Q1: What are React Hooks and why were they introduced?

A1: React Hooks are functions that let you use state and other React features in functional components. They were introduced to avoid the complexities of class components and promote reusable logic through custom hooks.

Q2: How does useEffect differ from componentDidMount, componentDidUpdate, and componentWillUnmount?

A2: useEffect combines all three lifecycle methods. It runs after every render by default. You can control its execution using a dependency array.

Q3: What is Redux and when should it be used?

A3: Redux is a state management library useful when multiple components need access to shared state. It enforces unidirectional data flow and centralizes state updates through actions and reducers.

Q4: Explain the role of middleware in Redux.

A4: Middleware in Redux provides a way to intercept and act on actions before they reach the reducer. Common use cases include async actions (redux-thunk) and logging.

Q5: What are controlled and uncontrolled components in React?

A5: Controlled components rely on React state for their values, while uncontrolled components manage their own internal state via refs.

Q6: How do you optimize performance in large React applications?

A6: Use React.memo, useCallback, useMemo, lazy loading, code splitting, and avoid unnecessary re-renders by tracking dependencies.

Q7: What is the difference between useCallback and useMemo?

A7: useCallback returns a memoized function, while useMemo returns a memoized value. Both optimize performance by preventing unnecessary recalculations.

Q8: How does Redux Toolkit simplify Redux usage?

A8: Redux Toolkit provides utilities like createSlice and configureStore that reduce boilerplate and promote best practices for Redux state management.

Q9: What is the purpose of keys in React lists?

A9: Keys help React identify which items have changed, are added, or are removed. They must be unique and stable for efficient reconciliation.

Q10: How can you handle side effects in Redux?

A10: Use middleware like redux-thunk for simple async logic or redux-saga for more complex workflows.