Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Estudio e implementación de arquitecturas RAG avanzadas: Mejora en asistentes virtuales con documentación corporativa

---

Alumno: Jorge Poza Tamayo

Tutor/a/es: Juan José Álvarez Sánchez

Estudio e implementación de arquitecturas RAG avanzadas: Mejora en asistentes virtuales con documentación corporativa

Jorge Poza Tamayo

Dedicado a mi padre.

Agradecimientos

Quisiera dedicar este trabajo, en primer lugar, a mi madre y a mi hermano. Vuestro apoyo incondicional, vuestra paciencia y vuestros ánimos en los momentos difíciles han sido el motor para llegar hasta aquí. Este logro es tan vuestro como mío.

Gracias a mi tutor, D. Juan José Álvarez Sánchez, por la confianza e ir adelante en mi idea de llevar a cabo este proyecto. Agradecer también al resto de profesores de la Escuela de Ingeniería de Informática de Segovia por los conocimientos trasmitidos a lo largo de todos estos años

Resumen

En el entorno corporativo actual, la gestión del vasto y creciente volumen de documentación interna representa un desafío crítico, que a menudo resulta en una pérdida de eficiencia y una barrera para el acceso rápido a la información. La aparición de LLMs (*Large Language Models*) ofrece una oportunidad sin precedentes para transformar la interacción con estas bases de conocimiento.

Este Trabajo de Fin de Grado aborda directamente esta problemática mediante el diseño y desarrollo de DocuBot, un asistente conversacional avanzado integrado en Microsoft Teams para la consulta de documentación.

La solución desplegada sobre la infraestructura de servicios PaaS de Microsoft Azure, se fundamente en una arquitectura RAG (*Retrieval Augmented Generation*) avanzada, que va más allá de los enfoques convencionales. La solución incorpora técnicas como la reescritura de consultas mediante un LLM para optimizar la intención del usuario, una búsqueda híbrida multi-índice que combina la relevancia semántica y de palabras clave, y un mecanismo de auditoría donde otro LLM verifica las fuentes para garantizar la máxima fiabilidad y trazabilidad de la respuesta.

Este documento detalla el proceso completo, desde la fundamentación teórica de las tecnologías subyacentes hasta la planificación, el diseño de la arquitectura, la implementación y las pruebas funcionales, culminando en un producto novedoso, robusto y seguro, diseñado para resolver una necesidad empresarial tangible.

Palabras claves: DocuBot, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Chatbot Corporativo, Azure, Microsoft Teams.

Abstract

In today's corporate environment, managing the vast and ever-growing volume of internal documentation presents a critical challenge, often leading to inefficiencies and hindering swift access to information. The emergence of LLMs (*Large Language Models*) offers an unprecedented opportunity to transform the interaction with these knowledge bases.

This project directly addresses this challenge through the design and development of DocuBot, an advanced conversational assistant integrated into Microsoft Teams for querying documentation.

Deployed on the Microsoft Azure PaaS infrastructure, the solution is built upon an advanced RAG (*Retrieval-Augmented Generation*) architecture that goes beyond conventional approaches. The solution incorporates advanced techniques such as LLM-based query rewriting to optimize user intent, a multi-index hybrid search combining semantic and keyword relevance, and an auditing mechanism where a second LLM verifies the sources to ensure the utmost reliability and traceability of the response.

This document details the entire process, from the theoretical foundations of the underlying technologies to the planning, architectural design, implementation, and functional testing, culminating in a robust, secure product designed to solve a tangible business need.

Keywords: DocuBot, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Enterprise Chatbot, Azure, Microsoft Teams.

Índice General

Lista de Figuras

Lista de Tablas

Introducción

1.1 Estructura del documento

Este presente documento se organiza en nueve capítulos diferenciados entre sí, de forma que cada uno de ellos agrupe contenidos:

- El primer capítulo, **Introducción**, en el que se encuentra este apartado, detalla aspectos iniciales como la motivación, el contexto y los objetivos específicos que se persiguen para crear una solución de IA fiable y de valor.
- En el segundo capítulo, **Estudio del alcance y viabilidad**, se analiza el panorama competitivo y el estado del arte para posicionar estratégicamente la solución. Este capítulo justifica la viabilidad del proyecto no solo desde una perspectiva técnica, sino también económica y de mercado, concluyendo con un análisis DAFO.
- El **Fundamento Teórico del Proyecto** constituye el pilar conceptual del trabajo y da las bases teóricas en las que se apoya el proyecto. Desde los fundamentos acerca de la Inteligencia Artificial y NLP, hasta los LLMs y arquitecturas RAG avanzadas, integración con Microsoft Teams y repaso del ecosistema Azure.
- En el cuarto capítulo, **Planificación y Gestión del Proyecto**, se detalla el rigor metodológico aplicado. Se explica cómo se ha adaptado un marco de trabajo ágil para gestionar un proyecto de IA, incluyendo la planificación temporal, la estimación de esfuerzos mediante Puntos de Historia, la presupuestación y la gestión proactiva de riesgos.
- El quinto capítulo corresponde con el **Análisis Funcional** del proyecto, y en él Se definen los roles del sistema (Usuario Final, Administrador), se especifican los requisitos funcionales a través de Épicas e Historias de Usuario, y se establecen los requisitos no funcionales críticos.
- Posteriormente, en el sexto capítulo, se describe el **Modelado de Datos** a nivel conceptual y lógico del proyecto.
- En el séptimo capítulo, se detalla el **Diseño y Arquitectura** de la aplicación. El capítulo comienza con la exposición del Diseño de Experiencia de Usuario del proyecto y continúa con la Arquitectura. Se detallan tanto la arquitectura lógica de componentes de software como la arquitectura física de despliegue en la nube.
- El octavo capítulo describe dos aspectos importantes: **Implementación y Despliegue del proyecto**. El enfoque que recibe este capítulo se centra en explicar qué cosas se han hecho, cómo y por qué.
- Por último, el documento finaliza con un noveno capítulo de **Conclusiones,** como resultados y aceptación, líneas de trabajo futuras, así como con conclusiones personales.

Finalmente, el **Apéndice A** ofrece un **Manual de Usuario y Administrador**, una guía práctica y concisa que facilita la adopción y gestión de la herramienta en un entorno real.

1.2 Motivación

En la era de la información, todas las organizaciones y particulares se enfrentaron a un gran problema que era pasar toda la información y conocimiento que generaban físicamente a formatos digitales para poder ser consumido desde sus ordenadores, una vez superado ese reto se ha llegado en los últimos años una paradoja: a pesar de tener la capacidad y tecnología para generar y almacenar volúmenes de datos y conocimiento sin precedentes, el acceso efectivo a esa información se ha vuelto cada vez más complejo y fragmentado.

Manuales de procedimiento, normativas de calidad, políticas de seguridad, documentación técnica y guías de usuario se encuentran dispersos en silos digitales como SharePoint, Google Drive, diversas intranets y sistemas de gestión documental. Este fenómeno de "sobrecarga de información" obliga a los empleados a navegar por una inmensidad de repositorios documentales, estructurados en carpetas que únicamente el autor conoce la lógica de organización, dependiendo de motores de búsqueda tradicionales que, basados en palabras clave, obligan al usuario a conocer el nombre exacto del documento y raramente proporcionan respuestas directas, sino una lista de documentos que deben ser revisados manualmente. Esta ineficiencia no solo frustra a los empleados, sino que también representa un coste oculto significativo en la productividad empresarial.

Paralelamente, estamos siendo partícipes de una revolución en la interacción humano-computadora impulsada por los Modelos Grandes de Lenguaje (LLMs). Se han puesto a disposición de forma generalizada y sin barreras a través de herramientas de IA conversacional que han redefinido las expectativas de los usuarios, quienes ahora demandan inmediatez, contexto y respuestas directas a sus consultas. Esta nueva realidad ha creado una brecha notable entre el avance de la tecnología de consumo y la funcionalidad, a menudo obsoleta, de las herramientas corporativas internas.

La motivación principal de este proyecto surge de la combinación de estos dos mundos: la necesidad empresarial de desbloquear de manera rápida y efectiva el valor de su conocimiento interno y la oportunidad tecnológica que ofrece la IA generativa para hacerlo de una manera fundamentalmente innovadora, abordando un problema de negocio real y persistente.

En el marco de este Trabajo de Fin de Grado, se ha decidido abordar este desafío no como un mero ejercicio teórico, sino como una oportunidad para diseñar y construir una solución de nivel empresarial. Este contexto académico ha permitido dedicar el tiempo y el rigor necesarios para investigar en profundidad las arquitecturas más avanzadas y aplicar las mejores prácticas de la ingeniería de software, resultando en una solución que es a la vez innovadora y robusta.

El enfoque no se limita a crear un chatbot, sino a investigar y construir los cimientos de un asistente de conocimiento corporativo inteligente. La idea es transformar la tediosa tarea de "buscar" información en una fluida "conversación" con un experto virtual que conoce a fondo la documentación de la empresa. Otra motivación clave es la democratización del conocimiento organizacional, un asistente como el propuesto

puede romper las barreras de información que tradicionalmente existen entre departamentos. Facilita enormemente el proceso de incorporación de nuevos empleados, proporcionándoles un "experto virtual" disponible 24/7 que puede. Además, asegura la consistencia de la información a lo largo de toda la empresa, garantizando que todos los empleados, sin importar su rol o antigüedad, accedan a la misma fuente de verdad actualizada.

La decisión de integrar esta solución en Microsoft Teams es estratégica, motivada por la necesidad de llevar la herramienta directamente al flujo de trabajo diario del usuario, las mejores herramientas son las que se integran directamente donde los usuarios ya trabajan, minimizando la barrera de entrada y maximizando la adopción en cualquier tipo de corporación.

Finalmente, un aspecto crucial que motiva este desarrollo es el desafío de la confianza y la fiabilidad en la IA. En un entorno profesional, una respuesta que no puede ser verificada carece de valor. Por ello, este proyecto se quiere alejar del modelo de "caja negra" y se enfoca en la transparencia. Es crucial entender el rol de la herramienta: es un “copiloto” diseñado para asistir, no para reemplazar, el juicio humano. La verificación final por parte del usuario es un pilar del sistema de IA, mostrar que puede no solo responder preguntas, sino también justificar sus respuestas citando las fuentes exactas, generando así la confianza indispensable para su uso en la toma de decisiones críticas.

En resumen, la motivación de este trabajo es doble: por un lado, dar respuesta a una necesidad actual, real y universal de las organizaciones modernas; y por otro, explorar y aplicar de manera responsable y segura las tecnologías de IA más avanzadas para crear una solución que sea a la vez potente, fiable y perfectamente integrada en el ecosistema empresarial.

1.3 Objetivos

Este Trabajo de Fin de Grado (TFG) representa una síntesis entre los fundamentos de la ingeniería de software adquiridos durante el grado así como una profunda investigación autodidacta en el campo de la Inteligencia Artificial Generativa, así como en el porfolio de tecnologías Azure. Mientras que las fases de análisis, diseño y gestión del proyecto se apoyan en metodologías de desarrollo de software consolidadas, la implementación técnica del núcleo de IA es el resultado de un aprendizaje para aplicar tecnologías de vanguardia a un problema empresarial real [1], también como en documentarlo con la rigurosidad que corresponde.

El proyecto aunque en esta memoria se presenta como un aplicativo para una gran corporación, no se concibe únicamente en origen como una aplicación monolítica para un único cliente o silo documental, sino además como un acelerador de soluciones o una plantilla de referencia. Su diseño modular y su configuración basada en variables de entorno le otorgan una naturaleza altamente replicable, permitiendo su adaptación e implementación en diferentes bases documentales con un esfuerzo mínimo de

personalización. En el contexto educativo de esta memoria se hablará siempre como un único sistema.

El objetivo principal de este trabajo es, por tanto, el diseño, desarrollo y documentación de un asistente de conocimiento corporativo para Microsoft Teams nivel empresarial, al que en adelante nos referiremos como **"DocuBot"**. Esta aplicación utilizará una arquitectura avanzada de RAG construida en un entorno cloud de Azure para proporcionar a los empleados respuestas precisas, contextualizadas y verificables, basadas exclusivamente en la base documental privada de la organización.

Este objetivo principal del TFG se desglosa en los siguientes objetivos específicos:

- Asumir un rol híbrido de Ingeniero de IA y Arquitecto Cloud, llevando a cabo un proceso de desarrollo completo que abarca:
    - El "front-end" conversacional: El diseño e implementación de la experiencia de usuario dentro de Microsoft Teams, utilizando exclusivamente Tarjetas Adaptables para toda la interacción.
    - El "back-end" distribuido: La arquitectura, orquestación e integración de un conjunto de servicios PaaS (*Platform as a Service*) en la nube de Azure, incluyendo los servicios de IA, bases de datos y lógica de aplicación.
    - Aplicar un proceso que va desde la planificación y el análisis inicial, pasando por el diseño y la implementación, hasta el despliegue en un entorno en la nube y la documentación final.
    - La gestión del proyecto mediante principios ágiles, desgranando el proyecto en épicas como grandes bloques funcionales, que a su vez se desglosaron en tareas concretas y manejables, organizadas en sprints.
- Implementar un pipeline avanzado de *Retrieval Augmented Generation* (RAG):
    - Desarrollar un sistema capaz de comprender las preguntas de los usuarios, buscar en la base documental privada mediante técnicas de búsqueda híbrida y generar respuestas en lenguaje natural, citando siempre las fuentes para garantizar la verificabilidad.
    - Investigar y dominar un stack tecnológico en Azure, incluyendo *Azure OpenAI* (para los LLMs y embeddings), *Azure AI Search* (para la búsqueda vectorial), *Azure Blob Storage* (almacenamiento en la nube) y *Azure Key Vault* (para la gestión segura de secretos).
- Diseñar una arquitectura de software limpia, escalable y segura en la nube de Azure:
    - Construir una arquitectura modular y desacoplada, separando claramente las responsabilidades en capas para facilitar su mantenimiento y evolución.
    - Gestionar todos los secretos de la aplicación de forma centralizada y segura.

- Elaborar una memoria técnica de gran detalle:
    - Documentar el proyecto para que sirva como guía práctica y estudio de caso sobre cómo abordar el diseño y la implementación de soluciones de IA fiables en un entorno empresarial, destacando los desafíos encontrados y las decisiones de diseño tomadas.

Alcance, Entorno y Viabilidad

En este capítulo se centra en contextualizar dicha solución que ha sido planteada en el primer capítulo a través de los objetivos y la naturaleza del proyecto. Se realizará en este un estudio del alcance funcional, se analizará su posicionamiento frente a las alternativas tecnológicas existentes (*state of the art*), y se justificará su viabilidad sino mediante el análisis del problema de negocio específico que resuelve en un entorno corporativo. El capítulo concluirá con un análisis DAFO que sintetiza la posición estratégica del sistema DocuBot.

2.1 Descripción del alcance del proyecto

El proyecto se constituye con un alcance determinado con el fin de satisfacer los objetivos anteriormente expuestos, el alcance define las capacidades concretas que DocuBot ofrece al usuario final en su primera versión. Este alcance se ha delimitado cuidadosamente para entregar un producto de valor tangible, dejando funcionalidades más complejas o que no sean necesarias para futuras evoluciones.

El alcance, por tanto, del proyecto, se centra en el desarrollo de un chatbot nivel empresarial multimodal (texto e imagen) que tenga:

- **Interfaz Conversacional en Microsoft Teams:** La interacción se realiza exclusivamente a través de una aplicación en Microsoft Teams, como canal principal de la comunicación.
- **Búsqueda Vectorial Asistida:** El usuario puede iniciar una búsqueda y, opcionalmente, acotarla mediante una selección de categorías de sistema predefinidas, permitiendo una búsqueda más dirigida.
- **Respuesta Sintetizada con Citaciones:** DocuBot proporciona respuestas en lenguaje natural sintetizadas a partir de los documentos internos, y cada respuesta va acompañada de un conjunto de citas verificables que apuntan a los documentos fuente.
- **Interacción con Documentos Fuente:** El usuario puede explorar los detalles de cada documento citado e incluso abrir el documento original directamente desde la tarjeta de respuesta.
- **Gestión de la Conversación:** DocuBot maneja el ciclo de vida de la conversación, incluyendo el inicio explícito de un nuevo diálogo y la gestión de la expiración de la sesión por inactividad.
- **Sistema de Feedback Detallado:** Se ha implementado un mecanismo de retroalimentación en dos fases: una valoración inicial rápida (útil/no útil) y un formulario opcional para comentarios más detallados sobre la calidad y relevancia de la respuesta.

Debido al diseño modular permite que este alcance definido sea la base sobre la que futuras implementaciones fuera del alcance de esta primera versión y se puedan añadir capacidades como paneles de analítica, soporte multicanal o ingesta de documentos autogestionada.

2.2 Antecedentes y Estado del arte

Para posicionar adecuadamente la solución DocuBot y comprender su relevancia, es fundamental analizar el panorama de herramientas y enfoques existentes que abordan el desafío de conectar eficientemente a los empleados con el conocimiento corporativo. Este análisis se divide en una breve retrospectiva histórica, una descripción del entorno actual de soluciones y una identificación de las deficiencias que el presente proyecto busca subsanar.

2.2.1 Breve Historia

La gestión del conocimiento corporativo ha evolucionado en paralelo a la propia tecnología de la información. En sus orígenes, el conocimiento residía en formatos físicos como manuales impresos y archivos, cuyo acceso era inherentemente lento y dependiente de la organización manual. La digitalización masiva trajo consigo los sistemas de ficheros en red, mejorando la durabilidad y el acceso remoto, pero manteniendo la carga de la localización de la información sobre el usuario, que necesitaba conocer la estructura de directorios para encontrar un documento.

El verdadero primer gran salto se produjo con la llegada de las intranets corporativas y los sistemas de gestión documental en las décadas de 1990 y 2000. Plataformas como las primeras versiones de SharePoint introdujeron los motores de búsqueda basados en palabras clave [2], una tecnología que indexaba datos de los documentos y permitía a los usuarios encontrar archivos relevantes a partir de términos de búsqueda. Este modelo, aunque revolucionario en su momento, se basaba en la coincidencia exacta de texto y carecía de una comprensión real del contexto o la intención del usuario, a menudo devolviendo resultados irrelevantes o dejando de encontrar información crucial por una simple diferencia en la terminología usada.

Han ido evolucionando plataformas como Microsoft 365 (SharePoint Online) o Confluence. Han mejorado sus motores de búsqueda con capacidades de ranking más avanzadas y ofrecen una mejor integración y colaboración. Sin embargo, su paradigma fundamental sigue siendo el de un "repositorio inteligente": el usuario busca y el sistema devuelve una lista de documentos. La tarea de leer, sintetizar y extraer la respuesta final sigue recayendo en el empleado.

2.2.2 Entorno Actual

En la actualidad, el sector de la gestión del conocimiento empresarial está experimentando una transformación sin precedentes, impulsada por la integración de la IA Generativa. El mercado ya no se define solo por la capacidad de almacenar y organizar documentos, sino por la habilidad de interactuar con ellos de forma inteligente. Este nuevo paradigma se caracteriza por:

- **Interfaces Conversacionales:** Los usuarios esperan poder "hablar" con sus datos, haciendo preguntas en lenguaje natural en lugar de formular consultas de búsqueda complejas.
- **Síntesis de Información:** Las herramientas más avanzadas ya no solo devuelven una lista de documentos, sino que sintetizan la información de múltiples fuentes para proporcionar una respuesta directa y coherente.
- **Contextualización Profunda:** Los sistemas son cada vez más conscientes del contexto del usuario (quién es, en qué está trabajando) para personalizar las respuestas y hacerlas más relevantes.
- **Integración en el Flujo de Trabajo:** La funcionalidad de búsqueda y respuesta se está incrustando directamente en las aplicaciones que los empleados ya utilizan a diario eliminando la necesidad de cambiar de contexto.

A continuación, se procede a analizar las plataformas líderes que compiten en este nuevo entorno, comparándolas con el enfoque de DocuBot.

Microsoft 365 Copilot

Microsoft 365 Copilot es, sin duda, el actor más dominante en el ecosistema empresarial de Microsoft. Se posiciona como un asistente de IA integrado en todo el conjunto de aplicaciones de Microsoft 365 (Word, Excel, Outlook, y muy notablemente, Teams) [3]. Copilot utiliza una arquitectura RAG propia muy sofisticada que se conecta al Microsoft Graph para acceder a los datos de un usuario (correos, documentos, chats, calendario) de forma segura.

- **Fortalezas:** Su principal ventaja es su integración nativa y profunda. No requiere una implementación separada; "simplemente funciona" dentro de las herramientas que los empleados ya usan. Su capacidad para correlacionar información entre diferentes aplicaciones (por ejemplo, resumir correos relacionados con un documento de Word) es extremadamente potente.
- **Posicionamiento frente a DocuBot:** Copilot es una solución de propósito general, diseñada para la productividad personal y de equipo. DocuBot, en cambio, se especializa en la consulta de corpus documentales corporativos y controlados (normativas, manuales de calidad, procedimientos técnicos), un caso de uso más específico.

Mientras Copilot responde basándose en "todo" a lo que un usuario tiene acceso y te puede ayudar en otras tareas como tomar notas en reuniones, realizar documentos y presentaciones y una infinitud de funcionalidades, DocuBot responde basándose en la base documental curada y definida por la organización.

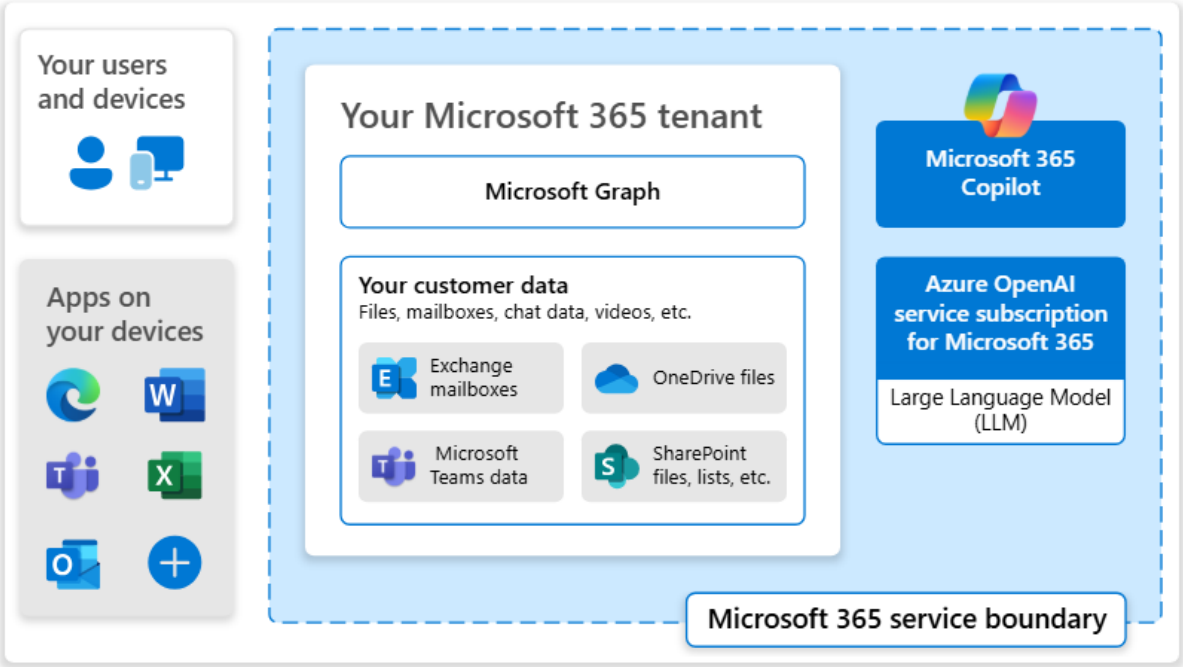


Figura 1. Capacidades Microsoft Copilot 365

Google NotebookLM

NotebookLM es la propuesta de Google en este mercado [4]. Está diseñado como un asistente de investigación personal que permite a los usuarios cargar sus propios documentos (desde Google Drive o subiéndolos directamente) y luego usar sus modelos para hacer preguntas, resumir y generar ideas a partir de ese contenido.

- **Fortalezas:** Su interfaz es muy limpia y se centra exclusivamente en la tarea de "razonar" sobre un conjunto de documentos proporcionado por el usuario. Es excelente para tareas individuales de análisis, investigación o estudio.
- **Posicionamiento frente a DocuBot:** NotebookLM es fundamentalmente una herramienta personal, no una solución empresarial escalable. No está diseñado para ser un asistente centralizado para toda una organización. Carece de los mecanismos de gestión de permisos centralizados, la integración con sistemas de identidad, y no está pensado para ser desplegado como un bot en plataformas colaborativas como Teams. DocuBot es, por definición, una solución multi-usuario y gestionada a nivel de organización.

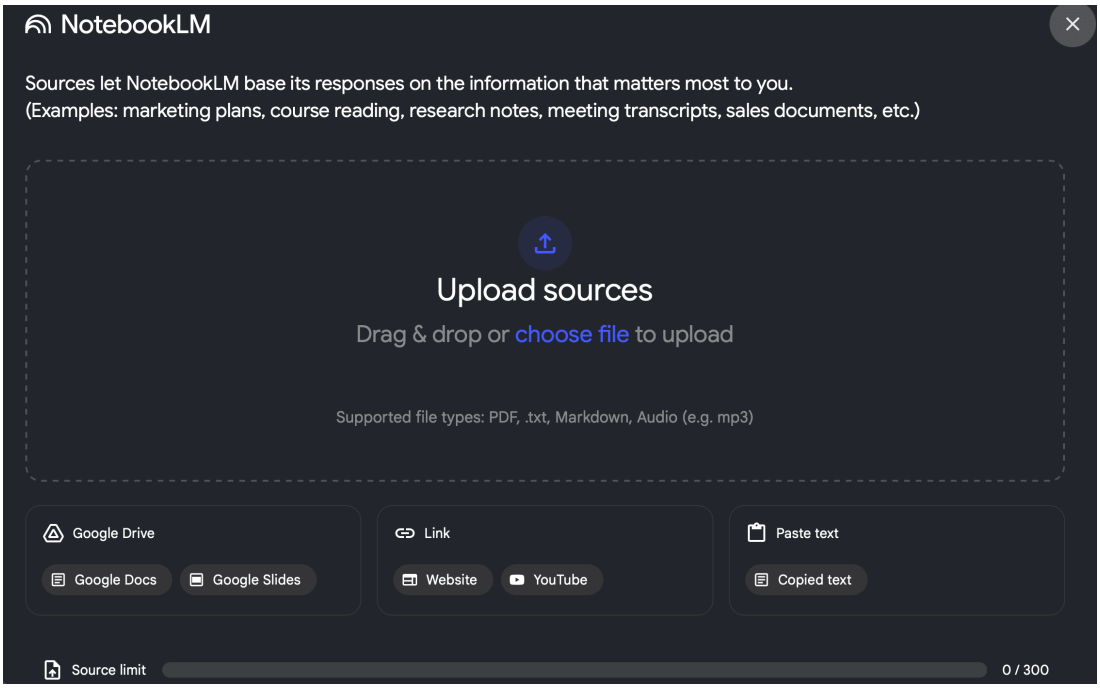


Figura 2. Interfaz NotebookLM

Glean

Glean es una de las startups en el campo de la búsqueda de información empresarial con IA [5]. Se especializa en conectarse a una amplia variedad de aplicaciones empresariales de terceros (Slack, Jira, Confluence, Salesforce, etc.). Su objetivo es crear un grafo de conocimiento de toda la empresa y permitir a los empleados buscar información y obtener respuestas sintetizadas a través de todas estas plataformas.

- **Fortalezas:** Su gran ventaja es su extensibilidad y neutralidad de plataforma. Es una solución ideal para empresas con un ecosistema tecnológico muy diverso en proveedores.
- **Posicionamiento frente a DocuBot:** Glean es una solución comercial de SaaS muy potente pero, al igual que Copilot, es en gran medida una "caja negra" propietaria. Las empresas pagan por usar el servicio, pero tienen un control limitado sobre la lógica interna, los modelos utilizados o los flujos de trabajo de IA. DocuBot ofrece una transparencia y personalización totales, permitiendo a una organización auditar, modificar y ser dueña de cada aspecto de su solución de IA. Es la opción para quienes buscan construir una capacidad estratégica interna en lugar de consumir un servicio de terceros.

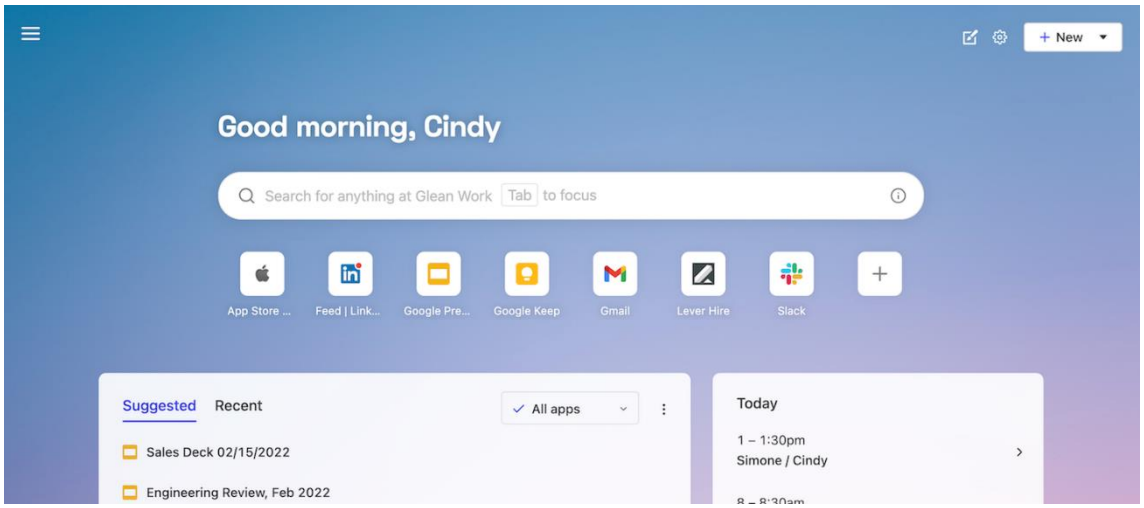


Figura 3. Interfaz inicial de Glean

2.2.3 Deficiencias Identificadas y Oportunidad de Proyecto

El análisis del estado del arte revela que, si bien existen soluciones de IA empresarial muy potentes, estas presentan deficiencias, especialmente cuando se evalúan desde la perspectiva de un caso de uso específico y controlado como el que puede abordar DocuBot. Estas deficiencias no invalidan a las plataformas comerciales, pero sí abren una clara oportunidad para una solución especializada y propietaria. Se expondrán en detalle las dos deficiencias más relevantes:

Deficiencia 1. El Modelo de Licenciamiento y su Ineficiencia en Costes

Las soluciones líderes como Microsoft 365 Copilot o NotebookLM operan bajo un modelo de licenciamiento rígido de coste por usuario y mes (aproximadamente 30€ en el momento de este estudio) [6], [7]. Si bien este modelo simplifica la adquisición, presenta una deficiencia económica fundamental para muchos escenarios corporativos:

- **Coste Desproporcionado para Uso Específico:** Una empresa se ve obligada a pagar el coste completo de la licencia para cada empleado, independientemente de si su uso se limita a una funcionalidad específica (como consultar normativas) o si aprovecha todo el espectro de capacidades de la herramienta. Para una organización donde solo un subconjunto de empleados necesita acceso intensivo al conocimiento corporativo, este modelo resulta económicamente ineficiente, pagando por una funcionalidad infrautilizada por la mayoría.
- **Falta de Escalabilidad:** El coste escala linealmente con el número de usuarios, no con el uso real. Esto penaliza a organizaciones donde muchos empleados podrían ser usuarios ocasionales del sistema de consulta documental.

La oportunidad para DocuBot es ofrecer un modelo de costes basado en el consumo real (*pay-as-you-go*). Al operar sobre la infraestructura de Azure, los costes se asocian directamente al número de llamadas a las APIs de IA y al almacenamiento en la nube, no a un número fijo de licencias. Esto permite a la empresa:

1. **Optimizar la Inversión:** Pagar únicamente por el valor que se extrae del sistema.
2. **Abrir el Acceso:** Ofrecer la herramienta a toda la organización sin incurrir en un coste recurrente prohibitivo por licencia, sabiendo que el coste total reflejará la demanda real.

Deficiencia 2. "Caja Negra": Potencia a costa de Control y Especialización

Como se mencionó en el posicionamiento, las plataformas comerciales son inherentemente "cajas negras". Esta opacidad, si bien simplifica la implementación, se convierte en una deficiencia crítica para casos de uso que requieren alta fiabilidad y auditabilidad.

- **Incapacidad de Ajustar el motor RAG:** Las empresas no pueden afinar el comportamiento del motor de IA para priorizar ciertos tipos de documentos sobre otros, o para adaptar el *prompting* a la jerga o a los matices específicos de su sector.

La oportunidad para DocuBot es ser la antítesis de la caja negra. Al ser un acelerador de código abierto, ofrece:

1. **Control Total sobre el Corpus Documental:** La empresa define con precisión la base de conocimiento, asegurando que las respuestas provengan de una única fuente de verdad.
2. **Personalización Profunda:** Permite modificar cada aspecto del pipeline RAG: desde el pre-procesamiento de los documentos y la estrategia de *chunking*, hasta el refinamiento de los *prompts* y la lógica de re-ranking de los resultados.

En síntesis, mientras el mercado se llena de soluciones de IA generalistas bajo modelos de suscripción fijos, emerge una necesidad clara de herramientas especializadas, controlables y económicamente eficientes. DocuBot se posiciona directamente en este nicho, ofreciendo a las organizaciones la capacidad de construir una solución de IA a medida, en lugar de simplemente consumir una.

2.2.4 Justificación de la Elección Tecnológica

La decisión de construir DocuBot sobre el ecosistema de Microsoft y su nube, Azure, no es casual, sino una elección estratégica fundamentada en la realidad del mercado empresarial actual. Dado que el objetivo es integrar el asistente directamente en el flujo de trabajo del empleado, se ha decido que fuera a través de **Microsoft Teams**, la elección de Teams como interfaz principal se justifica por la abrumadora cuota de mercado que Microsoft 365 ostenta en el entorno empresarial [8]. Al poder disponibilizar DocuBot en la plataforma donde la mayoría de los empleados ya colaboran, se maximiza la visibilidad y se minimizan las barreras para su adopción.

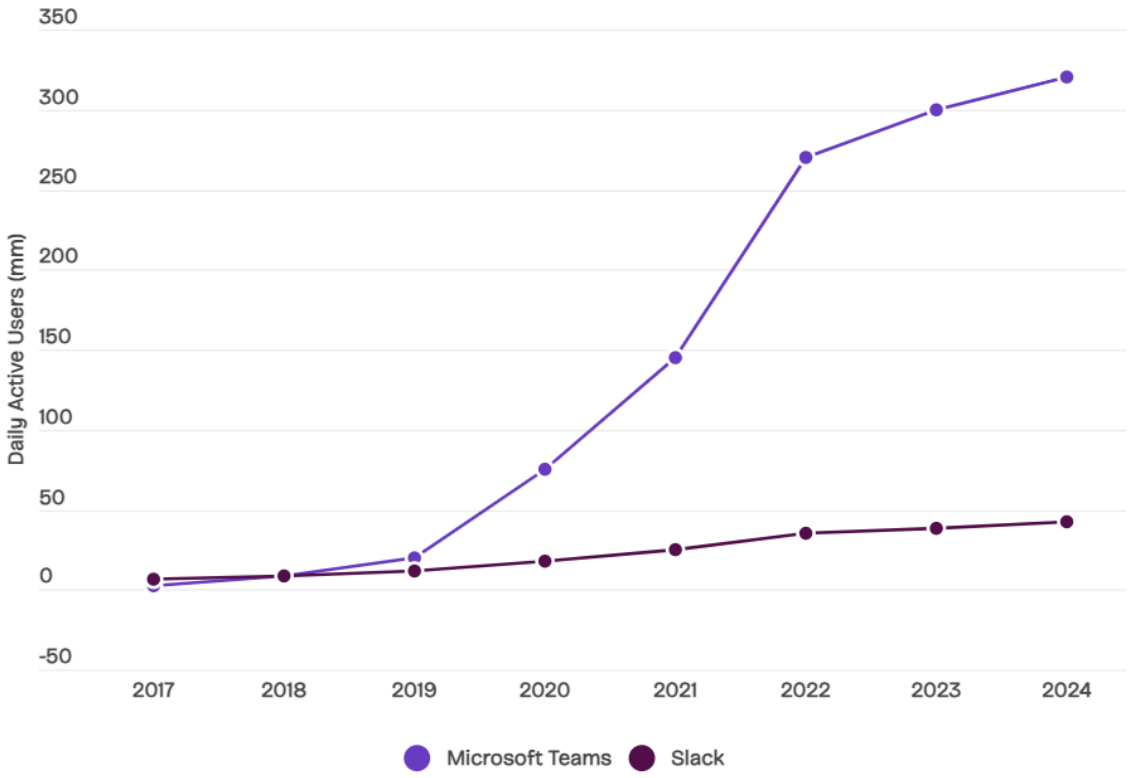


Figura 4. Comparativa uso Microsoft Teams vs competidores

La elección de Azure como plataforma tecnológica ofrece una profunda integración con el software empresarial de Microsoft lo convierte en la opción preferente para organizaciones que ya operan con Microsoft 365. Aunque existen varios competidores en el mercado de la nube, Microsoft Azure se consolida como el segundo actor principal en el mercado global [9].

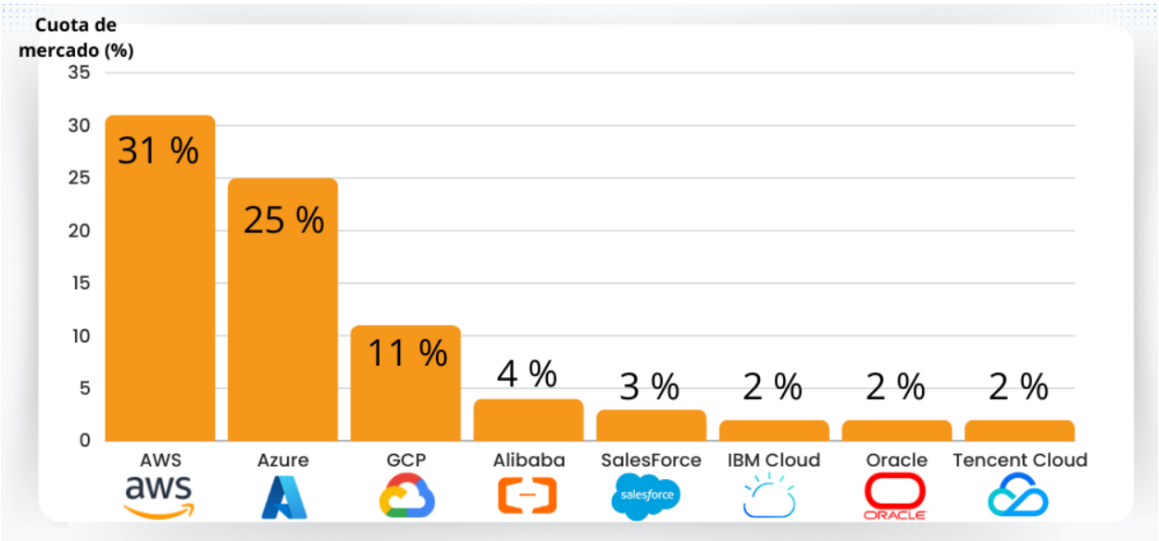


Figura 5. Cuota de mercado proveedores de Cloud.

En resumen, la dominancia de Microsoft en el software ofimático empresarial proporciona una base estable y segura [10]. La elección de Azure no solo responde a una conveniencia técnica derivada de la primera elección, sino a una estrategia de negocio que aprovecha la cuota de mercado y la confianza que las empresas depositan por lo general en Azure.

2.3 Análisis de Viabilidad

La viabilidad de un proyecto tecnológico como el que se tienen en este caso no solo depende de la posibilidad de construirlo, sino también de su adecuación al entorno como se ha ido viendo a lo largo de todo este capítulo y su justificación desde una perspectiva de negocio.

Técnicamente, la solución es factible gracias a la consolidación de los servicios en la nube de Microsoft Azure, que proporcionan acceso a modelos de IA de última generación (*Azure OpenAI*) y a potentes motores de búsqueda vectorial (*Azure AI Search*). El desarrollo se apoya en el patrón de arquitecturas RAG, que mitiga eficazmente las alucinaciones y los riesgos de respuestas incorrectas, y se acelera mediante librerías como la Microsoft Teams AI Library, que simplifican la interfaz y la integración con el ecosistema de Microsoft.

Desde el punto de vista operativo, DocuBot está diseñado para una adopción sin fricciones. Al integrarse como una aplicación nativa en Microsoft Teams, se inserta directamente en el flujo de trabajo diario de los empleados, eliminando la necesidad de aprender a usar una nueva herramienta. La interacción se realiza mediante lenguaje natural, lo que anula cualquier curva de aprendizaje técnica.

Finalmente, la viabilidad de negocio es la más contundente [11]. DocuBot aborda un problema empresarial fundamental: el tiempo y los recursos malgastados en la búsqueda de información. El retorno de la inversión (ROI) se manifiesta de forma directa a través de:

- **Aumento de la Productividad:** Al proporcionar respuestas instantáneas y precisas, se recuperan miles de horas de trabajo al año que los empleados dedican a buscar en repositorios de documentos o a consultar a sus compañeros.
- **Mitigación de Riesgos:** El acceso rápido a la información correcta y actualizada reduce la probabilidad de errores costosos derivados del uso de datos obsoletos o mal interpretados, mejorando la calidad y el cumplimiento normativo.
- **Aceleración de la Formación:** Actúa como un "experto virtual" para los nuevos empleados, permitiéndoles ser autónomos y productivos mucho más rápido, lo que reduce los costes y el tiempo asociados a los procesos de *onboarding*.

En conclusión, DocuBot no es solo un proyecto técnicamente realizable, sino una solución estratégica que ofrece un valor de negocio medible y significativo.

2.4 Análisis DAFO y Conclusiones

Para concluir este estudio de alcance y viabilidad, se presenta un análisis DAFO que sintetiza la posición estratégica del proyecto DocuBot. Este análisis permite consolidar los factores internos (Fortalezas y Debilidades) y externos (Oportunidades y Amenazas) que definen el entorno del proyecto, facilitando la toma de decisiones estratégicas.

Figura 6. Diagrama Análisis DAFO

El estudio realizado en este capítulo confirma que el proyecto DocuBot es viable, relevante y estratégico. Los beneficios potenciales en términos de productividad y eficiencia superan significativamente los riesgos y costes asociados.

El análisis DAFO sintetiza esta posición: las fortalezas del proyecto, como la personalización y la seguridad, le permiten capitalizar las oportunidades de mejora de la eficiencia empresarial. Al mismo tiempo, se reconocen las debilidades, como la dependencia de la calidad de los datos, y las amenazas del competitivo entorno tecnológico, estableciendo un marco realista para la gestión y evolución del proyecto.

Fundamento Teórico del Proyecto

En este capítulo se pretende describir, desde un punto de vista global, los pilares teóricos y tecnológicos sobre los que se asienta el proyecto DocuBot. El capítulo realizara un recorrido que comienza describiendo a nivel conceptual lo necesario para generar los LLM (*Large Language Model*), como los pilares del NLP (*Natural Language Processing*), explorando conceptos esenciales como la tokenización y la crucial transformación del texto en representaciones vectoriales mediante *embeddings*. A continuación, se analiza la evolución que ha llevado a la arquitectura *Transformer*, el cimiento de los LLM.

Una vez presentados los LLMs, se detallará su funcionamiento y sus capacidades, pero se pondrá detalle en sus limitaciones, como las alucinaciones y el conocimiento limitado, que justifican la necesidad de una arquitectura más robusta para un entorno empresarial. Como solución a estos desafíos, el capítulo se adentra en la arquitectura RAG (*Retrieval Augmented Generation*), el paradigma central de este proyecto. Se desglosarán sus componentes y su funcionamiento para garantizar respuestas fiables y trazables, explorando también técnicas avanzadas que definen el estado del arte y son usadas en este proyecto, hasta la transición hacia futuros sistemas de agentes.

Finalmente, se conectará la teoría con la práctica al describir la pila tecnológica de Microsoft Azure seleccionada para la implementación, justificando así la elección de cada servicio clave que permite materializar la arquitectura RAG de DocuBot.

3.1 Procesamiento Lenguaje Natural (NLP)

Antes de abordar el concepto de LLM de manera coherente, se requiere primero entender una serie de términos que nos ayudarán a ver mejor el panorama completo.

El NLP es la disciplina de la inteligencia artificial sobre la que se construyen los LLM. Dado que un LLM es, en esencia, un sistema avanzado especializado en tareas de NLP, su comprensión exige un conocimiento previo de los principios que rigen este campo. Esta sección sentará dichas bases, explicando los conceptos fundamentales del NLP y profundizando en las técnicas clave, como tokenización y embeddings, que son instrumentales tanto para el funcionamiento de los LLMs como para los componentes de recuperación de la arquitectura RAG.

3.1.1 Conceptos Básicos de NLP

El **Procesamiento del Lenguaje Natural** (NLP, por sus siglas en inglés) es una rama de la inteligencia artificial enfocada en facilitar la interacción entre las computadoras y el lenguaje humano. El término fue acuñado en la década de 1950, con pioneros como

Alan Turing, quien propuso el famoso "Test de Turing" (1950) como una medida de la capacidad de una máquina para exhibir comportamiento inteligente [12].

Para comprender el alcance del NLP, es útil conocer algunas de sus tareas más comunes. Aunque el campo es vasto, este proyecto se centra en varias capacidades clave que permiten su funcionamiento:

- Recuperación de Información: Es el proceso de buscar y obtener documentos o fragmentos de texto relevantes para una consulta del usuario, y pilar de una arquitectura RAG (la “R”)
- Análisis Semántico: Consiste en extraer el significado de una oración, teniendo en cuenta el contexto.
- Generación de Texto: Una vez recuperada y comprendida la información, esta tarea consiste en producir una respuesta coherente y en lenguaje natural. Será la fase final del RAG (la “G”).
- Resumen de Texto: Aunque no es una tarea explícita, el LLM realiza un resumen implícito al sintetizar la información de múltiples documentos fuente en una respuesta concisa y directa

Para realizar estas tareas, los sistemas de NLP utilizan diferentes técnicas y métodos, que pueden ser clasificados en dos grandes enfoques: el simbólico y el estadístico.

El enfoque simbólico se basa en el uso de reglas y representaciones lógicas del lenguaje. Este enfoque requiere un gran conocimiento del dominio y del idioma, puede ser muy preciso, pero es rígido, costoso de mantener y tiene una baja capacidad de adaptación a lenguaje imprevisto.

El enfoque estadístico se basa en el uso de modelos matemáticos y probabilísticos, que son entrenados a partir de grandes corpus de texto. Este enfoque requiere menos conocimiento previo y puede generalizar mejor a casos nuevos o desconocidos, pero suele tener un menor grado de exactitud y comprensión.

En los últimos años, se ha desarrollado un tercer enfoque, que combina los dos anteriores, y que se conoce como el enfoque de aprendizaje profundo o *deep learning*. Este enfoque utiliza redes neuronales artificiales, que son capaces de aprender representaciones complejas y abstractas del lenguaje, a partir de múltiples niveles de procesamiento y diferentes tipos de información. Este enfoque ha demostrado un gran avance en el rendimiento de muchas tareas de NLP.

3.1.2 Tokenización

La tokenización es el primer paso fundamental en el procesamiento computacional del lenguaje. Consiste en dividir un texto en unidades más pequeñas llamadas *tokens*, que pueden representar palabras, subpalabras o incluso caracteres. Siendo el token la unidad mínima de procesamiento en NLP. Este paso es esencial en cualquier sistema de NLP,

ya que permite descomponer el texto en fragmentos manejables para su análisis y manipulación.

Históricamente, se han utilizado varios enfoques:

- **Tokenización por palabras:** Divide el texto en palabras. Es intuitivo, pero ineficiente para manejar vocabularios grandes y palabras desconocidas (problema de *Out-of-Vocabulary* o OOV).
- **Tokenización por caracteres:** Divide el texto en letras individuales. Resuelve el problema de OOV, pero genera secuencias extremadamente largas, aumentando la carga computacional.

Los modelos de lenguaje modernos, emplean un enfoque intermedio y mucho más eficaz: la **tokenización por subpalabras**.

Esta técnica equilibra los dos extremos anteriores. Funciona aprendiendo las combinaciones de caracteres más frecuentes en un gran corpus de texto y fusionándolas en un solo token. De esta manera:

- Las palabras comunes (como "hola" o "sistema") pueden representarse con un único token.
- Las palabras más raras o complejas (como "desfragmentación") se descomponen en subpalabras con significado ("des", fragmenta", ción").

Este método ofrece dos ventajas cruciales para los LLMs:

1. Maneja un vocabulario finito y eficiente: Se evita el problema de OOV, ya que cualquier palabra nueva puede ser construida a partir de subpalabras conocidas.
2. Captura de información morfológica: El modelo puede entender la relación entre palabras como "correr", "corriendo" y "corredor" porque comparten un token raíz común.

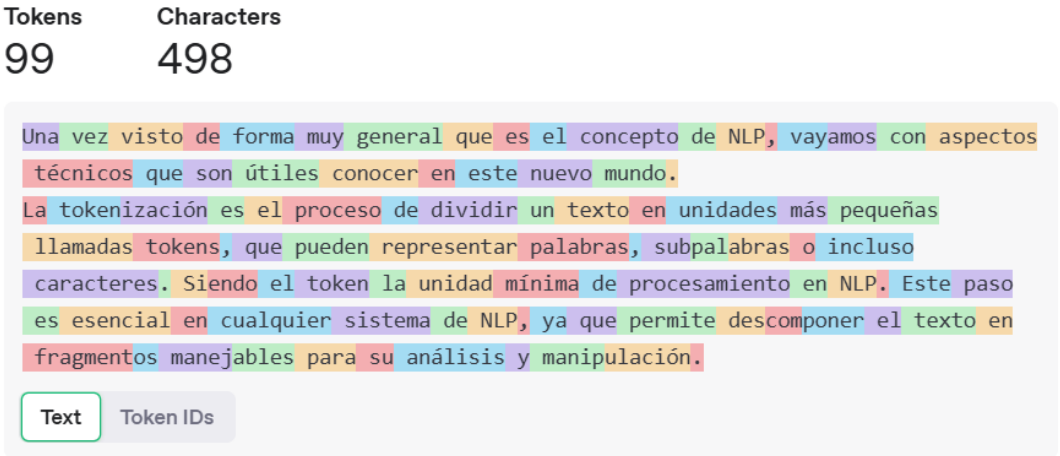


Figura 7. Sistema de Tokenización de OpenAI

Comprender la tokenización es fundamental en este proyecto por dos razones prácticas:

1. **Gestión del Límite de Contexto:** Los LLMs tienen una "ventana de contexto" máxima, es decir, un número límite de tokens que pueden procesar en una sola llamada. En nuestra arquitectura RAG, debemos asegurarnos de que la pregunta del usuario más el texto recuperado de los documentos no exceda este límite.
2. **Monitorización de Costes:** Los servicios como Azure OpenAI facturan por el número de tokens procesados (tanto en la entrada como en la salida).

Es importante señalar también que este es un cambio de paradigma respecto al NLP tradicional, donde eran comunes pasos de preprocesamiento manual. Los LLMs y modelos de embeddings modernos se entrenan con texto en su forma natural y son capaces de extraer el significado del contexto, incluyendo puntuación y mayúsculas.

3.1.3 Embeddings y Representaciones Vectoriales

Una vez que el texto se ha descompuesto en tokens, el siguiente paso crucial es transformar estas unidades discretas en una representación numérica que la máquina pueda procesar. Este proceso debe preservar el significado semántico del lenguaje, y para ello se utiliza una técnica fundamental conocida como embeddings. Un embedding es un vector denso de números que representa un fragmento de texto (una palabra, una frase o un documento entero) en un espacio matemático multidimensional. La propiedad esencial de este espacio es que la distancia y la dirección entre vectores corresponden a la relación semántica entre los textos que representan.

A continuación, se describen algunas de las principales técnicas de representación vectorial, desde métodos simples hasta embeddings avanzados, que son fundamentales en el procesamiento del lenguaje natural.

Fundamentos Matemáticos

Para representar el lenguaje en forma de vectores, es necesario exponer primero algunos conceptos básicos del álgebra lineal que serán utilizados por los modelos de embeddings.

Para evaluar la similitud entre vectores, se utilizan métricas como la similitud del coseno, esta medida calcula el coseno del ángulo entre dos vectores en el espacio. Un valor cercano a 1 indica que los vectores apuntan en direcciones muy similares (alta similitud semántica), un valor cercano a 0 indica ortogonalidad (poca o ninguna relación), y un valor cercano a -1 indica direcciones opuestas. Es la métrica por defecto en la mayoría de las bases de datos vectoriales, incluida la utilizada en este proyecto, por su eficacia al comparar la semántica independientemente de la magnitud de los vectores.

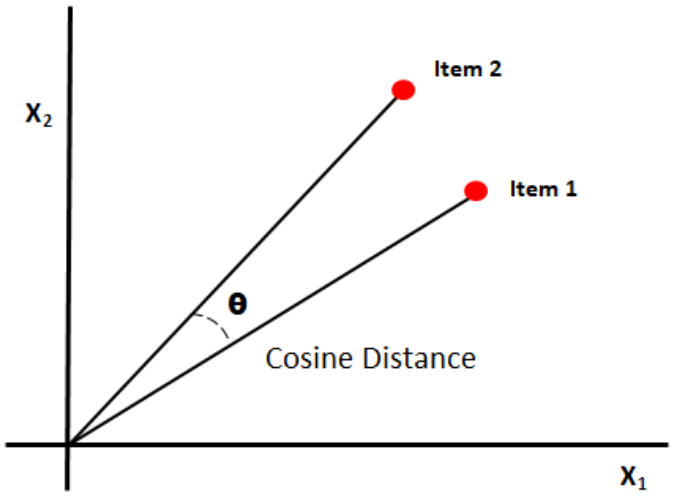


Figura 8. Similitud Coseno

Evolución de las Representaciones de Texto

Las técnicas de representación de texto han evolucionado desde enfoques básicos, como la codificación *one-hot* y el modelo de Bolsa de Palabras, hasta métodos de embeddings densos que capturan relaciones complejas entre palabras. A continuación se describen las principales técnicas de representación vectorial de texto simples.

La codificación one-hot es una de las primeras formas de representar texto, donde cada palabra del vocabulario se convierte en un vector binario con una dimensión por cada palabra del vocabulario. Por ejemplo, "gato" y "perro" no tienen ningún tipo de proximidad en este espacio de representación, a pesar de su relación conceptual.[13]

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Figura 9. One-Hot Encoding

El modelo de Bolsa de Palabras (BOW) es una técnica simple en el que un documento se representa como una colección de palabras, ignorando el orden y la gramática, y simplemente contando la frecuencia de aparición de cada palabra. Cada documento se convierte en un vector de frecuencias de palabras, donde cada dimensión corresponde a una palabra del vocabulario del corpus.

Vectores Densos

Con las representaciones de texto vistas hasta ahora no se pueden capturar las relaciones semánticas entre las palabras. Por ejemplo, las palabras "perro" y "gato" son semánticamente más cercanas que "perro" y "automóvil".

Los vectores densos ofrecen una solución más sofisticada. En lugar de asignar un número único a cada palabra, se representa cada palabra como un vector numérico de alta dimensión. Cada dimensión de este vector representa una característica semántica particular.

Ventajas de los vectores densos**:**

- **Representación única e interpretable:** Cada palabra tiene una representación única, y las dimensiones de los vectores a menudo pueden ser interpretadas en términos de características semánticas.
- **Captura de significado:** Los vectores densos capturan las relaciones semánticas entre las palabras. Por ejemplo, palabras con significados similares tenderán a tener vectores similares.
- **Reducción de dimensionalidad:** El número de dimensiones de los vectores densos es típicamente mucho menor que el tamaño del vocabulario, lo que hace que la representación sea más compacta y eficiente.

Word Embeddings

Una vez visto los avances que suponen los vectores densos en la representación vectorial, en lugar de codificar manualmente las representaciones de las palabras, como se hacía anteriormente, surge un nuevo enfoque: los word embeddings.

Los word embeddings son representaciones vectoriales densas de palabras a partir de grandes corpus de texto que mejoran significativamente las técnicas anteriores al capturar similitudes semánticas entre palabras en un espacio continuo de baja dimensionalidad (normalmente entre 100 y 300 dimensiones).

Las dimensiones de estos vectores no son tan fácilmente interpretables y cada dimensión es de valor real, pero las palabras similares tienen valores comparables en algunas dimensiones. Lo que permite incluso operaciones algebraicas que capturan relaciones entre palabras, un ejemplo que ilustra perfectamente esto es: “rey - hombre + mujer = reina” o “París - Francia + Italia = Roma”.

Embeddings Contextuales (BERT, GPT)

A diferencia de los métodos anteriores, que generan una representación fija para cada palabra, los embeddings contextuales como BERT y GPT producen representaciones que dependen del contexto en el que aparece la palabra [14]. Esto permite que estos modelos capten la polisemia, es decir, diferentes significados de una palabra según su

uso. De esta forma, el modelo es capaz de diferenciar entre "fui al banco a sacar dinero" y "me senté en el banco del parque".

Los modelos de embeddings modernos, como *text-embedding-ada-002* o los más recientes *text-embedding-3-large* de OpenAI, generan vectores de alta dimensionalidad (1536 y 3072 dimensiones respectivamente) que capturan matices semánticos con una precisión extraordinaria, siendo la tecnología clave para la búsqueda semántica.

Propiedades Clave de los Embeddings

Las representaciones vectoriales poseen varias propiedades que influyen en su capacidad de capturar relaciones semánticas.

- **Dimensionalidad**: Los embeddings de alta dimensionalidad, ofrecen mayor capacidad expresiva, aunque aumentan el costo computacional. En cambio, embeddings de baja dimensionalidad son más eficientes, aunque pueden perder información relevante.
- **Contextualización**: Los embeddings estáticos asignan una única representación por palabra sin considerar el contexto, mientras que los contextuales varían según el contexto, capturando la polisemia y mejorando el modelado de significados variados.
- **Visualización**: Adicionalmente técnicas como UMAP y PCA permiten proyectar los embeddings en espacios de menor dimensión, revelando agrupamientos semánticos y posibles sesgos en el modelo. Estas visualizaciones pueden mostrar cómo se agrupan palabras similares o cómo ciertos conceptos se relacionan en el espacio vectorial.[15]

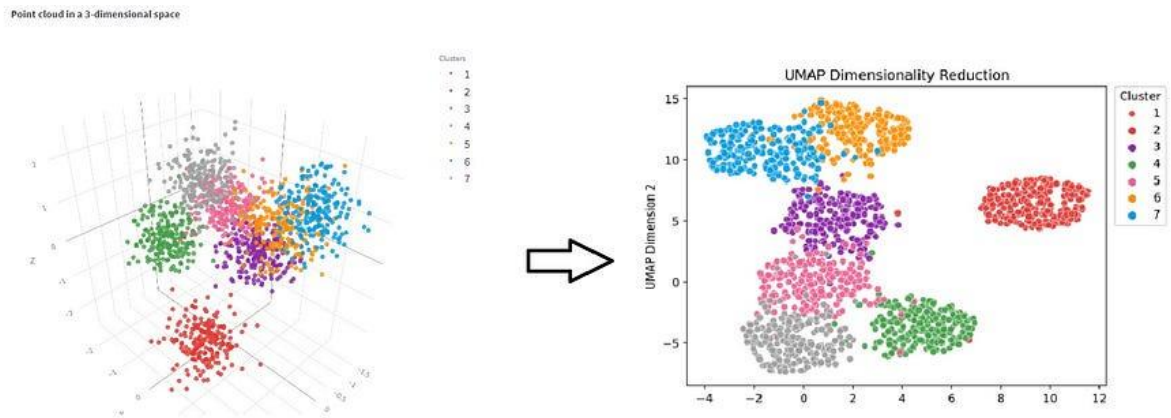


Figura 10. Reducción UMAP de dimensionalidad

3.1.4 Modelos de lenguaje y su evolución

Una vez que el texto se ha convertido en vectores numéricos (*embeddings*), el siguiente paso es procesar estas secuencias de vectores para comprender y generar lenguaje. El NLP ha experimentado una evolución significativa a lo largo de las últimas décadas. Desde los modelos probabilísticos tradicionales hasta las sofisticadas redes neuronales actuales. En esta sección, exploraremos los fundamentos y los desarrollos clave en la evolución de los modelos de lenguaje.

Redes Neuronales para NLP

Las Redes Neuronales Artificiales (RNA) son modelos computacionales inspirados en la estructura y el funcionamiento del cerebro humano para procesar información. Están diseñadas para reconocer patrones y aprender de los datos de manera similar a como lo hacen las neuronas biológicas. [16]

Consisten en una serie de nodos organizados por capas e interconectados que reciben información, la procesan y producen una salida. La capa de entrada recibe los datos y los envía a través de la red, mientras que la capa de salida produce la salida final. Entre la capa de entrada y la capa de salida, hay una o más capas ocultas que procesan la información.

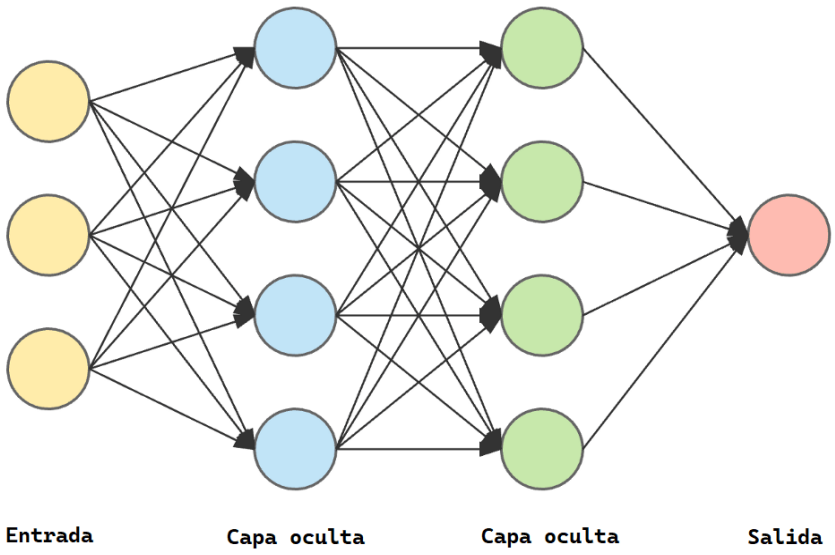


Figura 11. Estructura básica de una red neuronal

Dichos nodos, son conocidos como neuronas artificiales, además estos modelos son capaces de ajustar sus parámetros internos, conocidos como pesos.

El proceso de entrenamiento de una red neuronal implica proporcionarle datos de entrada y esperar una salida ya conocida. Se van ajustando los pesos de las conexiones en la red para mejorar la precisión. Este proceso se repite hasta que la red aprende a producir la salida deseada.

Las **funciones de activación** son una parte fundamental de cómo funcionan las redes neuronales. Cada neurona en la red neuronal utiliza una función de activación para determinar su salida en función de la entrada que recibe.

Las funciones de activación pueden ser lineales o no lineales. La elección de la función de activación puede tener un impacto significativo en la capacidad de la red para aprender y generalizar a nuevos datos. Algunas funciones de activación comunes incluyen la función sigmoide, la función ReLU y la función tangente hiperbólica.

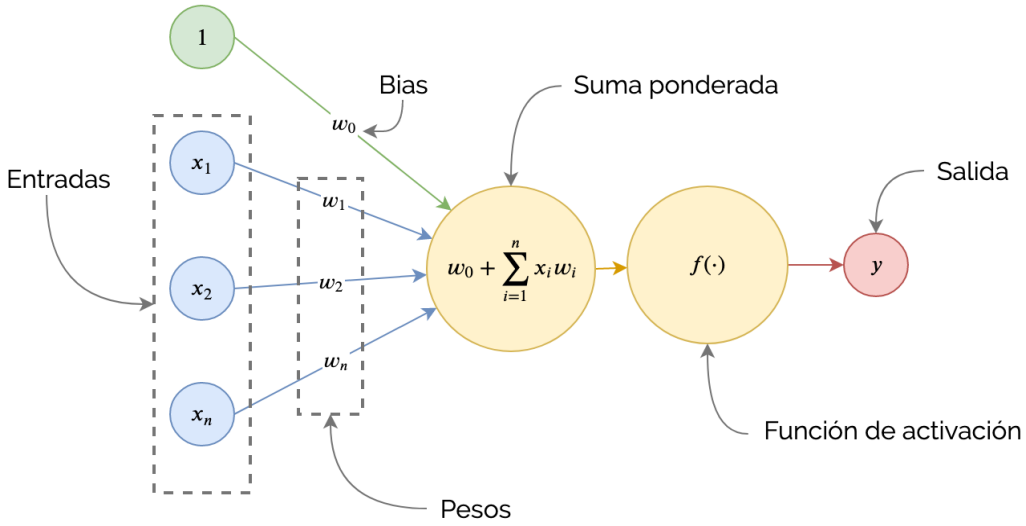


Figura 12. Componentes de una red neuronal

Tipos de Redes Neuronales

Las redes neuronales artificiales pueden clasificarse en función de cómo fluyen los datos desde el nodo de entrada hasta el nodo de salida, las más importantes para los actuales usos y como base de construcción de modelos más complicados son las llamadas feedforward, recurrentes o convolucionales.

Las redes neuronales feed-forward son las arquitecturas más sencillas, donde la información fluye en una sola dirección, desde la entrada hasta la salida, sin ciclos o retroalimentación. Esto significa que la salida de cada capa se utiliza como entrada para la siguiente capa hasta que se produce la salida final.

Estas redes siguen siendo útiles en aplicaciones de clasificación simple o representación de texto estática.

- **Redes Neuronales Recurrentes (RNN)**

  Las RNN, introducidas para modelar secuencias, están diseñadas para procesar datos secuenciales y temporales. Introducen conexiones recurrentes que permiten conservar información sobre estados previos, proporcionando una "memoria" de corto plazo. Esto significa que la salida de una neurona se utiliza como entrada para otra neurona, y así sucesivamente.

  Permiten que la información fluya entre palabras sucesivas de un texto. Una RNN cuenta con un estado oculto que se actualiza en cada paso de la secuencia, de modo

que puede mantener información del contexto previo dando esa capacidad de memoria que se ha comentado.

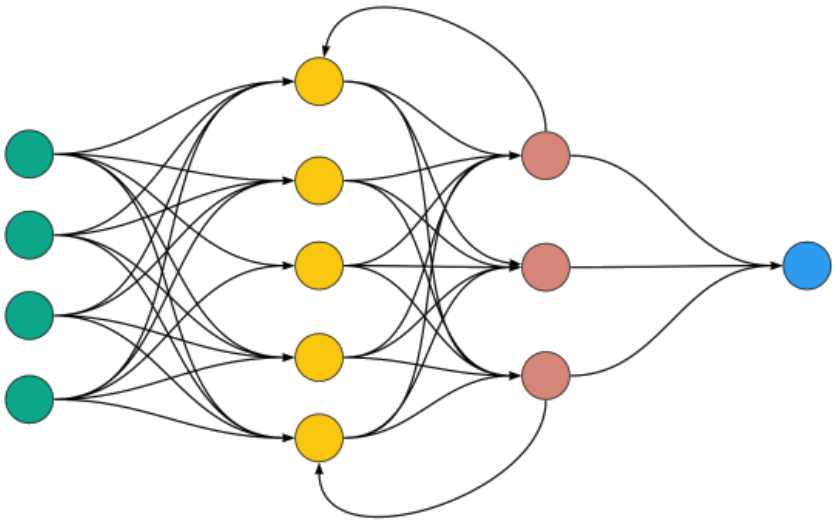


Figura 13. Arquitectura RNN

- **Redes Neuronales Convolucionales (CNN)**

Son un tipo de red neuronal que se utiliza principalmente para el procesamiento de imágenes y videos. En una red neuronal convolucional, las neuronas están organizadas en capas convolucionales, donde cada neurona está conectada solo a una región local de la capa anterior en lugar de a todas las neuronas de la capa anterior [17]. Esto permite que la red neuronal convolucional detecte características específicas en una imagen, como bordes y patrones, independientemente de su ubicación en la imagen.

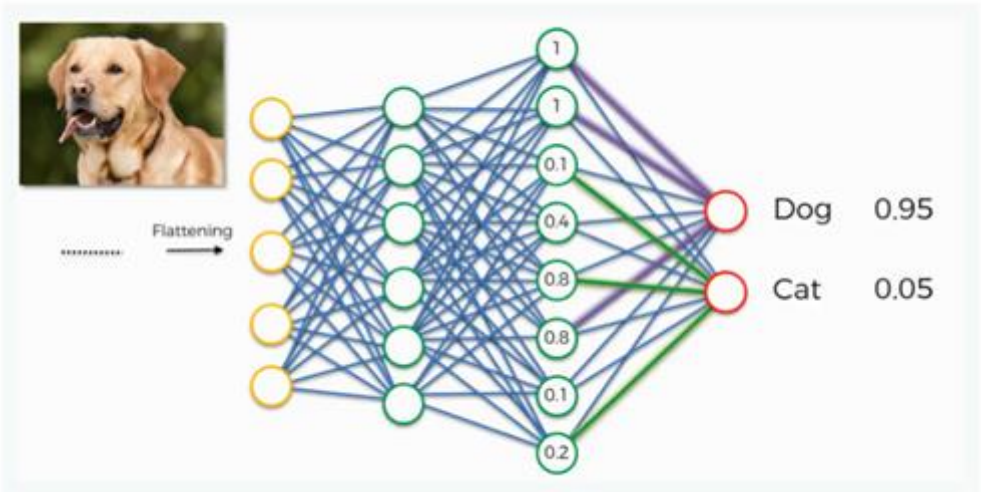


Figura 14. Ejemplo CNN

Sin embargo, las RNNs presentaban dos limitaciones críticas que obstaculizaban el progreso en el campo del NLP:

1. Problema de la Dependencia a Largo Plazo: A pesar de su memoria, las RNNs tenían dificultades para conectar palabras que estaban muy separadas en un texto largo. La información de las primeras palabras se "diluía" a medida que la red procesaba la secuencia.
2. Falta de Paralelización: Su naturaleza secuencial (procesar una palabra tras otra) impedía aprovechar el hardware moderno (GPUs) diseñado para cálculos en paralelo.

Estas limitaciones, crearon una necesidad de investigación: se requería una nueva arquitectura que pudiera procesar todos los elementos de una secuencia simultáneamente y que fuera capaz de determinar la importancia relativa de cada palabra con respecto a todas las demás, sin importar su distancia. Esta necesidad sentó las bases para la arquitectura Transformer.

3.1.5 Transformers y arquitecturas de atención (*Attention is all you need*)

La arquitectura Transformer, introducida en el influyente artículo "Attention is All You Need" (Vaswani et al., 2017),[18] representa un cambio en el NLP. Resolvió las limitaciones de las arquitecturas recurrentes (RNNs) al eliminar por completo la necesidad de procesamiento secuencial. En su lugar, introdujo el mecanismo de atención, que permite al modelo procesar todos los elementos de una secuencia simultáneamente y ponderar dinámicamente la influencia de cada palabra sobre las demás, capaces de capturar dependencias de largo alcance en las secuencias de texto.

Arquitectura General del Transformer

El modelo original del Transformer consta de dos componentes principales: un Codificador (*Encoder*) y un Decodificador (*Decoder*), pero utiliza exclusivamente los mecanismos de atención mencionados en lugar de recurrencia o convolución.

Cada uno de estos bloques contiene múltiples capas que procesan la información en paralelo y de forma eficiente.

- **El Codificador (*Encoder*)**: Su función es leer la secuencia de entrada completa y construir una representación numérica rica en contexto para cada token.[19]
- **El Decodificador (*Decoder*)**: Su función es generar la secuencia de salida, un token a la vez. En cada paso, no solo considera los tokens que ya ha generado, sino que también "presta atención" a las representaciones del codificador para asegurarse de que la salida sea coherente con la entrada.

Tanto el encoder como el decoder están compuestos por capas apiladas que contienen:

- **Capa de Multi-head Self-Attention.** Permite que cada token en la entrada preste atención a todos los demás tokens, capturando dependencias globales.
- **Red Feed-Forward (FFN).** Una red neuronal completamente conectada.

Estas sub-capas están conectadas mediante conexiones residuales y de normalización de capas (*Layer Normalization*) para estabilizar y facilitar el entrenamiento de redes profundas.

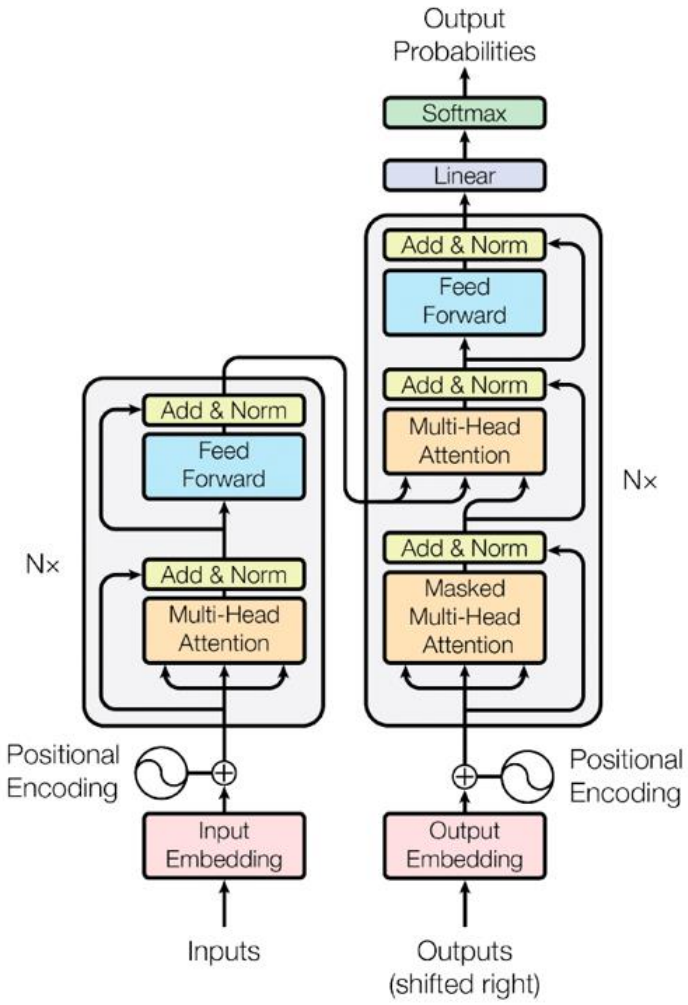


Figura 15. Arquitectura Transformer

Mecanismo de Atención (*Self Attention*)

La innovación clave del Transformer es el mecanismo de auto-atención. Permite que, al procesar un token, el modelo evalúe la importancia de todos los demás tokens de la misma secuencia y construya una nueva representación para el token actual basada en este contexto global.
Conceptualmente, el mecanismo funciona de manera análoga a una consulta en una base de datos. Para cada token de entrada, se generan tres vectores distintos apartir de su embedding:

- **Query (Q):** Representa la "pregunta" que el token actual está haciendo sobre los demás.
- **Key (K):** Representa la "etiqueta" o la naturaleza de cada uno de los otros tokens.
- **Value (V):** Representa el "contenido" o la información real de cada uno de los otros tokens.

El modelo calcula una puntuación de atención comparando el vector Query del token actual con el vector Key de todos los demás tokens. Estas puntuaciones se normalizan (usando una función softmax) para que sumen 1, convirtiéndose en pesos que indican cuánta "atención" se debe prestar a cada token.
Este proceso se realiza en paralelo para cada token, y mediante la técnica de **Atención Multi-cabeza (Multi-Head Attention)**, se repite varias veces con diferentes proyecciones lienales de Q, K y V. Esto permite al modelo capturar distintos tipos de relaciones (sintácticas, semánticas, etc.) simultáneamente.
Un ejemplo práctico para ilustrar este mecanismo: Consideremos la oración "El perro persigue al gato". Al aplicar mecanismo de atención:

- La palabra "perro" puede prestar más atención a "persigue" para entender que es el sujeto de la acción.
- "Gato" puede prestar atención a "al" y "persigue" para captar su papel como objeto.
- "El" puede enfocarse en "perro" para contextualizar su función como artículo definido.

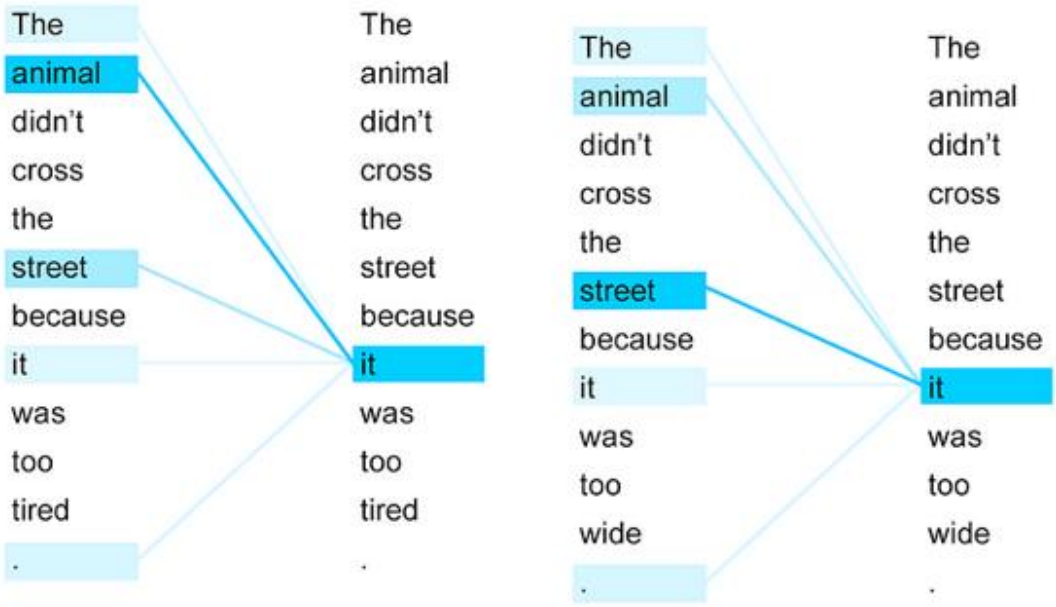


Figura 16. Ejemplo práctico del Mecanismo de Atención

3.2 Large Language Models (LLM)

La evolución del NLP ha alcanzado un punto culminante con el desarrollo de los **Grandes Modelos de Lenguaje** o ***Large Language Models*** **(LLM)**.

Partiendo de la arquitectura Transformer, el principio fundamental que permitió su desarrollo es la "Ley de Escala": al aumentar drásticamente el número de parámetros de un modelo Transformer y el volumen de sus datos de entrenamiento, no solo mejora su rendimiento en tareas existentes, sino que surgen capacidades completamente nuevas que no estaban presentes en modelos más pequeños [20]. Estos modelos a gran escala

pueden incorporar cantidades masivas de datos, a menudo de Internet, pero también de fuentes como Common Crawl, que comprende más de 50 000 millones de páginas web, y Wikipedia, que tiene aproximadamente 57 millones de páginas.

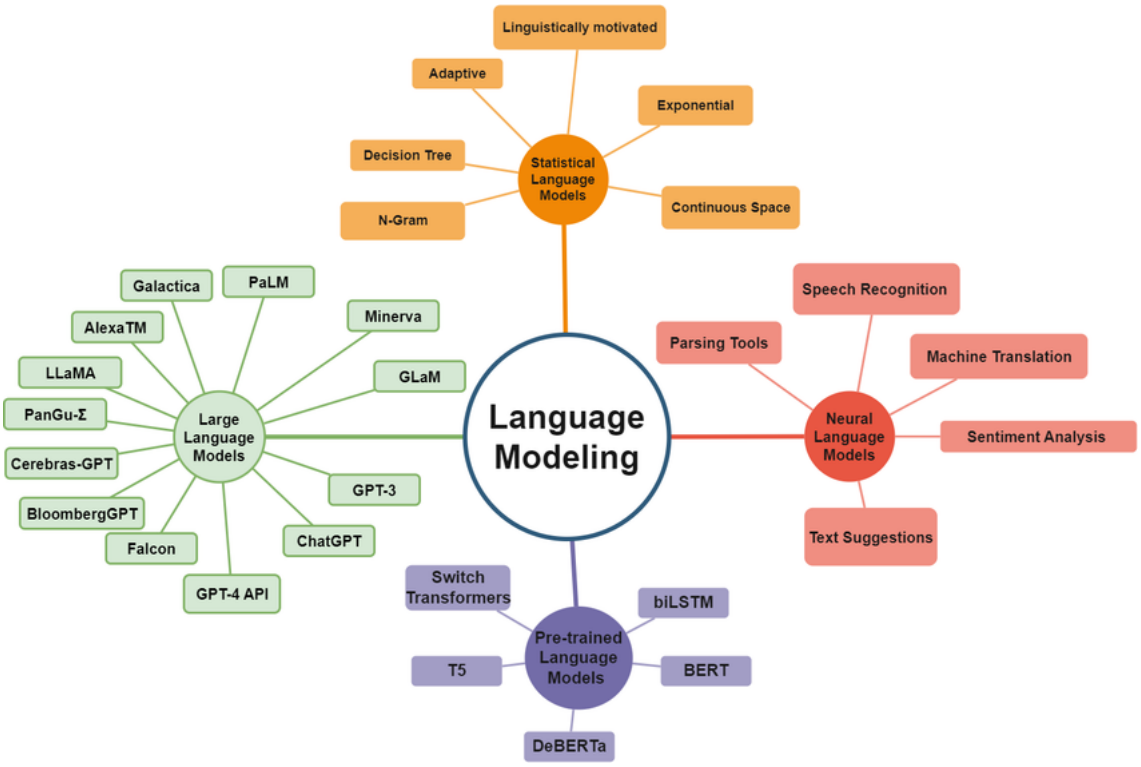


Figura 17. Tipos de modelado del lenguaje.[21]

3.2.1 Arquitecturas Fundamentales

A partir de la arquitectura Transformer original, la investigación se especializó en tres familias de modelos, cada una optimizada para un propósito diferente:

1. **Modelos Encoder-Only**
   - Utiliza solo el encoder del Transformer. Considera tanto el contexto izquierdo como el derecho de una palabra al generar su representación.
   - Ejemplo: BERT (*Bidirectional Encoder Representations from Transformers*).
   - Aplicaciones: Tareas de comprensión del lenguaje como clasificación, reconocimiento de entidades, y respuesta a preguntas. No es adecuado para generación de texto, ya que su arquitectura de encoder no está diseñada para producir secuencias de salida.
2. **Modelos Decoder-Only**
   - Utiliza solo la parte del decoder del Transformer. Utilizando una arquitectura autoregresiva que genera texto secuencialmente. El modelo predice la siguiente palabra considerando solo el contexto previo.

- Ejemplos: GPT (*Generative Pre-trained Transformer*), *LLaMA*, *Claude*.
- Aplicaciones: Generación de texto, completado de secuencias, y modelado de lenguaje autoregresivo.

3. **Modelos Encoder-Decoder**
   - Utilizan la arquitectura Transformer completa y son excelentes para tareas de transformación de secuencia a secuencia.
   - Ejemplo: T5 (*Text-to-Text Transfer Transformer*) y BART (*Bidirectional and Auto-Regressive Transformers*).
   - Aplicaciones: Traducción, resumen, conversión de texto a texto en diversas tareas.

Para el proyecto DocuBot, el objetivo principal es generar respuestas conversacionales fluidas y detalladas. Por lo tanto, la elección natural y fundamental es el uso de un modelo de la familia solo-Decoder (GPT), que es el estado del arte para tareas generativas.

3.2.2 Cómo Funciona un LLM (GPT)

Un LLM como GPT (*Generative Pre-trained Transformer*) se basa en el decoder del Transformer, utilizando una arquitectura autoregresiva que genera texto secuencialmente. El modelo predice la siguiente palabra considerando solo el contexto previo. Su funcionamiento interno, simplificado, sigue estos pasos:

1. **Entrada Tokenizada:** El texto de entrada (el *prompt*) es convertido en una secuencia de tokens numéricos por un tokenizador, como se describió en la sección 3.1.2.
2. **Procesamiento por el Transformer:** La pila de capas del decoder procesa esta secuencia. Gracias al mecanismo de auto-atención, cada token obtiene una representación contextualizada que considera toda la secuencia previa.
3. **Predicción del Siguiente Token:** Al final del proceso, el modelo genera una distribución de probabilidad sobre todo su vocabulario. El token con la mayor probabilidad es seleccionado (o se realiza un muestreo) como el siguiente token de la secuencia.
4. **Proceso Auto-regresivo:** Este nuevo token se añade a la secuencia de entrada, y todo el proceso se repite para generar el siguiente token, y así sucesivamente, hasta que se produce un token de "parada" o se alcanza la longitud máxima.

La serie GPT de OpenAI utiliza un tokenizador donde 1 token se asigna a alrededor de 4 caracteres que es la medida que se utilizará a partir de ahora.

Por tanto, simplificando un LLM como GPT se puede ver como un modelo probabilístico que predecir cuál es el siguiente token más probable en una secuencia, basándose en los patrones aprendidos durante el entrenamiento.

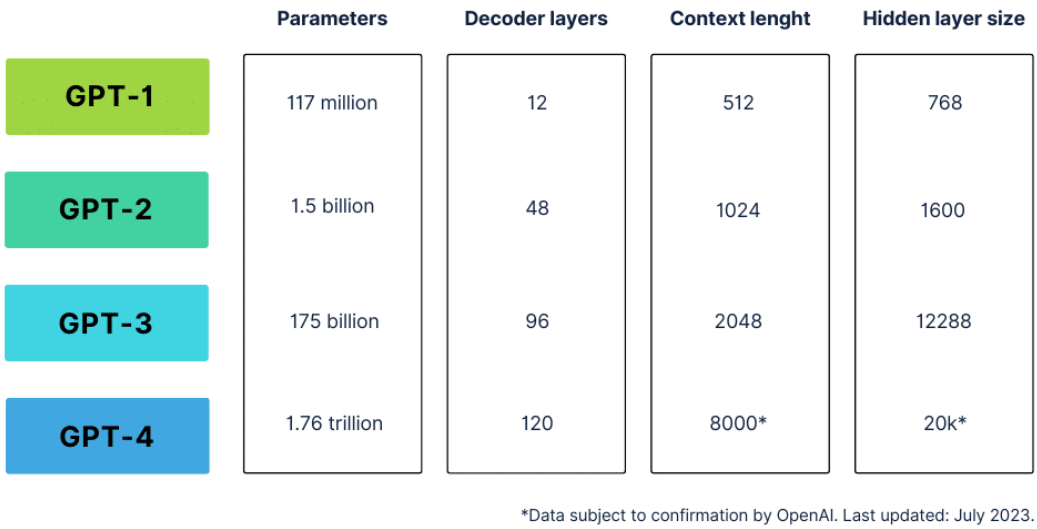


Figura 18. Evolución de los modelos GPT

3.2.3 Conceptos Clave de los LLMs

Más allá de su arquitectura, la aplicación efectiva de un LLM en un proyecto real como este depende de la comprensión de varios conceptos operativos.

Ventana de Contexto (Context Window)

El tamaño de contexto en un LLM es la cantidad máxima de tokens que el modelo puede "recordar" y utilizar al generar una respuesta o realizar una tarea. Este concepto es crucial porque define el alcance de la "memoria" activa del modelo: cuántas palabras o frases recientes puede procesar en cada interacción para tenerla en cuenta a la hora de generar una salida.

Tener en cuenta que este tamaño de contexto es temporal, es decir una vez que el tamaño de contexto se agota, el modelo no puede acceder a tokens anteriores que están fuera de esta ventana y recuerda por tanto solo los más cercanos a la interacción actual.

Prompts

Los prompts son instrucciones o entradas que se proporcionan a un LLM para guiar su generación de texto hacia una respuesta específica o para realizar una tarea particular. El prompting es, por tanto, el proceso de diseñar estas entradas de manera eficaz para obtener el output deseado del modelo.
El LLM genera su salida basándose en el prompt proporcionado, que puede incluir preguntas, indicaciones, contexto o ejemplos del tipo de respuesta que se espera. Debido

a que los LLM son modelos estadísticos entrenados en grandes cantidades de datos textuales, la formulación del prompt influye significativamente en la calidad y relevancia de la respuesta generada.

Multimodalidad

La multimodalidad en LLMs la capacidad de entender y generar información en múltiples formatos, como texto, imágenes y audio.[22] A diferencia de los modelos de lenguaje tradicionales que se limitan a datos textuales tanto de entrada y salida o incluso algunos modelos que han sido entrenados específicamente para ingerir una modalidad y producir otra modalidad diferente (modelos image to text, text to image…), los LLMs multimodales poseen la capacidad de procesar e interpretar el mundo a través de varias entradas sensoriales, al igual que los humanos. Los primeros modelos de este tipo CLIP y DALL-E.

Al combinar datos de diferentes modalidades, estos modelos obtienen una comprensión más profunda del contexto y el significado. Por ejemplo, un LLM multimodal podría analizar un informe no solo entendiendo el texto, sino también interpretando las imágenes asociadas, gráficos... proporcionado así una compresión completa de cada documento que será clave en nuestra aplicación.

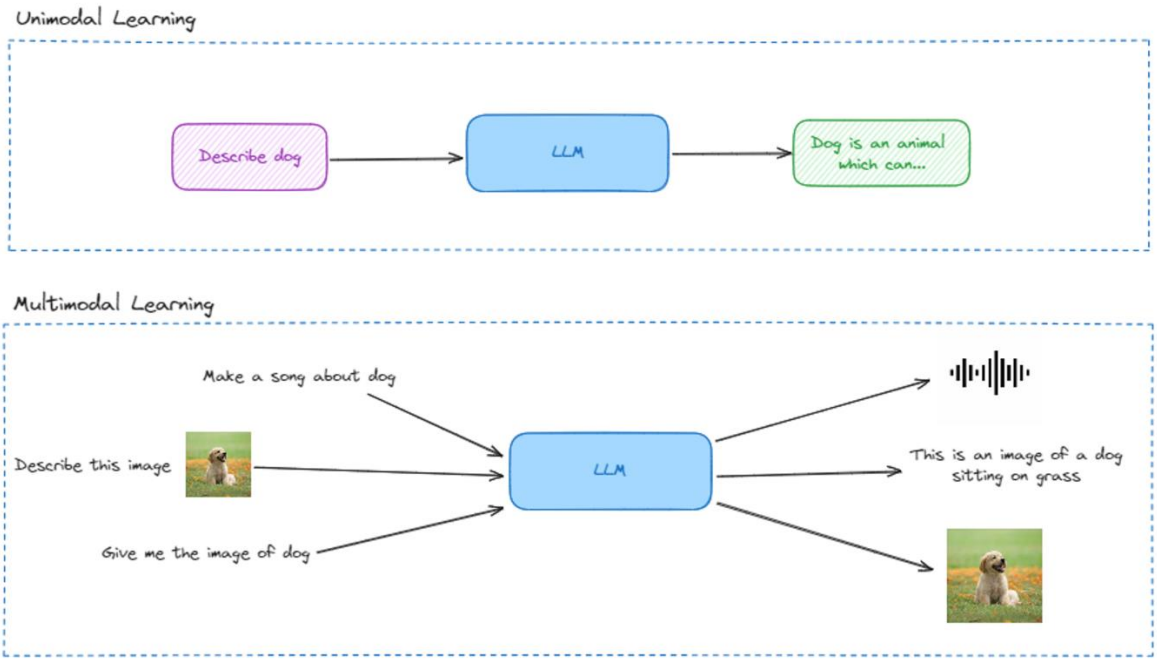


Figura 19. Funcionamiento modelo multimodal

3.2.4 LLMs en la Actualidad

Comprender los fundamentos de los LLMs es solo una parte de la ecuación. Para un proyecto práctico, es igualmente crucial analizar el ecosistema actual y tomar una

decisión informada sobre qué modelo y plataforma utilizar. El panorama actual se puede analizar a través de estos puntos [23].

- **Modelos con gran cantidad de parámetros**. Modelos cada vez más potentes entrenados con mayor cantidad de datos y por tanto con mayor conocimiento sobre la mayoría de los temas, estando ya en el orden de miles de billones o trillones de parámetros.
- **Modelos más pequeños especializados.** No siempre es recomendable según que tareas utilizar los modelos con más parámetros ya sea por su coste de uso, su alto coste de fine-tuning o su relativo poco conocimiento sobre nichos específicos de conocimiento. Para ello se están extendiendo los **Small Language Models** (SLM) modelos con menor número de parámetros del orden de pocos billones que pueden entrenarse para tareas muy específicas o dominios reducidos, pudiendo incluso desplegarse en una máquina en local sin necesidad de utilizar grandes clústeres de GPUs [24].
- **Modelos Open Source y de pago.** Gran variedad de modelos dado el auge en los últimos años del sector proporcionando numerosos modelos tanto modelos Open Source que puedes descargar y utilizar como modelos licenciados o de consumo por uso.
- **Aumento del tamaño de contexto.** Se ha ido aumentando rápidamente el tamaño de contexto o memoria de estos modelos lo que ha permitido mantener conversaciones más largas y poder pasar más información a los LLM en cada interacción, llegando incluso en los últimos modelos a poder pasar la Biblia como contexto, el orden de magnitud está ahora en 1-2 millones de tokens en los últimos LLMs de Google.
- **Modelos Multilingües**: Se han creado LLMs capaces de entender y generar texto en múltiples idiomas, ampliando su accesibilidad y utilidad a nivel global.
- **Integración Multimodal**: Como se ha visto antes combinado procesamiento de lenguaje y visión, permitiendo generar y comprender imágenes a partir de descripciones textuales. Modelos de audio y vídeo también han sido los mayores avances en los últimos meses.
- **Optimización de Recursos**: Se han desarrollado técnicas para reducir los requerimientos computacionales, como la destilación de conocimiento y cuantización, haciendo que LLMs sean más accesibles para organizaciones con recursos limitados.
- **Infraestructura en la Nube**: El acceso a LLMs a través de servicios en la nube ha democratizado su uso, permitiendo a empresas y desarrolladores integrar estas capacidades sin necesidad de infraestructura propia (Disponibilidad de los modelos a través de APIs)

Los LLM más potentes son proporcionados por empresas exclusivamente dedicadas a la Inteligencia Artificial Generativa (OpenAI, Claude) así como grandes tecnológicas (Meta, Google) cada una de estas empresas proporcionando sus propios modelos

basados en las diferentes arquitecturas vistas y cada uno con sus fortalezas y debilidades.

Vayamos con un repaso de las empresas más relevantes en este campo, para poder mencionar en que punto de madurez está cada una, que modelos ofrecen hoy en día y poder argumentar la elección de LLM que se hace en DocuBot.

OpenAI

OpenAI, liderado por Sam Altman, se consolidó como el primer líder y referente del sector tras el lanzamiento viral de ChatGPT. Su estrategia se centra en una búsqueda incesante de la escala, operando bajo la premisa de que modelos más grandes, entrenados con más datos, desbloquean capacidades de razonamiento y generación superiores. Su familia de modelos **GPT (*Generative Pre-trained Transformer*)**, con **GPT-4** y su iteración más reciente e interesante para el proyecto: **GPT-4o**, representan siempre el estándar de última tecnología contra el cual se mide el resto de la industria. Estos modelos son reconocidos por su excepcional capacidad de razonamiento complejo, su fluidez en la generación de texto y sus avanzadas capacidades multimodales, que les permiten procesar e interpretar texto, imágenes, audio y vídeo de forma nativa. El modelo de negocio de OpenAI se basa tanto en el modelo de suscripción para clientes finales como en el acceso a través de APIs, lo que les permite servir tanto al mercado de consumo como al empresarial, mientras que su alianza estratégica con Microsoft les proporciona la vasta infraestructura de supercomputación necesaria para entrenar y servir estos modelos masivos a través de Azure.

Antrophic

Anthropic, fundada por antiguos miembros de OpenAI, ha emergido como un competidor formidable al posicionarse con un fuerte énfasis en la seguridad, la ética y la creación de una IA "fiable y gobernable". Su familia de modelos **Claude**, cuya versión más potente es **Claude 3**, compite directamente en rendimiento con los mejores modelos de OpenAI y Google.

Google

Con un legado de investigación fundamental en IA, todo ello a partir del laboratorio inglés Google DeepMind, que incluye la propia invención de la arquitectura Transformer, Google ha posicionado la IA generativa como el núcleo de su estrategia futura. Su enfoque no es solo crear modelos potentes, sino integrarlos de manera profunda y sinérgica en todo su ecosistema de productos, así como convertirlos en aquellos con mayor tamaño de contexto del mercado. La familia de modelos **Gemini (Gemini 1.0 y 1.5)** fue diseñada desde su concepción para ser nativamente multimodal. En particular, **Gemini 1.5 Pro** ha causado un gran impacto gracias a su ventana de contexto masiva, capaz de procesar hasta 1 millón de tokens. Esto le permite analizar y razonar sobre volúmenes de información sin precedentes en un solo *prompt*, como libros

enteros o extensas bases de código. Estratégicamente, Google busca que Gemini potencie la Búsqueda, la suite de productividad Workspace y su buscador, de manera crucial, su plataforma en la nube Google Cloud (*Vertex AI*), compitiendo directamente por ser el gran proveedor empresarial.

Open Source: Meta y Mistral

En contraposición al modelo de negocio de los sistemas propietarios, existe un vibrante ecosistema de código abierto que está acelerando la democratización de la IA. **Meta** ha sido un actor clave en este movimiento con la liberación de su familia de modelos **LLaMA**. Al publicar los pesos de sus modelos, ha permitido que una comunidad global de desarrolladores, investigadores y empresas puedan experimentar, personalizar y construir sobre una base de alto rendimiento. Por otro lado, la startup europea **Mistral AI** ha ganado una enorme notoriedad al enfocarse en la eficiencia. En general, el movimiento de código abierto ofrece una alternativa centrada en la flexibilidad, el control y la eficiencia, impulsando la innovación fuera de los grandes laboratorios de IA.

Modelos Avanzados: "Razonamiento"

Es importante señalar que la investigación actual, año 2025, en IA generativa ya está avanzando más allá de los modelos que se han explicado en la sección superior. En los últimos meses han surgido arquitecturas de razonamiento [25]. Estos modelos como la serie *omini* de OpenAI o los *Deep Thinking* de Google, no están orientados a únicamente responder a una pregunta, sino que son capaces de descomponer un objetivo complejo en una secuencia de pasos, utilizar herramientas (como buscar en la web o ejecutar código) y planificar acciones para alcanzar dicho objetivo. Dado que el propósito central de este proyecto es perfeccionar la arquitectura RAG para proporcionar respuestas fiables basadas en un corpus documental, el análisis en detalle de estos modelos de razonamiento y agente, aunque fascinante, queda fuera del alcance de la evaluación tecnológica.

3.2.5 Limitaciones

A pesar de haber comentado en detalle sus capacidades, que son tremendas, los LLMs convencionales y de base presentan desafíos y limitaciones inherentes que son críticos de abordar, especialmente en un contexto empresarial como el de este proyecto:

- **Alucinaciones:** La limitación más conocida es la tendencia de los LLMs a "alucinar", es decir, en su esfuerzo por proporcionar una respuesta acaban generando información que es incorrecta, inventada o que no está respaldada por ninguna fuente, pero presentada con un tono de total confianza. Ocurre porque los modelos están diseñados para predecir la siguiente palabra en una secuencia,

no para verificar la veracidad de la información que generan. Esto es inaceptable en un entorno donde la precisión de la información es primordial.

- **Conocimiento Desactualizado:** Una de las mayores restricciones de los LLMs es que su conocimiento del mundo se congela en el momento en que finaliza su entrenamiento. No tienen acceso a información en tiempo real, lo que significa que no pueden responder a preguntas sobre eventos recientes o datos que han cambiado desde su última actualización. Por ejemplo, un LLM podría no conocer los resultados de un evento deportivo reciente o las últimas noticias económicas.
- **Falta de Conocimiento Específico de Dominio:** Los LLMs se entrenan con vastos conjuntos de datos de internet, lo que les proporciona un conocimiento general amplio. Sin embargo, carecen de la profundidad y la especificidad necesarias para dominios especializados como la jerga interna de una empresa, documentación técnica compleja o bases de datos propietarias. Para nuestro caso de uso esta es la limitación principal ya que los modelos no han sido entrenados con documentación específica de una empresa o documentación que no se encuentre pública en Internet.
- **Falta de Transparencia y Trazabilidad:** Cuando un LLM da una respuesta, es inherentemente una "caja negra". No proporciona las fuentes de su información, lo que hace imposible que un usuario verifique la veracidad de los datos o profundice en el contexto original.
- **Sesgos:** Al ser entrenados con vastas cantidades de texto de Internet, los LLMs pueden heredar y amplificar sesgos presentes en los datos (culturales, de género, etc.), lo que puede llevar a respuestas inapropiadas o injustas.
- **Seguridad y Privacidad:** Enviar datos sensibles de la empresa a una API de un LLM de terceros plantea preocupaciones significativas sobre la privacidad y la seguridad de los datos si no se gestiona a través de una plataforma empresarial segura como Azure.

3.2.6 Estrategias de Mejora

Ante estas limitaciones, surge la necesidad de adaptar o "especializar" un LLM de propósito general para un dominio o tarea específica, asegurando que sus respuestas sean fiables, precisas y seguras. Existen varias estrategias para lograr esto:

- **Ingeniería de Prompts (*Prompt Engineering*):** Es la técnica más básica. Consiste en proporcionar instrucciones y contexto detallado directamente en el prompt para guiar al modelo [26]. Aunque es útil, su capacidad para inyectar conocimiento a gran escala es limitada por la ventana de contexto, y el coste asociado que conllevaría inyectar en cada interacción todo el contexto disponible.

- **Ajuste Fino (*Fine-Tuning*):** Esta es una técnica avanzada que implica un re-entrenamiento del LLM con un *dataset* más pequeño y específico del dominio. El proceso ajusta los últimos pesos internos del modelo para que se especialice en el estilo, el tono y el conocimiento de los nuevos datos.
    - **Ventajas:** El conocimiento se integra directamente en el modelo, y puede ser muy eficaz para adaptar el comportamiento del LLM.
    - **Inconvenientes:** El *fine-tuning* sigue siendo un proceso muy **costoso y complejo** que requiere un dataset abundante, de alta calidad, experiencia técnica y recursos computacionales significativos. Además, no resuelve por completo el problema de las alucinaciones y hace que sea difícil actualizar el conocimiento del modelo (requiere un nuevo proceso de fine-tuning). Tampoco soluciona el problema de la trazabilidad, ya que el modelo sigue sin poder citar sus fuentes.

Dadas las desventajas del fine-tuning, especialmente la dificultad para mantener el conocimiento actualizado y la falta de trazabilidad, la industria ha ido centrándose en una tercera estrategia que se ha convertido en el estándar para aplicaciones empresariales basadas en conocimiento y sobre la que se basa todo este proyecto.

3.3 Arquitecturas RAG

Habiendo explorado ya como se construyen, todas las capacidades y debilidades de los LLMs se hace evidente que para aplicaciones empresariales se requiere una arquitectura que garantice la fiabilidad, actualidad y trazabilidad de la información. La Generación Aumentada por Recuperación (RAG, por sus siglas en ingles *Retrieval Augmented Generation*) representa un cambio de paradigma fundamental para todos los desafíos comentados. En lugar de depender únicamente de su memoria interna, un sistema RAG funciona más como un bibliotecario experto: no memoriza cada libro, sino que sabe exactamente dónde y cómo buscar la información más relevante para responder a una pregunta específica.

3.3.1 Definición y Principio Fundamental

La arquitectura RAG es un cambio en el diseño de sistemas de IA que mejora la calidad de las respuestas de un LLM al aumentar su contexto con información recuperada de una fuente de conocimiento externa y relevante.

El principio fundamental de RAG es la separación conceptual entre el conocimiento y la capacidad de razonamiento. En lugar de depender únicamente del conocimiento intrínseco del modelo por el entrenamiento (la información codificada en los pesos del modelo durante su entrenamiento), un sistema RAG aprovecha:

1. **Un "cerebro" de razonamiento:** El LLM, que actúa como un motor de lenguaje experto en comprender, sintetizar y generar texto de forma fluida y coherente.
2. **Una "biblioteca" de conocimiento externa:** Una base de datos o un índice de documentos (el *corpus*) que contiene la información específica, actualizada y verídica que el sistema debe utilizar.

Este proceso se puede visualizar en un flujo de tres pasos de alto nivel, como se ilustra en la siguiente Figura:

1. **Recuperación (*Retrieve*):** Ante una consulta del usuario, el sistema no acude inmediatamente al LLM. Primero, utiliza un módulo de recuperación para buscar en la biblioteca de conocimiento externa y obtener los fragmentos de información más relevantes para la consulta.
2. **Aumento (*Augmented*):** Los fragmentos de información recuperados se combinan con la pregunta original del usuario para construir un *prompt* enriquecido o aumentado. Este prompt proporciona al LLM todo el contexto necesario para formular una respuesta precisa.
3. **Generación (*Generation*):** El LLM recibe este prompt aumentado y genera una respuesta en lenguaje natural. Crucialmente, su tarea no es recordar información, sino sintetizar la que se le ha proporcionado en el contexto. Esto ancla con un prompt especializado para dicha tarea la respuesta a las fuentes autorizadas, reduciendo drásticamente las alucinaciones.

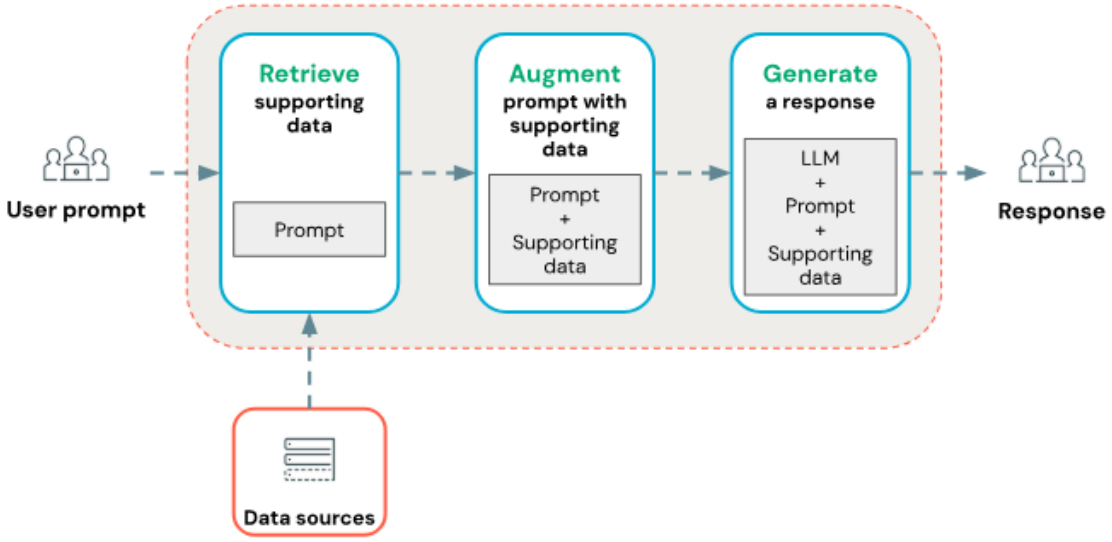


Figura 20. Arquitectura RAG básica

Al adoptar este enfoque, la arquitectura RAG no solo mitiga las limitaciones de los LLMs, sino que introduce nuevas capacidades. La más importante es la trazabilidad, ya que el sistema es consciente de qué fragmentos de información utilizó para generar su respuesta, lo que abre la puerta a citar fuentes y permitir la verificación por parte del usuario, un pilar fundamental para la confianza en sistemas de IA empresariales.

3.3.2 El Componente de Recuperación (*Retrieval*)

La premisa "basura entra, basura sale" (*Garbage In, Garbage Out*) es especialmente cierta en este contexto: un LLM, por muy potente que sea, no puede generar una respuesta precisa si se le proporciona información irrelevante o incorrecta. Por lo tanto, la eficacia de todo el sistema depende fundamentalmente de la capacidad de este módulo de recuperación para encontrar los fragmentos de conocimiento más relevantes para la consulta del usuario.

El diseño de un sistema de recuperación robusto implica tomar decisiones clave sobre dos aspectos: la estrategia de búsqueda y la estrategia de fragmentación de los datos.

Los documentos de una base de conocimiento (PDFs, documentos de Word, etc.) suelen ser demasiado largos para ser introducidos directamente en la ventana de contexto de un LLM o de un modelo de embeddings. Por ello, es necesario un paso de preprocesamiento conocido como chunking, que consiste en dividir los documentos originales en fragmentos más pequeños y manejables [27].

La estrategia de chunking es una decisión de diseño crítica, ya que un *chunk* mal definido puede cortar una idea a la mitad o carecer del contexto necesario para ser útil. Algunas estrategias comunes incluyen:

- **Fragmentación de Tamaño Fijo:** El método más simple, que divide el texto en fragmentos de un número fijo de caracteres o tokens.
- **Fragmentación con Solapamiento (*Overlap*):** Una mejora sobre el anterior, donde los fragmentos consecutivos comparten una pequeña porción de texto. Esto ayuda a preservar la continuidad semántica entre fragmentos.

La forma en que el sistema busca en la "biblioteca" de conocimiento determina qué documentos considera relevantes. Las estrategias principales son:

- **Búsqueda por Palabras Clave (*Keyword Search*):** Es el enfoque tradicional, basado en algoritmos como BM25, que buscan coincidencias exactas de las palabras de la consulta en los documentos [28]. Es muy eficaz para encontrar documentos que contienen términos específicos, acrónimos o códigos (p. ej., "procedimiento ISO-9001"). Sin embargo, fracasa cuando el usuario utiliza sinónimos o describe un concepto sin usar las palabras clave exactas presentes en el documento.
- **Búsqueda Semántica o Vectorial (*Semantic Search o Vector Search*):** Este enfoque, que ya se introdujo en la sección 3.1.3, supera las limitaciones de la búsqueda por palabras clave. Convierte tanto la consulta del usuario como los fragmentos de documentos en *embeddings* (vectores numéricos que representan el significado). Luego, en lugar de buscar palabras, busca los vectores de documentos más cercanos al vector de la consulta en un espacio multidimensional. Esto permite encontrar fragmentos que son **conceptualmente relevantes**, aunque no compartan ninguna palabra clave.

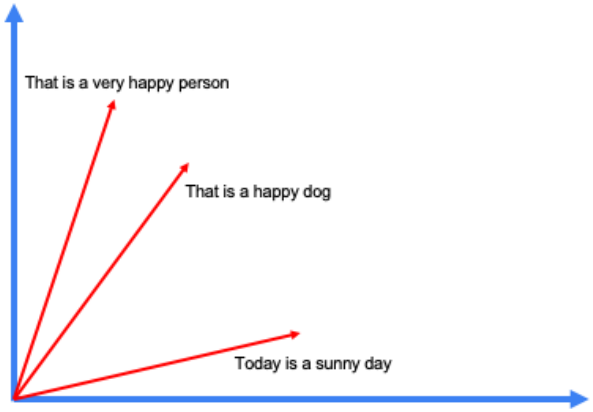


Figura 21. Ejemplo búsqueda vectorial

- **Búsqueda Híbrida (*Hybrid Search*):** Reconociendo que ambos enfoques tienen fortalezas complementarias, la búsqueda híbrida se ha convertido en la estrategia de vanguardia para los sistemas RAG.[29] Combina los resultados de una búsqueda por palabras clave y una búsqueda semántica, y luego utiliza un algoritmo de fusión y re-ranking (como *Reciprocal Rank Fusion* - RRF) para producir una lista final de resultados. Este método aprovecha la precisión de la búsqueda por palabras clave para términos específicos y la comprensión conceptual de la búsqueda semántica, ofreciendo la mayor robustez y relevancia.

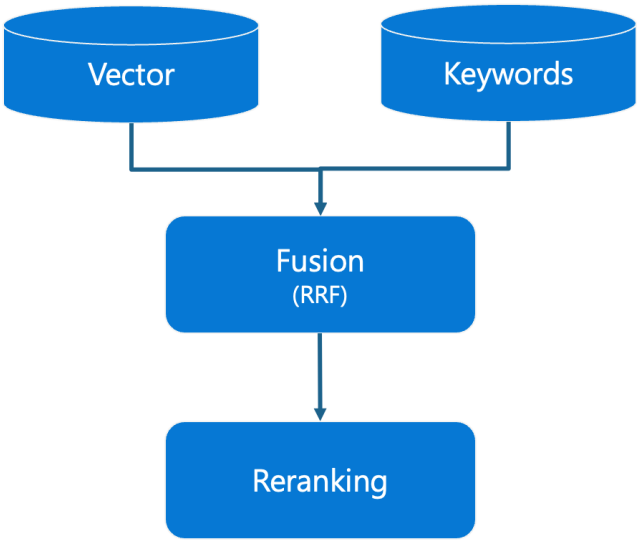


Figura 22. Flujo de búsqueda híbrida

La elección de una estrategia de fragmentación y un tamaño de *chunk* adecuados es un equilibrio entre proporcionar suficiente contexto en cada fragmento y mantenerlos lo suficientemente concisos para que el sistema de recuperación pueda identificar con precisión la información más relevante y para que quepan en el *prompt* final del LLM.

3.3.3 Trazabilidad y Citado de Fuentes

Una de las limitaciones más profundas de los LLMs en su forma pura es su naturaleza de "caja negra", que impide verificar el origen de la información que generan. La arquitectura RAG resuelve este problema de manera inherente, introduciendo el concepto de trazabilidad como una característica central del sistema. Esta capacidad no es un añadido, sino una consecuencia natural de su diseño, y constituye el pilar fundamental para construir confianza con el usuario.

La trazabilidad en un sistema RAG se refiere a la capacidad de rastrear el origen de cada afirmación en la respuesta generada hasta los fragmentos específicos de los documentos fuente que la respaldan. Esto es posible porque, a diferencia de un LLM estándar, el sistema RAG es consciente del contexto exacto que ha proporcionado al modelo para generar la respuesta.

Este proceso de atribución permite generar referencias precisas, como los números de cita que se ven comúnmente en los sistemas de Gen AI más avanzados y profesionales.

3.3.4 Ventajas Fundamentales de una Arquitectura RAG

A lo largo de esta sección, hemos desglosado los componentes y principios de la arquitectura RAG. A modo de conclusión, es útil sintetizar las ventajas fundamentales que hacen de RAG el enfoque preeminente para construir aplicaciones de IA fiables y basadas en conocimiento, y que justifican su elección como pilar central de este proyecto [30].

1. **Reducción Drástica de Alucinaciones y Aumento de la Fiabilidad:**
   Al anclar las respuestas del LLM a un conjunto de documentos verificables proporcionados en tiempo real, RAG minimiza la dependencia del modelo en su conocimiento interno.
2. **Conocimiento Dinámico y Siempre Actualizado:**
   A diferencia del fine-tuning, que requiere costosos procesos de re-entrenamiento para actualizar el conocimiento del modelo, la base de conocimiento de un sistema RAG es externa y desacoplada. Para actualizar la información del sistema, basta con añadir, modificar o eliminar documentos en la base de datos de recuperación.
3. **Transparencia y Trazabilidad Completa:**
   RAG transforma la "caja negra" de un LLM en un sistema transparente. Al ser consciente de los fragmentos de información utilizados para cada respuesta, el sistema puede citar sus fuentes y garantiza responder únicamente con dicha información.

4. **Eficiencia en Costes y Desarrollo:**
   Implementar una arquitectura RAG es, en la mayoría de los casos, significativamente más rápido y económico que entrenar un modelo desde cero o realizar un fine-tuning a gran escala.
5. **Control y Seguridad Granular:**
   Al separar la lógica de recuperación del LLM, se abre la puerta a implementar capas de control de acceso granulares. Es posible, por ejemplo, filtrar los documentos que el sistema de recuperación puede "ver" basándose en los permisos del usuario que realiza la consulta. Esto asegura que los usuarios solo reciban respuestas basadas en la información a la que están autorizados a acceder, una característica de seguridad indispensable en un entorno corporativo.

3.4 De RAG Avanzados hacia Agentes

La arquitectura RAG básica, como se ha descrito, resuelve las limitaciones fundamentales de los LLMs. Sin embargo, para construir sistemas verdaderamente robustos, eficientes y capaces de abordar consultas complejas, la comunidad de IA ha ido desarrollando en los últimos meses un conjunto de técnicas avanzadas que optimizan cada etapa del proceso. Este capítulo explora estas optimizaciones y traza el camino hacia el siguiente paradigma: los Agentes de IA, donde RAG actúa como el sistema de memoria y conocimiento fundamental.

3.4.1 Técnicas Avanzadas de RAG

Los sistemas RAG tradicionales suelen enfrentar desafíos críticos. Problemas como la baja precisión en la recuperación, la falta de conocimiento contextual en las respuestas y las dificultades para abordar consultas complejas pueden limitar su eficacia e impedir su uso en contextos empresariales. La necesidad de técnicas avanzadas de RAG es particularmente evidente en su capacidad para mejorar la precisión y la fiabilidad de las respuestas [31].

A continuación, se detallan algunas de las técnicas más importantes que transforman un RAG básico en una solución de nivel empresarial.

1. Optimización Pre-Recuperación (*Pre-Retrieval*): Mejorando la Pregunta.

- **Transformación de la Consulta (*Query Transformation*):** A menudo, la pregunta original del usuario no es la ideal para la búsqueda. Técnicas como:
  - **HyDE (*Hypothetical Document Embeddings*):** Se le pide a un LLM que genere una respuesta hipotética a la pregunta. Luego, se busca el embedding de esa respuesta hipotética, que suele ser semánticamente más rico que el de la pregunta corta [32].

- **Step-Back Prompting:** Se le pide a un LLM que dé un paso atrás y formule una pregunta más general o conceptual a partir de la pregunta específica del usuario. Se buscan documentos para ambas preguntas (la original y la general) para obtener un contexto más amplio.

2. Optimización de la Recuperación (*Retrieval*): Encontrando la Información Correcta.

- **Búsqueda Híbrida (*Hybrid Search*):** Como se introdujo anteriormente, esta técnica es el estándar de oro actual. Combina la precisión léxica de la búsqueda por palabras clave (ej. BM25), ideal para acrónimos, códigos y términos específicos, con la comprensión conceptual de la búsqueda vectorial. Los resultados de ambas búsquedas se fusionan de forma inteligente utilizando algoritmos como Reciprocal Rank Fusion (RRF), que prioriza los documentos que aparecen en los primeros puestos de ambas listas, ofreciendo lo mejor de ambos mundos.
- **Re-ranking:** La primera fase de recuperación (retrieve) debe ser rápida, aunque sacrifique algo de precisión. Una segunda fase de re-ranking refina los resultados.
  - **Cross-Encoders:** A diferencia de los modelos de embeddings que procesan pregunta y documento por separado, los cross-encoders los procesan juntos, obteniendo una puntuación de relevancia mucho más precisa. Son computacionalmente caros, por lo que se aplican solo a los N mejores resultados de la búsqueda inicial [33].

3. Optimización Post-Recuperación (*Post-Retrieval*): Refinando el Contexto.

- **Compresión de Contexto (*Context Compression*):** Los documentos recuperados pueden contener mucha información irrelevante. Se puede usar un LLM más pequeño para extraer solo las frases directamente relevantes a la pregunta del usuario de los chunks recuperados, antes de enviarlos al LLM principal. Esto optimiza el uso de la ventana de contexto y reduce el ruido.
- **RAG Multi-Salto o Recursivo (*Multi-hop/Recursive RAG*):** Para preguntas complejas que requieren sintetizar información de múltiples fuentes ("Compara las ventajas del producto A según el informe técnico con las desventajas mencionadas en las reseñas de clientes"). El sistema realiza una primera búsqueda, y la respuesta inicial genera nuevas preguntas que inician nuevas búsquedas, construyendo la respuesta paso a paso [34].

4. Estrategias de Indexación Avanzadas.

- **Chunking Semántico (*Semantic Chunking*):** En lugar de dividir por tamaño fijo, usar modelos de lenguaje para dividir los documentos en fragmentos que representen ideas o proposiciones completas y coherentes [35].
- **Indexación Multi-Vectorial (*Multi-Vector Indexing*):** En lugar de un solo vector por chunk, generar varios: un vector para un resumen del chunk, un

vector para una pregunta hipotética que el chunk respondería, etc. Esto aumenta las posibilidades de una coincidencia relevante.

3.4.2 Agentes de IA y el Rol Fundamental de RAG

A modo de avance de los siguientes pasos hacia donde se va a avanzar en este campo de la IA Generativa, si la arquitectura RAG dota a un LLM de conocimiento fiable, los **Agentes de IA** le dotan de capacidad de acción. Un agente es un sistema que utiliza un LLM como su "cerebro" de razonamiento para descomponer un objetivo complejo en una secuencia de pasos, seleccionar y utilizar herramientas para ejecutar esos pasos, y observar los resultados para alcanzar el objetivo de forma autónoma [36].

En este paradigma emergente, el agente puede interactuar con múltiples sistemas: buscar en la web, ejecutar código, consultar una base de datos o interactuar con APIs de software. Sin embargo, para que un agente sea verdaderamente útil en un entorno corporativo, su capacidad más crítica es acceder y razonar sobre información privada, específica del dominio y actualizada.

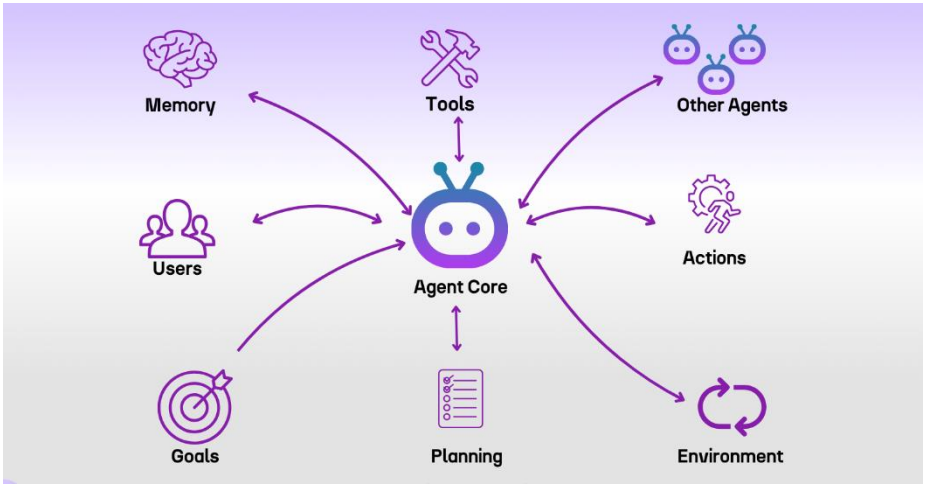


Figura 23. Sistemas de Agentes

Aquí es donde la arquitectura RAG va más allá su rol inicial. En lugar de ser el sistema completo, RAG se convierte en la herramienta de conocimiento fundamental del agente. Cuando un agente necesita responder a una pregunta como "¿Cuál es la política de la empresa sobre viajes internacionales?" o "Resume el feedback del cliente A sobre nuestro último lanzamiento", no depende de su conocimiento pre-entrenado, sino que invoca su herramienta RAG interna. Esta herramienta, conectada a la base de conocimiento corporativa (documentos de políticas, ERP, CRM, informes internos), recupera la información precisa y se la proporciona al "cerebro-orquestador" del agente para que pueda continuar con su plan de acción.

Por lo tanto, una arquitectura RAG robusta y avanzada, como la que se detalla en este proyecto, no debe verse como un fin en sí misma, sino como un prerrequisito esencial para la próxima generación de IA.

3.5 Tecnologías Utilizadas: Azure

En esta sección se presenta el *stack* tecnológico seleccionado para implementar los fundamentos teóricos descritos previamente. La elección de estas tecnologías no es arbitraria; responde a la necesidad de construir un sistema RAG empresarial, seguro, escalable y nativo del ecosistema de Microsoft, donde residirán los usuarios finales. Cada componente tecnológico ha sido escogido para desempeñar un rol específico dentro de la arquitectura de este proyecto

Para este proyecto, se ha optado por un ecosistema de servicios en la nube de **Microsoft Azure**.

Azure es la plataforma de computación en la nube de Microsoft, que ofrece un amplio catálogo de servicios integrados que abarcan desde el cómputo y el almacenamiento hasta la inteligencia artificial y el análisis de datos. Una de las principales ventajas de utilizar Azure es su modelo de **Plataforma como Servicio (PaaS)**. Los servicios PaaS, como los que se describirán a continuación, abstraen la complejidad de la infraestructura subyacente (servidores, sistemas operativos, redes), permitiendo a los desarrolladores centrarse exclusivamente en la lógica de la aplicación. Esto no solo acelera el desarrollo, sino que también garantiza alta disponibilidad, escalabilidad y seguridad gestionada por Microsoft, dotando al proyecto de una base tecnológica de calidad empresarial.

Para comprender la organización de los componentes que se describirán, es fundamental entender el modelo estructural de Azure. La plataforma organiza todos los servicios bajo una jerarquía lógica. En el nivel superior se encuentra la **Suscripción**, que funciona como un límite administrativo y de facturación. Dentro de una suscripción, los servicios se despliegan en **Grupos de Recursos** (*Resource Groups*). Un Grupo de Recursos es un contenedor lógico que agrupa todos los recursos relacionados de una solución para gestionarlos como una única entidad. Esto simplifica enormemente la administración del ciclo de vida (despliegue, actualización, eliminación), el control de acceso y el seguimiento de costes. Finalmente, cada servicio individual, como una base de datos o servicio de IA, es un **Recurso** que reside dentro de un grupo.

Todos los componentes tecnológicos de este proyecto se han consolidado dentro de un único Grupo de Recursos, garantizando así un entorno cohesivo y fácilmente gobernable.

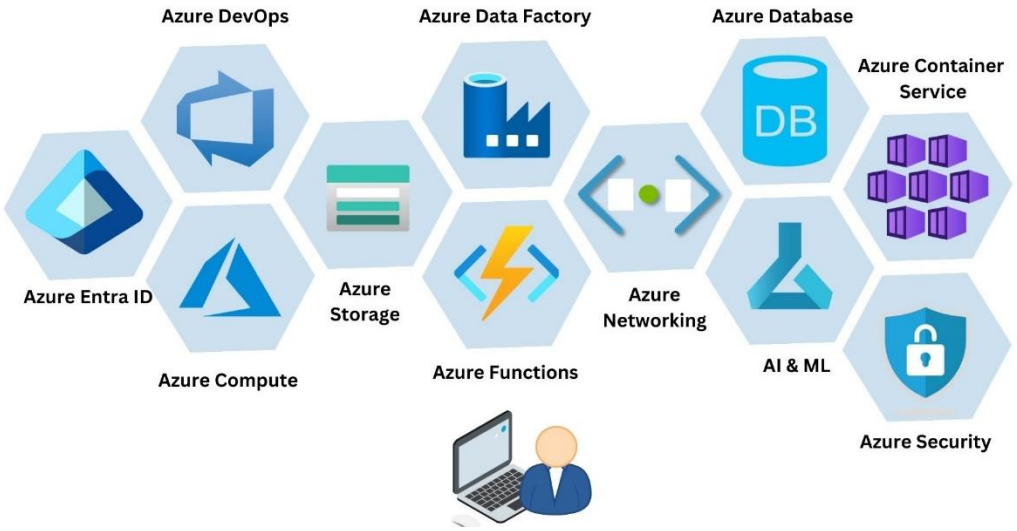


Figura 24. Top 10 Microsoft Azure Services más populares.

Este apartado servirá de base teórica para después comprender y glosar correctamente cuando se hable de los diferentes servicios a lo largo de la memoria, especialmente en el capítulo de Arquitectura Física que se abordará más adelante y da un porqué de que se hayan utilizado estas y no otras alternativas similares.

3.5.1. Azure App Service

Servicio PaaS para construir, desplegar y escalar aplicaciones web y APIs. Gestiona la infraestructura, permitiendo a los desarrolladores centrarse en el código.

- **Justificación:** Proporciona un entorno de ejecución robusto y gestionado para el código de DocuBot, manejando el escalado automático, la seguridad y la integración continua, lo cual es esencial para una aplicación de producción.

3.5.2. Azure Bot Service

Servicio que conecta el código de un bot con canales de comunicación como Microsoft Teams. Consumirá una instancia de las que se disponene con un Azure App Service.

- **Justificación:** Actúa como el puente indispensable y seguro que permite que el bot sea accesible desde Teams, traduciendo y enrutando la comunicación de manera fiable.

3.5.3. Azure OpenAI Service

Será la plataforma de modelos de IA Generativa. Azure OpenAI Service es un servicio que proporciona acceso a los modelos de lenguaje más avanzados de OpenAI (como la familia GPT y los modelos de embedding) a través de APIs REST. Este servicio opera dentro de la seguridad y disponibilidad regional de Microsoft Azure.

- **Justificación:** La principal razón para seleccionar Azure OpenAI Service, en lugar de acceder directamente a las APIs de otros proveedores de LLMs, es la seguridad de nivel empresarial. Al procesar información corporativa, es necesario que los datos no abandonen el entorno controlado de la organización. Azure OpenAI garantiza que tanto las consultas como los datos enviados al servicio se procesan dentro de la suscripción de Azure del cliente. Esta integración nativa con el resto del ecosistema Azure ofrece, además, una gestión unificada por una plataforma cloud consolidada.

3.5.4. Modelo GPT-4o

Será el LLM elegido en el aplicativo. GPT-4o es uno de los modelos de lenguaje de OpenAI surgido en verano de 2024 [37]. Es un modelo multimodal, diseñado para procesar y razonar de forma nativa sobre una combinación de entradas de texto, audio e imágenes, generando a su vez salidas en estos mismos formatos. Representa un avance significativo en la capacidad de los LLMs para comprender contextos complejos y variados.

- **Justificación:** Dentro del catálogo ofrecido por Azure OpenAI, se optó por el modelo GPT-4o por dos razones estratégicas clave:
    1. **Capacidad Multimodal:** La habilidad de GPT-4o para interpretar imágenes es fundamental para la fase de "Aumentación" del RAG, donde el contexto proporcionado al modelo puede incluir representaciones visuales de los documentos fuente. Esto permite una comprensión más profunda y precisa que la que se obtendría solo con texto.
    2. **Rendimiento y Razonamiento Avanzado:** Para un sistema RAG, donde el modelo debe analizar un gran volumen de texto recuperado y generar una respuesta coherente, precisa y bien fundamentada. Además de tener un tamaño de contexto considerable de 128.000 tokens de entrada, suficiente para DocuBot. Todo esto sin ser el modelo más costoso de los que dispone OpenAI que si bien son mejores, dispararían los gastos recurrentes de la aplicación.

3.5.5. Azure AI Search

Es el motor de búsqueda y base de datos vectorial. Azure AI Search es un servicio PaaS que ofrece capacidades avanzadas de búsqueda de información sobre contenido

heterogéneo. Aunque tradicionalmente se ha centrado en la búsqueda de texto completo, ha evolucionado para convertirse también en una potente **base de datos vectorial**, capaz de indexar y realizar búsquedas de similitud sobre embeddings de alta dimensionalidad.

- **Justificación:** Para la fase de "Recuperación" de un sistema RAG, se necesita un motor capaz de realizar búsquedas semánticas eficientes. Azure AI Search fue elegido sobre otras bases de datos vectoriales por su capacidad nativa de **búsqueda híbrida**, que combina la búsqueda vectorial (semántica) con la búsqueda léxica tradicional (palabras clave) [38].
- Para realizar esta extracción de los documentos de la base documental se necesita en particular una base de datos vectorial que permita almacenar esos documentos añadiendo un campo de embeddings para poder representar cada uno de esos documentos en un vector numérico de grandes dimensiones que represente toda la información semántica, y así poder obtener los documentos más similares a la pregunta del usuario según una medida, en este caso se ha decantado por la default como es la similitud coseno.
- **Modelo de Embeddings:** El modelo de embeddings elegido es el ***text-embedding-003-large*** de OpenAI, lanzado también en 2024 es el último modelo disponible, tiene más dimensiones (3072) que sus predecesores y mejora todas las métricas comparados con sus antecesores y competidores [39].

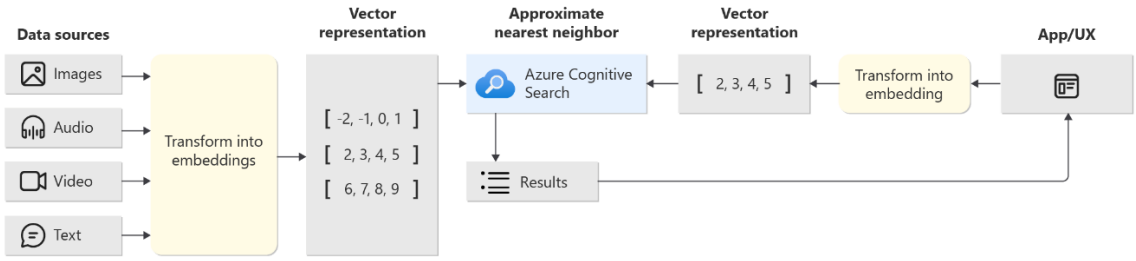


Figura 25. Ejemplo de Funcionamiento de Azure AI Search

3.5.6. Azure Cosmos DB

Azure Cosmos DB es una base de datos NoSQL distribuida globalmente y totalmente gestionada por Microsoft. Está diseñada para ofrecer tiempos de respuesta de milisegundos, escalabilidad automática e instantánea, y garantizar una alta disponibilidad. Soporta múltiples modelos de datos, incluyendo documentos, clave-valor y grafos [40].

- **Justificación:** Se seleccionó Cosmos DB frente a bases de datos relacionales tradicionales (como Azure SQL) por su flexibilidad de esquema. Los datos generados en una aplicación conversacional de IA (logs de conversación, estado de sesión, registros de feedback) son semi-estructurados y tienden a evolucionar. El modelo de datos basado en documentos JSON de Cosmos DB se alinea

perfectamente con los objetos de lenguajes como TypeScript y permite almacenar estas estructuras complejas sin las rigideces de un esquema relacional. Su baja latencia garantizada es fundamental para mantener una experiencia de usuario fluida en el chat.

3.5.7. Azure Blob Storage

Será nuestra forma de almacenar objetos, documentos en la nube. Azure Blob Storage es el servicio de almacenamiento de objetos de Microsoft, optimizado para almacenar cantidades masivas de datos no estructurados. Un "blob" (*Binary Large Object*) puede ser cualquier tipo de archivo de texto o binario, como un documento, una imagen o un vídeo.

- **Justificación:** La base de conocimiento de un sistema RAG se suele componer de archivos no estructurados (PDFs, documentos de Word, etc.). Se eligió Blob Storage como el repositorio para estos datos en bruto por ser la solución más costo-efectiva y escalable para este propósito. Blob Storage, ofrece un almacenamiento duradero y de bajo coste.

3.5.8. Azure Key Vault

Nos servirá en el futuro como gestión centralizada de secretos del aplicativo. Azure Key Vault es un servicio en la nube que permite almacenar y gestionar de forma segura secretos de aplicación, como claves de API, contraseñas, certificados o cadenas de conexión. Proporciona un control de acceso centralizado y un registro de auditoría completo [41].

- **Justificación:** La seguridad es un requisito no funcional crítico en cualquier aplicación. Key Vault se eligió para adherirse a las mejores prácticas de seguridad, eliminando la necesidad de almacenar credenciales sensibles en el código fuente o en archivos de configuración. Centralizar los secretos en Key Vault permite gestionar su ciclo de vida de forma independiente a la aplicación, mejorando drásticamente la postura de seguridad del proyecto y simplificando el cumplimiento de políticas de seguridad.

3.5.9. Microsoft Teams

Nos servirá de plataforma de interfaz conversacional. Microsoft Teams es la plataforma de colaboración y comunicación unificada de Microsoft que combina chat, videoconferencias, almacenamiento de archivos e integración de aplicaciones.

- **Justificación:** Como se ha mencionado anteriormente en otros capítulos, la decisión de utilizar Teams como la interfaz principal del sistema se basa en el principio de integración en el flujo de trabajo del usuario. En lugar de forzar a los usuarios a adoptar una nueva herramienta, el chatbot se despliega en el entorno que ya utilizan a diario. Esto no solo maximiza la adopción, sino que también aprovecha las capacidades nativas de Teams, como la autenticación implícita (que permite identificar al usuario de forma segura) y su soporte para interfaces de usuario a través de Tarjetas Adaptativas [42].

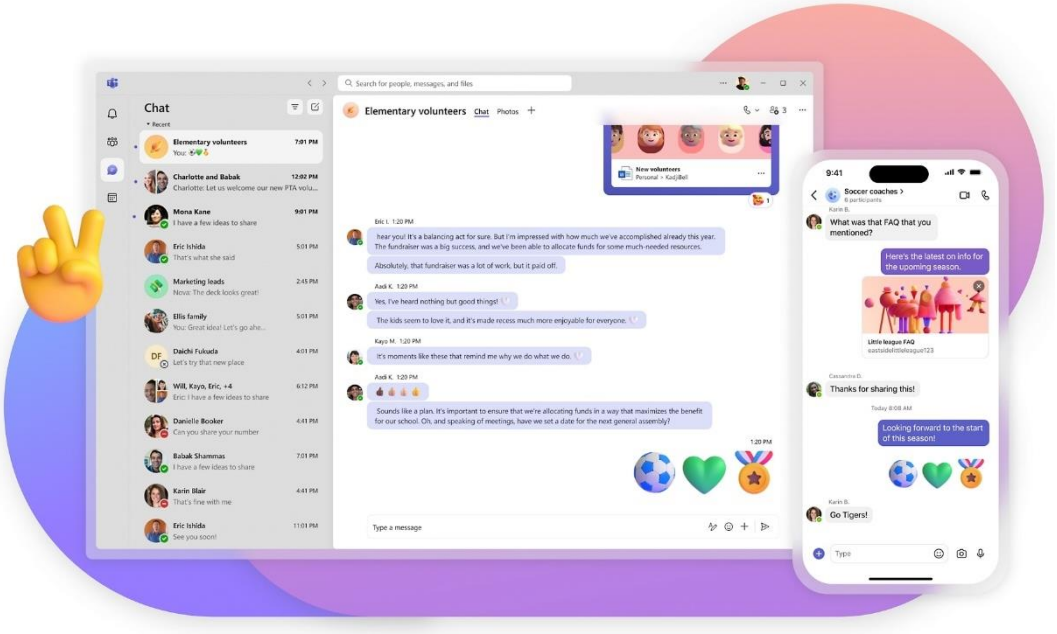


Figura 26. Interfaz de Microsfot Teams

Planificación y Gestión Del Proyecto

En este capítulo se detalla la metodología de gestión del proyecto llevada a cabo durante todo el proceso de desarrollo. Se comenzará describiendo la metodología para después mostrar como siguiendo esta metodología se ha llevado a cabo una estimación de tiempos del proyecto, así como una aproximación a la estimación de costes poniendo especial atención a los recursos humanos y a las particularidades de los proyectos basados en servicios de IA en la nube de Azure.
A continuación, se adjunta y detalla un presupuesto tanto a nivel académico como profesional estimado que comprenda los gastos para todas las herramientas necesarias para el desarrollo y despliegue de la aplicación, así como un coste aproximado del recurrente. Para acabar con un apartado de la gestión de posibles riesgos del proyecto.

4.1 Metodologías

La gestión de este proyecto se ha articulado mediante una adaptación de las metodologías ágiles, buscando aprovechar su flexibilidad, enfoque en la entrega incremental y capacidad de adaptación, incluso en un contexto de desarrollo individual. [43] Aunque tradicionalmente orientadas a equipos, sus principios fundamentales se han ido ajustando para optimizar el flujo de trabajo y conseguir finalizar este proyecto.

Esta adaptación metodológica se sustenta en los siguientes pilares:

1. **Estructura del Trabajo y Planificación Iterativa:**
    - **Organización en Sprints:** El desarrollo se ha dividido en sprints de 18-20 días naturales de duración. La elección de esta duración (aproximadamente dos semanas) busca un equilibrio entre la agilidad de ciclos cortos y la necesidad de un tiempo suficiente para completar bloques de trabajo significativos, manteniéndose dentro del límite recomendado de no exceder los 30 días por sprint [44].
    - **Entregas Incrementales y "Releases Privadas":** Cada sprint concluye con una "release privada". Es importante destacar que, especialmente en las fases iniciales del proyecto (donde se construye la infraestructura del backend, como el motor RAG), estos incrementales representan *funcionalidad interna probada y operativa*, más que una aplicación completamente desplegada y accesible para un usuario final. A medida que el proyecto avanza, los incrementales se vuelven más visibles, incluyendo el despliegue en Bot Service, la implementación de la recogida de feedback, mejora del tipo de búsqueda del retrieval, y mejoras específicas en la interacción en la última fase.
    - **Continuidad y Adaptabilidad:** Se ha procurado mantener una cadencia constante entre sprints siguiendo los principios de las metodologías ágiles,

no deberían existir descansos ni paradas de ningún tipo entre sprints. La metodología se diseñó con la flexibilidad necesaria para ajustar el cronograma de sprints, acomodando así la naturaleza individual, a tiempo parcial del proyecto y las contingencias laborales externas, sin perder la cadencia de entrega.

2. **Gestión de Requisitos y Estimación:**
    - **Desglose del Trabajo (Épicas y Tareas):** Para organizar el alcance del proyecto, se ha utilizado una estructura jerárquica. Las grandes funcionalidades o componentes del sistema se han definido como Épicas. Estas Épicas se descomponen en Tareas más específicas, que representan unidades de trabajo concretas a desarrollar. (Este desglose se ha gestionado mediante la herramienta JIRA de Atlassian, que se detallará posteriormente).
    - **Estimación con Puntos de Historia:** El esfuerzo requerido para completar cada Tarea se estima mediante Puntos de Historia. Esta métrica ágil valora de forma relativa la complejidad, el volumen de trabajo y la incertidumbre asociada.

    Nota: *Se ha optado por la estimación mediante Puntos de Historia en lugar de una estimación basada en tiempo (horas/días) debido a la naturaleza de por sí compleja e incierta del desarrollo de sistemas de IA dado el estado actual de la tecnología. Los Puntos de Historia permiten capturar de forma más fidedigna la complejidad y el riesgo asociados a tareas de investigación y desarrollo, como la optimización de prompts o el diseño de arquitecturas RAG avanzadas, donde una estimación temporal sería poco fiable.*

3. **Estimación de Costes:** La estimación de costes del proyecto se realiza teniendo en cuenta que es una aproximación. Esta cautela se justifica por la naturaleza novedosa del trabajo: al no existir proyectos análogos previos en los que basar las proyecciones, la precisión es limitada. Un componente significativo que impacta esta estimación es la extensa fase de investigación y autoaprendizaje indispensable para el proyecto como se ha podido observar por la extensión del capítulo de *Fundamentos Teóricos*. Dicha fase abarca tanto el dominio técnico de los recursos de Azure seleccionados como la consolidación de una base teórica robusta sobre tecnologías clave (LLMs, RAG, embeddings). Estos elementos de investigación dificultan una cuantificación exacta de los costes desde el inicio. El detalle de los gastos previstos para herramientas y recursos se expondrá en la sección correspondiente.
4. **Seguimiento del Proyecto:**
    - El seguimiento del progreso de las tareas y el avance dentro de los sprints se ha realizado utilizando herramientas de gestión de proyectos ágiles (JIRA).

4.1.1. Herramientas de Soporte a la Gestión y Desarrollo

Gestión de Tareas y Sprints con JIRA

Para la implementación práctica de la metodología ágil descrita y la gestión estructurada del trabajo, se seleccionó **JIRA** de ATLASSIAN como herramienta principal, se ha impuesto por su enfoque especial a proyectos ágiles comparadas a otras opciones como MS Project que están orientadas a equipos numerosos, metodología waterfall, diagramas Gantt complejos… aspectos más clásicos del desarrollo de software que no concuerdan con el proyecto que se está desarrollando. Esta plataforma facilitó la organización, seguimiento y planificación de todas las actividades del proyecto, desde la concepción de alto nivel hasta las tareas de desarrollo individuales. Se ha utilizado una instancia gratuita que provee JIRA, pero se ha utilizado dicha herramienta para que pueda ser trasladada a un entorno empresarial operativo sin problemas con su versión de pago.

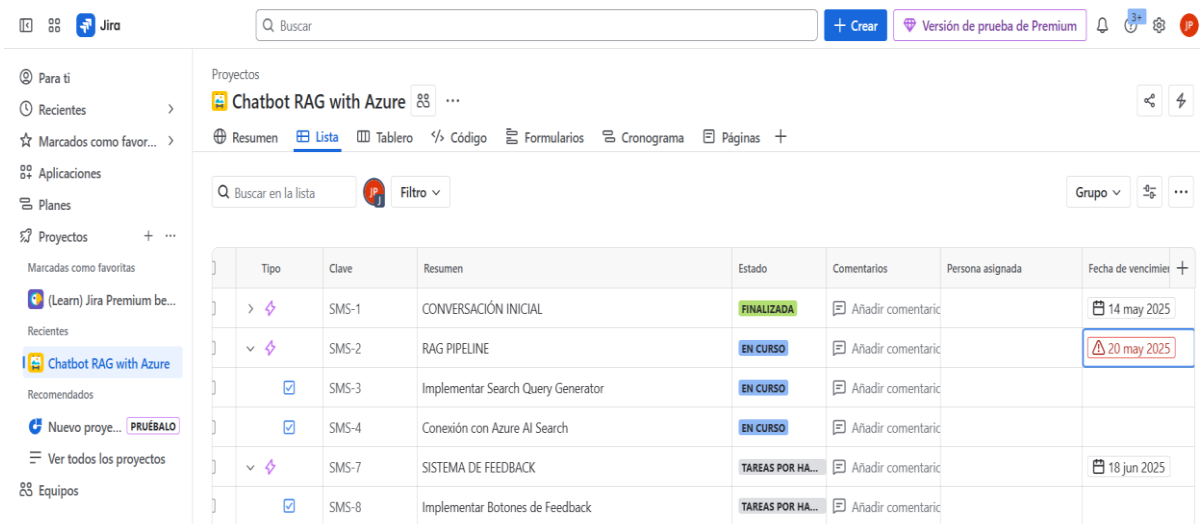


Figura 27. Interfaz Inicial del proyecto de JIRA

Organización Jerárquica del Trabajo

1. **Épicas (*Epics*):** Las grandes funcionalidades o componentes del proyecto se definieron como Épicas en JIRA. Estas representan bloques significativos de trabajo que agrupan un conjunto de requisitos relacionados y que suelen abarcar varios sprints para su completa implementación.
2. **Tareas (*Issues*):** Cada Épica se desglosó en unidades de trabajo más pequeñas y manejables, denominadas Tareas en el contexto de este proyecto (que en JIRA se gestionan como '*Issues*' de tipo '*Task*'). Estas representan los elementos individuales que se planifican y completan dentro de los sprints. Se pueden incluso dividir a mayores en subtareas que son puramente a nivel técnico.

Gestión del Product Backlog

Todas las Épicas e Historias de Usuario/Tareas identificadas conformaron el ***Product Backlog*** del proyecto en JIRA. Este backlog se mantuvo como una lista priorizada de todo el trabajo pendiente, revisándose y refinándose periódicamente para asegurar que reflejara las necesidades actuales del proyecto.

Sprint Planning

Al inicio de cada sprint, se llevó a cabo una planificación (*Sprint Planning*). Durante esta fase, se seleccionaron las Historias de Usuario/Tareas de mayor prioridad del *Product Backlog* que se estimaba podrían completarse dentro del sprint, conformando así el ***Sprint Backlog***.

Tablero de Sprint (Kanban Board)

El progreso de las tareas dentro de cada sprint se visualizó y gestionó a través de un tablero en JIRA (ya sea un tablero Kanban con columnas como 'Por Hacer', 'En curso', 'Listo'). Esto proporcionó una visión clara del estado del trabajo y ayudó a identificar posibles cuellos de botella.

Quiero mencionar que se han usado estas columnas que son las que ofrece el Tier Free de Jira por dafault, en caso de un desarrollo del aplicativo en productivo para un cliente final se podrían modificar a petición para poder adaptarse a cualquier metodología o nomenclatura de ella.

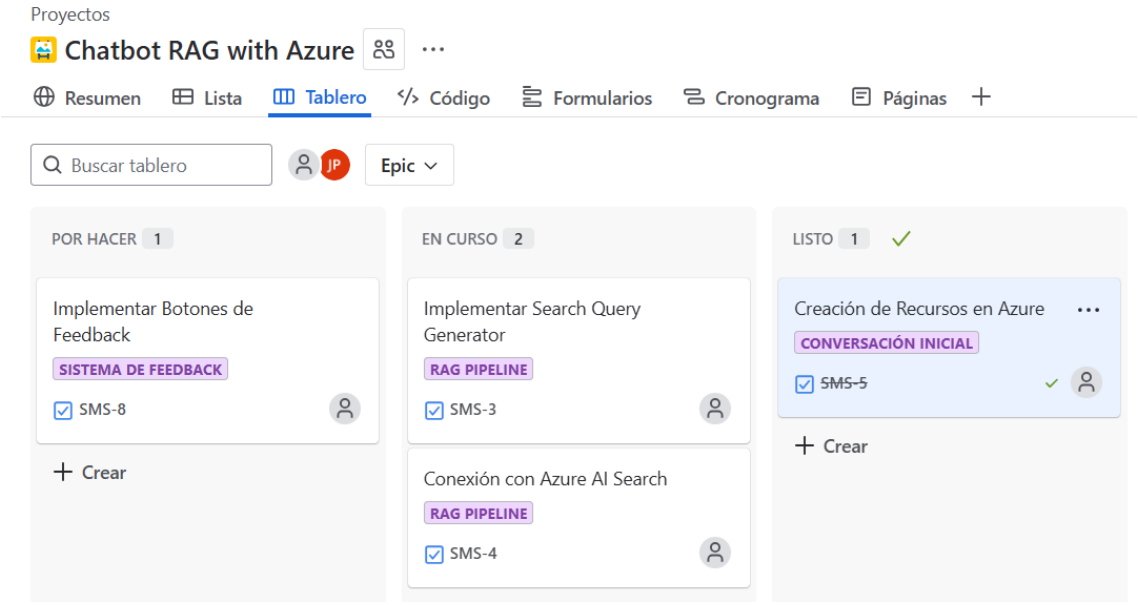


Figura 28. Tablero Kanban en JIRA

Adopción de Buenas Prácticas en el Desarrollo: Control de Versiones

Para asegurar la integridad del código, facilitar el seguimiento de cambios y aplicar buenas prácticas de desarrollo incluso en un proyecto individual, se adoptó un sistema

de control de versiones. Se utilizó Git como herramienta y GitHub como plataforma para el repositorio centralizado del proyecto. Se implementó una versión simplificada del flujo de trabajo Gitflow, utilizando principalmente una rama main para las versiones estables y ramas feature para el desarrollo de nuevas funcionalidades. Esta elección forma parte de la estrategia para gestionar el proyecto de forma ordenada y ágil, permitiendo un desarrollo iterativo y seguro. Los detalles específicos sobre esta adaptación de Gitflow y las razones de su simplificación se describirán en el Capítulo 8, dedicado a la Implementación y Despliegue.

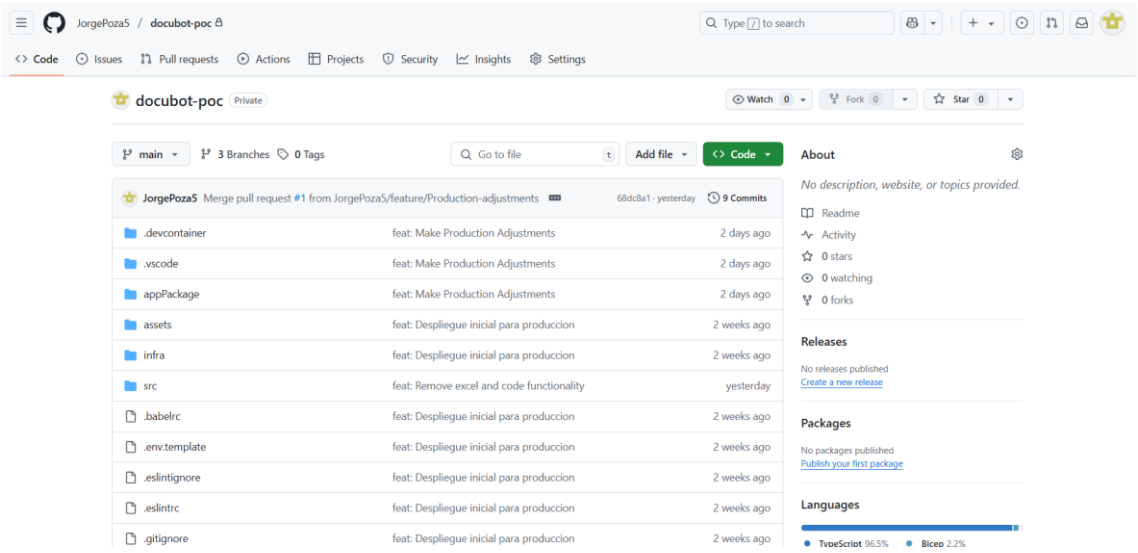


Figura 29. Interfaz de DocuBot en GitHub.

4.2 Estimación de Tiempos

Tras definir la metodología de gestión en la sección anterior, este apartado se centra en la planificación temporal y la estimación del esfuerzo requerido para el desarrollo del proyecto. A diferencia de proyectos de software con requisitos funcionales predecibles, la creación de una solución de Inteligencia Artificial como la presente implica un componente significativo de investigación y experimentación. Por ello, la planificación debe ser lo suficientemente flexible para acomodar esta naturaleza exploratoria, mientras proporciona una estructura clara para el avance del proyecto.

4.2.1. Planificación Temporal Inicial: Fases y Sprints

El ciclo de vida del proyecto se estructuró en dos fases principales: una fase conceptual intensiva seguida de una serie de sprints de desarrollo iterativo. A partir de ahora la duración media de un sprint de ha acordado en 20 días, ya que aunque es un poco más largo que las prácticas recomendadas al ser un proyecto en un contexto educativo realizado a tiempo parcial han hecho que se extienda en el tiempo.

Fase Conceptual y de Investigación (Duración: 2 meses)
El proyecto arrancó con una fase inicial no centrada en la codificación ni en la planificación en sí del mismo, sino en una investigación tecnológica profunda, del estado del arte y de posible cuota de mercado y competidores, indispensable para sentar las bases de una solución robusta y sobre las que se han basado todo el apartado de Fundamentos Teóricos.

La duración de esta fase y dado que ha sido previo a la planificación y por tanto al desglose en *sprints*, aunque no se organizó en sprints formales, tuvo una duración equivalente a 3 sprints de trabajo (20 días aproximadamente cada uno). Su objetivo no era producir un incremento de software, sino generar los activos intelectuales y de planificación necesarios para el desarrollo. Esta etapa fue crítica para mitigar riesgos técnicos y definir a posteriori una arquitectura viable. Las actividades principales incluyeron:

- **Análisis de Servicios Cloud:** Evaluación de las capacidades, limitaciones y modelos de coste de los servicios clave de Azure, como Azure OpenAI, Azure AI Search, Cosmos DB y Blob Storage.
- **Estudio de Teórico de Gen AI:** Investigación de las bases sobre las que se asientan el concepto de LLM, origen de todo el proyecto.
- **Estudio de Arquitecturas RAG:** Investigación de las mejores prácticas y patrones para el diseño de sistemas de RAG, incluyendo técnicas avanzadas como búsqueda híbrida y estrategias de *prompt engineering*, definición de *embeddings*…
- **Dominio del SDK:** Estudio en profundidad del SDK de Microsoft Teams AI, comprendiendo su modelo de eventos, la gestión del estado y las capacidades de integración de fuentes de datos. Revisión de ejemplos y plantillas del SDK.
- **Diseño Arquitectónico:** Creación de un diseño conceptual de la aplicación, definiendo la interacción entre los diferentes microservicios y componentes de la arquitectura.

Fase de Desarrollo por Sprints (4 meses)
Una vez finalizada la fase conceptual, el desarrollo se organizó en 6 sprints de aproximadamente 20 días naturales cada uno. Esta duración se consideró óptima para permitir el desarrollo de bloques funcionales completos, incluyendo las fases de experimentación necesarias. Durante estos sprints sí se aplicó la metodología ágil de forma estricta, con el objetivo de entregar un incremento funcional y probado al final de cada ciclo. El trabajo se distribuyó agrupando las 8 Épicas funcionales definidas para el proyecto, como se visualiza en el siguiente capítulo.

Quisiera hacer un inciso sobre la planificación de la **Épica 2: Motor de Razonamiento y Respuesta (RAG)**, que constituye el núcleo inteligente del sistema, es la más compleja y extensa del proyecto. Para gestionar eficazmente su desarrollo, se tomó la

decisión estratégica de dividir su implementación en dos sprints consecutivos. El Sprint 3 se centró en la parte de Recuperación (búsqueda híbrida, mejora de consultas, manejo de documentos), mientras que el Sprint 4 se enfocó en la parte
de Generación (integración con el LLM, generación de citas, formateo de respuestas). Esta división permitió abordar su complejidad de forma más manejable y asegurar la calidad de cada componente.

Se reconoce que una directriz común en metodologías ágiles es descomponer Épicas de gran tamaño en varias Épicas más pequeñas. Sin embargo, en este caso, se considera fundamental mantener la cohesión conceptual del flujo RAG completo bajo una única Épica. Separarla habría dificultado la visión integral del componente más crítico del sistema. Por lo tanto, se optó por una **división temporal** (en sprints) en lugar de una **división funcional** (en Épicas separadas), una adaptación pragmática que priorizó la claridad arquitectónica y la integridad de los futuros componente.

4.2.2. Puntos de Historia Aplicados a la Complejidad de IA

Como se justificó en la sección anterior, para estimar el esfuerzo de cada Tarea se ha utilizado la métrica ágil de Puntos de Historia (PH). Esta elección es especialmente pertinente para este proyecto, ya que permite capturar de forma más fidedigna la complejidad, el riesgo y la incertidumbre del desarrollo de sistemas de IA, en contraste con una estimación temporal que sería poco fiable [45].

Partiendo de esta base, se ha definido una escala de complejidad a nivel técnico que traduce estos conceptos abstractos a valores numéricos, permitiendo una planificación y priorización efectivas. La asignación de Puntos de Historia se ha adaptado para reflejar de manera fidedigna la distribución de tareas en este proyecto, desde las configuraciones más simples hasta la investigación más compleja, se ha establecido la siguiente escala de estimación basada en la serie de Fibonacci. Mencionar una vez mas que los puntos representan un valor relativo, y pueden ser diferentes para cada equipo o proyecto.

- **Complejidad MÍNIMA (1-2 PH):** Tareas bien definidas con baja incertidumbre.
    - *Ejemplos:* Creación de una tarjeta adaptativa estática, implementación de una función de utilidad simple.
- **Complejidad MEDIA (3 PH):** Tareas que requieren desarrollo de lógica de negocio o integración de varios componentes.
    - *Ejemplos:* Implementar un endpoint en el servidor Restify, desarrollar un manejador de eventos de Teams, implementar un gestor para interactuar con un servicio de Azure.
- **Complejidad ALTA (5 PH):** Desarrollo de componentes centrales con lógica de negocio compleja o múltiples dependencias, que orquestan otros servicios.

    - *Ejemplos:* Implementación completa del servicio para la autenticación y obtención de datos de usuario, diseño del flujo de orquestación principal para manejar las distintas interacciones.

- **Complejdidad MUY ALTA / INVESTIGACIÓN (8 PH):** Tareas complejas que combinan desarrollo con un componente significativo de incertidumbre e investigación. Requieren pruebas y validación iterativa.
    - *Ejemplos:*
        - Implementación de la búsqueda híbrida vecotrial.
        - Desarrollo para reescribir consultas.
- **Complejidad MÁXIMA / EXPERIMENTACIÓN PURA (13 PH):** Tareas cuyo resultado depende casi exclusivamente de la experimentación, el ajuste fino y la validación empírica. El esfuerzo no está en implementar mucho código, sino en iterar hasta encontrar la solución óptima, pudiendo ser transversal a la realización de otras tareas.
    - *Ejemplos:*
        - Ajuste Fino de Relevancia (Fine-Tuning de RAG): Analizar los resultados de búsqueda, probar diferentes técnicas de chunking de documentos, ajuste de parámetros…

4.2.3. Estimación de Tiempos por Sprint

Tras definir la metodología y las fases del proyecto en las secciones anteriores, este apartado se centra en la concreción temporal y la distribución del trabajo. Para ello, se presenta un cronograma que abarca los seis meses de duración total del proyecto, anclando cada fase y sprint a un periodo específico y cuantificando el esfuerzo del desarrollo mediante las métricas ágiles definidas.

La planificación se dividió en dos periodos principales, cubriendo desde la investigación inicial hasta la ejecución iterativa:

- **Fase Conceptual:** Periodo de 2 meses (Enero - Febrero 2025).
- **Fase de Desarrollo:** Periodo de 4 meses (Marzo - Julio 2025).

La planificación detallada de la fase de desarrollo se fundamenta en la métrica de estimación por Puntos de Historia, definida en la sección anterior. El siguiente paso consiste en distribuir el conjunto de Tareas del proyecto en la secuencia de los seis sprints de desarrollo. Cada sprint se planificó con un alcance medido en Puntos de Historia. Para ello, se seleccionaron tareas de diversas Épicas buscando un equilibrio en la carga de trabajo y asegurando que cada sprint entregara un incremento de valor coherente.

Sprint	Duración Estimada	Objetivo del Sprint /Incremento	Épicas Abordadas
Sprint 1	20 días	**Fundamentos y Seguridad:** Crear el esqueleto de la aplicación y el servicio de autenticación y autorización.	EPIC-01, EPIC-05
Sprint 2	20 días	**Interfaz Inicial e Ingesta de Conocimiento:** Desarrollar la selección de categorías y el sistema base de ingesta.	EPIC-01, EPIC-06
Sprint 3	20 días	**Núcleo RAG (Parte 1 - Recuperación):** Implementar la búsqueda híbrida y el reescritor de consultas.	EPIC-02 (Foco en Recuperación)
Sprint 4	20 días	**Núcleo RAG (Parte 2 - Generación y Citas):** Integrar el LLM principal y el sistema de generación de citas.	EPIC-02 (Foco en Generación)
Sprint 5	20 días	**Ciclo de Feedback y Persistencia:** Implementar la tarjeta de respuesta final, el sistema de feedback y el almacenamiento en Cosmos DB.	EPIC-03, EPIC-04
Sprint 6	20 días	**Optimización y Despliegue:** Realizar mejoras de robustez y ejecutar el despliegue final en Azure.	EPIC-07, EPIC-08

Tabla 1. Cronograma Desarrollo por Sprints y Foco Funcional

La anterior tabla detalla este cronograma, que actúa como el mapa de ruta de la fase de desarrollo. En ella se muestra el objetivo principal de cada sprint y se hace referencia a las Épicas Funcionales trabajadas, las cuales se definen y detallan en el Capítulo 5: Análisis Funcional.

La gestión de la carga de trabajo dentro de cada sprint se basó en la estimación de PH asignada a cada Tarea individual, como se describió en la sección 4.2.2. Durante la fase de planificación de cada sprint, se seleccionaban las tareas de mayor prioridad del backlog hasta completar una capacidad de trabajo considerada manejable para el ciclo de 20 días.

4.3 Presupuesto

En esta sección se presenta un análisis detallado de los costes asociados al proyecto, distinguiendo entre el coste de producción interno y el presupuesto de venta a un cliente final. Este desglose permite una visión completa tanto de la inversión necesaria para desarrollar la solución como de su valor en el mercado. El presupuesto se desglosa en tres áreas principales: los costes de los recursos humanos involucrados en el desarrollo, los costes de hardware y software necesarios, y los costes recurrentes de la infraestructura en la nube.

Es importante realizar los siguientes matices antes de presentar el desglose:

- **Contexto Académico vs. Profesional:** Se ha tenido acceso a créditos gratuitos de Azure (200 $ iniciales en la creación de una cuenta) y a licencias de software sin coste (GitHub for Students, JIRA Free Tier). Se detalla en el primer apartado.
- **Exclusión de Gastos Indirectos:** Se han obviado los costes indirectos como alquiler de oficinas, electricidad o servicios generales, asumiendo un entorno de trabajo en remoto. Si que se han añadido esos gastos a la hora del presupuesto final a cliente como gastos de estructura.
- **Herramientas Open Source:** No se presupuestan las herramientas de código abierto que han sido fundamentales, como Visual Studio Code, Git, Node.js o las librerías del ecosistema de TypeScript, pero se reconoce su valor esencial en el proyecto.
- **Periodo de Cálculo:** Los costes se calculan para la duración total del proyecto, estimada en **6 meses** (2 meses de fase conceptual y 4 meses de desarrollo por sprints).

El coste de producción representa la inversión directa y necesaria para llevar a cabo el desarrollo del proyecto desde cero. Se compone de los costes salariales del equipo de desarrollo y los costes de las herramientas y la infraestructura utilizadas durante la fase de creación.

4.3.1 Inversión Real en Contexto Académico

Antes de proceder al análisis detallado de un presupuesto profesional, es importante aclarar el coste real incurrido durante la ejecución de este proyecto en su marco académico. La finalidad de esta sección es transparentar que las cifras estimadas en los apartados siguientes corresponden a una simulación empresarial y no reflejan un desembolso económico personal.

El desarrollo de la solución se ha llevado a cabo optimizando al máximo los recursos disponibles, una práctica habitual en la fase de prototipado y validación de conceptos. Los costes se han minimizado mediante las siguientes estrategias:

- **Aprovechamiento de Créditos de Infraestructura Cloud:** Se ha hecho uso de los créditos gratuitos que Microsoft Azure ofrece a las cuentas. Estos créditos han sido suficientes para cubrir por completo los gastos de todos los servicios en la nube utilizados durante la fase de desarrollo, como Azure OpenAI, Azure AI Search, Azure App Service y Cosmos DB.
- **Recursos Tier Free Azure (F0).** Hay ciertos recursos de Azure, la mayoría de los usados en este proyecto, que siendo usados en contextos de PoC, por lo tanto con un uso relativamente bajo y sin posibilidad de escalabilidad ofrecen Tiers Gratuitos (F0) para poder consumir dichos recursos sin coste alguno.
- **Uso de Herramientas con Niveles Gratuitos:** Para la gestión del proyecto y el control de versiones, se han utilizado los planes gratuitos de herramientas

estándar en la industria como **JIRA (Free Tier)** y **GitHub**, que ofrecen las funcionalidades necesarias para un proyecto de esta escala sin coste asociado.

- **Aportación de Recursos Propios:** El desarrollo se ha sustentado sobre la infraestructura personal del desarrollador. Esto incluye el portátil HP y los servicios básicos como el Wi-Fi. Estos recursos, aunque esenciales para el proyecto, no se han imputado como un coste directo del desarrollo académico, ya que forman parte del equipamiento estándar preexistente.
- **Inversión Controlada para Simulación Empresarial:** La única inversión monetaria directa realizada fue la contratación de una suscripción mensual a **Microsoft 365 Enterprise (**aproximadamente **7 €/mes)**. Esta decisión se tomó para poder configurar un entorno de desarrollo realista, que incluyera un *tenant* propio con Microsoft Teams, SharePoint y acceso a Microsoft Graph API, replicando así las condiciones de un cliente final [46].
    - **Coste Real Incurrido:** El coste de esta suscripción fue de aproximadamente **7 € al mes**. Durante los 3 últimos meses del proyecto, en los que realmente se necesitó una organización de SharePoint y Teams, el desembolso total ascendió a: 7 €/mes * 3 meses = 21 €

Por lo tanto, el desembolso total para la realización de este proyecto académico ha sido mínimo y se ha limitado estrictamente a los costes de la suscripción de Microsoft 365. En definitiva, la ejecución del proyecto en un marco académico ha supuesto un coste monetario total de **21 €**, este análisis justifica la viabilidad económica del desarrollo en este contexto,

Las estimaciones de costes de recursos humanos, licencias de software y consumo de infraestructura que se presentan a continuación deben entenderse como un ejercicio de presupuestación profesional, diseñado para valorar el proyecto como si fuera un encargo comercial del sistema DocuBot.

4.3.2 Recursos Humanos

El principal coste en cualquier proyecto de IA es el derivado del tiempo y esfuerzo del equipo de desarrollo. Este proyecto ha sido ejecutado por un único desarrollador, una situación común en fases de prototipado o en la creación de una Prueba de Concepto (PoC) y más en entorno de IA donde como se ha ido comentando la incertidumbre y el descubrimiento de nuevas tecnologías está a la orden del día.

Para realizar una estimación de costes profesional, se analizará el trabajo realizado no como el de una sola persona, sino como el desempeño de un perfil técnico híbrido que ha asumido diferentes roles a lo largo del ciclo de vida del proyecto. Este enfoque permite valorar adecuadamente la diversidad de competencias requeridas.

Perfiles Profesionales Involucrados:

- ***AI Engineer*:** Durante la mayor parte del proyecto, el rol principal fue el de un ingeniero de IA, centrado en el diseño de la arquitectura RAG, el *prompt engineering*, la integración de modelos de lenguaje y el desarrollo general de DocuBot.
- ***Cloud Engineer*:** En fases específicas, como la configuración inicial de la infraestructura en Azure y el despliegue final, el rol se desplazó hacia el de un ingeniero Cloud/DevOps.
- ***Software Architect*:** Durante la fase conceptual, el rol asumido fue el de un arquitecto de software, encargado de diseñar la solución y seleccionar la pila tecnológica.

Metodología de Cálculo:

Se calculará el coste total basándose en una dedicación equivalente a una jornada laboral parcial (20 horas/semana) durante la duración del proyecto. En un contexto empresarial se puede justificar la jornada parcial por la inclusión de un mismo ingeniero en varios proyectos. Se utilizará un coste por hora promedio que refleje la naturaleza especializada y polivalente del perfil.

1. **Salario Anual de Referencia:** Se toma como referencia un salario bruto anual de **36.000 €** para un perfil de Ingeniero de IA / Cloud con la experiencia requerida para este proyecto en el mercado español [47].
2. **Coste Salarial por Hora:**
    - Horas anuales (jornada semanal de 40h): 8 horas/día * 220 días laborables/año = 1760 horas. Equivaldría a 1 FTE (*Full Time Equivalent*) que es la convención común que se utiliza en las empresas para el tiempo que trabaja una persona a tiempo completo en un año [48].
    - Coste salarial por hora: 36.000 € / 1760 horas ≈ **20,45 €/hora.**
3. **Duración y Alcance Total del Proyecto:** Para realizar un cálculo completo, la duración total del proyecto se estima en **6.5 meses**. Este periodo abarca no solo la fase conceptual (2 meses) y los seis sprints de desarrollo (4 meses), como ya se comentó en la fase de planificación, sino también una fase final de documentación y entrega final.

Nota sobre la Fase de Documentación y Transferencia: Se ha incluido una fase final de 10 días laborables (2 semanas) dedicada a la documentación técnica y la transferencia de conocimiento y de la solución. Aunque en este contexto académico se materializa como la redacción de la presente memoria y posterior presentación, en un entorno profesional esta etapa seguiría existiendo, siendo crucial y equivale a la creación de manuales de arquitectura, guías de despliegue y sesiones de formación para el cliente final. Su inclusión es indispensable para presupuestar un proyecto de manera integral y asegurar su mantenibilidad a futuro y abrir futuras relaciones con el mismo cliente.

4. **Cálculo de Horas Totales:** Se tiene en cuenta que la carga de trabajo en este proyecto ha sido de media 20 horas semanales o 0.5 FTE.
    - Fase Conceptual: 2 meses (aprox. 180 horas).
    - Fase de Desarrollo (6 sprints): 4 meses (aprox. 345 horas).
    - Fase de Documentación: 2 semanas (40 horas).
    - **Total de horas estimadas:** 180 + 345 + 40 = **565 horas**. Este total concuerda aproximadamente con los 6.5 meses de duración total del proyecto a tiempo parcial.

Cálculo del Coste Salarial Total:

Perfil Profesional	**Horas Totales Estimadas**	**Coste Salarial por Hora**	**Coste Salarial total**
Ingeniero Híbrido (IA/Cloud/Architect)	565 horas	20,45 €	**11.554 €**

Tabla 2. Cálculo de coste salarial

4.3.3 Hardware, Software e Infraestructura

Además del coste de los recursos humanos, el coste de producción incluye la inversión en las herramientas de desarrollo y, de manera fundamental en un proyecto nativo en la nube, el coste de los servicios de la plataforma cloud durante el ciclo de vida del desarrollo. En cuanto a Hardware hay que destacar que se ha optado por el renting de un equipo como se hace en las grandes empresas y no en el prorrateo de la compra de uno. A continuación, se desglosan estas partidas.

Recurso	**Proveedor**	**Modelo de Coste**	**Coste Desarrollo (6 meses)**
HARDWARE			
Estación de trabajo (Portátil)	HP	Renting	**300 €**
SOFTWARE (LICENCIAS)			
Repositorio de Código	GitHub	GitHub Enterprise (1 usuario)	20/mes € * 6 meses = **120 €**
Gestión de Proyecto	JIRA	Standard Plan (1 usuario)	10 €/ mes * 6 meses = **60 €**
INFRAESTRUCTURA CLOUD (AZURE)			
Azure OpenAI	Microsoft Azure	Pago por Uso	**~200 €**
Azure AI Search	Microsoft Azure	SKU Basic (S1)	60 €/mes * 6 meses = **360 €**

Azure App Service	Microsoft Azure	Plan Basic (B1)	70 €/mes * 6 meses = **420 €**
Azure Cosmos DB	Microsoft Azure	1000 RU/s	60 €/mes * 6 meses = **360 €**
Azure Blob Storage	Microsoft Azure	Pago por GB y Transacciones	~5 €/mes * 6 meses = **30 €**
Azure Key Vault	Microsoft Azure	Pago por Transacciones	~1 €/mes

Tabla 3. Coste de Hardware, Software e Infra

El subtotal de todo el coste de Hardware, Software e Infraestructura utilizado asciende a **2050 €** que se deberán tener en cuenta a la hora de realizar el presupuesto final.

4.3.4 Presupuesto Total a Cliente

Esta sección presenta el presupuesto que una empresa de consultoría tecnológica como la que se puede constituir a raíz de este tipo de proyectos facturaría a un cliente por el desarrollo de este proyecto. Este precio no solo cubre los costes de producción, sino que también incluye los costes de estructura de la empresa y su margen de beneficio.

Tarifa de Mercado (Coste Empresarial)

La tarifa que una empresa cobra por sus profesionales es superior a su coste salarial. Para este proyecto, se establece una tarifa de mercado de **40 €/hora** para un perfil híbrido de IA/Cloud.

Nota sobre la Tarifa de Mercado: Es fundamental matizar este valor. No representa el salario bruto que percibe el desarrollador (estimado en 20,45 €/hora), sino el coste total que el recurso supone para la empresa o la tarifa que esta facturaría a un cliente. Dicho valor incluye, además del salario bruto: las contribuciones a la seguridad social, los gastos de estructura (licencias comunes, infraestructura, administración, equipos auxiliares si fueran necesarios como RRHH, Financieros…), los costes de formación continua y el margen de beneficio empresarial. Por tanto, 40 €/hora es una estimación conservadora y realista del valor de mercado de una hora de trabajo de este perfil especializado.

Cálculo del Presupuesto de Venta:

Concepto	**Horas Totales Estimadas**	**Coste Salarial por Hora**	**Coste Salarial total**

Servicios Profesionales (Todos los Roles Incluidos)	565 horas	40 €	22.600 €

Tabla 4. Cálculo del presupuesto de ventas

Presupuesto Total de Venta al Cliente:

Concepto	**Coste**
Servicios Profesionales (Mano de Obra)	22.600 €
Hardware, Software e Infraestructura (Desarrollo)	2.050 €
PRESUPUESTO TOTAL DE VENTA	**24.650 €**

Tabla 5. Presupuesto Total de Venta

Estimación de Costes Operativos Recurrentes (OPEX)

Una vez que el proyecto se ha desarrollado y desplegado, entra en una fase de operación en la que incurre en costes mensuales recurrentes para su mantenimiento y funcionamiento. Estos costes operativos (OPEX) son fundamentales para la planificación a largo plazo del producto y serían asumidos por el cliente final o la entidad que gestiona la aplicación.

Recurso	**Proveedor**	**Modelo de Coste**	**Coste Operativo (Estim. Mensual)**
SOFTWARE (LICENCIAS)			
Repositorio de Código	GitHub	GitHub Enterprise (1 usuario)	20 €
Gestión de Proyecto	JIRA	Standard Plan (1 usuario)	10 €
INFRAESTRUCTURA CLOUD (AZURE)			
Azure OpenAI	Microsoft Azure	Pago por Uso	8,81€ / 1 millón Tokens
Azure AI Search	Microsoft Azure	SKU Basic (S1)	60 €
Azure App Service	Microsoft Azure	Plan Basic (B1)	70 €
Azure Cosmos DB	Microsoft Azure	1000 RU/s	60 €
Azure Blob Storage	Microsoft Azure	Pago por GB y Transacciones	~5 €
Azure Key Vault	Microsoft Azure	Pago por Transacciones	~1 €

Tabla 6. Estimación Costes Operativos Mensuales

El total de estos costes fijos recurrentes asciende a aproximadamente 250 € mensuales. A esta cifra deben sumarse los costes variables derivados del uso de los servicios de

Inteligencia Artificial de Azure OpenAI, que dependen directamente del número de consultas realizadas a DocuBot.

Estos costes variables se componen principalmente de dos elementos:

- **LLM (*GPT-4o*):** Es el principal motor del coste. Su precio se desglosa por consumo de tokens de entrada (contexto enviado) y de salida (respuesta generada). En este proyecto, se ha medido que cada consulta a DocuBot consume, de media, unos 7.000 tokens. La gran mayoría de estos corresponden a los tokens de entrada (pregunta del usuario, historial y, fundamentalmente, el contexto extraído de los documentos). La respuesta generada es mucho más corta, por lo que su coste de salida es mucho menor.
- **Modelo de Embeddings (*text-embedding-3-large*):** Este modelo se utiliza para convertir las preguntas de los usuarios en vectores para la búsqueda. Su coste es significativamente más bajo y, aunque se tiene en cuenta, resulta marginal en comparación con el coste del modelo *GPT-4o* en la operativa diaria.

Modelo	Precios (1M de tokens)
GPT-4o-2024-1120 Global	Entrada: €2,13530 Entrada en caché: €1,0677 Salida: €8,5412
text-embedding-3-large	€0,000112

Figura 30. Costes modelos Azure OpenAI API

Estos costes de servicios en LLM, irán reduciendo con el paso del tiempo pues según se ha podido comprobar a lo lardo del último año la gran competencia entre los proveedores de estas soluciones hacen que los precios de los LLM por millón de tokens se estén reduciendo drásticamente [49].

El coste de estos modelos puede ser monitorizado gracias al trabajo de generar el *logging* correspondiente en Cosmos DB.

Escenario de Coste Operativo Total

Para ofrecer una estimación tangible, se puede plantear un escenario de uso moderado para un equipo o departamento:

- **Supuesto:** 1.500 consultas a DocuBot al mes.
- **Cálculo de Tokens Totales:** 1.500 consultas/mes * 7.000 tokens/consulta = 10.500.000 tokens (10,5M tokens)

- **Estimación de Coste Variable (IA):** Asumiendo que la mayoría son tokens de entrada, el coste aproximado sería: 10,5M tokens * 8,54 €/M tokens ≈ 22,37 €
- **Coste Operativo Mensual Total (OPEX):**
    - Coste Fijo + Coste Variable (IA) = 250 € + 22,37 € = **272,37 €**

Este análisis detallado del OPEX permite al cliente final no solo conocer el coste de desarrollo, sino también prever con precisión los gastos de mantenimiento y operación de la solución, lo cual es fundamental para calcular el Retorno de la Inversión (ROI).

Análisis Comparativo y Presupuesto Final

El desglose realizado permite un análisis financiero básico de la viabilidad del proyecto en un entorno comercial.

- **Coste Total de Producción:**11.554 € (RRHH) + 2.050 € (HW y SW) = 13.604 €
- **Presupuesto de Venta:** 24.650 €
- **Margen Bruto del Proyecto:** 24.650 € - 13.604 € = **11.046 €**
- **Factor de Multiplicación (*Markup*):** 24.650 € / 13.604 € ≈ **1,81**

Conclusión del Análisis:
El presupuesto de venta estimado de **24.650 €** para el proyecto representa un factor de multiplicación (*markup*) de **1,81** sobre el coste total de producción. Este factor de multiplicación se encuentra dentro de un rango saludable para proyectos de consultoría tecnológica, validando que la tarifa de 40 €/hora es comercialmente sostenible.

Permite a una empresa cubrir todos sus costes directos e indirectos (gastos generales, impuestos, etc.) así como anticiparse a posibles retrasos, fallos de planificación o problemas no esperados que surjan por nuestro lado y, al mismo tiempo, generar un margen de beneficio robusto, lo que confirma la viabilidad económica del proyecto al estar en los márgenes deseados.

Adicionalmente, el cliente final o la empresa operadora debería presupuestar un mínimo coste recurrente de aproximadamente **250 € al mes** para el mantenimiento de la infraestructura en la nube. Como se ha calculado en el escenario de uso bajo-moderado, esto podría situar el OPEX total en torno a los **273 € mensuales**. Este coste operativo es una consideración crítica para la sostenibilidad a largo plazo de la solución y podría ser la base para un modelo de negocio tipo SaaS (*Software as a Service*) o un contrato de mantenimiento y soporte.

4.4 Gestión de Riesgos

La planificación de un proyecto tecnológico, especialmente en un campo tan dinámico como la Inteligencia Artificial, sería incompleta sin un análisis proactivo de los riesgos. Siguiendo las buenas prácticas recomendadas por marcos de gestión como el PMBOK [50], en esta sección se identifican las principales amenazas que podrían afectar a los objetivos del proyecto, se evalúa su probabilidad e impacto, y se definen estrategias de mitigación para minimizar su efecto.

El objetivo no es solo crear un listado estático, sino establecer un marco de pensamiento para anticipar desafíos y asegurar la viabilidad y calidad de la solución final. se ha optado por una matriz de gestión de riesgos consolidada.

En la siguiente tabla se detallan los riesgos identificados, su contexto específico para el proyecto DocuBot, una valoración cualitativa de su probabilidad e impacto, y las estrategias de mitigación y contingencia diseñadas.

ID	Riesgo	Probabilidad	Impacto	Nivel de Riesgo
R1	Técnico: Incompatibilidad o cambio disruptivo en el ecosistema tecnológico (SDKs, APIs).	Media	Alta	**Significativo**
R2	Calidad IA: Baja relevancia o precisión en las respuestas del motor RAG.	Alta	Alta	**Crítico**
R3	Costes: Superación de los costes estimados de la infraestructura de Azure.	Baja	Media	**Moderado**
R4	Planificación: Subestimación del esfuerzo en tareas de investigación y experimentación.	Alta	Media	**Significativo**
R5	Dependencia de Terceros: Degradación o caída del servicio en la API de Azure OpenAI.	Baja	Alta	**Significativo**

Tabla 7. Matriz Evaluación de Riesgos.

4.4.1 Estrategias de Gestión de Riesgos

A continuación, se detallan las estrategias de mitigación y contingencia diseñadas para cada uno de los riesgos priorizados.

- **R1 - Riesgo Técnico (Significativo):** La mitigación se centró en un diseño modular que aísla las APIs externas. Esto asegura que los cambios en el SDK o en los servicios de Azure tengan un impacto localizado y manejable.
- **R2 - Riesgo de Calidad de IA (Crítico):** Siendo el riesgo principal, su gestión fue un pilar del diseño. La mitigación proactiva incluyó una arquitectura RAG avanzada con búsqueda híbrida y un diseño de prompt con contexto ampliado. Como contingencia, el sistema de feedback del usuario es la herramienta clave para la mejora continua y el ajuste fino del modelo.

- **R3 - Riesgo de Costes (Moderado):** Se gestionó mediante la monitorización continua y el uso de Tiers de servicio de bajo coste (F0) durante el desarrollo.
- **R4 - Riesgo de Planificación (Significativo):** La incertidumbre inherente a la I+D se abordó utilizando Puntos de Historia para estimar la complejidad en lugar del tiempo. La planificación flexible de los sprints, con margen para la experimentación, fue fundamental para mitigar retrasos.
- **R5 - Riesgo de Dependencia (Significativo):** La resiliencia del sistema se garantizó por ejemplo implementando un mecanismo de reintentos con backoff exponencial en llamadas a OpenAI. De cara al usuario, se diseñaron mensajes de error claros que comunican fallos temporales en el servicio de IA, gestionando así las expectativas.

Análisis Funcional

En este capítulo se describirá y detallará el análisis funcional de la plataforma de IA conversacional. Hay que destacar que a diferencia de las aplicaciones de software tradicionales, donde el análisis funcional se centra en casos de uso y flujos de interacción discretos, en este proyecto el enfoque se adapta para describir el comportamiento de un sistema inteligente. La funcionalidad principal no reside en un conjunto de pantallas y botones, sino en un proceso cognitivo orquestado cuyo objetivo es comprender, razonar y responder a las consultas de los usuarios en lenguaje natural.

Por tanto, este análisis se articulará en torno a tres pilares fundamentales que se ajustan mejor a la naturaleza de una aplicación de Inteligencia Artificial:

1. **Identificación de Roles:** Define los distintos perfiles que interactúan con el ecosistema de la solución, no solo con la interfaz de chat, sino con el sistema en su totalidad.
2. **Capacidades Funcionales:** Describe las funcionalidades clave del sistema, alineadas con la metodología de gestión del proyecto basada en Épicas y Tareas, detallada en el capítulo anterior de metodología.
3. **Requisitos No Funcionales:** Especifica las cualidades críticas del sistema, como el rendimiento, la seguridad, la escalabilidad y, de manera crucial para una aplicación de IA, la fiabilidad y precisión de sus respuestas.

Este enfoque permite una descripción completa y precisa de lo que el sistema hace, para quién lo hace y con qué nivel de calidad y restricciones opera, proporcionando una visión integral de su valor y comportamiento.

5.1 Identificación de Roles/Actores

Seguimos con particularidades de un aplicativo de IA Generativa, ya que un proceso de desarrollo de software clásico, la fase de análisis se inicia definiendo "actores" que ejecutan "casos de uso". Siguiendo la metodología ágil adoptada en este proyecto, hablaremos de Roles. Estos roles representan a las entidades que interactúan con la plataforma o dependen de ella, pero su interacción va más allá de un simple flujo de acciones; definen el propósito y el contexto de uso del sistema.

Para este proyecto, se han identificado tres roles clave que abarcan todo el ecosistema de la solución, desde el consumo de información hasta su gestión y mantenimiento técnico.

- **Usuario Final (Empleado)**
    - **Descripción:** Es el rol principal y el consumidor final del chatbot. Representa a cualquier empleado de la organización con permisos para

utilizar la herramienta. Su objetivo principal es resolver dudas y obtener información precisa y contextualizada sobre los procedimientos, normativas y documentación interna de la empresa de manera rápida y eficiente, sin necesidad de buscar manualmente en múltiples repositorios.

- **Interacciones Clave**, que se van a relacionar con el Diagrama de Casos de Uso:
  - Iniciar y tener una conversación en Microsoft Teams.
  - Formular preguntas en lenguaje natural.
  - Interactuar con elementos de la interfaz, como la selección de categorías de conocimiento para acotar la búsqueda.
  - Recibir y procesar las respuestas presentadas en tarjetas adaptativas.
  - Consultar las fuentes originales de la información a través de los enlaces de las citas.
  - Proporcionar feedback explícito sobre la utilidad y calidad de las respuestas generadas por el sistema.
- **Implementación en el Sistema:** Este rol se corresponde con cualquier usuario de Azure Active Directory que tenga asignado el rol de aplicación específico que le concede permiso para interactuar con DocuBot, validado a través del servicio de Microsoft Graph.

Para visualizar de manera más clara las interacciones principales que el *Usuario Final* puede llevar a cabo con el sistema, se presenta a continuación un diagrama de Casos de Uso. Este diagrama modela las capacidades clave que el bot ofrece al usuario desde su perspectiva.

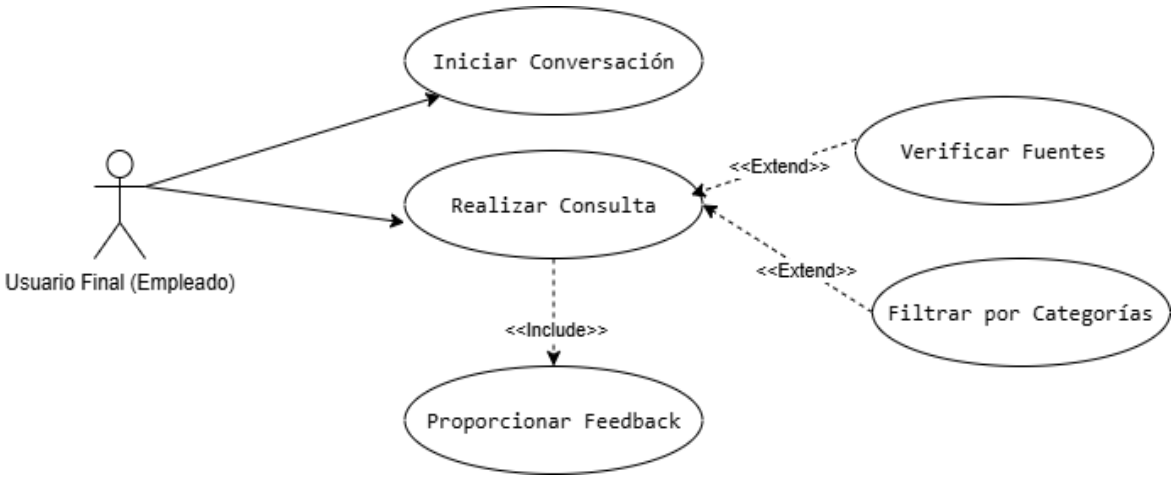


Figura 31. Casos de Uso Usuario Final

- **Administrador de Conocimiento (Rol de Negocio Necesario)**

- **Descripción:** Aunque este rol no interactúa directamente con la interfaz del chatbot, es funcionalmente crítico para la existencia y eficacia de la solución. Representa a la persona responsable de crear, mantener y asegurar la calidad de la base de conocimiento sobre la que opera el bot. El sistema RAG depende enteramente de la calidad y actualidad de la documentación que este rol gestiona.
- **Interacciones Clave (con el repositorio de documentos, no con el bot):**
  - Cargar nuevos documentos al repositorio central (Una biblioteca de SharePoint).
  - Actualizar versiones existentes de los procedimientos.
  - Retirar documentos obsoletos.
- **Implementación en el Sistema:** La funcionalidad del sistema depende de las acciones de este rol. Los procesos de ingesta de datos de DocBot están diseñados para consumir la información que este rol provee, convirtiéndola en un formato indexable y consultable por el motor de IA.

- **Desarrollador / Operador del Sistema (Administrador)**
  - **Descripción:** Es el rol técnico responsable del desarrollo, despliegue, monitorización y mantenimiento continuo de la plataforma DocuBot y toda su infraestructura en Azure.
  - **Interacciones Clave:**
    - Monitorizar los logs de la aplicación para detectar errores y analizar el comportamiento del sistema.
    - Analizar los datos de feedback y uso para identificar patrones, evaluar el rendimiento del modelo y planificar mejoras.
    - Ajustar y optimizar los prompts del sistema (ej: el prompt principal, el de generación de citas, el reescritor de querys..) para mejorar la calidad de las respuestas.
    - Gestionar el ciclo de vida de los recursos de Azure.

5.2 Especificación de Requisitos Funcionales

La especificación de los requisitos funcionales o capacidades del sistema del sistema se ha realizado siguiendo la metodología ágil descrita en el capítulo de Planificación. En lugar de utilizar un listado de requisitos tradicional, la funcionalidad se ha desglosado en **Épicas**, que representan grandes bloques de valor para la plataforma, no tanto al hacia el usuario como en el desarrollo software tradicional, y **Tareas**, que son las unidades de trabajo concretas que implementan dichas capacidades. Este enfoque

permite mantener una visión global del sistema a través de las Épicas, mientras se gestiona el desarrollo de forma granular con las Tareas.

A continuación, se presenta una visión global de las Épicas que definen las capacidades funcionales de la plataforma y, posteriormente, se desglosarán algunas de las más relevantes para ilustrar el proceso de especificación

5.2.1. Visión Global: Épicas Funcionales

Es el resumen de alto nivel de los requisitos funcionales. Las Épicas encapsulan las principales capacidades del sistema desde una perspectiva funcional. La siguiente tabla resume las diferentes Épicas que han guiado el desarrollo del proyecto.

ID-ÉPICA	Nombre Épica	Descripción
EPIC-01	Gestión Inicial de la Conversación	Define todas las funcionalidades iniciales que gobiernan la interacción del usuario con el bot a nivel de sesión: desde el inicio de una nueva conversación, el manejo del estado hasta el reinicio explícito del diálogo.
EPIC-02	Motor de Razonamiento y Respuesta (RAG)	Constituye el núcleo inteligente del sistema. Abarca el flujo completo desde que el usuario formula una pregunta hasta que el sistema genera una respuesta precisa, contextualizada y verificable, basada en la base de conocimiento interna.
EPIC-03	Sistema de Feedback	Engloba las capacidades que permiten al usuario valorar la calidad de las respuestas
EPIC-04	Persistencia de datos conversacionales	Define como el sistema registrar de forma persistente y estructurada tanto el feedback como los detalles de cada interacción para su posterior análisis y mejora.
EPIC-05	Integración y Seguridad	Define cómo el bot se integra de forma nativa y segura en el ecosistema de Microsoft Teams, incluyendo la autenticación de usuarios y la validación de permisos de acceso basada en roles de la organización.
EPIC-06	Gestión de la Base de Conocimiento	Describe las capacidades subyacentes del sistema para procesar, indexar y mantener actualizada la base de conocimiento documental. Aunque el proceso de carga de documentos es externo, esta épica cubre la parte del sistema que los ingiere y los prepara para ser consultados vía RAG.
EPIC-07	Robustez y Optimización	Añade pequeñas funcionalidades que se han ido recopilando, como gestión de la expiración por inactividad, adicción de un nuevo índice de

		búsqueda, opcionalidad del Feedback a solo manita abajo.
EPIC-08	Despliegue	Creación de todos los recursos definitivos necesarios en Azure y ajuste necesario, despliegue en Azure.

Tabla 8. Épicas Funcionales

5.2.2. Desglose de Épicas en Tareas Funcionales

Para ilustrar cómo cada Épica se materializa en funcionalidades concretas, a continuación se detallan las Tareas más representativas. Cada una de estas Tareas representa un requisito funcional específico del sistema, si queremos verlo como un desarrollo de software tradicional. Las Tareas se han reformulado siguiendo el formato de Historias de Usuario para centrar el desarrollo en el valor aportado.

Es importante destacar que, especialmente en las tareas técnicas internas del sistema (como las del RAG), el "rol" de la historia de usuario no siempre es el usuario final. A menudo, el rol es otro componente del propio sistema que "necesita" un resultado del componente anterior para poder realizar su trabajo. Este enfoque permite modelar las interacciones y dependencias internas del sistema, asegurando que cada pieza de la arquitectura se desarrolle con un propósito claro dentro del flujo de procesamiento general.

Épica 2: Motor de Razonamiento y Respuesta (Pipeline RAG)

Esta Épica es la parte más importante de DocBot a nivel técnico, como en carga de trabajo y hace posible la generación de la respuesta de la mejor manera posible. Su implementación se descompone en un pipeline de Tareas a menudo secuenciales donde la salida de una es la entrada de la siguiente, conformando el proceso de RAG avanzado que forma el aplicativo

- **Tarea 2.1: Mejora de la Consulta de Búsqueda**
    - **Descripción Funcional:** El proceso comienza optimizando la pregunta del usuario. El componente toma la entrada en lenguaje natural y el historial de la conversación y, mediante un LLM, la transforma en una consulta semántica precisa, ideal para un motor de búsqueda vectorial y que mejora claramente frente a la query directamente de usuario.
    -

TAREA 2.1	**MEJORA DE LA CONSULTA DE BÚSQUEDA**
DESCRIPCIÓN	**Como** el *Módulo de Búsqueda* **quiero** recibir una consulta optimizada en lugar del texto en bruto del usuario, **para** poder ejecutar una búsqueda vectorial y de texto con la máxima precisión en los índices.

DEPENDENCIAS	Ninguna dependencia técnica interna (es el primer paso del pipeline de razonamiento).
ACEPTACIÓN	- El componente debe ser invocado con la pregunta actual del usuario y el historial de la conversación. - La llamada al LLM para la reformulación de la consulta debe completarse con éxito. - La consulta generada debe ser una cadena de texto sintácticamente válida y semánticamente relevante.
SUB-TAREAS TÉCNICAS	- Implementar la clase de reescritor de consultas y su lógica de llamada a la API de OpenAI (3 P.H). - Diseñar y validar el prompt del sistema para la tarea de reformulación de consultas (1 P.H).

Tabla 9. Tarea Funcional 2.1 - Mejora de la Consulta de Búsqueda

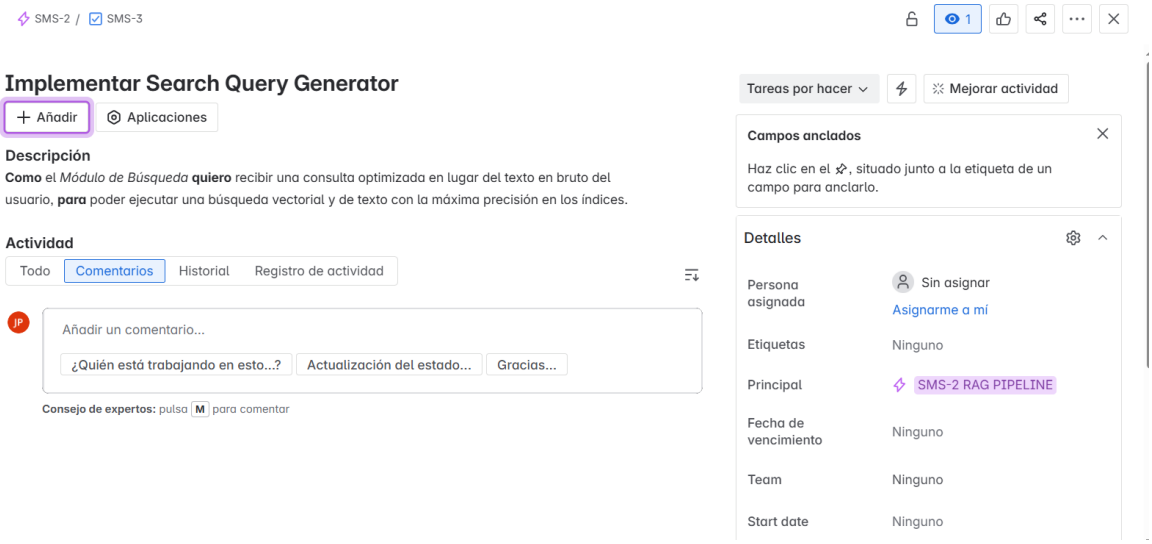


Figura 32. Ejemplo de Tarea Técnica en JIRA

- **Tarea 2.2: Búsqueda Híbrida y Recuperación de Contexto**
    - **Descripción Funcional:** Una vez optimizada la consulta, se realiza una búsqueda híbrida en los dos índices de Azure AI Search (resúmenes y texto completo), aplicando los filtros de categoría seleccionados. Los resultados se combinan y clasifican para obtener los fragmentos de documento más relevantes.

TAREA 2.2	**BÚSQUEDA HÍBRIDA Y RECUPERACIÓN DE CONTEXTO**

DESCRIPCIÓN	**Como** el *Generador de Respuestas*, **quiero** recibir un conjunto de fragmentos de texto relevantes y verificados, **para** tener el contexto necesario para construir una respuesta precisa y fundamentada, minimizando el riesgo de "alucinaciones".
DEPENDENCIAS	Tarea 2.1: Mejora de la Consulta de Búsqueda (requiere la consulta optimizada).
ACEPTACIÓN	- El buscador de la base vectorial debe ejecutar una búsqueda en ambos índices (resúmenes y texto completo). - La búsqueda debe aplicar correctamente los filtros de categoría seleccionados por el usuario. - El sistema de ranking debe combinar y ordenar los resultados de ambos índices de forma coherente. - Se debe recuperar una lista estructurada de documentos, incluyendo sus metadatos y URLs.
SUB-TAREAS TÉCNICAS	- Implementar la lógica de búsqueda híbrida (5 P.H). - Diseñar el algoritmo de ranking y combinación de resultados (3 P.H).

Tabla 10. Tarea Funcional 2.2 - Búsqueda Híbrida y Recuperación de Contexto

- **Tarea 2.3: Generación de Respuesta Contextualizada**
    - **Descripción Funcional:** Con los documentos recuperados como único contexto, se invoca al modelo de lenguaje principal. El prompt del sistema le instruye de forma estricta a sintetizar una respuesta que se base exclusivamente en la información proporcionada.
    - **Formato de Tarea (JIRA):**
        - **Como** el *Orquestador del Flujo*, **quiero** recibir un borrador de respuesta coherente y basado en el contexto, **para** poder iniciar el paso final de verificación y generación de citas.

- **Tarea 2.4: Generación de Citas Fiables y Presentación**
    - **Descripción Funcional:** El borrador de la respuesta y la lista de documentos fuente se entregan al generador de citas y a su vez verificador. Este componente, mediante una llamada a un LLM especializado, identifica qué documentos específicos respaldan las afirmaciones de la respuesta. Finalmente, el orquestador del flujo ensambla la respuesta y las citas en una tarjeta adaptativa para presentarla al usuario.

TAREA 2.4	**GENERACIÓN DE CITAS FIABLES Y PRESENTACIÓN**
DESCRIPCIÓN	**Como** un *Usuario Final*, **quiero** que la respuesta que veo en Teams esté respaldada por referencias claras y precisas, **para** poder confiar en la información y verificarla fácilmente.

DEPENDENCIAS	Tarea 2.3: Generación de Respuesta Contextualizada (requiere el borrador de la respuesta).
ACEPTACIÓN	- El componente debe ser invocado con la respuesta generada y la lista de documentos fuente. - El generador de citas debe devolver un JSON estructurado con los índices de los documentos citados. - La función de crear la salida debe procesar el JSON y renderizar correctamente los botones de las citas en la tarjeta adaptativa.
SUB-TAREAS TÉCNICAS	- Implementar generador de citas y su lógica de llamada a la API (2 P.H). - Orquestar la llamada para integrar la generación de citas en el flujo (1 P.H). - Actualizar la tarjeta adaptativa para manejar dinámicamente la sección de referencias (2 P.H).

Tabla 11. Tarea Funcional 2.4 - Generación de Citas Fiables y Presentación Final

Épica 3: Sistema de Feedback y Auditoría

Esta Épica asegura que el sistema sea transparente, medible y mejorable a lo largo del tiempo, cerrando el ciclo de la interacción con el usuario.

- **Tarea 3.1: Captura de Feedback del Usuario**
    - **Descripción Funcional:** La interfaz de usuario incluye controles de feedback (pulgar arriba/abajo). Una valoración negativa puede desencadenar la presentación de una segunda tarjeta que solicita comentarios detallados sobre la calidad y relevancia de la información.
    - **Formato de Tarea (JIRA):**
        - **Como** un *Usuario Final*, **quiero** una forma sencilla e intuitiva de valorar las respuestas del bot, **para** poder contribuir a la mejora de su precisión futura y utilidad general.

- **Tarea 3.2: Almacenamiento Persistente de Interacciones**
    - **Descripción Funcional:** El Orquestador es responsable de la persistencia de los datos. Cada interacción completa (pregunta, respuesta, citas, métricas de uso de tokens y metadatos del usuario) se guarda como un registro estructurado en Cosmos DB. Cuando un usuario proporciona feedback, el sistema localiza el registro original y lo actualiza, enriqueciéndolo con la valoración del usuario.

TAREA 3.2	**ALMACENAMIENTO PERSISTENTE DE INTERACCIONES**

DESCRIPCIÓN	**Como** un *Operador del Sistema*, **quiero** que cada interacción y su feedback asociado se almacenen de forma fiable y estructurada, **para** poder realizar análisis de rendimiento, identificar áreas de mejora y llevar a cabo auditorías de uso.
DEPENDENCIAS	- Tarea 2.4: Generación de Citas Fiables y Presentación Final (requiere el resultado completo del turno). - Tarea 3.1: Captura de Feedback del Usuario (requiere los datos del feedback).
ACEPTACIÓN	- La función saveTurn debe guardar un nuevo documento en Cosmos DB con todos los campos requeridos tras una respuesta exitosa. - La función updateTurn debe localizar el documento existente por su turnId y actualizarlo con los datos del feedback del usuario. - El esquema de datos en Cosmos DB debe ser coherente y no permitir registros incompletos.
SUB-TAREAS TÉCNICAS	- Diseñar el esquema del documento JSON para los turnos en Cosmos DB (2 P.H). - Implementar funciones de guardar turno y actualizar turno, incluyendo la lógica de escritura y actualización en Cosmos DB (3 P.H).

Tabla 12. Tarea Funcional 3.2 - Almacenamiento Persistente de Interacciones

5.3 Especificación de Requisitos No Funcionales

Los **Requisitos No Funcionales (RNF)** definen las cualidades y restricciones operativas del sistema. No describen una funcionalidad específica, sino los atributos de calidad que garantizan que la plataforma sea eficiente, segura, fiable y usable.

Este apartado es especialmente importante en este proyecto. Mientras que los requisitos funcionales describen *qué* hace el sistema, los no funcionales definen *cómo* lo hace. Para un sistema de IA, cualidades como la fiabilidad, la seguridad y el rendimiento no son secundarias, sino que son fundamentales para que el sistema sea aceptado y útil en un entorno corporativo como el que se quiere conseguir.

Categoría	ID	Requisito No Funcional
Rendimiento	RNF-01	**Respuesta en Tiempo Real.** El tiempo total de respuesta, desde que el usuario envía una pregunta hasta que recibe la tarjeta adaptativa con la respuesta, debe mantenerse en un rango aceptable (media inferior a 30 segundos) para no romper la fluidez de la conversación.
Seguridad	RNF-02	**Control de Acceso Basado en Roles.** El acceso a las funcionalidades del bot debe estar estrictamente restringido a los empleados autorizados de la organización.
	RNF-03	**Gestión Segura de Credenciales.** Todas las claves de API, cadenas de conexión y otros secretos no deben estar

		expuestos en el código fuente ni en archivos de configuración no seguros.
Escalabilidad	RNF-04	**Escalabilidad de Carga de Usuarios.** La arquitectura debe ser capaz de gestionar un aumento en el número de usuarios concurrentes sin una degradación significativa del rendimiento.
	RNF-05	**Escalabilidad de la Base de Conocimiento.** El sistema debe mantener su rendimiento de búsqueda y recuperación a medida que el volumen de documentos en la base de conocimiento crece.
Fiabilidad y Precisión	RNF-06	**Minimización de Alucinaciones.** La respuesta generada por el LLM debe estar estrictamente fundamentada en la información extraída de la base de conocimiento interna.
	RNF-07	**Precisión de las Citas.** Las referencias a los documentos fuente deben ser correctas y apuntar a los fragmentos que realmente respaldan las afirmaciones de la respuesta generada.
Usabilidad	RNF-08	**Interfaz Intuitiva y Clara.** La interacción con DocuBot debe ser sencilla y guiar al usuario, incluso si no tiene experiencia previa con sistemas de IA.
Trazabilidad	RNF-09	**Registro Completo de Interacciones.** Todas las interacciones significativas deben ser registradas para permitir la depuración, el análisis de uso y la mejora continua del sistema.

Tabla 13. Especificación de Requisitos No Funcionales

5.4 Requisitos de Información

Los Requisitos de Información describen las principales entidades de datos que el sistema debe almacenar y gestionar para dar soporte a sus funcionalidades. Aunque el chatbot no expone una base de datos tradicional al usuario, su funcionamiento depende de un modelo de información bien definido y distribuido en varios servicios de la plataforma Azure.

A continuación, se describen las entidades de información clave del proyecto.

- **RI-01: Estado de Conversación**
    - **Descripción:** Representa el estado temporal y volátil de una única sesión de conversación entre un usuario y DocuBot. Esta entidad es fundamental para mantener el contexto a lo largo de un diálogo.
    - **Implementación:** Almacenado en Cosmos DB, en el contenedor de conversaciones.
- **RI-02: Registro de Turno**

    - **Descripción:** Constituye el registro persistente y detallado de cada interacción completa de pregunta-respuesta. Esta entidad es la base para la auditoría, el análisis de rendimiento y el ciclo de feedback.
    - **Implementación:** Almacenado en Cosmos DB, en un contenedor separado (el de turnos).

- **RI-03: Base de Conocimiento Indexada**
    - **Descripción:** Es la representación procesada y optimizada de los documentos fuente de la organización. No contiene los documentos originales, sino fragmentos de texto (chunks), resúmenes, metadatos y, crucialmente, los vectores de embeddings que permiten la búsqueda semántica.
    - **Implementación:** Almacenado en Azure AI Search, distribuido en dos índices separados para optimizar la búsqueda híbrida.
- **RI-04: Configuración y Secretos del Sistema**
    - **Descripción:** Representa toda la información de configuración sensible y no sensible necesaria para que la aplicación se inicie y se conecte a los diferentes servicios de Azure.
    - **Implementación:** Almacenado de forma segura en Azure Key Vault.

Modelado de Datos

En este capítulo se describirán y explicarán los diferentes modelos y estructuras de datos que son fundamentales para el funcionamiento de DocuBot y que continúan con lo introducido brevemente en el capítulo de Análisis Funcional, en la parte de Requisitos de Información.

6.1 Contexto en Sistemas RAG

El modelado de datos es una parte esencial en el desarrollo de cualquier sistema de software, incluidas las aplicaciones basadas en IA como la nuestra, que utiliza un patrón de RAG. A diferencia de los sistemas transaccionales tradicionales donde el modelado se centra en bases de datos relacionales (con Diagramas Entidad/Relación) o NoSQL (con diagramas de colecciones/documentos para datos de aplicación), en un sistema RAG, el "modelado" se expande más allá de las bases de datos transaccionales tradicionales para abarcar cómo se prepara, se indexa y se utiliza el conocimiento para alimentar los modelos de lenguaje:

1. **El Corpus de Conocimiento:** El conjunto de documentos y metadatos que constituyen la "memoria" del sistema. El modelado de este corpus es crítico para la fase de Recuperación (Retrieval), ya que una estructura bien diseñada permite al sistema encontrar la información más relevante para la pregunta del usuario.
2. **El Índice de Búsqueda:** El esquema optimizado para la recuperación eficiente de información relevante de dicho corpus.
3. **Los Datos Operacionales:** La información generada durante la ejecución del bot, como el estado de la conversación, el historial de interacciones, el feedback del usuario y las métricas de uso. Este modelo es vital para la operación, auditoría y mejora continua del sistema.
4. **El Flujo de Datos:** Cómo estos diferentes tipos de datos se interconectan y transforman a lo largo del ciclo de vida de una consulta.

Si bien el modelado de datos sigue siendo crucial, es importante destacar por qué un Diagrama Entidad-Relación (E/R) tradicional, aunque útil en sistemas transaccionales, resulta insuficiente para capturar la complejidad de una arquitectura RAG como la de DocuBot. Las razones principales son: la naturaleza distribuida y heterogénea de los almacenes de datos (NoSQL, Search Index, Blob Storage); las relaciones conceptuales en lugar de claves; y el énfasis en el flujo dinámico de datos en lugar de un modelo estático. Por ello, en esta memoria se opta por un diagrama híbrido que combina la notación de componentes con la de entidades, complementado por diagramas de flujo y secuencia en este y capítulos posteriores para ofrecer una visión completa tanto de los datos en reposo como en movimiento.

6.2 Modelo Lógico de Datos

Una vez establecido el contexto del modelado de datos en sistemas RAG, esta sección profundiza en las estructuras de datos específicas que sustentan a DocuBot. A diferencia de un sistema monolítico, la arquitectura de DocuBot se basa en un conjunto de servicios de Azure especializados, cada uno con su propio modelo de datos optimizado para una función concreta: desde el almacenamiento de conocimiento preprocesado hasta el registro de interacciones en tiempo real.

Para este proyecto, se ha optado por una arquitectura de datos distribuida utilizando los servicios de Azure más adecuados para cada propósito: Azure AI Search para el corpus de conocimiento y Azure Cosmos DB para los datos operacionales. Así como Microsoft Graph para poder manejar a los usuarios que utilicen el sistema.

A continuación, se desglosa el modelo lógico de cada uno de estos componentes, mostrando cómo su diseño individual contribuye al funcionamiento global del sistema.

6.2.1 Origen de la Base Documental (SharePoint)

El fundamento de la inteligencia de DocuBot reside en su documentación, la cual está compuesta por la documentación interna de la empresa o cualquier corpus documental que se quiera explotar. Antes de ser procesados e ingeridos por el sistema RAG, estos documentos existen en su formato original y se encuentran como ya se ha comentado centralizados en una ubicación específica en SharePoint, siguiendo dentro del ecosistema corporativo de Microsoft, sirviendo como origen único de documentación para el sistema.

Características Relevantes del Corpus

- **Contenido Estructurado y No Estructurado:** Los documentos combinan texto no estructurado (párrafos de texto libre) con elementos altamente estructurados como tablas, listas numeradas y jerarquías de títulos. El éxito del sistema depende de su capacidad para interpretar ambos tipos de contenido en conjunto.
- **Riqueza Visual:** Una parte importante del conocimiento no está en el texto, sino en diagramas de flujo, esquemas de arquitectura, gráficos de resultados o imágenes de productos. Esta característica es la principal justificación para la implementación de una arquitectura RAG multimodal como DocuBot.
- **Multilingüismo:** El corpus puede contener documentos en español, inglés u otros idiomas, un factor a considerar en el preprocesamiento.
- **Metadatos Implícitos:** El nombre del archivo, las propiedades del documento actúan como metadatos valiosos. Aunque no todos se explotan en la versión actual, existe un campo de si el documento debe ser procesado o no por

DocuBot y el Grupo de Documentación se extrae y se utiliza para la realizar un filtrado de datos.

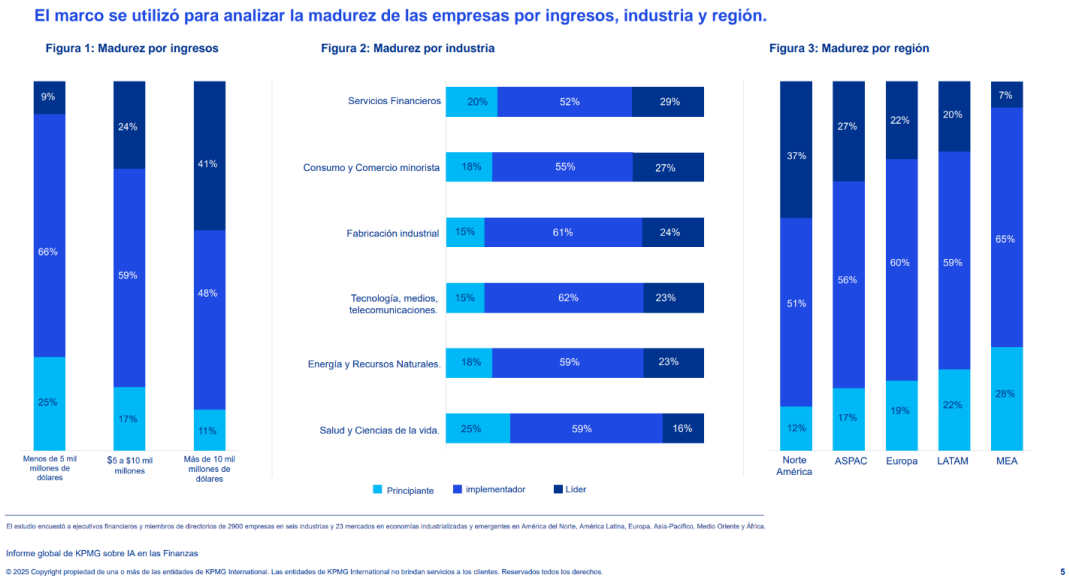


Figura 33. Ejemplo de Documentación Compleja

La selección y curación de este corpus se alinea con los objetivos iniciales del proyecto y define el alcance y los límites del conocimiento de DocuBot. El proceso de ingesta, que se detalla en el Capítulo 8, es el puente tecnológico que transforma estas fuentes originales en la base documental procesada que se describe a continuación.

6.2.2 Base Documental Procesada (Azure Blob Storage)

Una vez que los documentos del corpus original son procesados por el pipeline de ingesta, el resultado no se almacena como los archivos originales, sino como una colección de *Blobs* en Azure Blob Storage. Este almacenamiento representa una capa de datos intermedios, optimizada para ser consumida de manera eficiente por los componentes posteriores del RAG, como Azure AI Search y el LLM final.

La estructura en Blob Storage está organizada en contenedores, cada uno con un propósito específico.

Contenedor de Imágenes
Este contenedor almacena una representación visual de cada página de los documentos fuente, lo que añada la capacidad multimodal del sistema

- **Formato del Blob:** Imagen en formato **JPEG (.jpg)**.
- **Nombre:** [nombre_documento_original]_page_[numero_pagina].jpg (ej. PR-01-Control_Documental_v4.1_page_5.jpg). Esta convención de nomenclatura es determinista y crucial para la posterior correlación de artefactos.

Contenedor de Texto Completo

Este contenedor almacena el contenido textual extraído de cada página en un formato limpio y estructurado. Este es el texto optimizado para la fase de aumentación que se inyecta en el prompt del modelo de lenguaje para que genere la respuesta.

- **Formato del Blob:** Archivo de texto plano que contiene una estructura **JSON (.json)**.
- **Nombre:** Sigue la misma convención que las imágenes para una correlación directa: [nombre_documento_original]_page_[numero_pagina].json.
- **Contenido del JSON:** El contenido del blob es un objeto JSON simple que contiene todo el texto extraído de la página, así como descripciones de los elementos visuales, facilitando su posterior procesamiento.

Contenedor de Texto Plano

Este contenedor alberga el mismo texto extraído del contenedor anterior de cada página en su formato más simple, optimizado para la ingesta masiva y eficiente por parte de Azure AI Search, sin necesidad de parsear el JSON.

- **Formato del Blob:** Archivo de texto plano **(.txt)**.
- **Nombre**: [nombre_documento_original]_page_[numero_pagina].txt.

Contenedor de Resúmenes y Preguntas

Este contenedor almacena el resultado del procesamiento realizado por un LLM durante la fase de preprocesado. Para cada página, se genera un resumen y un conjunto de preguntas hipotéticas. El resumen condensa la información clave, mientras que las preguntas hipotéticas anticipan las posibles consultas de los usuarios

- **Formato del Blob:** Archivo de texto plano con contenido en formato **JSON (.json)**.
- **Nombre:** [nombre_documento_original]_page_[numero_pagina].json.
- **Contenido del JSON:** El objeto JSON contiene los datos enriquecidos que serán indexados en los campos primarios de búsqueda vectorial. Un campo de summary y otro de questions que será una lista.

Relación Conceptual entre los Blobs

Existe una relación conceptual 1 a 1 entre la imagen de una página y sus correspondiente JSONs y TXT. El sistema materializa esta relación a través de la convención de nomenclatura compartida, así como los metadatos utilizados que contienen las urls de los otros blobs.

Dentro de los metadatos que tiene cada Blob que se detallarán en capítulos siguientes se puede avanzar en esta fase que deberá haber aparte de las URL mencionadas para la relación de los diferentes Blobs de la misma página, el grupo de Documentación, número de página procesada, URL al documento original en SharePoint…

6.2.3 Índice para el RAG (Azure AI Search)

El corazón de la fase de recuperación de información reside en los índices de Azure AI Search. Estos no son simplemente un almacén, sino una estructura de datos optimizada para búsquedas complejas (textuales, vectoriales e híbridas). El éxito de DocuBot depende directamente de la calidad de este modelo.

Hemos definido una estructura de documento que se utiliza para poblar dos índices principales, uno optimizado para resúmenes y otro para texto completo. La estructura para el de resúmenes y que se puede extrapolar al del texto completo es la siguiente:

Campo	Tipo de Dato	Propósito en el Sistema RAG
chunk_id	String	Identificador único para cada fragmento de texto procesado.
parent_id	String	Identificador del documento original del que se extrajo el fragmento.
summary	String	Un resumen generado por IA del contenido de la página.
questions	String	Preguntas hipotéticas que podrían ser respondidas por este fragmento, generadas por IA para mejorar la búsqueda semántica.
title	String	Título del fragmento original
summary_vector	Collection(Edm.Single)	**Vector de embedding** del campo summary. Usado en el índice de resúmenes para la búsqueda vectorial.
questions_vector	Collection(Edm.Single)	**Vector de embedding** del campo questions. Usado en el índice de resúmenes para la búsqueda vectorial.
document_name	String	Nombre del archivo original. Usado para mostrar en las citas.
document_url	String	URL que apunta al documento original (PDF, etc.) en Azure Blob Storage.
image_blob_url	String	URL que apunta a la imagen (JPG) de la página en Azure Blob Storage.
page_number	String	Número de página del documento original de donde se extrajo el fragmento.
system	Collection(Edm.String)	Categoría(s) o sistema(s) a los que pertenece el documento. Clave para el filtrado.

Tabla 14. Esquema del Documento en Azure AI Search

6.2.4 Datos Operacionales y de Auditoría (Azure Cosmos DB)

Para el registro de las interacciones y el estado de las conversaciones, se utiliza Azure Cosmos DB, en su opción de base NoSQL. Este enfoque tiene como propósito de los datos **registros de auditoría y feedback**.

Cada vez que un usuario completa un ciclo de pregunta-respuesta, se crea un documento en la colección turns (*cosmos-db-container-turns*). Sin embargo, la aplicación se adhiere a una estructura consistente para garantizar la integridad de los datos y facilitar su posterior análisis. El esquema de este documento es el siguiente:

Campo (Atributo)	Tipo de Dato	Obligatorio	Descripción / Propósito
id	String	Sí	Identificador único del documento de turno, generado por la aplicación.
conversationId	String	Sí	Identificador de la conversación a la que pertenece este turno. Permite agrupar las interacciones.
country	String / Null	Sí	País del usuario, obtenido de Microsoft Graph para análisis de uso. Puede ser nulo si no está disponible.
companyName	String	Sí	Nombre de la compañía del usuario, obtenido de Microsoft Graph.
department	String	Sí	Departamento del usuario, obtenido de Microsoft Graph.
selectedCategories	Array de Strings	Sí	Contiene las categorías de documentación seleccionadas por el usuario al inicio. El array estará vacío si no se aplicó ningún filtro.
input	Objeto	Sí	Objeto que representa la entrada (prompt) completa enviada al modelo de IA.
input.content	Array	Sí	Contenido completo del prompt, que puede incluir fragmentos de texto e imágenes en formato Base64.
output	Objeto	Sí	Objeto que representa la salida (respuesta) completa recibida del modelo de IA.
feedbackUsage	Objeto	Sí	Objeto con las métricas de uso de tokens de la API de OpenAI para este turno.
citedDocumentNames	Array de Strings	Sí	Array con los nombres de los documentos fuente que el modelo citó en su respuesta.
userFeedback	Objeto	Sí	Objeto que almacena el feedback proporcionado por el usuario sobre la calidad de la respuesta.
userFeedback.useful	Boolean / Null	No	Valor booleano que indica la utilidad de la respuesta (true para útil, false

			para no útil). Es nulo si el usuario no proporciona feedback.
userFeedback.Quality	String / Null	No	Calificación numérica (1-10) de la calidad, proporcionada opcionalmente por el usuario.
userFeedback.IsDocumentationGiven	Boolean / Null	No	Valoración opcional sobre si la documentación citada fue relevante.
userFeedback.commentOverall	String / Null	No	Comentario de texto libre opcional proporcionado por el usuario.

Tabla 15. Diccionario Datos colección Turns

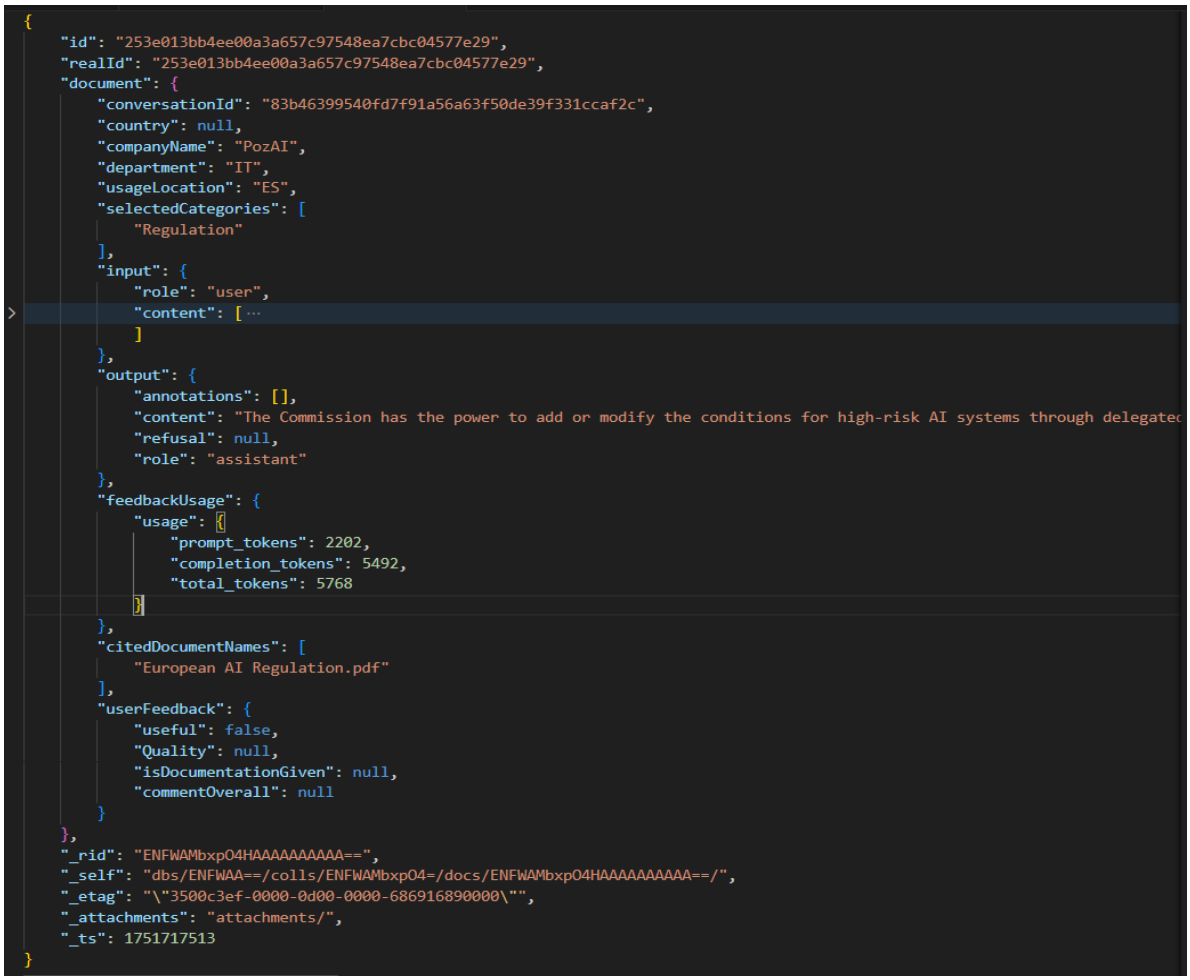


Figura 28. Ejemplo de documento de Turno en Cosmos DB

Para mantener el histórico de mensajes dentro de la misma conversación y dotar así de memoria a DocuBot se genera otro documento en el contenedor de conversaciones donde existirá un documento por usuario (*cosmos-db-container-conversations*). Estos documentos pueden tener varios turnos y se reinician una vez el usuario comienza una nueva conversación.

Con la combinación de estos dos contenedores se puede tener una visión completa para poder generar un futuro informe de uso, auditoría y de calidad de las respuestas en una herramienta de analítica como Power BI.

6.2.5 Datos de Autorización y Perfil de Usuario (Microsoft Graph)

A diferencia de los almacenes de datos persistentes como Azure AI Search o Cosmos DB, que son gestionados directamente por la aplicación, el sistema también consume datos en tiempo de ejecución desde una fuente externa: Microsoft Graph. Este servicio actúa como una API unificada para acceder a los datos de la organización en Microsoft 365, incluido el directorio de usuarios de *Azure Active Directory* (Azure AD).

Para el enriquecimiento de los registros de auditoría, así como los permisos de la aplicación. La selección de estos campos se ha realizado cuidadosamente para aportar valor analítico sin comprometer la privacidad ni almacenar información personal sensible innecesaria y así cumplir con la normativa respecto a la seguridad de los datos personales.

La estructura de datos consumida es la siguiente:

Campo	Tipo de Dato	Campo en Microsoft Graph	Propósito en el Sistema
companyName	String	companyName	En entornos multi-empresa dentro del mismo tenant, permite segmentar el uso y el feedback por compañía.
department	String	department	Facilita la identificación de los departamentos que más utilizan el bot, lo que puede guiar futuras mejoras o formaciones.
usageLocation	String	usageLocation	Aporta un dato de localización estandarizado.

Tabla 16. Esquema de Datos de Perfil de Usuario

Una vez descritas las estructuras de datos individuales que residen en cada uno de los servicios de Azure, es fundamental visualizar cómo estas se interconectan para formar un sistema. A diferencia de un modelo relacional tradicional, las relaciones en esta arquitectura son una combinación de dependencias lógicas, referencias por URL y flujos de datos que se materializan durante la ejecución de la aplicación.

La siguiente figura presenta un diagrama híbrido que consolida el modelo de datos completo del sistema, mostrando no solo las entidades clave y sus atributos más relevantes, sino también su ubicación física dentro de la arquitectura de Azure y las relaciones conceptuales que las unen.

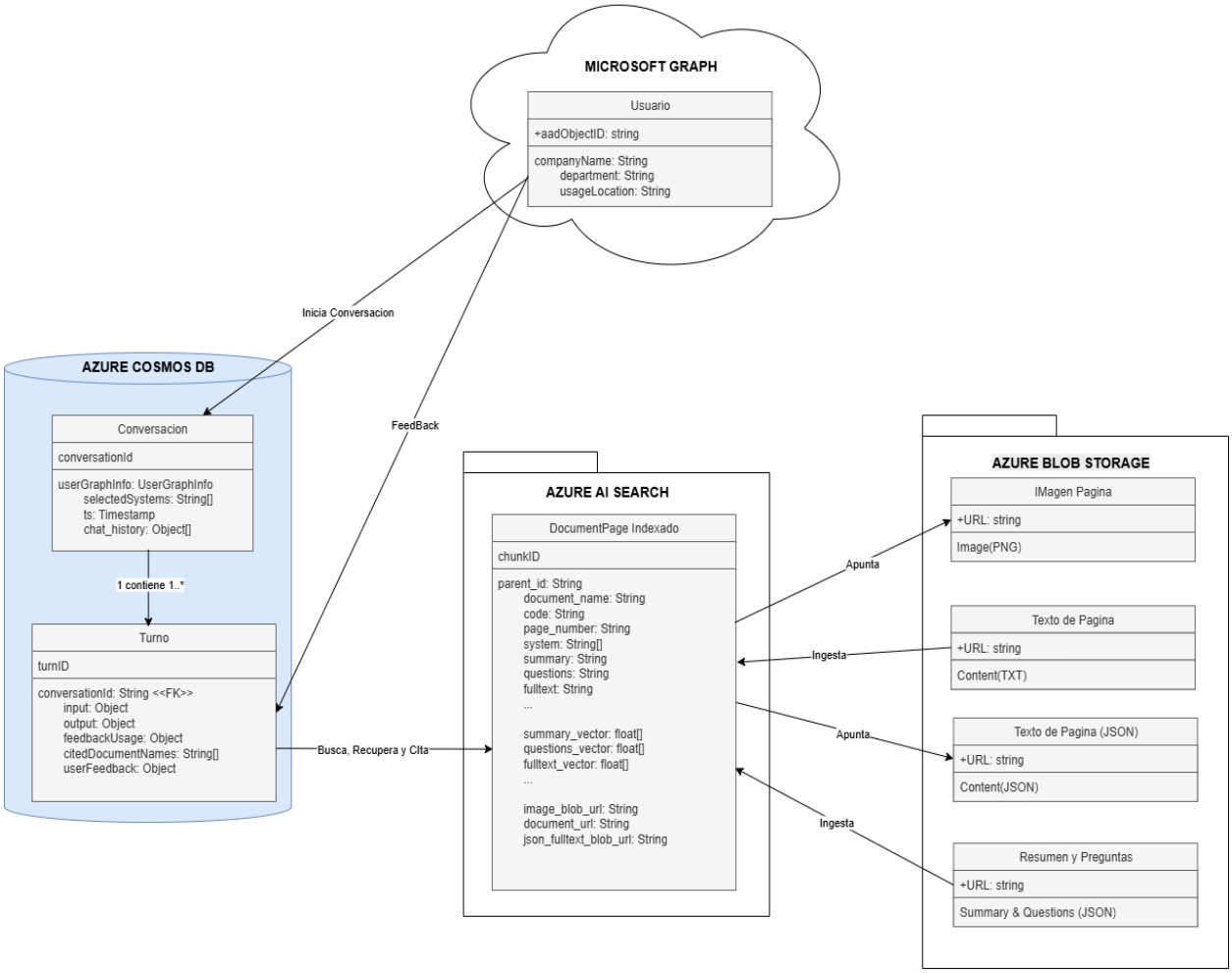


Figura 29. Modelado Lógico de Datos No Relacional

6.3 Flujo y Relaciones Conceptuales de los Datos

A diferencia de un modelo relacional tradicional con claves foráneas, las "relaciones" en este sistema RAG son a menudo conceptuales y se materializan a través del flujo de la aplicación, como se ha podido observar en el diagrama anterior.

1. Un Usuario (identificado por aadObjectId) inicia una Conversación.
2. La consulta del usuario, junto con los sistemas de Documentación del estado, se usa para buscar en Azure AI Search (modelo DocumentPage).
3. Las URLs de DocumentPage (image_blob_url, json_fulltext_blob_url) apuntan a los respectivos artefactos en Azure Blob Storage.
4. El contenido recuperado de Blob Storage y AI Search forma el contexto para el LLM.
5. La interacción completa (pregunta, respuesta, citas, feedback) se registra como un Turno en turnsStorage, vinculado a una Conversación.

6. Los citedDocumentNames en el Turno se derivan del campo document_name de las DocumentPage recuperadas.

Diseño y Arquitectura

El presente capítulo tiene como objetivo describir en profundidad el diseño y la arquitectura sobre los que se fundamenta el desarrollo de DocuBot como asistente basado en IA Generativa.

7.1 Introducción

La construcción de una solución de IA conversacional difiere sustancialmente del desarrollo de aplicaciones web tradicionales, ya que el diseño no se centra únicamente en la estructura de la información y la interfaz visual, sino, y de manera primordial, en la dinámica de la interacción, la fiabilidad de las respuestas y la orquestación de múltiples servicios cognitivos.

Se detallarán los pilares de la aplicación, comenzando por el diseño de la experiencia conversacional, un concepto que va más allá de la interfaz de usuario (UI) para abarcar el flujo completo de la interacción entre el usuario y DocuBot. Posteriormente, se describirá la arquitectura a diferentes niveles de abstracción, desde la visión conceptual que gobierna el sistema hasta los componentes lógicos de software y su despliegue físico en la nube de Microsoft Azure.

7.2 Experiencia Conversacional e Interfaz de Usuario

A diferencia de las aplicaciones tradicionales donde la Experiencia de Usuario (UX) se enfoca en la apariencia y navegación a través de pantallas web o móviles, en una aplicación conversacional el diseño se centra en la calidad, eficacia y confianza de la interacción. La interfaz principal no es un layout de página y lo visual de ella, sino el propio diálogo.

Conviene volver a recordar algo comentado durante el resto de capítulos, y es que se ha elegido Microsoft Teams como plataforma siendo una decisión estratégica fundamental para el diseño de la experiencia. Al ser la herramienta de comunicación y colaboración estándar en un gran número de corporaciones, los usuarios finales ya poseen una alta familiaridad con su entorno. Esto elimina la curva de aprendizaje asociada a una nueva aplicación, permitiendo que los usuarios interactúen con el chatbot de manera intuitiva desde el primer momento. El sistema se integra de forma nativa en el flujo de trabajo diario del empleado.

Este enfoque está en sintonía con recientes comentarios de líderes tecnológicos como Satya Nadella, CEO de Microsoft, quien ha pronosticado que la era de las aplicaciones SaaS tal como las conocemos está llegando a su fin, dando paso a los agentes de IA

como la nueva interfaz de usuario [51]. Según Nadella, la lógica de negocio se trasladará a estos agentes inteligentes que operarán a través de diversas aplicaciones.

Esta elección de plataforma influye directamente en los componentes de la interfaz. Microsoft Teams obliga al uso de Tarjetas Adaptativas (Adaptive Cards), un formato de fragmentos de UI s en formato JSON que se renderizan de forma nativa dentro del cliente de Teams. Este enfoque permite crear experiencias ricas, interactivas y visualmente consistentes con el entorno que el usuario ya conoce y utiliza. En lugar de desarrollar una interfaz desde cero, se aprovecha un framework robusto, centrando los esfuerzos en el diseño del contenido y la interacción de dichas tarjetas.

Por ello, para este proyecto, se habla de Experiencia Conversacional (CX), un concepto que engloba tanto los principios de la interacción como los componentes visuales que la soportan.

7.2.1 Características de la Experiencia Conversacional

La filosofía de diseño detrás de la experiencia conversacional se basa en una poderosa metáfora: el chatbot como un nuevo compañero de un equipo experto. Hasta ahora, un empleado que necesitaba consultar información sobre un tema específico dependía de la disponibilidad de un compañero experto en la materia. Este proyecto introduce al chatbot como un nuevo miembro de ese equipo, con ciertas mejoras: está disponible 24/7, proporciona respuestas inmediatas y mantiene un nivel de experiencia constante y actualizado basado en toda la documentación del sistema. Toda la CX se ha diseñado en inglés ante la previsión del despliegue DocuBot en una multinacional.

Partiendo de esta metáfora, el diseño se ha guiado por una serie de principios clave destinados a maximizar su utilidad y la confianza del usuario dentro del entorno de Microsoft Teams.

- **Claridad y Confianza:** El objetivo primordial no es solo proporcionar una respuesta, sino asegurar que esta sea fiable y verificable. Para lograrlo, el diseño incorpora un sistema de citación de fuentes explícito. Cada respuesta generada por el modelo de IA va acompañada de referencias a los documentos y páginas exactas de donde se extrajo la información.
- **Usabilidad y Guiado del Usuario:** Para evitar la frustración del usuario ante un lienzo en blanco, la conversación se inicia con un guiado proactivo. Al comenzar un nuevo chat, se presenta una tarjeta que permite al usuario acotar su búsqueda a categorías o sistemas específicos. Esta funcionalidad no solo mejora la relevancia de los resultados desde la primera interacción, sino que también educa al usuario sobre el alcance del conocimiento del bot. Adicionalmente, se ofrece un botón de "Nuevo Chat" para permitir al usuario reiniciar el contexto de la conversación de forma clara e inequívoca en cualquier momento.
- **Ciclo de Retroalimentación (*Feedback Loop*):** Un sistema de IA debe ser evaluable. Se ha integrado un ciclo de feedback como parte integral de la

experiencia post-respuesta. Mediante botones de "pulgar arriba/abajo", el usuario puede proporcionar una valoración rápida sobre la utilidad de la respuesta. Ante una valoración negativa, se presenta una segunda tarjeta que solicita un feedback más detallado sobre la calidad de la respuesta y la pertinencia de la documentación citada. Esta recopilación de datos es fundamental para futuras iteraciones de mejora y funcionalidades del sistema.

- **Manejo de Errores y Gestión de Expectativas:** DocuBot está diseñado para gestionar situaciones en las que no puede proporcionar una respuesta útil. Si la fase de recuperación no encuentra documentos relevantes para la consulta del usuario, el chatbot está programado para comunicarlo explícitamente, en lugar de intentar forzar una respuesta irrelevante o una respuesta que lleve a alucinaciones del LLM. Este manejo honesto de sus limitaciones es clave para mantener la confianza del usuario y gestionar correctamente sus expectativas.

Icono de la aplicación

El icono de DocuBot ha sido diseñado con ayuda de Imagen4, un modelo de Google de generación de imágenes. El tamaño del icono está restringido por la capacidad que te da Teams para cargar los logos de sus aplicaciones, que deben ser en 192x192.

Figura 34. Icono utilizado en DocuBot

7.2.2 Componentes de la Interfaz: Tarjetas Adaptativas

Como se ha dicho, la interfaz de usuario del chatbot se materializa a través de Tarjetas Adaptativas. Se han diseñado tres tarjetas principales que estructuran los puntos clave de la interacción con el usuario:

- **ResponseCard (Tarjeta de Respuesta):** Es el componente central de la interacción. Su diseño está cuidadosamente estructurado para presentar la información de manera clara y funcional.
    - **Cuerpo de la Respuesta:** Muestra el texto generado por el modelo de lenguaje.
    - **Sección de Referencias:** Un conjunto de botones donde cada uno corresponde a un documento fuente citado. Al ser pulsado, revela

detalles como el nombre del documento, las páginas referenciadas y un enlace directo para abrir el documento original.

- **Sección de Feedback:** Incluye los botones de pulgar arriba/abajo y el botón para iniciar una nueva conversación.

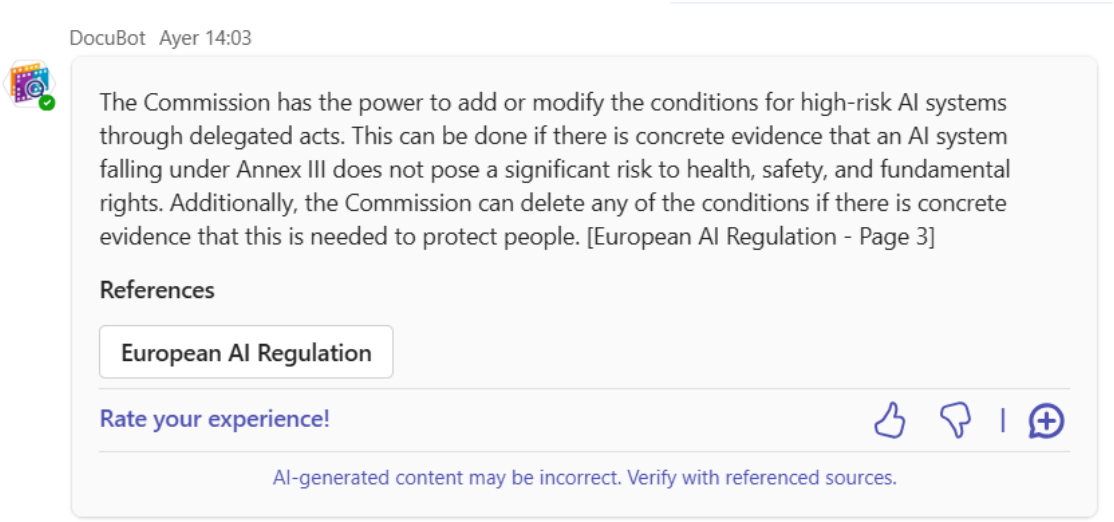


Figura 35. Ejemplo de ResponseCard.

- **CommentCard (Tarjeta de Comentarios):** Se utiliza para profundizar en el feedback del usuario. Es una tarjeta de entrada de datos que incluye:
  - Un selector numérico (1-10) para calificar la calidad de la respuesta.
  - Un selector booleano (Sí/No) para valorar la relevancia de los documentos citados.
  - Un campo de texto libre para comentarios adicionales.
  - Un botón de "Enviar" que encapsula los datos y los envía de vuelta al bot.

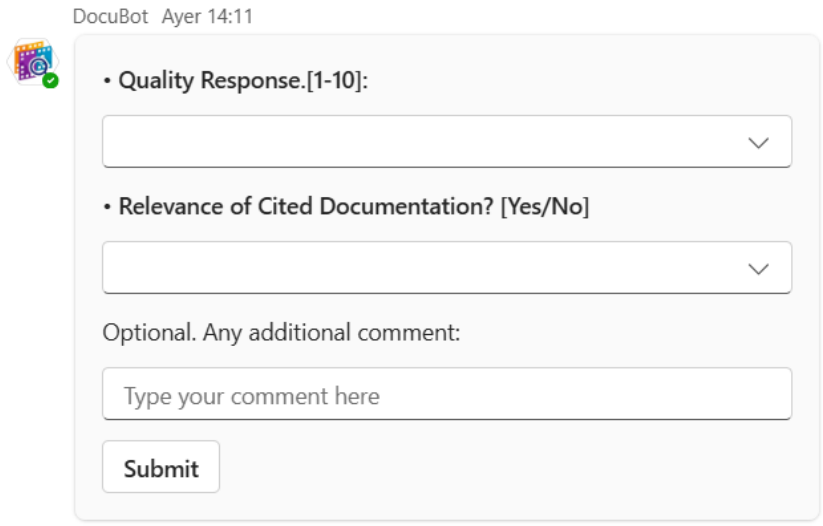


Figura 36. Ejemplo de CommentCard con feedback detallado

- **SystemSelectionCard (Tarjeta de Selección de Sistema Documental):** Presentada al inicio de la conversación, actúa como un formulario de filtrado previo.
    - Utiliza un multi-selector para que el usuario elija las áreas de conocimiento de su interés.
    - Ofrece dos acciones finales: "Confirmar Selección" para aplicar los filtros seleccionados y "Buscar en Todos los Sistemas" para proceder sin filtros, ofreciendo flexibilidad al usuario.

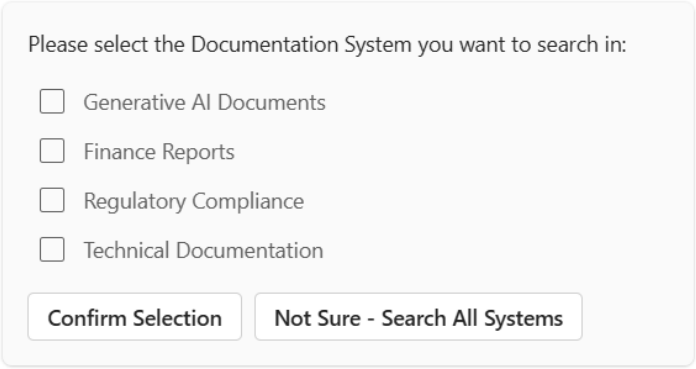


Figura 37. Ejemplo de tarjeta selección de documentación

7.3 Diseño de Arquitectura RAG Avanzada

La arquitectura de este sistema no sólo sigue el patrón de una aplicación de software tradicional, sino que se fundamenta en un paradigma de IA conocido como **RAG**. La base teórica de este tipo de arquitecturas se ha introducido y desgranado en el capítulo 3 que abordaba los fundamentos teóricos utilizados en el proyecto.

Se le han incorporado múltiples optimizaciones y características avanzadas para superar las limitaciones en un flujo RAG básico. En lugar de un simple proceso lineal de recuperar -> generar, este sistema emplea una arquitectura sofisticada y en múltiples etapas, diseñada para maximizar la precisión, la relevancia y la fiabilidad de las respuestas.

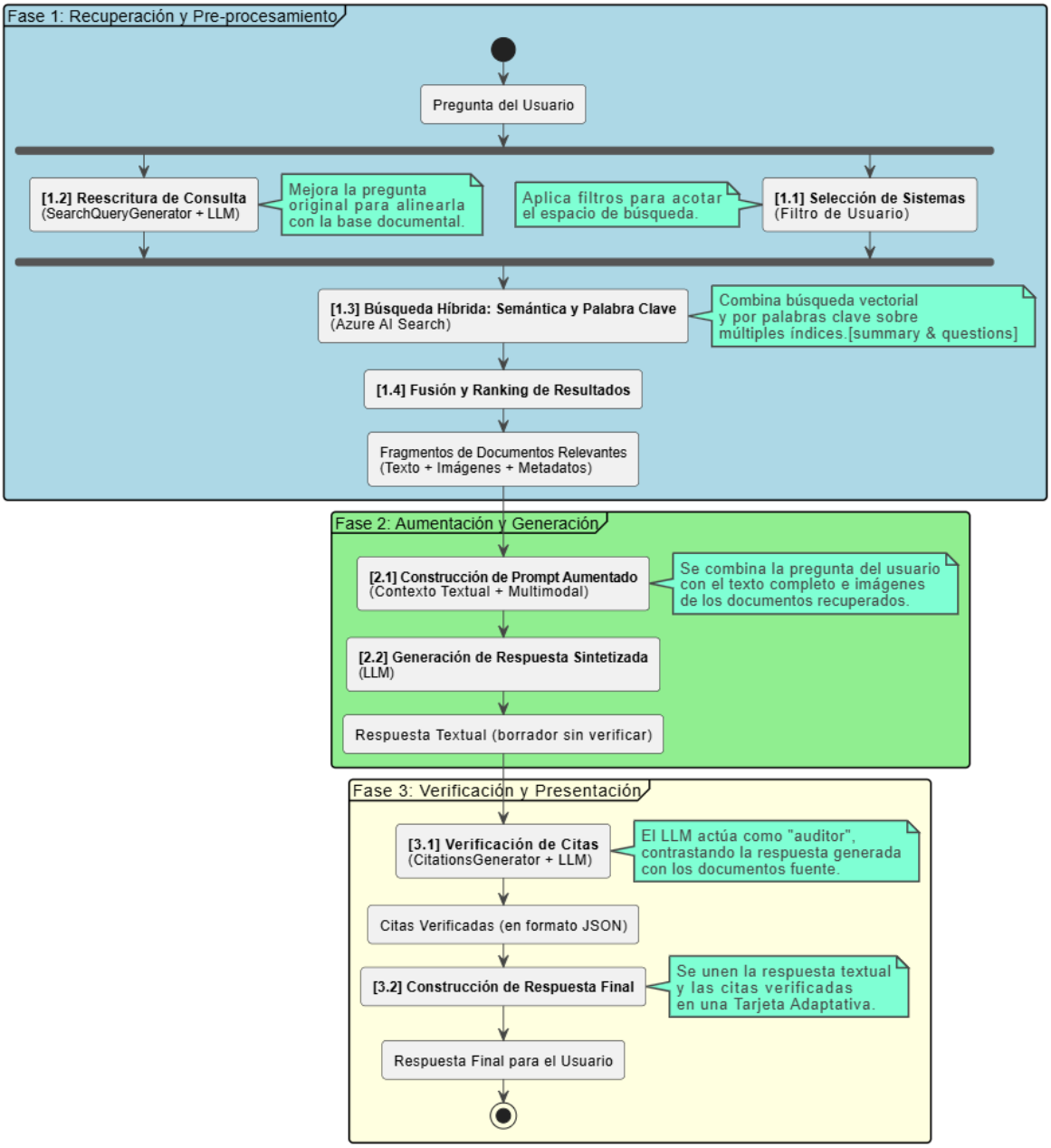


Figura 38. Diagrama flujo RAG avanzado DocuBot

A continuación, se detallan las particularidades y adaptaciones específicas que definen la arquitectura RAG de este proyecto, siguiendo las fases ilustradas en el diagrama anterior.

7.3.1 Proceso de Recuperación Multi-Índice y de Búsqueda Híbrida

El objetivo de esta fase es encontrar y extraer los fragmentos de información más relevantes de la base de conocimiento (el corpus documental de la empresa) en respuesta a la consulta del usuario.

A diferencia de un RAG clásico que podría depender de un único índice vectorial, el proceso de recuperación de este sistema está diseñado para ser mucho más exhaustivo y preciso, usando varias de las técnicas de RAG avanzadas:

1. **Reescritura de Consulta con Gen AI:** La primera particularidad es que la consulta del usuario no se utiliza directamente. El sistema emplea un componente *SearchQueryGenerator* que realiza una llamada preliminar al LLM. Este paso reescribe la consulta del usuario para enriquecerla semánticamente y alinearla con la terminología y estructura de la base documental. Esta optimización inicial es clave para mejorar significativamente la calidad de los resultados de búsqueda.
2. **Búsqueda sobre Múltiples Índices:** La búsqueda no se limita a un solo tipo de contenido. Se realiza sobre dos índices de Azure AI Search distintos y complementarios:
    - Un índice optimizado para resúmenes y pares de preguntas posibles, que captura la esencia de los documentos.
    - Un segundo índice que contiene el texto completo de los documentos, permitiendo encontrar detalles específicos y por claves.
3. **Búsqueda Híbrida:** Dentro de cada índice, se ejecuta una búsqueda híbrida que combina la búsqueda vectorial (*Cosine similarity*) con la búsqueda por palabras clave (*Keywords*). Esta técnica dual asegura que se recuperen tanto los documentos que son temáticamente relevantes como aquellos que contienen menciones literales, mitigando los fallos que cada tipo de búsqueda podría tener por separado.
4. **Filtro de grupos documentales:** La arquitectura integra la capacidad de filtrado del usuario. A partir de la selección de categorías realizada al inicio de la conversación, se construye dinámicamente un filtro que se aplica a la consulta de búsqueda. Esto garantiza que la recuperación se realice únicamente sobre el subconjunto de documentos pertinente, reduciendo el ruido y mejorando drásticamente la relevancia de los resultados desde el primer paso.
5. **Ranking y Fusión de Resultados:** Los resultados de ambos índices se fusionan mediante un sistema de puntuación que pondera la relevancia de cada documento en las diferentes búsquedas, siendo una adaptación de RFF (*Reciprocal Rank Fusion*). En lugar de usar el inverso del ranking, se optó por un sistema de puntos ponderados. Esto produce una lista final unificada y ordenada de los fragmentos de información más relevantes de todo el corpus documental.

Estrategia de Segmentación (Chunking) por Página

Un pilar en el diseño de la arquitectura RAG es la estrategia de segmentación, que define la granularidad de la información que se indexa y recupera. Para DocuBot, se tomó la decisión estratégica de adoptar una **segmentación por página**, tratando cada página de un documento original como una única unidad atómica de conocimiento.

Esta elección responde a tres objetivos clave del diseño:

- **Preservación del Contexto:** A diferencia de la segmentación por tamaño fijo, que puede cortar frases o ideas a la mitad, el chunking por página preserva de forma natural el contexto semántico y, crucialmente para la capacidad multimodal del sistema, el contexto visual (diagramas, tablas) concebido por el autor del documento.
- **Alineación con la Experiencia de Usuario:** La unidad técnica de recuperación ("chunk") se alinea directamente con la unidad de citación que el usuario final entiende ("el número de página"). Esto hace que la verificación de las fuentes sea intuitiva, fiable y sencilla a nivel técnico.
- **Simplificación Arquitectónica:** Esta decisión simplifica el diseño del CitationsGenerator y del ResponseCard, ya que no necesitan lógica adicional para reconstruir a qué página pertenece un fragmento de texto recuperado; la unidad recuperada es la referencia completa.

7.3.2 Inclusión del Contexto Multimodal

En la fase de aumentación, el objetivo principal es enriquecer la consulta original del usuario con la información más relevante extraída de la base de conocimiento. El resultado de esta etapa es un *prompt* final y detallado que se enviará LLM para generar una respuesta precisa.

El proceso se distingue por las siguientes particularidades:

- **Construcción del Contexto Textual:** A diferencia de una simple concatenación de texto, se extrae el contenido completo de los documentos almacenados (en este caso, blobs JSON). Esto proporciona un contexto denso y rico en detalles. Este componente es el encargado de descargar estos contenidos, que se unifican en una única cadena de texto para fundamentar la respuesta del LLM.
- **Incorporación de Contexto Multimodal:** La arquitectura del sistema está diseñada para ser multimodal. Se pueden descargar las representaciones visuales de las páginas de los documentos, como las imágenes en formato Base64. Estas imágenes se insertan junto al texto en el prompt final, permitiendo que el modelo multimodal analice la información visualmente. Esta capacidad permite que modelos con funcionalidades multimodales, como GPT-4o, no solo procesen el texto, sino que también interpreten elementos visuales como diagramas, tablas y gráficos. Esta característica es especialmente valiosa cuando las respuestas a

las consultas de los usuarios se encuentran parcial o totalmente en documentación técnica de carácter visual.

Esta fase de preparación del contexto es crucial, ya que al "aumentar" la pregunta inicial del usuario con fragmentos de documentos recuperados, se dota al LLM de toda la información necesaria para generar una respuesta coherente y bien fundamentada. Este enfoque, que separa la recuperación de información de la generación final, contribuye a mejorar notablemente la fiabilidad del sistema.

7.3.3 Proceso de verificación de citas

La fase final del proceso es la generación de la respuesta, la cual presenta otra innovación significativa en su diseño. En lugar de un enfoque con una única llamada al LLM, esta etapa se descompone en dos pasos desacoplados para incrementar la fiabilidad y la capacidad de justificación del sistema.

1. **Generación de la Respuesta Sintetizada:** En el primer paso, se realiza una llamada al LLM. La única responsabilidad del modelo en esta fase es sintetizar una respuesta coherente en lenguaje natural, basándose estrictamente en el contexto aumentado que se le ha proporcionado en la etapa anterior. Este enfoque deliberado evita que el LLM tenga que realizar simultáneamente las complejas tareas de responder y citar las fuentes.
2. **Generación y Verificación de Citas:** Una vez obtenida la respuesta textual, se realiza otra llamada al LLM, pero con un propósito distinto: actuar como un "auditor". Al modelo se le suministra la respuesta recién generada junto con los documentos fuente originales. Su tarea no es reformular la respuesta, sino analizarla y devolver una estructura de datos JSON que identifique con precisión qué documentos respaldan cada afirmación.

Este diseño de generación y citación desacoplada de la generación constituye una característica avanzada que diferencia a este sistema de una arquitectura RAG convencional. Al forzar una segunda pasada de verificación, no solo se incrementa drásticamente la fiabilidad de las citas, sino que también se asegura que el sistema pueda justificar el origen de su información, un requisito indispensable en un entorno corporativo que se necesitan arquitecturas mucho más fiables.

7.4 Arquitectura Lógica: Modelo de Componentes de Software

La arquitectura lógica de la aplicación se fundamenta en principios de diseño de software probados para crear sistemas modulares, mantenibles y escalables. Concretamente, se ha intentado adoptar el paradigma de la **Arquitectura**

Hexagonal (también conocida como "Puertos y Adaptadores"), una variante de la Arquitectura Limpia (Clean Architecture).

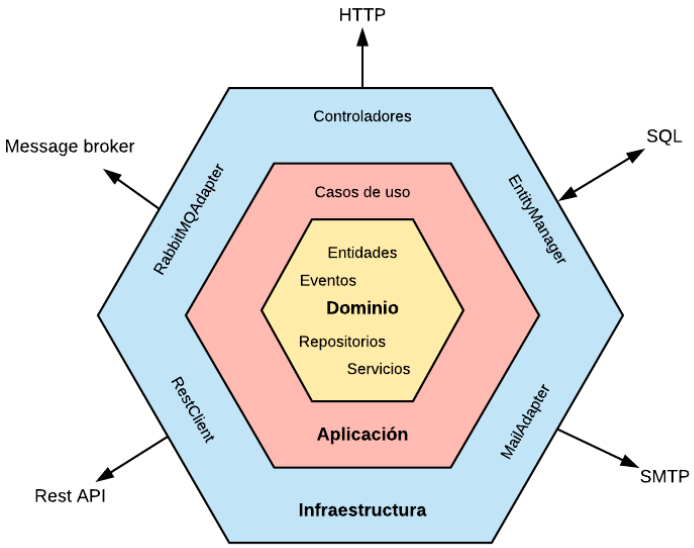


Figura 39. Arquitectura Hexagonal

El principio fundamental de la Arquitectura Hexagonal es proteger el núcleo de la aplicación (el dominio y la lógica de negocio) de las dependencias de tecnologías externas.[52] La comunicación entre el núcleo y el mundo exterior (como la API de Teams, la base de datos Cosmos DB o el servicio de OpenAI) se realiza a través de "puertos" (interfaces definidas por el núcleo) y "adaptadores" (las implementaciones concretas que "enchufan" la tecnología externa a esos puertos). Las dependencias siempre apuntan hacia el interior, hacia el dominio, lo que garantiza un sistema modular, fácil de mantener, así como promover una estricta separación de responsabilidades [12].

Esta elección es especialmente idónea para un sistema de IA conversacional como DocuBot, que debe orquestar múltiples servicios externos. Al aislar el núcleo, se podría, por ejemplo, cambiar la plataforma de chat de Microsoft Teams a Slack simplemente creando un nuevo "adaptador", sin modificar una sola línea de la lógica de negocio principal.

La aplicación de este paradigma se refleja directamente en la organización del código fuente, que sigue una estructura de capas clara.

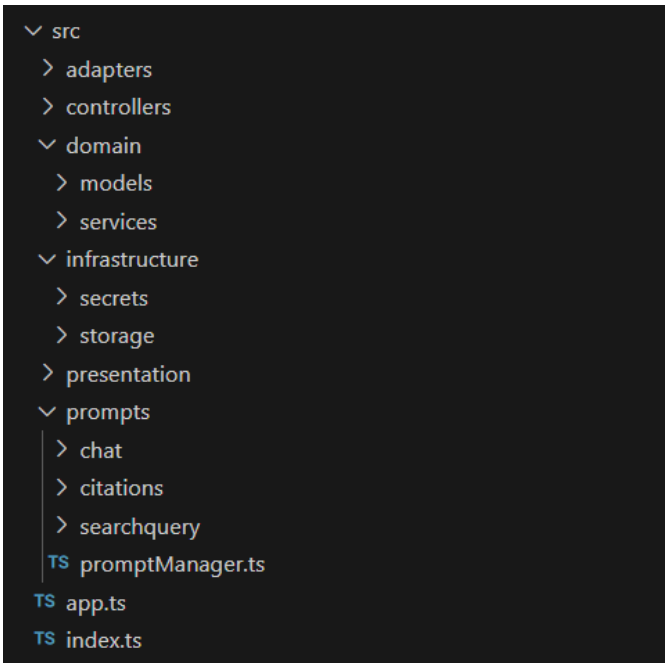


Figura 40. Estructura directorios del proyecto

Es importante señalar que el término "Hexagonal" es una metáfora visual para enfatizar la existencia de múltiples puntos de conexión (puertos) y no una prescripción estricta de tener seis lados o capas. En la práctica, esta separación se puede organizar en un modelo de capas lógicas. Para este proyecto, los componentes de la Arquitectura Hexagonal se han agrupado en el modelo de tres capas que se presentará a continuación: la Capa de Lógica de Negocio representa el núcleo central, mientras que las capas de Presentación y de Datos e Infraestructura constituyen los diferentes adaptadores que interactúan con el mundo exterior.

Para ilustrar esta arquitectura, se presentan dos diagramas con diferentes niveles de abstracción: una visión simplificada que introduce las capas principales y una visión completa que detalla los componentes de software dentro de cada capa y sus interacciones.

7.4.1 Arquitectura Lógica Simplificada

A alto nivel, el sistema se puede descomponer en tres capas lógicas fundamentales que interactúan entre sí, tal y como se representa en la siguiente figura.

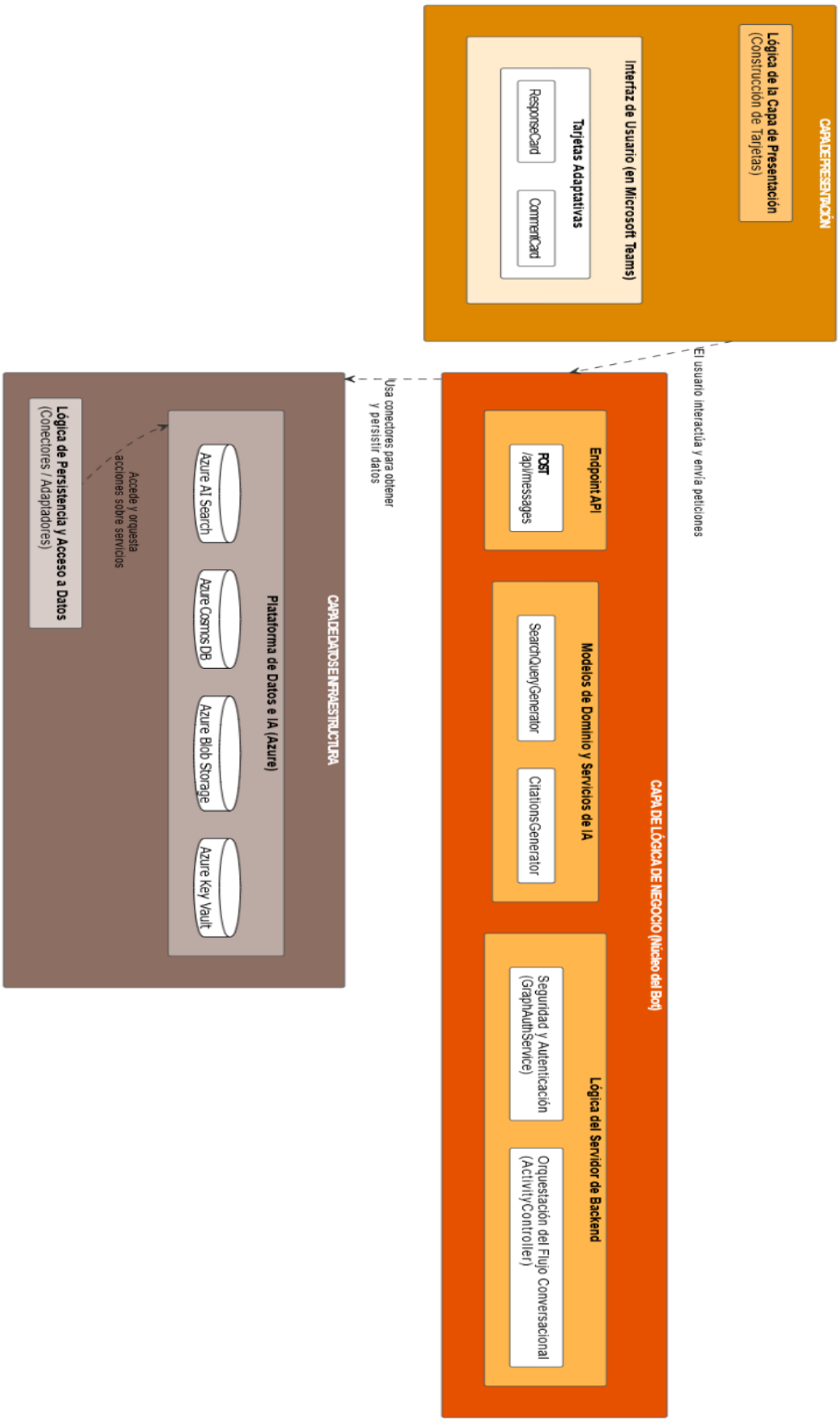


Figura 41. Arquitectura Lógica Simplificada

- **Capa de Presentación:**
    - **Lógica de la Capa de Presentación:** Encapsula la lógica para definir y construir la interfaz conversacional.
    - **Interfaz de Usuario (Tarjetas Adaptativas)**
- **Capa de Lógica de Negocio (El Núcleo del Bot):**
    - **Lógica del Servidor de Backend:** Incluye la orquestación del flujo conversacional y la gestión de la seguridad y autenticación.
    - **Modelos de Dominio y Servicios de IA:** Contiene las reglas y la lógica pura de la aplicación, como la generación de citas y la optimización de consultas.
    - **Endpoint de la API.**
- **Capa de Datos e Infraestructura:**
    - **Lógica de Persistencia y Acceso a Datos:** Contiene los "conectores" que se comunican con los servicios externos. No hay un único servicio de base de datos, sino un conjunto de servicios de Azure.

7.4.2 Arquitectura Lógica Completa

Profundizando en el detalle, la siguiente figura ilustra la arquitectura completa, mostrando los componentes de software específicos dentro de cada capa y el flujo de control y dependencias entre ellos. La cohesión del sistema se logra mediante el uso extensivo del patrón de Inyección de Dependencias (DI) [53], orquestado en una capa de ensamblaje que actúa como el contenedor de la aplicación.

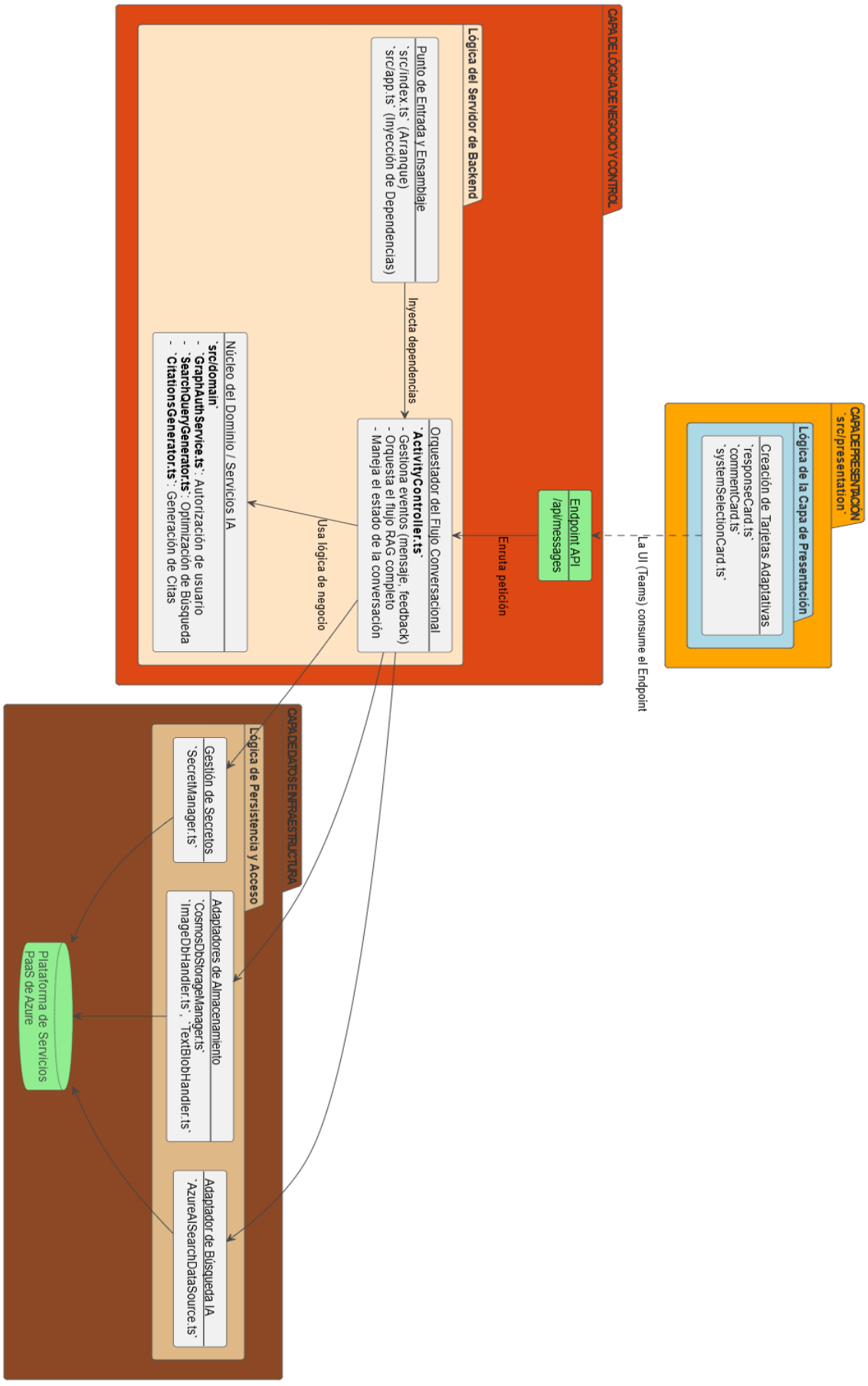


Figura 42. Arquitectura Lógica Completa

Capa de Presentación

Esta capa es responsable de definir cómo se presenta la información al usuario final. En el contexto de DocuBot, se traduce en la creación de las Tarjetas Adaptativas.

- **Lógica de la Capa de Presentación:** Se encarga de la construcción dinámica de los elementos de la interfaz. Esto incluye la lógica para poblar las tarjetas con los datos recibidos desde la capa de negocio (como la respuesta del LLM y las referencias documentales) y formatearlos adecuadamente en una estructura JSON.
- **Interfaz de Usuario Conversacional:** La interfaz se compone de Tarjetas Adaptativas (*ResponseCard, CommentCard, SystemSelectionCard*), que presentan la información y capturan las interacciones del usuario (clics en botones, selección de opciones, etc.).

Capa de Lógica de Negocio y Control

Esta es la capa más extensa y compleja de la aplicación. Está claramente subdividida en componentes con responsabilidades distintas:

- **Componente de Ensamblaje y Enrutamiento de Eventos:**
  Actúa como el punto de partida y el director de tráfico principal de la capa de negocio.
  - **Ensamblador de la Aplicación:** Es responsable de la construcción e inicialización de todos los componentes y servicios de la aplicación en el momento del arranque. Siguiendo el patrón de Inyección de Dependencias, este ensamblador instancia todos los servicios necesarios (como los conectores de datos y los servicios de IA) y los inyecta en los componentes que los utilizarán, principalmente en el controlador de flujo.
  - **Manejador de Eventos de la Plataforma:** Define las "rutas" lógicas de la aplicación conversacional. Mapea los eventos entrantes desde la plataforma de comunicación (por ejemplo, nuevo mensaje, usuario se une a la conversación, botón de tarjeta pulsado) a los métodos específicos dentro del Orquestador Conversacional.
- **Componente de Control de Flujo (Orquestador Conversacional):**
  Es el componente central que gestiona la lógica de la interacción de principio a fin. Orquesta las llamadas a los servicios apropiados en el orden correcto.
  - Al recibir un evento (una pregunta del usuario), este orquestador inicia una secuencia de operaciones: primero invoca al servicio de seguridad para validar los permisos del usuario; a continuación, llama al servicio de acceso a la base de conocimiento para recuperar documentos relevantes; después, pasa estos documentos al motor de generación de lenguaje para obtener una respuesta; y finalmente, coordina la presentación de esa respuesta al usuario.

- **Núcleo de Dominio (Reglas de Negocio y Lógica de IA):**
  Este es el corazón intelectual de la aplicación y contiene la lógica pura, completamente independiente de la tecnología externa.
    - **Servicios de Dominio:** Encapsulan reglas de negocio de alto nivel. Un ejemplo es el servicio de autorización, que contiene la lógica para determinar si un usuario tiene los roles y permisos necesarios para utilizar el sistema.
    - **Modelos y Lógica de IA:** Representan los conceptos y algoritmos clave que definen la inteligencia de la aplicación. En este proyecto, estos no son simples modelos de datos, sino componentes de servicio activos que tienen lógica compleja:
        - Un servicio para la generación de citaciones verificables, que sabe cómo analizar una respuesta y contrastarla con las fuentes.
        - Un servicio para la optimización de consultas de búsqueda, que aplica técnicas de IA para mejorar la pregunta del usuario antes de enviarla al motor de búsqueda.
- **API de Interacción (Endpoint):**
  La comunicación entre la plataforma externa (Microsoft Teams) y la capa de lógica de negocio se realiza a través de un único endpoint de API. Este punto de entrada recibe todas las actividades y las canaliza hacia el Componente de Ensamblaje y Enrutamiento de Eventos.

Capa de Datos e Infraestructura

Esta capa gestiona la persistencia de los datos y la comunicación con todos los servicios externos. Abstrae los detalles de la infraestructura, de ahí que se haya querido reflejar toda las conexiones externas en un mismo componente dentro del diagrama. Actúa como una colección de Adaptadores que traducen las necesidades de la aplicación en llamadas a servicios externos.

- **Teams Adapter:** Adaptador específico para la plataforma Microsoft Teams.
- **Lógica de Persistencia y Acceso a Datos:**
    - **Gestión de Persistencia:** Se encarga de guardar y recuperar el estado de la conversación y del registro detallado de cada turno (interacción, respuesta, feedback, métricas de uso) en una base de datos NoSQL, Azure Cosmos DB.
    - **Acceso a la Base de Conocimiento:** Gestiona la comunicación con el servicio de búsqueda empresarial, Azure AI Search. Es responsable de ejecutar las consultas de búsqueda híbrida (vectorial y por palabras clave) y de aplicar los filtros dinámicos sobre la base de conocimiento documental.

    - **Acceso a Almacenamiento de Blobs:** Proporciona los mecanismos para recuperar los artefactos de los documentos (imágenes de páginas, textos completos en formato JSON) desde Azure Blob Storage.
- **Gestor de Secretos:** Un componente de seguridad crucial que se comunica con un servicio de gestión de secretos centralizado para obtener de forma segura todas las credenciales, claves de API y cadenas de conexión necesarias para que la capa de infraestructura se comunique con los demás servicios.

La implementación específica de todos los componentes y las decisiones de codificación asociadas se detallan en las respectivas secciones del capítulo siguiente de Implementación.

7.4.3 Diagrama de Clases Núcleo de Aplicación

Para materializar el diseño lógico expuesto y ofrecer una visión más concreta de las interacciones a nivel de código, se presenta a continuación un diagrama de clases. Lejos de intentar representar la totalidad de las clases de la aplicación que resultaría inmanejable, este diagrama se centra en el componente orquestador principal, *ActivityController*, y sus colaboradores directos.

Se puede observar cómo *ActivityController* interactúa con un conjunto de servicios especializados. Estos incluyen servicios de dominio (como GraphAuthService para la lógica de autenticación o CitationsGenerator para la IA de citas) como adaptadores a la infraestructura (como AzureAISearchDataSource o CosmosDbPartitionedStorage), cada uno con una responsabilidad única.

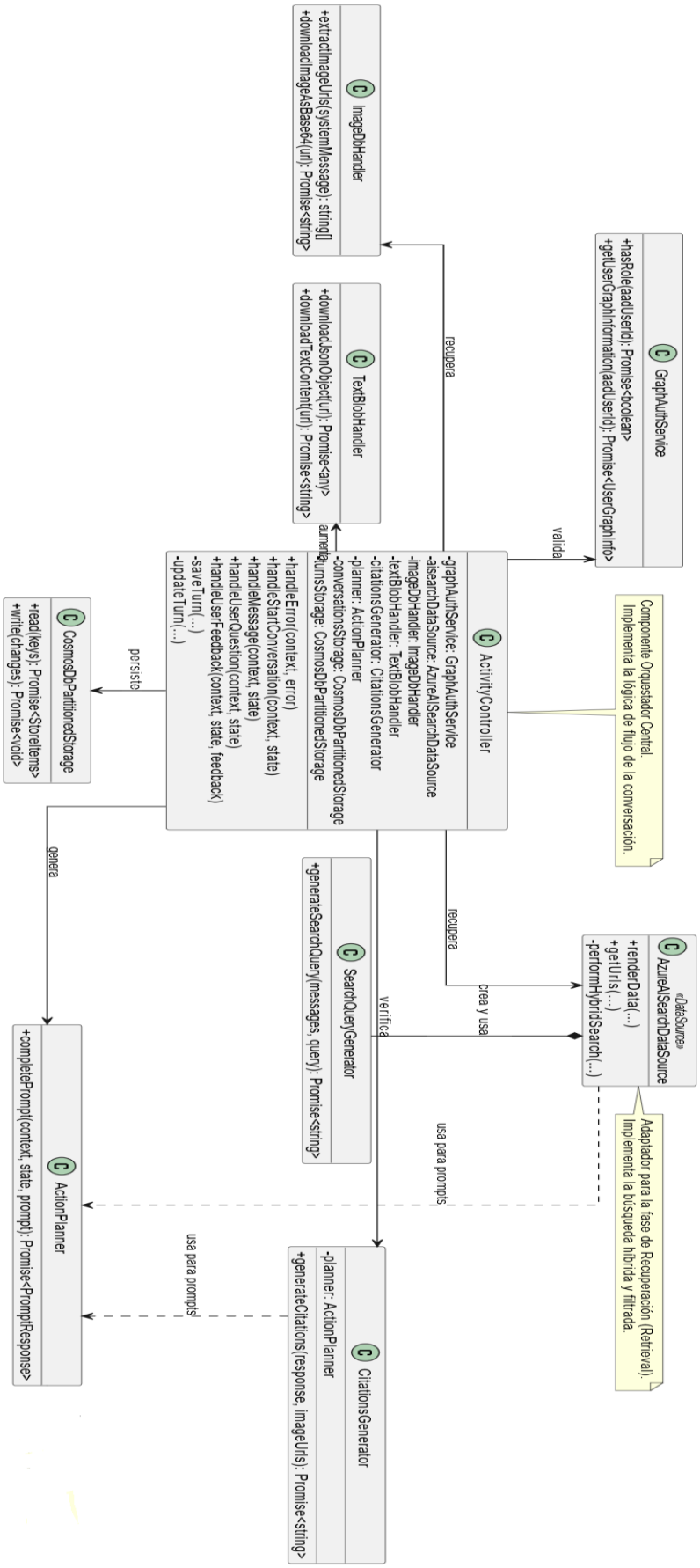


Figura 43. Diagrama Clases Núcleo Orquestador

7.5 Arquitectura Física: Diagrama de Despliegue

Toda arquitectura lógica debe materializarse en una arquitectura física que le dé soporte. En este proyecto, se ha optado por una arquitectura nativa en la nube, desplegada íntegramente sobre PaaS de Microsoft Azure. Como se ha comentado a lo largo de la memoria esta elección estratégica reduce la carga de gestión de la infraestructura, mejora la escalabilidad y la seguridad, y permite una integración nativa entre los distintos componentes.

La siguiente figura representa el diagrama de despliegue de la solución, mostrando los servicios de Azure involucrados y cómo los componentes lógicos de la aplicación se mapean a ellos dentro de una suscripción y un grupo de recursos dedicados. Muchos de estos servicios de Azure se comentaron en detalle sus funcionalidades durante el capítulo de Fundamentos Teóricos: Tecnologías Utilizadas

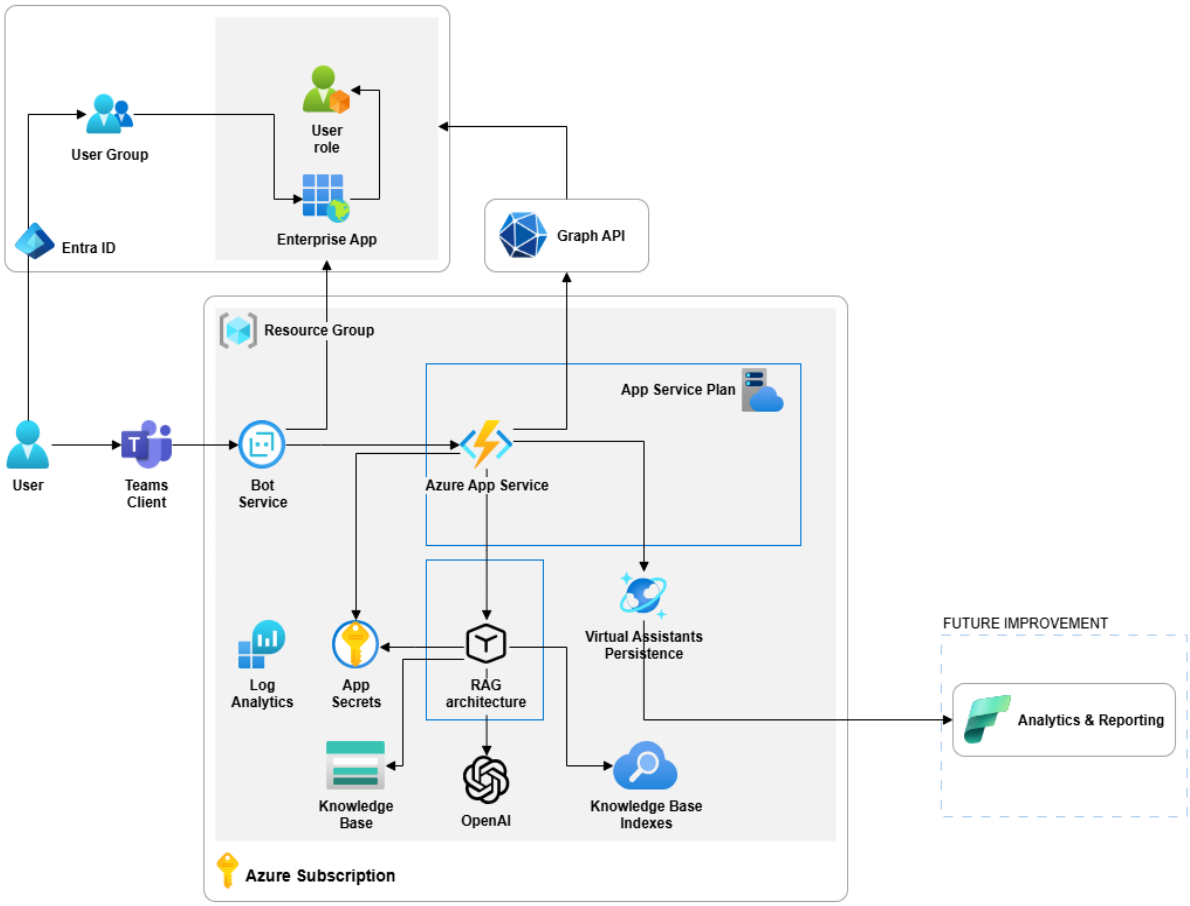


Figura 44. Diagrama de arquitectura física de los componentes Azure.

La arquitectura física se puede dividir en los siguientes bloques funcionales:

- **Plataforma de Interfaz y Conectividad:**
    - **Teams Client:** Es el cliente final donde el usuario interactúa con el chatbot.

  - **Azure Bot Service:** Actúa como el intermediario principal. Registra el bot, gestiona el canal de comunicación de Teams y enruta de forma segura los mensajes entre el cliente y la lógica de backend.

- **Entorno de Backend Conversacional**

  Este es el núcleo de la aplicación, donde se ejecuta toda la lógica. Se aloja como un servicio de cómputo gestionado en Azure.

  Azure App Service: Es el servicio PaaS que aloja la aplicación.

  - **Entorno de Ejecución (node.js)**: La aplicación está construida sobre el entorno de ejecución de Node.js.
  - **Framework de Backend (restify)**: Se utiliza como el framework web ligero para crear el servidor y exponer el único endpoint */api/messages*. Su función principal es recibir las peticiones HTTP del Azure Bot Service.
  - **Gestión de Variables de Entorno:** Middleware utilizado para cargar de forma segura la configuración de la aplicación a partir de variables de entorno.

- **Plataforma de Datos e Inteligencia Artificial:**
  Este es el conjunto de servicios que proporcionan las capacidades fundamentales de la aplicación.
  - **Azure OpenAI Service (OpenAI):** Provee acceso a los LLM, así como el modelo de embeddings utilizados en todo el flujo RAG.
  - **Azure AI Search (Knowledge Base Indexes):** Servicio que aloja los índices de búsqueda vectorial y de texto completo, actuando como el motor de la fase de recuperación.
  - **Azure Cosmos DB (Virtual Assistants Persistence):** Base de datos NoSQL utilizada para la persistencia del estado de la conversación y el almacenamiento de los registros de cada turno.
  - **Azure Blob Storage (Knowledge Base):** Servicio de almacenamiento de objetos donde residen los artefactos procesados de la base de conocimiento (imágenes de páginas, archivos JSON, etc.).

- **Plataforma de Seguridad y Operaciones:**
  Componentes transversales a todo el sistema que garantizan la seguridad y monitorización de la solución.
  - **Microsoft Entra ID (Entra ID):** Gestiona la identidad de los usuarios y de la aplicación. A través de una Aplicación, se definen los roles de usuario que determinan quién tiene permiso para usar DocuBot.
  - **Azure Key Vault (App Secrets):** Actúa como forma segura de almacenar todas las credenciales y secretos que la aplicación necesita para conectarse a los otros servicios.

    - **Microsoft Graph API:** Utilizada por la aplicación para consultar información del usuario (como su pertenencia a grupos o roles) directamente desde Microsoft Entra ID.

- **Mejora Futura:**
  El diagrama también contempla una futura integración con servicios de **Análisis y Reporting**, como Power BI. Los datos persistidos en Azure Cosmos DB podrían ser explotados para generar cuadros de mando sobre el uso, la efectividad de las respuestas y los temas más consultados, cerrando así el ciclo de vida del producto.

7.6 Flujo de la Aplicación: Diagrama de Secuencia

Para ilustrar de manera práctica cómo interactúan los distintos componentes de la arquitectura, se presenta un diagrama de secuencia que modela el flujo más completo y representativo de la aplicación: la respuesta a una pregunta de un usuario.

La siguiente figura detalla la secuencia de llamadas y el intercambio de datos entre el usuario, el servidor de la aplicación y los servicios de Azure, siguiendo las fases de la arquitectura RAG avanzada descrita previamente.

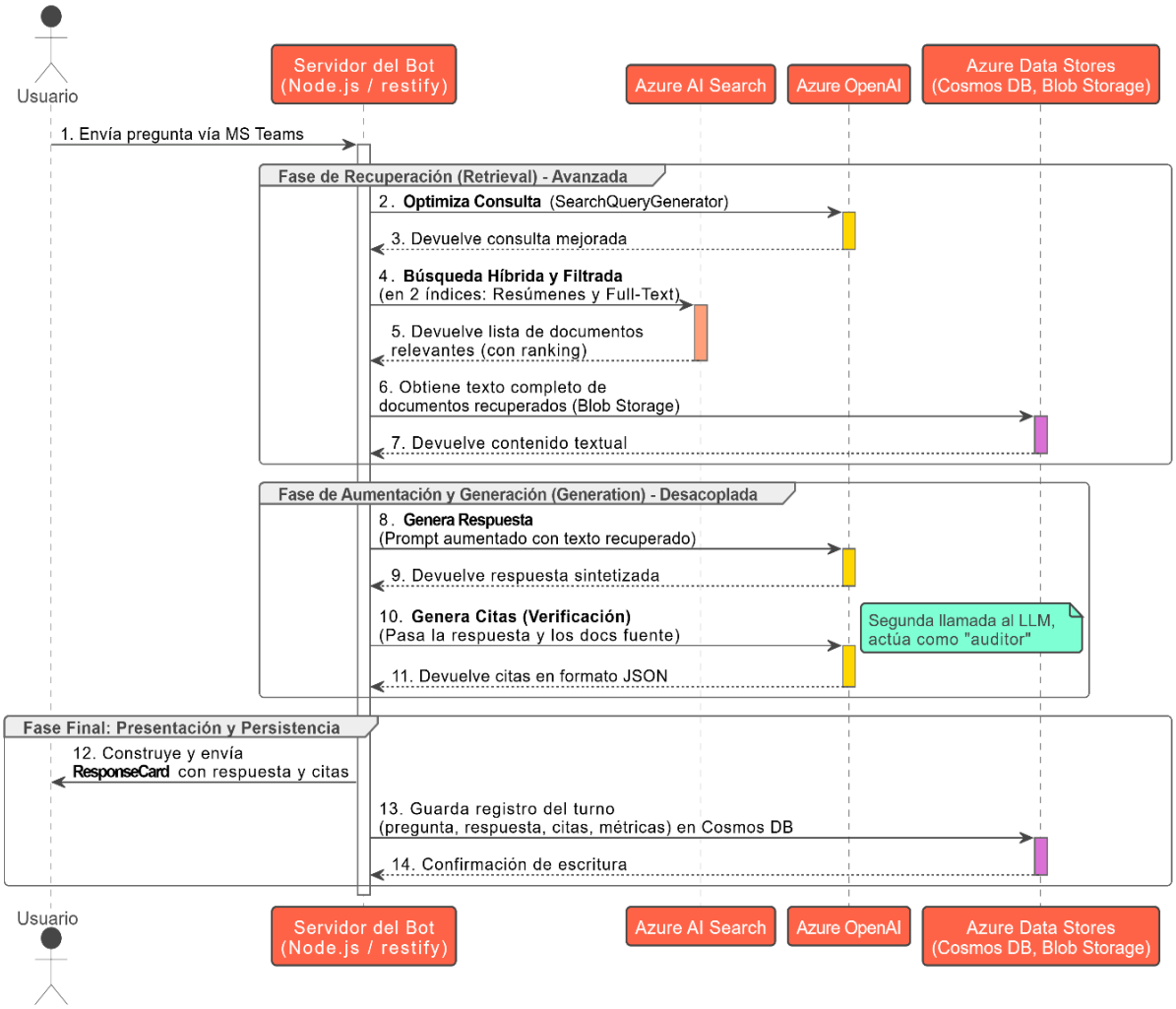


Figura 45. Flujo de Respuesta a una Pregunta.

El flujo, tal y como se representa en el diagrama, se desarrolla de la siguiente manera:

1. El Usuario envía una pregunta a través de la interfaz de Microsoft Teams. La petición es recibida por el Servidor del Bot.

Fase de Recuperación (Retrieval)

1. **Optimiza Consulta:** Para maximizar la precisión de la búsqueda, el servidor no utiliza la pregunta del usuario directamente. En su lugar, invoca al componente *SearchQueryGenerator*, el cual realiza una primera llamada a Azure OpenAI.
2. **Devuelve consulta mejorada:** Azure OpenAI devuelve 3 versiones de la pregunta original semánticamente enriquecidas y optimizadas de la pregunta original, además teniendo cuenta el histórico de la conversación si lo tuviera.
3. **Búsqueda Híbrida y Filtrada:** El servidor utiliza esta consulta mejorada para realizar una búsqueda en Azure AI Search. Esta búsqueda es híbrida (vectorial y por palabras clave) y se ejecuta sobre dos índices (resúmenes y texto completo),

aplicando los filtros de categoría que el usuario haya seleccionado. Otro aspecto que mencionar es que dentro del índice de resúmenes la búsqueda vectorial se realiza sobre 2 campos (summary y questions)

4. **Devuelve lista de documentos:** Azure AI Search devuelve una lista ordenada y puntuada de los documentos más relevantes, que a su vez son rerankeados.
5. **Obtiene texto completo:** Para construir el contexto, el servidor realiza una llamada a Azure Blob Storage para obtener el contenido textual completo de los documentos recuperados.
6. **Devuelve contenido textual:** Blob Storage devuelve el texto, que ahora servirá de base para la generación de la respuesta.

Fase de Aumentación y Generación (Generation)

1. **Genera Respuesta:** El servidor construye un *prompt* aumentado, que combina la pregunta original del usuario con el contenido textual recuperado y las imágenes de los documentos asociados a es documentos textuales recuperado, y realiza la llamada principal de generación a Azure OpenAI.
2. **Devuelve respuesta:** Azure OpenAI genera y devuelve la respuesta en lenguaje natural.
3. **Genera Citas (Verificación):** Aquí se ejecuta el paso clave de la arquitectura desacoplada. El servidor realiza una tercera llamada a Azure OpenAI, pasándole la respuesta recién generada y la lista de documentos fuente. En este paso, el LLM no genera contenido nuevo, sino que actúa como un "auditor".
4. **Devuelve citas en formato JSON:** El modelo devuelve una estructura JSON que mapea las afirmaciones de la respuesta a los documentos fuente, verificando así el origen de la información.

Fase Final: Presentación y Persistencia

1. **Construye y envía ResponseCard:** El servidor utiliza la respuesta y las citas para construir la Tarjeta Adaptativa final y la envía al Usuario.
2. **Guarda registro del turno:** De forma asíncrona, el servidor recopila toda la información de la interacción (pregunta, respuesta, citas, métricas, etc.) y la guarda en Azure Cosmos DB para su persistencia.
3. **Confirmación de escritura:** Cosmos DB confirma que el registro se ha guardado correctamente, finalizando el flujo.

Implementación y Despliegue

En este capítulo se detallará la implementación técnica de los componentes esenciales que conforman la solución DocuBot para Microsoft Teams. Antes de profundizar en los aspectos técnicos de la arquitectura propuesta en capítulos anteriores, se describirá primero como se han creado todos los recursos necesarios en Azure, después la metodología de control de versiones y el flujo de desarrollo que ha estructurado y organizado todo el proceso de codificación.

Como se ha ido reflejando a lo largo de toda la memoria, se diferencia de las aplicaciones web tradicionales que tienen una clara división entre *frontend* y *backend*, la arquitectura de este proyecto se articula en torno a un núcleo de orquestación de IA, una infraestructura de datos distribuida y una interfaz de usuario integrada nativamente en el cliente de Microsoft Teams.

El análisis se centrará en tres pilares fundamentales:

1. **El Núcleo Conversacional y la Orquestación de IA:** Se describirá el servidor de la aplicación, el controlador lógico que gestiona el flujo de la conversación y los componentes de inteligencia artificial que refinan y procesan la información.
2. **La Infraestructura de Datos y Conocimiento:** Se explorará cómo se estructura y consume la base de conocimiento en Azure AI Search, cómo se persiste el estado de la conversación y la auditoría en Cosmos DB, y cómo se gestionan los ficheros brutos en Azure Blob Storage.
3. **La Interfaz de Usuario en Microsoft Teams:** Se explicará cómo se construye la experiencia de usuario mediante el uso de Adaptive Cards y cómo se integra la aplicación en el ecosistema de Teams.

Finalmente, se abordará el proceso de despliegue de todos estos componentes en el entorno cloud de Microsoft Azure, detallando la configuración de los servicios y el ciclo de vida de la aplicación desde el desarrollo hasta la producción.

8.1 Provisionamiento de la Infraestructura en Azure

Antes de implementar y desplegar el código de la aplicación, es fundamental provisionar y configurar la infraestructura subyacente en la nube de Microsoft Azure. Este proceso implica la creación de todos los servicios PaaS que darán soporte a la solución, desde las bases de datos y los servicios de IA hasta el entorno de ejecución y la gestión de secretos.

8.1.1 Entorno Empresarial: Tenant de Microsoft 365

Antes de provisionar los recursos de infraestructura en Azure, fue necesario establecer un entorno que simulara de manera fidedigna el ecosistema corporativo en el que DocuBot operaría una vez desplegado. Un sistema de este tipo depende de estar conectado de servicios como Microsoft Teams, SharePoint Online y Microsoft Entra ID para la interfaz de usuario, el almacenamiento de documentos y la autenticación de usuarios, respectivamente.

Para este fin, se adquirió una licencia de **Microsoft 365 Empresa Básico**, lo que permitió la creación de un tenant de Microsoft 365 completamente funcional, como pudiera tener una organización de cualquier tamaño, la única diferencia radica en el número de licencias que tiene cada organización [46]. Este paso es crucial por varias razones:

- **Realismo:** Proporciona un entorno idéntico al que se encontraría en una empresa real, con usuarios, grupos, un sitio de SharePoint para la base documental y un cliente de Teams donde desplegar y probar DocuBot.
- **Integración de Identidad:** El tenant incluye una instancia de Microsoft Entra ID, fundamental para gestionar la identidad de la aplicación (App Registration) y los permisos de los usuarios (Roles).
- **Habilitador Funcional:** Sin un tenant de SharePoint Online, el proceso de ingesta de documentos no podría haberse implementado. Sin un entorno de Teams, la validación de la interfaz de usuario y la experiencia conversacional habría sido imposible.

La creación de este tenant ha sido un requisito técnico indispensable que constituyó la primera piedra de la infraestructura, permitiendo que el desarrollo y el despliegue se realizaran en un contexto realista y funcional.

8.1.2 Convención Nomenclatura Azure

La estrategia de provisionamiento se ha basado en dos principios clave para garantizar una gestión ordenada, segura y escalable:

1. **Centralización en un Grupo de Recursos:** Todos los recursos de Azure relacionados con esta solución se han desplegado dentro de un único Grupo de Recursos (Resource Group) denominado *rg-docubot-poc-westeu-001*. Esta buena práctica centraliza la gestión del ciclo de vida (despliegue, monitorización, eliminación), el control de acceso y el seguimiento de costes de todos los componentes de la aplicación como una única entidad.
2. **Adopción de una Nomenclatura Estándar:** Se ha seguido una convención que es una práctica recomendada en entornos empresariales por Azure [54].

La nomenclatura utilizada para la creación de los recursos sigue las buenas prácticas definidas por el propio Azure con el formato [tipo-recurso]-[nombre-proyecto]-

[entorno]-[region], permitiendo identificar de un vistazo el propósito de cada componente.

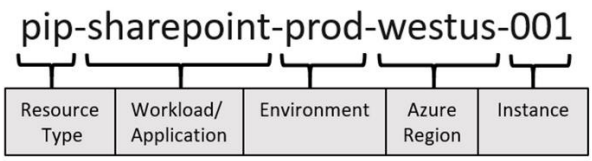


Figura 46. Ejemplo Nomenclatura Recurso Azure

Tipo de Recurso	Prefijo	Ejemplo Completo
Grupo de Recursos	rg	rg-docubot-poc-westeu-001
Cuenta de Almacenamiento	st	stdocubotpocwesteu
Bóveda de Claves	kv	kv-docubot-poc-westeu-001
Azure AI Search	ais	ais-docubot-poc-westeu-001
Azure OpenAI	oai	oai-docubot-poc-westeu-001
Base de Datos Cosmos DB	cosmos	cosmos-docubot-poc-westeu-001
Plan de App Service	plan	plan-docubot-poc-westeu-001
Aplicación (App Service)	app	app-docubot-poc-westeu-001

Tabla 17. Nomenclatura recursos Azure

8.1.3 Azure AI Search

Este recurso es el núcleo de la fase de Retrieval en la arquitectura RAG. Su función es indexar el contenido de los documentos corporativos y permitir búsquedas complejas y semánticas para encontrar la información más relevante a la pregunta de un usuario. Para lograrlo, la configuración de Azure AI Search no se limita a la creación del recurso principal, sino que implica la orquestación de cuatro subrecursos que definen un pipeline de indexación:

1. **Orígenes de Datos (Data Sources):** Especifican *de dónde* se obtendrá la información. Se pueblan durante la fase de ingesta y preprocesado de documentos.
2. **Conjuntos de Habilidades (Skillsets):** Definen las transformaciones de IA que se aplicarán a los datos de origen para enriquecerlos, como la generación de *embeddings*.
3. **Índices (Indexes):** Definen la estructura de los datos *una vez procesados*, es decir, el esquema sobre el cual se realizarán las búsquedas.
4. **Indexadores (Indexers):** Automatizan el proceso: conectan un origen de datos con un índice de destino, aplicando un conjunto de habilidades durante el camino para mover y transformar los datos.

A continuación, se detalla la configuración de cada uno de estos componentes y la estrategia general del servicio.

- **Nivel de servicio (SKU):** Se ha seleccionado el SKU Basic (S1), ya que ofrece un equilibrio adecuado entre coste y rendimiento para un entorno de producción inicial, permitiendo escalar si la demanda aumentase.
- Creación de **Índices:** Se han creado dos índices para dar soporte a la búsqueda híbrida: uno para resúmenes (*summaries-index*) y otro para el texto completo (*fulltext-index*). La definición de estos índices se realiza mediante una estructura JSON que especifica los campos, sus tipos de datos, y sus propiedades (si son filtrables, ordenables, etc.). A continuación, se muestra un extracto de la definición del índice de texto completo, destacando el campo vectorial que almacena los *embeddings*.

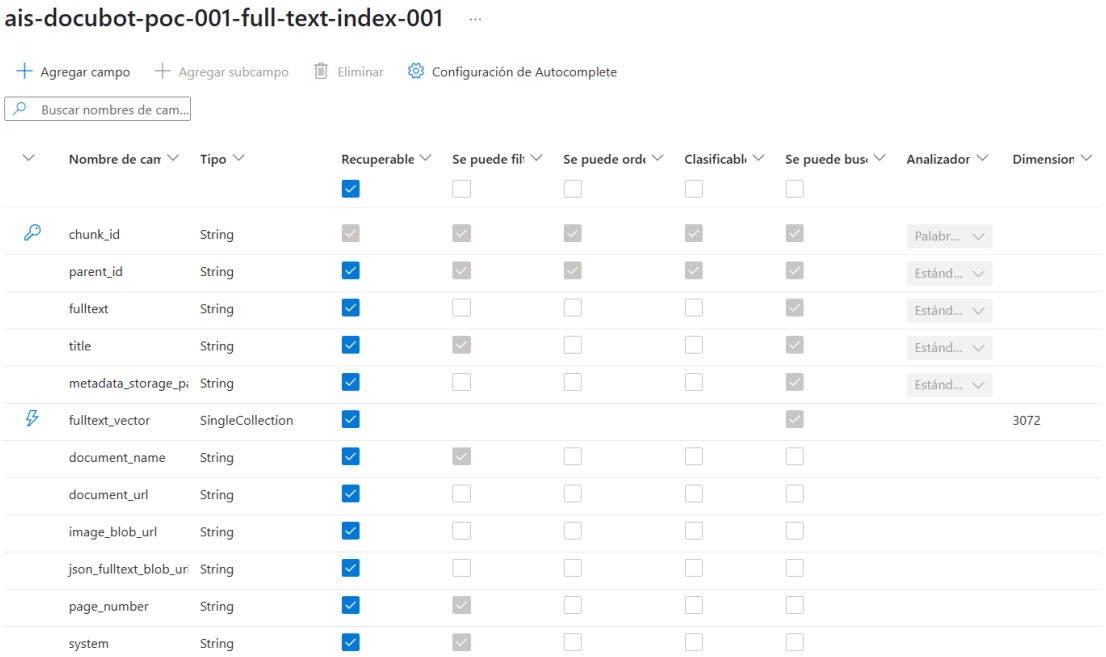


Figura 47. Campos del full-text index

- Creación de **DataSources**. Un origen de datos define la conexión a la fuente de la información que se va a indexar. En este proyecto, el conocimiento reside en documentos que han sido pre-procesados y almacenados en Azure Blob Storage. Se ha configurado un origen de datos que apunta al contenedor de Blob Storage donde se encuentran los archivos JSON o TXT con el texto extraído de cada página de documento.
- **Skillsets**: Es una secuencia de habilidades que procesan y enriquecen los datos en bruto antes de que lleguen al índice [55]. Para este proyecto, el skillset diseñado incluye *AzureOpenAIEmbeddingSkill*. Toma el contenido textual de un campo y realiza una llamada al modelo *text-embedding-large-003* para generar su representación vectorial. El vector resultante se mapea al campo correspondiente del índice.

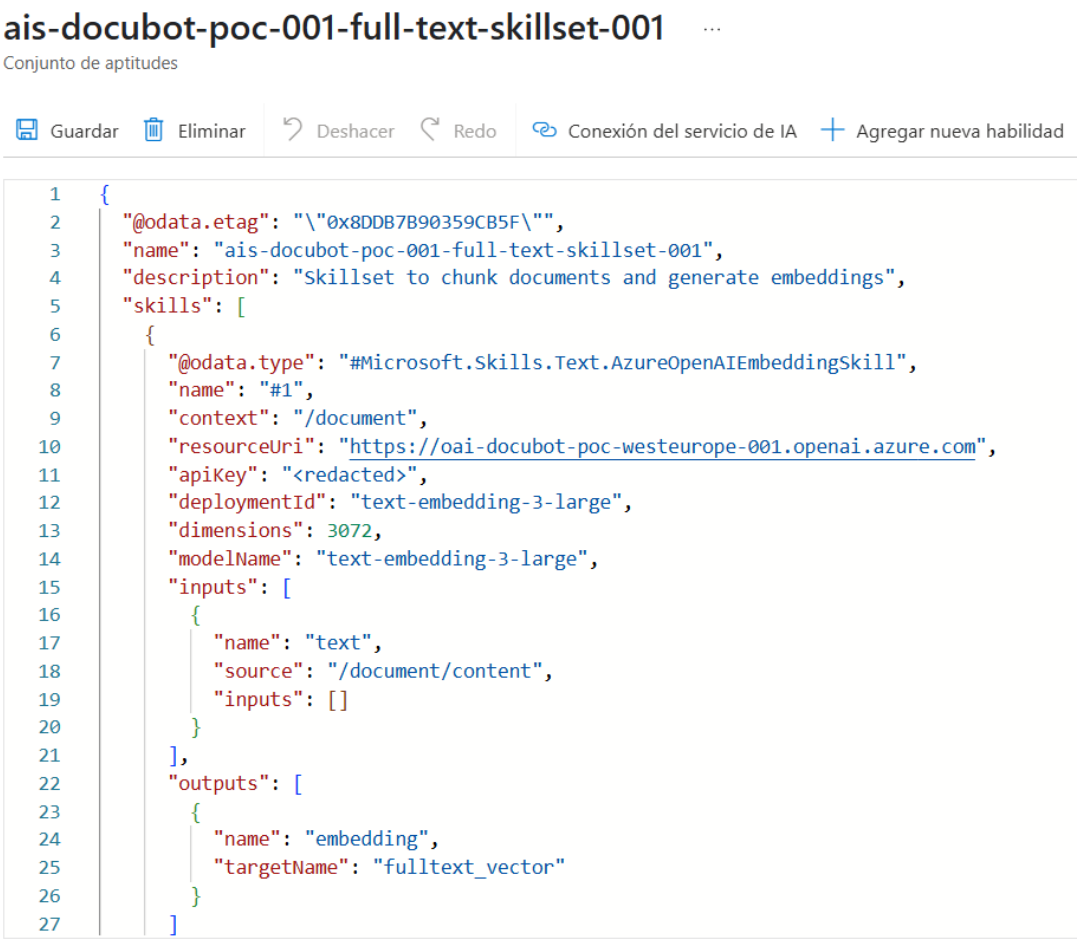


Figura 48. Ejemplo de creación de skillset

- El **indexador** es el componente que une todo el pipeline. Se configura para: primero conectar un origen de datos (Blob Storage) con un índice de destino (ej. fulltext-index). Aplicar un conjunto de habilidades (skillset) a los datos durante el proceso.

8.1.4 Cosmos Db

Azure Cosmos DB es la base de datos NoSQL con integración nativa con el ecosistema de Bot Framework a través de la librería Teams AI Library, el servicio se ha configurado de la siguiente manera:

- **Capacidad de Rendimiento:** Se ha configurado en modo Provisioned Throughput a nivel de base de datos. Esto permite compartir las Unidades de Solicitud (RU/s) entre los diferentes contenedores, optimizando los costes para una carga de trabajo que puede variar. En este caso se ha optado por provisonar 1000 RU/s de manera compartida.

Dentro de la base de datos *db-docubot-poc*, se han creado dos contenedores distintos, cada uno con un propósito específico:

1. Contenedor **conversations:** Su función es almacenar el estado de la conversación. Cada documento en este contenedor representa el estado completo de un diálogo específico, identificado por una clave única que combina el ID del canal, el ID del bot y el ID de la conversación. Esto asegura que cada interacción sea con memoria.
2. Contenedor **turns:** Este contenedor ha sido diseñado a medida para funcionar como un log de auditoría. A diferencia del contenedor de conversaciones, que puede ser modificado en cada turno, aquí se guarda un registro inmutable de cada ciclo completo de pregunta-respuesta-feedback. Cada documento representa un "turno" y almacena toda la información correspondiente.

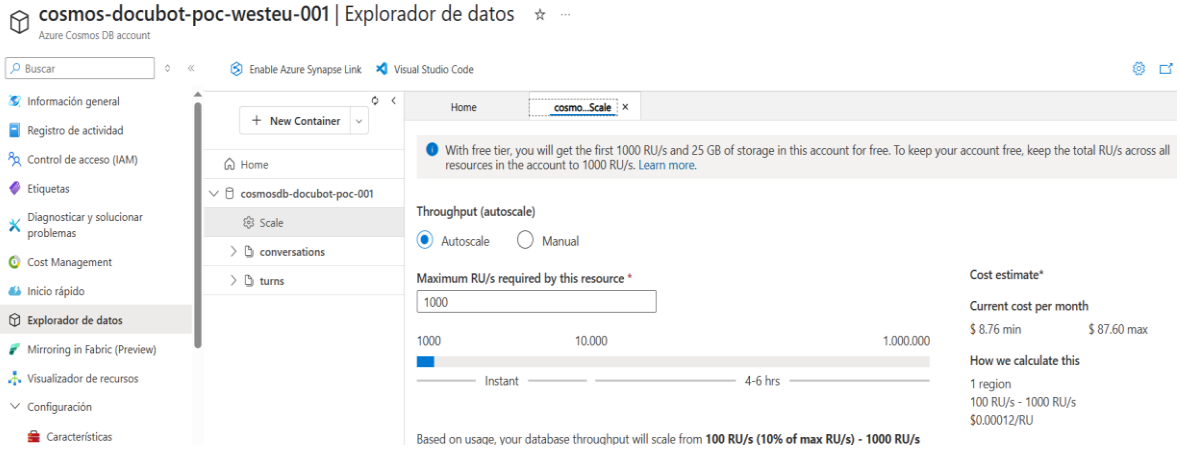


Figura 49. Configuración Inicial Cosmos DB

8.1.5 Blob Storage

Azure Blob Storage es el servicio de almacenamiento de objetos de Azure, diseñado en este `proyecto para guardar grandes cantidades de datos no estructurados. Actúa como el repositorio central para todos los blobs de datos generados durante la fase de pre-procesamiento de documentos.

Se han configurado cuatro contenedores principales para organizar los diferentes tipos de datos:

1. **Contenedor images-container:** Almacena las imágenes (JPG) extraídas de cada página de los documentos originales (PDFs, etc.). Cada imagen es una representación visual de una página.
2. **Contenedor full-text-container:** Almacena el contenido textual completo de cada página de documento, guardado en archivos JSON individuales. Cuando el bot necesita el contexto completo de un documento relevante para responder una pregunta, se utiliza para descargar el contenido de estos JSON "bajo demanda".

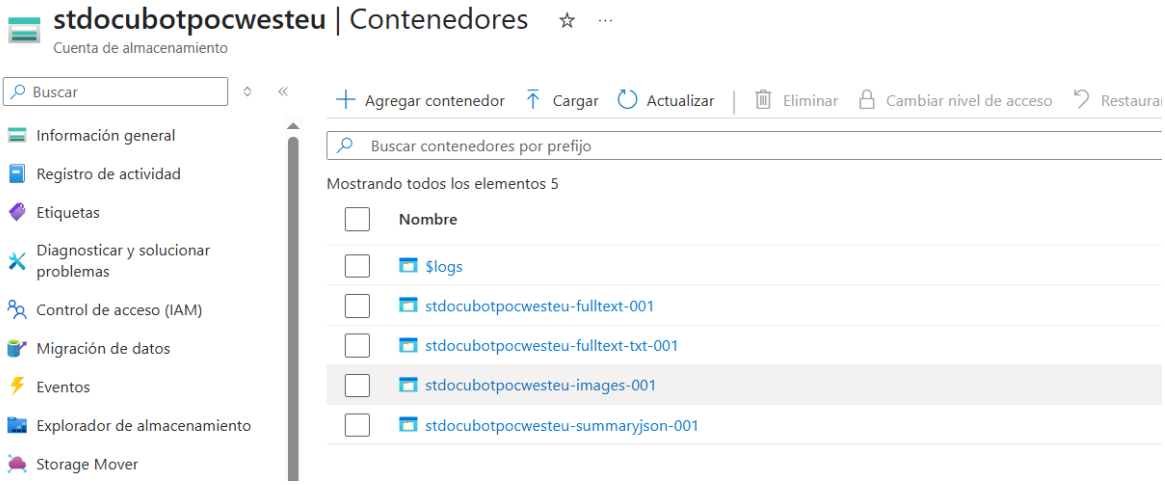
3. **Contenedor full-text-txt-container:** Almacena el mismo contenido que el contenedor anterior pero en vez de formato JSON en .txt para que pueda ser accedido en la fase de Retrieval por *AzureAISearchDataSource*.
4. **Contenedor summary-questions-container**: Este contenedor almacena datos enriquecidos por IA, generados durante el pre-procesamiento. Por cada página de documento, se crea un archivo JSON que contiene:
    - Un resumen (summary) del contenido de la página.
    - Una lista de posibles preguntas (questions) que podrían ser respondidas por esa página.
5. 

Figura 50. Contenedores de Azure Blob Storage

Esta duplicación dentro del método full-text es deliberada: el formato .txt está optimizado para la indexación masiva y eficiente por parte del indexador de Azure AI Search, mientras que el formato json es consumido en tiempo de ejecución para obtener la estructura del documento junto al texto.

El acceso a ambos contenedores se gestiona a través de las cadenas de conexión del servicio, que se almacenan de forma segura en Azure Key Vault.

8.1.6 Azure Open AI

El recurso Azure OpenAI proporciona acceso a los LLM de OpenAI, con las garantías de seguridad y cumplimiento de nivel empresarial de Azure. En este proyecto se aprovechan diferentes modelos desplegados para dos tareas muy específicas:

1. **Despliegue del Modelo de Lenguaje (*gpt-4o*):** Este es el modelo multimodal principal para las tareas generativas, con un tamaño máximo de contexto de 128.000 tokens.
    - Generación de Respuestas.

- Mejora de Consultas.
- Extracción de Citaciones.

2. **Despliegue del Modelo de Embeddings (*text-embedding-large-003*):** Este modelo es la base de la búsqueda semántica. Su función es convertir cadenas de texto en vectores, en este modelo en específico en vectores de 3072 dimensiones. Se utiliza en dos fases:
   - **Durante la Indexación:** Como parte del *skillset* de Azure AI Search.
   - **Durante la Búsqueda:** Cuando un usuario hace una pregunta, la aplicación llama a este modelo para convertir la pregunta en un vector. AI Search puede entonces comparar este vector con los vectores almacenados en el índice para encontrar los documentos más similares semánticamente, incluso si no comparten las mismas palabras clave.

Figura 51. Modelos desplegados en Azure OpenAI

8.1.7 App Service y Azure Bot

Estos dos recursos se despliegan en conjunto para alojar la aplicación y hacerla accesible desde Microsoft Teams.

App Service

Es el servicio PaaS que hospeda el código de la aplicación Node.js. Se encarga de gestionar el servidor web, el sistema operativo subyacente, el balanceo de carga y la escalabilidad, permitiendo que el desarrollo se centre exclusivamente en la lógica de la aplicación.

- **Plan de App Service (*plan-docubot-poc-westeu-001*):** Define los recursos computacionales (CPU, memoria, disco) asignados a la aplicación.

- **Configuración y Despliegue:** La configuración de la aplicación, incluidas las variables de entorno, se gestiona en el panel de Azure Portal del App Service.

Azure Bot

Este recurso no ejecuta código. Actúa como el registro de identidad del bot dentro del Microsoft Bot Framework. Es el que le permite comunicarse de forma segura con los servicios de Microsoft.

- **Endpoint de Mensajería:** En su configuración, se especifica la URL pública del App Service seguida del endpoint /api/messages. Esta es la dirección a la que el Bot Framework reenviará todas las actividades (mensajes, clics en botones, etc.) que el usuario realice en Teams.
- **Canales:** El recurso Azure Bot permite conectar la misma lógica de aplicación a múltiples plataformas. Para este proyecto, se ha habilitado y configurado el **canal de Microsoft Teams**, que adapta la comunicación y permite el uso de funcionalidades específicas de Teams.

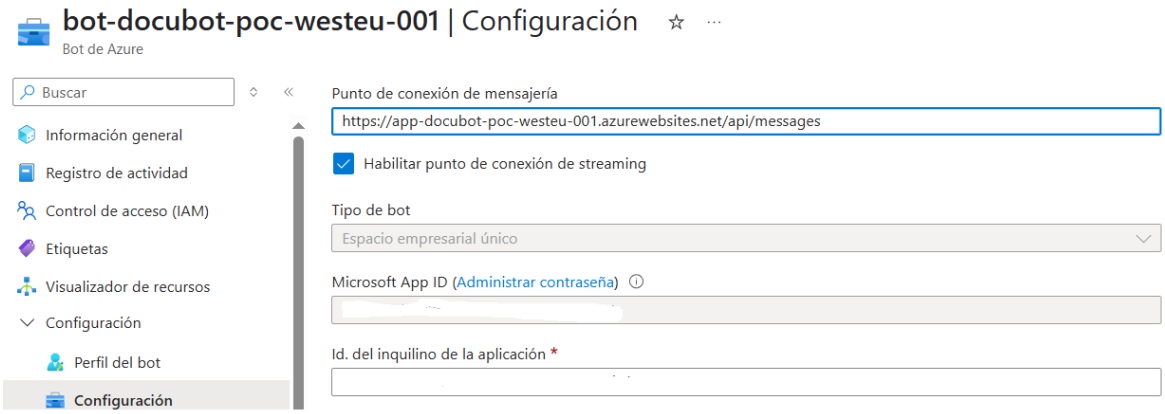


Figura 52. Configuración Bot Service

8.2 Flujo de Trabajo y Entorno de Desarrollo

Como se introdujo en el Capítulo de Planificación y gestión del Proyecto, la gestión del código fuente se ha realizado utilizando **Git**, con el repositorio alojado en **GitHub**. Para organizar el ciclo de desarrollo, se ha adoptado una adaptación simplificada del conocido modelo **Gitflow**, optimizada para la escala individual y naturaleza de este proyecto.

El modelo Gitflow tradicional, con sus diversas ramas develop, release y hotfix, da una estructura sólida para equipos de desarrollo grandes y ciclos de lanzamiento complejos. Sin embargo, para este proyecto se consideró que una versión más ágil y directa sería

más eficiente, manteniendo siempre los principios fundamentales de organización y estabilidad. Esta simplificación consistió en utilizar principalmente dos tipos de ramas:

- **Rama main:** Ha sido la única rama de larga duración, representando en todo momento el código estable y desplegable del proyecto. Cada fusión de código a la rama main simboliza una versión funcional o un incremento significativo del producto.
- **Ramas de Funcionalidad (feature/*):** Todo nuevo desarrollo, ya sea la implementación de una nueva característica, una mejora en el rendimiento o la corrección de un error se ha llevado a cabo en una rama de funcionalidad (feature) aislada. Estas ramas se crean siempre a partir del último estado de main y, una vez la funcionalidad estaba completa y probada de forma unitaria, se fusionaban de nuevo a main mediante una *Pull Request*.

Esta simplificación del flujo de trabajo se justifica por varias razones:

- **Eficiencia en un Proyecto Individual**. La complejidad de gestionar múltiples ramas de larga duración como develop y release no aportaba beneficios proporcionales al esfuerzo de mantenimiento, ya que la coordinación entre miembros de un equipo no era un factor.
- **Agilidad y Rapidez:** Un flujo más directo desde una rama feature a main ha permitido una mayor agilidad y una entrega más rápida de los incrementos funcionales.
- **Claridad del Repositorio:** Este enfoque mantiene una claridad meridiana sobre el estado del código: main contiene el código estable, mientras que las ramas feature/* contienen el trabajo en curso.

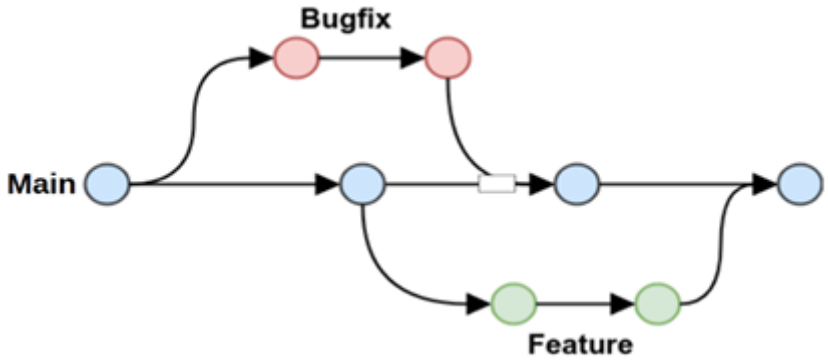


Figura 53. Diagrama Git simplificado

8.2.1 Entorno de Desarrollo Aislado y Reproducible

Para garantizar la consistencia y facilitar la configuración del complejo entorno de desarrollo que requiere este proyecto, se ha adoptado una estrategia basada en la tecnología de contenedores. En lugar de instalar manualmente las dependencias

(Node.js, Teams Toolkit, librerías específicas, etc.) directamente en el sistema operativo anfitrión, se ha utilizado la funcionalidad **Dev Containers** de Visual Studio Code [56].

Un Dev Container define un entorno de desarrollo completo, autocontenido y reproducible dentro de un contenedor Docker. Esto ofrece varias ventajas clave:

- **Consistencia:** Siempre se trabaja con exactamente las mismas versiones de herramientas y dependencias, eliminando el clásico problema de "en mi máquina funciona".
- **Aislamiento:** Las dependencias del proyecto no "contaminan" el sistema operativo principal.
- **Facilidad de Configuración:** Para empezar a trabajar, solo es necesario tener instalado Docker y Visual Studio Code con la extensión Dev Containers. El resto del entorno se construye automáticamente a partir de una configuración predefinida.

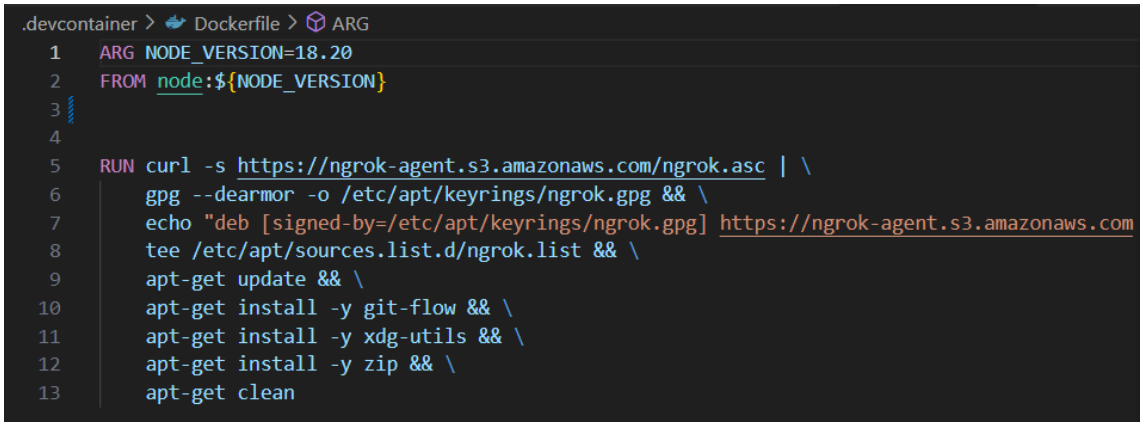


Figura 54. Dockerfile del DevContainer

La ejecución de Docker en un entorno Windows se ha realizado a través de **WSL** (Windows Subsystem for Linux). Esta arquitectura permite ejecutar un motor de Docker nativo de Linux directamente en Windows, ofreciendo un rendimiento superior y una mejor compatibilidad en comparación con soluciones más antiguas. La jerarquía resultante es un entorno de desarrollo robusto y eficiente, virtualizado sobre un subsistema Linux que a su vez se ejecuta en Windows.

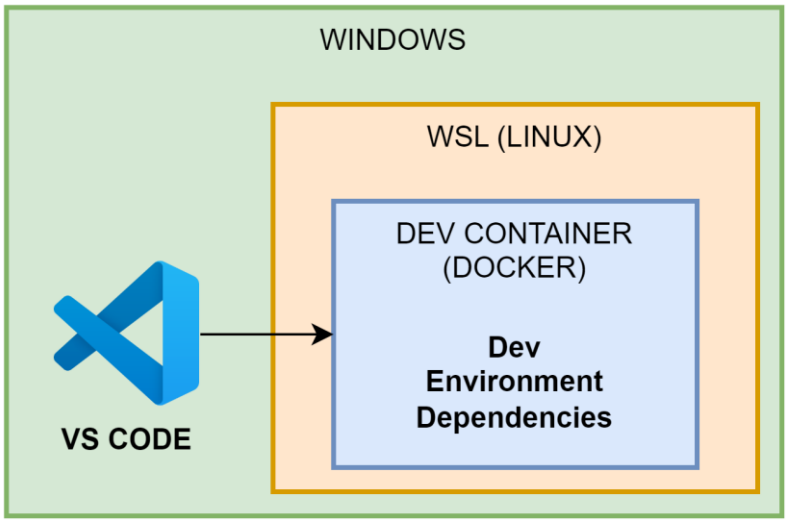


Figura 55. Esquema de Entorno Virtual Usado

8.3 Implementación Ingesta y Preprocesado

Antes de que DocuBot pueda responder a las preguntas de los usuarios, es necesario alimentar su base de conocimiento. Este proceso, de ingesta o preprocesado de documentación, no consiste simplemente en copiar ficheros, sino en un flujo de trabajo de preprocesado que transforma los documentos originales (PDFs) en un formato estructurado, enriquecido y optimizado para la búsqueda semántica a través de llamadas a GPT-4o a través de Azure OpenAI.

Para llevar a cabo esta tarea, se ha desarrollado un script de preprocesado en **Python 3.12**, independiente del bot principal, que actúa como un pipeline de ETL (*Extract, Transform, Load*). Esta función de prerpocesamiento de la base documental se encuentra al completo adjunto en uno de los repositorios y es el responsable de poblar los servicios de Azure Blob Storage (que después servirán de *datasource* para el AI Search) con la información necesaria.

El script se ejecuta bajo demanda del administrador (posible automatización en el futuro) y está diseñado para ser configurable, operando en diferentes "modos": Modo “summary” para generar solo resúmenes o modo “full_text” extraer texto completo según las necesidades. Todo el proceso está orquestado desde el fichero *process_pdfs.py.*

8.3.1 Entorno de Ejecución y Dependencias

El script se ejecuta en un entorno de **Conda** definido en el fichero environment.yml. La elección de Conda permite encapsular todas las dependencias en un entorno aislado y reproducible, asegurando que el script funcione de manera consistente.

environment.yml

```yaml
# environment.yml
name: docubot-preproc  # Nombre del entorno

channels:
  - conda-forge
  - defaults

dependencies:
  - python=3.9         # Usamos una versión estable de Python
  - pip
  # Dependencias que instalaremos con pip
  - pip:
    - office365-rest-python-client
    - azure-storage-blob
    - pdf2image
    - Pillow
    - requests
```

Figura 56. Entorno Ingesta y Preprocesado

8.3.2 Flujo de Procesamiento Orquestado

El script orquesta un flujo de trabajo secuencial y modular que se puede resumir en los siguientes pasos:

1. **Carga de Configuración:** Al iniciar, el script carga un fichero config.json que define todos los parámetros de la operación: credenciales de los servicios (SharePoint, Azure, OpenAI), nombres de los contenedores, y, de forma crucial, el "modo" de procesamiento. Esto permite, por ejemplo, ejecutar el script para generar resúmenes (*summary mode*) o para extraer el texto completo (*full_text mode*) de los documentos, cada uno con sus propios *prompts* y contenedores de destino.
2. **Conexión a las Fuentes de datos y Destinos:**
    - **SharePoint:** Se establece una conexión con la biblioteca de documentos de SharePoint especificada, utilizando un flujo de autenticación interactivo para mayor seguridad. El script verifica los permisos para asegurar que tiene acceso de lectura a los documentos.

```python
def get_sharepoint_context(config: Dict) -> Union[ClientContext, None]:
    """Conecta a SharePoint y verifica los permisos."""
    logging.info(f"Conectando a SharePoint en: {config['sharepoint']['site_url']}")
    try:
        # Usar flujo interactivo para la autenticación
        sp_ctx = ClientContext(config['sharepoint']['site_url']).with_interactive(
            tenant=config['sharepoint']['tenant'],
            client_id=config['sharepoint']['client_id']
        )

        web = sp_ctx.web
        sp_ctx.load(web)
        sp_ctx.execute_query()
        logging.info(f"Conexión exitosa a SharePoint. Título del sitio: {web.properties['Title']}")

        # Verificar permisos antes de continuar
        if not check_sharepoint_permissions(sp_ctx, config):
            logging.error("Permisos de SharePoint insuficientes. Corrige los permisos y vuelve a intentar.")
            return None

        return sp_ctx
```

Figura 57. Conexión a Sharepoint

- **Azure Blob Storage:** Se conecta a la cuenta de Azure Storage y verifica la existencia de los contenedores de destino (para imágenes, JSONs y txt), creándolos si no existen.

3. **Extracción (*Extract*): Identificación de Documentos a Procesar**
   - El script consulta la biblioteca de SharePoint para obtener una lista de todos los ficheros PDF que cumplen un criterio específico (en este caso, un campo de metadatos llamado IA con valor 1). Esto permite un control granular sobre qué documentos se incluyen en la base de conocimiento del bot y permite dejar documentos que no se quieran tener en cuenta.
4. **Transformación (*Transform*): Procesamiento de cada Documento**
   - Este es el paso más intensivo. Para cada PDF identificado, el script realiza un bucle por cada una de sus páginas, ejecutando la siguiente secuencia:
     a. **Descarga y Conversión:** Descarga el PDF a un directorio temporal y utiliza pdf2image para convertir la página actual en una imagen JPEG.
     b. **Análisis con IA (Visión y Lenguaje):** Envía esta imagen al modelo multimodal de Azure OpenAI (GPT-4o). El modelo "lee" la imagen de la página, se le proporciona un *prompt* específico dependiendo del modo de ejecución. La respuesta del modelo es un objeto JSON estructurado con la información extraída.
     c. **Enriquecimiento de Metadatos:** Se extraen metadatos relevantes del documento en SharePoint (ej. nombre, fecha de creación, sistema de documentación al que pertenece) para asociarlos a los datos generados.
5. **Carga (*Load*): Almacenamiento de los Artefactos en Azure**
   - Los resultados de la transformación se suben a Azure Blob Storage:

- La imagen de la página se guarda en un contenedor de imágenes.
- 2 blob de json que son para el output de los 2 métodos de procesamiento, uno para el summary y otro para el full_text.
- En modo full_text el texto extraído se aplana y se guarda como un fichero .txt en otro contenedor, optimizado así para la indexación de texto completo en Azure AI Search.

- Crucialmente, los blobs se interconectan mediante **metadatos**. Por ejemplo, el blob de la imagen contiene en sus metadatos la URL del JSON correspondiente y viceversa. Esto permite que, durante la ejecución del bot, se pueda navegar fácilmente entre los diferentes artefactos de una misma página.

El siguiente fragmento de *process_file* ilustra el núcleo de la fase de Transformación. Se ha seleccionado porque muestra la llamada a requests.post que envía la imagen de la página a la API de OpenAI, un ejemplo claro de cómo se utiliza la capacidad de visión del modelo para entender el contenido del documento (imágenes, diagramas, dibujos…).

src > process_pdfs.py

```
213  def process_file(item, sp_ctx: ClientContext, blob_service_client: BlobServiceClient, config: Dict, overwrite: bool):
235      with tempfile.TemporaryDirectory() as temp_dir:
253          for i, image in enumerate(images):
265              try:
266                  img_byte_arr = io.BytesIO()
267                  image.save(img_byte_arr, format='JPEG')
268                  image_bytes = img_byte_arr.getvalue()
269                  base64_image = base64.b64encode(image_bytes).decode('utf-8')
270
271                  headers = {"Content-Type": "application/json", "api-key": config["openai"]["api_key"]}
272                  payload = {
273                      "model": config["openai"]["model"], "response_format": {"type": "json_object"},
274                      "messages": [
275                          {"role": "system", "content": config["mode_config"]["system_prompt_text"]},
276                          {"role": "user", "content": [{"type": "text", "text": config["mode_config"]["prompt_text"]},
277                      ],
278                      "max_tokens": config["openai"]["max_tokens"]
279                  }
280
281                  logging.info("Enviando imagen a OpenAI para análisis...")
282                  response = requests.post(config["openai"]["api_url"], headers=headers, json=payload)
283                  response.raise_for_status()
284                  ai_response_str = response.json()["choices"][0]["message"]["content"]
285                  ai_data = json.loads(ai_response_str)
286              except Exception as e: ···
```

Figura 58. Llamadas al LLM en el Preprocesado

Este proceso de ingesta desacoplado del bot principal es fundamental para la arquitectura del sistema. Permite que la base de conocimiento del bot se actualice de forma asíncrona y controlada, sin afectar al chatbot, y asegura que los datos que consume Azure AI Search estén disponibles y en su última versión en todo momento.

8.3.3 Estructura de los Blobs Generados

El resultado final de este pipeline de preprocesado es un conjunto de blobs que está interconectados a través de los respectivos metadatos en Azure Blob Storage para cada página de cada documento procesado. La estructura de estos datos, especialmente de los ficheros JSON generados por la IA, es la base sobre la que se construye la base documental de DocuBot.

En el modo de resumen, el objetivo es que el LLM extraiga los conceptos clave, un resumen conciso y posibles preguntas que la página podría responder. Este formato es ideal para un índice de búsqueda semántica que prioriza la relevancia conceptual sobre el texto literal, y además al proporcionar 2 campos: preguntas y resumen permite realizar esa búsqueda semántica sobre una combinación de ambos.


```
{
  "summary": "The document outlines AI use cases categorized under various sectors, highlighting their applications and implications. In
non-banned biometrics, it includes systems like emotion recognition and biometric categorization. Critical infrastructure points to AI's
role in safety management for vital resources like electricity and water. Education and vocational training focuses on student assessment,
admission, and learning outcomes. Employment and self-employment covers AI in recruitment, contracts, and performance evaluations. Public
and private services discuss AI usage in assessing benefits, creditworthiness, and emergency calls. Law enforcement highlights risk
assessments, profiling, and polygraphs. Migration and border control involves health risk assessments and visa evaluations. Administration
of justice and democratic processes addresses AI in legal fact analysis and influencing elections, excluding indirect tools like campaign
optimizations.",
  "questions": "1. What AI systems fall under non-banned biometrics? 2. What role does AI have in the management of critical infrastructure?
3. How are AI systems used in education and vocational training? 4. What AI applications are listed for employment, workers management, and
self-employment? 5. How does AI assist in access to essential public and private services? 6. What are the AI use cases mentioned in law
enforcement? 7. How is AI utilized in migration, asylum, and border control management? 8. What applications of AI are discussed in
administration of justice and democratic processes?"
}
```


Figura 59. Ejemplo de json en modo Summary

En el modo de texto completo, el *prompt* instruye al LLM para que realice una transcripción literal y estructurada de todo el contenido de la página, a menudo organizándolo en secciones lógicas. Este formato es fundamental para el índice de búsqueda de texto completo y de keywords, así como para que DocuBot pueda citar fragmentos exactos en sus respuestas.


```
{
  "header": {
    "sectionNumber": "1",
    "title": "General Overview",
    "subtitle": "Executive summary"
  },
  "approvalStatement": "On May 2024, the Council approved the AI Act, bringing the legislative process to an end",
  "sections": [
    {
      "title": "Context",
      "content": [
        "The AI Act comes in response to the growing application and potential of AI systems in various sectors, along with the need to
address the potential risks and harms that these systems may cause to public interests, health, safety and fundamental rights protected by
the EU.",
        "The proposal was presented by the EC in 2021, followed by opinions from various bodies such as the ECB and the European Economic
and Social Committee.",
        "After the EP's and Council's adoption at first reading in May 2024, the document was published in the OJEU on 13 June 2024."
      ]
    },
    {
      "title": "Objective",
      "content": [
        "The main objective of the AI Act is to improve the functioning of the internal market by establishing a uniform legal framework for
the development, marketing, use and servicing of AI systems in the EU.",
        "This is done with the intention of promoting the adoption of human-centered and reliable AI, while ensuring a high level of
protection against the harmful effects of AI systems and supporting innovation."
      ]
```


Figura 60. Fragmento de json en modo FullText

Se genera otro contenedor de Blobs que contienen este mismo documento con toda la información de cada página pero en texto plano en un .txt para facilitar la ingesta por Azure AI Search.

Estos JSONs estructurados, TXT, junto con las imágenes y los metadatos asociados, forman el corpus procesado de conocimiento que Azure AI Search va a indexar. En la arquitectura RAG de DocuBot, el *AzureAISearchDataSource* buscará dentro de estos ficheros para encontrar los fragmentos más relevantes, que luego serán utilizados por el LLM para generar la respuesta final.

8.4 Implementación del Núcleo Conversacional y Orquestación de IA

El corazón del sistema es un servicio de Node.js que no solo responde a las interacciones del usuario, sino que orquesta un complejo flujo de trabajo de IA. Es responsable de interpretar la entrada, coordinar la recuperación de información, invocar modelos de lenguaje y formular una respuesta coherente y contextualizada.

La elección de la plataforma tecnológica ha sido estratégica y se basa en dos pilares:

1. **TypeScript:** Se ha utilizado TypeScript como lenguaje de programación. Su fiabilidad es crucial en una aplicación empresarial que integra múltiples servicios de Azure.
2. **Microsoft Teams AI Library (@microsoft/teams-ai):** La implementación se fundamenta en esta biblioteca oficial de Microsoft, disponible para Python y TypeScript/JavaScript [57]. Su uso es clave, ya que abstrae gran parte de la complejidad de la comunicación con el Bot Framework y proporciona herramientas de alto nivel como:
    - Un ActionPlanner para invocar modelos de lenguaje (como los de Azure OpenAI).
    - Gestión simplificada del estado de la conversación (TurnState).
    - Manejo de la moderación de contenido y bucles de reparación de diálogo.

La elección de esta librería, y por tanto de TypeScript, ha permitido centrar el desarrollo en la lógica de negocio y la orquestación de la IA, en lugar de en la comunicación con la API de Teams.

8.4.1 Arquitectura del Servidor y Composición Patron Builder

La arquitectura lógica, definida en el capítulo anterior bajo el paradigma de la Arquitectura Hexagonal, establece una separación estricta entre el núcleo de la aplicación y sus dependencias externas. Para materializar esta separación a nivel de código y gestionar la compleja inicialización de los componentes, se ha tomado una decisión de diseño crucial: la implementación del patrón de diseño Builder.

Este patrón se manifiesta en la interacción entre los ficheros *index.ts* (el punto de entrada) y *app.ts* (el ensamblador de la aplicación), y es fundamental para mantener el desacoplamiento definido en la arquitectura.

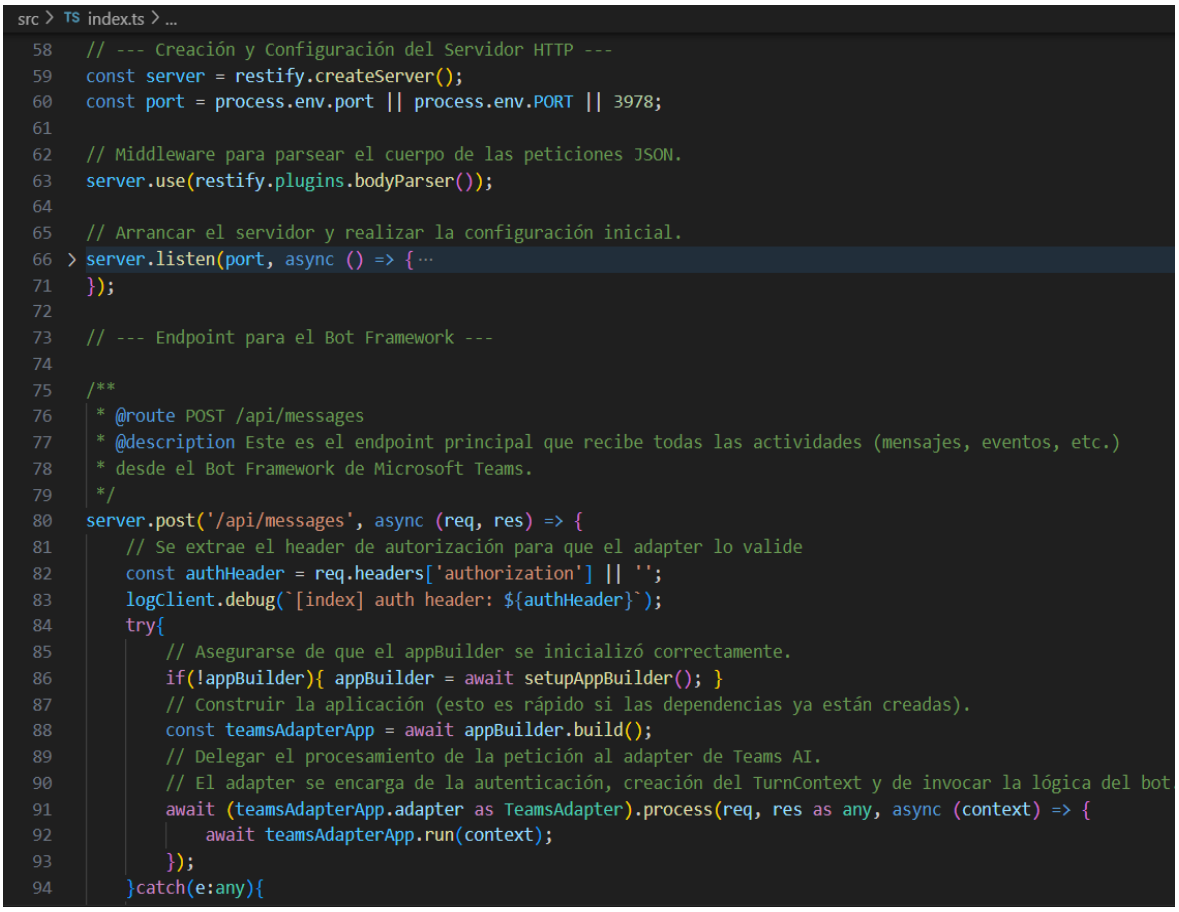
Punto de Entrada (*src/index.ts*): Adaptador de Infraestructura

Siguiendo el modelo de Puertos y Adaptadores, el fichero src/index.ts actúa como un adaptador de infraestructura primario. Su responsabilidad es ser la base de la aplicación es el servidor Node.js que actúa como el punto de escucha para todas las interacciones y delegarlas al núcleo de aplicación. Como servidor HTTP, se ha optado por Restify, una elección deliberada por su ligereza y alto rendimiento, ideal para un servicio de API que debe responder rápidamente al Bot Framework.

El fichero *src/index.ts* es el punto de entrada de la aplicación. Sus responsabilidades se centran en la inicialización y configuración del entorno:

- **Configuración Inicial:** Carga las variables de entorno desde un fichero .env. Se tiene además un *.env.template* para tener el esquema de las variables a usar.
- **Inicialización del Logger:** Se configura un logger asíncrono utilizando **Pino** con pino-pretty, que formatea los logs de manera legible durante el desarrollo.
- **Creación del Servidor:** Instancia el servidor Restify y define el endpoint principal /api/messages. Este endpoint es el receptor de todas las actividades (mensajes, eventos, interacciones) enviadas desde la plataforma de Microsoft Teams.
- **Arranque y Composición Inicial:** Una vez que el servidor se inicia y está listo para recibir peticiones, invoca a *setupAppBuilder*. Esta función es el primer paso en la composición de la aplicación**.**

El siguiente fragmento de código ha sido seleccionado para ilustrar el mecanismo de recepción y procesamiento de peticiones. Muestra cómo el servidor Restify, en su endpoint /api/messages, delega la petición HTTP cruda al TeamsAdapter. Este adaptador es el que traduce la petición en un TurnContext, el objeto central que encapsula toda la información de la interacción y que será utilizado por el resto de la aplicación.



```
src > TS index.ts > ...
58  // --- Creación y Configuración del Servidor HTTP ---
59  const server = restify.createServer();
60  const port = process.env.port || process.env.PORT || 3978;
61
62  // Middleware para parsear el cuerpo de las peticiones JSON.
63  server.use(restify.plugins.bodyParser());
64
65  // Arrancar el servidor y realizar la configuración inicial.
66  server.listen(port, async () => {···
71  });
72
73  // --- Endpoint para el Bot Framework ---
74
75  /**
76   * @route POST /api/messages
77   * @description Este es el endpoint principal que recibe todas las actividades (mensajes, eventos, etc.)
78   * desde el Bot Framework de Microsoft Teams.
79   */
80  server.post('/api/messages', async (req, res) => {
81      // Se extrae el header de autorización para que el adapter lo valide
82      const authHeader = req.headers['authorization'] || '';
83      logClient.debug(`[index] auth header: ${authHeader}`);
84      try{
85          // Asegurarse de que el appBuilder se inicializó correctamente.
86          if(!appBuilder){ appBuilder = await setupAppBuilder(); }
87          // Construir la aplicación (esto es rápido si las dependencias ya están creadas).
88          const teamsAdapterApp = await appBuilder.build();
89          // Delegar el procesamiento de la petición al adapter de Teams AI.
90          // El adapter se encarga de la autenticación, creación del TurnContext y de invocar la lógica del bot
91          await (teamsAdapterApp.adapter as TeamsAdapter).process(req, res as any, async (context) => {
92              await teamsAdapterApp.run(context);
93          });
94      }catch(e:any){
```

Figura 61. Creación del servidor Restify y endpoint de mensajes

El Ensamblador de la Aplicación (src/app.ts) y el Patrón Builder

Aquí reside el inicio de la arquitectura de software del proyecto. Para gestionar la complejidad de inicializar y conectar múltiples componentes asíncronos (clientes de Azure, modelos de IA, servicios de autenticación), se implementa el patrón Builder para orquestar la creación del objeto de aplicación, un proceso complejo debido a las múltiples dependencias asíncronas [58].

La clase *AppBuilder* es la implementación de este patrón. Su propósito no es ejecutar la lógica de negocio, sino construir el objeto de aplicación completo y funcional. El proceso es el siguiente:

1. **Recepción de Dependencias "Builder":** El constructor de *AppBuilder* recibe como parámetros no las dependencias finales, sino "builders" para cada una de ellas (GraphAuthServiceBuilder, OpenAIModelBuilder, AzureAISearchBuilder, ...). Se encapsula la lógica de obtención de secretos y configuración.
2. **Orquestación de la Construcción Asíncrona:** El método *build()* de *AppBuilder* llama de forma asíncrona a los métodos *build()* de cada "builder" individual para crear las instancias finales de los servicios.
3. **Inyección de Dependencias:** Una vez que todas las dependencias están construidas, *AppBuilder* las inyecta en los componentes que las necesitan. Por

ejemplo, instancia el *ActivityController* pasándole el servicio de Graph, el de AI Search y el planificador de IA, todos ya listos para usar.

4. **Ensamblaje Final:** Finalmente, crea la instancia principal de la Application de Teams AI, la configura con los manejadores de actividad (mapeando cada tipo de interacción a un método del *ActivityController*) y la devuelve.

Figura 62. Registro de actividades del bot

El uso del patrón Builder en este contexto es la clave para:

- **Mantener el Desacoplamiento:** El punto de entrada (index.ts) no sabe nada sobre cómo se construye Cosmos DB o AI Search. Solo conoce la interfaz del AppBuilder.
- **Gestionar la Asincronía:** Centraliza toda la lógica de await de las inicializaciones en un único lugar, simplificando el resto del código.

8.4.2 Monitorización y Logging con Pino

Una capacidad esencial para la mantenibilidad y depuración de la aplicación es su sistema de monitorización. Se ha implementado un robusto sistema de logging utilizando la librería **Pino**, una elección deliberada por su alto rendimiento y bajo overhead, ideal para aplicaciones Node.js en producción [59]. Esto proporciona una traza detallada que es invaluable para identificar comportamientos anómalos, optimizar el rendimiento y depurar errores.

La verbosidad del logging es configurable dinámicamente a través de la variable de entorno LOGGER_LEVEL. Por defecto, se establece en un nivel INFO para entornos de producción, pero puede ajustarse a DEBUG para obtener una traza mucho más

detallada durante la investigación de un incidente específico, sin necesidad de redesplegar el código.

8.4.3 El Orquestador Lógico: ActivityController

Si *index.ts* es el punto de entrada, ActivityController es el orquestador de DocuBot. Esta clase, instanciada en AppBuilder, recibe todas las dependencias necesarias mediante inyección (como el ActionPlanner, AzureAISearchDataSource, etc.), lo que facilita las pruebas unitarias y el desacoplamiento. Su funcionalidad es orquestar la secuencia de acciones para cada tipo de interacción del usuario.

Su método más importante es *handleUserQuestion*, ya que implementa el patrón **RAG**. El flujo que orquesta es el siguiente:

1. **Retrieval (Recuperación):** Invoca al servicio *AzureAISearchDataSource* para buscar los fragmentos de documentos más relevantes para la pregunta del usuario.
2. **Augmentation (Aumentación):** Utiliza los manejadores de Blob Storage para descargar el contenido completo (texto e imágenes) de los documentos recuperados, creando un contexto enriquecido para el prompt.
3. **Generation (Generación):** Invoca al ActionPlanner de la Teams AI Library, que se comunica directamente con el modelo de Azure OpenAI. El modelo recibe la pregunta del usuario junto con el contexto aumentado, así como el histórico de la conversación y genera una respuesta basada en la evidencia proporcionada.

Diseño del Prompt Principal de Generación

La fase de Generación culmina con la construcción de un prompt dinámico y multimodal que se envía al modelo GPT-4o. Este prompt provee de la capacidad de razonamiento de DocuBot y está cuidadosamente diseñado para maximizar la fiabilidad y minimizar las alucinaciones.

La estrategia del prompt se basa en los siguientes principios:

1. **Rol Explícito:** Al modelo se le asigna el rol de "asistente experto" para guiar su tono y estilo.
2. **Contexto Cerrado (*Grounding*):** Se le instruye explícitamente que debe basar su respuesta **única y exclusivamente** en el contexto proporcionado. Se le prohíbe el uso de conocimiento externo.
3. **Manejo de Incertidumbre:** Se le indica que si la respuesta no se encuentra en el contexto, debe comunicarlo claramente en lugar de intentar una respuesta.

A continuación, se muestra una plantilla del prompt utilizado:

src > prompts > chat > skprompt.txt

```
The following is a conversation with an AI assistant who is an expert in answering questions based on the provided context.

The context will be provided as a list of images, each image being a page of a document containing text, graphs, tables, and

Assistant's Instructions:

You should use a step-by-step reasoning process (chain-of-thought) to analyze the provided documents and generate the most ac

Use only the information contained in the provided documents to answer the user's question. Do not use any external knowledge

Before providing your answer, carefully verify that all information is accurate and directly supported by the provided docume

The answer can be constructed with information from one or more pages, but all pages must be from the same reference document

If the answer or part of it is based on an image, table, or diagram, explain their content in detail instead of just referenc

When including information from a source, reference each part of the answer using the following format: [Code + document name

If the answer is a list of terms or concepts, provide a brief explanation of each one.

If asking a clarifying question to the user would help improve the answer, ask the question.

If you cannot answer using the sources provided, indicate that you do not have sufficient information to answer. Return only

Remember to respond in a clear, concise, and professional manner, focusing on providing the most relevant and useful informat

Note: While you should perform step-by-step reasoning internally to arrive at the answer, only provide the final answer to th

In addition to the list of images given, also the transcription of that pages given as images is going to be provided to impr
Context from Documents:
 {{ $documentContext }}
```

Figura 63. Prompt Principal de Generación de respuesta final

Para ilustrar este flujo RAG, se ha seleccionado dos fragmentos de *handleUserQuestion*. Permite ver cómo se encadenan las llamadas a los diferentes servicios (AI Search, Blob Storage, OpenAI) para construir la respuesta final.

src > controllers > activityController.ts > ActivityController

```
export class ActivityController {

    /**
     * Handles user questions.
     * @param context - The turn context.
     * @param state - The application turn state.
     */
    private async handleUserQuestion(context: TurnContext, state: ApplicationTurnState): Promise<void> {
        this.logClient.info('[handleUserQuestion] Handling user question');
        let response: PromptResponse<string> | null = null; // Initialize response variable
        try {
            // 1. Retrieve relevant documents based on query and state (categories)
            this.logClient.debug('[handleUserQuestion] Retrieving documents from AI Search.');
            const documentsFromSearch = await this.aisearchDataSource.getUrls(context, state);
            this.logClient.debug(`[handleUserQuestion] Retrieved ${documentsFromSearch.length} documents from AI Search.`);
            this.logClient.debug(`[Handle User Question] Documents: ${JSON.stringify(documentsFromSearch)}`);
            if (documentsFromSearch.length === 0) {
                this.logClient.info('[handleUserQuestion] No relevant documents found by AI Search.');
                await context.sendActivity(process.env.NOT_INFORMATION_MESSAGE || "I couldn't find specific information related to your question
                await this.updateConversationTs(context, state); // Update timestamp even if no answer
                return;
            }
            const imagesUrls = documentsFromSearch.map(d => d.imageUrl).filter(Boolean) as string[];
            const docsUrlsBase64 = documentsFromSearch.map(d => d.docUrl).filter(Boolean) as string[];
            const documentTypes = documentsFromSearch.map(d => d.documentType).filter(Boolean) as string[];
            const jsonFullTextUrls = documentsFromSearch.map(d => d.jsonFulltextUrl).filter(Boolean) as string[];
            this.logClient.debug(`[Handle User Question] Images URLs: ${JSON.stringify(imagesUrls)}`);
            this.logClient.debug(`[Handle User Question] Full Text URLs: ${JSON.stringify(jsonFullTextUrls)}`);
```

Figura 64. (a) Fase de Recuperación: Búsqueda de documentos en Azure AI Search.

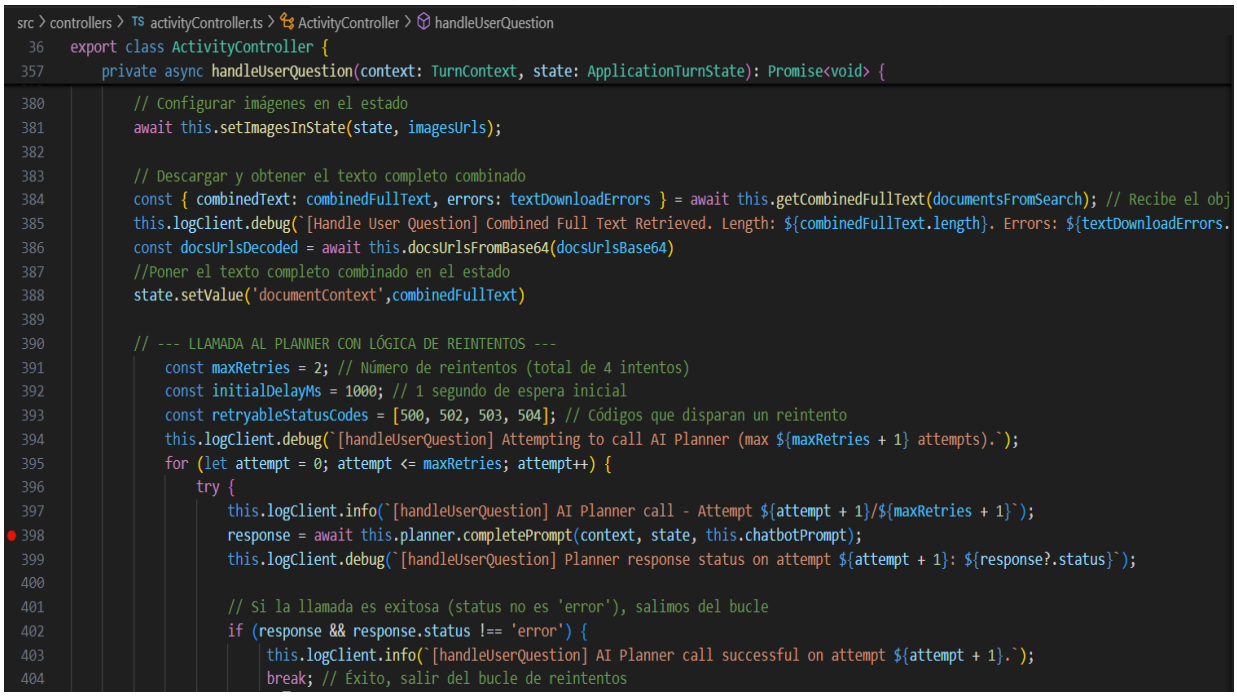


Figura 65. (b) Fases de Generación: Preparación del contexto y llamada al LLM

Esta secuencia de llamadas asíncronas encadenadas es la materialización de la arquitectura RAG en el código del sistema. Es importante destacar la robustez de la implementación en la fase de Generation: para garantizar la resiliencia del sistema ante fallos transitorios de la API de OpenAI (ej. errores 5xx), se ha implementado una lógica de reintentos. Si una llamada al modelo falla con un código de error recuperable, el sistema espera un tiempo creciente antes de reintentar, aumentando así las probabilidades de éxito sin sobrecargar el servicio. Hay que tener en cuenta que los modelos utilizados se disponibilizan con una capacidad de tokens por minuto que se puede alcanzar si hay simultáneamente muchos usuarios concurrentes.

8.4.4 Componentes de IA Auxiliares

Una característica avanzada de esta implementación es el uso de llamadas al LLM no solo para generar la respuesta final, sino también para asistir en los pasos intermedios del proceso:

- **SearchQueryGenerator:** En lugar de pasar la pregunta del usuario directamente a Azure AI Search para la búsqueda de las páginas relevantes, este módulo la pre-procesa. Utilizando el historial de la conversación, invoca al modelo de OpenAI para reformular la pregunta en una consulta de búsqueda más efectiva, así como para tener en cuenta el histórico de la conversación. Esto mejora significativamente la precisión de la fase de *Retrieval*.

Estrategia del Prompt de Reescritura. El prompt para esta tarea está diseñado para generar múltiples variantes de la consulta, aumentando así la probabilidad de encontrar documentos relevantes en l búsqueda sobre AI Search. Le pide al modelo que:

1. Analice la pregunta en el contexto del historial de la conversación.
2. Genere una lista de 3 consultas de búsqueda alternativas.
3. Devuelva el resultado en un formato JSON estructurado para un procesamiento automático sencillo.

src > prompts > searchquery > skprompt.txt

```
1  Below is a history of the conversation so far, and a new question asked by the user that needs to be answered by searching
2  You have access to Azure AI Search index with 100's of documents.
3  Generate 2/3 search query based on the conversation and the new question.
4  If the new input by the user is more than one question generate as many search queries as questions.
5  Do not include cited source filenames and document names e.g info.txt or doc.pdf in the search query terms.
6  Do not include any text inside [] or <<>> in the search query terms.
7  Do not include any special characters like '+'.
```

Figura 66. Prompt de reescritura de Query

- **CitationsGenerator:** La fiabilidad es uno de los aspectos clave. Tras recibir la respuesta del modelo, no siempre se puede garantizar que las referencias a los documentos fuente estén en un formato utilizable o por el contrario ha podido ser una alucinación del LLM. El *CitationsGenerator* resuelve este problema: toma la respuesta generada y la lista de documentos, y le pide al modelo, con un *prompt* que exige una salida JSON, que identifique explícitamente qué fragmentos de la respuesta corresponden a cada documento. En el caso de que no haya ningún fragmento que haya sido la base para generar la respuesta entonces se envía como si no se hubiese encontrado nada.

Estrategia del Prompt de Verificación de Citas. Este es un prompt crítico para la trazabilidad y está diseñado para ser muy estricto. Sus instrucciones clave son:

1. Asignar al modelo el rol de un "auditor de verificación de hechos" (fact-checking auditor).
2. Exigir que la salida sea obligatoriamente un objeto JSON. Esto permite un parseo fiable de la respuesta.
3. Instruir al modelo para que analice la respuesta generada y la compare con los documentos fuente, identificando el origen exacto (document_name, page_number) de cada afirmación.

Este diseño se alinea con la implementación mostrada anteriormente. El prompt es el siguiente

src > prompts > citations > skprompt.txt

```
You are a virtual assistant that helps to detect references given a text and a list of URLs provided by the user.
The references in the text are usually given as [Document][PageNumber of the document].
It is also possible that the reference is indicated as: "Reference: [Doc][PageNumber]"
In the document reference, the full name of the document may not be present, and it may be an identifier with letters
The URLs also have the full name of the document (including the identifier) and its page.
I want your response to be solely a JSON that is a list of numbers: {"Citations":[n1,n2,n3...]}.
I need each number to be the position that the used URL occupies in the list of URLs that I have provided.
Keep in mind that in a list, the first position corresponds to 0.
If you do not find the exact page in the URLs but there is a page from the same document, use that reference.
Do not repeat references.
```

Figura 67. Prompt para la generación de citas

El fragmento de *citationsGenerator.ts* se ha escogido para mostrar cómo se interactúa directamente con la API de Azure OpenAI (usando axios) para una tarea muy específica. Es un buen ejemplo de cómo se puede llamar de forma diferente, de manera más controlada y con requisitos de formato de salida estrictos (en este caso de tipo JSON), una capacidad clave para esta funcionalidad.

src > domain > models > citationsGenerator.ts > CitationsGeneratorBuilder > constructor

```
export class CitationsGenerator {

    public async generateCitations(responseWithoutCites: string, imagesUrls: string[]): Promise<string> {
        const queryMessages = await this.buildMessages(responseWithoutCites, imagesUrls);
        try {
            const response = await axios.post<OpenAIResponse>(
                this.apiUrl,
                {
                    messages: queryMessages,
                    max_tokens: this.tokenLimit,
                    temperature: 0.0,
                    n: 1,
                    response_format: { type: 'json_object' } // Añadir el formato de respuesta aquí
                },
                {
                    headers: {
                        'Content-Type': 'application/json',
                        'api-key': this.openaiApiKey
                    }
                }
            );

            if (response.data.choices.length > 0) {
                this.logClient.info(`[Citations Generator]Citations extraction: ${JSON.stringify(response.data.choices[0].message.content)}`);
                return response.data.choices[0].message.content.trim();
            } else {
                throw new Error('No search query generated');
            }
        } catch (error) {···
        }
```

Figura 68. Llamada a AzureOpenAI para la generación estructurada de citas

8.4.5 Entorno de Pruebas Local: El Teams Toolkit

Una de las complejidades del desarrollo de aplicaciones para Microsoft Teams es que, por su naturaleza, están diseñadas para ejecutarse en la nube y comunicarse con los servicios de Microsoft. Para superar este desafío y permitir un ciclo de desarrollo y depuración rápido y eficiente, se ha utilizado el Teams Toolkit, renombrada recientemente como una extensión para Visual Studio Code [60].

Esta extensión oficial de Microsoft proporciona un entorno de desarrollo local que actúa como un *sandbox*, simulando el comportamiento de la plataforma de Teams sin necesidad de desplegar continuamente los cambios en Azure.

Este entorno se fundamenta en dos componentes clave:

1. **Túnel de Desarrollo (*dev tunnel*):** El toolkit crea un túnel seguro desde una URL pública hasta el servidor Node.js que se ejecuta en la máquina local (*localhost*).
2. **Depuración Integrada:** Al ejecutar el proyecto a través del toolkit, el depurador de Visual Studio Code se adjunta automáticamente al proceso de Node.js.

El código está preparado para diferenciar entre el entorno local y el de producción, como se observa en la configuración del *TeamsAdapter*, que se ajusta para funcionar sin necesidad de credenciales de aplicación reales durante las pruebas locales.

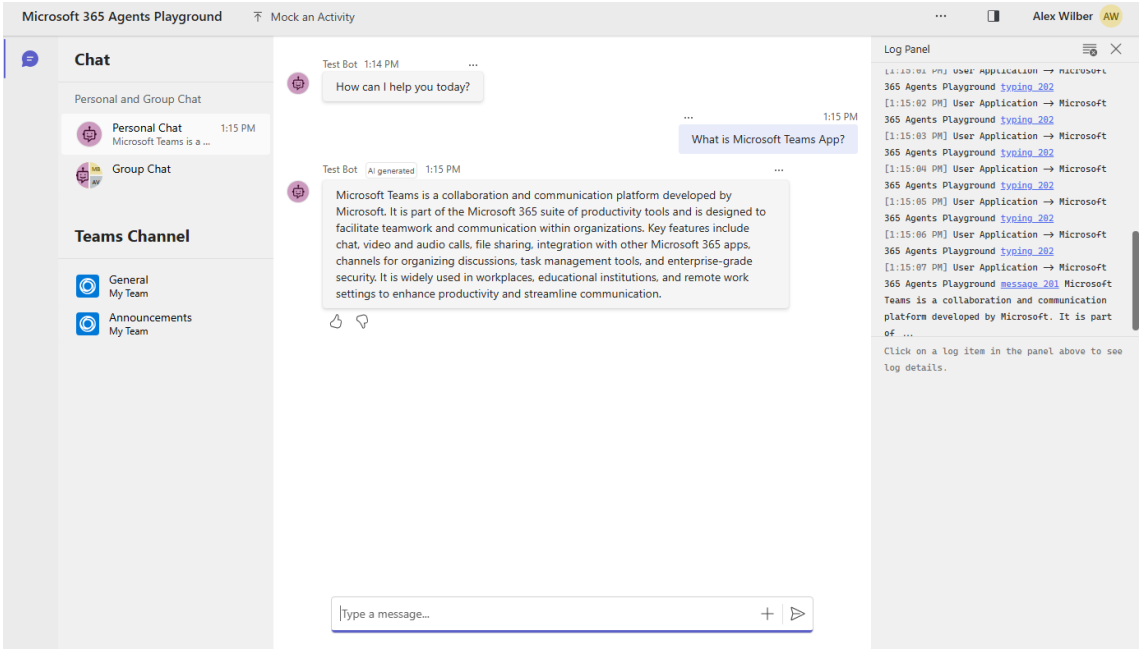


Figura 69. Sanbox de Teams Toolkit

Este entorno de pruebas ha sido fundamental para iterar rápidamente sobre la lógica del bot y el diseño de las Adaptive Cards, reduciendo drásticamente el tiempo entre la codificación de un cambio y su validación a nivel funcional, sin tener que hacer el despliegue definitivo al App Service hasta su productivización final.

8.5 Implementación de la Infraestructura de Datos

El núcleo conversacional necesita una infraestructura de datos robusta y bien diseñada que le proporcione el conocimiento necesario para responder, la persistencia para recordar y la seguridad para operar. Esta infraestructura se ha construido íntegramente

sobre servicios PaaS de Microsoft Azure, lo que garantiza escalabilidad, mantenibilidad y una integración nativa con el resto de la solución.

8.5.1 Base de Conocimiento: Azure AI Search

Como se ha comentado en numerosas ocasiones anteriormente, Azure AI Search es la pieza central de la fase de *Retrieval* en el patrón RAG. Actúa como la base de conocimiento indexada y consultable por DocuBot, permitiendo búsquedas rápidas y relevantes sobre un gran volumen de documentos.

La implementación aprovecha una de las capacidades más potentes de este servicio: la búsqueda híbrida, como se ha explicado en detalle en capítulos anteriores.

Para maximizar la relevancia, la búsqueda se realiza simultáneamente sobre dos índices diferentes:

1. Un índice optimizado para resúmenes y preguntas frecuentes de los documentos. Contiene campos con resúmenes concisos de cada página de documento y posibles preguntas que podrían responder. Su objetivo es capturar la esencia semántica del contenido para búsquedas vectoriales muy precisas.
2. Un índice que contiene el texto completo de cada página. Almacena el texto completo de cada página. Es fundamental para la búsqueda por palabras clave tradicional y actúa como complemento a la búsqueda vectorial.

El código en *azureAISearchDataSource.ts* fusiona los resultados de ambas búsquedas y de ambos índices mediante un sistema de puntuación y ranking personalizado. A cada resultado se le asignan puntos en función de su posición en la lista de resultados de cada índice. Luego, se suman estos puntos para obtener un totalPoints que determina el ranking final, asegurando que los documentos más relevantes, independientemente de cómo fueron encontrados, se presenten al modelo de lenguaje.

El fragmento de código a continuación muestra todo lo comentado anteriormente para el primero de los dos índices, el segundo de ellos es análogo.

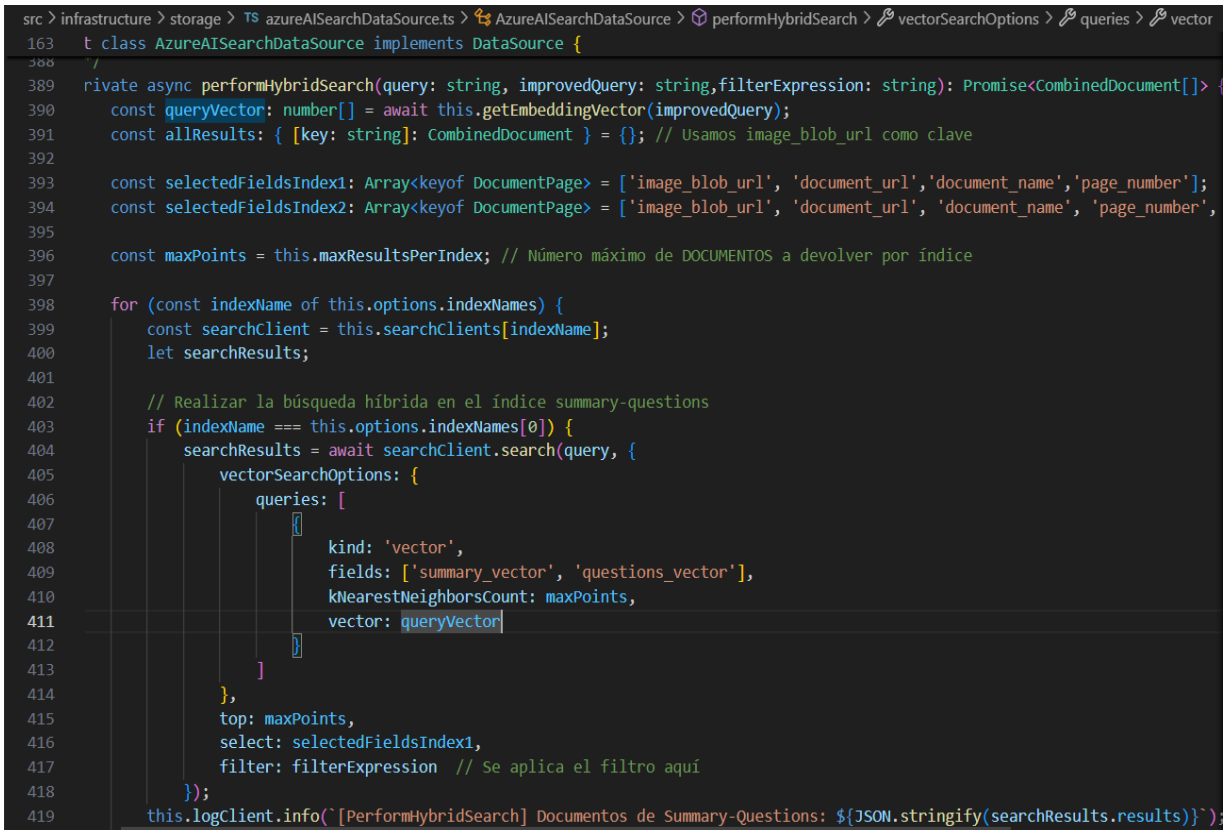


Figura 70. Implementación de la búsqueda híbrida y vectorial

8.5.2 Almacenamiento de Conversaciones y Turnos: Azure Cosmos DB

Azure Cosmos DB es la capa de persistencia de datos, los motivos de su elección han sido ampliamente explicados durante capítulos anteriores. Su naturaleza es ideal para almacenar los objetos JSON complejos que representan el estado de la conversación y los registros de feedback.

En el proyecto, Cosmos DB cumple dos roles críticos:

1. **Almacenamiento del Estado de la Conversación (*cosmos-db-container-conversations*):** Se utiliza la clase *CosmosDbPartitionedStorage*, proporcionada ya por el SDK de Bot Framework, para guardar automáticamente el estado de cada conversación. Esto incluye datos como el historial de chat, los filtros de sistema seleccionados por el usuario. Al persistir este histórico, permite a DocuBot tener memoria.
2. **Registro de Turnos (*cosmos-db-container-turns*):** De forma independiente, cada vez que el bot genera una respuesta, el *ActivityController* guarda un registro detallado de cada "turno" en una colección separada de Cosmos DB. Este registro, identificado por un turnId único, contiene la pregunta del usuario, la respuesta completa del modelo, el uso de tokens de IA, las fuentes citadas y el feedback posterior del usuario. Esta información es clave para el análisis de uso,

la depuración, la evaluación de la calidad del modelo y futuros reentrenamientos.

Esta separación garantiza dos objetivos clave: por un lado, la eficiencia en el mantenimiento del estado de la conversación; por otro, la creación de un registro de auditoría inmutable y completo, fundamental para análisis de negocio, evaluación de la calidad y cumplimiento normativo.

El método *saveTurn* en ActivityController es el responsable de construir este objeto de auditoría. El siguiente código ilustra perfectamente la estructura de datos que se persiste para cada turno. Es el punto de partida para cualquier análisis futuro sobre la efectividad, el uso y los costes.

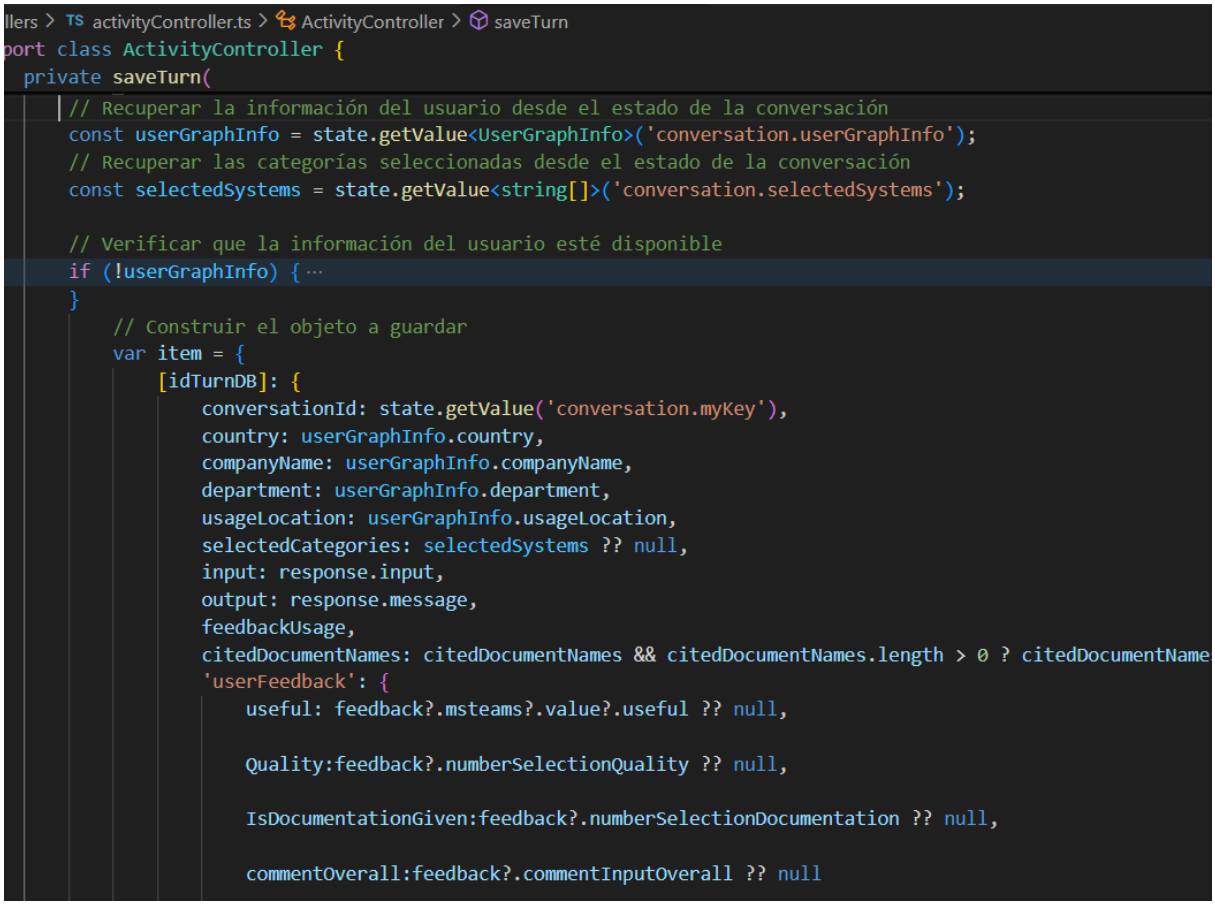


Figura 71. Guardar cada turno de conversación

8.5.3 Gestión Centralizada de Secretos: Azure Key Vault

Uno de los requerimientos con los que se ha basado la aplicación es intentar llevar una seguridad tipo empresarial y en lugar de tener claves de API, cadenas de conexión y otros secretos por el código o en ficheros de configuración, se ha implementado una estrategia de gestión de secretos centralizada utilizando Azure Key Vault.

La clase *SecretManager* es el único componente de toda la aplicación con la responsabilidad de comunicarse con Key Vault. Su implementación se basa en *DefaultAzureCredential* de la librería *@azure/identity*. Esta es la mejor práctica de seguridad en Azure, ya que permite una autenticación sin contraseñas cuando la aplicación se ejecuta en Azure App Service, *DefaultAzureCredential* utiliza automáticamente la Identidad Administrada (Managed Identity) del servicio para autenticarse en Key Vault, eliminando por completo la necesidad de gestionar claves o secretos para la propia aplicación.

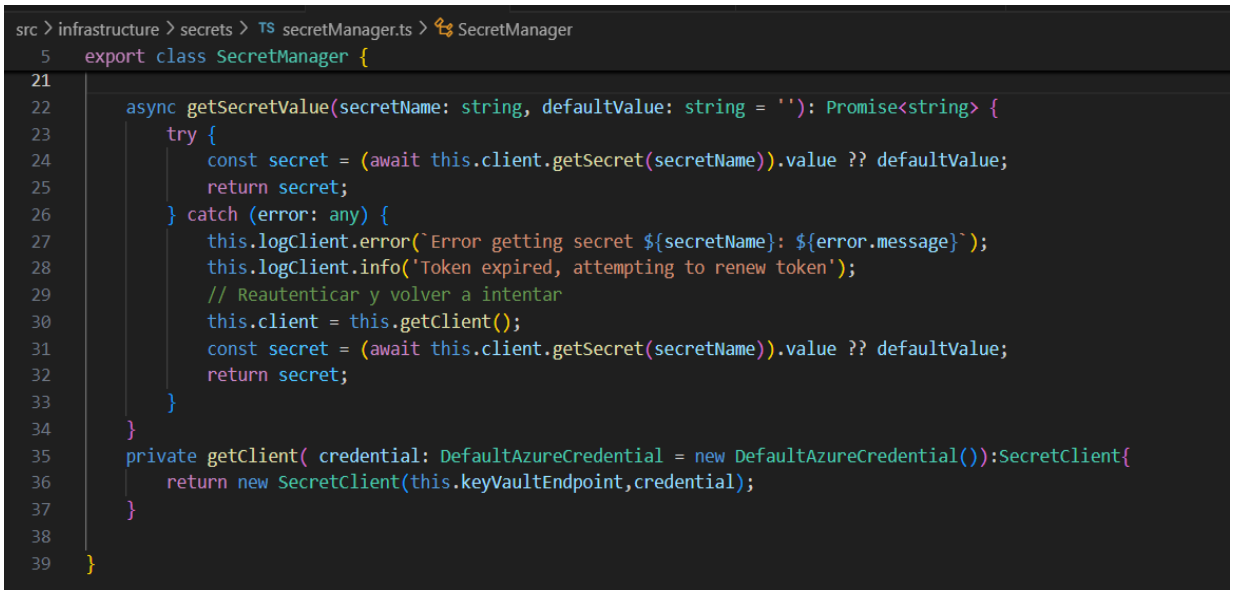


Figura 72. Implementación del cliente Azure Key Vault

8.6 Implementación de Interfaz de Usuario en Teams

A diferencia de una aplicación web convencional que se renderiza en un navegador, un bot de Microsoft Teams presenta su interfaz de usuario directamente dentro del cliente de Teams. Esta interfaz no se construye con HTML y CSS, sino con un conjunto de componentes nativos y, principalmente, con Adaptive Cards, un framework de Microsoft.

La implementación de la interfaz se ha enfocado en ser nativa, interactiva y coherente con la experiencia de usuario de Microsoft Teams.

8.6.1 Componentes de la Interfaz con Adaptive Cards

Las Adaptive Cards son el formato basado en JSON permite definir UIs que se adaptan automáticamente al tema (claro/oscuro), al factor de forma (escritorio/móvil) y a las directrices de diseño de Microsoft Teams. El proyecto separa claramente la definición

de estas tarjetas en el directorio ***src/presentation/***, siguiendo el principio de separación de funcionalidades.

Los componentes principales de la interfaz son:

- **systemSelectionCard.ts:** Actúa como el "formulario" inicial de la aplicación. Se presenta al usuario al comenzar una conversación y le permite filtrar el ámbito de la búsqueda mediante un conjunto de casillas de verificación (*Input.ChoiceSet*). Es clave para acotar el contexto y mejorar la fiabilidad de las respuestas.
- **responseCard.ts:** Es el componente de respuesta principal y el más complejo. Su diseño está pensado para presentar la información de forma clara y útil:
    - Muestra la respuesta generada por el LLM.
    - Incluye una sección de "Referencias" con botones interactivos (*Action.ShowCard*). Cada botón corresponde a un documento fuente citado. Al pulsar uno, se despliega una sub-tarjeta que muestra el nombre del documento, las páginas referenciadas y un enlace directo (*Action.OpenUrl*) al documento original.
    - Contiene acciones de feedback rápido (pulgar arriba/abajo) y un botón para iniciar una nueva conversación.
- **commentCard.ts:** Es un "formulario de feedback" que se presenta cuando un usuario indica que una respuesta no fue útil. Permite al usuario proporcionar una calificación numérica y comentarios de texto, enriqueciendo los datos de auditoría y futura mejora del aplicativo

A continuación con la siguiente figura muestro un extracto de la definición de la *responseCard*. Este código ha sido elegido porque muestra la creación de la sección de referencias, donde se generan dinámicamente los botones *Action.ShowCard*.

Figura 73. Generación de la Adaptve Card.

8.7 Despliegue en la Nube

Una vez implementados los componentes de software, el siguiente paso es desplegarlos en la infraestructura nube de Microsoft Azure. Mientras que los servicios de datos y IA (Cosmos DB, Azure OpenAI, AI Search, etc.) se han aprovisionado directamente en el portal de Azure, el servicio del bot, que contiene la lógica de negocio, requiere un proceso de empaquetado y despliegue específico.

Este procedimiento manual es robusto y adecuado para los despliegues iniciales del proyecto como en el de este TFG. La evolución natural de este proceso en un entorno de producción empresarial maduro sería su automatización completa mediante un pipeline de Integración y Despliegue Continuo (CI/CD) con herramientas como GitHub Actions o Azure DevOps.

8.7.1 Despliegue del Bot en Azure App Service

El proceso de despliegue manual consta de varios pasos, para simplificar su ejecución se ejecutan estos comandos directamente desde la consola de comandos de Visual Studio.

1. **Limpieza del Entorno (*npm run clean*):** El primer paso elimina cualquier artefacto de compilaciones anteriores para asegurar un empaquetado limpio.
2. **Compilación de la Aplicación (*npm run build*):** Se utiliza empaquetar todo el código TypeScript de la aplicación en un único fichero JavaScript. Durante este paso, se instalan todas las dependencias, incluidas las de desarrollo como Webpack.
3. **Limpieza de Módulos (*npm run clean_modules*):** Una vez compilada la aplicación, las dependencias de desarrollo ya no son necesarias. Este paso elimina el directorio node_modules para preparar el empaquetado final solo con las dependencias de producción.
4. **Empaquetado para Producción (*npm run package*):** Este es el paso final de la preparación. Primero se instalan únicamente las dependencias de producción. A continuación, todos los ficheros necesarios para ejecutar la aplicación en el servidor se comprimen en un único fichero. Azure App Service está optimizado para ejecutarse directamente desde un fichero *app.zip*, lo que acelera significativamente el despliegue y el tiempo de arranque.
5. **Despliegue en Azure (*az webapp deploy*):** Con el fichero app.zip listo y asumiendo que se ha realizado **az login** para saber el usuario en Azure y el grupo de recursos para el entorno de desarrollo, el despliegue se realiza con un único comando. Este comando sube el paquete *app.zip* al App Service especificado y lo pone en funcionamiento.

8.7.2 Configuración Final y Conexión con Microsoft Teams

Una vez que el App Service está en ejecución en Azure, se deben realizar dos pasos finales de configuración para que el bot este en correcto funcionamiento en Microsoft Teams:

1. **Registro del Bot (Azure Bot):** En el portal de Azure, se crea un recurso de tipo **Azure Bot**. Este recurso actúa como el registro de identidad del bot en el Bot Framework. En su configuración, se establece el Messaging endpoint para que apunte a la URL del Azure App Service desplegado (ej: *https://<nombre-del-app-service>.azurewebsites.net/api/messages*). También se configura el canal de "Microsoft Teams" dentro de Azure Portal.
2. **Variables Entorno en App Service:** El código de la aplicación depende de una serie de variables de entorno para funcionar. Mientras que en el desarrollo local estas se gestionan en un fichero .env, en producción se deben configurar de forma segura en la sección de Configuración > Variables de Entorno del App Service. Esta estrategia no consiste en eliminar toda la configuración del servicio, sino en gestionarla de forma segura y estructurada, diferenciando entre parámetros de comportamiento y el acceso a secretos.

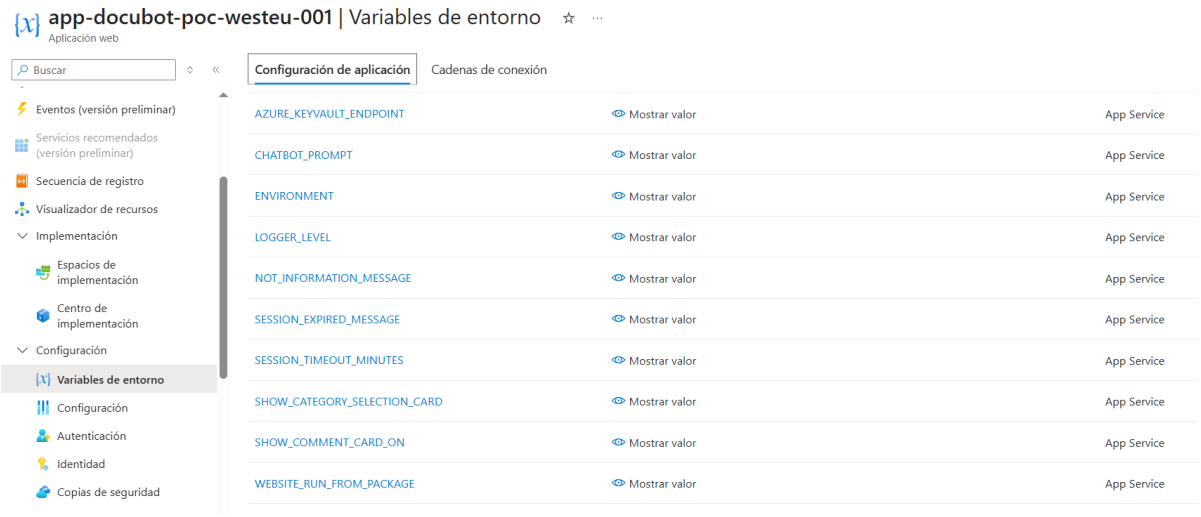


Figura 74. Variables de Entorno del App Service

3. **Manifiesto de la Aplicación (*manifest.json*):** Para que los usuarios puedan interactuar con el bot en Teams, se debe crear un fichero. Este fichero JSON describe la aplicación (nombre, descripción, iconos) que informa a Teams sobre sus capacidades. En este caso, define un bot y una extensión de mensajes para el comando "new" para nuevos mensajes. Este manifiesto se empaqueta en un fichero ZIP junto con los iconos y se puede cargar directamente en el administrador de Teams.

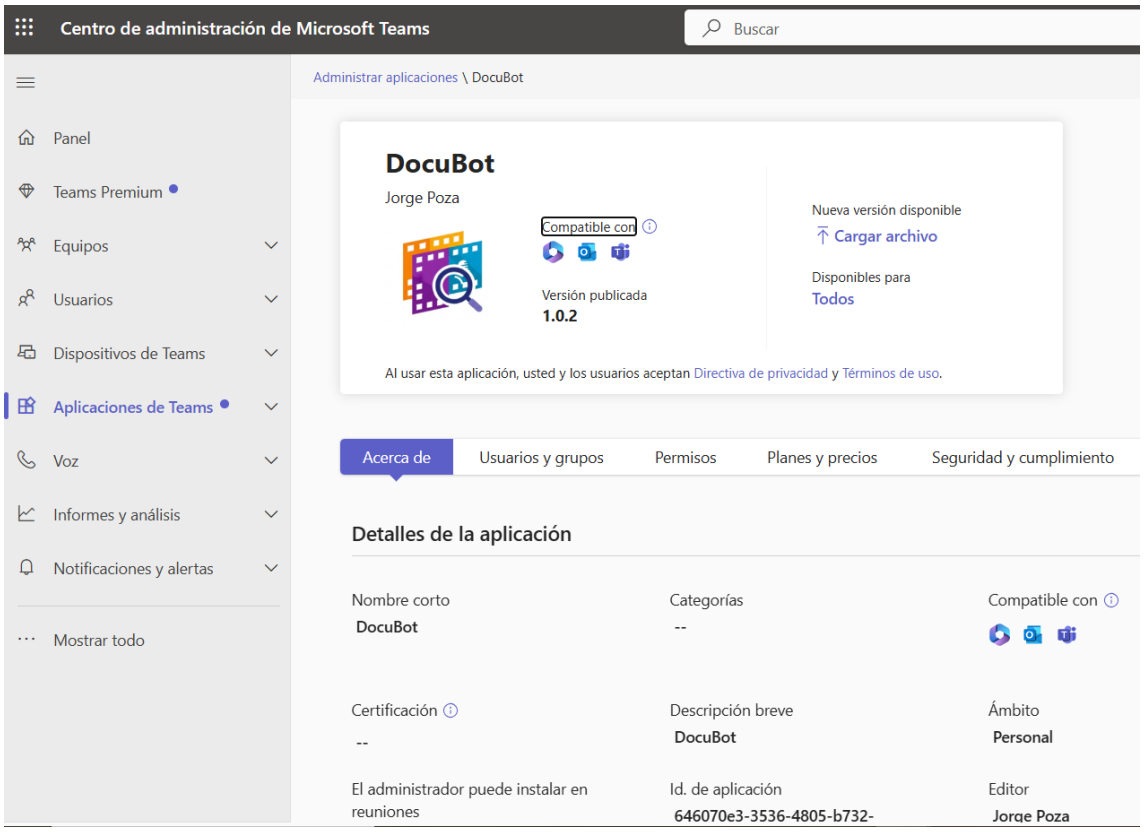


Figura 75. DocuBot en panel de Administración de Teams

8.8 Pruebas Realizadas y Evaluación de Resultados

En esta última sección del capítulo de implementación, este apartado se centra en su validación y en la evaluación de los resultados obtenidos para verificar el correcto comportamiento de la aplicación. Se ha optado por un enfoque de **pruebas de caja negra**, un método de validación de software en el que las funcionalidades de la aplicación se prueban sin conocimiento de la estructura del código interno, los detalles de implementación o los flujos de datos internos.

El objetivo de las pruebas de caja negra es validar que el sistema produce las respuestas o salidas esperadas (un mensaje, una Adaptive Card, una acción correcta) al recibir unas determinadas entradas (un comando del usuario, un clic en un botón, una pregunta específica).

PCN-01	**Inicio de conversación y selección de filtros de sistema**
Objetivo	Comprobar que al iniciar una nueva conversación, el sistema presenta correctamente la tarjeta de selección de sistemas y procesa la elección del usuario, almacenándola para futuras consultas
Prerrequisitos	El usuario ha instalado la aplicación del bot en Microsoft Teams.
Entrada	1. El usuario envía el mensaje new a DocuBot 2. El sistema presenta la Adaptive Card "Please select the system(s) you want to search in” 3. El usuario marca unas casillas (una o varias) 4. El usuario hace clic en el botón "Confirm Selection".
Salida Esperada	1. DocuBot responde con el mensaje de bienvenida. 2. Se muestra la Adaptive Card de selección de sistemas. 3. Tras la selección, el bot responde con un mensaje de confirmación, ej: "Okay, I will focus ..." 4. La tarjeta de selección de sistemas desaparece de la conversación.
Salida Obtenida	1. DocuBot respondió con el mensaje de bienvenida. 2. Se muestra la Adaptive Card de selección de sistemas. 3. Tras la selección, el bot responde con un mensaje de confirmación, ej: "Okay, I will focus ..." 4. La tarjeta de selección de sistemas desaparece de la conversación.

Tabla 18. PCN-01: Inicio de conversación y selección de filtros

PCN-02	**Consulta sin Resultados Relevantes**
Objetivo	Comprobar que el sistema maneja de forma controlada las consultas para las cuales no encuentra información relevante en su base de conocimiento, proporcionando una respuesta predefinida al usuario.
Prerrequisitos	No existen documentos en la base de conocimiento sobre "Mario Bros".
Entrada	El usuario envía la pregunta: ¿Quién es Mario Bros?
Salida Esperada	DocuBot responde con un mensaje de texto predefinido indicando que no pudo encontrar información relacionada con la pregunta en los

	documentos disponibles (ej: "I couldn't find specific information related to your question in the available documents.").
Salida Obtenida	DocuBot respondió con el mensaje: "I couldn't find specific information related to your question in the available documents.".

Tabla 19. PCN-02. Consulta sin Resultados Relevantes

PCN-03	**Flujo de feedback negativo y envío de comentario**
Objetivo	Verificar que el flujo de recopilación de feedback funciona correctamente cuando el usuario indica que una respuesta no fue útil
Prerrequisitos	DocuBot ha proporcionado una respuesta a una pregunta anterior mediante una responseCard.
Entrada	El usuario envía la pregunta: ¿Cómo se prepara una lasaña?
Salida Esperada	1. En la responseCard, el usuario hace clic en el botón de (pulgar abajo). 2. El sistema presenta la commentCard para feedback detallado. 3. El usuario selecciona "5" en "Quality Response", "NO" en "Relevance of Cited Documentation" y escribe "La respuesta no era lo que buscaba" en el campo de comentario. 4. El usuario hace clic en el botón "Submit".
Salida Obtenida	1. Apareció la tarjeta de comentarios detallados. 2. Tras el envío, DocuBot mostró el mensaje de agradecimiento y la tarjeta desapareció. 3. Se verificó en Cosmos DB que los datos del feedback (calidad: 5, documentación: no, comentario: "La respuesta no era lo que buscaba") se guardaron en el documento del turno.

Tabla 20. PCN-03. Flujo de feedback negativo y envío de comentario

PCN-04	**Manejo de sesión expirada**
Objetivo	Comprobar que el sistema detecta correctamente una sesión inactiva que ha superado el tiempo límite configurado (SESSION_TIMEOUT_MINUTES) y fuerza el reinicio de la conversación para garantizar un contexto limpio.
Prerrequisitos	1. El usuario ha tenido una conversación previa con DocuBot. 2. La variable de entorno SESSION_TIMEOUT_MINUTES está configurada a un valor específico (ej: 20 minutos). 3. El usuario no ha interactuado con DocuBot durante un tiempo superior al configurado.
Entrada	1. El usuario, cuya última interacción fue hace más de 20 minutos, envía un nuevo mensaje a DocuBot, por ejemplo: ¿puedes darme más detalles sobre el último punto? 2. El sistema recibe el mensaje y comprueba la marca de tiempo de la última actividad en el estado de la conversación.
Salida Esperada	1. El sistema ignora el contenido del mensaje del usuario (¿puedes darme más detalles...) porque la sesión ha expirado. 2. DocuBot responde con el mensaje predefinido para sesión expirada (ej: el valor de SESSION_EXPIRED_MESSAGE). 3. Se reinicia el estado de la conversación, borrando el historial de chat y los filtros anteriores.

Salida Obtenida	1. El sistema ignoró el contenido del mensaje y no intentó responder a la pregunta. 2. DocuBot respondió con el mensaje de sesión expirada.

Tabla 21. PCN-04. Manejo de sesión expirada

Estos resultados positivos validan de forma empírica la correcta implementación de:

- La orquestación del flujo conversacional en el ActivityController.
- La gestión de estado a través de CosmosDbPartitionedStorage.
- La correcta generación y procesamiento de las Adaptive Cards para la interacción con el usuario.

Evaluación de Rendimiento y Calidad

Más allá de la corrección funcional, es crucial evaluar el rendimiento del sistema y la calidad de sus respuestas para asegurar una experiencia de usuario satisfactoria.

- **Rendimiento (Tiempo de Respuesta):** Se midió el tiempo total transcurrido desde que el usuario envía una pregunta hasta que la *ResponseCard* se renderiza en el cliente de Microsoft Teams. El tiempo de respuesta promedio para una consulta estándar se situó en el rango de 12-15 segundos. Este tiempo es comprensivo, abarcando todo el pipeline RAG: la reescritura de la consulta, la búsqueda híbrida, la recuperación de artefactos y las llamadas secuenciales al LLM. Este rendimiento cumple con el Requisito No Funcional RNF-01, asegurando una interacción fluida que no rompe el ritmo de la conversación.
- **Calidad de las Respuestas de IA:** La evaluación de la calidad de las respuestas generadas es fundamental. Para un análisis algorítmico exhaustivo, existen marcos de trabajo especializados como RAGAS [61], que miden la fidelidad (*faithfulness*) y la relevancia semántica (*answer relevance*) de la respuesta. La integración de evaluación automática con estas herramientas constituiría una valiosa línea de trabajo futura para la optimización continua del sistema.

No obstante, para el alcance de este proyecto, se ha priorizado un método de validación de calidad más orientado al usuario final: el sistema de feedback integrado. La funcionalidad de valoración "útil/no útil" y la CommentCard para comentarios detallados actúan forma de evaluación de la calidad percibida por el usuario. Este mecanismo no solo valida la utilidad de la respuesta en un contexto real, sino que los datos recopilados y almacenados de forma estructurada en Cosmos DB sientan las bases para un futuro ciclo de mejora continua, permitiendo a los administradores del sistema identificar patrones, analizar respuestas mal valoradas y refinar los prompts o la base de conocimiento.

En síntesis, la fase de evaluación ha permitido confirmar que DocuBot es un sistema funcionalmente correcto, robusto y eficiente. Los resultados de las pruebas funcionales demuestran que la arquitectura implementada cumple con todos los requisitos definidos, mientras que el análisis de rendimiento valida que la solución opera dentro de parámetros aceptables para una experiencia de usuario fluida. DocuBot

está, por tanto, técnicamente validado y preparado para una siguiente fase de evaluación centrada en la experiencia y aceptación del usuario final.

Conclusiones

En este último capítulo se describen las posibles líneas de trabajo futuras que podrían elevar esta solución al siguiente nivel. Finalmente, se presentará una serie de conclusiones personales sobre el proceso de desarrollo y el aprendizaje obtenido a lo largo de este proyecto.

9.1 Líneas de trabajo futuras

La arquitectura actual, modular y basada en servicios de Azure, proporciona una base sólida para futuras expansiones. A continuación, se detallan cinco posibles líneas de trabajo que podrían ampliar significativamente las capacidades y la robustez de DocuBot.

9.1.1 Implementación de un Flujo de CI/CD.

Actualmente, el despliegue de la aplicación se realiza de forma manual. Una mejora fundamental sería la implementación de un pipeline de Integración Continua y Despliegue Continuo (CI/CD) utilizando herramientas como *GitHub Actions*. Este pipeline automatizaría el proceso desde la subida del código hasta el despliegue en Azure *App Service*, incluyendo pasos como la compilación, la ejecución de pruebas automatizadas y el empaquetado. La automatización reduciría drásticamente el riesgo de errores humanos, aumentaría la agilidad para lanzar nuevas versiones y garantizaría un proceso de despliegue consistente y repetible.

Se podrían explorar principalmente dos estrategias para activar el despliegue automático a Azure App Service mediante GitHub Actions:

1. **Despliegue en Merge a main:**
    - Cada vez que una rama (por ejemplo, una rama develop o una feature ya revisada y aprobada) se fusiona (merge) con la rama main, se activaría automáticamente el pipeline de CI/CD.
2. **Despliegue al Etiquetar una Release:**
    - **(Etiqueta de Release):** Una vez que el código en main se considera listo para una nueva versión, se crea una etiqueta Git (tag), por ejemplo, v1.2.0. La creación de esta etiqueta activaría el pipeline de CI/CD para desplegar la versión correspondiente.

9.1.2 Expansión a Otras Interfaces de Usuario

Ahora mismo si bien la integración nativa con Microsoft Teams es una de sus grandes fortalezas, la arquitectura del proyecto está diseñada para poder ser agnóstica a la interfaz. El núcleo conversacional y la infraestructura de datos no dependen de Teams. Sería relativamente sencillo desacoplar la capa de presentación y crear nuevos "frontales" para otras plataformas. Por ejemplo, se podría desarrollar una aplicación web independiente utilizando un framework como React o Vue, que consumiría la misma lógica del bot a través de una API. Esto permitiría ofrecer DocuBot a organizaciones que no utilizan Microsoft Teams o que desean integrarlo en sus propios portales internos, ampliando enormemente el alcance potencial.

9.1.3 Automatización del Pre-procesamiento de Documentos

El conocimiento de DocuBot depende de los documentos procesados y almacenados en Azure Blob Storage y AI Search. Actualmente, este proceso de ingesta se realiza de forma manual. Una evolución natural del proyecto sería crear un proceso de ingesta automatizado. Esto podría implementarse mediante una **Azure Function** que se dispare por ejemplo cada vez que se suba un nuevo documento a un SharePoint. Esta función se encargaría de:

1. Descargar los documentos nuevos de SharePoint.
2. Extraer el texto y las imágenes del documento.
3. Dividir el contenido en fragmentos lógicos (por páginas).
4. Generar los *embeddings* (vectores) para cada fragmento.
5. Indexar la información y los vectores en Azure AI Search.

Esta automatización garantizaría que la base de conocimiento de DocuBot se mantenga siempre actualizada de forma eficiente y sin intervención manual.

9.1.4 Utilización Power BI

Esta es una posible mejora presente casi desde el comienzo y que ya se ha ido comentando a lo largo de la memoria. El consumo de los datos de auditoría y feedback se realiza actualmente directamente desde Cosmos DB por el administrador pero debería realizarse desde un entorno de analítica de datos como Power BI donde tener los datos consolidados y poder realizar reportes visuales con las métricas de uso, costes… Así como poder agrupar los feedback de los usuarios para hacer análisis detallado de ello y estudiar futuras mejoras de DocuBot.

9.1.5 Despliegue con Infraestructura como Código

Para gestionar esta compleja infraestructura de red y los propios recursos de forma declarativa y repetible, se podría adoptar un enfoque de Infraestructura como Código utilizando herramientas como **Terraform**. Esto permitiría definir toda la arquitectura de Azure en ficheros de código, facilitando la creación de nuevos entornos (desarrollo, pruebas, producción), la recuperación ante desastres y la auditoría de la configuración.

9.2 Conclusiones Personales

La realización de este Trabajo de Fin de Grado ha sido una experiencia que ha superado con creces el ámbito de un proyecto académico. Ha supuesto una inmersión profunda en el mundo de la Inteligencia Artificial Generativa, un campo donde la línea entre la investigación teórica y la aplicación práctica es extraordinariamente fina, y en el que aspiro a desarrollar mi futuro profesional.

Una de las fases más cruciales y extensas fue, sin duda, la investigación preliminar, cuya profundidad se refleja en el capítulo de Fundamento Teórico. En un dominio tan novedoso, no existía un "manual de instrucciones" claro ni ejemplos a los que acudir; cada decisión de diseño, desde la estrategia de *chunking* hasta la arquitectura de búsqueda híbrida, exigía un riguroso proceso de comparación de *papers* y experimentación. Esta etapa inicial, fue fundamental para asentar las bases del proyecto, enseñándome a valorar la importancia de una base teórica sólida antes de escribir una sola línea de código.

El mayor desafío, sin embargo, fue gestionar la increíble velocidad a la que evoluciona este campo. El panorama de la IA ha ido cambiando radicalmente durante el desarrollo del proyecto: surgieron nuevos modelos, las capacidades multimodales evolucionaron y las técnicas de RAG avanzaron semanalmente. Esta rápida y vertiginosa evolución me obligó a adoptar una mentalidad de aprendizaje continuo.

Este proyecto me ha permitido consolidar y aplicar gran parte de los conocimientos adquiridos durante el grado. A su vez, me ha impulsado a profundizar de forma autodidacta en tecnologías de vanguardia como los LLMs multimodales, arquitecturas RAG y los servicios cloud de Azure.

En definitiva, este TFG es más que un proyecto técnico; es la demostración de que los principios sólidos de la ingeniería son más relevantes que nunca para abarcar la complejidad de la IA y construir soluciones que no solo sean inteligentes, sino también fiables y seguras. Ha sido un reto que me ha exigido ir más allá de lo aprendido hasta ahora, poniendo a prueba mi capacidad de resolución de problemas y que ha despertado en mí una pasión aún mayor por la tecnología y su futuro potencial.

Apéndice A. Manual de Usuario

En este apéndice se detalla el manual de uso de la aplicación para los distintos perfiles de usuario que interactúan con ella. El objetivo es proporcionar una guía clara y concisa para el usuario final sobre cómo utilizar DocuBot y unas indicaciones para el técnico sobre cómo administrar su base de conocimiento.

Las capturas de pantalla son meramente ilustrativas y han sido extraídas de una batería de pruebas, pero servirán como una referencia visual clara del flujo de interacción.

A.1 Manual de Usuario

Bienvenido a DocuBot. Este asistente en Microsoft Teams ha sido diseñado para ser tu experto de consulta sobre la documentación de la empresa. Su misión es proporcionarte respuestas rápidas y precisas, con disponibilidad 24/7, basadas siempre en la documentación oficial, ayudándote a encontrar la información que necesitas sin tener que buscar manualmente en múltiples repositorios.

A.1.1 Instalación de la Aplicación en Teams

Antes de poder realizar tu primera consulta, necesitas agregar el asistente a tu entorno de Microsoft Teams. El proceso es muy sencillo:

1. Abre tu aplicación de escritorio o web de Microsoft Teams.
2. En la barra lateral izquierda, haz clic en el icono de **"Aplicaciones"**.
3. En la tienda de aplicaciones, busca el nombre del asistente (“DocuBot") en la sección **"Creado para su organización"**.
4. Una vez localizado, haz clic sobre él para ver los detalles.
5. Pulsa el botón **"Agregar"**.

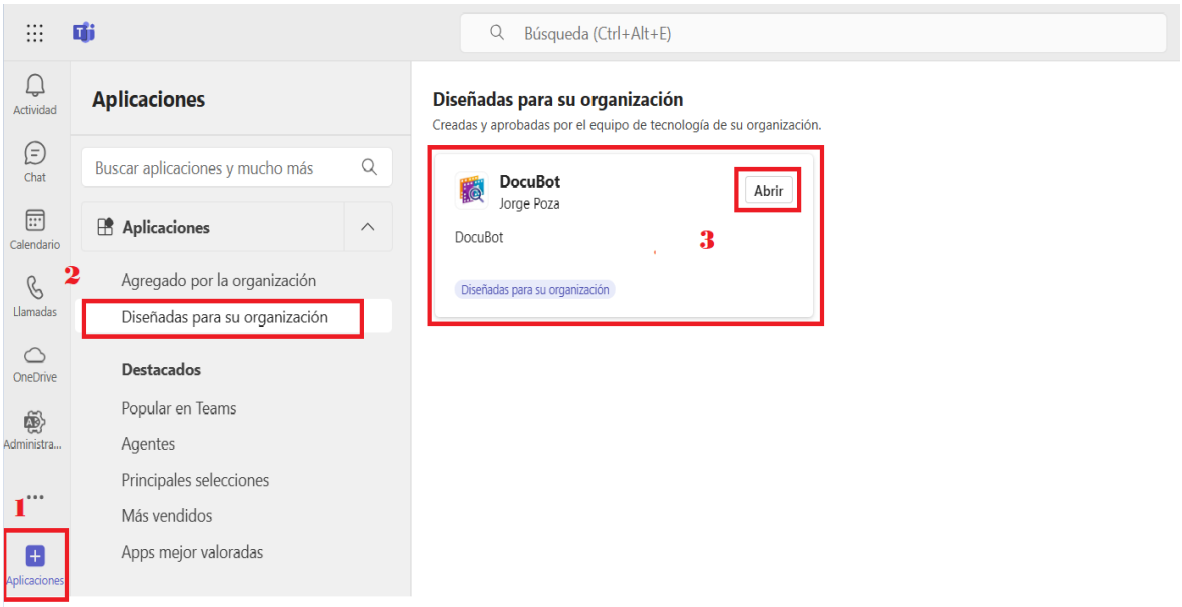


Figura 76. Instalación de DocuBot

Una vez agregado, DocuBot aparecerá en tu lista de chats y te enviará un mensaje de bienvenida, indicando que está listo para ser utilizado. Otras formas de acceder a DocuBot una vez instalado son a través de la barra lateral a través de aplicaciones o a través de la barra de búsquedas de Teams.

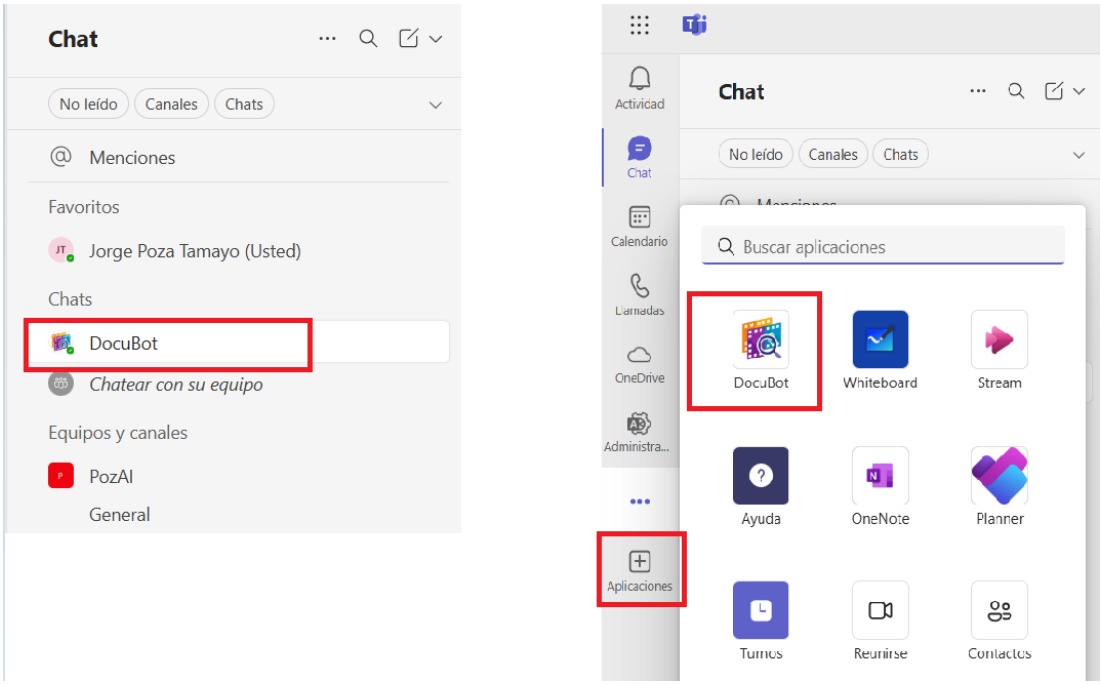


Figura 77. Formas de acceso a DocuBot

A.1.2 Iniciar o Reiniciar una Conversación

Aunque DocuBot te da la bienvenida al instalarlo, en cualquier momento se puede querer empezar una consulta desde cero para asegurarte de que no hay contexto de preguntas anteriores. El asistente ofrece varias formas de hacerlo:

1. **Comando de Chat**: En cualquier momento, puedes escribir new o nuevo en la caja de texto y pulsar Intro. Esta es la forma más directa de reiniciar el flujo.
2. **Botón en la Tarjeta de Respuesta**: Después de recibir una respuesta del asistente, la tarjeta de valoración incluye un icono de un bocadillo de chat (💬). Pulsar este botón tiene el mismo efecto que escribir new, borrando la conversación anterior y preparándose para una nueva consulta. Es la forma más cómoda de encadenar preguntas no relacionadas.
3. **Acción en la Barra de Redacción (*Message Extension*)**: Dependiendo de la configuración de Teams, puede que veas el icono del asistente debajo de la caja donde escribes tus mensajes. Al hacer clic en él, aparecerá una opción como "New". Al seleccionarla, se iniciará una nueva conversación.

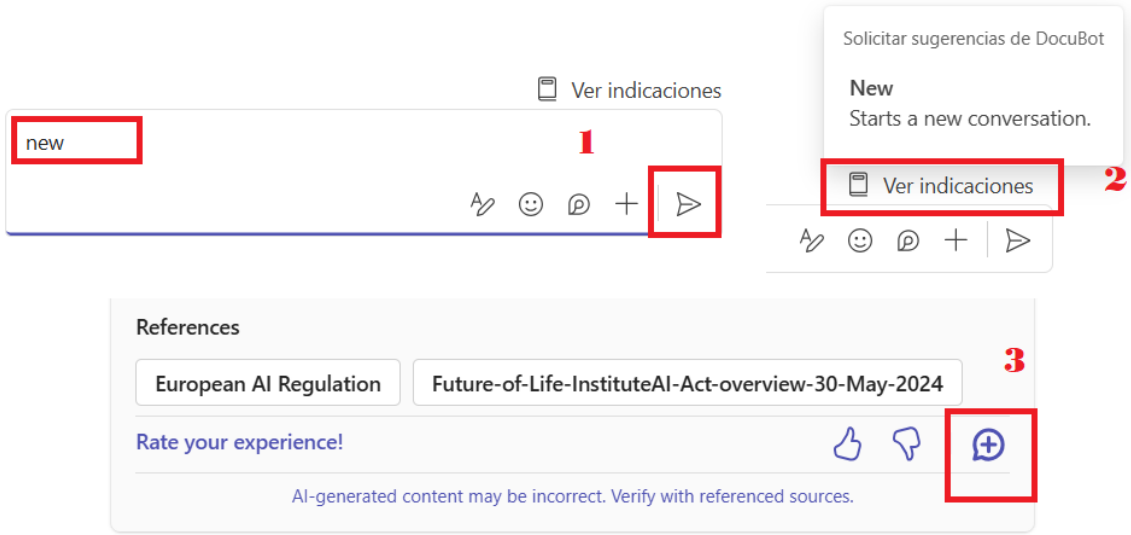


Figura 78. Formas de Nueva Conversación

Independientemente del método que elijas, el resultado será el mismo: DocuBot te presentará el mensaje de bienvenida y la tarjeta para seleccionar el ámbito de tu búsqueda, borrando la conversación previa y asegurando un punto de partida limpio.

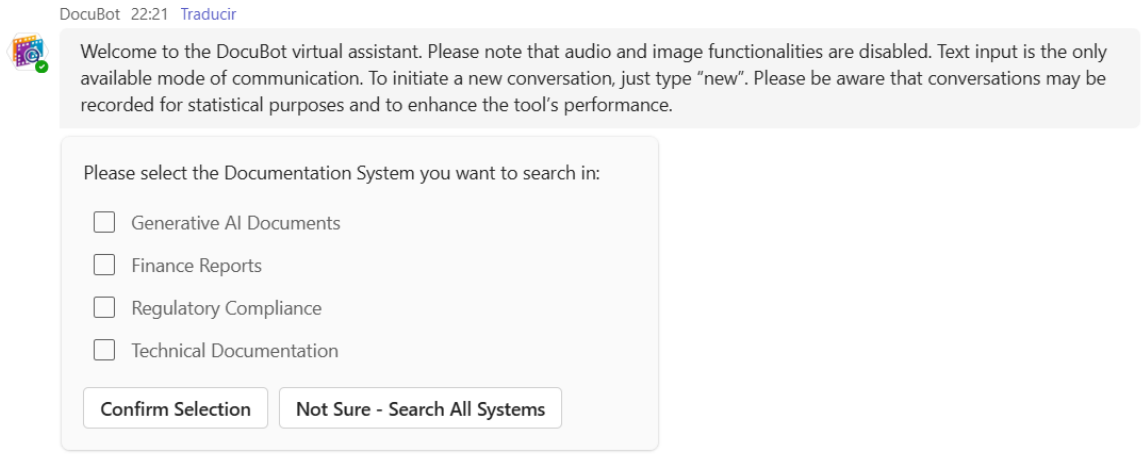


Figura 79. Mensaje Inicial DocuBot

En el caso de que no tuvieras permisos para utilizar DocuBot, se te indicará que no tiene permisos para utilizar dicho asistente virtual y deberá hablar con el administrador del sistema para que le incluya en el grupo de usuarios dentro de Microsoft Entra ID que tienen permisos de uso a la base documental y a DocuBot.

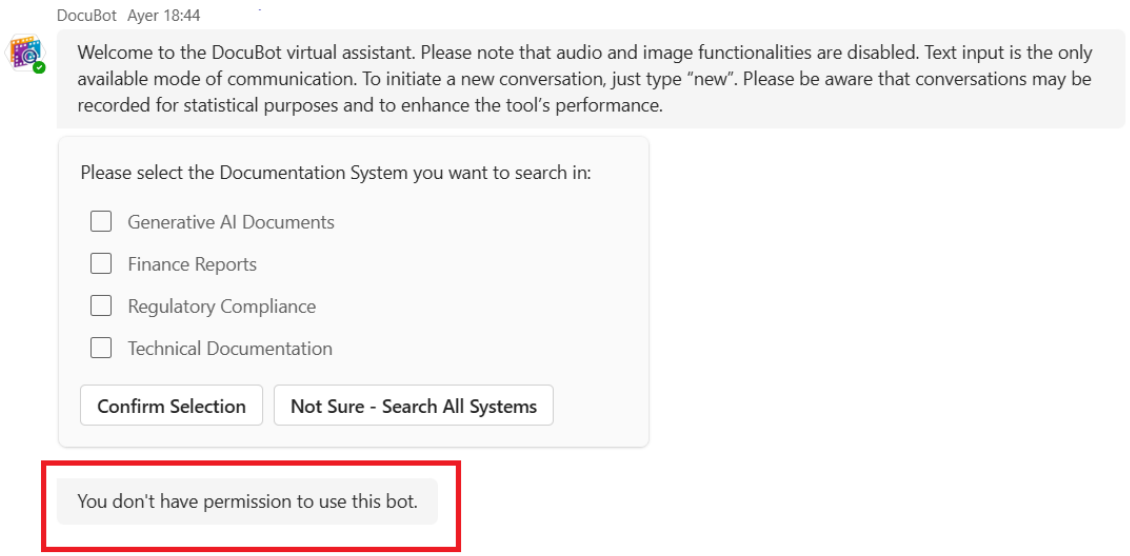


Figura 80. Usuario sin permisos de uso

A.1.3 Selección Sistemas Documentales

Tras iniciar una conversación, el asistente presentará una tarjeta interactiva para que puedas acotar el ámbito de tu consulta. Esto permite que la búsqueda sea más rápida y mejorará la calidad de la respuesta.

- **Selección Múltiple**: Puedes marcar una o varias casillas de los Sistemas de Documentación en los que se organiza la base documental consultada, sobre los que trata tu pregunta.
- **Confirmar Selección**: Una vez seleccionados los sistemas, pulsa el botón **"*Confirm Selection*"**. El bot confirmará tu elección y esperará tu pregunta.
- **Búsqueda General**: Si no estás seguro de a qué sistema pertenece tu duda, pulsa el botón **"*Not Sure - Search All Systems*"**. El asistente buscará en toda la base de conocimiento disponible.

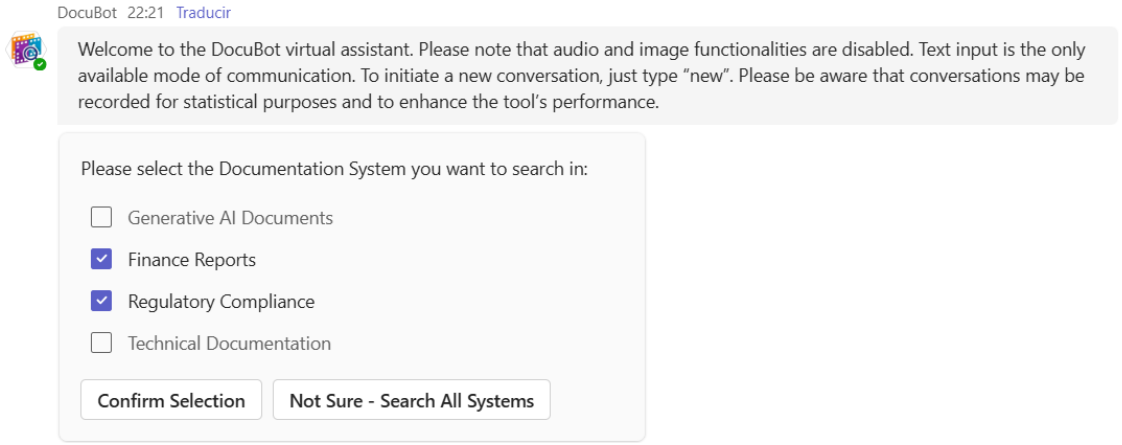


Figura 81. Tarjeta de selección de Grupo Documental

A.1.4 Realización Consultas

Esta sección detalla cómo interactuar eficazmente con DocuBot, desde el inicio de la consulta hasta la interpretación de sus respuestas.

Merece la pena recordar que para asegurar que cada consulta se trate de forma independiente, especialmente si vas a cambiar de tema, es muy recomendable iniciar una nueva conversación. También seleccionar los Grupos Documentales donde se puede responder a su consulta para acotar y mejorar la efectividad de la respuesta.

Escribe tu pregunta de forma clara. Mientras el asistente procesa tu solicitud (lo que tardará varios segundos), verás el indicador de escritura de Teams (tres puntos animados). Esto te confirma que tu pregunta ha sido recibida y se está generando una respuesta.

El asistente te devolverá la respuesta en una tarjeta interactiva. Es fundamental entender sus componentes:

1. **Respuesta Generada por IA**: El texto principal que sintetiza la información encontrada.

2. **Referencias**: Justo debajo, encontrarás los documentos fuente que DocuBot ha utilizado.

¡Atención! Cada vez que utilices DocuBot, las respuestas generadas incluirán referencias a la documentación disponible en la base documental. Esto es crucial porque la respuesta generada por la IA a veces puede ser incompleta o incluso incorrecta. Por lo tanto, **es responsabilidad del usuario consultar la documentación original en Sharepoint a través de los enlaces proporcionados para verificar y contrastar la respuesta generada.**

Al pulsar sobre cada referencia, se mostrará el nombre del documento, las páginas citadas y un enlace directo al archivo original para tu consulta.

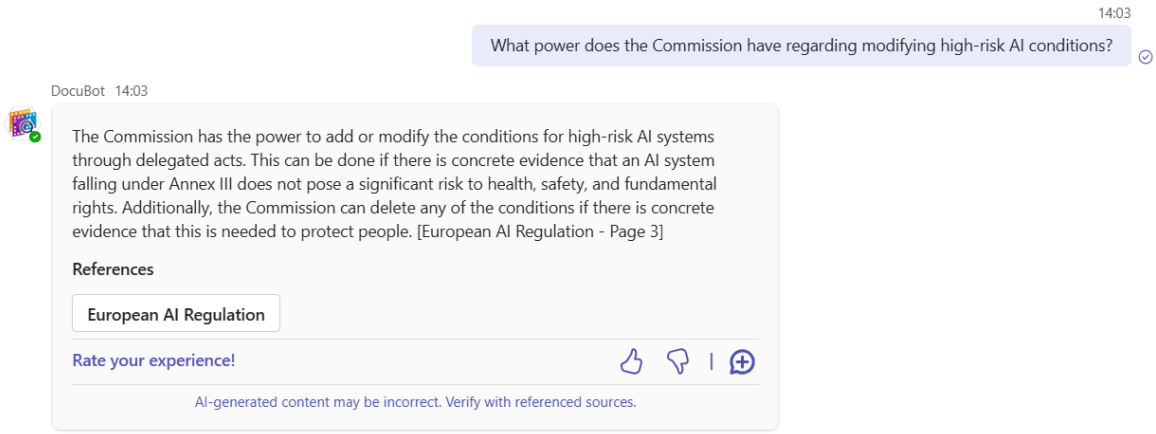


Figura 82. Ejemplo tarjeta respuesta

- **Historial de Conversación**: Durante una sesión activa (antes de reiniciarla), las respuestas generadas tienen en cuenta todo el historial de mensajes, lo que te permite hacer preguntas de seguimiento. Por ejemplo, si preguntas "¿Cuál es el procedimiento X?" y luego "¿Y quién es el responsable?", DocuBot entenderá que la segunda pregunta se refiere al procedimiento X.
- **Límites de Conocimiento**: El asistente solo puede responder preguntas basadas en la documentación que ha sido cargada y procesada. No podrá contestar a preguntas sobre temas que no estén en su base de conocimiento, opiniones, o información en tiempo real de internet. Si una pregunta está fuera de contexto, es probable que indique que no ha encontrado información relevante.
- **Sesión de conversación expirada**: DocuBot tiene un tiempo de sesión de conversación activo desde la última vez que has hablado de manera que si ha pasado demasiado tiempo desde tu última pregunta y lanzas una nueva consulta el histórico se borra y comienzas con una conversación limpia.

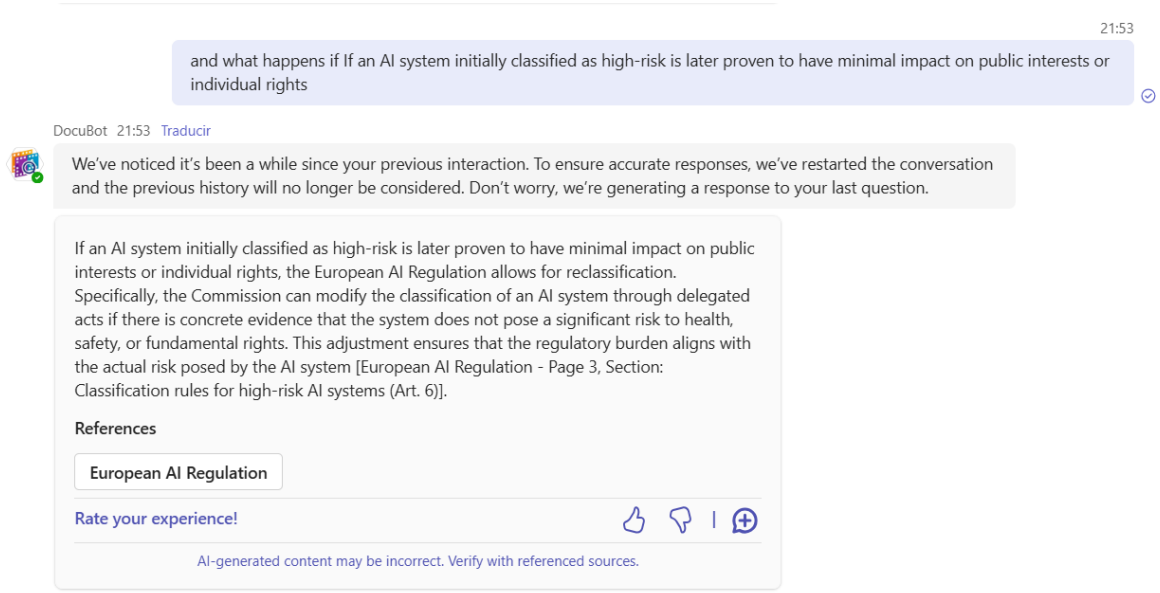


Figura 83. Ejemplo de sesión de conversación expirada

A.1.5 Feedback

Tu opinión es vital para mejorar la calidad de DocuBot. Por ello, cada respuesta viene acompañada de herramientas de valoración.

En la parte inferior de la tarjeta de respuesta, encontrarás una sección para valorar rápidamente tu experiencia:

- **Pulgar Arriba (Útil)**: Si la respuesta fue correcta y te ayudó.
- **Pulgar Abajo (No Útil)**: Si la respuesta fue incorrecta, incompleta o no relevante.

Si valoras una respuesta como "No Útil", el asistente te presentará una segunda tarjeta para que puedas proporcionar más detalles. Esta tarjeta incluye:

- **Calidad de la Respuesta**: Una valoración numérica del 1 al 10.
- **Relevancia de la Documentación**: Una selección de "Sí" o "No" sobre si los documentos citados eran pertinentes.
- **Comentario Adicional**: Un campo de texto opcional para que expliques con más detalle el problema.

Enviar este feedback es de gran ayuda para el equipo de mantenimiento para identificar áreas de mejora.

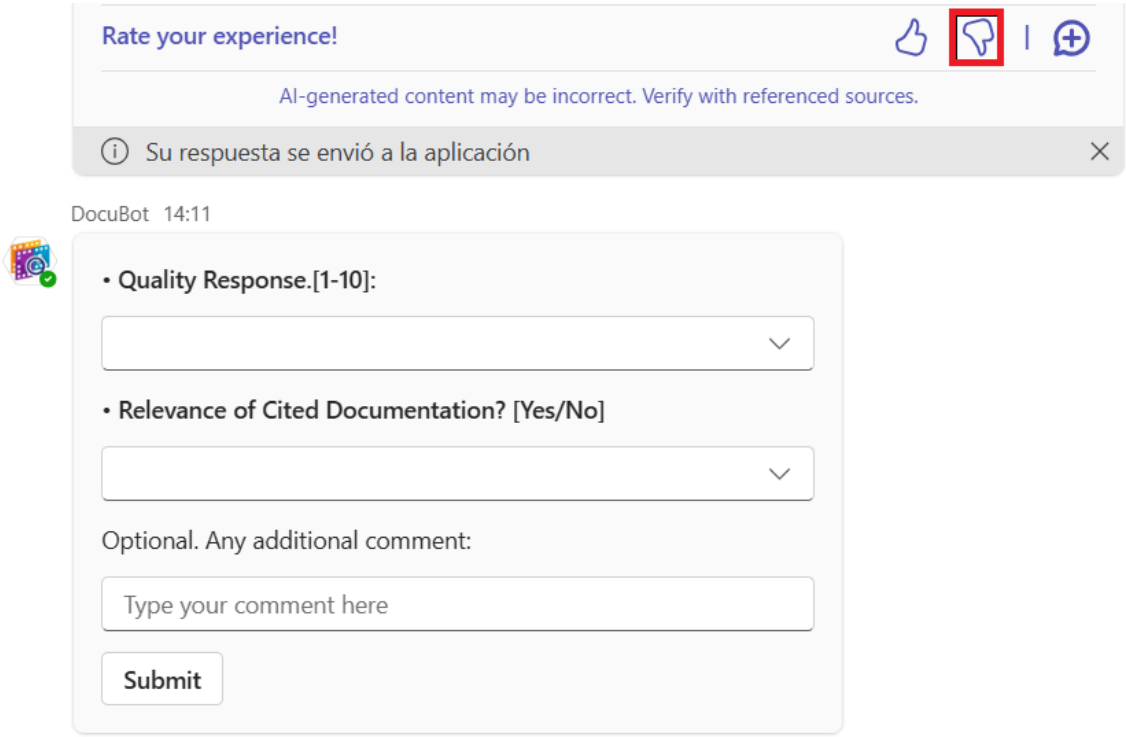


Figura 84. Tarjeta de Feedback

A.2 Manual de Administrador

El rol del Administrador es fundamental para garantizar que DocuBot sea una herramienta fiable, precisa y actualizada. A diferencia del usuario final, la interacción no es con la interfaz de chat, sino con la gestión de la base de conocimiento y la supervisión de la infraestructura tecnológica que soporta todo el stack en Azure.

Este manual detalla responsabilidades clave y los procedimientos asociados.

A.2.1 Gestión de la Base de Conocimiento

La inteligencia y utilidad de DocuBot dependen directamente de la calidad, actualidad y organización de la documentación fuente.

- **Repositorio Central**: Toda la documentación que debe consultar DocuBot (procedimientos, políticas, manuales, etc.) se gestiona exclusivamente desde una biblioteca de documentos designada en SharePoint.
- **Actualización y Versionado**: Es tu responsabilidad asegurar que solo las versiones más recientes y aprobadas de los documentos estén en este repositorio. En la librería Sharepoint existe un campo “IA” que indica si los documentos quieren ser incluidos o no en DocuBot. Los documentos obsoletos o en borrador

deben ser archivados, eliminados o dejar sin marcar para evitar que DocuBot proporcione información desactualizada.

- **Formato de Archivos**: Asegúrate de que los documentos se suban en formatos compatibles con el proceso de indexado (PDF).
- **Metadatos**: Asegúrate de que los metadatos de los documentos en SharePoint (como el "Documentation System") estén correctamente asignados. Estos metadatos son utilizados por DocuBot para filtrar las búsquedas según la selección del usuario.

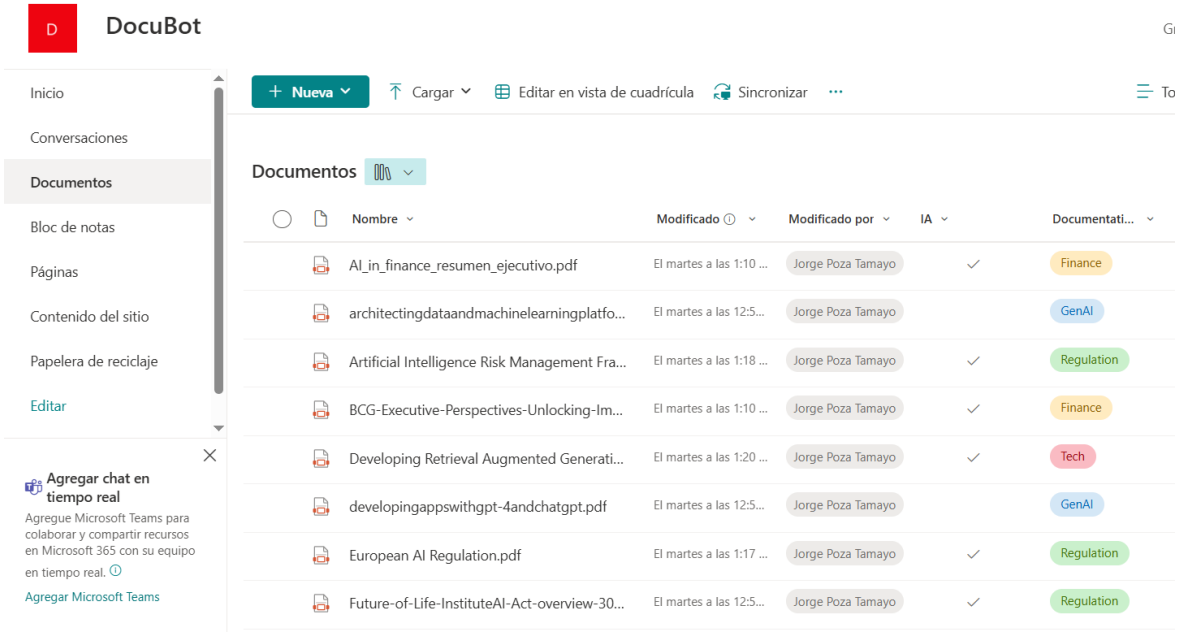


Figura 85. Base Documental en Sharepoint

Por otro lado el "indexado" es el proceso que lee los documentos de SharePoint y los "traduce" a un formato que DocuBot puede entender y consultar. Este proceso es el puente entre la documentación y la IA.

- **Activación del Proceso**: El indexado no es automático en tiempo real. Debe ser ejecutado deliberadamente cada vez que haya cambios significativos en la documentación. Este proceso se lanza a través de la ejecución de un script en Python desde un entorno de conda.

Advertencia: Si se actualiza un documento en SharePoint pero **no se ejecuta el proceso de indexado, DocuBot seguirá respondiendo, basándose en** la versión antigua de la información. La ejecución del pipeline de indexado es un paso crítico tras cada actualización de contenido.

A.2.2 Supervisión de la Infraestructura en Azure

La segunda gran responsabilidad es monitorizar la salud, el rendimiento y los costes de los servicios de Azure que componen la aplicación, serán gestionados desde Azure Portal con la cuenta de administrador del sistema.

- **Azure Key Vault**: El almacén seguro de todos los secretos (claves de API, contraseñas, cadenas de conexión). Se debe gestionar sus políticas de acceso.
- **Azure AI Search**: Monitoriza el estado de los índices y el uso del servicio. Si el volumen de documentos crece exponencialmente, puede ser necesario escalar este recurso.
- **Azure Cosmos DB**: Este servicio almacena dos tipos de datos críticos:
    - **Estado de la Conversación**: Permite al bot recordar el historial de chat en una sesión.
    - **Logs de Turnos**: Cada interacción (pregunta, respuesta, feedback, uso de tokens) se guarda como un registro detallado. Debes monitorizar el crecimiento y coste de esta base de datos.
- **Azure Blob Storage**: Verifica el estado de los contenedores donde se guardan los blobs con imágenes de las páginas, los resúmenes y los textos extraídos durante el preprocesamiento
- **Azure App Service**: Es el servicio que ejecuta el código del bot. Debes monitorizar su rendimiento.

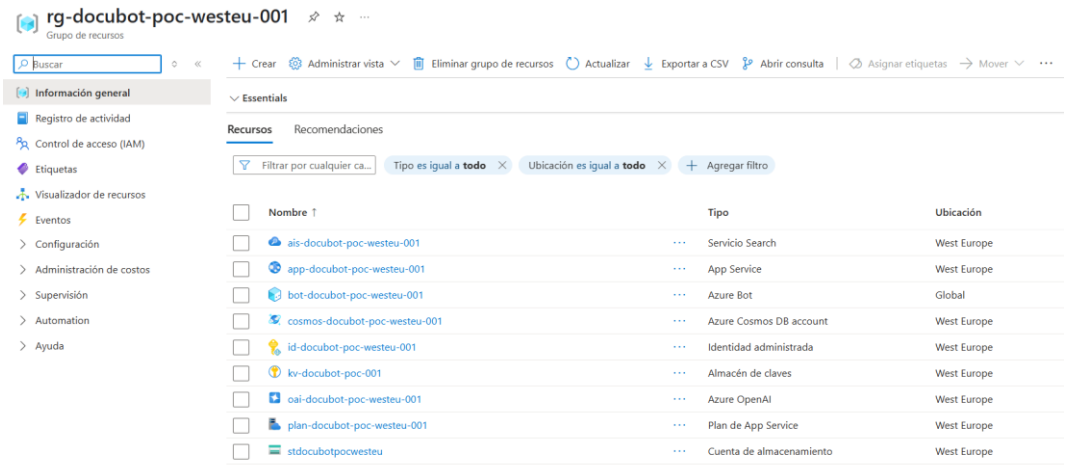


Figura 86. Azure Portal para el Administrador

El acceso a DocuBot está restringido para garantizar que solo los usuarios autorizados puedan utilizarlo. Esta gestión se realiza a través de **Microsoft Entra ID**.

- **Rol de Aplicación**: El acceso se concede mediante la asignación de un rol de aplicación específico a los usuarios.

- **Grupos de Seguridad**: La práctica recomendada es asignar este rol no a usuarios individuales, sino a un grupo en Entra ID.
- **Tu Responsabilidad**: Tu tarea consiste en añadir o eliminar usuarios de este grupo de seguridad a través del portal de Microsoft Entra ID para conceder o revocar el acceso a DocuBot. Esto centraliza y simplifica la administración de permisos.

Como administrador, es crucial mantener los costes operativos bajo control.

- **Azure Cost Management**: Utiliza esta herramienta en el portal de Azure para visualizar los costes desglosados por servicio. Te permite identificar qué componentes (p. ej., Azure OpenAI, Cosmos DB) están generando más gasto [62].
- **Presupuestos y Alertas**: Configura presupuestos mensuales para el grupo de recursos del proyecto. Asocia alertas a estos presupuestos para recibir notificaciones por correo electrónico cuando el gasto alcance ciertos umbrales.

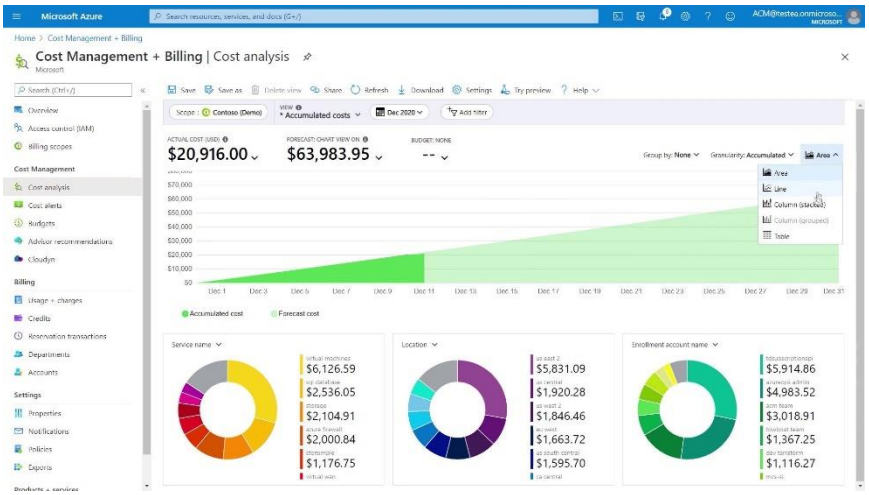


Figura 87. Ejemplo de Azure Cost Management

Bibliografía

[1] «Chatbots empresariales: Todo lo que necesitas saber en 2025». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://botpress.com/es/blog/enterprise-chatbots
[2] A. D. Martin, «20 años de SharePoint», Por una nube sostenible. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://blogs.encamina.com/por-una-nube-sostenible/20-anos-de-sharepoint/
[3] «Herramientas de productividad de IA para Microsoft 365 | Microsoft 365». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.microsoft.com/es-es/microsoft-365/copilot
[4] «Google NotebookLM | Note Taking & Research Assistant Powered by AI». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://notebooklm.google/
[5] «Work AI for all - AI platform for agents, assistant, search». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.glean.com/
[6] «Microsoft 365 Copilot: Planes y precios: IA para negocios | Microsoft 365». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.microsoft.com/es-es/microsoft-365/copilot/pricing
[7] «Google NotebookLM | Note Taking & Research Assistant Powered by AI». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://notebooklm.google/
[8] «Microsoft Teams Revenue and Usage Statistics (2025)», Business of Apps. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.businessofapps.com/data/microsoft-teams-statistics/
[9] J. Amengual, «Cuota de mercado de los proveedores de nube (Mayo 2024)», BlockStellart. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://blockstellart.com/cuota-de-mercado-de-los-proveedores-de-nube-mayo-2024/
[10] «Microsoft 365: Why is it Better than its Competitors Today?», The Bear Travel. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://thebear.travel/416/Microsoft-365:-Why-is-it-Better-than-its-Competitors-Today?
[11] «Top 22 benefits of chatbots for businesses and customers», Zendesk. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.zendesk.es/blog/5-benefits-using-ai-bots-customer-service/
[12] J. F. Gómez, «Arquitectura Hexagonal», Codely Pro. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://pro.codely.com/library/arquitectura-hexagonal-31201/
[13] «One Hot Encoding in Machine Learning», GeeksforGeeks. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.geeksforgeeks.org/machine-learning/ml-one-hot-encoding/
[14] Manikanth, «Understanding Contextualized Word Embeddings: The Evolution of Language Understanding in AI», Medium. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://manikanthgoud123.medium.com/understanding-contextualized-word-embeddings-the-evolution-of-language-understanding-in-ai-8bf79a98eb51
[15] F. Luna, «UMAP: An alternative dimensionality reduction technique», MCD-UNISON. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://medium.com/mcd-unison/umap-an-alternative-dimensionality-reduction-technique-7a5e77e80982
[16] «RPubs - Redes Neuronales». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://rpubs.com/luis_castillo/redes_neuronales

[17] «What are Convolutional Neural Networks? | IBM». Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://www.ibm.com/think/topics/convolutional-neural-networks
[18] A. Vaswani *et al.*, «Attention Is All You Need», 2 de agosto de 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.
[19] ifttt-user, «Explaining Attention in Transformers [From The Encoder Point of View] | Towards AI». Accedido: 7 de julio de 2025. [En línea]. Disponible en: https://towardsai.net/p/l/explaining-attention-in-transformers-from-the-encoder-point-of-view
[20] J. Vetterle, «Scaling Laws for LLM Pretraining», Jonas Vetterle Personal Page & Blog. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://www.jonvet.com/blog/llm-scaling-laws
[21] «(PDF) Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects», ResearchGate. Accedido: 13 de julio de 2025. [En línea]. Disponible en: https://www.researchgate.net/publication/372258530_Large_Language_Models_A_Comprehensive_Survey_of_its_Applications_Challenges_Limitations_and_Future_Prospects
[22] «A Comprehensive Guide to Multimodal LLMs and How they Work». Accedido: 7 de julio de 2025. [En línea]. Disponible en: https://www.ionio.ai/blog/a-comprehensive-guide-to-multimodal-llms-and-how-they-work
[23] A. Canales, «Análisis del Ecosistema 2025», Capitole. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://capitole-consulting.com/es/de-turing-a-agentes-autonomos-2025-ecosistema-llm/
[24] «Small Language Models (SLM): A Comprehensive Overview». Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://huggingface.co/blog/jjokah/small-language-model
[25] T. B. B. Lab, «Modelos de razonamiento en IA: Cómo funcionan», The Black Box Lab. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://theblackboxlab.com/modelos-de-razonamiento/
[26] «Guía de Ingeniería de Prompt – Nextra». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.promptingguide.ai/es
[27] A. Mishra, «Five Levels of Chunking Strategies in RAG| Notes from Greg's Video», Medium. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://medium.com/@anuragmishra_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d
[28] HeidiSteen, «Puntuación de relevancia BM25 - Azure AI Search». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/search/index-similarity-and-scoring
[29] «Una guía completa de búsqueda híbrida». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.elastic.co/es/what-is/hybrid-search
[30] Valentín, «Ventajas de tener un agente de IA con RAG en tu web para mejorar la atención al cliente», VA360 Academy. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://va360.academy/ventajas-de-tener-un-agente-de-ia-con-rag-en-tu-web-para-mejorar-la-atencion-al-cliente/
[31] Yugank .Aman, «Advanced RAG Techniques», Medium. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://medium.com/@yugank.aman/advanced-rag-techniques-0c283aacf5ba

[32] «Hypothetical Document Embeddings (HyDE)». Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://docs.haystack.deepset.ai/docs/hypothetical-document-embeddings-hyde
[33] «The aRt of RAG Part 3: Reranking with Cross Encoders | by Ross Ashman (PhD) | Medium». Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://medium.com/@rossashman/the-art-of-rag-part-3-reranking-with-cross-encoders-688a16b64669
[34] C. J. Yang, «Advanced RAG and the 3 types of Recursive Retrieval», Knowledge Graph RAG. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://medium.com/enterprise-rag/advanced-rag-and-the-3-types-of-recursive-retrieval-cdd0fa52e1ba
[35] P. Nayak, «Semantic Chunking for RAG», The AI Forum. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://medium.com/the-ai-forum/semantic-chunking-for-rag-f4733025d5f5
[36] A. L. Pascual, «Introduction to AI Agents», Medium. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://medium.com/@aleixlopez/introduction-to-ai-agents-62a790d0bc22
[37] «Hola, GPT-4o». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://openai.com/es-ES/index/hello-gpt-4o/
[38] HeidiSteen, «Consulta híbrida - Azure AI Search». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/search/hybrid-search-how-to-query
[39] «Nuevos modelos de integración y actualizaciones de API». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://openai.com/es-ES/index/new-embedding-models-and-api-updates/
[40] markjbrown, «Azure Cosmos DB». Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/cosmos-db/
[41] msmbaldwin, «¿Qué es Azure Key Vault?» Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/key-vault/general/basic-concepts
[42] v-aangie, «Información general sobre las tarjetas adaptables para Teams - Power Automate». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/power-automate/overview-adaptive-cards
[43] «Scrum for One: Adapting Agile Scrum Methodology for Individuals», Lucidchart. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.lucidchart.com/blog/scrum-for-one
[44] admin, «¿Cuánto debe durar mi sprint en Scrum? - NeuronForest». Accedido: 12 de julio de 2025. [En línea]. Disponible en: https://neuronforest.es/cuanto-debe-durar-mi-sprint-en-scrum/, https://neuronforest.es/cuanto-debe-durar-mi-sprint-en-scrum/
[45] J. Hoyos, «SCRUM y los puntos de historia, ¿Cómo funcionan?», Incentro. Accedido: 12 de julio de 2025. [En línea]. Disponible en: https://www.incentro.com
[46] «Microsoft 365 Empresa Básico | Microsoft 365». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://www.microsoft.com/es-es/microsoft-365/business/microsoft-365-business-basic
[47] Daniel, «Salario ingeniero de IA 2024: ¿Cuánto se puede ganar?», DataScientest. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://datascientest.com/es/salario-ingeniero-de-ia-2024

[48] «FTE (Equivalente a Tiempo Completo): ¿qué es y qué utilidades tiene para tu empresa?», Personio. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://www.personio.es/glosario/fte/
[49] B. Cottier, «LLM inference prices have fallen rapidly but unequally across tasks», Epoch AI. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://epoch.ai/data-insights/llm-inference-price-trends
[50] «8. Gestión de los Riesgos del Proyecto», La guia PMBOK. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://uacm123.weebly.com/8-gestioacuten-de-los-riesgos-del-proyecto.html
[51] admin, «Nadella's Radical Vision: How Agentic AI is changing SaaS forever», action.ai - Conversational Interfaces and Automated Customer Services. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://action.ai/nadellas-radical-vision-how-agentic-ai-is-changing-saas-forever/
[52] E. Salguero, «Arquitectura Hexagonal», Medium. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f
[53] «Inyección de dependencias en componentes | Software Crafters», Inyección de dependencias en componentes | Software Crafters. Accedido: 9 de julio de 2025. [En línea]. Disponible en: http://softwarecrafters.ioinyeccion-de-dependencias-componentes
[54] Zimmergren, «Definición de la convención de nomenclatura - Cloud Adoption Framework». Accedido: 9 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/cloud-adoption-framework/ready/azure-best-practices/resource-naming
[55] HeidiSteen, «Creación de un conjunto de aptitudes - Azure AI Search». Accedido: 9 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/search/cognitive-search-defining-skillset
[56] M. Hemphill, «Introduction to Dev Containers», Versent Tech Blog. Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://medium.com/versent-tech-blog/introduction-to-dev-containers-4c01cb1752a0
[57] surbhigupta12, «Introduction to Teams AI library - Teams». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/en-us/microsoftteams/platform/bots/how-to/teams-conversational-ai/teams-conversation-ai-overview
[58] F. Dimitrijeski, «Builder Pattern in TypeScript», Medium. Accedido: 11 de julio de 2025. [En línea]. Disponible en: https://medium.com/@filipdimitrijeski7/builder-pattern-in-typescript-a7787d180dfc
[59] «A Complete Guide to Pino Logging in Node.js | Better Stack Community». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://betterstack.com/community/guides/logging/how-to-install-setup-and-use-pino-to-log-node-js-applications/
[60] surbhigupta, «Debug bot using Agents Playground - Teams». Accedido: 8 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/en-us/microsoftteams/platform/toolkit/debug-your-agents-playground
[61] «Ragas». Accedido: 13 de julio de 2025. [En línea]. Disponible en: https://docs.ragas.io/en/stable/
[62] shasulin, «Introducción a Cost Management - Microsoft Cost Management». Accedido: 12 de julio de 2025. [En línea]. Disponible en: https://learn.microsoft.com/es-es/azure/cost-management-billing/costs/overview-cost-management