**MANDATORY INSTRUCTIONS**

* **Code should be modular and structured (not all files should be in one folder, should reusable and modules can be tested independently)**
* **Modules and Packages should be distinct to maintain scalability, reusability and testability**
* **Authentication should be implemented**
* **Unit test cases for all modules/components are mandatory**
* **Detailed documentation step by step procedure on how to run your application is also mandatory**

**Coding Exercise: Document Management and Basic Q&A Application**

Candidates are required to build a three-part application that involves backend services, frontend interface, and Basic Q&A features. 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: **Spring Boot based backend for document ingestion**, **.NET Core backend for user and document management**, and **Next.js frontend for user interaction**.

**Application Components**

1. **Spring Boot Backend (Document Ingestion and Basic Q&A)**
   * **Purpose :** Develop a backend application using **Spring Boot (Java)** to handle document ingestion, basic storage, and Basic retrieval Q&A.
   * **Key APIs:** 
     + **Authentication:**

**1.** Register, login, logout, and handle user roles (admin, editor, viewer).

* + - **Document Ingestion API:**
      1. Accepts document data (text, PDF, Docx, etc.).
      2. Stores content and metadata into relational database (Postgres/MySQL).
    - **Q&A API:**
      1. Accept user questions (keyword-based).
      2. Retrieves relevant documents using basic keyword matching or SQL queries (e.g. ILIKE or CONTAINS) as full-text search.
      3. Returns matching document snippets or summaries.
    - **Document Selection API:** 
      1. Allows users to filter documents by metadata (e.g. author, date, type).
      2. Supports pagination and sorting.
  + **Tools/Libraries:**
    - Sprint Boot (Java) for backend framework.
    - Spring Data JPA for database operations.
    - Postgres/MySQL for document storage. (Postgres preferred).
    - Swagger for API Documentation
    - Asynchronous programming for efficient handling of API requests.

1. **.NET Core Backend (User Management and Document Management)**
   * **Purpose:** Create a backend service using .NET Core 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 Spring Boot backend, possibly via a webhook or API call.
     + **Ingestion Management API**: Tracks and manages ongoing ingestion processes. Provide status updates or cancelation options.
   * **Tools/Libraries**:
     + ASP.NET Core for backend framework. (.Net 8/9 preferred).
     + C# for consistent, type-safe coding.
     + Entity Framework Core for Postgres integration.
     + JWT for authentication, with role-based authorization.
     + Http Client (or gRPC) to call Sprint Boot backend.
     + Microservices architecture to facilitate interaction between .NET Core and the Spring Boot backend.
2. **Next.js Frontend (User Interface for Management and Q&A)**
   * **Purpose:** Develop an Next.js frontend to handle user interactions with the backend services, document management, ingestion management, and a user-friendly 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.
   * **UI Considerations**:
     + Responsive design for multiple devices and browsers.
       1. Tailwind CSS or CSS module for styling across devices.
     + Modular, reusable components for better code structure.
     + State Management :
       1. React’s built-in Context API for lightweight global state.
     + API Integration:
       1. Next.js API Routes for server-side actions (e.g., auth).
     + Consistency with design patterns to ensure maintainability and scalability.

**Evaluation Criteria**

**Frontend**

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**:
   * Handle asynchronous operations using React’s useEffect and hooks.
   * Manage State.
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 (.NET Core)**

1. **Code Quality and Structure**:
   * C# usage with strong object-oriented principles.
   * Clean, well-documented, and easy-to-understand code structure.
2. **Data Modelling 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 Spring Boot 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 (Java - 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**:
   * Implement efficient batch Processing for large document uploads. (e.g., using Spring Batch or Message Queues)
   * Store documents and associate metadata (e.g., keywords, documents type) in a RDBMS (Postgres preferred).
3. **Q&A API Performance**:
   * Implement **full-text search** capabilities using **PostgreSQL’s tsvector** or **Elasticsearch** for high-performance document searching.
   * Use caching strategies (e.g., **Redis**) to store and quickly retrieve frequently queried data or document summaries.
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. **Inter-Service Communication**:
   * Design APIs that allow the .NET Core backend to trigger ingestion and access Q&A functionality seamlessly.
6. **Problem Solving and Scalability**:
   * Implement **retry mechanisms**, **backoff strategies**, and **batch processing** to handle large-scale document ingestion efficiently.
   * Use **message queues (e.g., RabbitMQ, Kafka)** for decoupling ingestion tasks and distributing the load across multiple services or worker nodes.

**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., authentication).
5. **Technical Knowledge**:
   * Demonstrate knowledge of HTTP/HTTPS, security, authentication, authorization, debugging, monitoring, and logging.
6. **Advanced Concepts**:
   * Showcase advanced concepts like React Context API, Redux 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.