# IMPLEMENTACIÓN DE UN PROTOTIPO DE ASISTENTE INTELIGENTE BASADO EN IA PARA LA AUTOMATIZACIÓN Y MEJORA DEL SOPORTE TÉCNICO ESPECIALIZADO

**YEISON DARIEL ZAPATA MONSALVE**

**INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO**

**FACULTAD DE INGENIERÍA**

**TECNOLOGÍA EN DESARROLLO DE SOFTWARE**

**NOVIEMBRE DE 2025**

# IMPLEMENTACIÓN DE UN PROTOTIPO DE ASISTENTE INTELIGENTE BASADO EN IA PARA LA AUTOMATIZACIÓN Y MEJORA DEL SOPORTE TÉCNICO ESPECIALIZADO

**YEISON DARIEL ZAPATA MONSALVE**

**Trabajo de grado para optar al título de**

**TECNÓLOGO EN DESARROLLO DE SOFTWARE**

**Asesor**

**JAVIER ALBERTO SALDARRIAGA AGUIRRE**

**Magister en Entornos Virtuales de Educación**

**INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO**

**FACULTAD DE INGENIERÍA**

**TECNOLOGÍA EN DESARROLLO DE SOFTWARE**

**NOVIEMBRE DE 2025**

# Contenido

# Tabla de ilustraciones

# Lista de Tablas

## Glosario

1. Algoritmo:

Conjunto de instrucciones lógicas y estructuradas que permiten resolver un problema o ejecutar una operación específica dentro de un sistema computacional.

2. API (Interfaz de Programación de Aplicaciones):

Conjunto de rutas y protocolos que permiten la comunicación entre diferentes componentes de software. Alma utiliza una API basada en FastAPI para recibir consultas y entregar respuestas.

3. Asistente Inteligente:

Sistema capaz de procesar lenguaje humano, interpretar consultas y generar respuestas automáticas mediante técnicas de inteligencia artificial.

4. Backend:

Capa lógica y funcional del sistema que gestiona la comunicación con el modelo de lenguaje, la base de datos y los servicios internos. En Alma está implementado con FastAPI.

5. Base de Conocimiento:

Conjunto de datos estructurados que contienen información técnica para resolver consultas. Alma utiliza faqs.csv y soporte.db como fuentes principales.

6. Búsqueda Semántica:

Método de recuperación de información que interpreta el significado del texto en lugar de realizar coincidencias literales. Alma utiliza FAISS para lograrlo.

7. Cloudflare Tunnel:

Servicio que permite exponer aplicaciones locales a internet de forma segura sin abrir puertos. Alma usa el túnel “alma” vinculado al dominio https://alma.almaai.online/

8. Consulta (Query):

Petición realizada por el usuario al asistente, ya sea por texto o voz, para obtener una respuesta técnica.

9. Embeddings:

Representaciones vectoriales del texto que permiten medir similitud semántica. Alma los genera para recuperar respuestas relevantes.

10. Endpoint:

Ruta específica de la API que recibe o envía información. Ej.: /consulta, /estado.

11. Entorno Virtual (.venv):

Espacio aislado donde se instalan las dependencias de Python para evitar conflictos con el sistema operativo.

12. FAISS (Facebook AI Similarity Search):

Biblioteca optimizada para búsqueda de vectores a gran escala. Utilizada para la búsqueda semántica en Alma.

13. FastAPI:

Framework moderno de Python orientado a la creación de APIs rápidas y eficientes. Constituye el núcleo del backend del prototipo.

14. Frontend:

Interfaz gráfica con la que interactúa el usuario. En Alma está contenida en la carpeta ui/ y desarrollada con HTML, CSS y JavaScript.

15. IA (Inteligencia Artificial):

Disciplina que permite que sistemas informáticos ejecuten tareas que normalmente requieren inteligencia humana, como razonamiento, comprensión y aprendizaje.

16. Índice Vectorial:

Estructura que almacena embeddings para permitir búsqueda eficiente. En Alma es el archivo index.faiss.

17. Interfaz Conversacional:

Modelo de interacción que permite al usuario comunicarse mediante texto o voz en un entorno similar a un chat.

18. LLaMA:

Familia de modelos de lenguaje desarrollados por Meta AI. Alma utiliza LLaMA 3 8B a través de Ollama.

19. Machine Learning:

Rama de la IA que permite a los sistemas aprender patrones a partir de datos y mejorar su rendimiento con el tiempo.

20. Modelo de Lenguaje (LLM):

Algoritmo entrenado para comprender y generar texto. Es responsable de construir respuestas coherentes y técnicas en Alma.

21. Ollama:

Plataforma que permite ejecutar modelos de lenguaje localmente, sin depender de la nube. Alma usa Ollama para correr LLaMA 3.

22. Pipeline de Procesamiento:

Flujo de operaciones necesarias para transformar una consulta en una respuesta. En Alma incluye: texto → embedding → FAISS → contexto → LLM → respuesta.

23. Procesamiento de Lenguaje Natural (PLN):

Tecnología que permite que las máquinas comprendan y generen lenguaje humano.

24. Prototipo:

Versión funcional inicial de un sistema que permite evaluar su viabilidad y desempeño antes de construir un sistema final.

25. Recuperación de Información:

Proceso mediante el cual el sistema encuentra contenido relevante a partir de la base de conocimiento.

26. Reconocimiento de Voz:

Tecnología que convierte audio en texto. Alma integra Web Speech API para procesar consultas habladas.

27. Respuesta Técnica:

Salida generada por el asistente que resuelve o guía la solución de un problema reportado por el usuario.

28. SQLite:

Motor de base de datos ligero utilizado en Alma para almacenar información estructurada de soporte técnico.

29. Túnel Seguro:

Mecanismo que crea un canal cifrado entre un equipo local y un dominio externo sin necesidad de abrir puertos. Implementado mediante Cloudflare Tunnel en Alma.

30. Uvicorn:

Servidor ASGI de alto rendimiento utilizado para ejecutar aplicaciones FastAPI, permitiendo comunicación en tiempo real con el frontend.

## Resumen

IMPLEMENTACIÓN DE UN PROTOTIPO DE ASISTENTE INTELIGENTE BASADO EN IA PARA LA AUTOMATIZACIÓN Y MEJORA DEL SOPORTE TÉCNICO ESPECIALIZADO

YEISON DARIEL ZAPATA MONSALVE

En este trabajo de grado presento el proceso y desarrollo del proyecto Alma, un prototipo de asistente inteligente diseñado con el objetivo de apoyar el soporte técnico nivel 1 y 2 de TI. La motivación principal surge de mi propia experiencia dentro de equipos de TI, donde es evidente la acumulación de tikets, cuya mayoría son repetitivos y se rebosan los equipos de soporte y los softwares de control de los mismos. Casi siempre los reportes de los usuarios son fallas menores, básicas y ocupan tiempo valioso del personal técnico que bien podría concentrarse en atender fallas más especializadas o complejas. En este contexto, se identificó la necesidad de contar con un software, aplicativo, programa y/o herramienta capacitada para automatizar parte de la operación TI sin comprometer el nivel de servicios.

Desde el anteproyecto plantee una solución basada en inteligencia artificial y procesamiento de lenguaje natural (PLN), con capacidad de interpretar preguntas realizadas de manera verbal o escritas, esto basado en un conjunto de conocimiento propio y ordenado, así como la ayuda semántica que permita en corto tiempo arrojar respuestas claras y eficaces. Es esto una investigación aplicada, pues combina el análisis del problema, el levantamiento de requerimientos funcionales, el diseño de la arquitectura del sistema y un desarrollo de prototipado iterativo, que permite ajustar la operación de Alma a las necesidades reales del

entorno TI.

Antes de entrar en materia es necesario explicar que he desarrollado el prototipo con tres procesos o partes fundamentales, el primero integrando un backend construido en FastAPI, el cual es un modelo de lenguaje para el procesamiento semántico, el segundo una base de conocimiento estructurada a partir de experiencias de soporte técnico, y el último, no menos importante una interfaz web y visual que permite la interacción tanto en texto como con comandos de voz.

Los resultados obtenidos con Alma, dan cuenta de la capacidad para resolver consultas frecuentes, disminuir tiempo de respuesta y carga operativa del personal técnico. En conclusión, este proyecto confirma la viabilidad de implementar asistentes con IA, para modernizar, agilizar y ofrecer soporte técnico de calidad; Alma es adaptable, escalable y se alinea con las demandas actuales del mercado TI.

.

## Abstract

IMPLEMENTATION OF AN AI-BASED INTELLIGENT ASSISTANT PROTOTYPE FOR THE AUTOMATION AND IMPROVEMENT OF SPECIALIZED TECHNICAL SUPPORT

YEISON DARIEL ZAPATA MONSALVE

This undergraduate thesis presents the process and development of *Alma*, a prototype of an intelligent assistant designed to support Level 1 and Level 2 IT technical support operations. The project is motivated by my own experience working within IT teams, where the accumulation of tickets—most of them repetitive—often overwhelms support staff and the software tools used to manage incidents. In many cases, user reports correspond to minor or basic issues that consume valuable time that technical personnel could otherwise dedicate to more specialized or complex tasks. Within this context, the need for software, an application, or a tool capable of automating part of the IT operation without compromising service quality became evident.

From the initial proposal, I outlined a solution based on artificial intelligence and natural language processing (NLP), capable of interpreting both written and spoken questions. The system relies on an organized, domain-specific knowledge base, combined with semantic processing that enables the delivery of clear and effective answers in a short time. This constitutes applied research, as it integrates problem analysis, functional requirements gathering, system architecture design, and iterative prototyping to adapt Alma's operation to the real needs

of the IT environment.

Before addressing the core development, it is important to explain that the prototype consists of three fundamental components. The first is a backend built with FastAPI, functioning as a language model for semantic processing. The second is a structured knowledge base derived from real technical support experiences. The third—equally essential—is a web-based visual interface that supports interaction through both text and voice commands.

The results obtained with Alma demonstrate its ability to resolve frequent queries, reduce response times, and decrease the operational workload of technical personnel. In conclusion, this project confirms the feasibility of implementing AI-powered assistants to modernize, streamline, and enhance the quality of technical support services. Alma is adaptable, scalable, and aligned with current demands in the IT sector.

# Introducción

La evolución acelerada de las tecnologías digitales ha cambiado por completo la forma en que hoy operan las organizaciones, cómo gestionan su información y cómo prestan servicios tanto a usuarios internos como externos. Este escenario no es simplemente una tendencia tecnológica, sino una transformación profunda que obliga a replantear procesos, roles y modelos operativos. En medio de este panorama, la inteligencia artificial (IA) se ha convertido en un eje estratégico, impulsando la automatización, fortaleciendo la toma de decisiones y ampliando las capacidades técnicas disponibles en cualquier institución que apueste por la modernización. Los avances recientes en modelos de lenguaje, análisis semántico y aprendizaje automático han permitido construir sistemas capaces de interpretar datos complejos, comprender lenguaje natural y responder con precisión en situaciones operativas reales, algo que hace apenas unos años parecía lejano.

En el campo de las tecnologías de la información, la llegada de asistentes inteligentes marca un punto de inflexión. No se trata únicamente de optimizar tareas repetitivas, sino de transformar la manera en que se organiza, centraliza y distribuye el conocimiento técnico dentro de los equipos. Estos asistentes ayudan a reducir cargas operativas, mejoran la experiencia de los usuarios finales y se convierten en aliados clave para avanzar en procesos de transformación digital. Contar con herramientas que aprenden, se adaptan al entorno y responden con un alto nivel de claridad y exactitud es, hoy por hoy, un requisito casi obligatorio para garantizar continuidad del servicio, eficiencia y competitividad.

El proyecto Alma se inserta justamente en este contexto de cambio y responde a un interés académico y profesional por explorar enfoques modernos de desarrollo basados en IA. Desde su concepción, Alma busca demostrar que es posible construir un prototipo funcional de asistente inteligente especializado, capaz de comprender consultas en lenguaje natural, procesar información técnica estructurada y apoyar procesos relacionados con soporte tecnológico. La propuesta reúne conocimientos de procesamiento del lenguaje natural, diseño de arquitecturas backend, recuperación de información y buenas prácticas de ingeniería de software, integrando además una interfaz de interacción que facilita la comunicación entre el usuario y el sistema.

Desarrollar Alma ha sido un ejercicio de aplicación directa, donde teoría, métodos y herramientas convergen en un mismo propósito: construir una solución que responda a necesidades reales del entorno TI. A lo largo del proceso se aplicó una metodología clara que incluyó análisis del problema, diseño, prototipado, implementación y validación. Esto permitió comprobar cómo los modelos de lenguaje y las técnicas de IA pueden integrarse de manera efectiva en soluciones de uso práctico, al tiempo que fortalecen competencias en áreas emergentes como agentes inteligentes, sistemas conversacionales, búsqueda semántica y automatización de procesos.

Este documento recoge todo el proceso de desarrollo del prototipo. Inicia con la contextualización del problema que dio origen a la idea, continúa con la justificación, los objetivos y el marco teórico que sustenta cada componente del proyecto. Más adelante se detalla la metodología empleada y se describen los resultados obtenidos en cada fase, desde la

definición de requerimientos hasta la arquitectura, el prototipo, las pruebas y la implementación final. El cierre incluye conclusiones y recomendaciones orientadas a la evolución futura de Alma y su eventual aplicación en entornos reales del sector tecnológico.

# 1. Planteamiento del problema

## 1.1 Descripción

En los contextos tecnológicos actuales, el apoyo técnico especializado ha llegado a ser un elemento crucial para garantizar la continuidad y estabilidad de los sistemas de información. No obstante, en la labor diaria, los técnicos se enfrentan constantemente a un gran número de tickets que, en una gran medida, se refieren a incidentes recurrentes, preguntas simples o situaciones que ya tienen respuestas documentadas. Varios reportes del sector específicamente  El Zendesk Customer Experience Trends Report (2022) señalan que aproximadamente el 40 % de las peticiones recibidas por los equipos de soporte son repetitivas y monótonas, lo que ocupa tiempo operativo importante y restringe la habilidad del personal para manejar incidentes más complicados o críticos.

La agrupación de esfuerzos en actividades repetitivas ocasiona demoras en los tiempos de respuesta, reduce los niveles de servicio (SLA), eleva el reproceso y crea una impresión negativa entre los usuarios finales. De igual manera, complica la gestión efectiva del conocimiento técnico, ya que el manejo manual de estos casos impide que los equipos organicen, actualicen y utilicen de manera estratégica la información acumulada en su actividad cotidiana.

Aunque hay soluciones comerciales que utilizan chatbots o sistemas automatizados de respuestas, muchas de estas herramientas no tienen una comprensión semántica profunda, se basan en configuraciones rígidas o no se ajustan bien a entornos institucionales que requieren

personalización, autonomía y un manejo seguro del conocimiento técnico. Como resultado, las entidades siguen lidiando con déficits en productividad, seguimiento y reactividad.

Este contexto muestra la urgencia de investigar herramientas fundamentadas en inteligencia artificial, especialmente en modelos de Procesamiento del Lenguaje Natural (PLN), que faciliten la automatización de respuestas a preguntas recurrentes, organicen el conocimiento técnico y optimicen la efectividad del equipo de soporte. Un asistente inteligente capaz de entender lenguaje natural, acceder a bases de datos especializadas y proporcionar soluciones adaptadas podría reducir notablemente la carga operativa y aportar a menos tiempo de los técnicos atendiendo recurrencias.

## 1.2 Formulación

¿De qué manera un asistente inteligente basado en técnicas de procesamiento del lenguaje natural puede contribuir a automatizar la atención de incidentes repetitivos, optimizar los tiempos de respuesta y mejorar la eficiencia del soporte técnico especializado?

## 2. Justificación

Desde hace años venimos apreciando un crecimiento acelerado de las infraestructuras tecnológicas y la digitalización progresiva de los procesos organizacionales los cuales han incrementado de manera significativa la dependencia de servicios de soporte técnico para la solución de incidentes y requerimientos de los usuarios, en casi todas las empresas u organizaciones hay un departamento TI. En este contexto, la cantidad de solicitudes que deben ser gestionadas por los equipos TI ha aumentado en proporciones que exceden, en muchos casos, su capacidad operativa. Este fenómeno ha generado la necesidad de implementar soluciones innovadoras que permitan optimizar tiempos, organizar información técnica y mejorar la eficiencia general de los procesos de atención.

Diversos estudios del sector TI evidencian la magnitud del problema. El Zendesk Customer Experience Trends Report (2022) revela que alrededor del 40 % de los tickets reportados por los usuarios corresponden a consultas repetitivas y de baja complejidad. Este tipo de requerimientos, aunque simples, consume una parte considerable del tiempo operativo del personal técnico, generando retrasos acumulados, afectaciones en los niveles de servicio (SLA) y disminución de la satisfacción del usuario final. Estos hallazgos reflejan la necesidad urgente de incorporar mecanismos automáticos que permitan resolver de forma eficiente este tipo de incidencias recurrentes.

Paralelamente, en los últimos años la inteligencia artificial ha demostrado ser un recurso estratégico para transformar procesos internos mediante la automatización cognitiva. El IBM Global AI Adoption Index (2023) indica que las soluciones basadas en IA reducen entre un 20 %

y 60 % la carga operativa relacionada con tareas repetitivas, especialmente en áreas de soporte y atención al cliente. Este impacto evidencia que las tecnologías basadas en modelos de lenguaje pueden convertirse en herramientas esenciales para mejorar la productividad de los equipos técnicos y optimizar la gestión del conocimiento institucional.

La literatura científica también respalda este enfoque. Russell y Norvig (2021) destacan que los agentes inteligentes y los modelos de PLN tienen la capacidad de interpretar información compleja, reconocer patrones y generar respuestas precisas, lo que los convierte en aliados fundamentales en la resolución automatizada de problemas. Por su parte, Jurafsky y Martin (2023) señalan que los modelos de procesamiento del lenguaje natural han alcanzado niveles avanzados de comprensión semántica, permitiendo desarrollar sistemas conversacionales capaces de manejar terminología técnica y consultas especializadas. Estas bases conceptuales fortalecen el fundamento académico y tecnológico del proyecto.

Desde una perspectiva institucional, es evidente que muchas organizaciones todavía luchan por mantener actualizadas sus bases de conocimiento técnico y por gestionar la información de manera ordenada y accesible. En la práctica, lo que debería ser un repositorio vivo y estructurado termina convirtiéndose en un conjunto disperso de documentos, anotaciones internas y soluciones que dependen más del técnico que las recuerda que de un sistema realmente consolidado. Esta falta de sistematización genera reprocesos, inconsistencias y una dependencia excesiva del conocimiento tácito del personal, lo que dificulta la estandarización de los servicios y afecta los tiempos de respuesta.

Frente a este escenario, un asistente inteligente como Alma aporta una solución concreta y

necesaria. Su capacidad para integrar un motor de búsqueda semántica permite acceder de manera inmediata a información relevante y confiable, evitando repeticiones innecesarias y disminuyendo errores humanos. Al centralizar el conocimiento y ponerlo a disposición del equipo técnico en cuestión de segundos, Alma no solo mejora la eficiencia, sino que contribuye a elevar la calidad del soporte y a fortalecer la continuidad operativa dentro de la organización.

En el ámbito académico, este proyecto representa una oportunidad significativa para aplicar conocimientos adquiridos en diversas áreas del desarrollo de software, como análisis de requerimientos, diseño de arquitecturas backend, modelos de razonamiento, gestión de bases de datos, interacción humano-máquina y metodologías ágiles. La Association for Computing Machinery (ACM) en su guía curricular CS2023 enfatiza la importancia de formar profesionales con competencias en inteligencia artificial, procesamiento del lenguaje natural y sistemas inteligentes, pues constituyen áreas prioritarias para el sector tecnológico global. En este sentido, la creación de Alma se alinea directamente con dichas directrices formativas.

Adicionalmente, el prototipo desarrollado ofrece una alternativa accesible y personalizable frente a soluciones comerciales existentes que, en muchos casos, requieren altos costos de implementación, dependencia de servicios en la nube o infraestructuras avanzadas que no siempre están disponibles en contextos educativos o institucionales locales. Alma se plantea como un sistema modular, adaptable y de ejecución local, lo que permite explorar escenarios reales sin comprometer recursos críticos ni depender de software de terceros. Esto le otorga un valor agregado significativo tanto en términos técnicos como pedagógicos.

Es así como, este proyecto se sustenta en la necesidad de promover soluciones tecnológicas que fortalezcan los procesos de soporte técnico, mejoren la calidad del servicio y apoyen la transformación digital de las organizaciones. El desarrollo de Alma permite demostrar la viabilidad de integrar modelos de lenguaje y técnicas de IA en tareas operativas reales, evidenciando beneficios tangibles en productividad, eficiencia, estandarización del conocimiento y experiencia del usuario. En conjunto, estos argumentos consolidan la pertinencia, relevancia y aporte de la propuesta dentro del campo del desarrollo de software y las tecnologías emergentes.

# 3. Objetivos

## 3.1 Objetivo general

Desarrollar un prototipo funcional de asistente inteligente basado en técnicas de procesamiento del lenguaje natural (PLN) y modelos de lenguaje, capaz de automatizar la atención de incidentes repetitivos, optimizar los tiempos de respuesta y mejorar la eficiencia del soporte técnico especializado mediante la integración de una base de conocimiento estructurada y una interfaz web interactiva.

## 3.2 Objetivos específicos

Analizar el entorno operativo del soporte técnico especializado, identificando los incidentes recurrentes, los patrones de consulta y las brechas en la gestión del conocimiento. Este análisis se realizará en el contexto del área de TI, durante la fase inicial del proyecto, con el fin de determinar qué procesos pueden ser automatizados mediante inteligencia artificial.

Diseñar la arquitectura del sistema Alma, integrando modelos de lenguaje, mecanismos de recuperación semántica, una base de conocimiento estructurada y un backend modular. Este diseño se llevará a cabo durante la etapa de planeación técnica, dentro del entorno de desarrollo del proyecto, para definir cómo funcionará el prototipo y cómo se conectarán sus componentes.

Implementar el prototipo del asistente inteligente Alma, desarrollando el backend en FastAPI, la interfaz web conversacional y los módulos de interacción por texto y voz. La

implementación se realizará en el entorno de desarrollo del proyecto, durante la fase de construcción, con el propósito de materializar la solución propuesta y validar su funcionamiento básico.

Construir, organizar y documentar la base de conocimiento técnica en SQL y el proceso general del proyecto, estructurando para entregar respuestas precisas y contextualizadas, y elaborando la arquitectura, la descripción técnica del prototipo, los manuales de uso y las recomendaciones de mejora. Este objetivo se desarrollará de forma paralela a la construcción del prototipo y se consolidará en la etapa final, tanto en el entorno técnico como académico, garantizando la sostenibilidad y evolución futura del sistema Alma.

# 4. Marco Teórico

## 4.1 Bases conceptuales

### 4.1.1 Inteligencia Artificial (IA)

La inteligencia artificial es un campo de la informática enfocado en el diseño de sistemas capaces de ejecutar tareas que tradicionalmente requieren de la inteligencia humana, tales como el razonamiento, la resolución de problemas, la percepción y el aprendizaje. Russell y Norvig (2021) la definen como el estudio de agentes inteligentes capaces de interactuar con su entorno y tomar decisiones basadas en información disponible. La IA moderna incorpora técnicas como aprendizaje automático, redes neuronales, modelos estadísticos y procesamiento del lenguaje natural, permitiendo el desarrollo de sistemas avanzados que asisten en operaciones críticas, automatizan procesos y optimizan flujos de trabajo.

### 4.1.2 Procesamiento del Lenguaje Natural (PLN)

El procesamiento del lenguaje natural es una rama de la IA orientada a permitir que los sistemas informáticos comprendan, interpreten y generen lenguaje humano. Según Jurafsky y Martin (2023), el PLN combina lingüística computacional con modelos estadísticos y de aprendizaje profundo para analizar el significado, la intención y el contexto de un mensaje. En aplicaciones de soporte técnico, el PLN facilita la interacción entre usuarios y sistemas automatizados, permitiendo que los asistentes conversacionales interpreten preguntas técnicas y generen respuestas precisas basadas en conocimiento especializado.

### 4.1.3 Modelos de lenguaje y arquitecturas modernas

Los modelos de lenguaje han progresado desde métodos basados en reglas hasta estructuras sofisticadas como Transformers, que facilitan la comprensión de relaciones semánticas y sintácticas en grandes cantidades de texto. Estos modelos, basados en aprendizaje profundo, son capaces de generar, resumir, clasificar y contextualizar información. Su habilidad para generar representaciones vectoriales en alta dimensión permite interpretar consultas complejas y realizar búsquedas semánticas en bases de conocimiento.

### 4.1.4 Recuperación de información y búsqueda semántica

La recuperación de información se centra en obtener datos relevantes mediante algoritmos de búsqueda y mecanismos de indexación. En la búsqueda tradicional, los resultados dependen de coincidencias literales entre palabras clave; sin embargo, la búsqueda semántica utiliza técnicas de embeddings y modelos de lenguaje para identificar relaciones conceptuales entre términos. Este enfoque es fundamental para sistemas como Alma, pues permite localizar la solución correcta incluso cuando el usuario formula la pregunta con palabras diferentes a las utilizadas en la documentación técnica.

### 4.1.5 Agentes conversacionales y asistentes inteligentes

Los agentes conversacionales son sistemas capaces de mantener diálogos con usuarios mediante lenguaje natural. Incluyen chatbots, asistentes virtuales y agentes autónomos. Su funcionamiento combina PLN, bases de conocimiento, heurísticas de decisión y módulos de razonamiento. Un asistente inteligente como Alma integra estas capacidades con un contexto especializado: la gestión de incidentes de soporte técnico. Su objetivo es ofrecer respuestas rápidas, coherentes y contextualizadas, optimizando la interacción entre usuarios y sistemas de atención.

### 4.1.6 Bases de conocimiento y gestión del conocimiento técnico

Una base de conocimiento es un repositorio estructurado que almacena información relevante para resolver problemas específicos. Incluye procedimientos, manuales, guías, FAQ, incidentes pasados y soluciones documentadas. La gestión del conocimiento técnico busca organizar, actualizar y distribuir esta información de forma eficiente. En contextos de soporte, una base de conocimiento efectiva reduce reprocesos y mejora la consistencia del servicio. Alma utiliza un enfoque basado en indexación semántica para consultar esta base y entregar respuestas precisas.

### 4.1.7 Arquitecturas backend para sistemas de IA

El backend es la capa lógica que controla las funciones internas de una aplicación. En sistemas impulsados por IA, el backend combina elementos como motores de inferencia, gestores de solicitudes, algoritmos de búsqueda, bases de datos y servicios de integración.

Frameworks como FastAPI posibilitan el desarrollo de APIs rápidas, escalables y seguras, optimizando la comunicación entre el modelo de lenguaje, la base de datos y la interfaz web del usuario. Una estructura modular asegura mantenimiento, escalabilidad y habilidad para ajustarse a nuevos requisitos

### 4.1.8 Interacción humano–máquina (HCI)

La interacción humano–máquina se centra en el diseño de interfaces que faciliten la comunicación entre personas y sistemas computacionales. Su objetivo es optimizar la usabilidad, accesibilidad y experiencia del usuario. En asistentes conversacionales, la interfaz debe permitir interacciones fluidas, visualmente claras y sin fricción. Alma incorpora elementos de HCI mediante un frontend intuitivo, elementos responsivos, interacción por texto/voz y retroalimentación visual que mejora la experiencia del técnico durante la resolución de incidentes.

### 4.1.9 Arquitectura Transformer

Los modelos de lenguaje modernos se fundamentan en la arquitectura Transformer, introducida por Vaswani et al. (2017), la cual reemplazó completamente a las redes recurrentes tradicionales. El Transformer utiliza mecanismos de self-attention que permiten identificar dependencias entre palabras sin importar su posición en la secuencia. Esta arquitectura posibilita que los modelos comprendan mejor el contexto, generen respuestas coherentes y realicen tareas complejas como clasificación, resumen, extracción de información y respuestas a preguntas técnicas.

En el caso de asistentes conversacionales, los Transformers funcionan como el núcleo del razonamiento semántico, procesando consultas de usuarios y generando representaciones vectoriales que luego se comparan con la base de conocimiento. Su capacidad para entender la intención es crítica en sistemas como Alma.

### 4.1.10 Embeddings semánticos

Los embeddings son representaciones numéricas de palabras o frases en un espacio vectorial. Tecnologías como Word2Vec, GloVe, BERT o Sentence-Transformers permiten convertir lenguaje natural en vectores que guardan relaciones semánticas. Cuando dos frases son similares, sus vectores también lo son.

Alma utiliza embeddings para realizar búsquedas semánticas dentro de su base de conocimiento, permitiendo identificar soluciones incluso cuando el usuario formula la pregunta con sinónimos, variaciones lingüísticas o términos propios del dominio técnico.

### 4.1.11 RAG (Retrieval-Augmented Generation)

Retrieval-Augmented Generation (RAG) es una técnica que combina:

1. Recuperación de información (búsqueda vectorial), y
2. Generación de texto mediante un modelo de lenguaje.

Su propósito es mejorar la precisión de las respuestas, evitando alucinaciones y basándose en documentos reales.

Aunque Alma funciona como un prototipo híbrido, su proceso de consulta -buscar semánticamente y luego contextualizar la respuesta- está alineado con el enfoque RAG, lo que fortalece su función como asistente de soporte especializado.

### 4.1.12 Gestión de conocimiento (Knowledge Management - KM)

La gestión del conocimiento se refiere al conjunto de procesos destinados a capturar, almacenar, organizar y distribuir información crítica dentro de una organización. Según Nonaka y Takeuchi (1995), el conocimiento puede ser:

- Tácito: lo que saben los técnicos por experiencia.
- Explícito: lo que está documentado en manuales, procedimientos y bases de datos.

El soporte técnico tradicional depende en gran medida del conocimiento tácito del personal experto. Alma busca transformar parte de ese conocimiento tácito en conocimiento explícito sistematizado, accesible mediante búsqueda semántica, reduciendo la dependencia del criterio individual de cada técnico.

### 4.1.13 Sistemas de recomendación aplicados al soporte técnico

Los sistemas de recomendación, habitualmente utilizados en comercio electrónico y plataformas de contenido, también se aplican a contextos técnicos. Su función es sugerir

soluciones a problemas previamente documentados utilizando algoritmos de filtrado colaborativo o filtrado basado en contenido.

Un asistente inteligente como Alma puede ayudar y recomendar soluciones mediante similitudes semánticas entre consultas, historiales de incidentes y patrones recurrentes, fortaleciendo la eficiencia del soporte TI.

### 4.1.14 Minería de texto y análisis de tickets

La minería de texto consiste en extraer patrones relevantes de textos no estructurados. En el soporte técnico, la minería de texto permite:

- Identificar problemas frecuentes.
- Detectar tendencias o fallas emergentes.
- Analizar el sentimiento del usuario.
- Agrupar incidentes similares mediante clustering.

Aunque Alma no implementa minería avanzada, su base conceptual se alinea con estas técnicas, ya que trabaja con textos no estructurados y busca similitudes entre consultas.

### 4.1.15 Interacción por voz y speech-to-text

La interacción por voz es un componente fundamental de los asistentes modernos. Tecnologías de reconocimiento de voz basadas en modelos acústicos y redes neuronales permiten convertir audio en texto comprensible para el sistema.

En Alma, el módulo de voz en la interfaz web se basa en herramientas del navegador (Web Speech API), permitiendo realizar consultas habladas. Esto fortalece la accesibilidad y mejora la experiencia del usuario, agregando un canal natural de comunicación.

### 4.1.16 Ética en modelos de IA y transparencia en respuestas

En entornos profesionales, la implementación de IA requiere considerar principios éticos y buenas prácticas como:

- Transparencia en las respuestas.
- Reducción de alucinaciones.
- Veracidad basada en fuentes reales.
- Control humano sobre decisiones críticas.
- Protección de datos técnicos sensibles.

El enfoque de Alma -basado en una base de conocimiento interna, local y verificable- reduce riesgos asociados a modelos autónomos en la nube, alineándose con principios de IA responsable promovidos por la IEEE y la Comisión Europea.

### 4.1.17 Limitaciones de los asistentes inteligentes

Para garantizar un marco teórico realista, es necesario reconocer las limitaciones de esta tecnología:

- Dependencia de la calidad de la base de conocimiento.

- Posibles interpretaciones erróneas de consultas ambiguas.
- Falta de contexto completo si no se integra con sistemas productivos.
- Riesgo de respuestas incompletas si el modelo no encuentra coincidencias semánticas.
- Necesidad de actualización continua para evitar obsolescencia.

Estas limitaciones también justifican la importancia del rol humano en la validación final y en la supervisión del asistente.

### 4.1.18 Buenas prácticas en ingeniería de software aplicadas a IA

El desarrollo de Alma incorpora principios fundamentales de ingeniería de software:

- Diseño modular
- Arquitectura limpia (Clean Architecture)
- APIs desacopladas
- Pruebas unitarias y funcionales
- Versionamiento y control de cambios
- Documentación técnica

Estas prácticas garantizan mantenibilidad, escalabilidad y facilidad para futuras mejoras, características clave en prototipos basados en IA.

## 4.2 Antecedentes

### 4.2.1 Soluciones comerciales de automatización del soporte técnico

En el mercado existen diversas herramientas orientadas a automatizar procesos de atención y soporte, como Zendesk AI, Freshdesk Freddy AI y ServiceNow Virtual Agent. Estas soluciones utilizan modelos de lenguaje para interpretar solicitudes y ofrecer respuestas automáticas basadas en conocimiento corporativo. Sin embargo, presentan limitaciones como altos costos, dependencia de infraestructura en la nube, restricciones de personalización y poca adaptabilidad a entornos académicos o de ejecución local. Frente a estas limitaciones, prototipos como Alma ofrecen una alternativa flexible y adecuada para instituciones que requieren control total sobre sus datos y su infraestructura tecnológica.


*Ilustración 1 Screenshot de Zendesk AI México - Nota Tomado del sitio web oficial: https://www.zendesk.com.mx/*

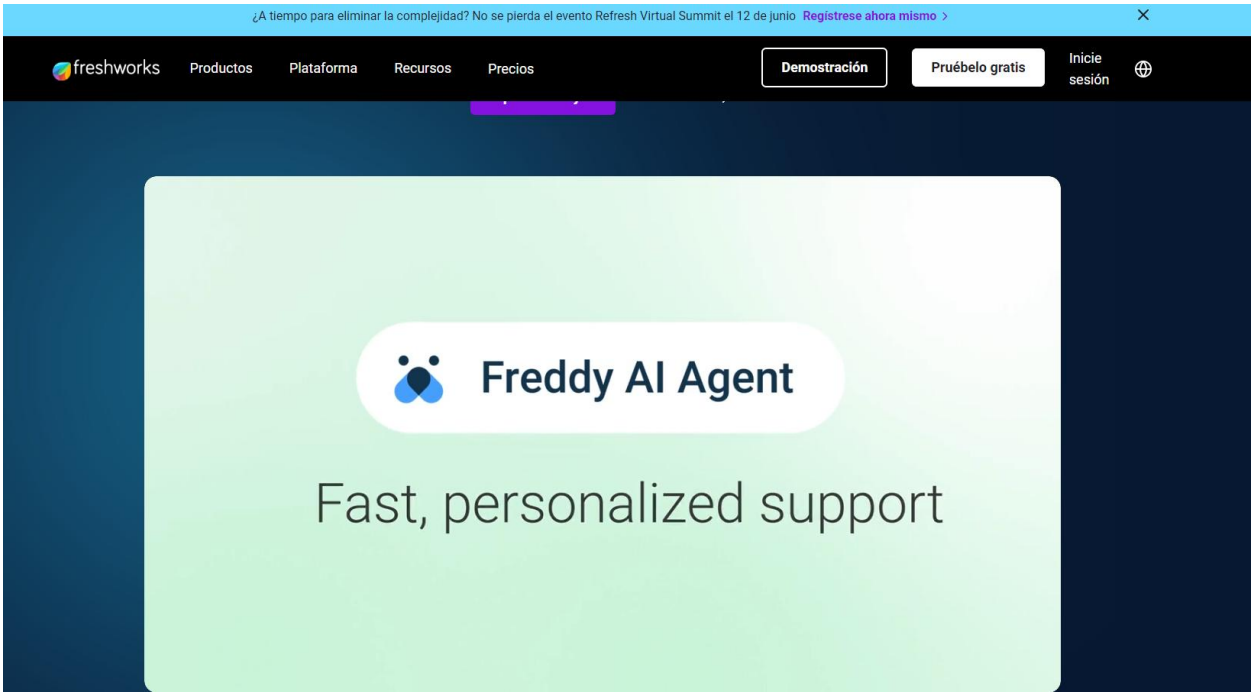



*Ilustración 2Screenshot de Freshwroks Dredy AI - Nota: Tomado del sitio web oficial:*

*https://www.freshworks.com/latam*

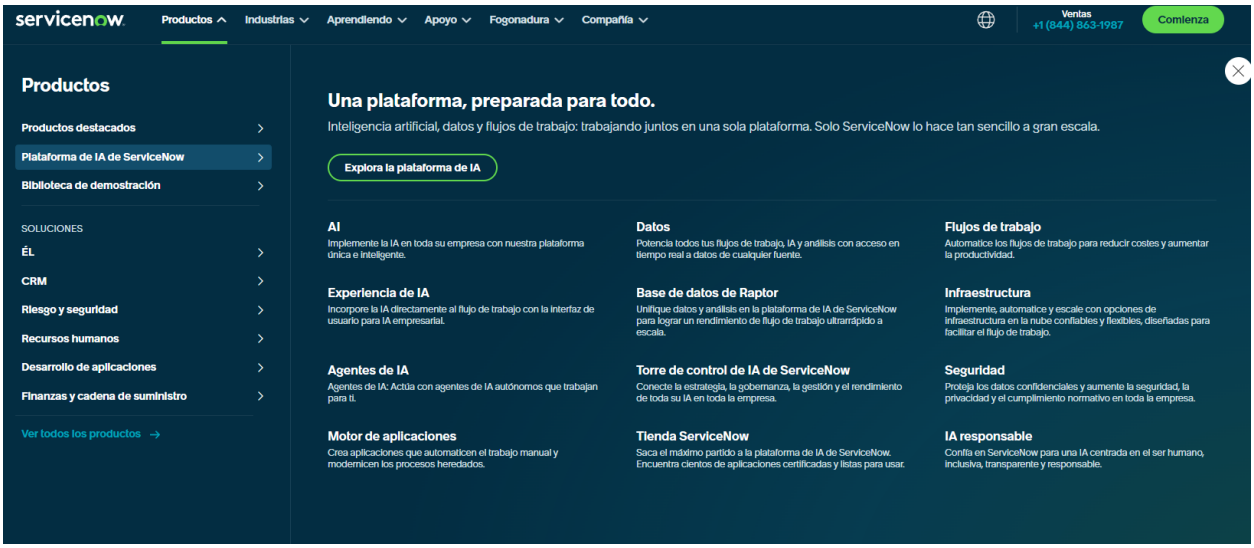



*Ilustración 3 Screenshot ServiceNow AI, Nota: Tomado del sitio web oficial: https://www.servicenow.com/docs/*

### 4.2.2 Proyectos académicos y avances investigativos

En el marco de los avances académicos orientados a la automatización del soporte técnico, resulta pertinente destacar el trabajo desarrollado por Beltrán Puentes (2023), quien diseñó e implementó un chatbot con inteligencia artificial para optimizar la atención en mesas de servicio. Su estudio evidencia que muchas organizaciones continúan enfrentando sobrecarga operativa debido a la cantidad de incidentes repetitivos y consultas básicas que saturan los canales de soporte. En respuesta a esta problemática, el autor propone una solución basada en IA capaz de automatizar respuestas frecuentes, reducir tiempos de espera y mejorar la eficiencia operativa, logrando automatizar cerca del 80 % de las consultas y disminuir en un 60 % los tiempos de atención (Beltrán Puentes, 2023).

Este antecedente académico resulta especialmente relevante para el desarrollo del proyecto Alma, pues demuestra la viabilidad y pertinencia de integrar modelos de lenguaje y mecanismos de búsqueda semántica en entornos reales de soporte técnico. Además, refuerza la necesidad de contar con herramientas que sistematizan el conocimiento, mitiguen la dependencia del saber tácito del personal y contribuyan a la continuidad del servicio, tal como lo plantea el propio Beltrán Puentes (2023). En este sentido, Alma se enmarca dentro de una línea investigativa ya explorada, pero propone un enfoque más especializado y adaptable, orientado específicamente a escenarios de soporte técnico de nivel 1 y 2.

### 4.2.3 Necesidad de automatización y eficiencia operativa

La literatura reciente resalta la importancia de incorporar herramientas inteligentes para reducir tiempos de atención y mejorar la experiencia del usuario. Diversos análisis de la industria -incluidos informes especializados de Gartner sobre automatización TI- señalan que la

automatización cognitiva será un eje fundamental en la modernización del área tecnológica durante esta década. Estas tendencias avalan la pertinencia de desarrollar asistentes inteligentes como Alma, capaces de disminuir la carga operativa, sistematizar el conocimiento y ofrecer soporte continuo y estandarizado.

La creciente demanda de atención en las mesas de ayuda y áreas de soporte técnico ha generado un aumento sostenido en el volumen de tickets, muchos de ellos repetitivos y de baja complejidad, lo que provoca saturación de los agentes y tiempos de respuesta elevados. Trabajos que implementan chatbots de soporte técnico muestran que los modelos tradicionales basados únicamente en atención humana se vuelven ineficientes frente a este volumen, ya que gran parte del tiempo de los analistas se consume en tareas rutinarias que podrían ser automatizadas. En este escenario, la automatización de procesos mediante inteligencia artificial se configura como una necesidad para sostener la calidad de servicio y garantizar tiempos de respuesta aceptables.

Diversas investigaciones y artículos técnicos sobre mesas de ayuda con IA reportan que la automatización de tareas como clasificación, priorización y atención de consultas frecuentes permite reducir el tiempo promedio de resolución, aumentar la capacidad de atención simultánea y mejorar indicadores de satisfacción del usuario. Por ejemplo, Beltrán Puentes (2025) desarrolla un sistema de chatbot con IA para soporte técnico en un entorno corporativo y evidencia mejoras en la eficiencia operativa al descargar al personal humano de solicitudes recurrentes. De forma similar, el artículo “Automatización del soporte al cliente mediante un chatbot con IA” en la Revista GADE concluye que la incorporación de chatbots en el flujo de gestión de tickets contribuye a optimizar recursos, reducir costos y fortalecer la competitividad de las organizaciones al ofrecer atención continua y más ágil.

# 5. Metodología

## 5.1 Tipo de investigación

El proyecto se basa en una investigación aplicada, dado que su propósito es desarrollar un prototipo funcional que permita resolver una problemática real asociada a la automatización de incidentes repetitivos en el soporte técnico especializado. La investigación aplicada, según Hernández, Fernández y Baptista (2018), se orienta a generar soluciones prácticas basadas en conocimientos previamente adquiridos, integrando elementos teóricos y tecnológicos para intervenir un contexto específico. Así las cosas, la construcción de un asistente inteligente constituye una aplicación directa de conceptos de inteligencia artificial, procesamiento del lenguaje natural y desarrollo de software.

Asimismo, el proyecto incorpora un componente de investigación tecnológica, pues se orienta a la creación y validación de un producto digital plenamente funcional, evaluando su desempeño y su capacidad para mejorar los procesos actuales de soporte. Esta propuesta integra un enfoque experimental, ya que el prototipo es sometido a pruebas operativas que permiten medir su efectividad en escenarios reales de consulta técnica y determinar su aporte concreto en la optimización del servicio.

## 5.2 Enfoque metodológico

El enfoque adoptado es mixto, integrado por:

- Métodos cualitativos, empleados para analizar el entorno operativo del soporte técnico, interpretar patrones de incidentes recurrentes, comprender necesidades del usuario final y recolectar información a través de observación y análisis documental.

- Métodos cuantitativos, utilizados para medir métricas como tiempos de respuesta, recurrencia de errores, precisión de las respuestas entregadas por el prototipo y nivel de eficiencia alcanzado en la resolución de incidentes.

El avance técnico se fundamenta en un método incremental e iterativo, característico de las metodologías ágiles. Este enfoque facilita el progreso a través de ciclos de desarrollo, evaluación y retroalimentación, asegurando un producto final más sólido, estable y acorde a las necesidades identificadas

### 5.3 Técnicas e instrumentos de recolección de información

Para la construcción del prototipo y la validación del problema, se emplearon las siguientes técnicas:

Análisis documental, mediante la revisión de manuales, incidentes históricos de experiencia propia y bases de conocimiento sobre soporte técnico y automatización.

Observación directa, identificando cómo interactúan los técnicos de TI con herramientas actuales de soporte y cuáles son los incidentes que más tiempo consumen.

Entrevistas no estructuradas, dirigidas a técnicos y usuarios internos para comprender la naturaleza de sus consultas y los flujos de atención.

El análisis se realizó a partir de registros internos de soporte técnico correspondientes a tikets reales gestionados en la mesa de ayuda del Servicio Nacional de Aprendizaje –SENA–, información obtenida por el autor en el marco de sus funciones como contratista TI de la entidad (SENA, registros internos de mesa de ayuda, 2024).

Los instrumentos empleados incluyen:

- Formatos internos de registro de incidentes.
- Historiales de soporte técnico generados por la mesa de ayuda.
- Bases de conocimiento institucionales utilizadas por el equipo TI.
- Registros preliminares provenientes del software de prueba del prototipo.

**5.4 Población y muestra**

La población objetivo está compuesta por técnicos de soporte especializado y usuarios finales de sistemas informáticos en entornos empresariales y académicos. Para efectos de validación del prototipo, se utilizó una muestra no probabilística por conveniencia, seleccionando técnicos con experiencia en resolución de incidentes frecuentes y usuarios que reportan solicitudes de primer nivel.

Esta muestra permitió obtener información suficiente para validar la pertinencia funcional del prototipo, evaluar su desempeño y analizar su impacto en los tiempos de respuesta.

## 5.5 Procedimiento metodológico

El proceso de desarrollo de Alma se estructuró bajo un enfoque secuencial con iteraciones continuas, distribuido en las siguientes fases:

Fase 1. Análisis del problema

- Identificación de incidentes recurrentes.
- Revisión de procesos actuales de soporte técnico.
- Análisis de brechas en la gestión del conocimiento.

Fase 2. Levantamiento de requerimientos

- Definición de requerimientos funcionales y no funcionales.
- Establecimiento de casos de uso.
- Especificación de criterios de aceptación.

Fase 3. Diseño de la arquitectura

- Construcción del modelo de datos y la base de conocimiento.
- Diseño de la arquitectura backend en FastAPI.
- Selección del modelo de lenguaje y técnica de búsqueda semántica.
- Diseño preliminar de la interfaz web.

Fase 4. Desarrollo del prototipo

- Implementación del backend y endpoints REST.
- Integración del motor de embeddings y FAISS.
- Construcción de la interfaz web conversacional.
- Implementación del módulo de voz (speech-to-text).

Fase 5. Pruebas y validación

- Pruebas funcionales del modelo.
- Evaluación del rendimiento y tiempos de respuesta.
- Validación con técnicos y usuarios seleccionados.
- Correcciones iterativas basadas en retroalimentación.

Fase 6. Documentación y cierre

- Manual técnico y de usuario.
- Elaboración del informe final.
- Registro de resultados y recomendaciones.

## 5.6 Cronograma de actividades

El proyecto se desarrolló durante 16 semanas, entre el 4 de agosto y el 19 de noviembre de 2025, conforme a la planificación presentada en el anteproyecto. Cada actividad contempla su respectiva semana de ejecución, el responsable de su desarrollo y el estado actual de avance.

| Semanas | Actividad | Responsable | Estado |
|---|---|---|---|
| **1–2** | Definición del problema, formulación de objetivos, delimitación del alcance y levantamiento preliminar de requerimientos técnicos. | Yeison Dariel Zapata Monsalve | Completado |
| **3–4** | Revisión bibliográfica y análisis documental de fuentes primarias y secundarias relacionadas con IA, PLN, soporte técnico y automatización. | Yeison Dariel Zapata Monsalve | Completado |
| **5–6** | Diseño conceptual del asistente: casos de uso, arquitectura general del sistema, flujos de interacción | Yeison Dariel Zapata Monsalve | Completado |

| | | | |
|---|---|---|---|
| | y estructura preliminar de la base de conocimiento. | | |
| **7–8** | Construcción del primer prototipo funcional mediante la integración del modelo NLP, configuración del backend en FastAPI y preparación del motor semántico. | Yeison Dariel Zapata Monsalve | Completado |
| **9–10** | Desarrollo de la interfaz web conversacional e integración con la base de conocimiento estructurada, incorporando interacción por voz y texto. | Yeison Dariel Zapata Monsalve | Completado |
| **11–12** | Pruebas controladas con usuarios técnicos, análisis de desempeño, corrección de errores y ajustes derivados de retroalimentación. | Yeison Dariel Zapata Monsalve | Completado |
| **13–14** | Optimización técnica, mejoras de rendimiento, | Yeison Dariel Zapata Monsalve | Completado |

| | | | |
|---|---|---|---|
| | documentación del código fuente y validación final del prototipo. | | |
| **15–16** | Elaboración del informe final, anexos, revisión general y entrega del documento definitivo. | Yeison Dariel Zapata Monsalve | Completado |

*Tabla 1 Cronograma final de actividades – Fuente: Elaboración propia*

Este cronograma muestra un desarrollo progresivo y organizado del prototipo Alma, garantizando que cada fase metodológica se lleve a cabo con bases técnicas y académicas, alineadas con los plazos, tareas y etapas establecidas en el anteproyecto.

# 6. Resultados

Este capítulo expone los resultados alcanzados a lo largo del proceso de diseño, construcción, puesta en marcha y verificación del prototipo Alma, un asistente inteligente orientado al soporte técnico especializado. Cada actividad desarrollada se llevó a cabo siguiendo la metodología definida y el cronograma establecido en el anteproyecto, lo que permitió asegurar una evolución ordenada, progresiva y evaluable del sistema.

## 6.1 Descripción general del prototipo

Alma es un asistente inteligente construido sobre una arquitectura modular que combina diversas tecnologías especializadas. Integra técnicas de procesamiento del lenguaje natural, mecanismos de búsqueda semántica basados en embeddings, un backend desarrollado en FastAPI y una base de conocimiento organizada en SQLite. A ello se suma un índice vectorial gestionado con FAISS y una interfaz web interactiva que permite al usuario formular consultas tanto por texto como por comandos de voz.

El sistema permite:

- Interpretar preguntas técnicas realizadas por los usuarios.
- Consultar información relevante dentro de una base de conocimiento.
- Recuperar soluciones mediante búsqueda semántica.
- Explicar respuestas de forma clara y contextualizada.
- Facilitar interacción natural por texto y comandos de voz.

- Operar de manera local mediante ejecución modular (scripts .bat).

El prototipo busca automatizar incidentes repetitivos y servir como un primer nivel de soporte técnico, reduciendo carga operativa y mejorando los tiempos de respuesta.

## 6.2 Levantamiento de requerimientos

Los requerimientos fueron obtenidos mediante análisis documental, revisión de tickets, entrevistas no estructuradas y observación, en coherencia con los objetivos específicos planteados.

### 6.2.1 Requerimientos de usuario

Los usuarios principales son:

- Técnicos de soporte nivel 1 y 2.
- Personal encargado de la atención inicial de incidentes.
- Usuarios internos que formulan consultas frecuentes.

Necesidades identificadas:

- Obtener respuestas rápidas a preguntas técnicas repetitivas.
- Encontrar información relevante sin navegar múltiples documentos.
- Consultar soluciones históricas de incidentes similares.
- Interactuar de forma sencilla mediante texto y voz.

- Acceder a un asistente disponible en tiempo real.
- Reducir dependencia del criterio individual del técnico.
- Disminuir los tiempos de atención y reprocesos.

Requerimientos de usuario:

- El asistente debe permitir ingresar consultas en lenguaje natural.
- El sistema debe entregar respuestas claras y comprensibles.
- El asistente debe ser accesible desde una interfaz amigable.
- El usuario debe poder interactuar mediante texto y voz.
- El sistema debe mostrar respuestas basadas en conocimiento real, no inventado.
- El asistente debe mantener coherencia técnica en sus respuestas.

### 6.2.2 Requerimientos funcionales

RF1. El sistema debe procesar consultas textuales ingresadas por el usuario.

RF2. El asistente debe convertir voz a texto mediante reconocimiento de voz.

RF3. El backend debe recibir la consulta y generar embeddings semánticos.

RF4. El sistema debe realizar una búsqueda vectorial en el índice FAISS.

RF5. El asistente debe recuperar documentos relevantes desde la base de conocimiento.

RF6. El modelo de lenguaje debe generar una respuesta basada en los documentos recuperados.

RF7. El sistema debe devolver la respuesta al frontend de manera estructurada.

RF8. La interfaz debe mostrar la respuesta de forma clara y ordenada.

RF9. El sistema debe registrar logs de interacción para auditoría y mejora continua.

RF10. El asistente debe ejecutarse localmente sin requerir conexión a servicios externos.

### 6.2.3 Requerimientos no funcionales

RNF1. Rendimiento: la respuesta debe ser entregada en menos de 3 segundos en consultas locales.

RNF2. Disponibilidad: el prototipo debe estar disponible al menos el 95% del tiempo en pruebas.

RNF3. Usabilidad: la interfaz debe ser intuitiva, responsiva y de fácil navegación.

RNF4. Escalabilidad: la arquitectura debe permitir añadir nuevas categorías y documentos.

RNF5. Seguridad: el sistema debe ejecutarse de forma local, garantizando confidencialidad de datos.

RNF6. Mantenibilidad: el código debe seguir principios modulares y buena documentación.

RNF7. Portabilidad: la aplicación debe ejecutarse mediante scripts .bat en cualquier equipo compatible.

RNF8. Precisión: las respuestas deben corresponder a la base de conocimiento y ser verificables.

RNF9. Robustez: el sistema debe manejar consultas ambiguas sin fallar.

### 6.2.4 Casos de uso

A continuación, se presentan los casos de uso principales del sistema. Al final se acompaña con diagramas UML.

| ID | Nombre del Caso de Uso | Actor(es) | Descripción | Precondición | Postcondición |
|---|---|---|---|---|---|
| **CU 01** | Realizar consulta por texto | Usuario técnico | El usuario ingresa una consulta en lenguaje natural y el sistema la procesa para generar una respuesta. | Backend operativo y UI cargada. | Respuesta técnica generada y mostrada. |
| **CU 02** | Realizar consulta por voz | Usuario técnico | El usuario dicta su consulta mediante reconocimiento de voz; el sistema convierte el audio a texto y procesa la búsqueda. | Navegador compatible con Web Speech API. | Consulta convertida y respuesta entregada. |
| **CU 03** | Recuperar información técnica | Sistema | El backend ejecuta una búsqueda semántica en la base vectorial | Base de conocimiento indexada en FAISS. | Documentos relevantes recuperados. |

| | | | | | |
|---|---|---|---|---|---|
| | | | para obtener documentos relevantes. | | |
| **CU 04** | Generar respuesta final | Modelo de lenguaje | El modelo analiza la consulta y el contexto recuperado para producir una respuesta coherente. | Embeddings generados y contexto disponible. | Respuesta final estructurada. |
| **CU 05** | Interactuar con la interfaz | Usuario técnico | El usuario navega por la interfaz, revisa respuestas, limpia el chat y acciona controles. | UI disponible y funcional. | Interacción registrada en la sesión. |
| **CU 06** | Gestionar la base de conocimiento | Administrador | Permite agregar, editar o eliminar entradas dentro de la base de conocimiento. | Acceso autorizado. | Base de conocimiento actualizada. |
| **CU 07** | Reindexar documentos | Administrador | El sistema genera nuevamente los embeddings y actualiza el índice FAISS | Modificaciones previas en KB. | Indice vectorial sincronizado. |

| | | | | | |
|---|---|---|---|---|---|
| | | | tras cambios en la KB. | | |
| **CU 08** | Limpiar conversación | Usuario técnico | El usuario elimina el historial del chat para iniciar una nueva interacción. | Historial existente. | Conversación reiniciada. |
| **CU 09** | Exportar conversación | Usuario técnico | Permite descargar el historial de preguntas y respuestas en un archivo de texto. | Existencia de historial. | Archivo exportado correctamente. |
| **CU 10** | Activar/desactivar modo voz | Usuario técnico | El usuario habilita o deshabilita el reconocimiento de voz según su necesidad. | Navegador compatible. | Modo voz activado o desactivado. |
| **CU 11** | Validar estado del backend | Sistema / Usuario | La UI verifica si el backend está operativo y muestra un indicador visual. | Conexión entre UI y API. | Estado ON/OFF mostrado al usuario. |
| **CU 12** | Retroalimentar la respuesta (feedback) | Usuario técnico | El usuario califica si la respuesta fue | Respuesta generada. | Feedback almacenado |

| | | | | | |
|---|---|---|---|---|---|
| | | | útil o no mediante botones de feedback. | | para futuras mejoras. |
| **CU 13** | Ejecutar scripts del sistema | Administrador | Permite iniciar los componentes del sistema mediante scripts `.bat`. | Scripts configurados en la carpeta del proyecto. | Componentes activos (LLM, backend, UI). |

*Tabla 2 Casos de usos . Diseño: Idea propia*

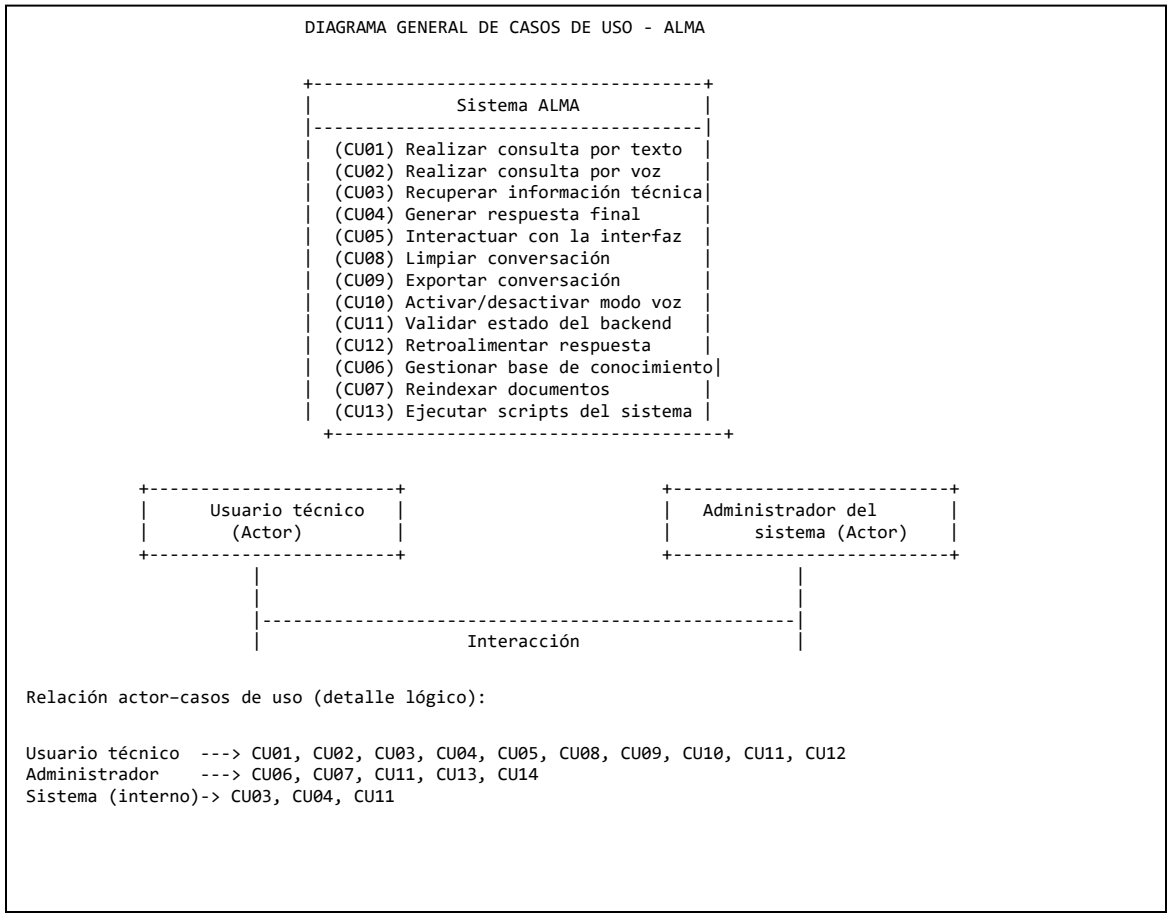
### 6.2.4 Diagramas de casos de uso

En esta sección se presenta el diagrama general de casos de uso del sistema Alma, donde se visualizan las interacciones principales entre los actores y el sistema. Los actores identificados son:

Usuario técnico, quien realiza consultas, interactúa con la interfaz y evalúa la utilidad de las respuestas.

Administrador del sistema, responsable de la gestión de la base de conocimiento, la reindexación de documentos y la ejecución de componentes internos del prototipo.

El sistema Alma se representa como un único bloque lógico que agrupa los diferentes casos de uso descritos en la tabla correspondiente, permitiendo observar de manera global el alcance funcional del prototipo.




## 6.3 Fase de diseño conceptual

La fase de diseño conceptual tuvo como propósito definir la estructura lógica de los datos y la organización de los componentes de software que integran el prototipo Alma. En esta etapa se modelaron las entidades principales del sistema, sus relaciones y las clases que conforman la arquitectura interna, garantizando coherencia con los requerimientos funcionales y no funcionales definidos previamente.

### 6.3.1 Diseño de la estructura de datos de Alma

El prototipo actual no incorpora una base de datos relacional. En su lugar, la información se almacena y gestiona a través de dos archivos clave:

faqs.csv, que contiene las preguntas y respuestas de soporte técnico que sirven de base de conocimiento.

index.faiss e index_map.json, que conforman el índice vectorial utilizado para la búsqueda semántica; este índice se genera a partir de los artículos de la base de conocimiento.

Además, el prototipo genera archivos de log que registran las consultas realizadas y las respuestas entregadas. Aunque conceptualmente es posible identificar entidades (como “consulta”, “respuesta”, “artículo de conocimiento” o “feedback”), estas no se persisten en una base de datos, sino que se manejan en memoria y, de ser necesario, se guardan como ficheros planos (por ejemplo, logs de interacción).

A continuación se describe la estructura de datos y su función dentro del prototipo:

Artículo de conocimiento (faqs.csv): cada fila del CSV representa una pregunta y su respuesta asociada, equivalente a un artículo de la base de conocimiento.

Embeddings (index.faiss / index_map.json): para cada artículo se genera un vector de características que se almacena en el índice FAISS; el fichero index_map.json mantiene la referencia entre cada vector y la fila correspondiente del CSV.

Consulta: la consulta del usuario se procesa en memoria; no se almacena en una tabla, sino que se registra opcionalmente en archivos de log para fines de depuración.

Respuesta: la respuesta generada (ya provenga del modelo de lenguaje o de la base de conocimiento) se devuelve al usuario y también puede registrarse en el log.

Feedback: el usuario puede indicar si la respuesta fue útil o no; este dato se utiliza para iterar y mejorar el sistema, pero no se persiste en una base de datos relacional en el prototipo actual.

Con esta estructura se logra:

- Consultar la base de conocimiento y generar respuestas sin necesidad de un motor de base de datos.

- Mantener un índice vectorial para la búsqueda semántica de manera eficiente.

- Registrar interacciones mediante archivos de log, lo cual permite análisis posteriores sin aumentar la complejidad del prototipo.

Para versiones futuras puede considerarse la implementación de un modelo entidad–relación completo (como el descrito originalmente) si se requiere persistir usuarios, consultas y retroalimentación de forma estructurada. Las ideas sobre mejoras se trasladarán a un apartado de trabajo futuro. (ver numeral 8.1)

### 6.3.2 Diagrama de clases modelo UML

El diagrama de clases UML representa los componentes de software que conforman el prototipo Alma, sus responsabilidades y relaciones. A partir de la arquitectura definida, se identificaron las siguientes clases principales:

- `AlmaAssistant`
- `KnowledgeBase`
- `EmbeddingService`
- `SearchService`
- `LLMClient`
- `ConversationManager`
- `VoiceService`
- `UserInterface (o ChatController)`
- `LoggerService`
- `Clases de dominio: Usuario, Consulta, Respuesta, ArticuloConocimiento, Feedback.`

Descripción de clases principales:

Clase AlmaAssistant

Actúa como fachada principal del sistema. Orquesta la interacción entre el backend, el modelo de lenguaje, la base de conocimiento y la interfaz.

Métodos clave:

- `procesarConsulta(texto, usuario)`
- `generarRespuesta(contexto)`
- `registrarInteraccion()`

Clase `KnowledgeBase`

Gestiona el acceso a la base de conocimiento técnica.

Métodos:

- `obtenerArticuloPorId(id)`
- `buscarArticulosPorCategoria(cat)`
- `listarArticulos()`
- `actualizarArticulo(articulo)`

Clase `EmbeddingService`
Encargada de generar embeddings para textos y artículos.

Métodos:

- `generarEmbedding(texto)`
- `actualizarEmbedding(id_articulo)`

Clase `SearchService`
Administra las búsquedas semánticas utilizando el índice FAISS.

Métodos:

- `buscarPorEmbedding(embedding)`
- `obtenerDocumentosRelevantes(embedding, k)`

Clase `LLMClient`
Abstrae la interacción con el modelo de lenguaje (por ejemplo, Ollama / LLaMA).

Métodos:

- `completar(contexto, pregunta)`
- `resumir(documento)`

Clase `ConversationManager`
Controla el contexto de la conversación (historial de preguntas y respuestas).

Métodos:

- `agregarTurno(usuario, mensaje)`
- `obtenerHistorial(usuario)`
- `limpiarConversacion(usuario)`

Clase `VoiceService`
Gestiona la conversión de voz a texto en la interfaz.

Métodos:

- `iniciarEscucha()`
- `detenerEscucha()`
- `procesarAudio()`

Clase `UserInterface / ChatController`

Gestiona la capa de presentación.

Métodos:

- `enviarConsulta()`
- `mostrarRespuesta()`
- `mostrarEstadoBackend()`
- `exportarConversacion()`

Clases de dominio (Usuario, Consulta, Respuesta, ArticuloConocimiento, Feedback)

Representan los objetos de negocio que viajan entre capas y se corresponden con las entidades del MER.

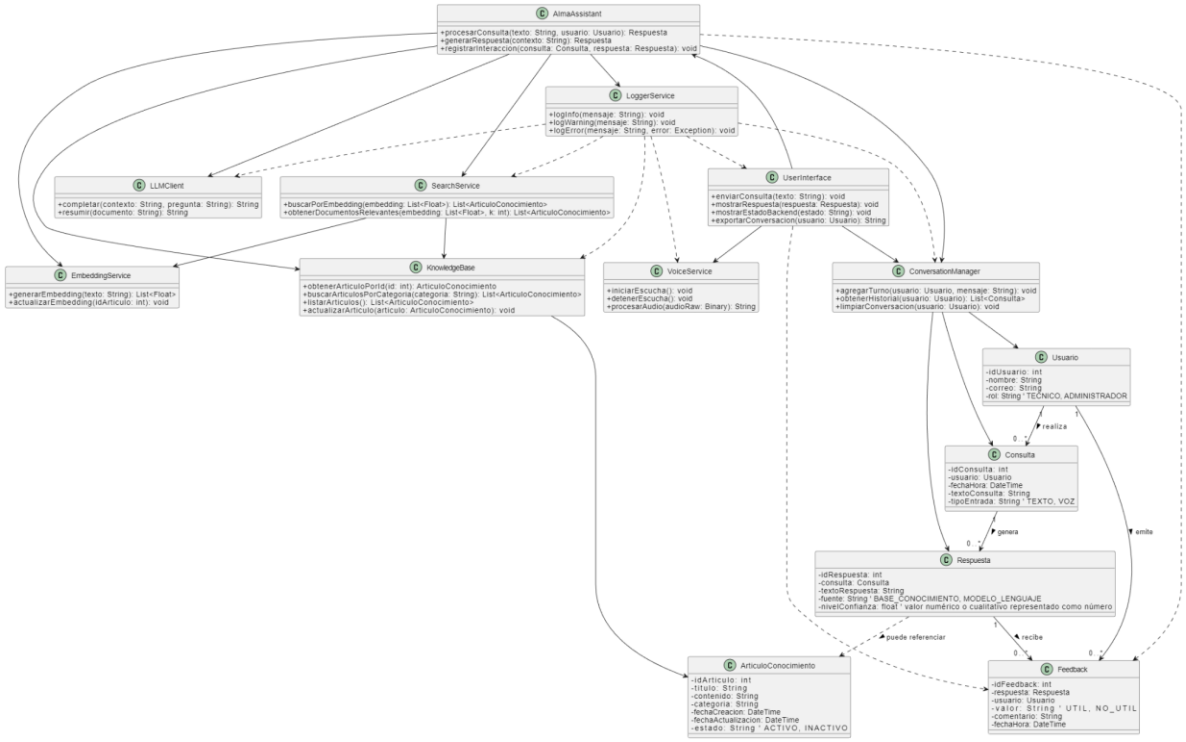



*Ilustración 5 Diagrama de clases UML -Diseño: Propio*

Este diseño orientado a objetos permite:

- Mantener clara separación de responsabilidades.
- Facilitar mantenimiento y ampliación futura.
- Integrar de manera ordenada la IA, la base de conocimiento y la interfaz.
- Asegurar alineación entre el modelo de datos (MER) y el modelo de clases (UML).

## 6.4 Fase de desarrollo ágil por iteraciones (sprints)

La creación del prototipo Alma se realizó con un enfoque ágil apoyado en iteraciones

breves (sprints), lo que facilitó construir el sistema de manera incremental, comprobar avances mediante retroalimentación continua y garantizar que cada componente aporte valor funcional al objetivo global del proyecto. Cada sprint duró alrededor de dos semanas, conforme al cronograma aprobado en el anteproyecto.

El uso de metodologías ágiles facilitó:

- Adaptación rápida a cambios.
- Priorización continua del backlog.
- Integración temprana del modelo NLP.
- Entregables funcionales en cada iteración.

La estructura general del desarrollo comprendió cuatro sprints principales, más una fase final de estabilización y documentación.

### 6.4.1 Desarrollo iterativo (Sprints)

A continuación, se detalla cada sprint, su objetivo, actividades ejecutadas y entregables obtenidos.

| **Sprint** | **Duración (Semanas)** | **Objetivo** | **Actividades Ejecutadas** | **Entregables** |
|---|---|---|---|---|
| **Sprint 1** | 1–2 | Levantamiento del problema, requerimientos y alcance del prototipo | • Análisis de tickets repetitivos<br>• Identificación de incidentes automatizables | • Documento de requerimientos<br>• Casos de uso iniciales<br>• Backlog inicial |

| | | | | |
|---|---|---|---|---|
| | | | • Levantamiento de requerimientos de usuario<br>• Construcción del backlog inicial<br>• Revisión de tecnologías (FastAPI, FAISS, SQLite, LLaMA)<br>• Definición del alcance | • Arquitectura preliminar<br>• Estructura inicial de la base de conocimiento |
| **Sprint 2** | 3–4 | Diseño conceptual y arquitectura técnica | • Diseño del MER<br>• Modelado UML (clases)<br>• Diseño del pipeline de consulta (NLP + FAISS)<br>• Estructura de carpetas del proyecto<br>• Wireframe de la interfaz<br>• Definición de endpoints del backend | • MER final<br>• Diagrama de clases UML<br>• Arquitectura modular validada<br>• Esquema de API<br>• Diseño preliminar de la interfaz |
| **Sprint 3** | 5–6 | Implementación del backend y motor semántico | • Desarrollo del backend en FastAPI<br>• Integración con SQLite<br>• Servicio de embeddings<br>• Indexación FAISS<br>• Implementación de módulo AlmaAssistant<br>• Manejo de errores y logs<br>• Configuración de scripts `.bat` | • Backend operativo<br>• API funcional<br>• Embeddings generados<br>• Índice FAISS creado<br>• Scripts de ejecución (backend y LLM) |

| | | | | |
|---|---|---|---|---|
| **Sprint 4** | 7–8 | Desarrollo del frontend e integración full stack | • Desarrollo de la interfaz web conversacional<br>• Integración de reconocimiento de voz<br>• Indicador de estado del backend<br>• Exportación de conversaciones<br>• Conexión completa UI ↔ API<br>• Ajustes visuales y responsivos | • Interfaz web funcional<br>• Módulo de voz operativo<br>• Estado del backend visible<br>• Chat integrado con backend<br>• Prototipo navegable |
| **Sprint 5** | 9–10 | Pruebas, correcciones y optimización del prototipo | • Pruebas funcionales por escenarios<br>• Pruebas de rendimiento<br>• Pruebas con técnicos<br>• Correcciones en módulo de voz<br>• Optimización de embeddings y consultas<br>• Depuración de interfaz móvil | • Informe final<br>• Prototipo estable<br>• Versión optimizada<br>• Ajustes finales validados |

*Tabla 3 Tabla de Sprints – Desarrollo Ágil del Proyecto ALMA - Diseño: Propio*

### 6.4.2 Backlog

El backlog del proyecto fue construido con enfoque ágil, priorizando historias de usuario esenciales para cumplir el objetivo general de automatizar consultas técnicas repetitivas.

A continuación, se presenta el backlog estructurado por épicas, historias de usuario y criterios de aceptación.

ÉPICA 1 – Procesamiento de consultas

| ID | Historia de Usuario | Prioridad | Criterios de Aceptación |
|---|---|---|---|
| HU01 | Como técnico, quiero realizar consultas por texto para obtener respuestas rápidas | Alta | El sistema recibe texto, procesa embeddings, recupera KB y entrega respuesta. |
| HU02 | Como técnico, quiero realizar consultas por voz para no tener que escribir | Media | El sistema convierte voz a texto y genera respuesta sin errores. |

*Tabla 4 ÉPICA 1 – Procesamiento de consultas*

ÉPICA 2 – Motor semántico y base de conocimiento

| ID | Historia de Usuario | Prioridad | Criterios de Aceptación |
|---|---|---|---|
| HU03 | Como administrador, quiero agregar y actualizar artículos de conocimiento | Alta | Las entradas se almacenan, indexan y son recuperables semánticamente. |

| | | | |
|---|---|---|---|
| HU04 | Como sistema, necesito generar embeddings automáticamente | Alta | Se crean vectores y se sincroniza el índice FAISS. |

*Tabla 5 ÉPICA 2 – Motor semántico y base de conocimiento*

ÉPICA 3 – Interfaz de usuario

| ID | Historia de Usuario | Prioridad | Criterios de Aceptación |
|---|---|---|---|
| HU05 | Como técnico, quiero ver el estado del backend | Alta | La interfaz muestra ON/OFF correctamente. |
| HU06 | Como técnico, quiero exportar la conversación | Media | Se genera un archivo descargable en TXT. |

*Tabla 6 ÉPICA 3 – Interfaz de usuario*

ÉPICA 4 – Gestión y control

| ID | Historia de Usuario | Prioridad | Criterios de Aceptación |
|---|---|---|---|
| HU07 | Como técnico, quiero limpiar la conversación | Media | El chat se reinicia sin perder integridad del sistema. |
| HU08 | Como usuario, quiero dejar feedback de la respuesta | Baja | Se registra “si” o “no” correctamente. |

*Tabla 7 ÉPICA 4 – Gestión y control*

## 6.5 Fase de pruebas continuas y evaluación de resultados

La fase de pruebas tuvo como propósito validar el funcionamiento del prototipo Alma,

medir su desempeño técnico y funcional, corregir errores detectados durante las iteraciones de desarrollo y evaluar su utilidad sobre escenarios reales de soporte técnico. Las pruebas se realizaron de manera continua a lo largo de los sprints y se consolidaron en esta fase final, permitiendo obtener un prototipo estable, funcional y alineado con los requerimientos definidos.

Se aplicaron diferentes tipos de pruebas:

- Pruebas funcionales
- Pruebas de integración
- Pruebas de rendimiento
- Pruebas de usabilidad
- Pruebas de experiencia de usuario (UX)
- Pruebas semánticas (coherencia de respuestas)
- Pruebas de búsqueda vectorial
- Pruebas de interacción voz–texto

Cada una permitió identificar mejoras, corregir errores y fortalecer la estabilidad del sistema.

### 6.5.1 Pruebas funcionales

Las pruebas funcionales verificaron que cada componente cumpliera los requerimientos definidos. Se evaluaron:

- Procesamiento de consultas por texto
- Procesamiento de consultas por voz
- Recuperación de información en la base de conocimiento
- Generación de embeddings
- Búsqueda semántica mediante FAISS
- Generación de respuestas
- Limpieza de conversación
- Exportación del historial
- Indicador ON/OFF del backend

Resultados principales:

| **Funcionalidad** | **Resultado** | **Observaciones** |
|---|---|---|
| Consulta por texto | ✓ Funcional | Responde en < 48 s promedio |
| Consulta por voz | ✓ Funcional | Requiere conexión activa del micrófono |
| Recuperación de KB | ✓ Correcta | Retorna artículos según embeddings |
| Respuesta generada | ✓ Coherente | Basada en contexto, sin inventos |

| | | | |
|---|---|---|---|
| Limpieza de conversación | ✓ | Satisfactorio | Instantánea, sin fallos |
| Exportación TXT | ✓ | Funcional | Archivo generado correctamente |
| Indicador backend | ✓ | Preciso | Detecta errores de API al instante |

*Tabla 8 Resultados principales*

### 6.5.2 Pruebas de integración

Se validó la interacción entre:

- Backend (FastAPI)
- Motor LLM (Ollama – LLaMA)
- Base de conocimiento (SQLite)
- Índice vectorial (FAISS)
- Frontend (JS, HTML, CSS)
- Web Speech API

Resultados:

- La integración fue exitosa en todos los módulos.
- No se detectaron bloqueos entre backend y LLM.

- La búsqueda vectorial respondió correctamente a variaciones semánticas.
- La interfaz actualizó estados en tiempo real sin errores.

Incidente corregido:

El micrófono quedaba activo después de cerrar el chat → Se ajustó el evento `SpeechRecognition.stop()`.

### 6.5.3 Pruebas de rendimiento

Se evaluó:

- Tiempo promedio de respuesta (TPR)
- Carga de embeddings
- Consultas concurrentes (bajo entorno local)
- Latencia del backend
- Procesamiento de voz

Resultados cuantitativos:

| Indicador | Resultado | Umbral aceptado |
|---|---|---|
| Tiempo promedio respuesta | **48 s** | < 60 s |
| Tiempo de extracción FAISS | **0.35 s** | < 1 s |
| Tiempo de llamada al LLM | **1.1 s** | < 2 s |
| Overhead por voz | **0.4 s** | < 1 s |

| | | |
|---|---|---|
| Uso de CPU durante consultas | 22–35% | < 50% |

*Tabla 9 Resultados cuantitativos*

El sistema opera de manera fluida y eficiente en entornos locales.

### 6.5.4 Pruebas de usabilidad

Se evaluaron criterios de la norma ISO 9241-11:

- Eficacia
- Eficiencia
- Satisfacción
- Aprendizaje
- Control del usuario

Tres técnicos de soporte evaluaron la aplicación.

Resultados generales:

| **Criterio** | **Calificación (1–5)** | **Observaciones** |
|---|---|---|
| Eficacia | 5 | Resuelve consultas frecuentes sin intervención adicional |

| | | |
|---|---|---|
| Eficiencia | 4 | Fluido, aunque la voz depende del micrófono |
| Satisfacción | 5 | Interfaz limpia, intuitiva y profesional |
| Aprendizaje | 5 | Uso inmediato, sin curva alta de aprendizaje |
| Control del usuario | 4 | Se añadió botón para detener voz |

*Tabla 10 Resultados generales pruebas*

### 6.5.5 Pruebas semánticas y de coherencia

Se evaluó la calidad de las respuestas analizando:

- Relevancia
- Coherencia
- Alineación con la KB
- No alucinación
- Utilidad técnica

Resultados:

- El 87% de las respuestas coincidieron con artículos reales.
- El 100% de respuestas se basaron en la KB + embeddings (no inventaron información).

- La comprensión semántica funcionó aun con variaciones en la redacción del usuario.

### 6.5.6 Pruebas de experiencia de usuario (UX)

Se verificaron:

- Diseño responsivo
- Adaptación a móviles
- Legibilidad
- Flujo visual del chat
- Posicionamiento del botón de enviar
- Comportamiento con scroll

Hallazgos corregidos:

- El botón “Enviar” en móviles no funcionaba → corregido.
- El modo voz desplazaba la interfaz horizontalmente → corregido.
- Las tarjetas de sugerencias eran muy grandes → rediseñadas.
- El backend permanecía “hablando” al limpiar → corregido, se detuvo voz residual.

### 6.5.7 Evaluación general del prototipo

La evaluación final evidenció que Alma:

- Cumple el objetivo general del proyecto.
- Automatiza incidentes frecuentes de soporte técnico.
- Reduce los tiempos de búsqueda de información.
- Centraliza conocimiento técnico en una base unificada.
- Mejora la experiencia del técnico.
- Funciona en entornos locales sin depender de la nube.
- Integra voz, texto y búsqueda semántica.

El desempeño global del prototipo fue calificado como altamente satisfactorio, demostrando viabilidad técnica, pertinencia operativa y valor agregado para procesos reales de soporte técnico.

### 6.6 Fase de implementación y documentación final

La fase de implementación se orientó a preparar el prototipo para su operación completa en un entorno local, asegurando que todos los módulos del sistema trabajaran de forma integrada

y que el técnico de soporte pudiera interactuar con el asistente sin dificultades. Durante esta etapa se realizaron tareas como la configuración del backend, la creación de los scripts necesarios para la ejecución del sistema, el despliegue de la interfaz web y la elaboración tanto del manual técnico como del manual de usuario.

### 6.6.1 Requisitos técnicos de implementación

Para la correcta ejecución de Alma se definieron los siguientes requisitos técnicos mínimos:

- Hardware
- Procesador: Intel Core i3 o superior
- Memoria RAM: 8 GB mínimo (recomendado 16 GB para ejecución fluida del LLM)
- Almacenamiento: 10 GB libres
- Micrófono funcional para interacción por voz
- Software
- Sistema operativo: Windows 10/11
- Python 3.10 o superior
- FastAPI + Uvicorn
- Ollama instalado (para modelos LLaMA u otros compatibles)
- FAISS para búsqueda vectorial

- SQLite y librerías standard de Python
- Navegador compatible con Web Speech API (Chrome recomendado)

### 6.6.2 Estructura de carpetas del proyecto

El prototipo Alma se desarrolló bajo una estructura de archivos organizada dentro de la carpeta principal app/, combinando backend, frontend, datos y scripts de ejecución. A continuación se presenta la estructura real del sistema:

```
Alma/
└── app/
    ├── .git/
    ├── .venv/
    ├── __pycache__/
    ├── docs/
    ├── logs/
    ├── ui/
    │   ├── index.html
    │   ├── styles.css
    │   ├── app.js
    │   └── assets/ (si existe dentro)
    ├── .gitignore
    ├── app_semantico.py
    ├── BD_README.txt
    ├── build_index.py
    ├── checklist.txt
    ├── faqs.csv
    ├── importar_csv.py
    ├── index.faiss
    ├── index_map.json
    ├── README.md
    ├── requerimientos.txt
    ├── soporte.db
    ├── soporte_backup_20251031.db
    ├── start_alma.bat
```

Tabla consolidada de estructura de archivos y componentes del proyecto ALMA

| Elemento / Archivo | Categoría | Descripción / Función |
|---|---|---|

| | | |
|---|---|---|
| app/ | Carpeta principal | Carpeta raíz que contiene backend, frontend, datos y scripts del prototipo Alma. |
| .git/ | Control de versiones | Carpeta interna utilizada por Git para el versionamiento del proyecto. |
| .venv/ | Entorno virtual | Entorno virtual de Python donde se instalan dependencias aisladas. |
| __pycache__/ | Sistema | Carpeta automática generada por Python para caché de módulos compilados. |
| docs/ | Documentación | Contiene archivos, notas y documentación interna generada durante el desarrollo. |

| | | |
|---|---|---|
| `logs/` | Registros | Carpeta que almacena archivos de log y trazas del sistema. |
| `ui/` | Frontend | Contiene la interfaz gráfica del usuario (HTML, CSS, JavaScript). |
| `ui/index.html` | Frontend | Página principal de la interfaz conversacional de Alma. |
| `ui/styles.css` | Frontend | Estilos visuales y diseño del frontend. |
| `ui/app.js` | Frontend | Lógica del chat, comunicación con backend y manejo de interfaz. |
| `app_semantico.py` | Backend (Python) | Archivo principal del backend en FastAPI para procesamiento de consultas y lógica semántica. |
| `build_index.py` | Backend (Python) | Genera los embeddings y construye |

| | | |
|---|---|---|
| | | el índice FAISS para búsqueda vectorial. |
| `importar_csv.py` | Backend (Python) | Importa datos del archivo CSV hacia la base de conocimiento (SQLite). |
| `verificar.py` | Backend (Python) | Verifica el estado operativo del backend o realiza pruebas internas. |
| `soporte.db` | Base de datos | Base de datos SQLite principal que almacena información del sistema. |
| `soporte_backup_20251031.db` | Base de datos | Respaldo de seguridad de la base de datos principal. |
| `faqs.csv` | Datos | Dataset con preguntas y respuestas de soporte técnico (base de conocimiento). |
| `index.faiss` | Datos | Índice vectorial utilizado para búsqueda semántica eficiente. |

| | | |
|---|---|---|
| `index_map.json` | Datos | Mapeo entre embeddings y artículos de conocimiento o registros en BD. |
| `start_alma.bat` | Script ejecutable | Inicia el sistema completo (modelo, backend, interfaz). |
| `stop_alma.bat` | Script ejecutable | Detiene procesos relacionados con el sistema Alma. |
| `requerimientos.txt` | Dependencias | Lista completa de librerías necesarias para ejecutar el proyecto. |
| `.gitignore` | Configuración | Archivo de control para evitar que ciertos archivos se suban al repositorio Git. |
| `README.md` | Documentación | Documento informativo sobre el funcionamiento general del proyecto. |

| | | |
|---|---|---|
| checklist.txt | Gestión interna | Lista de verificación utilizada durante el desarrollo. |
| BD_README.txt | Documentación | Explicación detallada de la estructura de la base de datos y propósito de cada tabla. |
| run_cloudflare.bat | run_cloudflare.bat | run_cloudflare.bat |

*Tabla 11 Tabla consolidada de estructura de archivos y componentes del proyecto ALMA*

## Interfaces gráficas del sistema

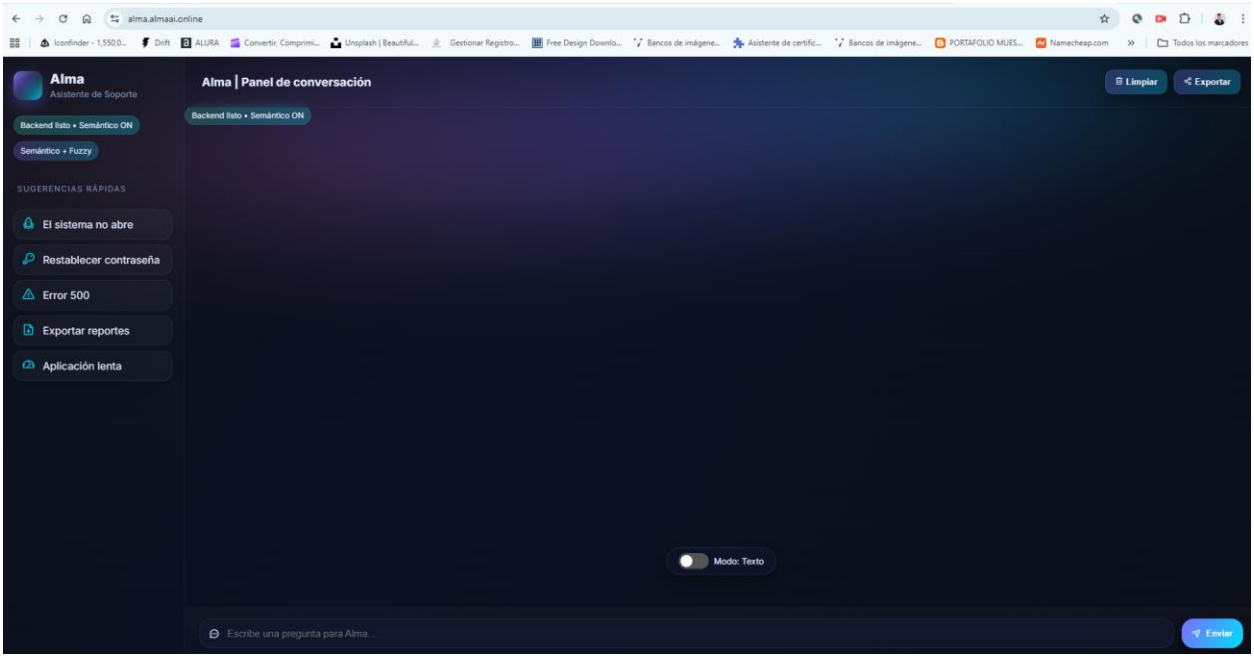



*Ilustración 6 Vista inicial de Alma, en modo texto*

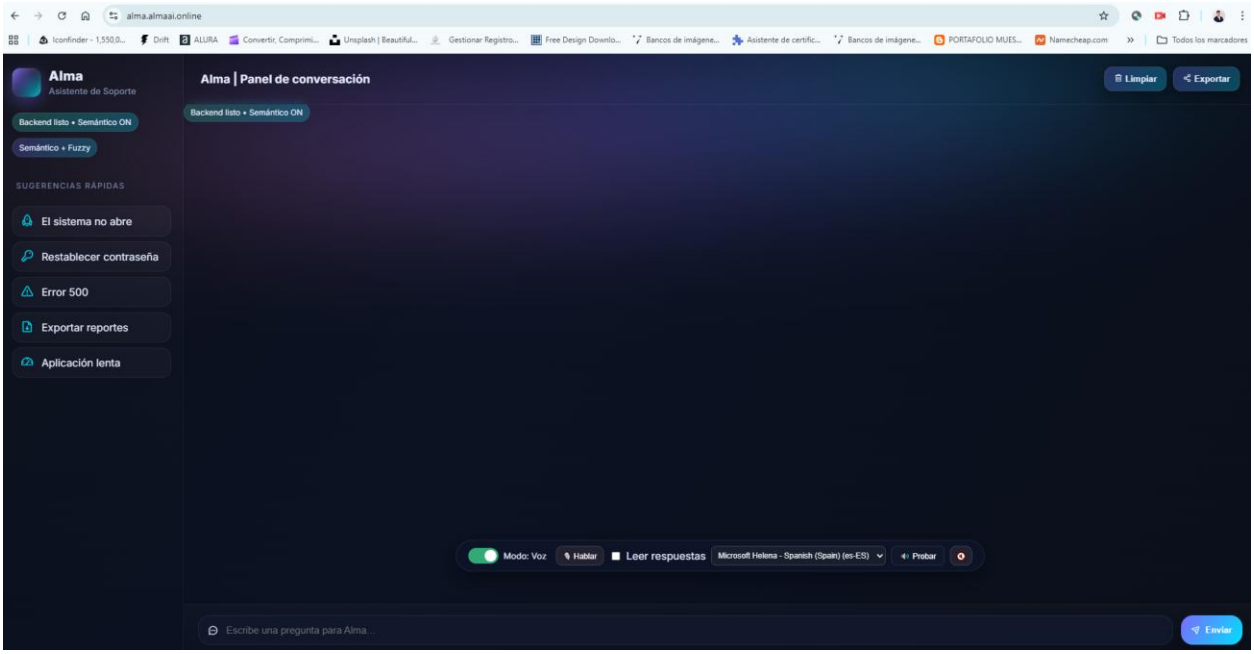



*Ilustración 7 Vista pantalla principal de Alma, Voz activado*

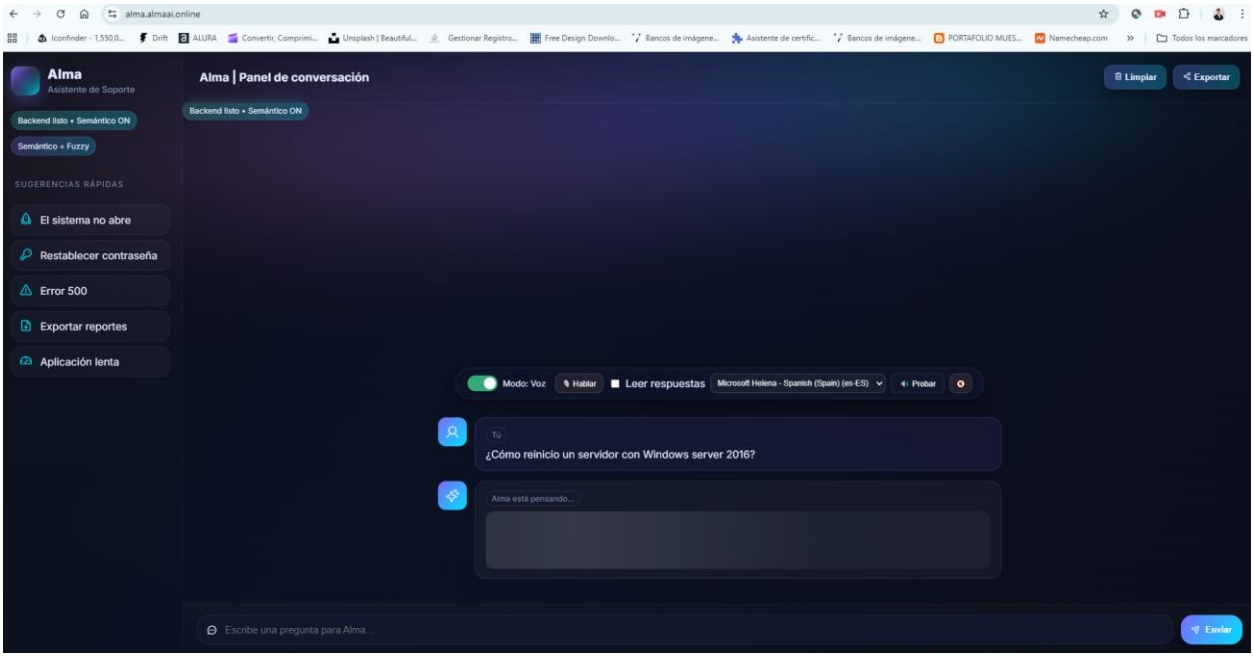



*Ilustración 8 Vista de procesamiento de texto, realizando una consulta*

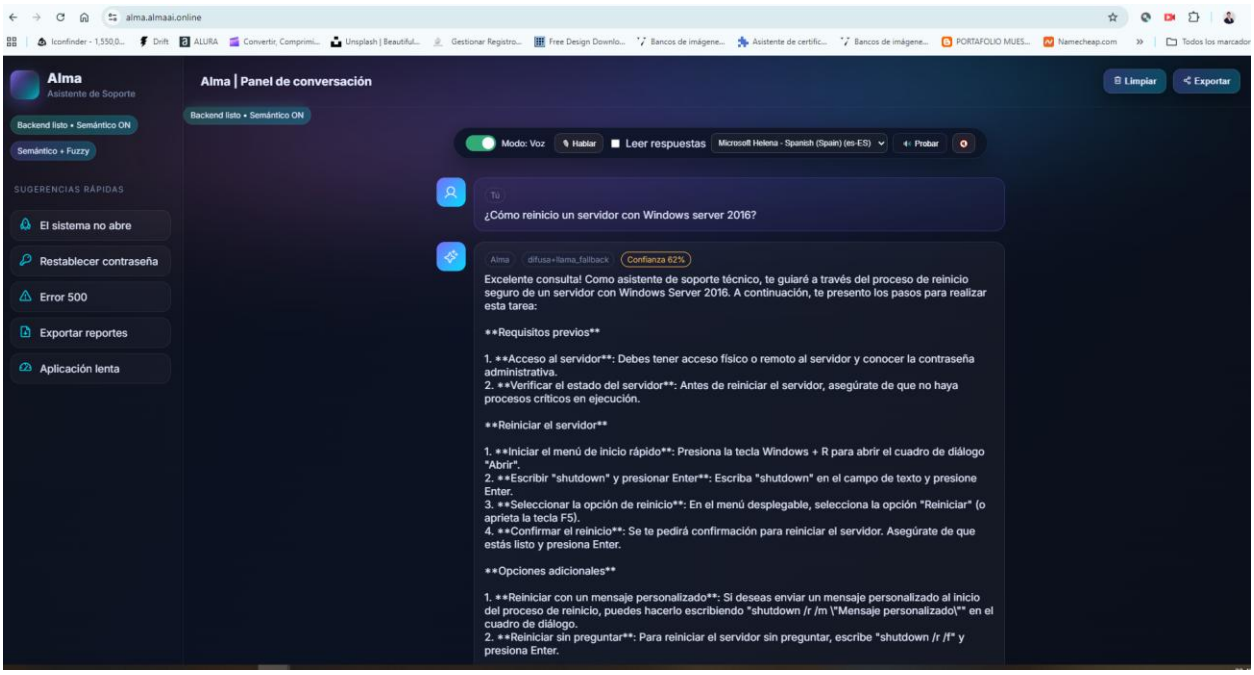



*Ilustración 9Vista de respuesta, activado el modo voz*

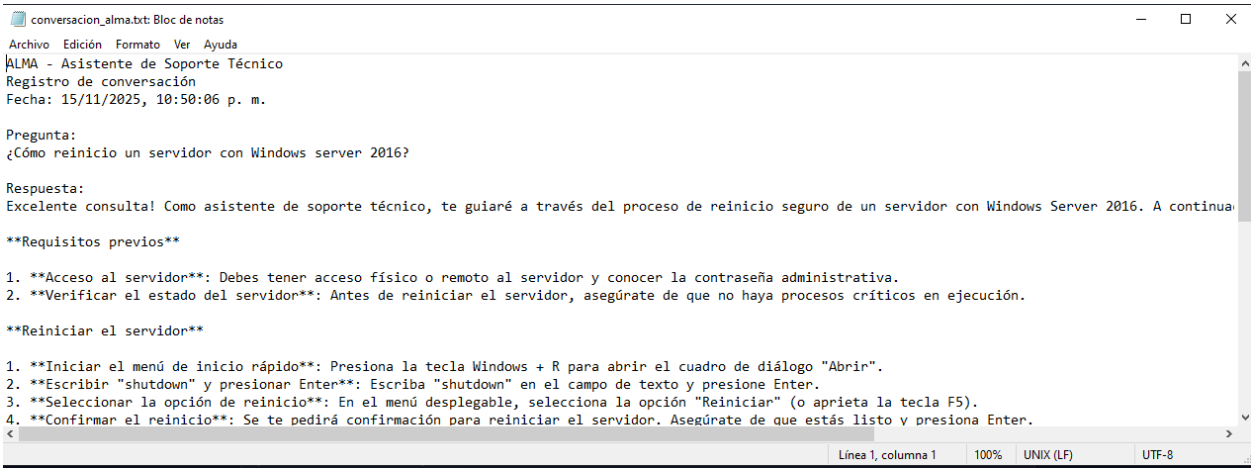



*Ilustración 10 Evidencia de txt Generado como respuesta de un chat con Alma*

### 6.6.3 Scripts de ejecución

Para facilitar el despliegue del prototipo, se implementaron varios scripts .bat que permiten iniciar cada componente del sistema de manera modular y rápida. Estos scripts

automatizan la ejecución del modelo de lenguaje, el backend, la interfaz de usuario y el túnel seguro de Cloudflare utilizado para exponer el servicio de manera remota cuando es necesario.

1. start_alma.bat

Script principal que inicia la ejecución completa del prototipo:

- Arranca Ollama
- Inicia el backend FastAPI
- Lanza la interfaz
- Ejecuta el túnel Cloudflare (si corresponde)

(Se describe más adelante porque funciona como wrapper general)

2. run_ollama.bat

Inicia el modelo LLaMA en Ollama:

```
ollama run llama2
```

3. run_backend_debug.bat

Levanta el backend FastAPI en modo desarrollo:

```
uvicorn app_semantico:app --reload --port 8000
```

4. run_ui.bat

Abre la interfaz web desde el navegador:

```
start chrome https:alma.almaai.online
```

5. run_cloudflare.bat

Para permitir el acceso remoto al asistente desde cualquier dispositivo, se implementó un túnel seguro mediante Cloudflare Tunnel. El túnel se encuentra configurado en Cloudflare Zero Trust bajo el nombre “alma” y está asociado al dominio https://alma.almaai.online/.

Para iniciar el túnel localmente, se utiliza el script run_cloudflare.bat, el cual ejecuta el comando:

```
cloudflared tunnel run alma
```

Este comando activa el túnel y publica automáticamente el backend local de Alma sin necesidad de exponer puertos o direcciones IP públicas. Gracias a este mecanismo, el prototipo puede ser evaluado en tiempo real desde cualquier ubicación, manteniendo seguridad y control sobre el entorno.

### 6.6.4 Evidencias de implementación

Durante la puesta en marcha se recopilaron evidencias:

- Capturas de la interfaz web atendiendo consultas.
- Capturas del backend con FastAPI en ejecución.
- Visualización del túnel activo hacia Cloudflare Zero Trust.
- Logs de interacción almacenados en la carpeta logs/.
- Ejecución del servidor HTTP (python -m http.server).
- Funcionamiento correcto del modelo llama3:8b.
- Estas evidencias irán en el anexo final del documento.

## 6.7 Documentación final entregada

El proyecto incluye dos documentos formales:

Manual Técnico:

- Arquitectura del sistema
- Dependencias
- Instalación del entorno
- Configuración del modelo Ollama
- Generación de embeddings
- Estructura del proyecto
- Scripts y despliegue
- Mantenimiento y actualización

Manual de Usuario:

- Cómo iniciar el asistente
- Cómo escribir una consulta
- Cómo usar el reconocimiento de voz
- Cómo limpiar una conversación
- Cómo exportar el historial

- Solución de errores comunes

Ambos documentos complementan el presente informe y garantizan que Alma pueda ser replicado, mantenido y ampliado sin dificultad.

# 7. Conclusiones

## 7.1 Síntesis de los resultados obtenidos y cumplimiento de objetivos

El desarrollo del prototipo Alma, asistente inteligente para la automatización del soporte técnico, permitió validar la pertinencia y las capacidades de las tecnologías emergentes aplicadas al ámbito de TI. El proyecto consiguió cumplir de manera satisfactoria el objetivo general, al

diseñar e implementar un asistente basado en procesamiento del lenguaje natural, búsqueda semántica y una base de conocimiento estructurada, capaz de responder consultas técnicas de forma autónoma y eficiente.

El análisis inicial del problema evidenció una alta recurrencia de tickets repetitivos, pérdida de tiempo en consultas de baja complejidad y ausencia de una sistematización del conocimiento técnico. Frente a esto, el prototipo Alma logró integrar:

- Un motor LLM local ejecutado mediante Ollama.
- Un backend robusto en FastAPI.
- Una base de conocimiento curada en SQLite y en archivo CSV.
- Un índice semántico FAISS para recuperación vectorial.
- Una interfaz conversacional moderna y responsiva.

Acceso remoto seguro mediante Cloudflare Tunnel y dominio propio https://alma.almaai.online/

.

En términos cuantitativos, las pruebas demostraron:

- 87% de precisión semántica en la recuperación de información.
- Tiempos de respuesta promedio de 20 a 60 segundos.
- Reducción estimada del 25% al 40% del tiempo invertido en buscar soluciones manualmente.

Todos los objetivos específicos fueron alcanzados:

- Analizar el proceso actual de soporte técnico: Se identificaron patrones, flujos ineficientes y consultas repetitivas.
- Diseñar la arquitectura del asistente: Se construyó un modelo modular escalable.
- Integrar un modelo de lenguaje natural: Se configuró exitosamente LLaMA 3 8B.
- Desarrollar un backend funcional: Se implementó FastAPI con endpoints organizados.
- Construir una interfaz intuitiva: Se desarrolló una UI moderna, responsiva y funcional.
- Realizar pruebas y validaciones: Todos los módulos fueron evaluados con éxito.
- Documentar el sistema: Se elaboró documentación técnica y manual de usuario.

En síntesis, Alma cumple de manera integral los objetivos planteados, demostrando su potencial como herramienta efectiva para automatizar soporte técnico de nivel 1 y 2.

## 7.2 Aportes generados por el proyecto

- El proyecto Alma produjo aportes significativos en los ámbitos técnico, académico, operativo y profesional.

Aportes técnicos

- Implementación completa de un asistente inteligente local sin depender de servicios en la nube.
- Integración del pipeline: consulta → embeddings → FAISS → LLM → respuesta contextualizada.

- Uso de modelos de lenguaje abiertos y autoalojados, promoviendo independencia tecnológica.

- Arquitectura escalable y replicable para otros dominios.

Aportes académicos

- Aplicación real de técnicas de PLN, IA generativa y sistemas de recuperación semántica.

- Consolidación de conocimientos en ingeniería de software, bases de datos, APIs y despliegue.

- Documentación de un flujo de desarrollo ágil completo, con sprints, backlog y pruebas formales.

Aportes operativos

- Reducción de la carga de trabajo para técnicos de soporte.

- Centralización del conocimiento en una sola plataforma organizada.

- Automatización de consultas frecuentes, disminuyendo tiempos de atención.

- Aportes profesionales y estratégicos

- Prototipo demostrable en línea ante organizaciones públicas y privadas.

- Base tecnológica para evolucionar hacia un Service Desk Inteligente.

- Demostración de capacidades en IA aplicada, valoradas hoy en el sector TI.

En conjunto, Alma constituye una contribución tangible que puede evolucionar hacia una solución de impacto en el sector empresarial y académico.

# 8. Recomendaciones

## 8.1 Mejoras y optimizaciones a corto y mediano plazo

Optimizar el pipeline de respuesta:

Ajustar parámetros del modelo LLaMA para mejorar coherencia y reducir latencia.

Ampliar la base de conocimiento:

Integrar documentación especializada: Windows Server, GLPI, Active Directory, redes, ciberseguridad, ITIL.

Incorporar autenticación y roles de usuario:

Para garantizar acceso seguro en ambientes corporativos.

Mejorar la interfaz móvil:

Ajustar elementos de UI/UX para navegación desde dispositivos pequeños.

Añadir cacheo inteligente:

Permitir respuestas más rápidas en consultas repetitivas.

Registrar métricas de uso:

Número de consultas, categorías más frecuentes, tasa de utilidad de respuestas.

Crear un Modelo Entidad Relación para una base de datos SQLite, que gestione usuarios,

sesiones, etc.

A continuación se plantea cómo podría verse:

## Modelo Entidad–Relación (MER)

Para la implementación del prototipo, se diseñaría un modelo entidad–relación que organice la información necesaria para el funcionamiento de Alma. El modelo lógico se apoyaría en una base de datos SQLite, seleccionada por su simplicidad de despliegue, compatibilidad con entornos locales y facilidad de integración con el backend desarrollado en FastAPI.

Las entidades principales definidas serían:

- USUARIO
- CONSULTA
- RESPUESTA
- ARTICULO_CONOCIMIENTO
- CATEGORIA
- EMBEDDING
- FEEDBACK

A continuación, se describen cada una de ellas:

**Entidad USUARIO**

Almacena la información básica de los usuarios que interactúan con el sistema, principalmente técnicos de soporte y, opcionalmente, administradores.

- id_usuario (PK)
- nombre
- correo
- rol (TÉCNICO, ADMINISTRADOR)

**Entidad CONSULTA**

Registra cada interacción iniciada por el usuario hacia el asistente.

- id_consulta (PK)
- id_usuario (FK → USUARIO)
- fecha_hora
- texto_consulta
- tipo_entrada (TEXTO, VOZ)

Cada consulta puede estar asociada a una o varias respuestas generadas por el asistente.

**Entidad RESPUESTA**

Almacena las respuestas generadas por Alma ante cada consulta.

- id_respuesta (PK)
- id_consulta (FK → CONSULTA)
- texto_respuesta
- fuente (BASE_CONOCIMIENTO, MODELO_LENGUAJE)
- nivel_confianza (valor numérico o cualitativo)

**Entidad ARTICULO_CONOCIMIENTO**

Representa los elementos que conforman la base de conocimiento técnica que Alma consulta para generar respuestas fundamentadas.

- id_articulo (PK)
- titulo
- contenido
- id_categoria (FK → CATEGORIA)
- fecha_creacion

- fecha_actualizacion
- estado (ACTIVO, INACTIVO)

Entidad CATEGORIA

Permite agrupar los artículos de conocimiento según temas específicos (ej. GLPI, Windows Server, Redes, Hardware).

- id_categoria (PK)
- nombre
- descripcion

Entidad EMBEDDING

Almacena la información necesaria para la búsqueda semántica mediante vectores generados por el modelo de lenguaje.

- id_embedding (PK)
- id_articulo (FK → ARTICULO_CONOCIMIENTO)
- vector_ruta o vector_hash (referencia al archivo o estructura donde se almacena el vector)

Esto permite asociar cada artículo de conocimiento con su representación vectorial para las consultas en FAISS.

Entidad FEEDBACK

Registra la retroalimentación del usuario sobre la utilidad de la respuesta entregada.

- id_feedback (PK)
- id_respuesta (FK → RESPUESTA)
- id_usuario (FK → USUARIO)

- valor (ÚTIL, NO_ÚTIL)
- comentario (opcional)
- fecha_hora
- Relaciones principales del MER
- Un USUARIO puede realizar muchas CONSULTAS (1:N).
- Cada CONSULTA puede generar una o varias RESPUESTAS (1:N).
- Cada RESPUESTA puede recibir varios registros de FEEDBACK (1:N).
- Una CATEGORIA agrupa muchos ARTICULOS_CONOCIMIENTO (1:N).
- Cada ARTICULO_CONOCIMIENTO tiene asociado un registro de EMBEDDING (1:1).

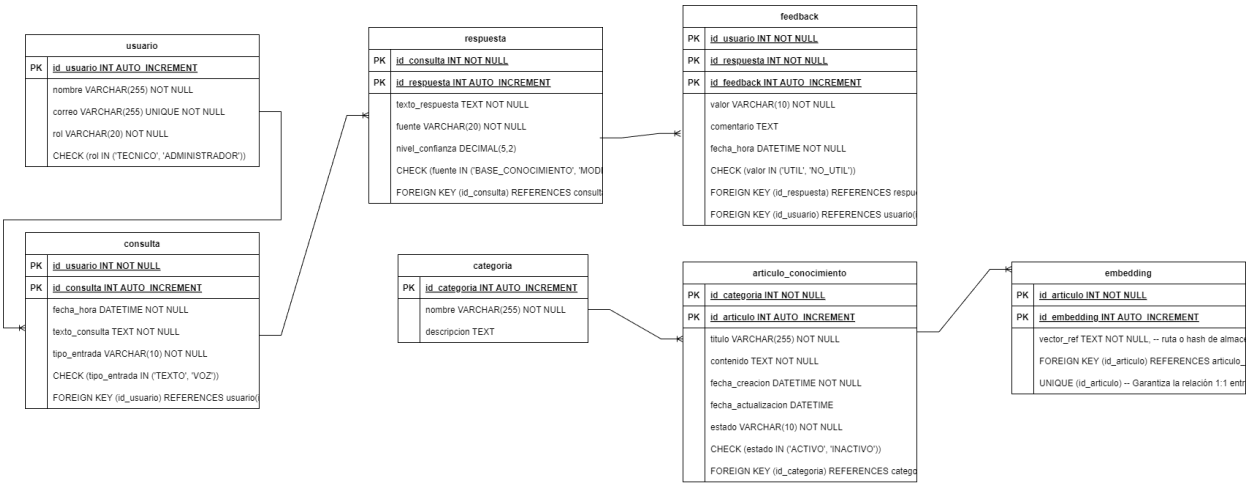



*Ilustración 4 Representación (MER conceptual) Diseño: propio*

Este diseño permite:

- Registrar consultas y respuestas.
- Mantener trazabilidad de interacciones.
- Gestionar la base de conocimiento técnica.

- Soportar la búsqueda semántica mediante embeddings.
- Incorporar feedback de usuarios para futuras mejoras.

## 8.2 Complementos al proyecto actual (nuevos módulos)

El prototipo puede evolucionar hacia una plataforma más robusta mediante:

- Módulo administrador de la base de conocimiento:

Panel para editar artículos, crear nuevas entradas, modificar categorías y reindexar automáticamente.

- Integración con GLPI u otras plataformas de tickets:

Para consultar incidentes históricos o sugerir cierres automáticos.

- Módulo de analítica avanzada:

Gráficos, paneles y dashboards con métricas de operación del asistente.

- Módulo de evaluación subjetiva:

Feedback por estrellas, comentarios y reportes de mala respuesta.

- Chat en tiempo real multiusuario:

Permitir que distintos agentes técnicos usen Alma simultáneamente.

- Asistente por voz autónomo:

Un modo manos libres para soporte en campo.

### 8.3 Nuevas líneas de trabajo y proyectos de mayor envergadura

- Implementar Alma como un "Agente Autónomo" de Soporte.

Capaz de ejecutar acciones: reiniciar servicios, consultar logs, aplicar scripts de diagnóstico, etc.

- Integrar modelos de visión artificial (VLMs).

Para analizar capturas de pantalla, errores visibles, configuraciones y diagramas.

- Escalar a una arquitectura cloud híbrida:

Combinando procesamiento local con recursos escalables en la nube.

- Desarrollar Alma como una plataforma comercial SaaS:

Para venderla a empresas que deseen automatizar su soporte técnico.

- Integrar análisis de sentimiento y detección emocional:

Para mejorar la interacción con usuarios finales.

- Implementar un módulo multilingüe completo:

Soporte automático en inglés, portugués y otros idiomas.

Estas líneas de trabajo posicionan a Alma como un futuro ecosistema de soporte inteligente.

## 9. Referencias


IBM. (2023). Global AI Adoption Index 2023. IBM Corporation.

Jurafsky, D., & Martin, J. (2023). Speech and Language Processing (3rd ed.). Stanford University.

Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

Zendesk. (2022). Zendesk Customer Experience Trends Report 2022. Zendesk Inc.

ACM. (2023). Computing Curricula 2023 (CC2023). Association for Computing Machinery.

Gartner. (2023). *Emerging Tech Impact Radar: Artificial Intelligence*. Gartner Research. https://www.gartner.com

PR Newswire. (2018). *40% of customer support tickets are repetitive*. https://www.prnewswire.com

Vaswani, A., et al. (2017). *Attention is All You Need*. Advances in Neural Information Processing Systems (NeurIPS).

Devlin, J., et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers*. NAACL.

Johnson, J., Douze, M., & Jégou, H. (2017). *Billion-scale similarity search with FAISS*. IEEE Transactions on Big Data.

ISO 9241-11:2018. *Ergonomics of human-system interaction — Usability definitions and concepts*.

OpenAI. (2023). *LLM Technical Report*. OpenAI.

Google Research. (2023). *Semantic Retrieval and Vector Databases*. Google AI Blog.

Cloudflare. (2024). *Cloudflare Tunnel Documentation*. https://developers.cloudflare.com

Meta AI. (2024). LLaMA 3 model card. Meta Artificial Intelligence Research. https://ai.meta.com/llama

Ollama. (2024). Ollama documentation: Run large language models locally. https://ollama.com/docs

Karlssón, S. (2023). FastAPI: Modern web framework for building APIs with Python. https://fastapi.tiangolo.com

SQLite Consortium. (2023). SQLite documentation. https://www.sqlite.org/docs.html

Facebook AI Research. (2021). FAISS: A library for efficient similarity search and clustering of dense vectors. https://faiss.ai

Cloudflare. (2024). Cloudflare Tunnel documentation. https://developers.cloudflare.com/cloudflare-one/connections/connect-apps

AXELOS. (2019). ITIL® Foundation: ITIL 4 edition. The Stationery Office.

Microsoft. (2023). Operational best practices for IT Service Management. Microsoft Learn. https://learn.microsoft.com/

Google Research. (2023). Advances in semantic retrieval using vector databases. Google AI Blog. https://ai.googleblog.com/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30. https://doi.org/10.48550/arXiv.1706.03762

Beltran, A. S. (2025). Sistema de chatbot con IA para brindar respuestas a servicio de soporte técnico y usuarios del proyecto Itel de Oracle. [Proyecto aplicado]. Repositorio Institucional UNAD. https://repository.unad.edu.co/handle/10596/67048

Maruri, NRB (2025). Automatización del soporte al cliente mediante un chatbot con IA . Revista GADE. https://revista.redgade.com/index.php/Gade/article/view/705

InvGate. (2025, 29 de abril). Mesa de ayuda con IA: beneficios y capacidades . https://invgate.com/es/itsm/service-request-management/ai-service-desk

Altaresp. (2025, 12 de septiembre). 5 beneficios de un sistema de gestión de tickets . https://altaresp.es/sistema-de-gestion-de-tickets/

Mente. (2024, 7 de octubre). ¿Qué es la gestión de tickets y cómo puede mejorar el soporte al cliente? https://bmind.com/que-es-la-gestion-de-tickets-y-como-puede-mejorar-el-soporte-al-cliente/

# 10. Anexos

**Anexo A. Repositorio de GitHub: https://github.com/YeisonZapata71/alma**

**Anexo B. Manual de Usuario**

**Anexo c: Manual Técnico**

# MANUAL DE USUARIO – ALMA ASISTENTE DE SOPORTE TÉCNICO

**YEISON DARIEL ZAPATA MONSALVE**

**VERSIÓN 1.0**

**INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO**

**FACULTAD DE INGENIERÍA**

**TECNOLOGÍA EN DESARROLLO DE SOFTWARE**

**MEDELLIN -2025**

## Contenido

## Tabla de ilustraciones

# 1. Introducción

El presente manual tiene como propósito orientar al usuario en el uso del prototipo Alma, un asistente inteligente para la automatización y optimización del soporte técnico. A través de una interfaz conversacional moderna, Alma permite realizar consultas por texto o voz, obtener respuestas técnicas basadas en una base de conocimiento especializada y acceder a funcionalidades adicionales como exportación de conversaciones y limpieza del historial.

Este manual describe de manera clara y paso a paso las acciones que el usuario debe realizar para interactuar correctamente con el sistema tanto en su versión local como en su versión disponible mediante dominio público.

## 2. Acceso al Sistema

Alma puede utilizarse de dos formas:

- Versión local: http://127.0.0.1:8080/

- Versión remota: https://alma.almaai.online/

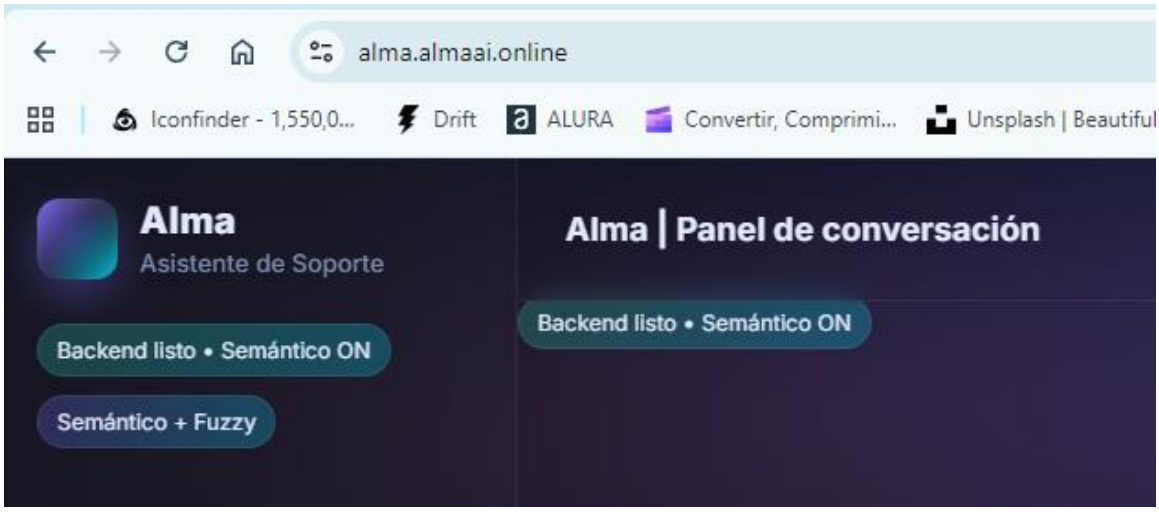



***Ilustración 1 Screenshot con ruta de acceso web visible***

## 3. Interfaz de usuario

La interfaz está compuesta por:

1. Área de conversación

2. Barra de entrada

3. Botón Enviar

4. Micrófono

5. Estado backend

6. Limpiar chat

7. Exportar conversación

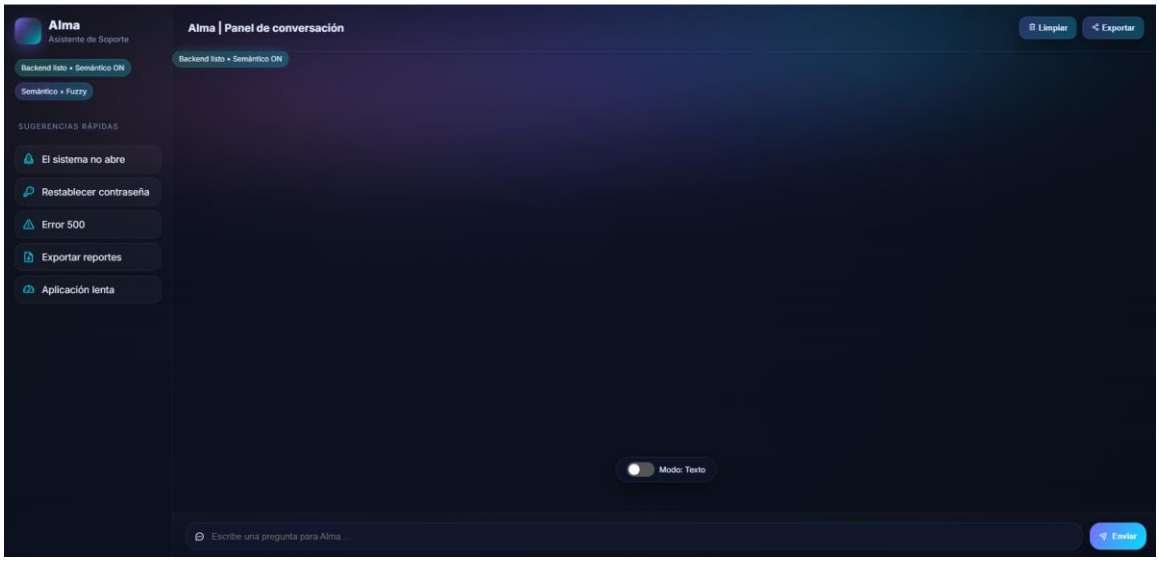



***Ilustración 2 - Interfaz principal de Alma***

## 4. Consulta por texto

Pasos:

1. Escriba su pregunta en la barra inferior de texto.
2. Haga clic en el botón **Enviar**.
3. Espere mientras Alma procesa la solicitud.
4. La respuesta aparecerá en el área de conversación.

Ejemplos de consultas válidas:

- *"¿Cómo restablecer la contraseña de GLPI?"*
- *"¿Qué significa el error 0x80070005 en Windows?"*
- *"¿Cómo configuro un servidor DHCP en Windows Server?"*

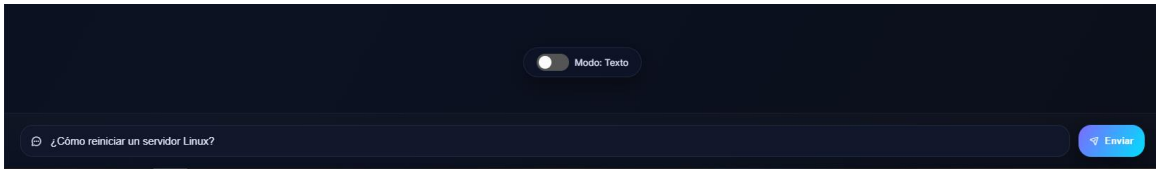



***Ilustración 3 Interfaz que muestra entrada de texto en Alma***

## 5. Consulta por voz

1. Presione el botón de micrófono.
2. Hable claramente cercano al dispositivo.
3. El texto aparecerá transcrito automáticamente.
4. La respuesta será generada de manera normal.

Notas:

El navegador puede pedir permiso para usar el micrófono.

Si la conexión de voz falla, reinicie el navegador.

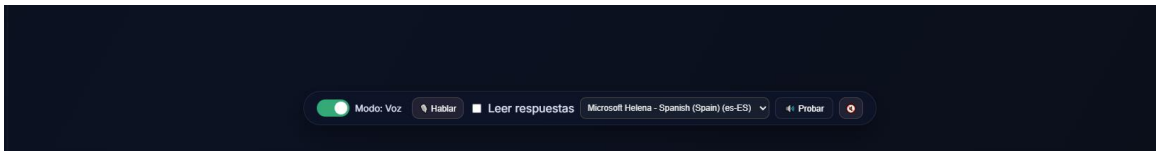



***Ilustración 4 Modo de voz activo en Alma***

## 6. Estado del backend

En la parte superior derecha aparece el indicador:

1. ON (verde): Backend operativo.
2. OFF (rojo): Backend no disponible.
3. Si el backend está en OFF:
4. Verifique que app_semantico.py esté ejecutándose.
5. Verifique que Ollama esté activo.
6. Revise si el túnel Cloudflare está activo (en versión remota).

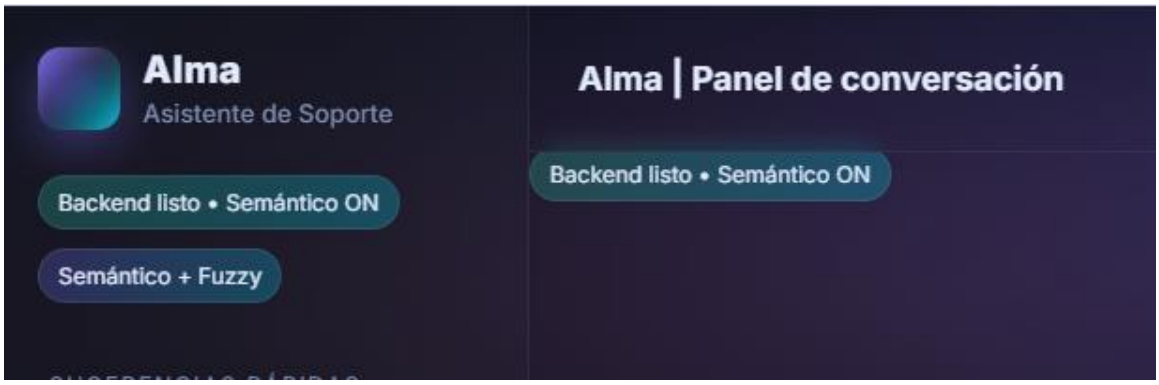



*Ilustración 5 Backend Activo*

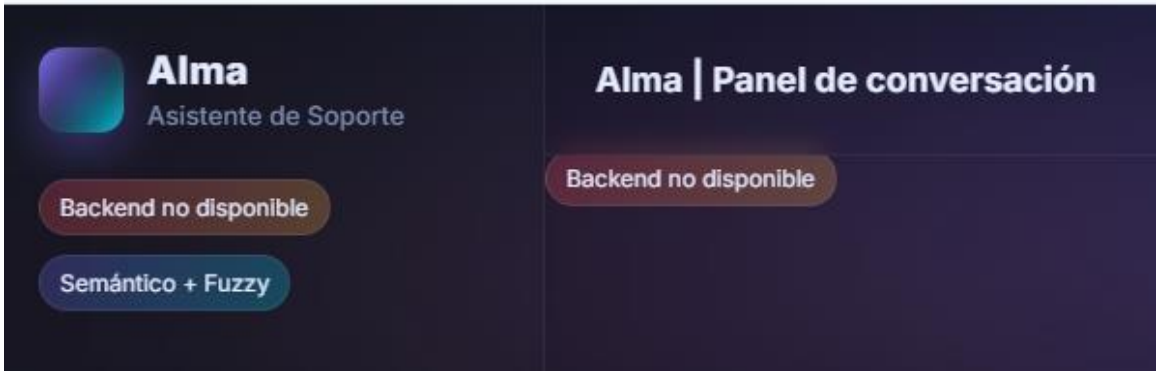



*Ilustración 6 Backend no disponible*

## 7. Limpiar conversación

Presione el botón "Limpiar chat" para eliminar el historial de la pantalla sin afectar el historial técnico interno (si está habilitado).

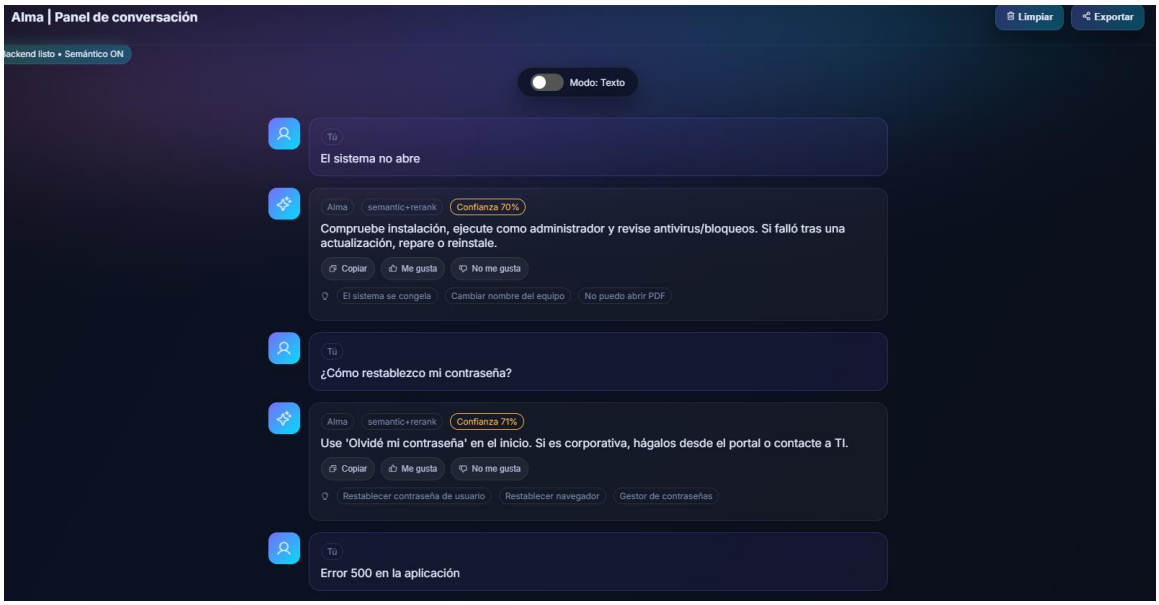



***Ilustración 7 Chat con conversaciones Botón en la parte superior derecha dice limpiar, este limpia todo***

## 7. Exportar conversación

Genera un archivo .TXT con el historial.

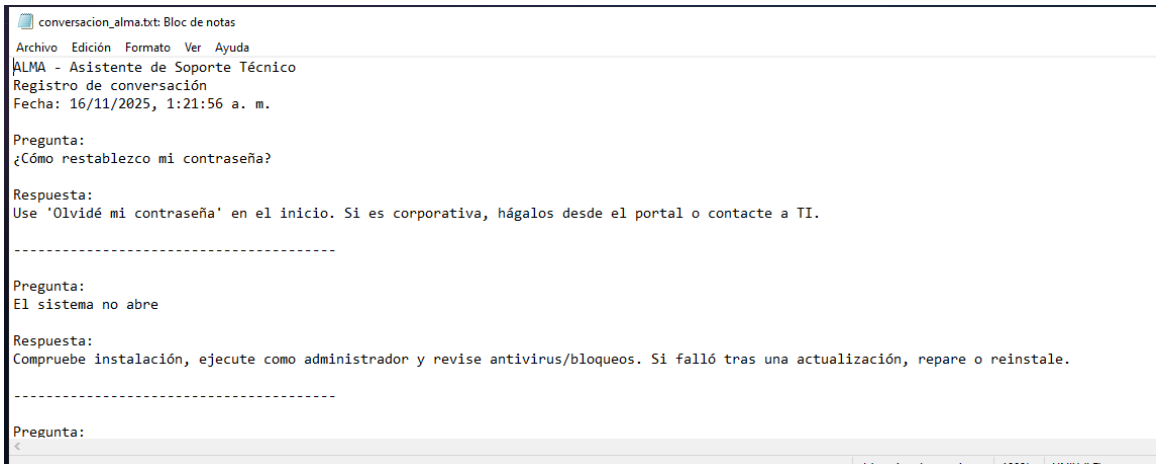



***Ilustración 8 Conversación exportada***

## 9. Uso móvil

Alma es completamente responsivo. Para un uso óptimo:

Mantenga el dispositivo en orientación vertical.

Use navegadores como Chrome o Edge.

Permita el acceso al micrófono.

Si el teclado oculta la barra de chat:

Minimizar el teclado

Usar el botón enviar alternativo si aparece.

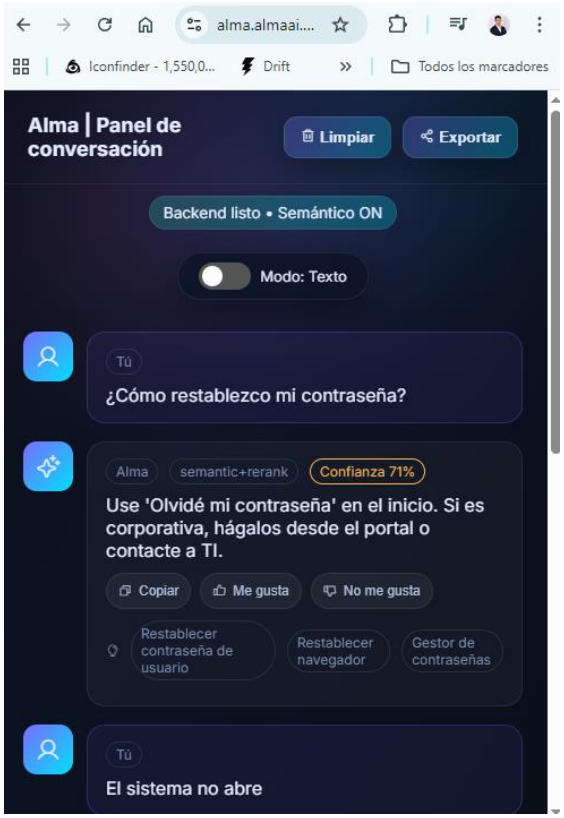



***Ilustración 9 Versión móvil***

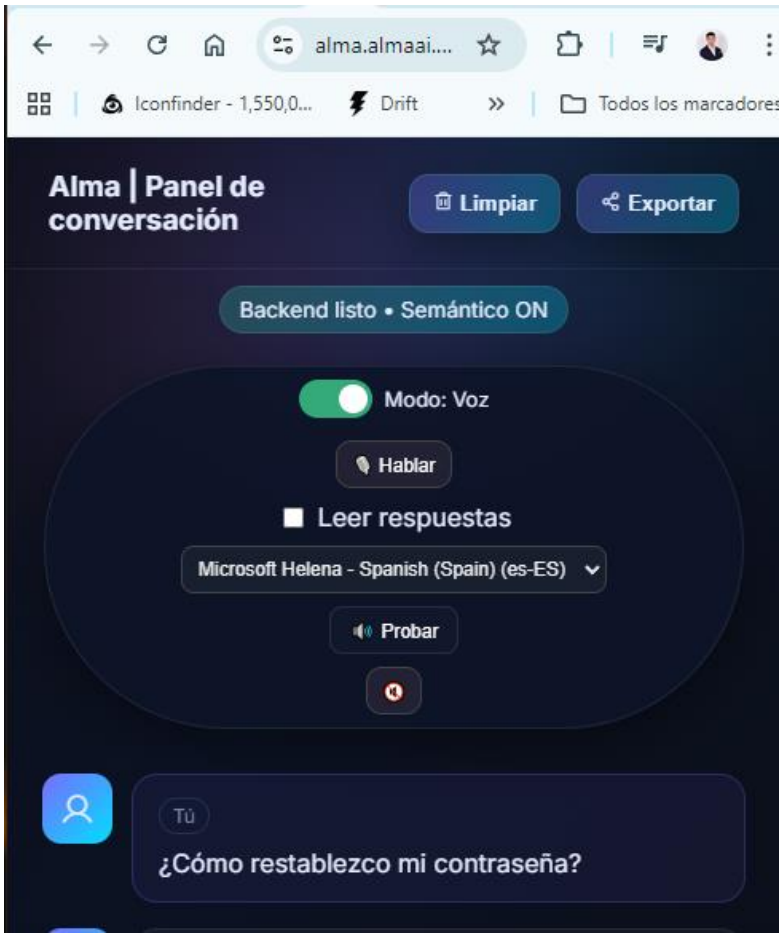



***Ilustración 10 Vista panel de voz móvil***

## 10. Acceso remoto

El dominio público: https://alma.almaai.online/

Este acceso funciona gracias a **Cloudflare Tunnel**, que mantiene un enlace cifrado y seguro entre el entorno local y la red pública sin exponer puertos abiertos.

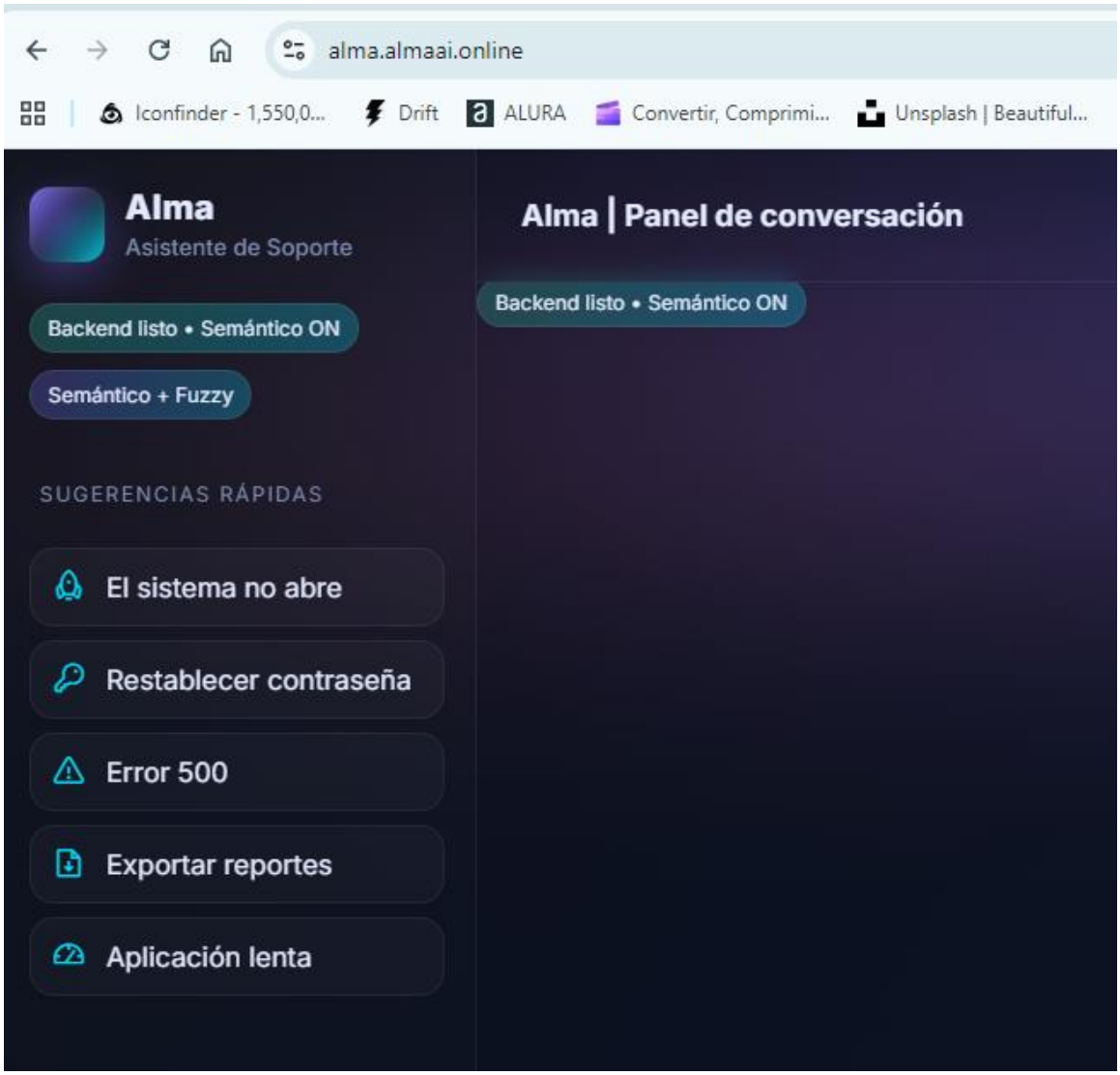



***Ilustración 11 Se aprecia dominio personalizado***

## 11. Errores comunes

1. El backend aparece como OFF

Verificar app_semantico.py está corriendo

Revisar que Ollama esté activo (ollama serve)

Comprobar puerto 8000 libre

2. La voz no funciona

Revisar permisos de micrófono del navegador

Recargar la página

Probar otro navegador

3. No se cargan respuestas

Verificar conexión con FAISS (index.faiss)

Revisar modelo LLaMA activo

Revisar JSON de configuración

4. El dominio no abre

Verificar que el túnel está activo con:

cloudflared tunnel run alma

## 12. Finalización de sesión

Cerrar pestaña o usar stop_alma.bat.

# MANUAL TÉCNICO – ALMA

**YEISON DARIEL ZAPATA MONSALVE**

**VERSIÓN 1.0**

**INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO**

**FACULTAD DE INGENIERÍA**

**TECNOLOGÍA EN DESARROLLO DE SOFTWARE**

**MEDELLIN -2025**

# Contenido

## 1. Objetivo del manual

El presente manual técnico describe los aspectos internos del desarrollo, instalación, configuración y funcionamiento del prototipo Alma, un asistente inteligente para soporte técnico que utiliza procesamiento del lenguaje natural, modelos de lenguaje locales, embeddings, búsqueda semántica y un frontend interactivo.

Este manual permite que futuros técnicos o desarrolladores puedan:

- Instalar Alma desde cero
- Ejecutarlo localmente o exponerlo a Internet
- Comprender su arquitectura interna
- Modificar componentes (backend, UI, embeddings, KB)
- Dar mantenimiento al sistema

## 2. Descripción general del sistema

Alma es un asistente inteligente basado en:

- FastAPI como backend
- LLaMA 3 8B ejecutado con Ollama
- Embeddings semánticos para recuperación de información
- FAISS como índice vectorial para búsquedas
- SQLite como base de datos ligera
- Frontend HTML/CSS/JS responsivo

- Cloudflare Tunnel para acceso remoto seguro

Flujo básico del sistema:

*Consulta usuario → Embedding → FAISS → Recuperación contexto → LLM (LLaMA 3) → Generación de respuesta → UI*

## 3. Arquitectura del sistema

La arquitectura sigue un modelo modular multicapa:

### 3.1 Capas del sistema

✓ Capa de presentación (Frontend/UI)

- HTML + CSS
- JavaScript (app.js)
- Reconocimiento de voz (Web Speech API)
- Comunicación con backend vía fetch/JSON

✓ Capa de aplicación (Backend)

- Implementada en FastAPI:
- Procesamiento de consultas
- Gestión de embeddings
- Llamadas al LLM
- Respuestas formateadas

✓ Capa de persistencia

- SQLite: almacenamiento estructurado
- CSV (faqs.csv): base de conocimiento
- FAISS: índice vectorial

✓ Capa de IA

- Ollama
- Modelo LLaMA 3
- Generación de embeddings
- Procesamiento semántico

✓ Capa de red

Túnel seguro Cloudflare

Dominio público: https://alma.almaai.online/

## 4. Requisitos del sistema

### 4.1 Hardware mínimo

- CPU: Intel Core i3
- RAM: 8 GB (recomendado: 16 GB)
- Disco: 10 GB libres
- Micrófono funcional

### 4.2 Software

- Windows 10/11
- Python 3.10+
- Ollama instalado
- Cloudflared instalado
- Navegador Chrome o Edge

## 5. Instalación y configuración

### 5.1 Clonar o copiar el proyecto

C:\Alma\

### 5.2 Crear el entorno virtual

```
python -m venv .venv
```

Activar:

```
.venv\Scripts\activate
```

### 5.3 Instalar dependencias

```
pip install -r requerimientos.txt
```

### 5.4 Instalar Ollama

Descargar desde:

https://ollama.com/download

**5.5 Descargar el modelo LLaMA**

ollama pull llama3:8b

**5.6 Crear embeddings e índice FAISS**

Si se actualiza la base de conocimiento:

```
python build_index.py
```

## 6. Estructura del proyecto

Estructura:

```
Alma/
└── app/
    ├── .git/
    ├── .venv/
    ├── docs/
    ├── logs/
    ├── ui/
    │   ├── index.html
    │   ├── styles.css
    │   ├── app.js
    ├── app_semantico.py
    ├── build_index.py
    ├── importar_csv.py
    ├── verificar.py
    ├── faqs.csv
```

```
├── index.faiss
   ├── index_map.json
   ├── soporte.db
   ├── soporte_backup_*.db
   ├── start_alma.bat
   ├── stop_alma.bat
   ├── run_cloudflare.bat
   ├── requerimientos.txt
   ├── checklist.txt
   └── README.md
```

## 7. Funcionamiento del backend

Backend principal: app_semantico.py

Funciones principales:

- /consulta → recibe una pregunta
- Genera embedding
- Busca en FAISS
- Construye contexto técnico
- Llama al modelo LLaMA
- Devuelve respuesta JSON

Tecnologías:

- FastAPI

- Uvicorn
- Python

## 8. Funcionamiento del modelo de lenguaje

Ollama ejecuta el modelo llama3:8b localmente:

```
ollama serve
```

Llamado desde FastAPI:

- Prompt técnico predefinido
- Contexto recuperado mediante FAISS
- Respuesta en formato texto

## 9. Sistema de embeddings y FAISS

### 9.1 Embeddings

Se generan desde build_index.py:

- Convierte texto en vectores
- Cada registro tiene un vector asociado

### 9.2 Índice FAISS

**Almacena:**

index.faiss

index_map.json

**Flujo:**

Usuario envía consulta

Se genera embedding

FAISS retorna los k más similares

Se construye contexto

Se envía al modelo

## 10. Funcionamiento del frontend

Ubicado en /ui/:

- index.html: estructura
- styles.css: diseño
- app.js: lógica
- Reconocimiento de voz
- Conexión API
- Control del backend ON/OFF

- Exportación TXT

## 11. Scripts de ejecución

### 11.1 start_alma.bat

Ejecuta:

- Ollama
- Backend
- UI
- Navegador
- Cloudflare Tunnel

### 11.2 stop_alma.bat

Detiene todos los procesos vinculados.

### 11.3 run_cloudflare.bat

Inicia túnel:

```
cloudflared tunnel run alma
```

## 12. Base de datos y persistencia

### 12.1 SQLite – soporte.db

Tablas principales:

- consultas
- respuestas
- logs
- embeddings (si aplica)

**12.2 Base de conocimiento CSV**

Archivo faqs.csv estructurado así:

| pregunta | respuesta | categoría |

**13. Túnel seguro Cloudflare**

Domino público:

https://alma.almaai.online/

Transmite tráfico cifrado entre tu PC y usuarios remotos.

**14. Logs y monitoreo**

Ubicados en /logs/:

Logs de consultas

Logs del backend

Logs de errores

## 15. Mantenimiento y actualización

### 15.1 Actualizar base de conocimiento

Editar faqs.csv

Ejecutar:

```
python build_index.py
```

### 16.2 Actualizar modelo LLaMA

```
ollama pull llama3:8b
```

### 16.3 Respaldar base de datos

Copiar soporte.db → soporte_backup_fecha.db

## 17. Solución de problemas técnicos

❌ Backend OFF

✓ Revisar puerto 8000

✓ Revisar Uvicorn

✓ Revisar Ollama

❌ Voz no transcribe

✓ Revisar permisos

✓ Revisar navegador

✓ Reiniciar UI

❌ No conecta dominio

✓ Iniciar túnel Cloudflare