7/03/24

Episode - 12

Let's Build our Store (Redux)

→ Disclamer : Redux is not mandatory in your application

→ Redux is mandatory when you are making a very big applicatio

→ Redux is a seperate library if React is seperate.

→ Redux is not the only library which is used for data/ State managemen

for example zustand

→ ★ Redux is primarly used for handling data/ States of our applicatio

→ ★ When we use redux then our debugging becomes easy.

→ Redux can also be used with other libraries & frameworks.

→ Redux offers us 2 libraries

- React-Redux
- ★ Redux-Toolkit (newer way of writing redux).
  - Along with react-redux.

★ React-Redux :- It is basically used to connect react with redux

★ Redux : It is a standalone state management library for Js app

★ Redux-Toolkit : Newer way of writing redux which enhances the development experience with Redux by providing additional tools.

↓

Standard way of writing redux.

Earlier we used to write redux with vanila redux / normal tradition redux.

↓

It was more complicated.

→ In vanila redux we had these 3 major problems which is now solved into ~~redux~~ - ~~too~~ redux toolkit.

① It was very complicated. → Now less complicated with RTK

② Needed to add allot of packages to get redux to do anything

③ It used to ~~redux~~ require too much boilerplate code.
↓ Now the code is short and sweet with RTK (Redux Toolkit)

★ We will learn RTK while building the cart feature in our app.

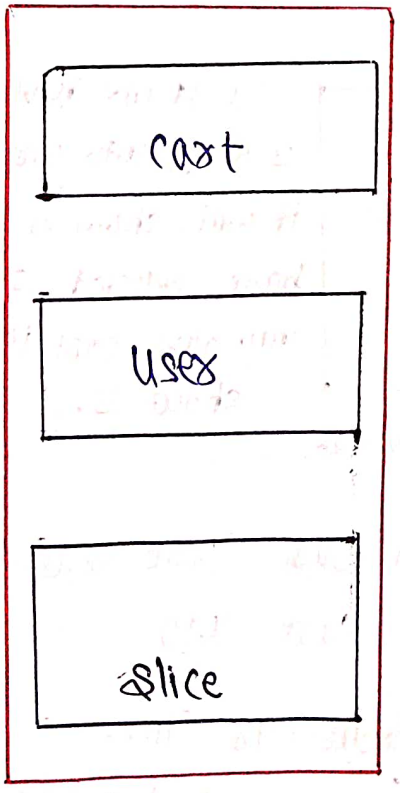
★ Redux Theory

→ What is a redux Store?

↳ Assume it as a very big JS object which is kept as in a global central place. (Any comp. can access that object)

→ Is it good to keep all data into redux Store?

↳ Yes, It is absolutely fine, our redux Store doesnot become clumsy we have slices in redux Store.



→ Slice is a small portion of our redux Store.

→ Why do we need Slices? and what are Slices?

Ans Logical Seperation of data are Slices.

for example:

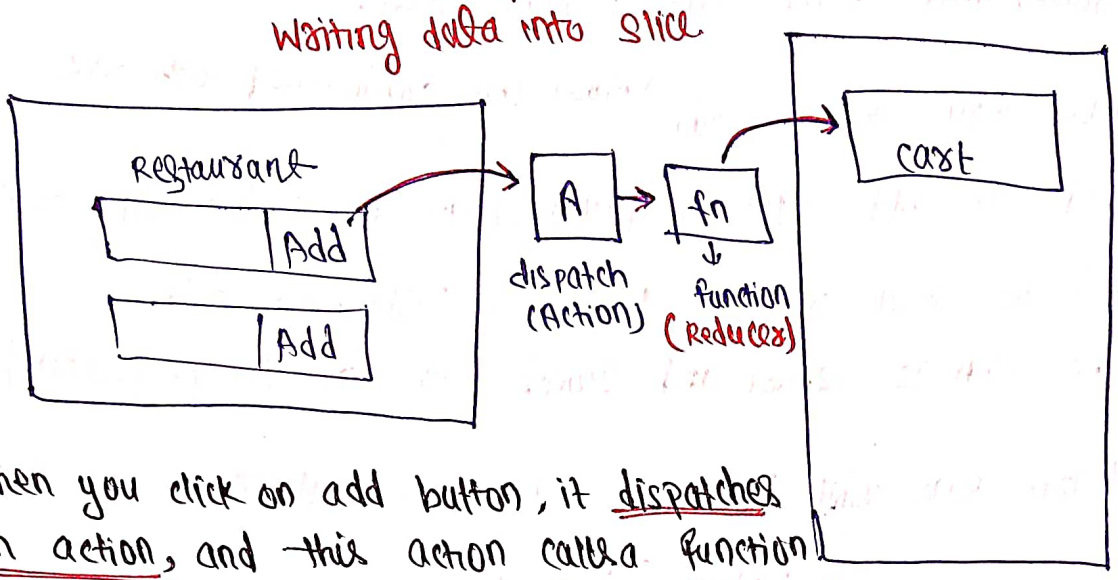
① If we want to store cart data into our store we can create a cart slice

② If we want to store the logged in user data we can create a loginslice

③ If we ~~want~~ want to have different themes (dark/white) we can create a slice related to theme data like colors etc.

→ We cannot directly modify our slice. There is a way for it.

Writing data into slice



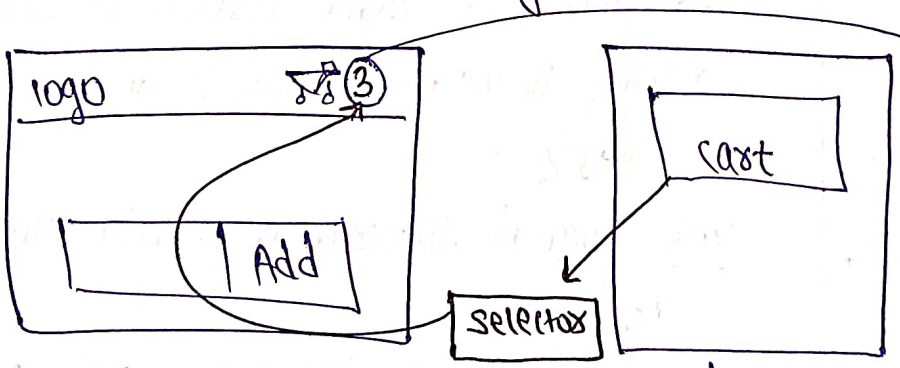
→ when you click on add button, it dispatches an action, and this action calls a function and it internally modifies the cart slice. Redux Store

→ That function is known as reducer function

→ When you click on add, it dispatches an action which calls a ~~dispatches an action which calls a~~ dispatches function which updates slice of our redux store.

→ Now how can we read data from slice?

↳ for this we use something known as Selector.



→ If we have added 3 items into our cart it will show 3, if we have selected 5 items into our cart it will show 5.

(Subscribing the store)

→ ~~Suppo~~ We want to extract data from our cart slice & show the number of cart items on cart logo.

→ for this we need to get/read data from our slice.

→ for this we will use selector. (To read the data from store & modify ~~text~~ cart number on header)

→ The phenomina of using selector is known as Subscribing to the Store.

→ After this when our cart number changes on header, as data changes in slice. Then we say that our Header component ~~cart~~ is Subscribed to our Store.

→ This means that our header compononent is now ~~with~~ is insinc with redux store.

conclusion: When we click on add button, it dispatches an action, that action calls a reducer function which modifies or updates our slice. Now as our header componont is Subscribed to our ~~cart~~ Redux store ~~Slice~~, ~~the~~ using a selector, our Number of cart items will increase.

* For mid & Small Size project we can use context, we don't even need redux for it.

→ Even in our big / scalable projects we can use context, but right now redux is in trend in market.

→ With context we have to create Sepexate context for Sepexate data. like for cart we can create cartContext, for use data we can create user context. But for doing this in redux, we can have a global store and inside that Store we can have different Slices.

→ Why we use redux is big projects, because redux is Scalable. as compare to context

→ Redux also provide Some extra stuffs. (middlewares, thunks etc

→ Steps :-

Redux toolkit name ↓ | It is just a normal js library name nothing else

1.) Intall ~~React toolk~~ @reduxjs/toolkit & react-redux.
2.) Build our Store.
3.) Connect our Store to our app.
4.) dispatch (action)
5.) Selector.

{ Earlier to work with redux, we used to intall allot of different packages to make redux powerful. }

09/03/24

Steps to use Redux toolkit :-

① Install react-redux & @reduxjs/toolkit.

② Build a Store.

```
import cartReducer from "./cartSlice";   → (2)
import {configureStore} from "@reduxjs/toolkit";

const myStore = configureStore({
    reducer : {
        cart : cartReducer,
    },
});
```

one big reducer for the whole app

It will store all the slices present in the Store.

It is the reducer of cart slice

It is the name of the slice present into our Store.

③ Now to mention slices and their reducers, we need to create them first 😊

```
import {createSlice} from "@reduxjs/toolkit";

const cartSlice = createSlice({
    name : "cart",
```

→ Name of our cart slice ✗.

```
initialState : {
    items : [],
},
```

It will store the initial state or initial data into our store's slice

→ Initially we are just storing an array (empty array).

```
reducers : {
    addItems : (state, action) => { state.items.push(action.payload) },
    removeItems : (state) => { state.items.pop() },
    clearItems : (state) => { state.items.length = 0 },
},
```

It will have 's', because it have multiple reducers.

This are our reducer functions

These are our actions

```
});
export const { addItems, removeItems, clearItems } = cartSlice.actions;
export default cartSlice.reducers;
```

from here we are doing named export of all of our different actions.

Not reducers x 2

→ We are exporting the reducer of our slice which will be received at our store, at line ④

(reading the data) ?

⑤ How to use this data

↳ `import {useSelector} from "react-redux";`

`const data = useSelector((store) => store.items);`

why store? .cart

{store → slice → your item}

↑ here we will get our data. / Read operation

4 ⑥ How to provide our store to all of our components.

In your app.js (root component)

→ `import mystore from "./mystore";`

→ `import {Provider} from "react-redux";`

```
<Provider store={mystore}>
    <Header/>
    :
</Provider>
```

my store

{Before using data, Firstly provide your store to the app}

⑥ How to do write operation ? → using dispatch

click → dispatch action → reducer func → change in store

→ import {useDispatch} from "react-redux";
import {addItems} from "./cartSlice";
const dispatch = useDispatch();

const onclickHandler = (items) => { dispatch(

returning is optional.

return (dispatch (addItems(items));

}

OR

data to store

const onclickHandler = (items) => dispatch (addItems(items));

Name of the action → you need to import it.

action's reducers

→ In addItems function why we have written action.payload?

In state we will get our initial value present in our store

addItems : (state, action) => state.items.push(action.payload);

When we do dispatch(addItems(items));

Then action looks like action: {

payload: items

};

Basically redux creates a payload key & put items at its value and provide this object to action parameter.

→ Mistake not to do while using useSelector().

If you are writing

```
const data = useSelector((store) => store.cart.item);
```

Then you are only subscribing to item which is inside cart, which is inside store. So your "data" will change only if there is a change in item.

If you write

```
const item = useSelector((store) => store);
const {data} = item.cart.store;
```

Then you are subscribing to the whole store, now suppose you have more than one slice into your store. (let's say loginuser) & if their is any change into userlogin slice then also our "data" will get updated. Because you have subscribed to the whole store. So subscribing to the whole store is not efficient at all.

→ Vanilla / older Redux vs Redux toolkit.

Write some action & reducer function

→ addItems : (state, action) => { state.item.push(item) }

action.payload

In above reducer function we are directly mutating / updating the state, and also we are not ~~updating~~ returning anything.

→ But in vanilla / older redux we could not directly mutate / update our state & also we need to return the new updated state.

Vanilla / older redux code for the same addItems

```
addItems: (state, action) => {
    const newState = [...state];
    newState.items.push(action.payload);
    return (newState);
};
```

→ This newState will replace the original state.

→ In redux-toolkit behind the scenes uses the ~~same~~ same above process and for doing this it uses immer js library.

→ immer js takes the previous / original store & makes a copy of the updated store, compare them & provide the difference between them.

→ While using immerjs we are not updating / mutating the original store, we are updating the draft or the copy of original store.

→ After completing the changes immerjs will update the original store

→ In removeItems we can't do this.

```
removeItems: (state) => { state = []; }
```

because here state is a local variable, what ever changes we are making changes into our local variable (draft / copy of original) Therefore it will not work if we need to do

```
state.cart.item.length = 0;
```

other way of doing it

```
removeItems: (state) => { return ({ item: [] }) };
```

★ This object will replace the previous state.

→ You can also use current to see the current state

```
removeItems: (state) => {
    console.log(state); ——→ This will not show you the current state property. properly.
    console.log(current(state));
}
```

→ Redux also helps in debugging with its redux devtool. So also use it.