**Full-Stack Task Manager**

**Task 1: Backend Logic (Node.js/Express)**

**Objective**: Implement a backend API using Node.js and Express to handle user authentication.

**Requirements**:

**1. User Authentication:**

- Create an API endpoint `/api/login` that accepts POST requests with JSON body containing `username` and `password`. **DONE**

- Validate the `username` and `password` against a predefined list of users stored in the database. **DONE**

- Return a JWT (JSON Web Token) if the authentication is successful, otherwise return an appropriate error message. **DONE**

**2. JWT Authentication Middleware:**

- Implement middleware to verify JWT tokens for protected routes. **DONE**

- Secure the task management API endpoints to require authentication. **DONE**

**Task 2: Database (SQLite/PostgreSQL/MongoDB)**

**Objective**: Set up a database to store user information and tasks. **DONE**

**Requirements:**

**1.** **User Table:**

- Create a table named `users` with columns for `id`, `username`, `password` (hashed), and any other relevant fields.

- Populate the `users` table with a few sample users. **DONE**

**2. Tasks Table**:

- Create a table named `tasks` with columns for `id`, `title`, `description`, `status`, `created\_at`, and any other relevant fields. **DONE**

**3.** **CRUD Operations:**

- Implement database queries to perform CRUD operations for managing users and tasks.

- Use proper error handling and transactions where necessary. **DONE**

**Task 3: Frontend (React/Next.js)**

**Objective:** Develop a dynamic user interface using React/Next.js for interacting with the backend APIs. **DONE**

**Requirements**:

**1. Authentication UI:**

- Implement a login page where users can input their username and password. **DONE**

- Upon successful authentication, store the JWT token securely for subsequent API requests. **DONE**

**2. Dashboard UI:**

- Display a list of tasks fetched from the backend API. **DONE**

- Provide options to add, edit, and delete tasks. **DONE**

- Implement pagination to limit the number of tasks displayed per page.**DONE**

**3. Task Management UI:**

- Include forms for adding and editing tasks. **DONE**

- Allow users to update the status of tasks (e.g., mark as completed). **DONE**

**4. Error Handling and Feedback:**

- Handle API request errors and display appropriate error messages to users.

- Provide feedback for successful operations. **DONE**

**5. UI Design and Responsiveness:**

- Design the UI with a focus on aesthetics and user experience. **DONE**

- Ensure the application is responsive and works well on different screen sizes and devices. **DONE**

**6. Routing:**

- Use React Router or Next.js routing for navigation between login and dashboard. **DONE**

- Implement proper route guarding to restrict access to the dashboard page if the user is not authenticated. **DONE**

**Submission Guidelines:**

- Set up a GitHub repository for your project and share the link with us.

- Include instructions on how to run both the frontend and backend applications locally and connect them together.

**UI Preview:**

Feel free to customize the UI according to your preferences and design skills.

This task should provide a comprehensive test of your full-stack development skills, including backend logic, database management, and frontend development with React/Next.js. If you have any questions or need clarification on any aspect of the task, feel free to ask!