(lecture - 12)

(let's build our store) Date.....

- We are going to build our own "store" (Redux) we are going to study how we ~~are going to do~~ can manage state of our application using Redux & how we can manage data of our application using Redux
- Redux works in the Data layer.
- UI & Data layer works in sync and build our own React app.
- Redux is not mandatory.
- When we are building small application or even a mid sized application, ~~Redux~~ we don't need redux.
- React and Redux are different libraries.
- Redux is not the only library that is used for managing states.
- other library - Zustand
- Redux offers state management & offers easy debugging
- It is not something which is tied to React, it's not just use along with React but it is popularly used with React that's why a lot of people think that React & Redux are or in the same thing.

Scanned with CamScanner

Date.....

- Redux offers state management and it does not need to be a React application itself, it works with other libraries and frameworks as well but it is heavily used with React.

- There are 2 libraries that Redux teams offer → (1) React-Redux - It is a kind of bridge betⁿ React & Redux

(2) Redux Toolkit - It is a newer way of writing Redux.

- We won't be using Vanila Redux (older way of writing Redux), we will be using Redux Toolkit & React-Redux.

— Redux Toolkit → Earlier we used to write it in a different way, now we used to write in a different way i.e. Redux Toolkit, it's now a standard way of writing Redux logic.

— It was originally created to address three common concerns

(1) Configuring a Redux store is too complicated — Redux was too complicated, Redux has the a huge initial learning curve, we have to learn a lot of things

Earlier while writing Vanilla Redux it was even more complicated, Redux toolkit made it little more easier, it is still complicated

Spiral

12:08

Date.....

But it is much more easier than what it is used to be.

Redux Toolkit offers a very less complicated version of Redux.

(2) we had to add a lot of packages to get Redux to do anything useful.

(3) Redux requires too much boilerplate code

Redux Toolkit →

- Redux store is kind of like a very big JS object with a lot of data inside it — and it is kept in a global central place.
- In our react application any component can access this redux store, it can write/read data from that store.
- And we keep most of the major data of our application into this redux store.
- Is it a good practice to keep all the data inside the whole big object / in the one big whole State Object ?

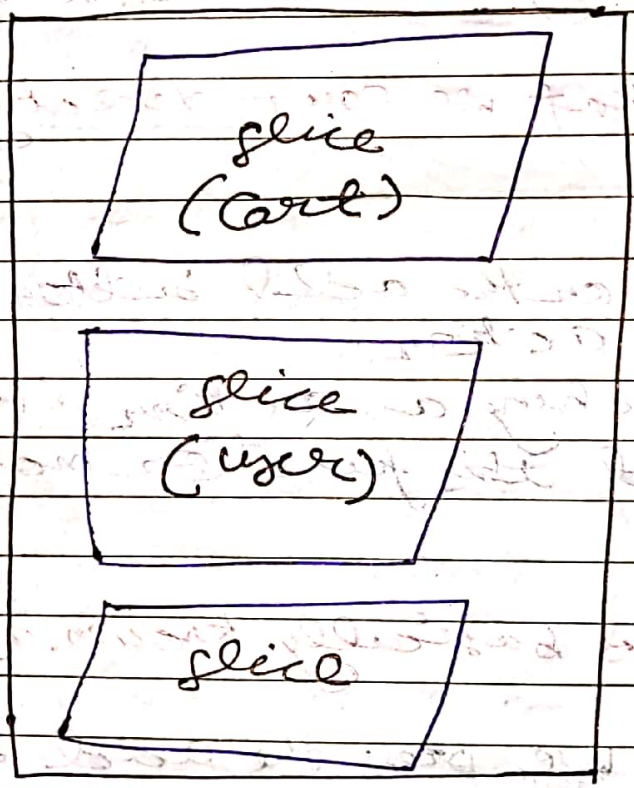
Yes, it is absolutely fine, there is no

Spiral

Date.....

problem with keeping a very big object with a lot of data inside it, it is completely fine but so that our Redux store doesn't become very big, very clumsy we have something known as slices inside our redux store.

- slices → Small portion of our Redux store.



- To keep data separate we make logical partition and these logical partition are slices.
- Suppose if we want to add cart data into our redux store we will create a separate slice, also like - logged in user info, user slice

Spiral

31:09

Date.....

- We will create a logical separation and we will make a small pieces inside our redux store.

- Initial cart slice can be empty array but later on as we put data inside it, it just modify the cart slice.

- We will click on the add button of item we can't directly add data to the cart slice.
Redux says that we can't directly modify the cart slice.

- If we click on the add button we have to dispatch an action.
after dispatching an action it calls a function and this function modifies the cart.

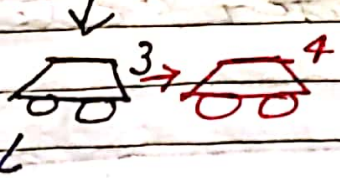
- This function is basically known as a Reducer

- ~~We~~ When we press the add button, it dispatches an action which calls the reducer function which updates the slice of our redux store.
We have written the data inside it.

- Suppose we want to read the data; for that we use something known as selector.

Date.....

- We will use a selector to read the data from our store and the selector will modify our react component



selector will give the data over here.

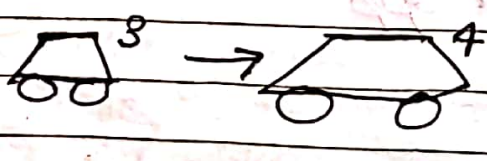
- When we use selector this phenomenon is known as subscribing to the store.

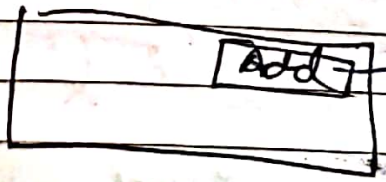
- So we say that the header component is subscribe to a store and when we say subscribe the store basically it is in sync with the store.

If the data inside our store changes our header component will update automatically.

Our Header component has subscribe the store.

How do we subscribe ? → using a selector

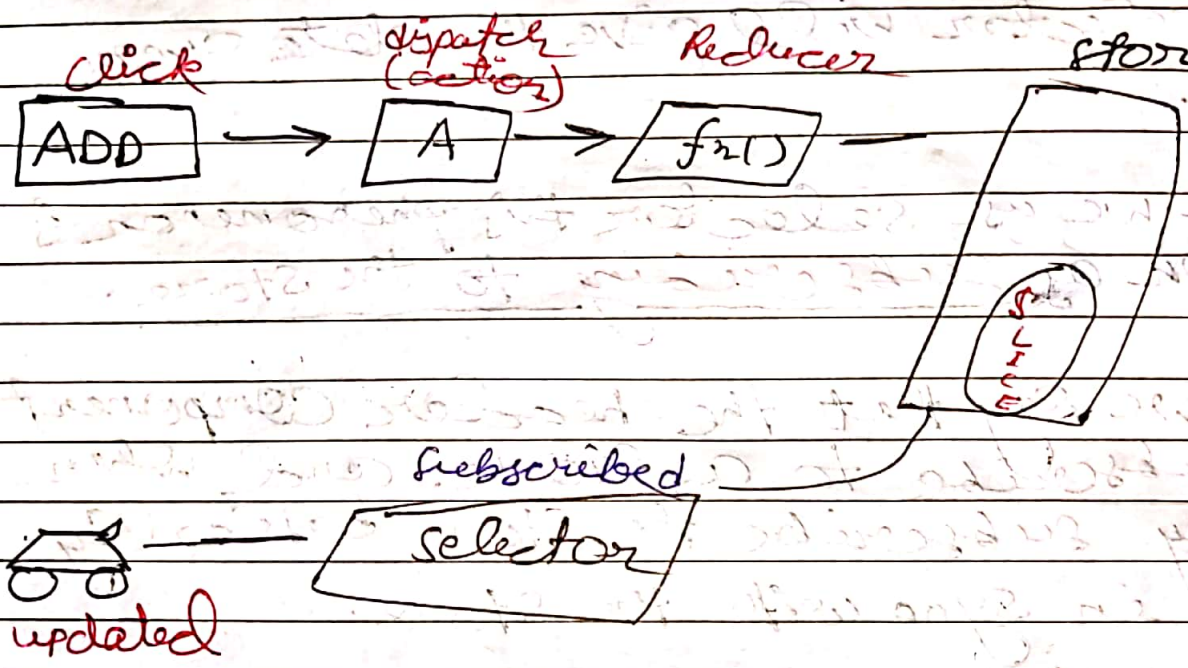




→ By clicking the add button the data on upper side changes.

Date.....

- As the header component is subscribed the store, data is updated on slice by reducer function which is called by dispatch action.



Redux Toolkit

- Install @reduxjs/toolkit and react-redux
- Build our store
- Connect our store to our app
- slice (cartSlice)
- dispatch(action)
- selector

- configureStore is coming from redux toolkit & Provider is coming from react-redux because providing store to the app is react-redux job which is a bridge bet redux & react app.

Date.....

- we will build our store in utils

appstore.js →

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

const appStore = configureStore ({

});

export default appstore;
```

App.js →

```
import { Provider } from "react-redux";
import appStore from "./utils/appstore";
return (
                    ↓ prop
<Provider store = {appStore} >

<UserContext.Provider value = {{ loggedInUser:
UserName, setUserName }} >

<div className = "app" >

<Header/>

<Outlet />

</UserContext.Provider>

</Provider>

) ;
```

Spiral

• cartSlice is a function, function takes a configuration to create a slice.

Date.....

• whenever we need to use store we have to wrap everything in provider, just like the context Provider

---

Utils → cartSlice.js

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

const cartSlice = createSlice({
  name: 'cart',
  initialState: {
    items: []
  },
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
    },
    removeItem: (state, action) => {
      state.items.pop();
    },
  },
```

It get access to the state (initial state) and action ↑ action

we can avoid writing action

Date.....

```
- clearCart : (state, action) => {
    state.items.length = 0;
  },
},
});
export const {addItem, removeItem, clearCart} = cartSlice.actions;
export default cartSlice.reducer;
```

• cartSlice will be kind of like a big object which has actions and reducer

→ {
  actions : {
    addItem
  },
  reducer
}

→ Reducers is basically an object, this object has different kind of actions that we can take like addItem, removeItem, clearCart and for each corresponding this we will have the reducer fn to update this state

Date.....

map to an action

appstore.js →

- we create a reducer in configureStore for adding slice.
- If we want to modify our store it has also a reducer for itself and that reducer combines the reducers of ~~that~~ the slices.
- This basically reducer is responsible to modify the appstore, it is a combination of different small stores

```
import { configureStore } from "@reduxjs/toolkit"
import cartReducer from "./cartSlice";

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

export default appStore;
```

Spiral

Date.....

- Basically we will use a selector, a selector is nothing but a hook inside react. This hook gives us the access to the store.

Header.js →

import { useSelector } from "react-redux";

// subscribing the store using a selector

const cartItems = useSelector((store) => store.cart.items);

- This selector basically helps us identify what portion of our store we need to read (subscribe).

<li className="px-4 font-bold text-xl">
Cart ({cartItems.length} items)

itemList.js →

<button
onClick={() => {handleAddItem}
// dispatch action

: import {useDispatch} from "react-redux";
import {addItem} from "../utils/cartSlice";

Date.....

const handleAddItem = () => {
// dispatch an action
const dispatch = useDispatch();

. const handleAddItem = () => {
dispatch(addItems("pizza"));
}

→ This is our action.payload.
Whatever we will pass here this will go to our reducer function action.

→ Whenever we are doing a selector make sure we are subscribing to the right portion to the store

const cartItems = useSelector((store) =>
store.cart.items);

→ If we don't subscribe to the right portion of the store it will be a big performance loss.

Spiral

1:53:16

Date.....

• we can do like this also.

```
const store = useSelector((store) => store);
const cartItems = store.cart.items;
```

→ now we are subscribing to whole store, earlier we were subscribing to small portion of the store and ~~extracting~~

→ we are subscribing to whole store and extracting the items.

→ But this is very less efficient

→ we don't want updates of whole store. to subscribe to

→ suppose if something is happening in another slice, it is nothing to do with cart.

→ when our application grows huge our store becomes very big so anything random changes in our store we don't want our component is affected by it.

→ we don't want to subscribe our whole store, a better performance way is to only subscribe to the specific portion of store.

• The name is selector because we are selecting a portion of store.

Spiral

Date.....

- When we create AppStore. Here the keyword is Reducer, because this is one big reducer and this reducer can have multiple reducers
- But when we are writing slice, we create multiple Reducers

export default cartSlice.reducer
→ we are exporting just a single reducer from it

- AppStore is a combination of small reducers of the slices and
  cartSlice reducer is a combination of the small reducer functions.

In Vanilla (older) Redux ⇒ Don't Mutate State
- returning was mandatory

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

Spiral

Date.....

• Now in Redux Toolkit We Have To Mutate The State

```
state.items.push(action.payload);
},
```

→ Behind the scenes Redux is creating immutable State (older redux logic), redux still doing like older redux but it is not asking developer to do.

→ Redux uses the Immer library to do this, basically immer library is kind of like finding the difference bet" the original state & mutated state and then it gives the new state which is a immutable state, a new copy of the state.

→ State is a local variable

```
//originalState = ["pizza"]
clearCart: (state, action) => {
  console.log(state); // [pizza]
  state = [];
  console.log(state) // []
```

→ It is not actually modifying the original State.

Spiral

Date.....

- A Redux Toolkit says we have to either mutate the state or return a new state existing

either,

state.items.length = 0 ; original state = []

or

return {items: []};

→ Redux Dev Tools — Chrome extension