1) Hooks: Prepare this in order

1. useState
2. useEffect
3. useContext
4. useReducer
5. useMemo
6. useCallback
7. useRef

2) Higher Order Components (HOC):

- What?
- When?
- Why?
- How?

3) Life Cycle Methods of Components:

- Class Components
- Mounting
- Updating
- Unmounting

4) State management (all about data):

- State/Props
- Props drilling
- Context

5) Redux or Zustand:

- How redux works?
- Why?
- When?
- Redux Toolkit (RTK)

6) Custom Hooks:

- When to use?
- Code
- Why? (to make code clean, maintainable, readable, reusable)

7) Lazy Loading (Very imp and highly asked):

- Code splitting
- Chunking
- Suspense

8) Virtual DOM:

- Reconciliation Algorithm
- React Fiber
- Renders
- Diff algorithm
- How does render work?

9) SSR vs CSR (important) : - What? - Difference - SEO and performance (SSR)	**10) Routing (Role-based access control-RBAC)** - react-router - How do you manage protected routes? - How do you handle routes? - query params - Dynamic routing
11) Testing - React Testing Library - Unit Testing	**12) Async Tasks** - API Calls - useEffect in depth - Events - Promises - setTimeout

13) Reusability, Readability, modularity, testability (Coding Practices)

14) Performance

- Lazy loading
- Asset optimization (how do you optimize js, css code)
- Writing optimized code
- Bundler
- CDN / Server level
- Rendering of components

15) Styling

- Tailwind
- StyleX
- Bootstrap
- Material UI
- Ant UI
- CSS / SCSS

16) Accessibility

17) Security

Hack for interview: Try to mention that the code you wrote is testable and try to write test cases

1) Hooks:

useState: Manages state in functional components.

useEffect: Handles side effects in functional components.

useContext: Accesses the context in functional components.

useReducer: Manages complex state logic with a reducer function.

useMemo: Memoizes values to optimize performance.

useCallback: Memoizes callback functions to prevent unnecessary renders.

useRef: Creates a mutable object that persists between renders.

2) Higher Order Components (HOC):

What?: Functions that take a component and return an enhanced version.

When?: Reuse component logic, share code, or manipulate component behavior.

Why?: Promotes code reusability and separation of concerns.

How?: Wrap a component with a function that adds or modifies its behavior.

3) Life Cycle Methods of Components:

Class Components: Traditional React components using ES6 classes.

Mounting: Component is being created and inserted into the DOM.

Updating: Component is being re-rendered as a result of changes.

Unmounting: Component is being removed from the DOM.

4) State management (all about data):

State/Props: Internal state for a component/external data passed to a component.

Props drilling: Passing props through multiple layers of components.

Context: Provides a way to pass data through the component tree without passing props.

5) Redux or Zustand:

How redux works?: Centralized state management using actions and reducers.

Why?: For managing complex application states.

When?: In large applications with a need for a single source of truth.

Redux Toolkit (RTK): Simplifies Redux setup and usage.

6) Custom Hooks:

When to use?: Extracting and reusing component logic.

Code: Functions prefixed with "use" returning stateful logic.

Why?: Enhances code organization, reusability, and readability.

7) Lazy Loading:

Code splitting: Breaking down the application into smaller parts.

Chunking: Loading only the necessary code chunks.

Suspense: Pausing rendering until a component is ready.

8) Virtual DOM:

Reconciliation Algorithm: Efficiently updates the UI based on state changes.

React Fiber: A reimplementation of React's core algorithm.

Renders: The process of updating the virtual DOM.

Diff algorithm: Compares the previous and current state to determine changes.

How does render work?: Updating the UI based on virtual DOM changes.

9) SSR vs CSR (important):

What?: Server-Side Rendering vs. Client-Side Rendering.

Difference: Where rendering occurs - server or client.

SEO and performance (SSR): Improved search engine optimization and initial load speed.

10) Routing (Role-based access control-RBAC):

react-router: Library for handling navigation in React applications.

How do you manage protected routes?: Utilize authentication and authorization checks.

How do you handle routes?: Define routes and components for navigation.

Query params: Additional information passed in the URL.

Dynamic routing: Creating routes dynamically based on data.

11) Testing:

React Testing Library: Testing library for React applications.

Unit Testing: Testing individual units of code.

Hack for interview: Emphasize writing testable code and demonstrate test cases.

12) Async Tasks:

API Calls: Fetching data from external sources.

useEffect in depth: Managing side effects, including async operations.

Events: Handling asynchronous events.

Promises: A pattern for handling asynchronous operations.

setTimeout: Delaying the execution of code.

13) Reusability, Readability, modularity, testability (Coding Practices):

14) Performance:

Lazy loading: Loading resources only when needed.

Asset optimization: Minifying and compressing JS/CSS code.

Writing optimized code: Following best practices for efficient code.

Bundler: Tools like Webpack to bundle and optimize code.

CDN / Server level: Distributing assets for faster loading.

Rendering of components: Optimizing rendering for better performance.

15) Styling:

Tailwind, StyleX, Bootstrap, Material UI, Ant UI, CSS / SCSS: Different styling approaches and libraries.