**7 Best Practices React State Management**

Are you tired of your React JS application becoming a tangled mess of states? Do you find it difficult to keep track of the flow of data across components? If you answered yes to any of these questions, you’re in luck! In this article, we’ll explore the best practices for managing state in React JS and show you how to keep your app state under control.

Table of Contents:

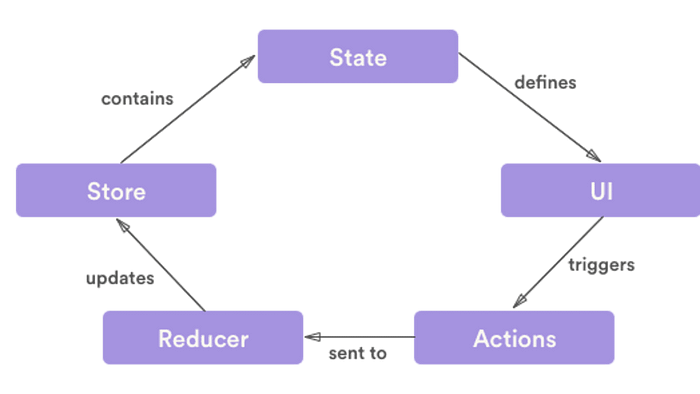
∘ [Why is State Management Important?](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#01f1)  
∘ [1. Understand React’s Local Component State](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#a3ca)  
∘ [2. Keep State Close to Where It’s Used](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#c719)  
∘ [3. Use Immutability for State Updates](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#b542)  
∘ [4. Leverage Redux or MobX for Advanced State Management](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#c8a8)  
∘ [5. Optimize State Updates with Memoization](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#54f8)   
∘ [6. Consider Using React Context API](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#5b05)  
∘ [7. Stay Up to Date with React’s Latest Features and Libraries](https://medium.com/@asiandigitalhub/7-best-practices-react-state-management-1dd1ce4eaa15#754d)

**Why is State Management Important?**

Before we dive into the best practices, let’s quickly recap why state management is crucial in React JS. State represents the data that changes over time in your application, and properly managing it ensures your app functions correctly.

It helps you avoid common pitfalls like data inconsistency, prop drilling, and unnecessary re-renders. By following these best practices, you’ll make your app more maintainable, scalable, and performant.

**1. Understand React’s Local Component State**

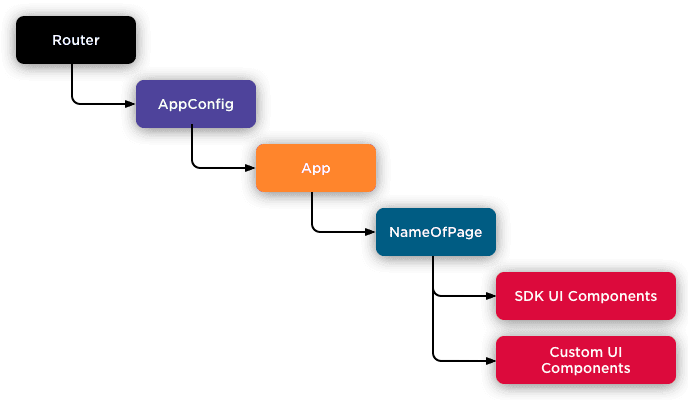


Source: LoginRadius

In React, state is managed within components using the useState or useReducer hooks. It’s essential to understand the local component state and how it differs from global state management libraries like Redux or MobX.

Local state should be used for data that is specific to a single component and doesn’t need to be shared across the entire application. For complex apps with multiple components, consider using a global state management solution.

**2. Keep State Close to Where It’s Used**

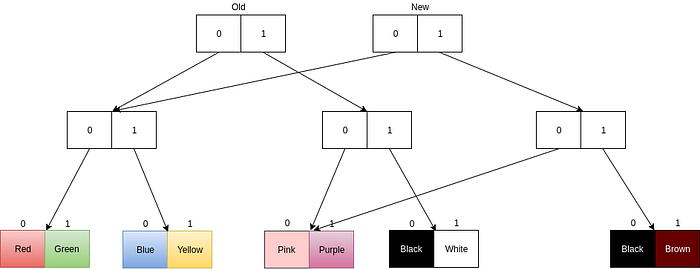


Source: Mobify Devcenter

A good practice is to keep the state as close to the components that need it as possible. This approach, known as “lifting state up,” reduces complexity and makes your app easier to reason about.

If multiple components need access to the same state, lift that state up to their nearest common ancestor. By doing so, you create a single source of truth for that state, making it easier to manage and update.

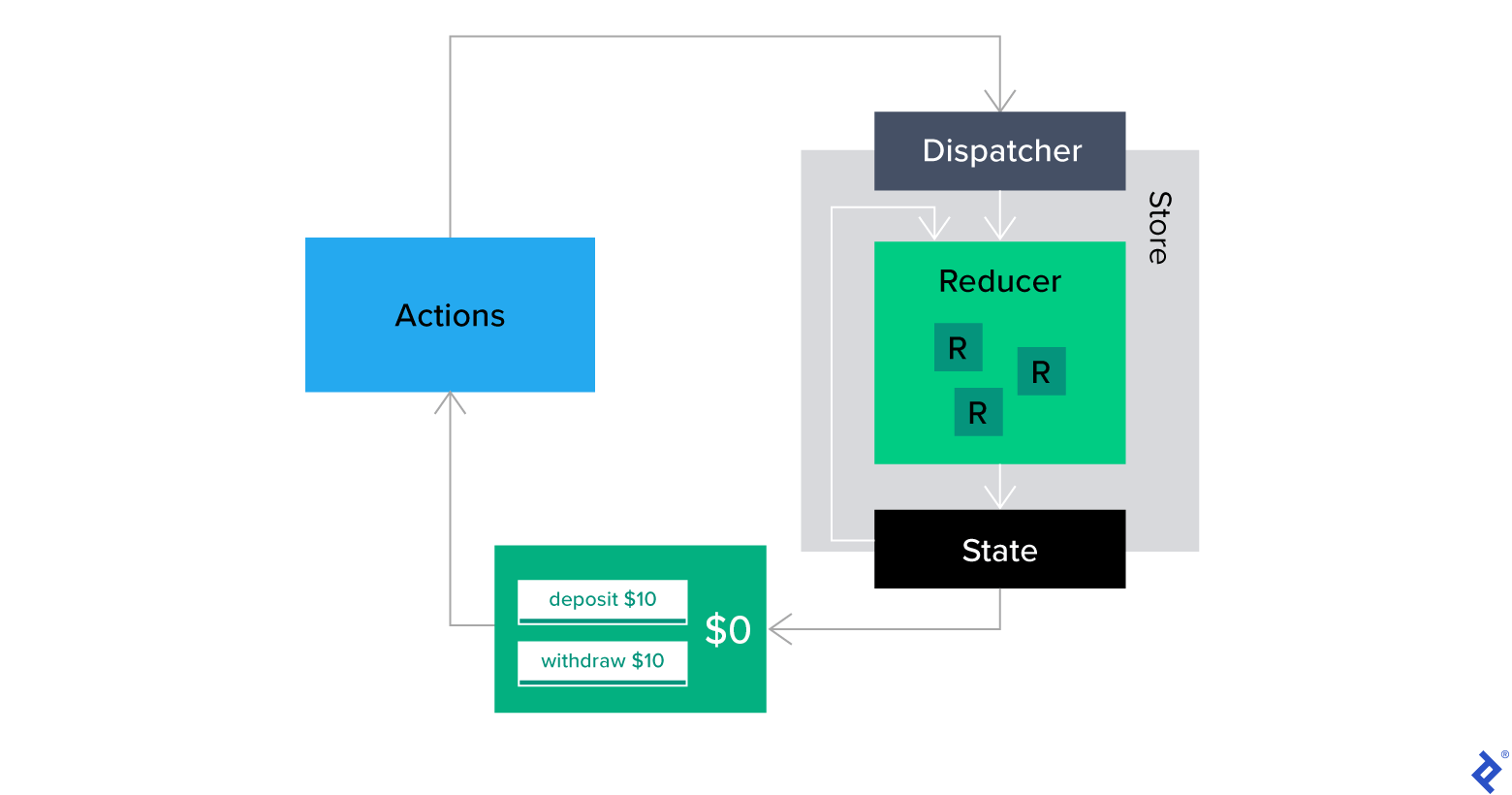
**3. Use Immutability for State Updates**



Source: HackerNoon

React’s state should never be mutated directly. Instead, use immutable data structures and techniques to update state. Immutable.js, Immer, or the spread operator are popular choices for creating new state objects without mutating the original. By following this practice, you ensure that your state updates are predictable and don’t introduce hard-to-debug side effects.

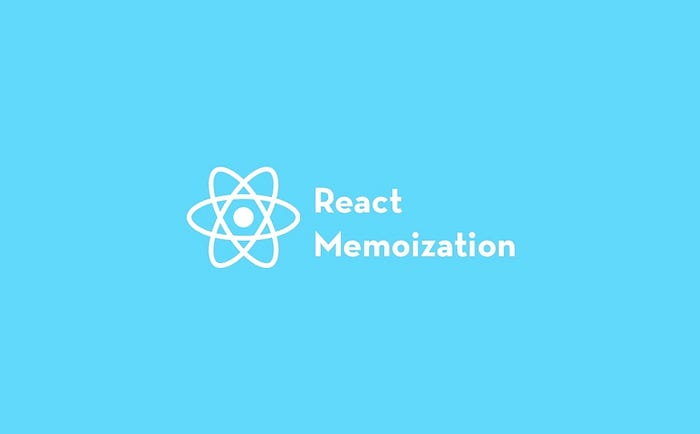
**4. Leverage Redux or MobX for Advanced State Management**



Source: Toptal

As your React JS application grows more complex, you might need a more robust state management solution. Redux and MobX are popular choices for managing global state and provide powerful tools to handle large-scale applications. However, only introduce these libraries if you genuinely need them. For smaller projects, local component state management might be sufficient.

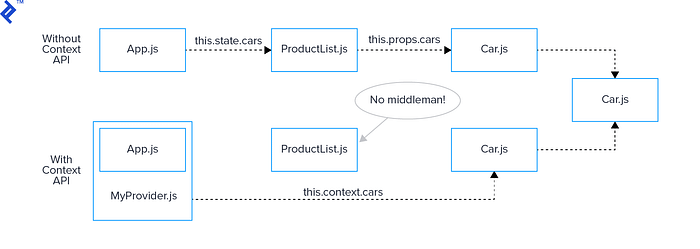
**5. Optimize State Updates with Memoization**



Source: alexdevero.com

React’s reconciliation algorithm can be expensive, especially if your app has a lot of state and complex UI components. To optimize performance, use memoization techniques like React.memo or useMemo to prevent unnecessary re-renders. These functions allow you to cache the result of a component based on its dependencies, reducing the workload on React’s reconciliation process.

**6. Consider Using React Context API**

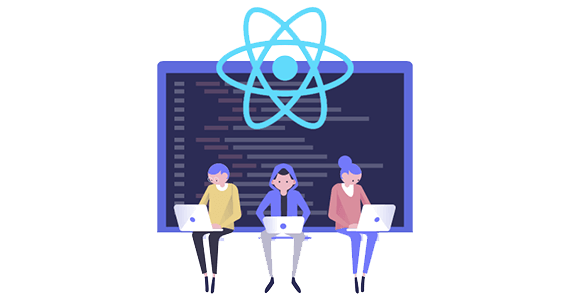


Source: Toptal

React’s Context API is a built-in solution for sharing state across components without resorting to a global state management library. It allows you to pass data through the component tree without passing props explicitly at every level.

Context is ideal for smaller applications or when you need to share state between a few closely related components. However, be cautious with using Context, as it can lead to performance issues if overused.

**7. Stay Up to Date with React’s Latest Features and Libraries**



Source: VimirLab

React JS is an evolving framework, with new features and libraries being introduced regularly. It’s essential to stay up to date with the latest advancements to maintain your app’s performance and effectiveness.

By following these best practices, you’ll be well-equipped to tackle state management challenges in your React JS applications. Remember to choose the right approach for your project’s size and complexity, and always strive for simplicity and maintainability.

Your knowledge journey doesn’t stop here! If you’ve found value in this article, there’s more to come. Stay connected by following us for exciting updates.