Mobile Programming - Lab 04

ReactJS Lab 1: todo list app

2025-10-20

Lab Project: The Ultimate React Todo List App

Project Goal

Build a fully functional Todo List application. Users will be able to view tasks, add new tasks, and mark them as complete. This project is the ideal way to practice components, props, state, event handling, and conditional rendering.

Setup

1. Create a new React project using your preferred tool:

- **Vite (Recommended):**
  `npm create vite@latest my-react-app -- --template react`

2. Navigate into the project directory: `cd my-react-app`
3. Start the development server: `npm run dev` (for Vite) or `npm start` (for Create React App).
4. Clean up the `src` folder by removing the boilerplate content inside `App.jsx`.

Milestone 1: Displaying a Static List of Todos

Focus: Creating reusable components, structuring the UI with JSX, and passing data down the component tree using props.

Tasks:

1. **Create Initial Data:**
   - Inside your `App.jsx` file, create a constant array named `initialTodos`.
   - This array will hold a few todo objects. Each object should have an `id` (e.g., 1), `text` (e.g., "Learn React"), and `isCompleted` (boolean, `false`).
2. **Create a `TodoItem` Component:**
   - Create a new file: `src/components/TodoItem.jsx`.

- This component will receive a single todo object via **props**.
- It should return a list item (`<li>`) element that displays the todo's `text`.

3. **Create a `TodoList` Component:**
   - Create a new file: `src/components/TodoList.jsx`.
   - This component will receive the `initialTodos` array as a prop.
   - It should use the `.map()` method to iterate over the array and render a `TodoItem` component for each todo.
   - **Important:** Remember to pass a unique `key` prop to each `TodoItem` in the list (e.g., `key={todo.id}`).
4. **Assemble in `App.jsx`:**
   - In `App.jsx`, import the `TodoList` component.
   - Render the `TodoList`, passing your `initialTodos` array to it as a prop.

**** Testing Your Progress (Milestone 1)****

- Your browser should display a static, unordered list of the todos from your `initialTodos` array.
- Open the browser's developer console. There should be **no errors or warnings**, especially about missing `key` props.
- Try changing the text of a todo in the `initialTodos` array in `App.jsx` and confirm that the UI updates automatically when you save the file.

Milestone 2: Adding New Todos with State

Focus: Making the application dynamic by managing the list of todos with the `useState` hook and handling user input events.

Tasks:

1. **Manage Todos with State:**
   - In `App.jsx`, import the `useState` hook from React.
   - Create a state variable to manage the list of todos. Initialize it with your `initialTodos` data:
     jsx `const [todos, setTodos] = useState(initialTodos);`
2. **Create an Input Form:**
   - In `App.jsx`, add an input field and a button to allow users to add new todos. A `<form>` element is best for this.
   - Create another piece of state to manage the value of the input field: `const [newTodoText, setNewTodoText] = useState('');`.
3. **Create an Event Handler:**

- Create a function `addTodo(todoText)`. This function will:
  - Create a new todo object with a unique `id` (you can use `Date.now()` for a simple unique ID), the provided `text`, and `isCompleted: false`.
  - Update the `todos` state by adding the new todo to the existing array. Use the spread (`...`) operator to do this immutably: `setTodos([...todos, newTodo]);`.

4. **Connect the Form to State and Handlers:**
   - Use the `onChange` event on the input field to update `newTodoText` as the user types.
   - Use the `onSubmit` event on the form. In the submission handler function, call `addTodo()` with the current `newTodoText`, and then clear the input field by setting `setNewTodoText('')`. Remember to call `event.preventDefault()` to stop the page from reloading.

Testing Your Progress (Milestone 2)

- You should now see an input field and a button on your page.
- Try typing in the input field. The text should appear.
- Add a new todo. It should instantly appear at the bottom of your list.
- Open the React DevTools. Find the `App` component and watch its `todos` state array. You should see the new todo object get added to the array when you click the button.

Milestone 3: Marking Todos as Complete

Focus: Handling interactions on child components, passing functions as props ("lifting state up"), and using conditional rendering to change UI based on state.

Tasks:

1. **Create the Toggle Function:**
   - In `App.jsx`, create a function `toggleTodo(idToToggle)`.
   - This function should update the `todos` state. To do this, `.map()` over the current `todos` array. For each `todo`, check if its `id` matches `idToToggle`.
   - If it matches, return a *new* todo object with its `isCompleted` property flipped (`...todo, isCompleted: !todo.isCompleted`).
   - If it doesn't match, just return the original `todo`.
   - Finally, call `setTodos()` with the new array you created.
2. **Pass the Function Down:**
   - Pass the `toggleTodo` function as a prop from `App` to `TodoList`, and then from `TodoList` to each `TodoItem`.
3. **Handle the Click Event:**

- In `TodoItem.jsx`, add an `onClick` event listener to the `<li>` element.
- When the `<li>` is clicked, it should call the `toggleTodo` function it received from props, passing up its own `id`.

4. **Apply Conditional Styling:**
   - In `TodoItem.jsx`, use a **ternary operator** inside the `style` prop or `className` prop of the `<li>`.
   - If `todo.isCompleted` is `true`, apply a style to cross out the text (e.g., `textDecoration: 'line-through'`). Otherwise, apply no special style.

Testing Your Progress (Milestone 3)

- Click on any todo item in the list. It should become crossed-out.
- Click on it again. The line-through style should disappear.
- Open the React DevTools and inspect the `App` component's state. When you click a todo, you should see the `isCompleted` boolean for that specific todo object flip between `true` and `false`.

Challenge Section

1. **Delete a Todo:**
   - Create a `deleteTodo(idToDelete)` function in `App.jsx`. This function should use the `.filter()` method to create a new array containing all todos *except* the one with the matching ID.
   - Add a "Delete" button inside each `TodoItem.jsx`.
   - Pass the `deleteTodo` function down as a prop and call it when the button is clicked.
2. **Component Composition (`TodoForm`):**
   - Create a new component, `src/components/TodoForm.jsx`.
   - Move the `<form>`, `<input>`, and `<button>` for adding a new todo into this new component.
   - The `App` component will now pass the `addTodo` function as a prop to `TodoForm`. The `TodoForm` component will be responsible for managing its own input state and calling the prop function on submit. This is a very common and important React pattern.
3. **Filter Todos:**
   - Add three buttons to `App.jsx`: "All", "Active", and "Completed".
   - Create a new piece of state to track the current filter: `const [filter, setFilter] = useState('all');`.
   - Before passing the `todos` to `TodoList`, create a new `filteredTodos` variable. Use an `if/else` or `switch` statement to filter the `todos` array based on the current `filter` state. Pass this `filteredTodos` array to the `TodoList` instead of the original one.