Section 8 – Redux

110. Section Overview

111. Redux introduction

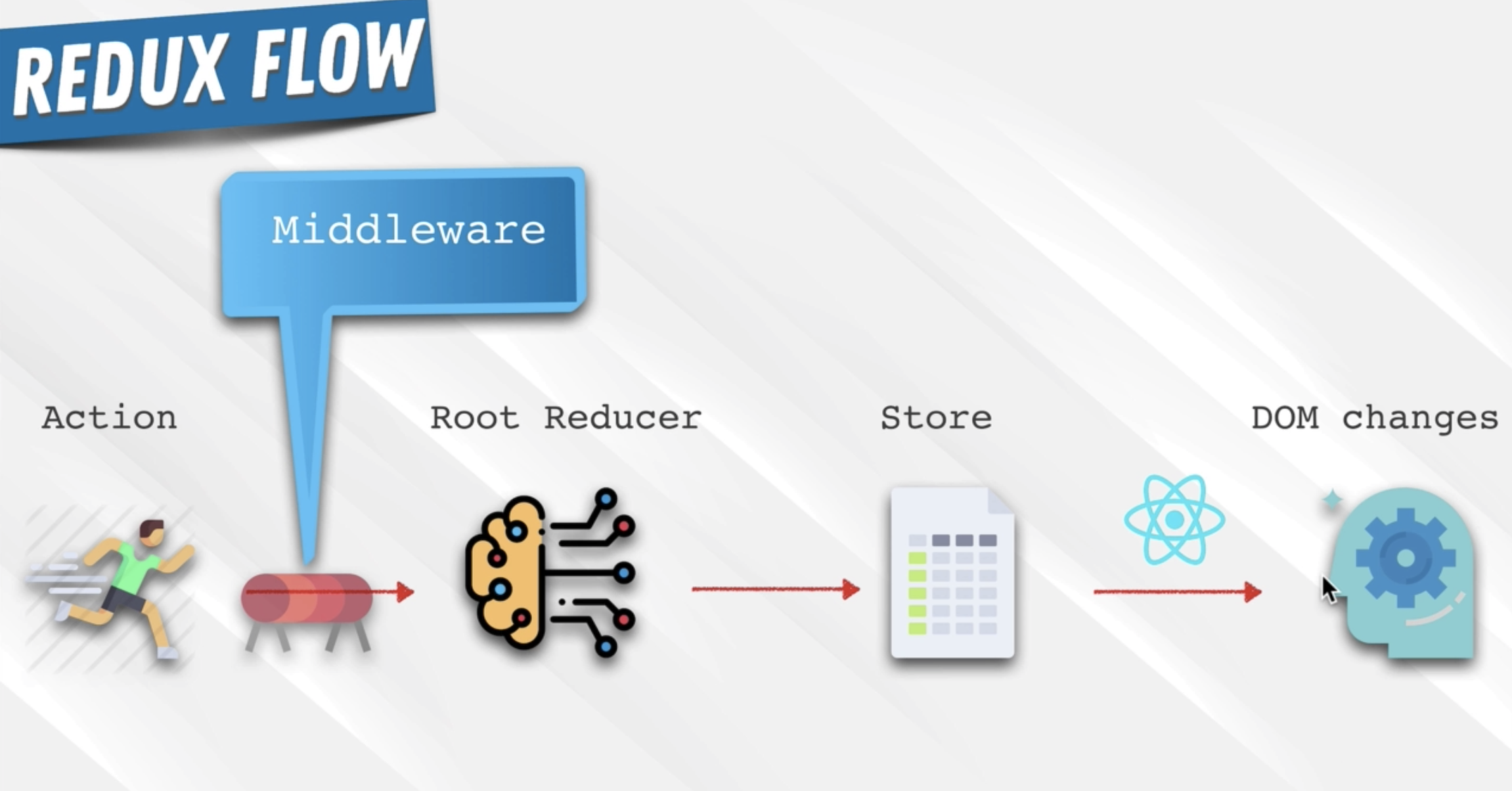
112. Redux Concepts

113. Redux in our application

114. Redux Action and Reducers

115. Setting up Redux 1

a. Redux flow



b. npm I redux redux-logger react-redux

c. Put the { Provider } to index.js from ‘react-redux’. Wrap the whole App so that the whole application will have an

access to all features related to store. It will be the parent of everything.

d. Write the store. Create the redux folder inside the src and create the root-reducer.js inside it.

**i. Root reducer will combine all the reducers in it.**

e. Create the user-reducer.js.

i. Put it inside the user folder.

ii. A reducer is a function that gets 2 properties namely the   
state and action.

iii. The state is an object which represents the initial or last state.

iv. Action is an object which consists of the following:

1. type: which is a string. Type is just a name that tells the specific action.

2. payload = is an action object that could be of anything

v. Assign the initial state: const INITIAL\_STATE = { currentUser: null }, put this as the default parameter for state = INITIAL\_STATE

vi. Use Switch Statement function to return the value of state base from the action. Default: return state is if there’s no matching action found, then the function will just return the current state.

vii. export default userReducer

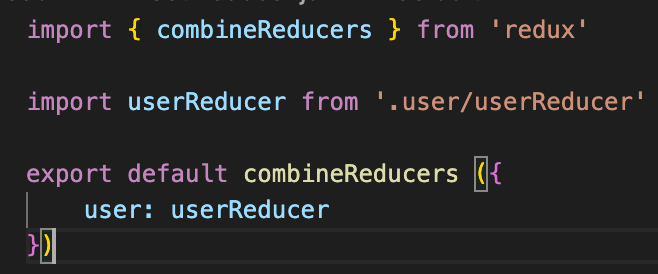


f. bring the userReducer to root-reducer.js

i. import { combineReducers } from ‘redux’

ii. import userReducer from ‘.user/user.reducer

iii. export default combineRedcers and put the userReducer inside the object. assign a key to userReducer



116. Setup Redux 2

a. Create the store

i. create the store.js inside the redux folder

ii. import { createStore, applyMiddleware } from ‘redux’

1. middleware is a function that catch the actions and display it before it reach the root reducer.

2. import logger from ‘redux-logger

iii. import root-reducer from ‘./root-reducer’

b. set up the middlewares

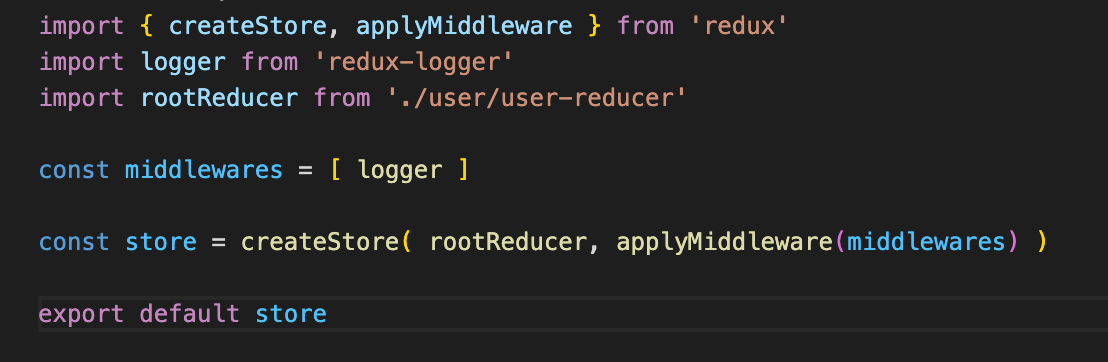
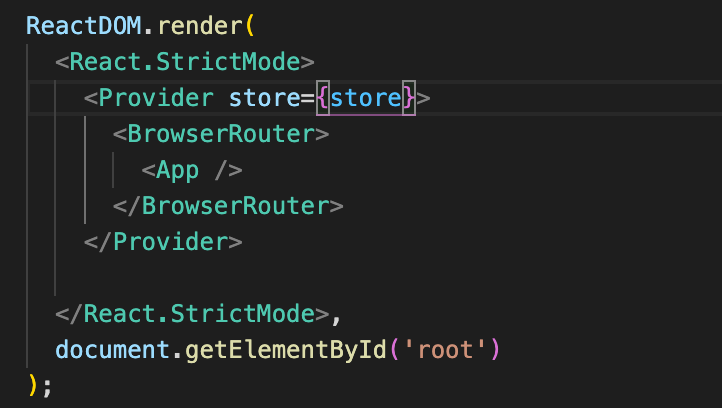
i. const middlewares = [ logger ] <<< results to an array

c. create the store

i. const store = createStore( rootReducer, applyMiddleware(…middlewares) )

d. export default store;

e. bring the store to App.js and pass it as a props in the Provider

f. Implement | Create Action – so that the redux can now start to store the value of user into the userReducer and out

to the <App>

i. create the action that will trigger the SET\_CURRENT\_USER inside the user-reducer.js

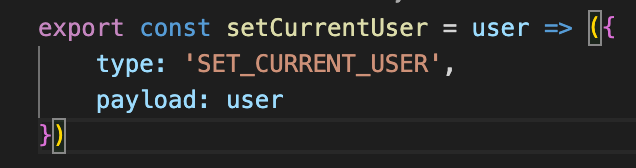
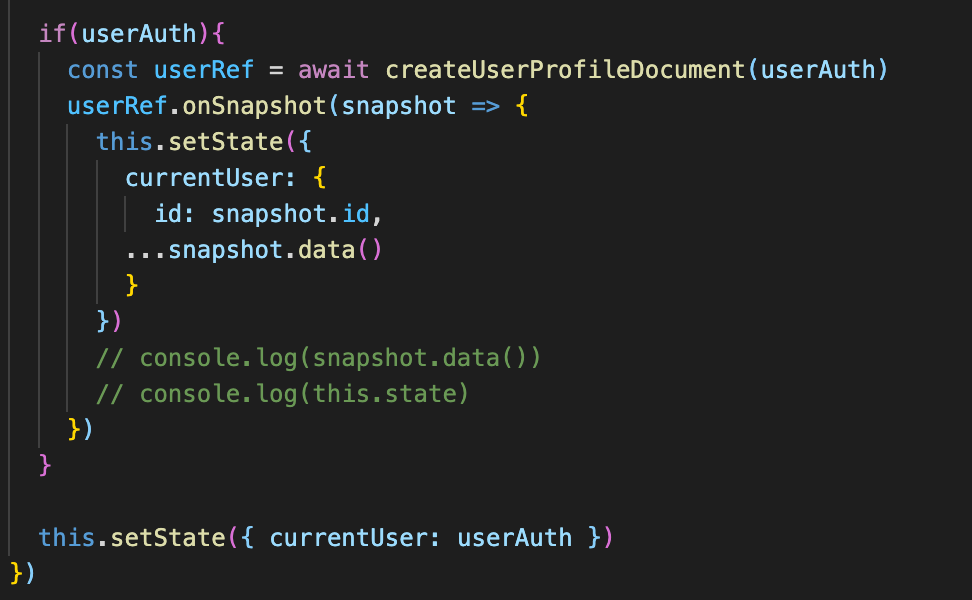
ii. create the user-actions.js inside the /redux/user

iii. action is a function that returns an object that consist of the type and payload

1. SET\_CURRENT\_USER should be the same as with the user-reducer as this is the string value expected

by the reducer.

2. user is an object that came from <App>. It’s either the userAuth or the snapShot

117. connect() and mapStateToProps

a. use the userReducer to <App> at the <Header> as the <Header is using the currentUser from the <App>

b. import { connect } from ‘react-reducer’ at the header.js

1. connect is an HOC that let’s the components access the features of redux

2. connect takes 2 functions; 1 is optional ( mapStateToProps ) and the 2nd is the upgraded component

(Header)

c. mapStateToProps can be of any name.

1. It will be the function that will allow the <Header> to access the state with the state being the

reducer (root-reducer.js)

2. it’s an object that will have the name of the properties will be the actual properties that needs to pass in.

3. It will get the ‘state‘ which is the root reducer, the top level root reducer.

i. currentUser: state.user.currentUser

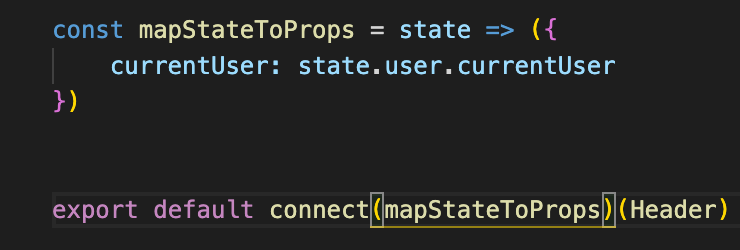
1. currentUser is the name of the property

2. state – is the root reducer

3. user (key) is a property of the root-reducer that will give the userReducer

4. currentUser is the property returned in the user-reducer.js

d. Remove the passing of the props to <Header> in the <App>



118. mapDispatchToProps

a. Update the <App> so that it can update the reducer value base from SET\_CURRENT\_USER of the

user.actions.js.

b. import { connect } from ‘react-redux’ & import { setCurrentUsr } from ‘./redux/user/user.actions.js’

c. export default connect(null, mapDispatchToProps)(App) <<< null because the currentUser is not needed in

the App

d. mapDispatchToProps is a function that gets the dispatch property and will return

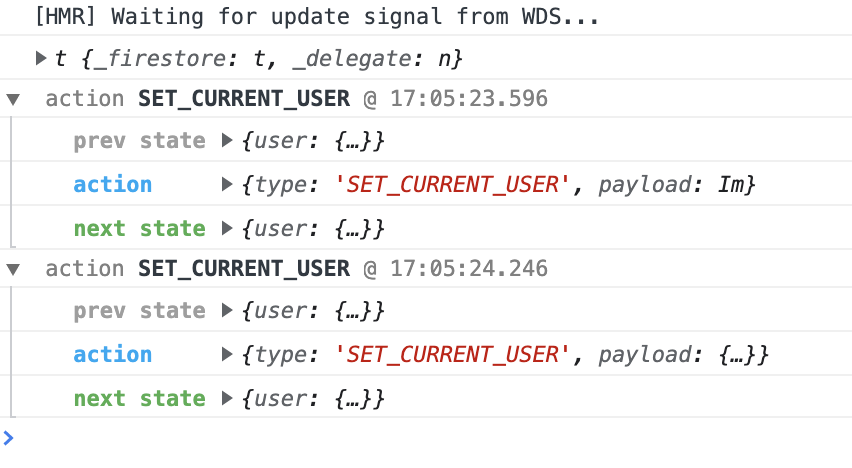
e. setCurrentUser can now replace the constructor and currentUser

f. replace the this.setState by currentUser

g. destructure the const { setCurrentUser } = this.props

h. remove the this.setState({currentUser:userAuth}) >>> setCurrentUser(userAuth)

i. at this point the logger is now displaying the actionin the console



119. User Redirect and User Action Type – this fir the user not to see the sign in page if there’s a successful signin.

a. 2 items needed:

i. At App.js, import { Redirect } from ‘react-router-dom

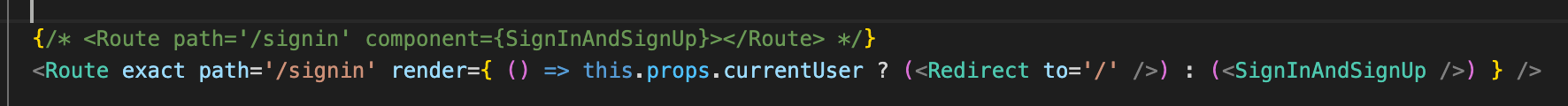
II. the user from root.reducer.js

1. use mapStateToProps = ({ user }) => ({ currentUser: user.currentUser })

2. this will grant access and get the state

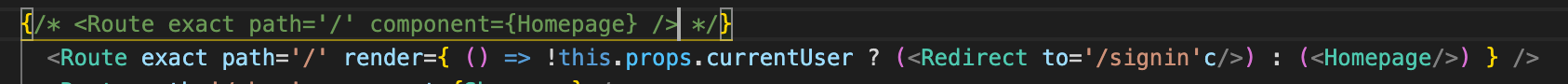
b. use the ternary operation to evaluate the presence or absence of currentUser and use the render function

to invoke what component needs to be rendered. Do this at the <Route component={SigninAndSignupPage}>



c. Replace the null by mapStateToProps to connect(mapStateToProps, mapDispatchToProps)(App)

d. did the same thing with the SIGN OUT button. If !this.props.currentUser.



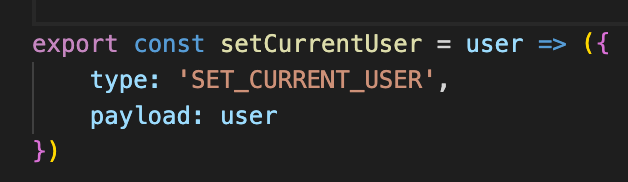
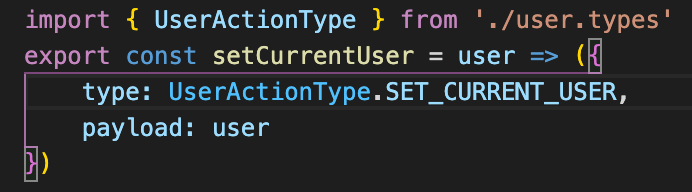
e. Improve the SET\_CURRENT\_USER to make it consistent to all places it can be use

i. create the set.types.js inside the user folder

ii. export const UserActionTypes = { SET\_CURRENT\_USER: ‘SET\_CURRENT\_USER }

iii. import UserActionTypes to user.action.js and all its been use

iv. then convert all SET\_CURRENT\_USER to UserActionTypes.SET\_CURRENT\_USER

120. Cart Component

a. Get the bag svg

b. create the cart icon component

121. Cart Drop down Component

a. create the cartDropdown component

b. import the <CustomButton>

c. import the <CartDropdown> to <Header> just outside the option. Watch out for the techque for dropdown

component.

122. Implementing redux in cart – this is to show and hide the <CartDropdown> component base in clicking the cart

icon. This method is to make the <CartDropdown> completely independent and reusable anywhere.

a. Create the redux files, cart.reducer.js, cart.types.js and cart.actions.js

b. Create the cartReducer. Set the const INITIAL\_SET = { hidden: true }

i. hidden: !state.hidden <<< this provides a toggle functionality rather than setting the state.

c. create the cart.types.js

i. export default CartActionTypes = { TOGGLE\_ART\_HIDDEN: ‘TOGGLE\_CART\_HIDDEN’ }

ii. then import this in the cartReducer

d. Create the art.actions.js

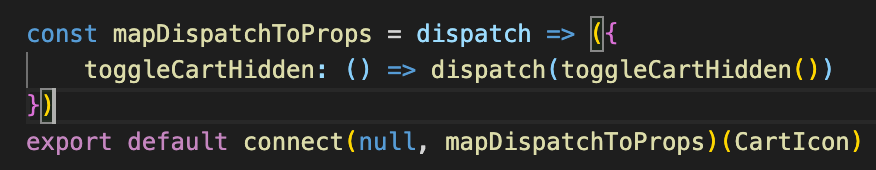
i. import the CartActionTypes

ii. no payload is required because state is not being set, instead only a toggle is trying to achieve

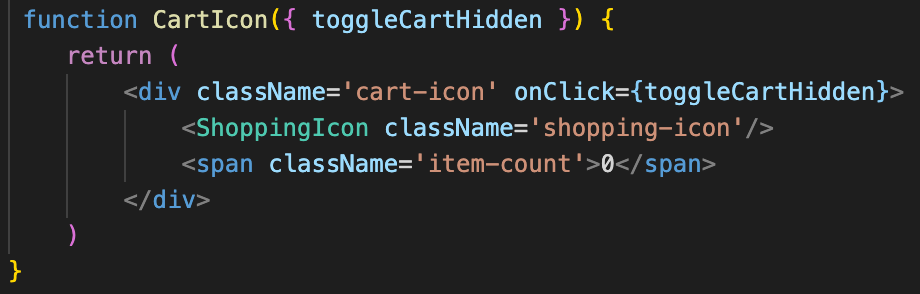
iii. type: CartActionTypes.TOGGLE\_CART\_HIDDEN <<< TCH is the initial string assigned from cartReducer

e. in the <Cart Icon> component, import the cart.actions.js and import { connect } from ‘react-redux’

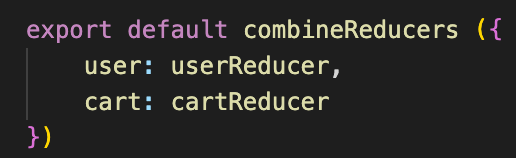
i. Create the mapDispatchToProps

ii. 

iii. pass the toggleCartHidden to the CartIcon and assign it in the onClick



f. Add the cartReducer to the roo-reducer



g. Destructure the state in the <Header>



123. Add to cart styling

a. Re-customize the <CustomButton> by dfining theclassNAme=’inveted’

b. Import <CustomButton> in the <CollectionItem />

c. re-style the <CollectionItem /> after the inclusion of the <CustomButton> hide initially the <CustomButton> and put it

inside the <CollectionItem /> box

124. Cart Item Reducer

a. Create the property that needs to set inside the reducer

i. in the cartReducer, add the new property cartItems: []. Use array as vsalue will be added or remove from it

depending on the available data

ii. in the cart.actions.type, add the new cart actions so that the reducer is aware of what is the addItem is to

be made.

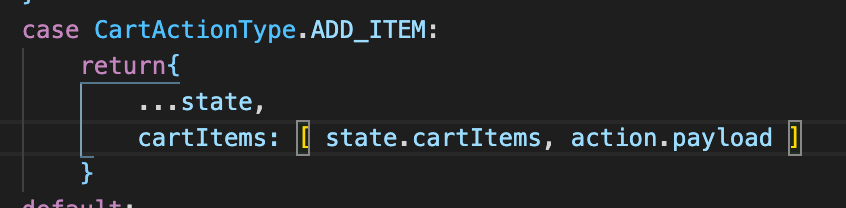
1. create the ADD\_ITEM: ‘ADD\_ITEM’

iii. create a function to add the item whenever the user clicks it

1. add a new case in the cartReducer

2. return an object that all of the state (…state) and cartItems. Include the old and the new (payload)

cartItems that got fired in the array. cartItems: [state.cartItems, action.payload]

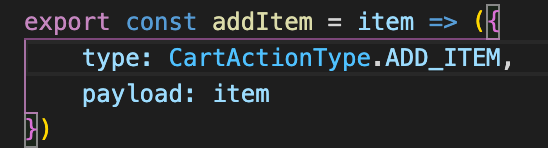


iv. in the cart.actions.js

1. create the new export const addItem = item => {type: CartActionItems.ADD\_ITEM, payload: item}

2. 1 is a function that will get the item that needs to be add in the cartItems array. It will return then an

object that consists the ADD\_ITEM and the payload



v. Bring the addItem to <CollectionItem > to be able to use this state

i. import { connect } from ‘react-redux’

ii. import { addItem } from ‘../redux/cart/cart.actions’

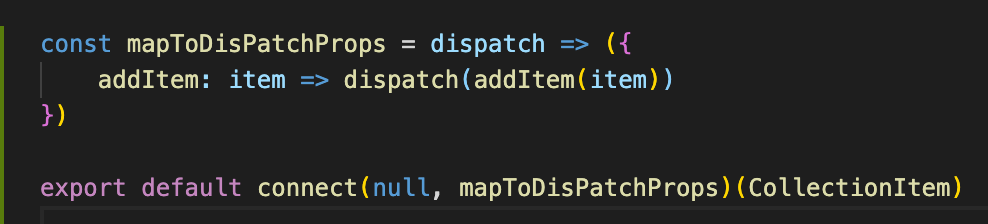
iii. create the mapDispatchToProps because it needs to dispatch the addItem

iv. addItem: is the imported function from cart-actions.js

v. whenever there’s an addItem, it will get the item as the property that will represent the addItem that

was passed in

vi. then dispath the addItem function creator pasing the (Item) as the payload.



vii. as there’s no need for mapStateToProps as of the moment.

vii. destructure the addItem in the CollectionItem

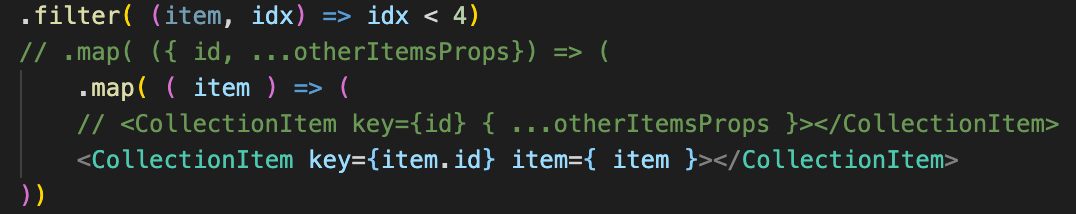
b. The <CollectionItem > has now access to addItem, It now needs to access the item that the <CollectionItem

> represents in order for the item be dispatch in the addItem

i. Tweak the <CollectionPreview >.

1. pass the whole item to the map in the <CollectionPreview > and pass the item={item}

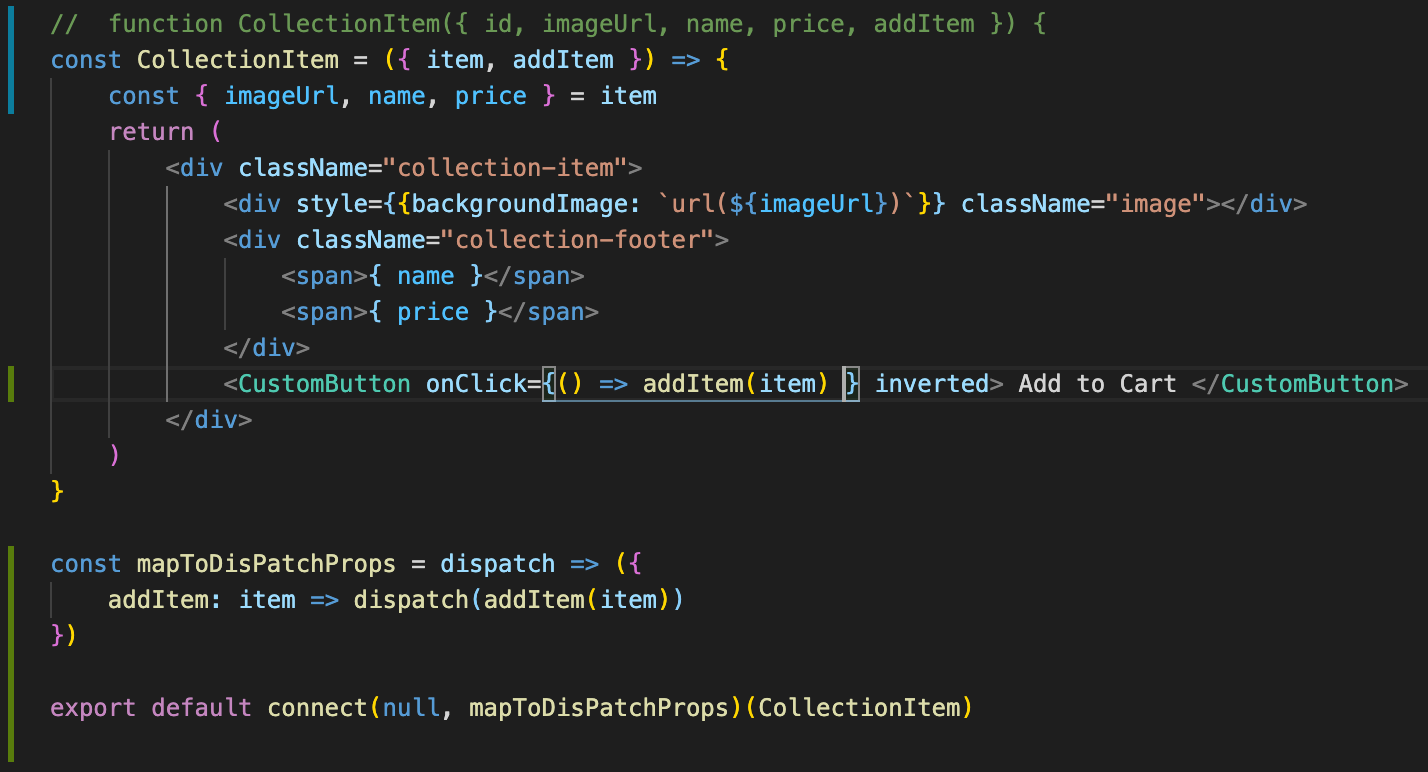
2.



c. back in the <CollectionItem> replace the destructured props by item. Remove the id as its not being use.

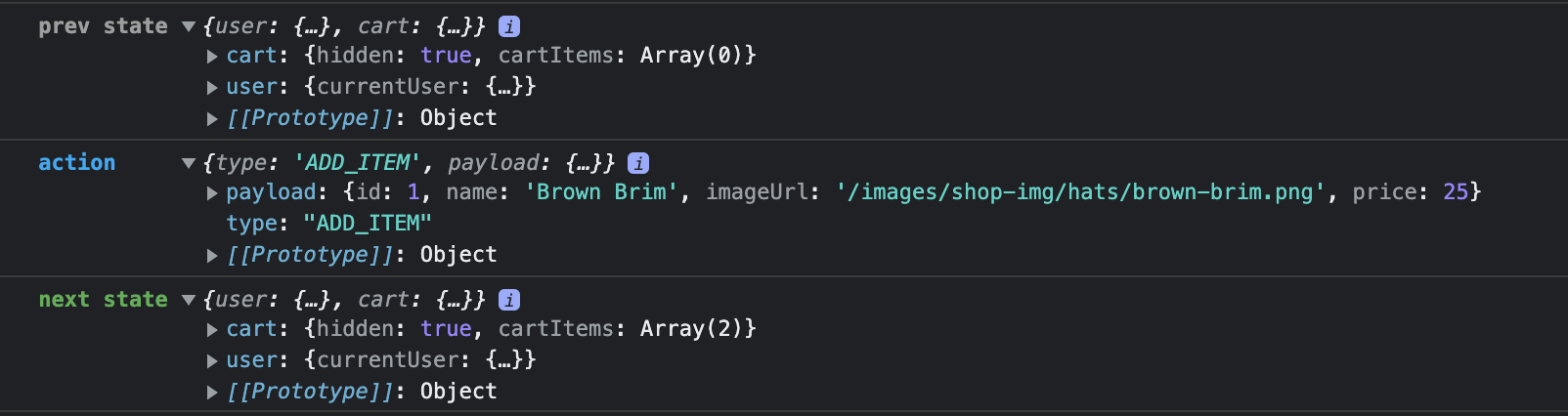
d. destructure the item, price, imageUrl = item. Then we can use now the item

e. pass the item as a props in the function of the <CustomButton onClick={() => addItem(item)}



f. the ADD\_ITEM action is now being fired and the cart is being updated though same items kept on

appearing.



g. pls note that the item used in the addItem function creator comes from the <CollectionPreview> items state

which was filetered then map

125. Adding multiple items to cart – grouping the cartItems inside the cart reducer. Create a utility that will Make sure that the

object pass will result to change only in the properties and will not make the component to re-render.

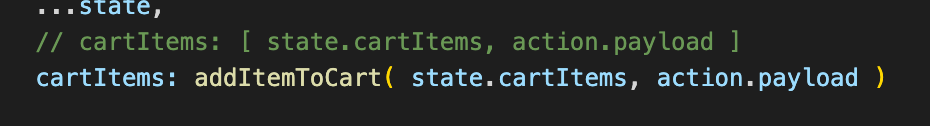
Utility function will allow to keep functions that may be needed in different locations.

a. create the cart.utils.js in cart folder



b. import the addItemToCart in cart.reducer

i. in the case, replace the value of cartItems by the addItemToCart( state.cartItems, action.payload )



c. replace the shop.data.js due to the id error

d. the quantity is now updating



127. Cart Item Component

a. Create the <CartItem />

b.