Universidad de Salamanca

Escuela Politécnica Superior de Zamora

Departamento de Informática y Automática

Grado en Ingeniería Informática en Sistemas de Información

TRABAJO FIN DE GRADO

Cerebro Digital: Aplicación de notas basada en Markdown con integración RAG y análisis semántico topológico

Autor: Oswaldo Enrique González Suárez

Tutores: María Luisa Pérez Delgado, Nicolás García Martín, Adrián Fernández Roldan

Fecha de adjudicación: Noviembre de 2025

Fecha de presentación: Julio de 2026

Zamora, 2026

INTENCIONALMENTE EN BLANCO

Resumen

Objeto del estudio

El presente Trabajo Fin de Grado tiene como objeto el diseño, desarrollo e implementación de *Cerebro Digital*, un sistema avanzado de Gestión del Conocimiento Personal (PKM) que integra Inteligencia Artificial local mediante una arquitectura de Generación Aumentada por Recuperación (RAG). El proyecto aborda una problemática real en el ámbito del software empresarial: la imposibilidad de combinar análisis inteligente de documentos con soberanía absoluta del dato, dado que las soluciones PKM actuales dependen de servicios de IA alojados en la nube.

Exposición del desarrollo

El sistema se ha desarrollado en colaboración tecnológica con la empresa HP SCDS durante un período de seis meses (noviembre de 2025 – mayo de 2026), bajo una metodología híbrida que combina principios de Scrum con Prototipado Evolutivo. La arquitectura resultante es un sistema de microservicios compuesto por un backend en Go (Golang) que implementa una API RESTful con Arquitectura Limpia, una base de datos documental MongoDB para la persistencia de notas y vectores de *embeddings*, y una interfaz de usuario en React bajo el paradigma *Glassmorphism*. El motor de IA ejecuta localmente los modelos Qwen3 (0.6b) y Nomic-Embed-Text a través del orquestador Ollama, sin transferir ningún dato a servidores externos. Adicionalmente, el sistema implementa un grafo de conocimiento interactivo basado en el algoritmo de Barnes-Hut, que traduce las referencias cruzadas entre documentos en una red topológica visualizable mediante simulación de partículas.

Fuentes de información

La documentación técnica del proyecto se apoya en las referencias oficiales de las tecnologías empleadas (Go, React, MongoDB y Ollama), así como en la literatura académica de referencia en los campos de la recuperación de información semántica [1], la representación gráfica de conocimiento [3], los modelos de lenguaje cuantizados [5] y los algoritmos de simulación de partículas [2]. La normativa aplicable en materia de protección de datos y calidad del software

también constituye una fuente primaria del diseño.

Aclaraciones

El sistema garantiza que los datos del usuario no abandonan en ningún momento el dispositivo local, tanto en la fase de vectorización (*embeddings*) como en la fase de inferencia del modelo de lenguaje. Esta restricción arquitectónica, lejos de ser una limitación, constituye el principio diferenciador del proyecto respecto a las soluciones del estado del arte.

Abstract

This dissertation details the development of *Cerebro Digital*, an advanced Personal Knowledge Management (PKM) system that integrates local Artificial Intelligence through a Retrieval-Augmented Generation (RAG) architecture. Conceived as a capstone project for the Degree in Computer Engineering in Information Systems at the University of Salamanca, and developed under the technological mentorship of HP SCDS, the system ensures absolute data sovereignty by processing large language models (LLMs) exclusively on edge computing devices.

The technical implementation leverages a microservices architecture using Go for concurrent backend processing and React for the presentation layer. The application applies graph theory algorithms — specifically the Barnes-Hut hierarchical approximation — to dynamically render topological networks from Markdown cross-references, and introduces a hybrid semantic-lexical search pipeline (Graph-RAG) designed to mitigate AI hallucinations. The project also documents the deployment of fully deterministic and reproducible environments using Nix, adhering to modern DevOps standards.

Agradecimientos

El desarrollo de este Trabajo Fin de Grado y la culminación de mis estudios universitarios no habrían sido posibles sin el apoyo incondicional de las personas que me han acompañado a lo largo de este viaje.

En primer lugar, quisiera expresar mi más sincero agradecimiento a mis tutores. A María Luisa Pérez Delgado, por parte de la Universidad de Salamanca, cuya dedicación, orientación y rigor académico han sido fundamentales para dar forma a este proyecto. Asimismo, a Nicolás García Martín y Adrián Fernández Roldan, del equipo de ingeniería de HP SCDS, por su invaluable acompañamiento tecnológico, sus auditorías y por brindarme la oportunidad de desarrollarme en un entorno profesional de tan alto nivel.

A mi familia, el verdadero motor de mi vida. A mis padres, a mi hermano y a mis abuelos, por inculcarme el valor del esfuerzo y por su aliento constante, que me ha dado la fuerza para superar cada obstáculo y acortar cualquier distancia.

De manera muy especial, quiero dar las gracias a mi tía, por haber sido el pilar que hizo posible mi llegada a España para cumplir el sueño de cursar esta beca de estudios; sin tu ayuda incondicional, esta etapa no habría sido posible. A mi prima, por haber sido mi familia más cercana y mi apoyo constante desde el primer día que pisé este país.

A mis amigos, los de siempre, aquellos que he mantenido cerca en el corazón a pesar de los kilómetros que nos separan, y a los grandes amigos que he tenido la suerte de hacer durante el camino. Gracias por hacer de estos años universitarios una experiencia inolvidable.

Finalmente, a mi novia. Gracias por tu paciencia infinita, tu comprensión en los momentos de mayor carga de trabajo y por estar siempre ahí para celebrar cada pequeño hito en este proyecto.

A todos ustedes, infinitas gracias.

Índice general

Resumen I

Abstract III

Agradecimientos IV

Índice de Fragmentos de Código IX

Índice de Anexos X

Índice de Siglas y Acrónimos XI

1. Introducción y Objetivos **1**
- 1.1. Contexto y motivación . . . 1
- 1.2. Objetivos del proyecto . . . 1
  - 1.2.1. Objetivos técnicos . . . 1
  - 1.2.2. Objetivos personales y competencias . . . 2
- 1.3. Normativa y estándares aplicables . . . 2

2. Marco Teórico y Estado del Arte **3**
- 2.1. Estado del arte en la Gestión del Conocimiento Personal . . . 3
- 2.2. Fundamentos matemáticos de la representación topológica . . . 4
- 2.3. Generación Aumentada por Recuperación (RAG) . . . 4
  - 2.3.1. Justificación de la selección de modelos (Edge AI) . . . 4
- 2.4. Persistencia vectorial y similitud del coseno . . . 5

3. Metodología y Planificación **6**
- 3.1. Elección de la metodología de desarrollo . . . 6
- 3.2. Herramientas de gestión y trazabilidad . . . 6
- 3.3. Planificación temporal y fases de ejecución . . . 6
- 3.4. Estudio de viabilidad y análisis de riesgos . . . 7
  - 3.4.1. Viabilidad técnica . . . 7
  - 3.4.2. Viabilidad económica . . . 8

3.4.3. Viabilidad legal y privacidad (RGPD) 8

4. Herramientas y Tecnologías **9**
4.1. Frontend y capa de presentación 9
4.2. Backend y lógica de negocio 9
4.3. Persistencia documental 10
4.4. Inteligencia Artificial y procesamiento semántico 10
4.5. DevOps, infraestructura y reproducibilidad 11

5. Desarrollo e Implementación **12**
5.1. Arquitectura del sistema 12
5.2. Arquitectura base e inicialización de conexiones 13
5.3. Extracción de metadatos semánticos y topológicos 13
5.4. Pipeline semántico y estrategia Graph-RAG híbrida 13
5.5. Renderizado topológico y Búsqueda Neural en React 14
5.6. Interfaces optimistas (Optimistic UI) y estado efímero 15

6. Aseguramiento de Calidad y Entornos **16**
6.1. Testing de lógica de negocio y benchmarks de rendimiento 16
6.2. Pruebas de integración HTTP y mocking 17
6.3. Infraestructura reproducible (Nix y Direnv) 17

7. Conclusiones y Trabajo Futuro **19**
7.1. Conclusiones técnicas y competenciales 19
7.2. Competencias adquiridas 20
7.3. Líneas de trabajo futuro 20

Glosario de Términos **21**

Bibliografía **24**

A. Estructuras de la Base de Datos Documental **26**

B. Manual de Usuario e Interfaz Gráfica **27**

Índice de figuras

3.1. Tablero Kanban y gestión de *Issues* en GitLab. 7

5.1. Arquitectura de microservicios de *Cerebro Digital*. 12
5.2. Pipeline de Generación Aumentada por Recuperación (Graph-RAG híbrido). 14

A.1. Inspección visual de los documentos y vectores persistidos en MongoDB. 26

B.1. Vista principal de gestión documental y panel de control lateral. 27
B.2. Representación global de la red topológica interactiva. 28
B.3. Detalle de la interacción neuronal y agrupaciones de conocimiento. 28
B.4. Interfaz del Asistente Inteligente operando el flujo RAG localmente. 29

Índice de tablas

2.1. Comparativa de herramientas PKM del estado del arte. 3
2.2. Comparativa de modelos LLM evaluados para Edge AI. 5

3.1. Planificación temporal del proyecto por sprints. 7

4.1. Endpoints de la API REST de Cerebro Digital. 10

6.1. Resultados de benchmarks sobre AMD Ryzen 7 5700U. 17

Índice de Fragmentos de Código

5.1. Inicialización resiliente de MongoDB (database.go) 13
5.2. Implementación del latido orgánico en Canvas HTML5 (GraphView.jsx) 14
6.1. Benchmarks para la evaluación de algoritmos de parsing (note_service_test.go) . . 16
6.2. Inyección de dependencias y simulación HTTP (note_handler_test.go) 17
6.3. Entorno de desarrollo determinista y aislado (shell.nix) 18

Índice de Anexos

- **Anexo A:** Estructuras de la Base de Datos Documental
- **Anexo B:** Manual de Usuario e Interfaz Gráfica

Índice de Siglas y Acrónimos

- **API:** *Application Programming Interface*
- **BFS:** *Breadth-First Search* (Búsqueda en Anchura)
- **BSON:** *Binary JSON*
- **CORS:** *Cross-Origin Resource Sharing*
- **CRUD:** *Create, Read, Update, Delete*
- **CSS:** *Cascading Style Sheets*
- **DOM:** *Document Object Model*
- **EPUB:** *Electronic Publication*
- **FPS:** *Frames Per Second*
- **GPU:** *Graphics Processing Unit*
- **HTTP:** *Hypertext Transfer Protocol*
- **I/O:** *Input/Output*
- **IA / AI:** Inteligencia Artificial / *Artificial Intelligence*
- **JSON:** *JavaScript Object Notation*
- **LOPDGDD:** Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales
- **LLM:** *Large Language Model*
- **NLP:** *Natural Language Processing*
- **PDF:** *Portable Document Format*
- **PKM:** *Personal Knowledge Management*
- **RAG:** *Retrieval-Augmented Generation*

- **RAM:** *Random Access Memory*
- **REST:** *Representational State Transfer*
- **RFC:** *Request for Comments*
- **RGPD:** Reglamento General de Protección de Datos
- **SDK:** *Software Development Kit*
- **SPA:** *Single Page Application*
- **SQL:** *Structured Query Language*
- **SVG:** *Scalable Vector Graphics*
- **TFG:** Trabajo Fin de Grado
- **UI:** *User Interface*
- **VRAM:** *Video Random Access Memory*

Capítulo 1

Introducción y Objetivos

1.1. Contexto y motivación

En la actual era de la información, los profesionales del desarrollo de software se enfrentan a un volumen de documentación técnica que crece de forma exponencial. La gestión tradicional de notas mediante estructuras jerárquicas rígidas de carpetas ha demostrado ser insuficiente para reproducir la forma en que el cerebro humano asocia ideas de manera orgánica y reticular [9].

Existen herramientas de Gestión del Conocimiento Personal (PKM) que introducen el concepto de enlaces bidireccionales, como Obsidian. Sin embargo, la mayoría carecen de capacidades analíticas avanzadas integradas de forma nativa, dependiendo en cambio de complementos de terceros que actúan como pasarelas hacia APIs de IA alojadas en la nube (OpenAI, Anthropic, etc.). Esta externalización presenta un riesgo crítico de privacidad cuando se maneja información corporativa o código fuente protegido.

Ante esta problemática, y en colaboración tecnológica con la empresa HP SCDS, surge la necesidad de un sistema que combine la ligereza del formato Markdown con la potencia del procesamiento de lenguaje natural ejecutado de forma estrictamente local (*Edge Computing*).

1.2. Objetivos del proyecto

El objetivo principal de este TFG es diseñar, desarrollar e implementar **Cerebro Digital**, una plataforma de gestión del conocimiento personal enfocada en la privacidad mediante computación en el borde de la red.

1.2.1. Objetivos técnicos

- **Arquitectura limpia y escalable:** Implementar un backend en Go usando únicamente su librería estándar, una base de datos documental (MongoDB) y una interfaz reactiva en React, siguiendo los principios de Arquitectura Limpia [6].

- **Integración de IA local:** Incorporar un flujo RAG orquestando el motor Ollama, garantizando que la vectorización y la inferencia no transfieran datos fuera del dispositivo del usuario.
- **Análisis topológico:** Desarrollar algoritmos que procesen referencias cruzadas (*Wiki-links*) para generar un grafo interactivo bidimensional impulsado por físicas de partículas con complejidad reducida mediante Barnes-Hut [2].
- **Reproducibilidad y DevOps:** Garantizar que el entorno de desarrollo sea determinista y portable en cualquier sistema UNIX/Linux mediante gestores de paquetes funcionales (Nix) y contenedores (Docker).

1.2.2. Objetivos personales y competencias

El desarrollo de *Cerebro Digital* ha constituido un reto académico integral orientado a consolidar competencias avanzadas en concurrencia con Golang, aplicación de estructuras algorítmicas clásicas (BFS, Barnes-Hut) en entornos web, y orquestación de infraestructuras reproducibles como DevOps en entornos empresariales reales.

1.3. Normativa y estándares aplicables

El diseño e implementación de *Cerebro Digital* se enmarca en el cumplimiento de los siguientes marcos normativos y estándares técnicos, cuya aplicación se detalla a lo largo de los capítulos correspondientes:

- **Reglamento (UE) 2016/679 (RGPD) y Ley Orgánica 3/2018 (LOPDGDD):** La arquitectura garantiza el cumplimiento del Reglamento General de Protección de Datos al procesar toda información del usuario de forma exclusivamente local, eliminando cualquier transferencia de datos personales a terceros [10].
- **ISO/IEC 25010:2011 — Calidad del Software (*SQuaRE*):** El diseño del sistema adopta los atributos de calidad definidos en esta norma, priorizando la eficiencia en el rendimiento (verificada mediante benchmarks), la seguridad (datos en local) y la mantenibilidad (Arquitectura Limpia).
- **CommonMark (RFC 7764):** El formato de almacenamiento de notas sigue la especificación CommonMark para garantizar la interoperabilidad y portabilidad del formato Markdown [11].
- **RFC 7231 — Semántica HTTP/1.1:** La API RESTful implementada cumple con la semántica de métodos HTTP definida en esta especificación del IETF, incluyendo los códigos de estado de respuesta.
- **RFC 8259 — El formato de intercambio de datos JSON:** Todos los mensajes de la API utilizan JSON como formato de serialización, conforme a este estándar del IETF.

Capítulo 2

Marco Teórico y Estado del Arte

La concepción arquitectónica de *Cerebro Digital* se fundamenta en la intersección de tres disciplinas de las Ciencias de la Computación: la teoría de grafos aplicada a la gestión de la información, el procesamiento de lenguaje natural (NLP) eficiente en dispositivos de consumo, y la ingeniería de software orientada a la soberanía del dato.

2.1. Estado del arte en la Gestión del Conocimiento Personal

Los sistemas de archivos han estructurado la información bajo arquitecturas jerárquicas estrictas. La PKM moderna argumenta que este paradigma es restrictivo y no refleja las redes de asociación cognitivas del cerebro humano [9].

El referente actual en el ámbito del PKM local es **Obsidian**, que popularizó el almacenamiento basado en Markdown y las bóvedas de conocimiento interconectadas mediante hiperenlaces bidireccionales. Sin embargo, el ecosistema actual presenta una limitación crítica: la integración de IA depende de complementos que actúan como pasarelas hacia APIs en la nube, rompiendo el principio fundamental de privacidad.

La Tabla 2.1 recoge una comparativa de las principales herramientas PKM del mercado frente a *Cerebro Digital*.

Tabla 2.1: Comparativa de herramientas PKM del estado del arte.

Herramienta	**IA Nativa**	**Grafo**	**Código abierto**	**Datos locales**	**Sin suscripción**
Obsidian	No (plugins)	Sí	No	Parcial	No (Pro)
Notion	Sí (nube)	No	No	No	No
Logseq	No	Sí	Sí	Sí	Sí
Roam Research	No	Sí	No	No	No
Cerebro Digital	**Sí (local)**	**Sí**	**Sí**	**Total**	**Sí**

2.2. Fundamentos matemáticos de la representación topológica

Para modelar la red de conocimiento, el sistema abstrae la información mediante la Teoría de Grafos [8]. La base documental se representa como un grafo dirigido $G = (V, E)$, donde:

- $V = \{v_1, v_2, \ldots, v_n\}$ representa el conjunto de vértices (notas Markdown).
- $E \subseteq V \times V$ es el conjunto de aristas dirigidas. Una arista $e = (v_i, v_j)$ existe si y solo si la nota v_i contiene una referencia sintáctica explícita (`[[...]]`) hacia la nota v_j.

Esta formalización permite analizar la densidad de la información mediante el grado de entrada ($\deg^-(v)$) y salida ($\deg^+(v)$) de cada nodo, identificando automáticamente conceptos nucleares o *hubs* de conocimiento.

Para la representación visual se implementa una simulación de *Force-Directed Graphs* [3], con dos fuerzas fundamentales:

1. **Fuerza de Atracción (Ley de Hooke):**

$$F_a = -k \cdot x$$

   donde k es la constante de rigidez del enlace y x el desplazamiento.

2. **Fuerza de Repulsión (Ley de Coulomb):**

$$F_r = k_e \frac{q_1 q_2}{d^2}$$

La aproximación iterativa de estas fuerzas se computa eficientemente utilizando el algoritmo de **Barnes-Hut** [2] mediante estructuras de datos *Quadtree*, reduciendo la complejidad computacional de $O(n^2)$ a $O(n \log n)$.

2.3. Generación Aumentada por Recuperación (RAG)

Los LLMs presentan una vulnerabilidad intrínseca conocida como "alucinación", generada por la desconexión temporal de su corpus de entrenamiento y la falta de contexto específico [1]. Para dotar al LLM del contexto privado del usuario sin incurrir en el elevado coste computacional del ajuste fino (*Fine-Tuning*), se implementa la arquitectura RAG [1], que requiere la transformación previa de fragmentos de texto en *embeddings*: representaciones matemáticas continuas en un espacio vectorial denso de alta dimensionalidad [4].

2.3.1. Justificación de la selección de modelos (Edge AI)

La viabilidad del proyecto imponía la ejecución de los modelos en hardware de consumo. Bajo esta restricción y por directriz del equipo de tutorización técnica, se descartaron modelos

fundacionales de gran tamaño debido a sus requerimientos de memoria VRAM, optando por modelos cuantizados [5].

La Tabla 2.2 resume los modelos evaluados durante el proceso de selección.

Tabla 2.2: Comparativa de modelos LLM evaluados para Edge AI.

Modelo	Parámetros	RAM (Q4)	Latencia RAG	Viabilidad Edge
Llama 3	8B	~5.0 GB	30–60 s	No
Mistral	7B	~4.5 GB	25–50 s	No
Qwen3	1.7B	~2.0 GB	12–20 s	Marginal
Qwen3	**0.6B**	**<1.5 GB**	**5–10 s**	**Sí**

El diseño emplea el modelo **Qwen3 (0.6b)** de Alibaba Cloud [18], que sometido a técnicas de cuantización radical (reduciendo la precisión de sus pesos a enteros de 4 u 8 bits [5]) permite una inferencia fluida en procesadores estándar utilizando menos de 1.5 GB de memoria RAM. Para la vectorización se emplea el modelo especializado **Nomic-Embed-Text** [19].

2.4. Persistencia vectorial y similitud del coseno

Las arquitecturas RAG a escala empresarial emplean Bases de Datos Vectoriales dedicadas (Pinecone, Milvus, ChromaDB). En *Cerebro Digital* se tomó la decisión fundamentada de consolidar la persistencia y la algoritmia en MongoDB y Go, por dos razones empíricas:

1. **Escala del dominio:** Un PKM personal raramente supera las decenas de miles de notas. Añadir un microservicio vectorial dedicado introduciría complejidad operativa y consumo de RAM injustificados en un entorno local.

2. **Eficiencia del cálculo algebraico en Go:** El lenguaje compilado Go permite computar la Similitud del Coseno directamente en memoria del servidor con latencias del orden de los microsegundos.

La métrica geométrica computada ignora la magnitud del vector y prioriza la superposición semántica, evaluando el ángulo θ entre el vector de la consulta (A) y el de la nota (B):

$$\text{similitud}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Capítulo 3

Metodología y Planificación

3.1. Elección de la metodología de desarrollo

La formulación estricta de requisitos iniciales resultaba inviable dada la incertidumbre en el rendimiento de los motores de IA locales. El ciclo de vida se ha regido por un enfoque híbrido que combina principios del **Desarrollo Ágil (Scrum)** con **Prototipado Evolutivo**, operando mediante iteraciones orientadas a valor y evaluación heurística de parámetros analíticos.

3.2. Herramientas de gestión y trazabilidad

Para garantizar la trazabilidad e integridad del desarrollo se implementó un flujo de trabajo apoyado íntegramente en el ecosistema **GitLab**:

- **Estructuración en Épicas e Issues:** Grandes hitos definidos como *Epics*, subdivididos en tareas atómicas (*Issues*).
- **Estrategia de ramificación GitFlow:** Características complejas aisladas en ramas antes de su consolidación en la rama principal.
- **Tablero Kanban:** Control visual de la deuda técnica (Figura 3.1).

3.3. Planificación temporal y fases de ejecución

El proyecto abarcó seis meses (noviembre de 2025 – mayo de 2026), estructurado bajo un enfoque *API-First* que prioriza la lógica de negocio antes que la capa visual. La Tabla 3.1 resume la planificación temporal por sprints.

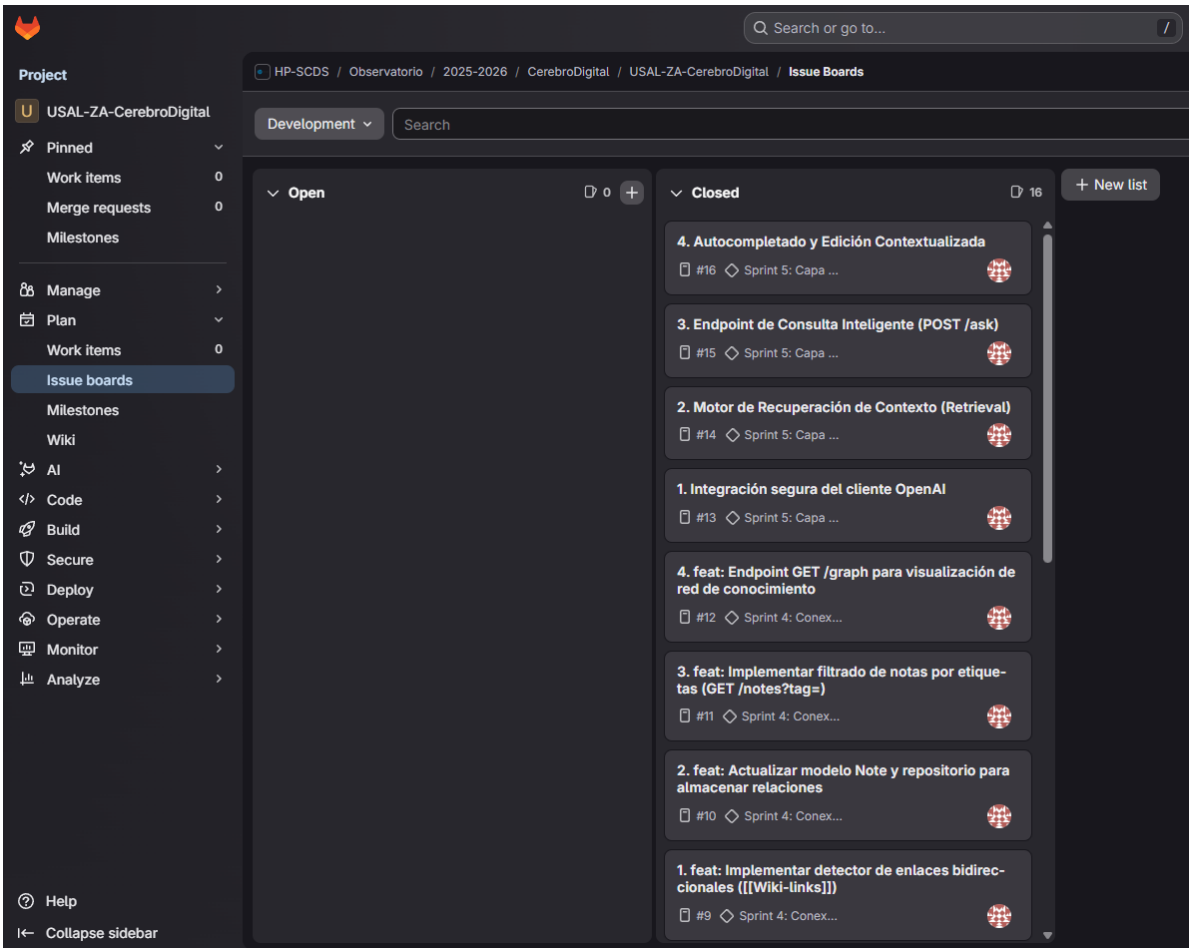


Figura 3.1: Tablero Kanban y gestión de *Issues* en GitLab.

Tabla 3.1: Planificación temporal del proyecto por sprints.

Sprint	Período	Objetivos principales	Estado
Sprint 1	Nov 2025	Entorno Nix, Docker, MongoDB, estructura Go base	Completado
Sprint 2	Nov – Dic 2025	CRUD de notas, driver MongoDB para Go, pruebas HTTP	Completado
Sprint 3	Ene – Feb 2026	Arquitectura limpia, parsing Regex de metadatos	Completado
Sprint 4	Feb – Mar 2026	Grafo topológico, Wiki-links, API de estructura	Completado
Sprint 5	Marzo 2026	Pipeline RAG, similitud del coseno, integración Ollama	Completado
Fase final	Abr – May 2026	Frontend React, BFS, Glassmorphism, QA general	Completado

3.4. Estudio de viabilidad y análisis de riesgos

3.4.1. Viabilidad técnica

El proyecto es técnicamente viable gracias a la madurez del ecosistema de código abierto. La orquestación del backend mediante **Go** garantiza una gestión de concurrencia eficiente para el

cálculo de distancias vectoriales, mientras que **React** asegura una renderización fluida del DOM. El principal riesgo técnico detectado fue el alto consumo de VRAM de los LLMs, mitigado mediante la adopción de modelos cuantizados (Qwen3 0.6b) y ejecución *Edge AI* a través de Ollama, tal como se documenta en la Tabla 2.2.

3.4.2. Viabilidad económica

La viabilidad financiera del proyecto es sólida. Al prescindir de infraestructuras en la nube y APIs de IA de pago, los costes de despliegue y mantenimiento se reducen a cero. Todas las herramientas empleadas (Nix, Docker, MongoDB Community, Go, Node.js) operan bajo licencias de software libre (MIT, Apache 2.0, GPL), haciendo la solución económicamente sostenible y distribuible a nivel empresarial.

3.4.3. Viabilidad legal y privacidad (RGPD)

La arquitectura del proyecto aborda el mayor desafío actual en la gestión del conocimiento corporativo: el cumplimiento del **RGPD** (Reglamento (UE) 2016/679) [10] y la **LOPDGDD** (Ley Orgánica 3/2018). Al operar bajo el paradigma de computación local, los datos del usuario nunca abandonan el disco duro del dispositivo. No existen transferencias de red hacia servidores de terceros ni para la vectorización ni para la generación de texto, eliminando los riesgos de exposición o brechas de seguridad asociados a plataformas centralizadas.

Capítulo 4

Herramientas y Tecnologías

Para la construcción de *Cerebro Digital* se ha orquestado un *stack* tecnológico moderno, priorizando el rendimiento nativo frente al uso de *frameworks* sobrecargados.

4.1. Frontend y capa de presentación

La interfaz de usuario es una *Single Page Application* (SPA) altamente interactiva:

- **React y Vite:** Base de la SPA. El flujo de datos se controla mediante *Hooks* nativos (`useState`, `useEffect`, `useMemo`) y la API del navegador (`localStorage`) para la persistencia efímera, reduciendo el tamaño del *bundle* final [12].
- **React Router DOM:** Interceptación de rutas en el lado del cliente y parametrización de vistas sin recarga del DOM.
- **React-Force-Graph-2D:** Motor de renderizado en *Canvas* HTML5 que traduce la estructura JSON de vértices y aristas en una simulación de partículas estable, ejecutando el motor de físicas *d3-force* [13, 14].
- **Tailwind CSS:** Implementación del paradigma visual *Cyber Minimalism* y *Glassmorphism* mediante un *framework* de estilos utilitarios, con texturas de ruido a través de filtros SVG.
- **React-Markdown:** Parseo en tiempo real del formato Markdown a HTML seguro.

La Tabla 4.1 documenta los endpoints REST que expone el backend, consumidos por la capa de presentación.

4.2. Backend y lógica de negocio

El núcleo del servidor y la API RESTful se desarrollaron en **Go (Golang)**, seleccionado por su eficiencia en la gestión de memoria y su modelo de concurrencia nativa (*Goroutines*). A partir de

Tabla 4.1: Endpoints de la API REST de Cerebro Digital.

Método	Endpoint	Descripción	Auth.
GET	`/health`	Diagnóstico del servidor	No
GET	`/notes`	Lista todas las notas del usuario	No
POST	`/notes`	Crea una nueva nota	No
GET	`/notes/{id}`	Obtiene una nota por identificador	No
PUT	`/notes/{id}`	Actualiza el contenido de una nota	No
DELETE	`/notes/{id}`	Elimina una nota	No
GET	`/graph`	Devuelve la topología del grafo	No
POST	`/ai/query`	Lanza una consulta al motor RAG	No
POST	`/ai/embed`	Vectoriza una nota manualmente	No

la versión 1.22, la librería estándar `net/http` incorpora un enrutador que soporta la declaración directa de métodos y variables de ruta (`GET /notes/{id}`), eliminando la necesidad de *frameworks* de terceros como Gin o Fiber [15]. Esto permitió implementar una estricta Arquitectura Limpia [6] que separa la comunicación web, los servicios lógicos y los repositorios de datos.

4.3. Persistencia documental

Para el almacenamiento de las entidades se seleccionó **MongoDB** [16]. Su naturaleza NoSQL documental se alinea con la estructura expansiva y semiestructurada del formato Markdown. El controlador oficial para Go (`mongo-driver`) permite mapear estructuras en BSON, almacenando eficientemente grandes matrices numéricas de precisión decimal (los *embeddings*) sin la fricción de un esquema relacional estricto.

4.4. Inteligencia Artificial y procesamiento semántico

La capa neuronal opera de forma exclusivamente local:

- **Ollama:** Motor de inferencia local responsable de la orquestación, carga en memoria y aceleración por hardware de los modelos cuantizados [17].
- **Qwen3 (0.6b):** LLM desarrollado por Alibaba Cloud [18], seleccionado por su balance entre precisión semántica y bajo consumo de recursos en hardware de consumo (véase Tabla 2.2).
- **Nomic-Embed-Text:** Modelo especializado en vectorización densa (traducción de texto a coordenadas espaciales multidimensionales) [19].
- **go-openai SDK:** Librería cliente en Go que aprovecha la compatibilidad de Ollama con la especificación de OpenAI, apuntando al puerto local (`localhost:11434/v1`).

4.5. DevOps, infraestructura y reproducibilidad

- **Nix y Direnv:** El ecosistema de desarrollo se aisló mediante **Nix**, un gestor de paquetes puramente funcional. En conjunción con **Direnv**, se diseñó un entorno virtual efímero que carga automáticamente las versiones exactas del compilador de Go, Node.js y las variables de entorno al acceder al directorio del proyecto. Esta solución erradica la divergencia de configuraciones entre estaciones de trabajo (el antipatrón "en mi máquina sí funciona").
- **Docker:** Contenerización de la instancia de MongoDB [20], garantizando entornos de almacenamiento aislados y descartables durante las fases de pruebas.

Capítulo 5

Desarrollo e Implementación

Este capítulo detalla la implementación técnica de los componentes críticos del sistema. El desarrollo no se gestó de forma aislada, sino bajo un marco de co-desarrollo tutelado por el equipo de ingeniería de **HP SCDS**. Las directrices corporativas establecidas en reuniones quincenales determinaron el *stack* tecnológico definitivo y garantizaron que el diseño cumpla con los estándares de rendimiento y mantenibilidad del sector.

5.1. Arquitectura del sistema

El sistema sigue una arquitectura de microservicios distribuida en tres capas funcionales. La Figura 5.1 ilustra la relación entre los componentes principales.

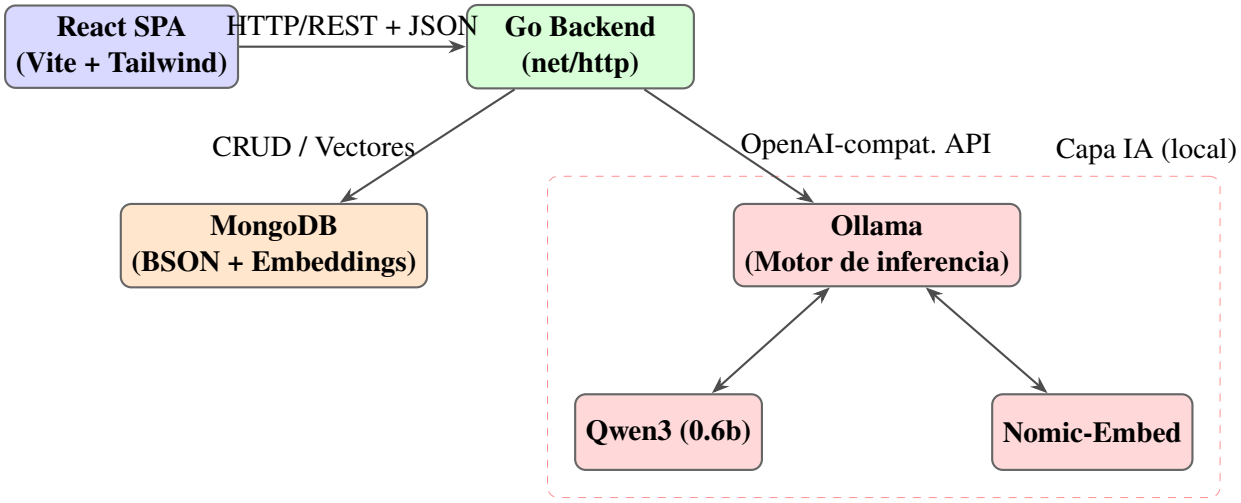


Figura 5.1: Arquitectura de microservicios de *Cerebro Digital*.

5.2. Arquitectura base e inicialización de conexiones

Se ha aplicado el principio de separación de responsabilidades (*Separation of Concerns*). Un elemento crítico del backend es la gestión resiliente de la persistencia. Como se observa en el Listado 5.1, la inicialización de MongoDB se orquesta combinando los patrones *Graceful Shutdown*, delegación por variables de entorno y prevención de bloqueos mediante contextos con límite de tiempo (`WithTimeout`).

```
func InitDB() {
  mongoURI := os.Getenv("MONGO_URI")
  if mongoURI == "" {
    log.Println("AVISO: MONGO_URI no definida. Usando credenciales por defecto
    ...")
    mongoURI = "mongodb://admin:password@localhost:27017"
  }

  clientOptions := options.Client().ApplyURI(mongoURI)

  // Contexto con timeout para evitar bloqueos (Deadlock prevention)
  ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
  defer cancel()

  var err error
  client, err = mongo.Connect(ctx, clientOptions)
  if err != nil {
    log.Fatal("ERROR: Error fatal al crear cliente Mongo:", err)
  }
}
```

Listado 5.1: Inicialización resiliente de MongoDB (database.go)

5.3. Extracción de metadatos semánticos y topológicos

El sistema procesa dinámicamente el contenido mediante algoritmos de *parsing* basados en Expresiones Regulares. Durante el guardado de una nota, la capa de servicios (*Service Layer*) inspecciona el cuerpo del texto para extraer las entidades categóricas (*hashtags*) y las adyacencias estructurales (*Wiki-links*).

Esta lógica algorítmica abstrae las referencias bidireccionales contenidas entre dobles corchetes (`[[...]]`) y las transforma en las aristas (*edges*) que conformarán la topología matemática del grafo de conocimiento.

5.4. Pipeline semántico y estrategia Graph-RAG híbrida

La Figura 5.2 ilustra el flujo completo del motor RAG implementado.

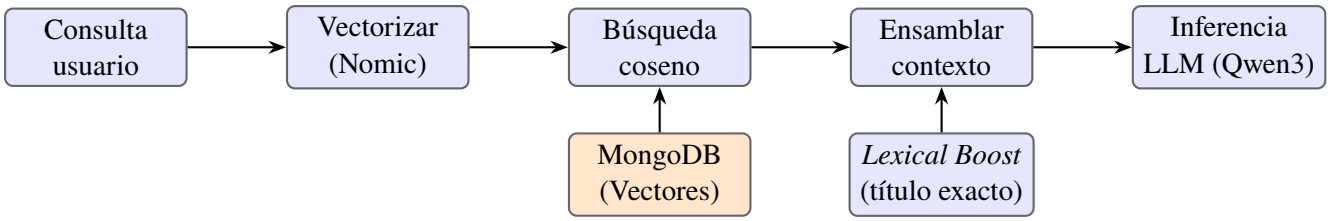


Figura 5.2: Pipeline de Generación Aumentada por Recuperación (Graph-RAG híbrido).

La integración de la IA amplía el modelo RAG estándar [1] mediante una estrategia **Graph-RAG híbrida**: si una nota recuperada contiene enlaces directos a otros documentos, dichos documentos se recuperan recursivamente e inyectan en el contexto final, proporcionando al LLM un mapa conceptual hiper-conectado.

Para evitar la abstracción excesiva en búsquedas específicas, el sistema implementa un **Impulso Léxico** (*Lexical Boost*): si la consulta coincide exactamente con el título de una nota, se inyecta un incremento ponderado al cálculo de similitud.

5.5. Renderizado topológico y Búsqueda Neural en React

La representación de la red de conocimiento delega el cálculo espacial al motor *d3-force* embebido en un lienzo *Canvas* HTML5. Para evitar bloqueos del hilo principal de React durante el cálculo continuo de distancias euclidianas, la simulación física se desacopla del ciclo de renderizado.

Para dotar a la topología de un comportamiento orgánico durante las consultas semánticas, se ha implementado un efecto visual denominado **Búsqueda Neural**: cuando el motor identifica un nodo coincidente, el componente altera dinámicamente sus propiedades físicas mediante funciones trigonométricas dependientes del tiempo, generando una propagación de luz (*bloom*) sobre el vértice (Listado 5.2).

```
ctx.beginPath();
ctx.arc(node.x, node.y, baseRadius, 0, 2 * Math.PI, false);

if (isMatch) {
  const time = Date.now() / 300;
  const breath = Math.sin(time + node.x) * 4; // Expansión radial periódica

  ctx.fillStyle = '#ffffff';
  ctx.shadowColor = nodeColor;
  ctx.shadowBlur = 60 + breath * 5; // Bloom interactivo proporcional al pulso
}
ctx.fill();
```

Listado 5.2: Implementación del latido orgánico en Canvas HTML5 (GraphView.jsx)

5.6. Interfaces optimistas (Optimistic UI) y estado efímero

Para mitigar la latencia inherente a las operaciones de red y garantizar una respuesta visual instantánea, el Frontend adopta el patrón de **Actualización Optimista**: las mutaciones de los componentes renderizan el nuevo estado inmediatamente, asumiendo que la transacción HTTP será exitosa. Si el servidor devuelve un error, el cliente ejecuta un *rollback* silente. El estado conversacional con la IA se gestiona como información efímera en la Web Storage API (`localStorage`), aliviando la carga sobre la base de datos.

Capítulo 6

Aseguramiento de Calidad y Entornos

Este capítulo documenta las estrategias aplicadas para garantizar la fiabilidad del código y la reproducibilidad del entorno de ejecución.

6.1. Testing de lógica de negocio y benchmarks de rendimiento

La validación de la lógica del dominio se ha estructurado utilizando el enfoque de pruebas guiadas por tablas (*Table-Driven Tests*). Para funciones críticas como la extracción de metadatos, se definieron matrices de estructuras que evalúan exhaustivamente casos límite (*edge cases*), utilizando la librería `reflect` para comparaciones profundas (`DeepEqual`).

Al tratarse de una aplicación intensiva en lectura, se garantizó que los algoritmos de extracción por expresiones regulares no introdujeran cuellos de botella, mediante **Pruebas de Rendimiento (Benchmarks)** con la estructura `testing.B` (Listado 6.1).

```
func BenchmarkExtractTags(b *testing.B) {
  texto := "Nota de ejemplo con varios hashtags como #tecnologia, " +
           "#programacion y #golang para medir la latencia del Regex."

  b.ResetTimer() // Excluye el tiempo de inicialización de datos
  for i := 0; i < b.N; i++ {
    extractTags(texto)
  }
}
```

Listado 6.1: Benchmarks para la evaluación de algoritmos de parsing (note_service_test.go)

La Tabla 6.1 recoge los resultados empíricos obtenidos sobre hardware de consumo (**AMD Ryzen 7 5700U**).

Los resultados confirman que el microservicio en Go es capaz de analizar más de 80.000 documentos por segundo sin bloquear el hilo principal, validando el uso de Expresiones Regulares nativas frente a motores de indexación pesados.

Tabla 6.1: Resultados de benchmarks sobre AMD Ryzen 7 5700U.

Benchmark	**Latencia media**	**Aprox. operaciones/s**	**Método**
`BenchmarkExtractTags`	11.353 ns/op	~88.000 docs/s	Regex nativo Go
`BenchmarkExtractLinks`	6.866 ns/op	~145.000 docs/s	Regex nativo Go
Inferencia RAG completa	5–10 s	—	Qwen3 0.6b + Ollama

6.2. Pruebas de integración HTTP y mocking

Las validaciones de la capa de red se instrumentaron con la librería nativa `net/http/httptest`. Para desacoplar las pruebas de los servicios externos (Ollama, MongoDB), se explota la **Inyección de Dependencias Ligera** mediante funciones de primera clase (*First-Class Functions*). Como se ilustra en el Listado 6.2, se aplica la técnica de *Mocking*, sobrescribiendo temporalmente la función `UpdateNoteFunc` con una clausura que simula una respuesta exitosa.

```
func TestUpdateNoteHandler_Success(t *testing.T) {
  oldFunc := UpdateNoteFunc
  defer func() { UpdateNoteFunc = oldFunc }()

  // Mock: evitamos llamar a la IA o a la base de datos real
  UpdateNoteFunc = func(id string, note models.Note) error {
    return nil
  }

  goodJSON := []byte(`{"title": "Actualizado", "body": "Texto nuevo"}`)
  req, _ := http.NewRequest("PUT", "/notes/12345", bytes.NewBuffer(goodJSON))
  rr := httptest.NewRecorder()

  handler := http.HandlerFunc(UpdateNoteHandler)
  handler.ServeHTTP(rr, req)

  if status := rr.Code; status != http.StatusOK {
    t.Errorf("Esperábamos status 200, obtuvimos %v", status)
  }
}
```

Listado 6.2: Inyección de dependencias y simulación HTTP (note_handler_test.go)

6.3. Infraestructura reproducible (Nix y Direnv)

El entorno de trabajo implementa **Nix** a través de un archivo de manifiesto (`shell.nix`) que declara de forma inmutable el estado exacto de las dependencias a nivel de sistema operativo (Listado 6.3). Al integrarse con **Direnv**, el sistema operativo intercepta el acceso al directorio del

proyecto y despliega instantáneamente el entorno virtual. Esta solución garantiza que el proyecto pueda ser auditado, compilado y ejecutado obteniendo un entorno binariamente idéntico.

```
let
  nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-25.11
    ";
  pkgs = import nixpkgs { config = { allowUnfree = true; }; overlays = []; };
in pkgs.mkShell {
  name = "cerebro-digital";
  packages = with pkgs; [
    go       # Compilador Go (versión fijada)
    air      # Recarga automática en desarrollo
    mongosh  # Cliente ligero para inspección de la BD
  ];
}
```

Listado 6.3: Entorno de desarrollo determinista y aislado (shell.nix)

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones técnicas y competenciales

El desarrollo de *Cerebro Digital* ha culminado en la consolidación de un sistema PKM integral, funcional y centrado en la soberanía de los datos. A continuación se exponen las principales conclusiones agrupadas conforme a los criterios de evaluación del TFG.

Respecto a los objetivos planteados: todos los objetivos técnicos definidos en la Sección 1.3 se han alcanzado satisfactoriamente. El sistema implementa una arquitectura RAG local funcional, un grafo interactivo basado en teoría de grafos, una capa de pruebas con benchmarks verificables, y un entorno de desarrollo reproducible mediante Nix. La colaboración con HP SCDS ha garantizado, además, que el diseño cumpla con los estándares de calidad exigibles en un entorno empresarial real.

Respecto a la dificultad y originalidad: el principal desafío arquitectónico fue la adopción del lenguaje Go y la transición hacia una Arquitectura Limpia. La gestión de la concurrencia mediante *goroutines* para el cálculo de distancias vectoriales, y la implementación de la estrategia Graph-RAG híbrida con Impulso Léxico, constituyen contribuciones técnicas originales que van más allá de los sistemas RAG estándar de la literatura.

Respecto al rigor técnico: los benchmarks empíricos demuestran que el microservicio en Go procesa más de 80.000 documentos por segundo en hardware de consumo (Tabla 6.1), y que el modelo Qwen3 0.6b cuantizado es viable para inferencia Edge AI con latencias de 5 a 10 segundos (Tabla 2.2). La adopción de la Similitud del Coseno como métrica de recuperación es coherente con la literatura de recuperación de información [7].

Respecto a la normativa aplicable: la arquitectura garantiza el cumplimiento del RGPD [10] y la LOPDGDD al procesar todos los datos de forma local. El diseño adopta los atributos de calidad de la norma ISO/IEC 25010:2011 y la API sigue los estándares RFC 7231 y RFC 8259, tal como se detalla en la Sección 1.3.

Limitación identificada: los modelos cuantizados de 0.6b parámetros presentan una tendencia a la imprecisión semántica más elevada que los modelos de mayor tamaño. Este fenómeno es un *trade-off* de ingeniería previsible al confinar modelos generativos a procesadores de consumo,

parcialmente mitigado mediante la calibración de la temperatura algorítmica y el Impulso Léxico.

7.2. Competencias adquiridas

El desarrollo de este TFG ha permitido consolidar competencias en concurrencia con Golang, algoritmia de grafos en entornos web, orquestación de infraestructuras reproducibles, colaboración en ciclos de revisión de código empresariales y aplicación práctica de marcos regulatorios (RGPD) en decisiones de diseño arquitectónico.

7.3. Líneas de trabajo futuro

El ciclo evolutivo del software deja abiertas las siguientes líneas de investigación:

- **Desacoplamiento de Ollama:** Refactorizar el motor de *embeddings* para ejecutar el modelo directamente en el binario de Go a través de librerías de inferencia nativas (como `llama.cpp`), reduciendo la dependencia de servicios de terceros.
- **Indexación documental expandida:** Ampliar la Búsqueda Neural para que los algoritmos de vectorización procesen documentación binaria externa (PDF, EPUB, DOCX), enriqueciendo el grafo de conocimiento privado del usuario.
- **Base de datos vectorial dedicada:** Para casos de uso empresarial con decenas de miles de documentos, evaluar la integración de un índice HNSW (*Hierarchical Navigable Small World*) mediante una base de datos vectorial embebida, sin comprometer el principio de funcionamiento local.

Glosario de Términos

El presente glosario recoge y define los conceptos técnicos específicos empleados a lo largo de esta memoria que, por su especialización, pueden requerir una explicación más detallada que la proporcionada por el Índice de Siglas y Acrónimos.

Alucinación (IA) Fenómeno por el cual un modelo de lenguaje grande genera información plausible pero factualmente incorrecta o inventada. Se produce cuando el modelo no dispone de contexto suficiente y extrapola a partir de patrones estadísticos de su corpus de entrenamiento. En *Cerebro Digital*, este problema se mitiga mediante la arquitectura RAG, que inyecta contexto verificado antes de cada inferencia.

Arquitectura Limpia (*Clean Architecture*) Patrón de diseño de software propuesto por Robert C. Martin [6] que organiza el código en capas concéntricas con dependencias unidireccionales: la capa más interna (dominio y lógica de negocio) no conoce las capas externas (infraestructura, base de datos, interfaz de usuario). En este proyecto se aplica separando los *handlers* HTTP, la capa de servicios y los repositorios MongoDB en paquetes independientes de Go.

Barnes-Hut (algoritmo) Algoritmo de aproximación jerárquica para el cálculo de fuerzas en sistemas de n partículas [2]. Agrupa partículas distantes mediante una estructura de datos *Quadtree*, sustituyendo el cálculo individual por una fuerza agregada del centro de masa del grupo. Reduce la complejidad computacional de $O(n^2)$ a $O(n \log n)$, lo que permite renderizar grafos de conocimiento con cientos de nodos a una tasa de fotogramas fluida.

***Edge Computing* / Edge AI** Paradigma de procesamiento en el que la computación se realiza en el propio dispositivo del usuario (el "borde" de la red), en contraposición al modelo centralizado en la nube. En este proyecto, tanto la vectorización de documentos como la inferencia del LLM se ejecutan íntegramente en la máquina local, lo que garantiza privacidad absoluta y elimina la latencia de red.

***Embedding* (vectorización)** Representación matemática de un fragmento de texto como un vector numérico de alta dimensionalidad (en este sistema, 768 dimensiones) en un espacio semántico continuo. Textos con significados similares producen vectores próximos en dicho espacio, lo que permite realizar búsquedas por similitud conceptual en lugar de coincidencia literal de palabras clave. El modelo Nomic-Embed-Text es el responsable de esta transformación en *Cerebro Digital*.

Force-Directed Graph Técnica de representación visual de grafos [3] que modela las relaciones entre nodos como fuerzas físicas: los nodos conectados se atraen (resortes, Ley de Hooke) y todos los nodos se repelen entre sí (cargas electrostáticas, Ley de Coulomb). La simulación itera hasta alcanzar un equilibrio de fuerzas, produciendo una disposición espacial que revela la estructura topológica del grafo de forma intuitiva y orgánica.

Glassmorphism Tendencia de diseño de interfaces de usuario caracterizada por el uso de fondos traslúcidos con efecto de cristal esmerilado (*frosted glass*), logrado mediante la propiedad CSS `backdrop-filter: blur()`. En *Cerebro Digital* se complementa con gradientes radiales matemáticos y texturas de ruido (*Noise Textures*) generadas a través de filtros SVG para conseguir un acabado visual de alta fidelidad.

Goroutine Unidad de ejecución concurrente ligera del lenguaje Go, administrada por el *runtime* del propio lenguaje en lugar de por el sistema operativo. Su coste de creación es órdenes de magnitud inferior al de un hilo del sistema operativo (~2 KB de pila inicial frente a los ~2 MB de un hilo POSIX), lo que permite lanzar miles de goroutines simultáneas. En este proyecto se utilizan para paralelizar el cálculo de la Similitud del Coseno sobre el conjunto de *embeddings* almacenados.

Graph-RAG híbrido Extensión de la arquitectura RAG estándar [1] que enriquece el contexto inyectado al LLM con información de la estructura topológica del grafo de conocimiento. Si una nota recuperada por similitud semántica contiene *Wiki-links* hacia otros documentos, dichos documentos se incorporan recursivamente al contexto. Este diseño proporciona al modelo un mapa conceptual hiper-conectado, mejorando la coherencia y precisión de las respuestas generadas.

Grafo dirigido Estructura de datos matemática $G = (V, E)$ en la que el conjunto de aristas E está compuesto por pares ordenados de vértices (v_i, v_j), es decir, cada arista tiene una dirección de origen a destino. En este proyecto, el grafo de conocimiento es dirigido porque un *Wiki-link* en la nota A hacia la nota B no implica necesariamente que B enlace de vuelta a A.

Impulso Léxico (*Lexical Boost*) Mecanismo de ajuste ponderado aplicado al cálculo de la Similitud del Coseno cuando la consulta del usuario coincide exactamente con el título de una nota almacenada. Este impulso incrementa artificialmente la puntuación de relevancia de dicha nota, priorizándola en la ventana de contexto enviada al LLM. Resuelve el problema de la abstracción excesiva en búsquedas muy específicas donde la similitud semántica pura podría no ser suficiente.

Markdown Lenguaje de marcado ligero creado por John Gruber en 2004, diseñado para ser legible en texto plano y convertible a HTML de forma sencilla. Su especificación está formalizada en el estándar CommonMark (RFC 7764) [11]. Constituye el formato de almacenamiento

central de *Cerebro Digital* por su portabilidad, interoperabilidad y la garantía de que el usuario mantiene siempre la propiedad de sus datos en un formato abierto.

Modelo cuantizado LLM al que se han reducido la precisión numérica de sus parámetros (pesos) desde el formato de punto flotante de 32 bits (`FP32`) a representaciones de menor precisión como enteros de 4 u 8 bits (`Q4`, `Q8`) [5]. La cuantización reduce drásticamente el consumo de memoria RAM y VRAM (por un factor de 4 a 8×), con una pérdida marginal y aceptable de calidad en la generación de texto. Hace viable la ejecución de LLMs en hardware de consumo sin aceleración GPU dedicada.

Soberanía del dato Principio arquitectónico y regulatorio según el cual el propietario de los datos mantiene el control total sobre dónde se almacenan y procesan, sin depender de terceros. Constituye el principio diferenciador de *Cerebro Digital* y la base de su cumplimiento con el RGPD [10], al garantizar que ningún dato del usuario abandona el dispositivo local durante ninguna fase del procesamiento.

Wiki-link Sintaxis de referencia cruzada entre documentos, popularizada por los sistemas tipo wiki, que consiste en encerrar el título de un documento destino entre dobles corchetes: `[[Nombre del documento]]`. En *Cerebro Digital*, el motor de *parsing* detecta esta sintaxis mediante expresiones regulares y transforma cada referencia en una arista del grafo de conocimiento, generando así la topología de la red de forma automática sin intervención del usuario.

Bibliografía

[1] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020.

[2] J. Barnes y P. Hut, "A hierarchical $O(N \log N)$ force-calculation algorithm," *Nature*, vol. 324, n.º 6096, pp. 446–449, 1986.

[3] T. M. J. Fruchterman y E. M. Reingold, "Graph Drawing by Force-directed Placement," *Software—Practice and Experience*, vol. 21, n.º 11, pp. 1129–1164, 1991.

[4] A. Vaswani *et al.*, "Attention Is All You Need," en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[5] T. Dettmers, A. Pagnoni, A. Holtzman y L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[6] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[7] C. D. Manning, P. Raghavan y H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[8] R. Diestel, *Graph Theory*, 5.ª ed. Springer, 2017.

[9] T. Vander Wal, "Folksonomy," *Off the Top Blog*, feb. 2007. [En línea]. Disponible en: http://vanderwal.net/folksonomy.html. [Último acceso: 29-05-2026].

[10] Parlamento Europeo y Consejo de la Unión Europea, "Reglamento (UE) 2016/679 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales (RGPD)," *Diario Oficial de la Unión Europea*, L 119, pp. 1–88, may. 2016.

[11] J. MacFarlane *et al.*, "CommonMark Spec," versión 0.31.2, 2024. [En línea]. Disponible en: https://spec.commonmark.org/. [Último acceso: 29-05-2026].

[12] Meta Open Source, "React: The library for web and native user interfaces," 2024. [En línea]. Disponible en: https://react.dev/. [Último acceso: 29-05-2026].

[13] V. Asturiano, "react-force-graph," repositorio GitHub, 2024. [En línea]. Disponible en: https://github.com/vasturiano/react-force-graph. [Último acceso: 29-05-2026].

[14] M. Bostock, "d3-force: Force-directed graph layout using velocity Verlet integration," repositorio GitHub, 2024. [En línea]. Disponible en: https://github.com/d3/d3-force. [Último acceso: 29-05-2026].

[15] The Go Authors, "Go Documentation," 2024. [En línea]. Disponible en: https://go.dev/doc/. [Último acceso: 29-05-2026].

[16] MongoDB Inc., "MongoDB Documentation," 2024. [En línea]. Disponible en: https://www.mongodb.com/docs/. [Último acceso: 29-05-2026].

[17] Ollama, "Ollama Official Documentation & GitHub Repository," 2024. [En línea]. Disponible en: https://github.com/ollama/ollama. [Último acceso: 29-05-2026].

[18] Alibaba Cloud, "Qwen — Official Documentation & Models," 2024. [En línea]. Disponible en: https://qwenlm.github.io/. [Último acceso: 29-05-2026].

[19] Nomic AI, "Nomic Embed: Open Source Text Embeddings," repositorio GitHub, 2024. [En línea]. Disponible en: https://github.com/nomic-ai/nomic. [Último acceso: 29-05-2026].

[20] Docker Inc., "Docker Documentation," 2024. [En línea]. Disponible en: https://docs.docker.com/. [Último acceso: 29-05-2026].

Apéndice A

Estructuras de la Base de Datos Documental

A continuación se exponen las colecciones en formato BSON almacenadas en el clúster local de MongoDB. Como se observa en la Figura A.1, se evidencia la persistencia de los vectores matemáticos (*embeddings*) y los metadatos topológicos extraídos.

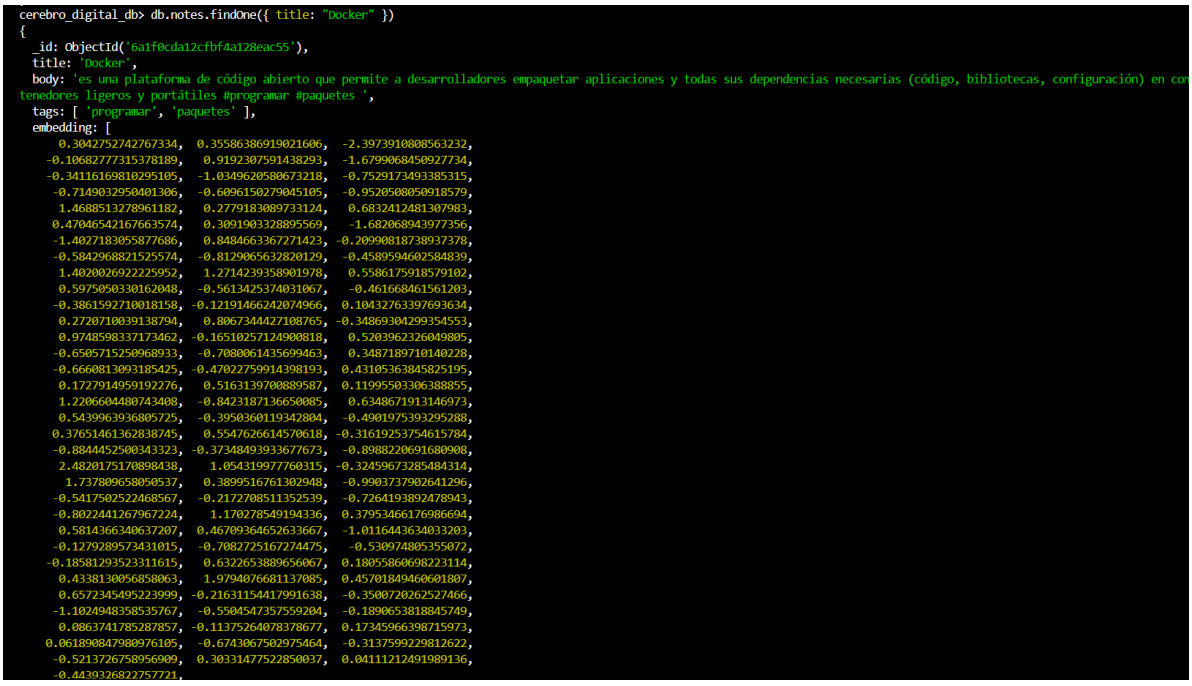


Figura A.1: Inspección visual de los documentos y vectores persistidos en MongoDB.

Apéndice B

Manual de Usuario e Interfaz Gráfica

El producto final es una *Single Page Application* (SPA) que materializa los requerimientos de interacción del sistema. El diseño sigue principios de *Cyber Minimalism* y *Glassmorphism*, con una paleta de colores de bajo contraste para reducir la fatiga visual durante la gestión intensiva del conocimiento.

La Figura B.1 muestra la vista principal de la base de datos documental. En este panel se orquesta el sistema CRUD, permitiendo al usuario redactar en Markdown mientras el servicio en Go extrae las referencias semánticas y las entidades categóricas.

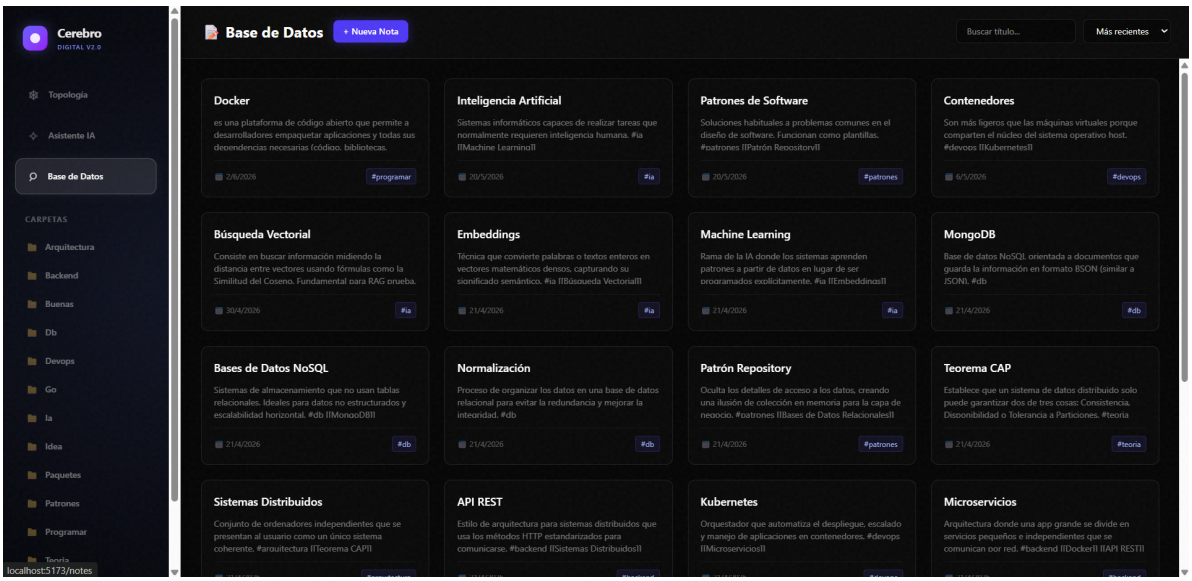


Figura B.1: Vista principal de gestión documental y panel de control lateral.

La Figura B.2 ilustra la renderización global del grafo de conocimiento, donde el motor Barnes-Hut computa las fuerzas de repulsión en tiempo real.

En la Figura B.3, la interacción directa con un nodo activa la *Búsqueda Neural*, alterando las propiedades físicas del nodo y agrupando visualmente los *clusters* de conocimiento vinculados.

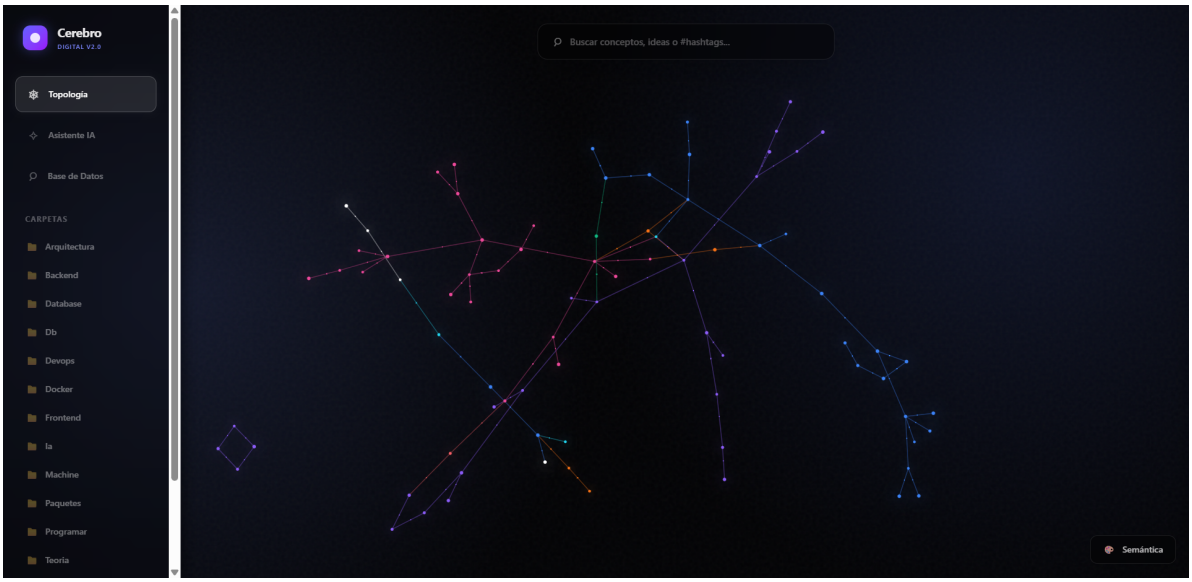


Figura B.2: Representación global de la red topológica interactiva.

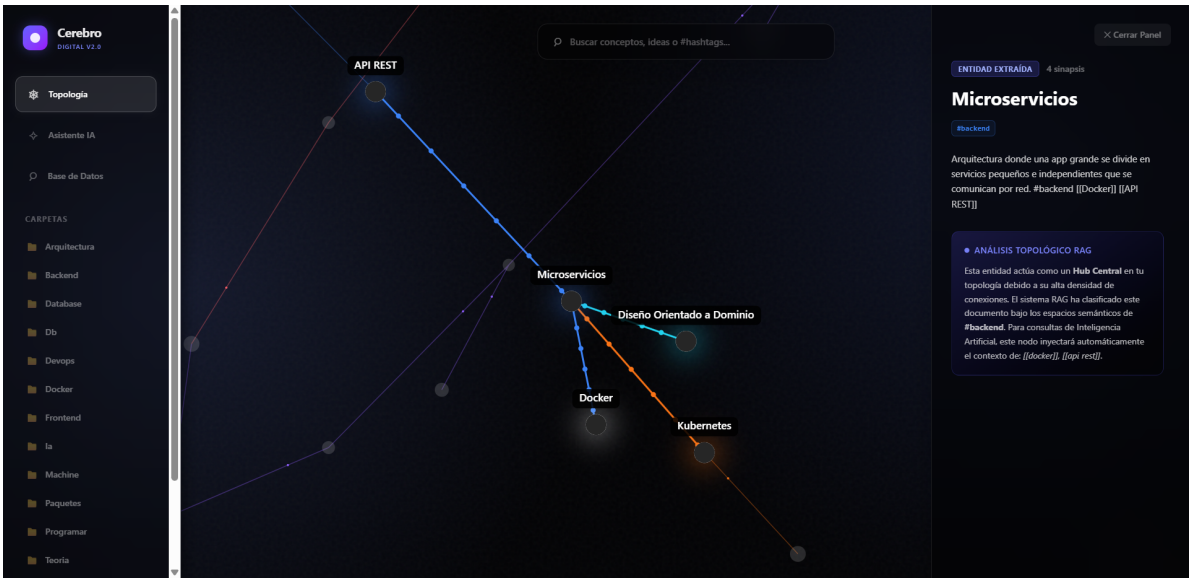


Figura B.3: Detalle de la interacción neuronal y agrupaciones de conocimiento.

La Figura B.4 detalla la vista del Asistente Inteligente. La interfaz consolida el flujo Graph-RAG híbrido: el sistema recupera matemáticamente el contexto local relevante y lo inyecta en la ventana de *tokens* del modelo LLM cuantizado, generando respuestas precisas y bien formateadas.

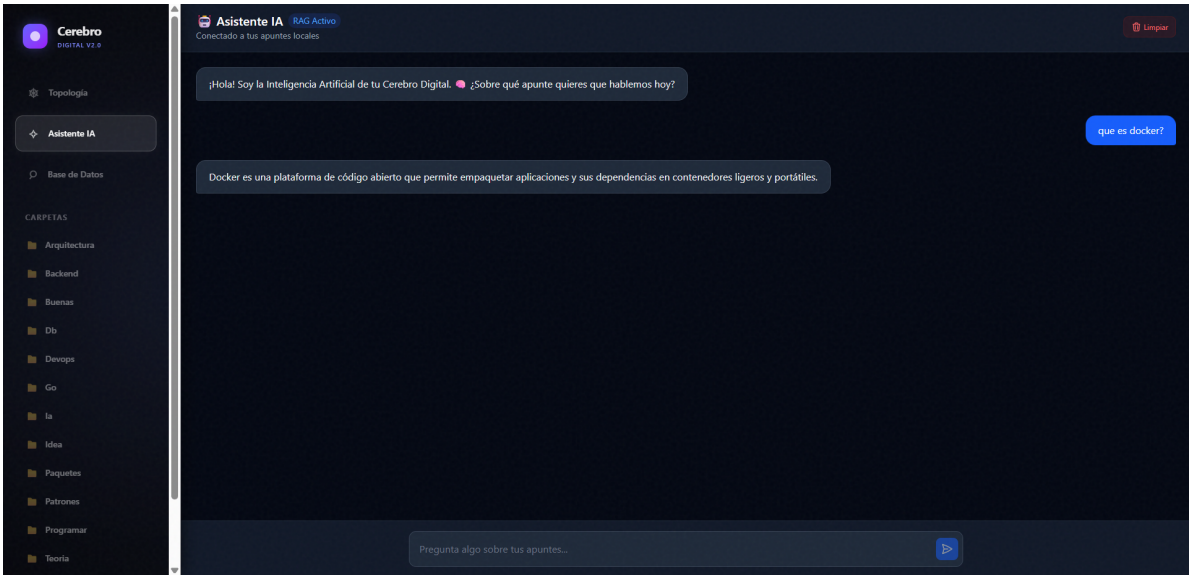


Figura B.4: Interfaz del Asistente Inteligente operando el flujo RAG localmente.