Project Task: Todo App with FastAPI and PostgreSQL (NO UI)

Project Overview

What the project is about:
Build a backend diary/todo application using FastAPI, which allows users to create, read, update, and delete todo items. The app will integrate with a PostgreSQL database, enabling you to learn about RESTful API design, database integration, and backend development practices.

What You Have to Do

- **Environment Setup:**
  - Install Python (3.9+), FastAPI, Uvicorn, SQLAlchemy, and a PostgreSQL driver (like `psycopg2`).
  - Create a virtual environment and set up a Git repository.
- **Project Structure:**
  - Organize your project into logical directories (e.g., `app/`, `models/`, `schemas/`, and `routes/`).
- **Database Integration:**
  - Set up a PostgreSQL database.
  - Use SQLAlchemy to define models for diary/todo items.
- **API Development:**
  - Create RESTful API endpoints in FastAPI for diary/todo CRUD operations.
  - Validate input and output using Pydantic models.
  - Create swagger api.
- **Testing & Documentation:**
  - Write tests for your endpoints (using FastAPI's TestClient or pytest).
  - Document the API endpoints and project structure.

What You Will Learn

- **Backend Frameworks:**
  - How to build a backend service with FastAPI and understand its routing, dependency injection, and asynchronous capabilities.
- **Database Integration:**
  - Configuring and interacting with PostgreSQL using SQLAlchemy.
  - Implementing CRUD operations and handling database transactions.
- **API Design:**
  - Creating clean, well-documented RESTful APIs.
  - Using Pydantic for data validation and serialization.
- **Project Organization & Testing:**

- Structuring a backend project for maintainability and scalability.
- Writing tests to ensure reliability and correctness.

Project Task: Chat with PDF using OpenAI LLM API and Gradio

Project Overview

What the project is about:
Develop a backend-powered chat application where users can upload a text PDF file, extract its content, and interact with it by asking questions. The app leverages OpenAI's LLM API to generate responses based on the uploaded document. The user interface is built with Gradio for simplicity and rapid development.

What You Have to Do

- **Environment Setup:**
  - Install Python (3.9+), Gradio, OpenAI SDK, and a PDF extraction library like PyPDF2.
  - Configure your project's virtual environment and manage dependencies.
- **Project Structure:**
  - Organize the project files logically (e.g., `app.py` for main code, `utils/` for helper functions).
- **PDF Upload & Text Extraction:**
  - Implement a feature to allow users to upload a text-based PDF file.
  - Use PyPDF2 to extract text from the uploaded PDF.
- **LLM Integration:**
  - Integrate the OpenAI LLM API to process user questions.
  - Construct a prompt that includes (a summary of) the extracted PDF text and the user's query.
- **Chat Interface Development:**
  - Build an interactive UI with Gradio that supports file upload, question input, and displays the conversation history.
- **Testing & Deployment:**
  - Test the application thoroughly.
  - Optionally deploy the app for broader access.

Notes:

No data storage needed, Chat can restart on a new session.

What You Will Learn

- **LLM & API Integration:**
    - How to integrate and interact with the OpenAI API for natural language processing tasks.
- **PDF Text Extraction:**
    - Techniques to extract and preprocess text data from PDFs using PyPDF2.
- **Interactive UI Development:**
    - Building a simple, interactive web interface using Gradio.
- **Prompt Engineering:**
    - Crafting effective prompts that combine document context and user queries for accurate LLM responses.