Coding Exercise: Document Management and RAG-based Q&A Application

Candidates are required to build an application that involves backend services and Q&A features powered by a Retrieval-Augmented Generation (RAG) system. The application aims to manage users, documents, and an ingestion process that generates embeddings for document retrieval in a Q&A setting.

Application Components

1. **Python Backend (Document Ingestion and RAG-driven Q&A)**
    - **Purpose:** Develop a backend application in Python to handle document ingestion, embedding generation, and retrieval-based Q&A (RAG).
    - **Key APIs:**
        - **Document Ingestion API**: Accepts document data, generates embeddings using a Large Language Model (LLM) library, and stores them for future retrieval.
        - **Q&A API**: Accepts user questions, retrieves relevant document embeddings, and generates answers based on the retrieved content using RAG.
        - **Document Selection API**: Enables users to specify which documents to consider in the RAG-based Q&A process.
    - **Tools/Libraries**: Any one of the following
        - Use Ollama Llama 3.1 8B model/Langchain/Llama Index library or OpenAI API or Hugging Face Transformers
        - Database for storing embeddings (Postgres preferred).
        - Asynchronous programming for efficient handling of API requests.

Evaluation Criteria

Backend (Python - Document Ingestion and Q&A)

1. **Code Quality**:
    - Asynchronous programming practices for API performance.
    - Clear and concise code, with emphasis on readability and maintainability.
2. **Data Processing and Storage**:
    - Efficient embedding generation and storage.
    - Ability to handle large datasets (e.g., large volumes of documents and embeddings).
3. **Q&A API Performance**:
    - Effective retrieval and generation of answers using RAG.
    - Latency considerations for prompt response times.
4. **Inter-Service Communication**:
    - Design APIs that allow the backend to trigger ingestion and access Q&A functionality seamlessly.
5. **Problem Solving and Scalability**:
    - Demonstrate strategies for large-scale document ingestion, storage, and efficient retrieval.
    - Solution for scaling the RAG-based Q&A system to handle high query volumes.

End-of-Development Showcase Requirements

At the end of the development, candidates should demonstrate the following:

1. **Design Clarity**:
    - Show a clear design of classes, APIs, and databases, explaining the rationale behind each design decision.
    - Discuss non-functional aspects, such as API performance, database integrity, and consistency.
2. **Test Automation**:
    - Showcase functional and performance testing.
    - Cover positive and negative workflows with good test coverage (70% or higher).
3. **Documentation**:
    - Provide well-documented code and create comprehensive design documentation.
4. **3rd Party Code Understanding**:
    - Explain the internals of any 3rd-party code used (e.g., libraries for LLM or authentication).
5. **Technical Knowledge**:
    - Demonstrate knowledge of HTTP/HTTPS, security, authentication, authorization, debugging, monitoring, and logging.
6. **Advanced Concepts**:
    - Usage of design patterns in code.
7. **Test Data Generation**:
    - Demonstrate skills in generating large amounts of test data to simulate real-world scenarios.
8. **Deployment and CI/CD** (Applicable to All Components):
    - **Dockerization**: Dockerize each service, making it easily deployable and portable.
    - **Deployment Scripts**: Provide deployment scripts to run the application on Docker or Kubernetes, compatible with any cloud provider (e.g., AWS, Azure, GCP).
    - **CI/CD Pipeline**: Implement a CI/CD pipeline for each component to automate testing, building, and deployment.
