React Demo app – Invoices Module

Redux is a state management library that gives you access to the state anywhere in your components without the need to pass props. So it can be used with any front-end libraries like Angular and React but it works best with React. ‘react-redux’ is the official library that connects the two.

How to install redux in your applicatons:

npm install --save redux react-redux

* Redux is a state management library that gives you access to the state anywhere in your components without the need to pass props. So it can be used with any front-end libraries like Angular and React but it works best with React. ‘react-redux’ is the official library that connects the two.
* First thing to understand about Redux is something called the store. It’s where the entire state of your application will live. This is the first main benefit of using Redux. Instead of having to manage the state in different components we have to only manage it in one single place called the store. The store is an object which has some methods in it that allows us to get the current state of our application, subscribe to changes or update the existing state of our application. This is great because now we don’t have to pass down data from the parent component to deeply nested child components through props. So anytime a component needs data it can ask the store and the store will provide it with the data.

import { createStore } from 'redux';

const store = createStore();

* The createStore method will allow us to create the store but we are not done yet. This method needs a special argument and this argument goes by a special name called the ‘reducer’. Let’s create a separate folder called reducers. So under crud-redux/src create a folder called ‘reducers’.

const postReducer = (state = [], action) => {

}

export default postReducer;

* We will fill in the contents of that function a bit later. Now let’s understand another important concept in Redux called actions. Actions are nothing but plain Javascript objects with a type property. This type property describes the event that is taking place in the application. This event can be anything from incrementing a counter to adding items in an array. These actions help us track the different events that are happening in our application. The structure of an action is as follows-

{

type: 'EVENT\_NAME'

}

An action can have any number of properties but it must have a type property. So an action can include data like so

{

type:'ADD\_ITEM',

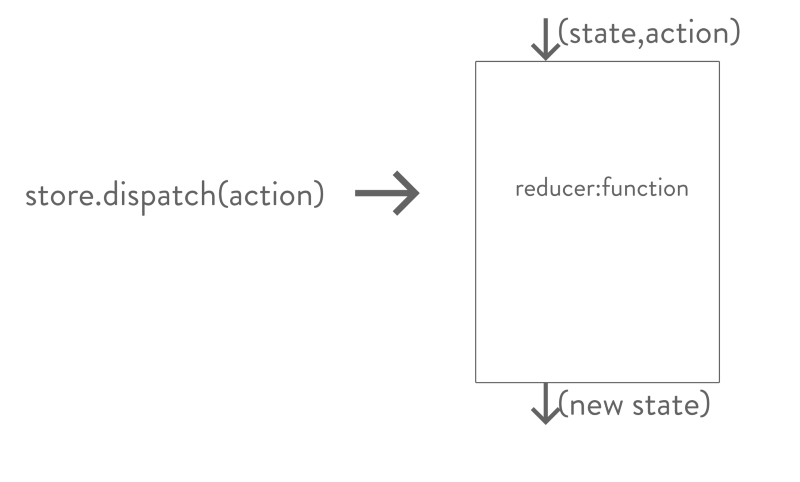
name: 'Redux'

}

In this example the event name is ‘ADD\_ITEM’ and the data is the name property with a value of ‘Redux’. Now another important term that is used alongside actions is called dispatch. When we say ‘dispatch an action’ we simply mean call the dispatch method which is inside the store object with an action.

What the heck is the reducer?

Well it’s nothing but a function that takes the current state and an action that was dispatched as it’s parameters and returns the new state.



* Now the question is how does the reducer go about producing the new state for the application. Well that is pretty simple, it first checks which type of action was dispatched and based on it returns the new state.

const postReducer = (state = [], action) => {

switch(action.type) {

case 'ADD\_POST':

return state.concat([action.data]);

default:

return state;

}

}

export default postReducer;

* Now what is happening here is that we are using a ‘switch statement’ and we are switching based on the value of action.type. If the value is ‘ADD\_POST’ we are returning a new array containing action.data. Basically whenever the ‘ADD\_POST’ event happens we want to push some data into the state array.Now what is action.data? Well it’s nothing but an object with our individual post title and the post message. One thing to note here is that the reducer function expects a default value for the state. Here we are using ES6 default-parameter syntax to add that. The default value for the state here is an empty array. One other thing to note is that a reducer must always have the default clause inside the switch statement. In the default clause we simply return the state. This is done so that in case none of the action.type value matches any of the cases we simply return the state.
* The Provider component uses something called React Context which allows you to pass the store object to any components that needs to access it without the need to pass props. Here we are wrapping the App component which is our parent component with the Provider component so that all the child components in our app can get access to the store. The Provider component takes the store as a prop.

import postReducer from './reducers/postReducer';

const store = createStore(postReducer);

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>, document.getElementById('root'));

* It seems like our data is being captured properly. Great all is left now is to dispatch an action. To do that we will make use of the connect() function provided by the react-redux library. Now this is where things might get a bit tricky but I will try my best to explain it. We know that our state lives inside this object called the store and this store has it’s own set of methods for getting the current state of our application, updating the state of our application and subscribing for changes. We have already discussed one of these methods called dispatch. We need dispatch whenever we want to pass some action to the reducer to tell some sort of event has happened and then the reducer can decide what to do with the state. But to do that we need access to dispatch. Won’t it be great if we somehow got access to the dispatch method as a prop. That is what connect() allows you to do. connect() returns a function which takes in your current component as an argument and returns a new component with dispatch method as it’s prop. The main idea to remember is that connect will ultimately return a new component which has the dispatch method as a prop.

export default connect()(component-name)

Example :

import {connect} from 'react-redux';

* Remember that connect() gives you access to dispatch as a prop. Here once we have captured the data from the form we dispatch the action using this.props.dispatch() passing in the data object with a type of ‘ADD\_POST’.
* Great, now we have added some data in our state but we can’t see any of those changes reflected in our application so let’s fix that. Before doing that let’s understand one more important thing about connect(). This special function provided by the react-redux library gives you access to dispatch whenever you call it wrapping the component-name as an argument to the function that is returned. We have seen this syntax which is as follows-

export default connect()(component-name)

* Additionally, connect can do more. It can give you access to your state which is living inside your store object. To get access to your state, we need to use a special function called mapStateToProps. This function does exactly what it is named, map the state from the store object to the props object in your components. Let’s define this mapStateToProps function-

const mapStateToProps = (state) => {

return {

posts: state

}

}

* The argument to mapStateToProps is our application state. To understand this better, imagine whatever you pass inside the mapStateToProps argument is your state. Next question to ask is what is that state is it an array or an object or something else? Well that will depend on what you have defined it in your reducer. Since we have only one reducer which is the postReducer, we know that the state is an array.
* The entire state of our application lives inside an object called the store. In order to update the state we need to dispatch an action. Actions are nothing but Javascript objects with a type property which describes the event taking place. Events can be anything from updating counters to adding posts like we have seen above. Once the action has been dispatched, it is received by the reducer. The reducer takes in the current state of the application and the dispatched action and produces the next state of the application based on action.type.
* For our React application to use the Redux store, we use the Provider component provided by the react-redux library and put it as the root of all the components.
* In order to access our Redux store within our React components we use the special connect() function. This function gives us access to dispatch and when we pass in mapStateToProps it gives us access to the state. mapStateToProps is a function that takes in the state of our application as a parameter and returns an object with keys of that object becoming the props of the component so that whenever we use this.props.key\_name we get back the state we need.