Full-Stack Developer Assignment

Overview

Create a full-stack web application for a task management system using React for the frontend and Node.js with Express for the backend.

The application should allow users to:

− Register and log in with authentication (JWT).

− Create, read, update, and delete (CRUD) users.

− Create, read, update, and delete (CRUD) tasks.

− Assign tasks to different users.

− Attach up to 3 documents (PDF format) to tasks and view those documents when viewing task details.

− Filter and sort tasks based on status, priority, and due date.

− Store data in PostgreSQL or MongoDB.

− Store files in either local machine or any cloud storage such as S3

− Ensure proper API documentation and include automated tests.

− Containerize the application using Docker.

Requirements

AUTHENTICATION & AUTHORIZATION:

• Implement user registration and login with password hashing.

• Use JWT-based authentication.

• Ensure users can only manage their own tasks unless they have an admin role.

• Secure frontend routes based on authentication and authorization roles.

FRONTEND REQUIREMENTS:

• Use React for the frontend.

• Implement Redux, Context API, or any state management library for state management.

• Use React Router for navigation.

• Build a responsive UI with CSS.

• Implement client-side form validation.

• Consume backend APIs efficiently and handle loading/error states.

• Display real-time task updates using WebSockets.

• Allow users to upload documents when creating or updating a task.

• Provide a way to view attached documents when viewing task details.

BACKEND REQUIREMENTS:

• Implement RESTful APIs following best practices.

• CRUD operations for Users and Tasks.

• Admin users should be able to create, edit, and delete any user.

• Users should have attributes: email, password, role.

• Tasks should have attributes: title, description, status, priority, due date, assigned to, and attached documents.

• Implement filtering, sorting, and pagination when listing users and tasks.

• Implement file upload functionality for attaching documents to tasks.

• Provide API endpoints to retrieve and download attached documents.

DATABASE:

• Use MongoDB (Preferably Docker containers that can be run locally).

• Define appropriate schema/models with relationships.

• Store file metadata in the database while keeping actual files in a storage solution (e.g., cloudinary).

GIT & VERSION CONTROL:

• Use Git for version control with meaningful commit messages.

• Provide a gitignore file to exclude unnecessary files.

• Use feature branches and make pull requests for code reviews (optional).

DOCKER & DEPLOYMENT

• Containerize the application using Docker.

• Provide a Docker Compose file for easy setup.

• Ensure the application can run with a single command (docker-compose up).

• Bonus: Deploy the application to Vercel.

TESTING

• Write unit and integration tests for authentication and task management.

• Use Jest, Mocha, PyTest, or Cypress depending on the chosen tech stack.

• Ensure at least 80% test coverage.

DOCUMENTATION

• Provide a well-structured README.md with setup instructions.

• Document API endpoints using Swagger or Postman collection.

• Include a brief explanation of the project structure and design decisions.

DELIVERABLES

• A public GitHub repository with the project.

• README.md with setup instructions.

• Automated test cases.

• API documentation (Postman collection).

• Docker configuration files (Dockerfile & docker-compose.yml).

EVALUATION CRITERIA

• Code quality and structure.

• Proper use of authentication and authorization.

• API design and adherence to RESTful principles.

• Database schema design and queries.

• Implementation of filtering, sorting, and pagination.

• Git commit history and best practices.

• Quality and coverage of test cases.

• Clarity and completeness of documentation.

• Responsiveness and usability of the frontend.

- Proper handling of file uploads and retrieval.