Mastering React Essentials

Context, Reducers, Refs, Conditionals, Lists & Routing

useContext: Sharing Data Without Prop Drilling

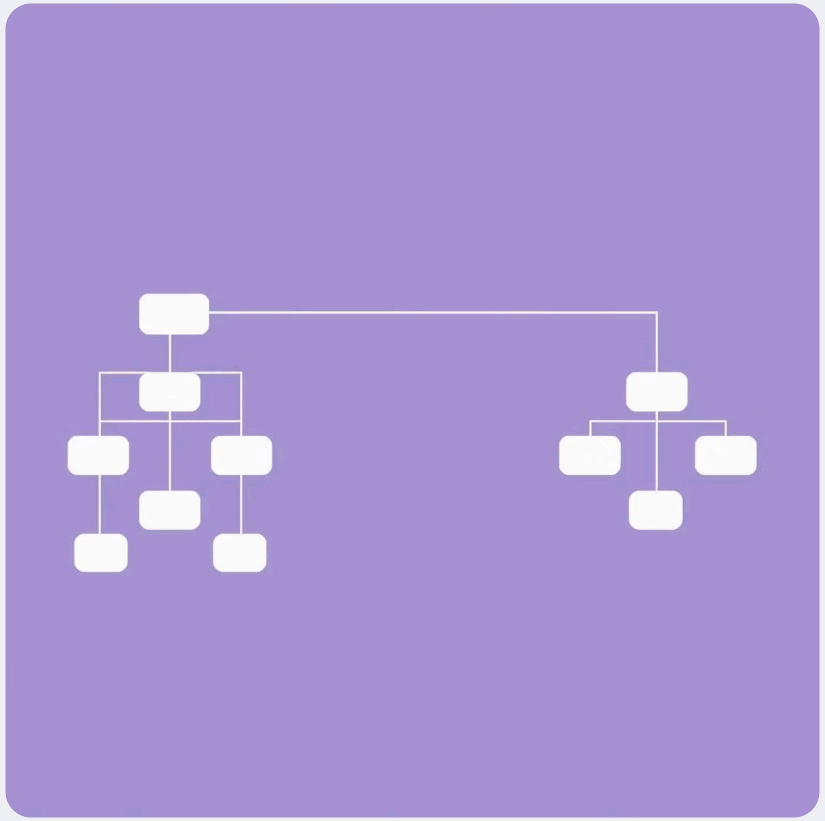
React's `useContext` Hook lets components access shared data deep in the tree without passing props manually. It simplifies data flow significantly.

The Problem: Prop Drilling

Passing data down through many layers of components, even if intermediate components don't need it.

The Solution: useContext

Create a context, provide the data at the top, and consume it directly where needed, eliminating unnecessary prop chains.



useReducer: Managing Complex State Logic

`useReducer` centralises state updates via a reducer function, similar to Redux but built-in. It's ideal for complex state changes and can be combined with `useContext` to provide global state and dispatch.

Before: useState for Tasks

```
const [tasks, setTasks] = useState([]);
const addTask = (text) => {
  setTasks([...tasks, { id: Date.now(), text }]);
};
```

After: useReducer for Tasks

```
const [tasks, dispatch] = useReducer(tasksReducer, []);
const addTask = (text) => {
  dispatch({ type: 'added', id: Date.now(), text });
};
```

The reducer function handles various actions like 'added', 'changed', or 'deleted', making state logic predictable and testable.

useRef: Persisting Values Without Re-renders

`useRef` holds mutable values that persist across renders without triggering re-renders. It's excellent for direct DOM interaction, storing timers, or retaining previous state values.

- **DOM Access**

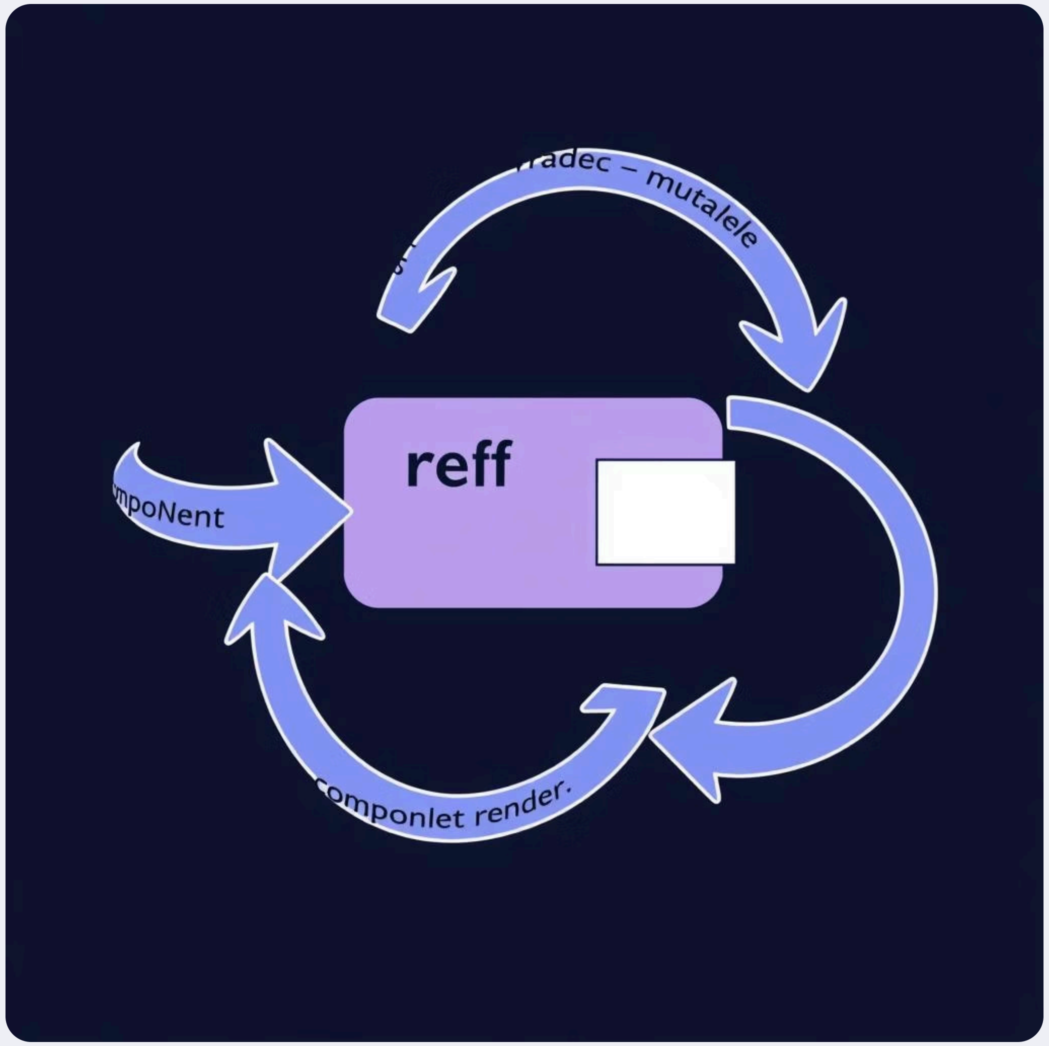
  Directly interact with DOM elements, such as focusing an input field on component mount.

- **Mutable Values**

  Store any mutable value (e.g., a timer ID) that needs to persist across renders without causing UI updates.

- **Tracking Previous State**

  Keep a reference to the previous value of a prop or state variable.



React Router: Navigating Single Page Apps

React Router enables client-side routing, rendering components based on the URL without full page reloads. This creates a seamless, app-like experience within a single page application (SPA).

URL Change

User navigates or types a new URL, e.g., `/about`.

Route Match

React Router matches the URL to a defined `<Route>`.

Component Render

The corresponding component is rendered, dynamically updating the UI.

Key components include `<BrowserRouter>` to enable routing, `<Routes>` to group routes, and `<Route>` to define specific paths to components.

Links: Navigating Without Reloads

The `<Link>` component replaces standard `<a>` tags for internal navigation within your React application. It prevents full page reloads, making transitions instantaneous and preserving application state.

Traditional HTML Link

```
<a href="profile.html">Go to Profile</a>
```

Causes a full page refresh, losing state.

React Router Link

```
<Link to="/profile">Go to Profile</Link>
```

Updates URL and renders component instantly, maintaining state.

This declarative approach to navigation is fundamental for building smooth and efficient single-page applications.

Conclusion: Building Scalable React Apps

Mastering these core React concepts empowers you to build robust, maintainable, and performant applications.

1

State Management

`useContext` and `useReducer` offer a powerful, scalable approach to managing complex global state.

2

Performance

`useRef` enables efficient DOM manipulation and value persistence without unnecessary re-renders.

3

Dynamic UIs

Conditionals and list rendering provide the tools for creating responsive and interactive user interfaces.

4

Seamless Navigation

React Router and `<Link>` deliver smooth, client-side routing for an enhanced user experience.

Embrace these tools to tackle modern web development challenges effectively.