Step 1: Set Up Git Repository

1. **Create a new directory for the project**:

Open a terminal and run:

mkdir todo-list-app

cd todo-list-app

1. **Initialize a new Git repository**:

Inside the todo-list-app directory, initialize Git:

git init

1. **Create a .gitignore file**:

This file helps Git ignore specific files or folders. Since you’ll have dependencies saved in node\_modules/ (which can get very large), it’s best to exclude them from the repository. This command creates a .gitignore file with node\_modules/ listed, so it won’t be tracked by Git.

echo "node\_modules/" > .gitignore

Step 2: **Verify the setup**

* Once the installation finishes, you should see a bunch of new files in the project directory.
* To test if everything is set up correctly, start the development server:
* npm start
* Run the create-react-app Command

npx create-react-app .

* Once the setup is complete, start the server from the terminal by running:

npm start

* default browser at <http://localhost:3000>

Step3: install dependencies

* for this project use the default dependencies. Later if we like to add new libraries, we can do so with npm install <package-name>

Step4: Project Structure

1.**Create a components folder**:

* Inside the src directory, create a new folder called components. This is where all React components (individual pieces of the app) will live.

2. **Create Component Files**:

* Inside src/components/, create the following component files:
  + TodoItem.js: This component will represent a single to-do item.
  + TodoList.js: This component will render a list of TodoItem components.
  + AddTodo.js: This component will contain the form to add new to-do items.

Step 5: Develop the Application Components

**1. App Component (App.js)**

**Purpose**: This is the main component of the app, holding the to-do data and the logic for managing tasks (adding, editing, deleting, marking as completed).

**Code Breakdown**:

* **useState**: This React hook creates the todos state, which stores the list of tasks. setTodos is used to update this list whenever a task is added, edited, deleted, or marked as completed.
* **Functions**:
  + **addTodo function**: Adds a new task to the existing todos list and updates the state.
  + **toggleComplete function**: Changes the completion status of a task by toggling the completed property.
  + **deleteTodo function**: Removes a task from the todos list by filtering it out based on the task’s index.
  + **editTodo function**: Updates the text of a specific task in the todos list.

**Components inside**:

* **<AddTodo />**: A form component for adding new tasks to the list.
* **<TodoList />**: A component for displaying all tasks in the todos list.

**In Summary**: The App component controls the data and functions for adding, editing, deleting, and marking tasks as completed. It passes these as props to child components (AddTodo and TodoList).

**2. AddTodo Component (AddTodo.js)**

**Purpose**: This component is a form that lets users type a new task and add it to the list.

**Code Breakdown**:

* **useState**: Creates a state variable newTodo to store the text of the new task as the user types.
* **handleSubmit function**:
  + **Prevents page refresh**: Calls e.preventDefault() to stop the form submission from refreshing the page.
  + **Adds the new task**: If newTodo isn’t empty, it uses the addTodo function from App to add the new task.
  + **Clears input**: Resets newTodo to an empty string after the task is added, clearing the input box.

**In Summary**: The AddTodo component manages user input for new tasks and updates the App component with new tasks while resetting the input form.

**3. TodoList Component (TodoList.js)**

**Purpose**: This component receives the list of tasks from App and renders each task as a TodoItem component.

**Code Breakdown**:

* **todos.map**: Loops through the todos list and creates a TodoItem for each task. Each task gets its own TodoItem component.
* **key**: Each TodoItem receives a unique key (here, index) to help React track changes and improve performance.

**In Summary**: TodoList serves as a container for TodoItem components, ensuring each task in the list is displayed as a separate item.

**4. TodoItem Component (TodoItem.js)**

**Purpose**: This component represents an individual task. It displays the task’s text, a checkbox for marking it as completed, and buttons for editing or deleting the task.

**Code Breakdown**:

* **useState for Editing**:
  + **isEditing**: Controls whether the task is in edit mode (where the text can be modified) or view mode.
  + **editedText**: Holds the updated text while editing.
* **Functions**:
  + **toggleComplete**: Toggles the completed status of the task by calling the toggleComplete function passed as a prop from App.
  + **handleEdit**: Switches the task into edit mode.
  + **handleSave**: Saves the edited text by calling editTodo, exits edit mode, and updates the task text.
* **Elements**:
  + **input type="checkbox"**: A checkbox representing whether the task is completed, where checked is set to todo.completed.
  + **span and input**: Displays the task text as a span when not editing, or an editable input field when in edit mode.

**In Summary**: TodoItem manages individual tasks, allowing users to mark them as complete, edit the text, or delete the task entirely.

Step 6: Create a Remote Repository on GitHub

* + 1. add origin with the URL: git remote add origin <https://github.com/ytt2028/todolistapp.git>
    2. Push the code

git push -u origin main