**DIFFERENT STATE MANAGEMENT AND WHEN TO USE THEM OVER THE OTHERS**

7 Top React State Management Libraries

Before we dive into the different state mangement tools, let’s dig a little into what state mangement is about.

State Management involves the process of managing data accross your application. State Management is crucial to help communicate states between multiple components in a way that’s easy and scalable.

Developers use **React hooks** to access and share component states across different components. But when dealing with a large number of components, the complexity becomes too much for React hooks to handle. Developers need to leverage **React state management** libraries in such cases. But how do you choose the state management library that’s best for you? It depends….And that’s why you should keep reading .

There are so many React state management libraries but we’ll be dealing with 7 of them now; Redux, Hookstate, Recoil, Jotai, Rematch, Zustand, MobX. **NB:** I will be dwelling on redux more and for that i will reserve it for the last

**Top 7 React state management libraries**

**HOOKSTATE: A state management** based on native React hooks and has virtually no boilerplate or actions or reducers. Hookstate is unique in the way that it tracks rendered and updated states. It uses concepts like local state and global state that are very similar to native React state management. But it also introduces other concepts like **nested state, scoped state, and asynchronous state**. These features allow developers to access complex states, render large states, and use ‘promises’ to delay certain actions.Hookstate works almost exactly like the React useState Hook. In fact, with Hookstate, creating a global state is just as easy as creating a local state with the useState Hook.

**RECOIL:** Recoil is known for being developed by Facebook. The main concepts used in Recoil are **atoms** and **selectors.** Recoil is a brand-new experimental JavaScript state management library that addresses many of the problems developers face when developing large applications using the existing Context API.Recoil helps you eliminate the unnecessary re-renders which happens while using the Context API or Redux. A case study of where an app is using the Context API, when the state in the context is changed, all the components in which even the data is not changed will also re-render. Then Recoil saves the day using an Atom, an individual state, the components which are subscribed to atoms only re-render. Atoms are units of state, which can be subscribed individually unlike many state management systems. When an atom is updated, each component that is subscribed to it is re-rendered with the updated value. A **selector** is a pure function that accepts an atom as an input. When an atom is updated which is subscribed by the selector, the selector function will be re-evaluated.

**JOTAI:** Jotai is based on the new Recoil pattern and library by Facebook. Jotai is like Recoil.

Joitai Mascot

But where its standout features include in its smaller size, cleaner API, greater TypeScript support, and deeper maturity. If you need to read and write atoms outside React, Jotai provides store API.

**REMATCH**: Rematch is Redux best practices without the boilerplate. Rematch was built on Redux.

**ZUSTAND**: [Zustand](https://github.com/pmndrs/zustand) is a fast and scalable state management solution built by the developers of [Jotai](https://jotai.org/) and [React springs](https://react-spring.io/). Zustand is known for its simplicity, using hooks to manage states without boilerplate code .Also, it renders components only if their state changes.

**MobX**: MobX follows the OOP paradigm and uses the observer/observables pattern to manage state. MobX creates an observable data model that your observers or components can refer to. It then tracks the data accessed by each component and renders it whenever the data changes.

**REDUX:** Redux is used to maintain and update data across your applications for multiple components to share, all while remaining independent of the components.

**Without Redux,** you would have to make data dependent on the components and pass it through different components to where it’s needed. In the picture below, we only need some data in the parent and inner child components, but we’re forced to pass it to all the components (including the child component that doesn’t need it) to get it to where it’s needed.

**With Redux,** we can make state data independent of the components, and when needed, a component can access or update through the Redux store.

**HOW REDUX WORKS:**

There is a central store that holds the entire state of the application. Each component can access the stored state without having to send down props from one component to another.There are three core components in Redux — actions, store, and reducers.

* **ACTIONS:** are the only way you can send data from your application to your Redux store.Actions are plain JavaScript objects that must have
* a type property to indicate the type of action to be carried out, and
* a payload object that contains the information that should be used to change the state.

And actions are executed using thedispatch() method which sends the action to the store

* **REDUCERS:** Reducers are pure functions that take the current state of an application, perform an action, and return a new state.
* **STORE:** The store is a “container” (really a JavaScript object) that holds the application state, and the only way the state can change is through actions dispatched to the store.

Read up complete documentation at <https://medium.com/@nicsylvia15f/different-state-management-and-when-to-use-them-over-the-others-ca979efbc7ae>