Assignment 12 - Let's Build Our Store

Q1 useContext vs Redux

⇒ useContext and Redux are both tools used for state management in React applications, but they serve different purposes and have different use cases

useContext
This is a hook provided by React, used to manage global state or pass data across components in React applications. useContext allows data to be passed through the component tree without the need for prop drilling, typically suitable for managing a smaller amount of global state in smaller or medium-sized application.

redux
It's a state management library that provides a single global state store to manage the entire application's state. Redux employs the concepts of action and reducers to update state and can be used in conjunction with React. However, it requires specific architecture and setup, commonly employed in larger or more complex applications or scenarios requiring intricate state management.

Q2 Advantages of using Redux Toolkit over Redux?

⇒ Redux Toolkit is a set of utility functions and abstractions that simplifies and streamlines the process of working with Redux. It is designed to address some of the common pain points and boilerplate associated with using plain Redux.

Advantages:

1) It helps us write less code. It provides shortcuts that save us from typing a lot of repetitive code

2) It makes things like fetching data simpler. It has a tool called createAsyncThunk that handles async actions in a way that's easy to understand and use.

3) Setting up your Redux store is easier with Redux Toolkit. It has a function called configureStore that simplifies the process, and it comes with sensible defaults, so you don't have to configure everything from scratch.

4) It has built-in debugging support. Enabling it is as easy as adding one line of code when setting up your store.

5) Working with immutable data is usually a bit tricky. It uses a library called Immer to handle this behind the scenes, so you can write more straightforward and readable code.

Q3 Explain Dispatcher?

⇒ In Redux, a dispatch is not a standalone concept; instead it's a term often used to refer to a function called dispatch. We use it to send actions to the store. An action is a plain JavaScript object that describes what should change in the applications state.

When we want to update the state in our Redux store, we create an action and dispatch it using the dispatch function.

```
// Code
const myAction = { type: 'INCREMENT' };
store.dispatch(myAction);
```

Q4 Explain Reducer?

⇒ In Redux Toolkit, the createSlice function is commonly used to create reducers. It simplifies the process of defining actions and the corresponding reducer logic, reducing boilerplate code.

We use createSlice to define a "slice" of your Redux store. A slice includes actions, a reducer, and the initial state. Reducers take in two things: previous state and an action. Then it returns the new updated instance of state. Whenever an action is dispatched, all the reducers are activated. Each reducer filters out the action using a switch statement switching on the action type. Whenever the switch statement matches with the action passed, the corresponding reducers take the necessary action to make the update and return a fresh new instance of the global state.

Q5 Explain Slice?

⇒ In Redux Toolkit, a slice is a collection of Redux-related code, including reducer logic and actions, that corresponds to a specific piece of the application state. Slices are created using the createSlice utility function provided by Redux Toolkit. The primary purpose of slices is to encapsulate the logic related to a specific part of the code more modular and easier to manage.

It auto generates your action creators and types which you have to define them as constants before redux-toolkit.

Q6 Explain Selector?

⇒ In Redux Toolkit, a selector is a function that extracts specific piece of data from the Redux store. It allows you to compute derived data from the store state and efficiently access specific parts of the state tree.

Redux Toolkit provides the createSlice and createAsyncThunk utilities along with the createSelector function from the reselect library to help manage selectors easily.

Selectors encapsulate logic for data retrieval, optimizing performance, and promoting code reusability. By using, memoization, selectors cache results to prevent unnecessary re-renders, thus enhancing overall application efficiency.

Q7 Explain createSlice and the configuration it takes

createSlice() → A function that accepts an initial state, an object of reducer functions, and a "slice name", and automatically generates action creators and action types that correspond to the reducers and state

initialState → The initial state value for the slice. This is the starting point for your state before any actions are dispatched.

reducers → An object where each key-value pair represents a reducer function. The keys are the names of the actions, and the values are the reducer logic.

//Code

```
import {createSlice} from '@reduxjs/toolkit'

const initialState = {value: 0}

const counterSlice = createSlice({
    name: 'counter',
    initialState,
    reducers: {
        increment(state) {
            state.value++
        },
        decrement(state) {
            state.value--
        },
    },
})

export const {increment, decrement} = counterSlice.actions
export default counterSlice.reducer
```