Agent Framework for RAG-Based Question Generation and Summarization

This technical assessment evaluates the AI Engineering ability to design, build, and deploy a robust **multi-agent system** for generating **MCQ and fill-in-the-blank questions** and **summaries** from PDF files, leveraging **Large Language Models (LLMs)** and **Retrieval Augmented Generation (RAG)**. Key skills assessed include multi-agent orchestration, RAG pipeline implementation, LLM integration, robust FastAPI endpoint development, Docker containerization for deployment and adherence to strong software engineering practices.

Please Do:

- **Include Instructions**: Provide instructions on how to run the code.
- **Include Sample Questions:** Provide a list of generated questions
- **Keep it Simple**: Ensure the solution is simple and concise.
- **Submit Within Time**: Return the zipped solution within **72 hours** by email to *technicaltest@alefeducation.com*; or provide a link to a Google Drive folder with the zipped solution.
- **Ask Questions**: If you have any questions, email *technicaltest@alefeducation.com*.

Please Don't:

- Do not build a user interface; the focus is on the technical solution and not the interface.
- Perform Unnecessary Tasks: Do not perform exploratory data analysis, model evaluation.
- Please make sure that you complete the task by yourself, the purpose of the task is to assess your capabilities.

Functional Test Conditions:

- Upload a sample PDF file "**A quick Algebra Review**" and ingest into a Vector DB with appropriate metadata.
- Generate Multiple Choice Questions based on the topics in the PDF file

Technical Requirements:

1. Design a multi-agent system that integrates with LLMs to generate questions from an uploaded file. You are required to have **two agents** for the question generation endpoint. One agent will take in the retrieved content and create questions, and a second agent will be the evaluator for the generated questions.
2. Use libraries like Langchain, LangGraph, Huggingface, FastAPI to facilitate the development.
3. Use openly available LLMs available via APIs from providers like Github Models, OpenRouter, OpenAI, Google AI Studio, or Huggingface.
4. Use an open-source vector database to store and retrieve relevant information efficiently.
5. Create a Dockerfile with commands to run the application and expose the endpoints. If the container fails to build, start, or expose the endpoints, the submission will be disqualified.

Endpoints:

1. **/ingest**: Upload and process the PDF file. It should return the Table of Contents for the uploaded document.
2. **/generate/questions**: Generate MCQ questions based on the PDF. Given a concept or a text query, the endpoint should generate the questions according to the retrieved documents. The input is text where the agent will orchestrate the flow from retrieval to question creation to the evaluation.

Assessment Criteria:

- Should cover all Technical requirements.
- Should cover the functional test case.
- Demonstrate clean coding and test driven development.
- You are expected to be able to modify and implement extra requirements during the live code interview.
- Should be organized to demonstrate as a python production code file rather than a .ipynb file used for experimentation.