Promtior Challenge

Participant: Santiago Díaz
For the position of: AI Engineer

1. Project Overview

The goal of this project is to develop a Retrieval-Augmented Generation (RAG) chatbot that answers questions about Promtior's services and founding date. The chatbot uses LangChain for the RAG pipeline, Ollama for running the llama2 model locally, and FastAPI for serving the application as an API.

Key Features

- Data Ingestion: Scrapes Promtior's website and parses the challenge PDF to create a combined dataset.
- RAG Pipeline: Uses LangChain to create a retrieval-augmented generation pipeline with open-source embeddings and the llama2 model.
- API Endpoint: Exposes the chatbot functionality via a FastAPI endpoint.
- Cloud Deployment: Deploys the FastAPI app to AWS ECS using AWS Copilot, with Ollama hosted on an EC2 instance.

2. Project Structure

The project is organized into three main folders:
data/

- Contains raw and processed data:
    - website_text.txt: Text scraped from Promtior's website.
    - pdf_text.txt: Text extracted from the challenge PDF.
    - combined_data.txt: Combined text used as context for the RAG pipeline.

src/

- Contains the application logic:
    - data_scraping.py: Scrapes Promtior's website and saves the text.
    - pdf_parser.py: Parses the challenge PDF into text.
    - data_combiner.py: Combines the website and PDF text into a single file.
    - my_rag_pipeline.py: Implements the RAG pipeline using LangChain.
    - app.py: Defines the FastAPI endpoint and starts the server.

doc/

- Contains project documentation (this pdf).

3. Implementation Details

Data Preparation:

I created 3 scripts to generate data about Promtior:

- data_scraping.py scrape Promtior's website and extract relevant text that then is saved in data/website_text.txt
- pdf_parser.py parse the challenge pdf to text and save it in data/pdf_text.txt
- data_combiner.py combines both .txt archives into one called data/combined_data. Which is the one I use to give context to the rag chain

RAG Pipeline:

After having that I created the file my_rag_pipeline.py that contains all the logic of the solution:

- Loads and splits into chunks the previously mentioned data
- Creates the Vector Store using an Open-source HuggingFace embeddings, then the embeddings are stored in a FAISS vector store for efficient similarity search.
- The llama2 model is initialized using Ollama and configured to run locally
- Defines the prompt template to guide the LLM in generating responses based on the retrieved context
- The retrieval and generation components are combined into a single RAG chain using LangChain

API Endpoint:

The app.py script defines a FastAPI endpoint (/promtior-chatbot) that accepts user queries and returns responses generated by the RAG pipeline. The API is served using Uvicorn.

4. Local Testing

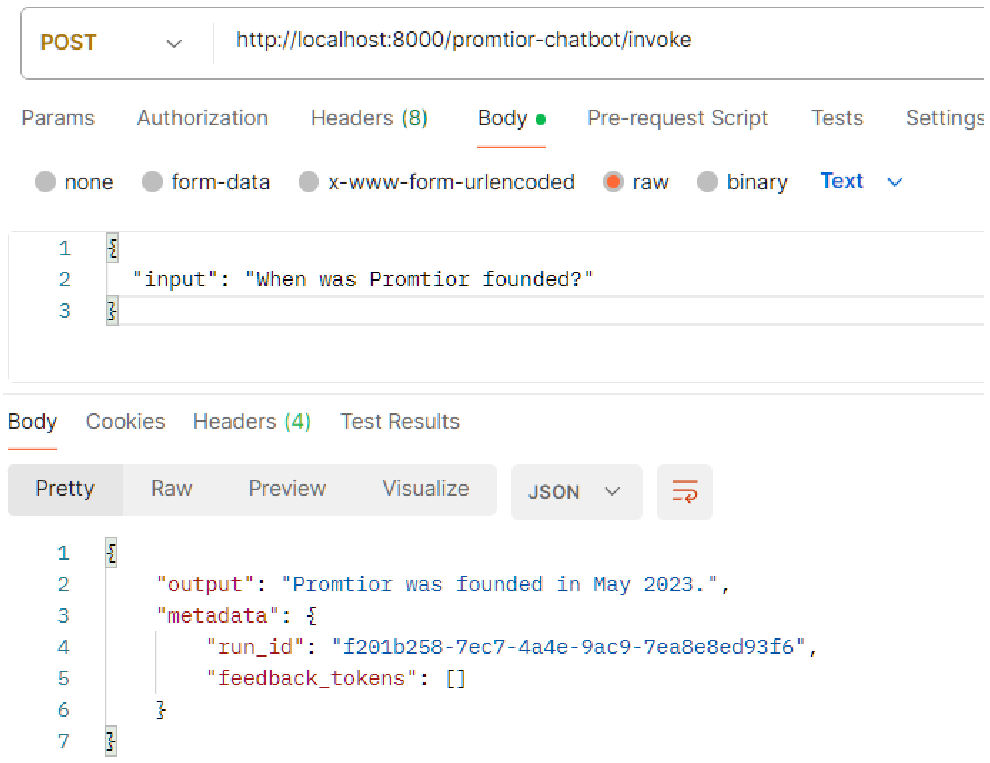
1. Start the Ollama server and pull the llama2 model:

```
ollama serve
ollama pull llama2:7b
```

1. Start the FastAPI server:

```
uvicorn src.app:app --host 0.0.0.0 --port 8000
```

Test with Postman:
Send a POST request to http://localhost:8000/promtior-chatbot/invoke with a JSON payload:



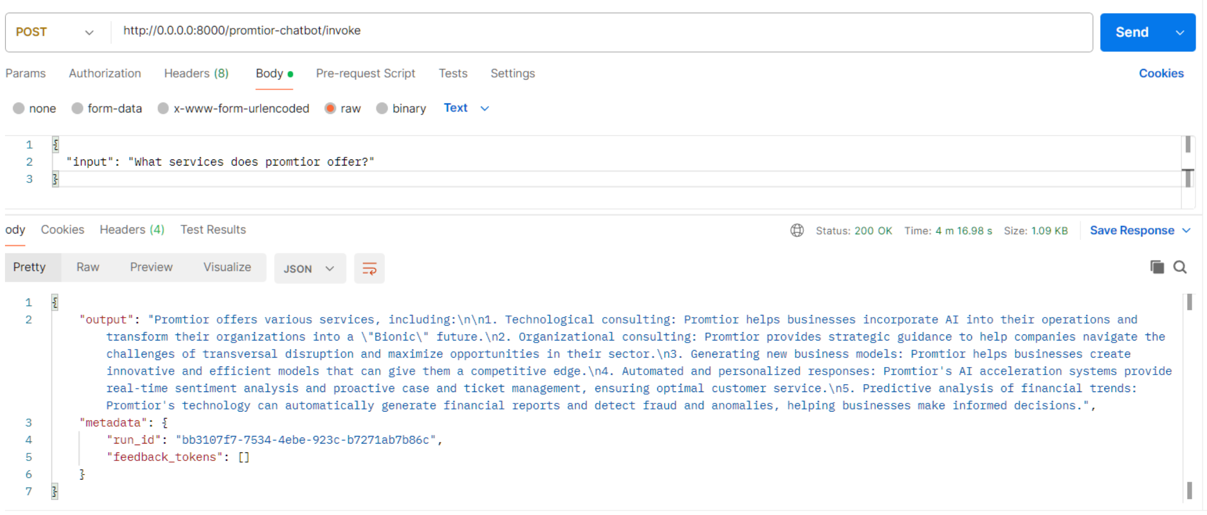
POST http://localhost:8000/promtior-chatbot/invoke

Body: raw, Text

```
{
  "input": "When was Promtior founded?"
}
```

Response Body (Pretty, JSON):

```
{
    "output": "Promtior was founded in May 2023.",
    "metadata": {
        "run_id": "f201b258-7ec7-4a4e-9ac9-7ea8e8ed93f6",
        "feedback_tokens": []
    }
}
```



POST http://0.0.0.0:8000/promtior-chatbot/invoke

Body: raw, Text

```
{
 "input": "What services does promtior offer?"
}
```

Status: 200 OK Time: 4 m 16.98 s Size: 1.09 KB

Response Body (Pretty, JSON):

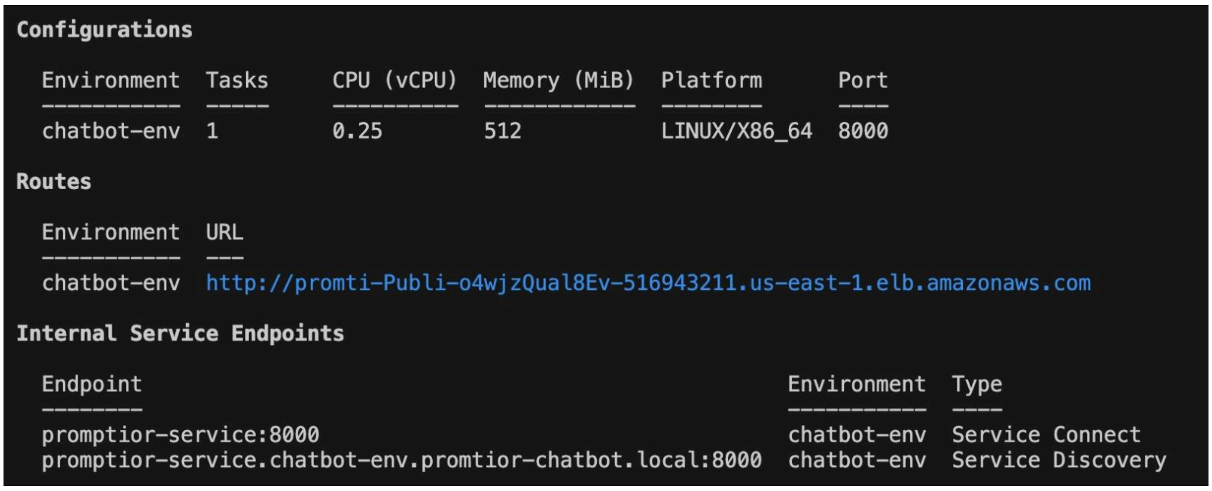
```
{
    "output": "Promtior offers various services, including:\n\n1. Technological consulting: Promtior helps businesses incorporate AI into their operations and
        transform their organizations into a \"Bionic\" future.\n2. Organizational consulting: Promtior provides strategic guidance to help companies navigate the
        challenges of transversal disruption and maximize opportunities in their sector.\n3. Generating new business models: Promtior helps businesses create
        innovative and efficient models that can give them a competitive edge.\n4. Automated and personalized responses: Promtior's AI acceleration systems provide
        real-time sentiment analysis and proactive case and ticket management, ensuring optimal customer service.\n5. Predictive analysis of financial trends:
        Promtior's technology can automatically generate financial reports and detect fraud and anomalies, helping businesses make informed decisions.",
    "metadata": {
        "run_id": "bb3107f7-7534-4ebe-923c-b7271ab7b86c",
        "feedback_tokens": []
    }
}
```

Now my application is correctly running on my local machine.

5. Cloud Deployment

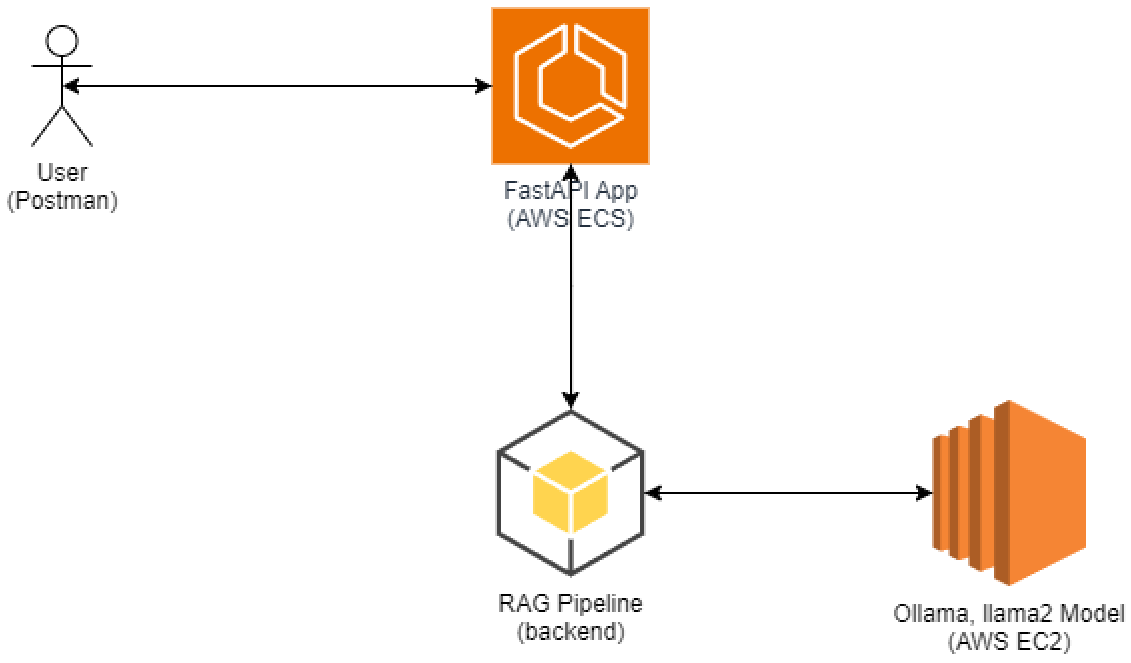
First to ensure the application has access to Ollama, I deployed an instance of Ollama in EC2 making sure that the application has access to it through the url: `http://ec2-3-85-233-72.compute-1.amazonaws.com:11434`

Then to deploy the API I created a Dockerfile and built a docker image with the fastAPI and then deploy it in ecs aws with the command: `copilot init --app [application-name] --name [service-name] --type 'Load Balanced Web Service' --dockerfile './Dockerfile' --deploy`



```
Configurations

  Environment  Tasks     CPU (vCPU)  Memory (MiB)  Platform     Port
  -----------  -----     ----------  ------------  --------     ----
  chatbot-env  1         0.25        512           LINUX/X86_64  8000

Routes

  Environment  URL
  -----------  ---
  chatbot-env  http://promti-Publi-o4wjzQual8Ev-516943211.us-east-1.elb.amazonaws.com

Internal Service Endpoints

  Endpoint                                                Environment  Type
  --------                                                -----------  ----
  promptior-service:8000                                  chatbot-env  Service Connect
  promptior-service.chatbot-env.promtior-chatbot.local:8000  chatbot-env  Service Discovery
```

6. Component Diagram



Explanation of Interactions

- User → FastAPI App:

The user sends a query (e.g., "What services does Promtior offer?") to the FastAPI endpoint.

- FastAPI App → RAG Pipeline:

The FastAPI app forwards the query to the RAG pipeline.

- RAG Pipeline → Ollama on EC2:

The RAG pipeline sends the query and retrieved context to the llama2 model hosted on Ollama (EC2).

- Ollama on EC2 → RAG Pipeline:

The llama2 model generates a response and sends it back to the RAG pipeline.

- RAG Pipeline → FastAPI App:

The RAG pipeline returns the generated response to the FastAPI app.

- FastAPI App → User:

The FastAPI app sends the response back to the user.

7. Challenges

The main challenge I encountered was deploying the application in the cloud because Ollama needed to be running and accessible to the FastAPI API. To solve this, I deployed Ollama on an EC2 instance, configured it to listen on all interfaces, and ensured the FastAPI app could connect to it by pointing to the EC2 instance's public IP. Additionally, I used AWS Copilot to automate the deployment of the FastAPI app to ECS, simplifying the process and ensuring scalability.