1. **Setup React-Redux.**

We need to install ‘redux’, ‘react-redux’ as well. ‘react-redux’ gives us some utilities function to work with react. **npm i redux react-redux**

Folder structure:

\*Hold all the redux related work inside **redux** folder

\*make folder for each features . Like counter is features, so make folder for **counter** folder

\*make action js file to hold all the action type of this feature. Make **actionTypes.js** and write all the action as constant variable and export them

// actions types

export const INCREMENT = "counter/increment";

export const DECREMENT = "counter/decrement";

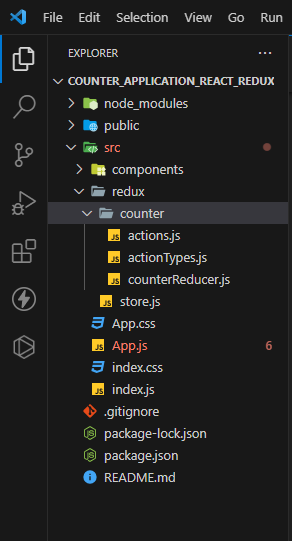
convention: action types write with features like ‘counter/increment’. Because an application can have many increment for many features. So meaning and best practice is above way.

\*make **actions.js** to hold all the action creators. Which maybe takes parameter and return an object which may contains {type:’’, payload:’’, and so on}

\*make a reducer file. **counterReducer.js** .

Lession: Mainly create reducer, actions creator, action types for each features. Mainly for each state management

We only one store, which we will put inside redux folder. Make a store for counter and export.



Now we will create a provider to wrap our application

return (

    <Provider store={store}>

        <Count totalCount={totalCount} />

    </Provider>

  );

Now store is available to the whole application. Now any component or nested component can subscribe this store.

Which part will subscribe this store, only those component will get the access to the store.

**HOC(Higher order component):** that takes argument as a function and return a new function

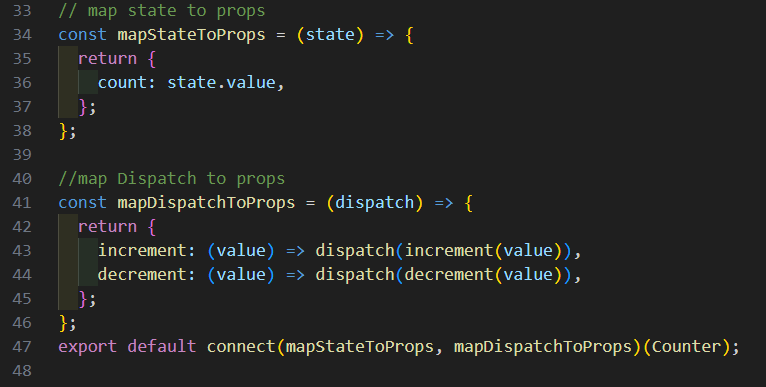
Lession: To connect store with specific component we need HOC. When we call **connect().** It give HOC. And we need to pass parameter to this HOC. The parameter will be mainly component that we want to add with store.

We need to export mainly this connect() function. This we need to two parameter with two object. One is state and other is dispatch

export default connect(mapStateToProps, mapDispatchToProps)(Counter);

which prop we pass to the mapStateToProps we can receive it into component. As well as what we pass to the mapDispatchToProps we got as parameter. We can pass props as well to the component. Also we can receive it into component parameter. Also we can receive component props inside mapStateToProps

**OLD way: to connect component and redux**



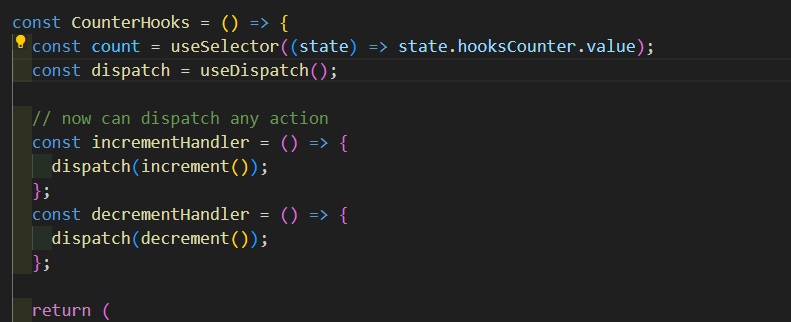
**Lession: Lecture: useSelector();**

**useSelector()** hooks: to get specific state we need this hooks. Always put this hooks at top level

this hooks mainly give us current state. it take a function as parameter. This function take a state.

**useDispatch()** hooks: this give us dispatch function.

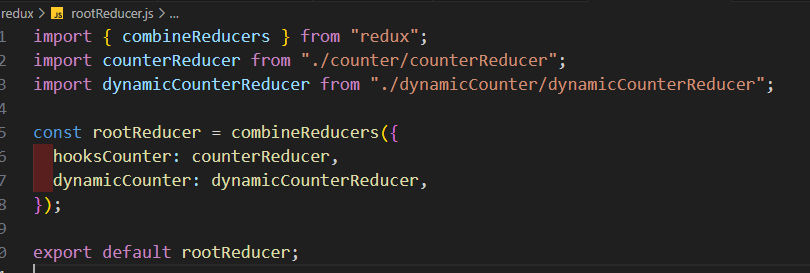
useSelector() and useDispatch() hooks



**New lesion:** We can have many reducer for many cases. But createStore() take only one parameter.

That is why need to use **combineReducers()**. It return us an object. We pass an object to it and then we put many reducer as an object property.

After that we pass the returned object into createStore() hooks that combineObject() give us.





**Note:** we can get props inside both mapStateToProps and mapDispatchToProps function. We can use the the that is passed inside components.

**Middleware concept of redux**

When an action is dispatched, it go to the reducer then reducer change the new state.

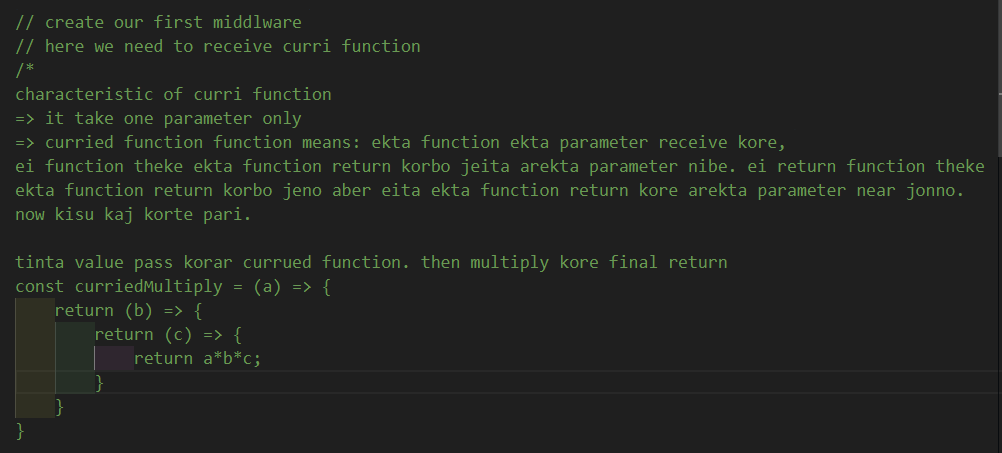
Action dispatch => reducer => new state.

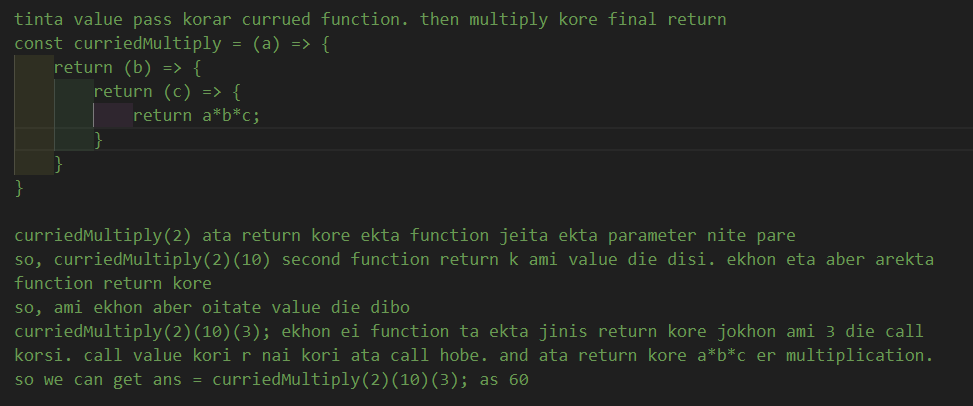
Middleware is junction before reducer and after action dispatch. We can do many thing like data fetching, user check, etc before an action is go to the reducer. Mainly middleware is an interceptor.

Action dispatch => middleware => reducer => new state

Before we see **createStore(reducer)** take a parameter. But it takes another parameter which is middleware. **applyMiddleware()** need to call inside createStore(). We need to pass middleware inside applyMiddleware() method. We can pass multiple middleware.

Each middleware should be a **curried** function





// first parameter is state, second is next, third is action. redux pass them

// state means previous state, next means when we call it, then it go to reducer otherwise it holds

// action means current action object.

const myLogger = (state) => (next) => (action) => {

  console.log(`Action: ${JSON.stringify(action)}`);

  console.log(`Before: ${JSON.stringify(store.getState())}`);

};

const store = createStore(rootReducer, applyMiddleware(myLogger));

**Note:** must return next() with action parameter. Like next(action)

**multiple middleware**

we can pass multiple middleware inside applyMiddleware() function. We can use **redux-logger** package for managing middleware. We can pass custom and package middleware as well.

// counter store

import myLogger from "./middlewares/myLogger";

import logger from "redux-logger";

const store = createStore(rootReducer, applyMiddleware(logger, myLogger));

export default store;

**Redux dev tools**

To use redux dev tools. We need to install ‘redux-devtools-extension’ npm package. It connect the redux extension with our vs code. This package give us a middleware. composeWithDevTools() function. We need to wrap applyMIddleware()

// counter store

import { applyMiddleware, createStore } from "redux";

import rootReducer from "./rootReducer";

import myLogger from "./middlewares/myLogger";

import logger from "redux-logger";

import {composeWithDevtools} from "redux-devtools-extensions"

const store = createStore(rootReducer, composeWithDevtools(applyMiddleware(logger, myLogger)));

export default store;