6. State Management and Lifting State Up

Last updated by | Subramanya Dixit | May 5, 2025 at 7:59 PM GMT+5:30

📘 Why State Management Is Important

Managing state effectively allows different components in a React app to stay in sync. It improves data flow, component behavior, and user interaction, especially in larger applications.

🧠 Key Concepts

✅ Local Component State

State that is specific to one component.

```
function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>{count}</button>;
}
```

✅ Lifting State Up

Share state between sibling components by moving it to the closest common parent.

```
function Parent() {
  const [value, setValue] = useState('');
  return (
    <>
      <Input value={value} onChange={setValue} />
      <Display value={value} />
    </>
  );
}

function Input({ value, onChange }) {
  return <input value={value} onChange={(e) => onChange(e.target.value)} />;
}

function Display({ value }) {
  return <p>{value}</p>;
}
```

✅ Prop Drilling

Passing state or functions deeply through many components.

✅ Global State (via Context)

Used when state needs to be shared across many parts of the app.

```
const ThemeContext = React.createContext();

function App() {
  const [theme, setTheme] = useState('light');
  return (
    <ThemeContext.Provider value={theme}>
      <Child />
    </ThemeContext.Provider>
  );
}

function Child() {
  const theme = useContext(ThemeContext);
  return <div>Theme: {theme}</div>;
}
```

📊 Visual Overview

```
flowchart TD
  A[Parent Component] --> B[Input Child Component]
  A --> C[Display Child Component]
  B -->|onChange| A
  A -->|state| C
```

💡 Guidelines

- Start with local state; lift it when needed.
- Avoid unnecessary prop drilling.
- Use Context for global state.
- Keep state as close as possible to where it's used.

📝 Practice Exercises

1. Create a parent-child form with state lifted to the parent.
2. Update sibling components based on shared parent state.
3. Add global theme state using `useContext`.
4. Refactor a deeply nested component to avoid prop drilling.

❓ Quiz Questions

1. What does "lifting state up" mean?
a) Storing state in Redux
b) Passing state from child to parent
✅ **c) Moving shared state to the closest common ancestor**
d) Avoiding useState

2. When should you use Context API?
a) For every component
b) Only for class components
✅ **c) When state needs to be shared across many components**
d) To avoid using props at all

---

3. What is prop drilling?
a) Using multiple contexts
b) Changing props in place
✅ **c) Passing data through many levels of components**
d) Breaking the render tree

---

4. What is a drawback of lifting state too far up the tree?
a) React crashes
✅ **b) Unnecessary re-renders and complexity**
c) Memory leaks
d) It disables hooks
