How React Components React to Each Other In React, components can communicate via props and state. Parent components pass data to child components through props, and child components can send data back using callback functions. React's virtual DOM ensures efficient updates and renders only the parts of the UI that change.

React Lifecycle Hooks in Functional Components React's lifecycle methods (such as `componentDidMount`, `componentWillUnmount`, etc.) are typically used in class components. In functional components, React hooks like `useEffect` serve the same purpose. The `useEffect` hook runs side effects like data fetching or subscriptions when a component mounts or updates.

useState `useState` is a hook used to declare state in functional components. It allows you to create a state variable and update it.

Props `Props` are immutable and passed from parent to child components. They allow parent components to provide data to child components but cannot be modified by the child component itself.

useMemo
`useMemo` is a React hook used to memoize the result of a computation to avoid recalculating it unnecessarily. It optimizes performance by recomputing only when its dependencies change. Useful for expensive calculations.

useCallback
`useCallback` is a React hook used to memoize a function so that it doesn't get recreated on every render. It is beneficial when passing callbacks to child components to prevent unnecessary re-renders.

memo
`memo` is a higher-order component that wraps a React component to memoize its rendering. It ensures the component only re-renders when its props change, improving performance for components with static or rarely changing props.

useRef
`useRef` is a React hook that creates a mutable reference object. It is often used to persist values across renders without causing re-renders, or to directly access and manipulate DOM elements.

What is Redux:
Redux is a state management library for JavaScript applications, commonly used with React. It provides a predictable state container to manage the application's state globally.

Why:
Redux helps manage complex application states, especially in large-scale apps. It centralizes state, making debugging, testing, and sharing data between components easier.

How:

1. **Dispatch**: Use `dispatch` to send actions to the store to update the state.
2. **Select**: Use `useSelector` or `mapStateToProps` to retrieve specific parts of the state from the store.