Coding Exercise: Task Management Web Application

Objective: Your task is to design and implement a simple Task Management Web Application using a frontend framework of your choice and a backend API service.

Requirements:

1. **Backend API**:
   - Create a RESTful API or a GraphQL API to manage tasks.
   - Implement the following endpoints or equivalents:
     - **GET /tasks**: Fetch all tasks.
     - **GET /tasks/:id**: Fetch a single task by ID.
     - **POST /tasks**: Add a new task.
     - **PUT /tasks/:id**: Update a task by ID.
     - **DELETE /tasks/:id**: Delete a task by ID.
   - Use a persistent data storage (e.g., a relational database, NoSQL database, or an ORM like SQLAlchemy for Python).
   - Include error handling (e.g., for incorrect route access or invalid data inputs).
2. **Frontend**:
   - Use a frontend framework or library (e.g., React, Vue, Angular).
   - Implement the following views/pages:
     - **List View**: Display all tasks with the ability to delete a task.
     - **Details View**: Display details of a single task.
     - **Add/Edit View**: A form to add a new task or edit an existing one.
   - Implement responsive design (either using a framework like Bootstrap, Tailwind, or manual CSS).
   - Connect the frontend to the backend API to perform CRUD operations.
3. **Authentication** (Bonus):
   - Implement a simple user authentication system.
   - Users should be able to register, log in, and log out.
   - Only logged-in users can create, update, or delete tasks.
4. **Deployment** (Bonus):
   - Deploy the frontend and backend services, either separately or as a combined app, to a cloud provider of your choice (e.g., AWS, Heroku, Netlify, Vercel).
   - Provide the URL for the live application.

5. **Documentation**:
    - Include a README.md file describing the project setup, endpoints/APIs, and any other necessary information.
    - Document the API using tools like Swagger or Postman, or within the README.md.

Evaluation Criteria:

- **Backend Logic**: Structure, clarity, error handling, and efficiency of the server-side code.
- **Frontend Design**: Responsiveness, usability, and aesthetics of the user interface.
- **Data Persistence**: Correct implementation of the database, including schema design and data retrieval/update methods.
- **Code Quality**: Clear, readable, and maintainable code with appropriate comments.
- **Authentication**: Secure implementation of user authentication and authorization (if attempted).
- **Deployment**: Successful deployment of the application (if attempted).
- **Documentation**: Clarity and completeness of the README.md and API documentation.

Submission:

- Provide a link to the GitHub/GitLab repository containing the code.
- If deployed, provide the URL for the live application.
- Include any necessary setup instructions and credentials in the README.md.

Note: This exercise tests a candidate's abilities across both frontend and backend domains, simulating real-world full-stack development scenarios. The candidate should ensure that they pay attention to the user experience, application design, and data integrity.