**REDUX:**

We want to separate the view from the data(state). It is use for state management on client side.

Redux toolkit is build upon redux, it is not redux. Redux toolkit give us the guidelines. Redux is independent of pattern, we can do any pattern.

Redux create predictable state container. The state you update at one place, it will be updated everywhere where ever it is use. Redux creates store. And all states remain in store from where, we can access them and update them

Main installation:

yarn add redux react-redux

createStore() requires reducer, so we need to create reducer to pass in it

similar to context api, which provide us with provider so we wrap all of our components under provider so they all can access all the data in context. Redux also provides provider which we use to wrap index.js components and it requires store, like context provider requires value. So this provider give access to store data to all App and child components

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { Provider } from 'react-redux';

import { store } from './store/store';

ReactDOM.render(

  <React.StrictMode>

    <Provider store={store}>

    <App />

    </Provider>

  </React.StrictMode>,

  document.getElementById('root')

);

Reducer is a pure function

useDispatch():

Reducer works on messages instead of lots of useState. So to update state or data, we need to pass the messages. Dispatch function which uses useDispatch hook is used to pass the messages

Dispatch function takes object which takes message. And this message will be pass to reducer to take the action based on the type of message and update state.

We pass default initial state(which is a object) to state, so that we can update its state as per require action.

SPREAD OPERATOR:

Reducer function takes all the values from the existing state, and change or update them. For this, we use spread operator OR you can say it copy everything from which we applied spread operator. And after copying all the things, we can add new properties or update the previous properties

Var user = {name:”Muneeb”, email:”abc@gmail.com”}

Updating user;

Var newUser = {…user, name:”Zia Khan”, age:24}

So what we does is, we copy all the properties from the user, then update name, and add age property as well and store it in newUser.

Payload:

And to send data to reducer like to send some value to reducer so that it update the previous value based on the new value, we use payload property as well

dispatch ( {type:"INCREMENT\_BY\_AMOUNT", payload: Number(value)} )

So this time, it will take some value, and update counter based on that value, so we accept that value in payload. We can also pass object to payload like payload: {name:”Muneeb”, age:”20”}

And we access the information from payload using action.payload

        case "INCREMENT\_BY\_AMOUNT":{

            return {

                ...state, counter: state.counter+action.payload

            }

Complete reducer uptill now.

// Var user = {name:”Muneeb”, email:”abc@gmail.com”}

// Updating user;

// Var newUser = {…user, name:”Zia Khan”, age:24}

// So what we does is, we copy all the properties from the user, then update name, and add age property as well and store it in newUser.

const initialState={

    counter: 0

}

export const counterReducer = (state={initialState}, action)=>{

    switch(action.type){

        case "INCREMENT":{

            return {...state, counter: state.counter+1}

        }

        case "DECREMENT":{

            return {...state, counter: state.counter-1}

        }

        case "INCREMENT\_BY\_AMOUNT":{

            return {

                ...state, counter: state.counter+action.payload

            }

        }

    }

}

Now to update state everywhere in our app, we pass this reducer in createStore hook in store.js file

And remember, we have call that store in Provider tag in index.js, so all components of app will have access to store data and updation will be performed in all over the app.

useSelector():

Now we have updated the store, but we are not using counter from store till now. To access or use values from the store, we have a hook called useSelector() that takes function and in that function, we can access values from the state

As in this case, we have counter in the state, so to use counter from state in our app, we can access it like;

    //const [counter, setCounter] = useState(0);

    const [value, setValue]= useState(0);

    const dispatch = useDispatch();

    const counter = useSelector( (state)=>{

        return state.counter

    } );

Now we don’t need useState for the counter as we are using dispatch which pass message to reducer to update the state.

import { useState } from "react";

import { useDispatch, useSelector } from 'react-redux';

function Counter() {

    //const [counter, setCounter] = useState(0);

    const [value, setValue]= useState(0);

    const dispatch = useDispatch();

    const counter = useSelector( (state)=>{

        return state.counter

    } );

  return (

    <div>

        <div>Counter:{counter}</div>

        <div>

            <button onClick= {()=>{

                //setCounter(counter+1)

                dispatch ( {type:"INCREMENT"} )

            }}>Increament</button>

            <br/>

            <button onClick= {()=>{

                //setCounter(counter-1)

                dispatch ( {type:"DECREMENT"} )

            }}>decreament</button>

            <br/>

        </div>

        <br/>

        <div>

            <input type='text' onChange={(e)=>{

                setValue(e.target.value)

            }}/>

            <br/>

            <button onClick= {()=>{

                //setCounter(counter+Number(value))

                dispatch ( {type:"INCREMENT\_BY\_AMOUNT", payload: Number(value)} )

                }}>Increament by value</button>

        </div>

    </div>

  );

}

export default Counter;

Conclusion:

1. First set provider in index.js

    <Provider store={store}>

    <App />

    </Provider>

Their will be a single store, that work as a central part consisting of all the states using reducer.

1. Create directory store, and file store.js in it. We use createStore hook and pass reducer to it. It contains all the data to update, actions to performed on the state based on the messages coming from the dispatch.
   1. Rule#1: reducer function should be a pure function. Means it does not do any kind of side effects.
   2. Rule#2: default should be their which consist of state. So initially it does not give us the null.
2. When we are going to update the store, we use useDispatch function naming it as dispatch, and when we use value or state, we use useSelector();
3. We call dispatch and pass the object with message which we want to send to reducer, so that reducer take action based on that message. If we also want to send data with dispatch, we use payload and we can use it reducer with action.payload.
4. useSelector is used to update UI by getting data from the store.