Introduction to the React and Redux Challenges

React is a view library that you provide with data, then it renders the view in an efficient, predictable way. Redux is a state management framework that you can use to simplify the management of your application's state.

In a React Redux app, you create a single Redux store that manages the state of your entire app. Your React components subscribe only to the pieces of data in the store that are relevant to their role. Then, you dispatch actions directly from React components, which then trigger store updates.

Redux is not designed to work with React out of the box, you need to use the react-reduxpackage. It provides a way for you to pass Redux stateand dispatchto your React components as props.

**1. Getting Started with React Redux**

class DisplayMessages extends React.Component {

// change code below this line

constructor (props) {

super(props);

this.state = {

input: '',

messages: []

}

}

// change code above this line

render() {

return <div />

}

};

**2. Manage State Locally First**

class DisplayMessages extends React.Component {

constructor(props) {

super(props);

this.state = {

input: '',

messages: []

}

this.submitMessage = this.submitMessage.bind(this);

this.handleChange = this.handleChange.bind(this);

}

// add handleChange() and submitMessage() methods here

handleChange(event) {

this.setState({input: event.target.value});

}

submitMessage() {

this.setState(

{

messages: [...this.state.messages, this.state.input],

input: ""

}

);

}

render() {

const messageOutput = this.state.messages.map(i => <li key = {i + 1}>{i}</li>);

return (

<div>

<h2>Type in a new Message:</h2>

{ /\* render an input, button, and ul here \*/ }

<input

value = {this.state.input}

onChange = {this.handleChange}

/>

<br/>

<button onClick={this.submitMessage}>Add message</button>

<ul>

{messageOutput}

</ul>

{ /\* change code above this line \*/ }

</div>

);

}

};

## Type in a new Message:

  
Add message - button



**3. Extract State Logic to Redux**

// define ADD, addMessage(), messageReducer(), and store here:

const ADD = 'ADD';

const addMessage = (message) => {

return {

type: ADD,

message: message

}

}

const initialState = {

messages: []

};

const messageReducer = (state = initialState.messages, action) => {

switch(action.type) {

case ADD:

return [...state, action.message];

break;

default:

return state;

}

};

const store = Redux.createStore (messageReducer);

**4. Use Provider to Connect Redux to React**

// Redux Code:

const ADD = 'ADD';

const addMessage = (message) => {

return {

type: ADD,

message

}

};

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

switch (action.type) {

case ADD:

return [

...state,

action.message

];

default:

return state;

}

};

const store = Redux.createStore(messageReducer);

// React Code:

class DisplayMessages extends React.Component {

constructor(props) {

super(props);

this.state = {

input: '',

messages: []

}

this.handleChange = this.handleChange.bind(this);

this.submitMessage = this.submitMessage.bind(this);

}

handleChange(event) {

this.setState({

input: event.target.value

});

}

submitMessage() {

const currentMessage = this.state.input;

this.setState({

input: '',

messages: this.state.messages.concat(currentMessage)

});

}

render() {

return (

<div>

<h2>Type in a new Message:</h2>

<input

value={this.state.input}

onChange={this.handleChange}/><br/>

<button onClick={this.submitMessage}>Submit</button>

<ul>

{this.state.messages.map( (message, idx) => {

return (

<li key={idx}>{message}</li>

)

})

}

</ul>

</div>

);

}

};

const Provider = ReactRedux.Provider;

class AppWrapper extends React.Component {

// render the Provider here

render ()

{

return (

<Provider store={store}>

<DisplayMessages/>

</Provider>

)

}

// change code above this line

};

**5. Map State to Props**

const state = [];

// change code below this line

const mapStateToProps = (state) => {

return {

messages: state

}

};

**6. Map Dispatch to Props**

const addMessage = (message) => {

return {

type: 'ADD',

message: message

}

};

// change code below this line

const mapDispatchToProps = (dispatch) => {

return {

submitNewMessage: function(message) {

dispatch(addMessage(message));

}

}

};

**7. Connet Redux to React**

const addMessage = (message) => {

return {

type: 'ADD',

message: message

}

};

const mapStateToProps = (state) => {

return {

messages: state

}

};

const mapDispatchToProps = (dispatch) => {

return {

submitNewMessage: (message) => {

dispatch(addMessage(message));

}

}

};

class Presentational extends React.Component {

constructor(props) {

super(props);

}

render() {

return <h3>This is a Presentational Component</h3>

}

};

const connect = ReactRedux.connect;

// change code below this line

const ConnectedComponent = connect(mapStateToProps, mapDispatchToProps)(Presentational);

**8. Connect Redux to the Message App**

// Redux:

const ADD = 'ADD';

const addMessage = (message) => {

return {

type: ADD,

message: message

}

};

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

switch (action.type) {

case ADD:

return [

...state,

action.message

];

default:

return state;

}

};

const store = Redux.createStore(messageReducer);

// React:

class Presentational extends React.Component {

constructor(props) {

super(props);

this.state = {

input: '',

messages: []

}

this.handleChange = this.handleChange.bind(this);

this.submitMessage = this.submitMessage.bind(this);

}

handleChange(event) {

this.setState({

input: event.target.value

});

}

submitMessage() {

const currentMessage = this.state.input;

this.setState({

input: '',

messages: this.state.messages.concat(currentMessage)

});

}

render() {

return (

<div>

<h2>Type in a new Message:</h2>

<input

value={this.state.input}

onChange={this.handleChange}/><br/>

<button onClick={this.submitMessage}>Submit</button>

<ul>

{this.state.messages.map( (message, idx) => {

return (

<li key={idx}>{message}</li>

)

})

}

</ul>

</div>

);

}

};

// React-Redux:

const mapStateToProps = (state) => {

return { messages: state }

};

const mapDispatchToProps = (dispatch) => {

return {

submitNewMessage: (newMessage) => {

dispatch(addMessage(newMessage))

}

}

};

const Provider = ReactRedux.Provider;

const connect = ReactRedux.connect;

// define the Container component here:

const Container = connect(mapStateToProps, mapDispatchToProps)(Presentational);

class AppWrapper extends React.Component {

constructor(props) {

super(props);

}

render() {

// complete the return statement:

return (

<Provider store={store}>

<Container/>

</Provider>

);

}

};

**9. Extract Local State into Redux**

// Redux:

const ADD = 'ADD';

const addMessage = (message) => {

return {

type: ADD,

message: message

}

};

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

switch (action.type) {

case ADD:

return [

...state,

action.message

];

default:

return state;

}

};

const store = Redux.createStore(messageReducer);

// React:

const Provider = ReactRedux.Provider;

const connect = ReactRedux.connect;

// Change code below this line

class Presentational extends React.Component {

constructor(props) {

super(props);

this.state = {

input: ''

\*\*\*removed messages property, these messages will be managed by Redux

}

this.handleChange = this.handleChange.bind(this);

this.submitMessage = this.submitMessage.bind(this);

}

handleChange(event) {

this.setState({

input: event.target.value

});

}

submitMessage() {

this.props.submitNewMessage(this.state.input);

modify method so that the method dispatches submitNewMessage() from this.props, and pass in the current message input from local state as an argument.

this.setState({

input: ''

\*\*\*removed messages property here as well because it was removed from local state, “messages: this.state.messages.concat(currentMessage)”,

});

}

render() {

return (

<div>

<h2>Type in a new Message:</h2>

<input

value={this.state.input}

onChange={this.handleChange}/><br/>

<button onClick={this.submitMessage}>Submit</button>

<ul>\*\*\*state to props – it maps the messages received from props rather that the state

{this.props.messages.map( (message, idx) => {

return (

<li key={idx}>{message}</li>

)

})

}

</ul>

</div>

);

}

};

// Change code above this line

const mapStateToProps = (state) => {

return {messages: state}

};

const mapDispatchToProps = (dispatch) => {

return {

submitNewMessage: (message) => {

dispatch(addMessage(message))

}

}

};

const Container = connect(mapStateToProps, mapDispatchToProps)(Presentational);

class AppWrapper extends React.Component {

render() {

return (

<Provider store={store}>

<Container/>

</Provider>

);

}

};

**10. Moving Forward From Here**

When working with “npm and a file system on your own machine”, import all dependencies

// import React from 'react'

// import ReactDOM from 'react-dom'

// import { Provider, connect } from 'react-redux'

// import { createStore, combineReducers, applyMiddleware } from 'redux'

// import thunk from 'redux-thunk'

// import rootReducer from './redux/reducers'

// import App from './components/App'

// const store = createStore(

// rootReducer,

// applyMiddleware(thunk)

// );

// ReactDOM.render(

// <Provider store={store}>

// <App/>

// </Provider>,

// document.getElementById('root')

// );