20 JULY MONDAY

JUN 2020

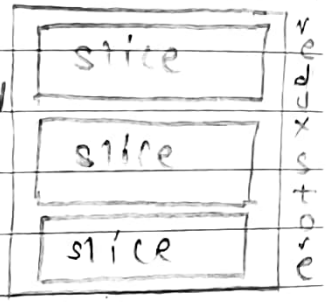
M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

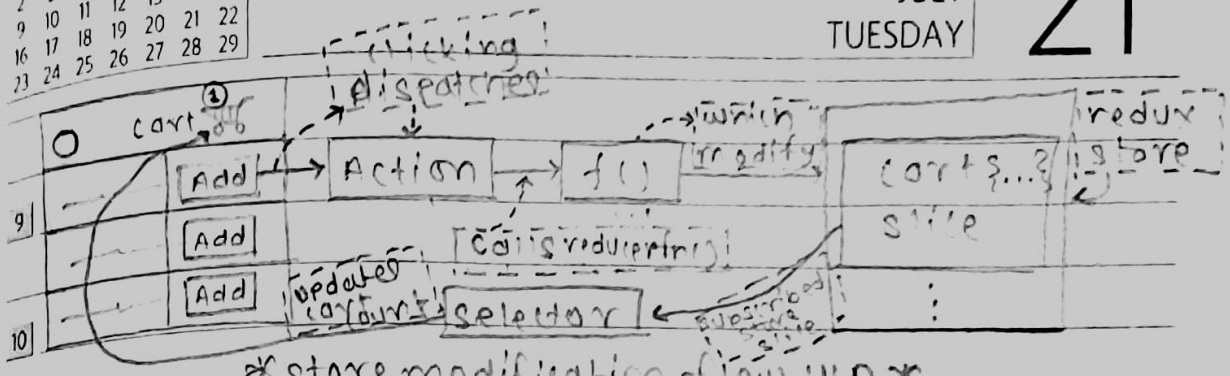
* EP12 ⇒ Redux :

- When we click "add" btn on any item on app; it gets added to cart [using redux we can handle this action].

+ Redux store :

- Massive JS object kept in central place. Any comp. can access / R / W to this object.
- Major app. data is kept here.
- keeping such massive amt. of data in a single object is fine; but @ the same time it can become complex to store & retrelve this data.
- Hence "slices" are used in redux store.
- slices are part of redux store; which is small portion. To keep data separate; some lugical partitions are created known as slices.
- E.g for cart data; separate slice'll be created. Also to store data for logged in user; separate slice is created.
- ∴ slices : user / cart / theme etc.
- We can not modify slice directly. To modify cart (slice); When we click on "add" btn; an "action" is dispatched. Which in-turns call a function which modifies "cart" slice.
- The function called after dispatching an action; is known as "reducer".



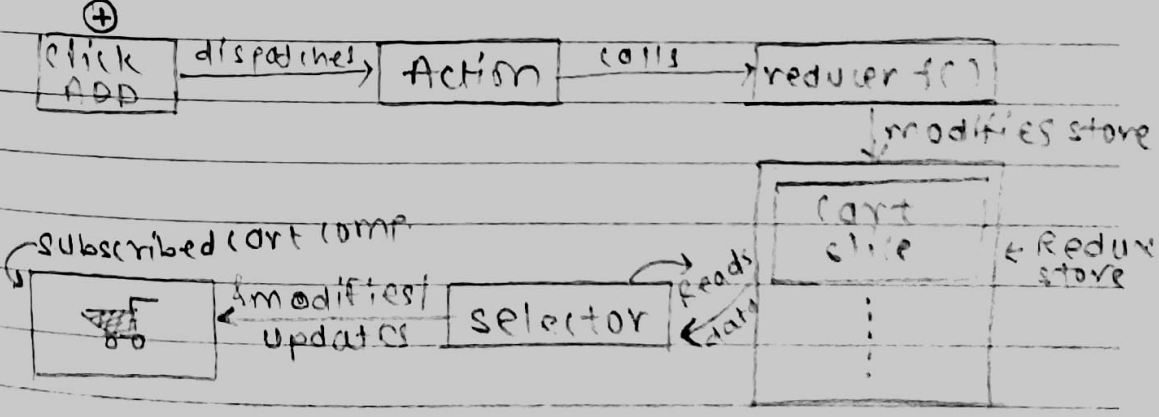


* store modification flow HLD *

∴ When we click "add", it dispatches an action; which in turn calls "reducer" function which modifies "cart slice" of a redux store.

- A 'selector' is used to read data from a redux store slice to update UI layer,
e.g We need to read data from "cart slice" of store to update count of items in cart to show in real time to user.

* When we use "selector"; the particular comp (which needs to be updated by reading store slice) is "subscribed" to store. i.e. that comp. is in sync with store; in terms When store is updated; that comp'll also be modified / updated



*Reducer can modify app. bcz it can access "initialState" which is configured in "createSlice()".

22 JULY WEDNESDAY

∴ reducerEg : (state, action) => {... }
(state: ~~Init: action~~ state)
→ "ll modify "state" based on "action".

JUN 2020						
M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

(19)

*Creating store:

9 - configureStore() is used to create store which is part of RTK. slices are added into this
10 method.
- After creation; it's provided to "root" of an "app".
11 [where div w/ className "root" or "app"].
The "provider" is part of "react-redux" library.
12 bcz it's react specific & not part of RTK.
- "store" that we've created is passed as an
1 argument to "provider":

<provider store={...}> → appStore.
2 <div className="app">
...
3 </div>
</provider>

4 - If only small portion of app needs to use store; only that part is wrapped in "provider".
5 - To create a "slice" into store; "createSlice()" is used [by RTK].
6 "createSlice()" takes configuration to create a slice as: {}:

{ ① name: "name of slice",
② initialState: {what this slice'll contain & what that will be initially", }
③ reducers: { reducer funcs related to actions". }

(20)

※ Reducer can modify app. b/c it has an access to "state" i.e. "initialState" declared in config. of "createSlice()".

∴ structure of reducer'll be:

```
reducers: {
    reducerNm: (state, action) => {
        // operⁿ [state.items.push(action.payload)];
    }
}
```

∵ Reducer'll modify "state" based on "action".

- structure of reducers for cart:

```
reducers: {
    addItem: (state, action) => {
        state.items.push(action.payload);
    }
}
```

∴ When we click "add", we dispatch an "action" which invokes "addItem" reducer. We get "payload" (means info. of to be added in cart).

We push it into "items" array w/ is initially empty.

+ We need to export actions & reducers from slice mandatorily.

Syntax for export:

- export const { } = cartSlice.actions;
- export default cartSlice.reducers;

- export const { action1, action2, ... } = slice.actions

this syntax helps to individually export actions defined in "reducers" of "createSlice()".

"reducer" object passed to "configureStore()" works as a big reducer (main one) of whole app.

reducers exported from slices are passed to it as small/sub-reducers.

- Passing reducers to "reducer" (single/multiple):

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

i.e. each slice's reducer is added here.

* Mutation of state occures, when we create "actions" in "reducers" property of "createSlice()".

Mutation = directly modifying state.

*selector:

- Used to subscribe the store.

In react-redux, we get "useSelector" hook which is used to access store.

i.e. we are subscribing to store using selector

- in this hook; we can specify which portion of store we want to subscribe; which will be again a smaller part of "slice" which will be holding the related data.
e.g for cart; we need only "store.cart" to access. & we won't access/subscribe to store.user or store.theme in cart logic.
∴ hook is written as:
const cart = useSelector((store) => store.cart.items);
- useSelector() is basically used to access state of a redux store. It takes "selector" function as an argument & called w/ store state.

* Dispatching an action:
- Just like "useSelector()" is available to subscribe to store & read data; "useDispatch()" is used to dispatch an action.
+ construction of useDispatch():
- const dispatch = useDispatch();
//...
dispatch(addItem("cart"));
- "dispatch" will work as a function which takes an argument of an action exported from "createSlice()".
→ So now when we click "add"; an action is dispatched by "dispatch" w/ name "addItem()" which will modify "cart" slice.

whatever we pass here is given as an argument to reducer as an action which will be action.payload.
addItem: (state, action)

27 JULY MONDAY

- When we "dispatch", redux constructs an object like: { payload: "pizza", } → not simple like this much complex in real life [this is ex.]
when we "dispatch" an action of addItem as:
dispatch(addItem("pizza"));.
- Redux takes this constructed object & pass as a 2nd argument (action) to reducer in createSlice:

```
const cartSlice = createSlice({
  //...
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
    },
    //...
  },
});
```

object { ... payload: "pizza", ... } → action.payload ∴ we'll get "pizza" here.

* Difference bet^n following 3 calls:

```
<button
  ① onClick={handleAddItem}
  ② onClick={() => handleAddItem(item)}
  ③ onClick={handleAddItem(item)}
></button>
```

③ → Is immediately invoking the function
② → Is callback; it'll be invoked on some trigger or action.
① → callback w/o arguments. If we want to pass arguments to callback, need to use way ② of passing handler.

* Imp things abt redux:

(1) While using "useSelector()"; make sure to subscribe to right portion of store; otherwise it'll ~~case~~ cause performance issues.

Eg (1) const cartItems = useSelector(store => store.cart.items);

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

- Both (1) & (2) is correct; but (2) is less efficient.
Bcz store contains multiple slices such as "user" to login info; "theme" for app theme etc.
- So If we subscribe to whole store; cart will be subscribed to unrelated store updates such as "user logged in" which can cause performance issues.
- When (1) is used: ~~action~~ "cart" will update only when "items" array in "cart" slice changes.

(2) Confusion between "reducer" & "reducers":
- When we are building store; "reducer" is used. bcz it's whole big & only one "reducer" for entire app. It can've multiple smaller "reducers".
- When we create "slice"; We create "reducers".

- When we export from "slice"; we export only singular reducer as:

`export default cartSlice.reducer;`

biz "reducer" is a comb^n of multiple smaller "reducers".

(3) Mutation of states:

- for RTK; immer is used BTS by redux for simplification of updation of state by directly mutating it in reducer itself.
- But in a reducer; When we try to modify "state" which is local variable; it doesn't work.

*i.e. clearCart: (state) => {
    state = [ ];
}

This does not work & cart'll ~~be~~ not be emptied biz we are modifying local variable only. [which is just creating reference by doing: state = [ ].]

* clearCart: (state) => {
    state.~~cart~~.items = length = 0;
}

When above behavior is used; we are "mutating" state & not local variable. Hence; in this way; state which is mutated will point to "this".

%% ~~return [ ];~~ works similar way.
return { items: [ ] }

%% TS quick start{ RTK queries quick start

S	M	T	W	T	F	S	AUG 2020
						1	
30	31						
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	

(26)

WK 31 • 212-154

we've to return } whole state of items

JULY THURSDAY 30

return{items: [] }

- The behavior of "~~return []~~" works b/z of strategy of redux which says:
 "Either mutate the existing state or return a new state."

(10) RTK Query ⇒ To read.

→ In older Redux; Redux Thunks were used to store data rcvd from an API call to store asynchronously. [Thunk is a middleware & used in certain design pattern].

→ In newer version; this Redux Thunk middleware design pattern is replaced w/ RTK Queries.