1. **useContext vs Redux**

|  |  |
| --- | --- |
| useContext | Redux |
| useContext is a hook. | Redux is a state management library. |
| It is used to share data. | It is used to manage data and state. |
| Changes are made with the Context value. | Changes are made with pure functions i.e. reducers. |
| We can change the state in it. | The state is read-only. We cannot change them directly. |
| It re-renders all components whenever there is any update in the provider’s value prop. | It only re-render the updated components. |
| It is better to use with small applications. | It is perfect for larger applications. |
| It is easy to understand and requires less code. | It is quite complex to understand. |
| Built-in tool that ships with React | Additional installation Required, driving up the final bundle size |
| Requires minimal Setup | Requires extensive setup to integrate it with a React Application |
| Specifically designed for static data, that is not often refreshed or updated | Works like a charm with both static and dynamic data |
| Adding new contexts requires creation from scratch | Easily extendible due to the ease of adding new data/actions after the initial setup |
| Debugging can be hard in highly nested React Component Structure even with Dev Tool | Incredibly powerful Redux Dev Tools to ease debugging |
| UI logic and State Management Logic are in the same component | Better code organization with separate UI logic and State Management Logic |

1. **Advantage of using Redux Toolkit over Redux**

* Redux Toolkit simplifies store setup down to a single clear function call, while retaining the ability to fully configure the store's options if you need to
* Redux Toolkit eliminates accidental mutations, which have always been the #1 cause of Redux bugs
* Redux Toolkit eliminates the need to write any action creators or action types by hand
* Redux Toolkit eliminates the need to write manual and error-prone immutable update logic
* Redux Toolkit makes it easy to write a Redux feature's code in one file, instead of spreading it across multiple separate files
* Redux Toolkit offers excellent TS support, with APIs that are designed to give you excellent type safety and minimize the number of types you have to define in your code
* RTK Query can eliminate the need to write any thunks, reducers, action creators, or effect hooks to manage fetching data and tracking loading state

1. **Explain Dispatcher.**

* [dispatch() is the method used to dispatch actions](https://redux.js.org/api/store#dispatchaction) and trigger state changes to the store.
* react-redux is simply trying to give you convenient access to it.
* The dispatch is not available on props if you do pass in actions to your connect function.
* Components cannot directly modify the store. So to do that we need to dispatch an action.
* This action will call a reducer function and that will modify the slice of the store.

1. **Explain Reducer.**

* In redux, the reducers are the **pure functions** that contain the logic and calculation that needed to be performed on the state.
* These functions accept the initial state of the state being used and the action type.
* It updates the state and responds with the new state. This updated state is sent back to the view components of the react to make the necessary changes.
* Basically, In short, we can say that Reducer’s work is to return the updated state and to also describe how the state changes.

1. **Explain slice.**

* A Redux Slice is a collection of reducer logic and actions for a single feature of our app.
* The name “slice” comes from the idea that we’re splitting up the root Redux state object into multiple “slices” of slate.
* A slice is the portion of Redux code that relates to a specific set of data and actions within the store 's state.

1. **Explain selector.**

* A “selector” is simply a function that accepts Redux state as an argument and returns data that is derived from that state.
* They can compute derived data, allowing Redux to store the minimal possible state. Selectors are also very efficient.
* Selector means you are subscribing to the store i.e. syncing with the store and automatically update on UI.

1. **Explain createSlice and the configuration it takes.**

* 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.
* This API is the standard approach for writing Redux logic.
* createSlice accepts a single configuration object parameter, with the following options:
  1. **initialState**[​](https://redux-toolkit.js.org/api/createslice" \l "initialstate" \o "Direct link to heading)
     1. The initial state value for this slice of state.
     2. This may also be a "lazy initializer" function, which should return an initial state value when called. This will be used whenever the reducer is called with undefined as its state value, and is primarily useful for cases like reading initial state from localStorage.
  2. **name**[​](https://redux-toolkit.js.org/api/createslice#name)
     1. A string name for this slice of state. Generated action type constants will use this as a prefix.
  3. **reducers**[​](https://redux-toolkit.js.org/api/createslice#reducers)
     1. An object containing Redux "case reducer" functions (functions intended to handle a specific action type, equivalent to a single case statement in a switch).

### extraReducers[​](https://redux-toolkit.js.org/api/createslice" \l "extrareducers" \o "Direct link to heading)

* + 1. 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.
    2. extraReducers allows createSlice to respond to other action types besides the types it has generated.
    3. As case reducers specified with extraReducers are meant to reference "external" actions, they will not have actions generated in slice.actions.