Chatbot assistant Promtior

Technical test - AI Engineer

Facundo Barboza

facubarboza1702@gmail.com

09/11/2024

Table of Contents

1. Project Overview

The Promtior Assistant chatbot was created to respond to user inquiries by utilizing rich context from a variety of sources, such as PDF documents and webpages. Contextualized replies and conversational logic are handled by the solution using the LangChain framework.

Approach

Data Extraction: Information is retrieved from the Promtior website (www.promtior.ai) and the PDF document of the Technical test (AI Engineer). Both sources are combined into a single vector store for efficient querying.

Question Processing: The system reformulates user questions based on the chat history and retrieves relevant answers using a GPT-powered model. For that, there was a connection to the OpenAI API.

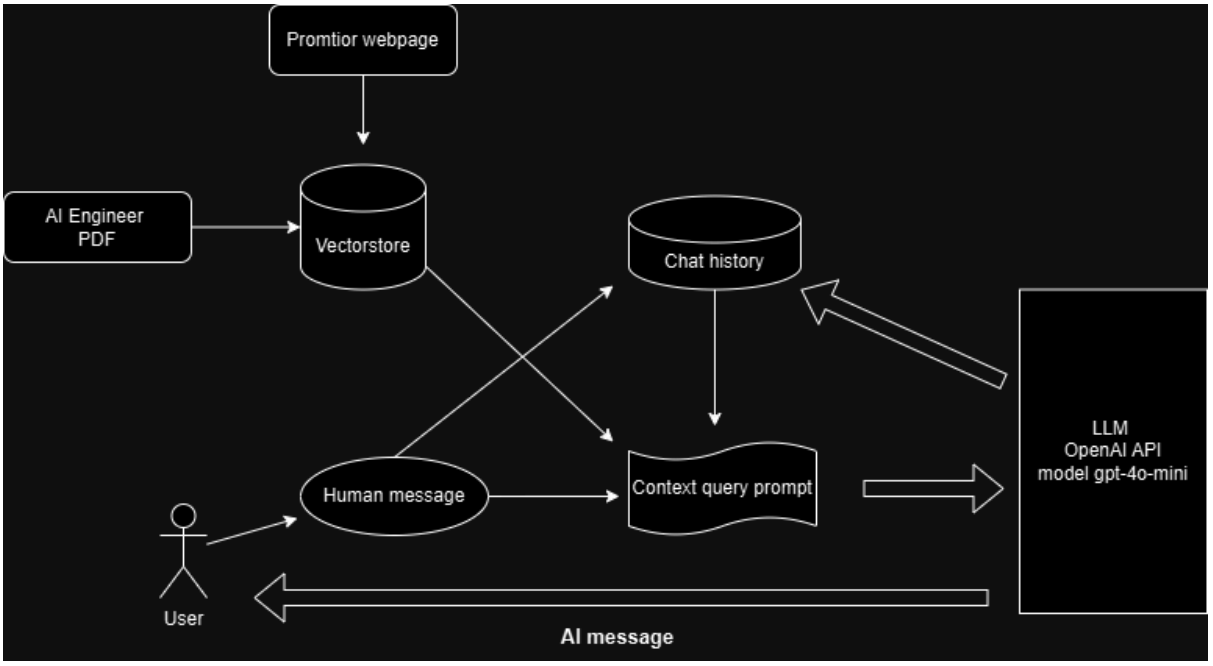
Deployment: The application was containerized with Docker and deployed to *Railway*.

Challenges and Solutions

One of the challenges faced was combining information from multiple sources.In order to solve this the webpage and PDF content were processed and indexed into a unified vector store.

Another one to mention might be providing accurate, contextualized responses.That is the reason why LangChain was used to reformulate user queries based on prior conversation history.

2. Component Diagram



The diagram illustrates the architecture of the Promtior assistant chatbot and how it processes user queries by utilizing multiple data sources and maintaining a conversational context.

Flow:

- The *user* sends a query and it is processed as a *HumanMessage*.
- The chatbot retrieves relevant data from the *vectorstore* (web page and PDF) and combines it with the *chat history*.
- The *context query prompt* is generated, which enriches the input for the *LLM*.
- The LLM processes the prompt and generates a response (*AI Message*).
- The response is returned to the *user* and appended to the *chat history*.

3. Implementation Details

Files

- **app.py**:
    - The primary entry point for the application.
    - Exposes two routes:
        - /: A welcome page.
        - /chat: A POST endpoint for handling user queries and returning chatbot responses.
    - Manages chat history to provide context for follow-up questions.

- **chatbot_pipeline.py**:
    - Defines the main pipeline for processing user queries:
        - Loads data from the web page and PDF. For the web page context retriever, the WebBaseLoader module was used. And in the case of PDF, PyPDFLoader.
        - Uses LangChain to manage the query resolution and response generation.
    - Handles contextualization for user questions.

- **Dockerfile**:
    - Encapsulates the application in a Docker container for portability and easy deployment.

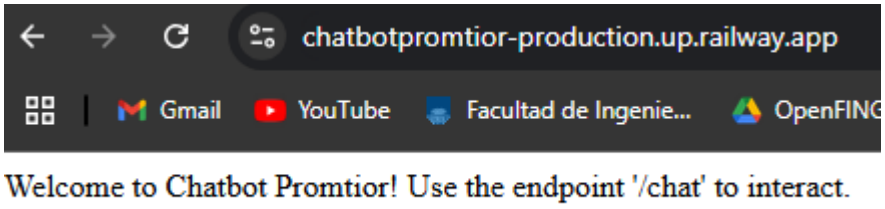
Technologies Used

- **LangChain**: For conversation management and context handling.
- **Flask**: To expose the chatbot as a web API.
- **Docker**: To package the application for deployment.
- **Railway**: To simplify the deployment process.

4. Deployment

The chatbot was deployed on **Railway** using the project's GitHub repository. Railway streamlined the deployment process with the following steps:

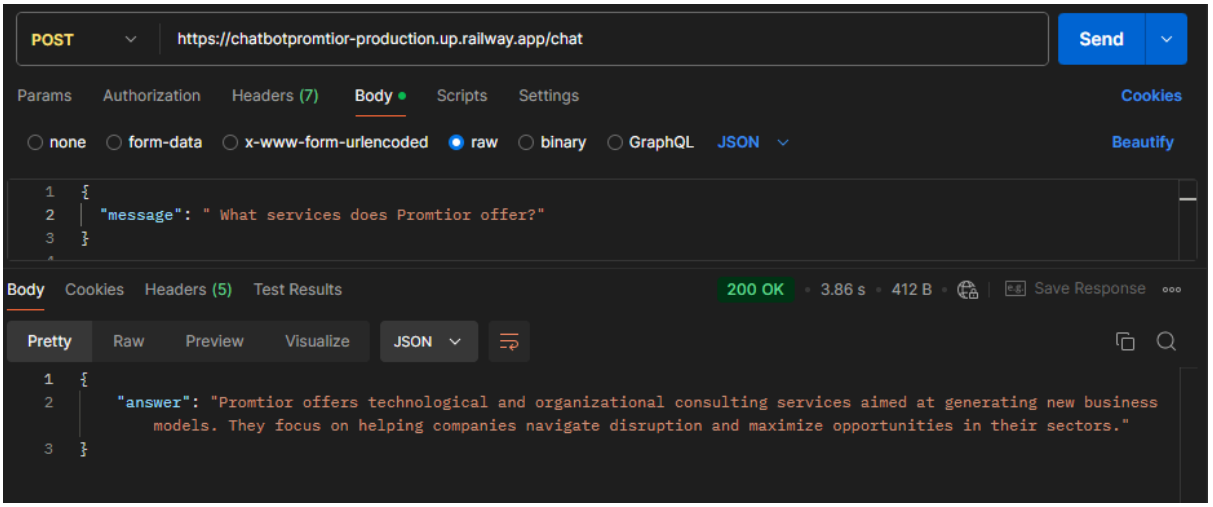
1. **Project Setup**:
    - The GitHub repository was connected directly to Railway.
    - Railway automatically detected the Dockerfile and built the environment based on the specified dependencies.
2. **Environment Variables**:
    - Required keys (*LANGCHAIN_API_KEY, OPENAI_API_KEY*) were configured directly in Railway's environment settings.
3. **Testing**:
    - Railway provided a public URL to test the chatbot's functionality post-deployment. This URL is the following: https://chatbotpromtior-production.up.railway.app/

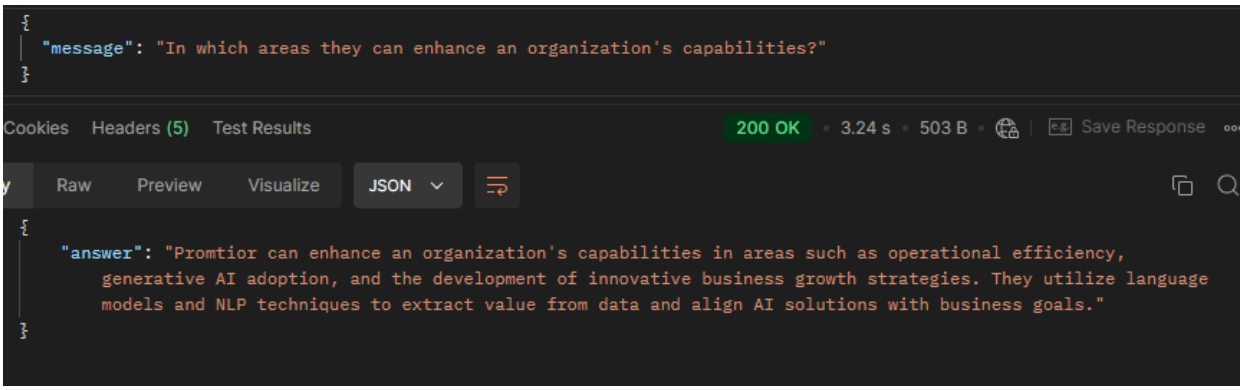


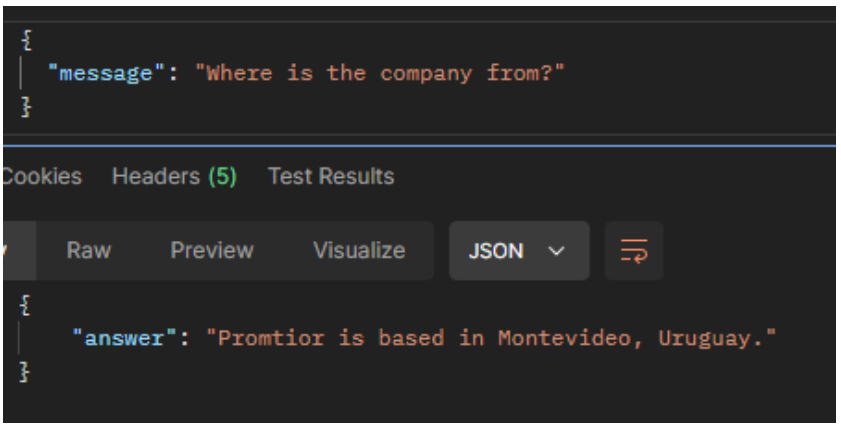
This approach simplified the deployment process, enabling a focus on the chatbot's logic and functionality without worrying about underlying infrastructure.

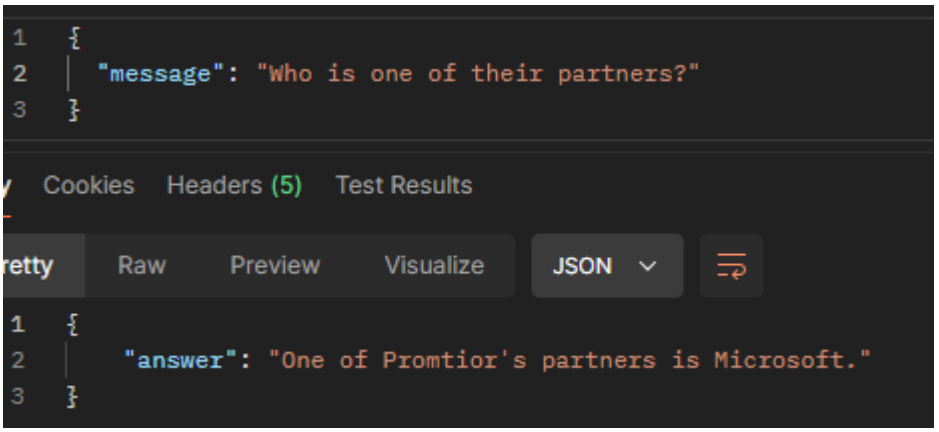
5. Testing

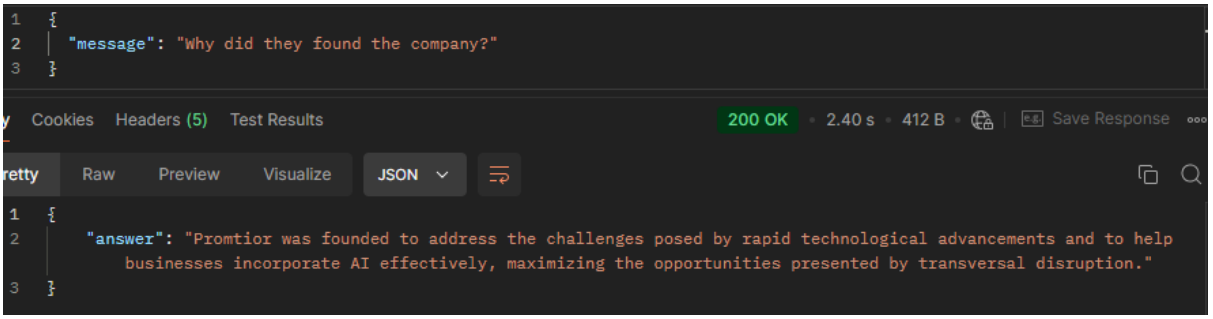
In order to test the chatbot, I used **Postman**. So once the app is running by making POST requests to the "/chat" endpoint the user can interact with the chatbot. Below there are some images with examples of different queries made by an user and its corresponding responses.

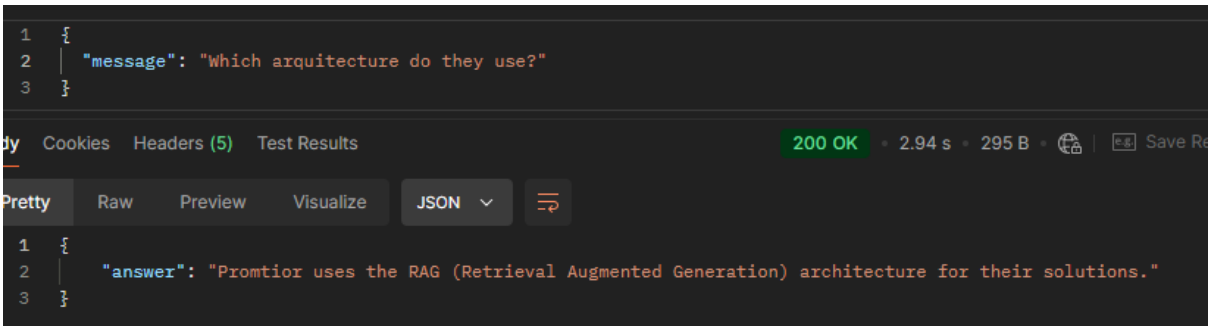





{
"message": "Where is the company from?"
}
Cookies Headers (5) Test Results
Raw Preview Visualize JSON
{
"answer": "Promtior is based in Montevideo, Uruguay."
}


1 {
2 "message": "Who is one of their partners?"
3 }
Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON
1 {
2 "answer": "One of Promtior's partners is Microsoft."
3 }


1 {
2 "message": "Why did they found the company?"
3 }
Cookies Headers (5) Test Results
200 OK 2.40 s 412 B Save Response
Pretty Raw Preview Visualize JSON
1 {
2 "answer": "Promtior was founded to address the challenges posed by rapid technological advancements and to help businesses incorporate AI effectively, maximizing the opportunities presented by transversal disruption."
3 }


1 {
2 "message": "Which arquitecture do they use?"
3 }
Cookies Headers (5) Test Results
200 OK 2.94 s 295 B
Pretty Raw Preview Visualize JSON
1 {
2 "answer": "Promtior uses the RAG (Retrieval Augmented Generation) architecture for their solutions."
3 }