



RIDUNAJ
Repositorio Institucional
Digital UNAJ



Universidad Nacional
ARTURO JAURETCHE

Práctica Profesional Supervisada

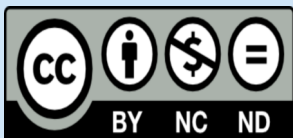
Serapio, Alan Nicolas

Sistema multiagente para lectura y comprensión de documentos con inteligencia artificial

Instituto de Ingeniería y Agronomía

2025

Carrera: Ingeniería en Informática



Esta obra está bajo una Licencia Creative Commons.
Atribución – No comercial – Sin obra derivada 4.0
<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Documento descargado de RID - UNAJ Repositorio Institucional Digital de la Universidad Nacional Arturo Jauretche

Cita recomendada:

Serapio, A. N. (2025). *Sistema multiagente para lectura y comprensión de documentos con inteligencia artificial* [Práctica Profesional Supervisada, Universidad Nacional Arturo Jauretche].
<https://rid.unaj.edu.ar/handle/123456789/3617>

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

*Sistema multiagente para lectura y comprensión de
documentos con inteligencia artificial*

Serapio, Alan Nicolás

Florencio Varela, diciembre, 2025

Estudiante

Nombre y Apellido: Alan Nicolas Serapio

Correo electrónico: serapio_alan@hotmail.com

Organización donde se realiza la Práctica Profesional Supervisada

Nombre de la Empresa- Entidad- Institución: Universidad Nacional Arturo Jauretche

Dirección Av. Calchaquí 6200, Florencio Varela, (1888) Buenos Aires, Argentina

Teléfono +54 11 4275-6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en
aplicaciones de interés social, Instituto de Ingeniería y Agronomía

Tutor organizacional

Nombre y apellido: Prof. Ing Lucas Maximiliano Olivera

Correo electrónico: olivera@unaj.edu.ar

Docente Supervisor

Nombre y apellido: Prof. Dr Marcelo Angel Cappelletti

Correo electrónico: mcappelletti@unaj.edu.ar

Coordinador de la Carrera de Ingeniería en Informática

Nombre y apellido: *Dr. Ing. Morales, Martín*

Correo electrónico: martin.morales@unaj.edu.ar

Resumen

La acelerada producción de artículos científicos plantea un desafío significativo para la comunidad, limitando la capacidad de asimilación de nueva información. Por este motivo, se desarrolla un **Sistema Neuro-Analítico Multiagente** con el propósito de complementar y acelerar el **análisis automatizado de documentos científicos**.

El enfoque metodológico principal se basa en una arquitectura distribuida de microservicios que implementa la **Recuperación Aumentada (RAG)** y utiliza **Modelos de Lenguaje Grande (LLMs)**. El sistema se organiza en tres planos funcionales: el Plano de Orquestación Conversacional, el Plano de Procesamiento Documental y el Plano de Persistencia y Estado.

Los resultados clave incluyen un sistema capaz de realizar la carga y procesamiento completo de documentos, indexación semántica, recuperación vectorial (RAG), generación de resúmenes contextuales e interacción conversacional fluida. Además, se incorpora la funcionalidad de Reconocimiento Óptico de Caracteres (OCR) tradicional y multimodal.

Como conclusión, se ha logrado desarrollar un sistema robusto que facilita la organización, generación de relaciones semánticas y síntesis de literatura científica mediante una arquitectura modular y escalable. Esto permite la consulta contextualizada y el análisis inteligente de *papers*, proveyendo una herramienta eficiente para la asimilación de conocimiento.

Abstract

The rapid production of scientific articles poses a significant challenge to the community, limiting the capacity to assimilate new information. For this reason, a Multi-Agent Neuro-Analytical System is being developed to complement and accelerate the automated analysis of scientific documents.

The main methodological approach is based on a distributed microservices architecture that implements Augmented Retrieval (RAG) and utilizes Large Language Models (LLMs). The system is organized into three functional planes: the Conversational Orchestration Plane, the Document Processing Plane, and the Persistence and State Plane.

Key results include a system capable of performing complete document loading and processing, semantic indexing, vector retrieval (RAG), contextual summary generation, and fluid conversational interaction. Furthermore, traditional and multimodal Optical Character Recognition (OCR) functionality is incorporated. In conclusion, a robust system has been developed that facilitates the organization, generation of semantic relationships, and synthesis of scientific literature through a modular and scalable architecture. This enables contextualized search and intelligent analysis of papers, providing an efficient tool for knowledge assimilation.

Dedicatorias y agradecimientos

En primer lugar, quisiera darle un reconocimiento especial a aquellos profesores y compañeros que me ayudaron en mi trayecto, sin ustedes el camino hubiese sido mucho más arduo, gracias A mi familia, a mi hermana y a mi padre, cuyo respaldo fue invaluable a la hora de poder terminar con mis estudios, Muchas Gracias, y espero algún día poder devolver, aunque sea un poco de lo mucho que me ayudaron .

Finalmente ,a mi madre, cuya insistencia en el bienestar de mi futuro y fe en mí, hizo posible que llegara hasta este punto. Solo puedo decir, te quiero ma, gracias por creer en mí, espero que sea que donde estes, te encuentres bien.

Contenido

Contenido

1.	Introducción.....	10
	Objetivos.....	10
2.	Glosario.....	12
3.	Arquitectura y tecnologías utilizadas.....	20
	Lenguajes de programación.....	20
	Editor de Código.....	20
	Frameworks.....	21
	Base de datos.....	23
	Modelos de Lenguaje grande.....	24
	Modelos de lenguaje grande (LLM) y su arquitectura base.....	25
4.	Marco teórico.....	29
	Paradigmas del diseño de software.....	29
	La Representación vectorial del lenguaje y espacio semántico.....	30
	Sistema RAG (Retrieval-Augmented Generation).....	32
	Orquestación de Agentes y Grafos de Tareas.....	33
	Grafos determinísticos vs Grafos no determinísticos.....	35
	El módulo de upload como routing estatico.....	35
	El módulo conversacional como grafo dinámico guiado por modelos.....	36
	Contraste conceptual desde la teoría de grafos en LangGraph.....	38
5.	Visión General del sistema.....	40
	Funcionalidad General del Sistema.....	41
	Microservicios.....	42
	Orchestration Pattern.....	42
	Event-Driven Reasoning (LLM-driven).....	43
	Pipe-and-Filter en el procesamiento documental.....	43
	Gateway Pattern (ms-db).....	44
	Adapter Pattern (tool registry).....	44
	Client–Orchestrator–Service Architecture.....	45
	Rol del LLM en la Arquitectura.....	45
6.	Metodologías.....	47
	Microservicio de base de datos.....	47
	Arquitectura interna del microservicio.....	47
	Model Layer (Capa de Modelos).....	48

Repository Layer (Capa de Repositorios).....	49
Service Layer (Capa de Servicios).....	49
Router Layer (Capa de Ruteo)	50
Microservicio de procesamiento y clasificación.....	50
Componentes del sistema.....	50
Microservicio de backend.....	60
Arquitectura interna	60
Módulo de orquestación del chat	61
Estructura interna del módulo y sus responsabilidades	62
Módulo de orchestracion del upload.....	67
Microservicio de front end.....	69
Flujo interno.....	70
7. Resultados.....	72
Flujo de trabajo - Ciclo Operativo Global	72
8. Conclusiones.....	93
9. Bibliografía.....	95
10. Recursos web	96

Ilustración 1 - Ecosistema LangChain	22
Ilustración 2 - Arquitectura Transformer	26
Ilustración 3 - La regla de la dependencia de arquitectura limpia	29
Ilustración 4 - Sistema Rag	33
Ilustración 5 - Modelo de Langgraph.....	34
Ilustración 6 - Flujo de nodos	35
Ilustración 7 - Grafo Routing estatico.....	36
Ilustración 8 - Grafo Routing Dinamico	38
Ilustración 9 - Diagrama C1 del sistema.....	40
Ilustración 10 - Adapter: tool registry.....	45
Ilustración 11 - Diagrama C2 del sistema.....	46
Ilustración 12 - Diagrama C3 del microservicio de BD	48
Ilustración 13 - Representacion de nodos en Neo4j.....	49
Ilustración 14 - ejemplo de router para un usuario	50
Ilustración 15 - Agente de embedding	52
Ilustración 16 - Agente de extraccion	54
Ilustración 17 - Ejemplo de la devolución de datos del agente Retriever.....	56
Ilustración 18 – adaptador para la base de datos.....	62
Ilustración 19 – Registro centralizado	63
Ilustración 20 - Ejemplo de herramienta - tool_retriever.....	64
Ilustración 21 - Ejemplo de prompts usados en el sistema	65
Ilustración 22 - Vista principal de la interfaz web	74
Ilustración 23 - vista del icono para cargar los documentos.....	74
Ilustración 24 - carga de documentos en la interfaz web.....	75
Ilustración 25 - Implementación de la carga de documentos.....	75
Ilustración 26 - validación de agente de extracción en el microservicio	76
Ilustración 27 - Backlog del agente de extracción en el microservicio	76
Ilustración 28 - validación de agente de extracción de imágenes	77
Ilustración 29 - backlog del agente de imagenes	77
Ilustración 30 – Extracción de imágenes en Postman.....	78
Ilustración 31 - inicio de lotes de las imágenes	78
Ilustración 32 - Terminación correcta del agente de extraccion de imagenes	79
Ilustración 33 - ejemplo de ejecución de un servicio de Fastapi	81
Ilustración 34 - ejemplo de inicio de next.js	81
Ilustración 35 - conversación iniciada	82
Ilustración 36 – Estado inicial – respuesta de langgraph.....	82
Ilustración 37 - Conversacion persistente	83
Ilustración 38 - modelo de uso de la herramienta de retriever.....	83
Ilustración 39 - razonamiento del modelo	84
Ilustración 40 - estados del grafo al traer la informacion	84
Ilustración 41 – obtención de información de documentos	85
Ilustración 42 - Recuperacion de texto	85
Ilustración 43 - Inicio de la recuperación mediante LangGraph.....	86
Ilustración 44 - Eleccion de la herramienta	86
Ilustración 45 - uso del agente Retriever	87
Ilustración 46 - busqueda mediante Vectorizacion.....	87
Ilustración 47 - Resultado de la herramienta de recuperacion.....	88

Ilustración 48 - Búsqueda con múltiples herramientas	88
Ilustración 49 – búsqueda de imagen relevante en base 64	89
Ilustración 50 - Recuperación de las imágenes por página.....	90
Ilustración 51 - imagen en base64	90
Ilustración 52 - información ocr multimodal de la imagen recuperada	91
Ilustración 53 - uso de los agentes ocr -summarize y ocr-fetch.....	91
Ilustración 54- Ejemplo conversación entre usuario y asistente	92

1.Introducción

El siguiente informe tiene como finalidad, presentar, desarrollar y exponer las distintas formas que tiene un sistema de múltiples agentes para poder mediante modelos y agentes un análisis automatizado de documentos científicos.

En la actualidad, la información abunda y la población científica aumenta exponencialmente, lo cual hace que la divulgación científica se mueva a un ritmo acelerado, esto hace que cada vez aparezcan nuevos y más complejos artículos científicos, si bien, esto no es inherentemente malo, supone un problema, ya que el tiempo no es suficiente para poder prestarle a cada artículo la atención adecuada, por lo tanto, este sistema viene a ayudar a resolver esta cuestión, si bien este sistema no tiene como finalidad reemplazar a la lectura de los textos, su función es más bien complementar la lectura pudiendo utilizar la inteligencia artificial para acelerar el ritmo en el cual se pueda aprender la información de los textos, así como también poder buscar desde varios artículos información relevantes sobre un tema y poder hacer un resumen o mapa cognitivo del mismo

Objetivos

Desarrollar un Sistema Neuro-Analítico Multiagente que, utilizando una arquitectura de Recuperación Aumentada (RAG) y modelos de Lenguaje Grande (LLMs), permita la consulta contextualizada y la síntesis automatizada de información contenida en documentos científicos (papers)

- Implementación de segmentación en los documentos
- Creación de vectorización en los segmentos de los documentos para la recuperación de segmentos de texto
- Diseñar el módulo de RAG
- Crear entornos de consultas
- Hacer resúmenes de la información recuperada
- Diseño de extracción de imágenes para futuras consultas

- Hacer OCR(Optical Character Recognition) tradicional ,es decir, reconocer caracteres y palabras para producir una transcripción de texto, así como OCR multimodal que no solo "lee" los caracteres, sino que comprende la disposición espacial mediante un modelo de LLM y permite devolver una respuesta al usuario

La aplicación es una arquitectura distribuida donde agentes inteligentes, equipados para capacidades avanzadas de procesamiento de lenguaje natural, colaboran en distintas tareas, como pueden ser: la extracción de datos, la recuperación de información , los resúmenes automatizados, entre otras rutinas.

El sistema busca facilitar la organización y generación de relaciones semánticas, así como poder generar síntesis de literatura científica en dominios específicos.

2. Glosario

	Término	Definición
A		
	<ul style="list-style-type: none">● APIs (Interfaces de Programación de Aplicaciones)	Conjunto definido de reglas, métodos, especificaciones y modelos de comunicación que permiten a diferentes componentes de software interactuar entre sí. Una API expone funcionalidades o datos de un sistema para que otros programas puedan utilizarlos de forma controlada y estandarizada.
	<ul style="list-style-type: none">● Auto Recursivo	Propiedad de una función, procedimiento o modelo que se invoca a sí mismo como parte de su propio proceso de ejecución. La estructura auto recursiva genera una cadena de llamadas que se resuelve cuando se alcanza una condición base claramente definida.
	<ul style="list-style-type: none">● API REST	Es una API que utiliza protocolos web (principalmente HTTP) y sigue las directrices de REST. Se caracteriza por: <ul style="list-style-type: none">○ Orientación a Recursos: Trata cada pieza de datos o funcionalidad como un recurso (ej. /usuarios, /productos/123).○ Uso de Verbos HTTP: Utiliza los métodos HTTP estándar para realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar) sobre esos recursos.<ul style="list-style-type: none">GET: Leer un recurso.POST: Crear un nuevo recurso.PUT/PATCH: Actualizar un recurso.DELETE: Eliminar un recurso.

B

- **Back end** Es la parte de un sistema o aplicación que funciona en segundo plano, invisible para el usuario, y se encarga de la lógica, el procesamiento de datos y la infraestructura del servidor
- **Bag-of-Words** Representación vectorial de texto que convierte un documento en una colección no ordenada de términos, donde la estructura sintáctica y el orden de las palabras se ignoran. Cada palabra se representa por su frecuencia o presencia dentro del corpus.

C

- **Comunicación Sin Estado, (Stateless Communication)** Cada solicitud HTTP se trata de forma independiente, sin depender del historial de solicitudes previas.
 - **No hay Memoria de Sesión en el Servidor:** El servidor no guarda ni almacena ninguna información (o "estado") sobre la sesión o las peticiones previas de ese cliente entre una solicitud y la siguiente.
 - **Cada Solicitud es Completa:** El cliente debe incluir toda la información necesaria (autenticación, datos, contexto) en cada solicitud para que el servidor pueda procesarla correctamente.
- **Compatibilidad Cross Platform** Capacidad de un software para ejecutarse de forma consistente en múltiples sistemas operativos, arquitecturas o dispositivos sin requerir cambios significativos en su código ni en su comportamiento.
- **Comportamiento Reactivo** Modelo de interacción donde el sistema responde de manera continua y no bloqueante a eventos externos, cambios de estado o flujos de datos. Implica

propagación automática de actualizaciones, asincronía y manejo eficiente de eventos.

- **CRUD** Acrónimo de las cuatro operaciones básicas de la gestión de datos, Crear, Leer, Actualizar y Eliminar (Create, Read, Update, Delete en inglés)

D

- **Debugger Integrado** Herramienta embebida dentro de un entorno de desarrollo que permite inspeccionar la ejecución del código paso a paso, visualizar estados internos, establecer puntos de interrupción y detectar errores de lógica o comportamiento.

- **Deep Learning** Subcampo del aprendizaje automático basado en arquitecturas de redes neuronales profundas que utilizan múltiples capas de procesamiento para aprender representaciones jerárquicas de los datos. Permite modelar relaciones complejas en imágenes, texto, audio y otras modalidades.

E

- **Escalabilidad Horizontal** Estrategia de crecimiento que consiste en incrementar la capacidad del sistema añadiendo múltiples instancias o nodos idénticos para distribuir la carga. El sistema mantiene funcionalidad uniforme y se adapta al aumento de tráfico mediante replicación.

- **Estado Conversacional (State Machine)** Mecanismo basado en una máquina de estados finitos donde cada paso de una conversación corresponde a un estado bien definido. Las transiciones dependen del input del usuario, el contexto previo y las reglas que rigen el flujo conversacional.

- **Endpoints** Los endpoints en microservicios son puntos de acceso (URLs) específicos que cada servicio independiente expone para que otros servicios o clientes puedan interactuar con su funcionalidad y datos, comunicándose típicamente vía HTTP (REST), exponiendo lógica de negocio, estado y metadatos, y permitiendo que la arquitectura sea escalable y mantenible, funcionando como puertas de entrada para solicitudes y respuestas.

F

- **Fácil de Escalar en el Tiempo** Propiedad arquitectónica y de diseño que permite extender, actualizar o aumentar la capacidad del sistema sin reescrituras profundas, manteniendo coherencia con el crecimiento sostenido de usuarios, datos o funcionalidades.

- **Framework** Infraestructura de software que provee estructuras, convenciones, bibliotecas y patrones predefinidos para acelerar el desarrollo y estandarizar la lógica interna de una aplicación. Define un flujo controlado donde el desarrollador completa las partes específicas.

H

- **Hooks** Mecanismo que permite interceptar, extender o personalizar el comportamiento de un sistema en puntos clave de su ejecución. Un hook expone eventos o ciclos internos donde se pueden agregar funciones adicionales sin modificar el código base.

I

- **IntelliSense** Sistema de autocompletado inteligente que ofrece sugerencias contextuales, documentación integrada, inferencia de tipos y navegación asistida del código, facilitando el desarrollo y reduciendo errores.

- **Interfaz Funcional** Tipo de interfaz o contrato que define exactamente una única función abstracta. Es fundamental en paradigmas que utilizan expresiones lambda o programación funcional dentro de lenguajes orientados a objetos.

L

- **Lenguaje de Alto Nivel Tipado Orientado a Objetos** Lenguaje de programación que abstrae detalles de bajo nivel, utiliza un sistema de tipos estático o dinámico y organiza su estructura alrededor del paradigma orientado a objetos. Facilita modelos complejos mediante clases, objetos, herencia y polimorfismo.
- **LLM (Modelo de Lenguaje Grande)** Modelo estadístico entrenado con enormes cantidades de texto para aprender patrones lingüísticos complejos y generar, clasificar, transformar o razonar sobre lenguaje natural. Utiliza arquitecturas profundas como transformers y opera mediante predicción contextual.

M

- **Marketplace** Plataforma que permite publicar, distribuir, descubrir e integrar aplicaciones, modelos, componentes o servicios desarrollados por terceros. Establece un ecosistema modular y escalable basado en intercambio de recursos.
- **Machine learning** El Machine Learning (ML), o Aprendizaje Automático, es una rama de la Inteligencia Artificial que permite a los sistemas aprender de los datos, identificar patrones y tomar decisiones o hacer predicciones con mínima intervención humana, en lugar de ser programados explícitamente para cada tarea.

- **Microservicios:** También llamados arquitectura de microservicios, es un método arquitectónico nativo de la nube en el que una sola aplicación se compone de muchos componentes o servicios más pequeños acoplados de forma flexible e implementables de forma independiente.

N

- **No structured query language (NOSQL)** Es un enfoque de administración de bases de datos que permite diseños con esquemas más flexibles comparados a los de Sql, que permiten escalar en aplicaciones modernas

P

- **Patrón Decorador** Patrón estructural que permite extender funcionalidades de un objeto sin modificar su código original. Envuelve el comportamiento mediante capas adicionales que agregan capacidades de forma dinámica y transparente.
- **Patrón Gateway** Es un diseño arquitectónico que actúa como un único punto de entrada para todas las solicitudes de clientes (móviles, web) hacia un sistema de microservicios, enrutando las peticiones a los servicios de backend adecuados, centralizando funciones como la autenticación, el registro, el balanceo de carga y la agregación de respuestas, simplificando la comunicación y desacoplando al cliente de la complejidad de múltiples servicios
- **Patrón Service Locator** Patrón de arquitectura que centraliza la gestión y provisión de dependencias en un único componente llamado localizador de servicios, el cual entrega instancias a los módulos que las solicitan en tiempo de ejecución.

- **Pydantic** Biblioteca de validación y modelado de datos basada en Python que utiliza tipado fuerte, parseo automático y verificación estructural para garantizar la consistencia de los datos utilizados en APIs y sistemas de backend.

R

- **ReDoc** Herramienta de código abierto para la generación de documentación de APIs. Toma un archivo de y genera una documentación estática de alta calidad y orientada a la lectura. Se caracteriza por su diseño elegante de una sola página y tres paneles, optimizado para la experiencia del lector.

- **Representational State Transfer (REST)** Es un conjunto de seis restricciones arquitectónicas (como Cliente-Servidor, Sin Estado, Cacheable, Interfaz Uniforme, etc.) ideadas por Roy Fielding para guiar la creación de sistemas distribuidos, siendo la World Wide Web su ejemplo más prominente.

S

- **Seamless Git Integrado** Integración fluida y continua de herramientas Git dentro de un entorno de desarrollo, permitiendo realizar commits, branches, merges, revisiones y operaciones de control de versiones sin abandonar la interfaz principal
- **Single page application (SPA)** Es una implementación de aplicación web, que carga un único documentos web, y luego actualiza el contenido del cuerpo cuando se debe mostrar contenido diferente
- **structured query language (Sql)** Es un lenguaje de programación diseñado para actualizar, obtener, y calcular información en bases de datos relacionales, mediante esquemas rígidos y tablas estructuradas.
- **Streaming de datos** La transferencia y procesamiento continuo de datos en tiempo real a medida que se generan, a diferencia del

procesamiento tradicional por lotes que los agrupa y procesa en intervalos.

- **Swagger UI**

Interfaz de usuario interactiva y de código abierto que visualiza y facilita la interacción con recursos de una API definida. Su característica principal es que permite a los desarrolladores probar en tiempo real las distintas rutas (*endpoints*) de la API directamente desde el navegador.

T

- **Tiempo Real**

Capacidad de un sistema para procesar datos y producir respuestas con latencias muy bajas, garantizando interacción inmediata o casi inmediata. Se utiliza en comunicación instantánea, monitoreo, streaming y sistemas sensibles al tiempo.

3.Arquitectura y tecnologías utilizadas

Lenguajes de programación

En este proyecto se usó como lenguaje base Python para la realización, Python es un sistema creado a finales de los años 80, pero fue el 20 de febrero de 1991, por el desarrollador informático holandés Guido van Rossum⁽¹⁾ que vuelve pública y accesible a todas las personas la primera versión de Python (Python 0.9.0).

La razón concluyente por la cual integrar este lenguaje al sistema es debido a sus inherentes beneficios que ofrece, un lenguaje de alto nivel tipado orientado a objetos con bibliotecas públicas que están continuamente en desarrollo por una comunidad atenta y en crecimiento, siendo esta una de las más grandes actualmente para 2025⁽²⁾.

Editor de Código

El proyecto se basó en la naturaleza flexible de un programa que pueda ser usado en cualquier ordenador, por lo tanto, se debía poder tener una computadora que no requiera de muchos recursos, esto llevó a que, en vez de usar un entorno de desarrollo, se usará un editor de código ligero que pudiese configurarse de manera que pudiera personalizarse para mejorar la eficiencia del sistema adaptándose a las necesidades del usuario.

Siguiendo esta perspectiva de trabajo, se encuentra en el editor de código visual studio code una gran opción disponible, desarrollada a principios de 2011, cuando se unió a Microsoft el desarrollador Eric gamma, quien inició el desarrollo de un editor web ligero llamado Monaco, que sería la piedra angular por la cual luego se sentaría las bases para lo que más adelante se convertiría en visual studio code.

Anunciado en abril de 2015 , fue finalmente lanzado al público el año siguiente, el 14 de abril de 2016, ya desde un principio Visual studio code se separó de sus competidores ofreciendo algunas características que eran innovadoras en su época como, por ejemplo, intellisense, un debugger integrado, seamless git integrado, un Marketplace y compatibilidad cross platform

Frameworks

FastAPI

Creado el 5 de diciembre de 2018 por el desarrollador de software Sebastian Ramirez, debido a su necesidad un framework web que sea de alto rendimiento para crear aplicaciones de tipo Api que usen Python

De esa idea nació Fastapi, un marco de trabajo que se caracteriza por su velocidad de desarrollo, rendimiento, fácil de escalar en el tiempo, documentación integrada automática con Swagger ui y ReDoc, una validación de datos que utiliza sugerencias de tipo Pydantic, esto combinado a que está diseñado para microservicios del tipo backend con aplicaciones hacia machine learning y aplicaciones de tiempo real, su soporte nativo para programación asíncrona permite implementar streaming de datos, también facilita la gestión de cargas pesadas, como análisis documental, y su tipado basado en Python mejora la claridad y robustez del código del lado del servidor.

Su estructura ligera y modular es adecuada para arquitecturas distribuidas donde cada microservicio se centra en una sola responsabilidad, estas características eran justamente las que se buscaban para este tipo proyecto y fue la razón de su elección.

Langchain

El sistema modelado inicialmente necesitaba de un marco de trabajo de código abierto para crear aplicaciones basadas en modelos de lenguaje de gran tamaño (LLM) para experimentar con diversos modelos de aprendizaje lógico (LLM) que permitan encontrar la solución óptima para sus aplicaciones

Todas estas cuestiones Langchain podía resolverlas cómodamente, fundado en 2022, la empresa Langchain inc, por el ingeniero Harrison Chase, bajo la premisa de que los modelos de lenguaje, para alcanzar su máximo potencial, deben poder interactuar con entornos externos y razonar sobre información contextualizada, más allá de los límites internos de entrenamiento que posean.

De esa visión nace el núcleo del marco de trabajo Langchain, es decir, estructuras que conectan un modelo de lenguaje con múltiples componentes (como bases de datos, APIs o sistemas de recuperación de información) para ejecutar tareas complejas mediante pasos estructurados.

Entre sus características más destacadas se encuentran:

- Modularidad: cada componente (modelos, memorias, herramientas, agentes, etc.) puede intercambiarse o combinarse según el diseño de la aplicación.
- Integración de memoria: permite mantener el contexto conversacional o de ejecución a lo largo del tiempo.
- Herramientas y agentes: introduce la figura de agentes capaces de decidir qué acción ejecutar en función de la información y herramientas disponibles.
- Compatibilidad amplia: soporta múltiples modelos de lenguaje (OpenAI, Anthropic, Google, HuggingFace, Ollama, entre otros).
- Ecosistema expansivo: dispone de conectores para bases de datos vectoriales, motores de búsqueda, frameworks de orquestación, facilitando la conexión entre la inteligencia del modelo y datos reales.

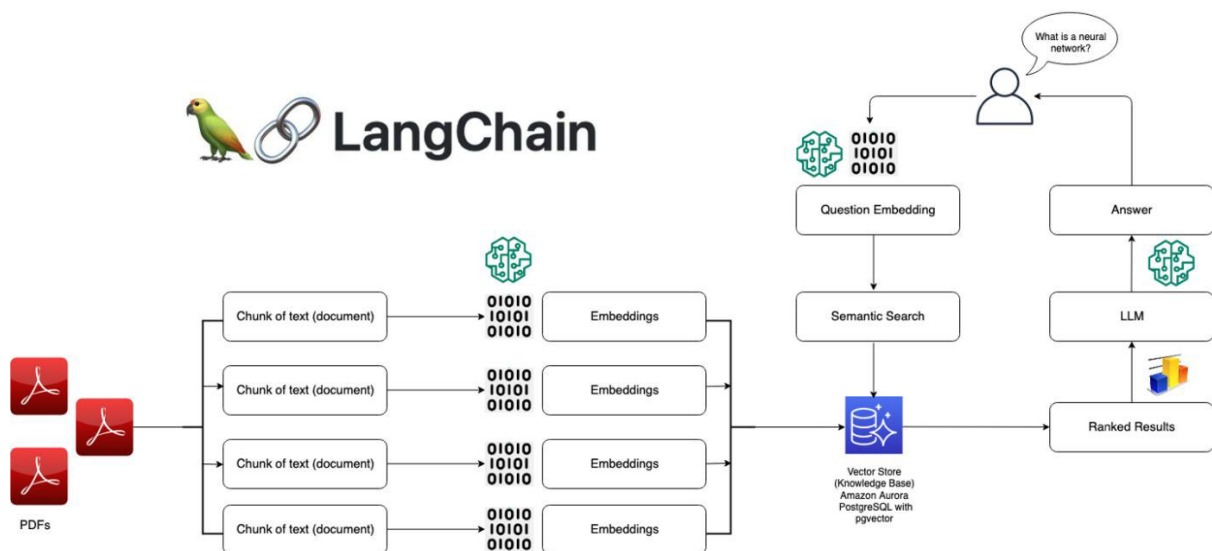


Ilustración 1 - Ecosistema LangChain

LangGraph

Creado por la empresa langchain Inc, dirigida por Harrison Chase, en 2024 se lanzó LangGraph es una librería de código abierto para agentes de IA diseñada para construir, implementar y gestionar flujos de trabajos complejos de agentes de IA generativos.

LangGraph usa un concepto donde cada nodo del grafo representa un paso en el cálculo, creando esencialmente un grafo de estado. Este enfoque permite que el grafo conserve

información sobre los pasos anteriores, posibilitando el procesamiento continuo y contextual de la información a medida que avanza el cálculo. Los usuarios pueden gestionar todos los grafos con estado de LangGraph mediante sus API.

Herramientas

Es una interfaz funcional que expone una capacidad o acción específica a un modelo de lenguaje. Esencialmente actúa como la lógica necesaria para realizar una acción externa del modelo de lenguaje grande (LLM), permitiéndole poder interactuar con sistemas ajenos y realizar tareas que el modelo no podría hacer por su cuenta.

En Python, el decorador `@tool` (de LangChain) es el mecanismo que hace que una función sea utilizable por el LLM:

- Toma una función estándar de Python.
- Genera automáticamente el **esquema JSON** a partir de los *docstrings* (descripción) y las anotaciones de tipo (*type hints* de los parámetros).
- Registra la función para que el agente la pueda **consultar** y **ejecutar** cuando el LLM solicite esa herramienta.

En resumen, la **Tool** es el puente que transforma la **intención** y el **razonamiento** del LLM en **interacción** con el microservicio que tiene las rutinas de las herramientas.

Base de datos

La elección de la base de datos requería de una base de datos, que permitiese manejar relaciones complejas, que sea intuitivo para el desarrollo, y que permitiese analizar y consultar grandes bancos de datos de manera estable y eficientemente

Para resolver estas cuestiones, se decidió usar Neo4j, primero conceptualizada a principios de los años 2000 por su creador Emil Eifrem, cuando se dio cuenta de varios errores y límites que se presentaban en bases de datos RDBMS,

Para subsanar esos problemas que encontró, se creó Neo4j, y se desplegó la primera versión en 2003, pero no fue sino hasta 2010, tres años después de la creación de la compañía de Neo4j que se liberó la versión pública de neo4j.

Neo4j permite hacer el almacenamiento de las relaciones de forma nativa como punteros directos, lo que permite recorrer de manera tanto eficiente como veloz rutas complejas o patrones en redes densas, el diseño en grafos permite un diseño con una mayor comprensión visual y facilita una comprensión rápida del diseño, estas características de neo4j fueron la principal razón de su elección para el sistema.

Lenguaje de consulta declarativo de grafos

El sistema de gestión de base de datos neo4j trae consigo su propio lenguaje de consulta, este lenguaje de consulta de grafos Cypher, se concibió como un lenguaje interpretado donde cada línea se lee y ejecuta secuencialmente. El mismo se diseñó con énfasis en el recorrido de rutas, la coincidencia de patrones y un estilo declarativo.

Cypher Query Language (Cypher) es un lenguaje de consulta de grafos declarativo diseñado para expresar consultas de coincidencia de patrones de forma intuitiva. Desarrollado originalmente para Neo4j, ahora es independiente del proveedor y compatible con múltiples plataformas.

Modelos de Lenguaje grande

El sistema tiene las tareas predeterminadas que son las encargadas de hacer el grueso del sistema, para ello en ciertos puntos, las mismas usan modelos de LLM, Ollama, MiniCPM, entre otros.

Para realizar estas rutinas se utilizan modelos de LLM, que son modelos de aprendizaje profundo (Deep learning) construidos con arquitecturas de redes neuronales, principalmente de la arquitectura transformer, que utiliza dos mecanismos fundamentales:

- **Mecanismo de Autoatención (Self-Attention):** Este es el corazón del Transformer. Permite al modelo **ponderar la importancia** de cada palabra (o *token*) en relación con todas las demás palabras de la secuencia de entrada. Esto es esencial para entender el **contexto completo** y las dependencias a largo plazo dentro de una oración o un documento, lo que mejora drásticamente la coherencia y precisión de las respuestas.
- **Naturaleza Secuencial (Generación de Tokens):** Los LLMs son intrínsecamente **modelos generativos** que predicen el **siguiente token** en una secuencia basándose en

todos los tokens que le precedieron. Esto es lo que les permite "escribir" o "completar" texto de forma fluida.

Modelos de lenguaje grande (LLM) y su arquitectura base

La arquitectura Transformer fue diseñada en el 2017 por un equipo de investigadores Google Brain en un documento científico (titulado "**Attention Is All You Need**") en el cual se planteaba un nuevo diseño de las arquitecturas de redes neuronales, un diseño de redes neuronales creado para procesar secuencias (texto, audio, código) de forma altamente eficiente sin necesidad de procesarlas en orden.

En lugar de recorrer la información paso a paso, usa un mecanismo llamado **atención** que permite que cada parte de la secuencia "mire" y relacione directamente con cualquier otra parte, sin importar la distancia entre ellas.

La arquitectura Transformer consta de dos componentes principales: el Codificador (*Encoder*) y el Decodificador (*Decoder*)

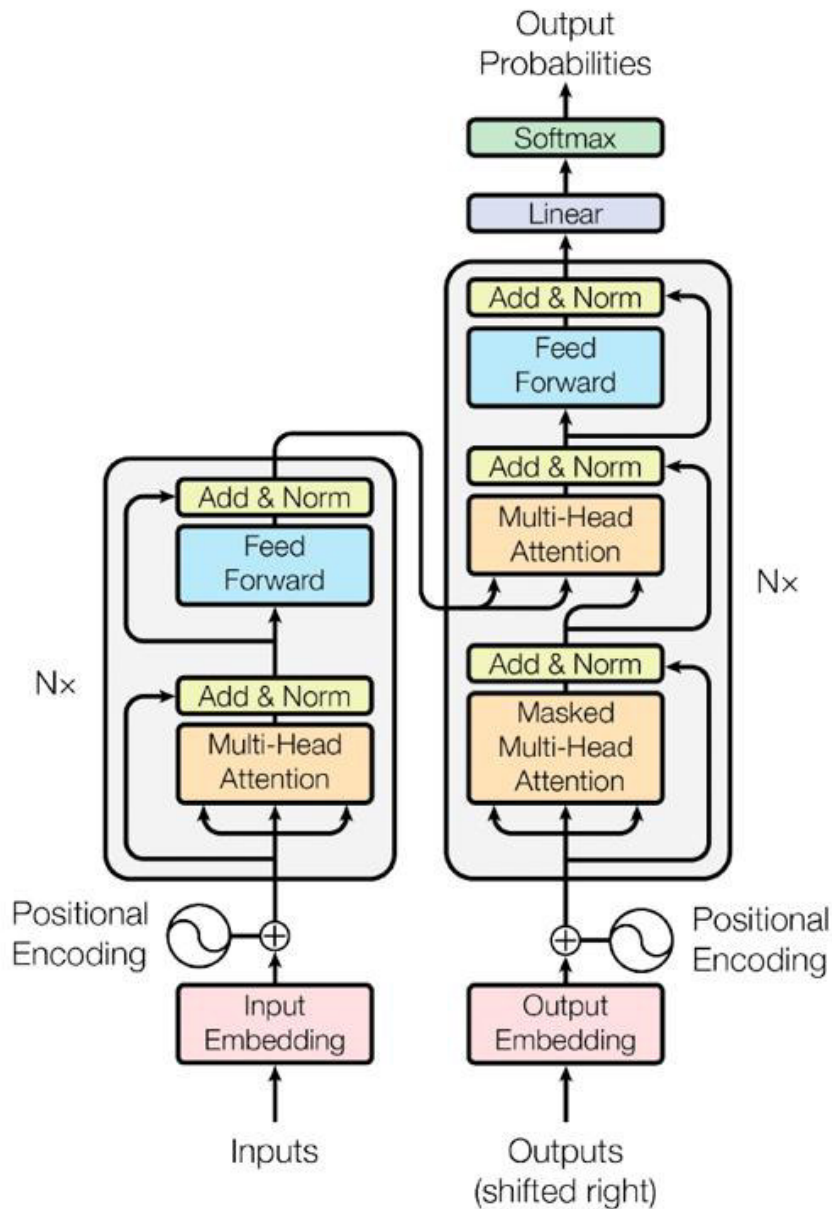


Ilustración 2 - Arquitectura Transformer

El **Codificador** procesa la secuencia de entrada completa (por ejemplo, una frase o una pregunta) para crear una representación de contexto rica en significado.

- **Input Embedding:** El texto de entrada se convierte en vectores numéricos (*Embeddings*).
- **Positional Encoding:** Se agrega información sobre la posición de cada palabra, ya que, el Transformer procesa todas las palabras simultáneamente.

- **Multi-Head Attention:** Este mecanismo permite al codificador ponderar la importancia de cada palabra en relación con todas las demás palabras de la entrada. Un ejemplo podría ser un ratón, que según el contexto podría referirse a una pieza de hardware o a un animal.
- **Add & Norm:** Las conexiones residuales (Add & Norm) ayudan a mantener el flujo de información
- **Feed Forward:** Esta capa permite introducir la capacidad de no linealidad para la red.
- **Capas Repetidas (N x):** El proceso se repite N veces (varias capas apiladas) para refinar progresivamente el entendimiento del contexto.

El **Decodificador** toma el vector de contexto del Codificador y genera la secuencia de salida (por ejemplo, la traducción, la respuesta o el resumen) palabra por palabra (*token por token*).

- **Output Embedding y Positional Encoding:** Similar a lo que sucede en el codificador, solo que convierte la secuencia de **salida** que se genera en vectores.
- **Masked Multi-Head Attention:** Esta es la clave de la generación. Se llama "enmascarada" porque el decodificador solo puede atender a las palabras que ya ha generado (es decir, no puede ver el futuro). Esto garantiza la generación secuencial del texto.
- **Multi-Head Attention (Codificador-Decodificador):** Esta es la conexión crítica. Permite al decodificador enfocarse en las partes más relevantes del vector de contexto producido por el codificador para guiar la generación de la siguiente palabra.
- **Linear:** Esta capa actúa como un clasificador preliminar, recibe un vector flotante por cada posición de la palabra que se está prediciendo y devuelve un vector de números reales, donde cada número corresponde a un token único en el vocabulario, es decir que asigna una puntuación bruta a cada palabra. Estos números se denominan logits.

- **Softmax:** Convierte las puntuaciones brutas (los *logits*) en una **distribución de probabilidad** significativa, esto se resuelve aplicando una función exponencial a cada *logit* y luego normalizando el resultado para que todos los valores sumen “1”, el resultado final es un vector donde cada elemento es la **probabilidad** de que el *token* correspondiente sea el siguiente en la secuencia.

El modelo entonces elige el *token* que tiene la **probabilidad más alta** en esta distribución como la predicción para el siguiente elemento de la frase que se está generando.

4. Marco teórico

Paradigmas del diseño de software

La arquitectura limpia es un enfoque de diseño de software propuesto por Robert C. Martin, que tiene como objetivo principal crear sistemas que sean independientes de detalles de implementación, como frameworks, bases de datos, y la interfaz de usuario.

Su definición formal se centra en una estricta separación de responsabilidades a través de capas concéntricas y la aplicación de una Regla de Dependencia fundamental, la regla implica que Las capas interiores no deben saber nada sobre las capas exteriores, el código en un círculo exterior puede depender del código en un círculo interior y el código en un círculo interior no debe depender del código en un círculo exterior.

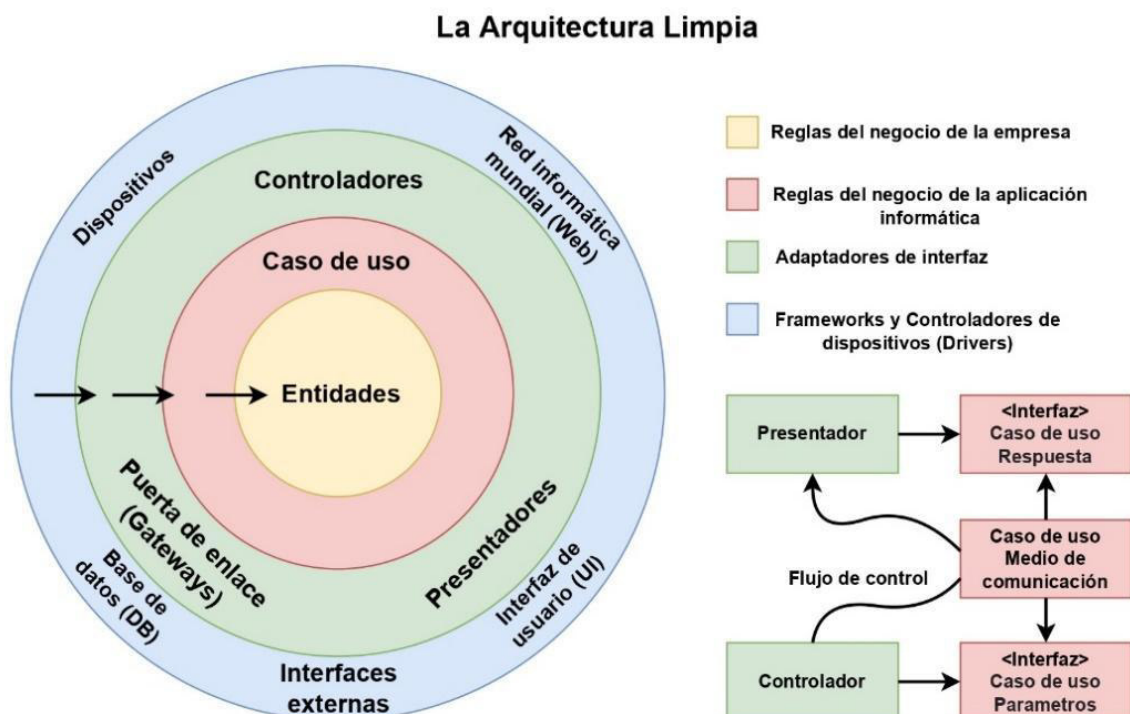


Ilustración 3 - La regla de la dependencia de arquitectura limpia

La Representación vectorial del lenguaje y espacio semántico

La **Representación Vectorial del Lenguaje** se materializa a través de un **Embedding** (o incrustación vectorial). Un embedding es un vector de números reales de alta dimensionalidad (típicamente entre 384 y 1536 dimensiones, dependiendo del modelo).

Se utilizan para representar segmentos de texto en un formato numérico manipulable por algoritmos de recuperación y modelos de aprendizaje profundo.

Estos vectores densos (es un vector donde la mayoría o todas las entradas son números distintos de cero) capturan patrones semánticos, sintácticos y contextuales presentes en el lenguaje, permitiendo que palabras, frases o *chunks* completos puedan ser comparados mediante operaciones geométricas.

- **Propósito:** Su objetivo es mapear el contenido textual de un segmento (*chunk*) a un punto en un **espacio vectorial** donde la **distancia geométrica** entre dos puntos es inversamente proporcional a la **similitud semántica** de sus textos correspondientes. Esto significa que textos conceptualmente equivalentes o relacionados tienden a ubicarse cercanos entre sí, incluso si no comparten vocabulario explícito, mientras que textos no relacionados aparecen separados en regiones distintas del espacio.
- **Fundamento Matemático:** Si el texto A es semánticamente similar al texto B, entonces la distancia entre el vector V_A y el vector V_B en el espacio vectorial (medida, por ejemplo, mediante la **Similitud del Coseno**) será pequeña. Esta relación se evalúa usualmente mediante Similitud del Coseno, que mide el ángulo entre los vectores, ignorando diferencias de magnitud. Valores cercanos a 1 indican alta similitud, mientras que valores cercanos a 0 indican baja correspondencia semántica. En cambio, si los textos no están relacionados, la similitud del coseno será baja y, por tanto, la distancia geométrica será mayor.

$$\text{similitud del coseno}(V_A * V_B) = \frac{V_A * V_B}{\|V_A\| * \|V_B\|}$$

Además, estos vectores no poseen dimensiones interpretables de forma individual; el significado está distribuido en todas las componentes. La estructura del espacio vectorial emerge del entrenamiento de modelos neuronales que aprenden a organizar la

semántica del lenguaje en regiones coherentes. Este comportamiento continuo permite que variaciones pequeñas en el texto produzcan ajustes graduales en la representación vectorial.

En la práctica, los embeddings suelen normalizarse a una norma unitaria, lo que facilita comparaciones estables y aumenta el rendimiento de los métodos de búsqueda aproximada utilizados en sistemas de recuperación semántica.

Esta normalización es especialmente importante en los algoritmos de *Approximate Nearest Neighbor* (ANN), técnicas diseñadas para localizar los vectores más cercanos en espacios de alta dimensionalidad sin evaluar todas las distancias de forma exhaustiva.

Los métodos ANN funcionan construyendo índices que exploran solo una fracción del espacio vectorial, manteniendo alta probabilidad de encontrar los vecinos más relevantes.

Entre sus enfoques más utilizados se encuentran:

- **HNSW (Hierarchical Navigable Small World Graph):** estructura basada en grafos con múltiples niveles que permite navegar progresivamente desde capas superiores (más dispersas) hacia niveles inferiores (más densos).
- **IVF (Inverted File Index):** particiona el espacio vectorial en *clusters* y busca únicamente en las particiones relevantes.
- **PQ (Product Quantization):** comprime los vectores en subespacios para acelerar comparaciones reduciendo precisión mínima necesaria.

Los métodos ANN funcionan de manera más eficiente cuando los vectores presentan distancias acotadas, una distribución uniforme en el espacio y métricas de similitud reducidas a operaciones simples.

Al proyectar todos los embeddings sobre una hiperesfera unitaria y eliminar las diferencias de escala, la normalización permite mejorar la navegación del índice, reduciendo el ruido en las comparaciones y aumentando la probabilidad de encontrar vecinos relevantes con bajo costo computacional.

Sistema RAG (Retrieval-Augmented Generation)

El paradigma Retrieval-Augmented Generation combina técnicas de recuperación de información con generación de texto condicionada por modelos de lenguaje grandes, permitiendo que las respuestas se fundamenten en evidencia documental externa y actualizable, en lugar de depender exclusivamente del conocimiento parametrizado del modelo.

El flujo completo de un sistema RAG se organiza en dos grandes fases:

Extracción y limpieza del texto desde documentos no estructurados.

- Se realiza la segmentación semántica (chunking) en unidades coherentes que respeten la estructura lógica del documento (párrafos, encabezados, ítems de lista, etc.).
- Generación de embeddings: cada fragmento se codifica mediante un modelo encoder (bi-encoder) en un vector denso que captura su significado semántico.
- Indexación vectorial: los vectores se almacenan en una base de datos especializada que permita búsquedas aproximadas de alta dimensión.

Recuperación y generación

- La consulta del usuario se convierte en un embedding utilizando el mismo modelo encoder.
- Se ejecuta una búsqueda de similitud (generalmente coseno) para recuperar los k fragmentos más relevantes del índice vectorial.
- Los fragmentos recuperados, junto con sus metadatos de origen, se insertan en el contexto del modelo generativo.
- El LLM genera la respuesta final condicionada a dicha evidencia, pudiendo incluir citas explícitas.

Este enfoque ofrece tres ventajas estructurales:

- Reducción drástica de alucinaciones al anclar la generación en evidencia real.
- Actualización instantánea del conocimiento mediante la incorporación o eliminación de documentos.

- Trazabilidad y citación automática de fuentes.

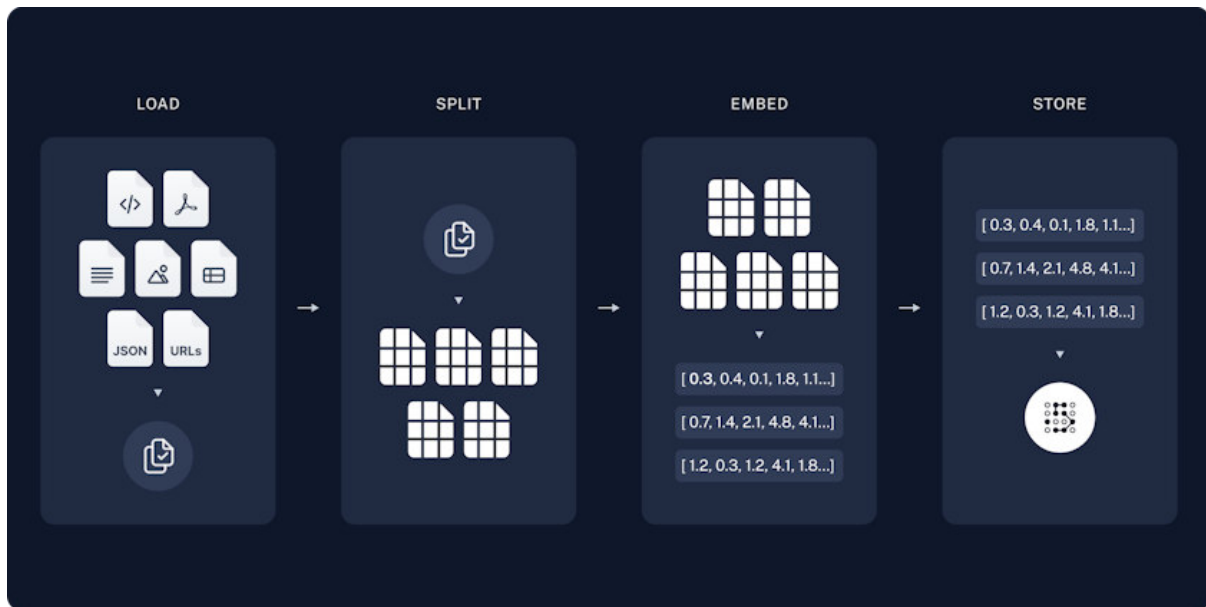


Ilustración 4 - Sistema Rag

Orquestación de Agentes y Grafos de Tareas

Grafo: es una estructura matemática que permite representar relaciones entre objetos, se compone de un conjunto de vértices o nodos, y líneas, que se llaman aristas o arcos, que los conectan.

En LangGraph esta estructura se representa como un mapa de decisiones con memoria, con nodos que representan las acciones que realiza el agente, y las aristas que representan las acciones que el orquestador va a tomar a continuación.

- **Grafo cíclico:** un grafo cíclico es cualquier grafo que contiene al menos un ciclo y es esencial para la ejecución de agentes, esto significa que existe una ruta que comienza y termina en el mismo nodo, formando un bucle dentro del grafo. Los flujos de trabajo complejos suelen implicar dependencias cíclicas, donde el resultado de un paso depende de los pasos anteriores del bucle.
- **Nodos:** En LangGraph, los nodos representan componentes o agentes individuales dentro de un flujo de trabajo de las IA. Se puede pensar en los nodos como “actores” que interactúan entre sí de una manera específica.

- Aristas:** En Python, las aristas son una función que determina qué nodo ejecutar a continuación en función del estado actual. Pueden ser bifurcaciones condicionales o transiciones fijas.

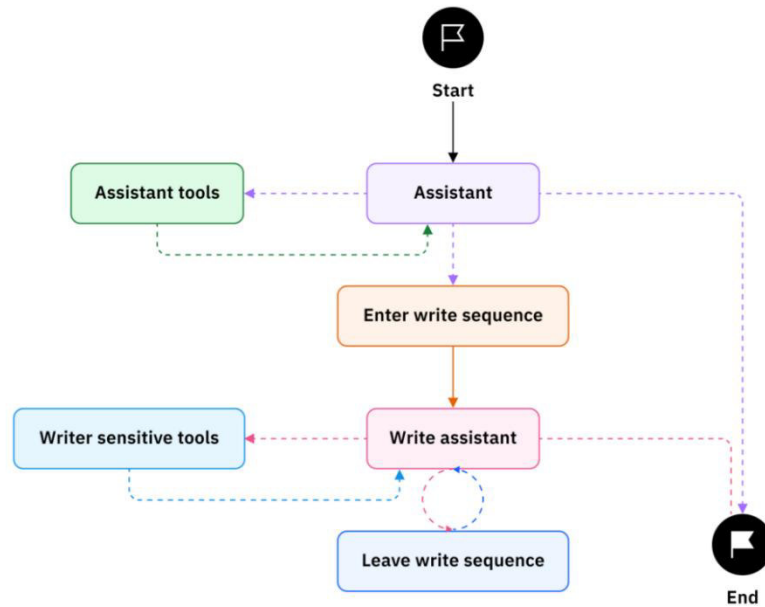


Ilustración 5 - Modelo de Langgraph

Agentes: la versión formal de un agente en Langgraph se podría definir como una función determinística o estocástica que, dada un estado (state) y un input, produce un nuevo estado y/o una acción, y que se modela como un nodo dentro de un grafo dirigido (DAG o grafo cíclico permitido) que compone un *workflow de agentes*.

Siendo la representación matemática: Un agente A es una función $A : S \rightarrow S'$

La versión explícita representado como $A(S:I) \rightarrow (S', O)$

donde:

- S = estado actual (por ej. memoria, historial, variables)
- I = input del entorno o de otros nodos
- S' = nuevo estado
- O = salida (por ejemplo, una llamada a herramienta, texto, decisión)

Y en la práctica podría verse dando un ejemplo como el siguiente:

```
def agent_node(state):
```

```
    # lee el estado
```

```
# genera output
# actualiza estado
return new_state
```

Es decir, el agente es una transformación del estado, no una entidad persistente la inteligencia proviene del modelo de LLM que sea contenido en los nodos.

Los agentes se conectan mediante edges que permiten definir el flujo de control, permitiendo ciclos de así ser necesario (esto se denomina comportamiento reactivo /auto recursivo)

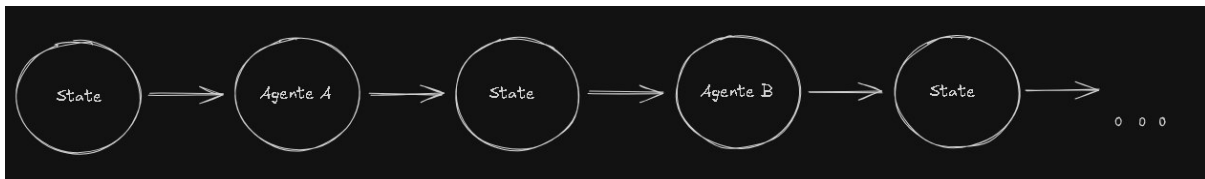


Ilustración 6 - Flujo de nodos

Finalmente, cabe aclarar que en LangGraph un agente no es definido por su complejidad, sino por el contrato de entrada y salida de este.

Grafos determinísticos vs Grafos no determinísticos

En el sistema se pueden denotar 2 grandes patrones descriptos por Langgraph, routing estáticos (grafos determinísticos) y routing dinámico (no determinístico)

El módulo de upload como routing estatico

Tomando como ejemplos el proceso de ingesta documental, compuesto por las etapas *upload* → *viability* → *extract* → *metadata* → *extract_images*, se ajusta al patrón de **routing estático**. En este tipo de grafos, cada transición entre nodos se encuentra predefinida y no admite variación derivada ni de la entrada en tiempo real ni del razonamiento del modelo.

Desde una perspectiva teórica, este flujo puede interpretarse como la instanciación de un **pipeline secuencial con topología fija**, donde el grafo no permite expresar condiciones internas ni ramificación dinámica.

Esta naturaleza del pipeline se traduce en tres propiedades fundamentales que LangGraph asocia a estos tipos de grafo:

- **Invariancia estructural:** ninguna etapa modifica la posibilidad de que la siguiente deba ejecutarse.
- **Determinismo fuerte:** el comportamiento del sistema es una función pura del documento de entrada; no existe variabilidad inducida por heurística o razonamiento probabilístico.
- **Trazabilidad absoluta:** al no haber caminos alternativos, el flujo es completamente auditable y replicable.

Este enfoque es consistente con la función del módulo: garantizar que el documento sea transformado de un objeto a una unidad estructurada, el módulo encarna un caso de **computación declarada y no deliberativa**, donde la semántica del proceso no depende de interpretación, sino de reglas formales preestablecidas.



Ilustración 7 - Grafo Routing estatico

El módulo conversacional como grafo dinámico guiado por modelos

El módulo conversacional (*orchestrator_chat*), en contraste, se fundamenta en la dinámica que LangGraph asocia a los **model-driven graphs**, donde las transiciones no están fijadas de antemano, sino que emergen del razonamiento del modelo en cada iteración. Aquí, el nodo central de razonamiento funciona como un “operador de transición probabilístico” que, basándose en el estado acumulado, determina cuál será la siguiente acción del grafo. Esta dinámica es precisamente la que LangGraph describe como:

LLM-controlled branching

Es la **acción específica** de un LLM al decidir qué camino tomar en un punto de decisión predefinido dentro de un flujo de trabajo. Es el mecanismo de toma de decisiones.

Dynamic planning

Es la **capacidad general** del LLM para generar, modificar y corregir la secuencia de pasos (el plan) en tiempo real basándose en el contexto y los resultados intermedios. Es el "cerebro" del agente.

Adaptive execution graphs

Es la **estructura arquitectónica** (como un diagrama de flujo avanzado) que se utiliza para organizar y gestionar los estados y las transiciones (bifurcaciones) de un agente. Es el "mapa de ruta"

Desde un punto de vista teórico, este tipo de grafo introduce una forma limitada de **no determinismo semántico**, donde cada transición está condicionada por la evaluación del estado, incluyendo el historial conversacional, el contexto documental, la salida previa de las herramientas y las reglas implícitas de decisión incorporadas en el modelo.

El grafo deja de ser una secuencia fija y se transforma en un **espacio de posibles recorridos**, donde el modelo actúa como un **planificador introspectivo**, seleccionando rutas en función de la intención declarada por el usuario detectada por el modelo.

LangGraph considera este paradigma como un ejemplo de **control de flujo emergente**, donde la lógica de ejecución no está completamente codificada, sino **delegada** a un agente inferencial. Las propiedades características son:

- **Rutas dependientes del estado:** el recorrido del grafo cambia en función del contenido generado hasta el momento.
- **No determinismo controlado:** el grafo no es arbitrario, pero tampoco es fijo; las decisiones se derivan del modelo.
- **Iteración deliberativa:** el grafo puede entre nodos de razonamiento y nodos de acción, formar bucles de planificación, ejecución y replanificación.

Este comportamiento es estructuralmente distinto del grafo determinista del módulo de ingesta: mientras la ingesta transforma datos siguiendo reglas prescritas, el módulo conversacional **delibera** sobre los datos recibidos y permite al modelo decidir cómo proceder. La función profunda de este grafo no es producir **estructura**, sino producir **interpretación**.

Grafo No determinísticos (Routing Dinámico)

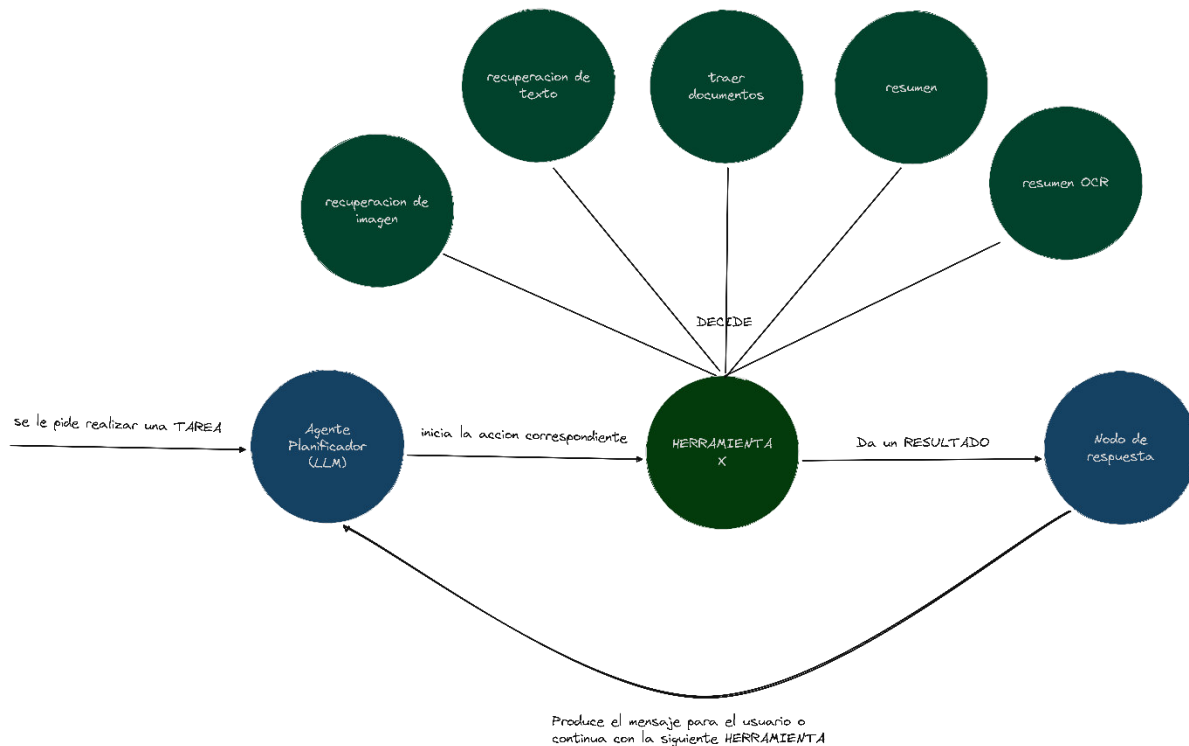


Ilustración 8 - Grafo Routing Dinámico

Contraste conceptual desde la teoría de grafos en LangGraph

A nivel teórico, ambos módulos representan dos formas de computación complementarias dentro del marco que LangGraph promueve:

El grafo determinista

Corresponde a la computación **procedimental**, donde todas las operaciones están definidas por el diseñador del sistema. El sistema no necesita *agencia*, sino *regularidad*. La ausencia de decisiones basadas en modelos garantiza una consistencia ontológica del documento como entidad procesada.

El grafo dinámico basado en modelos

Corresponde a la computación **agentiva**, donde un modelo de lenguaje participa en la determinación del recorrido del grafo. Aquí el sistema no solo procesa, sino que también

razona sobre el procesamiento. Es un tipo de grafo donde las transiciones no están contenidas únicamente en la estructura, sino también en la inferencia.

Desde la teoría de LangGraph, ambos enfoques pueden coexistir dentro de un mismo sistema porque representan dos abstracciones distintas del papel de los nodos:

- en la ingesta, los nodos representan **transformaciones deterministas sobre un objeto**,
- en la conversación, los nodos representan **posibilidades de acción seleccionadas por un planificador probabilístico**.

Este contraste subraya la complementariedad epistémica entre ambos módulos: el primero crea conocimiento procesado de manera estable, el segundo lo activa de manera flexible.

- El módulo *orchestrator_upload* se ajusta al modelo de **grafo determinista**, donde la secuencia es fija y la computación se concibe como una transformación ordenada del documento.
- El módulo *orchestrator_chat* adopta el modelo de **grafo dinámico guiado por inferencia**, donde la computación emerge de un ciclo deliberativo entre el estado del sistema y el razonamiento del modelo.

El sistema, por lo tanto, articula dos formas de organización del flujo que LangGraph reconoce como fundamentales: la computación que **produce estructura** y la computación que **produce decisiones**. La primera sostiene la ontología documental; la segunda habilita la agencia conversacional y el razonamiento sobre esa ontología.

5. Visión General del sistema

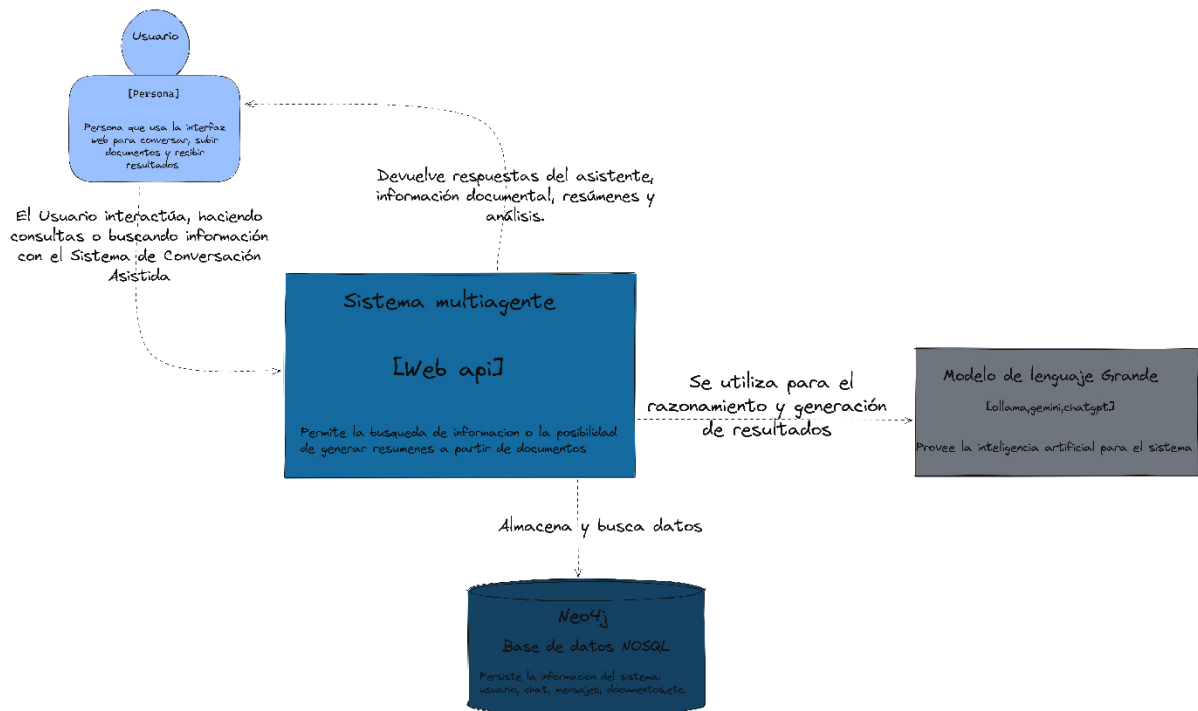


Ilustración 9 - Diagrama C1 del sistema

El sistema se desarrolla implementando una arquitectura distribuida orientada al procesamiento inteligente de datos sobre documentos científicos. Para lograr tal cometido, se ha implementado una aplicación web API.

La arquitectura está basada en microservicios, esto significa que adopta un enfoque donde cada uno, funciona como una unidad autónoma con su propia responsabilidad, ciclo de vida, api expuesto, con comunicación **REST** y orquestación modular mediante LangGraph y Fastapi.

El sistema se organiza alrededor del microservicio de backend, que actúa como el **orquestador lógico**, coordinando:

- La interacción con el usuario en tiempo real,
- El razonamiento dinámico del LLM,
- El uso de herramientas externas (RAG, resúmenes, obtención de documentos),
- La persistencia distribuida de información conversacional y documental.

A gran escala, el sistema se divide en tres grandes planos funcionales:

Plano de Orquestación Conversacional: este es el motor de razonamiento dinámico basado en LangGraph, que se ejecuta en el backend principal.

Plano de Procesamiento Documental: el microservicio que se dedica a la extracción, segmentación semántica del texto, incorporación vectorial para cada uno de los segmentos, y recuperación vectorial.

Plano de Persistencia y Estado: este microservicio está especializado en abstraer todo acceso a la base de datos de Neo4j y permite almacenar la información de los usuarios, chat, mensajes, documentos, etc.

Funcionalidad General del Sistema

El sistema proporciona las siguientes capacidades integrales:

- **Carga y procesamiento completo de documentos** (PDF, textos, etc.).
- **Indexación semántica y generación de fragmentos (chunks)** optimizados para LLMs.
- **Recuperación vectorial (RAG)** basada en similitud semántica.
- **Generación de resúmenes contextuales.**
- **Interacción conversacional fluida** con razonamiento en múltiples pasos.
- **Persistencia completa de chats y mensajes** en una base de grafos.
- **Selección dinámica de herramientas** por parte del LLM.

El backend y los microservicios de procesamiento operan sin estado persistente, toda la persistencia del sistema se delega al microservicio de la base de datos. esto permite escalabilidad horizontal.

El uso de herramientas registradas (tools) hace posible agregar nuevas capacidades sin modificar el núcleo conversacional, el orquestador solo necesita registrar una nueva herramienta y el LLM puede comenzar a utilizarla sin cambios estructurales.

Estilo Arquitectónico

El sistema implementa un conjunto de patrones arquitectónicos complementarios que permiten estructurar una plataforma escalable, modular y capaz de coordinar tanto microservicios especializados como modelos de lenguaje de gran escala. Estos patrones contribuyen a desacoplar funcionalidades, reducir la complejidad del sistema, habilitar capacidades avanzadas de razonamiento automatizado, recuperación documental y persistencia coherente.

- **Microservicios**
- **Orchestration Pattern**
- **Event-Driven Reasoning** (LLM-driven)
- **Pipe-and-Filter** (en el procesamiento documental)
- **Gateway Pattern** (microservicio de la base de datos)
- **Adapter Pattern** (registros de herramientas)
- **Client–Orchestrator–Service Architecture**

Microservicios

La plataforma está organizada en unidades funcionales autónomas, cada una responsable de un dominio específico. Por ejemplo, el microservicio de la base de datos(ms-db) gestiona exclusivamente la persistencia de chats y mensajes en Neo4j, mientras que otros servicios externos pueden conectarse sin afectar el núcleo del sistema.

Cada microservicio expone únicamente endpoints REST, lo que permite independencia en ciclos de despliegue, escalabilidad horizontal selectiva y tolerancia a fallos localizada.

Orchestration Pattern

La orquestación implica que el control del flujo recae en un único componente central (el orquestador del chat), que decide qué pasos ejecutar en cada etapa: recuperar memoria, invocar herramientas, interpretar resultados, generar respuestas o persistir interacciones.

Esto se materializa a través del **agente orquestador**, que evalúa el estado conversacional (state machine), decide si debe activar la recuperación de documentos (tool retrieval), llamar a la herramienta de resumen (tool summarize) o generar una respuesta directa, y luego delega la ejecución en los módulos correspondientes.

Event-Driven Reasoning (LLM-driven)

El comportamiento del sistema se rige por una secuencia de eventos desencadenados por el propio modelo de lenguaje. El agente central no sigue un flujo rígido; por el contrario, **cada paso está condicionado por la salida del LLM**, que analiza el estado actual y emite una instrucción implícita del tipo:

- “invocar la herramienta X”
- “realizar recuperación de contexto”
- “finalizar la respuesta”

Este enfoque convierte las llamadas a herramientas, la selección de documentos, la desambiguación y la decisión de detenerse en **eventos generados dinámicamente** por el razonamiento del modelo. El orquestador, a través del motor **LangGraph**, **gestiona las transiciones del grafo** en respuesta a estos eventos, estableciendo el ciclo iterativo de **Percepción–Decisión–Acción** que guía el flujo de la conversación hasta su resolución.

Pipe-and-Filter en el procesamiento documental

El sistema de ingestión y análisis de documentos implementa un pipeline en el cual cada etapa procesa el resultado de la anterior, sin compartir estado. Por ejemplo:

- Carga del documento.
- Extracción de texto.
- Detección de columnas o estructuras.
- Limpieza y normalización.
- Segmentación en chunks.
- Indexación de vectores.

Cada módulo cumple la función de un **filtro**, y la salida se encadena hacia el siguiente en un flujo lineal. Esto permite optimizar o reemplazar cualquiera de las etapas sin modificar el resto del pipeline, manteniendo una arquitectura de procesamiento transparente y extensible.

Gateway Pattern (ms-db)

El microservicio de la base de datos implementa un patron del tipo gateway que encapsula toda la lógica de acceso a Neo4j, por lo tanto, el backend nunca interactúa directamente con la base de datos; en su lugar:

- envía solicitudes HTTP estandarizadas,
- recibe datos ya estructurados,
- delega completamente la gestión de nodos, relaciones y consultas Cypher.

Adapter Pattern (tool registry)

La interacción entre el orquestador y las herramientas se realiza mediante un **registro de herramientas**, que actúa como adaptador entre:

- El formato que el LLM necesita para describir herramientas,
- Y la implementación concreta de cada herramienta en el backend.

El registro provee descripciones estandarizadas, normaliza I/O y permite agregar nuevas herramientas sin modificar el flujo principal del orquestador.

El LLM solo observa un catálogo coherente, ignorando la complejidad interna de cada módulo. Esto habilita la extensibilidad dinámica en la plataforma.

```

tool_registry.py M X
app > orchestrator_chat > adapters > tool_registry.py > get_registered_tools
1 from app.orchestrator_chat.tools.tool_get_documents import get_user_documents
2 from app.orchestrator_chat.tools.tool_retriever import retriever_tool
3 from app.orchestrator_chat.tools.tool_summarize import summarizer_tool
4 from app.orchestrator_chat.tools.tool_ocr_summarize import ocr_summarize_tool
5 from app.orchestrator_chat.tools.tool_image_fetcher import ocr_fetch_image_tool
6
7 TOOLS = [
8     get_user_documents,
9     retriever_tool,
10    summarizer_tool,
11    ocr_summarize_tool,
12    ocr_fetch_image_tool
13 ]
14
15
16 def get_registered_tools():
17     """
18     Devuelve la lista de herramientas registradas (objetos @tool)
19     para ser usadas dinámicamente por el LangGraph Orchestrator.
20     """
21     return TOOLS
22
23
24 def get_tool_names():
25     """
26     Devuelve solo los nombres de las herramientas registradas.
27     Útil para debugging o introspección del sistema.
28     """
29     return [getattr(t, "name", getattr(t, "__name__", "unknown_tool")) for t in TOOLS]
30

```

Ilustración 10 - Adapter: tool registry

Client–Orchestrator–Service Architecture

El sistema completo puede describirse como una arquitectura tripartita:

- **Cliente:** Es el que emite las consultas y recibe respuestas en forma conversacional.
- **Orquestador:** Interpreta la intención del usuario, decide los pasos siguientes, activa las herramientas necesarias y coordina los servicios.
- **Servicios:** Ejecutan las operaciones especializadas (persistencia, búsqueda vectorial, extracción, resúmenes externos, etc.).

Rol del LLM en la Arquitectura

El modelo de lenguaje no es simplemente un generador de texto, sino un **planificador autónomo** dentro del orquestador, esto significa que puede:

- Interpretar el mensaje del usuario
- Decidir qué herramientas usar
- Razonar sobre documentos
- Coordinar múltiples consultas externas
- Generar salidas finales coherentes y justificadas

Este rol sitúa al LLM como una capa de control inteligente, integrándose con el backend pero no reemplazando sus responsabilidades arquitectónicas.

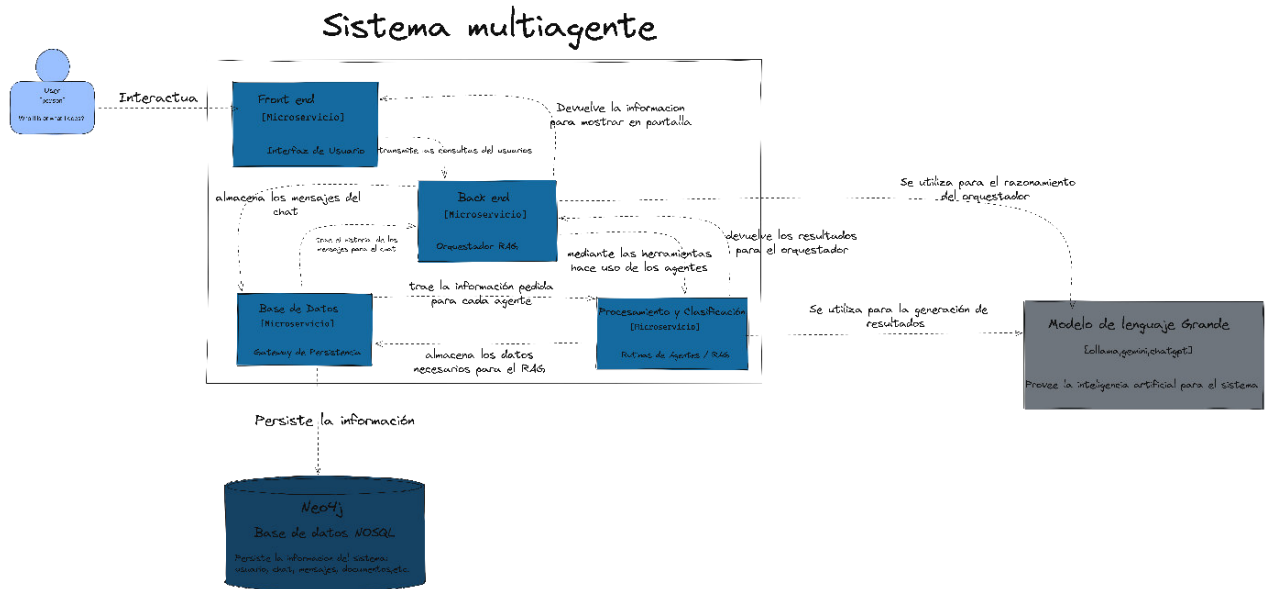


Ilustración 11 - Diagrama C2 del sistema

6. Metodologías

A continuación, se detalla la función específica de cada uno de los microservicios que componen al sistema.

Microservicio de base de datos

En un principio, debido a que eran pocas rutinas que usaban la base de datos se usaba una conexión directa con neo4j, pero a medida que el sistema aumentaba, cada vez más se acumulaba la duplicación de lógica para el acceso a la base de datos. Se necesitaba entonces crear un microservicio que evitara problemas de este tipo.

Este sistema de persistencia central tiene como objetivo ser tanto el único punto por el cual se pueda acceder a los datos, de esta forma los otros microservicios solo pueden acceder a los datos a través de una comunicación API REST

Arquitectura interna del microservicio

El sistema se divide en 4 módulos, el módulo del chat, para los mensajes del usuario y asistente, el módulo de usuario, para poder definir al usuario al momento de crearlo y que pueda acceder a su repositorio de documentos, el módulo de búsqueda (search), que permite acceder a la búsqueda de vectores de los textos en los documentos y el módulo del documento, que es el que se encarga de definir como se guardan los documentos, las imágenes, sus “trozos” de textos entre otros.

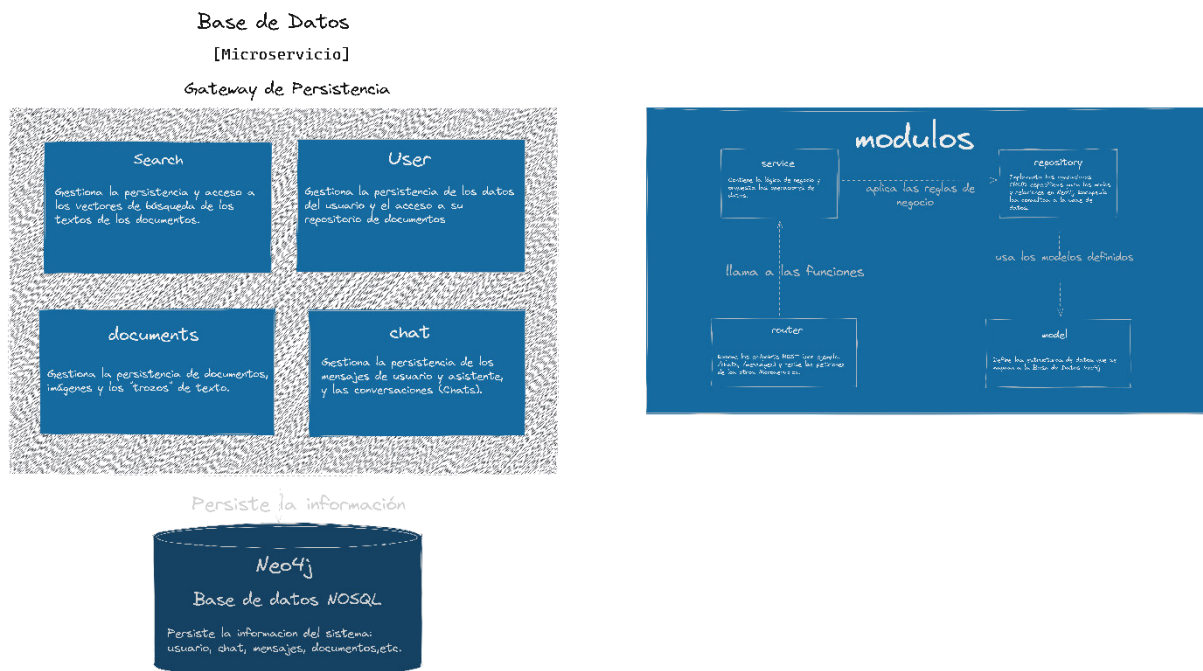


Ilustración 12 - Diagrama C3 del microservicio de BD

Model Layer (Capa de Modelos)

Esta es la capa de modelos, es la que permite definir las entidades que se manejan en el sistema, en este caso, se podría tomar el módulo de chat, los modelos de chat y message.

- **Chat:** Es el contenedor de las conversaciones del usuario, donde se agrupan los mensajes del usuario y asistente. Un chat puede tener muchos mensajes y un usuario puede tener muchos chats
- **Message:** Un mensaje es una unidad de conversación dentro del chat.

Formato Orientado a Grafos:

- En Neo4j, los datos se representan como **nodos** y **relaciones**. Un nodo podría representar a un chat o un mensaje, y las relaciones entre nodos pueden modelar las interacciones entre chats y mensajes.
- La relación entre un Chat y los Message se define como chat-> has message ->message



Ilustración 13 - Representación de nodos en Neo4j

Repository Layer (Capa de Repositorios)

En la capa de repositorio, se implementa el acceso a la base de datos (en este caso, **Neo4j**). Esta capa sigue el patrón **Repository** que proporciona una interfaz para acceder a los datos sin exponer detalles de la base de datos subyacente.

- Los repositorios contienen las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para trabajar con los nodos y relaciones en la base de datos.
- **CRUD en Neo4j:** Por ejemplo, crear un Chat o un Message, obtener todos los mensajes dentro de un Chat, actualizar el contenido de un mensaje, o eliminar un chat (y, potencialmente, todos los mensajes relacionados).

Service Layer (Capa de Servicios)

La capa de servicios contiene la **lógica de negocio** de la aplicación. Aquí es donde se realiza el procesamiento de los datos y la interacción entre los modelos.

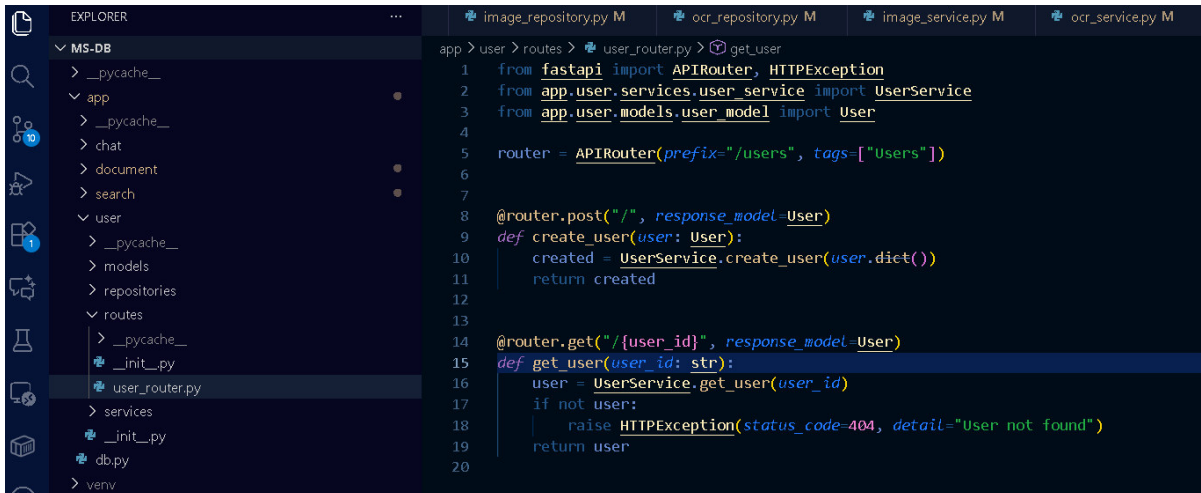
- **Lógica de creación:** Crear un nuevo chat o mensaje
- **Lógica de búsqueda:** Buscar un chat específico o los mensajes de un chat
- **Relaciones entre chats y mensajes:** Cuando se crea un mensaje, este tiene que estar asociado con un chat.

El servicio se comunica con los repositorios para obtener o modificar los datos.

Router Layer (Capa de Ruteo)

La capa de ruteo, en este caso, es la que expone **endpoints REST** (las rutas HTTP) a través de **FastAPI**.

Estos endpoints estarán conectados a los servicios. Es decir, cuando un usuario haga una solicitud, el router llamará al servicio correspondiente para procesar la solicitud, y el servicio a su vez interactúa con los repositorios para acceder a los datos de la base de datos.



```

EXPLORER
├── MS-DB
│   ├── __pycache__
│   └── app
│       ├── __pycache__
│       ├── chat
│       ├── document
│       ├── search
│       └── user
│           ├── __pycache__
│           ├── models
│           ├── repositories
│           └── routes
│               ├── __pycache__
│               ├── __init__.py
│               └── user_router.py
│                   ├── __init__.py
│                   ├── services
│                   ├── __init__.py
│                   ├── db.py
│                   └── venv
├── image_repository.py M
├── ocr_repository.py M
├── image_service.py M
└── ocr_service.py M

app > user > routes > user_router.py > get_user
1 from fastapi import APIRouter, HTTPException
2 from app.user.services.user_service import UserService
3 from app.user.models.user_model import User
4
5 router = APIRouter(prefix="/users", tags=["Users"])
6
7
8 @router.post("/", response_model=User)
9 def create_user(user: User):
10     created = UserService.create_user(user.dict())
11     return created
12
13
14 @router.get("/{user_id}", response_model=User)
15 def get_user(user_id: str):
16     user = UserService.get_user(user_id)
17     if not user:
18         raise HTTPException(status_code=404, detail="User not found")
19     return user
20
    
```

Ilustración 14 - ejemplo de router para un usuario

Microservicio de procesamiento y clasificación

Este es el microservicio responsable de todas las rutinas del sistema, de realizar la carga de documentos a generar la extracción de los datos, hacer la incrustación de vectores en los datos (embedding) o extraer las imágenes del documento hasta analizar la viabilidad de los documentos, generar metadatos, hacer la recuperación de la información y generar resúmenes. Todas estas rutinas son las que luego las tool del backend llaman para que el modelo de LLM, pueda generar un flujo dinámico en respuesta a la solicitud del usuario

Componentes del sistema

A continuación, se dará un detalle de los agentes, con consideración especial para aquellos que hacen el grueso del sistema.

Agente de upload

Este es el punto de entrada del sistema, la función principal es la de recibir un documento y almacenarlo en el entorno local del microservicio.

Este agente es la interfaz que permite transformar un flujo de datos binarios en una referencia estándar lista para ser usada por otros módulos del sistema.

Agente de viability:

Este módulo tiene como objetivo determinar si un archivo que el usuario subió al sistema es adecuado para ser procesado por otros módulos.

Este agente básicamente revisa que el archivo cumpla con reglas básicas (que no esté vacío, corrupto, que tenga texto, etcétera). Este filtro permite garantizar que solo documentos válidos ingresen al flujo formal de extracción

Agente de metadatos:

Este agente tiene como propósito extraer, y almacenar metadatos formales del documento: título, autores, tamaño, dimensiones físicas, cantidad de página, codificación interna y otros atributos.

Agente de embedding:

Este agente transforma los segmentos que se extrajeron del texto y les asigna una incrustación vectorial para su posterior recuperación usando la librería MiniLM.

```
app > agents > embedding_agent.py > ...
1  from sentence_transformers import SentenceTransformer
2
3  _model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")
4
5  def generate_embedding(text: str) -> list[float]:
6      #Genera el embedding de un texto usando MiniLM.
7      if not text or not text.strip():
8          return []
9      embedding = _model.encode(text)
10     return embedding.tolist()
11
```

Ilustración 15 - Agente de embedding

Agente de extracción:

Este módulo es responsable de la ingesta y preparación de documentos no estructurados (ej. archivos PDF) para su posterior uso en un sistema de generación aumentada por recuperación (RAG).

El proceso se divide en cinco etapas fundamentales: Extracción de texto, mapeo de origen, Chunking estratégico, generación de Embeddings e indexación

Extracción de Texto

La función clave aquí es que se utiliza la librería PyMuPDF (fitz) para procesar archivos PDF, un estándar en el manejo de documentos.

En la segmentación, el texto de cada página se somete a una segmentación inicial a nivel de párrafo, esto se lleva a cabo empleando una expresión regular sofisticada (un ejemplo de lo anterior sería la que se usa en el sistema “(re.split(r'(?:\n\s*\n+)|(?!<=[\.\?!\])\s*\n', text))”) que intenta dividir el texto basándose en saltos de línea múltiples (típicos de separación de párrafos) o en un salto de línea que sigue a una puntuación terminal (punto, interrogación, exclamación).

Cada párrafo extraído se almacena junto a su número de página original. Este metadato es vital para la trazabilidad y la citación en la fase de respuesta del RAG, permitiendo al modelo señalar la fuente exacta de la información.

Estrategia de Chunking Adaptativo (Tokenización)

Se implementa una estrategia de división de texto (chunking) que prioriza la cohesión semántica sobre la simple división por tamaño fijo.

Unidad Semántica Base: El proceso principal de chunking en el componente de extracción opera sobre los párrafos ya extraídos y mapeados. Esto asegura que, en la medida de lo posible, un chunk no divida una idea a mitad de un párrafo, preservando la coherencia.

Fallback a Tokenización: Se utiliza la librería Tiktoken para medir la longitud de cada párrafo en tokens. Si un párrafo supera un umbral predefinido (e.g., el doble del `chunk_size`), se aplica una técnica de sub-chunking por tokens.

Esto permite manejar párrafos excepcionalmente largos (como descripciones técnicas densas) de forma efectiva, dividiéndolos en fragmentos de tamaño máximo (500 tokens) para la memoria del modelo y para optimizar la búsqueda por embeddings.

Metadatos de Chunk: Cada fragmento resultante se enriquece con metadatos como un `chunk_id` único, un `index` secuencial y el `token_count` real.

Generación de Embeddings

Se utiliza el **agente de embedding** que actúa dedicado al procesamiento de vectorización y que utiliza un modelo de lenguaje pre-entrenado (minilm).

Cada segmento de texto se transforma en un vector numérico de alta dimensión. Este permite capturar el significado semántico del mismo. Estos embeddings son la base de la fase de recuperación, ya que permiten buscar fragmentos de texto relevantes basándose en la similitud vectorial con la consulta del usuario.

En el sistema RAG, la función de generar el embedding (`generate_embedding`) es crítica ya que genera:

- **Entrada:** Un *chunk* de texto coherente (resultante de la fase de *chunking*).
- **Proceso:** El *chunk* ejecuta el modelo de *embedding* (MiniLM).
- **Salida:** Un vector numérico que se adjunta como el campo "embedding" a cada *chunk* de datos.

```

embedding_agent.py  extraction_agent.py  image_extraction_agent.py  image_extraction_router.py
app > agents > extraction_agent.py > extraction_agent
1  import uuid
2  import fitz # PyMuPDF
3  import tiktoken
4  import requests
5  import re
6
7  DB_SERVICE_URL = "http://127.0.0.1:8080" # ms-db
8  EMBEDDING_SERVICE_URL = "http://127.0.0.1:8081" # ms-procesamiento y clasificación
9
10
11 // # 1. genera los embeddings
12 def generate_embedding(text: str) -> list:
13     try:
14         resp = requests.post(
15             f"{EMBEDDING_SERVICE_URL}/embeddings/minilm",
16             json={"text": text},
17             timeout=300
18         )
19         resp.raise_for_status()
20         return resp.json().get("embedding", [])
21     except Exception as e:
22         print(f"[WARN] No se pudo generar embedding: {e}")
23     return []
24
25
26 # 2. Divide en párrafos
27 def split_by_paragraphs(text: str) -> list[str]:
28     paragraphs = re.split(r'(?:\n\s*\n+)|(?<=[\.\?!\])\s*\n', text)
29     return [p.strip() for p in paragraphs if len(p.strip()) > 50]
30

```

Ilustración 16 - Agente de extracción

Indexación y Persistencia

La fase final en el componente de extracción es la persistencia de los datos procesados. Se consolida una estructura de datos completa que incluye metadatos del documento y una lista de todos los chunks procesados, cada uno con su texto, embedding, número de página de origen y orden indexado.

Este objeto estructurado se envía al microservicio de base de datos (neo4j). La base de datos es responsable de indexar los embeddings para la búsqueda rápida por similitud.

Agente de extracción de imágenes

Este agente permite identificar, extraer y normalizar las imágenes extraídas directamente desde el PDF o el archivo original.

Extracción estructural: obtiene todos los objetos gráficos que no pertenecen al texto plano, evitando interpretaciones heurísticas o conversiones prematuras. Esto asegura que diagramas, fórmulas renderizadas como imágenes y gráficos empotrados sean recuperados sin pérdida de datos.

Normalización y registro en ms-db: una vez extraídas, las imágenes se envían al microservicio de la base de datos, donde quedan almacenadas con identificadores persistentes. Estos identificadores permiten que otros agentes, por ejemplo, módulos OCR multimodales, puedan acceder a cada imagen en forma independiente, procesarla, interpretarla o vincularla con texto.

Agente de Recuperación

El Retriever Agent es el componente que permite que el sistema "encuentre" información relevante cuando el usuario realiza una consulta. Funciona como un motor de recuperación semántica construido sobre los embeddings generados en la etapa anterior.

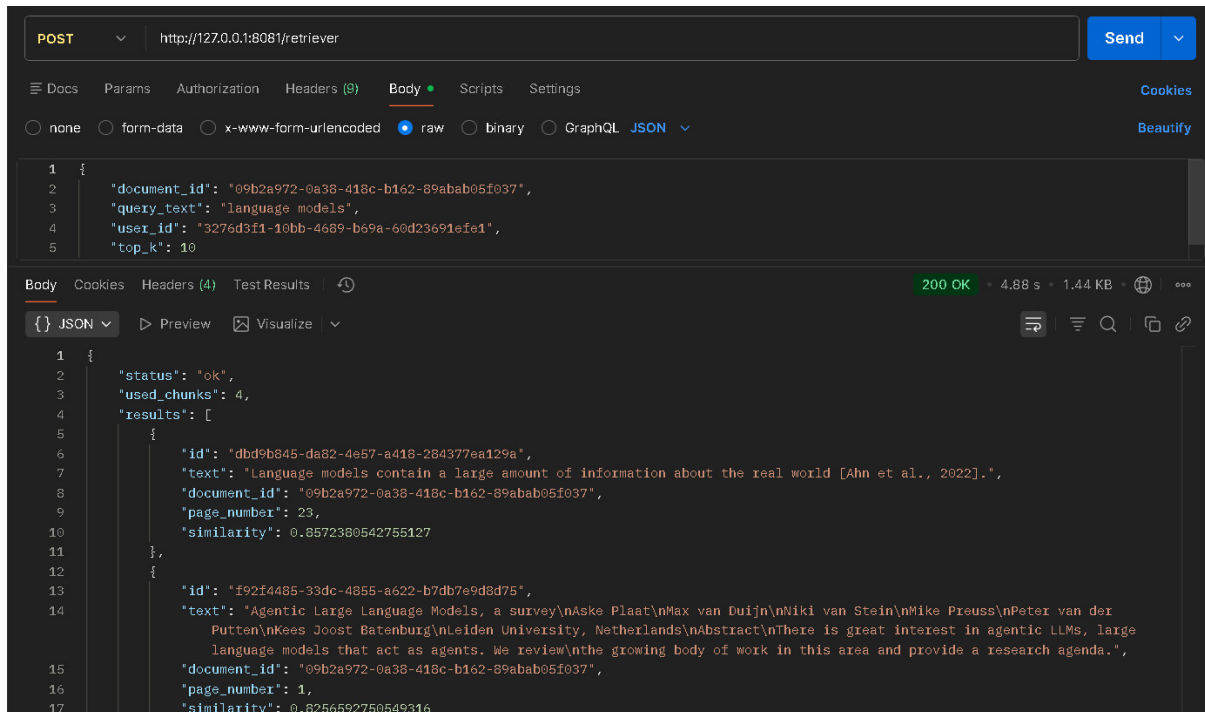
Su operación se divide en tres pasos:

Procesamiento de la consulta: toma el texto de la pregunta del usuario y genera su embedding correspondiente, usando el mismo modelo que se aplicó a los documentos.

Búsqueda en ms-db: solicita al microservicio una operación de búsqueda basada en distancias vectoriales. En lugar de buscar coincidencias textuales exactas, identifica los textos recuperados más próximos semánticamente a la intención del usuario.

Selección y retorno de evidencia: devuelve los segmentos más relevantes según una cantidad fija, priorizando aquellos que pueden servir como evidencia sólida para tareas posteriores (resumen, respuesta asistida, verificación, etc.).

Este agente no genera texto ni interpreta contenidos, la única función que tiene es recuperar el contexto adecuado, lo que convierte al pipeline en un sistema que responde basándose en información real contenida por el usuario en lugar de depender únicamente de un modelo generativo.



```
POST http://127.0.0.1:8081/retriever
Body
raw
1 {
2   "document_id": "09b2a972-0a38-418c-b162-89abab05f037",
3   "query_text": "language models",
4   "user_id": "3276d3f1-10bb-4689-b69a-60d23691eFe1",
5   "top_k": 10
}

Body
JSON
1 {
2   "status": "ok",
3   "used_chunks": 4,
4   "results": [
5     {
6       "id": "dbd9b846-da82-4e57-a418-284377ea129a",
7       "text": "Language models contain a large amount of information about the real world [Ahn et al., 2022].",
8       "document_id": "09b2a972-0a38-418c-b162-89abab05f037",
9       "page_number": 23,
10      "similarity": 0.8572380542755127
11    },
12    {
13      "id": "f92f4485-33dc-4855-a622-b7db7e9d8d75",
14      "text": "Agentic Large Language Models, a survey\nAske Plaat\nMax van Duijn\nNiki van Stein\nMike Preuss\nPeter van der Putten\nKees Joost Batenburg\nLeiden University, Netherlands\nAbstract\nThere is great interest in agentic LLMs, large language models that act as agents. We review\nthe growing body of work in this area and provide a research agenda.",
15      "document_id": "09b2a972-0a38-418c-b162-89abab05f037",
16      "page_number": 1,
17      "similarity": 0.8256592750549316
18    }
19  ]
20 }
```

Ilustración 17 - Ejemplo de la devolución de datos del agente Retriever

Agente de resumen

El Summarize Agent es el módulo del pipeline que produce transformaciones semánticas complejas y que interactúa directamente con las consultas del usuario. Su misión es sintetizar información proveniente de los documentos y presentarla de forma clara, cohesiva y centrada en la necesidad planteada.

El flujo interno del agente incluye varias etapas:

Obtención de evidencia mediante el Retriever Agent: En lugar de leer documentos completos, solicita únicamente los chunks con mayor relevancia para la pregunta del usuario.

Construcción de un cuerpo coherente: Combina los fragmentos seleccionados, resolviendo repeticiones, ruidos o discontinuidades producto de la segmentación original. Esta etapa garantiza que el LLM trabaje sobre un texto bien formado.

Síntesis mediante un modelo LLM: Utiliza un modelo generativo (por ejemplo, Ollama) para producir un resumen o explicación que capture lo esencial del material. El resultado no es una copia condensada, sino una reconstrucción semántica basada en evidencia.

Registro en ms-db: el resultado final se guarda como un nuevo mensaje dentro del chat del usuario, asegurando trazabilidad del razonamiento y manteniendo coherencia con el historial conversacional del sistema.

Este agente es donde se produce el salto desde datos fragmentados hacia una explicación comprensible, contextual y útil.

Agente de resumen más información OCR

Este agente tiene como función principal integrar diversas fuentes de texto (la información de los textos recuperados, y el texto extraído por OCR de una imagen) para luego generar un resumen final utilizando el agente de summarize.

Agente de búsqueda de OCR

Este agente se encarga de traer la primera página (siendo que se recuperan con el agente de retrieve, siendo la más relevante la que tenga un valor más cercano a la búsqueda en concreto) que tenga una imagen asociada en la base de datos de microservicios, y luego devolver esa imagen y el número de página.

Administrador de modelos

Debido a que los modelos locales consumen una gran cantidad de RAM, se necesitaba de una forma para poder cargar modelos de manera que siempre estén activos únicamente cuando se los llamen, para esto se creó el Model manager que es un administrador de recursos, que permite seleccionar dinámicamente que motor de IA, se utiliza según la solicitud.

Ocr baseline

El enfoque de estos tipos de modelos se basa en analizar la imagen en términos puramente gráficos: identifica contrastes, bordes y patrones geométricos que coinciden con formas predefinidas de caracteres.

El flujo es el mismo para todos los componentes de este estilo (baseline ocr, paddle baseline ocr, trocr baseline ocr), las imágenes del documento se recuperan desde el microservicio de la base de datos, una vez en memoria, el tratamiento es sencillo: cada imagen se decodifica, se abre y se entrega a un motor que intenta reconocer letras, números o símbolos presentes en ella siguiendo reglas fijas de detección.

No hay comprensión del contexto, ni análisis del diseño de la página, ni capacidad de interpretar elementos gráficos como tablas o bloques de texto reorganizados.

Para compensar la falta de inteligencia visual, la estrategia del baseline se apoya en la eficiencia. El trabajo se divide entre los núcleos de la CPU mediante un esquema de paralelismo clásico basado en hilos. Esto permite que muchas imágenes se procesen simultáneamente, obteniendo resultados rápidos y constantes.

Aunque la calidad del texto puede ser inferior en documentos complejos o de mala calidad, el rendimiento suele ser bastante alto en casos donde: las páginas no tienen ruido (errores o corrupción de datos), impresiones nítidas, PDFs digitales o material con poca variación visual.

Una vez que cada página ha sido procesada, el backend envía los resultados al microservicio de base de datos.

Ocr multimodal

Los modelos de OCR multimodal son componentes que no utilizan un motor de OCR clásico, sino un modelo de inteligencia artificial.

En el OCR multimodal, la extracción de texto no surge de un mecanismo que busque patrones de letras directamente en la imagen, sino de un proceso más complejo en el que el modelo primero interpreta la escena visual completa y luego produce una respuesta textual basada en lo que “comprende”.

La imagen no se analiza como un simple conjunto de píxeles para detectar formas de letras; en cambio, los modelos multimodales convierten la imagen en representaciones internas de alto nivel donde identifican objetos, estructuras, relaciones espaciales y fragmentos de texto presentes en el contenido visual.

Una vez procesada la imagen, el modelo utiliza su componente lingüístico para generar una descripción en lenguaje natural que corresponde al texto que detecta en la escena. Es una extracción que mezcla percepción visual con generación de lenguaje: el modelo “ve” lo que hay en la imagen, pero también “entiende” qué tipo de información es, cómo está organizada y qué segmentos de esa escena corresponden a texto significativo. Por eso puede recuperar texto incluso cuando está inclinado, deformado, embebido en señales visuales irregulares o mezclado con diagramas.

La lógica central no cambia, aunque el modelo subyacente sea Ollama, Florence, MiniCPM o Gemini; para el microservicio de procesamiento y clasificación todos ellos se comportan como una unidad capaz de recibir una imagen, interpretar su contenido mediante un proceso de razonamiento visual y devolver una descripción textual del texto que haya identificado.

Como estos modelos consumen muchos recursos, la ejecución nunca es masiva ni totalmente paralela: se limita la cantidad de imágenes procesadas simultáneamente para no saturar la GPU (o en el caso de este sistema CPU). Esto convierte al flujo en un recorrido controlado, donde cada lote de páginas se envía al motor multimodal con una concurrencia regulada. Mientras el sistema avanza, cada resultado viaja de regreso al microservicio de base de datos, acompañado de información relevante como el motor utilizado, la versión del modelo y el número de página, lo que permite reconstruir luego el proceso completo con claridad y trazabilidad.

Criterio de selección entre tipos de modelos en el sistema

La selección entre el OCR baseline y el OCR multimodal se basa principalmente en el costo computacional y en la necesidad real de precisión extrema.

En el sistema, el OCR baseline se utiliza como opción predeterminada en prácticamente todos los casos, porque es mucho más ligero, procesa las páginas con gran velocidad y ofrece resultados suficientemente buenos cuando los documentos son digitales, están bien escaneados o tienen texto claro y sin complicaciones.

Por otro lado, el OCR multimodal solo se utiliza en contadas ocasiones, generalmente una sola vez por documento, cuando el baseline devuelve un nivel de confianza muy bajo o cuando se detecta que el documento pertenece a categorías especialmente difíciles (fotografías de documentos, texto sobre imágenes, manuscritos, páginas muy ruidosas o con diseños complejos). De esta forma se aprovecha la alta capacidad de comprensión visual del modelo multimodal únicamente cuando es estrictamente necesario, reservando los recursos pesados para esos casos excepcionales.

Microservicio de backend

Este es el microservicio principal en donde se encuentran las responsabilidades de orquestación predeterminísticas y dinámicas teniendo los módulos principales del sistema, los cuales sirven para la utilización de las rutinas creadas en el microservicio de procesamiento y clasificación, en el backend existen 3 módulos, el de usuario, que se encarga de crear, verificar y permitir iniciar sesión.

Para que cada usuario pueda manejar su propia colección de documentos, se utiliza el módulo de upload que se encarga de la subida del documento, su viabilidad, la extracción de datos, incrustación de vectores en los segmentos del documento, así como también de la extracción de las imágenes y crear metadatos para el mismo.

Por último, existe el módulo que se encarga de usar dinámicamente las herramientas restantes, el módulo de orquestación del chat que es el que se encarga de construir el grafo de langgraph, dentro del mismo existe un agente que actúa como enrutador mediante el chat que permite según la intención del usuario definir que herramienta va a utilizar

Arquitectura interna

Componentes clave:

- **LangGraph Orchestrator:** coordina el flujo de conversación y las herramientas.
- **Tool Registry:** registra dinámicamente las herramientas disponibles (`retriever_tool`, `summarizer_tool`, `get_user_documents`, `ocr_summarize_tool`, `ocr_fetch_image_tool`).
- **Adapters:** encapsulan la interacción con demás microservicios.
- **Controller / Service:** manejan las peticiones REST y gestionan el streaming de respuestas.

Módulo de orquestación del chat

Este es el módulo principal del sistema, por el cual se cruzan todas las interacciones que se crearon con este y otros microservicios, las interacciones entre usuario y modelo, almacenamiento de datos, mecanismos de recuperación de documentos, entre otros.

Su función es coordinar, regular y resolver la comunicación entre un LLM que opera mediante un grafo de razonamiento (LangGraph) y un ecosistema externo compuesto por microservicios, herramientas semánticas y pipelines documentales.

Todo el flujo conversacional y los procesos de adquisición, búsqueda y síntesis de información se canalizan a través de esta pieza, por lo que su diseño presenta características arquitectónicas más estrictas que el resto del sistema.

El flujo del sistema combina:

- **Orchestration Pattern:** El orquestador controla el flujo y dicta el orden de las operaciones, incluidos los llamados a las herramientas, la carga del historial, construcción de los prompts y la recuperación de documentos indexados.
- **Event-Driven Reasoning (LLM-driven):** Las decisiones no provienen de estados finitos codificados, sino que nodo a nodo, el modelo de lenguaje grande (LLM) razona sobre el estado global del sistema. Cada paso del grafo responde a la decisión del modelo.
- **Pipe-and-Filter:** El sistema procesa progresivamente el input del usuario a través de filtros y herramientas: búsqueda de documentos, extracción contextual, resúmenes, operaciones semánticas, etc.
- **Adapter Pattern:** Se emplean adaptadores para desacoplar el orquestador de los microservicios. Esto permite mantener interfaces estables aun cuando cambien los servicios externos.
- **Gateway Pattern:** aplicado al microservicio ms-db: Los adaptadores internos (como db adapter) funcionan como puerta de entrada única hacia el microservicio de persistencia.

Estructura interna del módulo y sus responsabilidades

El módulo de la orquestación conversacional se basa en los principios de LangGraph, siendo que cada nodo es una interacción, y cada interacción se representa mediante estados transferidos entre nodos, donde algunos nodos ejecutan lógica determinística y otros dependen del razonamiento del módulo de lenguaje grande en uso.

Esta arquitectura posibilita una conversación entre usuario y asistente, en la cual el asistente puede analizar la consulta del usuario para poder definir los pasos a seguir.

Adaptadores

Uno de los pilares del orquestador es la existencia de un adaptador de microservicios que permite mantener una separación estricta entre la lógica conversacional y el microservicio de base de datos (ms-db).

Este adaptador(db adapter) presenta una interfaz estable y bien definida para tareas como la creación de chats, el almacenamiento de mensajes, la recuperación del historial y el acceso a documentos del usuario. Además, el uso de un cliente HTTP asíncrono con manejo explícito de errores garantiza que las fallas de red no afecten el razonamiento del grafo.



```
tool_registry.py M db_adapter.py X
app > orchestrator_chat > adapters > db_adapter.py > get_user_documents_from_db
1 import httpx
2
3 BASE_MS_DB = "http://127.0.0.1:8080"
4
5 async def create_chat_in_db(user_id: str, title: str = "Nueva conversación (Orchestrator)":
6     async with httpx.AsyncClient() as client:
7         r = await client.post(f"{BASE_MS_DB}/api/chats/", json={"user_id": user_id, "title": title})
8         r.raise_for_status()
9         return r.json()
10
11 async def save_message_in_db(payload: dict):
12     async with httpx.AsyncClient() as client:
13         r = await client.post(f"{BASE_MS_DB}/api/messages/", json=payload)
14         r.raise_for_status()
15         return r.json()
16
17 async def get_messages_from_db(chat_id: str):
18     async with httpx.AsyncClient() as client:
19         r = await client.get(f"{BASE_MS_DB}/api/messages/{chat_id}")
20         r.raise_for_status()
21         return r.json()
22
23
```

Ilustración 18 – adaptador para la base de datos

Tools

El orquestador necesita ofrecer al modelo un conjunto de herramientas disponible durante la ejecución, para hacer esto se incorpora al sistema un registro centralizado (registry_tool), este

archivo permite centralizar las referencias de todas las funciones decoradas como herramientas(`@tools`).

En términos arquitectónicos, este registro opera como un patron Service Locator, suficiente para que LangGraph materialice acciones externas sin acoplar la lógica del grafo a implementaciones específicas.

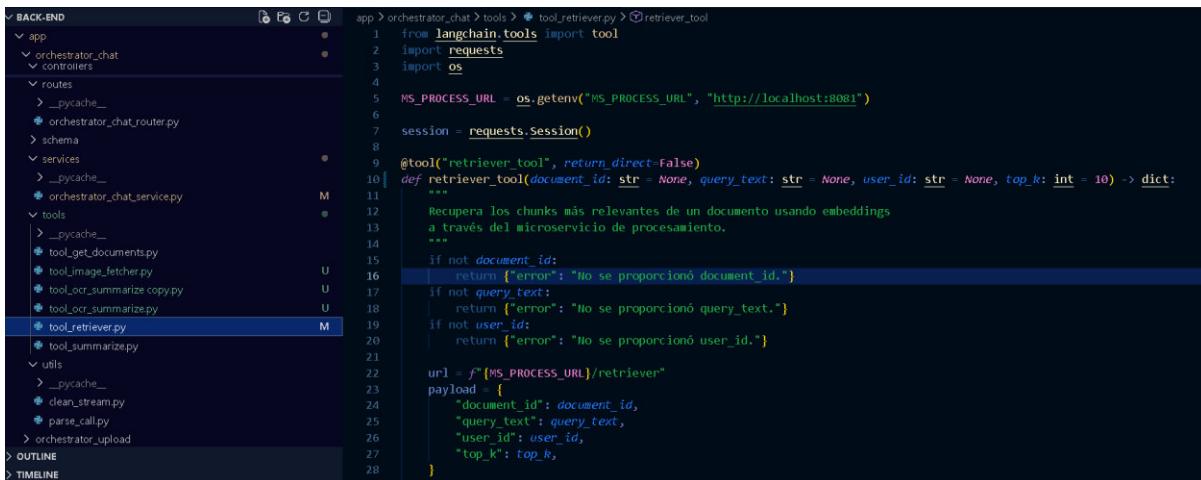
```
app > orchestrator_chat > adapters > tool_registry.py > get_tool_descriptions
1 from app.orchestrator_chat.tools.tool_get_documents import get_user_documents
2 from app.orchestrator_chat.tools.tool_retriever import retriever_tool
3 from app.orchestrator_chat.tools.tool_summarize import summarizer_tool
4 from app.orchestrator_chat.tools.tool_ocr_summarize import ocr_summarize_tool
5 from app.orchestrator_chat.tools.tool_image_fetcher import ocr_fetch_image_tool
6
7 TOOLS = [
8     get_user_documents,
9     retriever_tool,
10    summarizer_tool,
11    ocr_summarize_tool,
12    ocr_fetch_image_tool
13 ]
14
15
16 def get_registered_tools():
17     """
18     Devuelve la lista de herramientas registradas (objetos @tool)
19     para ser usadas dinámicamente por el LangGraph Orchestrator.
20     """
21     return TOOLS
22
23
24 def get_tool_names():
25     """
26     Devuelve solo los nombres de las herramientas registradas.
27     Útil para debugging o introspección del sistema.
28     """
29     return [getattr(t, "name", getattr(t, "__name__", "unknown_tool")) for t in TOOLS]
```

Ilustración 19 – Registro centralizado

Herramientas

Las herramientas que dispone el orquestador al momento de actuar son:

- `get_user_documents`: Extrae la lista de documentos del usuario desde la base de datos
- `retriever_tool`: Ejecuta la búsqueda vectorial usando el índice del usuario.
- `summarizer_tool`: Realiza síntesis a partir del contenido de chunks recuperados.
- `ocr_summarize_tool`: Realiza síntesis a partir del contenido de chunks recuperados + la información de la imagen obtenida por la tool de `ocr_fetch_image`.
- `ocr_fetch_image_tool`: Recupera imágenes a partir de la información traída por la herramienta de `retriever`.



```

1 from langchain.tools import tool
2 import requests
3 import os
4
5 MS_PROCESS_URL = os.getenv("MS_PROCESS_URL", "http://localhost:8081")
6
7 session = requests.Session()
8
9 @tool("retriever_tool", return_direct=False)
10 def retriever_tool(document_id: str = None, query_text: str = None, user_id: str = None, top_k: int = 10) -> dict:
11     """
12     Recupera los chunks más relevantes de un documento usando embeddings
13     a través del microservicio de procesamiento.
14     """
15     if not document_id:
16         return {"error": "No se proporcionó document id."}
17     if not query_text:
18         return {"error": "No se proporcionó query_text."}
19     if not user_id:
20         return {"error": "No se proporcionó user_id."}
21
22     url = f"{MS_PROCESS_URL}/retriever"
23     payload = {
24         "document_id": document_id,
25         "query_text": query_text,
26         "user_id": user_id,
27         "top_k": top_k,
28     }

```

Ilustración 20 - Ejemplo de herramienta - tool_retriever

Memoria de las conversaciones

Para la memoria del sistema no se utilizan mecanismos internos del grafo, sino que se utiliza el microservicio ms-db, Gracias a eso el historial es persistente, multi-sesion y evita depender de buffers que se limpian entre sesiones.

Estado del orquestador como eje

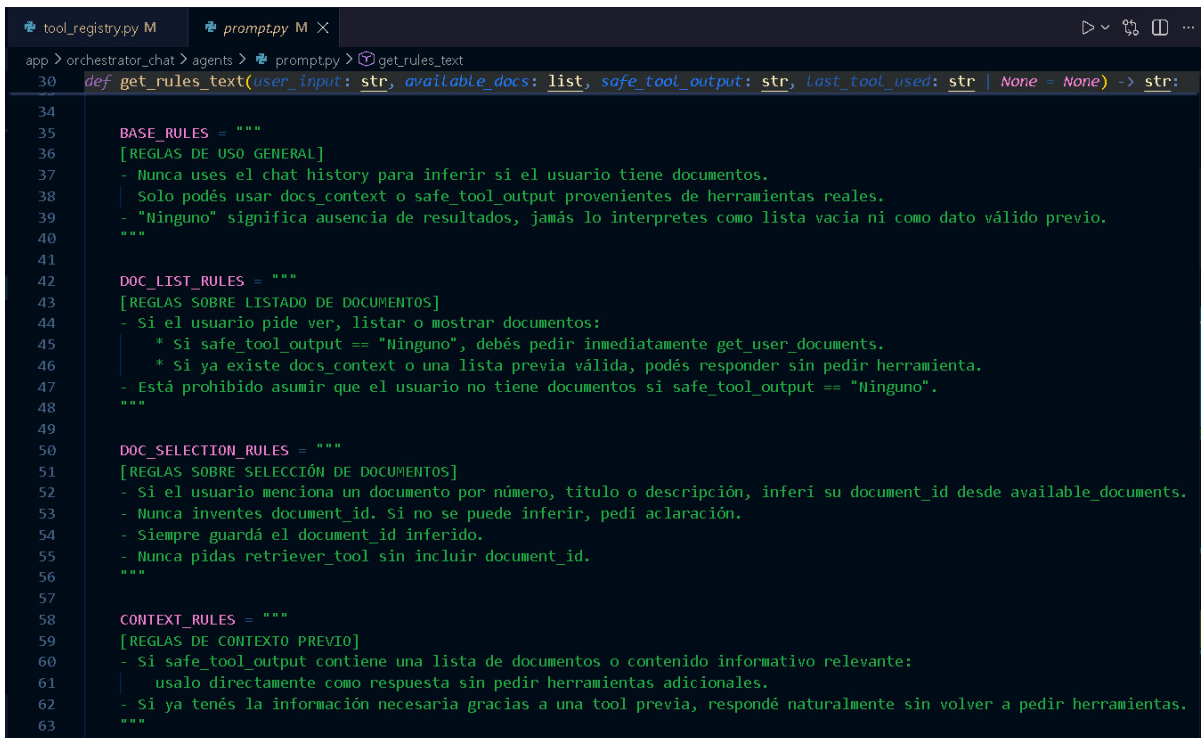
El estado del orquestador constituye la parte más crítica del sistema, en él confluye la entrada del usuario, el historial recuperado, los resultados de la herramienta previas y los documentos que previamente cargo el usuario.

El grafo se compone de nodos con los estados necesarios para la resolución de la siguiente acción, cada nodo del grafo recibe esos estados, lo transforman y devuelven una nueva versión. Este mecanismo evita cualquier pérdida de información y permite un razonamiento progresivo, ya que el modelo dispone en cada paso de un panorama completo de lo que ha ocurrido hasta el momento

Razonamiento basado en el modelo

El modelo de razonamiento se basa en la construcción de un prompt jerárquico que combina información sobre las herramientas registradas, reglas contextuales que se generan dinámicamente, según la salida previa de herramientas, documentos disponibles del usuario, historial completo.

Este ensamblado, proporciona al modelo un marco teórico suficiente para poder decidir la siguiente acción, si debe invocar una herramienta, cuál es la más adecuada y con qué parámetros debe realizarla.



```

30 def get_rules_text(user_input: str, available_docs: list, safe_tool_output: str, last_tool_used: str | None = None) -> str:
34
35     BASE_RULES = """
36     [REGLAS DE USO GENERAL]
37     - Nunca uses el chat history para inferir si el usuario tiene documentos.
38     | Solo podés usar docs_context o safe_tool_output provenientes de herramientas reales.
39     - "Ninguno" significa ausencia de resultados, jamás lo interpretes como lista vacía ni como dato válido previo.
40     """
41
42     DOC_LIST_RULES = """
43     [REGLAS SOBRE LISTADO DE DOCUMENTOS]
44     - Si el usuario pide ver, listar o mostrar documentos:
45     | * Si safe_tool_output == "Ninguno", debés pedir inmediatamente get_user_documents.
46     | * Si ya existe docs_context o una lista previa válida, podés responder sin pedir herramienta.
47     - Está prohibido asumir que el usuario no tiene documentos si safe_tool_output == "Ninguno".
48     """
49
50     DOC_SELECTION_RULES = """
51     [REGLAS SOBRE SELECCIÓN DE DOCUMENTOS]
52     - Si el usuario menciona un documento por número, título o descripción, inferí su document_id desde available_documents.
53     - Nunca inventes document_id. Si no se puede inferir, pedí aclaración.
54     - Siempre guardá el document_id inferido.
55     - Nunca pidas retriever_tool sin incluir document_id.
56     """
57
58     CONTEXT_RULES = """
59     [REGLAS DE CONTEXTO PREVIO]
60     - Si safe_tool_output contiene una lista de documentos o contenido informativo relevante:
61     | usalo directamente como respuesta sin pedir herramientas adicionales.
62     - Si ya tenés la información necesaria gracias a una tool previa, respondé naturalmente sin volver a pedir herramientas.
63     """

```

Ilustración 21 - Ejemplo de prompts usados en el sistema

El proceso continúa con la interacción con el modelo, donde se consideran tanto respuestas síncronas (elementos guardados en la base de datos) como asíncronas (razonamiento del modelo) y se aplica un mecanismo de limpieza al texto generado para resolver ruido o elementos no deseados.

Finalmente, la actualización del estado integra la herramienta seleccionada, los parámetros derivados del razonamiento y los resultados de operaciones anteriores.

Implementación del Sistema RAG

El orquestador implementa un esquema RAG explícito mediante las herramientas que se encuentran en la carpeta tools. La primera de ellas, “get_documents” es la encargada de recuperar la lista de documentos del usuario, funciona como la percepción del grafo, ya que le permite determinar qué información tiene disponible el usuario, La segunda herramienta, “retriever” dedicada a la recuperación vectorial, constituye el núcleo del componente semántico del RAG, extrayendo los fragmentos relevantes según la consulta del usuario.

La tercera herramienta, “summarize” produce la síntesis final de la información recuperada, generando resúmenes, explicaciones y reconstrucciones de contenido extenso.

Este esquema RAG no está embebido en el prompt del modelo, sino que se ejecuta como una secuencia transparente de operaciones: primero recuperación, luego razonamiento y finalmente síntesis.

Integración general con langgraph

La integración con LangGraph convierte al orquestador en un grafo dinámico donde el modelo asume el papel de planificador. Cada herramienta corresponde a un nodo del grafo, cada paso del razonamiento permite reconsiderar el plan y cada actualización del estado deja explícito el avance del diálogo. Esta arquitectura evita flujos rígidos o predeterminados y habilita a las decisiones emergentes basadas en el contexto de la conversación entre usuario y asistente.

En el sistema se utilizan mecanismos de trazabilidad para poder seguir el flujo resultante entre nodos, además de auditar y reproducir el proceso en su totalidad, esto garantiza que todas las transiciones sean interpretables y verificables mediante logs, lo cual es esencial para un sistema donde el control del flujo está parcialmente delegado a un modelo estadístico.

Base de datos y los microservicios

El orquestador de langgraph se comunica con el microservicio de la base de datos a través del adaptador que se mencionó en el apartado de adaptadores (db adapter) lo que permite mantener un desacople fuerte entre lógica y persistencia. Gracias a esto, el microservicio puede escalar, migrar o modificarse sin afectar el funcionamiento del grafo.

Además, la independencia de implementación hace posible reutilizar el motor conversacional en distintos entornos, incluso fuera del contexto en el que fue concebido originalmente.

Flujo interno

- El usuario envía un mensaje desde el frontend.
- El backend llama al `orchestrator_chat`.
- El orquestador de langGraph evalúa la intención y selecciona la herramienta adecuada:
 - Si se solicita información → usa `tool_retriever`.
 - Si se pide resumen → usa `tool_retriever` y luego usa `tool_summarize`
 - Si el usuario pregunta por sus documentos → usa `tool_get_documents`.

- El resultado se pasa a Ollama para generar la respuesta final.
- El mensaje y la respuesta se guardan en ms-db.

Módulo de orkestracion del upload

Este módulo tiene como función tomar un archivo que el usuario decida, procesarlo en distintas etapas mediante agentes del microservicio de procesamiento y clasificación, construir una representación estructurada del documento que pueda utilizarse posteriormente en tareas de recuperación semántica.

Esta responsabilidad lo convierte en un **orquestador de ingesta de documentos**, cuya tarea es transformar documentos crudos en entidades analíticas completas, capaces de participar en flujos RAG.

Dicho proceso no se limita a la extracción de texto, sino que abarca la armonización de imágenes, metadatos, bloques textuales, páginas, chunks semánticos y cualquier componente necesario para permitir razonamiento sobre contenido complejo.

El grafo tiene 5 nodos, donde cada **nodo** representa una etapa del pipeline:

- upload → Envía el archivo al microservicio de procesamiento y clasificación.
- viability → valida el documento.
- extract → obtiene el texto y genera el id del documento.
- metadata → extrae los metadatos del documento.
- extract_images → extrae las imágenes asociadas del documento.

El módulo entonces se organiza en torno a estas series de capas bien definidas, cada una con responsabilidades específicas dentro del proceso de ingestión.

La primera de ellas es el controlador de **upload**, que representa el punto de entrada desde FastAPI. Esta capa recibe el documento, los datos del usuario y cualquier metadato o instrucción adicional que pueda modificar el flujo de procesamiento. Su única responsabilidad es validar la entrada y delegar la orquestación al servicio central del módulo.

Después se ejecuta la etapa de **viability**, que determina si el documento enviado es viable para su análisis. Este paso constituye un filtro temprano que evita el procesamiento de archivos corruptos, ilegibles o incompatibles con el pipeline documental. La verificación puede considerar aspectos como integridad estructural, formato, tamaño, legibilidad básica o coherencia interna. Desde el punto de vista arquitectónico, esta fase evita trabajos innecesarios en etapas posteriores y protege el sistema de fallos derivados de archivos inapropiados. La viabilidad no implica extraer contenido, sino confirmar que el documento puede ser tratado con garantías por las siguientes capas de procesamiento

Superada la validación, el pipeline avanza hacia la etapa de **extract**, uno de los momentos centrales del proceso. Aquí se obtiene el texto del documento y se genera el *doc id* asociado, que servirá como referencia estructural permanente en ms-db. La extracción se realiza sin alterar el contenido ni reorganizar su estructura lógica, respetando la disposición original del texto en el archivo. Este comportamiento es crucial para preservar la fidelidad documental y asegurar que el análisis posterior (como la segmentación o la indexación) opere sobre datos que reflejan con precisión el contenido original del usuario. El *doc id* generado en esta etapa se convierte en el identificador primario que enlaza todas las entidades relacionadas con el documento: metadatos, imágenes, páginas y futuros chunks.

La etapa siguiente corresponde a **metadata**, cuyo propósito es obtener y normalizar información descriptiva del documento. Los metadatos típicamente incluyen elementos como título, autor, fechas de creación, tipo de archivo, dimensiones y cualquier otro atributo inherente al contenido original. Esta fase permite que el documento adquiera una identidad más rica dentro del ecosistema de microservicios, facilitando búsquedas posteriores, validaciones internas y procesos de indexación.

Finalmente, el flujo ejecuta **extract_images**, que produce una representación explícita de todas las imágenes contenidas en el documento. Este paso recorre las distintas secciones del archivo y extrae cada imagen asociada, generando una colección identificable y persistente en ms-db. La separación física entre texto e imágenes es fundamental para habilitar capacidades multimodales en el futuro y para permitir que el sistema procese, analice o indexe recursos visuales de manera independiente. Además, esta etapa asegura que el documento ingestado no se limite al contenido textual y que pueda preservarse toda su riqueza informacional, manteniendo una fidelidad estructural completa respecto al original.

Cada nodo usa un **Adapter** que encapsula las llamadas HTTP al servicio externo.

Flujo interno

- El backend recibe un archivo PDF.
- Llama al `orchestrator_upload`, que ejecuta el grafo `upload` → `viability` → `extract` → `metadata` → `images`.
- Si todas las etapas son exitosas, retorna el `doc_id` y metadatos al backend.
- Si falla una etapa, lanza una excepción que se propaga con mensaje detallado.

Microservicio de front end

Para que los usuarios pudieran hacer uso del sistema se diseñó el Front end, el cual cumple el rol de la interfaz principal entre el usuario y el conjunto de microservicios. De esta forma se puede realizar la interacción entre usuario y asistente de manera que la conversación sea accesible al usuario, como se pensó en primera instancia que el proyecto fuese una web API, se decidió usar el framework `next.js`, la librería `react`, junto con lenguajes como `js`, `html` y `css`

La selección de tecnologías responde tanto a necesidades funcionales como a criterios de diseño arquitectónico.

React y **Next.js** se eligieron por su capacidad para construir interfaces altamente interactivas y actualizables en tiempo real, algo fundamental para este sistema, ya que el punto central era el chat que permite realizar una conversación progresiva entre usuario y modelo. Además, su organización basada en componentes y hooks lo que facilita la separación de responsabilidades, incrementa la modularidad del sistema y permite escalar la aplicación manteniendo claridad en el código. `Next.js` aporta capacidades de optimización integradas, enrutamiento simplificado y un rendimiento adecuado para aplicaciones con múltiples vistas.

La combinación de **JavaScript moderno**, **HTML** y **CSS modular** completa el entorno, garantizando compatibilidad con navegadores estándar y permitiendo un control preciso del diseño visual.

En conjunto, estas tecnologías brindan un equilibrio entre eficiencia, mantenibilidad y escalabilidad. Permiten construir una interfaz responsiva capaz de interactuar con un

ecosistema de microservicios mediante mecanismos avanzados como el streaming, al mismo tiempo que mantienen un diseño limpio y extensible.

Flujo interno

El frontend no contiene lógica de negocio ni de procesamiento; su responsabilidad se centra únicamente en representar la información en pantalla, gestionar la interacción del usuario y coordinar la comunicación con los microservicios backend.

Los elementos visuales se definen mediante componentes React, que renderizan el HTML estructurado y se aplican estilos mediante los módulos de CSS.

Los componentes controlan exclusivamente la presentación, mientras que la lógica se traslada a los hooks (diseñados en javascript)

Manejo de interacción y flujo interno mediante hooks personalizados

El frontend implementa un conjunto de hooks propios que encapsulan la lógica relacionada con mensajes, carga de archivos y gestión de estado. Estos hooks permiten que los componentes se mantengan enfocados en la presentación.

El manejo de mensajes, tanto enviados como recibidos, se organiza mediante un hook que administra el historial del chat, el estado del input y la creación de nuevos mensajes. Este flujo, se integra en JavaScript gracias a las APIs asíncronas del navegador.

La modularización mediante hooks permite generar un sistema flexible y organizado que separa las responsabilidades en capas lógicas.

El frontend se comunica con distintos microservicios implementados en **FastAPI** mediante solicitudes **HTTP convencionales**. Esta integración aprovecha la eficiencia de FastAPI, que maneja las solicitudes y respuestas de manera estándar.

Cuando un usuario envía un mensaje, el frontend lo transmite al **servicio orquestador**, el orquestador **espera a que el proceso completo del modelo finalice** antes de devolver la respuesta.

- El navegador procesa la **respuesta** una vez que esta ha sido recibida.

- La interfaz de usuario se actualiza **una sola vez** con el texto final.

Los microservicios encargados de manejo documental o análisis también se integran de forma directa. FastAPI acepta datos provenientes de **FormData**, lo que permite subir PDF u otros tipos de archivo sin pasos intermedios.

7.Resultados

Este apartado tiene objetivo demostrar la viabilidad y la funcionalidad del sistema de **Análisis Automatizado de Documentos Científicos**

Dado que el principal aporte de esta tesis reside en la integración metodológica de la Arquitectura de Microservicios con un Sistema Multi-Agente (MAS) orquestado por LangGraph, esta sección se enfoca en la **validación operativa** de dicha arquitectura.

Se presentará evidencia detallada de que:

- El sistema **ejecuta con éxito** el ciclo de vida completo del análisis de un documento científico.
- Los **cuatro microservicios** principales están **correctamente desacoplados** y la comunicación entre ellos es funcional.
- La **lógica de orquestación de LangGraph** se traduce eficazmente en la invocación de las Rutinas del sistema.

La presentación se basa en la **evidencia visual (Figuras)** que documentan la implementación y el flujo de trabajo.

Flujo de trabajo - Ciclo Operativo Global

El flujo macro del sistema se puede resumir en el siguiente esquema.

- Un usuario inicia sesión
- Carga un documento mediante una web
- Se analiza, si pasa las revisiones, se guarda para posterior uso
- El texto es segmentado y se incrusta la vectorización en cada segmento
- Se extraen las imágenes
- El usuario inicia la conversación con el asistente, genera consulta o habla con el modelo.
- El backend central inicializa el estado y delega al orquestador.
- El LangGraph ejecuta un ciclo de razonamiento iterativo.
- Si el LLM lo decide, se invoca una herramienta:

- listado de documentos
- recuperación documental
- resumen
- La herramienta se comunica con un microservicio externo.
- El microservicio procesa la solicitud y devuelve los datos estructurados esperados.
- El orquestador integra los resultados, contextualiza con la información de la memoria, la conversación, y genera la respuesta final.
- La respuesta es transmitida al usuario.
- Los mensajes de la conversación se persisten mediante el microservicio de la base de datos para posteriores usos.
- Se esperan nuevas consultas

Demostración del Flujo de Análisis (Frontend)

Se mostrará la interfaz de usuario, validando la interacción del sistema con el usuario y presentando la salida final del análisis estructurado por los agentes.

Validación de la Integración Inter-Servicio

Se documentará la comunicación a nivel de *backend*, ilustrando la secuencia de la llamada del Microservicio de Backend/LangGraph al Microservicio de procesamiento y clasificación, confirmando el correcto funcionamiento de las *tools* del sistema multi-agente.

Evidencia del Desacoplamiento Arquitectónico

Finalmente, se presentarán diagramas y capturas que confirman la independencia de los microservicios, verificando la implementación de la metodología de arquitectura distribuida.

Microservicio de front end

La vista de la página web se maneja en principio como una SPA, donde en principio solo cambia la interfaz del chat.

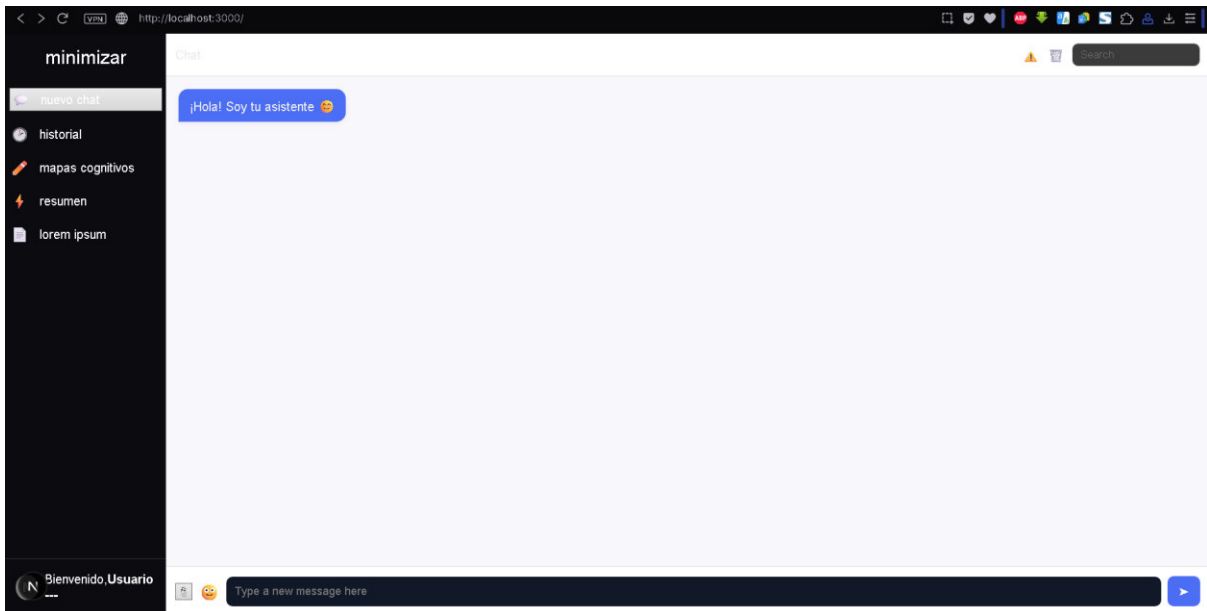


Ilustración 22 - Vista principal de la interfaz web

Vista de la carga de documentos

Dentro de la vista principal, se puede observar a la izquierda de la barra de búsqueda, un icono de clip, que es el encargado de poder cargar los documentos que el usuario desee.

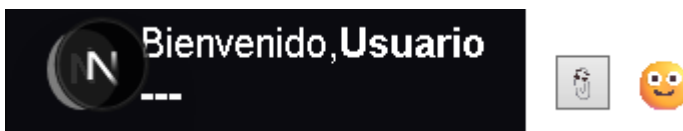


Ilustración 23 - vista del icono para cargar los documentos

Al pulsar el botón del icono del clip, saldrá una opción para poder cargar el documento desde la carpeta de descargas del usuario

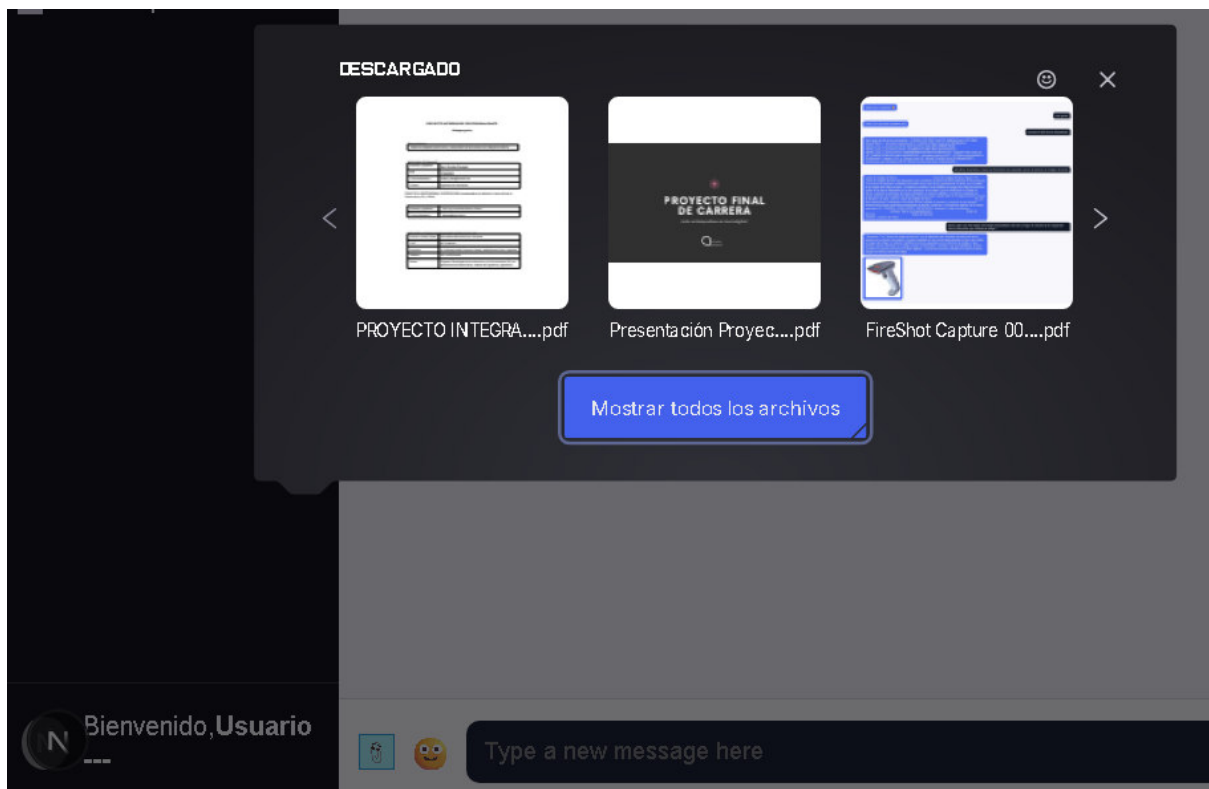


Ilustración 24 - carga de documentos en la interfaz web

Implementación de la carga de documentos en el next.js



Ilustración 25 - Implementación de la carga de documentos

Microservicio de Procesamiento y clasificación

Al hacer la carga del documento se desencadenará las acciones descritas en el backend

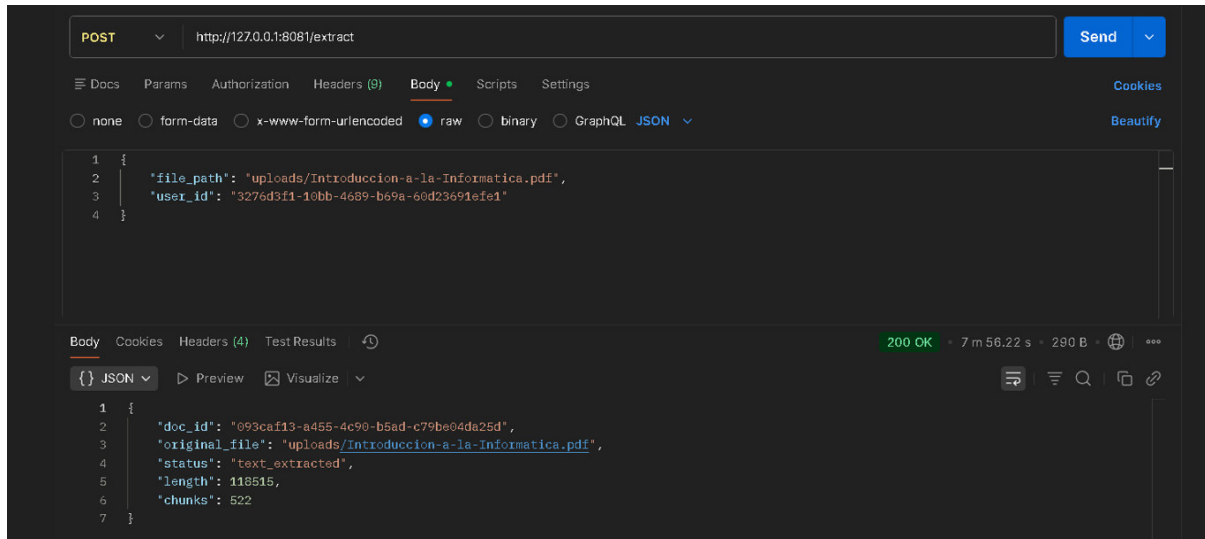


Ilustración 26 - validación de agente de extracción en el microservicio

Se extraerán del documento la información en trozos de textos y se le aplicara una incrustación vectorial a cada uno de ellos

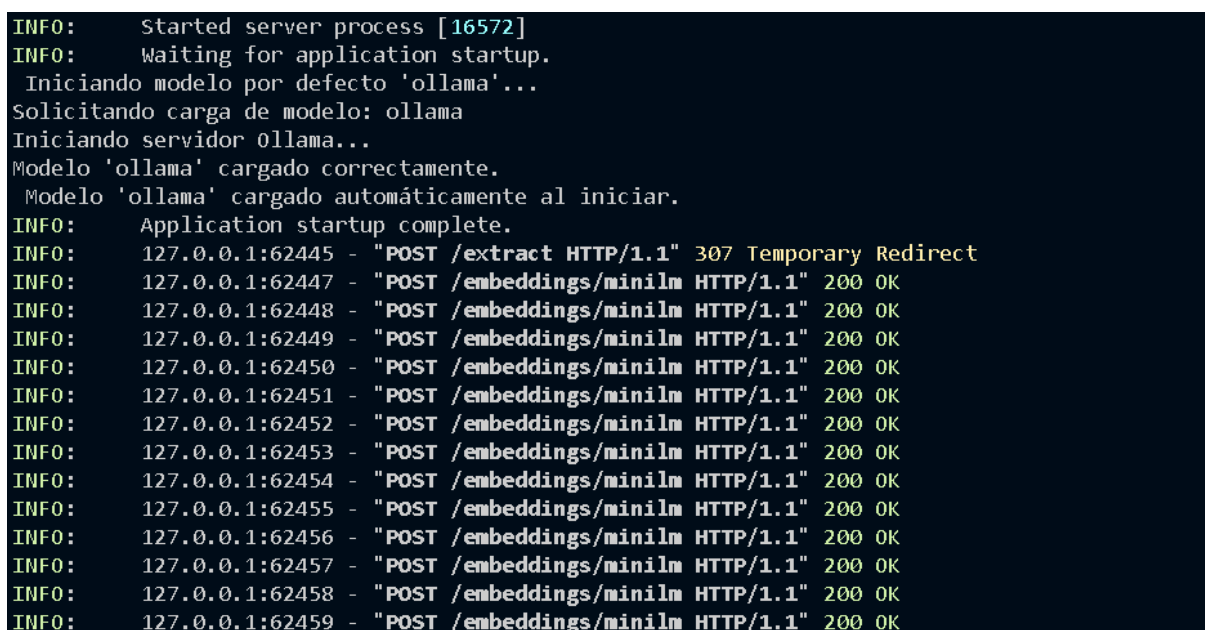


Ilustración 27 - Backlog del agente de extracción en el microservicio

Luego de esto se realizará la extracción de imágenes del documento

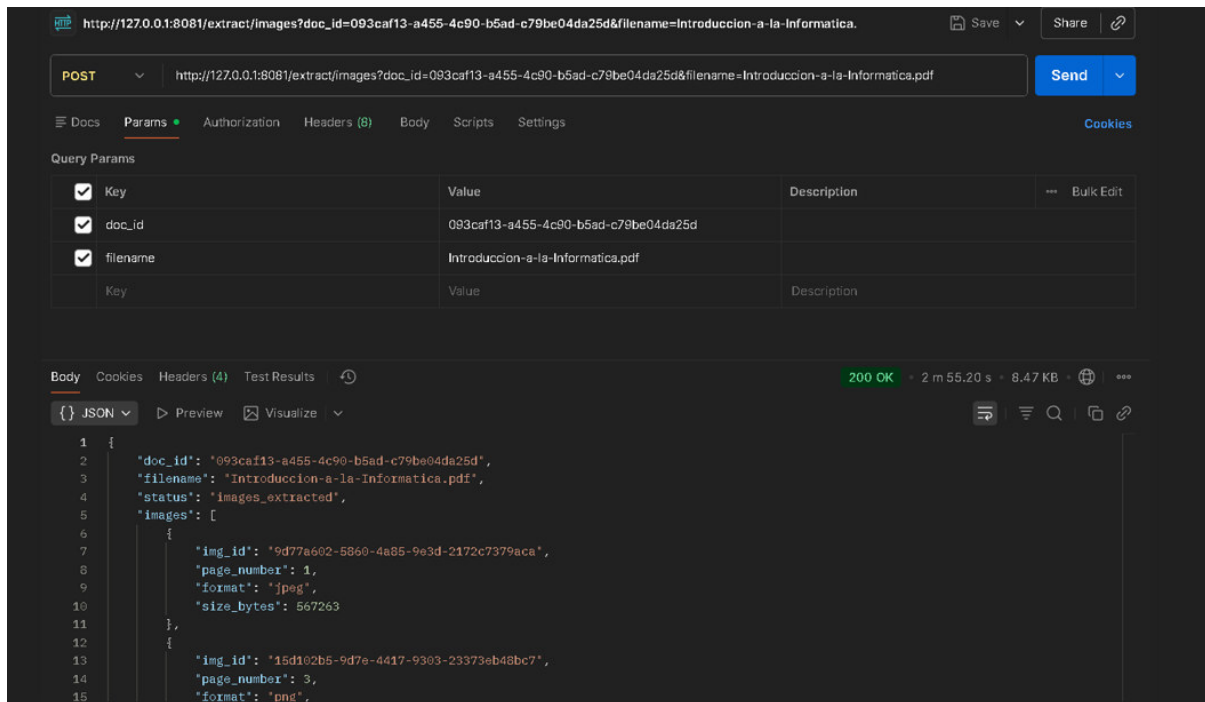


Ilustración 28 - validación de agente de extracción de imágenes

En el backlog se muestra cómo se guardan correctamente las imágenes

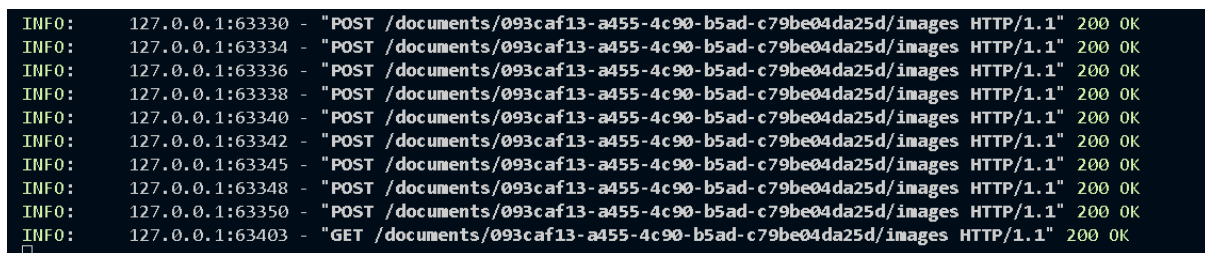


Ilustración 29 - backlog del agente de imagenes

Otro agente importante al igual que el agente de ocr baseline, es el agente de ocr multimodal, que dependiendo de quien sea el modelo que use para generar el acto multimodal (en este caso, debido a restricciones de memoria y capacidad computacional se usa un modelo pequeño de ollama, lo que hace que las respuestas generadas no sean tan eficientes)

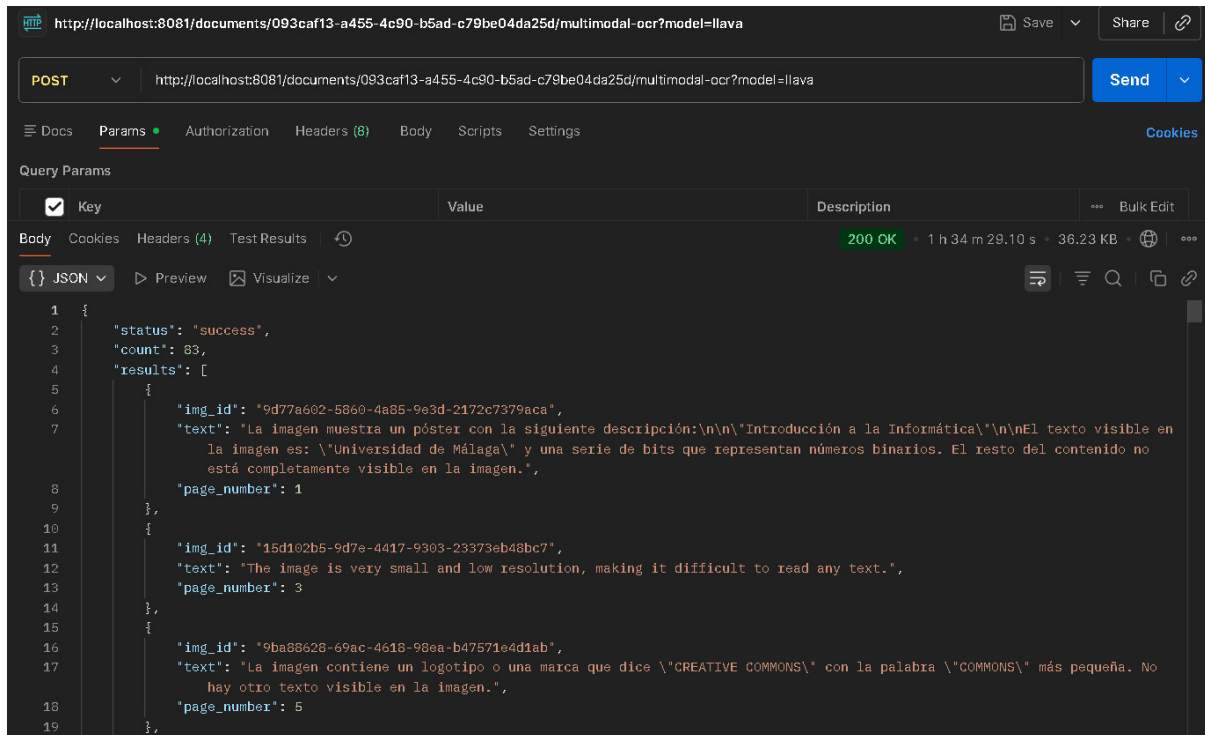


Ilustración 30 – Extracción de imágenes en Postman

Se procesan las imágenes de un documento en lotes con hilos y a medida que se realizan los lotes, se van guardando

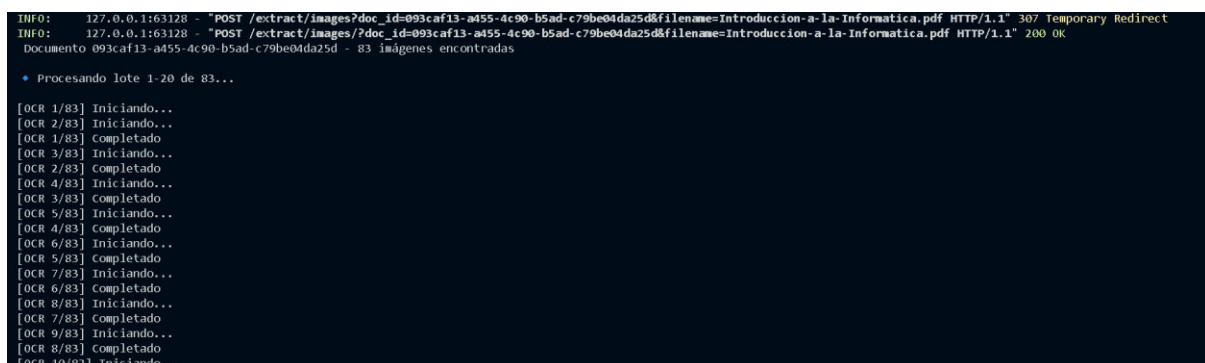
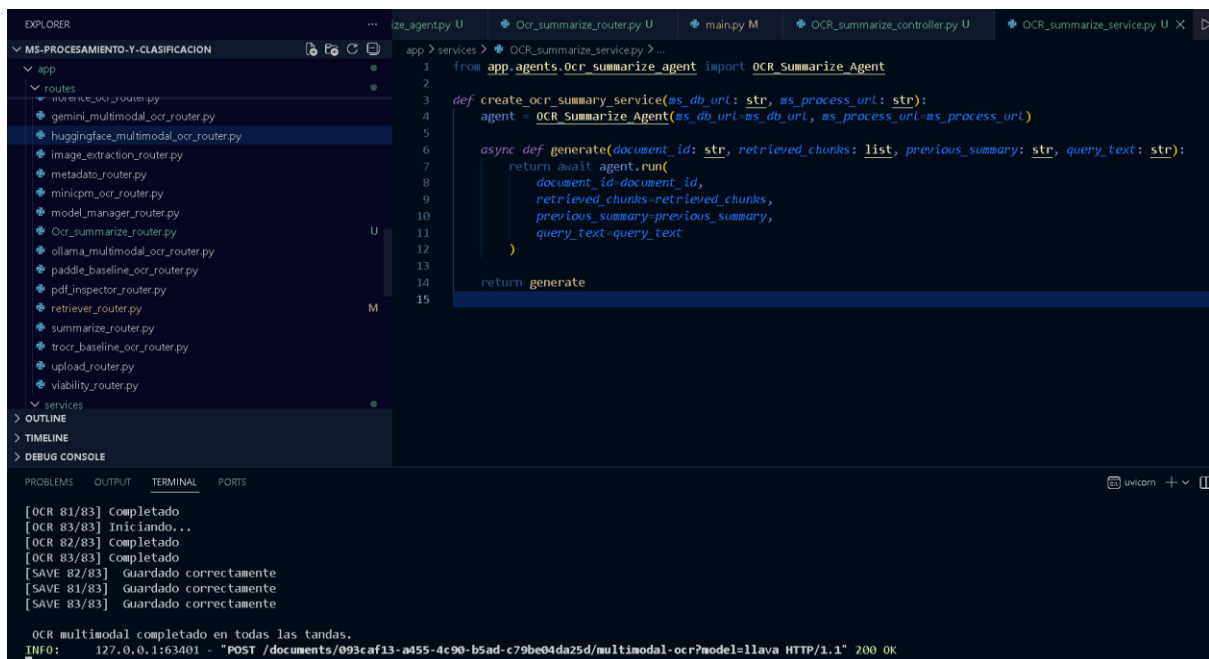


Ilustración 31 - inicio de lotes de las imágenes



```

1 from app.agents.ocr_summarize_agent import OCR_Summarize_Agent
2
3 def create_ocr_summary_service(ms_db_url: str, ms_process_url: str):
4     agent = OCR_Summarize_Agent(ms_db_url=ms_db_url, ms_process_url=ms_process_url)
5
6     async def generate(document_id: str, retrieved_chunks: list, previous_summary: str, query_text: str):
7         return await agent.run(
8             document_id=document_id,
9             retrieved_chunks=retrieved_chunks,
10            previous_summary=previous_summary,
11            query_text=query_text
12        )
13
14     return generate
15

```

```

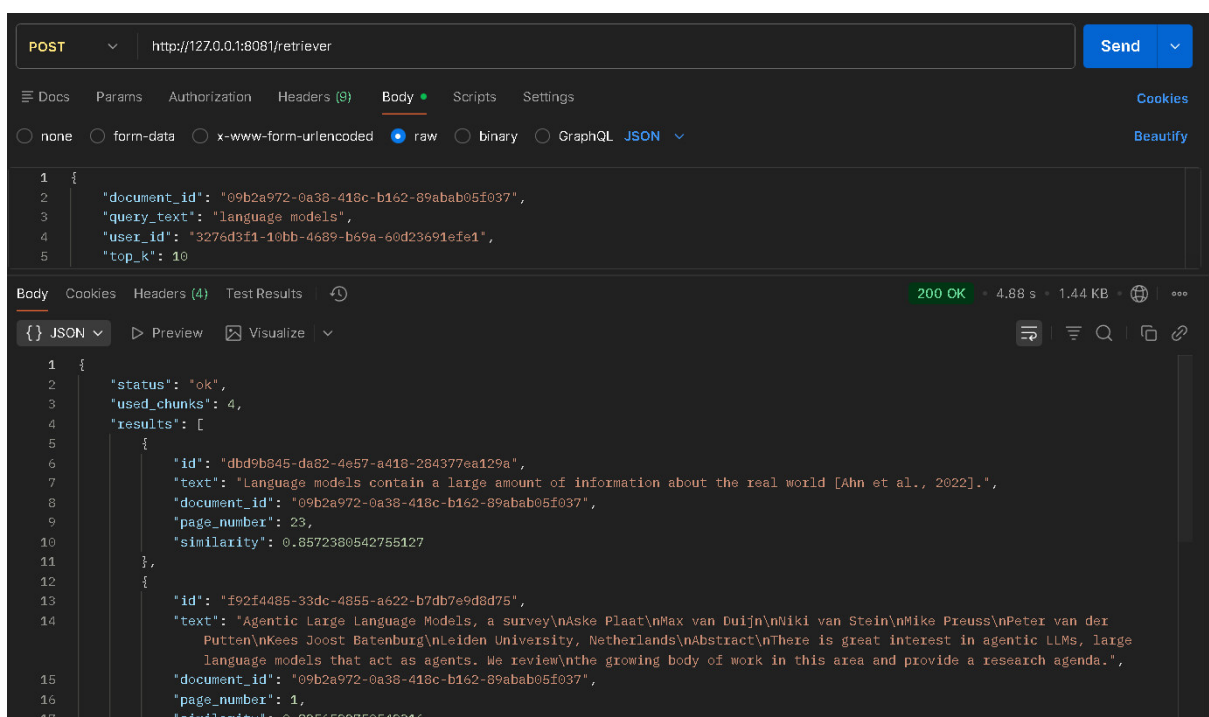
[OCR 81/83] Completado
[OCR 83/83] Iniciando...
[OCR 82/83] Completado
[OCR 83/83] Completado
[SAVE 82/83] Guardado correctamente
[SAVE 81/83] Guardado correctamente
[SAVE 83/83] Guardado correctamente

OCR multimodal completado en todas las tandas.
INFO: 127.0.0.1:63401 - "POST /documents/093caf13-a455-4c90-b5ad-c79be04da25d/multimodal-ocr?model=llava HTTP/1.1" 200 OK

```

Ilustración 32 - Terminación correcta del agente de extracción de imágenes

Una mención especial es para el agente de retriever que devuelve la información de los textos embebidos, en la imagen siguiente se puede ver no solo como es la recuperación sino también el dato de similarity que sirve para entender que tan cercano es el texto recuperado con respecto a la búsqueda.



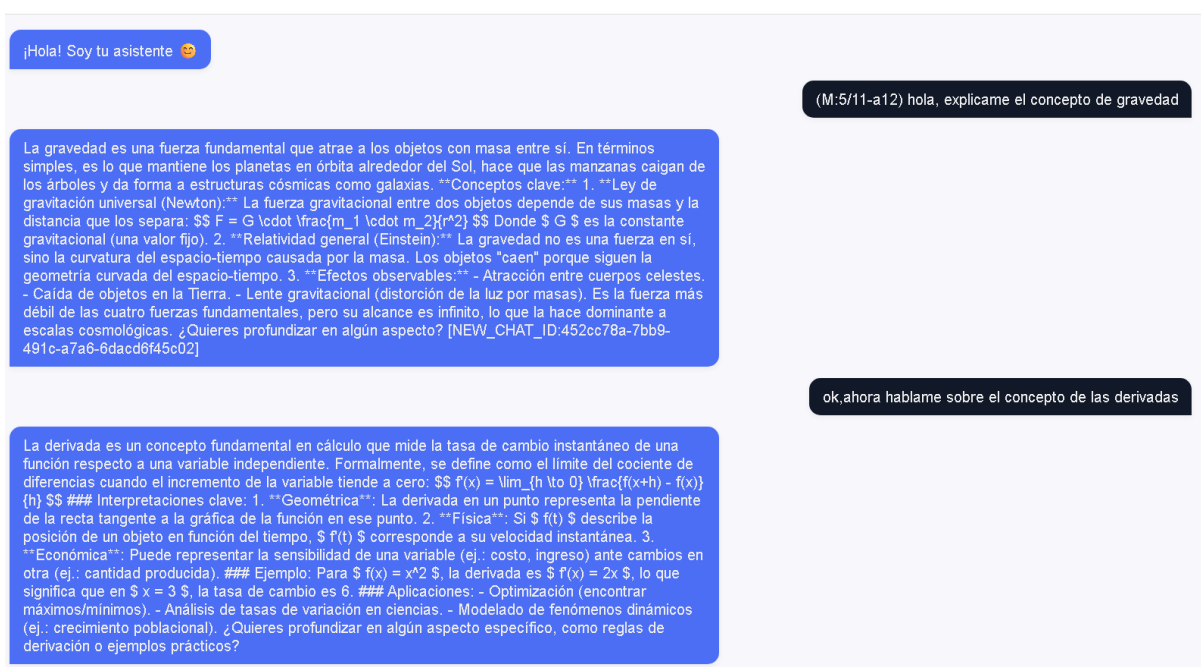
```

POST http://127.0.0.1:8081/retriever
Body
{
  "document_id": "09b2a972-0a38-418c-b162-89abab05f037",
  "query_text": "language models",
  "user_id": "3276d3f1-10bb-4689-b69a-60d23691efe1",
  "top_k": 10
}
200 OK - 4.88 s - 1.44 KB
Body
{
  "status": "ok",
  "used_chunks": 4,
  "results": [
    {
      "id": "dbd9b845-da82-4e57-a416-284377ea129a",
      "text": "Language models contain a large amount of information about the real world [Ahn et al., 2022].",
      "document_id": "09b2a972-0a38-418c-b162-89abab05f037",
      "page_number": 23,
      "similarity": 0.8572380542755127
    },
    {
      "id": "f92f4485-33dc-4855-a622-b7db7e9d8d75",
      "text": "Agentic large Language Models, a survey\nAske Plaat\nMax van Duijn\nNiki van Stein\nMike Preuss\nPeter van der Putten\nKees Joost Batenburg\nLeiden University, Netherlands\nAbstract\nThere is great interest in agentic LLMs, large language models that act as agents. We review\nthe growing body of work in this area and provide a research agenda.",
      "document_id": "09b2a972-0a38-418c-b162-89abab05f037",
      "page_number": 1,
      "similarity": 0.8256592750549316
    }
  ]
}

```

Finalmente, un ejemplo de una conversación con el modelo de LLM en una conversación sin herramientas.

El modelo de orquestador de langgraph y por lo tanto el conversacional es el de servicio de Google, mientras que, debido a las restricciones a peticiones con herramientas multimodales, se usa un modelo local para las (qwen3:8b), por eso las respuestas en el agente de ocr multimodal difieren tanto en calidad.



¡Hola! Soy tu asistente 😊

(M:5/11-a12) hola, explicame el concepto de gravedad

La gravedad es una fuerza fundamental que atrae a los objetos con masa entre sí. En términos simples, es lo que mantiene los planetas en órbita alrededor del Sol, hace que las manzanas caigan de los árboles y da forma a estructuras cósmicas como galaxias. **Conceptos clave:** 1. **Ley de gravitación universal (Newton):** La fuerza gravitacional entre dos objetos depende de sus masas y la distancia que los separa: $F = G \cdot \frac{m_1 \cdot m_2}{r^2}$ Donde G es la constante gravitacional (una valor fijo). 2. **Relatividad general (Einstein):** La gravedad no es una fuerza en sí, sino la curvatura del espacio-tiempo causada por la masa. Los objetos "caen" porque siguen la geometría curvada del espacio-tiempo. 3. **Efectos observables:** - Atracción entre cuerpos celestes. - Caída de objetos en la Tierra. - Lente gravitacional (distorción de la luz por masas). Es la fuerza más débil de las cuatro fuerzas fundamentales, pero su alcance es infinito, lo que la hace dominante a escalas cosmológicas. ¿Quieres profundizar en algún aspecto? [NEW_CHAT_ID:452cc78a-7bb9-491c-a7a6-6dacc6f45c02]

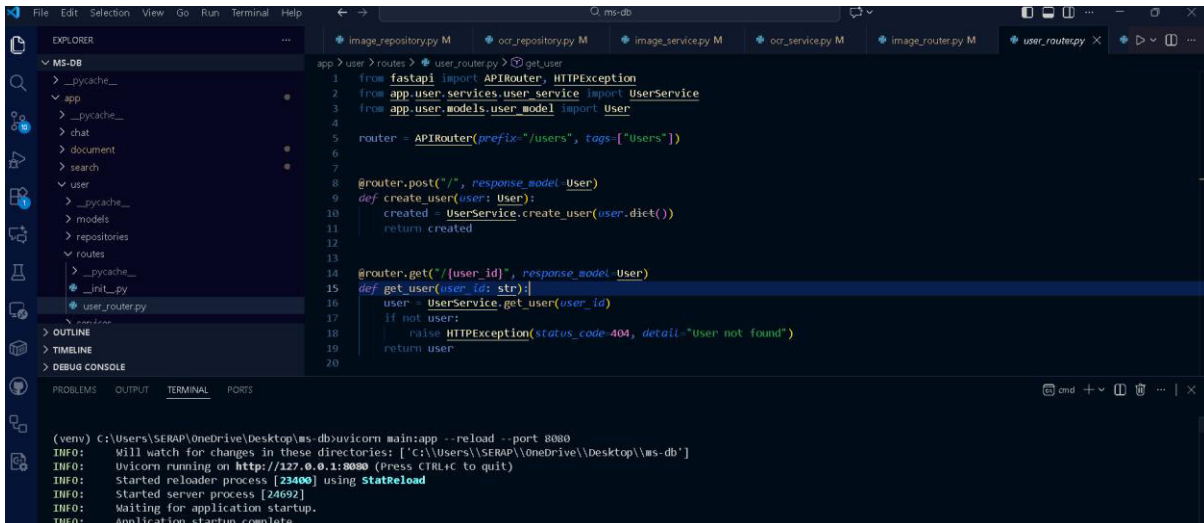
ok,ahora hablame sobre el concepto de las derivadas

La derivada es un concepto fundamental en cálculo que mide la tasa de cambio instantáneo de una función respecto a una variable independiente. Formalmente, se define como el límite del cociente de diferencias cuando el incremento de la variable tiende a cero: $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$ **Interpretaciones clave:** 1. **Geométrica:** La derivada en un punto representa la pendiente de la recta tangente a la gráfica de la función en ese punto. 2. **Física:** Si $f(t)$ describe la posición de un objeto en función del tiempo, $f'(t)$ corresponde a su velocidad instantánea. 3. **Económica:** Puede representar la sensibilidad de una variable (ej.: costo, ingreso) ante cambios en otra (ej.: cantidad producida). **Ejemplo:** Para $f(x) = x^2$, la derivada es $f'(x) = 2x$, lo que significa que en $x = 3$, la tasa de cambio es 6. **Aplicaciones:** - Optimización (encontrar máximos/mínimos). - Análisis de tasas de variación en ciencias. - Modelado de fenómenos dinámicos (ej.: crecimiento poblacional). ¿Quieres profundizar en algún aspecto específico, como reglas de derivación o ejemplos prácticos?

Figure 1 - Ejemplo validación conversacional con el modelo de LLM

Microservicio de base de datos

Se inicia los microservicios.



```

1 from fastapi import APIRouter, HTTPException
2 from app.user.services.user_service import UserService
3 from app.user.models.user_model import User
4
5 router = APIRouter(prefix="/users", tags=["Users"])
6
7
8 @router.post("/", response_model=User)
9 def create_user(user: User):
10     created = UserService.create_user(user.dict())
11     return created
12
13
14 @router.get("/{user_id}", response_model=User)
15 def get_user(user_id: str):
16     user = UserService.get_user(user_id)
17     if not user:
18         raise HTTPException(status_code=404, detail="User not found")
19     return user
20

```


```

(venv) C:\Users\SERAP\OneDrive\Desktop\ms-db\uvicorn main:app --reload --port 8080
INFO: Will watch for changes in these directories: ['C:\Users\SERAP\OneDrive\Desktop\ms-db']
INFO: Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
INFO: Started reload process [23400] using StatReload
INFO: Started server process [24692]
INFO: waiting for application startup.
INFO: Application startup complete.

```

Ilustración 33 - ejemplo de ejecución de un servicio de Fastapi

Microservicio de Front end



```

1 chatWindow {
2   background: var(--background);
3   color: var(--foreground);
4   display: flex;
5   flex-direction: column;
6   width: 100%; /* ocupa todo el espacio disponible */
7   height: 100%;
8   background: #1f88fc;
9 }
10
11
12 .header {
13   display: flex;
14   justify-content: space-between;
15   align-items: center;
16   padding: 12px;

```

```

PS C:\Users\SERAP\OneDrive\Desktop\front-end> npm run dev
% PS C:\Users\SERAP\OneDrive\Desktop\front-end> npm run dev

> front-end@0.1.0 dev
> next dev --turbo

Warning: Next.js inferred your workspace root, but it may not be correct.
We detected multiple lockfiles and selected the directory of c:\Users\SERAP\package-lock.json as the root directory.
To silence this warning, set "turbo.root" in your next.js config, or consider removing one of the lockfiles if it's not needed.
See https://nextjs.org/docs/app/api-reference/config/next-config-js/turbopack#root-directory for more information.
Detected additional lockfiles:
* C:\Users\SERAP\OneDrive\Desktop\front-end\package-lock.json

▲ Next.js 15.5.3 (Turbo)
- Local:    http://localhost:3000
- Network:  http://192.168.56.1:3000

✓ Starting...
✓ Ready in 2.9s

```

Ilustración 34 - ejemplo de inicio de next.js

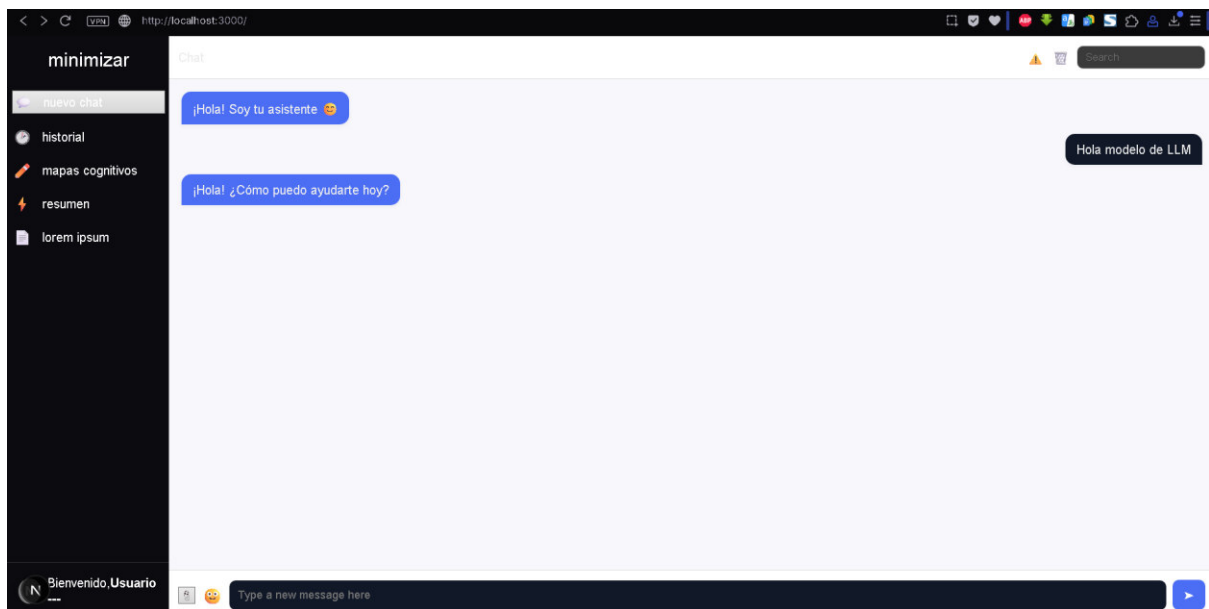


Ilustración 35 - conversación iniciada

Microservicio de Backend

Se muestra cómo se inicia el grafo del orquestador de LangGraph, a través de la consulta del usuario en la interfaz web.

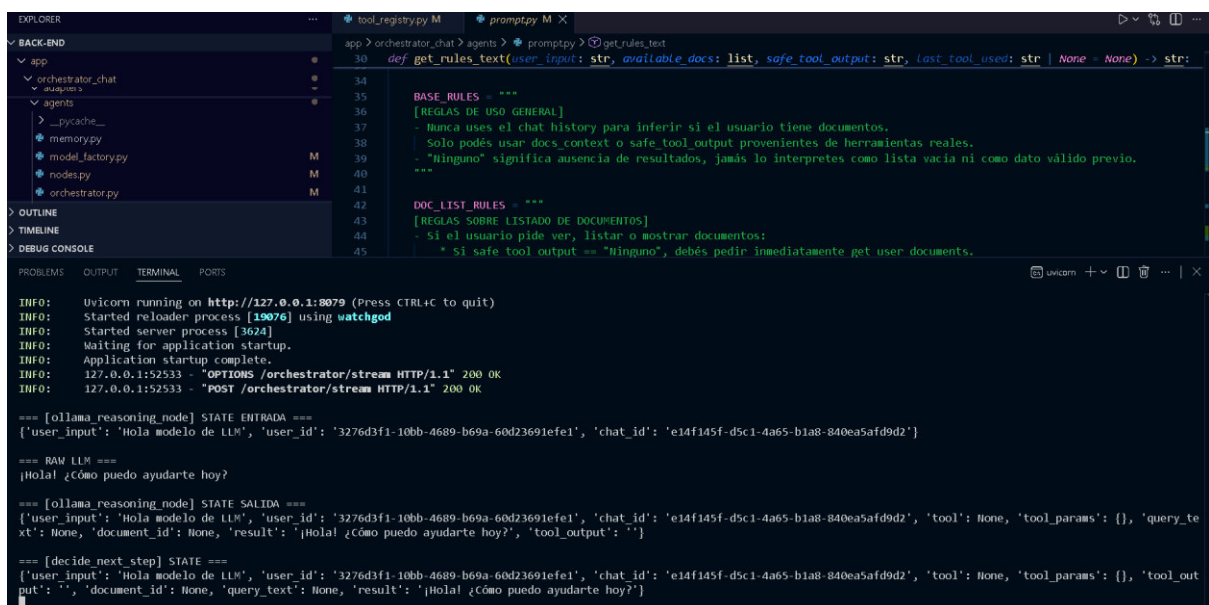


Ilustración 36 – Estado inicial – respuesta de langgraph

Microservicio de la base de datos

Se guardan los mensajes de la conversación

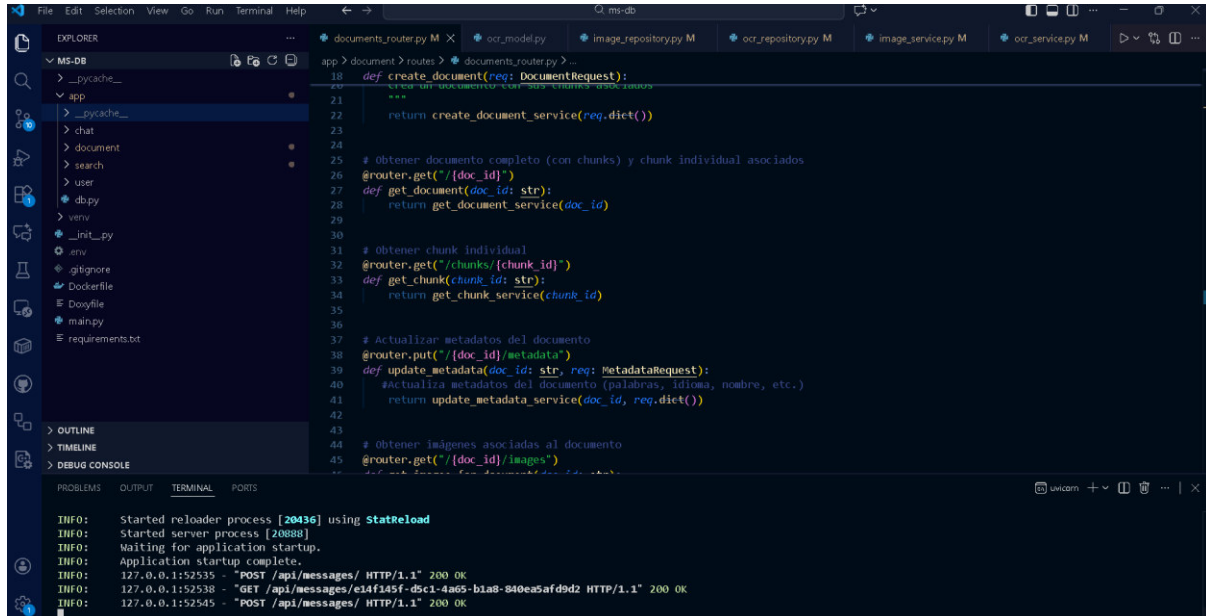


Ilustración 37 - Conversacion persistente

Microservicio de front end

Se indica al modelo de LLM que realice una acción, el modelo decide a partir de las herramientas dadas y por los límites de los prompts fijados

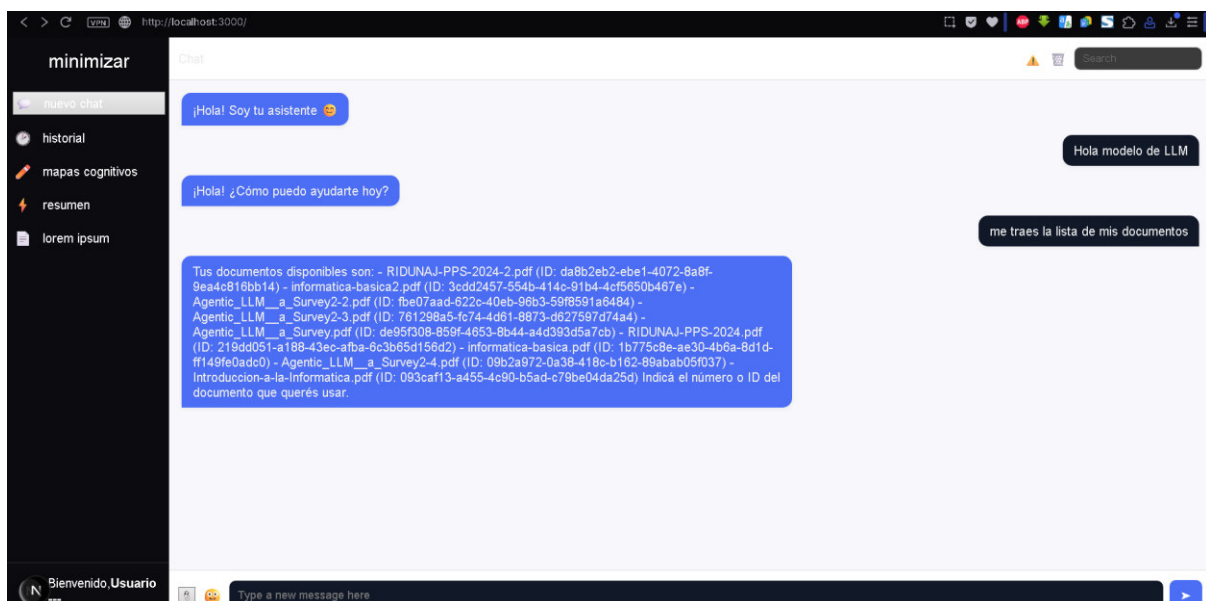


Ilustración 38 - modelo de uso de la herramienta de retriever

Microservicio de la Base de datos

En este caso la herramienta (tool: get documents) del microservicio de backend gracias al adapter de la base de datos, trajo la información de los documentos del usuario, sin necesidad de tener que usar una rutina de otro microservicio, simplemente usando el endpoint de document/user además como en cada paso de la conversación se guardan los mensajes de la conversación

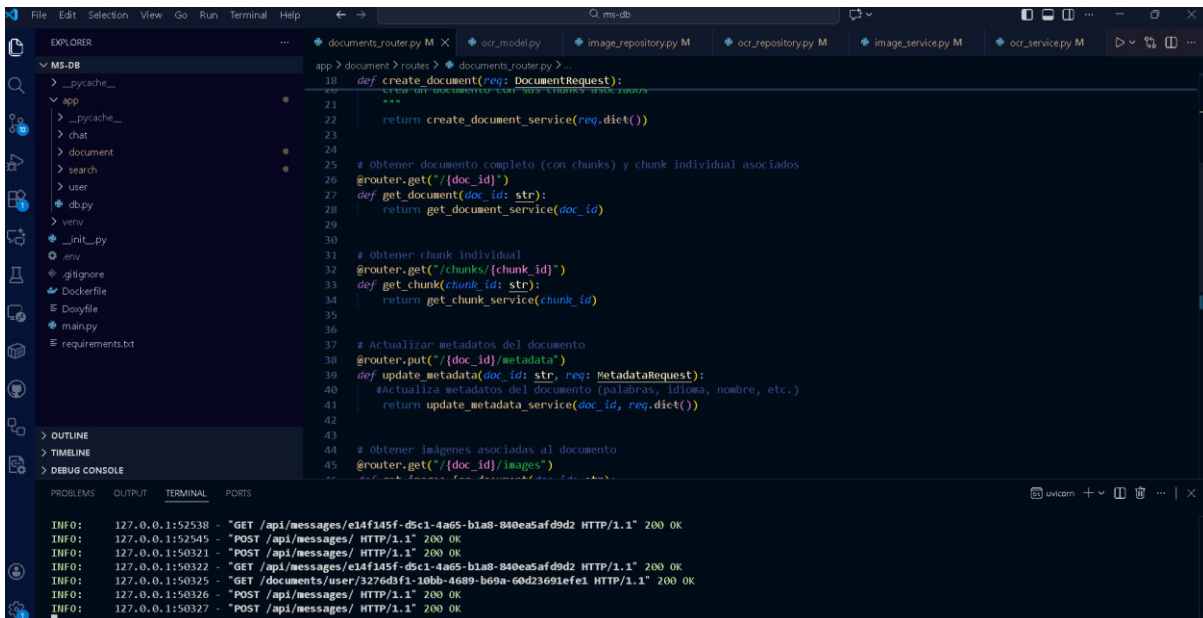


Ilustración 41 – obtención de información de documentos

Microservicio de Frontend

En esta ocasión se pide información más relevante acerca de un documento específico, el modelo de LLM deberá razonar y decidir no solo que herramienta usar, sino también que documento, además que deberá usar los estados de la información para no perder datos importantes entre ciclos del grafo.

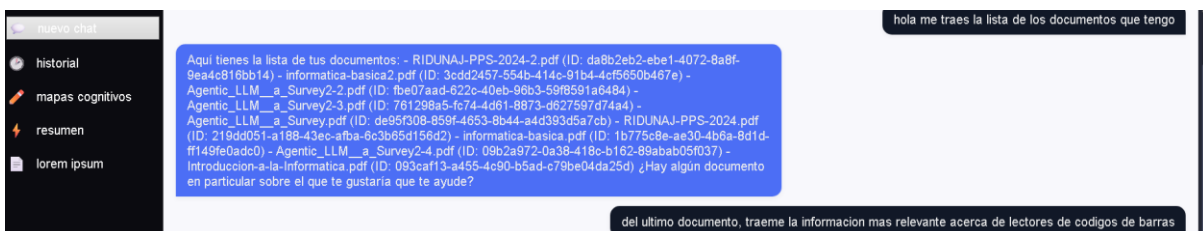
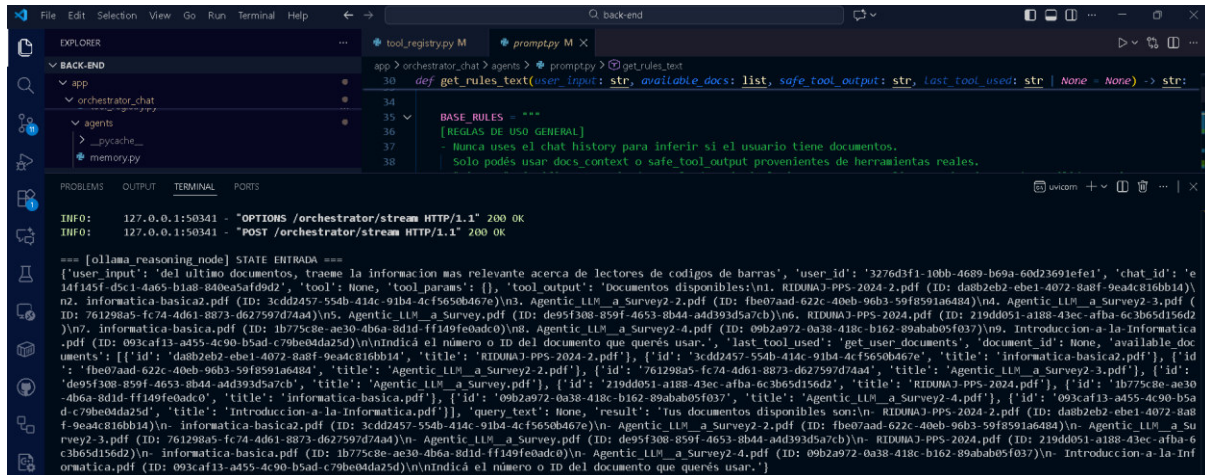


Ilustración 42 - Recuperacion de texto

Microservicio de Backend

El modelo de LLM, analiza la petición y actúa en consecuencia, se realiza la recuperación de textos, es decir, se emplea la herramienta de retrieveer



```

File Edit Selection View Go Run Terminal Help
back-end
EXPLORER
BACK-END
  app
    orchestrator_chat
      agents
        _pycache_
        mycapy
tool_registry.py M
prompt.py M X
app > orchestrator_chat > agents > prompt.py > get_rules_text
30 def get_rules_text(user_input: str, available_docs: list, safe_tool_output: str, last_tool_used: str) -> str:
31
32
33
34
35
36
37
38
BASE RULES - ***
[REGLAS DE USO GENERAL]
- Nunca uses el chat history para inferir si el usuario tiene documentos.
- Solo podés usar docs.context o safe_tool_output provenientes de herramientas reales.
PROBLEMS OUTPUT TERMINAL PORTS
INFO: 127.0.0.1:50341 - "OPTIONS /orchestrator/stream HTTP/1.1" 200 OK
INFO: 127.0.0.1:50341 - "POST /orchestrator/stream HTTP/1.1" 200 OK

=== [ollama_reasoning_node] STATE ENTRADA ===
{'user_input': 'del ultimo documentos, trae la informacion mas relevante acerca de lectores de codigos de barras', 'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'tool': None, 'tool_params': {}, 'tool_output': 'documentos disponibles:\n1. RIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\n2. informatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\n3. Agentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\n4. Agentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\n5. Agentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\n6. RIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\n7. informatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\n8. Agentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\n9. Introduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nIndicó el número o ID del documento que querés usar.', 'last_tool_used': 'get_user_documents', 'document_id': None, 'available_documents': [{'id': 'da8b2eb2-eb1-4072-8a8f-9e4dc816bb14', 'title': 'RIDUUAJ-PPS-2024-2.pdf', 'id': '3cdd2457-554b-414c-91b4-4cf5650b467e', 'title': 'informatica-basica2.pdf', 'id': 'fbe07aad-622c-40eb-96b3-59f8591a6484', 'title': 'Agentic_LLM_a_Survey2-2.pdf', 'id': '761298a5-fc74-4d61-8873-d627597d74a4', 'title': 'Agentic_LLM_a_Survey2-3.pdf', 'id': 'de95f308-859f-4653-8b44-aad393d5a7cb', 'title': 'Agentic_LLM_a_Survey.pdf', 'id': '219dd051-a188-43ec-afba-6c3b65d156d2', 'title': 'RIDUUAJ-PPS-2024.pdf', 'id': '1b775c8e-ae30-4b6a-8d1d-ff149fe0adco', 'title': 'informatica-basica.pdf', 'id': '09b2a972-0a38-418c-b162-89abab05f037', 'title': 'Agentic_LLM_a_Survey2-4.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'title': 'Introduccion-a-la-Informatica.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d'}], 'query_text': None, 'result': 'tus documentos disponibles son:\nRIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\ninformatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\nAgentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\nAgentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\nAgentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\nRIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\ninformatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\nAgentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\nIntroduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nIndicó el número o ID del documento que querés usar.'}

=== [ollama_reasoning_node] STATE SALIDA ===
{'user_input': 'del ultimo documento, trae la informacion mas relevante acerca de lectores de codigos de barras', 'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'tool': None, 'tool_params': {}, 'tool_output': 'documentos disponibles:\n1. RIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\n2. informatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\n3. Agentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\n4. Agentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\n5. Agentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\n6. RIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\n7. informatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\n8. Agentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\n9. Introduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nIndicó el número o ID del documento que querés usar.', 'last_tool_used': 'get_user_documents', 'document_id': None, 'available_documents': [{'id': 'da8b2eb2-eb1-4072-8a8f-9e4dc816bb14', 'title': 'RIDUUAJ-PPS-2024-2.pdf', 'id': '3cdd2457-554b-414c-91b4-4cf5650b467e', 'title': 'informatica-basica2.pdf', 'id': 'fbe07aad-622c-40eb-96b3-59f8591a6484', 'title': 'Agentic_LLM_a_Survey2-2.pdf', 'id': '761298a5-fc74-4d61-8873-d627597d74a4', 'title': 'Agentic_LLM_a_Survey2-3.pdf', 'id': 'de95f308-859f-4653-8b44-aad393d5a7cb', 'title': 'Agentic_LLM_a_Survey.pdf', 'id': '219dd051-a188-43ec-afba-6c3b65d156d2', 'title': 'RIDUUAJ-PPS-2024.pdf', 'id': '1b775c8e-ae30-4b6a-8d1d-ff149fe0adco', 'title': 'informatica-basica.pdf', 'id': '09b2a972-0a38-418c-b162-89abab05f037', 'title': 'Agentic_LLM_a_Survey2-4.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'title': 'Introduccion-a-la-Informatica.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d'}], 'query_text': None, 'result': 'Aquí tienes la lista de tus documentos:\n\nRIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\ninformatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\nAgentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\nAgentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\nAgentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\nRIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\ninformatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\nAgentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\nIntroduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nHay algún documento en particular sobre el que te gustaría que te ayude?'}

=== RAW LLM ===
tool:retriever_tool
{'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'document_id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'query': 'lectores de c odigos de barras'}

=== [ollama_reasoning_node] STATE SALIDA ===
{'user_input': 'del ultimo documento, trae la informacion mas relevante acerca de lectores de codigos de barras', 'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'tool': 'retriever_tool', 'tool_params': {'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'document_id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'query': 'lectores de codigos de barras'}, 'query_text': 'lectores de codigos de barras', 'tool_output': 'documentos disponibles:\n1. RIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\n2. informatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\n3. Agentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\n4. Agentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\n5. Agentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\n6. RIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\n7. informatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\n8. Agentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\n9. Introduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nIndicó el número o ID del documento que querés usar.', 'last_tool_used': 'get_user_documents', 'document_id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'available_documents': [{'id': 'da8b2eb2-eb1-4072-8a8f-9e4dc816bb14', 'title': 'RIDUUAJ-PPS-2024-2.pdf', 'id': '3cdd2457-554b-414c-91b4-4cf5650b467e', 'title': 'informatica-basica2.pdf', 'id': 'fbe07aad-622c-40eb-96b3-59f8591a6484', 'title': 'Agentic_LLM_a_Survey2-2.pdf', 'id': '761298a5-fc74-4d61-8873-d627597d74a4', 'title': 'Agentic_LLM_a_Survey2-3.pdf', 'id': 'de95f308-859f-4653-8b44-aad393d5a7cb', 'title': 'Agentic_LLM_a_Survey.pdf', 'id': '219dd051-a188-43ec-afba-6c3b65d156d2', 'title': 'RIDUUAJ-PPS-2024.pdf', 'id': '1b775c8e-ae30-4b6a-8d1d-ff149fe0adco', 'title': 'informatica-basica.pdf', 'id': '09b2a972-0a38-418c-b162-89abab05f037', 'title': 'Agentic_LLM_a_Survey2-4.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'title': 'Introduccion-a-la-Informatica.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d'}], 'query_text': 'lectores de codigos de barras', 'result': 'lectores de codigos de barras'}
  
```

Ilustración 43 - Inicio de la recuperación mediante LangGraph



```

INFO: 127.0.0.1:50433 - "POST /orchestrator/stream HTTP/1.1" 200 OK

=== [ollama_reasoning_node] STATE ENTRADA ===
{'user_input': 'del ultimo documento, trae la informacion mas relevante acerca de lectores de codigos de barras', 'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'tool': None, 'tool_params': {}, 'tool_output': 'documentos disponibles:\n1. RIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\n2. informatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\n3. Agentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\n4. Agentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\n5. Agentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\n6. RIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\n7. informatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\n8. Agentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\n9. Introduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nIndicó el número o ID del documento que querés usar.', 'last_tool_used': 'get_user_documents', 'document_id': None, 'available_documents': [{'id': 'da8b2eb2-eb1-4072-8a8f-9e4dc816bb14', 'title': 'RIDUUAJ-PPS-2024-2.pdf', 'id': '3cdd2457-554b-414c-91b4-4cf5650b467e', 'title': 'informatica-basica2.pdf', 'id': 'fbe07aad-622c-40eb-96b3-59f8591a6484', 'title': 'Agentic_LLM_a_Survey2-2.pdf', 'id': '761298a5-fc74-4d61-8873-d627597d74a4', 'title': 'Agentic_LLM_a_Survey2-3.pdf', 'id': 'de95f308-859f-4653-8b44-aad393d5a7cb', 'title': 'Agentic_LLM_a_Survey.pdf', 'id': '219dd051-a188-43ec-afba-6c3b65d156d2', 'title': 'RIDUUAJ-PPS-2024.pdf', 'id': '1b775c8e-ae30-4b6a-8d1d-ff149fe0adco', 'title': 'informatica-basica.pdf', 'id': '09b2a972-0a38-418c-b162-89abab05f037', 'title': 'Agentic_LLM_a_Survey2-4.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'title': 'Introduccion-a-la-Informatica.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d'}], 'query_text': None, 'result': 'Aquí tienes la lista de tus documentos:\n\nRIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\ninformatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\nAgentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\nAgentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\nAgentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\nRIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\ninformatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\nAgentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\nIntroduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nHay algún documento en particular sobre el que te gustaría que te ayude?'}

=== RAW LLM ===
tool:retriever_tool
{'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'document_id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'query': 'lectores de c odigos de barras'}

=== [ollama_reasoning_node] STATE SALIDA ===
{'user_input': 'del ultimo documento, trae la informacion mas relevante acerca de lectores de codigos de barras', 'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'tool': 'retriever_tool', 'tool_params': {'user_id': '3276d3f1-10bb-4689-b69a-60d23691efe1', 'chat_id': 'e14f145f-d5c1-4a65-b1a8-840ea5afd9d2', 'document_id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'query': 'lectores de codigos de barras'}, 'query_text': 'lectores de codigos de barras', 'tool_output': 'documentos disponibles:\n1. RIDUUAJ-PPS-2024-2.pdf (ID: da8b2eb2-eb1-4072-8a8f-9e4dc816bb14)\n2. informatica-basica2.pdf (ID: 3cdd2457-554b-414c-91b4-4cf5650b467e)\n3. Agentic_LLM_a_Survey2-2.pdf (ID: fbe07aad-622c-40eb-96b3-59f8591a6484)\n4. Agentic_LLM_a_Survey2-3.pdf (ID: 761298a5-fc74-4d61-8873-d627597d74a4)\n5. Agentic_LLM_a_Survey.pdf (ID: de95f308-859f-4653-8b44-aad393d5a7cb)\n6. RIDUUAJ-PPS-2024.pdf (ID: 219dd051-a188-43ec-afba-6c3b65d156d2)\n7. informatica-basica.pdf (ID: 1b775c8e-ae30-4b6a-8d1d-ff149fe0adco)\n8. Agentic_LLM_a_Survey2-4.pdf (ID: 09b2a972-0a38-418c-b162-89abab05f037)\n9. Introduccion-a-la-Informatica.pdf (ID: 093caf13-a455-4c90-b5ad-c79be04da25d)\n\nIndicó el número o ID del documento que querés usar.', 'last_tool_used': 'get_user_documents', 'document_id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'available_documents': [{'id': 'da8b2eb2-eb1-4072-8a8f-9e4dc816bb14', 'title': 'RIDUUAJ-PPS-2024-2.pdf', 'id': '3cdd2457-554b-414c-91b4-4cf5650b467e', 'title': 'informatica-basica2.pdf', 'id': 'fbe07aad-622c-40eb-96b3-59f8591a6484', 'title': 'Agentic_LLM_a_Survey2-2.pdf', 'id': '761298a5-fc74-4d61-8873-d627597d74a4', 'title': 'Agentic_LLM_a_Survey2-3.pdf', 'id': 'de95f308-859f-4653-8b44-aad393d5a7cb', 'title': 'Agentic_LLM_a_Survey.pdf', 'id': '219dd051-a188-43ec-afba-6c3b65d156d2', 'title': 'RIDUUAJ-PPS-2024.pdf', 'id': '1b775c8e-ae30-4b6a-8d1d-ff149fe0adco', 'title': 'informatica-basica.pdf', 'id': '09b2a972-0a38-418c-b162-89abab05f037', 'title': 'Agentic_LLM_a_Survey2-4.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d', 'title': 'Introduccion-a-la-Informatica.pdf', 'id': '093caf13-a455-4c90-b5ad-c79be04da25d'}], 'query_text': 'lectores de codigos de barras', 'result': 'lectores de codigos de barras'}
  
```

Ilustración 44 - Eleccion de la herramienta

Microservicio de Procesamiento y clasificación

Mediante el modelo de LLM y gracias a las reglas del prompt se puede sacar la parte más importante de la frase (en el caso de la búsqueda anterior sería: “lectores de código de barra”) para usar esa información y pasársela al tool de retrieveer

La herramienta hace uso del agente de recuperación en el microservicio de procesamiento y clasificación o de la información de la frase (almacenada en los estados)

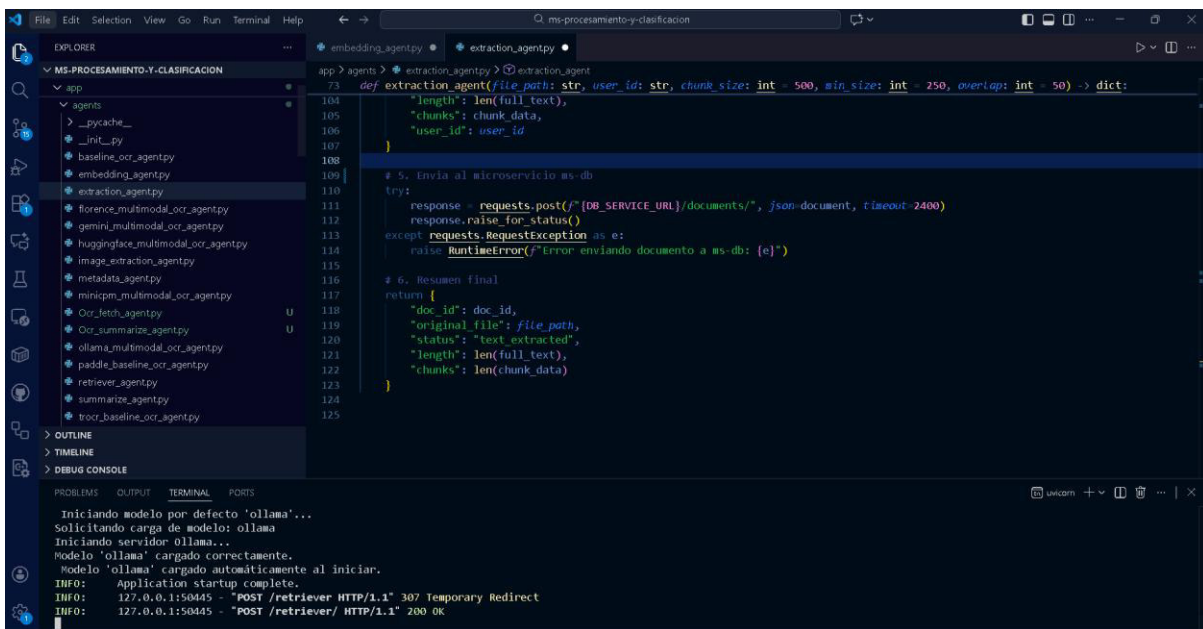


Ilustración 45 - uso del agente Retriever

Microservicio de la Base de datos

El agente de retriever, toma la elección de la frase más relevante que el usuario dio en la búsqueda, para hacer uso del módulo de búsqueda(search) de la base de datos.

Esta permite hacer la búsqueda sin necesidad de guardar la información y aplicar distintas formas de búsquedas de ser necesaria, aunque se usa la forma por defecto que es generar el embedding de la búsqueda y buscar por similitud en el documento especificado.

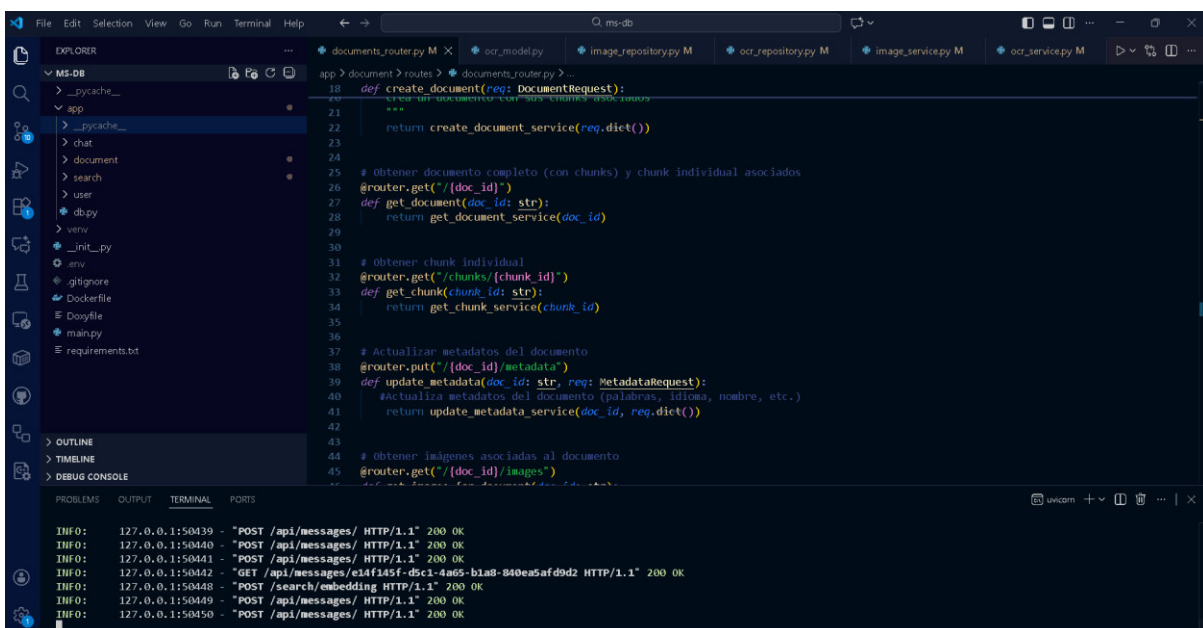


Ilustración 46 - búsqueda mediante Vectorización

Se recuperan los textos cuya similaridad sean los más cercanos a la búsqueda y se devuelve la información al agente de retriever, este termina de devolver la información a la tool que los requirio, para luego actualizar los estados del grafo, el LLM se da cuenta que ya tiene la información necesaria y termina de mostrar la información por pantalla

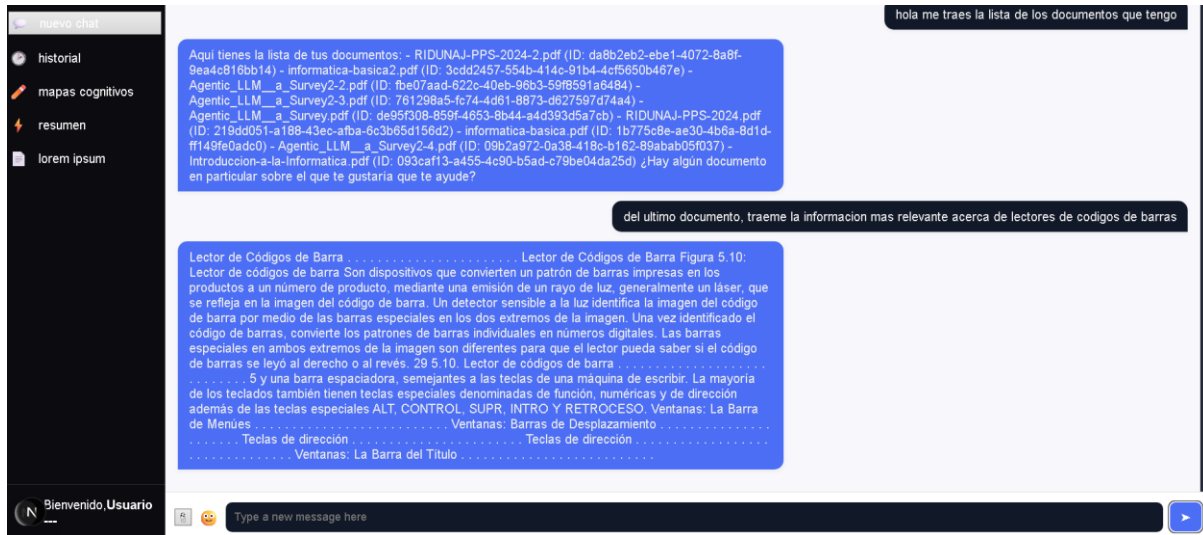


Ilustración 47 - Resultado de la herramienta de recuperacion

Microservicio de Front end

Finalmente, como última parte de las herramientas accesibles al modelo de Langgraph, encontramos, que el usuario pide no solo un resumen, si no que traiga una imagen relevante acerca de la información generada por la recuperación de texto.

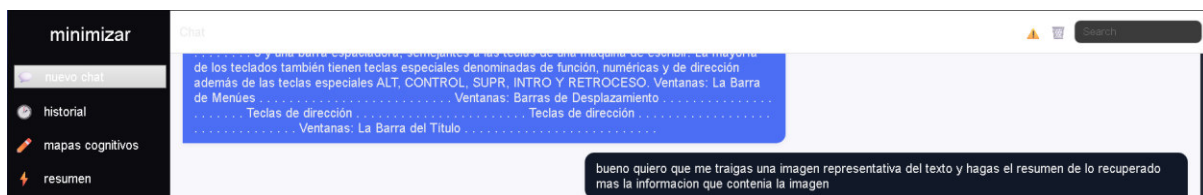


Ilustración 48 - Búsqueda con multiples herramientas

Microservicio de Backend

Nuevamente el orquestador de LangGraph deberá razonar exactamente qué hacer y cómo, empezará por tomar el valor de la frase y mediante los prompts del sistema elegir la herramienta más adecuada

Siguiendo la corriente que el usuario pide, se usara primero la herramienta de traer imágenes(tool: ocr_fetch_image) que usa el estado guardado de los datos de la tool retrieveer (las páginas de los textos recuperados), una vez que se trae la lista de páginas, se pasa a buscar la imagen.

De la lista de las páginas, se usa la más relevante (es decir que sea la de mayor similaridad) , una vez recuperado la información se utiliza la tool de ocr_summarize que toma el id de la imagen para poder buscar la información del ocr multimodal que ya está guardado en la base de datos



```

=== [tool_node] SIATE ENTRADA ===
{
  "user_input": "bueno quiero que me traigas una imagen representativa del texto y hagas el resumen de lo recuperado mas la información que contenia la imagen",
  "user_id": "3276d3f1-10bb-4689-b69a-60d23691efe1",
  "chat_id": "e1af1d5f-d5c1-4a65-b1a8-840ea5af9d2",
  "tool": "ocr_fetch_image_tool",
  "tool_params": {
    "user_id": "3276d3f1-10bb-4689-b69a-60d23691efe1",
    "chat_id": "e1af1d5f-d5c1-4a65-b1a8-840ea5af9d2",
    "document_id": "093caf13-a455-4c90-b5ad-c79be04da25d",
    "page_number": 29,
    "tool_output": "Lector de códigos de Barra"
  }
}

...
Lector de códigos de Barra
Figura 5.10: Lector de códigos de barra
son dispositivos que convierten un patrón de barras impresas en los productos a un número de producto, mediante una emisión de un rayo de luz, generalmente un láser, que se refleja en la imagen del código de barra. Un detector sensible a la luz identifica la imagen del código de barra por medio de las barras especiales en los dos extremos de la imagen. Una vez identificado el código de barras, convierte los patrones de barras individuales en números digitales. Las barras especiales en ambos extremos de la imagen son diferentes para que el lector pueda saber si el código de barras se leyó al derecho o al revés.

...
Un detector sensible a la luz identifica la imagen del código de barra por medio de las barras especiales en los dos extremos de la imagen. Una vez identificado el código de barras, convierte los patrones de barras individuales en números digitales. Las barras especiales en ambos extremos de la imagen son diferentes para que el lector pueda saber si el código de barras se leyó al derecho o al revés.

...
Una barra espaciadora, semejantes a las teclas de una máquina de escribir. La mayoría de los teclados también tienen teclas especiales denominadas de función, numéricas y de dirección además de las teclas especiales (ALT, CONTROL, SUPR, INTRO Y RETROCESO).

...
Ventanas: Barras de Desplazamiento
Ventanas: La Barra de Dirección
Ventanas: La Barra del Título

...
last_tool_used: "retriever_tool",
document_id: "093caf13-a455-4c90-b5ad-c79be04da25d",
available_documents: [
  {
    "id": "da82eb2-ebe1-4072-8af-9eac816bb14",
    "title": "RIDUWAJ-PPS-2024-2.pdf",
    "id": "3cdd2457-554b-414c-91b4-4cf5650b467e",
    "title": "informatica-basica2.pdf",
    "id": "fbc07aad-622c-40eb-96b3-59f8591a6484",
    "title": "Agentic LLM a Survey2.2.pdf",
    "id": "761298a5-fc74-4d61-8873-d62759d74aa4",
    "title": "Agentic LLM a Survey2.3.pdf",
    "id": "de95f308-859f-4653-8b44-a4d39d35a7cb",
    "title": "Agentic LLM a Survey.pdf",
    "id": "219dd051-a188-43ec-afba-6c3b05d136da",
    "title": "RIDUWAJ-PPS-2024.pdf",
    "id": "1b775c8e-ae30-4b6a-bd1d-ff149f60adc",
    "title": "informatica-basica.pdf",
    "id": "09b29972-8a38-418c-b162-89ab005f0377",
    "title": "Agentic LLM a Survey2.4.pdf",
    "id": "093caf13-a455-4c90-b5ad-c79be04da25d",
    "title": "Introducción a la Informática.pdf"
  ],
  "query_text": "lecturas de códigos de barras",
  "retrieval_text": "Lector de códigos de Barra"
}

...
Lector de códigos de Barra
Figura 5.10: Lector de códigos de barra
son dispositivos que convierten un patrón de barras impresas en los productos a un número de producto, mediante una emisión de un rayo de luz, generalmente un láser, que se refleja en la imagen del código de barra. Un detector sensible a la luz identifica la imagen del código de barra por medio de las barras especiales en los dos extremos de la imagen. Una vez identificado el código de barra, convierte los patrones de barras individuales en números digitales. Las barras especiales en ambos extremos de la imagen son diferentes para que el lector pueda saber si el código de barras se leyó al derecho o al revés.

...
Una barra espaciadora, semejantes a las teclas de una máquina de escribir. La mayoría de los teclados también tienen teclas especiales denominadas de función, numéricas y de dirección además de las teclas especiales (ALT, CONTROL, SUPR, INTRO Y RETROCESO).

...
Ventanas: Barras de Desplazamiento
Ventanas: La Barra de Dirección
Ventanas: La Barra del Título

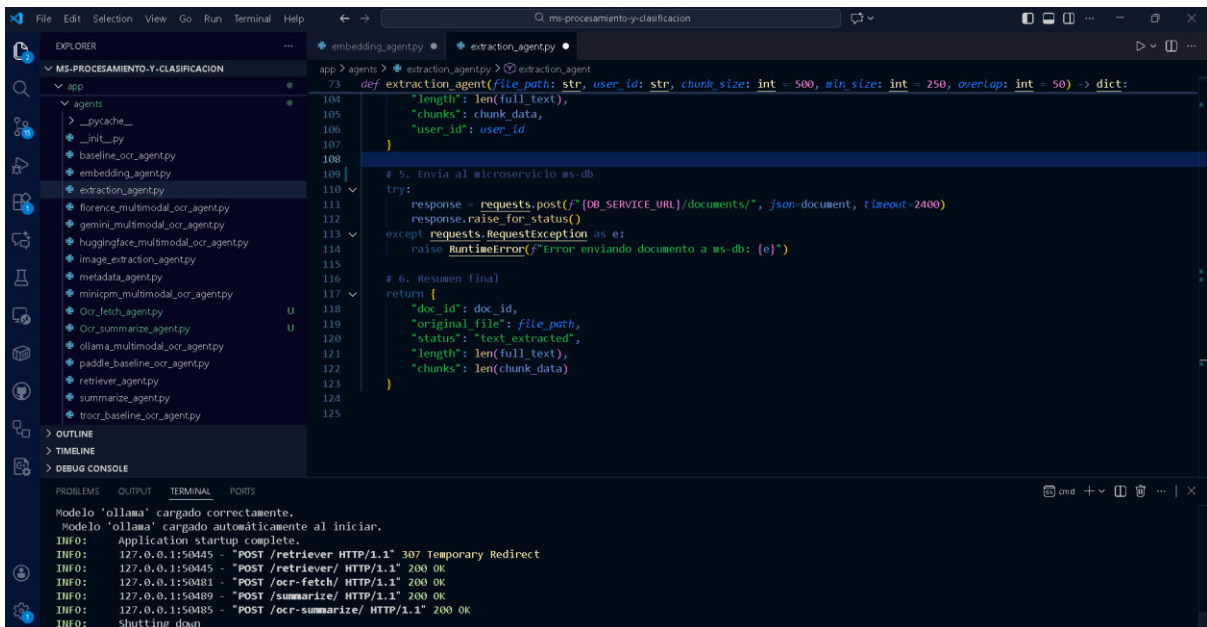
...
retriever_results: [
  {
    "id": "05187fd-6ed7-401c-ac53-ca50e087aebb",
    "text": "Lector de códigos de Barra",
    "document_id": "093caf13-a455-4c90-b5ad-c79be04da25d",
    "page_number": 9,
    "similarity": 0.9099175930023193,
    "id": "cd02723d-72eb-431e-ad63-076c7917e536",
    "text": "Lector de códigos de Barra\nFigura 5.10: Lector de códigos de barra\nson dispositivos que convierten un patrón de barras impresas en los productos a un número de producto, mediante una emisión de un rayo de luz, generalmente un láser, que se refleja en la imagen del código de barra. Un detector sensible a la luz identifica la imagen del código de barra por medio de las barras especiales en los dos extremos de la imagen. Una vez identificado el código de barra, convierte los patrones de barras individuales en números digitales. Las barras especiales en ambos extremos de la imagen son diferentes para que el lector pueda saber si el código de barras se leyó al derecho o al revés.",
    "document_id": "093caf13-a455-4c90-b5ad-c79be04da25d",
    "page_number": 46,
    "similarity": 0.8684346675872803,
    "id": "c1a517c1-f0dc-490f-88fa-25cd9af2e1f1",
    "text": "29\n5.10. Lector de códigos de barra"
  },
  {
    "document_id": "093caf13-a455-4c90-b5ad-c79be04da25d",
    "page_number": 14,
    "similarity": 0.80884163665771484,
    "id": "c784845d-2193-4484-b874-e5c63a7a93d5",
    "text": "5\nny una barr"
  }
]

```

Ilustración 49 – búsqueda de imagen relevante en base 64

Microservicio de Procesamiento y clasificación

Se usa el agente de recuperación de imágenes, y se traen todas las imágenes de la lista de páginas, pero solo se guarda la imagen más relevante, vía la similiaridad más alta.



```


def extraction_agent(file_path: str, user_id: str, chunk_size: int = 500, min_size: int = 250, overlap: int = 50) -> dict:
    """
    Función que extrae el texto de un documento y lo envía al microservicio ms-db.
    """
    # 5. Envía al microservicio ms-db
    try:
        response = requests.post(f"{DB_SERVICE_URL}/documents/", json=document, timeout=2400)
        response.raise_for_status()
    except requests.RequestException as e:
        raise RuntimeError(f"Error enviando documento a ms-db: {e}")

    # 6. Resumen final
    return {
        "doc_id": doc_id,
        "original_file": file_path,
        "status": "text extracted",
        "length": len(full_text),
        "chunks": len(chunk_data)
    }

```

Microservicio de la Base de datos

Se realiza la búsqueda mediante el id recuperado de la imagen, para solicitar la información obtenida por el agente ocr multimodal.



```

INFO: 127.0.0.1:50487 - "GET /images/df1a7190-a8fd-477b-94d0-42b7c6679a23/multimodal-ocr HTTP/1.1" 200 OK
INFO: 127.0.0.1:50507 - "POST /api/messages HTTP/1.1" 200 OK
INFO: 127.0.0.1:50507 - "POST /api/messages/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:50508 - "POST /api/messages/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:50509 - "POST /api/messages/ HTTP/1.1" 200 OK


```

Ilustración 52 - información ocr multimodal de la imagen recuperada

Microservicio de Procesamiento y clasificación

Ocr summarize usa el agente de summarize para tomar todo el texto recuperado (textos recuperados mediante retriever y la información que tenía la imagen recuperada del ocr multimodal) y hacer un resumen mediante el modelo de LLM.

Esta información se la pasa al backend para que el orquestador pueda terminar con el ciclo del grafo y mostrar en pantalla la información pedida.



```

def extraction_agent(file_path: str, user_id: str, chunk_size: int = 500, min_size: int = 250, overlap: int = 50) -> dict:
    """
    Función que extrae el texto de un documento y lo envía al microservicio ms-db.
    """
    # 5. Envía al microservicio ms-db
    try:
        response = requests.post(f"{DB_SERVICE_URL}/documents/", json=document, timeout=2400)
        response.raise_for_status()
    except requests.RequestException as e:
        raise RuntimeError(f"Error enviando documento a ms-db: {e}")

    # 6. Resumen final
    return {
        "doc_id": doc_id,
        "original_file": file_path,
        "status": "text extracted",
        "length": len(full_text),
        "chunks": len(chunk_data)
    }

```

Ilustración 53 - uso de los agentes ocr -summarize y ocr-fetch

8. Conclusiones

El desarrollo del sistema permitió la validación integral de la arquitectura propuesta, basada en microservicios con orquestación centralizada. esta configuración demostró ser la adecuada para cumplir con los objetivos de escalabilidad, modularidad y extensibilidad planteadas al inicio del proyecto.

La separación funcional entre orquestación, el procesamiento documental y la persistencia de los datos, mediante interfaces REST, permitió asegurar un desacoplamiento efectivo que posibilita la evolución independiente de cada componente sin comprometer la coherencia global.

La incorporación de patrones arquitectónicos como gateway y adapter además permitió la capacidad de abstraer el acceso a la base de datos y de poder extender dinámicamente las funciones del orquestador mediante herramientas especializadas.

La integración de LangGraph permitió implementar un flujo de razonamiento dinámico basado en ciclos de percepción - decisión - acción, lo que luego se revela en desafíos a la hora de la trazabilidad de los estados, control de variabilidad de los nodos, la definición de reglas del modelo y el manejo del estado conversacional.

Al ir resolviendo estos problemas permitieron validar la robustez de la arquitectura y subrayaron la importancia del diseño de interfaces y del modelado del comportamiento del LLM

En términos de seguridad, el enfoque de microservicios demostró ser capaz de reducir la probabilidad de errores en el futuro, al restringir el acceso del modelo a recursos sensibles y aplicar estrictamente el principio de mínimo privilegio.

En cuanto al rendimiento, el razonamiento iterativo tuvo problemas en cuanto a se refiere al accionar local, lo que evidencio costos operativos inherentes, lo que permite establecer líneas claras para futuras optimizaciones.

Desde una perspectiva funcional, el sistema alcanzo los objetivos propuestos por este trabajo, el de construir una aplicación web capaz de analizar documentos científicos mediante la

construcción de agentes especializados, integrando segmentación, vectorización, inclusión de un sistema RAG, ocr tradicional así como multimodal y resúmenes mediante los modelos de LLM.

El resultado fue una herramienta efectiva al momento de acelerar el entendimiento de documentos y también para su exploración, suministrando al usuario de un entorno interactivo de consulta contextualizada.

Finalmente, en el plano formativo, la práctica profesional supervisada, me significó un avance sustancial en el desarrollo de mis habilidades profesionales. el proyecto me permitió profundizar en arquitecturas distribuidas, patrones de diseño, iniciarme en el mundo de LLM, integrar los mismos entornos reales , manejo de los estados en sistemas complejos y buenas prácticas de la ingeniería aplicadas a sistemas de IA.

Este proceso fortaleció mi criterio técnico y la capacidad para enfrentar problemas de diseño arquitectónico con una mirada más analítica.

En síntesis, el trabajo integro con éxito una arquitectura robusta, escalable, extensible con un motor de razonamiento dinámico, generando una aplicación web que satisface el problema por el cual se inició el proyecto y consolidando mi crecimiento profesional en áreas claves de la ingeniería de software.

9. Bibliografía

[1] Plaat, A., van Duijn, M., van Stein, N., Preuss, M., van der Putten, P., & Batenburg, K. J. (2025). *Agentic Large Language Models, a survey* (arXiv:2503.23037 [cs.AI]). Repositorio de preimpresiones de arXiv. <https://arxiv.org/abs/2503.23037>

[2] Y. Shen *et al.*, "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face", **arXiv**, [revisado Diciembre 2023]. Disponible en: <https://arxiv.org/abs/2303.17580>

[3] Wooldridge, M. (2009). Multiagent Systems. En *An Introduction to Multiagent Systems* (2.^a ed., pp. 1–28). John Wiley & Sons

[4] Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). *Generative Agents: Interactive Simulacra of Human Behavior*. arXiv. [\[https://arxiv.org/abs/2304.03442\]](https://arxiv.org/abs/2304.03442)

[5] Han, S., Xia, P., Zhang, R., Sun, T., Li, Y., Zhu, H., & Yao, H. (2025). *MDocAgent: A Multi-Modal Multi-Agent Framework for Document Understanding*. arXiv. [\[https://arxiv.org/abs/2503.13964\]](https://arxiv.org/abs/2503.13964)

10. Recursos web

- [1] Van Rossum, G. (s. f.). *Guido van Rossum's Home Page*. GitHub. ⁽¹⁾
<https://gvanrossum.github.io/>
- [2] GeeksforGeeks. (2023, 22 de diciembre). *History of Python*. ⁽²⁾
<https://www.geeksforgeeks.org/python/history-of-python/>
- [3] Amazon Web Services. (s. f.). *¿Qué es Python? AWS*. <https://aws.amazon.com/es/what-is/python/>
- [4] Python Institute. (s. f.). *About Python*. <https://pythoninstitute.org/about-python>
- [5] The VS Code Team. (2016, abril 14). *Visual Studio Code 1.0!* Visual Studio Code.
<https://code.visualstudio.com/blogs/2016/04/14/vscode-1.0> code.visualstudio.com
- [6] Yadav, R. (s. f.). *The untold story of Visual Studio Code: A revolution in software development*. dev.to. <https://dev.to/rajeshkumaryadavdotcom/the-untold-story-of-visual-studio-code-a-revolution-in-software-development-44pp>
- [7] tiangolo. (s. f.). *Historia, diseño y futuro — FastAPI*.
<https://fastapi.tiangolo.com/es/history-design-future/> fastapi.tiangolo.com
- [8] Ahmed, S. (s. f.). *FastAPI – Advantages and disadvantages*. Medium.
<https://medium.com/@shariq.ahmed525/fastapi-advantages-and-disadvantages-fe1e5190317d>
- [9] Amazon Web Services. (s. f.). *What is LangChain*. <https://aws.amazon.com/es/what-is/langchain/>
- [10] LangChain. (s. f.). *State of AI agents*. <https://www.langchain.com/stateofaiagents>
- [11] IBM. (s. f.). *LangGraph*. <https://www.ibm.com/think/topics/langgraph>
- [12] Dam, J. D. (s. f.). *Guía completa: LangGraph y el futuro de los agentes de IA en 2025*. Medium. <https://medium.com/@jddam/guía-completa-langgraph-y-el-futuro-de-los-agentes-de-ia-en-2025-2f34ceaa456f>
- [13] Delftswa. (s. f.). *Neo4j: capítulo sobre Neo4j*.
<https://delftswa.gitbooks.io/desosa2016/content/neo4j/chapter.html>
- [14] TigerGraph. (s. f.). *Cypher Query Language*.
<https://www.tigergraph.com/glossary/cypher-query-language/>

[15] Estrada, J. L., & Chimal, D. C. (2022). *Arquitectura de la Información: Diseño de la Usabilidad en la Web*. *RIDE Revista Iberoamericana para la Investigación y el Desarrollo Educativo*, 12(24), e016.

<https://www.redalyc.org/journal/3442/344268257016/html/>