Full-Stack React and Node.js Developer Internship Assignment

Assignment Title:

Build a Food Delivery System: Backend API with Node.js and Frontend Dashboard with React.js

Note: We also consider submissions that include only the **Frontend** or only the **Backend** for evaluation.

Objective:

The goal is to assess your skills in Node.js and React.js by designing a food delivery system that includes user authentication, menu management, and order tracking.

Part 1: Backend (Node.js)

Requirements:

1. **Express Server Setup:**
    - Set up an Express server that listens on port 5000 (or any other port).
2. **MongoDB Database:**
    - Use MongoDB Atlas or a local MongoDB instance for database storage.
    - Create two models:
        - **User Model:**
            - `username`: A string (required).
            - `password`: A hashed string (required).
        - **Menu Model:**
            - `name`: A string (required).
            - `category`: A string (e.g., Appetizers, Main Course, Desserts).
            - `price`: A number (required).
            - `availability`: A boolean (default: true).
        - **Order Model:**
            - `userId`: Reference to the User who placed the order.
            - `items`: Array of menu items (menu item ID and quantity).
            - `totalAmount`: Calculated total price.
            - `status`: String (e.g., "Pending", "Completed").

        - `createdAt`: Timestamp (auto-generated).

3. **API Endpoints:**
    - **Authentication:**
        - `POST /register`: Register a new user.
        - `POST /login`: Login a user and return a JWT token.
    - **Menu Management:**
        - `GET /menu`: Fetch all menu items.
        - `POST /menu`: Add a new menu item.
        - `PUT /menu/:id`: Update a menu item.
        - `DELETE /menu/:id`: Delete a menu item.
    - **Order Management:**
        - `POST /order`: Place an order with selected menu items and quantities.
        - `GET /orders`: Fetch all orders of a logged-in user.
4. **Validation & Error Handling:**
    - Validate required fields like `username`, `password`, and `menu item fields`.
    - Handle invalid data gracefully (e.g., missing fields, incorrect data types).

Tech Stack:

- Node.js, Express.js
- MongoDB
- Mongoose for schema management

Part 2: Frontend (React.js)

Requirements:

1. **React Application Setup:**
    - Use `Create React App` or an alternative setup to create the project.
2. **Pages and Components:**
    - **Login Page:**
        - A login form that accepts username and password.
        - On successful login, store the JWT token locally.
    - **Menu Page:**
        - Display all menu items in a grid layout.
        - Add options to create, update, and delete menu items.
    - **Cart Component:**
        - Allow users to add menu items to a cart with quantities.
    - **Order Page:**
        - Display the cart items, calculate the total price, and allow the user to place an order.
        - After placing the order, show the user their order history.
3. **State Management:**

    - Use `React Context` or `Redux` to manage the application state (e.g., user session, menu items, cart).
4. **API Integration:**
    - Use `Axios` or the Fetch API to interact with the backend API for CRUD operations.
5. **Styling:**
    - Use a CSS framework like TailwindCSS, Material-UI, or Bootstrap.
    - Ensure responsiveness for both desktop and mobile views.

3. UI and Styling:

- Use **CSS** or a UI framework like **Bootstrap** or **Material-UI** to style the application.
- Ensure the interface is **user-friendly**, with intuitive forms for adding, editing, and deleting tasks.
- The app should be **responsive**, working well on both desktop and mobile devices.

Bonus Features (Optional, for Extra Credit):

For extra credit, you may implement the following features:

- **Search/Filter**: Allow users to search tasks by title or filter tasks by status (completed or pending).
- **Pagination/Infinite Scroll**: Implement pagination or infinite scroll for displaying large task lists.
- **Authentication**: Implement JWT-based authentication for users to register, log in, and manage tasks individually.
- **Sorting**: Enable sorting of tasks by title, creation date, or status.

Deliverables:

1. **Code Repository**:

    - A **public GitHub repository** containing the complete source code for both the front-end and back-end.
    - Ensure your code is well-organized, and commit messages are clear and meaningful.
    - Include a **README file** that includes:
        - Setup instructions for both the front-end and back-end.
        - A brief project description and feature list.
        - Any assumptions, challenges, or limitations faced during development.
2. **Code Walkthrough Video**:

    - Create a **screen recording** that walks through the following:
        - **Code Structure**: Explain the folder and file structure of both the front-end and back-end.
        - **Back-End Explanation**: Walk through the Express server setup, the API routes, and database interactions.
        - **Front-End Explanation**: Demonstrate the React components, explain state management, and show how data is fetched from the API and rendered.
        - **API Integration**: Show how the front-end communicates with the back-end via API calls (GET, POST, PUT, DELETE).
        - **Deployment**: Walk through deploying both the front-end and back-end, showing the live version of the application.

3. **Deployment**:

    - **React Front-End**: Deploy the React application using a platform like **Vercel**, **Netlify**, or **GitHub Pages**.
    - **Node.js Back-End**: Deploy the Node.js server using platforms like **Heroku**, **Railway**, or **Render**.
    - **MongoDB Database**: Use MongoDB Atlas or another cloud database service to host the database and connect it to the back-end.
    - Provide links to the deployed live application:
        - **Front-End URL** (e.g., https://your-app-name.vercel.app)
        - **Back-End URL** (e.g., https://your-api.herokuapp.com)

4. **Live Demo**:

    - Ensure both the front-end and back-end are deployed and fully operational. Provide links to the live application and ensure functionality like task management works properly in a browser.

Submission Instructions:

- **File Naming Conventions**:

    - **GitHub Repository**: Name your repository as `full-stack-task-management-app`.
    - **Code Walkthrough Video**: Name your video file as `task-management-walkthrough-[YourName].mp4`.
    - **GitHub Username**: Please ensure your GitHub username is included in the repository URL and the submission form.
    - **UI Design**: You are free to use a design library like **Material-UI**, **Bootstrap**, or create your own simple design. The focus is on functionality, but a clean, user-friendly UI is encouraged.

- **Submission Form**:
  - Submit your completed assignment via the **Google Form here** (make sure to follow the naming conventions for the GitHub repository and video).
  - Include the following links in the form:
    - **GitHub Repository** link.
    - **Code Walkthrough Video** link.
    - **Front-End Deployment URL**.
    - **Back-End Deployment URL**.

Evaluation Criteria:

Your submission will be evaluated based on the following:

- **Code Quality**: Clean, modular, and well-structured code following best practices.
- **Functionality**: The application should meet all core requirements (add, edit, delete, and view tasks).
- **Error Handling**: Proper validation and error handling implemented both on the front-end and back-end.
- **User Interface**: A responsive, intuitive, and visually appealing UI.
- **Documentation**: A well-written README with clear setup instructions and explanations.
- **Deployment**: Successful deployment with both front-end and back-end accessible online.