Week 2 React Weekend Task: Building a Simple To-Do Application

---

Objective:
In this weekend task, you will apply the concepts learned in the React course to build a simple To-Do application. This task will help you consolidate your understanding of React components, state management, component lifecycle methods, and React Hooks. You will create a dynamic to-do list where users can add, view, edit, delete, and filter tasks. The task covers essential React topics, including functional components, hooks (`useState`, `useEffect`), and handling events.

Task Overview:

You will be required to:

1. Set Up the React Environment:

- Install **Node.js** and **npm** (if not already installed).
- Set up a new React project using `create-react-app`.

```
bash
Copy code
npx create-react-app todo-app
```

- Ensure that the environment is correctly set up and running.

2. Create the To-Do Application Structure:

- Inside the `src/` folder, create the following directories and files:
  - **components/**: For storing all the React components.
  - **App.js**: The main file that will hold the layout and logic.
  - **styles/**: For storing CSS styles.

3. Implementing the To-Do List with Functional Components:

- **Problem**: Create the main structure for your To-Do app with the following components:
  - `Header`: Displays the title of the application.
  - `TodoInput`: Allows users to enter a new task.
  - `TodoList`: Displays the list of tasks.
  - `TodoItem`: Represents an individual task with an option to mark it as complete, edit it, or delete it.

- **Hint**: Use React functional components and props to pass data between components.

---

4. Managing State with the `useState` Hook:

- **Problem**: Use the `useState` hook to manage the state of your to-do list.
    - Create state variables for storing tasks (as an array).
    - Handle the addition of new tasks. Each task should include a unique ID, title, and a `completed` status (default to `false`).
    - Ensure that when a new task is added, the input field is cleared.
- **Code Example**:

```
const [tasks, setTasks] = useState([]);
const [newTask, setNewTask] = useState("");

const handleAddTask = () => {
  setTasks([...tasks, { id: Date.now(), title: newTask, completed:
false }]);
  setNewTask(""); // Clear the input after adding a task
};
```

5. Editing and Deleting To-Do Items:

- **Problem**: Implement functionality to edit and delete tasks.
    - For editing: Allow users to click on a task, change the title, and save the update.
    - For deleting: Provide a delete button next to each task that removes it from the list.
- **Hint**: Use the `map()` function to display the list of tasks and the `filter()` function to delete a task by its unique ID.

---

6. Component Lifecycle with `useEffect`:

- **Problem**: Use the `useEffect` hook to demonstrate component lifecycle behavior.
    - Display a message in the console whenever the `tasks` state is updated (i.e., when a new task is added, deleted, or edited).
    - Implement logic to save the tasks to **localStorage** every time they change, so the to-do list persists when the page is refreshed.
- **Code Example**:

```
useEffect(() => {
  console.log("Task list updated");
  localStorage.setItem("tasks", JSON.stringify(tasks));
}, [tasks]);
```

7. Filtering To-Do Items:

- **Problem**: Add a feature to filter tasks based on their completion status.
    - Create buttons to filter tasks: "All", "Active", "Completed".

    - Use the `filter()` function to show only the relevant tasks based on the selected filter.
- **Hint**: Add a state variable for the current filter and conditionally render the tasks based on their `completed` status.

---

8. Styling the To-Do Application:

- **Problem**: Style your To-Do app using CSS to make it visually appealing.
    - Use flexbox or CSS grid to create a clean layout.
    - Add some hover and click effects to the buttons.
    - Use different colors to indicate the completion status of tasks (e.g., strikethrough for completed tasks).

9. Handling Events:

- **Problem**: Handle user interactions using React event handlers.
    - Add `onClick` events to buttons for adding, editing, deleting, and filtering tasks.
    - Use `onChange` events to update the input field as the user types a new task.

---

10. BONUS: Improving the To-Do App with Advanced Features (Optional):

- Implement drag-and-drop functionality using the `react-dnd` library to reorder tasks.
- Add due dates for tasks using the `react-datepicker` library and display overdue tasks with a warning message.
- Add the ability to prioritize tasks with a dropdown selection.

Submission:

- Submit the full project folder in a compressed format (.zip).
- Ensure all functionalities are working as expected.
- Include a README file with instructions on how to install dependencies and run the project.