**Theory Chapter 12**

**REDUX: PREDICTABLE, CENTRALIZED , FLEXIBLE ,DEBUGGABLE**

1. Redux is a predictable state container for JavaScript apps.
2. Easy to test , behaves consistently on different env. (dev,qa,prod)
3. Greater developer experience in terms of debugging
4. Works with the ui layer

**Three principles of Redux**

1. Single source of truth for the whole application
2. State are read only, the only way to change the state is by dispatching actions
3. State is changed on in pure functions (Reducers)

**Q: useContext Vs Redux**

1. useContext comes with react and for redux we need to install a library from a package manager
2. Middleware is supported in redux which result in support of logging information and doing side effects in that . Context does not support middleware, but we can use useEffect to have any side effects in context
3. Debugging is easy in redux cz of redux tool kit extension as compared to context api
4. Redux is mainly used for frequently changed data , and context mainly used for more of static data.
5. Change in the value of context results in re-rendering of the whole component tree while change in the value of redux causes the re-rendering of the component which is consuming that value
6. Configuration is more complex in redux as compared to context

**Q: Advantages of using Redux toolkit over redux**

**A: Redux toolkit** speeds up the development process

Toolkit was basically designed to solve the following problems

1. Configuration of redux store is more complicated
2. Need to add more packages so that one can use redux to its full potential
3. Requires too much boilerplate code

**Q. Explain dispatcher , reducer , slice , selector**

**A: NOTE: When a user clicks on button it dispatches an action which carry the updated state which calls a reducer function which in turn modifies the state in a slice of a store and we can subscribe to a slice of store to component by using useSelector.**

**STATE:** In the redux api state is the data which is managed by redux and used by the application . It is the complex object. You can put anything in the state which can be converted in the json, State is basically an object which hold other complex object/data.

**ACTION:** Action is basically a plain object. Action main intent is to change the data in the store. There is no other way than action carry the data to store. It has a type field which tells the type of action being performed and other than that it carry the updated data to the store. Type is basically a string

**REDUCER (PURE FUNCTION – function which produces the same output for the same input):** Reducer is a function which accepts the state and action and returns the updated state. They should free from any side effect , api calls

**DISPATCHER:** Dispatch function accepts an action or an async action. It may dispatches one or more actions to reducer

**Base dispatch** which is provided by store without any middleware – it synchronously dispatches action to the store’s reducer

**Middleware wrapped base dispatch:**  it allows dispatch to handle async actions along with the sync action

**Async Actions:** it is an action which is send to dispatcher but that action is not ready to be consumed by the reducer.it is transformed by middleware into series of actions before sending it to the base dispatcher

**CREATESLICE:** It is function that accepts inititalstate and an object of reducers functions.it then automatically generated the actions and actions types which corresponds to that state and reducer.

**Reducers** on slice An object containing Redux "case reducer" functions (functions intended to handle a specific action type, equivalent to a single case statement in a switch).

This object will be passed to [createReducer](https://redux-toolkit.js.org/api/createReducer), so the reducers may safely "mutate" the state they are given.

const yourSlice = createSlice({

  name: "Any name",

  initialState: { cartItems: [] },

  reducers: {

    reducerName :(state,action)=>{

       state.cartItems = action.payload

    }

  },

});

export const { reducerName } = yourSlice.actions;

export default yourSlice.reducer;

**extraReducers** in slice: One of the key concepts of Redux is that each slice reducer "owns" its slice of state, and that many slice reducers can independently respond to the same action type. extraReducers allows createSlice to respond to other action types besides the types it has generated.

**SELECTOR:**  deriving  logic of state in store is written in function , we call these functions as selector. This function takes the store and returns a part of a store.

Eg : const selectEntities = state => state.entities

Good practice to make a selector function name start with select

Selectors are basically defined in the 2 ways

1. Along side the reducer
2. In the components using useSelector hook