Modern React Redux

What's in the Course

- ★ Complete guide to React
- ★ In depth look at How React Works under the hood
- ★ Complete Guide to Redux: Reducers, Actions and Action Creators
- ★ How to Make Redux Work with React
- ★ Complete Guide to React-Router
- ★ In depth look at how React-Router works under the Hood
- ★ Google and Facebook Oauth logins
- ★ How to setup authentication in React-Redux with JWT tokens

We also put everything together in the final project to build a complete front end app with React-Redux.

Course Intro

-We will put together a React-Redux app with Routing and Authentication and learn how it works at a deep level.

-We will work directly in the development environment, we will not use JS Fiddle or Codepen.

-I will teach you all the practical concepts

-We will focus on the main fundamental concepts of React and Redux and avoid styling.

-I advise you not to delete code unless specified.

Overview

Outcomes:

- Setup Redux Actions
- Setup a Redux Reducer
- Setup Action Creators
- Setup a React Container

Technologies Used:

- React
- Redux
- React-Redux
- Redux Thunk

Separation of Concerns

Traditional Web App: App is separated based on programming language. Example, HTML handles layout, Javascript handles all the business logic and CSS handles styling.

HTML, Javascript, CSS is kept in separate files.

Single Page App: App is separated based on components. Each component is responsible for one main functionality of the app. Example, addpost.js component only handles submitting a post and editpost.js is only responsible for editing posts.

The UI layout, styling and business logic can be contained in the same file with JSX

React

Props

- Short for Properties
- Functional Components: Accessed with the syntax “props.” (name of user defined property)
- Class Components: Accessed with the syntax “this.props” (name of user defined property)
- Part of the 1 way data binding pattern
- Read only

Components

Functional Components:

- Set up usually as an arrow function
- Returns JSX directly
- Not aware of state
- Can have optional props

Class Components:

- Set up as an ES6 JS class
- Also referred to as containers
- Have a render function that returns JSX
- Aware of state
- Can have optional props

State

-Temporary Data

Examples: Whether a user is logged in or not, the page a user is on, or whether a dialog box is open.

Examples of Non Temporary Data: blog posts, comments and user profile data.

In React state is a Javascript object, a key-value pair.

-Changed with the setState() method.

Changing State

Done Through the setState() function

3 Aspects of changing state:

1. Do not Mutate state directly
2. Change the state by referencing the previous state.
3. React will merge the old state with the new state.

Ternary Expression

{condition ? code block if true : code block if false}

Lifecycle Methods

- ComponentDidMount()
- ComponentWillUnmount()

Redux

The three Principles of Redux

1st Principle

There is only one source of truth.

2nd Principle

State is read-only.

3rd Principle

Changes are made with Pure Functions

Actions

- Redux Specific
- Can be dispatched from React
- Describe what will happen to the state
- Have a required “type” property
- Can have any other optional properties

For example, an action with the type “login_success” might change a property of Redux state called isAuthenticated from false to true.

Action Creators

- Can dispatch actions asynchronously
- a function which dispatches a normal action
- Allows for dynamically setting the payload property
- No changes required to the reducer

For example if you want to save the user input to the state. You would setup the action creator with an arrow function like so:

(user_input_text) => (type: “USER_INPUT”, payload: {text: user_input_text} }

Reducers

- Directly change the Redux state
- Must be “pure”, or Can’t be async
- Takes “state” and “action” as parameters
- Usually setup as a switch statement
- Each case statement has to match an action type

```
case ACTION_TYPES.LOGIN_SUCCESS:
  return  {
    ...state,
    isAuthenticated: true,
  }
```

mapStateToProps()

- Essentially, how to get the state from Redux
- Property in the return statement is user defined

- Set the value to “state.” (name of reducer if more than one) “ . ” (name of property from reducer)
- In the React render method use “this.props.” name of property you set.

mapDistachToProps()

- How to change the state by dispatching actions
- Actions can be plain or async action creators
- Actions can have an optional payload

- To dispatch actions use the syntax “this.props.” (name of the property you set)

connect ()

-Connects the React Container with the Redux store

Read and Actions Container	Read Only Container	Dispatch Actions Only Container
connect (mapStateToProps, mapDispatchToProps)(Container)	connect (mapStateToProps)(Container)	connect (null, mapDispatchToProps)(Container)

	React	Redux	React-Redux
Update State	this.setState()	Dispatch actions to the reducer store.dispatch()	mapDispatchToProps() this.props.dispatch_action() Then dispatch actions to the reducer.
Read State	this.state.state_property	store.getState()	mapStateToProps() this.props.state_property

Routing

Traditional vs SPA Routing

Traditional Routing:

- Multiple HTML Files
- New server Request with each route change
- Browser Reload on each route change
- Relatively Slower performance

Single Page App Routing:

- Single HTML File
- Only 1 server request for initial web site loading
- A new component renders for each route change
- Browser does not reload with route change
- Very fast performance

Google O-Auth setup

1. Create a Project.
2. Go to Credentials
3. Select Create Oauth Client ID
4. Setup the consent screen
5. Enter Name of Application and put auth0.com as the authorized domain
6. Save
7. Select Web Application
8. For Javascript origins put your application auth0 domain.
9. For URI redirect put your auth0 domina and add /login/callback

Facebook O-Auth setup

1. Click on Apps and hit add new app
2. Click on Facebook login
3. Next you will see the quickstart on the left hand side tab
4. Choose the web app
5. Add auth0.com to valid URLs
6. Click on the Facebook Login > settings tab in the left hand side and enter you auth0 domain in the valid 0auth redirect URIs
7. After click on the Settings > Basic tab where you will find your client id and client secret.
8. Next copy these credentials in to the auth0 dashboard.

A Complete React-Redux Frontend App

Why I didn’t cover Redux Thunk

- Any functionality offered by Redux Thunk can be accomplished without Redux Thunk
- Redux Thunk is mainly Syntax sugar, meaning it makes code slightly easier to read.
- The difference is so small that in my opinion it is not worth the time to learn it for such a slight advantage.

Link Comparing Redux Thunk with regular action creators, the difference in code is quite literally 2 words.

https://stackoverflow.com/questions/34570758/why-do-we-need-middleware-for-async-flow-in-redux

Hooks

React Hooks

Hooks are referring to being able to “hook” react class functionality to a functional component.

Main purpose of hooks is to give class functionality to react functional components, mainly to allow functional components to read and update state.

This both makes development easier and improves performance.

2 Rules of Using Hooks

1. **No nested hooks calls**

Wrong:

```
if (true) { useEffect( () => {return value} ) }
```

Right:

```
useEffect( () =>  true ?  return value  :  null )
```

2. **Do not call hooks outside of the component function**

4 Main Hooks Overview

useState()	**useEffect()**	**useContext()**	**useReducer()**
Similar to this.setState(). Updates local component state.	Similar to componentDidMount() Use when you want to call function automatically	Similar to react-redux. Allows you to access and update the global context state through the React context API.	Similar to react-redux. Allows you update the local component state through redux actions and reducers. **DOES NOT** update state globally by itself

useState()

const [value, setValue] = useState(initalState)

[state value, set state function]

Set State: setValue(value + 1)
Read State: { value } in render method.

Component re-renders automatically when state changes.

Variable Names are user defined

Without Array destructuring:
const value = useState(0)[0]
const setValue = useState(0)[1]

Multiple Properties:
const [value, setValue] = useState(initalState)
const [value2, setValue2] = useState(initalState)

useEffect()

```
useEffect(() => {
        () => { return value }
})
```

useEffect will be called every time a component renders

```
useEffect(() => {
        () => { return value }
}, [] )
```

useEffect will be called when the component mounts

```
useEffect(() => {
        () => { return value }
}, [state.value] )
```

useEffect will be called when state.value changes

useReducer()

```
import * as Reducer from './store/hooks/reducer';
Import * as ACTIONS from './store/actions/actions';

const [state, dispatch] = useReducer(Reducer.reduxReducer,  Reducer.initialState)
        [ value, function ]
```

Regular Action: dispatch(type: "ACTION")
Action Creator: (payload) => dispatch(ACTIONS.create_action(payload))
Read State: { state.state_property } in render method. Same state property that you setup in the initial state, in the reducer.

What is Context?

- Context is separate from React Hooks
- Is a way to pass down props from a parent component to a deeply nested child component
- In regular React props can only be passed to a direct child component of a parent component.
- A parent can’t pass props to a child of a child component.
- Is what allows us to have a global state

useContext()

In separate context.js file:

```
const Context =  React.createContext({
        prop1: false
    })


export default Context;
```

Import the context in your root App.js file:

```
import Context from './utils/context';
```

Import and setup the actions and reducers here as well. See the useReducer() Syntax guide for more details.

useContext() cont.

In root App.js file. Wrap your routes with Provider:

```
 <Context.Provider
        value={{
           state_prop1: false,
           update_state: () => return newValue
        }}
    <Routes />
</Context.Provider>
```

**Important:

The “prop1” that we setup in the beginning in the context.js file is irrelevant and is overridden here when we initialize the provider.

Setup up all your global state and dispatches here in the root App.js file and not the context.js file.

useContext() cont.

Import the Context into a component and initialize it like so:

```
import Context from '../utils/context';

const context = useContext(Context)
```

After initializing context you can access the global state.

Read State: { context.state_prop1 } in render method

Set State: { () => context.update_state() } in event handler

**You can update the global state through either useReducer() or useState() in App.js

Migration Guide from Redux to React hooks

- Actions do not need to be changed at all.
- Reducers do not need to changed either.
- Simply export both the initial state and the reducer instead of just the reducer. Do not use “export default”
- Then Pass in both the reducer and initialState to the useReducer() hook
- Export and Import actions as normal.
- Actions are dispatched as normal

	React	Redux	React-Redux	React useState()	React useReducer()	React useContext()
Update State	this.setState()	Dispatch actions to the reducer store.dispatch()	mapDispatchToProps() this.props.dispatch_action() Then dispatch actions to the reducer.	setValue(value)	dispatch(type: ‘string of action’) **Or** dispatch(ACTIONS.function())	{() = > context.name_of_property()}
Read State	this.state.state property	store.getState()	mapStateToProps() this.props.state_property	{name_of_property} Name of property user defined in local component	{state.name_of_property} Name of property comes from Reducer	{context.name_of_property} Name of property comes from value prop of Context Provider

The Final Project

Building upon a existing basic React Redux App. This will allow you to see how React Hooks compare to React - Redux side by side.

A lot of current projects still use Redux so it’s important to understand both approaches.

We will not change the existing React Redux code but build React Hooks separately from scratch