Test Task: Full-Stack & AI-Enabled Web Application Challenge
Estimated Time: 1–3 hours
Objective:
Develop a simple Q&A web app that demonstrates your skills in full-stack development, AI integration, and rapid prototyping. The task covers:

- Web Development: Building a modern web application (front-end and back-end) with RESTful API endpoints.
- AI Integration: Calling a generative AI model to produce responses based on retrieved knowledge.
- Data Handling: Implementing a basic knowledge base lookup (via CSV/SQL/in-memory data).
- Testing: Demonstrating basic automated testing strategies.
- Documentation & Communication: Clear setup, code structure, and thoughtful comments.

Task Description
1. Build the Application
Front-End

- User Interface:
  Create a minimal user interface (using a framework of your choice such as React, Vue, Angular, or server-rendered templates). The UI should include:
  - An input field for the user to type a question.
  - A display area to show the AI-generated answer.
  - *(Optional)* A log of past questions and answers.

Back-End

- Language & Framework:
  Use Python (preferably with FastAPI or Flask). Other languages/frameworks are acceptable, but Python is preferred to simplify AI/ML integration.
- Endpoints:
  1. POST /api/ask:
     - Accepts a JSON payload with a user question.
     - Performs a retrieval step: Query a small knowledge base (e.g., a CSV with 5–10 FAQ entries) using a simple keyword search or embedding-based lookup.
     - Generates a final answer: Feed the relevant context(s) together with the user question to a generative AI model (e.g., via the OpenAI API, a local model integrated through LangChain, etc.). The prompt should direct the model to incorporate and reference the retrieved context.
     - Returns the generated answer in a JSON response.
  2. *(Optional)* GET /api/history:
     - If you implement a history/log feature, expose an endpoint to fetch previous Q&A pairs.
- Data Layer:
  You can use a simple in-memory list, a CSV file stored in memory, or a minimal SQL database (with a table such as faq_entries having columns like id and content).

AI / RAG Integration

- Retrieval-Augmented Generation:
  Include a step where you extract relevant context from your small knowledge base before generating the answer with the AI model.
- Generative AI Model:
  Integrate with a model through services such as OpenAI API, or use a local model (with frameworks like LangChain) to generate a coherent answer based on both the user's question and the retrieved context.

2. Testing & Quality Assurance

- Automated Testing:
  Write at least one meaningful automated test:
  - Unit Test: Test the retrieval function with a sample query.
  - Integration Test: Test the /api/ask endpoint to ensure it returns a response with the correct JSON structure.
- Code Quality:
  Ensure your code is modular, well-commented, and follows best practices.

3. Project Setup, Documentation, and Git Upload

- Project Setup:
  - Provide a requirements.txt, Pipfile, or poetry.lock file listing all dependencies.
  - Include instructions in a README.md on how to set up and run the application, as well as how to run the tests.
  - *Optional:* Provide Docker configuration (such as a docker-compose.yml or Dockerfile).
- Git Upload:
  Important: Create a Git repository for your solution (on GitHub or GitLab), commit your code, and push it. Include the repository URL in your submission so we can review your work.
- README.md Content:
  Your README should include:
  - Setup Instructions: How to install dependencies and run the application.
  - Execution Instructions: How to start the server and the front-end.
  - Testing: How to run the tests.
  - Approach Explanation: A brief overview of your design choices, especially regarding the retrieval process and the AI integration.
  - Known Limitations/Trade-Offs: Any points you might improve given more time.

4. Bonus (Optional)

If you have extra time or want to demonstrate additional skills:

- Multi-Agent Systems:
  Add a second processing step (e.g., a “summarizer agent” that refines the generated answer or a “moderator agent” that verifies information).
- Human-in-the-Loop:
  Implement a feature allowing the user to confirm or edit the retrieved context before final generation.
- Additional Technologies:
  Incorporate protocols or advanced tools (e.g., knowledge graphs, vLLM, ollama) if you are familiar with them.

Submission Instructions

1. Develop your solution locally following the task guidelines.
2. Upload your complete solution to a Git repository.
3. Share the repository URL in your submission (e.g., include it in your email or application portal).

Please focus on demonstrating functionality, clarity, and best practices rather than a fully production-ready solution. We’re eager to see your innovative approach and well-documented code.
Good luck!