Implementing Redux

1. Review -
   1. React component subscribes to the store by useSelector
   2. How do we write our reducers??
   3. Map our state logically across different reducer
   4. within each –
      1. we need to create an object to store your initial state values that you want that reducer to be connected to
      2. create a pure function that takes two parameters: state and action
         1. state should default to your initial state object
      3. Create a switch statement that checks the action.type and performs the logic to create the new state object
      4. Return the new state. This updates the state in Redux and re-render subscribed React components.

*Object.assign() syntax; target, origin, {payload: payloadVal}*

* *Assign() will overwrite props that are new/different but keep the rest.*
* *We need to make a new Object because states are immutable (read-only) and values within it cannot be overwritten.*

dispatchEvent({

type: CHANGE\_COUNT,

payload: 5

})

const intialState = { count: 0 }

function countReducer(state = initialState, action) {

*switch* (action.type) {

*case* CHANGE\_COUNT:

*return* Object.assign({}, state, { count: action.payload })

*default*:

*return* state;

}

};

1. When Redux is initialized, it will call all reducers and pass in “action” as “null”
   1. And render the page as necessary.
   2. There is nothing that disallows us from adding multiple payloads
      1. But often, multiple payloads may be added as a nested object
2. How do we create our store??
   1. Combine your reducers into a single object
   2. Import object returned by combineReducers and pass into createStore method from redux
   3. This is also where we can add in redux devtools.
   4. REMEMBER WHAT YOU NAMED YOUR REDUCER!
   5. Usually, we have reducers in a separate file
      1. And each reducer has its own initial state
   6. And we import them to index.js via combineReducers redux method.
      1. And combining them creates our total initial state.
   7. Basically, we need an individual reducer for each state
      1. (count, user, etc)
      2. And each reducer has its own initial state
         1. Initial state for count: 0
         2. Initial state for user: …?
      3. Then the combineReducers collects all the initial states
      4. And creates a new “initial state”
      5. And we do this inside store.js (2nd half of code below)

*Syntax:*

*Starts in index.js ↓*

*import* { combineReducers } *from* 'redux';

*import* countReducer *from* './countReducer.js'

*import* userReducer *from* './userReducer.js'

const reducers = combineReducers({

counter: countReducer,

user: userReducer

})

*//store.js*

*import* { createStore } *from* 'redux'

*import* reducers *from* './index.js'

const store = createStore(reducers);

*export* *default* store;

1. On initialization
   1. We pass the reducers the initial state and “null”
   2. Each reducer has its own initial state gets passed through the closure
   3. To “state” via const store = createStore(reducers)
2. Calling useDispatch within a component returns Redux’s dispatch function
   1. Invoke dispatch in handlers for state-changing events (onClick, onChange, etc)
   2. When invoked, these handlers will **dispatch actions**
   3. This is accomplished by invoking action creators that will return action objects, which are passed into dispatch.
      1. **We will write the action creators that make the action object that holds the type and payload data.**
   4. This is helpful because we **can’t use async functions in useDispatch/useSelector** but we can in our action creators.
3. How do we create an action creator?
   1. **Caveat – if there is a typo in the action creator,**
   2. **It will not catch the error; it will instead use the default.**
   3. Import action types into the action creator…
   4. Create actions.js or something like that that will help you keep track of your action creators.
   5. Redux Toolkit – very helpful. Do this extension if you can.
4. Keep in mind the data flow chart in mind.
   1. Store -> react component -> action -> reducer
5. How do we make our store available to react???
   1. Import “Provider” react-redux, the store from store.js, and your top.level component
   2. Wrap your top level component in the Provider component
   3. Pass the store as a prop of the Provider.
      1. This isn’t giving us access to “store” –
      2. Rather, it is giving us access to “useSelector” and “useDispatch” that are connected to the store.
      3. And thus, we can use these methods to access the store from any level.

*import* { render } *from* 'react-dom'

*import* { Provider } *from* 'react-reudux'

*import* App *from* './components/All'

*import* store *from* './store.js'

render(

<Provider *store*={store}>

<App />

</Provider>,

document.querySelector('#root')

);

1. *Usage note on useSelector and useDispatch;*
   1. *We ideally want to use these as little as possible.*
2. How do we access store values from within components?
   1. Call useSelector within a component to access the Redux store.
   2. useSelector takes a selector function as a callback, invokes it on the store, and returns the specified piece of state.
   3. Components can call it multiple times to subscribe to different parts of the store.
   4. Whenever the component rerenders, each call to useSelector will check to see if its piece of the store has been updated.
   5. if so, it runs the selector to retrieve the new data.

*//each invocation should return a \*separate\* part of the state/store.*

const counter = useSelector(store => store.counter.count)

const user = useSelector(state => state.user)