We are going to take an example of a simple counter application using **React.js and Redux.**First, we need to setup a react environment.

**Step 1:  Configure the project.**

Create one project folder and in that create one file called **package.json.**Copy the following code into it.

{

"name": "reduxapp",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"start": "webpack-dev-server"

},

"author": "Some Name",

"license": "ISC",

"devDependencies": {

"babel-core": "^6.24.0",

"babel-loader": "^6.4.1",

"babel-preset-es2015": "^6.24.0",

"babel-preset-react": "^6.23.0",

"babel-preset-stage-3": "^6.22.0",

"webpack": "^2.3.2",

"webpack-dev-server": "^2.4.2"

},

"dependencies": {

"react": "^15.4.2",

"react-dom": "^15.4.2",

"react-redux": "^5.0.6",

"redux": "^3.7.2"

}

}

Switch to a terminal and type the following command.

npm install

Next step will be to create the**webpack.config.js**file in the root folder.

// webpack.config.js

module.exports = {

entry: './src/main.js',

output: {

filename: 'bundle.js'

},

module: {

loaders: [

{

loader: 'babel-loader',

test: /\.js$/,

exclude: /node\_modules/

}

]

},

devServer: {

port: 3000

}

};

Also, create one file called in root called **.babelrc**

{

"presets": ["es2015", "react", "stage-3"]

}

Make one file in the root called **index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Redux Tutorial 2017</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

Go to a terminal and type this command.

npm start

Server will start at this URL: [http://localhost:3000](http://localhost:3000/)

## ****Step 2: Create a main.js file inside src folder.****

*// main.js*

import React from 'react';

import { render } from 'react-dom';

import { Provider } from 'react-redux';

import App from './components/App';

render(

<Provider>

<App />

</Provider>,

document.getElementById('root')

)

I have included the react particular dependency, but also I have included react-redux. It provides us a store through our entire application. So our App element is wrapped around **Provider.**

## ****Step 3: Create components directory inside src.****

Make one directory and inside make our most important component file called **App.js**

// App.js

import React from 'react';

const App = () => {

<div className="container">

App Component

</div>

}

export default App;

**App.js** the component where our other components are included. We are creating the simple counter application. However, we need to define first how many components are required to complete the application.

There are mainly two types of components when you are dealing with React and Redux.

1. **Smart Component**
2. **Dumb Component**

## ****Smart Component****

The smart component is the kind of component, which directly interacts with the state of our application. It has access to the store and it can either dispatch the actions or get the current state of our application. It is the smart components because when the store is changed, by default, it subscribes the new state and changes the view according to it. In our application, there are **three**smart components.

1. **Counter.js**
2. **AddCounter.js**
3. **RemoveCounter.js**

All these smart components are put in the **containers folder,**which I will create later in this article.  
**Container components**only contain that components that are smart.

## ****Dumb Component****

**App.js**is the **Dumb**component, it includes the child component but, it does not interact the store. So we put that component inside **components folder.**

## ****Step 4: Create container dir. Inside src folder.****

We need to create three **container components**inside this directory as I have mentioned before.

1. We are creating **AddCounter.js** component

*// AddComponent.js*

import React, { Component } from 'react';

import { connect } from 'react-redux';

import { addCounter } from '../actions';

import { bindActionCreators } from 'redux';

class AddCounter extends Component {

constructor(props) {

super(props);

}

render() {

return (

<div className="container">

<form>

<div className="field is-grouped">

<div className="control">

<button className="button is-primary"

onClick={(e) => {e.preventDefault();this.props.dispatch(addCounter())}}>

Add

</button>

</div>

</div>

</form>

</div>

)

}

}

function mapDispatchToProps(dispatch) {

return { actions: bindActionCreators(addCounter, dispatch) }

}

export default connect(mapDispatchToProps)(AddCounter);

Here, I need to explain lots of things so let us get started.

If **AddCounter**component is container component, then it connects to the **store** remember, that is why it is a smart component.

So the last line of the code is that we are connecting our component to the **Redux store.**

When the user clicks the button according to Redux principle, it dispatches an action, so we need to pass dispatch function as a property to this component.

Now, it dispatches the action, and in our scenario, it named as **addCounter().**So addCounter return an action.

## ****Create an Action****

Make one folder inside src called **actions**and in that create one file called **index.js**

*// index.js*

import \* as actionType from './ActionType';

export const addCounter = () => ({

type: actionType.ADD\_COUNTER,

payload: 1

});

So as I mentioned, It returns an object describes our **actions**.

Here, I have also included one more file called**ActionType.js**

This file is needed because all the action names are constant and if we create one file, which only exports the naming constant then it is easy for us to define any action without any typos because actions names are on the strings.

## ****Make ActionType.js****

This file is inside **actions directory.**

*// ActionType.js*

export const ADD\_COUNTER = 'ADD\_COUNTER';

export const REMOVE\_COUNTER = 'REMOVE\_COUNTER';

Now, coming to the point, we are at **AddCounter.js** file, right?

*// AddCounter.js*

import React, { Component } from 'react';

import { connect } from 'react-redux';

import { addCounter } from '../actions';

import { bindActionCreators } from 'redux';

class AddCounter extends Component {

constructor(props) {

super(props);

}

render() {

return (

<div className="container">

<form>

<div className="field is-grouped">

<div className="control">

<button className="button is-primary"

onClick={(e) => {e.preventDefault();this.props.dispatch(addCounter())}}>

Add

</button>

</div>

</div>

</form>

</div>

)

}

}

function mapDispatchToProps(dispatch) {

return { actions: bindActionCreators(addCounter, dispatch) }

}

export default connect(mapDispatchToProps)(AddCounter);

**Function mapDispatchToProps** is needed because we need to pass dispatch as a property to our component and also we need to bind the actions with this component.

## ****Step 5: Create reducer directory inside src and make one file****

Make one file called **counterReducer.js**inside **reducer directory.**

// counterReducer.js

import \* as actionType from '../actions/ActionType';

const counterReducer = (state = 0, action) => {

let newState;

switch (action.type) {

case actionType.ADD\_COUNTER:

return newState = state + action.payload;

case actionType.REMOVE\_COUNTER:

return newState = state - action.payload;

default:

return state

}

}

export default counterReducer;

Please note that I have described both cases in here

1. **Increment (for add +1 in counter state)**
2. **Decrement (**decrease **one from counter state)**

**If you carefully see the above cases then, I have not modified the state directly. See, I have defined a new variable and then assign new state in that variable and return that variable.**

So, it is a pure function, which is not mutating any store state, just take the old state value and add that old value plus new value and assign it to the variable and return the new state of our application. This is the central principle of **Redux** after all.

Make one file inside **reducer directory called index.js**

*// index.js*

import { combineReducers } from 'redux';

import counterReducer from './counterReducer';

const counterApp = combineReducers({

counterReducer

})

export default counterApp

Now, this is our single and final store. This is the store we include in our **main.js**file.

**combineReducer function**as the name suggests to combine all reducers in one **store** and return as a **global application state object.**

Now, our **main.js** file will look like this. I have passed the store to our entire application. Also Included the reducers.

*// main.js*

import React from 'react';

import { render } from 'react-dom';

import { createStore } from 'redux';

import { Provider } from 'react-redux';

import App from './components/App';

import reducer from './reducers';

const store = createStore(reducer);

render(

<Provider store={store}>

<App />

</Provider>,

document.getElementById('root')

)

## ****Step 6: Create same smart component for RemoveCounter.js****

Make **RemoveCounter.js** inside **container directory.**

*// RemoveCounter.js*

import React, { Component } from 'react';

import { connect } from 'react-redux';

import { removeCounter } from '../actions';

import { bindActionCreators } from 'redux';

class RemoveCounter extends Component {

constructor(props) {

super(props);

}

render() {

return (

<div className="container">

<form>

<div className="field is-grouped">

<div className="control">

<button className="button is-primary"

onClick={(e) => {e.preventDefault();this.props.dispatch(removeCounter())}}>

Remove

</button>

</div>

</div>

</form>

</div>

)

}

}

function mapDispatchToProps(dispatch) {

return { actions: bindActionCreators(removeCounter, dispatch) }

}

export default connect(mapDispatchToProps)(RemoveCounter);

Now, switch to **src  >>  actions  >>  index.js** and add new action for **removeCounter**. So our final file looks like this.

*// index.js*

import \* as actionType from './ActionType';

export const addCounter = () => ({

type: actionType.ADD\_COUNTER,

payload: 1

});

export const removeCounter = () => ({

type: actionType.REMOVE\_COUNTER,

payload: 1

});

We have already defined the decrement **reducer**.

// counterReducer.js

import \* as actionType from '../actions/ActionType';

const counterReducer = (state = 0, action) => {

let newState;

switch (action.type) {

case actionType.ADD\_COUNTER:

return newState = state + action.payload;

case actionType.REMOVE\_COUNTER:

return newState = state - action.payload;

default:

return state

}

}

export default counterReducer;

## ****Step 7: Now final smart component remains is Counter.js****

*// Counter.js*

import React, { Component } from 'react';

import { connect } from 'react-redux';

class Counter extends Component {

constructor(props){

super(props);

}

render(){

return (

<div className="cotainer">

<div className="notification">

<h1>

{this.props.count}

</h1>

</div>

</div>

)

}

}

function mapStateToProps(state){

return {

count: state.counterReducer,

};

}

export default connect(mapStateToProps)(Counter);

This is the smart component, so we need to connect it with **Redux store. So**we have connected it with the store and fetched the latest state from the store and display it.

**mapStateToProps**maps the state to the props of current component and shows the data as a property of the component.

## ****Step 8: Include all the components into the App.js file.****

*// App.js*

import React from 'react';

import Counter from '../containers/Counter';

import AddCounter from '../containers/AddCounter';

import RemoveCounter from '../containers/RemoveCounter';

const App = () => {

return (

<div className="container">

<Counter></Counter><br />

<div className="columns">

<div className="column is-11">

<AddCounter></AddCounter>

</div>

<div className="column auto">

<RemoveCounter></RemoveCounter>

</div>

</div>

</div>

)

}

export default App;

Finally, all the code is over, and if you have not started the development server, then please start via the following command.

npm start

If you switch to this URL: [http://localhost:3000](http://localhost:3000/)