PROYECTO

PROMTIOR CHATBOT

Promtior

ENERO
2026

Resumen

En este proyecto se implementó un chatbot para la empresa Promtior capaz de responder preguntas sobre la misma utilizando agentes de IA con capacidades de Retrieval-Augmented Generation (RAG).
Los datos disponibles para el agente serán la página web de la empresa y un pdf que contiene información adicional.

Componentes

- **Backend (python):** API que utiliza langserve y FastAPI para disponibilizar un agente de IA implementado con la librería langchain. Permite recuperar información de las sesiones e invocar el agente, permitiendo la posibilidad de streamear el contenido.

- **Frontend (React + vite):** Aplicación web que interactua con el backend para dar una mejor experiencia de usuario y una mejor interfaz gráfica al usuario final.

- **Infraestructura**
  - **Vectorstore (FAISS):** Almacena embeddings sobre el pdf y la web que contienen información relevante de la empresa.
  - **Azure:** VM en la nube que contiene la aplicación de producción.
  - **OpenAI (externo):** Motor de inferencia externo.
  - **Ollama**: Motor de inferencia local.
  - **Redis**: Base de datos en memoria para persistencia de conversaciones.
  - **NGINX**: Reverse proxy que maneja el enrutamiento.
  - **Docker**: Orquestación de todos los servicios con red aislada.
  - **GithubActions**: Despliegue automático de toda la aplicación en VM de Azure.

Flujo de datos

- El usuario selecciona el modelo que quiere utilizar (Ollama u OpenAI)
- El usuario envía una pregunta desde el frontend
- NGINX enruta la petición al contenedor API
- El agente LangChain busca contexto relevante en FAISS
- Se consulta Redis para mantener el historial de conversación
- El contexto + pregunta se envían a Ollama/Mistral u OpenAI/gpt-4o-mini
- La respuesta generada se devuelve al usuario

Desafios

- **Uso de langhcain:** La implementación de una solución en langchain sin previos conocimientos resultó un desafio. Si bien la documentación es precisa, se tuvieron múltiples dificultades para determinar las librerías y sintaxis correcta debido a múltiples versiones y cambios recientes sobre la librería.

- **Comprensión del funcionamiento completo del agente:** Dado que la librería presenta múltiples abstracciones, fue necesario profundizar en las mismas para comprender el flujo de datos.

- **Uso de múltiples modelos para el agente:** se decidió utilizar ambos modelos propuestos, Ollama (local) y OpenAI (externo), lo cual llevo a múltiples problemas, principalmente debido a que la librería esta preparada y pensada para trabajar sobre un único modelo.

- **Lectura del stream desde frontend:** el código utilizado para la correcta lectura del stream fue implementado desde cero dada la falta de experiencia al trabajar con este tipo de respuestas. Sumado a esto, la estructura del stream varía entre modelos.

- **Tiempo de construcción de imagen de backend:** Al construir la imagen del backend se instalan las librerias necesarias de python, lo cual aumenta considerablemente el tiempo de ejecución del pipeline. Se utilizó una cache de docker para no reinstalar las librerías en cada ejecución.