Here are 100 advanced-level React.js interview questions designed to challenge your deep understanding of React:

**Advanced React Questions**

1. How does React’s reconciliation algorithm work? Can you explain the process of diffing?
2. What is React Fiber and how does it improve the React rendering process?
3. Explain the concept of **Concurrency** in React. How does it affect component rendering?
4. How does **React’s Virtual DOM** differ from the actual DOM? What are the performance implications of using the Virtual DOM?
5. How can you optimize React application performance by memoizing components?
6. How does **React Suspense** work in terms of loading and handling asynchronous data?
7. What are **error boundaries** in React and how do you handle errors in React components?
8. Can you explain **lazy loading** components in React? How do you handle fallback UI during lazy loading?
9. How do **React Hooks** like useEffect and useLayoutEffect work under the hood?
10. How would you use **React context** in large-scale applications without causing unnecessary re-renders?
11. How do you handle **state management** in a large React application with multiple layers of components?
12. Can you explain the concept of **Higher Order Components (HOCs)** and their usage in React?
13. How does **React.memo()** optimize performance in functional components?
14. What is **useCallback** and when should you use it to optimize performance in React?
15. What are the differences between **React.useEffect()** and **componentDidUpdate()**?
16. How does **React’s Context API** perform compared to traditional state management solutions like Redux?
17. How do you implement **server-side rendering (SSR)** in React, and what are the benefits?
18. How does **hydration** work in React for server-side rendered applications?
19. What is **React Suspense for data fetching**, and how does it handle concurrent rendering?
20. Can you explain the concept of **React portals** and when to use them?
21. What are **React’s Fiber Reconciler** and its impact on performance and rendering behavior?
22. How does **React’s event delegation** system work under the hood?
23. Explain **React’s context** provider pattern and how to use it for global state management.
24. How do you handle **conditional rendering** with complex logic in React?
25. How do you implement **stateful components** using **React hooks**?
26. What are **React fragments** and why are they used?
27. How do you perform **state persistence** across sessions in React apps?
28. How does **React’s Hooks API** change the way you think about state and side effects in functional components?
29. What are the performance trade-offs of using **useState** vs **useReducer** in React?
30. What is the purpose of **React’s key** prop in list rendering, and how does React handle list updates internally?
31. How do you handle **deeply nested data** with **React Context** without triggering unnecessary re-renders?
32. What is the **useImperativeHandle** hook, and how does it work with **React.forwardRef**?
33. How can you optimize **large lists** in React using **virtualization** techniques?
34. How would you handle **complex animations** in React, and what libraries would you consider?
35. How does **Redux middleware** like **redux-saga** or **redux-thunk** work to handle side effects in React apps?
36. How would you architect a **multi-step form** with **React Hooks** and validate each step individually?
37. What is the purpose of **React.memo()** and how does it prevent unnecessary re-renders in functional components?
38. How can you handle **data fetching and caching** in a React application using **React Query** or **Apollo Client**?
39. Explain how you would implement **infinite scrolling** with **React** and **windowing techniques**.
40. What is **React Profiler**, and how can it help you identify performance bottlenecks in a React application?
41. How do you use **React Router v6** features like useRoutes and nested routing?
42. What is the **useEffect** dependency array and how does React optimize effects based on the dependencies?
43. Can you explain **server-side rendering (SSR)** vs **client-side rendering (CSR)** in React?
44. What is the **key difference between useLayoutEffect and useEffect** in React, and when should you use one over the other?
45. How do you implement **dynamic imports** in React and handle loading fallback components?
46. What are **React Hooks** like useReducer and when would you use them instead of useState?
47. How do you handle **user authentication** and **authorization** in React using JWT tokens?
48. Can you explain how **React’s Context API** can be used to manage global state, and how to prevent unnecessary re-renders?
49. What is the role of **Redux DevTools** in debugging React apps, and how do you use it effectively?
50. How would you implement **state management** with **Redux Toolkit** and **slice reducers** in a large React app?
51. What are **custom hooks**, and how can you build them to encapsulate complex logic in React?
52. How does **React’s synthetic event system** differ from the native DOM event system?
53. How do you create a **global loading spinner** using **React context** and **hooks**?
54. How would you implement **authentication with OAuth 2.0** in React, and how would you handle tokens securely?
55. What are **render props** in React, and how are they different from HOCs (Higher Order Components)?
56. How does **React’s shouldComponentUpdate** work, and how can it help optimize re-rendering?
57. How do you manage **error boundaries** and handle errors in nested child components in React?
58. What is the **useLayoutEffect** hook, and when is it necessary to use it over useEffect?
59. What is **React Router’s useNavigate** hook, and how does it differ from useHistory?
60. How do you implement **real-time updates** in React using WebSockets or Server-Sent Events (SSE)?
61. Can you explain **React’s reconciler** and how React determines which components to re-render?
62. How would you implement **multilingual** or **internationalized** applications in React?
63. How does **React’s Fiber architecture** improve rendering performance in React applications?
64. Can you explain the difference between **local state** and **global state** in React, and when each should be used?
65. How do you implement **drag-and-drop functionality** in React using libraries like react-dnd or react-beautiful-dnd?
66. How would you handle **real-time form validation** using React hooks and context?
67. How does **React Suspense for data fetching** work alongside React’s Concurrent Mode?
68. Can you explain **React’s concurrent rendering** and how it improves user experience and performance?
69. How do you optimize the **rendering of large lists** using **windowing** and **virtualization** techniques in React?
70. How would you handle **time zones** and **date formatting** in a React app using libraries like date-fns or moment.js?
71. What is **code-splitting** in React, and how does it improve the performance of a large application?
72. How do you use **React Query** for managing server-state, and how does it differ from **Redux**?
73. How does **React’s Context API** handle performance issues when using large or deeply nested contexts?
74. What are **React’s StrictMode** and its impact on identifying potential problems during development?
75. How does **React Suspense** handle **code splitting** and **lazy loading**?
76. How would you implement **state persistence** for offline access in React using **IndexedDB** or **localStorage**?
77. How do you implement **custom components** in React with advanced pattern-based design?
78. Can you explain the usage of **forwardRef** and its advantages in passing refs to child components?
79. How do you test **asynchronous functions** and **data fetching** in React applications?
80. How does **React’s error handling** work with the use of **Error Boundaries**?
81. What are **controlled** and **uncontrolled components** in React, and how do they differ?
82. How would you manage **accessibility (a11y)** in a complex React application?
83. How do you manage **form state** in React using **React Hook Form** or **Formik**?
84. How does **React Suspense** and **Concurrent Mode** interact to improve user experience?
85. What is **static typing** with **TypeScript** in React, and how does it improve development?
86. How does **Redux-Toolkit** simplify Redux implementation and improve code organization?
87. How do you implement **client-side routing** and **nested routing** with **React Router v6**?
88. How do you handle **animations** in React using libraries like react-spring or framer-motion?
89. How would you implement **state persistence** with **Redux** using **redux-persist**?
90. How do you create and manage **API clients** in React with **Axios** or **fetch**?
91. How do you handle **API pagination** in React using **useEffect** and **React Query**?
92. How does **React’s Hooks API** provide better encapsulation of logic compared to class components?
93. Can you explain how to use **React Suspense** to handle **fallback UI** while loading content?
94. What is the role of **custom hooks** in React and how do they improve code reusability and modularity?
95. How does **Concurrent Mode** in React enhance app performance and how can it be enabled?
96. How do you optimize **code splitting** in a React application using **React.lazy** and **Suspense**?
97. How do you create a **high-performance table/grid component** using **React** and **virtualization**?
98. How would you implement **real-time notifications** in React using **WebSockets** or **Firebase**?
99. How would you handle **multi-user collaboration** features in React apps (e.g., Google Docs-style real-time collaboration)?
100. How does **server-side rendering (SSR)** in React improve SEO and initial load performance?

These questions test a deeper understanding of React's internals, optimizations, performance considerations, and advanced features, and they require experience and knowledge of React's advanced concepts.