## **Frontend To-Do Application Competency Task**

### **Objective**

Design and implement a single-page To-Do application using **React (TypeScript)**. The application will simulate interactions with a REST API by using **mock data**. This task focuses on frontend skills, including component design, state management, API integration, and user experience.

### **Features**

#### **1. Core Functionality**

* **To-Do List Display**: The main view should show a list of To-Do items. Each item should display its title, a description, and a checkbox to mark it as complete.
* **Add New To-Do**: A form or input field to add new To-Do items. Submitting the form should add the new item to the list.
* **Update To-Do**: Users should be able to edit an existing To-Do item. This could be done by clicking on the item to open an editable view or a modal.
* **Mark as Complete**: The checkbox on each item should toggle its completion status.
* **Delete To-Do**: A button or icon on each item to remove it from the list.

#### **2. Mock API Integration**

Instead of a real backend, you'll simulate API calls using **mock data** and asynchronous functions. This demonstrates your ability to handle real-world API patterns without a running backend.

* Simulate a GET request to fetch the initial To-Do list.
* Simulate POST, PUT, and DELETE requests for creating, updating, and deleting To-Do items.
* Each "API call" should have a simulated delay (e.g., using setTimeout) to replicate network latency. This is crucial for demonstrating proper loading state management.

#### **3. State Management and UI/UX**

* **Loading States**: Display a loading indicator (like a spinner or skeleton loader) while "fetching" or "saving" data.
* **Error Handling**: Show a clear error message (e.g., a red text block or a toast notification) if a simulated API call fails.
* **Strict Typing**: Use **TypeScript** throughout the application. Define clear interfaces for your To-Do items and API responses. **Avoid using any**.
* **Component Structure**: The application should be composed of well-defined, reusable components (e.g., ToDoList, ToDoItem, AddToDoForm).
* **Responsive Styling**: Apply minimal styling with plain CSS or a framework like Tailwind CSS to ensure the application looks good on both desktop and mobile screens.

#### **4. React Fundamentals**

* Use modern React hooks like useState and useEffect.
* Demonstrate a clear understanding of component lifecycles and side effects.

### **Submission Requirements**

* **GitHub Repository**: Create a public GitHub repository with your completed project. The code should be well-organized and include clear README.md instructions on how to install and run the application.
* **Code Quality**: The code should be clean, well-commented, and follow best practices.
* **Functionality**: All features listed above must be fully implemented.