Manipulating the State

IIHT

Time To Complete: 10 to 12 hr

CONTENTS

1 Project Abstract

State manipulation is fundamental to building dynamic and interactive React applications. This project introduces participants to modifying component state based on user actions. Using a Task List application as the context, users will learn how to toggle data within a list by updating state immutably. The application demonstrates how to lift state up, pass down handler functions as props, and respond to events by conditionally rendering updated UI—all key practices in real-world React development.

2 Problem Statement

You are required to develop a React-based Task List application where users can mark tasks as completed or incomplete. The main `App` component stores the list of tasks using the `useState` hook. Each task is rendered using the `Task` component, which receives its data and a toggle function via props. When a user clicks a button, the completion status of the task should be toggled, updating the UI in real time. This project focuses on understanding how to manipulate state in a predictable and immutable way in React.

3 Proposed Manipulating the State Application Wireframe

UI needs improvisation and modification as per given use case and to make test cases passed.

3.1 ScreenShots

Task List

Task 1

This is the first task

Status: Incomplete

Mark Completed

Task 2

This is the second task

Status: Incomplete

Mark Completed

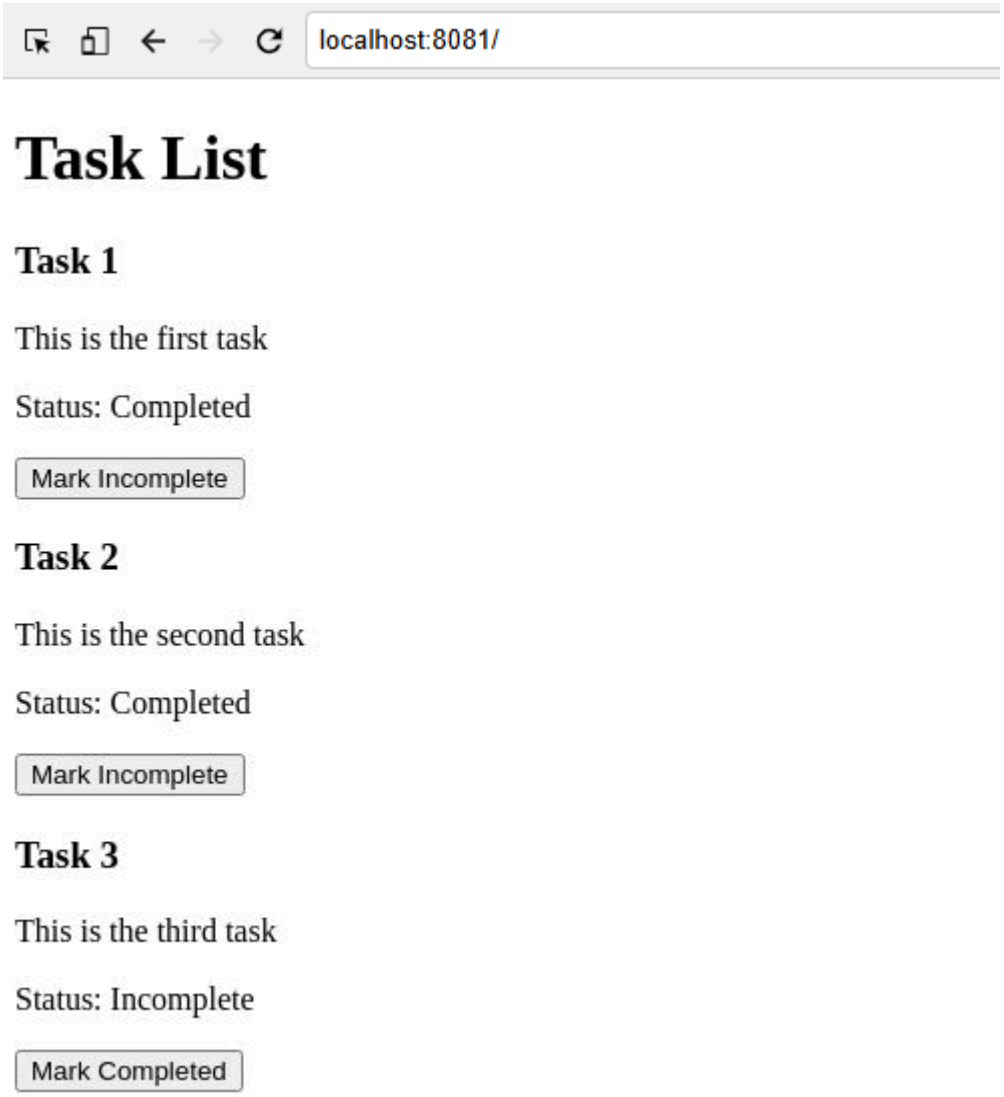
Task 3

This is the third task

Status: Completed

Mark Incomplete

Update Task



4 BUSINESS-REQUIREMENT:

As an application developer, develop the Manipulating the State (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Welcome Page	As a user I should be able to visit the welcome page as the default page. Acceptance criteria:

		1. Display a heading **“Task List”**. 2. Render a list of tasks using the `Task` component. 3. Each task must show: ● Name ● Description ● Completion status (Completed or Incomplete) 4. Each task must have a **button** that: ● Toggles its `completed` status between `true` and `false` ● Updates the UI instantly upon click **State & Method Overview:** **AppComponent:** **State Variables:** 1. tasks ● Type: `array` ● Purpose: Stores a list of task objects. ● Each task contains: ➔ `name` (string) ➔ `description` (string) ➔ `completed` (boolean) **Method: `toggleCompletion(index):`** 1. Toggles the `completed` value for the selected task by: ● Mapping through the `tasks` array ● Identifying the task by index ● Creating a new version of the task with flipped `completed` status ● Updating the entire list using `setTasks()` **Props & Child Interaction:** **Task Component** **Props Received:** 1. `task`: An object with task details.

		2. `onComplete`: A method (from parent) to toggle the task's status. **Responsibilities:** 1. Display task name and description. 2. Show current **status** (Completed / Incomplete). 3. Render a **button** that triggers the `onComplete` method. ● Button text changes dynamically based on the current status. **HTML Structure:** 1. **App Component:** ● Top-level `<div>` contains: ➔ `<h1>` with "Task List" ➔ A list of `<Task />` components generated using `.map()` 2. **Task Component:** ● Inside a `<div>`: ➔ Heading (`<h3>`) for task name ➔ Paragraph (`<p>`) for description ➔ Paragraph for status (`<p>Status: ...`) ➔ A `<button>` to toggle status ➢ Label changes based on whether the task is completed **Dynamic Behavior:** 1. Button click calls `toggleCompletion()` in the parent via prop. 2. UI updates in real-time by flipping the value of `completed`. 3. State is updated immutably using `.map()` and `setTasks`. **** Kindly refer to the screenshots for any clarifications. ****

5 CONSTRAINTS

1. You should be able to press the "TAB" key and "SHIFT + TAB" to navigate from top field to bottom field and vice-versa.

6 MANDATORY ASSESSMENT GUIDELINES

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to
   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. This editor Auto Saves the code.
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
   Note: The application will not run in the local browser
7. You can follow series of command to setup React environment once you are in your project-name folder:
   a. npm install -> Will install all dependencies -> takes 10 to 15 min.
   b. npm run start -> To compile and deploy the project in browser. You can press the <Ctrl> key while clicking on localhost:4200 to open the project in the browser -> takes 2 to 3 min.
   c. npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.
8. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.