PAGE NO.:

⇒ RTK (Redux Toolkit)

- createSlice ?
- Configure store?
- actionCreators
- createAction
- useDispatch ?
- useSelector ?
- How to connect React with redux toolkit ?
- If we use RTK then, is we need redux ? yes, we need package for communication b/w them redux and RTK.

⇒ Redux

Redux is a pattern and library for managing and updating application state, using events called "actions". It serves as a centralized store for state that needs to be used across your entire application, with rules ensuring that the state can only be updated in a predictable fashion.

↓
which component read/update data in the redux store. It can be used while testing

- **Redux main topics :-**

(1) Action (what to do ?)
(2) Reducer (How to do ?)
(3) Store (object which holds the state of the application)
(4) Functions associated with the store like createStore(), dispatch(action) and getState().

PAGE NO.:

① Action

Actions are plain javascript objects that have a 'type' field. Actions only tell what to do, but they don't tell how to do?.

ex- return {
    type: 'Increment',
    payload: number
}

- Action creator (who creates action)

A pure function which creates an action.

ex-
```
export const incNumber = (num) => {
    return {
        type: 'Increment'
        payload: number
    }
}
```

- Action creator is reusable, portable and easy to ~~use~~ test.

② Reducer

- Reducers are functions that take the current state and action as arguments, and return a new state result.

ex- const initialState = 0;

```
const changeNumber = (state = initialState, action) =>
    {
      switch (action.type) {
          case "Increment": return state +
                                action.payload;
          case: "DECrement": return state - 1;
          default: return state;
      }
    }
```

(3) Store

- The redux store brings together the state, actions, and reducers that make up your app.
- It's important to note that you will only have a single store in a redux application.
- Every Redux store has single root reducer function.

ex- For creating redux store -

→ import {createStore} from "redux";
const store = createStore(rootReducers);

PAGE NO.:

Redux Principles :-

① Single source of truth

- The global state of your application is stored as an object inside a single store.

② State is Read-only

- The only way to change the state is to dispatch an action.

③ Immutability, one-way data flow, Predictability of outcome.

④ changes are made with pure reducer functions.

⇒ RTK

→ Slices in RTK?

A function that accepts a "slicename", an initial state, and an object of reducer functions, and automatically generates action creators and action types that correspond to the reducer and state.

- Slice is nothing but it is a part of the store.

- After creating slice, we can see actions of that slice that created automatically behind the scenes - slice.actions (console.log it)

⇒ extraReducers in RTK :-

- If we want that any micro-reducer or miniReducer of a slice can be used inside another slice then we can use the concept of "extraReducer". That is we can share a microreducers of a slice between multiple slices. ~~if the task is same related to each other~~

OR

- If action is supposed to handle by one reducer then use reducers.
- If the action is supposed to handle by multiple reducers then use "extra Reducers".

- extraReducer is referencing the actual original microReducer that we create in slice & if we ~~create~~ remove actual original microReducer then extraReducer will give us error. because it was referencing to it.

- Eg. I have 2 slices i.e, adminSlice and userSlice, ~~if I want~~ adminSlice has list of admins and userSlice has list of users. 'clearUser' reducer (micro) delete all the users and same thing we want in adminSlice where the same logic is implemented for deleting all user from adminSlice. Then we can make microReducer of userSlice is defined to be as extraReducer in the slice of user. And then we can reuse the same logic for clearing value of state in adminSlice also.
- So, it only share the logic of the code ~~not~~ & execution depends on dispatch().

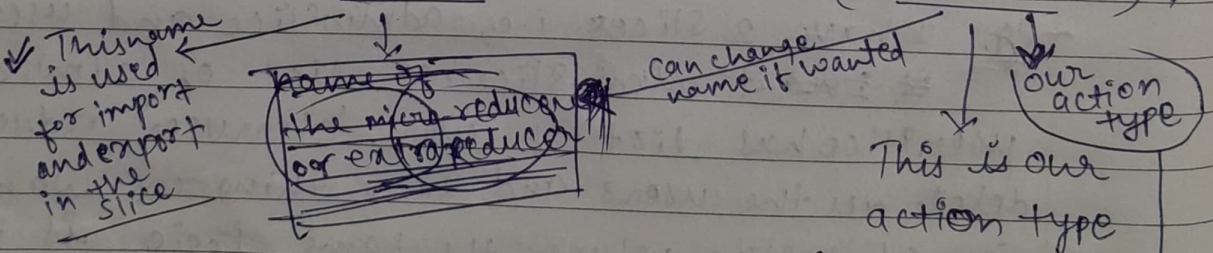
- createAction() is a action creator and action creators creates action object.

⇒ createAction() function in RTK :-

- We have seen that 'extraReducer' is helpful and increases code reusability by sharing the logic of the 'reducer' present inside a slice to other slices's reducers.
  - but if the main original reducer will be deleted or removed from the code then it will throw an error because the extraReducer is pointing to the main actual reducer of the slice that code is shared. and if the main/original reducer removed then to which code, the extraReducer will points or references.

- - For solving this problem, we have createAction() function - (for creating actions using createAction() make folder 'actions' and 'index.js' inside that folder for defining/using this function.

  - import { createAction } from "@reduxjs/toolkit".
- export const deleteUser = createAction("delete users");



- we have created action for that reducer used in extraReducer. so no worry if the main original reducer of slice removed from code.

⇒ we can import 'deleteUsers' from actions/index.js to use the dispatch() and pass this 'deleteUsers' inside the dispatch.

PAGE NO.:

- Now we can import this 'deleteUser' in the any slice where we are writing extraReducer code.

- import {deleteUser} from '.../action/index'

- extraReducer is used in object of createSlice after reducer just like another key-value pair.

- extraReducers(builder) { → no need to pass sliceName. actions. name of micro-reducer here now.
  builder.addCase(deleteUsers, () => {
    return []
  })
}

- and also now, we don't to export that particular micro-reducer from this slice like - export const {deleteUsers} = sliceName.actions
  ↓ no need to export like this

- because now our deleteUser is totally independent.

export const deleteUsers = createAction("deleteUsers")

- Note:- When we used createAction() function in the 'index' file inside 'actions' folder then when we create action object and exported it in a single line. So, the left part side is used for export & import while the argument or string passed in createAction() method, that string is used in slice file where we write code extra Reducer in addCase() function.