



Understanding Redux: Beginner's guide to modern state management

May 12, 2020 - 8 min read

Aaron Xie



So, you know JavaScript. And you know frontend framework like React. You're off to the races, building your amazing single-page-application and expanding it.

Over time, your [React](#) application becomes more complex, with more app components, and more data going in and out of it. Now you're running into issues, and managing multiple simultaneous processes is turning into a headache. How do you keep it all in line?

The answer is to use **Redux**.

Today we'll learn:

- [What is state management?](#)
- [What is Redux, and why do we need it?](#)
- [Benefits and limitations of Redux](#)



- [Main concepts of Redux](#)
- [Getting started with React-Redux](#)
- [Next Steps](#)

What is state management?

State, a term from React, is an object that holds data contained within a component. It determines how a component behaves and renders. State is a central component to making dynamic pages through conditional rendering.

An easy way to grasp this concept is to understand the user interface as a function of state, meaning that a developer can change the appearance of the web application depending upon the data held by the state. Imagine you are building a [to-do-list application with React](#).

You have a todo-item component, and you want to program the component so that when a user clicks the item, it gets crossed out. You can have a state variable called `isFinished` and have its value be either `True` or `False`. Depending upon the value of `isFinished`, the todo-item can be programmed to be crossed out.

State management is simply the management of the state of multiple user interface controls or components. As developers work on larger, more complex applications, they begin using external tools to better manage the state of their application.

To make state management easier, developers often use state management libraries that allow them to create a model of their app state, update the state of components, monitor and observe changes to the state, and read state values.

Because state can be messy to manage, especially when there are a number of dynamic components, utilizing a state management system will help your future debugging.

Some popular state management tools:

- Redux



- Vuex
- Mobx
- Apollo Link State
- Unstated
- Flux

What is Redux, and why do we need it?



Redux

Redux is a lightweight state management tool for JavaScript applications, released in 2015 and created by Dan Abramov and Andrew Clark.

Redux is the most popular state management solution, helping you write apps that behave in the same way, are easy to test, and can run the same in different environments (client, server, native). One of the key ways Redux does this is by making use of a redux store, such that the entire application is handled by one state object.

According to its official documentation, Redux was founded on three core principles:

- The state of your whole application is stored in an object tree within a single store.
- Ensure the application state is read-only and requires changes to be made by emitting a descriptive action.



- To specify how the state tree is transformed by actions, you write pure reducer functions.

With the entire state of your application centralized in one location, each component has direct access to the state (at least without sending props to child components, or callback functions to parent components).

Master Redux.

Educative's **Redux course** teaches you how to use Redux with React and javascript so that you can develop powerful web applications.

[Learn Redux](#)

With the **hooks** functionality and **Context API** incorporated into React, some have questioned whether Redux is still necessary to build a larger react application. The answer is yes. Though you may not need it for a simple React application, you will need it when your applications become more complex. The Context API is not a replacement for Redux.

Problems with the Context API arise when your app expands. In a larger application, the order in which data moves can be important. With Redux, you can keep a record of the changes in your state and time travel back to these changes.

Furthermore, Redux is more efficient than React stand-alone because Context often forces re-renders.

Furthermore, while Context API has made it easier to pass data between components without using Redux, it's not a state manager, which means you're missing out on a lot of other features. Redux offers tools that make it incredibly easy for you to debug, test, and track your state.

To be sure, Redux provides scalability, easy debugging, and middlewares. It's also important to note that Context and Redux cannot be compared in the same category, as Redux is decoupled from the UI layer and is a state management system, while Context is not.



Benefits and limitations of Redux


- **State transfer:** State is stored together in a single place called the ‘store.’ While you don’t need to store all the state variables in the ‘store,’ it’s especially important to when state is being shared by multiple components or in a more complex architecture.

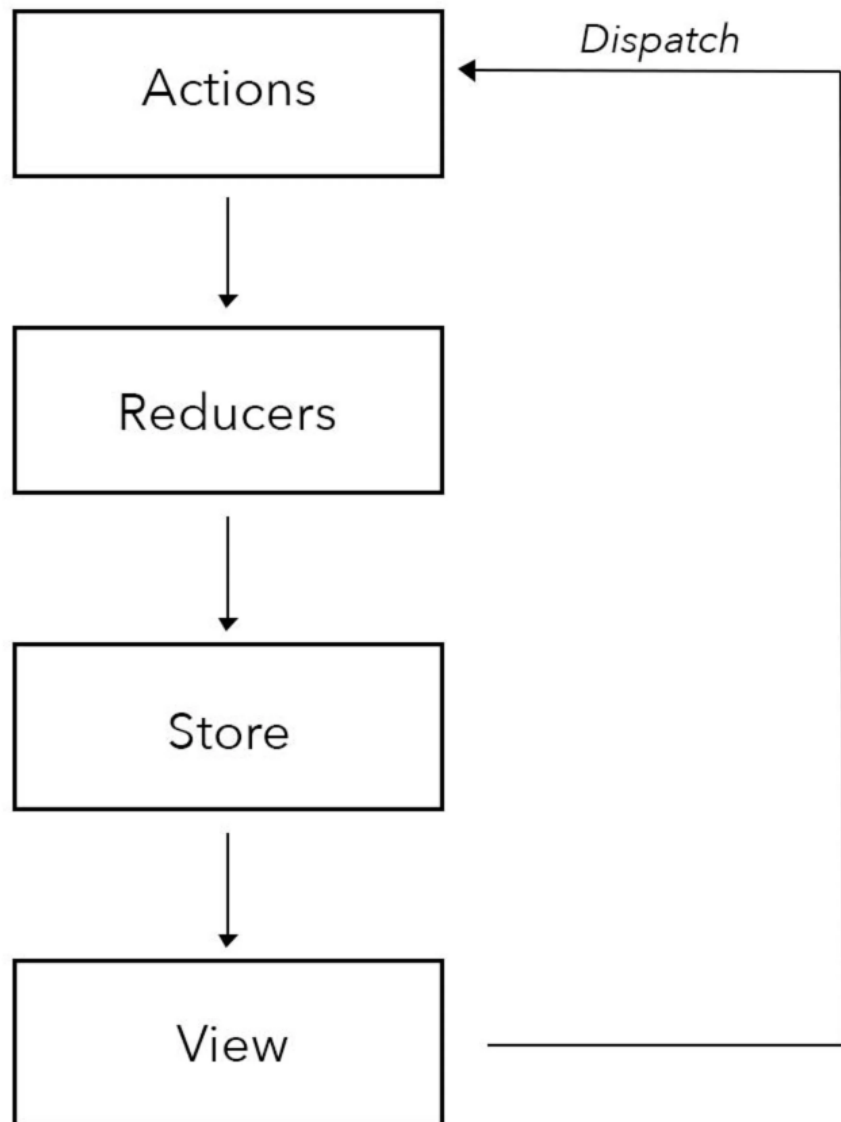
As your application grows larger, it can be increasingly difficult to identify the source of the state variables, which is why a ‘store’ is useful. It also allows you to call state data from any component easily.

- **Predictability:** Redux is “a predictable state container for JavaScript apps.” Because reducers are pure functions, the same result will always be produced when a state and action are passed in. Furthermore, the slices of state are defined for you, making the data flow more predictable.
- **Maintainability:** Redux provides a strict structure for how the code and state should be managed, which makes the architecture easy to replicate and scale for somebody who has previous experience with Redux.
- **Ease of testing and debugging:** Redux makes it easy to test and debug your code since it offers powerful tools such as Redux DevTools in which you can time travel to debug, track your changes, and much more to streamline your development process.

While Redux is something that every developer should consider utilizing when developing their application, it’s not for everyone. Setting up the Redux architecture for your application can be a difficult and seemingly unnecessary process when you’re working with a small application. It may be unnecessary overhead to use Redux unless you’re scaling a large application.

Main concepts of Redux

Naturally, using an external solution for state management means being familiar with a few rules in the development process. Redux introduces **actions**, **action creators**, **reducers**, and **stores**. Ultimately, these concepts are used to create a  simple state management architecture.



Action

Action is static information about the event that initiates a state change. When you update your state with Redux, you always start with an action. Actions are in the form of Javascript objects, containing a **type** and an optional **payload**.

Action creators

These are simple functions that help you create actions. They are functions that return action objects, and then, the returned object is sent to various reducers in the application.

Reducer



A reducer is a pure function that takes care of inputting changes to its state by returning a new state. The reducer will take in the previous state and action as parameters and return the application state. As your app grows, your single reducer will be split off into smaller reducers that manage certain parts of the state tree.

Redux store

The Redux store is the application state stored as objects. Whenever the store is updated, it will update the React components subscribed to it. You will have to create stores with Redux. The store has the responsibility of storing, reading, and updating state.

Getting started with Redux

Although Redux is used with other JavaScript libraries like [Angular](#) or [Vue.js](#), it's most commonly used for React projects. Let's take a look at a basic implementation of React-Redux.

```
1 import { createStore } from 'redux'
2
3 function count(state = 0, action) {
4   switch (action.type) {
5     case 'increase':
6       return state + 1
7     case 'decrease':
8       return state - 1
9     default:
10      return state
11   }
12 }
13
14 let store = createStore(counter)
15
16 store.subscribe(() => console.log(store.getState()))
17
18 store.dispatch({ type: 'increase' })
19 store.dispatch({ type: 'decrease' })
```



- **Line 3 - 12:** This implements a reducer, a pure function with `(state, action) => state` signature. The function transforms the initial state into the next state based on the `action.type`.
- **Line 14:** Creates a Redux store, which holds the state of the app. Its API is `{ subscribe, dispatch, getState }`. The `createStore` is part of the Redux library.
- **Line 16:** `subscribe()` is used to update the UI in response to state changes.
- **Line 18 - 19:** An action is dispatched to mutate the internal state.

Next steps

Now, you might feel ready to begin testing the waters with Redux, but don't get ahead of yourself. Redux has a pretty big learning curve initially. It's tougher to pick up on your own.

To help you along the way, Educative has created the [Understanding Redux](#) course, which is designed to give you a primer on the basics of Redux. It's an interactive course, with plenty of coding challenges to make learning Redux not just easier, but more fun.

By the end, you'll build skypey, a modern messaging app to bring your JS, React, and Redux skills altogether.

Happy learning!

Continue reading about React and Redux

- [Introducing React Design Patterns: Flux, Redux, and Context API](#)
- [React Router Tutorial: Adding Navigation to your React App](#)
- [React Bootstrap Tutorial: upgrade React apps with a CSS framework](#)

