SevenApps Backend Developer Take-Home Challange: PDF Chat API

Overview

This is designed to thoroughly assess your skills in Python programming, API development, integration with external services, using Large Language Models and to understand your problem-solving skills .

In this challenge, you'll be tasked with building a FastAPI application that enables users to interact with PDF files and being able to chat with it!

Project Description

Your primary objective is to develop a sophisticated FastAPI application that serves as the backbone of a PDF chat service.

This application will provide robust endpoints for uploading PDF documents and

The core functionality of your application should encompass:

1. Efficient uploading and processing of PDF documents
2. Intelligent extraction and storage of text content and associated metadata
3. Seamless integration with the Gemini API for advanced natural language processing
4. Provision of an intuitive and responsive chat functionality based on PDF content
5. Implementation of comprehensive error handling and detailed logging mechanisms
6. Ensuring top-notch security measures and addressing scalability concerns
7. Optimizing performance for handling large volumes of data and concurrent users

Detailed Tasks

1. Set up your Large Language Model (LLM)

While we highly recommend leveraging Google's Gemini API for this project due to its advanced capabilities and free tier, we understand the importance of flexibility in choosing the right tools for the job. As such, you have the liberty to select any state-of-the-art Large Language Model for inference. Here are some options to consider:

- **Google Gemini API (recommended and free)**
- Local models using Ollama or LM Studio (free)
- OpenAI's GPT models (GPT-3.5, GPT-4)
- Anthropic's Claude

For the purposes of this comprehensive guide, we'll focus on setting up and utilizing the Gemini API, given its powerful features and accessibility.

1.1 Setting up the Gemini API

Follow these detailed steps to set up your Gemini API access:

1. Open your web browser and navigate to https://ai.google.dev/aistudio
2. Sign in with your Google account. If you don't have one, you'll need to create it.
3. Once logged in, locate and click on the "Create" button in the top-right corner of the page.
4. From the dropdown menu, select "API key".
5. A new API key will be generated. Carefully copy this key and store it securely.

For in-depth implementation instructions and best practices, we strongly recommend referring to the Gemini API Python Quickstart Documentation. This resource provides valuable insights into initializing the API client, sending requests, and handling responses effectively.

1.2 API Key Management

Implementing secure key management is crucial for maintaining the integrity and security of your application. Follow these best practices:

1. Utilize environment variables to store sensitive information, including API keys.
2. Create a `.env` file in your project's root directory to securely store your Gemini API key locally.
3. Leverage a robust library like `python-dotenv` to efficiently load environment variables in your application.
4. Ensure that your `.env` file is explicitly added to your `.gitignore` file to prevent accidental exposure of sensitive information in version control.

2 Chat With PDF API

2.1 FastAPI Endpoint Implementation

Develop the following FastAPI endpoints to facilitate PDF upload and chat functionality:

2.1.1 PDF Upload

- **Request Method:** POST
- **Endpoint:** `/v1/pdf`
  - **Description:** Endpoint for uploading and registering a PDF
  - **Input:**
    - **Description:** Multipart form data containing the PDF file
    - **Example**

    ```
    curl -X POST "http://localhost:8000/v1/pdf" \
    -F "file=@/path/to/your/pdf/file.pdf"
    ```

  - **Output:**
    - **Description:** JSON response with the generated PDF ID
    - **Example**

    ```
    {
        "pdf_id": "unique_pdf_identifier"
    ```

```
}
```

- **Process:**

```
a. Validate the uploaded file (file type, size limits)
b. Generate a unique identifier for the PDF
```

2.1.1.1 State Management

Implement a robust state management system with the following features:

1. *Design a data structure to efficiently store PDF(s).*
2. *Implement error handling to manage various PDF formats and potential parsing issues.*
3. *Store extracted text along with associated metadata (e.g., filename, document ID, page count) in a structured format.*
4. *Consider implementing text preprocessing techniques to enhance the quality of extracted content.*

2.1.2 Chat with PDF

- **Request Method:** POST
- **Endpoint:** `/v1/chat/{pdf_id}`
  - **Description:** Endpoint for interacting with a specific PDF
  - **Input:** JSON body containing the user's message

```
{
    "message": "What is the main topic of this PDF?"
}
```

  - **Output:** JSON response with the AI-generated answer

```
{
    "response": "The main topic of this PDF is..."
}
```

  - **Process:**

```
a. Validate the pdf_id and retrieve the associated PDF content
b. Use the Gemini API to generate a response based on the PDF
content and user query
```

2.3 LLM Integration

Implement seamless integration with the Gemini API for advanced natural language processing:

1. *Initialize the Gemini API client using the securely stored API key.*
2. *Develop a function to generate context-aware prompts based on PDF content and user queries.*
3. *Implement error handling for API rate limits, timeouts, and other potential issues.*
4. *Consider implementing caching mechanisms to improve response times for frequently asked questions.* (Optional)

For detailed implementation guidance, refer to the Gemini API documentation and best practices for prompt engineering and API integration.

Problems to tackle (Bonus points!)

- **Problem 1:** Is using the Gemini 1.5 Flash that has 1 Million context size enough or Retrieval-Augmented Generation (RAG) is a better approach?
- **Problem 2:** Having 1 Million context size is great but output tokens are limited to 8196, how would you queries that has more than 8196 tokens?
- **Problem 3:** Writing unit tests are great for ensuring the app works just fine, but how would you evaluate the performance of the Large Language Model?

2.5 Error Handling and Logging

Implement comprehensive error handling and logging mechanisms:

1. Develop a custom error handling middleware for FastAPI to catch and process exceptions.
2. Implement detailed logging for all critical operations, including PDF processing, API calls, and state management.
3. Use a structured logging format (e.g., JSON) for easy parsing and analysis. (*Optional*)
4. Implement different log levels (**DEBUG**, **INFO**, **WARNING**, **ERROR**) for granular control over log output. (*Optional*)

Ensure that all API responses include meaningful error messages and appropriate HTTP status codes for various error scenarios.

2.6 Testing

Make sure you add some tests to ensure

1. Unit tests for individual components (e.g., PDF processing, Gemini API integration, state management).
2. Integration tests to verify the entire application flow, including API endpoints and error handling.

2.7 Documentation

Provide comprehensive documentation for the project:

1. Develop a comprehensive README.md file including:
    - Project overview
    - Detailed setup instructions, including environment configuration
    - Explanation of API endpoints with request/response examples
    - Testing procedures and instructions for running the test suite

Ensure all documentation is clear, concise, and kept up-to-date with any changes to the project.

Evaluation Criteria

Criteria	Description	Weight
Functionality	Correct implementation of all required features	40%
Code Quality	Clean, readable code following Python best practices	25%
API Design	Well-structured endpoints and appropriate use of HTTP methods	20%
Documentation	Clear and complete documentation, including API docs and setup instructions	10%
Error Handling	Robust error handling and informative error messages	5%

Tip (Bonus Points!)

You can use any library that will help you with the task, including but not limited to:

- **Langchain**
  - See: https://python.langchain.com/
- **LLama-Index**
  - See: https://docs.llamaindex.ai/

Additional Considerations

Deadline: Please submit your completed project within 7 days of receiving this challenge.

Note: While we appreciate timely submissions, we value quality over speed. Take the time you need to showcase your best work!

Good luck! We look forward to reviewing your submission.

Need any help?

Feel free to reach out to directly from elif@sevenapps.co if you have any questions or need any help.