**React.js and React Native Interview Questions**

**React.js**

**1. What is React, and why is it popular for front-end development?**

*React is an open-source JavaScript library developed by Facebook, primarily used for building user interfaces, especially single-page applications (SPAs). One reason for its popularity is its component-based architecture, which promotes reusability and modularity, allowing developers to build complex UIs out of small, isolated, and reusable pieces of code. Another key advantage is its use of a virtual DOM, which optimizes rendering by minimizing direct manipulation of the actual DOM. This leads to improved performance, especially in applications that require frequent updates. Additionally, React’s unidirectional data flow simplifies state management, making it easier to track and debug application state.*

**2. Explain the concept of a component in React.**

*In React, a component is essentially a self-contained, reusable piece of the user interface. Components encapsulate their own structure, styling, and behavior, which makes it easy to compose complex interfaces by combining multiple components. They can be functional or class-based, and they support data binding, lifecycle management, and event handling. By breaking down an application into components, React encourages developers to follow a modular design, improving maintainability and scalability.*

**3. What is the virtual DOM, and how does React use it for rendering?**

*The virtual DOM is an in-memory representation of the actual DOM. React maintains a virtual DOM to track changes in the component tree, which helps in optimizing updates to the real DOM. When a component’s state or props change, React first updates the virtual DOM, calculates the difference (or "diff") between the new and previous versions, and then applies only the necessary changes to the actual DOM. This process, known as reconciliation, minimizes direct DOM manipulation and improves the performance and responsiveness of the application.*

**4. What are props in React, and how do they differ from state?**

*Props, short for "properties," are read-only inputs passed from a parent component to a child component. They allow data to flow down the component hierarchy, enabling components to communicate with each other. Unlike state, which is managed within the component itself and can change over time, props are immutable—once a component receives props, it cannot directly modify them. This helps ensure that data flows in a predictable manner and that child components are purely functional, reacting to data changes rather than altering them.*

**5. How does React handle state, and what is the difference between state and props?**

*React handles state by allowing components to store and manage internal data that can change over time. State is specific to a component and is mutable, meaning the component can update its state as needed. Props, on the other hand, are external, read-only inputs received from a parent component. The primary difference is that state is managed within a component and directly affects the component’s rendering when it changes, while props are passed down from parent to child and are generally used for data display rather than internal data management.*

**6. Explain functional vs. class components. When would you use each?**

*Functional components are simpler and rely on functions to define the component. With the introduction of hooks, they can manage state and side effects, which traditionally only class components could handle. Class components, on the other hand, are ES6 classes that extend from React.Component and include lifecycle methods. Since hooks were introduced in React 16.8, functional components have become the preferred choice for most cases due to their simplicity and readability. Class components may still be used when dealing with legacy code, but for new projects, functional components are generally recommended.*

**7. What are hooks in React? Name a few common hooks and their uses.**

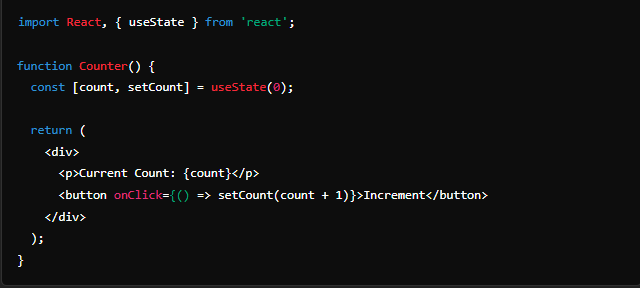
*Hooks are special functions introduced in React 16.8 that enable functional components to use state and other React features without converting to class components. Some commonly used hooks include:*

* **useState**: Manages local component state.
* **useEffect**: Handles side effects like data fetching or updating the DOM.
* **useContext**: Provides access to context values without prop drilling.
* **useReducer**: Manages more complex state logic, similar to Redux reducers.

*Hooks simplify the component lifecycle and make code more modular and maintainable.*

**8. What is useState and how does it work? Provide an example.**

*The useState hook is used to add state to functional components. It returns an array with two elements: the current state value and a function to update it. Each time the update function is called, React re-renders the component with the new state value. Here’s an example:*

*In this example, count is initialized to 0, and clicking the button increments the count by updating the state using setCount.*

**9. Describe useEffect and give an example of how to use it.**

*The useEffect hook is used to manage side effects in functional components, such as data fetching, subscriptions, or DOM manipulation. It runs after the component renders, and you can specify dependencies to control when it should re-run. For example, here’s how to use useEffect to fetch data on initial render:*



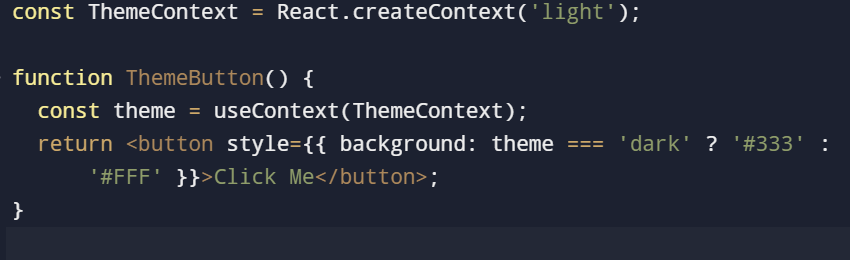
*Here, useEffect fetches data from an API when the component mounts. Since the dependency array is empty, it only runs once.*

**10. What is the difference between useEffect and componentDidMount?**

*componentDidMount is a lifecycle method in class components that runs after the component mounts, similar to how useEffect with an empty dependency array runs once when a functional component mounts. However, useEffect is more versatile as it can run multiple times depending on its dependency array, making it suitable for a wider range of cases, such as responding to prop or state changes. useEffect is also compatible with functional components, which are now more commonly used in modern React development.*

**11. How does the useContext hook work, and when would you use it?**

*The useContext hook allows functional components to access values from a React Context without needing to use props for passing data down multiple levels. This is particularly useful for managing global data such as user authentication status, theme settings, or any shared data across various parts of an application. When using useContext, a component can directly subscribe to context updates, improving readability and simplifying component communication in deeply nested component trees.*



**12. What is the purpose of useReducer, and how does it differ from useState?**

*useReducer is a hook designed for managing more complex state logic, especially when state depends on multiple actions or when updates are tightly related to previous state values. Unlike useState, which provides a simpler state-setting function, useReducer requires a reducer function to handle updates. The reducer function takes the current state and an action, returning a new state based on the action type.*

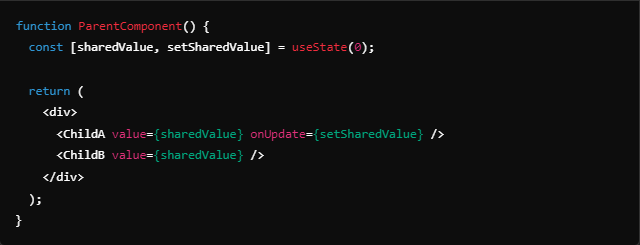
*Use useReducer when the state management is more complex than simple value updates, as in cases of handling form data or managing complex UI state.*



**13. Explain how lifting state works in React and provide an example.**

Lifting state involves moving shared state to the closest common ancestor of components that need to access it. By lifting the state up, both components can read and update the state, allowing them to stay in sync.

Example: If you have two sibling components that both need access to the same state value, you’d move the state to their parent component and pass it down as props.



Here, sharedValue is lifted to the ParentComponent, and both ChildA and ChildB have access to it.

**14. What is React’s Context API, and how is it used?**

*The Context API allows for creating global data that can be accessed by any component in a React app, without prop drilling. It’s useful for values that need to be shared across multiple components, like themes, authentication status, or language preferences.*

*To use Context, you create a Context with React.createContext, then wrap your component tree in a provider component. Any descendant can then access the context value with the useContext hook or the Context.Consumer component.*

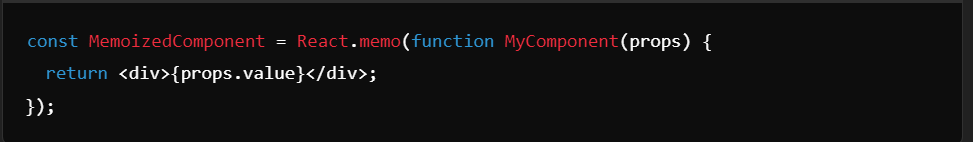
**15. What is Redux, and how does it complement React for state management?**

*Redux is a predictable state container for JavaScript applications, often used with React to manage application-wide state in a centralized store. It works especially well for complex applications with deeply nested components or where multiple components need to share and update state consistently. Redux provides a single source of truth, uses a strict unidirectional data flow, and offers a reliable way to debug state changes over time with tools like Redux DevTools.*

*In React applications, Redux is helpful for managing state that’s global to the app, such as user authentication, application settings, or data that’s frequently used across multiple components.*

**16. What is memoization in React, and how does React.memo work?**

*Memoization in React is a performance optimization technique that prevents re-rendering of components when their props haven’t changed. React.memo is a higher-order component that wraps a functional component and memoizes the rendered output based on props. This means that if the props don’t change, React will skip re-rendering the component, improving performance.*



Here, MyComponent will only re-render if props.value changes.

**17. Explain the concept of higher-order components (HOCs) in React.**

*Higher-order components (HOCs) are functions that take a component as an argument and return a new component with added functionality. HOCs are used for code reuse, such as adding authentication, theming, or logging to components. They allow us to enhance components by wrapping them with extra functionality without modifying their core behavior.*



*In this example, withLogging is an HOC that logs a message whenever the WrappedComponent renders.*

**18. What is React Router, and how does it help with single-page applications (SPAs)?**

*React Router is a library for managing navigation and routing in React applications, allowing developers to create dynamic routing for SPAs. By using React Router, we can define routes that render specific components based on the URL path, enabling a seamless user experience without reloading the page. It allows for nested routing, URL parameters, and history manipulation, which enhances the functionality of single-page applications.*

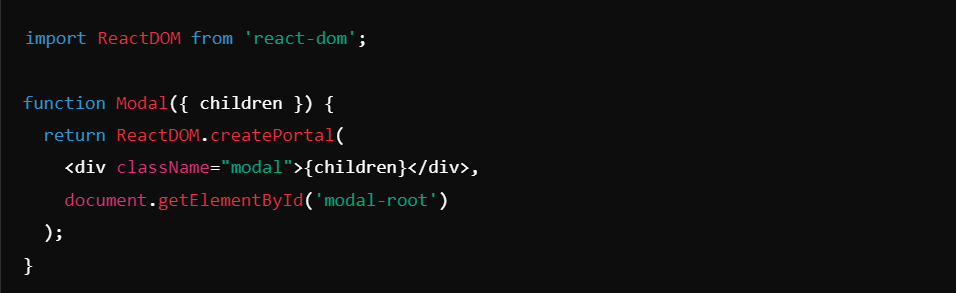
**19. How can you optimize the performance of a React application?**

*Optimizing a React application can involve several techniques, including:*

1. **Using React.memo** to prevent unnecessary re-renders.
2. **Applying useCallback and useMemo hooks** for memoizing functions and values.
3. **Code splitting** with React.lazy and Suspense to load only the necessary parts of the application.
4. **Implementing virtualization** for lists using libraries like react-window to render only visible items.
5. **Optimizing dependency arrays** in hooks to avoid unnecessary executions.
6. **Reducing the size of assets** and using efficient data-fetching strategies like pagination and caching.

**20. What are portals in React, and when would you use them?**

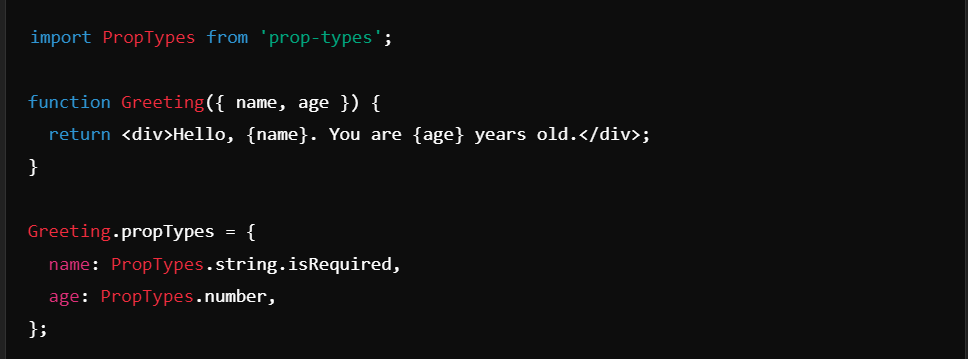
*Portals provide a way to render components outside of the main DOM hierarchy of the parent component. They are often used for rendering elements like modals, tooltips, or dropdowns, which might need to break out of the component structure to avoid being clipped or hidden by overflow styling.*



*Here, Modal renders content into the DOM node with the ID modal-root, outside of its parent’s DOM hierarchy.*

**21. What is PropTypes, and why is it useful in React applications?**

*PropTypes is a type-checking library for React, used to specify the types of props that a component should receive. It helps catch potential bugs by ensuring that the data passed to components matches expected types, making the code more reliable and easier to debug. PropTypes is especially helpful in larger applications where components are reused across different parts of the app and helps developers understand the structure of props at a glance.*



**22. How does server-side rendering (SSR) work in React, and when is it beneficial?**

*Server-side rendering (SSR) in React involves rendering components on the server rather than the client. When a user requests a page, the server generates the HTML content, which is then sent to the browser. This results in faster initial page loads, better SEO (especially for content-heavy applications), and improved performance on low-powered devices. React frameworks like Next.js are commonly used to implement SSR.*

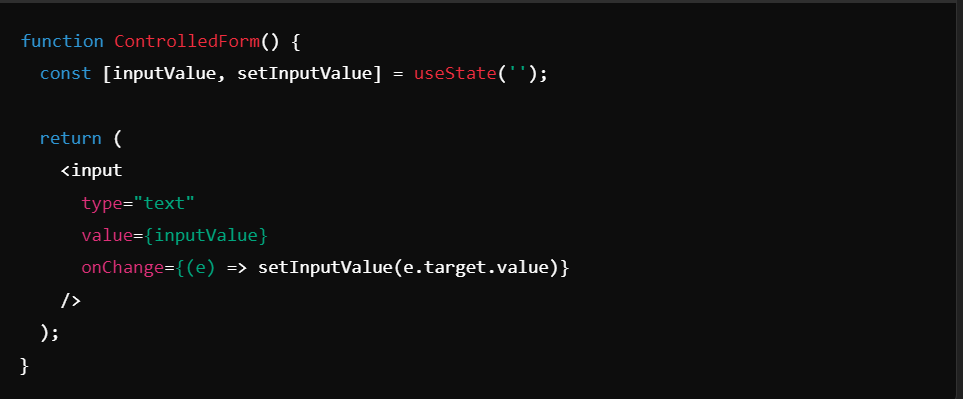
***23. Explain how React.lazy and Suspense work for code-splitting.***

*React.lazy and Suspense enable code-splitting, which is the process of splitting a large application into smaller chunks so that only the necessary code is loaded when needed. React.lazy allows components to be loaded lazily, meaning they’re only fetched when they’re about to be rendered. Suspense is used to wrap React.lazy components and define a fallback UI (like a loading spinner) to display while the component is loading.*

**

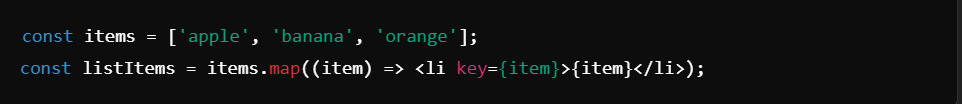
**24. What is a controlled component in React?**

*A controlled component in React is one where the form data is handled by the component’s state rather than directly in the DOM. This means the component’s input fields are bound to state variables, allowing React to control their values. Controlled components make it easier to manage and validate form data, as the state and input values are always synchronized.*

**

**25. What are keys in React, and why are they important?**

*Keys are unique identifiers assigned to elements in lists or dynamic collections in React. They help React efficiently update the DOM by identifying which items have changed, been added, or removed. Without keys, React would re-render all list items on each update, but with keys, React can optimize rendering by targeting specific items that need changes.*

**

**26. How does React handle events, and what is event delegation?**

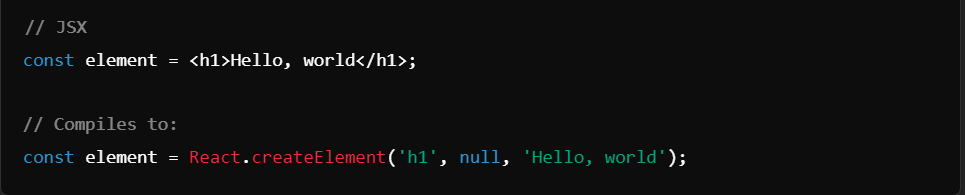
*React handles events using its own synthetic event system, which normalizes events across different browsers. Instead of attaching an event listener to each DOM element, React uses a technique called event delegation, where a single event listener is attached to the root of the DOM. When an event occurs, it “bubbles up” to the root, where React can efficiently handle it, improving performance and reducing memory usage.*

**27. What is reconciliation in React?**

*Reconciliation is the process React uses to compare the virtual DOM with the actual DOM and determine the most efficient way to update the UI. When a component’s state or props change, React creates a new virtual DOM tree, compares it to the previous tree, and calculates the minimal set of changes required. React then applies those changes in a process called “diffing” to update only the affected parts of the real DOM.*

**28. Explain React.createElement. How does JSX compile into this function?**

*React.createElement is the fundamental function used by React to create virtual DOM elements. JSX, the syntax that looks like HTML within JavaScript, is syntactic sugar that compiles down to React.createElement calls. When JSX is processed, it’s translated into React.createElement calls, which then return objects representing elements that React uses to construct the virtual DOM.*

****

**29. How do error boundaries work in React, and how can you create one?**

*Error boundaries are React components that catch JavaScript errors in their child components, preventing the entire app from crashing. They catch errors during rendering, in lifecycle methods, and in constructors of the component tree below them. Error boundaries are created by implementing componentDidCatch and getDerivedStateFromError in class components.*

**

*In this example, if an error occurs in any child component of ErrorBoundary, it will display a fallback message instead of breaking the entire app.*

**10. What is a custom hook, and how do you create one?**

*Custom hooks in React allow you to reuse stateful logic across components. They are JavaScript functions that use built-in hooks to encapsulate and share logic in a more modular way. Custom hooks are especially useful for encapsulating complex logic like data fetching, form handling, or subscription management.*

**

*This useFetchData hook can now be used to fetch data from any API and will return data and loading values that can be used in components.*

**31. Explain the difference between React.PureComponent and React.Component.**

*The main difference between React.PureComponent and React.Component is in how they handle updates. React.PureComponent performs a shallow comparison of props and state to determine if the component needs re-rendering. If neither has changed, it skips the re-render to optimize performance. In contrast, React.Component re-renders whenever its parent re-renders, regardless of whether props or state have changed. PureComponent is ideal when you want to avoid unnecessary re-renders in components with large data or complex trees. However, it only performs a shallow comparison, so nested objects may not work as expected if they’re mutated instead of replaced.*

**32. What is useLayoutEffect, and how does it differ from useEffect? When should you use it?**

*useLayoutEffect is similar to useEffect, but it fires synchronously after all DOM mutations and before the browser paints. This makes it ideal for tasks that need to measure or manipulate the DOM immediately after a render, such as animations or scroll position adjustments. useEffect, on the other hand, is asynchronous and fires after the DOM has painted, making it preferable for side effects that don’t affect layout.*

*Use useLayoutEffect sparingly and only for cases where you need to ensure the DOM is up-to-date before the user sees it. Excessive use can block the rendering process, potentially impacting performance.*

**33. How do React’s Context API and Redux differ? In what scenarios would you prefer one over the other?**

*The Context API provides a way to pass data through the component tree without manually passing props at each level. It’s best suited for relatively simple, global state like theme or language, where minimal logic is involved. Redux, on the other hand, is a more comprehensive state management solution that helps maintain complex, application-wide state. Redux’s strict immutability, middleware support, and developer tools make it a better choice for applications with complex state management needs, multiple interdependent data sources, or more advanced side-effect handling.*

*Generally, if the global state is relatively simple and doesn’t require complex logic or debugging tools, the Context API is a good choice. For larger applications with complex state logic, Redux is often the better choice.*

**34. What is the significance of the dependency array in useEffect? How does it impact component behavior?**

*The dependency array in useEffect tells React when the effect should re-run. React only re-runs the effect when the dependencies in this array change, optimizing the component’s performance. If the array is empty, the effect runs only once, similar to componentDidMount. Without dependencies, the effect re-runs on every render. Failing to include necessary dependencies can lead to stale closures, where the effect uses outdated values.*

*Choosing dependencies carefully is crucial for preventing unintended behaviors, such as infinite re-renders or using outdated data.*

**35. Explain how React.lazy and Suspense help with code-splitting. Can they handle server-side rendering?**

*React.lazy and Suspense enable code-splitting by loading components only when they are needed, reducing the initial bundle size. With React.lazy, components are fetched lazily as they’re rendered. Suspense allows you to specify fallback content (such as a loading spinner) while the component is loading.*

*However, React.lazy and Suspense currently work only for client-side rendering. For server-side rendering, libraries like Next.js handle code-splitting and data fetching on the server, providing SSR-ready alternatives to React.lazy.*

**36. What are forward refs (React.forwardRef), and when would you use them?**

*React.forwardRef allows a parent component to directly access the child component’s reference, which can be useful for interacting with DOM elements or APIs in the child component. This is commonly used when you want to control an element from a parent, like focusing an input, triggering a modal, or implementing animations.*

**

**37. How does error handling differ between synchronous and asynchronous code in React?**

*In synchronous code, errors can be caught using try-catch blocks or error boundaries at the component level. React’s error boundaries handle errors in lifecycle methods and render functions for synchronous code. However, error boundaries don’t catch errors in asynchronous code, such as in setTimeout or async calls.*

*For asynchronous code, errors should be handled within the asynchronous function itself, typically by using .catch() or surrounding await statements in a try-catch block. Custom hooks or error-handling functions are often useful for handling async errors across components.*

**38. Explain React’s StrictMode. What are its benefits, and how does it affect development?**

*StrictMode is a development-only feature in React that helps identify potential issues in an application. It highlights unsafe lifecycles, warns about deprecated APIs, and double-invokes certain methods to help identify side effects that may cause issues. It’s beneficial for maintaining code quality and ensuring future compatibility with React’s features.*

*While StrictMode does not affect production behavior, it is crucial for detecting issues in development and helping teams write safer, more optimized code.*

**39. How does Concurrent Mode in React work, and what problems does it solve?**

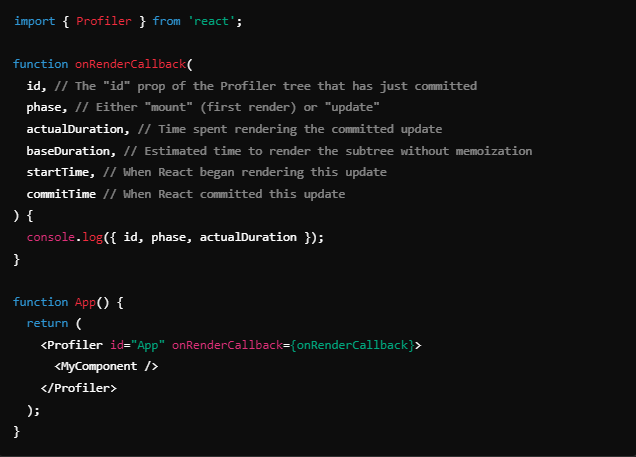
*Concurrent Mode allows React to prepare multiple tasks in the background without blocking the main thread. This means React can prioritize rendering important updates, such as user interactions, and can pause or abort less critical updates when necessary. It’s useful for managing complex UI updates smoothly, even when dealing with data-heavy operations or intensive computations.*

*Concurrent Mode is not fully released yet, but it has features like useTransition, which provides better control over UI transitions and rendering priority, improving responsiveness in applications.*

**40. What is the Profiler API, and how can it help optimize a React app?**

*The Profiler API in React is a tool for measuring component render times and analyzing performance bottlenecks. It records data on when components mount, re-render, and unmount, helping developers identify unnecessary re-renders or expensive operations. This data is accessible in the React DevTools, making it easier to target and optimize parts of an application that may be impacting performance.*

*For instance, you can wrap components in the Profiler to measure and analyze their render times:*

**

*Using the Profiler, you can measure and improve your app’s performance based on concrete data.*

**41. How can you conditionally apply hooks in React? Why can’t hooks be conditional in regular circumstances?**

*In React, hooks cannot be conditionally applied because they rely on a strict order to ensure consistent behavior across renders. React depends on the order in which hooks are called to track state and effects correctly. Conditional hooks would break this order, causing errors and unpredictable behavior.*

*However, you can create conditional logic inside hooks by moving the condition inside the hook itself. For instance, use conditional logic within a useEffect or custom hook rather than conditionally applying the hook at the component level.*

**42. What are compound components, and how would you build one in React?**

*Compound components are a pattern in React where components are designed to work together to build more complex components. They allow for flexible structure and configuration, as components can share state and logic. For example, a <Dropdown> component might include <Dropdown.Item>, <Dropdown.Toggle>, and <Dropdown.Menu> as compound components that interact.*

*To create a compound component, you can use React’s Context API to share state between child components, or pass props from a parent component to children for customization.*

**

**43. Explain the purpose of React’s useImperativeHandle hook. How does it interact with refs?**

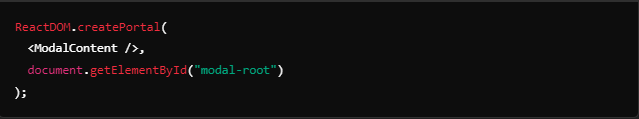
*The useImperativeHandle hook customizes the instance value that a parent component receives when using ref on a child component. It’s useful for exposing specific methods or properties from the child component to the parent, typically for DOM manipulation or other imperative operations.*

*When using React.forwardRef with useImperativeHandle, you can control what the ref object exposes to its parent, providing an interface for only specific functionalities.*

**

**44. What is a React portal, and how does it work? Provide an example use case.**

*A React portal allows you to render a child component into a DOM node outside of the component’s immediate parent hierarchy. Using ReactDOM.createPortal, you can specify where in the DOM the child should be rendered. Portals are commonly used for modals, tooltips, and pop-ups, where you want to keep the component visually on top of the rest of the UI while it remains part of the React component tree.*

**

**45. How do you manage accessibility (a11y) in a React application?**

*Managing accessibility involves using semantic HTML elements, ARIA roles, keyboard navigation, and screen reader support. React’s accessible APIs, such as aria-\* attributes, make it easier to add accessibility features. Libraries like react-aria provide components and hooks for accessible UI.*

*Key accessibility practices include providing alt text for images, using landmarks for screen readers, ensuring keyboard focus states, and testing with screen readers.*

**46. What is tree-shaking, and how does it affect a React app’s build size?**

*Tree-shaking is an optimization technique that removes unused code from the final JavaScript bundle. It relies on ES6 module imports and helps reduce the bundle size by eliminating unused exports. Tree-shaking is particularly beneficial in large applications, ensuring that only the code actually used by the app is included in the final build.*

*React apps benefit from tree-shaking because it reduces unnecessary code from dependencies and unused component files, resulting in smaller and faster bundles.*

**47. What are controlled and uncontrolled components in React? When would you use each?**

*Controlled components rely on React state to manage their values, with state changes triggered by user input. In contrast, uncontrolled components use the DOM to manage their state internally, and values are accessed via refs.*

*Controlled components are generally preferred for form handling due to their predictable nature, while uncontrolled components are suitable for simpler forms or cases where immediate access to the value is unnecessary.*

**48. How would you optimize a React app with large lists of data (e.g., with react-window)?**

*react-window optimizes large lists by virtualizing them, rendering only the visible portion in the viewport and updating the DOM as the user scrolls. This drastically reduces memory and CPU usage.*

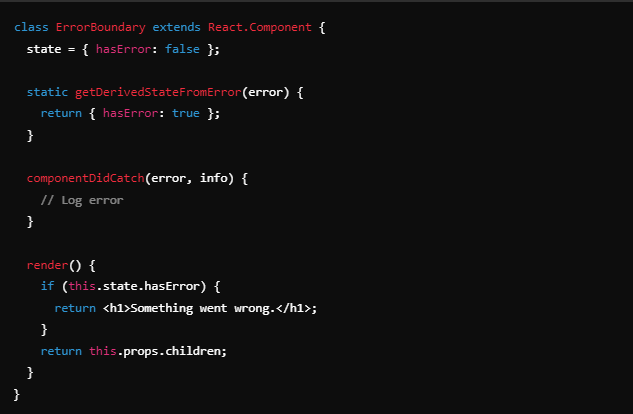
*Other techniques include memoizing list items with React.memo and using shouldComponentUpdate or useCallback to prevent unnecessary re-renders.*

**49. Explain the use of shouldComponentUpdate in a class component and how it affects performance.**

*shouldComponentUpdate is a lifecycle method in class components that allows developers to control whether a re-render should occur when props or state change. It’s useful for optimizing performance by preventing unnecessary re-renders. By returning false, you can skip rendering for updates that don’t require UI changes, saving on rendering cost.*

**50. What is an Error Boundary in React, and how do you create one?**

*Error boundaries are components that catch JavaScript errors in their children’s component tree, log them, and display a fallback UI. They’re created using a class component that implements componentDidCatch and getDerivedStateFromError.*

**

**51. How can you memoize functions in React? Explain the difference between useMemo and useCallback.**

*Memoization in React improves performance by caching the results of expensive calculations or functions. useMemo is used for memoizing computed values, whereas useCallback memoizes function instances. useCallback is often used when passing functions to child components, ensuring the function reference remains consistent.*

**52. How does reselect work with Redux for state optimization?**

*reselect creates memoized selectors, which compute derived state based on Redux state and cache the results. This prevents recalculation and re-renders when the relevant parts of the state haven’t changed. reselect is especially useful for complex, derived data that would otherwise need recalculating on every state change.*

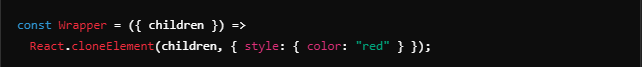
**53. How would you handle session or token expiration in a React application?**

*Handling session expiration typically involves detecting token expiration based on its lifetime or by checking with the server. You can set up interceptors in axios or other HTTP clients to detect 401 responses and prompt re-authentication or refresh the token. State management libraries like Redux can also help manage global session state.*

**54. Explain how React.cloneElement works and a use case where it’s useful.**

*React.cloneElement creates a copy of a React element, allowing you to modify its props or children. It’s commonly used to customize children passed to a component, such as adding event handlers or injecting additional props.*

*Example use case: Wrapping child components with additional behavior:*

**

**55. What is the purpose of React.Children utility?**

*React.Children provides utilities for working with component children, especially useful when handling an unknown number of children. Functions like React.Children.map, React.Children.forEach, and React.Children.toArray allow for safe manipulation of children components.*

*This utility is essential when building flexible components, such as custom layout components, where you need to process each child individually.*

**56. Explain the use of CSS-in-JS libraries (e.g., styled-components) and their advantages in React apps.**

*CSS-in-JS libraries like styled-components allow developers to write CSS directly in JavaScript files. With styled-components, you create styled React components with encapsulated styles, making it easy to keep style and logic together, which improves component modularity and reusability. It eliminates the need for globally scoped styles and helps avoid conflicts by generating unique class names for each styled component.*

*Advantages of CSS-in-JS include:*

* *Scoped styling that applies only to the component where it's used, preventing global styles from unintentionally affecting other parts of the app.*
* *Dynamic styling based on props or application state, enabling styles to change as the component’s data changes.*
* *Theme support, making it easier to create and apply consistent themes across an application.*

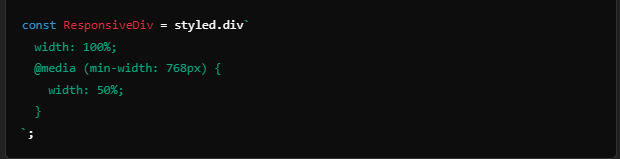
*Example of styled-components:*

**

**57. How would you handle media queries and responsive design in React?**

*For responsive design, there are a few common approaches in React:*

1. ***CSS Media Queries****: With CSS-in-JS libraries like styled-components, you can add media queries directly to components for responsive styles.*

**

***2. JavaScript-based Media Queries****: Libraries like react-responsive enable you to apply media queries in JavaScript by using conditions in components to adapt their layout.*

***3. CSS Frameworks and Grid Systems****: Using frameworks like Bootstrap or CSS Grid can help with responsiveness through pre-defined grid classes or grid layouts.*

***4. Custom Hooks for Responsive Values****: For fully dynamic layouts, you can use custom hooks (e.g., useWindowDimensions) to check window dimensions and apply conditional rendering or styling.*

**58. Describe a scenario where you would use React.Context instead of Redux.**

*The React.Context API is ideal for managing state that needs to be accessible across a few components or specific areas of an application but does not require the complex structure and middleware that Redux provides. For example, managing themes, authenticated user data, or language preferences are good use cases for React.Context, where you only need to pass values to specific sections without extensive action handling or reducer logic.*

*In cases where global state management is minimal and primarily read-only or where data changes are infrequent, React.Context can simplify state management without introducing Redux.*

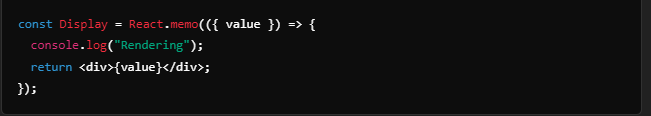
**59. How does React.memo work, and when should you use it in functional components?**

*React.memo is a higher-order component that memoizes the result of a component to prevent unnecessary re-renders. When a component wrapped in React.memo receives the same props on subsequent renders, React skips rendering the component and reuses the previous render output.*

*You should use React.memo when:*

* *You have a functional component that receives props and does not rely heavily on re-renders.*
* *The component re-renders frequently, but the props often remain the same (e.g., pure display components).*

*Example of React.memo usage:*

**

**60. What are some common anti-patterns in React development?**

*Avoiding anti-patterns is important for maintainable and efficient React code. Here are a few common anti-patterns:*

1. ***Directly Modifying State****: Mutating state directly, rather than creating a new copy, can cause issues with React’s rendering cycle. Use setState for updates to ensure changes are tracked.*
2. ***Excessive Use of Inline Functions****: Using inline functions like onClick={() => handleClick()} can cause unnecessary re-renders since a new function is created on every render. Instead, use useCallback to memoize functions when needed.*
3. ***Overusing Context for Large State Management****: While the Context API is useful, overloading it with large application state can lead to performance bottlenecks, as any change in context data re-renders all components consuming it. For complex state, consider Redux or other state management libraries.*
4. ***Ignoring Dependency Arrays in Hooks****: Leaving out dependencies or providing incorrect dependencies in hooks like useEffect can cause unexpected behaviors, such as infinite loops or missing updates.*
5. ***Re-rendering Large Lists Without Optimization****: Rendering large lists without using libraries like react-window or pagination can lead to performance issues, especially on low-performance devices. Virtualization techniques should be used when displaying many items.*
6. ***Not Cleaning Up in useEffect****: Leaving side effects, like subscriptions or event listeners, active without cleanup can lead to memory leaks and performance issues. Always include clean-up code in useEffect where necessary.*

**React Native**

**1. What is React Native, and how does it differ from React?**

*React Native is a framework that enables building mobile applications for iOS and Android using JavaScript and React. Unlike standard React, which targets the DOM for web applications, React Native translates React components to native components, like UIView on iOS and View on Android, to provide a native look and feel.*

**2. How does React Native bridge the gap between JavaScript and native code?**

*React Native uses a "bridge" to communicate between JavaScript and native code (Java, Objective-C/Swift). The bridge allows JavaScript code to call native modules and vice versa, which enables access to device features (camera, GPS, etc.) while keeping JavaScript and native code separate yet able to work together efficiently.*

**3. Explain the concept of a virtual DOM in React Native.**

*The virtual DOM in React Native is a lightweight copy of the actual UI tree. Changes in components trigger virtual DOM updates, and React Native uses a diffing algorithm to apply only necessary updates to the actual native views. This minimizes re-rendering and optimizes performance, similar to React on the web but with a focus on native UI components.*

**4. How do you set up navigation in a React Native application?**

*React Navigation is commonly used for setting up navigation. It provides navigators like StackNavigator, TabNavigator, and DrawerNavigator to manage screens and navigation. You install the @react-navigation libraries, configure navigators, and wrap your app in a NavigationContainer.*

**5. What is the difference between StackNavigator and TabNavigator in React Navigation?**

*StackNavigator organizes screens in a stack, allowing users to navigate back to previous screens like a stack of cards. TabNavigator displays tabs, often at the bottom of the screen, for switching between major app sections without a "back" concept. Both provide different navigation experiences based on app requirements.*

**6. How does StyleSheet work in React Native, and how does it differ from CSS in React?**

*React Native’s StyleSheet is an abstraction for defining styles that closely resemble CSS but are optimized for mobile. Unlike CSS, styles in React Native are inline and do not support cascading or inheritance. Properties and values may differ slightly, and everything uses Flexbox for layout.*

**7. Explain the purpose of Flexbox in React Native layout design.**

*Flexbox is the layout model used in React Native to create responsive designs. It allows items to be arranged in rows or columns and easily aligned, spaced, and resized within their containers, which is crucial for building adaptable mobile layouts.*

**8. What are the different ways to handle state in React Native?**

*State can be managed in React Native using useState and useReducer for local state, Context API for global state, or libraries like Redux and MobX for more complex, centralized state management across the app.*

**9. What is AsyncStorage, and how do you use it for persistent storage?**

*AsyncStorage is a simple key-value storage system for storing data persistently on the device, similar to localStorage in web apps. You can store small amounts of data, like user preferences or tokens, by using AsyncStorage.setItem() to save and AsyncStorage.getItem() to retrieve.*

**10. Explain the purpose of FlatList and SectionList in React Native.**

*FlatList efficiently renders large lists by lazily loading items, while SectionList is used for grouped lists with section headers. Both provide optimized performance, but SectionList is ideal for displaying data in categories, while FlatList is better for simpler lists.*

**11. How does useEffect work differently in React Native compared to React?**

*useEffect behaves similarly in both, managing side effects (like fetching data) based on dependencies. However, React Native apps might rely more on lifecycle events due to device-specific actions, like GPS location or hardware features, which may impact the effect timing or dependencies.*

**12. What is Animated API, and how is it used for animations in React Native?**

*The Animated API provides a flexible way to create smooth animations. It supports animations on properties like position, scale, and opacity using methods like Animated.timing, Animated.spring, and allows for complex sequences or parallel animations.*

**13. Explain the purpose of PanResponder in React Native.**

*PanResponder enables handling complex touch gestures, such as dragging and swiping. It tracks touch movements and provides callback functions for gesture events, allowing you to build custom gesture-based interactions, like sliders or draggable items.*

**14. What are gesture handlers, and why are they important in React Native?**

*Gesture handlers, provided by react-native-gesture-handler, offer advanced gesture handling with native feel. They prevent issues with touch performance and gesture conflicts that can occur in the default React Native touch system, especially when working with complex animations or nested touchable elements.*

**15. How do you handle media (e.g., images and videos) in React Native?**

*For images, use the Image component, supporting caching and resizing. For videos, react-native-video provides a customizable video player. Both should be optimized in terms of size and resolution to minimize memory usage and loading time.*

**16. Explain the difference between TouchableOpacity, TouchableHighlight, and TouchableNativeFeedback.**

*TouchableOpacity provides a fading effect, TouchableHighlight darkens or highlights the background, and TouchableNativeFeedback shows native ripple effects (mainly for Android). Each offers different feedback styles for a more interactive experience based on platform or design.*

**17. What is Expo, and how does it simplify React Native development?**

*Expo is a toolkit for building React Native apps without needing to set up native code. It simplifies development by providing libraries, testing tools, and a managed workflow, allowing for faster prototyping. However, some native modules may be restricted without "ejecting" from Expo.*

**18. How would you handle device orientation changes in React Native?**

*Use the Dimensions API to listen for screen size changes and adjust layouts accordingly, or useWindowDimensions for dynamic layout updates. You can also use react-native-orientation for more control over orientation events and lock orientation on specific screens if needed.*

**19. Explain the purpose of NativeModules in React Native.**

*NativeModules allow JavaScript code to interact with native functionalities or libraries not directly available in React Native. By creating native modules in Swift, Objective-C, or Java, and exposing them through NativeModules, you can access device-specific features or custom functionalities.*

**20. How do you manage form validation in React Native?**

*Form validation can be managed with libraries like Formik and Yup, which simplify state management and validation logic. Alternatively, react-hook-form provides a lightweight way to handle forms in functional components, making validations and error handling more manageable.*

**21. What are the best practices for optimizing React Native app performance?**

*To optimize performance in React Native, you can follow these best practices:*

* *Use FlatList and SectionList for rendering large lists efficiently.*
* *Leverage the shouldComponentUpdate lifecycle method in class components or React.memo in functional components to prevent unnecessary re-renders.*
* *Optimize images by using appropriate sizes and formats (e.g., PNG, JPG).*
* *Minimize the number of state updates and use functional updates when possible.*
* *Use native modules for performance-intensive tasks.*
* *Profile the app using tools like the React Native Performance Monitor and the React DevTools Profiler to identify bottlenecks.*

**22. How can you enable offline support in a React Native app?**

*Offline support can be implemented using libraries like AsyncStorage for local data storage, redux-persist to save Redux state, and react-query or axios with retry logic for managing API calls. Additionally, service workers or caching strategies can be employed to handle requests and store data for offline access.*

**23. What are the differences between Android and iOS development in React Native?**

*Differences in React Native development for Android and iOS include:*

* *UI Components: Some components behave differently or are styled uniquely on each platform (e.g., TouchableNativeFeedback on Android vs. TouchableOpacity on iOS).*
* *Navigation: The navigation paradigm may differ, with different conventions for back navigation and headers.*
* *Native Modules: Platform-specific code is often required to access native APIs, leading to variations in implementation.*
* *Testing and Debugging: Tools may differ slightly, requiring platform-specific adjustments.*

**24. How does react-native-reanimated work, and why is it used?**

*react-native-reanimated is a library that provides a more powerful and performant way to create animations in React Native. It allows for animations to be declared and run on the UI thread, leading to smoother animations by avoiding the overhead of the JavaScript thread. This is particularly useful for complex gestures and animations that require high performance.*

**25. Explain how React Native handles platform-specific code with Platform API.**

*React Native's Platform API allows you to write platform-specific code by checking the platform at runtime. You can use Platform.OS to conditionally render components or apply styles based on whether the app is running on iOS or Android. For example:*

*javascript*

*Copy code*

*import { Platform } from 'react-native';*

*const buttonStyle = Platform.OS === 'ios' ? styles.iosButton : styles.androidButton;*

**26. What is the purpose of useRef in React Native?**

*The useRef hook is used to persist values across renders without causing re-renders. In React Native, it's commonly used to access and interact with a DOM element or a child component directly, such as controlling focus on an input field or triggering animations.*

**27. How do you handle network requests in React Native?**

*Network requests in React Native can be handled using the built-in fetch API or libraries like axios. For better management of requests, especially in larger apps, you can use hooks like useEffect to fetch data and state management solutions (like React Context, Redux, or MobX) to store and manage the data.*

**28. What is Fast Refresh in React Native, and how does it improve development?**

*Fast Refresh is a feature that provides instantaneous feedback during development by preserving the component state when you edit your code. It allows developers to see changes in real-time without losing their current application state, which significantly enhances the development experience.*

**29. What are deep linking and universal links in the context of React Native?**

*Deep linking allows you to link directly to a specific content or feature within your app from external sources (like URLs). Universal links are a type of deep link that works on both iOS and Android, allowing users to open the app directly from a web link. In React Native, libraries like react-navigation and react-native-deep-linking can facilitate implementing deep links.*

**30. How do you integrate native modules in a React Native project?**

*To integrate native modules, you need to create the module in native code (Java for Android, Objective-C/Swift for iOS), expose it to JavaScript using the React Native Bridge, and then call it from your React Native components. This typically involves defining the module, registering it, and using NativeModules in your JavaScript code.*

**31. What are the limitations of React Native compared to fully native development?**

*Limitations of React Native include:*

* *Performance: React Native may not achieve the same performance as fully native apps, particularly for CPU-intensive tasks.*
* *Access to Native APIs: Some native APIs may not be available in React Native or may require additional bridging, making it more cumbersome to implement certain features.*
* *Third-party Libraries: You may encounter fewer libraries compared to native ecosystems, or some may not be maintained well.*
* *Complex UIs: For very complex UIs or animations, fully native development may be more suitable.*

**32. How do you handle notifications in a React Native app?**

*Notifications can be handled using libraries like react-native-push-notification or Firebase Cloud Messaging (FCM) for both local and remote notifications. Set up involves configuring the necessary permissions, handling incoming notifications, and creating notification handlers for different app states (foreground, background, or killed).*

**33. Explain the use of SafeAreaView in React Native.**

*SafeAreaView is a component that renders content within the safe area boundaries of a device, which is particularly important on devices with notches or rounded corners. It helps prevent content from being clipped or obscured by system UI elements, ensuring a better user experience.*

**34. What is the difference between useCallback and useMemo in React Native?**

*useCallback and useMemo are both hooks that optimize performance by memoizing values. useCallback returns a memoized version of a callback function that only changes if its dependencies change, preventing unnecessary re-renders of child components. useMemo, on the other hand, memoizes a computed value and re-calculates it only when its dependencies change. Use useCallback for functions and useMemo for values.*

**35. How does TypeScript integrate with React Native?**

*TypeScript integrates with React Native by adding type safety to your components and props. You can set up TypeScript by creating a new React Native project with TypeScript templates or adding TypeScript to an existing project. Type definitions are available for React Native, allowing for improved code completion, error detection, and overall maintainability.*

**36. What is react-native-gesture-handler, and how does it improve interactions?**

*react-native-gesture-handler is a library that provides a more robust gesture system for React Native. It replaces the default touch handling system with a native gesture system, improving performance and enabling complex gesture recognition. It allows for handling gestures more intuitively, making it easier to create rich interactions without conflicts.*

**37. How do you implement push notifications in React Native?**

*To implement push notifications, you can use libraries like react-native-push-notification or integrate with Firebase Cloud Messaging (FCM). The process involves setting up your app to handle notifications, configuring permissions, and writing code to handle incoming notifications, as well as displaying them to the user.*

**38. What are code-push updates, and how do they work in React Native?**

*CodePush is a service that allows you to deploy mobile app updates directly to users' devices without going through the app store approval process. It enables you to push updates for JavaScript, images, and other assets. You integrate it by using the react-native-code-push package and can configure it to automatically check for updates when the app starts or when it resumes from the background.*

**39. Explain the purpose of AppRegistry in React Native.**

*AppRegistry is the entry point for React Native applications. It is used to register the main application component and manage the application lifecycle. AppRegistry ensures that your application runs in the correct environment and allows you to run multiple apps from the same JavaScript bundle, useful for developing and testing multiple apps.*

**40. What is react-native-paper, and how does it help in building UI?**

*react-native-paper is a library that provides a collection of customizable and accessible Material Design components for React Native. It simplifies the UI building process by offering pre-built components like buttons, cards, and dialogs, adhering to Material Design guidelines, which helps in creating a consistent and visually appealing interface quickly.*

**41. Explain the React Native architecture and how it interacts with native code*.***

*React Native uses a bridge architecture to enable communication between JavaScript and native modules on iOS and Android. The JavaScript code runs on a separate JavaScript thread, which communicates with the native platform through a bridge. The bridge serializes JavaScript commands and passes them to the native side, which then executes the commands using native UI components or modules.*

*This structure allows React Native to render platform-specific UI elements and access native features. While this architecture enables cross-platform development, it can impact performance due to the bridge’s asynchronous communication.*

**42. How does the Fabric renderer improve React Native performance?**

*The Fabric renderer, part of the new React Native architecture, improves performance by reducing the bridge bottleneck and enabling synchronous communication between JavaScript and native modules. Fabric enhances concurrency, allowing the UI to remain responsive even when handling complex updates.*

*It also integrates with React's Concurrent Mode, allowing more granular control over rendering priorities, so only essential components update while others defer their updates, leading to smoother user experiences.*

**43. What are the differences between React Native's native modules and TurboModules?**

*Native modules are React Native’s way of accessing native platform features, but they rely on the bridge, causing potential performance issues due to asynchronous communication.*

*TurboModules, part of the new architecture, use a different approach that supports synchronous communication where possible. They enhance performance by allowing JavaScript to load only the modules needed at runtime (lazy loading) and enable faster, more direct communication with native code, reducing bridge latency.*

***44. Explain the role of Hermes in React Native and its benefits.***

*Hermes is a JavaScript engine optimized for React Native, especially on Android. It improves performance by providing faster startup times, reduced memory usage, and optimized bytecode that speeds up the execution.*

*Benefits of using Hermes:*

* *Reduced app size due to optimized bytecode.*
* *Faster app startup time because of bytecode pre-compilation.*
* *Lower memory consumption, which improves app performance on low-end devices.*

*To enable Hermes, add the enableHermes flag to the build.gradle file.*

**45. How would you troubleshoot a memory leak in a React Native app?**

*To troubleshoot memory leaks:*

1. *Use tools like Xcode’s Instruments (for iOS) or Android Studio’s Profiler to monitor memory usage.*
2. *Identify and eliminate components or hooks that aren’t cleaned up properly, such as unsubscribed listeners or unmounted components.*
3. *Check for uncleaned useEffect side-effects and ensure event listeners are removed in cleanup functions.*
4. *Use profiling tools like why-did-you-render to detect unnecessary re-renders.*

**46. What are code-push updates in React Native, and how do you implement them?**

*Code-push is a service from Microsoft that allows for over-the-air (OTA) updates to React Native apps. It enables developers to push updates directly to users without requiring an app store update. Code-push can only update JavaScript and asset files, not native code.*

*To implement:*

1. *Install react-native-code-push and configure it in the app.*
2. *Push updates using the CodePush CLI.*
3. *Integrate the CodePush API into the app to check for updates on launch.*

**47. How does the bridge work in React Native, and what is its impact on performance?**

*The bridge in React Native is an asynchronous communication layer between JavaScript and native code. It allows the JavaScript thread to send commands to native components and receive responses back.*

*Impact on performance: Since the bridge relies on asynchronous, serialized messages, it can create a bottleneck, especially for heavy tasks or animations, leading to potential lag in complex apps.*

**48. What are best practices for handling API calls in React Native?**

*Best practices include:*

* *Using a library like Axios or fetch for making HTTP requests.*
* *Storing API responses in a state management library (e.g., Redux, React Query) to avoid redundant network calls.*
* *Utilizing useEffect to trigger requests on component mount and useCallback to avoid unnecessary re-fetching.*
* *Implementing proper error handling and loading states for better UX.*

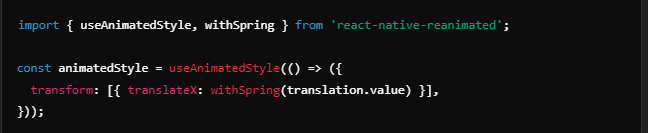
**49. How do you handle gesture conflicts in React Native using react-native-gesture-handler?**

*To resolve gesture conflicts with react-native-gesture-handler:*

* *Wrap root components in GestureHandlerRootView.*
* *Use priority settings like waitFor and simultaneousHandlers to specify the order of gesture handling.*
* *Apply PanGestureHandler, TapGestureHandler, etc., directly on conflicting views and manage state to determine which gestures to respond to.*

**50. Explain how to use react-native-reanimated for complex animations.**

*With react-native-reanimated, you define animations as declarative blocks, allowing the library to handle them efficiently on the native side. This avoids the bridge and improves performance for complex animations.*

**

**51. What is a NativeEventEmitter in React Native, and when would you use it?**

*NativeEventEmitter is used to listen for events sent from native modules. It's commonly used when integrating native SDKs (e.g., handling barcode scans, GPS updates) and allows JavaScript to respond to real-time data from the native side.*

**52. How would you implement deep linking in a React Native application?**

*Deep linking allows you to open specific screens in an app using a URL.*

1. *Configure URL schemes in the app’s native files (e.g., AndroidManifest.xml for Android and Info.plist for iOS).*
2. *Use @react-navigation/deep-linking in a navigation stack to handle URL patterns and route users to specific screens.*

**53. What is the BackHandler API, and how can you control navigation behavior with it?**

*BackHandler is an API in React Native to listen for the hardware back button on Android.*

*To control navigation, use BackHandler.addEventListener to override the default behavior. For example, you can prompt a user before closing the app or navigate to a specific screen.*

**54. Explain how to set up push notifications with react-native-push-notifications.**

*To set up push notifications:*

1. *Install react-native-push-notifications and follow platform-specific setup steps (e.g., configuring Firebase Cloud Messaging for Android).*
2. *Register for notifications in the app code and handle permissions.*
3. *Use PushNotification.localNotification to schedule local notifications or PushNotification.configure to handle remote notifications.*

***55. How does SafeAreaView improve user experience across different devices?***

*SafeAreaView helps avoid content overlap with notches, status bars, and other screen obstructions on various devices. It adds padding around the screen content, automatically adjusting based on device-specific safe areas, like the top notch on iPhone X and beyond. This enhances user experience by ensuring that important content is not obscured, providing a consistent layout across different devices.*

**56. What is Fast Refresh, and how does it differ from Hot Reload in React Native?**

*Fast Refresh is React Native's tool for instantly updating the UI with code changes, combining the best of live reloading and hot reloading. Unlike Hot Reload, which sometimes failed to maintain state or could be unreliable, Fast Refresh is more consistent and preserves the component state across reloads by default. It improves productivity by giving immediate feedback without a full reload, making it particularly effective for iterative UI development.*

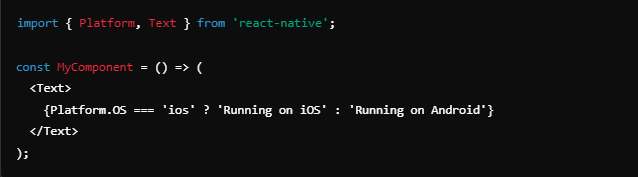
**57. How do you debug performance issues in a React Native app?**

*To debug performance issues:*

1. *Use the React DevTools Profiler to analyze component render times.*
2. *Enable Hermes (on Android) and inspect the performance traces.*
3. *Use Xcode Instruments (iOS) or Android Studio Profiler for memory and CPU usage.*
4. *Check for unnecessary re-renders using the why-did-you-render library.*
5. *Optimize FlatList and ScrollView components by avoiding anonymous functions as props and using PureComponent or React.memo for child components.*

**58. What are platform-specific code practices in React Native, and how do you use Platform.OS?**

*Platform-specific code practices involve writing conditional logic or files tailored for Android and iOS differences. Using Platform.OS, you can implement code based on the current platform. For example:*

*Files can also be named with .ios.js or .android.js to automatically target platform-specific versions of a component.*

**59. How do you set up dynamic themes in React Native using a theme provider?**

*To set up dynamic themes:*

1. *Define a ThemeProvider (e.g., from styled-components or @react-navigation/native) that provides light and dark theme objects.*
2. *Use context or state to toggle between themes, setting colors and styles dynamically.*
3. *Wrap the application with ThemeProvider and access theme values through the useTheme hook or styled components.*

*This allows users to switch themes dynamically, enhancing accessibility and UX.*

**60. Explain FlatList optimizations and when to use SectionList instead.**

*FlatList optimizations include using keyExtractor for stable keys, setting initialNumToRender, and enabling windowSize and maxToRenderPerBatch to improve performance with large datasets. FlatList is best for simple lists without sections.*

*SectionList should be used when data needs to be grouped into sections, like a list with headers. It has built-in support for rendering section headers and is optimized for structured data, which enhances performance and user navigation in structured lists.*

**61. What are TouchableOpacity and TouchableNativeFeedback, and how do they differ?**

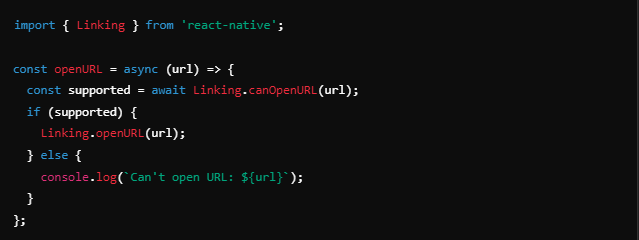
*Both TouchableOpacity and TouchableNativeFeedback provide touchable elements but with different styles and behaviors:*

* *TouchableOpacity is cross-platform, with a fading opacity effect upon press.*
* *TouchableNativeFeedback uses native touch feedback (like ripple effects) on Android, providing a more platform-native feel.*

*Use TouchableOpacity for a consistent look across platforms and TouchableNativeFeedback when you want Android-native touch feedback.*

**62. What is the Linking API in React Native, and how would you use it to open URLs?**

*The Linking API in React Native allows you to interact with external URLs and deep link into other apps. To open a URL:*

*Use Linking for external URLs, phone numbers, and deep links to enhance inter-app navigation.*

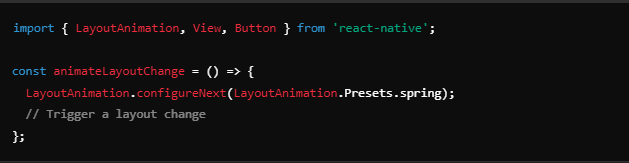
**63. Explain the purpose of AppRegistry in a React Native app.**

*AppRegistry is the entry point for all React Native apps, responsible for registering the root component, which starts the app. AppRegistry.registerComponent tells React Native which component to render as the main app, linking JavaScript code with the native app.*

*It’s also used for rendering other types of applications, such as rendering into a root view for web support or handling headless tasks.*

**64. How would you implement animations using the LayoutAnimation API?**

*LayoutAnimation allows easy transitions for layout changes in React Native. It automatically animates views when their layout changes (e.g., size or position).*

*Use LayoutAnimation for quick, fluid layout transitions without needing complex animations.*

**65. What are some common performance bottlenecks in React Native, and how do you mitigate them?**

*Common performance bottlenecks include:*

* *Excessive re-renders: Use React.memo, useCallback, and useMemo to memoize components and functions.*
* *Large lists: Optimize FlatList or SectionList with lazy loading, item recycling (e.g., react-window), and stable keys.*
* *Heavy animations: Use react-native-reanimated or LayoutAnimation for performant animations.*
* *Inefficient images: Optimize image sizes, use caching, and prefer FastImage on Android for smoother loading.\**

*Mitigating these issues helps improve responsiveness, reduce memory usage, and enhance the overall user experience.*

**66. How does react-native-fs work for file handling, and what are common use cases?**

*react-native-fs is a library that provides file system access in React Native, allowing you to perform operations like reading, writing, deleting, and managing files on the device's file system. Common use cases include:*

* *Downloading and storing files, such as images, documents, or videos.*
* *Creating and managing local databases or caches by saving data as files.*
* *Reading and writing configuration or user-generated files, such as text files or JSON data.*
* *Uploading files to remote servers.*

*To use react-native-fs, you would typically import it and call its methods, like readFile, writeFile, downloadFile, etc., passing the appropriate path and options.*

**67. Explain how to handle authentication and session management in React Native.**

*Authentication and session management in React Native can be handled using various approaches:*

* *Use libraries like react-navigation for managing navigation and state based on user authentication status.*
* *Store user credentials or tokens securely using AsyncStorage, SecureStore, or other secure storage libraries.*
* *Implement authentication flows with forms and APIs, using fetch or axios for network requests to log in, sign up, or log out users.*
* *Utilize context or state management libraries like Redux or MobX to manage global authentication state and provide user data throughout the app.*
* *Handle session expiration by checking token validity and redirecting users to the login screen when their session expires.*

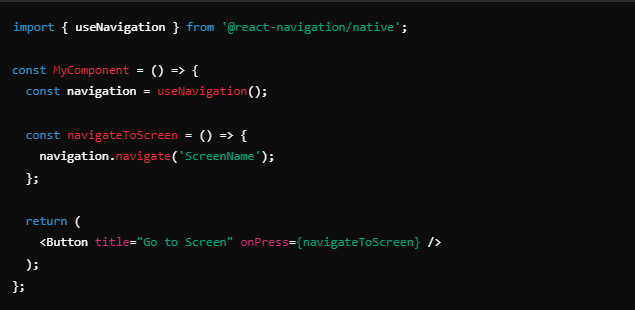
***68. How would you integrate a native Android/iOS library into a React Native project?***

*Integrating a native library into a React Native project involves the following steps:*

1. *Create or add the native library to your project in either the Android or iOS folders.*
2. *Expose the native functionality to JavaScript by creating a bridging module. This typically involves writing a Java (for Android) or Objective-C/Swift (for iOS) module that implements the native code and exposes methods to React Native.*
3. *Use NativeModules from react-native in your JavaScript code to call the exposed native methods.*
4. *Configure any necessary permissions or settings in the native project files, such as the AndroidManifest.xml for Android or the Info.plist for iOS.*
5. *Ensure proper linking of the library using either react-native link (for older versions) or by manually adding the library to the appropriate project files.*

***69. What is react-navigation’s useNavigation hook, and how do you use it?***

*The useNavigation hook is a part of react-navigation that allows functional components to access the navigation object without needing to pass it as a prop. This makes it easier to navigate programmatically within components. You can use it as follows:*

*This hook provides access to the navigation functions, such as navigate, goBack, setParams, etc., allowing you to control navigation directly from your component.*

**70. Explain differences between using Expo and a React Native CLI project.**

*The main differences between using Expo and a React Native CLI project include:*

* ***Development Environment:***
  + *Expo provides a managed environment with a set of pre-built components and APIs, making it easier to get started without configuring native build tools.*
  + *React Native CLI allows for full control over the native code and libraries, which is beneficial for complex applications or those needing custom native modules.*
* ***Native Code:***
  + *Expo limits the use of custom native modules; you can only use the APIs and libraries that Expo supports.*
  + *React Native CLI allows you to create and link custom native modules, giving you more flexibility.*
* ***Build Process:***
  + *Expo simplifies the build process with tools like Expo Go for testing on devices without a native build.*
  + *React Native CLI requires setup and configuration for building and deploying apps to Android and iOS.*
* ***Over-the-Air Updates:***
  + *Expo provides built-in over-the-air updates, making it easy to push updates to users without going through app stores.*
  + *React Native CLI requires additional tools (like CodePush) for implementing similar functionality.*

*Choosing between them depends on the project requirements, the need for native functionality, and developer experience.*