QUALCO

Assessment for Senior-Level Full-Stack Developer

Contents

Project Title: Task Management Application

1. Overview

The goal is to build an advanced task management system with real-time collaboration features. Use **React** for the front-end and **Java (Spring Boot)** for the back-end. The application should allow authenticated users to perform CRUD operations on tasks, collaborate in real time, and manage projects.

2.Requirements

2.1 Front-End: React

1. **Components**:
   - **TaskList**: Displays tasks within a project.
   - **TaskForm**: Form to add or edit tasks.
   - **ProjectDashboard**: Overview of all projects and their associated tasks.
   - **UserManagement**: Manage users and permissions.
2. **State Management**:
   - Use **Redux** or **Context API** for managing state across the application.
3. **Real-Time Collaboration**:
   - Implement real-time updates using **WebSockets** (e.g., **Socket.IO** or similar library).
4. **Routing**:
   - Use **React Router** for navigation between different views (projects, tasks, user management).
5. **Styling**:
   - Use libraries like **Material-UI** or **Tailwind CSS** for a responsive and modern UI.
6. **Testing**:
   - Write unit tests with **Jest** and **React Testing Library**.

2.2 Back-End: Java (Spring Boot)

1. **Endpoints**:

- GET /projects: Fetch all projects.
- POST /projects: Create a new project.
- GET /projects/{id}/tasks: Fetch tasks within a project.
- POST /projects/{id}/tasks: Create a new task in a project.
- PUT /tasks/{id}: Update a task.
- DELETE /tasks/{id}: Delete a task.
- POST /register: Register a new user.
- POST /login: User authentication.

2. **Real-Time Communication**:
   - Implement **WebSockets** for real-time task updates and notifications.
3. **Database**:
   - Use **PostgreSQL** or **MySQL** for data persistence.
4. **Model**:
   - Project class with id, name, description, and createdDate.
   - Task class with id, title, description, status, and projectId.
   - User class with id, username, password, and role.
5. **Security**:
   - Implement JWT-based authentication and role-based authorization.
6. **Testing**:
   - Write unit tests with **JUnit** and **Mockito**.

3. Project Structure Guidelines

```
senior-level-task-app/
| ├── frontend/
|   | ├── src/
|   |   | ├── components/
|   |   |   | ├── TaskList.js
|   |   |   | ├── TaskForm.js
|   |   |   | ├── ProjectDashboard.js
|   |   |   | └── UserManagement.js
|   |   | ├── redux/
|   |   |   | ├── store.js
|   |   |   | └── tasksSlice.js
|   |   | ├── App.js
```

```
|   |   └── index.js
|
└── backend/
    ├── src/
    │   ├── main/
    │   │   ├── java/
    │   │   └── com/example/taskmanager/
    │   │       ├── model/
    │   │       │   ├── Project.java
    │   │       │   ├── Task.java
    │   │       │   └── User.java
    │   │       ├── controller/
    │   │       │   ├── ProjectController.java
    │   │       │   ├── TaskController.java
    │   │       │   └── AuthController.java
    │   │       ├── service/
    │   │       │   ├── ProjectService.java
    │   │       │   ├── TaskService.java
    │   │       │   └── UserService.java
    │   │       ├── repository/
    │   │       │   ├── ProjectRepository.java
    │   │       │   ├── TaskRepository.java
    │   │       │   └── UserRepository.java
    │   │       └── security/
    │   │           ├── JwtUtil.java
    │   │           └── SecurityConfig.java
```

4. API Specification

Method	Endpoint	Description
GET	/projects	Fetch all projects
POST	/projects	Create a new project
GET	/projects/{id}/tasks	Fetch tasks within a project
POST	/projects/{id}/tasks	Create a new task in a project
PUT	/tasks/{id}	Update a task
DELETE	/tasks/{id}	Delete a task
POST	/register	Register a new user
POST	/login	User authentication

5. Submission Guidelines

1. **GitHub Repository**:
   - Submit the project via a GitHub repository.
2. **Documentation**:
   - Include a README.md with:
     - Project overview
     - Setup instructions for both front-end and back-end
     - Real-time collaboration explanation
3. **Code Quality**:
   - Ensure clean, modular, and well-documented code.
4. **Testing**:
   - Include unit tests for both front-end and back-end components.

6. Evaluation Criteria

1. **Functionality**:
    - Are CRUD operations implemented correctly?
    - Does real-time collaboration work as expected?
2. **Authentication & Authorization**:
    - JWT-based authentication and role-based access control.
3. **Real-Time Features**:
    - Real-time updates using WebSockets.
4. **Code Quality**:
    - Clean, modular, and documented code.
5. **UI/UX Design**:
    - Modern and responsive design with intuitive navigation.
6. **Testing**:
    - Adequate test coverage for critical components.