Practical Assignment for Junior Developer position

Assignment Overview:

You are tasked with building a small web application that allows users to manage a list of tasks. The application should have a front-end interface for user interaction and a back-end server to handle data storage and retrieval.

Duration:

48 hours

Evaluation Criteria:

- **Code Quality**: Clean, well-documented code with meaningful comments and proper use of version control (e.g., Git).
- **Functionality**: The application should meet all functional requirements outlined below.
- **Responsiveness**: The application should be responsive and work well on both desktop and mobile devices.
- **User Interface Design**: A clean, intuitive, and visually appealing user interface.

Functional Requirements:

Front-End:

1. Task List Page:

- Display a list of tasks.
- Each task should show a title, description, and status (completed or not).
- Users should be able to add new tasks.
- Users should be able to edit existing tasks.
- Users should be able to delete tasks.
- Users should be able to mark tasks as completed or not completed.

2. Add/Edit Task Form:

- A form for adding a new task or editing an existing task.
- The form should have fields for title and description.
- The form should have a checkbox or toggle for marking the task as completed.

3. Responsiveness:

- The application should be fully responsive and usable on both desktop and mobile devices.

Back-End:

1. Task Management API:

- Provide RESTful API endpoints to create, read, update, and delete tasks.
- Use a persistent data storage method (e.g., a database) to store tasks.

2. Endpoints:

- `GET /tasks`: Retrieve a list of tasks.
- `GET /tasks/:id`: Retrieve a specific task by ID.
- `POST /tasks`: Create a new task.
- `PUT /tasks/:id`: Update an existing task by ID.
- `DELETE /tasks/:id`: Delete a task by ID.

Technical Requirements:

Front-End:

- **Framework:** Use a modern front-end framework (e.g., React, Vue, Angular).
- **CSS:** Use a CSS framework (e.g., Bootstrap, Tailwind) or write your own responsive CSS.
- **State Management:** Use state management appropriate to the chosen framework (e.g., Redux for React).
- **Form Handling:** Proper handling and validation of form data.

Back-End:

- **Language:** Use a back-end language and framework of your choice (e.g., Node.js with Express, Python with Flask/Django).

- **Database:** Use a relational database (e.g., SQLite, PostgreSQL) or a NoSQL database (e.g., MongoDB).

- **API:** Implement and document RESTful API endpoints.

- **Security:** Basic security measures for API endpoints.

<u>Submission Guidelines:</u>

1. Git Repository: Host your project on a public Git repository (e.g., GitHub, GitLab).

2. Documentation: Include a README file with:

  - An overview of the project.
  - Instructions on how to set up and run the project locally.
  - Brief documentation of your API endpoints.

3. Deployment: Optionally, deploy your application to a free hosting service (e.g., Vercel for front-end, Heroku for back-end) and provide the URL in the README.

<u>Tips for Success:</u>

- Focus on completing the core features first before adding any extra functionality.
- Ensure your application is responsive and accessible.
- Write clean, maintainable code and include comments where necessary.
- Test your application thoroughly to ensure all features work as expected.

Good luck!