50+ ReactJS Interview Questions 2025: Common, Basic & Advanced

Basic ReactJS Interview Questions for Freshers:

It's essential for aspiring developers to build a strong foundation in ReactJs. So, if you're new to React and prepping for your first interview, this section can be your React bootcamp. We'll explore some fundamental React concepts through commonly asked basic interview questions.

1. What is React?

React is a JavaScript library for building user interfaces. It allows you to create reusable components that manage their own state and efficiently update the UI.
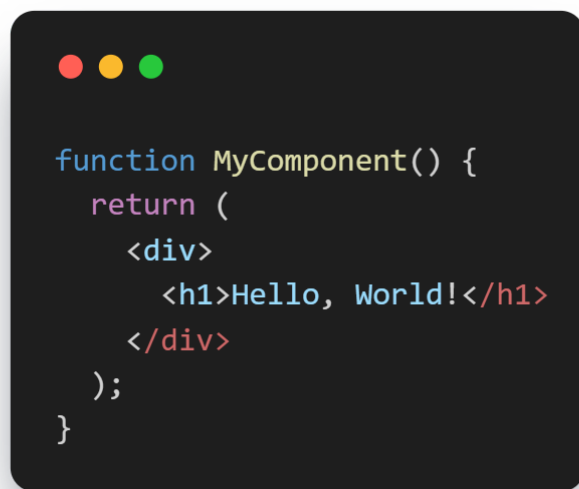
2. What are Components in React?

Components are the building blocks of React applications. They are reusable pieces of code that encapsulate functionality and UI. You can think of them as independent modules that display a specific part of your application's interface.

3. Explain JSX in React.

JSX (JavaScript XML) is a syntax extension that lets you write HTML-like structures within your JavaScript code. It improves readability and makes it easier to visualize the structure of your UI components.

Example:

```
function MyComponent() {
  return (
    <div>
      <h1>Hello, World!</h1>
    </div>
  );
}
```

4. Explain the concept of the Virtual DOM.

The Virtual DOM (Document Object Model) is a concept implemented in React that provides a programming API that works like a lightweight copy of the actual DOM. This means that whenever a component's state changes, the Virtual DOM gets updated instead of the real DOM. React then efficiently updates the real DOM to match the Virtual DOM, minimizing performance costs and enhancing user experience.

5. Distinguish between a Class component and a Functional component.

Class components are ES6 classes that extend from 'React.Component' and can hold and manage local state and lifecycle methods. On the other hand, Functional components are simpler and primarily used for rendering UI without handling state or lifecycle methods, although with React Hooks, they are now capable of using both.

6. How do you create a React component?

There are two main ways to create React components:

- **Class-based components:** These use the `class` keyword and lifecycle methods to manage state and handle events.
- **Functional components:** These are simpler functions that return JSX code and can leverage React Hooks for managing state and side effects.

7. What are Props in React?

Props are read-only properties passed down from parent components to child components. They act like arguments, providing data to child components without modifying their internal state.

Example:

```
function Greeting(props) {
  return (
    <p>Hello, {props.name}!</p>
  );
}
```

8. What's the difference between Props and State in React?

- Props: Read-only data passed down from parent to child components. Used for customization without changing internal state.
- State: Internal data managed by a component. State can be changed, making it ideal for keeping track of user inputs, events, and data that changes over time.

9. What does the render() method do in React components?

The render() method is essential in class components. It examines this.props and this.state and returns one of the following: React elements, arrays and fragments, portals, string and numbers, Booleans or null. This output represents what should be displayed on the screen.

10. What are keys in React and why are they important?

Keys are special string attributes that you need to include when creating lists of elements. They help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity, enhancing performance during updates.

11. What is an event in React?

In React, an event is similar to events in plain JavaScript—actions like clicks, form submissions, or key presses. React wraps these events in its own SyntheticEvent wrapper to ensure consistency across different browsers.

12. How do you handle events in React?

Handling events in React is straightforward: you use event handlers. These are functions you write to execute when an event occurs. For example, you might have a button that needs to handle a click event, which you can set up like this: <button onClick={handleClick}>Click me!</button>, where handleClick is the function that runs when the button is clicked.

13. What is a stateful component?

A stateful component in React is one that can hold and change state over time. These components are usually class components but can also be functional components using hooks like useState. They are handy when your component needs to remember something or be interactive.

14. What is a stateless component?

Conversely, a stateless component is one that doesn't manage any state. These often serve as presentational components, merely rendering UI elements based on the props they receive. Stateless components can be functional components without any hooks for state management.

15. How do you pass data between components in React?

Passing data between components in React is done through props (short for properties). You pass data from parent components to child components as arguments to the child component in the JSX where it's used.

16. What are controlled components?

In React, a controlled component is one that manages its own state and updates based on user input. For example, form elements like inputs often need to be controlled components, whereas React handles the form data.

17. How do you update the state of a component?

To update the state of a component in React, you use the setState method in class components or the setter function from useState in functional components. It's important to remember that state updates may be asynchronous and should not rely on the previous state directly.

18. What is the significance of the componentDidMount lifecycle method?
componentDidMount is a lifecycle method in class components that is called after the component is rendered for the first time. This is the perfect place to initiate API calls, set timers, or handle any interactions that require the DOM nodes to be present.

19. Explain the purpose of the useState hook.

The useState hook is a fundamental hook in React for adding state to functional components. It allows you to add and manage state in a component without converting it into a class component.

20. What is the useEffect hook and how is it used?

The useEffect hook lets you perform side effects in your components. These can be anything from fetching data to directly interacting with the DOM. It can be configured to run after every render or only when certain values change.

Intermediate level- ReactJs Interview Questions and Answers

Now that you're familiar with the basics of React, it's time to move towards more complex concepts. In this section, we'll cover topics like higher-order components, the React lifecycle, and state management, among others. The following set of questions and answers have been carefully curated to provide you with a comprehensive understanding of intermediate-level React concepts.

21. What are higher-order components?
Higher-order components (HOCs) are a powerful pattern used in React to enhance components with additional functionality. An HOC is a function that takes a component and returns a new component. It's useful for reusing code, logic, and bootstrap abstraction in React applications.

22. Explain the lifecycle of a React component.
The lifecycle of a React component can be divided into three phases: mounting, updating, and unmounting. Mounting is when the component is being created and inserted into the DOM. Updating occurs when a component is re-rendered due to changes in props or state. Unmounting is the final phase when the component is removed from the DOM. Lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount allow developers to hook into these phases for managing operations appropriately.

23. How can you handle forms in React?
Forms in React can be handled using controlled components where form data is handled by the state within the component. Each state mutation has a corresponding handler function, making it straightforward to modify or validate user input.

24. What is lifting state up in React?
Lifting state up is a common pattern for sharing state between multiple components. It involves moving state to the nearest common ancestor of the components that require it. This way, state can be passed down as props to the components that need it, ensuring consistent data and behavior.

25. How does React implement the re-rendering of components?
React implements re-rendering through its reconciliation algorithm, where it updates the DOM based on the changes in the component's state or props. React efficiently updates only the parts of the DOM that actually changed, rather than re-rendering everything, which enhances performance.

26. What are controlled components?
Controlled components are those where React controls the values of input elements. The input form elements, such as <input>, <textarea>, and <select>, have their values controlled by React's state, and their values change via state, not directly from user input.

27. What are uncontrolled components?
Uncontrolled components work like traditional HTML form inputs, where the forms naturally keep some internal state. In React, uncontrolled components are managed using a ref to get form values from the DOM instead of handling the form state via state.

28. Explain the concept of virtual DOM and how it differs from real DOM.
The virtual DOM is a lightweight copy of the real DOM. It is a concept implemented by React that allows for efficient updates to the UI by minimizing direct manipulations of the real DOM, which can be slow. When a component's state changes, React creates a new virtual DOM and compares it with the previous version. Only the differences are updated in the real DOM.

29. How do you optimize performance in a React application?
Optimizing performance in a React application can involve several strategies, such as using shouldComponentUpdate or React.memo to prevent unnecessary re-renders, code-splitting to reduce the size of bundles loaded initially, and using lazy loading for components.

30. What is the context API?
The Context API is a way for a React app to effectively produce global variables that can be passed around. This is the alternative to "prop drilling" or moving props from grandparent to child to parent, and so on. Context is often used to share data such as user authentication, themes, or a language preference.

31. How do you use refs in React?
Refs in React are used to get references to a DOM node or an instance of a component in a React Application. Refs are created using React.createRef() and attached to React elements via the ref attribute.

32. Explain forward refs in React.
Forward refs in React allow you to pass a ref down to a child component. This is particularly useful in higher-order components or when you need the parent component to directly interact with child component DOM nodes.

33. What are synthetic events in React?
Synthetic events in React are wrapper objects around the native event. They combine the behavior of different browser's native events into one API, ensuring that the events behave identically across all browsers.

34. How do you implement error handling in React components?
Error handling in React components can be achieved using error boundaries. An error boundary is a component that catches JavaScript errors in its child component tree, logs those errors, and displays a fallback UI instead of the component tree that crashed.

35. What are portals in React?
Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component. This is commonly used for modals, tooltips, and floating menus.

36. How does React Router work?
React Router is a library that enables dynamic routing in a web app. It keeps the UI in sync with the URL, allowing you to handle routing declaratively. It works by changing your application's components depending on the browser's URL, without reloading the page.

37. What is the difference between React Router and traditional routing?
React Router uses client-side routing, where the routing is handled internally by the JavaScript that is loaded on the page, without the need for page reloads. Traditional routing, on the other hand, involves requests to a server and reloading the entire page with new content.

38. How do you implement code-splitting in React?
Code-splitting in React can be implemented using React.lazy and Suspense. This allows you to split your code into separate chunks which can be loaded on demand. It is particularly useful for improving the initial load time of the application.

Advanced ReactJS Interview Questions for Experienced:

Mastering advanced React concepts is crucial for handling complex projects and architectural challenges. Our "Advanced ReactJS Interview Questions for Experienced" section delves into advanced topics that seasoned developers often encounter. From state management strategies to handling side effects, hooks, and server-side rendering, this section is designed to test and expand your mastery of React.

39. What are the different ways to manage State in a React application?
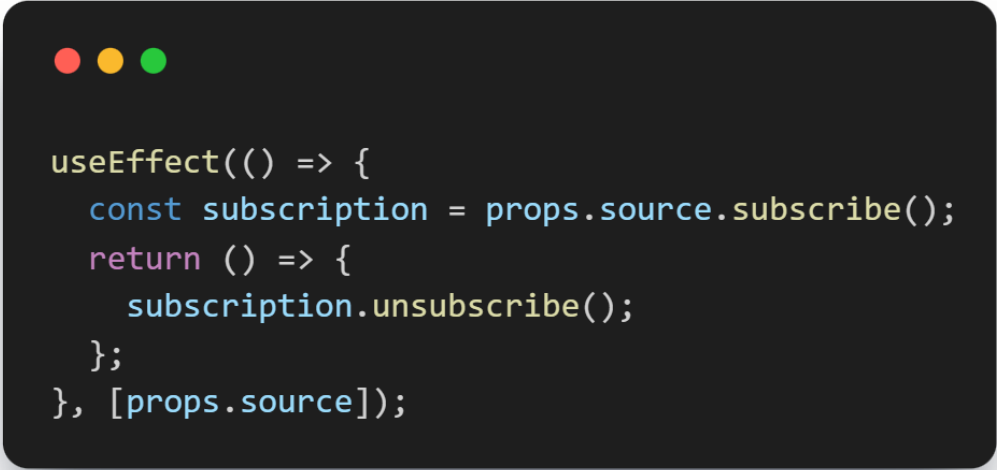
React offers multiple ways to manage state, each with its own use case. Here are the common approaches:

- Local state: Managed within a component using `useState` or `useReducer`.
- Global state: Tools like Redux or Context API help manage state that is accessible by any component in the application.
- Server state: Data fetched from an external server which can be managed via React Query or SWR.
- URL state: State represented in the URL parameters accessible via React Router.

40. How do you handle side effects in React components?

Side effects are operations affecting other components or that involve asynchronous operations. React uses the `useEffect` hook to handle side effects, such as API calls, subscriptions, or manually manipulating the DOM.

Example:

```
useEffect(() => {
  const subscription = props.source.subscribe();
  return () => {
    subscription.unsubscribe();
  };
}, [props.source]);
```

41. Explain the concept of hooks in React. What problems do they solve?

Hooks are functions that let you "hook into" React state and lifecycle features from function components. They allow you to write functional components with the same capabilities as class components, making your code cleaner and easier to maintain.

Problems Solved by Hooks:

- **Code Reusability:** Share logic across components without resorting to higher-order components (HOCs).
- **Component Composition:** Build complex UIs by combining simpler functional components.
- **State Management:** Use the `useState` hook to manage component state within functional components.

42. How would you implement global state management in React without using external libraries?

To manage global state without external libraries, the Context API can be utilized effectively. It allows you to share values between components without having to explicitly pass a prop through every level of the tree.

Example:

```
const MyContext = React.createContext(defaultValue);

function MyComponent() {
  return (
    <MyContext.Provider value={/* some value */}>
      {/* child components */}
    </MyContext.Provider>
  );
}
```

43. What is React Fiber?

React Fiber is a complete reimplementation of the React core algorithm. It enhances the suitability of React for areas like animation, layout, and gestures. Its main goal is to enable incremental rendering of the virtual DOM.

44. How do you handle server-side rendering with React?

SSR allows you to render your React application on the server, improving initial page load times and SEO. Libraries like Next.js simplify SSR implementation in React projects.

Example with Next.js:

```
function HomePage() {
  return <div>Welcome to Next.js!</div>
}

export async function getServerSideProps(context) {
  return { props: {} };
}
```

45. What are the common performance issues in React applications? How do you troubleshoot them?

Performance issues in React often include:

● Unnecessary Re-renders: Optimize components using shouldComponentUpdate (class components) or React.memo (functional components) to prevent unnecessary re-renders.
● Large Virtual DOM Diffs: Break down complex components into smaller ones to minimize the amount of DOM that needs to be updated.
● Excessive Prop Drilling: Use Context API or state management solutions to avoid passing props through multiple levels of components.

46. How do you secure a React application?

Just like any web application, React applications need to be secured. Here are some key areas to focus on:

● Sanitize User Input: Prevent XSS attacks by sanitizing any user-provided data before displaying it on the UI.
● Secure API Communication: Use HTTPS for API communication to encrypt data transmission.

● Implement Authentication and Authorization: Control user access to specific features and data based on their roles.

47. What are the pros and cons of using Redux?

Redux is a popular state management library, but it's not always necessary. Here's a quick breakdown:

Pros:
● Centralized State: Keeps all application state in one place, making it easier to manage and reason about.
● Predictable Updates: Makes application flow more predictable and easier to debug.
● Large Community and Ecosystem: Plenty of resources and tools available for working with Redux.

Cons:

● Complexity: Setting up and managing Redux can add complexity to smaller applications.
● Boilerplate Code: Requires writing additional code for actions, reducers, and middleware.

48. How do you integrate TypeScript with React?

TypeScript can be integrated by creating React components with TypeScript. This adds static type checking, enhancing the reliability and maintainability of the application.

Example:

```
interface AppProps {
  message: string;
}

const App: React.FC<AppProps> = ({ message }) => <div>{message}</div>;
```

49. Explain the main principles of Redux.

Redux follows three fundamental principles:

- Single source of truth: The state of your entire application is stored in one object tree.
- State is read-only: The only way to change the state is to emit an action.
- Changes are made with pure functions: Reducers are pure functions that take the previous state and an action to compute the next state.

50. How do you handle asynchronous actions in Redux?

Asynchronous actions in Redux are handled by middleware like Redux Thunk or Redux Saga. These allow you to write action creators that return a function instead of an action.

Example with Redux Thunk:

```
const fetchData = () => {
  return (dispatch) => {
    fetch('url')
      .then(response => response.json())
      .then(data => dispatch({ type: 'FETCH_DATA_SUCCESS', payload: data }))
  };
};
```

51. What is React Suspense and how do you use it?

React Suspense lets you specify the loading indicator in case some components in the tree below it are not yet ready to render. It's used for code splitting and lazy loading components.

Example:

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <OtherComponent />
    </React.Suspense>
  );
}
```

52. How do you test React components?

Popular testing libraries for React include:

- **React Testing Library:** A lightweight library focused on testing components in isolation from implementation details.
- **Jest:** A popular testing framework that can be used to write unit and integration tests for React components.

Example:

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  expect(screen.getByText(/learn react/i)).toBeInTheDocument();
});
```

53. What is the use of static type checking in React?

Static type checking helps identify potential errors in your code before runtime. TypeScript, as mentioned earlier, is a popular choice for adding static typing to React projects. It improves code readability, maintainability, and helps catch errors early in the development process.

```
import PropTypes from 'prop-types';

function MyComponent({ message }) {
  return <div>{message}</div>;
}

MyComponent.propTypes = {
  message: PropTypes.string.isRequired
};
```

54. Explain the role of immutability in React.

Immutability is a core concept in React, especially when working with state and props. It helps prevent unexpected mutations and enables optimized performance with pure components.