1. **What is React, and how does it differ from other JavaScript frameworks/libraries?**

React is an open-source JavaScript library used for building user interfaces (UIs). It was developed by Facebook and is known for its **component-based architecture**. Unlike other frameworks like Angular or Vue, React focuses solely on the view layer, leaving the choice of other technologies like routing or state management to the developer.

1. **What are the advantages of using React?**

* Use of Virtual DOM to improve efficiency
* Gentle learning curve
* SEO friendly
* Reusable components
* Huge ecosystem of libraries to choose from

1. **Explain the component lifecycle in React.**

In React, components have various lifecycle methods that are invoked at different stages of a component's existence. The key stages include:

* component creation (mounting)
* updates (updating)
* removal (unmounting).

1. **What are React Hooks, and how do they work?**

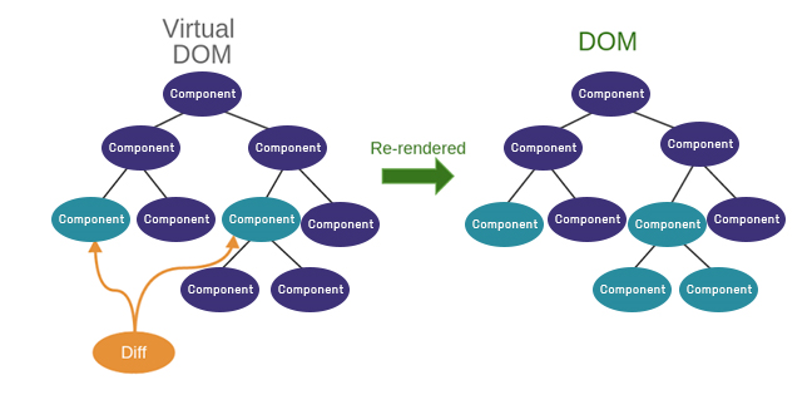
React Hooks are functions that allow functional components to manage state and side effects. Commonly used hooks include **useState** for managing component state and **useEffect** for handling side effects. Hooks provide a more concise and readable way to work with component logic.

1. **What is JSX, and why is it used in React?**

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript files. It's used in React to define the structure of UI components. JSX is transpiled into regular JavaScript by tools like Babel before it's rendered in the browser.

1. **What is the Virtual DOM, and why is it important in React?**

The Virtual DOM is a virtual representation of the actual DOM in memory. React uses it to optimize UI updates. When changes occur in the component's state, React creates a virtual tree, compares it to the previous one, and then updates the real DOM only with the differences (the changed component’s children are also re-renders). This process results in faster and more efficient UI updates.



1. **How do you manage state in React?**

State management in React can be achieved using the **useState** hook for functional components. For more complex state management, you can use libraries like **Redux** or even the **Context API** to share state between components.

1. **What is the difference between state and props in React?**

* **Props** (short for properties) are used to pass data from a parent component to a child component. They are immutable and provide a way for components to communicate with each other.
* **State**, on the other hand, is used to manage data that can change over time within a component itself. Components can have their own local state, and when state changes, React re-renders the component to reflect the new state.

1. **What is prop drilling, and how can you avoid it?**

Prop drilling occurs when you pass data from a parent component down to deeply nested child components through multiple intermediary components. To avoid it, you can use **React's Context API** to provide data to components without the need for props drilling, or use state management libraries like **Redux**.

1. **What are the rules that must be followed while using React Hooks?**

There are 2 rules which must be followed while you code with Hooks:

* React Hooks must be called only at the top level. It is not allowed to call them inside the nested functions, loops, or conditions.
* It is allowed to call the Hooks only from the React Function Components.

1. **Why is there a need for using keys in Lists?**

* Keys help react identify which elements were added, changed or removed.
* Keys should be given to array elements for providing a unique identity for each element.
* Without keys, React does not understand the order or uniqueness of each element.
* With keys, React has an idea of which particular element was deleted, edited, and added.
* Keys are generally used for displaying a list of data coming from an API.

1. **What is React Router, and how do you implement routing in a React application?**

React Router is a library for adding client-side routing to React applications. You can set up routes using **BrowserRouter**, define routes with **Route** components, and navigate between them using **Link** components.

* **BrowserRouter**: It is a router implementation that will make use of the HTML5 history API (pushState, popstate, and event replaceState) for keeping your UI to be in sync with the URL. It is the parent component useful in storing all other components.
* **Routes**: It is a newer component that has been introduced in the React v6 and an upgrade of the component.
* **Route**: It is considered to be a conditionally shown component and some UI will be rendered by this whenever there is a match between its path and the current URL.
* **Link**: It is useful in creating links to various routes and implementing navigation all over the application. It works similarly to the anchor tag in HTML.

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

Performance optimization in React involves techniques like:

* code splitting (loading only necessary code for each route)
* lazy loading (loading components as needed)
* memoization (caching expensive calculations)
* implementing the **shouldComponentUpdate** method to prevent unnecessary renders.

1. **If you were working on a React application that was rendering a page very slowly, how would you go about investigating and fixing the issue?**

The first step is to use the **Profiler** tool provided within the **React Developer Tools** browser plugin, which is available for Google Chrome and Mozilla Firefox. The Profiler tool allows developers to **find components that take a long time to render** or are rendering **more frequently than necessary**.

One of the most common issues in React applications is when components re-render unnecessarily. **React.memo()** prevents unnecessary re-rendering of function components - if the props have not changed, then the component will not re-render. While this is a useful tool, the shallow comparison brings with it an additional performance penalty, so it can have a negative performance impact if used incorrectly. By using the React Profiler, performance can be measured before and after using this tool to ensure that performance is actually improved by making a given change.

1. **What is mutation in React?**

**State mutation** in React refers to the act of **directly modifying the state of a component by changing its values or properties without following the recommended practices and conventions** for state management. In React, state mutation is considered a bad practice and can lead to unexpected behavior and bugs in your application. React encourages developers to follow a set of rules and guidelines for managing state to ensure the predictable and reliable behavior of components.

1. **Loss of React's Reconciliation**: React uses a process called reconciliation to efficiently update the user interface when state or props change. When you mutate state directly, React may not detect the change, leading to inconsistencies between the component's rendered output and its internal state.
2. **Unpredictable Behavior**: Mutating state directly can lead to unexpected and hard-to-debug behavior. Since React relies on a virtual representation of the DOM and a virtual representation of state, directly mutating state bypasses React's mechanisms for managing these aspects.
3. **Performance Concerns**: React's virtual DOM and reconciliation process are designed to optimize updates for performance. State mutations can interfere with these optimizations, potentially causing your application to be slower than it could be.
4. **How can mutation be avoided?**

Use **setState** or **State Hooks**: Always use the useState hook in functional components to update state. This method ensure that React is aware of the state change and can trigger the necessary updates to the component's UI.

const [count, setCount] = useState(0);

setCount(count + 1);

When updating state, create new objects or arrays with the updated values rather than modifying the existing state. This ensures that you're not directly mutating the state object, which can lead to problems.

setTodos((prevTodos) => [...prevTodos, newTodo]);

1. **What is Redux? Explain its basic concepts.**

Redux Toolkit is a library that simplifies and streamlines the process of managing application state with Redux. It provides utilities and best practices to help manage the complexity of state management, making it an ideal choice for React applications.

* **Actions** are plain JavaScript objects that describe changes to the application state.
* **Reducers** are pure functions that specify how the application's state changes in response to actions.
* The **store** is a single source of truth that holds the entire application state.