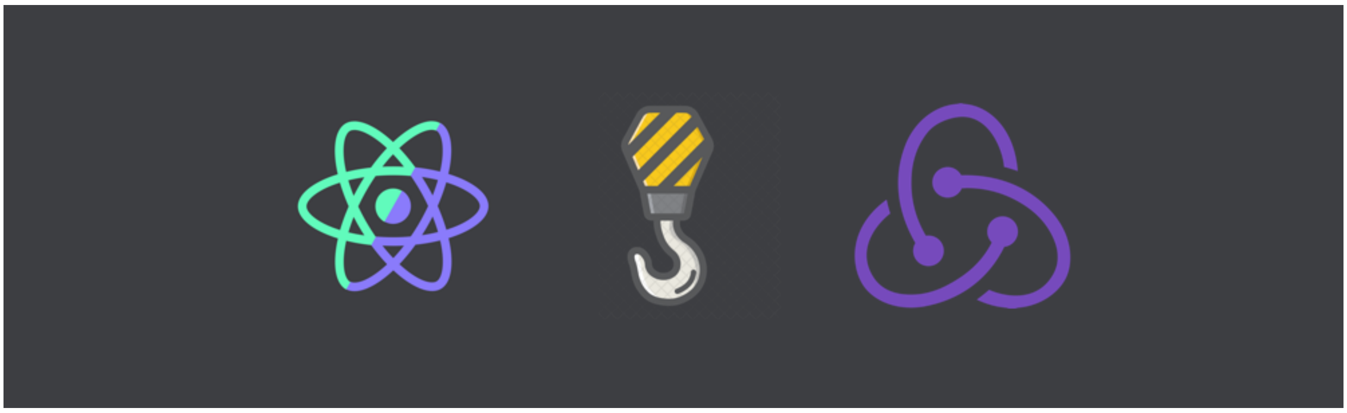
Alex Mendes

70 Followers About Follow

Introduction on React-Redux using Hooks (`useSelector` & use `Dispatch`)

Alex Mendes Nov 25, 2019 · 4 min read



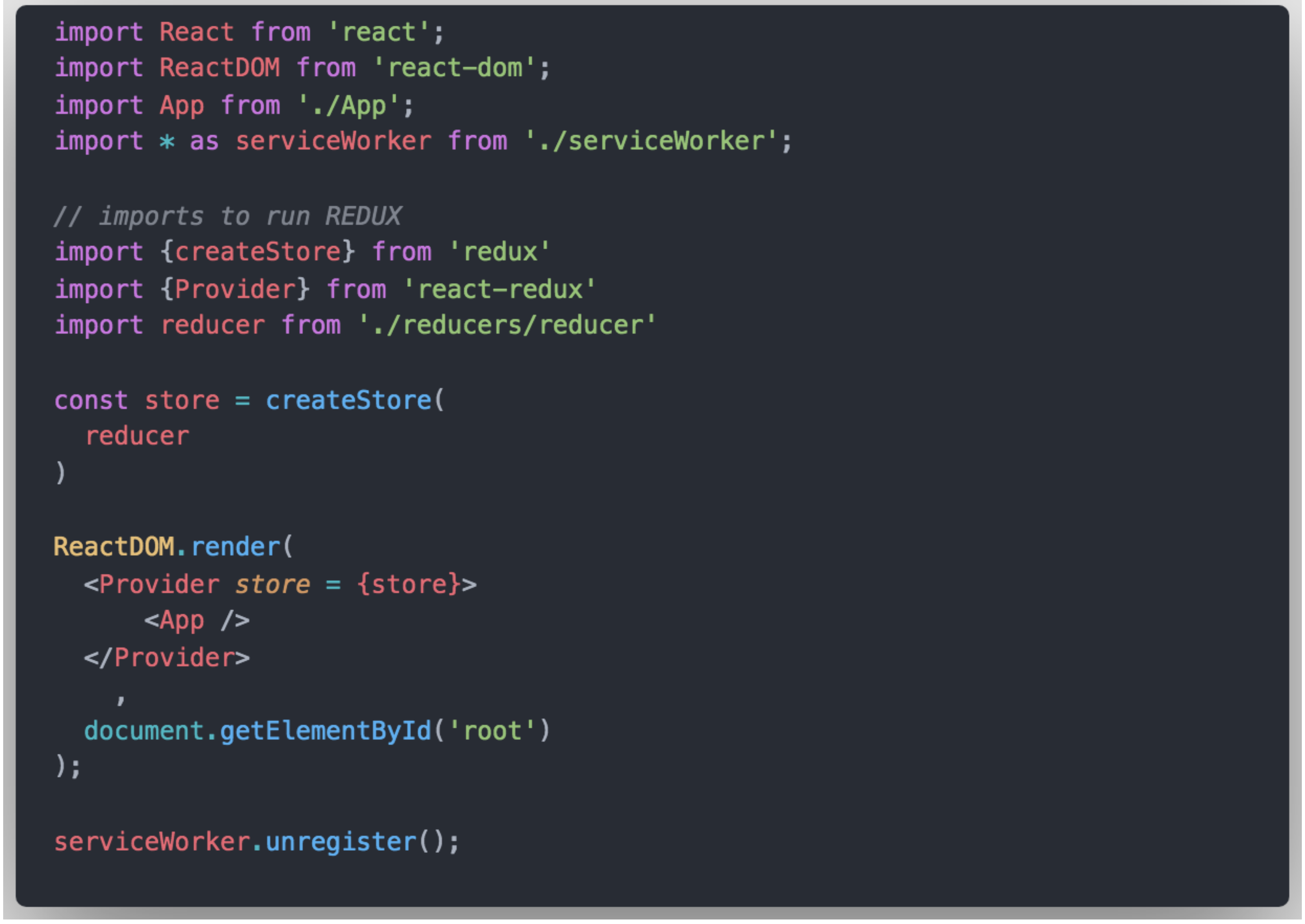
Those who have worked with Redux in React know that apart from being a great state management tool, Redux can be a little tedious at the beginning with the boilerplate/setup code that doesn't help at all to understand the data flow of the application.

And even when you get used to writing the extra amount of code, you might keep asking yourself "why so much preparation for something that is supposed to make my code run faster and cleaner?"

Well, if you are those like me that loved the simplicity and the code abstraction that hooks allow you to write JSX, I'm sure you will keep asking yourself why you haven't learned to write Redux code using `useSelector` and `useDispatch` before. On this blog, we are going to start developing an application with no Redux setup and build it all the

First, let's go to the basics:

After installing Redux and React-Redux inside your root project directory, import and write the following code into index.js



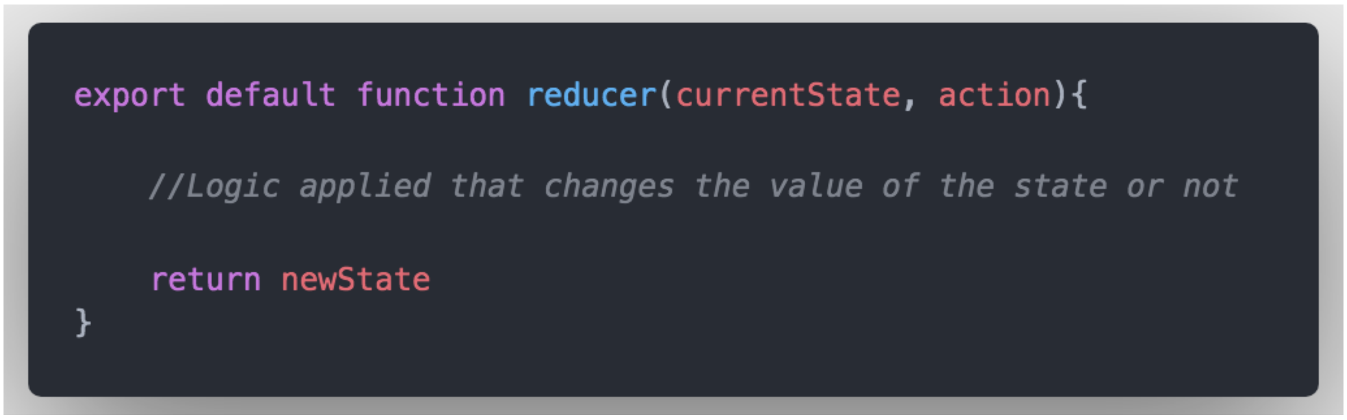
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import * as serviceWorker from './serviceWorker';

// imports to run REDUX
import {createStore} from 'redux'
import {Provider} from 'react-redux'
import reducer from './reducers/reducer'

const store = createStore(
  reducer
)

ReactDOM.render(
  <Provider store = {store}>
      <App />
  </Provider>
    ,
  document.getElementById('root')
);

serviceWorker.unregister();
```

All three imports are going to serve as the connection between Redux and our application.

`Provider` will serve as the component bridge between the Redux store and our entire app, so we are going to wrap the App component with this component provided to us from the `react-redux` library.

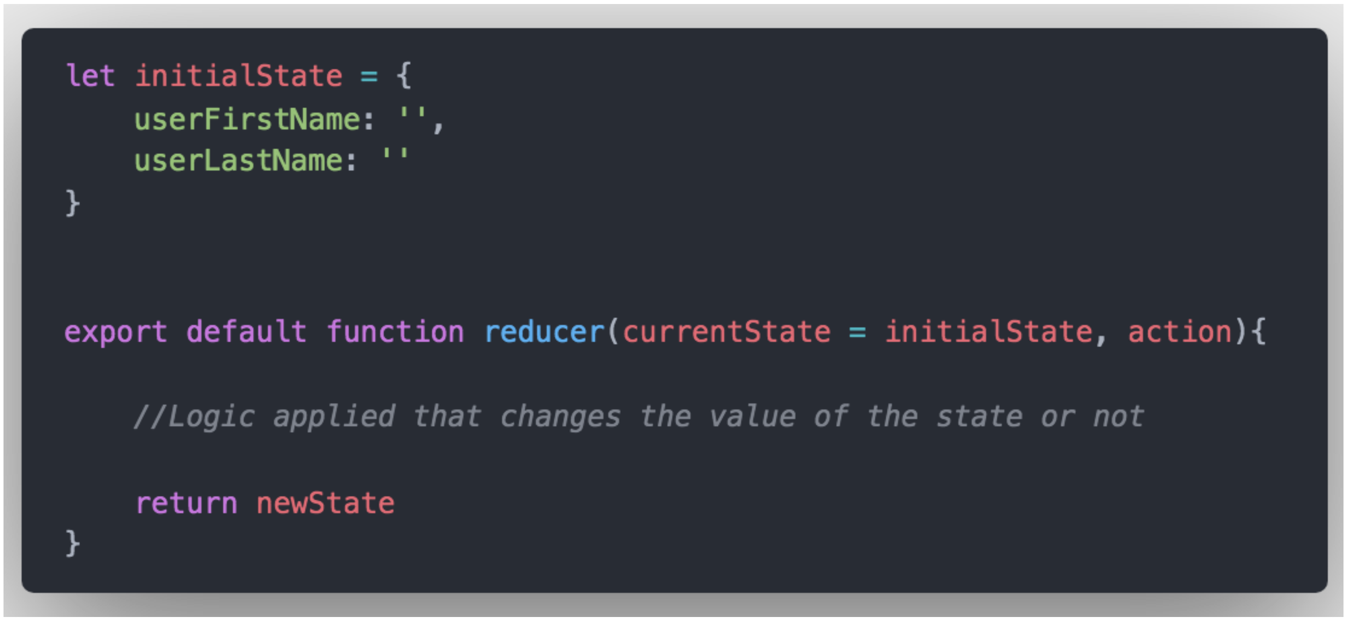
`store` by convention, and is going to be equals to the function **`createStore`** , also from **`react-redux`** library, that will accept a **`reducer`** as an argument.

We are going to be in charge of creating the reducer, which is nothing more than a function that takes two arguments: The ***`current state`*** and the **`action`** and returns the **`new state`** .



```
export default function reducer(currentState, action){

    //Logic applied that changes the value of the state or not

    return newState
}
```

The parameter **`currentState`** needs a default value equals to the initial structure of the store, and for that, we are going to create an object called **`initialState`** and assign it as the default value of **`currentState`** .



```
let initialState = {
    userFirstName: '',
    userLastName: ''
}


export default function reducer(currentState = initialState, action){

    //Logic applied that changes the value of the state or not

    return newState
}
```

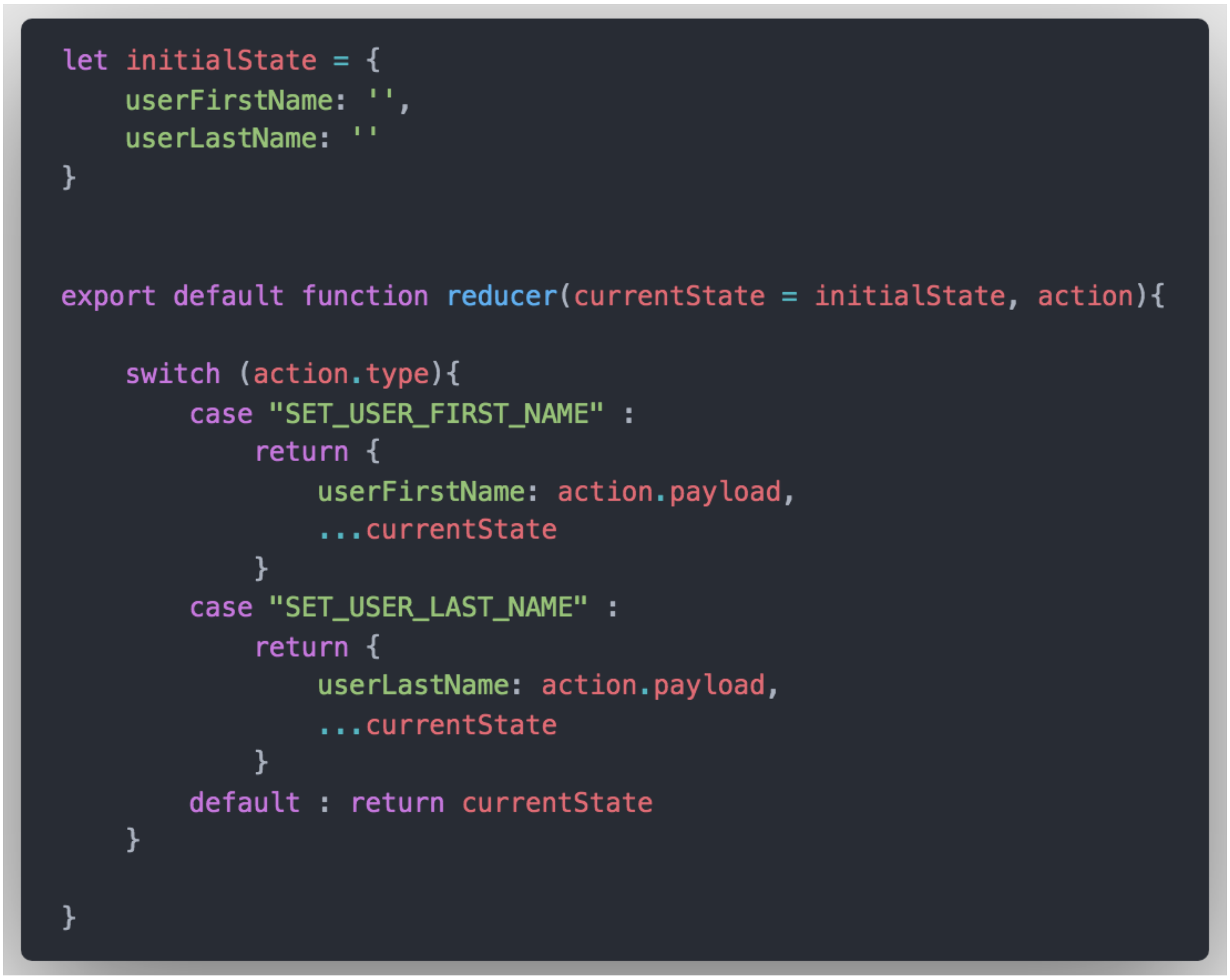
Now let's tackle the action, something that might be a little difficult to understand at first.

it to be upper case and snack case) and the `payload` or `data` that is coming from the action.

i.e: If we want to change or set the state with a new user, the action would look something like this:

```
const action = {type: "SET_USER_FIRST_NAME", payload: "Alex"}
```

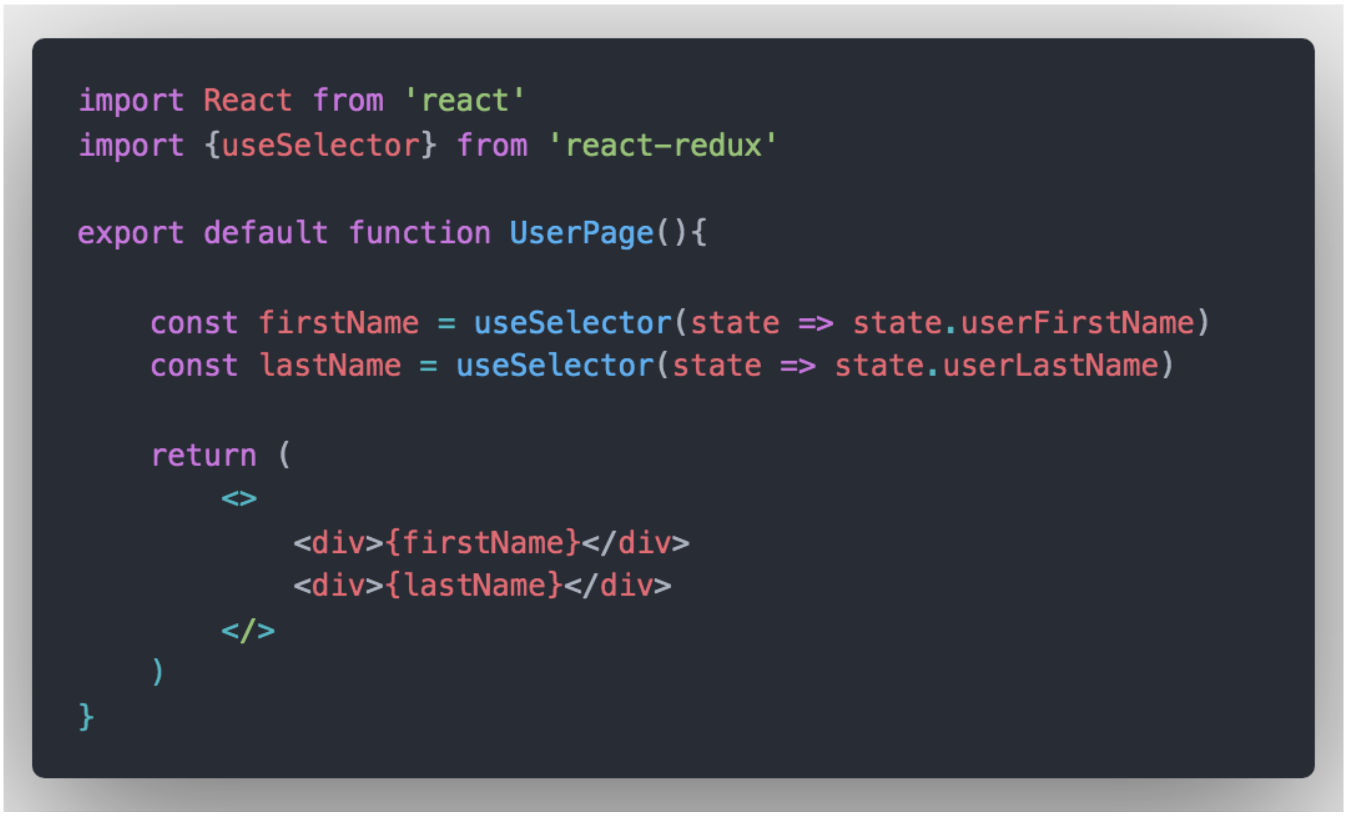
By having an idea of how the action will look like, we can start writing the logic inside our reducer, and although we could use an `if/else` statement, let's opt for a `switch` statement in order to make our code look cleaner. This way, we can insert both the current state and the action in the reducer function and the logic should look something like this.



```
let initialState = {
    userFirstName: '',
    userLastName: ''
}


export default function reducer(currentState = initialState, action){

    switch (action.type){
        case "SET_USER_FIRST_NAME" :
            return {
                userFirstName: action.payload,
                ...currentState
            }
        case "SET_USER_LAST_NAME" :
            return {
                userLastName: action.payload,
                ...currentState
            }
        default : return currentState
    }

}
```

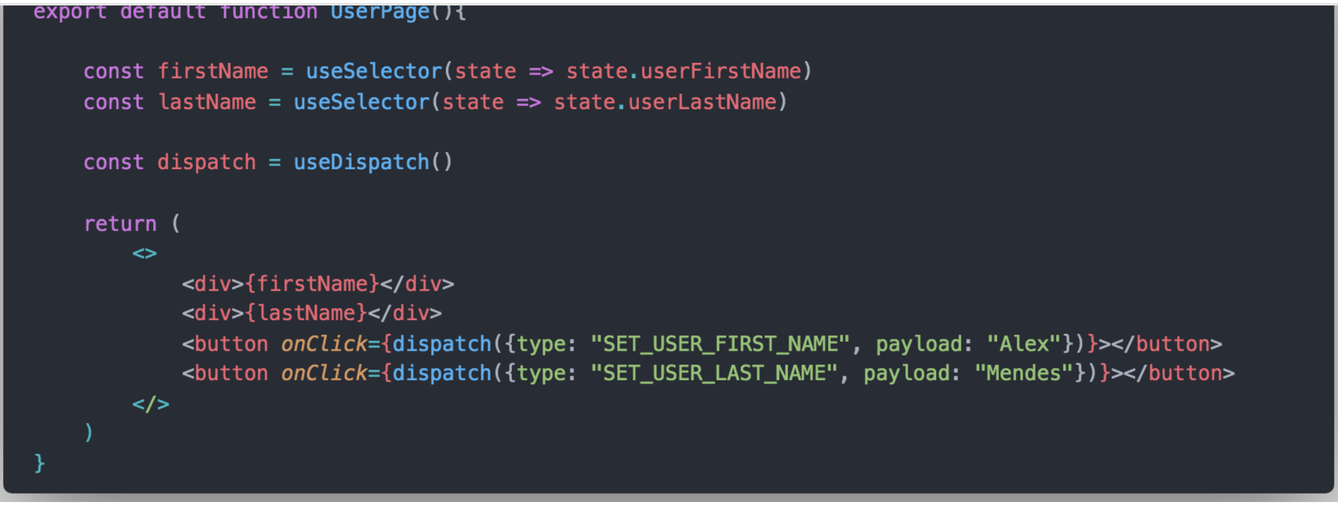
of the code abstraction of writing Redux code and make possible to read and dispatch (send) data from the store a lot easier from any functional component.

Let's start by importing `useSelector` from the `react-redux` library. `useSelector` is a function that takes the current state as an argument and returns whatever data you want from it. It's very similiar to `mapStateToProps()` and it allows you to store the return values inside a variable within the scope of you functional components instead of passing down as props.



```
import React from 'react'
import {useSelector} from 'react-redux'

export default function UserPage(){

    const firstName = useSelector(state => state.userFirstName)
    const lastName = useSelector(state => state.userLastName)

    return (
        <>
            <div>{firstName}</div>
            <div>{lastName}</div>
        </>
    )
}
```

Now let's try to dispatch an action to the store by importing `useDispatch` and implementing it to the scope of the functional component.


```
export default function UserPage(){

    const firstName = useSelector(state => state.userFirstName)
    const lastName = useSelector(state => state.userLastName)

    const dispatch = useDispatch()

    return (
        <>
            <div>{firstName}</div>
            <div>{lastName}</div>
            <button onClick={dispatch({type: "SET_USER_FIRST_NAME", payload: "Alex"})}></button>
            <button onClick={dispatch({type: "SET_USER_LAST_NAME", payload: "Mendes"})}></button>
        </>
    )
}
```

Similar to `useSelector` , `useDispatch` is a function that we import from `react-redux` library and assign it to a variable. And with this, we are able to dispatch any action to the store by simply adding an action as an argument to the new variable like the code above and that's it.

You are set up to start using this syntax to every functional component that needs to either read data from the state or dispatch an action to it without having to write the extra amount of code when dealing with class components.

Thank you for reading.

React Redux Hooks Store State