**Redux:**

**package.json**

{

"name": "redux\_learning\_demo",

"version": "0.1.0",

"private": true,

"dependencies": {

"bootstrap": "^4.2.1",

"react": "^16.7.0",

"react-dom": "^16.7.0",

"react-redux": "^6.0.0",

"react-scripts": "^2.1.2",

"redux": "^4.0.1"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": "react-app"

},

"browserslist": [

">0.2%",

"not dead",

"not ie <= 11",

"not op\_mini all"

]

}

Redux is not a part of React. It is a separate javascript library that you can use with javascript or any other javascript framework.

* It makes state management much easier.

Commands to create a redux project:

1. npm init
2. npm install redux –save

const redux = require('redux')

const createStore = redux.createStore;

// Reducer

const rootReducer = (state, action) => {

return state;

};

// store

const store = createStore(rootReducer);

console.log(store.getState());

Here, we are using nodejs syntax.

We have created store [const store = createStore(rootReducer);]

We have to pass a reducer to a state through which store knows what is the state.

rootReducer is a reducer which is just returning a state which doesnot contain anything yet as we have not initialized anything yet to state. So, we will be getting *undefined* as an output when will will print state in console.log(state.getState()).

const redux = require('redux')

const createStore = redux.createStore;

const initialState = {

counter: 0

}

// Reducer

const rootReducer = (state = initialState, action) => {

return state;

};

// store

const store = createStore(rootReducer);

console.log(store.getState());

Now, we have introduced a constant naming initialState which is an Object having counter as one of the variable.

Now, we are assigning this initialState to state while defining rootReducer as:

// Reducer

const rootReducer = (state = initialState, action) => {

return state;

};

Now store contains this reducer which is returning a state which now has been initialized.

So, we will get in console.log(state.getState())

* { counter: 0 }

As far as now we have created the store and passed a reducer to it.

Now, we want want dispatch an action from the store. We can do this like

// Dispatch actions

store.dispatch({

type: "INCREMENT\_COUNTER"

});

store.dispatch({

type: "ADD\_COUNTER",

value: 10

});

These are two actions that has been dispatched by the store.

Now, these actions will be going to rootReducer that we created.

\*REMEMBER\*: We created rootReducer by using two arguments:

1. State
2. Action

As soon as we dispatch the actions it will go to all the reducers(in our case only one reducer is there). Now, we can add our own logic in our reducer like this:

const redux = require('redux')

const createStore = redux.createStore;

const initialState = {

counter: 0

}

// Reducer

const rootReducer = (state = initialState, action) => {

if (action.type === "INCREMENT\_COUNTER") {

return {

...state,

counter: state.counter + 1

}

}

if (action.type === "ADD\_COUNTER") {

return {

...state,

counter: state.counter + action.value

}

}

return state;

};

// store

const store = createStore(rootReducer);

console.log("Before dispatching an action: " + JSON.stringify(store.getState()));

// Dispatch actions

store.dispatch({type: "INCREMENT\_COUNTER"});

store.dispatch({type: "ADD\_COUNTER", value: 10});

console.log("After dispatching an action: " + JSON.stringify(store.getState()));

Output:

Before dispatching an action: {"counter":0}

After dispatching an action: {"counter":11}

As, store is dispatching actions two times so, two times the state will get manipulated in the reducer. First time it will increment by 1 according to our login and second time it will get incremented by 10. Hence : {"counter":11}

After every action gets dispatched, our state may or may not changes. Here we are using console.log(state.getState()) many times to know about the state and this becomes very tedious job because we have to guess about the changes of the state.

Hence, we use the concept of subscribe. We write the subscribe method after creating the store. If any dispatch method is used then this subscribe will automatically run to inform that Hey, your state may have changed!

store.subscribe(() => {

console.log("['SUBSCRIBE'] " + JSON.stringify(store.getState()))

});

const redux = require('redux')

const createStore = redux.createStore;

const initialState = {

counter: 0

}

// Reducer

const rootReducer = (state = initialState, action) => {

if (action.type === "INCREMENT\_COUNTER") {

return {

...state,

counter: state.counter + 1

}

}

if (action.type === "ADD\_COUNTER") {

return {

...state,

counter: state.counter + action.value

}

}

return state;

};

// store

const store = createStore(rootReducer);

console.log("Before dispatching an action: " + JSON.stringify(store.getState()));

store.subscribe(() => {

console.log("['SUBSCRIBE'] " + JSON.stringify(store.getState()))

});

// Dispatch actions

store.dispatch({

type: "INCREMENT\_COUNTER"

});

store.dispatch({

type: "ADD\_COUNTER",

value: 10

});

console.log("After dispatching an action: " + JSON.stringify(store.getState()));

**Connecting Redux with React**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import { createStore } from 'redux';

import reducer from '../src/store/reducer/reducer';

import { Provider } from 'react-redux';

const store = createStore(reducer);

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

So we have learned that in order to use the reducers we have to use store and we have to pass reducer to it.

That’s how we import createStore using ES6 syntax:

import { createStore } from 'redux';

we have to have reducer in order to pass it into store.

import reducer from '../src/store/reducer/reducer';

We have to use higher order component so as to enable the store to other components

import { Provider } from 'react-redux';

So, to connect the Component we have to export the component with the use of connet of ‘react-redux’. connect is function which returns a function which takes Component as a parameter. So, connect itself is not a higher order Component but its returns a higher order component.

reduxtest.js

const { createStore } = require('redux');

// creating state

const initialState = {

age: 21

};

/\*\*

\* creating reducer

\* It takes two parameters

\* 1. state

\* 2. action

\*/

const myReducer = (state = initialState, action) => {

// copy of state as we do not mutate state directly

const newState = {...state};

if (action.type === 'ADD') {

newState.age += 1;

}// end of if

// it’s almost like a setState in react

return newState; }

// creating the store

const store = createStore(myReducer);

// creating an action which is type ADD

store.dispatch({ type: 'ADD' });

console.log(store.getState());

**C:\NotBackedUp\myworkspace\_VisualStudio\youtube\_redux>** node reduxtest.js

{age: 22 }

**#Adding new action:**

const { createStore } = require('redux');

// creating state

const initialState = {

age: 21

};

/\*\*

\* creating reducer

\* It takes two parameters

\* 1. state

\* 2. action

\*/

const myReducer = (state = initialState, action) => {

// copy of state as we do not mutate state directly

const newState = { ...state };

if (action.type === 'ADD') {

newState.age += 1;

}// end of if

if (action.type === 'SUB') {

newState.age -= 1;

}

// its almost like a setState in react

return newState;

}

// creating the store

const store = createStore(myReducer);

console.log('Initial state: ' + JSON.stringify(store.getState()));

// creating an action which is type ADD

store.dispatch({ type: 'ADD' });

console.log('state after ADD ' + JSON.stringify(store.getState()));

// creating an action which is type SUB

store.dispatch({ type: 'SUB' });

console.log('state after SUB: ' + JSON.stringify(store.getState()));

**C:\NotBackedUp\myworkspace\_VisualStudio\youtube\_redux>** node reduxtest.js

Initial state: {"age":21}

state after ADD {"age":22}

state after SUB: {"age":21}

**Subscribe**

const { createStore } = require('redux');

// creating state

const initialState = {

age: 21

};

/\*\*

\* creating reducer

\* It takes two parameters

\* 1. state

\* 2. action

\*/

const myReducer = (state = initialState, action) => {

// copy of state as we do not mutate state directly

const newState = { ...state };

if (action.type === 'ADD') {

newState.age += 1;

}// end of if

if (action.type === 'SUB') {

newState.age -= 1;

}

// its almost like a setState in react

return newState;

}

// creating the store

const store = createStore(myReducer);

store.subscribe(() => {

console.log('state changed ' + JSON.stringify(store.getState()))

})

console.log('Initial state ' + JSON.stringify(store.getState()))

// creating an action which is type ADD

store.dispatch({ type: 'ADD' });

// creating an action which is type SUB

store.dispatch({ type: 'SUB' });

**C:\NotBackedUp\myworkspace\_VisualStudio\youtube\_redux>** node reduxtest.js

Initial state: {"age":21}

state after ADD {"age":22}

state after SUB: {"age":21}

**passing value with actions:**

const myReducer = (state = initialState, action) => {

// copy of state as we do not mutate state directly

const newState = { ...state };

if (action.type === 'ADD') {

newState.age += action.val;

}// end of if

if (action.type === 'SUB') {

newState.age -= action.val;

}

// its almost like a setState in react

return newState;

}

// creating the store

const store = createStore(myReducer);

store.subscribe(() => {

console.log('state changed ' + JSON.stringify(store.getState()))

})

console.log('Initial state ' + JSON.stringify(store.getState()))

// creating an action which is type ADD

store.dispatch({ type: 'ADD', val:10 });

// creating an action which is type SUB

store.dispatch({ type: 'SUB', val:12});

**C:\NotBackedUp\myworkspace\_VisualStudio\youtube\_redux>** node reduxtest.js

Initial state: {"age":21}

state after ADD {"age":22}

state after SUB: {"age":21}

**Redux with React**

**Commands:**

npm install redux --save

npm install react-redux –save

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

/\*\*

\* Provider allows us to inject the global store

\*/

import { Provider } from 'react-redux';

import { createStore } from 'redux';

import reducer from './reducers/reducer'

const store = createStore(reducer);

/\*\*

\* Provider allows us to inject the global store

\* Store is now available to the entire application

\*/

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

**reducer,js**

const initialState = {

age: 21

};

const reducer = (state = initialState, action) => {

const newState = { ...state };

if (action.type === 'AGE\_UP') {

newState.age++;

}

if (action.type === 'AGE\_DOWN') {

newState.age--;

}

return newState;

};

export default reducer;

import React, { Component } from 'react';

import './App.css';

/\*\*

\* connect is a higher order component

\* that is used to connect store and the App Component

\* to the redux component

\*/

import { connect } from 'react-redux';

class App extends Component {

render() {

return (

<div className="App">

<div>Age: <span>{this.props.age}</span> </div>

<button onClick={this.props.onAgeUp}>Age Up</button>

<button onClick={this.props.onAgeDown}>Age down</button>

</div>

);

}

}

const mapStateToProps = (state) => {

return {

age: state.age

}

};

/\*\*

\* @param {\*} dispatch

\* oneAgeUp is an annonymous function

\* onAgeDown is an annonymous function

\*/

const mapDispatchToProps = (dispatch) => {

return {

onAgeUp: () => dispatch({ type: 'AGE\_UP' }),

onAgeDown: () => dispatch({ type: 'AGE\_DOWN' })

}

};

/\*\*

\* Here, we are connect our

\* mapStateToProps, mapDispatchToProps

\* to our store so that it is available in App Component

\*/

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