**Documentación del Chatbot RAG Promtior**

**1. Objetivo de la documentación**

En este documento presento cómo abordé el reto de construir un asistente conversacional para Promtior utilizando un enfoque RAG. Aquí detallo la lógica de implementación, los principales desafíos y las tecnologías empleadas para que pueda valorarse la solución propuesta.

**2.Overview del proyecto**

Este proyecto monta un pipeline RAG que usa un PDF como única fuente de conocimiento. Primero cargo y divido el documento en fragmentos con LangChain, luego genero embeddings con MiniLM y los indexo en FAISS. Cuando llega una consulta, recupero los trozos más relevantes, compongo un prompt y lo envío al modelo Mixtral-8x7B vía Together.ai. Finalmente limpio y post-proceso la respuesta antes de devolverla al usuario a través de FastAPI.

**3.Funcionamiento del proyecto**

Este proyecto implementa un chatbot basado en la arquitectura RAG que responde preguntas sobre la compañía Promtior tomando su información directamente del pdf, mas adelante voy a explicar porque opté por usar como fuente de conocimiento el pdf y no hice scraping a la página de la compañía.   
Cuando un usuario envía una consulta, el sistema:  
1. Divide el PDF en fragmentos mediante langchain

2. Genera embedding con el modelo “all-miniLM” de Hugging Facel.

3. almacena y recupera vectores usando FAISS

4. Recupera los fragmentos más relevantes y los combina en un promt

5. Envía el prompt a un modelo LLM “Mixtral”.

6. Post-procesa la respuesta, filtra prefijos, saludos y referencias, y la devuelve al usuario.

**4.Tecnologías usadas**Python 3 con FastApi y Uvicorn  
Pipeline RAG: LAngChain + FAISS + Embeddings MiniLM  
LLM remoto: Mixtral-8x7B vía [Together.ai](http://together.ai)  
Despliegue: AWS EC2 (SSH + nohup) + Vercel para el frontend

Next.Js + Redux Toolkit + Redux Persist + Tailwind: frontend para probar el chatbot de manera más cómoda

**5.Estructura de carpetas del proyecto**

**api/routes/chat.py**: endpoint de la API  
**core/embeddings.py:** lógica de creación y carga del índice FAISS

**core/prompts.py:** definición del system prompt y plantillas  
**core/rag\_chain.py:** función ‘ask\_question’’ con flujo de RAG y saludos  
**doc/promtior.pdf:** fuente de conocimiento

**doc/diagrama.png:** Diagrama que pide el challenge  
**models/faiss\_index:** índice FAISS generado automáticamente

**main.py:** inicializacion de FastApi

**6.Desafios encontrados**

Encontré desafío en todo. Nunca había usado Python, pero me enfoqué en ver qué tecnologías ocupar e investigué la arquitectura de carpetas. Me apoyé en la IA para que me ayude a “modularizar” (como hacemos en React) los distintos módulos de Python. Vi videos, revisé la documentación de LangChain y exploré repositorios de chatbots, porque nunca había implementado uno.

Una vez que arranqué con el código, tuve problemas con Ollama y Llama 2 estos respondían, pero tardaban más de un minuto. Al investigar descubrí que requieren GPU, y como no tengo, opté por un modelo remoto más liviano.

La IA me recomendó Llama 3, pero recibía mal el contexto y en vez de dar respuestas concisas me lanzaba más preguntas. Finalmente usé Mixtral-8x7B, que es más robusto y seguro, y las respuestas mejoraron muchísimo.

También, con preguntas vagas como “¿qué ofrece?” o “¿cuándo se fundó?”, el modelo no sabía de quién hablaba y devolvía fragmentos genéricos de cualquier parte del PDF.

Intenté scraping de la web, pero las respuestas no coincidían con mis preguntas y devolvió información genérica o errónea, Por eso elegí el PDF como fuente de conocimiento porque además de describir de manera breve los servicios que ofrece, también contiene fecha exacta de la fundación de la compañía.

**7.Links de redes personales**

* [*Linkedin*](https://www.linkedin.com/in/joseenriquez80/)
* [*GitHub*](https://github.com/JoseEnriquez88)
* [*Portfolio*](https://enriquez-jose.vercel.app/)
* *Mail:* [*enriquez.jose@gmail.com*](mailto:enriquez.jose@gmail.com)