

# React and Redux: Lesson 4

## Introduction to Redux

William Abboud

\* Covers React v15 - 16



# What is Redux ?

Docs: <https://redux.js.org/>



# Redux

Redux is a tiny JavaScript library for managing state in an entire app. It is not linked to React and can be used with other libraries but is commonly paired with React and solves a common problem of **shared state**.

A close-up, low-angle shot of a green rope net structure, likely a climbing net. The ropes are thick and green, with silver-colored metal rings at the intersections. The background is a soft, out-of-focus mix of purple and pink hues. The word "State" is overlaid in the center in a light blue, sans-serif font.

State

Photo by Clint Adair on Unsplash



# Revisitting state

State is the part or parts of your app that holds data and any other supplementary information to the current behaviour of your app.

E.g State in React components is used to hold data for that component and 99% of cases that data is linked to how the component is rendered or what the component does.



# The State problem in React

The problem with the state in React is the one way data flow nature of React. If a component at the top of the tree holds a state that 5 components 10 levels down in the tree need then you will need to pass that state data through every node in order to get it to the bottom nodes.



# How Redux helps ?

Redux gives you the ability to allocate the whole app's state in one single object and then you can grab your data from there. Like a global storage.

Redux's state holder is called the **Store**. The store holds state and that state is where your data lives.

The **Store** is the first core piece of Redux and is the embodiment of the first principle of Redux:

***Single source of truth*** *The state of your whole application is stored in an object tree within a single store.*



# The Store

The **Store** is manager of everything in Redux's architecture.

It dispatches **Actions**, it holds and controls access to the applications' state and notifies any subscriber when the state has changed.

The **Store** take a **Reducer** function which returns the whole App' state. This **Reducer** is called every time an action is dispatched and can be referred to as the main **Reducer**.

A Redux application typically has only one **Store**.

Reducer function

Old State is passed in

Next action

```
1 import { createStore } from 'redux';  
2  
3 const AppStore = createStore((state = {}, action) => {  
4   return {  
5   };  
6 });
```

This object is your state





# How Redux helps ? Continued...

The state is a read-only. The only way to change the state is to emit an **Action**.

**Actions** are the second core piece of Redux. An **Action** is an object describing what change occurred. You can think of **Actions** as just messages.

**Actions** are useful because they define a single way to inform the whole system what occurred and what change to the state is needed.

*State is read-only. The only way to change the state is to emit an action, an object describing what happened.*




# Actions

**Actions** are just messages informing the system what happened. Technically an **Action** is just a JavaScript object with a “type” property.

The “type” property specifies what is the **action**. Other than that the content of an **action** is up to you. You have total freedom of defining the payload because the **action** is just an object.

To send an **Action** use the **dispatch()** method defined in the **Store**.

The store’s main **Reducer** will be called every time an **Action** is sent.



```
8  const action = {  
9    type: "SOME_ACTION",  
10   content: "hello world"  
11  };  
12  
13  AppStore.dispatch(action);  
14
```



# How Redux helps ? Continued...

The **Store** keeps the state.

The State update is triggered by an **Action**.

Finally we update the state from within a **Reducer**.

A **Reducer** is just a function which takes the old state and the next action and returns new state based on the old state and the action.

The name **Reducer** comes from the function you supply to the method [Array.prototype.reduce\(\)](#).

*Changes are made with pure functions To specify how the state tree is transformed by actions, you write pure reducers.*



# Reducer

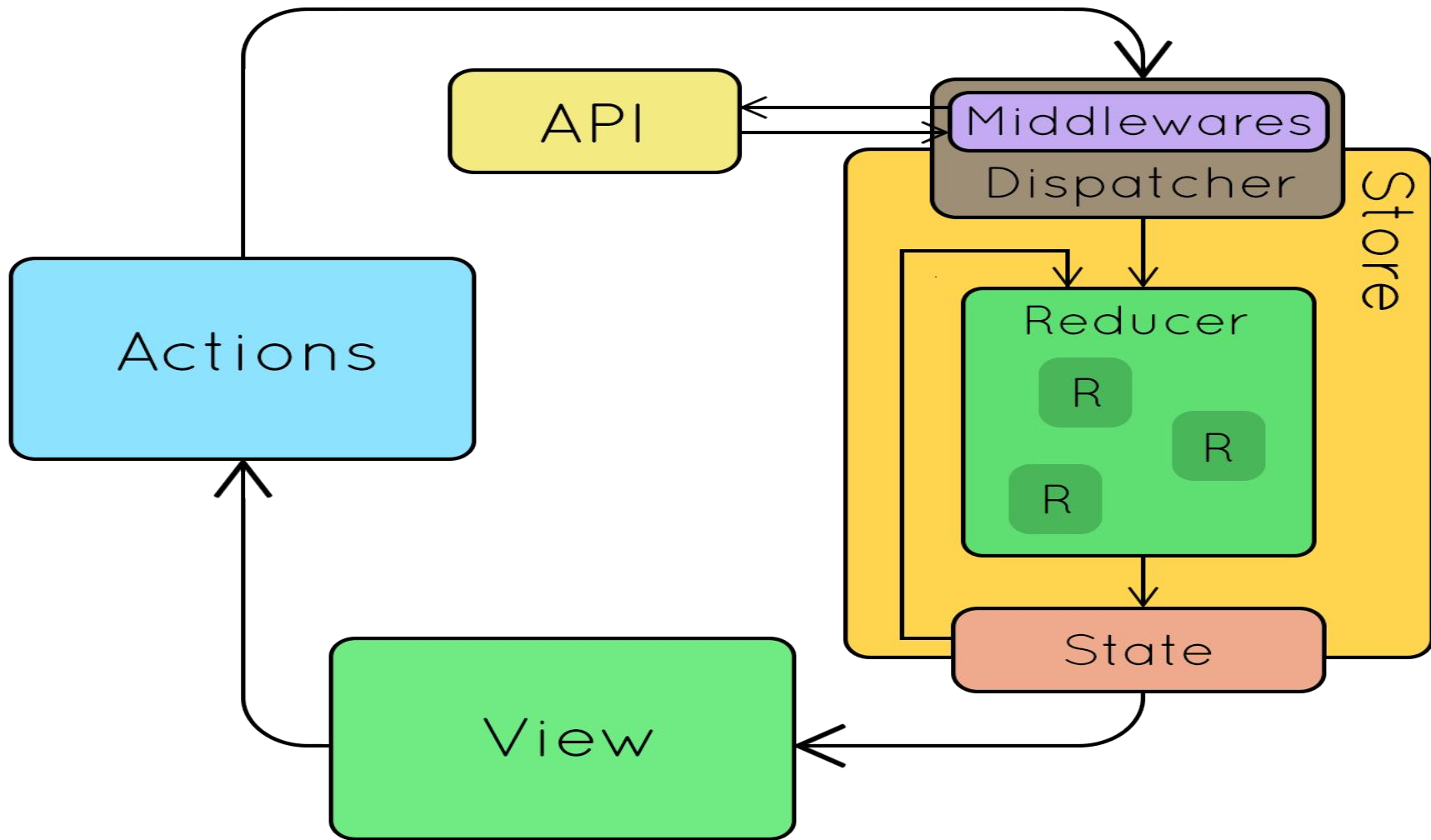
The **Reducer** is perhaps the most important and complicated to grasp part of Redux's architecture.

A **Reducer** has to be a pure function, given the same inputs it always produces the same output. It has to have no side effects and be responsible for rendering the next state.

When creating the **Store** you pass in a **Reducer**, the main **Reducer**. It is called every time an **Action** is dispatched and returns the next state for the whole app.



```
2  
3  function mainReducer(oldState, action) {  
4    return oldState;  
5  }  
6  
() 7  const AppStore = createStore(mainReducer);  
8
```





# Subscribe

The **Store** has a special function **subscribe(listenerFn)** which is used to add subscribers.

When the state changes the subscribers are invoked.