PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

CARRERA DE: TECNOLOGÍAS DE LA INFORMACIÓN



Trabajo de Titulación

Tema:

Diseño y desarrollo de un prototipo de Help Desk con asistencia de soporte técnico basado en IA para la gestión del área de TI dentro de un entorno empresarial hipotético

AUTOR:

Jhosua Esteban Fuertes Meléndez

DIRECTOR:

Juan Francisco Chafla Altamirano

QUITO DM, JUNIO DE 2026

DEDICATORIA

A mi familia, por su apoyo incondicional y por confiar en todo lo que soñé. Espero seguir contando con su guía para tener una vida mejor.

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a mi madre y a mi padre, quienes han sido un pilar fundamental en mi formación académica. Su paciencia, apoyo constante y confianza en mis aspiraciones me han impulsado a seguir adelante y a formarme como un gran ser humano y un excelente profesional.

Asimismo, agradezco a mi tutor, el Ing. Juan Francisco Chafla, por su objetividad en cada corrección y observación realizada durante el desarrollo de mi trabajo de titulación. Su acompañamiento y orientación fueron fundamentales para la culminación y entrega de este proyecto.

RESUMEN

Help Desk es una herramienta de software especializado en la gestión de incidencias de usuarios o clientes pertenecientes a una empresa, a través de la cual brinda soporte técnico mediante diferentes canales de comunicación, como correo electrónico, chatbots, números telefónicos o portales de usuarios. En la actualidad, estos sistemas se han modernizado y adaptado a los constantes cambios tecnológicos, siendo uno de los más relevantes la incorporación de modelos de lenguaje de gran tamaño (LLM) en los procesos de automatización de los flujos de trabajo. Esta integración potencia de manera considerable la eficiencia operativa, lo que se traduce en tiempos de respuesta más rápidos, una clasificación de tareas más precisa y una mejora en la calidad del servicio ofrecido. Ante este contexto, se plantea el desarrollo de un prototipo de *Help Desk* con asistencia de soporte técnico basado en IA, que funcione de manera complementaria con el técnico del área de TI y brinde un servicio capaz de integrar herramientas emergentes como LangChain, LangGraph, RAG. Estas tecnologías ayudan a procesar la consulta del usuario y generar una respuesta acorde con su requerimiento. Para el desarrollo del prototipo, se diseñaron e implementaron dos bases de datos: una vectorial y una relacional. La base de datos vectorial se encargó de almacenar la base de conocimiento del sistema, correspondiente a la documentación oficial utilizada para la recuperación de información. Por su parte, la base de datos relacional permitió almacenar la información asociada a cada consulta, como la clasificación de ticket, el estado del requerimiento y los datos del usuario.

El prototipo, estructurado bajo una arquitectura por capas, fue probado en diez escenarios distintos bajo condiciones que simulaban un entorno de trabajo hipotético. Los resultados demostraron una gestión adecuada, con una tasa de resolución de incidencias del 80% y un tiempo promedio de respuesta de 8,79 segundos, excluyendo el caso de la consulta escalada al especialista del área. Estas pruebas evidencian que es posible integrar herramientas basadas en inteligencia artificial para brindar soporte técnico personalizado. Asimismo, demuestran que dichas herramientas pueden incorporarse en sistemas Help Desk tradicionales con el propósito de mejorar los procesos de automatización de tareas y optimizar la gestión de incidencias.

Palabras clave: Help Desk, LangChain, LangGraph, RAG, LLMs, TI.

CONTENIDO

ÍNDICE DE FIGURAS VIII
ÍNDICE DE TABLAS X
LISTA DE ABREVIATURAS XI
CAPÍTULO I: INTRODUCCIÓN 1
1.1. Justificación 2
1.2. Planteamiento del problema 3
1.3. Objetivos 4
1.3.1. Objetivo general 4
1.3.2. Objetivos específicos 4
1.4. Alcance 5
CAPÍTULO II: MARCO TEÓRICO Y CONCEPTUAL 6
2.1. Antecedentes 7
2.2. Help Desk 8
2.3. Gestión de incidencias dentro del área de TI 9
2.4. Soporte técnico enfocado a la inteligencia artificial 12
2.5. Conceptos básicos de inteligencia artificial 12
2.5.1. Procesamiento del lenguaje natural 13
2.5.2. Aprendizaje automático 14
2.5.3. Representación del conocimiento y razonamiento 15
2.6. ChatBots 15
2.6.1. Chatbots basados en reglas 16
2.6.2. Chatbots conversacionales 16
2.6.3. Chatbots contextuales 17
2.7. Large Language Models (LLM) 17
2.7.1. Modelos LLM actuales 18
2.8. RAG 19
2.8.1. Limitaciones y fortalezas 20
2.8.2. Funcionamiento en sistemas de soporte Help Desk 21
2.9. Bases de datos vectoriales 22
2.9.1. Estructura y funcionamiento 22
2.9.2. Sistema basado en búsqueda semántica 24
2.10. Tecnologías y herramientas de desarrollo del prototipo 27

2.10.1. LangChain 27
2.10.2. LangGraph 27
2.10.3. Streamlit 28
2.10.4. FastAPI 29
CAPÍTULO III: DISEÑO DEL PROTOTIPO 30
3.1. Arquitectura del sistema 31
3.1.1. Diseño de la arquitectura del sistema Help Desk 32
3.2. Diseño de la base de datos 34
3.2.1. Diseño de modelo entidad-relación 35
3.2.2. Reglas de cardinalidad 37
3.3. Diseño de la interfaz gráfica 39
3.3.1. Estructura general de la interfaz 40
3.3.2. Visualización de la interfaz gráfica 42
CAPÍTULO IV: DESARROLLO DEL PROTOTIPO 46
4.1. Configuración del entorno de desarrollo 47
4.1.1. Estructura del proyecto 47
4.2. Funcionamiento de la aplicación 50
4.2.1. Implementación de documentos en el sistema RAG 50
4.2.2. Configuración inicial de RAG 53
4.2.3. Configuración avanzada de RAG 57
4.3. Creación e integración del grafo con LangGraph 61
4.3.1. Human in the loop 64
4.3.2. Funciones de enrutamiento 66
4.3.3. Conexión del grafo 67
4.3.4. Checkpoints y persistencia de estado 68
CAPÍTULO V: ANÁLISIS DE RESULTADOS 70
5.1. Pruebas del sistema 71
5.1.1. Fortalezas y limitaciones del prototipo 81
5.2. Métricas de evaluación 82
5.2.1. Cálculo de la confianza 83
5.2.2. Evaluación del tiempo de respuesta y resolución de incidencias del prototipo 84
CAPÍTULO VI: CONCLUSIONES Y RECOMENDACIONES 87

6.1. Conclusiones 88
6.2. Recomendaciones 89
BIBLIOGRAFÍA 92

ÍNDICE DE FIGURAS

Figura 1 Esquema de trabajo de Help Desk. 9
Figura 2 Diagrama de flujo de Help Desk y Service Desk. 11
Figura 3 Proceso de Recuperación-Aumento-Generación (RAG). 19
Figura 4 Estructura y funcionamiento de una base de datos vectorial. 23
Figura 5 Estructura y funcionamiento de un sistema de búsqueda. 24
Figura 6 Estructura y funcionamiento de un sistema de búsqueda basado en LLMs. 25
Figura 7 Estructura y funcionamiento de la comparación de la consulta vs documentos. 26
Figura 8 Arquitectura organizada por capas del prototipo Help Desk. 32
Figura 9 Modelo Entidad-Relación del prototipo Help Desk. 35
Figura 10 Flujo de trabajo del prototipo Help Desk: caso de estudio. 37
Figura 11 Funciones del departamento de TI. 39
Figura 12 Apartado visual del especialista TI organizado por bloques. 41
Figura 13 Apartado visual del usuario organizado por bloques. 41
Figura 14 Página principal del sistema Help Desk. 42
Figura 15 Diseño de interfaz gráfica del usuario (Prototipo Help Desk). 43
Figura 16 Diseño de interfaz gráfica del especialista (Prototipo Help Desk). 44
Figura 17 Diseño del panel de control (Prototipo Help Desk). 45
Figura 18 Sitio de descarga del entorno PyCharm. 47
Figura 19 Estructura del proyecto Help Desk. 48
Figura 20 Archivo “configuración.py” del proyecto Help Desk. 49
Figura 21 Diagrama de funcionamiento prototipo Help Desk. 50
Figura 22 Formato del documento FAQ (preguntas frecuentes) prototipo Help Desk. 51
Figura 23 Formato del documento guía de resolución prototipo Help Desk. 52
Figura 24 Formato del documento manual de usuario prototipo Help Desk. 52
Figura 25 Configuración inicial de fragmentación de documentos. 53
Figura 26 Función de carga de documentos al sistema Help Desk. 54
Figura 27 Obtención del tipo de documento. 54
Figura 28 Función de generador de id y de división de documentos. 55
Figura 29 Creación de la base de datos vectorial (ChromaDB) del sistema Help Desk. 55
Figura 30 Carga de la base de datos vectorial (ChromaDB) del sistema Help Desk. 56
Figura 31 Configuración inicial del sistema RAG. 56
Figura 32 Test de prueba del funcionamiento de RAG. 57
Figura 33 Inicialización de RAG dentro del sistema Help Desk con ChromaDB. 58
Figura 34 Proceso de carga de la base vectorial al sistema Help Desk. 58
Figura 35 Proceso de búsqueda de información en el sistema RAG. 59
Figura 36 Generación de respuesta al usuario con base en el modelo GPT. 60
Figura 37 Método para medir el nivel de confianza de la respuesta. 60
Figura 38 Definición del estado. 61
Figura 39 Inicialización del modelo GPT. 62
Figura 40 Búsqueda de información relevante. 63
Figura 41 Clasificación del contexto RAG. 63

Figura 42 Proceso de escalamiento del requerimiento. 64
Figura 43 Proceso de respuesta del especialista del área. 65
Figura 44 Proceso de asignación para la respuesta final. 66
Figura 45 Función de enrutamiento basada en la clasificación del requerimiento. 66
Figura 46 Función de enrutamiento basada en la respuesta del técnico. 67
Figura 47 Conexión y creación del grafo final con LangGraph. 68
Figura 48 Compilación del grafo del prototipo de Help Desk. 69
Figura 49 Test de prueba Nº1 (Ingreso de consulta). 71
Figura 50 Test de prueba Nº1 (Gestión de la consulta). 72
Figura 51 Test de prueba Nº1 (Historial del procesamiento). 73
Figura 52 Test de prueba Nº1 (Respuesta final generada por RAG). 73
Figura 53 Test de prueba Nº2 (Ingreso de consulta). 74
Figura 54 Test de prueba Nº2 (Gestión de la consulta). 74
Figura 55 Test de prueba Nº2 (Contexto recuperado por RAG). 75
Figura 56 Test de prueba Nº2 (Historial del procesamiento). 75
Figura 57 Test de prueba Nº3 (Panel del especialista). 76
Figura 58 Test de prueba Nº4 (Panel del especialista). 77
Figura 59 Test de prueba Nº5 (Panel del especialista). 78
Figura 60 Test de prueba Nº6 (Panel del especialista). 79
Figura 61 Test de prueba Nº7 (Panel del especialista). 80
Figura 62 Módulo de consulta de tickets (Pruebas del sistema). 84
Figura 63 Tiempo de respuesta por ticket. 85
Figura 64 Resolución de incidencias. 85
Figura 65 Resumen de resultados obtenidos. 86

ÍNDICE DE TABLAS

Tabla 1 Fortalezas y limitaciones del prototipo de Help Desk. 81
Tabla 2 Métricas de evaluación del prototipo Help Desk. 82
Tabla 3 Ecuación de estimación de la confianza. 83

LISTA DE ABREVIATURAS

HTML	*HyperText Markup Language* (Lenguaje de Marcado de Hipertexto)
BD	*Database* (Base de datos)
CSS	*Cascading Style Sheets* (Hojas de Estilo en Cascada)
API	*Application Programming* Interface (Interfaz de Programación de Aplicaciones)
FAQ	*Frequently Asked Questions* (Preguntas Frecuentes)
GPT	*Generative Pre-trained Transformer* (Transformador Generativo Preentrenado)
UI	*User Interface* (Interfaz de Usuario)
TI	*Information Technology* (Tecnologías de la Información)
RAG	*Retrieval-Augmented Generation* (Generación Aumentada por Recuperación)
MD	*Markdown* (Lenguaje de marcado ligero)
ITIL	*Information Technology Infrastructure Library* (Biblioteca de Infraestructura de Tecnologías de la Información)
GUI	*Graphical User Interface* (Interfaz Gráfica de Usuario)
IDE	*Integrated Development Environment* (Entorno de Desarrollo Integrado)
IA	*Artificial Intelligence* (Inteligencia Artificial)
LLM	*Large Language Model* (Modelo de Lenguaje de Gran Tamaño)
VPN	*Virtual Private Network* (Red Privada Virtual)

CAPÍTULO I: INTRODUCCIÓN

La inteligencia artificial forma parte fundamental de la vida del ser humano. Con el paso del tiempo, surgen nuevos modelos de inteligencia artificial impulsados por herramientas como ChatGPT, Claude o Gemini, que procesan información, toman decisiones autónomas y son capaces de dar solución a problemáticas complejas del mundo real.

Estos modelos pueden ser de gran utilidad para los sistemas *Help Desk*, ya que brindan atención y soporte personalizado a los distintos usuarios que forman parte de una empresa, además de reducir la carga de trabajo del técnico o especialista designado en el área de Tecnologías de la Información, a través de un *chatbot* basado en inteligencia artificial.

Estos sistemas ya están integrados dentro de la infraestructura tecnológica de las empresas, pero muchos no están actualizados ni mucho menos adaptados a los modelos de lenguaje de gran tamaño (LLM).

En este capítulo se exponen los siguientes puntos: la justificación, el planteamiento del problema, los objetivos guía y el alcance que delimita el proyecto, estableciendo un marco preliminar para el diseño y desarrollo de un prototipo de sistema *Help Desk* enfocado en la gestión de incidencias dentro del área de TI.

1.1. Justificación

El área de TI (Tecnologías de la Información) es esencial para asegurar el funcionamiento de equipos y servicios tecnológicos en una organización. Sin embargo, enfrenta retos como la gestión compleja de solicitudes, la saturación del servicio y la sobrecarga del personal, lo que afecta la calidad del soporte técnico.

Los *Help Desk* empresariales automatizan preguntas y procesos generales para los usuarios o clientes. Su IA (Inteligencia Artificial), con base en un conocimiento previo, utiliza RAG (Generación Aumentada por Recuperación) para fortalecer los modelos de lenguaje LLM y minimizar el sesgo y la incertidumbre en la información.

Por lo tanto, para desarrollar un prototipo de *Help Desk*, se busca simular, a través de un aplicativo web, un escenario real de operación en una empresa hipotética, evitando comprometer cualquier información sensible. Se adoptará un enfoque práctico y aplicable que responda a las necesidades actuales de un entorno laboral.

Este trabajo se enfoca en automatizar la atención de incidencias para mejorar los tiempos de respuesta mediante la integración de IA. Incluye la clasificación de tickets, la generación automática de respuestas y el uso de asistentes virtuales, lo que optimiza la gestión del área de TI y aporta una alternativa innovadora y adaptable a entornos empresariales.

1.2. Planteamiento del problema

Los *Help Desk* son herramientas fundamentales dentro del ámbito de TI, ya que permiten automatizar la asistencia y centralizar las distintas solicitudes en un único servicio. En un entorno empresarial real, es frecuente que surjan situaciones como inconvenientes de conexión a una VPN o fallas en la instalación de software en los equipos de trabajo, las cuales requieren una respuesta o solución oportuna.

Sin embargo, la implementación de un *Help Desk*, por sí sola, no asegura una gestión adecuada de los problemas, ya que resulta necesario comprender su estructura de funcionamiento y proporcionar a sus agentes los recursos pertinentes para el cumplimiento eficiente de sus labores.

Con base en la problemática planteada centrada en la óptima gestión de incidencias del área de TI surge la siguiente pregunta principal para el trabajo:

¿Cómo se puede desarrollar un prototipo de *Help Desk* incorporando algoritmos de inteligencia artificial con respuestas automáticas mediante el uso de RAG que garantice la atención de los usuarios de manera rápida y personalizada?

Preguntas Secundarias:

¿Qué herramientas se deben tener en consideración para el desarrollo y diseño de este prototipo de *Help Desk*?

¿Cuáles son los pasos por seguir para el uso correcto de este prototipo y qué funciones podrían implementarse a futuro para una mejora continua?

Este trabajo de titulación brindará una alternativa que automatice gran parte de los procesos de asistencia de soporte técnico en un solo sistema orientado al área de TI de una empresa hipotética que desconoce o quiere implementar este tipo de tecnologías.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar un prototipo de *Help Desk* con asistencia de soporte técnico basado en IA, mediante un aplicativo web que simule un entorno de trabajo y permita gestionar más de un 50% el número de incidencias relacionadas con el área de TI.

1.3.2. Objetivos específicos

- Diseñar e implementar un módulo de gestión de incidencias que permita la automatización de tickets y brinde asistencia mediante un *chatbot*.
- Construir una base de datos vectorial que procese consultas de datos no estructurados de forma segura, confiable y escalable.
- Implementar el *Help Desk* en un entorno simulado que represente el flujo de trabajo del área de TI dentro de un contexto empresarial hipotético.
- Evaluar el desempeño del prototipo de *Help Desk* en términos de confiabilidad, tiempos de respuesta y resolución de incidencias.

1.4. Alcance

En este trabajo de titulación se desarrollará un prototipo de *Help Desk* con asistencia de soporte técnico basado en IA dentro de un entorno empresarial hipotético en el que se implementarán herramientas que integren funcionalidades relacionadas con IA.

Adicionalmente se analizará el desempeño y las limitaciones de cada una de ellas y se determinarán las implicaciones que tienen dentro de procesos de trabajo que aún no están automatizados con el fin de comprobar un incremento superior al 50% en la atención de incidencias.

Su diseño e implementación se construirán bajo un aplicativo web en el que un algoritmo de inteligencia artificial decidirá si las solicitudes o peticiones que realiza el usuario deban ser respondidas por ella misma mediante RAG o deba escalar la ayuda a un humano. Se evaluarán ambos casos una en donde la IA deba responder y otra donde la persona lo deba hacer. Se determinará el grado de confiabilidad de la respuesta con base en la petición realizada.

Como es una versión reducida de un *Help Desk* empresarial no se inmiscuirá en aspectos técnicos ni mucho menos se implementará este prototipo en entidades estatales o privadas ya que se requeriría información adicional como flujos de trabajo dentro de las áreas de TI, así como capacitaciones al respectivo personal.

Este trabajo servirá de guía y como base para proyectos de mayor escala a pesar de su simplicidad ayudará a comprender de mejor forma cómo estos sistemas automatizan procesos y cómo optimizan el servicio de asistencia técnica.

CAPÍTULO II: MARCO TEÓRICO Y CONCEPTUAL

En este capítulo se expone el marco teórico y conceptual que sustenta el diseño y desarrollo del prototipo de sistema *Help Desk* basado en inteligencia artificial. Como punto de partida, se abordan los conceptos básicos que componen un *Help Desk* y su relación con la gestión de incidencias en el área de TI. Asimismo, se redefine el concepto de soporte técnico desde una perspectiva acorde con los modelos de inteligencia artificial modernos.

De igual manera, se analiza la estructura y el funcionamiento tanto de las bases de datos vectoriales como de las técnicas de mejora de respuesta en relación con el contexto de la pregunta o petición del usuario, en el que RAG desempeña un papel fundamental. Posteriormente, se profundiza en los diferentes *frameworks*, como LangChain, LangGraph, Streamlit y FastAPI, los cuales brindan apoyo para el diseño, modelación y desarrollo del *backend* del sistema.

Con la única finalidad de establecer un análisis más sólido que permita comprender el funcionamiento y la importancia de cada uno de estos elementos dentro del planteamiento de la propuesta.

2.1. Antecedentes

A pesar de que en los últimos años los *Help Desk* han ganado relevancia debido al creciente desarrollo tecnológico, el cual ha generado una mayor dependencia del software por parte de los usuarios, su aplicación no se ha limitado únicamente a sectores estratégicos como la industria, la salud o la educación, sino que también se ha extendido a pequeñas y medianas empresas.

En estos contextos, se implementan canales de soporte que incorporan *chatbots* a través de APIs, capaces de crear y gestionar solicitudes mediante un sistema de tickets. Dicho modelo puede extrapolarse igualmente a entornos corporativos, en los cuales se gestionan procesos entre usuarios de una misma organización. En ese sentido, el *Help Desk* tiene como propósito principal establecer un grupo de personas encargado de proporcionar soporte para la ejecución de las tareas del personal de trabajo (Muns, 1993).

Aunque su uso no es reciente, sino que se remonta a décadas anteriores, desde la aparición del primer ordenador personal en 1981, los primeros *Help Desk* eran bastante rudimentarios, ya que carecían de personal debidamente capacitado y de las herramientas o bases de datos que posteriormente se incorporaron a inicios de siglo. Con el surgimiento del marco de referencia *ITIL (Information Technology Infrastructure Library)*, se introdujo el concepto de *service desk*, que en la actualidad constituye el canal principal de comunicación para atender las consultas de los clientes en relación con el área de TI.

En la actualidad, existen diversas formas de implementar un *Help Desk*, que van desde el fortalecimiento del servicio de soporte técnico mediante tecnologías como HTML5 y MySQL, hasta soluciones adaptadas a aplicaciones CAPLA dentro de arquitecturas ya funcionales. En este contexto, el presente trabajo de titulación propone una alternativa basada en el diseño y desarrollo de un prototipo de *Help Desk* inteligente, concebido en una versión reducida, pero práctica, dentro de un entorno empresarial hipotético.

Para ello, se integrarán técnicas como RAG y diversos *frameworks*, con el fin de dar una atención más eficiente y optimizar la interacción con el usuario.

2.2. Help Desk

Es un sistema que brinda soporte al proceso de gestión tecnológica; está conformado por una serie de recursos humano-máquina que proveen asistencia a usuarios informáticos en los diferentes niveles jerárquicos de una empresa, adaptándose a las necesidades de esta.

El *Help Desk* es un área especializada en soporte técnico por teléfono, de manera presencial o vía acceso remoto o página web (Bulchand y Melián, 2010). Atiende las diferentes incidencias que los usuarios puedan presentar en su área de trabajo a través de entornos asistidos por computadoras o software especializado.

Según la biblioteca de infraestructura de la tecnología de información (ITIL) dentro de las empresas consolidadas al ISO/IEC 20000 (Roncancio Roncancio & Mesa Páez, 2011), un *Help Desk* tiene la posibilidad de ofrecer una vasta gama de servicios centralizados además de formar parte de un *service desk* (centro de servicios) con el fin de incluir dentro de la red de asistencia de TI prácticas de administración mejoradas.

Además, conjuga el talento humano y recursos tecnológicos, para solucionar iteraciones de incidencias de manera integral (Jaramillo Castro & Morocho Puchaicela, 2016). Todo esto se traduce en tareas que guarden algún tipo de relación con los servicios de asistencia de TI, en el que cada problema se clasifica de acuerdo con su nivel de prioridad.

Los usuarios son atendidos en el orden en el que se receptan las incidencias; sin embargo, dependiendo de la gravedad de la situación, este criterio puede variar. Y de ser necesario el técnico de turno se vea en la necesidad de asistir al lugar del incidente. En gran parte de los casos, esta actividad implica una serie de responsabilidades que han sido abordadas de manera imparcial e improvisada, recurriendo a metodologías rudimentarias para el cumplimiento de la gestión de servicios.

Para cubrir las necesidades de una empresa u organización que requiere asistencia en sus servicios de soporte, como mínimo se deberá tomar en cuenta los siguientes lineamientos:

- “Automatización de procesos enfocados en la atención de incidencias, con el fin de reducir la contratación excesiva de personal, así como del tiempo necesario para ejecutar cada tarea”.

- “Establecimiento de una jerarquía de prioridades a través de un reglamento o normativa que regule los riesgos e implicaciones que conllevan los servicios de soporte técnico”.
- “Evaluación de la calidad del servicio en términos de atención de incidencias”.

2.3. Gestión de incidencias dentro del área de TI

La gestión de incidencias dentro del área de TI está relacionada intrínsecamente con algunos conceptos tradicionales de atención al cliente: el punto único de contacto, el ticket, la incidencia, la solicitud de servicio, la clasificación, la priorización, el seguimiento, el escalado y la base de conocimiento.

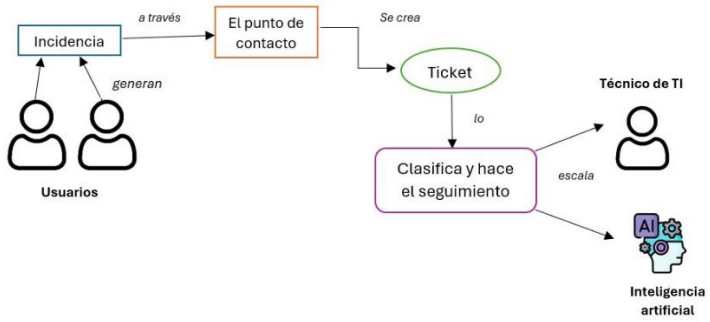


Figura 1 Esquema de trabajo de Help Desk.

Elaboración propia.

A través de la figura 1 se explica a breves rasgos el flujo de trabajo que desempeña un *Help Desk* dentro de un entorno empresarial hipotético a un nivel general:

1. Uno o varios usuarios generan una incidencia a través de un punto de contacto que se encarga de centralizar la atención de los requerimientos.
2. Se registra el ticket en forma de solicitud o pregunta dentro del mismo sistema.
3. El sistema *Help Desk* clasifica los tickets y con base en el nivel de prioridad o complejidad lo escala a un técnico de TI o a la misma inteligencia artificial.
4. Atiende el requerimiento del usuario y finaliza la tarea.
5. Repite el mismo procedimiento para la siguiente incidencia.

Para un análisis más profundo y detallado de este flujo de trabajo se requiere estructurar al *Help Desk* en tres niveles. En el primer nivel se identifica al usuario, aquel que se encarga de realizar la solicitud o de reportar el problema. Si la problemática se resuelve en esta primera

instancia, la solicitud se cierra y su solución se actualiza en la documentación correspondiente. Caso contrario, y de ser necesario, el requerimiento es escalado a un técnico cuya resolución será atendida en el segundo nivel.

El segundo nivel está conformado por los especialistas del *Help Desk*, que en gran parte de los casos llegan a ser los técnicos del área de TI. Muchos de ellos dedican esfuerzos a la atención de incidencias mientras desarrollan otras actividades. Su rol es importante ya que poseen la experiencia necesaria para solventar cualquier problema que se origine en el primer nivel.

Finalmente, el tercer nivel se caracteriza por encargarse de la atención de las áreas que se encuentran fuera del *Help Desk*, como la administración de bases de datos, el desarrollo de programas, el soporte técnico y la administración de redes (Roncancio Roncancio & Mesa Páez, 2011).

Cabe resaltar las destrezas, aptitudes y conocimientos del especialista, ya que este proporcionará las respectivas aclaraciones al usuario ante cualquier duda que pueda surgir respecto de la solución de su requerimiento, con base en un razonamiento lógico que identifique las posibles fortalezas y debilidades de los resultados obtenidos.

Por lo general suelen ser individuos cuyo perfil se ajusta a áreas de conocimiento relacionadas con el desarrollo de software, redes, electrónica o temas relacionados con la tecnología.

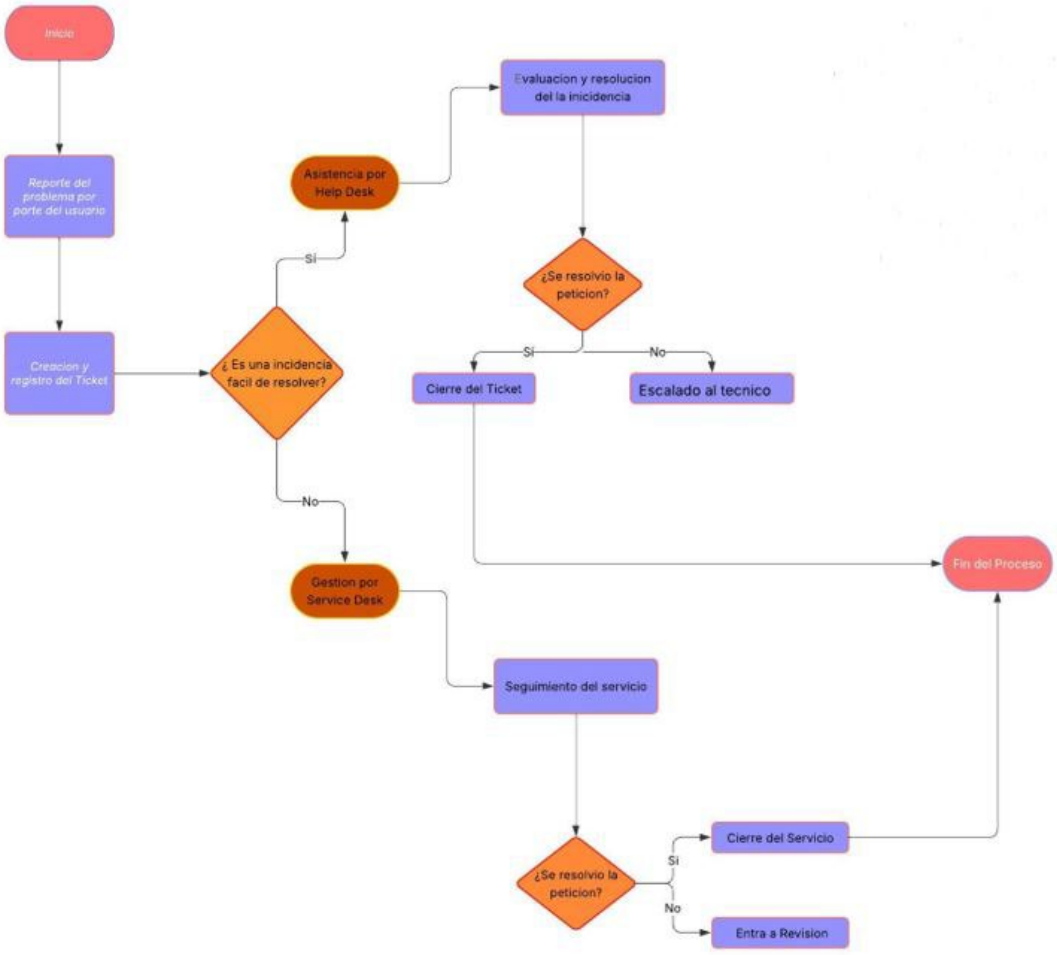


Figura 2 Diagrama de flujo de Help Desk y Service Desk.

Elaboración propia.

En lo que respecta a la figura 2 en este se compara dos términos que, a pesar de sonar similares, suelen confundirse y estos son: *Help Desk* y *Service Desk*. Su diferencia radica en sus alcances y objetivos, ya que este último no solo atiende incidencias, sino que crea canales de comunicación entre usuarios y técnicos, con la finalidad de dar seguimiento a los procesos del servicio y poder gestionarlos de forma integral dentro del área de TI.

2.4. Soporte técnico enfocado a la inteligencia artificial

Consiste en adaptar e implementar modelos y herramientas de inteligencia artificial en servicios de atención, gestión y resolución de incidencias dentro del flujo de trabajo en el área de TI.

En contrapartida con su versión más tradicional, que depende en mayor medida de la intervención directa del especialista, este incorpora mecanismos de mejora continua a los procesos de análisis de solicitudes, interpretación de lenguaje natural, restauración de información relevante para la obtención de respuestas más confiables en relación con el contexto de la pregunta.

En resumidas palabras, la inteligencia artificial permite automatizar tareas como la clasificación de tickets, la obtención de datos relevantes a partir de información alojada en documentación oficial con la finalidad de brindar una atención inicial al usuario mediante asistentes virtuales o *chatbots*. Todo esto constituye una base fundamental en el diseño y desarrollo de un sistema *Help Desk*.

Sin embargo, el soporte técnico asistido por inteligencia artificial no tiene la intención de reemplazar al técnico o especialista del área sino más bien reducir su carga de trabajo, así como de optimizar el tiempo que necesita para atender un requerimiento en un entorno laboral cada vez más competitivo.

2.5. Conceptos básicos de inteligencia artificial

La inteligencia artificial forma parte esencial en la vida cotidiana del ser humano. Con el pasar del tiempo surgen nuevos modelos de IA impulsados por herramientas como ChatGPT, Claude o Gemini que procesan información, toman decisiones autónomas y son capaces de dar soluciones a problemáticas complejas del mundo real.

La inteligencia artificial es una de las ramas de la informática que se encarga del estudio del razonamiento humano, focalizándose principalmente en cuatro definiciones diferentes: actuar como las personas, razonar como las personas, razonar racionalmente y actuar racionalmente. A pesar de que cada enfoque trata de formular y validar su propia definición, todos comparten un punto en común: la imitación del pensamiento humano.

Como una ciencia que busca la comprensión profunda de la inteligencia, trata de entender los alcances, límites y capacidades de la mente humana, lo que resulta un problema de alta complejidad. Por ende, las áreas fundamentales de la inteligencia artificial son:

- Representación del conocimiento y razonamiento
- Aprendizaje automático
- Procesamiento del lenguaje natural
- Visión por computadoras
- Robótica
- Reconocimiento automático del habla

De entre todas estas áreas de conocimiento la que más resalta es procesamiento de lenguaje natural ya que permite a la máquina entender, interpretar y manejar el lenguaje utilizado por los seres humanos (Sacoto, 2021). Aunque existen seis áreas fundamentales, en este trabajo solo se abordan tres: el aprendizaje automático, la representación del conocimiento y razonamiento, y el procesamiento del lenguaje natural.

2.5.1. Procesamiento del lenguaje natural

Durante la última década, el procesamiento del lenguaje natural ha experimentado cambios impulsados por una evolución constante de los modelos de alto aprendizaje y por una transformación en las infraestructuras tecnológicas que ha marcado un creciente interés en la automatización de tareas o procesos internos dentro de una empresa u organización.

Debido a su impacto significativo en diversas áreas de conocimiento, así como en las industrias manufactureras, esta está redefiniendo la manera en la que se procesan y gestionan los datos en respuesta a un mercado que no solamente debe comprender cómo comunicarse con una máquina, sino más bien entender el proceso de administrar grandes volúmenes de información sin comprometer la seguridad de sus usuarios.

El procesamiento de un computador se compone de dos fases las cuales son: el preprocesamiento de datos ingresados, el desarrollo de algoritmos, o la implementación de otros ya existentes para completar la tarea del procesamiento del lenguaje natural (Figueroa Sacoto, 2021). Todo ello se traduce en un proceso de limpieza y preparación de los datos de entrada, con

base en un filtro que determine el tipo de dato o formato que necesita el algoritmo para funcionar correctamente.

Este provee un filtro al *Help Desk* para la depuración, clasificación y asignación de información, mediante el cual se determina el formato y el tipo de dato que mejor se ajusta al algoritmo de búsqueda y garantice su correcto funcionamiento.

2.5.2. Aprendizaje automático

Este otro concepto esencial en inteligencia artificial, ya que permite comprender, desde un punto de vista lógico y matemático, un conjunto complejo de datos. Se trata de una técnica que permite reconocer patrones en el procesamiento de información relevante con la finalidad de mejorar el desempeño en cada tarea específica. En comparación a la programación estructural, esta no requiere programar cada regla de manera explícita.

La funcionalidad del aprendizaje automático se basa en tres términos esenciales que lo componen:

- Datos: Es la información con la que se entrena a cada modelo.
- Algoritmo: Secuencia ordenada y finita de instrucciones que se le integra al modelo para que procese los datos.
- Modelo: Es el resultado del aprendizaje, ideal para realizar predicciones.

Mediante el desarrollo de algoritmos, es que se pueden crear herramientas capaces de segmentar la información en datos cuyos patrones responden a resultados cuantificables.

En el marco de desarrollo del prototipo del *Help Desk*, el aprendizaje automático se integra dentro del sistema con la finalidad de analizar y clasificar incidencias, cuyo objetivo sea sugerir y mejorar los tiempos de respuesta. En consecuencia, los procesos de atención del soporte técnico dejan de ser sistemáticos y manuales para volverse más dinámicos e inteligentes, ajustándose a cada petición que realice el usuario.

2.5.3. Representación del conocimiento y razonamiento

Constituyen un pilar fundamental en procesos de imitación de comportamientos de naturaleza humana por parte de las máquinas, esto se traduce en la capacidad de tomar decisiones complejas ante una problemática planteada. Básicamente, la representación de conocimiento es exteriorizar un sistema de creencias, intenciones y juicios de un agente basado en inteligencia artificial, de modo que pueda desarrollar un razonamiento automatizado.

Su principal función consiste en crear modelos y estructuras para representar la información y el conocimiento, de modo que los sistemas inteligentes puedan utilizarlos (European Information Technologies Certification Institute, s. f.). En cambio, la representación del razonamiento es un método de análisis de modelos de conocimientos que funciona con base en la deducción, la lógica e inducción, con el fin de obtener nueva información que contribuya al proceso de mejoramiento y comprensión del contexto.

Aunque la representación de conocimiento está más ligada a métodos de codificación de información, en el que un sistema basado en inteligencia artificial es libre de tomar decisiones, también sirve de puente entre los datos brutos y el razonamiento significativo. En el momento en el que se representa el conocimiento en forma estructurada, los sistemas de inteligencia artificial pueden interpretar datos, extraer inferencias y aplicar técnicas de razonamiento para resolver problemas (European Information Technologies Certification Institute, s. f.).

Esta terminología se subdivide en dos conceptos diferentes, pero intrínsecamente relacionados con el aprendizaje de los modelos o estructuras basadas en inteligencia artificial. En el sistema *Help Desk*, ayudan a codificar los datos y a generar nuevos conocimientos, tratando de emular un proceso similar a la sinapsis, con la intención de mejorar la comprensión lectora y la capacidad de razonamiento.

2.6. Chatbots

La automatización de la información mediante el uso de *bots* —software con inteligencia artificial para programar "todo tipo de cosas" (Colle, 2017, p. 24) — está revolucionando la forma en la que se percibe al mundo y cada una de las pequeñas partes que lo constituyen, en un panorama donde ofertar nuevas formas creativas de trabajo está transformando a la industria como consecuencia del desarrollo de nuevas herramientas tecnológicas, como los chatbots.

Los *chatbots* son programas que interactúan con usuarios empleando un lenguaje natural con el propósito de simular una conversación humana (Herrero-Diz & Varona-Aramburu, 2018).

Aunque también son conocidos como herramientas construidas sobre algoritmos de aprendizaje, con la capacidad de redactar y programar respuestas que se adaptan al contenido de interés del usuario fortaleciendo la eficiencia operativa dentro de los servicios de atención. Con base en su funcionalidad los *chatbots* se clasifican en: *chatbots* basados en reglas, *chatbots* conversacionales y *chatbots* contextuales.

Un *chatbot* permite capturar la conversación para hacer su análisis o su exploración cognitiva y afectiva, y para ver las percepciones de los usuarios sobre un tema, interacción, situación o contexto concretos (García Brustenga et al., 2018). En los sistemas *Help Desk*, el tipo de chatbot que mejor se adapta a las necesidades de la empresa dentro del marco de seguimiento de atención y gestión de incidencias es el *chatbot* contextual, ya que, como se detalla posteriormente, brinda atención personalizada a través de asistentes virtuales basados en el procesamiento de lenguaje natural, lo que permite generar una interacción más auténtica con el usuario.

2.6.1. Chatbots basados en reglas

Este tipo de *chatbots* está desarrollado sobre una base de instrucciones predefinida, ya que, mediante la interacción con el usuario, este debe limitarse a ciertos parámetros durante la conversación, debido a que no tienen la capacidad de pensar o tomar decisiones propias, como lo haría un modelo de inteligencia artificial avanzado.

Suelen ser útiles en tareas sencillas y cíclicas, y funcionan muy bien con palabras clave; sin embargo, carecen de un flujo de trabajo dinámico. Son *chatbots* ideales para responder a consultas comunes y son aplicables a la tarea de soporte al cliente (Pola & Sheela, 2021).

2.6.2. Chatbots conversacionales

Son programas que integran inteligencia artificial y están diseñados para mantener una interacción con el usuario a través del lenguaje natural, con el fin de establecer un entorno en el que el contexto de la conversación siga una línea de aprendizaje. Aunque su creación no es reciente, han adquirido cierta popularidad en aplicaciones de mensajería como WhatsApp o Telegram, y cada vez más empresas buscan integrarlos en sus sistemas de soporte técnico.

En entornos corporativos los *bots* conversacionales ayudan a mejorar la productividad, así como la atención a los diferentes requerimientos que puedan presentarse en el área de TI, mediante el uso de asistentes virtuales con la finalidad de brindar un correcto acompañamiento al usuario durante la resolución del problema.

2.6.3. Chatbots contextuales

Son sistemas conversacionales que siguen una línea de aprendizaje; esto quiere decir que recopilan información de interacciones pasadas con el usuario para mejorar el tiempo y la coherencia de las respuestas. Del mismo modo maneja una mejor interpretación de la intención del usuario, no limitándose únicamente al contexto de la conversación, sino también adaptándose al tema, situación o necesidad de este.

No suelen funcionar bajo líneas de comando predefinidas, como ocurre con los *chatbots* basados en reglas, sino más bien hacen un análisis más profundo de la conversación y responden según el contexto presentado.

2.7. Large Language Models (LLM)

Los LLM son modelos de inteligencia artificial entrenados para percibir, interpretar y generar lenguaje natural. Debido a que utilizan técnicas de análisis y producción de texto, pueden desempeñar diversas tareas en el ámbito de la lingüística, como resumir textos, redactar correos o ensayos y programar código, entre otras.

Por ello, estos modelos deben ser entrenados con grandes volúmenes de datos para detectar patrones y establecer relaciones en el lenguaje. Gracias a esto, pueden formular respuestas más coherentes y acordes con el contexto de las entradas de información.

Antes de que se acuñara el término de LLM, existieron varios intentos de generar texto automáticamente, los cuales se construyeron sobre arquitecturas de redes neuronales recurrentes, con variantes en *Long Short-Term Memory* (LSTM) y las *Gated Recurrent Units* (GRU) (Dávila Romero, 2025).

2.7.1. Modelos LLM actuales

Con el surgimiento de la arquitectura *transformer*, muchas empresas han tratado de entrenar sus propios modelos con la intención de integrarlos como servicios de paga o, simplemente, liberarlos como modelos de código abierto (*open source*). Entre los más destacados se encuentran:

- **GPT:** Por sus siglas en inglés, *Generative Pre-trained Transformer*, se utiliza como una herramienta que permite una comunicación más fluida, adaptada a las capacidades y necesidades de cada usuario. En consecuencia, es posible diseñar y desarrollar interfaces que integren diversas funcionalidades, acorde con los parámetros establecidos por el cliente.
- **Gemini:** Es un modelo desarrollado por Google. Este modelo destaca por su amplia ventana de contexto (un concepto que representa la "memoria" del modelo). Al igual que GPT, es de pago, y de hecho es su principal competidor (Romero, 2025).
- **Claude:** Es un modelo desarrollado por *Anthropic* y de gran interés para los científicos de datos. Vio la luz por primera vez en octubre del 2024 y, su versión Claude lidera actualmente la mayoría de los puntos de referencia, convirtiéndolo en el modelo LLM más sólido. Su principal fortaleza radica en el análisis de información extensa, lo que resulta útil en tareas de síntesis a gran escala.
- **Llama:** Es un modelo LLM de código abierto (*open source*) creado por Meta, diseñado para generar respuestas de forma similar a GPT, aunque con un número limitado de instrucciones según el área de conocimiento y con sesgos que pueden afectar a la exactitud de la información devuelta. Aun así, su versatilidad lo convierte en una herramienta capaz de igualar funciones de modelos más sólidos y conocidos.

Para este caso de estudio, el modelo que mejor se ajusta al desarrollo del prototipo de *Help Desk* es GPT en su versión mini, ya que permite realizar consultas rápidas, automatizar tareas repetitivas, como la clasificación de tickets, y brindar apoyo al personal técnico con base en parámetros predefinidos en la documentación, la cual sirve como guía de conocimiento para dar respuestas más coherentes, entre otras funciones.

2.8. RAG

Es una técnica cuyo funcionamiento se enfoca en la adición de flujos de trabajo a modelos de alto aprendizaje, con el fin de mejorar sus respuestas mediante la modificación del contexto con información relevante que guarde algún tipo de relación con la pregunta provista. Por lo general, RAG trabaja a la par con bases de datos vectoriales capaces de extraer fragmentos de información provenientes de documentos, necesarios para responder a la pregunta del usuario.

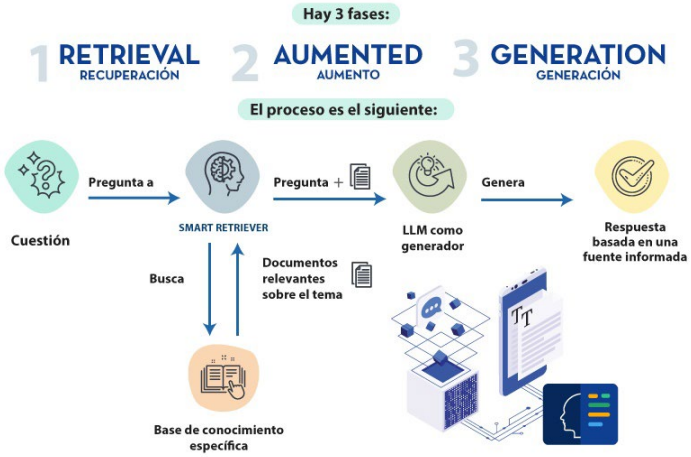


Figura 3 Proceso de Recuperación-Aumento-Generación (RAG).

Nota. Tomado de *RAG - Recuperación Aumento Generación: La llave que abre la puerta de la precisión a los modelos del lenguaje*, por A. Alija, 2024, datos.gob.es.

En la figura 3 se explica el funcionamiento de RAG a través de una estructura básica ya que durante la última década se ha promovido su uso debido a la capacidad de sus modelos de recuperación para buscar información útil y de sus modelos generativos para proporcionar respuestas coherentes a las solicitudes del usuario, a partir de datos previamente recuperados y enriquecidos. Esta característica constituye un factor clave en el desarrollo de proyectos, aunque su eficacia depende esencialmente del volumen y la calidad de los datos provistos.

En campos como la medicina, RAG se ha utilizado para apoyar diagnósticos clínicos, generar resúmenes de literatura científica y desarrollar *chatbots* capaces de responder consultas médicas con precisión (Miao et al., 2024). En el ámbito educativo, ha demostrado ser una herramienta muy útil y práctica, con aplicaciones en la generación de contenidos didácticos, incluidos ejercicios y sus soluciones, adaptados a necesidades específicas (Murtaza et al., 2022; Rivero Zazo, 2024).

El enfoque de RAG como técnica de mejora de modelos LLM, a través de la integración de conocimiento procedente de fuentes externas, posee un amplio alcance en distintos contextos de aplicación.

Su fuerte radica en que contribuye a reducir las alucinaciones de los modelos, lo que se traduce en fuentes de información más confiables y verificables. Dentro del entorno de desarrollo de un *Help Desk*, permite que el *chatbot* ejecute consultas con base en una documentación más detallada y contextualizada. Por ende, los tiempos de respuesta se reducen considerablemente, generando una impresión positiva en los usuarios y reduciendo la carga operativa del equipo de trabajo.

2.8.1. Limitaciones y fortalezas

Esta técnica, conocida como RAG, se apoya enormemente en el funcionamiento mediante *prompting* con el que cuentan los LLM (Cohen Solano, 2023). Gracias a ello, es posible dotar al modelo del contexto necesario para que pueda responder específicamente en situaciones que se ajusten a la información previamente incorporada a través de *queries* "aumentadas". Entre sus múltiples fortalezas se destacan:

- Respuestas más precisas y contextualizadas en el marco de aprendizaje del modelo LLM, ya que no se limita únicamente a ser entrenado con información base, sino que aborda un enfoque orientado a la expansión del conocimiento, lo que le permite realizar una búsqueda más especializada y apoyados en datos de mayor confiabilidad.
- Su uso es altamente escalable y eficiente, ya que, a partir de fragmentos de información, puede construir sólidas bases de conocimientos mediante procesos de entrenamiento óptimos.
- Implementa mejoras al comportamiento de los *chatbots* para que recopilen información más actualizada y se adapten a las bases de conocimiento del aplicativo.

  En lo que respecta a sus limitaciones se puede detallar lo siguiente:

- No pueden formular respuestas sin una base de conocimiento que contenga información exacta, actualizada y relevante. Si la documentación está incompleta o carece de coherencia, pues el resultado obtenido también se verá afectado.

- Aunque RAG trata de reducir tanto los sesgos como la información inventada no lo hace en su totalidad. Por consiguiente, el modelo puede cometer errores al interpretar el contexto de la pregunta y generar información basada en fuentes no verificables.
- Debido a su compleja implementación, se convierte en una herramienta difícil de usar, ya que requiere de un motor de búsqueda, procesos de indexación, documentación sólida y la integración de una base de datos vectorial.

En términos generales, el uso de la técnica RAG permite desarrollar aplicaciones con costos bajos de implementación ofreciendo mejores resultados y convirtiéndolo en un recurso esencial en la industria.

2.8.2. Funcionamiento en sistemas de soporte Help Desk

La aplicación de RAG en sistemas similares a *Help Desk* cuenta con algunos casos reales. Por ejemplo, *Jira Service Management* emplea agentes virtuales para responder solicitudes, que en caso de ser necesario son derivadas a un agente humano.

Otro caso es *ServiceNow*, cuya arquitectura basada en RAG permite recuperar artículos con información relevante y contextualizar las respuestas a partir de información empresarial.

Del mismo modo, Intercom implementa este enfoque en su motor Fin AI, con el fin de proporcionar información más confiable cuyas respuestas están sustentadas en artículos, revistas, informes o cualquier otro medio electrónico que le sea de ayuda.

2.9. Bases de datos vectoriales

Ya que se necesita una base de datos para almacenar la información o conocimiento recopilado cuyo acceso no debe delimitarse a procesos de vectorización por cada petición aplicada como son las bases de datos tradicionales que no tienen la capacidad de manejar datos no estructurados (Hernández Suárez, 2024).

En consecuencia, surgen las bases de datos vectoriales. Aunque se han estudiado a fondo los cambios de algoritmos en aplicaciones con grandes volúmenes de datos, especialmente en modelos avanzados de aprendizaje, estos han modificado los requisitos de almacenamiento.

Estos son grandes repositorios de almacenamiento, indexación y búsqueda de vectores con alta dimensionalidad, es decir, son representaciones numéricas de datos textuales e imágenes (Hernández Suárez, 2024). Su funcionamiento es crucial en la transformación de datos en *embeddings* vectoriales, como lo refleja la búsqueda semántica. Esto se traduce en resultados de búsqueda más precisos y rápidos, mejorando la fiabilidad de los resultados.

2.9.1. Estructura y funcionamiento

Los gestores de bases de datos ahora deben ofrecer procesamiento de consultas seguro, confiable y escalable para datos no estructurados.

Según la revista VLDB, los sistemas gestores de bases de datos vectoriales enfrentaban retos de ambigüedad semántica, tamaño de los vectores, altos costes de la similitud semántica y la falta de una estructura de indexación. La superación de estos retos ha dado lugar a nuevos enfoques para el procesamiento, almacenamiento e indexación de consultas, así como para la optimización y ejecución de estas (Pan et al., 2024).

Y esto, en un sistema Help Desk, podría ser muy útil, ya que recuperar información por similitud semántica ayudaría a procesos de búsqueda más eficientes para el usuario cuyas consultas se formulan de manera diferente. Un error muy común dentro de estos entornos sería un “error al iniciar sesión”, que puede ser reescrito de varias formas por distintos usuarios, como, por ejemplo: “Outlook no abre”, etc.

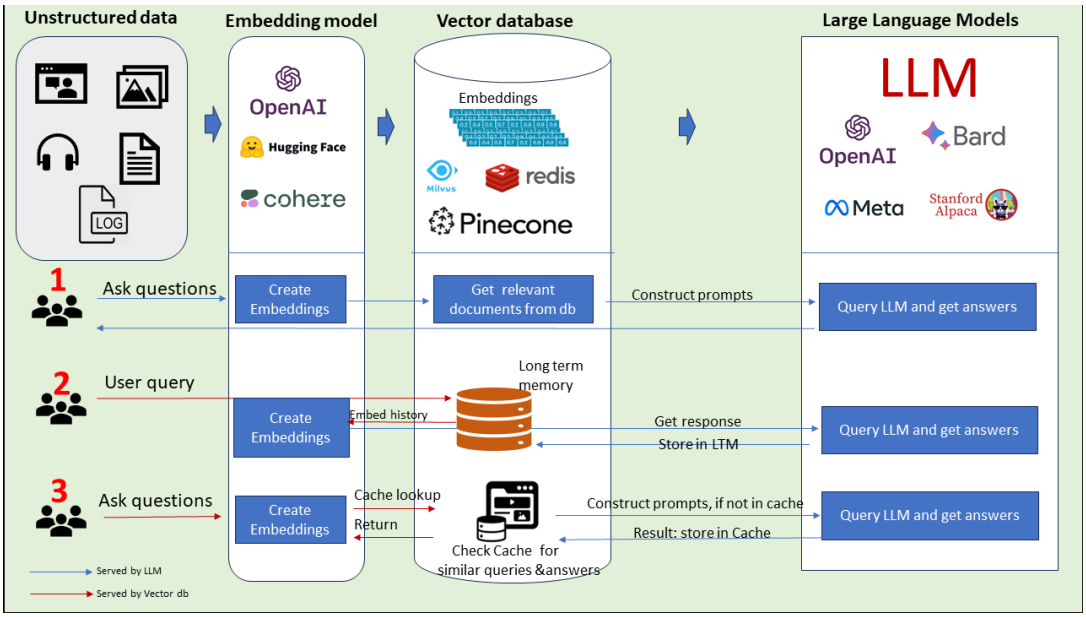


Figura 4 Estructura y funcionamiento de una base de datos vectorial.

Nota. Adaptado de *Bases de Datos Vectoriales: Teoría y Práctica con ChromaDB y LangChain,* por A. Berrospi Casano, 2024, Medium.

En lo referente a la figura 4 en esta se detalla, de manera general, el funcionamiento de una base de datos vectorial, ya que necesita de una entrada de datos, como imágenes, documentos y audios para ejecutar sus respectivos procesos de almacenamiento. Con la finalidad de convertir dichos datos en vectores del tipo numérico (lista de números) mediante el uso de un modelo *embedding*. El resultado debe ser leído e interpretado por el LLM para dar una respuesta coherente y adaptada al contexto de la consulta del usuario.

Entonces, las bases de datos vectoriales tienen el deber de dar sentido a esas consultas que pueden estar conectadas a un mismo problema, a pesar de que no se use la misma terminología.

Según la documentación técnica, el sistema funciona conectado a una base de conocimiento que facilita la búsqueda y generación automática de respuestas. Dependiendo de la intención de la consulta, puede escalarla a un agente humano.

Lo que resulta muy útil en sistemas *Help Desk* ya que almacenar los manuales, procedimientos o preguntas frecuentes en una documentación con una base sólida de

conocimiento permite no solo buscar por coincidencia de palabras, sino más bien por el contenido relacionado con la pregunta.

2.9.2. Sistema basado en búsqueda semántica

Para definir el concepto de búsqueda semántica, es necesario comprender antes qué es un sistema de búsqueda de información en el que se realiza una consulta sobre una base de documentos, ya sean estos relativamente cortos o extensos, pero con la finalidad de dar con una solución que responda a lo solicitado por el usuario.

La idea general de un sistema de búsqueda es que la consulta sea dirigida al sistema y que este, posteriormente, analice los diferentes documentos almacenados en la base de datos para encontrar posibles coincidencias que se ajusten a la pregunta planteada.

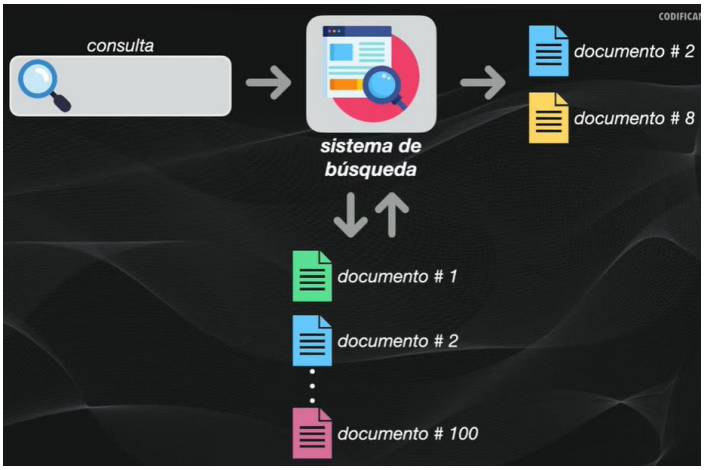


Figura 5 Estructura y funcionamiento de un sistema de búsqueda.

Nota. Captura de pantalla de *¿Qué es la búsqueda semántica?,* por Codificando Bits, s. f., en YouTube.

A través de la figura 5 se representa de forma gráfica el funcionamiento de un sistema de búsqueda. Partiendo de una base compuesta por 100 documentos, el sistema de búsqueda retorna la información de uno o dos de esos textos, en los que probablemente se pueda encontrar la respuesta.

La implementación de estos sistemas es simple, ya que se lo hace a través de un método de búsqueda por palabras clave. Sin embargo, no resulta del todo útil para problemas de la vida real, ya que basa su respuesta en el mayor número de coincidencias. Por ende, en una pregunta más específica y mejor formulada, tiende a devolver información errónea.

En otras palabras, no logran entender el contexto de la pregunta, ni mucho menos el de los documentos, para devolver una respuesta adecuada; y, por lo tanto, estos sistemas no funcionan en la práctica.

Es aquí donde los sistemas de búsqueda semántica superan este tipo de limitación, ya que no basan su funcionamiento únicamente en el análisis de las palabras individuales, sino en el significado completo de los textos, tanto de la consulta como de los documentos contenidos en una base de datos.

El eje central en un sistema basado en búsqueda semántica es un modelo de lenguaje, LLM. Funciona de la misma manera que un sistema de búsqueda tradicional con la diferencia de que el cerebro del sistema va a ser el LLM en cuestión.

Por ende, la búsqueda semántica puede entenderse como una aplicación del procesamiento del lenguaje natural que permite encontrar y recuperar documentos existentes que respondan a una búsqueda con base en su significado. Es decir, que tanto el significado de la consulta como el de los documentos se analizan mediante un método de extracción de información apoyado en un gran modelo de lenguaje para retornar los documentos con la información más confiable y verificable.

A partir de la figura 6 se puede observar, de forma más general, el funcionamiento previamente descrito de un sistema basado en búsqueda semántica.

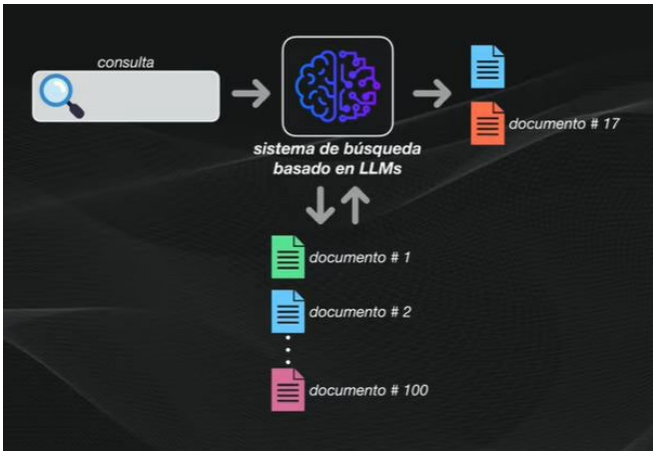


Figura 6 Estructura y funcionamiento de un sistema de búsqueda basado en LLMs.

Nota. Captura de pantalla de *¿Qué es la búsqueda semántica?,* por Codificando **Bits**, s. f., en YouTube.

A continuación, se detalla paso a paso el proceso de comparación que realiza un sistema que emplea búsqueda semántica, a través de la figura 7:

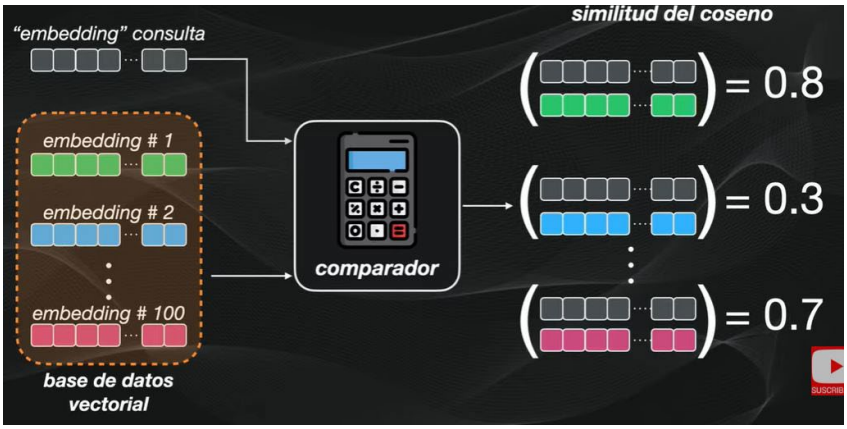


Figura 7 Estructura y funcionamiento de la comparación de la consulta vs documentos.

Nota. Captura de pantalla de *¿Qué es la búsqueda semántica?,* por Codificando Bits, s. f., en YouTube.

1. Codifica los documentos como embeddings y almacena su información en una base de datos vectorial.
2. Codifica la consulta, lo que significa que el usuario debe digitar la información correspondiente a su pregunta o solicitud.
3. Busca los documentos que semánticamente respondan a esa consulta y que tengan significados asociados a ella. En este proceso de búsqueda se utiliza la operación matemática denominada "similitud del coseno" que permite comparar vectores, es decir, los embeddings y los devuelve en números de entre 0 y 1.

Cuanto más cercano se encuentre de 0, más diferentes serán los *embeddings* del documento y de la consulta; mientras que, cuanto más cercano se encuentre de 1, más similares serán semánticamente dichos *embeddings*.

En definitiva, los resultados que más se acercan al 1 son los que contendrán la información más adecuada para dar solución al requerimiento del usuario. Por esta razón, su uso es bastante recurrente en sistemas *Help Desk* que buscan integrar algoritmos basados en inteligencia artificial.

2.10. Tecnologías y herramientas de desarrollo del prototipo

A continuación, se enumerarán una serie de herramientas tecnológicas que intervendrán en el diseño y desarrollo del prototipo del *Help Desk*, entre las cuales se encuentran: LangChain, LangGraph, Streamlit, FastAPI.

2.10.1. LangChain

Dentro del campo de la IA generativa es un *framework open source* que permite la experimentación con modelos de lenguaje externos con el uso de claves API privadas (García Barrena & Borrego Díaz, 2024). Incluye métodos y herramientas con librerías de OpenAI para configurar agentes o IA generativa, permitiendo crear texto mediante operaciones sucesivas.

Su importancia radica en sus agentes ya que estos pueden tomar decisiones de recuperación de información además de conceder permisos para el uso de herramientas externas con el único fin de dar respuestas más contextualizadas y alineadas al campo de estudio recabando datos a partir de documentación previamente definida.

La memoria de LangChain permite conservar datos de interacciones anteriores, lo que mejora la continuidad del servicio. Esto resulta beneficioso durante el desarrollo de sistemas *Help Desk*, ya que contribuye a integrar modelos y una arquitectura capaz de gestionar asistentes personalizados.

2.10.2. LangGraph

LangGraph es un framework construido sobre LangChain que introduce una forma de modelar los flujos de trabajo de los agentes a partir de grafos (Fernández Zarza, 2026). Este enfoque ofrece un alto grado de control mediante un sistema de estados compartidos que avanza por un grafo, generando resultados intermedios en contextos válidos para ejecutar la siguiente acción. Así, el desarrollador define qué pasos se ejecutan, su orden y condiciones.

Además, LangGraph se encarga de simplificar la creación y la gestión de agentes de IA, los cuales son componentes modulares diseñados para ejecutar tareas específicas de traducción. A su vez aprovechan las potentes capacidades semánticas de los grandes modelos lingüísticos (LLM), como GPT-4o, para garantizar traducciones precisas y contextualmente relevantes, manteniendo la escalabilidad y la retención del contexto (Wang & Duan, 2024).

La función primordial de LangGraph consiste en modelar diversos procesos, tales como *workflows* o agentes que integran ejecución durable, persistencia y *human in the loop*. Por ello, en un sistema de *Help Desk* basado en IA, se puede emplear para administrar el flujo de trabajo, lo que resultaría de gran ayuda para mantener el contexto, pausar procesos y retomar conversaciones.

Aunque LangGraph tiene una presencia limitada en sistemas de *Help Desk* dentro del ámbito de TI, sí existen opciones como *customer support*, *customer service* o *call center* que siguen una misma lógica de atención, recuperación de información, enrutamiento y escalado.

La IA *Assistant* de Klarna, desarrollada con LangGraph y LangSmith, ofrece soporte en pagos, reembolsos y derivaciones. Logró reducir el tiempo de respuesta a las consultas de sus usuarios. Este es el caso más parecido a lo que LangGraph puede ofrecer a sistemas de soporte y ayuda como lo es un *Help Desk*.

2.10.3. Streamlit

Streamlit, bajo un contexto de diseño, ayuda a aliviar la carga de trabajo, centrando los esfuerzos en la funcionalidad del aplicativo. Además, es una librería de Python que facilita la programación de aplicaciones web, sintetizando las líneas de código y ahorrando tiempo en tareas destinadas al *backend* o en el uso de otros lenguajes de programación como Angular, HTML o CSS.

Este framework interviene principalmente en el diseño de la interfaz gráfica del sistema *Help Desk* y lo representa como un aplicativo web interactivo en el que se emplea un número limitado de líneas de código. Además, sus funcionalidades encajan perfectamente con los requerimientos de este trabajo de titulación, ya que permite emplear formularios de tickets, paneles de seguimiento y da soporte a la gestión documental.

2.10.4. FastAPI

FastAPI es otro framework de Python utilizado para la construcción de APIs de una manera rápida y fácil de programar (Martín Hernández, 2022). Garantiza el funcionamiento óptimo del backend y de las respectivas respuestas dadas bajo ciertas peticiones del usuario. Su empleabilidad ha escalado niveles de confianza bastante altos debido a su facilidad en la creación de APIs de código reducido y su velocidad al momento de ejecutar pruebas durante el desarrollo.

Su funcionamiento está estrechamente vinculado con el *backend* del *Help Desk*, lo que implica la creación de APIs encargadas de gestionar toda la logística del aplicativo web. Entre sus tareas se incluyen el registro de tickets, la consulta de incidencias, la autenticación de usuarios y la conexión del chatbot para enviar respuestas al *frontend*.

CAPÍTULO III: DISEÑO DEL PROTOTIPO

Este capítulo presenta tanto el diseño de la interfaz gráfica de usuario como el de la base de datos vectorial, en la que se incorpora un panel de control que detalla el flujo del sistema, el número de tickets activos y el estado de RAG. Para su desarrollo y funcionamiento no se emplea el uso de herramientas como HTML, JavaScript y CSS, sino que se parte de una premisa que ha revolucionado el diseño de aplicaciones web: Streamlit, una herramienta en la que no existe una separación entre la lógica de *backend* y *frontend*, ni tampoco archivos de configuración complejos o servidores que montar.

Simplemente basta con escribir unas pocas líneas de código que empiezan con st. en Python para dotar de funcionalidad a la aplicación. El corazón de Streamlit es su modelo de ejecución reactiva, por lo que, en cada interacción realizada por el usuario, este ejecuta un script de arriba hacia abajo para recalcular toda la interfaz basándose en un nuevo estado.

Además, que, con su biblioteca de componentes preconstruidos, no será necesario diseñar burbujas de *chat*, manejar *layouts* complejos o crear campos de entrada personalizados para el *chatbot* sino poder iterar rápido y enfocarse en la lógica de la conversación. Streamlit traduce las intenciones de diseño en una interfaz clara, funcional sin escalabilidad extrema ni arquitecturas complejas para el usuario.

A lo largo de este capítulo también se describe la arquitectura del sistema y los diferentes *frameworks* que garantizan el correcto funcionamiento del prototipo de *Help Desk*. Del mismo modo se estructura cada uno de los módulos operativos que la componen, como el módulo de gestión de tickets y el módulo de *chatbot* integrado con inteligencia artificial, entre otros.

3.1. Arquitectura del sistema

El sistema *Help Desk* propuesto para un entorno empresarial hipotético encaja perfectamente en una arquitectura modular del tipo cliente-servidor, organizada por capas, ya que está dividido en módulos, en la que cada uno cumplen una función distinta pero que se complementan en su trabajo. Esta estructura favorece a la operatividad del sistema mediante la gestión de tickets, recuperación de información mediante RAG, almacenamiento de datos, ejecución de la lógica del negocio y la presentación de una interfaz gráfica funcional para la correcta interacción con el usuario.

Por ende, en este tipo de arquitectura se pueden identificar dos posiciones: el cliente, que viene a ser el usuario final que accede al aplicativo mediante un navegador web, y el servidor, donde se encuentran almacenando todos los datos, reglas y lógicas de la aplicación (Carlos Miguel Jaramillo Castro, 2016).

El diseño de este tipo arquitectura puede resultar útil para el desarrollo de aplicaciones de red, así como de aplicaciones que constan de una interfaz gráfica (GUI), por lo general los programas se almacenan del lado del usuario, y la lógica del negocio del lado del servidor como resultado permite validar los datos del usuario en el cliente. En definitiva, elegir entre una o dos capas depende de las necesidades de la aplicación y más si se quiere englobar todas las funciones en un solo proceso o descentralizarlo (Acosta Gonzaga et al., 2006).

3.1.1. Diseño de la arquitectura del sistema Help Desk

A través de la figura 8 se detalla capa por capa el funcionamiento de la arquitectura del prototipo de *Help Desk*:



Figura 8 Arquitectura organizada por capas del prototipo Help Desk. Elaboración propia.

A pesar de que en la figura 8 no se incluye la capa del usuario, se sobreentiende que su participación es esencial en el desarrollo de este sistema, ya que es quien lleva a cabo las

solicitudes o consultas del soporte. Aunque no interviene directamente en los procesos de gestión de información, ni mucho menos en la lógica de programación, su interacción con el sistema determina qué tipo de acciones puede llevar a cabo. Además, en esta capa no solamente participa el usuario como tal, sino que también es necesario mencionar el técnico o especialista del área.

Al tratarse de un entorno simulado, el rol de especialista será desempeñado por cualquier persona que mantenga algún tipo de relación con el área de tecnologías de la información o afines. Entre sus principales tareas se destacan la revisión de los tickets asignados y el apoyo tanto en la documentación técnica como en las sugerencias provistas por la inteligencia artificial para responder dudas o consultas de carácter general por parte del usuario.

Una vez dictaminadas las razones por las que no se incluyó la capa del usuario en la figura 8 se tomará como punto de partida la capa de presentación.

- **Capa de presentación:** Como se puede apreciar en la figura 8, en esta capa se incluyen componentes básicos de la interfaz gráfica del usuario, los cuales determinan el tipo de solicitud que se va a realizar, así como todo el proceso de gestión de tickets, que abarca desde la revisión de la incidencia hasta su respectiva resolución. En otras palabras, se encarga de recabar datos y mostrar resultados a través de interfaces gráficas dinámicas, cuya función no se centra en procesos complejos del sistema.
- **Capa de la lógica de negocio:** En esta capa se recepta toda aquella información proveniente de la interfaz, además de determinar el flujo de trabajo del sistema como se puede observar en la figura 8. Lo más destacable es la conexión entre el *frontend*, el backend con la capa de inteligencia artificial. Asimismo, aquí también se llevan a cabo procesos de validación de datos, asignación de incidencias y seguimiento de su ciclo de vida.
- **Capa de inteligencia artificial:** Aquí se encuentra el cerebro del sistema y, mediante técnicas de razonamiento, se facilitan los procesos de interpretación de la consulta, así como de la búsqueda por similitud y coincidencia de palabras, con la finalidad de encontrar información más relevante que ayude a generar respuestas más contextualizadas. En esta capa se retoman conceptos como LangChain, LangGraph,

RAG, el modelo GPT y el procesamiento de lenguaje natural para comprender procesos de tratamiento y de gestión de datos como se detalla en la figura 8.

- **Capa de almacenamiento de datos:** En esta capa se abarcan todos aquellos métodos que permiten enviar y almacenar información mediante un gestor de base de datos. Constituye la base de conocimiento del sistema, la cual, a su vez, se representa en documentación técnica que contiene datos relevantes acerca de procedimientos, preguntas frecuentes o soluciones previas del área. Además, almacena tanto los embeddings de los documentos como la asignación y el estado de los tickets. Todo se representa en la figura 8.

3.2. Diseño de la base de datos

Para el diseño de la base de datos relacional del prototipo de *Help Desk*, se plantea un modelo de entidad-relación el cual funciona de manera asincrónica con una base de datos vectorial. Aunque se trata de dos conceptos que difieren mucho en su funcionalidad, la realidad es que en la práctica mantienen una relación complementaria.

Mientras la base de datos relacional se encarga de administrar toda la información estructural y transaccional del sistema, como usuarios, tickets, seguimientos e historiales de navegación, se propone el uso de PostgreSQL. Por su parte la base de datos vectorial almacena los embeddings o vectores para que, por medio de RAG, se pueda recuperar la información y ejecutar procesos de búsqueda semántica; para ello, se propone el uso de ChromaDB.

Su vínculo se establece a través de un registro documental y de una tabla de sincronización vectorial, en la que se puede relacionar la documentación a través de una indexación en la base vectorial. En otras palabras, el *backend* es el puente de unión entre estas dos, más específicamente en la capa de la lógica del negocio donde se comparten los metadatos, se asignan los identificadores a los documentos y se generan las tablas de indexación.

3.2.1. Diseño de modelo entidad-relación

Como se puede observar en el diagrama de la figura 9, este se encuentra constituido por 12 entidades más una entidad que funge como núcleo central. Entre las que se pueden nombrar están: la base de conocimiento, los fragmentos del documento, la recuperación de información mediante RAG, la consulta RAG, el usuario, los embeddings y las asignaciones de ticket. Del mismo modo, destaca la entidad ticket, por ser el eje central del funcionamiento del sistema.

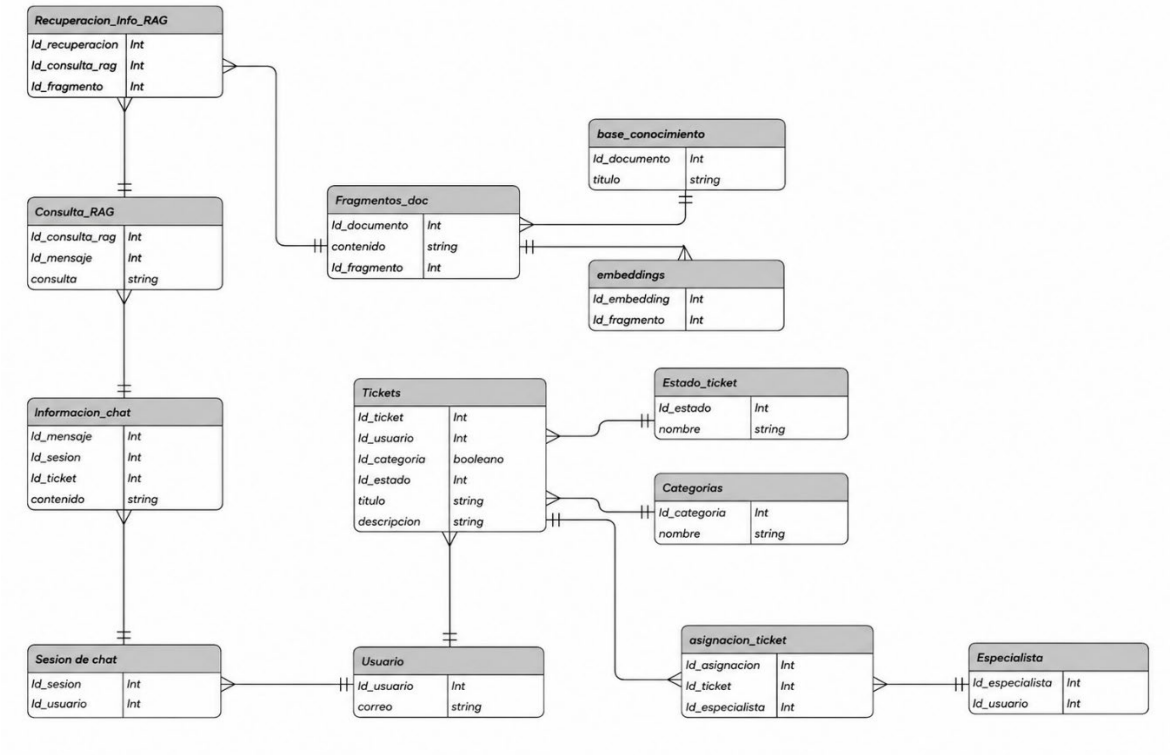


Figura 9 Modelo Entidad-Relación del prototipo Help Desk. Elaboración propia.

- **La entidad usuario:** Almacena la información relacionada con los usuarios del sistema. Representa a todos los individuos que interactúan con el aplicativo web. Además, ayuda a identificar al sujeto de la consulta, es decir, a quien genera el ticket e inicia una sesión de chat.
- **La entidad especialista:** Representa al técnico del área o el encargado de atender las diferentes incidencias provenientes del usuario, y mantiene una estrecha relación con la entidad usuario.
- **La entidad categoría:** Sirve para clasificar el tipo de ayuda que va a recibir el usuario con base en el ticket generado. Determina si la asistencia de soporte técnico debe ser

escalada a un especialista o si el propio sistema puede resolverla a través de un modelo LLM.

- **La entidad estado_ticket:** Representa la situación actual de la incidencia dentro del flujo de trabajo del sistema. Permite identificar los posibles estados de un ticket, como pendiente o resuelto.
- **La entidad ticket:** Constituye el eje central del sistema, cuya función es registrar cada petición o solicitud hecha por el usuario. Aquí se encuentra contenida la información relacionada con el problema reportado y, además, establece vínculos con otras entidades para volver más dinámico al proceso de atención.
- **La entidad asignación_ticket:** Su tarea consiste en registrar el ticket y dirigirlo a un especialista con base en la complejidad del requerimiento. Su objetivo es que cada incidencia esté asociada a un técnico o responsable para su posterior resolución.
- **La entidad sesión_chat:** Representa la primera interacción que el usuario realiza con el sistema, en la que reporta su problema al *chatbot* a través de un mensaje que posteriormente será procesado y devuelto.
- **La entidad información_chat:** Almacena el contenido del mensaje enviado desde la entidad sesión_chat el cual sirve como pauta para interpretar la consulta.
- **La entidad base_conocimiento:** Funciona como un repositorio de almacenamiento para toda aquella información relacionada con la documentación del sistema, la cual permite responder las consultas mediante RAG.
- **La entidad fragmentos_doc:** Contiene los separadores de la documentación. Esto permite estructurar al documento en bloques pequeños y manipulables para generar procesos de indexación, lo que a su vez facilita la búsqueda semántica.
- **La entidad embeddings:** Representa los vectores generados a partir del contenido del documento. Estos contienen los datos más relevantes asociados a la similitud con la consulta, es decir, aquella que más se aproxima a los resultados esperados por el usuario.
- **La entidad consulta_RAG:** Lleva un registro de todas las consultas procesadas. Además, recepta la información proveniente de la entidad información_chat para posteriormente activar un mecanismo de búsqueda que se ajuste a lo solicitado por el usuario con base en la documentación oficial del sistema.

- **La entidad recuperación_RAG:** Almacena los resultados obtenidos en la entidad consulta_RAG especificando cada fragmento relevante recuperado y que sirve como base de sustentación para la respuesta generada por el sistema.

3.2.2. Reglas de cardinalidad

Con relación al diagrama de la figura 9 la entidad ticket se relaciona con cuatro entidades: usuario, estado, categoría y asignación de ticket. Un usuario puede generar varios tickets, mediante el cual un ticket puede tener varias asignaciones que, dependiendo del caso, puede ser dirigido a un especialista que asume el rol de técnico del área. Hasta que no se dé una respuesta al requerimiento, los tickets pueden mantener un estado pendiente o resuelto y, según su categoría, pueden o no ser escalados a un técnico.

Además, un usuario puede mantener varias sesiones de chat, en el que puede digitar varios mensajes. Estos, a su vez, sirven de guía para que RAG recupere los datos más relevantes de cada consulta, para que posteriormente varios procesos de recuperación de información se apliquen a un fragmento de documento y estos sean convertidos en varios vectores. A su vez, los fragmentos del documento deben estar vinculados a una sola base de conocimiento, la cual servirá para sustentar las respuestas a los diferentes requerimientos planteados.

A partir del siguiente caso, se representa de forma gráfica las cardinalidades antes descritas, lo que sirve como complemento para entender, a un nivel más general, cómo funcionan estas relaciones entre entidades.

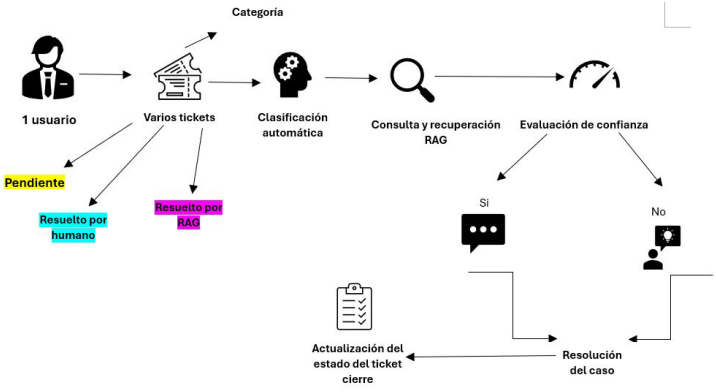


Figura 10 Flujo de trabajo del prototipo Help Desk: caso de estudio.

Elaboración propia.

Como punto de referencia, se toma la figura 10 para explicar más a detalle el funcionamiento del caso y por qué es importante comprender su flujo de trabajo.

- **Usuario – Ticket**
    1. Un usuario puede generar varios tickets, por ende, la cardinalidad es:
       1 usuario → N tickets
    2. Una categoría puede estar asociada a muchos tickets, pero un ticket solo puede pertenecer a una sola categoría, por ende, su cardinalidad es:
       1 ticket → 1 categoría 1 categoría → N tickets
    3. Un estado puede estar presente en varios tickets, pero un ticket solo puede permanecer en un solo estado, por ende, su cardinalidad es:
       1 estado → N tickets 1 ticket → 1 estado
    4. Un usuario puede entablar varias sesiones de chat, pero una sesión de chat solo puede estar sujeta a una sola consulta, por ende, su cardinalidad es:
       1 usuario → N sesiones de chat 1 sesión de chat → 1 consulta
    5. Un documento puede dividirse en varios fragmentos, pero un fragmento puede dividirse en varios *embeddings* o vectores, por ende, su cardinalidad es:
       1 documento → N fragmentos 1 fragmento → N vectores
    6. Una consulta RAG puede recuperar varios fragmentos, pero cada registro de recuperación está asociado únicamente a una consulta, por ende, su cardinalidad es:
       1 consulta RAG → N fragmentos N recuperación RAG → 1 consulta

Asimismo, se añade un componente de confiabilidad dentro del flujo de trabajo que dictamina si la consulta puede ser resuelta por el chatbot o si debe ser escalado al especialista del área.

3.3. Diseño de la interfaz gráfica

Para el diseño de la interfaz gráfica del prototipo *Help Desk* se empleará uno de los *frameworks* más prácticos de Python: Streamlit, el cual permite convertir pocas líneas de código en un aplicativo web interactivo y funcional, sin necesidad de integrar herramientas como JavaScript, CSS ni *frameworks frontend* complejos.

Además, al tratarse de una herramienta de código abierto (*open source*) su diseño ofrece una mayor escalabilidad para el desarrollo de aplicaciones de datos y de paneles de control orientados a la supervisión de procesos de aprendizaje automático.

Dicha interfaz permite visualizar el estado del ticket, así como el panel de control encargado de gestionar el flujo del sistema.

Como se trata de una versión reducida del *Help Desk* que emplean las grandes corporaciones, pues su diseño está limitado a simular un gestor de incidencias especializado en la atención de procesos relacionados con el área de TI como se puede ver representado en la figura 11 de entre todos los servicios que esta puede ofrecer. Está enfocado especialmente en brindar una interacción intuitiva y una navegación sencilla, pero clara, manteniendo la facilidad de uso para el usuario.



Figura 11 Funciones del departamento de TI.

Nota. Tomado de *Funciones del departamento de TI: Importancia, Funciones y Roles,* por A. Navarro, 2023, Sugeris.

A pesar de tener un diseño básico, la idea es comprender cómo se lleva a cabo el proceso de atención de solicitudes o problemáticas, así como sus respectivas soluciones, ya sea por parte de un especialista del área o de la propia inteligencia artificial.

El tipo de usuarios al que está orientado se compone esencialmente de los empleados de una empresa y de los especialistas o técnicos encargados de responder a los requerimientos.

3.3.1. Estructura general de la interfaz

La interfaz gráfica está estructurada por módulos con el fin de brindar una interacción más fluida con el sistema, sin incorporar apartados visuales complejos que puedan confundir al usuario. Además, como su diseño sirve como base para proyectos futuros, con el tiempo se podrán agregar más funcionalidades, así como reorganizar todos sus componentes iniciales para que se adapten a los nuevos esquemas de trabajo de las empresas.

Constará de dos apartados: uno destinado al usuario, en el que debe ingresar su correo electrónico y la descripción del problema para registrar su consulta mediante la creación de un ticket. Con esta información, el sistema ejecuta los procedimientos correspondientes para la atención de incidencias y resuelve los requerimientos por las dos alternativas previamente descritas.

El segundo apartado destinado para el especialista del área, en el que se receptan todos los tickets y se despliega un panel interactivo en el que se detalla la confianza y la justificación de la respuesta, las fuentes consultadas, la categoría, el contexto para el agente y un cuadro de texto para resolver la incidencia, solo se despliega en el caso en el que el *chatbot* conversacional no lo pueda solventar.

Sin embargo, si se trata de un caso que no requiere intervención de un técnico pues el *chatbot* conversacional se encarga de generar una respuesta automática, basado en su nivel de confianza, para responder al requerimiento en relación con la documentación oficial. Y de manera complementaria se visualiza el estado del ticket para dar seguimiento a la incidencia dentro del sistema.

A continuación, se presenta una interfaz gráfica en la que se describe la organización de cada componente por bloques:

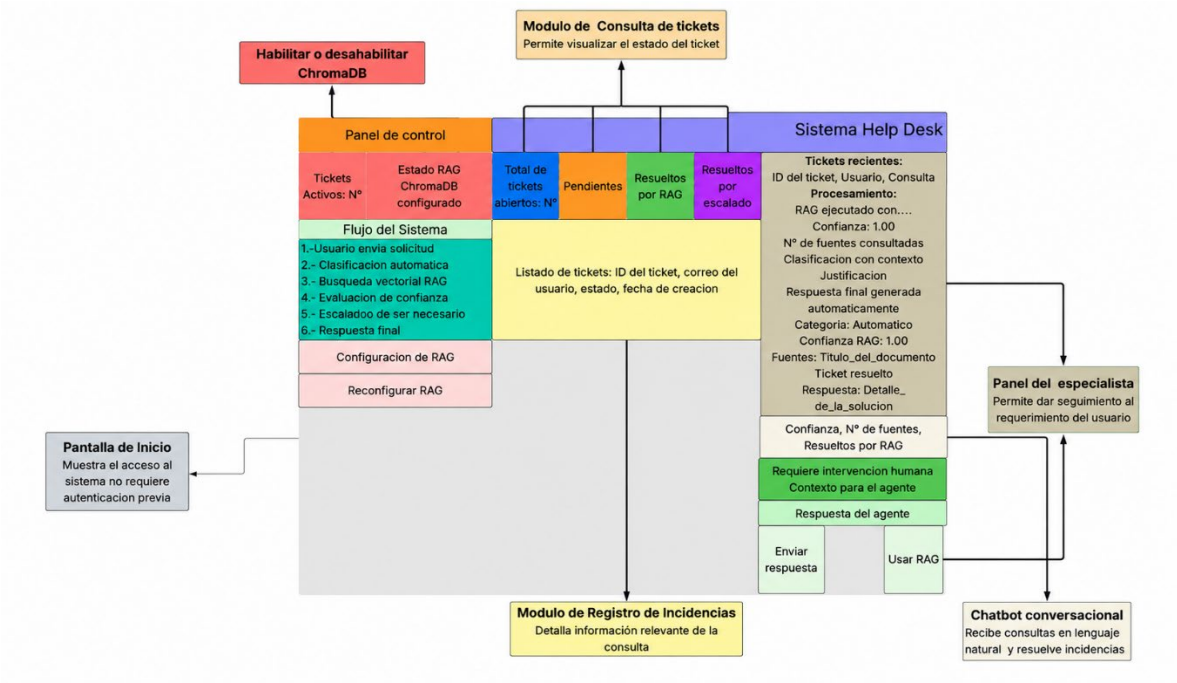


Figura 12 Apartado visual del especialista TI organizado por bloques. Elaboración propia.

Como se puede observar en la figura 12, esta detalla el apartado destinado al especialista o técnico del área, en el que podrá realizar un correcto seguimiento y posterior atención al número de requerimientos entrantes. En ella se describe cada módulo y su función dentro del sistema, además de especificarse el flujo en el que se ejecuta cada acción, lo cual ayuda a entender de manera más general el proceso de gestión de incidencias.

De igual forma se diseña un apartado visual destinado para el usuario en el que se describe cada componente de su estructura por bloques:

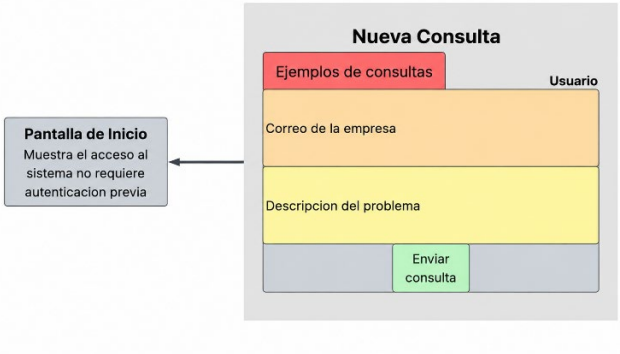


Figura 13 Apartado visual del usuario organizado por bloques. Elaboración propia.

Como se puede apreciar en la figura 13, esta describe dos bloques en los que se ingresan tanto el correo del usuario como la descripción de la consulta. Aunque parece una interfaz sencilla, sirve como ejemplo para el caso de estudio planteado. Además, consta de un botón desplegable con ejemplos de consultas o dudas más comunes que se le pueden presentar a un usuario.

3.3.2. Visualización de la interfaz gráfica

Una vez estructurada la interfaz gráfica, se exponen las correspondientes evidencias visuales, cuyo fin es mostrar de forma más clara toda la organización del sistema *Help Desk*, así como de todos los componentes o módulos que la conforman. Así justificando las estructuras expuestas en las figuras 11 y 12, lo que facilita una mejor comprensión de sus diseños en concordancia con los procesos de gestión de incidencias designados en el flujo de trabajo.

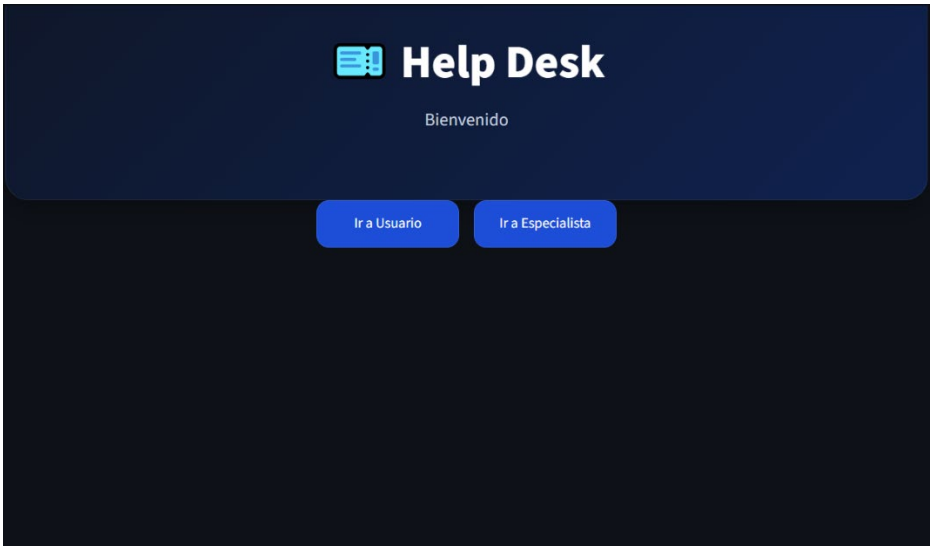


Figura 14 Página principal del sistema Help Desk. Elaboración propia.

La figura 14 presenta una página de inicio que no fue contemplada en el diseño preliminar del sistema; sin embargo, esta sirve de puente entre las pestañas del usuario y el especialista, para una mayor movilidad entre ambas. Consta de dos botones: uno para dirigirse a la ventana del usuario y el otro para dirigirse a la ventana del especialista. Aunque su diseño es bastante básico, no afecta directamente al funcionamiento del prototipo.



Figura 15 Diseño de interfaz gráfica del usuario (Prototipo Help Desk). Elaboración propia.

La figura 15 presenta la ventana de visualización del usuario, en la que se muestra un botón que despliega una serie de dudas o consultas más comunes dentro de un entorno de trabajo. Además, consta de dos campos de textos en los que el usuario ingresa el correo asociado a su empresa, así como la descripción del requerimiento que desea que se le atienda para posteriormente enviarlo y que sea receptado por el especialista del área. Asimismo, se integra un botón de "Regresar a la página principal", el cual no fue contemplado en el diseño original.

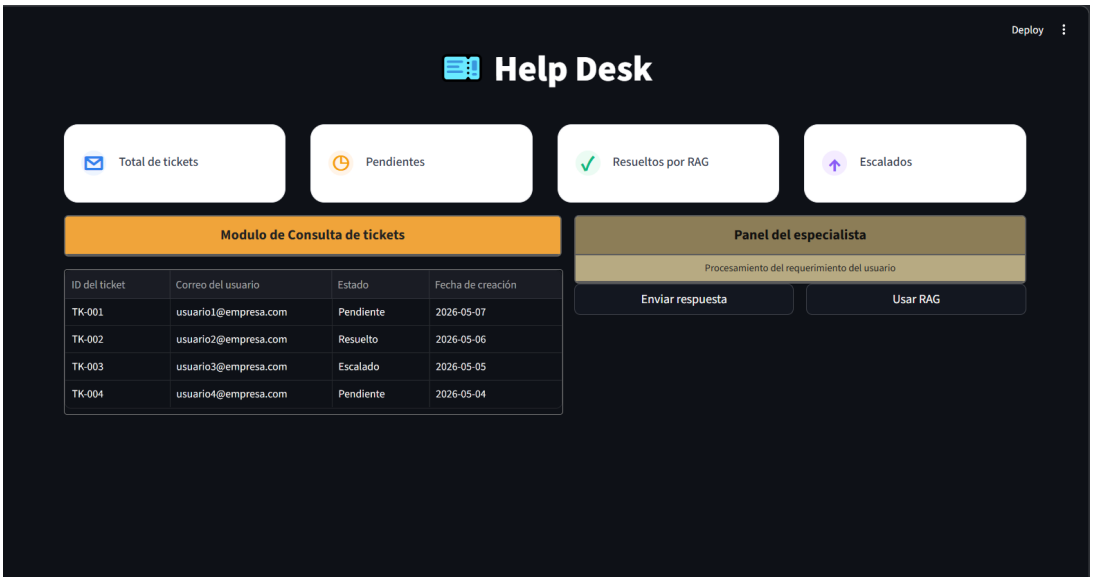


Figura 16 Diseño de interfaz gráfica del especialista (Prototipo Help Desk). Elaboración propia.

Esta figura 16 presenta la ventana de visualización del especialista, en la que se gestiona el procesamiento de la consulta. Esta se divide en tres módulos. El primero detalla el total de tickets, así como el estado de cada uno. Cabe resaltar que el cuadro denominado “Resueltos por RAG” corresponde a los requerimientos que son resueltos mediante el *chatbot* conversacional o, en este caso, por la IA.

El segundo módulo define una tabla que enlista los tickets generados durante todo el proceso de atención, lo que ayuda a llevar un registro más detallado de cada incidencia. Finalmente, el tercer módulo, y el más importante, es aquel en donde se lleva a cabo el procesamiento del ticket, en el que se dictamina si la consulta debe ser resuelta por la IA o, simplemente escalada al técnico de TI.

Para el caso del técnico del área, se despliega un campo de texto en el que este debe incluir la solución para posteriormente enviarla al correo del usuario. En cambio, cuando la consulta es resuelta por la IA simplemente envía automáticamente la respuesta al correo correspondiente.

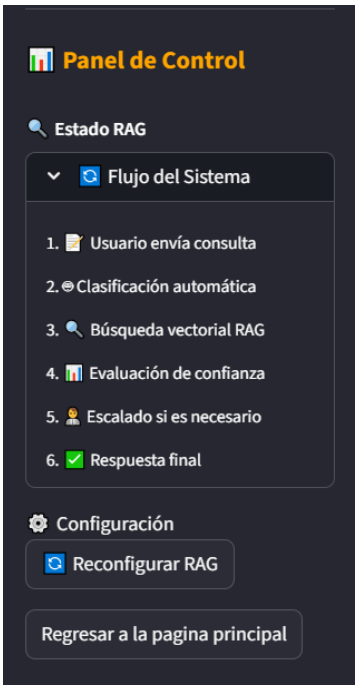


Figura 17 Diseño del panel de control (Prototipo Help Desk). Elaboración propia.

Esta figura 17 presenta un panel de control que se incorpora en la ventana de visualización del especialista como barra lateral izquierda. Este describe con más detalle el flujo de trabajo que lleva a cabo el prototipo de *Help Desk*.

CAPÍTULO IV: DESARROLLO DEL PROTOTIPO

Este capítulo contiene la descripción técnica del desarrollo e implementación del prototipo del sistema *Help Desk*, abarcando la instalación inicial del entorno, así como la programación y configuración de las diferentes herramientas y funcionalidades que se integran en él. El sistema se programa en PyCharm IDE, elegido por albergar una amplia gama de librerías, por su facilidad de uso y por permitir la creación de archivos con extensión .py para las interfaces y funciones, y .db para la construcción de la base de datos.

El eje central del sistema se fundamenta en la aplicación de la técnica RAG, la cual contribuye a reducir sesgos en la información recuperada además de disminuir la presencia de datos irrelevantes para la consulta. Complementada con el método de búsqueda semántica, esta se constituye como medio esencial para obtener respuestas coherentes y contextualizadas a través del modelo GPT.

Por otra parte, la estructura del proyecto está organizada mediante la carpeta *pages/,* cuyos archivos están vinculados a un archivo principal. Esto resulta conveniente, ya que permite distribuir y reducir la carga de trabajo lo que resulta en tiempos de respuesta más rápidos.

Además, se hace uso de Python en su versión más reciente, la 3.14.4, así como de la API correspondiente destinada a los modelos de alto aprendizaje y del *framework* de LangChain para el acceso a la base de datos vectorial. También se incluyen componentes de procesamiento de *queries,* así como para la generación de *embeddings* y *prompting*.

El aplicativo se ejecuta en un servidor web local generado por el *framework* Streamlit, encargado principalmente de la implementación de la interfaz gráfica. Por ende, tanto la base de datos vectorial como la relacional se alojan de manera local. En consecuencia, la validación y las pruebas de usuario se realizan de forma integral, sin necesidad de desplegar el sistema en un servidor externo.

4.1. Configuración del entorno de desarrollo

Debido a que el sistema *Help Desk* está desarrollado en su totalidad dentro del entorno PyCharm, su proceso de instalación y configuración resulta sencillo. Por ende, la creación de proyectos orientados a aplicaciones de índole general se facilita.

Antes de usar el entorno, primero se debe tener instalado Python en su versión más actual. Una vez hecho este paso, por defecto se crea un archivo con el nombre "main.py" en el que es posible programar. Para acceder a este entorno, es necesario dirigirse al sitio oficial de JetBrains, como se muestra en la figura 18.

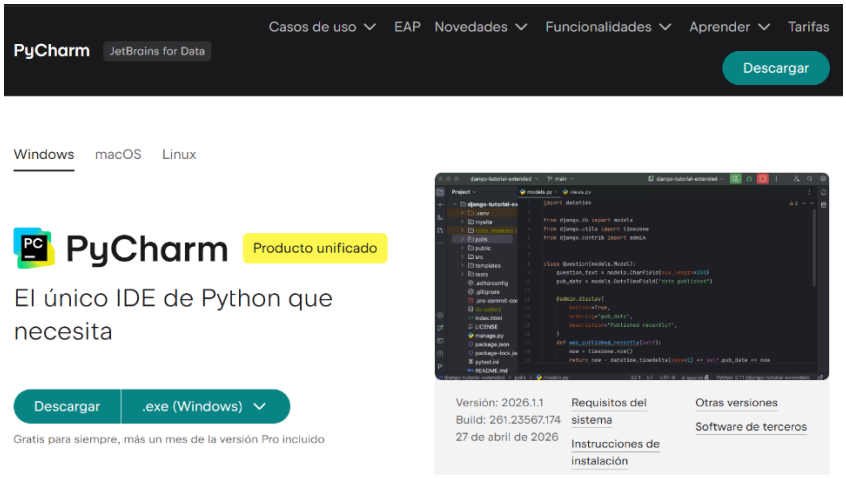


Figura 18 Sitio de descarga del entorno PyCharm.

Nota. Captura de pantalla tomada de *Download PyCharm: The Python IDE for data science and web development,* por JetBrains, s. f., JetBrains. Recuperado el 11 de mayo de 2026.

Cabe resaltar que para que el entorno esté en condiciones óptimas de trabajo hay que importar una serie de librerías como pandas, streamlit, vvid y datetime, etc. Posteriormente, y durante el desarrollo, se incorporan librerías adicionales con base en las necesidades del sistema.

4.1.1. Estructura del proyecto

A continuación, se define la organización de los archivos, carpetas y módulos que componen el prototipo *Help Desk*. Cuya importancia radica en la distribución sistemática de las funcionalidades del sistema, así como del mantenimiento constante del código lo que da adaptabilidad al aplicativo web y permite hacer correcciones en puntos clave para una mejora continua. Por ende, facilita la comunicación entre las diferentes capas de su arquitectura.

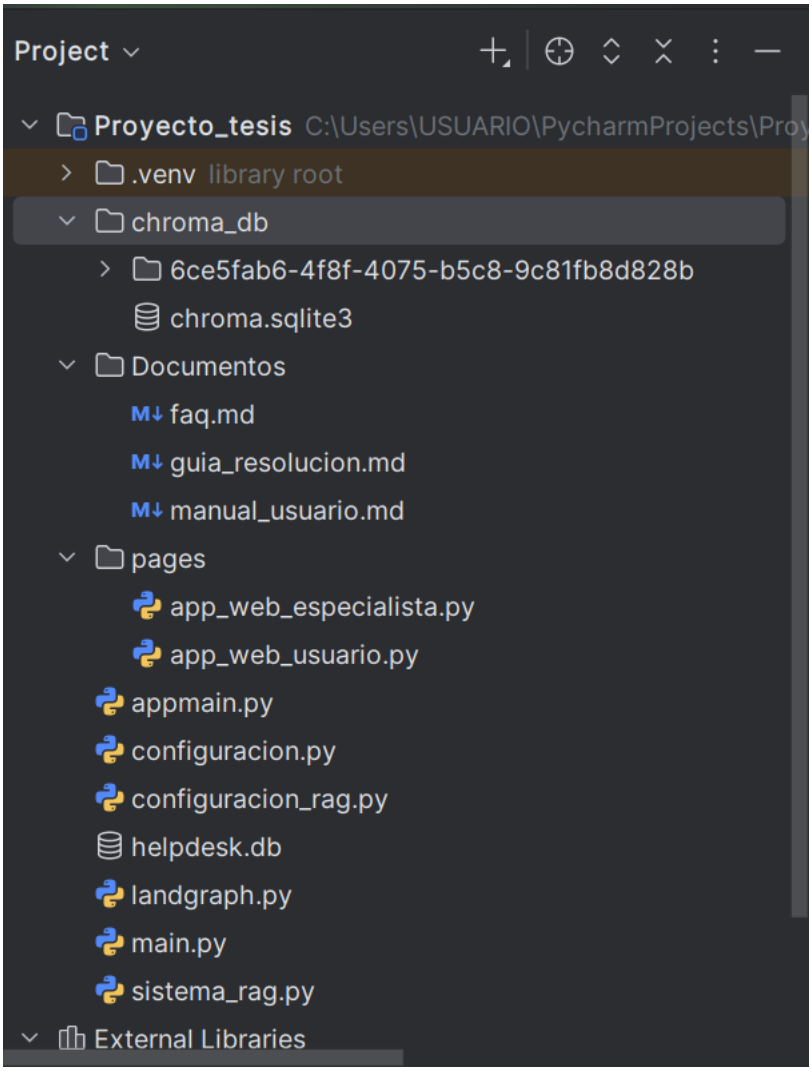


Figura 19 Estructura del proyecto Help Desk. Elaboración propia.

En la figura 19 se presenta la estructura del aplicativo web. Como primer paso, se crea un directorio acompañado de la ruta en el que se almacena el proyecto; en este caso; dicho directorio recibe el nombre "Proyecto_tesis". Dentro del mismo se crean 3 subcarpetas: documentos, pages y chroma_db.

- Documentos: En esta carpeta se almacena la base de conocimiento del sistema. Consta de tres archivos: uno de preguntas frecuentes, otro correspondiente a una guía de resolución y, por último, un manual de usuario, todos con una extensión .md, lo que reduce los costos económicos de los LLM.

- Chroma_db: En esta carpeta se aloja la base de datos vectorial ChromaDB; esto significa que en ella se almacenan los *embeddings*, metadatos y fragmentos indexados de la base de conocimiento.
- Pages: En esta carpeta se alojan archivos relacionados con las interfaces gráficas, tanto del usuario como del especialista, que para el caso de estudio en cuestión corresponden a "app_web_especialista.py" y "app_web_usuario.py".

Además, se cuenta con la presencia de un archivo principal denominado "appmain.py", desde el cual se inicia la aplicación y se visualiza la página de inicio del *Help Desk*. Adicionalmente, también se incluyen dos archivos que se encargan de las configuraciones generales y de la configuración inicial de RAG, denominados "configuracion.py" y "configuracion_rag.py", respectivamente.

- Configuracion.py: Dentro de este archivo se definen los parámetros de ChromaDB, el modelo GPT que se utiliza, las claves API y las rutas correspondientes a las carpetas como se puede visualizar en la figura 20.

```
1  CHROMADB_PATH = "C:\\Users\\USUARIO\\PycharmProjects\\Proyecto_tesis\\chroma_db"
2  DOCS_PATH = "C:\\Users\\USUARIO\\PycharmProjects\\Proyecto_tesis\\Documentos"
3  EMBEDDINGS_MODEL = "text-embedding-3-large"
```

Figura 20 Archivo "configuración.py" del proyecto Help Desk. Elaboración propia.

- Configuración_rag.py: En este archivo se procesan los documentos destinados al sistema RAG. Dicho proceso consiste en cargar la documentación, dividirla en fragmentos, transformarla en forma de embeddings y almacenarla en ChromaDB.

Y, como también se puede observar en la figura 19, existen archivos con el nombre de "landgraph.py" y "sistema_rag.py", los cuales se encargan principalmente del flujo del sistema y de la lógica de RAG, respectivamente.

- Sistema_rag.py: La función de este archivo radica en ejecutar el código correspondiente a la técnica de recuperación de información, con base en el mayor número de coincidencias encontradas entre el contexto del modelo y los fragmentos más relevantes.

Y, finalmente, se encuentra el archivo "helpdesk.db", que alberga la base de datos relacional del sistema, donde se registran y almacenan los usuarios, tickets, estados y respuestas.

4.2. Funcionamiento de la aplicación

A partir de la figura 21 se muestra el proceso completo del *prompt* de un usuario cuando es expuesto al *chatbot* conversacional. Para este caso, se empleó un *query sencillo* con el fin de identificar los procesos a los que es sometido antes de generar una respuesta definitiva. Estos se dividen en 5 pasos (sin tener en consideración la recepción de *prompts* ni mucho menos cuando se le da una solución al usuario final).

Cabe aclarar que los elementos expuestos en el diagrama son ejemplos y no representan con exactitud la respuesta que genera el sistema *Help Desk*.

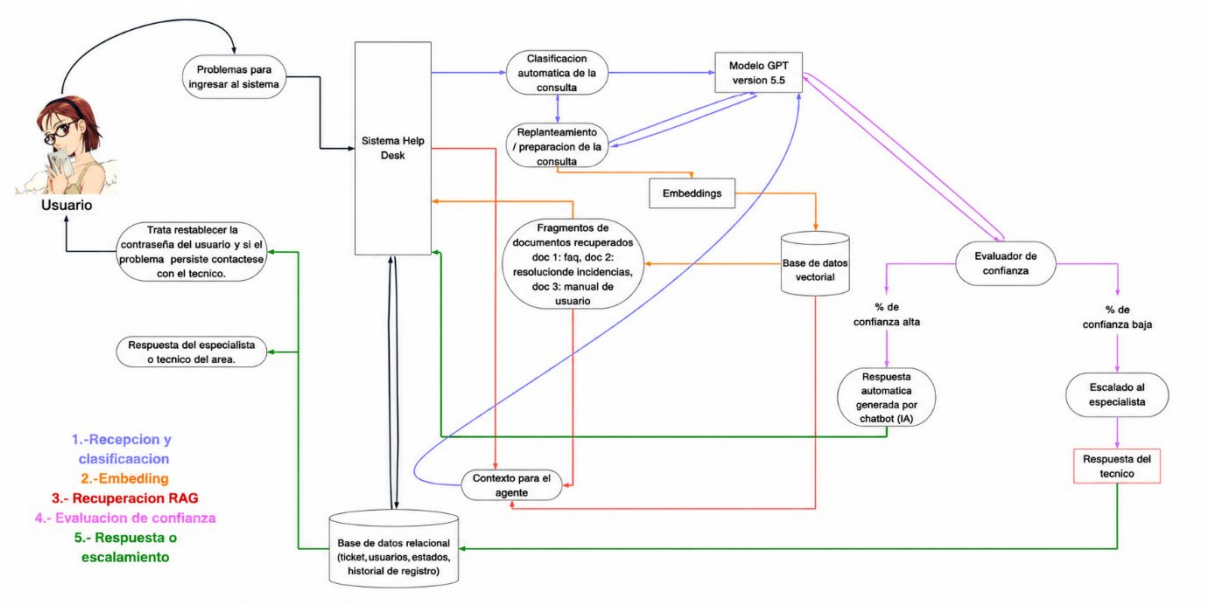
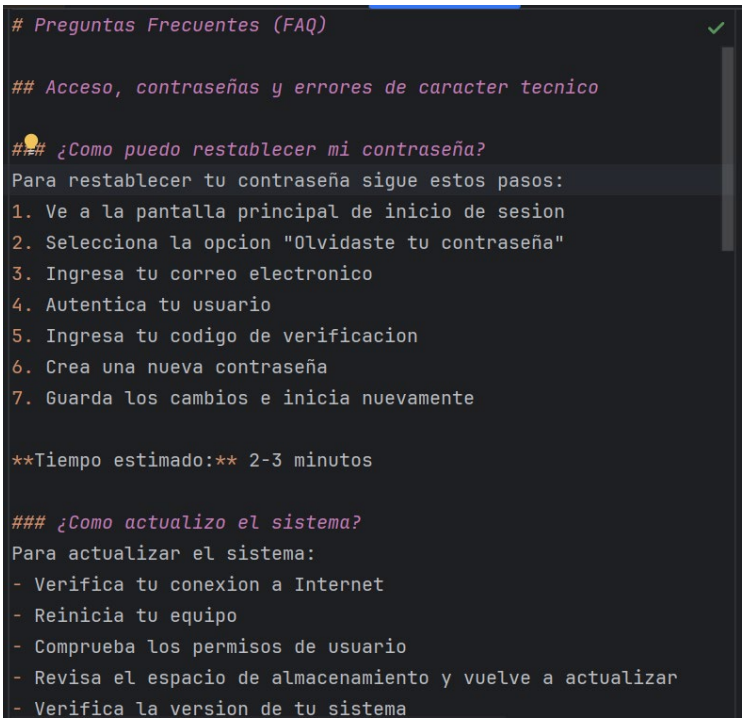


Figura 21 Diagrama de funcionamiento prototipo Help Desk. Elaboración propia.

Para el desarrollo del prototipo *Help Desk, en primera instancia,* tanto el *backend* como el *frontend* se implementan de manera local. Posteriormente se despliegan mediante el *framework* streamlit.

4.2.1. Implementación de documentos en el sistema RAG

El texto que hace alusión a los "fragmentos de documentos recuperados", en relación con el diagrama de la figura 20, corresponde a la base de conocimiento del sistema, descrito anteriormente en el inciso de la estructura del proyecto. Cada documento cuenta con un formato específico previamente definido, con la finalidad de que el sistema RAG pueda analizar y recuperar la información contenida en cada uno de ellos.



```
# Preguntas Frecuentes (FAQ)

## Acceso, contraseñas y errores de caracter tecnico

### ¿Como puedo restablecer mi contraseña?
Para restablecer tu contraseña sigue estos pasos:
1. Ve a la pantalla principal de inicio de sesion
2. Selecciona la opcion "Olvidaste tu contraseña"
3. Ingresa tu correo electronico
4. Autentica tu usuario
5. Ingresa tu codigo de verificacion
6. Crea una nueva contraseña
7. Guarda los cambios e inicia nuevamente

**Tiempo estimado:** 2-3 minutos

### ¿Como actualizo el sistema?
Para actualizar el sistema:
- Verifica tu conexion a Internet
- Reinicia tu equipo
- Comprueba los permisos de usuario
- Revisa el espacio de almacenamiento y vuelve a actualizar
- Verifica la version de tu sistema
```

Figura 22 Formato del documento FAQ (preguntas frecuentes) prototipo Help Desk. Elaboración propia.

En la figura 22 se muestra el formato del documento faq.md, en el cual se pueden visualizar dos apartados: uno relacionado con la recuperación de contraseña y el otro con la actualización del sistema. Cada apartado cuenta con iconos y separadores únicos para diferenciar la información presentada.

El texto marcado en morado corresponde a los títulos #, subtítulos ## y subdivisiones ### de cada documento. Para la pregunta "¿Cómo restablezco mi contraseña?", se optó por utilizar numeración, ya que la respuesta es una secuencia de pasos. Además, se agrega un markdown al final del apartado "Tiempo estimado".

En cambio, para la pregunta "¿Cómo actualizo el sistema?", se optó por utilizar viñetas, ya que las consultas que guarden relación con esta pregunta pueden no tener una solución directa, y de ser necesario requerir la intervención de un técnico o especialista del área.

El resto de la información de este documento presenta una estructura similar, por lo que no es necesario abordar cada punto de manera detallada. Además, este formato sirve de ejemplo para agregar más datos en un futuro, criterio que se aplica también en el resto de la documentación.

```
### Error 415 - Formato de archivo no reconocido
**Síntomas:** El sistema no acepta el archivo adjunto
**Solución:**
- Verifica el formato del archivo
- Cambiar el formato del archivo
- Guardar nuevamente el documento en el formato definido
- Actualiza la pagina
- Contacta soporte si el error persiste
```

Figura 23 Formato del documento guía de resolución prototipo Help Desk.

Elaboración propia.

Aunque el formato del documento guía_resolucion.md, mostrado en la figura 23, no difiere demasiado del primer documento presentado, hay que tener en consideración que el tipo de contenido y los separadores utilizados para estructurar la respuesta es diferente.

```
### Configuración de Notificaciones
Puedes configurar:
- Notificaciones por correo electronico
- Alertas sobre cambios de estado
- Avisos de nuevos comentarios
- Recordatorios personalizados
- Notificaciones dentro del sistema
```

Figura 24 Formato del documento manual de usuario prototipo Help Desk.

Elaboración propia.

La información presentada en la figura 24 cuyo contenido no mantiene relación directa con algún error o problema específico, como sí ocurre en el caso de los dos documentos anteriores. Más bien, este apartado es para solicitar algún tipo de guía o asesoramiento. El formato de respuesta que utiliza este inciso se basa en la combinación de un separador y un *markdown* descritos anteriormente: las viñetas y el resaltado en negrita. Asimismo, todos estos elementos forman parte del proceso *embedding*, referente al diagrama de la figura 21.

Una vez creada la base de conocimiento se procede a implementar la funcionalidad encargada de dividir los documentos en diferentes fragmentos, convertirlos en vectores y crear una base de datos vectorial para almacenarlos.

Además, se realiza una configuración inicial del sistema RAG por medio de una API key, con la finalidad de verificar su correcto funcionamiento. Esta clave permite que el proyecto se conecte al servicio externo de OpenAI y utilice su modelo de *embeddings.* De esta manera, garantiza que el acceso se limite al propietario de la API key, evitando que cualquier persona pueda utilizarlo sin autorización. Asimismo, ayuda a comparar la pregunta del usuario con los fragmentos de los documentos previamente definidos.

4.2.2. Configuración inicial de RAG

A continuación, se muestra el código que se implementa dentro del archivo configuración_rag.py. La explicación a detalle de las librerías importadas en el proyecto queda excluida por cuestiones de optimización documental, ya que describir el nombre y funcionalidad de cada una implicaría abordar aspectos técnicos con un grado alto de complejidad y extensión.

```
class procesador_documento:  6+ usages
    """Procesador de documentos para el sistema RAG."""

    def __init__(self, ruta_document: str = "Documentos",
                 ruta_Chroma: str = "./chroma_db",
                 tam_frag: int=1000, solapamiento: int=200):
#Rutas donde se alojan la base de conocimiento, la  base de datos vectorial y los embedd
        self.docs_path = Path(ruta_document)
        self.chroma_path = Path(ruta_Chroma)
        self.embeddings = OpenAIEmbeddings(model=EMBEDDINGS_MODEL)

#Configuracion para dividir el documento en fragmentos mas pequeños
        self.text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=tam_frag,
            chunk_overlap=solapamiento,
            length_function=len,
            separators=["\n #","\n ##","\n ###","\n\n",
                        "\n", ".", "!", "?", ",", " ", ""]
        )
```

Figura 25 Configuración inicial de fragmentación de documentos. Elaboración propia.

Como se puede visualizar en la figura 25, se definen las rutas en las que se almacena la información relacionada con la base de conocimiento; además, se describen los separadores que sirven de referencia para que RAG divida el documento en fragmentos pequeños.

```python
def cargar_documents(self) -> List[Document]:  1 usage
    if not self.docs_path.exists():
        raise FileNotFoundError(f"Documentos no encontrados: {self.docs_path}")

    loader = DirectoryLoader(
        path=str(self.docs_path),
        glob="*.md",
        loader_cls=TextLoader,
        loader_kwargs={"encoding": "utf-8"}
)
    documents=loader.load()

    for documento in documents:
        ruta_archivo = documento.metadata.get("source","")
        #Extrae el nombre del archivo sin extension
        nombre_archivo = Path(ruta_archivo).stem
        documento.metadata.update({"Nombre_archivo": nombre_archivo,
                                   "Tipo_documento": self.obtener_tipo_documento(nombre_archivo),
                                   "Id_documento": self.generar_documento_id_documento(documento.page_content),
                                   "Fuente_de_origen":str(ruta_archivo),})
    print(f"✅ Se cargaron {len(documents)} documentos correctamente.")
    return documents
```

Figura 26 Función de carga de documentos al sistema Help Desk. Elaboración propia.

La función que se muestra en la figura 26 representa el proceso de carga de la documentación almacenada en la carpeta Documento. En primera instancia, comprueba la existencia de dicha carpeta y, posteriormente, lee únicamente los archivos con extensión .md a través de *DirectoryLoader* y *TextLoader*. Una vez cargados, los enriquece con metadatos, como el nombre del archivo, el tipo de documento, el identificador único y la ruta de origen. Esto permite estructurar los datos de manera más organizada y facilitar el proceso que usa RAG para recuperar la información.

```python
def obtener_tipo_documento(self, nombre_archivo: str) -> str:  1 usage
    nombre_archivo=nombre_archivo.lower()
    if "faq" in nombre_archivo:
        return "Preguntas frecuentes"
    elif "manual_usuario" in nombre_archivo or "manual" in nombre_archivo:
        return "Manual de usuarios"
    elif "guia_resolucion" in nombre_archivo or "guia" in nombre_archivo:
        return "Guia de resolucion de incidencias"
    else:
        return "Documentacion general"
```

Figura 27 Obtención del tipo de documento. Elaboración propia.

La función presentada en la figura 27 clasifica los documentos antes de almacenarlos en la base de datos vectorial. Dicha clasificación se realiza con base en el nombre del archivo, el cual sirve como identificador para diferenciar el contenido de la documentación.

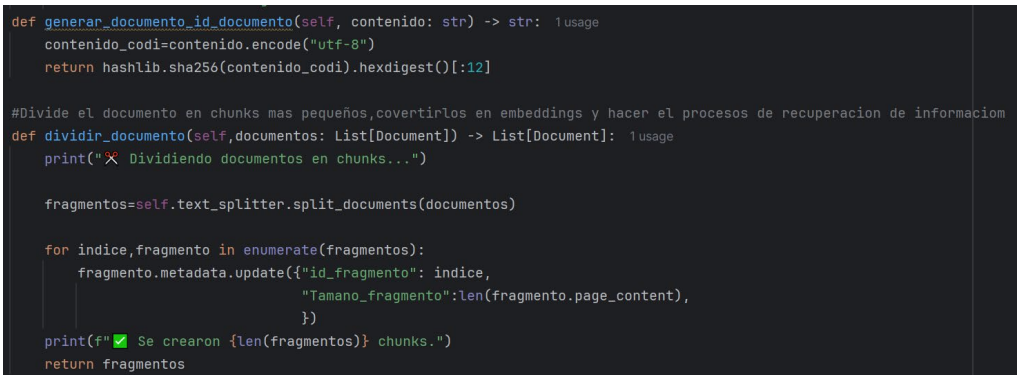


Figura 28 Función de generador de id y de división de documentos.

Elaboración propia.

A continuación, en la figura 28, se presentan las funciones encargadas de generar el identificador único del documento y de dividir su contenido en *chunks* o fragmentos más pequeños para su posterior conversión en *embeddings* o vectores.

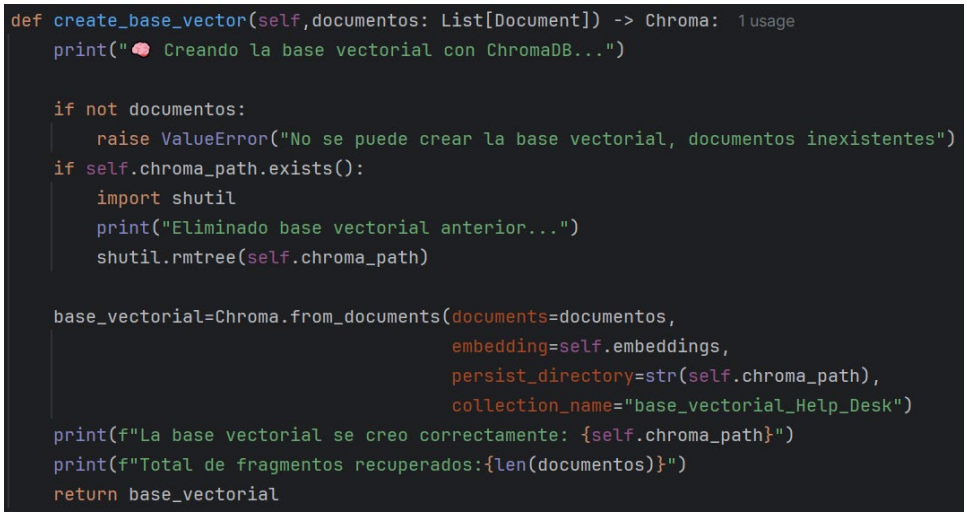


Figura 29 Creación de la base de datos vectorial (ChromaDB) del sistema Help Desk.

Elaboración propia.

En la figura 29 se muestra el código encargado de crear la base vectorial del sistema Help Desk a partir de los fragmentos recuperados. Para almacenar los documentos en ChromaDB, se hace uso del modelo de *embeddings,* el cual transforma el contenido en vectores. Posteriormente, dicha base se guarda en una ruta definida y queda disponible para que la utilice el sistema RAG.

```python
def cargar_base_vectorial(self) -> Chroma:  1 usage
    print("Buscando base vectorial...")

    if not self.chroma_path.exists():
        raise FileNotFoundError(f"Base vectorial no encontrada: {self.chroma_path}")

    base_vectorial=Chroma(persist_directory=str(self.chroma_path),
                          embedding_function=self.embeddings,
                          collection_name="base_vectorial_Help_Desk")
    print(f"La base vectorial se cargo correctamente: {self.chroma_path}")
    return base_vectorial
```

Figura 30 Carga de la base de datos vectorial (ChromaDB) del sistema Help Desk. Elaboración propia.

En la figura 30 se muestra el código encargado de buscar la ruta de alojamiento de la base de datos vectorial creada previamente. Una vez localizada, se carga dicha base en el prototipo de *Help Desk*.

```python
# Carga el sistema RAG en base chorma creado
def configuracion_rag(self, reconstruir_base:bool=False):  3+ usages
    if self.chroma_path.exists() and not reconstruir_base:
        print("Base vectorial encontrada...")
        return self.cargar_base_vectorial()

    documentos=self.cargar_documents()

    if not documentos:
        print("No se hallaron documentos para procesar")
        return None
    fragmentos=self.dividir_documento(documentos)
    base_vectorial=self.create_base_vector(fragmentos)
    print("Sistema RAG configurado correctamente")
    return base_vectorial
```

Figura 31 Configuración inicial del sistema RAG. Elaboración propia.

Finalmente, en la figura 31 se presenta la función encargada de permitir que el sistema pueda hacer uso de la base de conocimiento.

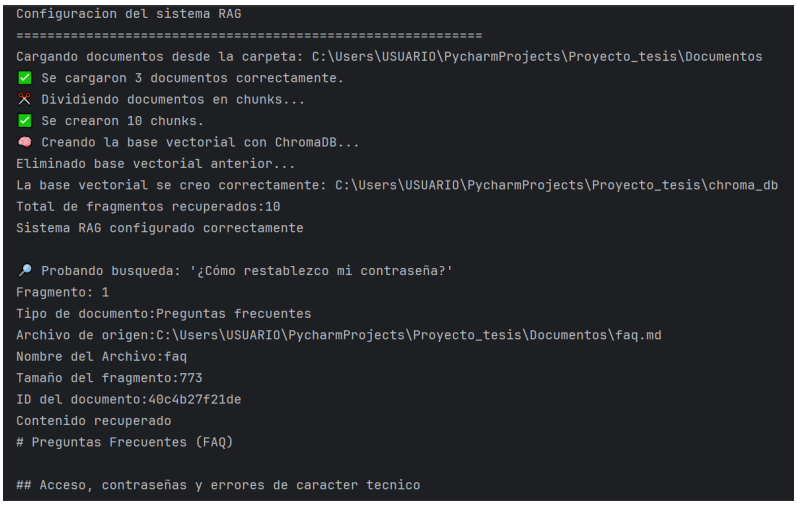


Figura 32 Test de prueba del funcionamiento de RAG. Elaboración propia.

Como se puede visualizar en la figura 32, se ejecuta un test del sistema RAG para comprobar su correcto funcionamiento. Además, como resultado, el contenido recuperado de cada documento está enriquecido con metadatos, los cuales permiten diferenciar el tipo de información que se tiene en consideración para la respuesta al usuario.

4.2.3. Configuración avanzada de RAG

Una vez que se implementa la configuración inicial de RAG en el prototipo *Help Desk,* se procede a optimizar su funcionamiento con *MultiQueryRetriever*. Para ello, se crea una clase auxiliar dentro del fichero sistema_rag.py, la cual permite recuperar los documentos de la base de datos vectorial de una manera eficiente.

MultiQueryRetriever es un componente de LangChain que mejora el sistema de búsqueda de documentos dentro de un entorno que emplea RAG, ya que se basa en múltiples versiones de la consulta del usuario. En términos generales, considera varias formas en las que un usuario podría plantear su pregunta, lo que permite ampliar la búsqueda en la base vectorial ChromaDB.

```python
class VectorRAGsystem:  2+ usages

    def __init__(self,ruta_chroma:str="chroma_db",cantidad_documentos:int=3,
                 nombre_coleccion:str="base_vectorial_Help_Desk",):
        self.chroma_path = Path(ruta_chroma)
        self.cantidad_documentos = cantidad_documentos
        self.nombre_coleccion = nombre_coleccion
        self.embeddings=OpenAIEmbeddings(model=EMBEDDINGS_MODEL)
        self.llm=ChatOpenAI(model="gpt-4o-mini",temperature=0)
        self.vectorstore=None
        self.retriever=None
        logging.basicConfig(level=logging.INFO)
        logging.getLogger("langchain.retrievers.multi_query").setLevel(logging.INFO)
        self.cargar_vector()
```

Figura 33 Inicialización de RAG dentro del sistema Help Desk con ChromaDB. Elaboración propia.

Como se visualiza en la figura 33, se describe el código correspondiente al constructor de la clase VectorRAGSystem, encargado de inicializar la configuración avanzada de RAG. Esta función o método __init__ define la cantidad de documentos que se recuperan por cada consulta, además de cargar el modelo *embeddings,* así como el modelo GPT. Dichos modelos cumplen la función de convertir la información en vectores y de generar las respuestas para el usuario, con base en la lógica de funcionamiento de un *chatbo*t conversacional.

Cabe resaltar que todas las funciones se construyen bajo esta clase VectorRAGSystem.

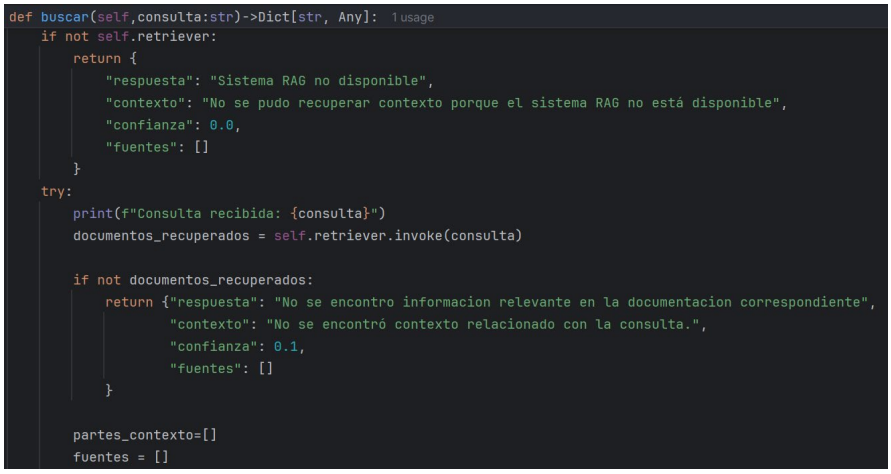
```python
def cargar_vector(self):  1 usage
    try:
        print("Buscando base de datos vectorial del sistema Help Desk....")
        if not self.chroma_path.exists():
            print(f"Base de datos vectorial no encontrada: {self.chroma_path}")
            self.vectorstore=None
            self.retriever=None
            return None

        self.vectorstore=Chroma(
            persist_directory=str(self.chroma_path),
            embedding_function=self.embeddings,
            collection_name=self.nombre_coleccion,
        )

        recuperador_base=self.vectorstore.as_retriever(search_type="similarity",
                                                       search_kwargs={"k":self.cantidad_documentos})
        self.retriever=MultiQueryRetriever.from_llm(retriever=recuperador_base,
                                                    llm=self.llm,
                                                    prompt=self._get_multi_query_prompt())
        print("Sistema RAG inicializado correctamente")
    print(f"Base vectorial cargada desde: {self.chroma_path}")
    return self.vectorstore
except Exception as error:
    print(f"Error al cargar la base vectorial{str(error)}")
    self.vectorstore=None
    self.retriever=None
    return None
```

Figura 34 Proceso de carga de la base vectorial al sistema Help Desk. Elaboración propia.

El código de la figura 34 corresponde al proceso asociado con la carga de la base vectorial ChromaDB y la configuración de *MultiQueryRetriever,* con el fin de activar la función de búsqueda de RAG. Para ello, se debe tener en consideración la existencia de la ruta en donde se almacena ChromaDB; de lo contrario, se produce una falla en la carga de la documentación correspondiente.



```
def buscar(self,consulta:str)->Dict[str, Any]:  1 usage
    if not self.retriever:
        return {
            "respuesta": "Sistema RAG no disponible",
            "contexto": "No se pudo recuperar contexto porque el sistema RAG no está disponible",
            "confianza": 0.0,
            "fuentes": []
        }
    try:
        print(f"Consulta recibida: {consulta}")
        documentos_recuperados = self.retriever.invoke(consulta)

        if not documentos_recuperados:
            return {"respuesta": "No se encontro informacion relevante en la documentacion correspondiente",
                    "contexto": "No se encontró contexto relacionado con la consulta.",
                    "confianza": 0.1,
                    "fuentes": []
            }

        partes_contexto=[]
        fuentes = []
```

Figura 35 Proceso de búsqueda de información en el sistema RAG.

Elaboración propia.

En lo que respecta al código de la figura 35, se describe una función que se encarga de la búsqueda dentro de RAG, tomando como referencia la consulta ingresada por el usuario. En caso de que RAG esté configurado correctamente, ejecuta la búsqueda mediante *MultiQueryRetriever,* con el fin de recuperar los fragmentos más relevantes desde ChromaDB. Además, esta función prepara la lista que contendrá el contexto y las fuentes de donde se obtuvo la información, para posteriormente generar de manera automática una respuesta dirigida al usuario.

```python
def _generar_respuesta(self,consulta:str,contexto:str)->str:  1 usage
    prompt = f"""
    Eres un asistente de soporte técnico de un sistema Help Desk.

    Tu tarea es responder la consulta del usuario utilizando únicamente la información
    del contexto recuperado desde la base de conocimiento.

    Consulta del usuario:
    {consulta}

    Contexto recuperado:
    {contexto}

    Instrucciones:
    - Responde de forma clara, directa y útil para el usuario.
    - No generes preguntas alternativas.
    - No reformules la consulta.
    - No digas que estás usando RAG.
    - No inventes información que no esté en el contexto.
    - Si el contexto no es suficiente, indica que el caso debe ser revisado por un especialista.

Respuesta final para el usuario:
```

Figura 36 Generación de respuesta al usuario con base en el modelo GPT. Elaboración propia.

La figura 36 describe una función que se encarga de generar la respuesta final dirigida al usuario, tomando como referencia el modelo GPT y el contexto de la consulta recuperado desde la base de conocimiento. Es importante tener en consideración el *prompt* de instrucciones, que el modelo usa como guía para dar una respuesta coherente y contextualizada.

```python
def _calcular_confianza(self,consulta:str,documentos:list[Document])->float:  1 usage
    if not documentos:
        return 0.0
    texto_consulta={texto.lower()
                    for texto in consulta.split()
                    if len(texto) > 2}
    if not texto_consulta:
        return 0.3
    puntacion_relevancia=0
    total_texto_contexto=0
    for documento in documentos[:3]:
        contenido=documento.page_content.lower()
        texto_contenido=contenido.split()

        total_texto_contexto+=len(texto_contenido)

        coincidencia= sum(1 for texto in texto_consulta
                          if texto in contenido)
        puntacion_relevancia+=coincidencia


        confianza_base=puntacion_relevancia/len(texto_consulta)
        confianza_base=min(confianza_base,1.0)

        bonus_documentos=min(len(documentos)/4.0,0.2)
        bonus_contenido=min(total_texto_contexto/1000.0,0.1)

        confianza_final=confianza_base+bonus_documentos+bonus_contenido
        confianza_final=min(confianza_final,1.0)
        return round(confianza_final,2)
```

Figura 37 Método para medir el nivel de confianza de la respuesta. Elaboración propia.

Para finalizar el proceso de configuración de RAG, en la figura 37 se describe una función que se encarga de estimar el nivel de confianza de la respuesta generada. Esta función compara y contrasta palabras que guarden algún tipo de similitud entre la consulta y la documentación disponible, con el objetivo de encontrar el mayor número de coincidencias posibles.

Mediante este proceso de búsqueda, se establece un cálculo cuyos valores oscilan entre 0.0 y 1.0, con el fin de determinar el grado de relevancia de la información recuperada, la cual sirve como parámetro base dentro de los sistemas *Help Desk*.

4.3. Creación e integración del grafo con LangGraph

En este apartado se crea el grafo con LangGraph y se integra con el sistema de consultas RAG previamente configurado. Este proceso se lleva a cabo dentro del fichero "landgraph.py", en el cual se define el estado, es decir, la información actual del aplicativo web y la forma en que esta se procesa a través de un flujo de trabajo. Dicho flujo establece el orden en el que se ejecuta cada tarea, como guardar, actualizar información.

```
class HelpDeskEstado(TypedDict,total=False):
    consulta_usuario:str
    respuesta_rag:Optional[str]
    confianza:float
    fuentes:List[str]
    contexto_rag:Optional[str]
    clasificacion_requerimiento:str
    requiere_escalamiento:bool
    respuesta_tecnico:Optional[str]
    respuesta_final:Optional[str]
    historial:Annotated[List[str],add]
```

Figura 38 Definición del estado. Elaboración propia.

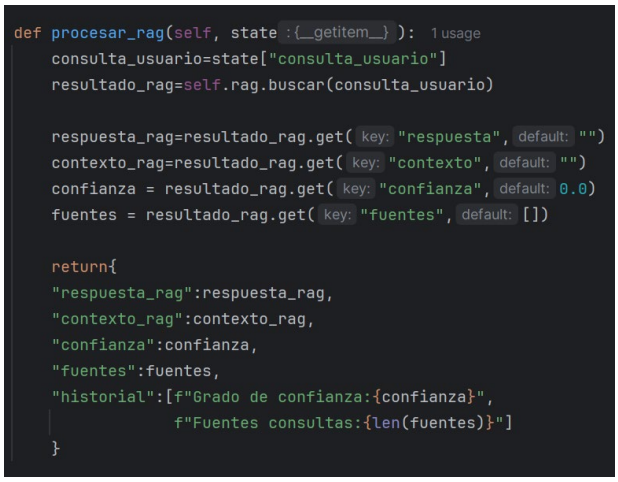
Como se puede observar en la figura 38, se definen los atributos que se utilizan desde los nodos de LangGraph, los cuales sirven como una estructura para organizar la información que se almacena durante el proceso de atención de una consulta. Esta información es presentada al especialista del área cuando gestiona una incidencia. A continuación, se describe el rol que desempeña cada atributo especificado en el código.

- **El campo consulta_usuario:** Almacena la información relacionada con la pregunta o requerimiento hecho por el usuario.
- **El campo respuesta_rag:** Almacena la respuesta recuperada mediante RAG.
- **El campo confianza:** Almacena el porcentaje o nivel de confianza que tiene el sistema para responder a una consulta determinada.
- **El campo fuentes:** Guarda las fuentes de información consultadas por RAG desde la base de conocimiento.
- **El campo de contexto_rag:** Almacena la información relevante previamente recuperada, la cual sirve de referencia para responder al requerimiento del usuario.
- **El campo clasificación_del_requerimiento y requiere_escalamiento:** Indica si el requerimiento puede ser atendido por la IA o escalado al especialista del área correspondiente.
- **El campo respuesta_técnico y respuesta_final:** Almacena la información de la respuesta proporcionada por el técnico, en caso de haberse escalado. Caso contrario, se almacena la respuesta generada por la IA.
- **El campo historial:** Registra el flujo de trabajo del sistema que se representa mediante una lista de pasos consecutivos, que permite observar todo el proceso de la consulta, desde su recepción hasta su resolución.

```
class HelpDeskgrafo:  1 usage
    def __init__(self):
        self.llm = ChatOpenAI(model="gpt-4o-mini",temperature=0.1)
        self.rag = VectorRAGsystem(ruta_chroma=CHROMADB_PATH)
        self.graph = None
```

Figura 39 Inicialización del modelo GPT. Elaboración propia.

En referencia a la figura 39, se define un constructor encargado de inicializar el modelo GPT, instanciar la ruta de la base vectorial y preparar la variable para almacenar el grafo.



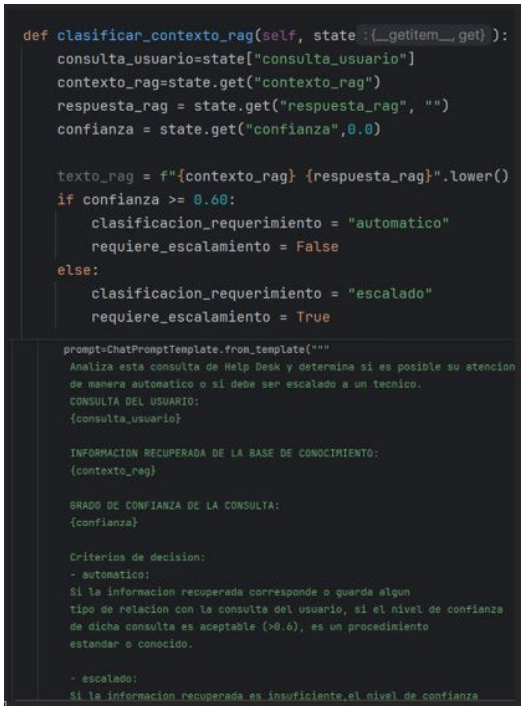
```
def procesar_rag(self, state :{__getitem__} ):  1 usage
    consulta_usuario=state["consulta_usuario"]
    resultado_rag=self.rag.buscar(consulta_usuario)

    respuesta_rag=resultado_rag.get( key: "respuesta", default: "")
    contexto_rag=resultado_rag.get( key: "contexto", default: "")
    confianza = resultado_rag.get( key: "confianza", default: 0.0)
    fuentes = resultado_rag.get( key: "fuentes", default: [])

    return{
    "respuesta_rag":respuesta_rag,
    "contexto_rag":contexto_rag,
    "confianza":confianza,
    "fuentes":fuentes,
    "historial":[f"Grado de confianza:{confianza}",
                 f"Fuentes consultas:{len(fuentes)}"]
    }
```

Figura 40 Búsqueda de información relevante. Elaboración propia.

En lo que respecta al código de la figura 40, se especifica una función encargada de ejecutar la búsqueda y recuperación de información relevante desde la base de conocimiento. Entre el contenido que devuelve se encuentra el nivel de confianza, el número de fuentes encontradas, etc. Dichos datos constituyen un estado, el cual se usa para la retroalimentación de los nodos posteriores.



```
def clasificar_contexto_rag(self, state :{__getitem__, get} ):
    consulta_usuario=state["consulta_usuario"]
    contexto_rag=state.get("contexto_rag")
    respuesta_rag = state.get("respuesta_rag", "")
    confianza = state.get("confianza",0.0)

    texto_rag = f"{contexto_rag} {respuesta_rag}".lower()
    if confianza >= 0.60:
        clasificacion_requerimiento = "automatico"
        requiere_escalamiento = False
    else:
        clasificacion_requerimiento = "escalado"
        requiere_escalamiento = True

    prompt=ChatPromptTemplate.from_template("""
     Analiza esta consulta de Help Desk y determina si es posible su atencion
     de manera automatico o si debe ser escalado a un tecnico.
     CONSULTA DEL USUARIO:
     {consulta_usuario}

     INFORMACION RECUPERADA DE LA BASE DE CONOCIMIENTO:
     {contexto_rag}

     GRADO DE CONFIANZA DE LA CONSULTA:
     {confianza}

     Criterios de decision:
     - automatico:
     Si la informacion recuperada corresponde o guarda algun
     tipo de relacion con la consulta del usuario, si el nivel de confianza
     de dicha consulta es aceptable (>0.6), es un procedimiento
     estandar o conocido.

     - escalado:
     Si la informacion recuperada es insuficiente,el nivel de confianza
```

Figura 41 Clasificación del contexto RAG. Elaboración propia.

El código correspondiente a la figura 41 describe un método encargado de clasificar el requerimiento del usuario a partir del ingreso de la consulta. Para ello, se utiliza un *prompt*, el

cual ayuda al modelo GPT a definir los criterios de evaluación; estos se traducen en términos de una atención personalizada a través de IA o un escalamiento al especialista del área. En caso de que el modelo muestre fallas durante el proceso de clasificación, el sistema Help Desk utiliza el nivel de confianza como respaldo para la toma de decisiones.

4.3.1. Human in the loop

En el campo del desarrollo de la inteligencia artificial, las aplicaciones o agentes que se incorporan dentro de modelos LLM no siempre funcionan de manera completamente autónoma, ya que en ciertas partes críticas de determinados procesos es necesaria la intervención humana. Esto puede ocurrir, por ejemplo, para validar decisiones o proporcionar información adicional.

En este caso de estudio, el uso de LangGraph facilita la implementación de esquemas denominados *human in the loop,* es decir, la integración del ser humano dentro del grafo y del bucle de ejecución. Esto se realiza mediante interruptores de ejecución y persistencia del estado.

La idea consiste en pausar la ejecución de un grafo en un punto específico, permitir que el humano intervenga, ya sea observando, modificando el estado o añadiendo información adicional, y luego reanudar el flujo de ejecución exactamente en el punto en el que se detuvo.

Por ende, se utilizan dos directivas después de la compilación del grafo: *interrupt before,* que permite interrumpir la ejecución antes de llegar a un nodo específico, e *interrupt after,* que realiza un proceso similar, pero después de llegar a un nodo determinado.

```
def preparar_escalamiento(self, state :{get} ):  1 usage
    consulta_usuario=state.get("consulta_usuario","")
    confianza=state.get("confianza",0.0)
    clasificacion_requerimiento=state.get(
        "clasificacion_requerimiento",
        "escalado")
    return {
        "requiere_escalamiento":True,
        "historial":[
            "El requerimiento fue trasladado a soporte tecnico"
            f"Consulta del usuario: {consulta_usuario}",
            f"Grado de confianza: {confianza}",
            f"Clasficacion del requerimiento: {clasificacion_requerimiento}",
        ]
    }
```

Figura 42 Proceso de escalamiento del requerimiento. Elaboración propia.

El código que se observa en la figura 42 corresponde a una función encargada de preparar la información necesaria para derivar el requerimiento al especialista del área de TI, cuando este no puede ser atendido automáticamente por la IA. En este caso, el sistema recupera la

información relacionada con la consulta del usuario, destacando el nivel de confianza y la clasificación del requerimiento. Posteriormente actualiza e indica el número de incidencias que requieren escalamiento.

```python
def procesar_respuesta_tecnico(self,state :{get} ):  1 usage
    respuesta_tecnico=state.get("respuesta_tecnico","")

    if respuesta_tecnico:
        return {
            "respuesta_final":respuesta_tecnico,
            "requiere_escalamiento":False,
            "historial":[
                "El tecnico del area atendio el requerimiento y de una respuesta"
                "La respuesta final fue actualizada y redefinida por el tecnico "
            ]
        }
    return {"requiere_escalamiento":True,
            "historial":[
                "El requerimiento esta en proceso de ser atendido por el tecnico "
            ]}
```

Figura 43 Proceso de respuesta del especialista del área. Elaboración propia.

Referente a la figura 43, se muestra el código correspondiente a la función encargada de procesar la respuesta del técnico del área. En esta función se verifica si el técnico ha proporcionado una solución al requerimiento escalado. En caso de existir una respuesta pertinente, esta se almacena bajo el nombre de respuesta_final, la cual actualiza el estado del requerimiento y lo marca como finalizado. Por el contrario, si la respuesta aún no tiene una solución definitiva, pues el requerimiento permanece en un estado de espera hasta que sea atendido.

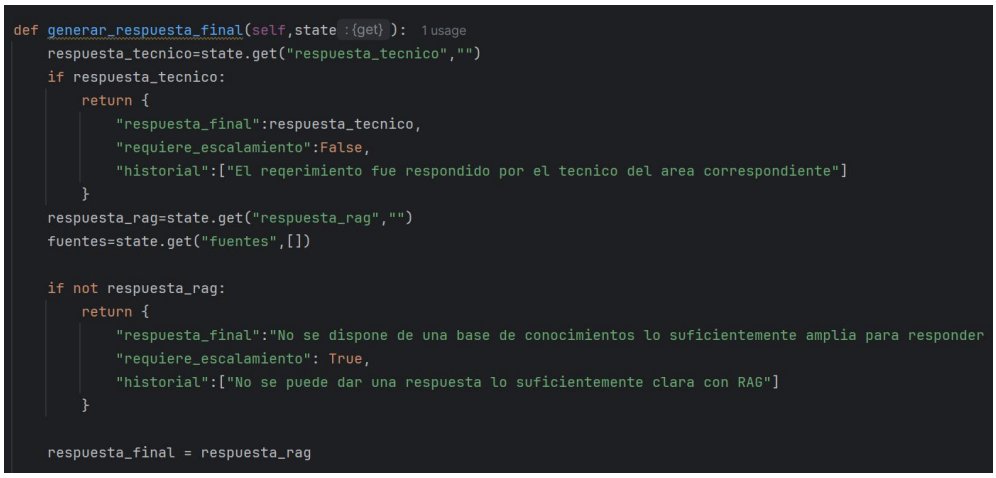


Figura 44 Proceso de asignación para la respuesta final. Elaboración propia.

Finalmente, el código correspondiente a la figura 44 se encarga de determinar el actor más adecuado para la consulta del usuario en correlación con la respuesta final. Dicho actor se representa como la IA o como el especialista del área.

En caso de que la respuesta sea proporcionada por el especialista, la función verifica su existencia y la almacena como respuesta final. De lo contrario, utiliza la respuesta recuperada mediante RAG y la complementa con las fuentes consultadas, con el fin de dar por terminado todo el proceso de atención del requerimiento.

4.3.2. Funciones de enrutamiento

Una vez que se implementa toda la funcionalidad de los nodos, el siguiente paso consiste en definir una estructura para el grafo. Para ello, es necesario incorporar algunas funciones adicionales para controlar el flujo de ejecución. Estas funciones se conocen como funciones de enrutamiento que se encargan de determinar el camino que debe seguir el requerimiento durante todo su ciclo.



Figura 45 Función de enrutamiento basada en la clasificación del requerimiento. Elaboración propia.

El código que hace referencia a la figura 45 corresponde a la función encargada de definir el camino que debe seguir la consulta después del proceso de clasificación con contexto RAG. En otras palabras, esta función determina si es conveniente generar una respuesta de manera automática o escalarla a un técnico del área correspondiente para su posterior resolución.

```python
def decidir_desde_tecnico(self,state :{get} ):  1 usage
    respuesta_tecnico=state.get("respuesta_tecnico","")
    if respuesta_tecnico:
        return "procesar_respuesta_tecnico"
    else:
        return "esperar_respuesta_tecnico"
```

Figura 46 Función de enrutamiento basada en la respuesta del técnico.

Elaboración propia.

El código correspondiente a la figura 46 muestra una función encargada de decidir si el flujo del grafo debe continuar con el procesamiento de la respuesta del técnico o mantenerlo en espera. Esto significa que el sistema mantiene el estado del requerimiento como pendiente hasta la respectiva intervención del técnico.

4.3.3. Conexión del grafo

Para finalizar con la creación del grafo con LangGraph, se establece la conexión entre los nodos y el control de flujo que define el orden de ejecución de cada proceso dentro del sistema *Help Desk.* Este flujo inicia con la recepción de la consulta del usuario y continúa con su gestión mediante RAG.

Dicha conexión es clave en la atención del requerimiento, ya que permite determinar el agente responsable de generar la respuesta, ya sea este el LLM integrado con IA o el especialista del área correspondiente.

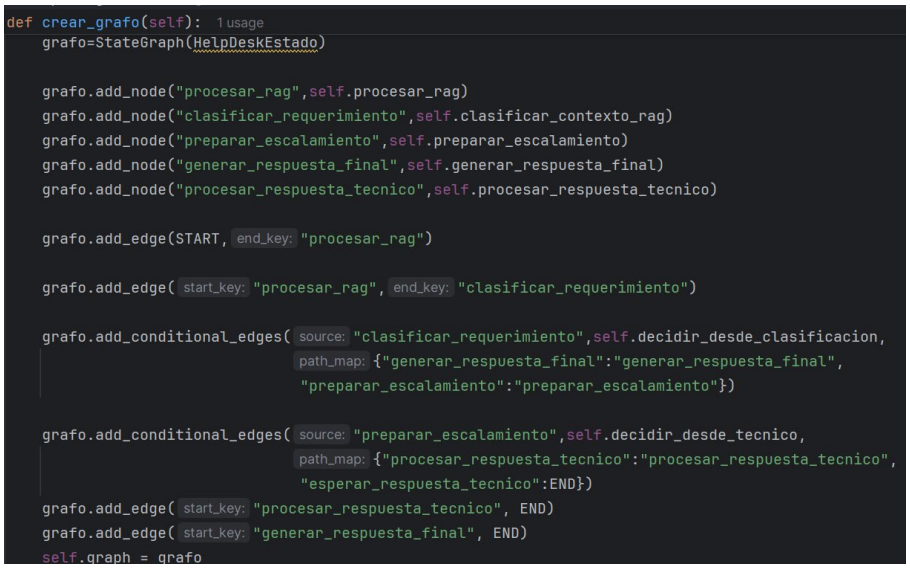


Figura 47 Conexión y creación del grafo final con LangGraph. Elaboración propia.

En referencia al código de la figura 47, se muestra una función encargada de incorporar cada nodo que representa una etapa del flujo del sistema *Help Desk* dentro de la estructura del grafo previamente creada. Para ello, se define el mecanismo de procesamiento de la consulta mediante RAG y su respectiva clasificación de acuerdo con su nivel de complejidad.

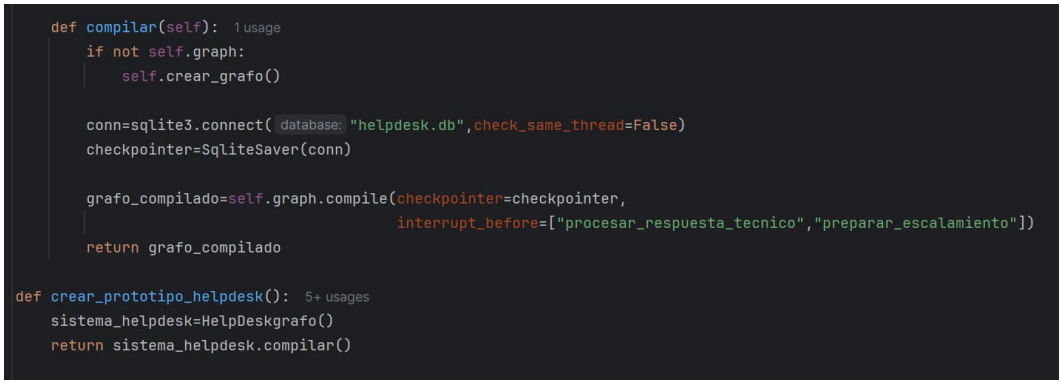
Todas estas consignas fueron descritas en los apartados anteriores; por lo tanto, para evitar repeticiones innecesarias, se excluye la explicación detallada de cada nodo en esta sección.

4.3.4. Checkpoints y persistencia de estado

Una de las abstracciones interesantes que proporciona LangGraph son los *checkpoints,* los cuales están relacionados con la persistencia de la ejecución y el estado del grafo. En el desarrollo de aplicativos webs que implementan grafos grandes, con nodos que pueden tener tiempos de ejecución relativamente altos, resulta conveniente pausar la ejecución del grafo en un momento determinado para realizar algún tipo de acción o, simplemente reanudar el proceso de ejecución en el punto en el que se detuvo.

Esto es precisamente lo que permiten los *checkpoints.* LangGraph incorpora de forma nativa un sistema de puntos de control que permiten persistir el estado de un grafo en pasos significativos, es decir, durante las transiciones entre nodos. De esta manera, se conserva el estado general de la ejecución, lo que permite pausar o reanudar *workflows,* así como incorporar técnicas de *human in the loop*.

Una vez definido este concepto, el grafo se compila y se le proporciona un objeto *checkpoint*, el cual brinda varias formas de guardar el estado del grafo en una base de datos, como SQLite, mediante clases auxiliares. De esta forma, si el sistema se reinicia o sufre un apagón inesperado, la ejecución puede reanudarse y mantener todas las características del estado guardadas. Este proceso se puede visualizar en la figura 48.



```
    def compilar(self):  1 usage
        if not self.graph:
            self.crear_grafo()

        conn=sqlite3.connect( database: "helpdesk.db",check_same_thread=False)
        checkpointer=SqliteSaver(conn)

        grafo_compilado=self.graph.compile(checkpointer=checkpointer,
                                           interrupt_before=["procesar_respuesta_tecnico","preparar_escalamiento"])
        return grafo_compilado

def crear_prototipo_helpdesk():  5+ usages
    sistema_helpdesk=HelpDeskgrafo()
    return sistema_helpdesk.compilar()
```

Figura 48 Compilación del grafo del prototipo de Help Desk. Elaboración propia.

CAPÍTULO V: ANÁLISIS DE RESULTADOS

Este capítulo presenta el funcionamiento y la implementación del prototipo de sistema *Help Desk,* evaluando su comportamiento bajo determinados escenarios de operación hipotéticos. Las pruebas se realizan en un entorno empresarial simulado, en el cual el programador asume los roles de usuario y especialista del área, con el fin de ejecutar los respectivos test de prueba.

Dichas pruebas se desarrollan a partir de ejemplos de consultas comunes, en el que el usuario plantea su problemática y proporciona su correo electrónico como dato de autenticación y validación para el procesamiento del requerimiento. Posteriormente, la información ingresada se compara con la base de conocimiento previamente definida y se almacena en la base de datos vectorial. A partir de estos datos, el sistema recupera el contexto necesario para generar una respuesta con relación a la consulta del usuario.

Asimismo, se consideran consultas cuya información no guarda relación directa con la base de conocimiento. En ambos escenarios, se utilizan métricas de evaluación que permiten determinar el grado de confianza que tiene el sistema para responder al requerimiento. Si este valor supera el umbral establecido de 0.60, se genera automáticamente una respuesta y se da por finalizado el proceso de atención del ticket. En caso contrario, el requerimiento es escalado y se habilita un cuadro de texto para que el especialista del área proporcione una respuesta adecuada.

Durante este capítulo se describe el entorno de prueba, el contexto recuperado por el agente y la respuesta proporcionada por la IA mediante RAG en distintos escenarios de consultas simuladas. Posteriormente, se presenta la documentación correspondiente a los resultados de experimentación, así como el análisis del comportamiento de los LLM en aplicativos webs orientados a la gestión de servicios dentro del área de TI.

Los criterios de evaluación consideran aspectos como tiempos de respuesta, el nivel de confianza de la información recuperada y la reducción de la carga operativa del técnico.

5.1. Pruebas del sistema

Este apartado presenta las pruebas de funcionamiento del prototipo *Help Desk*, con el propósito de verificar que los principales componentes del sistema se integran de manera adecuada. Para ello, se contemplan diversos escenarios de consulta, incluyendo casos no relacionados con la base de conocimiento, con el fin de aportar más dinamismo y realismo al proyecto. De esta manera, se evalúa la capacidad del sistema para brindar soporte técnico.



Figura 49 Test de prueba N°1 (Ingreso de consulta). Elaboración propia.

Como se puede observar en la figura 49, se presenta la interfaz gráfica del usuario con el funcionamiento integrado del sistema. En esta pantalla se detalla el correo electrónico del usuario y la descripción del problema. Para este caso, se utiliza un ejemplo de consulta seleccionado desde el recuadro superior. Una vez ingresada la información, se envía la consulta y se espera unos segundos mientras se procesa.

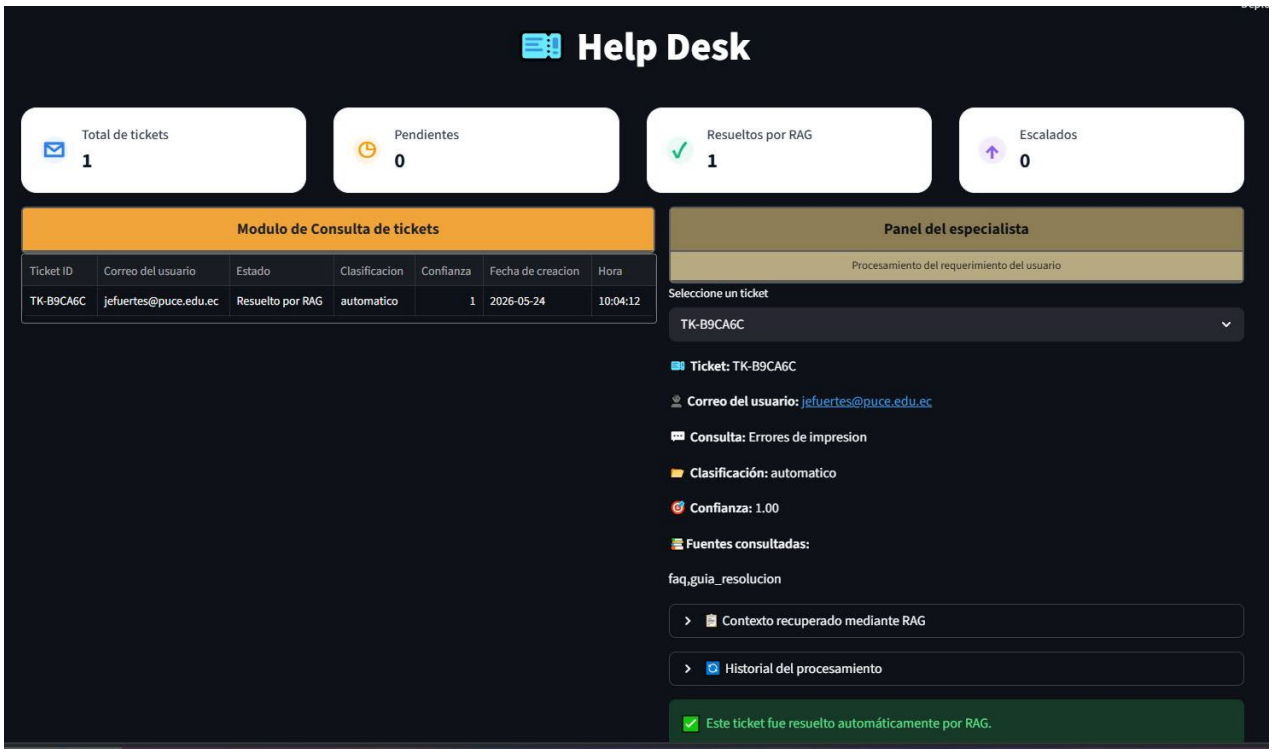


Figura 50 Test de prueba Nº1 (Gestión de la consulta). Elaboración propia.

En lo que respecta a la figura 50, se puede observar la interfaz gráfica correspondiente al especialista del área con el funcionamiento integrado del sistema. Lo más destacable es el estado del ticket, el cual se encuentra como resuelto por RAG. En otras palabras, la IA generó una respuesta automática debido a que contaba con un nivel de confianza suficiente para atender el requerimiento y no escalarla a un técnico del área.

Además, como se puede observar en el panel del especialista, al ticket se le asigna un código único, junto con un nivel de confianza de 1.0. También se muestran las fuentes de donde el sistema obtuvo la información para responder al requerimiento; en este caso, corresponden a los documentos faq y guía_resolucion.

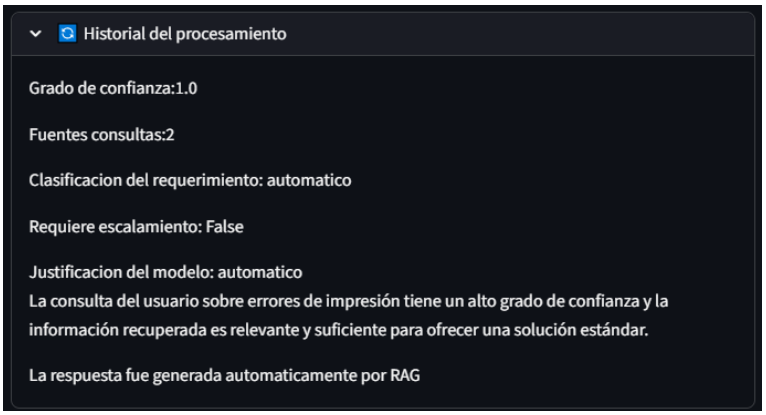


Figura 51 Test de prueba N°1 (Historial del procesamiento). Elaboración propia.

Lo que se observa en la figura 51 corresponde a información previamente presentada en la figura 48. Sin embargo, en este caso se incorpora la justificación del modelo, el cual explica la razón de la clasificación del requerimiento y, en consecuencia, determina el tipo de solución que debe proporcionarse. En este escenario hipotético, el tipo de justificación corresponde a una atención automática.

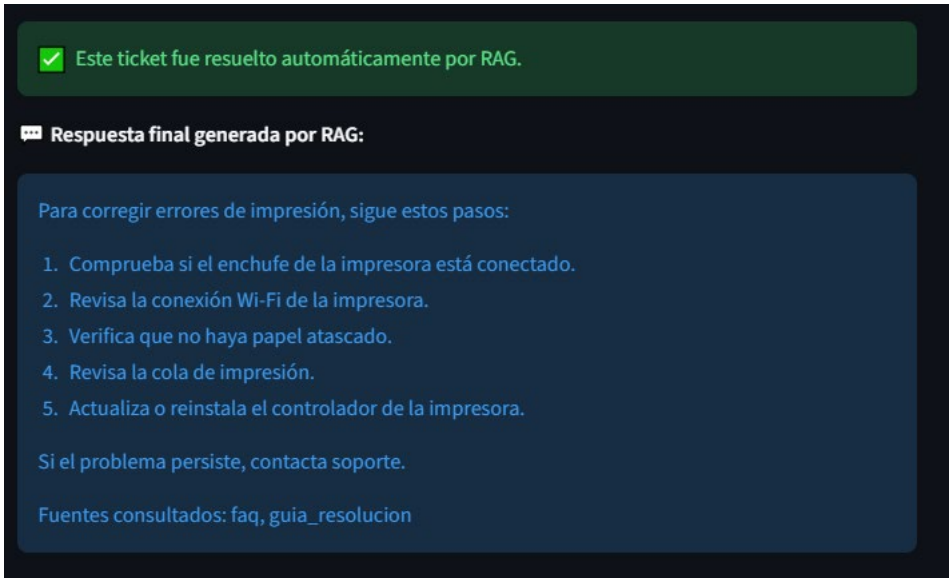


Figura 52 Test de prueba N°1 (Respuesta final generada por RAG). Elaboración propia.

Finalmente, la figura 52 presenta la respuesta final generada por el sistema, la cual es entregada al usuario como solución a su requerimiento. Cabe resaltar que el contexto recuperado mediante RAG solo se despliega cuando el nivel de confianza está por debajo de 0.60; por esta razón, en este caso no fue necesario incluirlo.

A continuación, se plantea una consulta más compleja, la cual representa un caso en el que es necesaria la intervención del técnico del área.



Figura 53 Test de prueba Nº2 (Ingreso de consulta). Elaboración propia.

Al igual que en la figura 47, la figura 53 detalla el correo electrónico del usuario y la descripción del problema. Sin embargo, a diferencia del caso anterior, no se selecciona un ejemplo de consulta del cuadro superior, sino que se plantea una consulta propia.

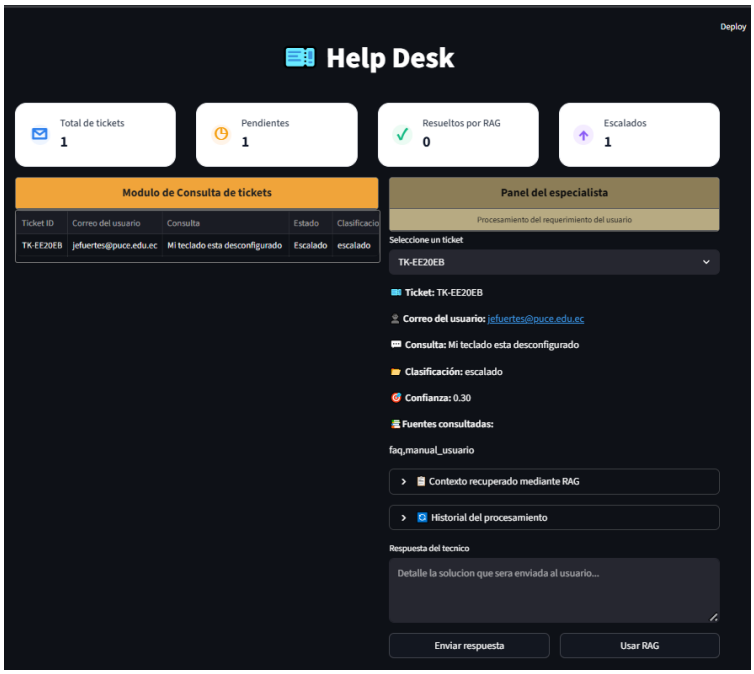


Figura 54 Test de prueba Nº2 (Gestión de la consulta). Elaboración propia.

La figura 54 presenta la misma información que la figura 48; sin embargo, en este caso, el nivel de confianza se encuentra por debajo del umbral del 0.60. Además, la clasificación del requerimiento aparece como “escalado”, lo que indica que es necesaria la intervención de un técnico para atender el caso de manera personalizada. De igual forma, en la tabla del módulo de

consulta de tickets, el estado del requerimiento se muestra como “escalado”. Una vez que el especialista envía la respuesta, el estado se actualiza y aparece como “resuelto por el especialista”.

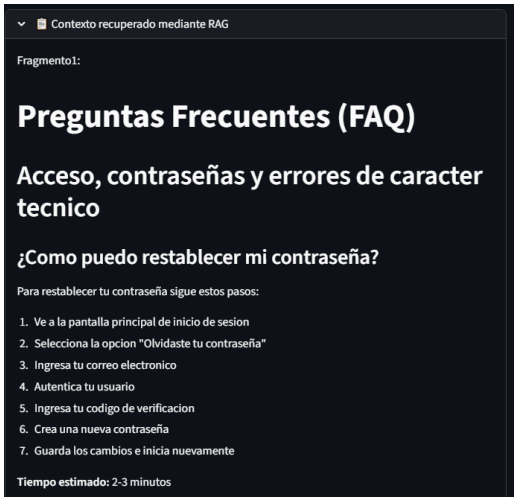


Figura 55 Test de prueba Nº2 (Contexto recuperado por RAG). Elaboración propia.

En referencia a la figura 55, se presenta el contexto proporcionado por el agente cuando el requerimiento es escalado. Esta información sirve como base de apoyo para que el especialista del área genere una respuesta coherente y acertada a la consulta del usuario.

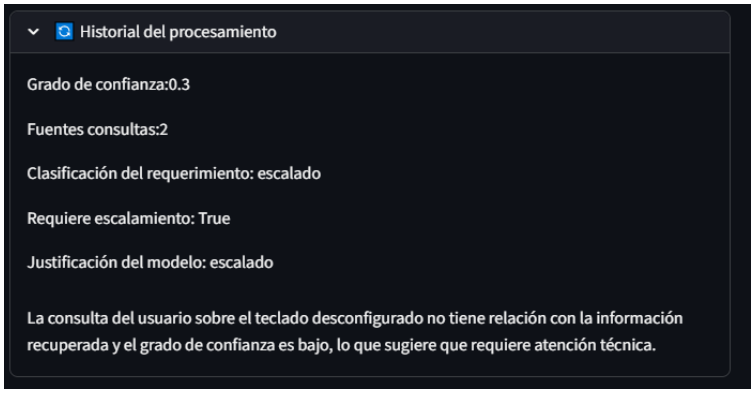


Figura 56 Test de prueba Nº2 (Historial del procesamiento). Elaboración propia.

Lo que se observa en la figura 56 corresponde a la justificación del modelo. En este escenario hipotético, dicha justificación corresponde a una atención escalada, debido a que no existe la suficiente confianza para responder al requerimiento.

A partir de las siguientes pruebas de funcionamiento, solo se incorpora el panel del especialista, ya que en este apartado se encuentra toda la información relacionada con la consulta, así como el proceso de asignación del ticket y la respuesta proporcionada por el sistema.

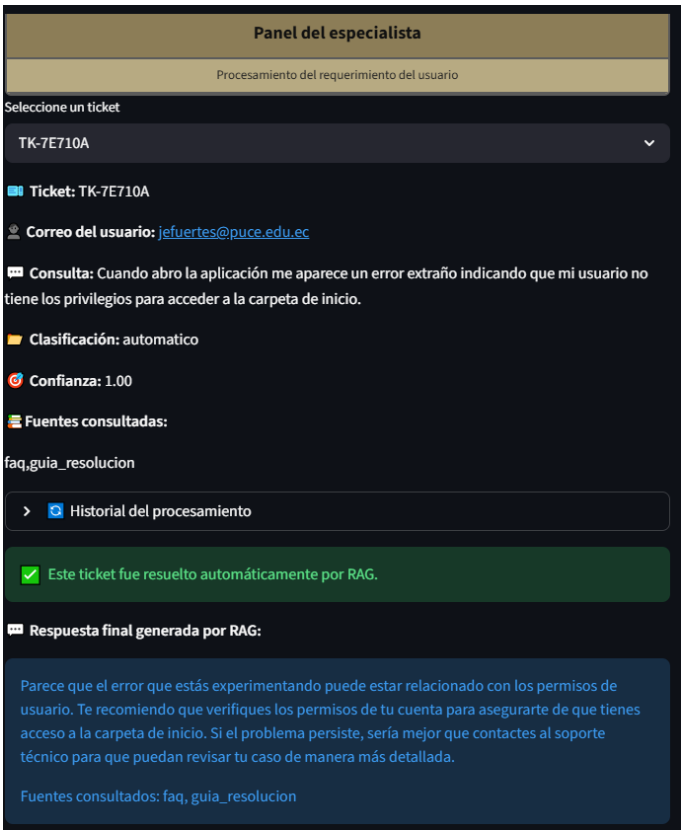


Figura 57 Test de prueba Nº3 (Panel del especialista). Elaboración propia.

Para este caso, la figura 57 presenta una consulta más compleja, en la que el usuario manifiesta problemas para acceder a una carpeta. Esta consulta contiene un mayor nivel de detalle en comparación con las dos consultas anteriores y genera una respuesta orientada a una recomendación sin una sucesión de pasos a seguir.

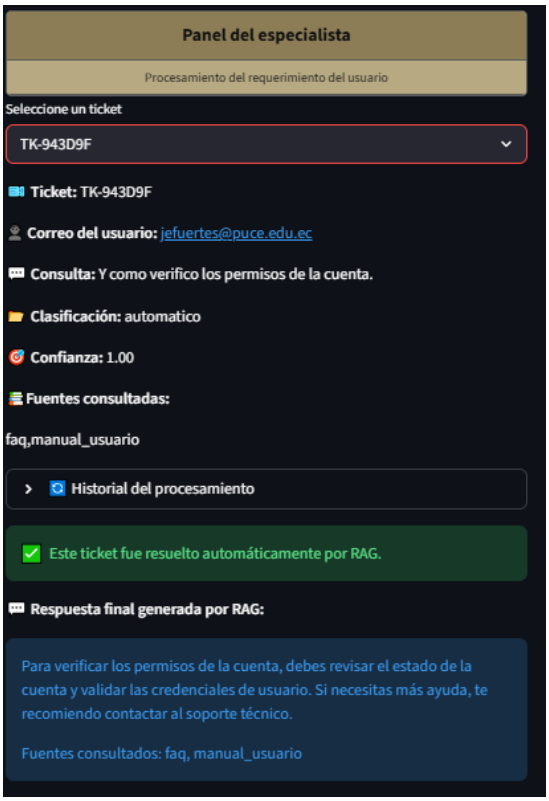


Figura 58 Test de prueba N°4 (Panel del especialista). Elaboración propia.

Lo que se observa en la figura 58 corresponde a una consulta realizada a partir de la respuesta anterior, en la que el usuario mantiene una duda sobre cómo verificar los permisos de la cuenta. Ante esta situación, el sistema genera una respuesta manteniendo el contexto de la consulta previa. En definitiva, esto demuestra que es posible plantear preguntas o dudas sobre requerimientos anteriores sin necesidad de especificar nuevamente el problema inicial.

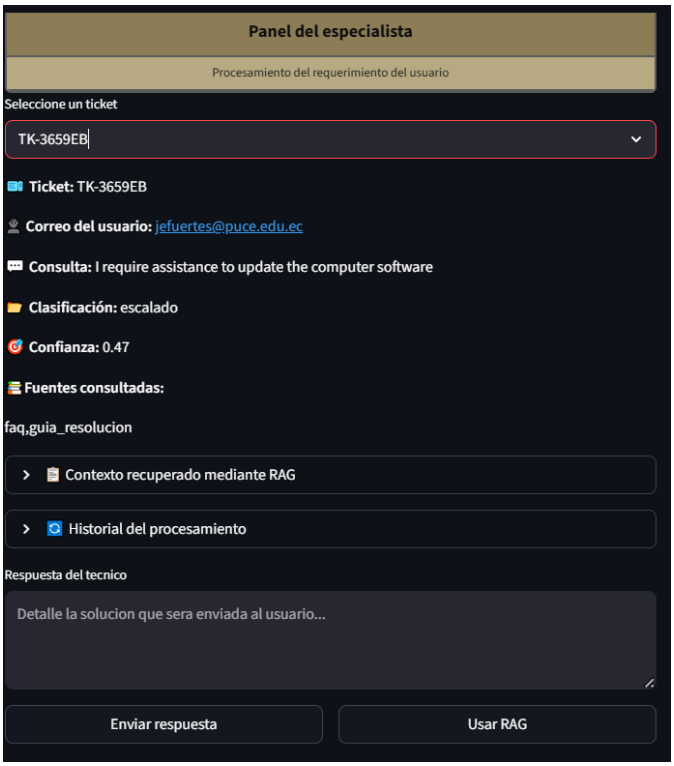


Figura 59 Test de prueba N°5 (Panel del especialista). Elaboración propia.

Como se puede observar en la figura 59, se realiza una consulta relacionada con la actualización del sistema operativo, pero redactada en inglés. Esta prueba tuvo como objetivo verificar si el sistema *Help Desk* podía interpretar consultas en otros idiomas. Sin embargo, en este escenario, no logró generar la respuesta automática, debido a que no presentaba el nivel de confianza suficiente para atender el requerimiento.

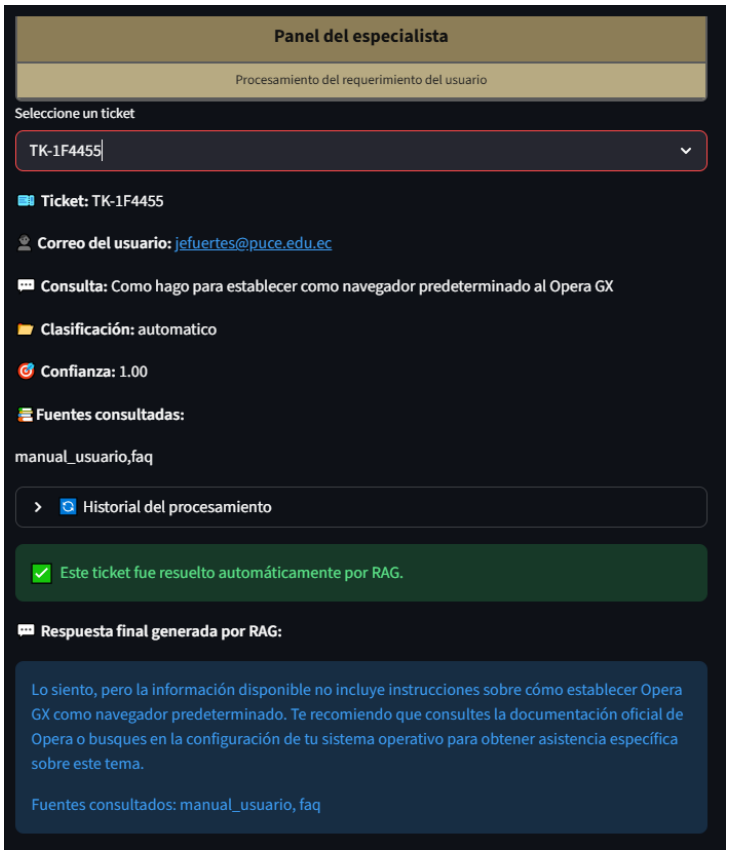


Figura 60 Test de prueba N°6 (Panel del especialista). Elaboración propia.

Respecto a la figura 60, se presenta una consulta cuya información no se encuentra disponible en la base de conocimientos establecida. En este escenario, se planteó una pregunta relacionada con el navegador Opera GX. Aunque el Help Desk alcanzó el nivel de confianza suficiente y generó una respuesta, esta no fue la esperada, ya que proporcionó una solución parcial al requerimiento y orientó al usuario a buscar información adicional en la documentación oficial de Opera.

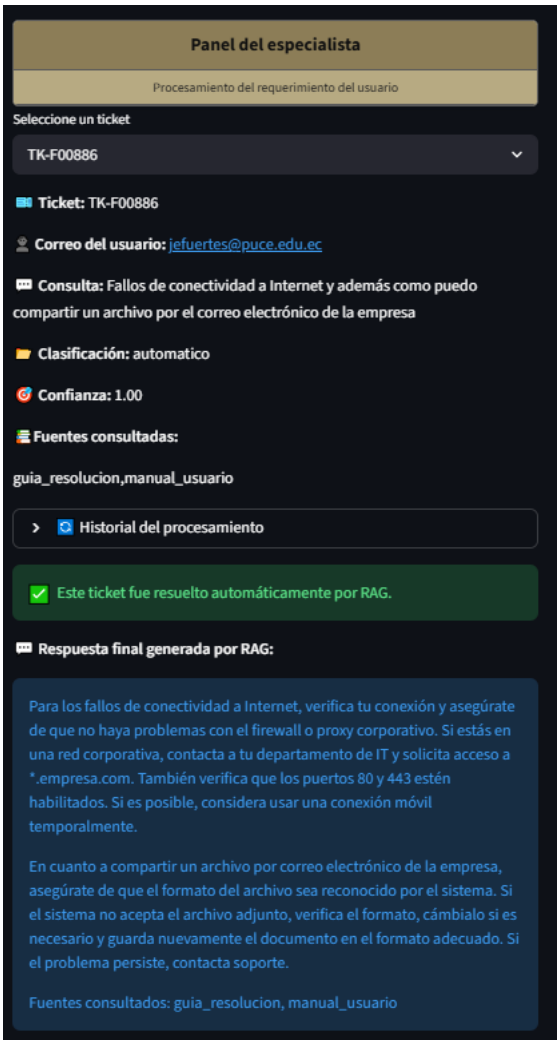


Figura 61 Test de prueba N°7 (Panel del especialista). Elaboración propia.

En la consulta presentada en la figura 61, se concatenaron dos peticiones diferentes sin relación directa entre ellas. El sistema Help Desk gestionó cada solicitud como un caso independiente y generó dos respuestas distintas, sin mezclar ambos contextos. Esto demuestra que es posible redactar más de una petición dentro de una misma consulta, siempre y cuando el aplicativo pueda identificar y fragmentar de manera correcta cada requerimiento.

5.1.1. Fortalezas y limitaciones del prototipo

A continuación, se describen las fortalezas y limitaciones observadas en el prototipo *Help Desk,* tomando como referencia la tabla 1. Cabe resaltar que las observaciones presentadas corresponden a los dos escenarios de prueba planteados durante la evaluación del sistema.

Tabla 1 *Fortalezas y limitaciones del prototipo de Help Desk.*

Fortalezas	Limitaciones
1.- El sistema responde a requerimientos simples sin la intervención de un especialista del área.	1.- El sistema depende estrictamente de una base de conocimientos bien definida para generar una respuesta coherente y contextualizada.
2.- Permite generar respuestas basadas en RAG.	2.- Si el contenido de la documentación está mal estructurado o no cuenta con la información suficiente, el contexto recuperado por RAG puede presentar inconsistencias.
3.- Permite clasificar las incidencias que ingresan al sistema entre casos automáticos y casos que requieren escalamiento.	3.- Un valor mal estimado en el criterio de la confianza puede causar errores en la clasificación de la incidencia.
4.- Cuando un requerimiento es escalado, el sistema proporciona al especialista del área el contexto de la consulta para facilitar la atención.	4.- En algunos casos, el contexto proporcionado puede no estar completamente relacionado con la consulta del usuario, por lo que no se debe depender exclusivamente de este resultado.
5.- Permite realizar un seguimiento del estado del ticket.	5.- El sistema requiere de una API privada para utilizar el modelo GPT, lo que representa un costo adicional en el desarrollo del sistema.

Elaboración propia.

5.2. Métricas de evaluación

Las métricas de evaluación de desempeño que se aplican al funcionamiento del prototipo Help Desk se basan en tres criterios principales: confiabilidad, tiempo de respuesta y resolución de incidencias. Mediante la tabla 2 se describe la función correspondiente de cada criterio.

Tabla 2 *Métricas de evaluación del prototipo Help Desk.*

Métrica de evaluación	***Función***	***Forma de medición***	***Interpretación***
Confiabilidad	Evalúa el nivel de certeza que tiene el sistema para responder a la consulta del usuario.	Se expresa como un valor de entre 0 y 1.	Valores inferiores a 0.6 el sistema escala el requerimiento a un técnico. Valores superiores a 0.6 el sistema genera una respuesta automática.
Tiempo de respuesta	Evalúa la eficiencia del sistema durante el proceso de gestión del requerimiento.	Se mide en segundos o minutos y depende de la capacidad de procesamiento del sistema.	Si el sistema cuenta con la información suficiente, responde el requerimiento en un corto lapso. En cambio, si el sistema no cuenta con la información suficiente, el requerimiento se escala a un técnico para su respectiva revisión.
Resolución de incidencias	Evalúa la cantidad de tickets resueltos correctamente, ya sea mediante RAG o por el técnico del área.	Se mide mediante la comparación entre el total de tickets generados y el total de tickets resueltos.	Se registró un total de 10 tickets en el sistema. De ellos, 2 quedaron pendientes, 7 fueron resueltos mediante RAG y 1 fue resuelto por especialista.

Nota. Elaboración propia.

5.2.1. Cálculo de la confianza

Para calcular el nivel de confianza obtenido por el sistema se establece una ecuación general, la cual se detalla en la tabla 3:

Tabla 3 *Ecuación de estimación de la confianza.*

Variables	*Ecuación*
P: *Número de palabras relevantes de la consulta.* ***D:*** *Número de documentos recuperados por RAG.* ***Mi:*** *Número de coincidencias encontradas entre la consulta del usuario y el documento recuperado i.* ***Ti:*** *Número total de palabras del documento recuperado i.*	$Confianza = Confianza_{base} + Bonus_{documento} + Bonus_contenido$ $Confianza_{base} = \frac{\sum_{i=1}^{3} M_i}{P}$ $Bonus_{documentos} = \frac{D}{4}$ $Bonus_{contenido} = \frac{\sum_{i=1}^{3} T_i}{1000}$

Elaboración propia.

Como primer paso, se definen las variables que intervienen en el cálculo de la confianza. Entre ellas, la más relevante es la variable "D", ya que representa el número de documentos recuperados desde la base de conocimiento del sistema. Ante la falta de documentos, el resto de los cálculos pierden validez, debido a la falta de información para establecer una comparación con la consulta del usuario. Cabe resaltar que la ecuación general está compuesta por tres sub-ecuaciones: la confianza base, el bonus_contenido y el bonus_documento.

La ecuación confianza_base representa la suma de las coincidencias (M_i), desde el documento 1 hasta el documento 3, dividido para el conjunto de palabras, representado por (P).

Las ecuaciones bonus_contenido y bonus_documento corresponden a cálculos adicionales que aumentan el nivel de confianza final del sistema dentro de un valor límite.

En lo que respecta a la ecuación bonus_documento, esta estima un valor proporcional a la cantidad de documentos recuperados. Esto quiere decir que, mientras más alto sea el número de documentos recuperados, mayor será el bonus proporcionado; sin embargo, el resultado no debe sobrepasar el valor de 0.2. Este límite se establece en la función _calcular_confianza dentro del archivo sistema_rag.py.

Del mismo modo, la ecuación bonus_contenido estima un valor proporcional a la cantidad de texto recuperado. Para ello, se realiza la suma total de las palabras presentes en los tres primeros documentos recuperados y se usa el valor 1000 para normalizar dicha cantidad de texto. En otras palabras, mientras más alto sea el número de coincidencias encontradas, mayor será el resultado obtenido; no obstante, este valor no debe sobrepasar el 0.1. Al igual que en la ecuación bonus_documento, este límite se establece dentro de la misma función del archivo descrito previamente.

Para obtener el resultado final de la confianza, el valor calculado se redondea a un máximo de dos decimales.

5.2.2. Evaluación del tiempo de respuesta y resolución de incidencias del prototipo

Para medir el tiempo de respuesta y la resolución de incidencias, se ejecutan diez consultas diferentes como punto de referencia. A partir de estas pruebas, se elaboran dos gráficas que permiten visualizar el desempeño del sistema.

Modulo de Consulta de tickets

Ticket ID	Correo del usuario	Consulta
TK-8FFED0	jefuertes@puce.edu.ec	Perdida o recuperacion de datos
TK-D17685	jefuertes@puce.edu.ec	Mi teclado esta desconfigurado
TK-36AAD7	jefuertes@puce.edu.ec	Cambios en los roles y permisos del usuario
TK-8CACF5	jefuertes@puce.edu.ec	Cuando trato de conectarme a una pagina me aparece un error relacionado con lo co
TK-1F4455	jefuertes@puce.edu.ec	Como hago para establecer como navegador predeterminado al Opera GX
TK-5247BD	jefuertes@puce.edu.ec	El monitor de mi computadora no funciona
TK-F00886	jefuertes@puce.edu.ec	Fallos de conectividad a Internet y además como puedo compartir un archivo por el
TK-3659EB	jefuertes@puce.edu.ec	I require assistance to update the computer software
TK-7E710A	jefuertes@puce.edu.ec	Cuando abro la aplicación me aparece un error extraño indicando que mi usuario no
TK-366338	jefuertes@puce.edu.ec	Las credenciales de la empresa para iniciar sesión están fallando

Figura 62 Módulo de consulta de tickets (Pruebas del sistema). Elaboración propia.

La figura 62 muestra una tabla en la que se registra el total de consultas realizadas por el usuario, así como la información relacionada con el proceso de gestión del requerimiento. Estos datos sirven como base para la elaboración de las gráficas correspondientes al tiempo de respuesta y a la resolución de incidencias.

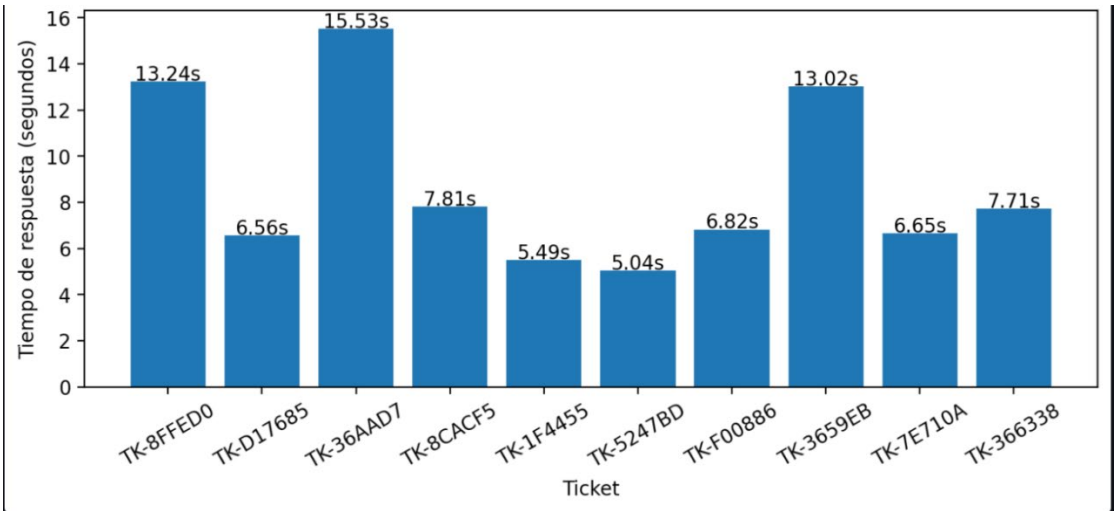


Figura 63 Tiempo de respuesta por ticket. Elaboración propia.

La figura 63 muestra un gráfico de barras correspondiente a la métrica de evaluación del tiempo de respuesta. En este gráfico se presenta el código del ticket asociado a cada consulta, junto con su respectivo tiempo de procesamiento. Para su elaboración, se utilizaron las librerías time, encargada de calcular el tiempo de respuesta, y matplotlib, empleada para representar los resultados de manera gráfica.

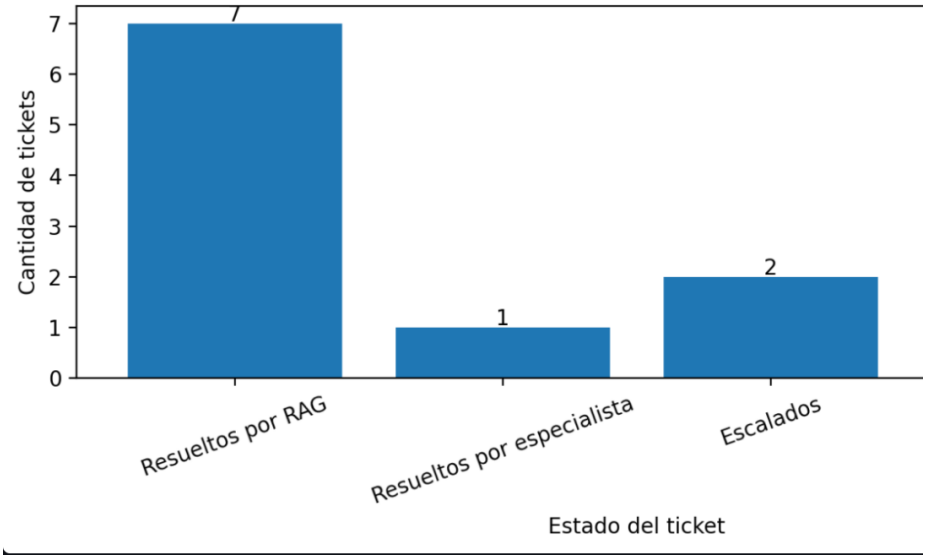


Figura 64 Resolución de incidencias. Elaboración propia.

La figura 64 muestra el gráfico de barras correspondiente a la métrica de resolución de incidencias. En este gráfico se presenta la cantidad de tickets resueltos por RAG, así como aquellos resueltos por el especialista. Además, se observa que dos tickets aún se mantienen en

estado de escalado, por lo que se encuentran a la espera de la respuesta por parte del especialista del área.

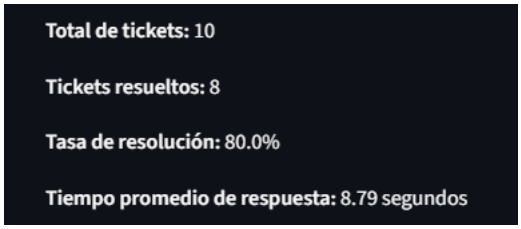


Figura 65 Resumen de resultados obtenidos. Elaboración propia.

Finalmente, en la figura 65 se muestra un resumen general de las métricas de evaluación aplicadas al prototipo de *Help Desk*. En este caso, se registró un total de 10 tickets, de los cuales 8 fueron atendidos con éxito, lo que representa una tasa de resolución del 80%. Además, se presenta el tiempo promedio en el que el sistema proporciona una respuesta al usuario, excluyendo los casos que todavía se mantienen en espera de atención.

Estos resultados evidencian un desempeño adecuado del sistema en el proceso de gestión y atención de las consultas. Tanto la tasa de resolución como el tiempo promedio de respuesta se obtienen mediante operaciones matemáticas básicas.

CAPÍTULO VI: CONCLUSIONES Y RECOMENDACIONES

Este capítulo presenta un resumen sucinto de los resultados obtenidos durante el desarrollo del prototipo *Help Desk* basado en LLM, así como los hallazgos identificados durante su implementación. Además, se busca validar cada uno de los objetivos propuestos mediante conclusiones que recopilen la información más relevante en torno al proyecto planteado, el cual integra herramientas y funcionalidades prácticas en el campo de la inteligencia artificial.

Asimismo, se destacan tanto las limitaciones y deficiencias encontradas como las fortalezas y aportes que el sistema puede brindar a proyectos de mayor escala, a pesar de su simplicidad. Adicionalmente, se concluye el proceso de análisis del comportamiento del sistema bajo determinados escenarios de prueba, los cuales permiten evaluar el rendimiento en términos de tiempo de respuesta y confiabilidad. Dichos parámetros proveen valores esenciales para medir el desempeño y establecer un flujo de trabajo que se adapte mejor a las necesidades de las empresas.

También se establecen recomendaciones y sugerencias orientadas al mejoramiento continuo del prototipo, considerando aspectos relacionados con la incorporación de nueva información en la base de conocimiento, el aumento del número de variables para formular una ecuación de confianza más sólida, la integración de filtros de seguridad que no comprometan la información sensible de la empresa, así como la adaptación e implementación del sistema en entornos reales de trabajo.

Durante este capítulo, se busca sintetizar los resultados alcanzados y proyectar una visión más amplia sobre los algoritmos de inteligencia artificial y su rol en aplicativos webs orientados al soporte técnico. Esto tiene como finalidad consolidar fuentes de información más coherentes y contextualizadas durante el proceso de atención del requerimiento.

6.1. Conclusiones

El prototipo del sistema *Help Desk* desarrollado tuvo como objetivo integrar herramientas basadas en inteligencia artificial dentro de un aplicativo web orientado al área de TI de una empresa. Para ello, se emplearon técnicas como RAG y procesamiento de lenguaje natural, con el fin de mejorar los procesos relacionados con la gestión de requerimientos.

En términos generales, se puede afirmar que la ejecución de la aplicación fue satisfactoria, ya que se aplicaron métricas de evaluación para medir su desempeño y, en su mayoría, los resultados obtenidos fueron positivos. Esto se evidenció, teniendo en consideración el contexto, la confianza y el estado de cada consulta realizada por el usuario.

De igual manera, existe un margen de mejora considerable en comparación a sistemas *Help Desk* tradicionales. Sin embargo, dada la manera en que fue desarrollado el prototipo, su escalabilidad resultó accesible, debido a los niveles bajos de esfuerzo técnico que requiere para adaptarse o ampliarse.

El sistema se compuso de dos subsistemas principales: el subsistema del *chatbot* y el subsistema de gestión de incidencias. Ambos permitieron automatizar el proceso de creación y asignación de tickets. En este caso, se utilizó un *chatbot* contextual con la capacidad de responder a las preguntas realizadas por el usuario, apoyándose en un modelo que integra procesamiento de lenguaje natural. Su objetivo fue identificar los patrones de búsqueda más comunes e interpretarlos para facilitar el análisis del requerimiento.

En este contexto, la aplicación también cumplió con el desarrollo de un módulo de gestión de incidencias, encargado de administrar cada ticket generado a partir de una consulta. Este objetivo se logró mediante el diseño de una base de conocimiento que acepta únicamente archivos con extensión .md, lo cual permite optimizar el almacenamiento en la base de datos vectorial.

Dicha base de datos contribuyó al procesamiento de datos no estructurados provenientes de la consulta del usuario, ya que, al incorporar un principio basado en búsqueda semántica, permitió fragmentar la información ingresada y convertirla en vectores. Esto facilitó la identificación de similitudes dentro de la base de conocimiento y la generación de una respuesta automática.

Aplicar un flujo de trabajo al sistema *Help Desk* permitió establecer una estructura que evidencia la gestión del requerimiento del usuario. Este proceso contempla la asignación del

ticket y su clasificación con base en el nivel de confianza, convirtiendo dicho valor en un factor clave para determinar el tipo de atención que debe ejecutarse, ya sea mediante la generación de una respuesta automática con RAG o mediante el escalamiento al especialista del área. De esta manera, se cumple el objetivo planteado y, consecuentemente, se logra simular un proceso de soporte técnico similar al que podría usarse en una empresa, con mayor adaptabilidad a distintos escenarios.

De igual forma, se puede afirmar que el uso de RAG en este tipo de aplicaciones representa una pieza clave debido a sus capacidades avanzadas de respuesta. Además, esta técnica se complementa adecuadamente con herramientas como LangGraph y LangChain, lo que permite reducir costos asociados al desarrollo e implementación. Esto abre nuevas posibilidades de aplicación tanto en ámbitos académicos como en entornos empresariales de carácter general.

Y en relación con la meta establecida de gestionar mas del 50% de las incidencias en el marco de trabajo del área de TI, se demostró, a partir de los escenarios de prueba realizados, que el sistema alcanzo una tasa de resolución del 80%. Este resultado evidencia que la integración de inteligencia artificial permitió atender la mayor parte de los requerimientos dentro del entorno simulado. Por ende, el prototipo no solo cumplió el propósito inicial de este proyecto, sino que también presentó un desempeño favorable en la automatización y gestión de incidencias.

6.2. Recomendaciones

Además de los resultados obtenidos, se consideran ciertos aspectos orientados a la mejora continua del sistema en futuras etapas de extensión y actualización. Dado que este prototipo fue planteado como una versión reducida de un sistema *Help Desk* empresarial, su desarrollo permitió mejorar algunos aspectos relacionados con la gestión y atención de consultas dentro del área de TI. En este sentido, se priorizaron funcionalidades estrechamente vinculadas con la IA, con la finalidad de mejorar la efectividad del prototipo.

Es importante mencionar que, para llevar el sistema Help Desk a producción, se recomienda ampliar la base de conocimiento. Esto no solo permitirá generar respuestas más coherentes y contextualizadas, sino también mejorar la calidad de información proporcionada al usuario. Además, sería conveniente incorporar varios tipos de archivos, no únicamente documentos que contengan texto, sino también imágenes o audios.

De esta manera, una captura de pantalla o un recurso multimedia podría ser interpretado por el sistema sin que el usuario tenga que redactar completamente el problema, facilitando respuestas más adaptadas a sus necesidades.

Otra recomendación consiste en establecer un criterio de confianza más sólido y complejo que el utilizado actualmente. Por motivos de optimización, la ecuación encargada de calcular la confianza se limita principalmente a la cantidad de documentos incorporados en la base de conocimiento. Por esta razón, si en el futuro se desea realizar una actualización o ampliar la documentación, también será necesario ajustar dicha ecuación en función de los cambios realizados, lo que podría convertirse en una tarea laboriosa si no se automatiza correctamente.

También se recomienda utilizar versiones de LLM más actuales. En este proyecto se trabajó con un modelo GPT-4 en su versión mini, debido a que no se requería procesar grandes volúmenes de datos. Sin embargo, en entornos corporativos más robustos, el uso de modelos más avanzados se convierte en una necesidad, ya que permitiría definir *prompts* más detallados y alineadas con las reglas establecidas por el área de TI de la empresa.

Asimismo, se deben considerar los costos adicionales asociados al uso de estos modelos, ya que se requiere una API privada para ejecutar cada consulta. No obstante, desde una perspectiva costo-beneficio, esta inversión no representaría un gasto excesivo, considerando que podría reducir la contratación de personal técnico adicional, y mejorar significativamente la automatización de procesos.

Finalmente, es importante destacar que este prototipo está diseñado con base en una arquitectura por capas, lo que lo convierte en una aplicación versátil y sujeta a mejoras continuas, tanto en su *frontend* como en su *backend*. Teniendo esto en consideración, la aplicación no debe entenderse como un producto listo para llevar a producción, sino más bien como una propuesta que sirva de guía y ejemplo para el desarrollo de sistemas más especializados dentro de flujos de trabajo más complejos.

Como observación adicional, el sistema requiere una conexión a internet estable para su funcionamiento.

Con base en los resultados obtenidos, así como en las limitaciones y fortalezas observadas, se considera necesario ampliar el campo de estudio y evaluar su aplicación en condiciones más reales, con el fin de validar su adaptabilidad y funcionamiento.

Aunque existen documentos o informes relacionados con las herramientas de inteligencia artificial incorporadas en este prototipo, su libre acceso es todavía limitado y su difusión no es suficientemente amplia. Por ello, resulta necesario promover el desarrollo de nuevas publicaciones de carácter científico, que permitan ampliar la perspectiva que se tiene sobre la inteligencia artificial y redefinir su enfoque en áreas orientadas al soporte técnico.

BIBLIOGRAFÍA

Acosta Gonzaga, E., Álvarez Cedillo, J. A., & Gordillo Mejía, A. (2006). *Arquitecturas en N-Capas: Un sistema adaptivo*. *Polibits*, 34, 34–37. https://doi.org/10.17562/PB-34-7

Alija, A. (2024, 24 de enero). RAG - *Retrieval Augmented Generation: La llave que abre la puerta de la precisión a los modelos del lenguaje*. datos.gob.es. https://datos.gob.es/es/blog/rag-retrieval-augmented-generation-la-llave-que-abre-la-puerta-de-la-precision-los-modelos-del

García Barrena, J., & Borrego Díaz, J. (2024). *Evaluación de guardarraíles de modelos LLM ante la generación de artículos sesgados*. En Investigaciones DACIU. Ingeniería y Arquitectura (n.º 2, pp. 66–76). Fundación Avanza. https://hdl.handle.net/11441/171372

Berrospi Casano, A. (2024, 28 de febrero). *Bases de datos vectoriales: Teoría y práctica con ChromaDB y LangChain*. Medium. https://medium.com/datapath/bases-de-datos-vectoriales-teor%C3%ADa-y-pr%C3%A1ctica-con-chromadb-y-langchain-0140ab635a24

García Brustenga, G., Fuertes-Alpiste, M., & Molas-Castells, N. (2018). *Briefing paper: Los chatbots en educación*. eLearn Center, Universitat Oberta de Catalunya. https://openaccess.uoc.edu/handle/10609/85786

Jaramillo Castro, C. M., & Morocho Puchaicela, D. A. (2016). *Sistema Help Desk, utilizando ITIL para la provisión del servicio en el departamento de mantenimiento y soporte técnico de la Universidad Nacional de Loja*. Revista Tecnológica - ESPOL, 29(1), 155–169. https://rte.espol.edu.ec/index.php/tecnologica/article/view/514

JetBrains. (s. f.). *Download PyCharm: The Python IDE for data science and web development*. Recuperado el 11 de mayo de 2026, de https://www.jetbrains.com/pycharm/download/

Codificando Bits. (s. f.). *¿Qué es la búsqueda semántica?* [Video] YouTube. https://www.youtube.com/watch?v=l_fdkLQ8s2s

Cohen Solano, K. (2023). *Recuperación Aumentada Generativa (RAG) para la creación de chatbot de dominio específico* [Trabajo de grado de pregrado, Universidad de los Andes]. Repositorio Institucional Séneca. https://hdl.handle.net/1992/73530

Colle, R. (2017). *Algoritmos, grandes datos e inteligencia en la red: Una visión crítica*. Universidad de Alicante. https://doi.org/10.14198/MEDCOM/2017/9_cmd

European Information Technologies Certification Institute. (s. f.). *Knowledge representation and reasoning*. https://eitc.org/research-opportunities/new-media-and-new-digital-economy/ai-machine-learning-deep-learning-and-neural-networks/ai-research-and-applications/knowledge-representation-reasoning-and-logic/knowledge-representation-and-reasoning

Fernández Zarza, I. (2026). *Desarrollo de un agente de inteligencia artificial en una plataforma financiera multiagente* [Trabajo de fin de grado, Universitat Autònoma de Barcelona]. Dipòsit Digital de Documents de la UAB. https://ddd.uab.cat/record/326556

Martín Hernández, S. (2022). *Herramienta para el análisis de datos en Twitter* [Trabajo de fin de grado, Universidad de Salamanca]. Gredos. https://hdl.handle.net/10366/151366

Herrero-Diz, P., & Varona-Aramburu, D. (2018). *Uso de chatbots para automatizar la información en los medios españoles*. El Profesional de la Información, 27(4), 742–749. https://doi.org/10.3145/epi.2018.jul.03

Muns, R. (1993). The Help Desk Handbook: *The Help Desk Institute guide to help desk operations and problem management*. Help Desk Institute.

Pan, J. J., Wang, J., & Li, G. (2024). *Survey of vector database management systems*. The VLDB Journal, 33, 1591–1615. https://doi.org/10.1007/s00778-024-00864-x

Dávila Romero, M. Á. (2025). *Agente LLM para la recolección de datos* [Trabajo de fin de máster, Universidad Pontificia Comillas]. Repositorio Comillas. http://hdl.handle.net/11531/96780

Rivero Zazo, I. (2024). *Inteligencia artificial aplicada a la generación de contenidos didácticos en Física y Química para 3.º de la ESO*.

Roncancio Roncancio, J. E., & Mesa Páez, L. O. (2011). *Business case para la definición y caracterización del catálogo de servicios de la mesa de ayuda en la empresa BASIC 3000 a través de ITIL Foundations V.3* [Proyecto de grado, Universidad Libre]. Repositorio Institucional Unilibre. https://hdl.handle.net/10901/8844

Figueroa Sacoto, S. S. (2021). *Diseño y desarrollo de un chatbot usando redes neuronales recurrentes y procesamiento de lenguaje natural para tiendas virtuales en comercio electrónico* [Tesis de grado, Universidad Politécnica Salesiana]. Repositorio Institucional de la Universidad Politécnica Salesiana. http://dspace.ups.edu.ec/handle/123456789/21195

Hernández Suárez, A. (2024). *Sistema de búsqueda inteligente aplicando Retrieval-Augmented Generation* [Tesis de maestría, Universidad Politécnica de Madrid]. Archivo Digital UPM. https://oa.upm.es/83447/

Wang, J., & Duan, Z. (2024). *Agent AI with LangGraph: A modular framework for enhancing machine translation using large language models*. arXiv. https://arxiv.org/abs/2412.03801