Redux Js

Question 1: What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store.

Redux is a predictable state container for JavaScript applications, commonly used with libraries like React to manage the application's state in a consistent and predictable way. It helps in managing the state of an application and ensures that data flows in a clear and controlled manner.

Actions:

- Actions are plain JavaScript objects that describe **what happened** in the application. They have a type property that tells Redux what kind of action is being performed.
- Actions may also carry additional data (called "payload") to give more context about the event.

Reducers:

- Reducers are pure functions that define how the state of the application changes in response to actions. A reducer takes two parameters: the current state and an action, and it returns a new state.
- Reducers are responsible for handling actions and updating the state.

Store:

- The store is a centralized object that holds the entire state of the application. It is created using the createStore function in Redux.
- The store provides methods to get the current state, dispatch actions to change the state, and subscribe to state changes.

Why Redux is used in React applications:

- **Predictable State Management**: With Redux, state is predictable because it is stored centrally and only changes through actions and reducers, making it easy to debug and trace changes.
- **Centralized State**: Redux allows the entire state of the app to be in one place, which simplifies passing data between components, especially in large applications.
- **Easier State Sharing**: Components can subscribe to the store and update their data when the state changes, making it easy to share state across many components.
- **Separation of Concerns**: Redux separates the state management from the UI logic, helping developers maintain clear, organized code.

Question 2: How does Recoil simplify state management in React compared to Redux?

- Simpler API: Recoil boasts a more intuitive and beginner-friendly API. It leverages React's context API under the hood, making it easier to understand and use for developers familiar with React's core concepts.

- Atom-Based State: Recoil centers around the concept of "Atoms," which are units of state. This granular approach makes it easier to manage and reason about different parts of your application's state.
- Selectors: Recoil provides a powerful mechanism for deriving complex state from other Atoms and Selectors. This allows you to create derived data, effectively caching and optimizing calculations.
- No Global Store: Unlike Redux, Recoil doesn't require a single, global store. State can be more localized and managed within specific components or parts of your application, leading to better code organization and easier testing.
- Less Boilerplate: Recoil generally requires less boilerplate code than Redux. Setting up and managing state with Recoil can be more concise and streamlined.

  Practical:- Slice:-

```
import { createSlice } from "@reduxjs/toolkit";


const initialState = {
 cartitem: [], // 0 display
 cartopen: false
};


// Reducer + Actions
const cartSlice = createSlice({
 name: "cart",
 initialState,
 reducers: {
  additem(state, action) {
   const existingItem = state.cartitem.find(item => item.id === action.payload.id);


   if (existingItem) {
    existingItem.quantity += 1; // Increase quantity if item already exists
   } else {
    state.cartitem.push({ ...action.payload, quantity: 1 }); // Add new item with quantity 1
   }
  },
```

```
    removeitem(state, action) {
       state.cartitem = state.cartitem.filter(item => item.id !== action.payload);
    },
    incrementqty(state, action) {
     state.cartitem = state.cartitem.map(item =>
       item.id === action.payload ? { ...item, quantity: item.quantity + 1 } : item
     );
    },
    decrementqty(state, action) {
     state.cartitem = state.cartitem.map(item =>
       item.id === action.payload && item.quantity > 1
         ? { ...item, quantity: item.quantity - 1 }
         : item
     );
    },
    togglecart(state) {
     state.cartopen = !state.cartopen;
    }
  }
});

export const { additem, removeitem, incrementqty, decrementqty, togglecart } =
cartSlice.actions;
export default cartSlice.reducer;
import { createSlice } from "@reduxjs/toolkit";

const initialState={
    wishitem:[],// 0 display
    wishopen:false
}
```

```
//reducer + action

const wishSlice=createSlice({
   name:'wish',//reducer
   initialState,
   reducers:{
      //action

      additemwish(state,action){

        const newitem =action.payload.id;

         const existingid =state.wishitem.find(item=>item.id == action.payload);

         if(existingid){

            existingid.quantity++
         }else{
            state.wishitem.push(action.payload)
         }
      },
      removeitemwish(state,action){

         state.wishitem=state.wishitem.filter(item=>item.id !== action.payload)
      },
      incrementqtywish(state,action){
         state.wishitem.map(item=>{
            if(item.id == action.payload){
               item.quantity++
            }
         })
```

```
    },
    decrementqtywish(state,action){
      state.wishitem.map(item=>{
        if(item.id == action.payload){
          item.quantity--
        }
      })
    },
    togglewish(state,action){
      state.wishopen =action.payload
    }
  }
})

export const
{additemwish,removeitemwish,incrementqtywish,decrementqtywish,togglewish}=wishSlice.
actions;

export default wishSlice.reducer
```

Store:-

```
import { configureStore } from "@reduxjs/toolkit";
import cartSlice from "../slice/cartSlice";
import wishSlice from "../slice/wishSlice"

 const mystore = configureStore({
   reducer:{
     cart:cartSlice,
     wish:wishSlice
   }
})

export default mystore;
```