DECEMBER 10, 2018 / #REACT

How to use Redux in ReactJS with real-life examples

by Nazare Emanuel Ioan

Since I started to work with ReactJS, at Creative-Tim, I've only used it to create simple react apps, or templates if you will. I have used ReactJS only with create-react-app and have never tried to integrate it with something more.

A lot of our users have asked me or my team if the templates created by me had Redux on them. Or if they were created in such manner that they could be used with Redux. And my answer was always something like: "I haven't worked with Redux yet and I do not know what answer I should give you".

So here I am now, writing an article about Redux and how it should be used in React. Later on, in this article, I am going to add Redux on top of one of the projects that I have worked over the last one and some years.

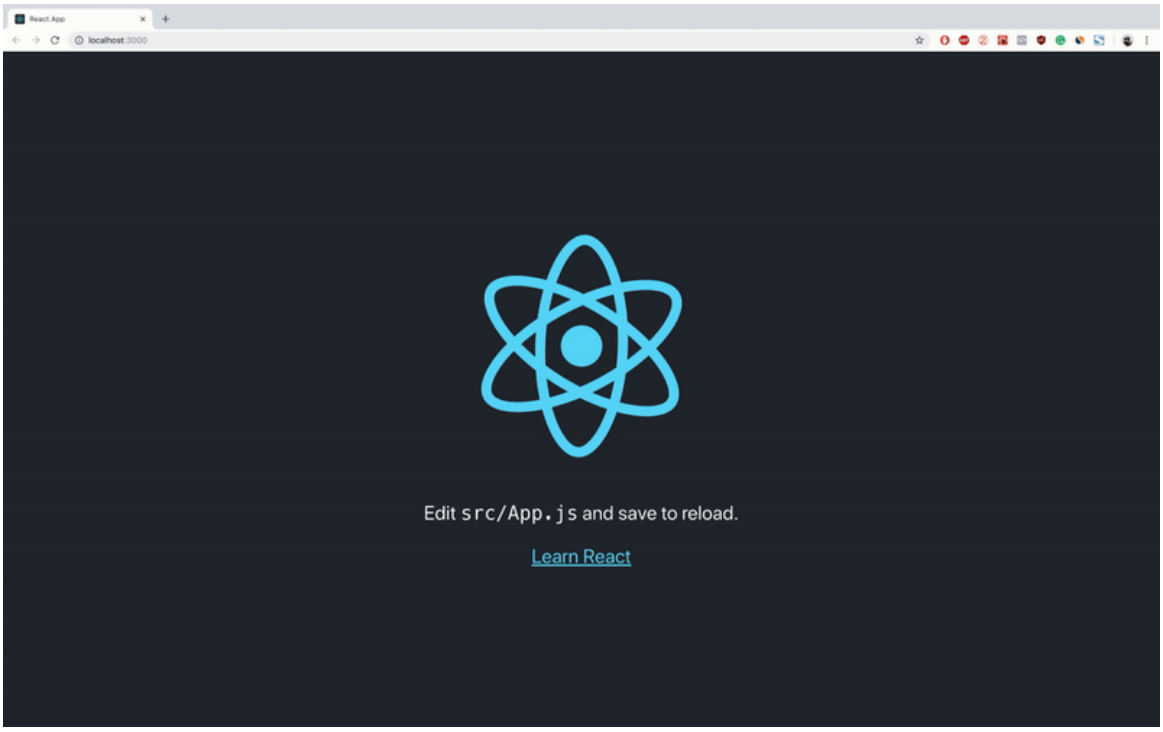
Good to know before we go ahead and wrestle with these two libraries:

- My Node.js version at the time of writing this post was 10.13.0 (LTS)
- If you want to use Webpack instead, then you can read my Webpack article, and combine what I am showing you there with what I am going to show you here.

Creating a new ReactJS based project and adding Redux to it

First things first let's create a new react app, cd into it and start it.

```
create-react-app react-redux-tutorial
cd react-redux-tutorial
npm start
```



paragraph, an anchor to the React website and the official ReactJS icon rotating.

I haven't told you guys what we are going to use Redux for, or what are we doing here. And this is because I needed the above gif image.

To make this tutorial article light weight and easy to understand, we are not going to build something very complex. We are going to use Redux to make the above React image stop or start rotating.

So this being said, let's go ahead and add the following **Redux** packages:

```
npm install --save redux react-redux
```

redux v4.0.1

- What Redux does in a very general sense, is that it creates a global state for the whole application, that can be accessed by any of your component
- It is a state management library
- You have only one state for your whole app, and not states for each of your components

react-redux v5.1.1

- The official docs state: *It lets your React components read data from a Redux store, and dispatch actions to the store to update data*

NOTE: *If you have problems with the above command, try installing the packages separately*

When working with Redux, you will need three main things:

- actions: these are objects that should have two properties, one describing the **type** of action, and one describing what should be changed in the app state.
- reducers: these are functions that implement the behavior of the actions. They change the state of the app, based on the action description and the state change description.
- store: it brings the actions and reducers together, holding and changing the state for the whole app — there is only one store.

As I've said above, we are going to stop and start the React logo spinning. This means we are going to need two actions as follows:

1 — Linux / Mac commands

```
mkdir src/actions
touch src/actions/startAction.js
touch src/actions/stopAction.js
```

```
mkdir src\actions
echo "" > src\actions\startAction.js
echo "" > src\actions\stopAction.js
```

Now let's edit the **src/actions/startAction.js** as follows:

```
export const startAction = {
  type: "rotate",
  payload: true
};
```

So, we are going to say to our reducer that the type of the action is about the *rotation* (**rotate**) of the React logo. And the state for the rotate of the React logo should be changed to **true** — we want the logo to start rotating.

Now let's edit the **src/actions/stopAction.js** as follows:

```
export const stopAction = {
  type: "rotate",
  payload: false
};
```

So, we are going to say to our reducer that the type of the action is about the *rotation* (**rotate**) of the React logo. And the state for the rotate of the React logo should be changed to **false** — we want the logo to stop rotating.

1 — Linux / Mac commands

```
mkdir src/reducers
touch src/reducers/rotateReducer.js
```

2 — Windows commands

```
mkdir src\reducers
echo "" > src\reducers\rotateReducer.js
```

And, add the following code inside of it:

```
export default (state, action) => {
  switch (action.type) {
    case "rotate":
      return {
        rotating: action.payload
      };
    default:
      return state;
  }
};
```

So, the reducer will receive both of our actions, both of which are of type **rotate**, and they both change the same state in the app — which is *state.rotating*. Based on the payload of these actions, *state.rotating* will change into **true** or **false**.

an action and we forget to add a case for that action. This way we do not delete the whole app state — we simply do nothing, and keep what we had.

The last thing that we need to do is to create our store for the whole app. Since there is only one store / one state for the whole app, we are not going to create a new folder for the store. If you want, you can create a new folder for the store and add it there, but it's not like with the actions, for example, where you can have multiple actions and it looks better to keep them inside a folder.

So this being said we are going to run this command:

1 — Linux / Mac command

```
touch src/store.js
```

2 — Windows command

```
echo "" > src\store.js
```

And also add the following code inside it:

```
import { createStore } from "redux";
import rotateReducer from "reducers/rotateReducer";
```

```
export default configureStore;
```

So, we create a function named **configureStore** in which we send a default state, and we create our store using the created reducer and the default state.

I'm not sure if you've seen my imports, they use absolute paths, so you might have some errors due to this. The fix for this is one of the two:

Either

1 — Add a .env file into your app like so:

```
echo "NODE_PATH=./src" > .env
```

Or

2 — Install cross-env globally and change the start script from the package.json file like so:

```
npm install -g cross-env
```

And inside package.json

Now that we have set up our store, our actions and our reducer we need to add a new class inside the **src/App.css** file. This class will pause the rotating animation of the logo.

So we are going to write the following inside **src/App.css**:

```
.App-logo-paused {
  animation-play-state: paused;
}
```

So your **App.css** file should look something like this:

```
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 40vmin;
}

/* new class here */
.App-logo-paused {
  animation-play-state: paused;
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
```

```
.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Now, we only need to modify our **src/App.js** file so that it listens to our store state. And when clicking on the logo, it calls one of the start or stop actions.

First things first, we need to connect our component to our redux store so we import **connect** from **react-redux**.

```
import { connect } from "react-redux";
```

After this, we'll export our App component through the connect method like this:

```
export default connect()(App);
```

```
import { startAction } from "actions/startAction";
import { stopAction } from "actions/stopAction";
```

Now we need to retrieve the state from our store and to say that we want the start and stop actions to be used for changing the state.

This will be done using the connect function, which accepts two parameters:

- **mapStateToProps:** this is used to retrieve the store state
- **mapDispatchToProps:** this is used to retrieve the actions and dispatch them to the store

You can read more about them here: react-redux connect function arguments.

So let's write inside our App.js (at the end of the file if you may):

```
const mapStateToProps = state => ({
  ...state
});

const mapDispatchToProps = dispatch => ({
  startAction: () => dispatch(startAction),
  stopAction: () => dispatch(stopAction)
});
```

```
export default connect(mapStateToProps, mapDispatchToProps)(App);
```

And right now, inside our App component, we can access the store state, the startAction and stopAction through props.

Let's change the **img** tag to:

```
<img
  src={logo}
  className={
    "App-logo" +
    (this.props.rotating ? "":" App-logo-paused")
  }
  alt="logo"
  onClick={
    this.props.rotating ?
      this.props.stopAction : this.props.startAction
  }
/>
```

So, what we are saying here is, if the store state of rotating (**this.props.rotating**) is true, then we want just the *App-logo* **className** to be set to our **img**. If that is false, then we also want the *App-logo-paused* class to be set in the **className**. This way we pause the animation.

Also, if **this.props.rotating** is **true**, then we want to send to our store for the **onClick** function and change it back to **false**, and vice-versa.

We are almost done, but we've forgot something.

For this, we go inside **src/index.js**, we import a **Provider** from **react-redux**, and the newly created store like so:

```
import { Provider } from "react-redux";

import configureStore from "store";
```

- Provider: makes the Redux store available to any nested components that have been wrapped in the connect function

After this, instead of rendering our App component directly, we render it through our Provider using the store that we've created like so:

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

Here we could have used the **configureStore** function with some other state, for example ***configureStore({ rotating: false })***.

So, your **index.js** should look like this:

```
import configureStore from "store";
// new imports stop

import './index.css';

import App from './App';
import * as serviceWorker from './serviceWorker';

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

serviceWorker.unregister();
```

Let’s go ahead and see if our redux app works:

react and **redux** in action

1 — Linux / Mac command

```
touch src/actions/rotateAction.js
```

2 — Windows command

```
echo "" > src\actions\rotateAction.js
```

And add this code:

```
const rotateAction = (payload) => {
  return {
    type: "rotate",
    payload
  }
```

We are going to send an action of type rotate, with a payload that we are going to get in the App component.

Inside the src/App.js component, we need to import our new action creator:

```
import rotateAction from "actions/rotateAction";
```

Add the new function to the mapDispatchToProps like so:

rotateAction: will receive a (payload) and will dispatch the rotateAction with the payload

Change the **onClick** function to:

```
onClick={() => this.props.rotateAction(!this.props.rotating)}
```

And finally, add our new action creator to the **mapDispatchToProps** like this:

```
rotateAction: (payload) => dispatch(rotateAction(payload))
```

We can also delete the old imports for the old actions, and delete

```
import React, { Component } from 'react';
// new lines from here
import { connect } from "react-redux";
import rotateAction from "actions/rotateAction";

//// new lines to here

import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    console.log(this.props);
    return (
      <div className="App">
        <header className="App-header">
          <img
            src={logo}
            className={
              "App-logo" +
              (this.props.rotating ? "":" App-logo-paused")
            }
            alt="logo"
            onClick={
              () => this.props.rotateAction(!this.props.rotating)
            }
          />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </div>
```

```
const mapStateToProps = state => ({
  ...state
});
const mapDispatchToProps = dispatch => ({
  rotateAction: (payload) => dispatch(rotateAction(payload))
});

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

A real-life example with Paper Dashboard React

Paper Dashboard React — Product Gif

As you will see in the above gif image, I am using the right menu to change the colors of the menu on the left. This is achieved by using component states, and by passing that state from a parent component to the two menus and some functions to change that state.

small diagram on how the app works at the moment

I thought it would be a nice example, to take this product and replace the component states with Redux.

You can get it in these 3 ways:

1. Download from creative-tim.com
2. Download from Github
3. Clone from Github:

```
git clone https://github.com/creativetimofficial/paper-dashboard-react.gi
```

Now that we have this product, let's cd into it and install again redux and react-redux:

```
npm install --save redux react-redux
```

After this, we need to create the actions. Since in the right menu we have 2 colors that set the background of the left menu, and 5 colors that change the color of the links, we need 7 actions, or 2 actions

1 — Linux / Mac commands

```
mkdir src/actions
touch src/actions/setBgAction.js
touch src/actions/setColorAction.js
```

2 — Windows commands

```
mkdir src\actions
echo "" > src\actions\setBgAction.js
echo "" > src\actions\setColorAction.js
```

After this, let's create the actions code as follows:

— src/actions/setBgAction.js

```
const setBgAction = (payload) => {
  return {
    type: "bgChange",
    payload
  }
}
export default setBgAction;
```

— src/actions/setColorAction.js

```
    payload
  }
}
export default setColorAction;
```

Now, as in the first part, we need the reducer:

1 — Linux / Mac commands

```
mkdir src/reducers
touch src/reducers/rootReducer.js
```

2 — Windows commands

```
mkdir src\reducers
echo "" > src\reducers\rootReducer.js
```

And the code for the reducer:

```
export default (state, action) => {
  switch (action.type) {
    case "bgChange":
      return {
        ...state,
        bgColor: action.payload
      };
    case "colorChange":
```

```
    default:
      return state;
  }
};
```

As you can see here, unlike our first example, we want to keep our old state and update its contents.

We also need the store:

1 — Linux / Mac command

```
touch src/store.js
```

2 — Windows command

```
echo "" > src\store.js
```

The code for it:

```
import { createStore } from "redux";
import rootReducer from "reducers/rootReducer";

function configureStore(state = { bgColor: "black", activeColor: "info" }
  return createStore(rootReducer,state);
```

Inside the src/index.js we need:

```
// new imports start
import { Provider } from "react-redux";

import configureStore from "store";
// new imports stop
```

And also, change the **render** function:

```
ReactDOM.render(
  <Provider store={configureStore()}>
    <Router history={hist}>
      <Switch>
        {indexRoutes.map((prop, key) => {
          return <Route path={prop.path} key={key} component={prop.compor
        })}
      </Switch>
    </Router>
  </Provider>,
  document.getElementById("root")
);
```

So the **index.js** file should look like this:

```
import React from "react";
import ReactDOM from "react-dom";
import { createBrowserHistory } from "history";
import { Router, Route, Switch } from "react-router-dom";
```

```
// new imports stop

import "bootstrap/dist/css/bootstrap.css";
import "assets/scss/paper-dashboard.scss";
import "assets/demo/demo.css";

import indexRoutes from "routes/index.jsx";

const hist = createBrowserHistory();

ReactDOM.render(
  <Provider store={configureStore()}>
    <Router history={hist}>
      <Switch>
        {indexRoutes.map((prop, key) => {
          return <Route path={prop.path} key={key} component={prop.compor
        })}
      </Switch>
    </Router>
  </Provider>,
  document.getElementById("root")
);
```

Now we need to make some changes inside **src/layouts/Dashboard/Dashboard.jsx**. We need to delete the state and the functions that change the state. So go ahead and **delete these bits of code:**

The constructor (between lines 16 and 22):

```
constructor(props){
  super(props);
  this.state = {
    backgroundColor: "black",
    activeColor: "info",
```

The state functions (between lines 41 and 46):

```
handleActiveClick = (color) => {
    this.setState({ activeColor: color });
  }
handleBgClick = (color) => {
  this.setState({ backgroundColor: color });
}
```

The sidebar **bgColor** and **activeColor** props (lines 53 and 54):

```
bgColor={this.state.backgroundColor}
activeColor={this.state.activeColor}
```

All of the FixedPlugin props (between lines 59–62):

```
bgColor={this.state.backgroundColor}
activeColor={this.state.activeColor}
handleActiveClick={this.handleActiveClick}
handleBgClick={this.handleBgClick}
```

So, we remain with this code inside the Dashboard layout component:

```
import React from "react";
// javascript plugin used to create scrollbars on windows
```

```
import Footer from "components/Footer/Footer.jsx";
import Sidebar from "components/Sidebar/Sidebar.jsx";
import FixedPlugin from "components/FixedPlugin/FixedPlugin.jsx";

import dashboardRoutes from "routes/dashboard.jsx";

var ps;

class Dashboard extends React.Component {
  componentDidMount() {
    if (navigator.platform.indexOf("Win") > -1) {
      ps = new PerfectScrollbar(this.refs.mainPanel);
      document.body.classList.toggle("perfect-scrollbar-on");
    }
  }
  componentWillUnmount() {
    if (navigator.platform.indexOf("Win") > -1) {
      ps.destroy();
      document.body.classList.toggle("perfect-scrollbar-on");
    }
  }
  componentDidUpdate(e) {
    if (e.history.action === "PUSH") {
      this.refs.mainPanel.scrollTop = 0;
      document.scrollingElement.scrollTop = 0;
    }
  }
  render() {
    return (
      <div className="wrapper">
        <Sidebar
          {...this.props}
          routes={dashboardRoutes}
        />
        <div className="main-panel" ref="mainPanel">
          <Header {...this.props} />
          <Switch>
            {dashboardRoutes.map((prop, key) => {
              if (prop.pro) {
                return null;
              }
              if (prop.redirect) {
```

```
              );
            })}
          </Switch>
          <Footer fluid />
        </div>
        <FixedPlugin />
      </div>
    );
  }
}

export default Dashboard;
```

We need to connect the **Sidebar** and **FixedPlugin** components to to the store.

For **src/components/Sidebar/Sidebar.jsx**:

```
import { connect } from "react-redux";
```

And change the export to:

```
const mapStateToProps = state => ({
  ...state
});

export default connect(mapStateToProps)(Sidebar);
```

```
import { connect } from "react-redux";
import setBgAction from "actions/setBgAction";
import setColorAction from "actions/setColorAction";
```

And the export should now be:

```
const mapStateToProps = state => ({
  ...state
});

const mapDispatchToProps = dispatch => ({
  setBgAction: (payload) => dispatch(setBgAction(payload)),
  setColorAction: (payload) => dispatch(setColorAction(payload))
});

export default connect(mapStateToProps, mapDispatchToProps)(FixedPlugin);
```

We are going to have these next changes:

- anywhere you find the word **handleBgClick**, you'll need to change it to **setBgAction**
- anywhere you find the word **handleActiveClick**, you'll need to change it to **setColorAction**

So, the FixedPlugin component should now look like this:

```
import React, { Component } from "react";

import { connect } from "react-redux";
```

```
class FixedPlugin extends Component {
  constructor(props) {
    super(props);
    this.state = {
      classes: "dropdown show"
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    if (this.state.classes === "dropdown") {
      this.setState({ classes: "dropdown show" });
    } else {
      this.setState({ classes: "dropdown" });
    }
  }
  render() {
    return (
      <div className="fixed-plugin">
        <div className={this.state.classes}>
          <div onClick={this.handleClick}>
            <i className="fa fa-cog fa-2x" />
          </div>
          <ul className="dropdown-menu show">
            <li className="header-title">SIDEBAR BACKGROUND</li>
            <li className="adjustments-line">
              <div className="badge-colors text-center">
                <span
                  className={
                    this.props.bgColor === "black"
                      ? "badge filter badge-dark active"
                      : "badge filter badge-dark"
                  }
                  data-color="black"
                  onClick={() => {
                    this.props.setBgAction("black");
                  }}
                />
                <span
                  className={
                    this.props.bgColor === "white"
                      ? "badge filter badge-light active"
```

```
        this.props.setBgAction("white");
      }}
    />
  </div>
</li>
<li className="header-title">SIDEBAR ACTIVE COLOR</li>
<li className="adjustments-line">
  <div className="badge-colors text-center">
    <span
      className={
        this.props.activeColor === "primary"
          ? "badge filter badge-primary active"
          : "badge filter badge-primary"
      }
      data-color="primary"
      onClick={() => {
        this.props.setColorAction("primary");
      }}
    />
    <span
      className={
        this.props.activeColor === "info"
          ? "badge filter badge-info active"
          : "badge filter badge-info"
      }
      data-color="info"
      onClick={() => {
        this.props.setColorAction("info");
      }}
    />
    <span
      className={
        this.props.activeColor === "success"
          ? "badge filter badge-success active"
          : "badge filter badge-success"
      }
      data-color="success"
      onClick={() => {
        this.props.setColorAction("success");
      }}
    />
    <span
```

```
        }
        data-color="warning"
        onClick={() => {
          this.props.setColorAction("warning");
        }}
      />
      <span
        className={
          this.props.activeColor === "danger"
            ? "badge filter badge-danger active"
            : "badge filter badge-danger"
        }
        data-color="danger"
        onClick={() => {
          this.props.setColorAction("danger");
        }}
      />
    </div>
  </li>
  <li className="button-container">
    <Button
      href="https://www.creative-tim.com/product/paper-dashboar
      color="primary"
      block
      round
    >
      Download now
    </Button>
  </li>
  <li className="button-container">
    <Button
      href="https://www.creative-tim.com/product/paper-dashboar
      color="default"
      block
      round
      outline
    >
      <i className="nc-icon nc-paper"></i> Documentation
    </Button>
  </li>
  <li className="header-title">Want more components?</li>
  <li className="button-container">
```

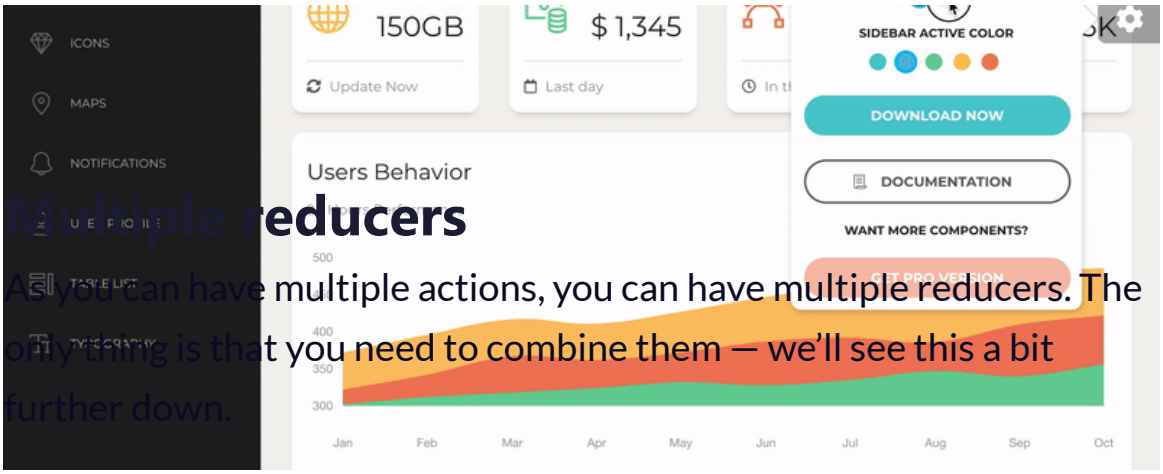
```
              round
              disabled
            >
              Get pro version
            </Button>
          </li>
        </ul>
      </div>
    </div>
  );
 }
}

const mapStateToProps = state => ({
  ...state
});

const mapDispatchToProps = dispatch => ({
  setBgAction: (payload) => dispatch(setBgAction(payload)),
  setColorAction: (payload) => dispatch(setColorAction(payload))
});

export default connect(mapStateToProps, mapDispatchToProps)(FixedPlugin);
```

And we are done, you can start the project and see how everything works fine:



Multiple reducers

As you can have multiple actions, you can have multiple reducers. The only thing is that you need to combine them — we'll see this a bit further down.

Let's go ahead and create two new reducers for our app, one for the **setBgAction** and one for the **setColorAction**:

1 — Linux / Mac commands

```
touch src/reducers/bgReducer.js
touch src/reducers/colorReducer.js
```

2 — Windows commands

```
echo "" > src\reducers\bgReducer.js
echo "" > src\reducers\colorReducer.js
```

After this, let's create the reducers' code as follows:

— src/reducers/bgReducer.js

```
        ...state,
        bgColor: action.payload
      };
    default:
      return state;
  }
};
```

— src/reducers/colorReducer.js

```
export default (state = {} , action) => {
  switch (action.type) {
    case "colorChange":
      return {
        ...state,
        activeColor: action.payload
      };
    default:
      return state;
  }
};
```

When working with combined reducers, you need to add a **default state** in each of your reducers that are going to be combined. In my case, I've chosen an empty object i.e. **state = {}**;

And now, our **rootReducer** will combine these two as follows:

— src/reducers/rootReducer.js

```
export default combineReducers({
  activeState: colorReducer,
  bgState: bgReducer
});
```

So, we say that we want the **colorReducer** to be referred by the **activeState** prop of the app state, and the **bgReducer** to be referred by the **bgState** prop of the app state.

This means that our state will no longer look like this:

```
state = {
  activeColor: "color1",
  bgColor: "color2"
}
```

It will now look like this:

```
state = {
  activeState: {
    activeColor: "color1"
  },
  bgState: {
    bgColor: "color2"
  }
}
```

Since we've changed our reducers, now we've now combined them

```
import { createStore } from "redux";
import rootReducer from "reducers/rootReducer";

// we need to pass the initial state with the new look
function configureStore(state = { bgState: {bgColor: "black"}, activeStat
  return createStore(rootReducer,state);
}
export default configureStore;
```

Since we've changed the way the state looks, we now need to change the props inside the **Sidebar** and **FixedPlugin** components to the new state object:

— **src/components/Sidebar/Sidebar.jsx**:

Change **line 36** from

```
<div className="sidebar" data-color={this.props.bgColor} data-active-colc
```

to

```
<div className="sidebar" data-color={this.props.bgState.bgColor} data-act
```

`e.bgColor` . And all the `this.props.activeColor` to `this.props.activeState.activeColor` .

So the new code should look like this:

```
import React, { Component } from "react";

import Button from "components/CustomButton/CustomButton.jsx";

import { connect } from "react-redux";
import setBgAction from "actions/setBgAction";
import setColorAction from "actions/setColorAction";

class FixedPlugin extends Component {
  constructor(props) {
    super(props);
    this.state = {
      classes: "dropdown show"
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    if (this.state.classes === "dropdown") {
      this.setState({ classes: "dropdown show" });
    } else {
      this.setState({ classes: "dropdown" });
    }
  }
  render() {
    return (
      <div className="fixed-plugin">
        <div className={this.state.classes}>
          <div onClick={this.handleClick}>
            <i className="fa fa-cog fa-2x" />
          </div>
          <ul className="dropdown-menu show">
            <li className="header-title">SIDEBAR BACKGROUND</li>
            <li className="adjustments-line">
              <div className="badge-colors text-center">
```

```
            : "badge filter badge-dark"
        }
        data-color="black"
        onClick={() => {
          this.props.setBgAction("black");
        }}
      />
      <span
        className={
          this.props.bgState.bgColor === "white"
            ? "badge filter badge-light active"
            : "badge filter badge-light"
        }
        data-color="white"
        onClick={() => {
          this.props.setBgAction("white");
        }}
      />
    </div>
  </li>
  <li className="header-title">SIDEBAR ACTIVE COLOR</li>
  <li className="adjustments-line">
    <div className="badge-colors text-center">
      <span
        className={
          this.props.activeState.activeColor === "primary"
            ? "badge filter badge-primary active"
            : "badge filter badge-primary"
        }
        data-color="primary"
        onClick={() => {
          this.props.setColorAction("primary");
        }}
      />
      <span
        className={
          this.props.activeState.activeColor === "info"
            ? "badge filter badge-info active"
            : "badge filter badge-info"
        }
        data-color="info"
        onClick={() => {
```

```
        className={
          this.props.activeState.activeColor === "success"
            ? "badge filter badge-success active"
            : "badge filter badge-success"
        }
        data-color="success"
        onClick={() => {
          this.props.setColorAction("success");
        }}
      />
      <span
        className={
          this.props.activeState.activeColor === "warning"
            ? "badge filter badge-warning active"
            : "badge filter badge-warning"
        }
        data-color="warning"
        onClick={() => {
          this.props.setColorAction("warning");
        }}
      />
      <span
        className={
          this.props.activeState.activeColor === "danger"
            ? "badge filter badge-danger active"
            : "badge filter badge-danger"
        }
        data-color="danger"
        onClick={() => {
          this.props.setColorAction("danger");
        }}
      />
    </div>
  </li>
  <li className="button-container">
    <Button
      href="https://www.creative-tim.com/product/paper-dashboar
      color="primary"
      block
      round
    >
      Download now
```

```
                href="https://www.creative-tim.com/product/paper-dashboar
                color="default"
                block
                round
                outline
              >
                <i className="nc-icon nc-paper"></i> Documentation
              </Button>
            </li>
            <li className="header-title">Want more components?</li>
            <li className="button-container">
              <Button
                href="https://www.creative-tim.com/product/paper-dashboar
                color="danger"
                block
                round
                disabled
              >
                Get pro version
              </Button>
            </li>
          </ul>
        </div>
      </div>
    );
  }
}

const mapStateToProps = state => ({
  ...state
});

const mapDispatchToProps = dispatch => ({
  setBgAction: (payload) => dispatch(setBgAction(payload)),
  setColorAction: (payload) => dispatch(setColorAction(payload))
});

export default connect(mapStateToProps, mapDispatchToProps)(FixedPlugin);
```

Thanks for reading!

If you've enjoyed reading this tutorial please share it. I am very keen on hearing your thoughts about it. Just give this thread a comment and I'll be more than happy to reply.

Special thanks should also go to Esther Falayi for his tutorial which has given me some much needed understanding on **Redux**.

Useful links:

- Get the code for this tutorial from Github
- Read more about ReactJS on their official website
- Read more about Redux here
- Read more about React-Redux
- Check out our platform to see what we are doing and who we are
- Get Paper Dashboard React from www.creative-tim.com or from Github
- Read more about Reactstrap, the core of Paper Dashboard React

Find me on:

- Email: manu@creative-tim.com
- Facebook: https://www.facebook.com/NazareEmanuel
- Instagram: https://www.instagram.com/manu.nazare/