Episode 12 - Lets build our Store -

We have two layers in our frontend / react app,
→ UI layer - the dom part, jsx.
→ Data Layer - variables, states, data.

We want, UI layer to be sync with Data layer, they work together in sync in a react application.

Redux - It is a state management tool, it works in the data layer, and is used in big-big applications as alternative to Context API.

→ Redux is mandatory for all react applications, use it only in very very big applications.

→ There are other state management library like zustand.
→ Redux is separate library from React, they are different.

Advantages of Redux -
① → In large applications, redux offers, handeling data, managing store.
② → Application becomes easier to debug. using Redux Dev Tools.

Three things Redux team offers to us.
→ Redux (vanilla Redux) — old way of writing Redux.
→ React - Redux - Acts as bridge b/w React & Redux
→ Redux Toolkit (RTK) - Standard new way of writing Redux

Currently Used.

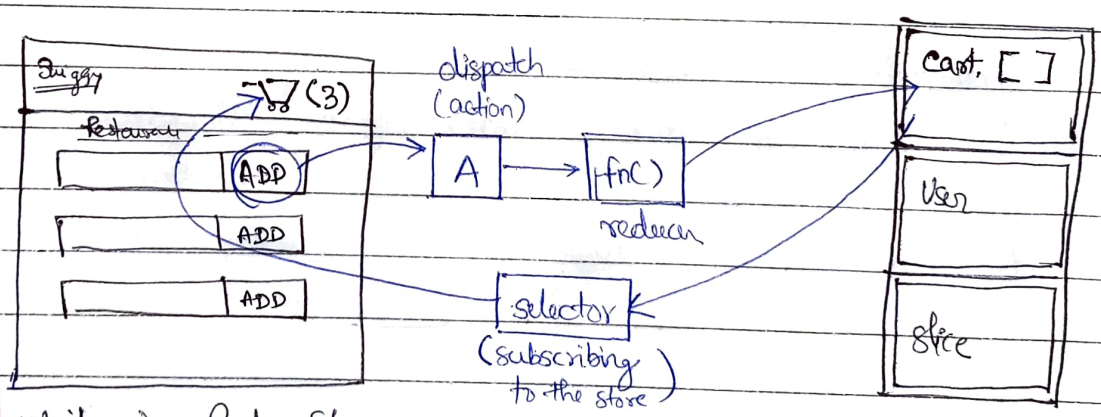
Gargon - So now when people say Redux they means RTK.

Redux Store

Redux Store is a big js object, with lot of data inside it, kept in a global central page, and any component of our app can access it, and can read and write data in it.

And we keep most of the major data of our app inside this redux store.

Is it good to store all the data into this big object?
Yes its completely fine. But so that our redux store does not become very big & clumsy we have Slices in our Redux store.



→ Write in Redux Store

We cannot directly update something in redux store.
When we press Add button, it dispatches an action, which calls the reducer function which updates the slice of our ReduxStore.

→ Read in reduxStore.

We use "Selector", and this selector will give us data. When we use selector, this phenomena is known as "subscribing to the store".

So we say that header component is subscribed to our store means they our Header is in sync with the store: i.e if data in my store changes, the Header component will update automagically.

★ Using a selector we/Component can subscribe to the store.

★ If we/ are subscribed to the store, our component will update automagically/Component with change in store.. Means Component is in sync with that store/slice.

Steps –

- Install @reduxjs/toolkit and react-redux.
- Build our Store
- Connect our store to our app.
- Slice
- dispatch (action)
- Selector.

i) Building Store – Create appStore.js file in utils

```
import { configureStore } from "@reduxjs/toolkit";
const appStore = configureStore();
export default appStore;
```

Now we need to provide this store to our application. In our main component where we define routes, provide the appstore to the outer most component i.e wrap it with Provider.

* You can also provide store to a specific component also.

ii) Creating Provider - App.js.

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

let App = () => {
    ---
    return (
      <Provider store={appStore}>
        <UserContext.Provider value={---}>
          <div class="App">
            ---
          </div>
        </UserContext.Provider>
      </Provider>
```

iii) Creating Card Slice - 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) => {
            state.items.pop();
        },
```

reducer func

actions

// mutating the state

```
        clearCard : (state) => {
            state.items.length = 0; },
    },
});

export const { addItem, removeItem, clearItem } = cartSlice.actions;
export default createSlice.reducers;
```

iv) Now we have to add this slice to our store

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

const appStore = configureStore({
    reducer: {
        cartReducer,
        userReducer,
    },
});
export default appStore;
```

So for appStore also we have a big reducer, that contains small reducers of the slices present in appStore.

v) Reading the data from cart slice.

We will use the selector i.e useSelector hook from "react-redux" library. We can use this anywhere in our app where we provided appStore.

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

// this is subscribing to the store using selector.
const cart = useSelector((store) => store.cart.items);
```

vi) Writing in a Slice.

So for this we have to dispatch an action, and we have "useDispatch" hook for this.

```
import { useDispatch } from "react-redux";
- - -
- -

const ItemList = () => {
  - - - - -
  const dispatch = useDispatch();
  - - - - -
  dispatch(addItem("pizza"));
  - - -
}
```

this will go to cart reducer addItem as action = { payload: "pizza", }

in second argument.

Miss Conceptions.

⭐⭐→ In reading the slice we actuall don't read anything but we only just subscribe to the store so that when slice's state change, they will automaticall rerender the subscribed components and display according to new slice's state. We don't have to manually update the subscribed components with usestate or something.

→ While subscribing to the store, we can subscribe to the whole store, but it is a bad practice, because any small change in our store in any slice will trigger the rerender of the components subscribed to whole store.

So we have to use useSelector wisely to subscribe just the required / specific portion of store or slice.

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

bad way, not efficient

```
const cartItems = useSelector ((store) => {
    return store.cart.items; });
// good way.
```

→ reducer — big reducer — app reducer — Combination of small reducers.

reducers — many reducers of a slice — Combination of diff. functions.
functions.

→ We have to mutate the state, ~~otherwise~~ i.e change the original state, because the state variable is local variable, so we just directly modify state, and immer will handle mutating.

RTK says - either Mutate the existing state

OR

Return a new state.

```
originalState = { items: [ "pizza" ] };

state.items.length = 0 ;   // Means mutating the state
                           // originalState = []

return {items: []};  // dont do nothig with state,
                     // just return a new state.
                     // [] will be replaced inside OriginalState
                     =[]
```

★ Redux Dev Tools.

★ RTK Query of Redux Toolkit - ~~was~~ is replacement of old .vanil js's "redux middlewares & thunks." for making api requests.