Assignment :- 2

Assignment: Building a Dynamic Task List Application

In this assignment, you will create a dynamic task list application using React functional components. The application will allow users to add tasks, mark tasks as completed, and filter tasks based on their completion status.  
Use css modules and bootstrap for styling

Use react router for routing

### **Task 1: Setting Up the Project (Topics: Create React App)**

* Create a new React application using Create React App.
* Set up the basic project structure and make sure it's running without errors.

### **Task 2: Implementing Components and Props (Topics: Components and Props)**

* Create a functional component called Task that receives task as a prop and displays the task details.
* Create another functional component called TaskList that receives an array of tasks as a prop and renders a list of Task components.

### **Task 3: Managing State and Lifecycle (Topics: State and Lifecycle)**

* Implement a state variable tasks in your main App component to store the list of tasks.
* Add a form component (TaskForm) allowing users to add new tasks. When a new task is submitted, update the tasks state to include the new task.

### **Task 4: Handling Events (Topics: Handling Events)**

* Implement an event handler function to toggle the completion status of a task when it's clicked.
* Attach the event handler to the Task component so that when a user clicks on a task, it toggles its completion status.

### **Task 5: Implementing Conditional Rendering (Topics: Conditional Rendering)**

* Add a button to toggle between showing all tasks, completed tasks, and incomplete tasks.
* Implement logic to filter the tasks based on their completion status and render the filtered tasks in the TaskList component accordingly.

### **Task 6: Using Hooks (Topics: useState, useEffect, useCallback, useMemo, useRef)**

* Use the useState hook to manage the state of tasks and the filter type.
* Use the useEffect hook to update the document title with the number of incomplete tasks whenever the tasks state changes.
* Use the useCallback hook to memoize the event handler function for toggling task completion.
* Use the useRef hook to create a ref for the input field in the TaskForm component and focus on it when the component mounts.

### **Task 7: Creating Custom Hooks (Topics: Custom Hooks)**

* Create a custom hook called useLocalStorage that takes a key and initial value as arguments and returns a stateful value stored in localStorage, along with a function to update it.
* Use the useLocalStorage hook to persist the tasks state in localStorage so that tasks persist even after a page refresh.

### **Task 8: Adhering to Rules of Hooks (Topics: Rules of Hooks)**

* Ensure that all hooks are called at the top level of your functional components, not inside loops or conditions.
* Make sure to follow the order of hooks: useState, useEffect, useCallback, useMemo, useRef.

### **Task 9: Implementing Controlled Components (Topics: Controlled Components)**

* Convert the input field in the TaskForm component into a controlled component by storing its value in the component's state.
* Implement an event handler to update the input field's value as the user types.