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

Candidates are required to build a three-part application that involves backend services, frontend interface, 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. The exercise is divided into three main components: **Python-based backend for document ingestion**, **NestJS backend for user and document management**, and **Angular frontend for user interaction**.

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**:
        - Use LLM libraries (e.g., OpenAI API or Hugging Face Transformers).
        - Database for storing embeddings (Postgres preferred).
        - Asynchronous programming for efficient handling of API requests.

2. **NestJS Backend (User Management and Document Management)**
    - **Purpose:** Create a backend service using NestJS to manage user authentication, document management, and ingestion controls.
    - **Key APIs:**
        - **Authentication APIs**: Register, login, logout, and handle user roles (admin, editor, viewer).
        - **User Management APIs**: Admin-only functionality for managing user roles and permissions.
        - **Document Management APIs**: CRUD operations for documents, including the ability to upload documents.
        - **Ingestion Trigger API**: Allows triggering the ingestion process in the Python backend, possibly via a webhook or API call.
        - **Ingestion Management API**: Tracks and manages ongoing ingestion processes.
    - **Tools/Libraries**:
        - TypeScript for consistent type management.
        - Database integration (Postgres recommended).
        - JWT for authentication, with role-based authorization.
        - Microservices architecture to facilitate interaction between NestJS and the Python backend.

3. **Angular Frontend (User Interface for Management and Q&A)**

- **Purpose:** Develop an Angular-based frontend to handle user interactions with the backend services, document management, ingestion management, and RAG-based Q&A interface.
- **Key Pages/Features**:
  - **Sign Up, Login, and Logout**: User authentication interface.
  - **User Management**: Admin-only access for managing users and assigning roles.
  - **Document Upload and Management**: Interface to upload and manage documents.
  - **Ingestion Management**: Interface to trigger and monitor ingestion status.
  - **Q&A Interface**: A user-friendly interface for asking questions, receiving answers, and displaying relevant document excerpts (using RAG).
- **UI Considerations**:
  - Responsive design for multiple devices and browsers.
  - Modular, reusable components for better code structure.
  - Consistency with design patterns to ensure maintainability and scalability.

Evaluation Criteria

Frontend (Angular)

1. **Code Quality**:
   - TypeScript expertise, modular UI component development, and adherence to design patterns.
   - Readable, well-documented, and simple code structure.
2. **Web Services Integration**:
   - Ability to consume APIs effectively and handle asynchronous operations.
3. **CSS and Design**:
   - Proficiency in CSS for a visually appealing, responsive UI.
   - Demonstration of user-centered design thinking, including consistent UX and accessibility.
4. **Performance and Testing**:
   - Automated testing of the UI.
   - Web app optimized for high performance (Google Page Speed Insights score of 90% or above).
   - Considerations for handling large-scale usage (e.g., handling 1 million users).
5. **Additional Skills**:
   - Usage of website analytics to track and improve user experience.
   - Problem-solving approach and demonstrated thought for large-scale application viability.

Backend (NestJS)

1. **Code Quality and Structure**:
   - TypeScript usage with strong object-oriented principles.
   - Clean, well-documented, and easy-to-understand code structure.
2. **Data Modeling and Design**:
   - Design a robust database schema, including generating a large dataset (e.g., 1000+ users with roles, 100000+ entities).
   - Demonstrate methods to create realistic test data.
3. **API Development and Testing**:

    - REST API design and automated testing.
    - Microservices architecture integration to handle the Python backend for ingestion.
4. **Authentication and Authorization**:
    - Implementation of JWT-based authentication with role-based access control.
    - Demonstration of secure and scalable authentication for high volumes of users.
5. **Additional Skills**:
    - Knowledge of microservices and inter-service communication.
    - Problem-solving skills and scalability considerations for handling large datasets and user traffic.

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 NestJS 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**:
    - Showcase advanced concepts like RxJS, NgRx, and ORM where applicable.
    - 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.

Special Requirement for Frontend-Only Developers

If the candidate is a frontend-only developer, they should create a **mocking service** that simulates backend services. This service should provide random responses that enable the frontend application to work as expected and ensure comprehensive testing of frontend functionality.

No cloud provider account or LLM will be provided for this exercise. Use any free / low end LLM (Accuracy of response in Q&A is NOT a criteria for evaluation).