# Coding Exercise: Document Management and RAG-based QsA Application

Candidates are required to build an application that involves backend services and QCA 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 QCA setting.

# Application Components

1. **Python Backend (Document Ingestion and RAG-driven QsA)**
   * **Purpose:** Develop a backend application in Python to handle document ingestion, embedding generation, and retrieval-based QCA (RAG).

# Key APIs:

* + - **Document Ingestion API**: Accepts document data, generates embeddings using a Large Language Model (LLM) library, and stores them for future retrieval.
    - **QsA 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 QCA 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 QsA)**

1. **Code Quality**:
   * Asynchronous programming practices for API performance.
   * Clear and concise code, with emphasis on readability and maintainability.

# Data Processing and Storage:

* + Efficient embedding generation and storage.
  + Ability to handle large datasets (e.g., large volumes of documents and embeddings).

# QsA API Performance:

* + Effective retrieval and generation of answers using RAG.
  + Latency considerations for prompt response times.

# Inter-Service Communication:

* + Design APIs that allow the backend to trigger ingestion and access QCA functionality seamlessly.

# Problem Solving and Scalability:

* + Demonstrate strategies for large-scale document ingestion, storage, and efficient retrieval.
  + Solution for scaling the RAG-based QCA system to handle high query volumes.

# End-of-Development Showcase Requirements

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

# 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.

# Test Automation:

* + Showcase functional and performance testing.
  + Cover positive and negative workflows with good test coverage (70% or higher).

# Documentation:

* + Provide well-documented code and create comprehensive design documentation.

# 3rd Party Code Understanding:

* + Explain the internals of any 3rd-party code used (e.g., libraries for LLM or authentication).

# Technical Knowledge:

* + Demonstrate knowledge of HTTP/HTTPS, security, authentication, authorization, debugging, monitoring, and logging.

# Advanced Concepts:

* + Usage of design patterns in code.

# Test Data Generation:

* + Demonstrate skills in generating large amounts of test data to simulate real- world scenarios.

1. **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.