Project Overview

Overview

The objective was to implement a RAG-based chatbot capable of answering questions about the Promtior website using **LangChain** and **LangServe**, as specified in the assessment guidelines.

The final architecture includes:

- **LangChain** for RAG orchestration
- **LangServe (FastAPI)** to expose the chain as an API
- **Vector store** for semantic retrieval
- **Ollama (llama2)** as LLM
- **Streamlit UI** as chat interface
- Website content + PDF presentation as indexed sources

Project Overview

Overview

The objective was to implement a RAG-based chatbot capable of answering questions about the Promtior website using **LangChain** and **LangServe**, as specified in the assessment guidelines.

The final architecture includes:

- **LangChain** for RAG orchestration
- **LangServe (FastAPI)** to expose the chain as an API
- **Vector store** for semantic retrieval
- **Ollama (llama2)** as LLM
- **Streamlit UI** as chat interface
- Website content + PDF presentation as indexed sources

Model Evaluation and Architectural Decisions

Although the assessment allowed either OpenAI or Ollama, I decided to evaluate multiple configurations to determine the most optimal solution:

1. **OpenAI (GPT-4o-mini and GPT-4o)**
2. **Ollama (llama2)**
3. Different embedding configurations (Ollama embeddings vs OpenAI embeddings)

The goal of this experimentation was to evaluate:

- Answer quality
- Retrieval precision
- Latency
- Cost and deployment complexity

Through testing, I observed that **Ollama (llama2)** produced more consistent answers for this specific RAG use case, especially when retrieving structured company information such as services and founding date.

For embeddings, I evaluated alternatives and selected the configuration that provided the most stable retrieval behavior.

This iterative experimentation allowed me to validate that the final architecture was not chosen arbitrarily, but based on comparative performance.

UI Decision

Although not explicitly required, I added a **Streamlit UI** to improve usability and evaluation experience.
This avoids CLI-based interaction and provides a conversational interface while preserving the required LangChain + LangServe architecture.

Main Challenge: Deployment

The primary challenge was deployment, not implementation.

Railway

Initial deployment attempt used Railway for simplicity.
However, Ollama models exceeded the resource limits of the free tier, causing instability.

AWS EC2

A second attempt used AWS EC2 for greater resource control.
Instance-type and account-level quota limitations restricted deployment within the available timeframe.

Final State

- Full RAG pipeline works locally via Docker Compose.
- Architecture is containerized and cloud-ready.
- Deployment steps are documented.
- Cloud deployment remains a next step.

Next Steps

- Cloud deployment on AWS with appropriate instance sizing
- Unit and integration testing
- Observability (logging, metrics, tracing)
- Retrieval optimization (re-ranking / improved query strategies)