# ****TaskPro Application - Technical Documentation****

### 1. Project Overview

**Technology Stack:**

* **Frontend:** React.js + TypeScript
* **Styling:** Tailwind CSS
* **State Management:** Zustand
* **Icons:** Lucide React
* **Notifications:** React Hot Toast
* **API:** JSONPlaceholder (for fetching tasks)
* **Authentication:** Custom implementation with local storage
* **Deployment:** Netlify

### 2. Project Structure

plaintext

Copy code

src/

├── components/

│ ├── auth/

│ │ └── RegisterForm.tsx

│ ├── tasks/

│ │ └── TaskItem.tsx

│ └── ui/

│ └── Button.tsx

├── pages/

│ ├── Dashboard.tsx

│ ├── Login.tsx

│ └── Register.tsx

├── store/

│ ├── useAuthStore.ts

│ └── useTaskStore.ts

├── services/

│ └── api.ts

└── types/

└── task.ts

### 3. Features Implementation

#### 3.1 Authentication System

**Authentication Flow:**

1. **Registration (/register):**
   * User fills out the registration form.
   * Form validation is implemented.
   * User data is stored.
   * Redirects to login upon successful registration.
2. **Login (/login):**
   * User enters credentials.
   * Credentials are validated.
   * Authentication token is stored.
   * Redirects to the dashboard on successful login.
3. **Protected Routes:**
   * Checks the authentication status.
   * Redirects unauthorized users to login.

#### 3.2 Task Management

**Task Operations:**

1. **Create Task:**
   * Adds a task with a title.
   * Generates a unique ID for the task.
   * Sets initial completed status.
   * Stores the task creation timestamp.
2. **Update Task:**
   * Allows users to edit task titles.
   * Toggles the completion status.
   * Displays a success notification upon update.
3. **Delete Task:**
   * Removes a task from the store.
   * Displays a confirmation notification after deletion.

### 4. State Management

#### 4.1 Auth Store

ts

Copy code

interface AuthState {

user: User | null;

token: string | null;

isAuthenticated: boolean;

register: (name: string, email: string, password: string) => Promise<void>;

login: (email: string, password: string) => Promise<void>;

logout: () => void;

}

#### 4.2 Task Store

ts

Copy code

interface TaskState {

tasks: Task[]; // List of tasks

filter: TaskFilter; // Current task filter (e.g., completed, pending)

isLoading: boolean; // Loading state for fetching tasks

error: string | null; // Error state if any operation fails

// Methods for task operations

}

### 5. API Integration

**API Endpoints:**

* **GET** /todos: Fetch all tasks
* **POST** /todos: Create a new task
* **PATCH** /todos/:id: Update a task's details
* **DELETE** /todos/:id: Delete a specific task

These endpoints interact with the **JSONPlaceholder** API for simulating task management operations.

### 6. UI Components

#### 6.1 Reusable Components

* **Button Component:**
  + variant: 'primary' | 'secondary' | 'danger' | 'success'
  + isLoading?: boolean
  + icon?: React.ReactNode

#### 6.2 Task Components

* **TaskList:** Displays the list of tasks, can be filtered by completion status.
* **TaskItem:** Displays an individual task with its title, status, and actions (edit, delete).
* **TaskForm:** A form to create or edit tasks.
* **TaskFilter:** Provides an interface to filter tasks by status (completed or pending).

### 7. Styling and Theme

#### 7.1 Dark Mode

ts

Copy code

// Theme Toggle

const { isDark, toggleTheme } = useTheme();

* The application supports dark mode, toggled with a button that applies the 'dark' class to the HTML element.

#### 7.2 Responsive Design

css

Copy code

/\* Tailwind CSS Breakpoints \*/

sm: 640px // Mobile

md: 768px // Tablet

lg: 1024px // Desktop

xl: 1280px // Large Desktop

The application is fully responsive and adjusts the layout based on the screen size using Tailwind CSS breakpoints.

### 8. Error Handling

**Error States:**

* **API Errors**: Errors during fetching, creating, updating, or deleting tasks.
* **Form Validation Errors**: Invalid input during task creation or editing.
* **Authentication Errors**: Invalid credentials or missing tokens.
* **Loading States**: Indicating the application is waiting for an API response.

**Notification System:**

* Success messages: toast.success('Operation successful')
* Error messages: toast.error('Operation failed')

This notification system helps provide real-time feedback to users on their actions.

### 9. Local Development

**Setup Instructions:**

bash

Copy code

# Install dependencies

npm install

# Start the development server

npm run dev

# Build for production

npm run build

# Preview production build

npm run preview

1. **Install Dependencies:** Install all required packages with npm install.
2. **Development Server:** Run npm run dev to start the local development server.
3. **Production Build:** Use npm run build to build the application for production.
4. **Preview Build:** Run npm run preview to preview the production build locally.

### 10. Deployment

#### 10.1 Build Process

bash

Copy code

# Production build

npm run build

# Output: /build directory

The production-ready application is built into the /build directory, ready for deployment.

#### 10.2 Netlify Deployment

**Steps to Deploy on Netlify:**

1. **Push Code to GitHub:**
   * Initialize a Git repository and commit your code.

bash

Copy code

git init

git add .

git commit -m "Initial commit"

1. **Connect Repository to Netlify:**
   * Create a new site on [Netlify](https://app.netlify.com/).
   * Connect your GitHub repository to Netlify.
2. **Configure Build Settings:**
   * Set the **Build Command** to npm run build.
   * Set the **Publish Directory** to build/.
3. **Deploy:**
   * Click **Deploy site** on Netlify.
   * After deployment is complete, Netlify provides a live URL for your application.

Netlify automatically handles continuous deployment, so every new commit pushed to the connected GitHub repository will trigger a rebuild and deployment.

### 11. Future Enhancements

**Planned Features:**

* Real backend integration (for persistent data storage).
* Email verification during registration.
* Password reset functionality.
* Task categories and priorities (e.g., high, medium, low).
* Due dates and task reminders.
* Search functionality for tasks.
* Collaborative features like sharing tasks with others or teams.

### Conclusion

This documentation provides a complete guide to the architecture, functionality, and deployment process of the **Task Tracker Application**. The application is designed for easy use and quick deployment through **Netlify**, with all core task management features integrated and tested.

For further enhancements, backend integration and additional features will be implemented to extend the application's capabilities.