Documentación Challenge Final – PI Consulting

GET TALENT – IA

Desarrollado por:

Johana Belen Escudero

Índice

Información General del Proyecto

Objetivo

El objetivo para este proyecto es:

Creación de un chatbot utilizando una arquitectura RAG (Retrieval-Augmented Generation) para abordar un problema específico de la vida real utilizando de manera efectiva los conceptos y técnicas a lo largo del curso, como así también tecnologías nuevas investigadas por el alumno para poder llegar a una solución más enriquecedora e innovadora.

Contexto organizacional

Este proyecto se desarrolla para la empresa PI Data Strategy & Consulting, entidad privada que se dedica a brindar servicios y soluciones integrales de infraestructura y procesamiento de datos complejos. Las áreas en las que se especializa son: analítica Avanzada, Machine Learning, Inteligencia Artificial, Business Intelligence, entre otras.

Descripción general del producto

UOCRAIA es un chatbot diseñado para asistir a empleados y delegados del sector de la construcción afiliados al gremio UOCRA, proporcionando información clara, precisa y accesible sobre sus derechos y obligaciones laborales.

Este proyecto surge de la realidad que enfrentan muchos trabajadores, especialmente en regiones del sur del país, donde el nivel educativo suele ser bajo y, en muchos casos, no se requiere haber finalizado la educación secundaria para ingresar al sector. Esta situación genera una serie de problemáticas, como:

- Aprovechamiento por parte de los empleadores.
- Mal pago de las horas trabajadas.
- Falta de provisión de herramientas necesarias.

Estas prácticas son ilegales, pero la falta de conocimiento y acceso a la información deja a los trabajadores en situación de vulnerabilidad. **UOCRAIA** busca cerrar esta brecha de información, empoderando a los empleados con herramientas para conocer y hacer valer sus derechos laborales.

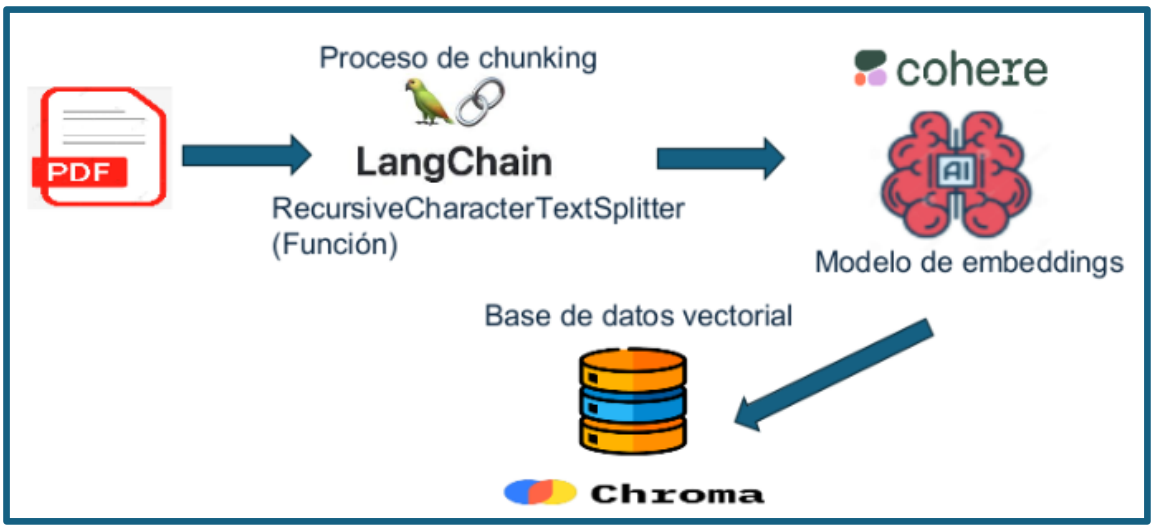
UOCRAIA se va a alimentar de dos fuentes de información:

- Convenio Colectivo de Trabajo 76/75
- Convenio Colectivo de Trabajo Yacimientos petrolíferos y gasíferos

Información técnica del proyecto

Arquitectura – Primera parte (Ingesta de documentos)

A continuación se describe la arquitectura propuesta, se va a dividir en dos partes para facilitar la explicación:

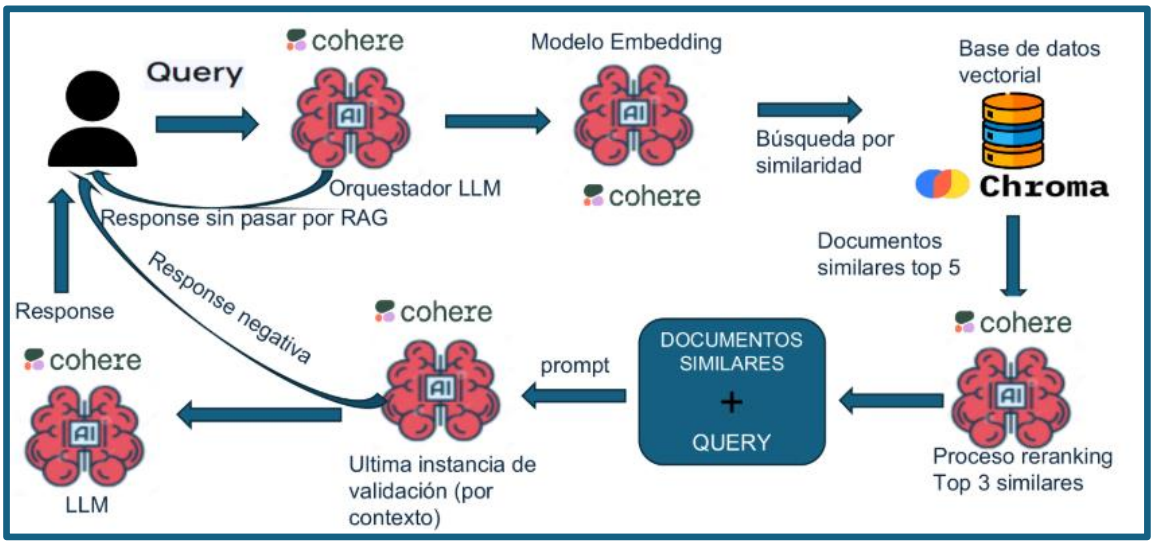


- Se van a ingestar los dos pdfs a nuestra aplicación por un endpoint (‘/POST/chromadb/{document_name} que va a recibir por parámetro el nombre del archivo.
- Este documento va a pasar por un proceso de chunking por medio de la función recursivecharactertextsplitter (tamaño de chunking de cada documento) proporcionada por el framework langchain.Opté por un tamaño de chunking(fragmentos más chicos del documento) que abarque cada articulo de los convenios, al estar estructurado la mayoría del contenido de los archivos de esa manera, me pareció buena estrategia dividirlos de esa forma. Esta información (tamaño de chunk, overlap, y demás datos que voy a necesitar) están precargados en el fichero de metadata.
- Después, estos chunks pasan por el modelo de embedding proporcionado por Cohere con el objetivo de obtener su representación vectorial, el modelo que utilicé fue “embed-multilingual-v3.0” porque es uno de los mas nuevos, acepta una dimensionalidad de 1024, que estimo que bastante potente y nos ayuda después a la búsqueda semántica que más adelante se va a detallar y por último es multi lenguaje, es decir soporta al español que para este proyecto es clave.

- Por último, se van a alojar en una base de datos vectorial, en este caso ChromaDB, que es ideal para almacenar y gestionar grandes cantidades de vectores.

  Es importante mencionar que todo este proceso no es necesario hacerlo ya que la base de datos vectorial se encuentra persistida. Lo cual significa que los documentos con sus respectivos embeddings ya van a estar en la base de datos vectorial cuando inicialicen la aplicación. Esto era un punto importante a realizar.

Arquitectura – Segunda Parte (Consulta del usuario)



- El proceso empieza por la consulta de un usuario, esta consulta va a pasar por una primera instancia que es el Orquestador LLm proporcionado por Cohere, este orquestador va a decidir si la consulta va a pasar por nuestro RAG o no, por ejemplo si el usuario nos dice “hola” “chau” “hasta luego” “gracias” “como te llamas?” etc, estas preguntas no requieren que pase por nuestro proceso y nuestro orquestador puede contestarlas sin ningún tipo de problema, de esta manera se otimiza mucho nuestra aplicación filtrando este tipo de consultas. En cambio, si la pregunta es: “es verdad que tengo que percibir un adicional por trabajar en las alturas? Porque es algo que nunca me han pagado”. Esta es una pregunta mucho mas compleja que va a pasar a la siguiente instancia de nuestro proceso.

- El siguiente paso es realizar el proceso embedding a la query del usuario (input_type = search query para búsquedas optimas) para posteriormente hacer el proceso de recuperacion (retrieve) a la base de datos vectorial, es decir , mediante el calculo de producto punto entre el vector de la consulta y los vectores almacenados nos ayuda a medir la similitud entre ellos, identificando los chunks mas relevantes para obtener nuestro contexto y así poder responder a la consulta del usuario.
- Luego, se pasa a la siguiente instancia de nuestra aplicación que es el proceso reranking. Después de calcular la similitud, se aplica un nuevo modelo para evaluar y reordenar los resultados según su relevancia real. Utilicé el modelo rerank-v3.5 porque es uno de los modelos más nuevos y optimizados de Cohere y además es multilenguaje,
- Una vez obtenido nuestro contexto recuperado de nuestra base de datos vectorial y haber pasado por el proceso de reranking, se pasa por una ultima instancia de validación. Que consiste en que el modelo basándose del contexto como su fuente de información va a analizar si la consulta del usuario puede ser respondida o no, si no puede ser respondida (por ejemplo “te gusta la coca cola?”) el proceso termina con un mensaje: “Lo lamento, solo puedo responder preguntas que estén relacionadas con los convenios colectivos de trabajo ¿tienes alguna otra pregunta en mente?”. En cambio, si la consulta puede ser contestada, pasa a la instancia final.
- Por último, habiendo pasado por todas las instancias previas, el modelo está preparado para responder a la consulta del usuario y así finalizando el proceso.

  Es importante mencionar que para cada instancia que se haya involucrado un modelo de IA generativa, se le especificó un prompting con instrucciones claras, distintas personalidades y/o roles de como debía responder en cada etapa para poder llegar al resultado final. Además, que también por lo general asignándole una temperatura muy baja para evitar “alucinaciones” y que para que cada pregunta se conteste de la misma forma.

Tecnologías utilizadas

Se utilizaron varias tecnologías, algunas explicadas en este curso y otras no que investigué y apliqué para este challenge Final:

Frontend:

Una de las sugerencias que seguí fue implementar una interfaz gráfica para facilitar la interaccioón con la API, para esto utilicé:

- **Streamlit:** librería de Python que me facilitó realizar toda la interfaz gráfica de una manera sencilla y rápida. Al principio pensé por usar React pero luego investigando un poco opté por streamlit ya que también se usa Python y es recomendable para productos MVP.

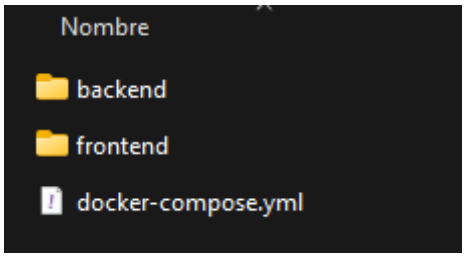
Backend:

- **FastAPI:** framework ideal para construir APIs con Python
- **ChromaDB:** Base de datos vectorial optimizado para la búsqueda y almacenamiento de embeddings.
- **Cohere:** Plataforma de IA que ofrece modelos de lenguaje para diversas funciones, en este proyecto se utilizó para generación de texto y procesamiento de embeddings.
- **Langchain:** Framework open-source diseñado para facilitar la creación de aplicaciones basados en arquitectura RAG. En este caso se utilizó para la técnica de chunking y herramientas como PyPDF para leer el contenido de un PDF.

Es importante mencionar que para todo el despliegue (tanto frontend como backend) fueron gestionados y configurados con **Docker,** herramienta de código abierto que investigué para aplicar en este proyecto facilitándome generar un contenedor, que me provee de un entorno aislado y ligero que incluye todo lo necesario para ejecutar la aplicación en distintas computadoras, como el código y las dependencias, por ejemplo.

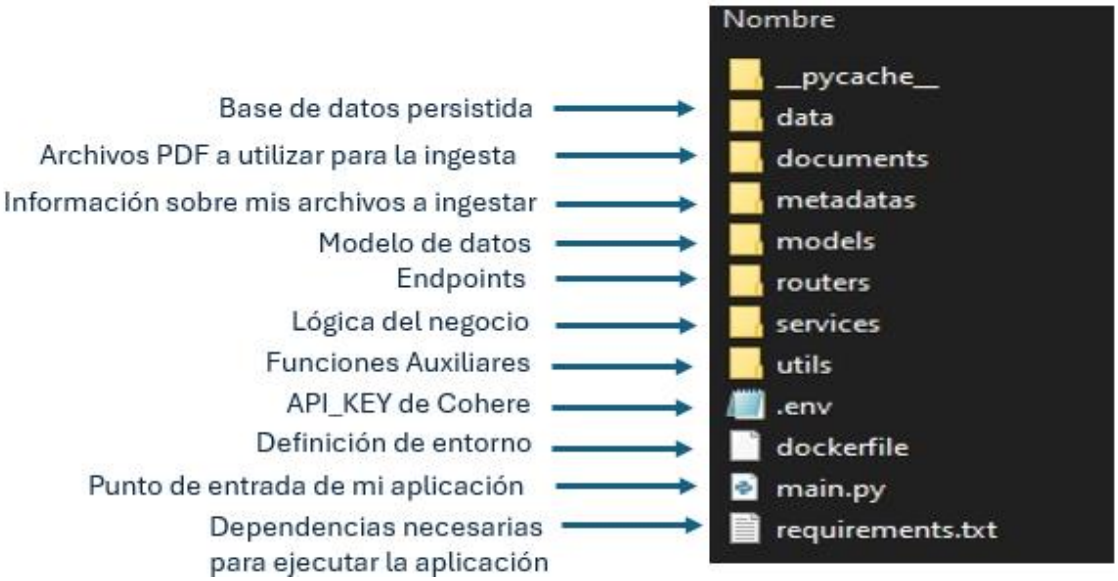
Estructura de la aplicación

Dentro del fichero app, nos encontraremos con lo siguiente:



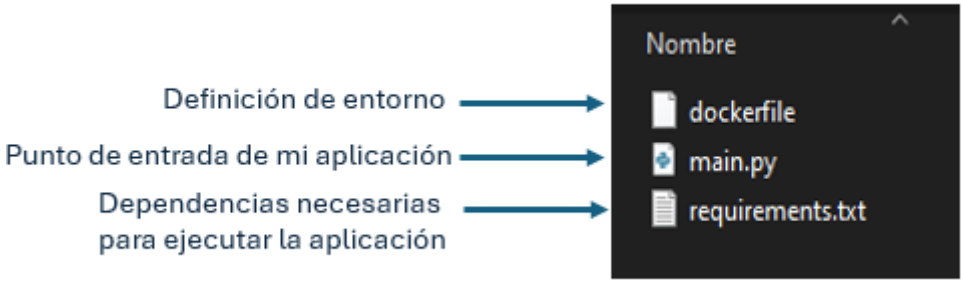
Dos ficheros principales: uno para el backend, que maneja la lógica de negocio, el procesamiento de datos y la comunicación con la base de datos, y otro para el frontend, que gestiona la interfaz de usuario e interactúa con el backend para mostrar la información. Sumado a esto tenemos un docker-compose.yml, esto nos facilita la creación, configuración y administración de aplicaciones multicontenedor, permitiendo iniciar todos los servicios necesarios con un solo comando.

Backend:



Lo organicé de esta manera para que esté lo mas modularizado posible, siguiendo buenas prácticas y ayudando a que el código sea más legible.

Frontend:



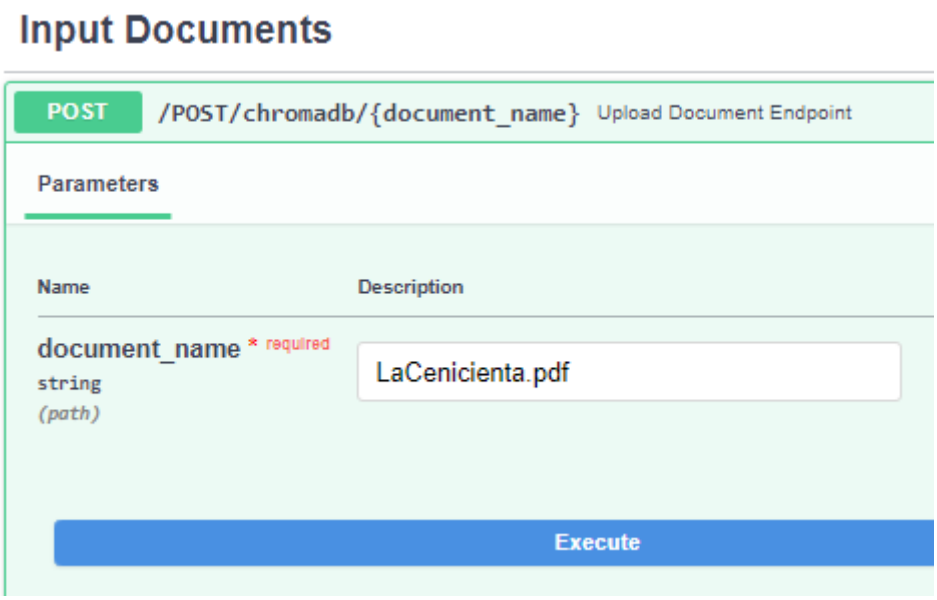
Paso a paso de instalación

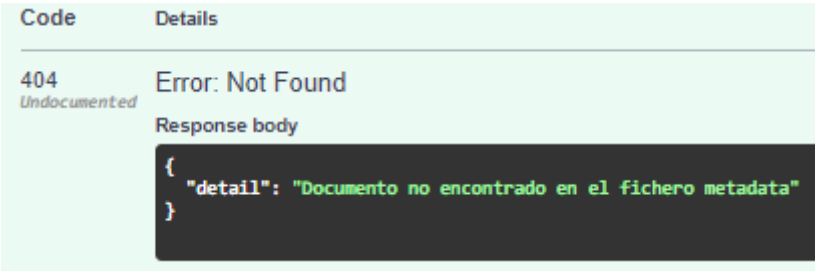
- Se debe tener Docker Desktop instalado en la computadora, si no lo tiene lo puede descargar acá
- Descargar el repositorio del proyecto en GitHub
- Paso de sugerencia: dentro del fichero backend, modificar el archivo .env y cambiar el código de api_key de Cohere por el propio de ustedes.
- Abrir CMD, posicionarse en el fichero /app del proyecto y ejecutar el comando: “docker-compose up --build” sin las comillas para que levante los servicios y cree los contenedores. Los servicios de backend y frontend se ejecutarán en http://localhost:8000/ y http://localhost:8501/ respectivamente.

Ejemplos de prueba

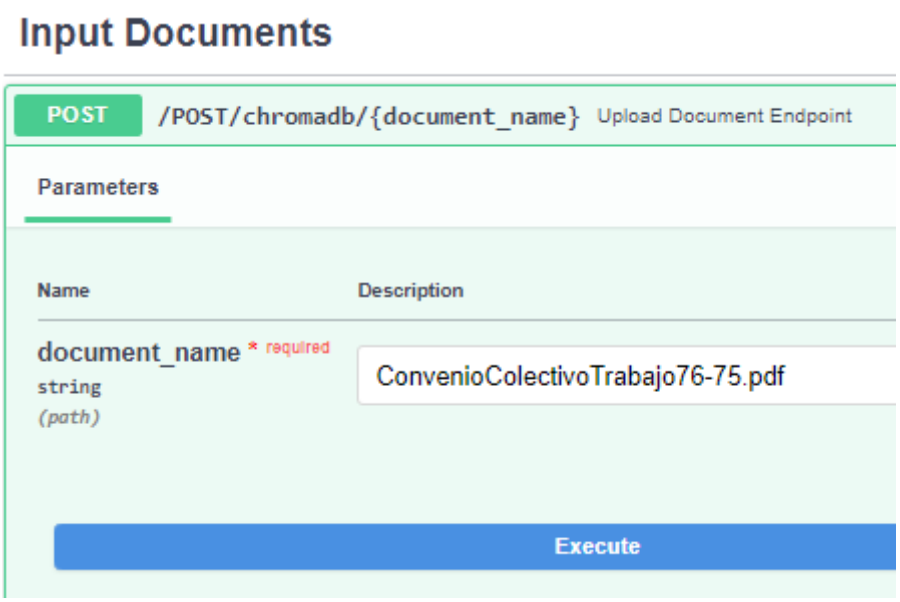
Carga de documentos

- Cargar un documento que no se encuentra en el fichero metadata generará un código de error http 404:

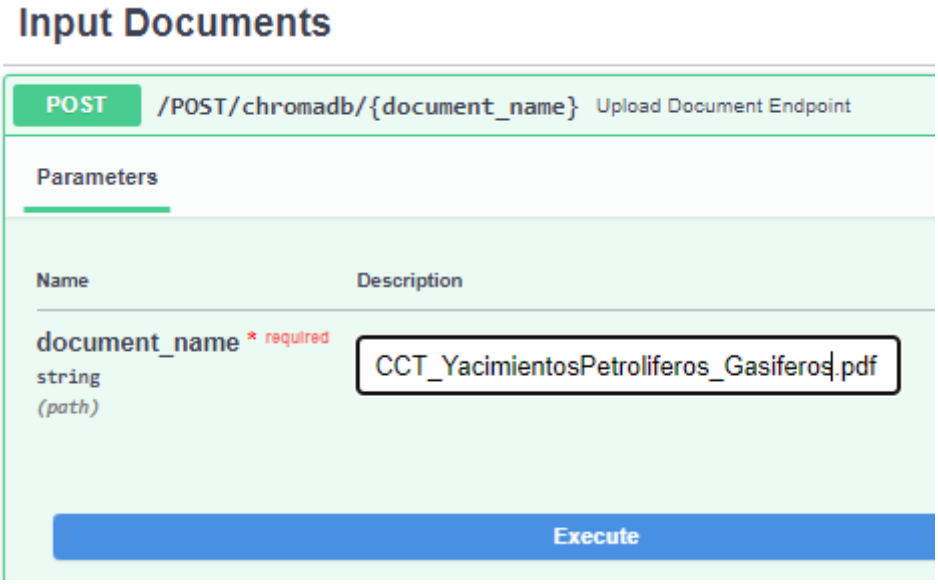


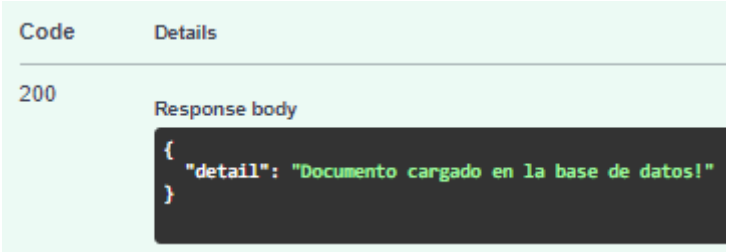


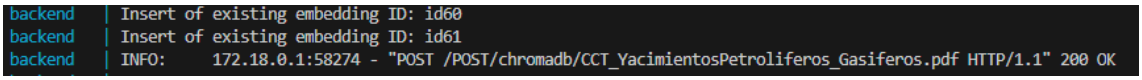
- Cargar un documento a la base de datos vectorial que ya se ha cargado previamente generará un error http 409



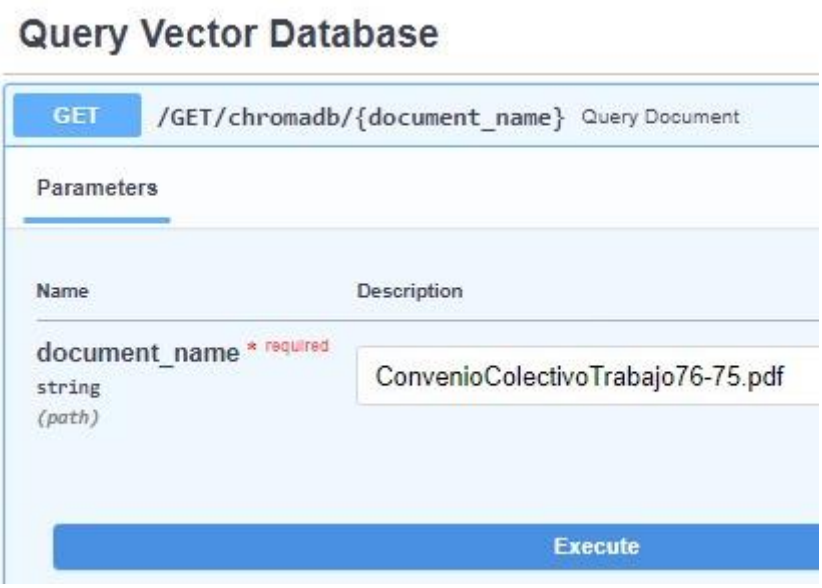
- Ejemplos exitosos:

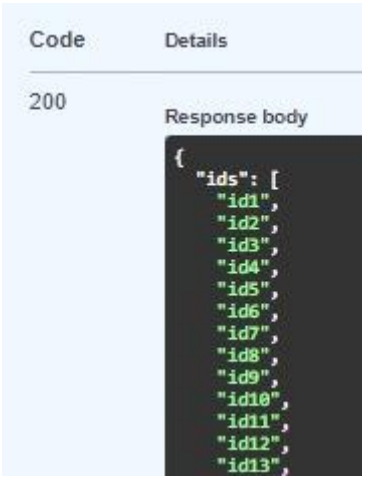






Después existe otro método GET, que no se especificó en el enunciado pero lo creé a modo de prueba para consultar lo que tenía mi colección en ChromaDB

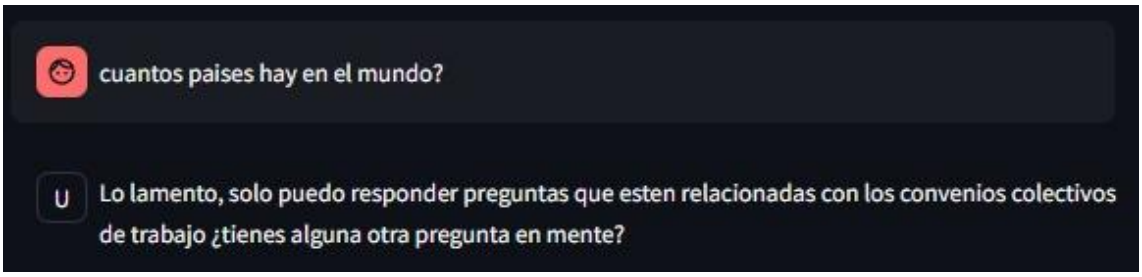


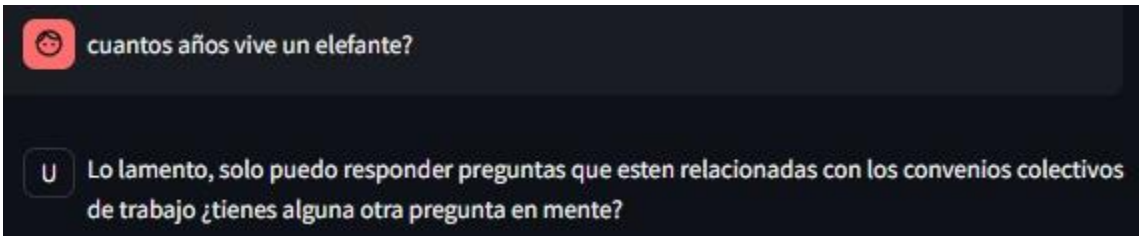


Consulta query del usuario

- Ejemplo que no pasa por RAG:

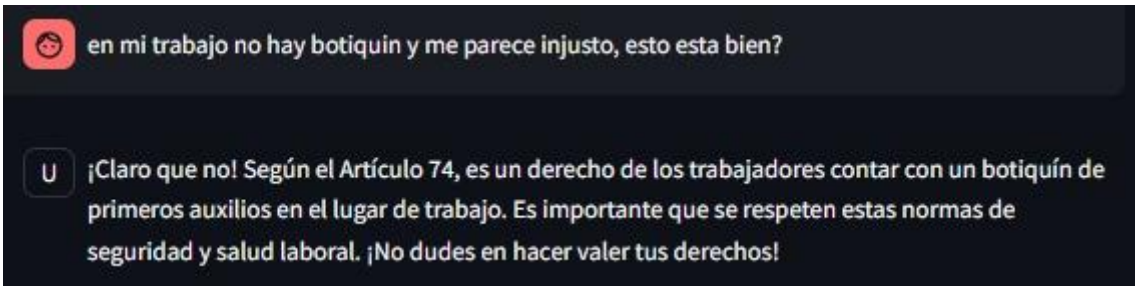
- Ejemplos que no pasa la validación por contexto:

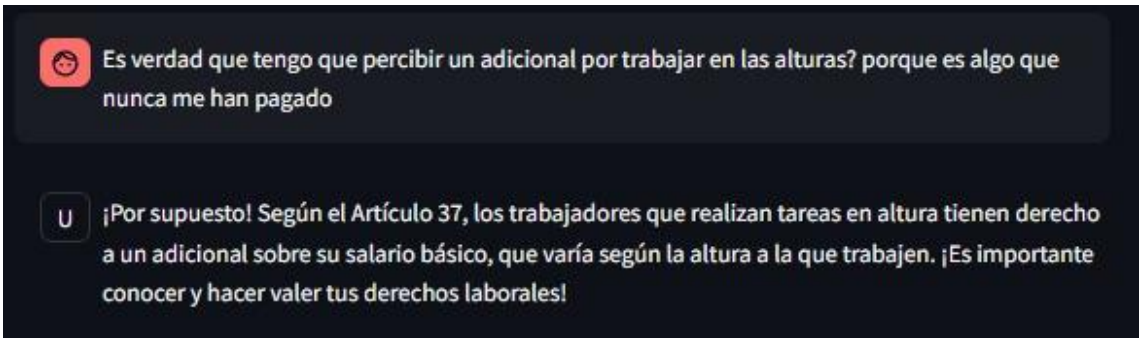


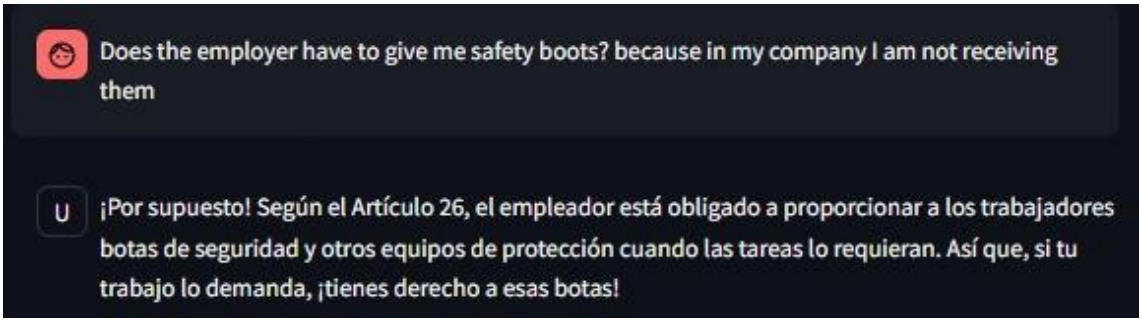


Téngase en cuenta que, al no poder contestar la pregunta de acuerdo al contexto, siempre va a contestar de la misma manera

- Ejemplos de éxito:







También entiende las preguntas en otros idiomas y responde en español.

Conclusión personal

Siento que aprendí muchísimo a lo largo del curso y espero haberlo demostrado en este challenge final. Soy muy apasionada a la tecnología y más que nada la inteligencia artificial. Considero que la tecnología tiene que ser un cambio cultural grande en todos los ámbitos y no centrarse solamente en un sector, siento que aporté con un granito de arena a esta nueva revolución industrial que se está viviendo hoy en día. Es por esto que me inspiré en crear UOCRAIA.

La verdad es que se me pasó el tiempo volando este mes con ustedes. Estoy muy agradecida con PI por la oportunidad.