BRAIN STATION 23 PLC.

Java Assignment

Please use spring framework and spring boot

1. User Registration and Authentication:

Task:

Allow users to register with a username, email, and password. Implement user authentication using JWT (JSON Web Tokens).

Details:

- **User Registration:**
  - Create a REST endpoint (`/api/register`) that accepts a POST request with JSON payload containing `username`, `email`, and `password`.
  - Validate input data (e.g., check for unique username, valid email format, strong password).
  - Hash the password securely before storing it in the database.
  - Upon successful registration, respond with a JSON object confirming the registration.
- **User Authentication:**
  - Create a REST endpoint (`/api/login`) for user login.
  - Accept a POST request with `username` and `password`.
  - Validate credentials against the stored hashed password.
  - If valid, generate a JWT containing user information.
  - Respond with the JWT and user details.

2. Task Management:

Task:

Authenticated users should be able to create, update, and delete tasks. Tasks should have a title, description, and status.

Details:

- **Task Model:**
  - Design a "tasks" table in a relational database to store task information.
- **Task Creation:**
  - Create a REST endpoint (`/api/tasks`) to handle task creation.
  - Accept a POST request with JSON payload containing `title`, `description`, and `status`.
  - Validate input data and ensure the user is authenticated using the JWT.
  - Store the task in the database and respond with the created task details.
- **Task Retrieval:**
  - Create a REST endpoint (`/api/tasks`) to retrieve all tasks for the authenticated user.
  - Use the JWT to authenticate the user.
  - Respond with a JSON array containing task details.
- **Task Update:**
  - Create a REST endpoint (`/api/tasks/:id`) to update a task by ID.
  - Accept a PUT request with JSON payload containing fields to be updated (e.g., `status`).
  - Validate input data and ensure the user is authenticated.
  - Update the task in the database and respond with the updated task details.
- **Task Deletion:**
  - Create a REST endpoint (`/api/tasks/:id`) to delete a task by ID.
  - Accept a DELETE request and ensure the user is authenticated.
  - Delete the task from the database and respond with a confirmation message.

3. REST API:

Task:

Implement a RESTful API for task management with endpoints for user registration/login and CRUD operations for tasks.

Details:

- **Endpoint Details:**
  - Design RESTful endpoints for user registration, user login, task creation, task retrieval, task update, and task deletion.
  - Follow REST principles (use appropriate HTTP methods, handle errors with status codes).

4. Data Validation:

Task:

Implement server-side validation for user registration and task creation. Validate input data to meet specified criteria.

Details:

- **Validation Rules:**
  - Define validation rules for fields such as password complexity, email format, and required fields.
  - Ensure that usernames and emails are unique.
  - Validate input data before processing requests.
- **Error Handling:**
  - Implement error handling to provide meaningful error messages in case of validation failures.

5. Database Operations:

Task:

Use a relational database to store user and task data. Implement database operations for CRUD functionality.

Details:

- **Database Integration:**
  - Connect your application to a relational database (e.g., MySQL, PostgreSQL) using an ORM (Object-Relational Mapping) library.
  - Implement functions/methods to perform CRUD operations on the database.

6. Unit Testing:

Task:

Write unit tests for at least two functions/methods in your application using a testing framework of your choice (e.g., Jest, Mocha).

Details:

- **Testable Functions:**
  - Choose functions/methods that handle critical logic, such as user authentication or task creation.
  - Write test cases to cover different scenarios (positive and negative cases).

7. API Call:

Task:

Implement a feature where the frontend makes API calls to retrieve and display tasks. Use a frontend framework or library.

Details:

- **Frontend Integration:**
  - Develop a simple frontend application using a plain js, html, css or framework or library (e.g., React, Vue, Angular).
  - Implement functionality to fetch and display tasks by making API calls to your backend.

8. Authentication and Authorization:

Task:

Ensure that only authenticated users can create, update, and delete tasks (created by them). Implement role-based authorization. Administrator can view all task and perform edit and delete operation.

Details:

- **JWT Authentication:**
  - Use JSON Web Tokens (JWT) for user authentication.
  - Include the JWT in the headers of protected endpoints.
  - Verify the JWT on the server to authenticate users.
- **Authorization:**
  - Implement role-based authorization to distinguish between regular users and administrators.
  - Allow only authorized users to perform certain actions (e.g., task creation, deletion).

9. Security Measures:

Task:

Implement input validation to prevent SQL injection attacks. Protect against XSS and CSRF attacks.

Details:

- **Input Validation:**
  - Sanitize and validate user inputs to prevent SQL injection attacks.
  - Use parameterized queries or prepared statements for database operations.
- **XSS Protection:**
  - Sanitize user-generated content before rendering it in the frontend to prevent Cross-Site Scripting (XSS).
- **CSRF Protection:**

- Implement Cross-Site Request Forgery (CSRF) protection using tokens.
- Include CSRF tokens in forms and verify them on the server side.

10. Submission Guidelines:

Details:

- **Codebase:**
  - Organize your code in a modular structure.
  - Include comments for clarity and documentation.
- **Documentation:**
  - Provide a README file with setup instructions and API documentation.
  - Include details about API endpoints, expected request/response formats, and authentication.
- **Testing:**
  - Include unit tests and instructions on how to run them.
  - Ensure proper error handling and status codes in your API responses.
- **Git Repository:**
  - Use version control (e.g., Git) and submit a link to your repository.

11. Bonus Points:

Details:

- **Additional Features:**
  - Implement additional features like sorting tasks, filtering, or pagination.
  - Demonstrate secure password storage practices (e.g., using bcrypt).

Evaluation Criteria:

- **Correctness and Functionality:**
  - Evaluate the correctness and functionality of implemented features.

- **Code Quality:**
  - Assess the quality of code in terms of organization, readability, and best practices.
- **REST Principles:**
  - Check if RESTful principles are followed in API design.
- **Security Measures:**
  - Evaluate the effectiveness of security measures, including input validation and protection against common vulnerabilities.
- **Authentication and Authorization:**
  - Check the implementation of user authentication and role-based authorization.
- **Frontend Integration:**
  - Assess the successful integration of frontend and backend components.
- **Documentation and Testing:**
  - Review the completeness of documentation and the effectiveness of unit tests.
- **Bonus Tasks:**
  - Consider bonus tasks for additional points.

Submission Guidelines:

- Commit your project to a public GitLab/Github repository from the initial setup.
- Share the GitLab/Github repository URL so we can track your progress and review your code.
- Include a comprehensive README file that outlines your approach, architecture diagram, screenshots, and future scope ideas.
- Commit your code frequently with informative messages that reflect your problem-solving process.

This task provides a platform to showcase your skills and passion for this expertise development. We encourage you to be creative, demonstrate your expertise, and impress us with your ability to build robust and scalable applications. Good luck!