## **What are the major features of React?**

The major features of React are:

* It uses **VirtualDOM** instead of RealDOM considering that RealDOM manipulations are expensive.
* Supports **server-side rendering**.
* Follows **Unidirectional** data flow or data binding.
* Uses **reusable/composable** UI components to develop the view.

## **What are Pure Components?**

*React.PureComponent* is exactly the same as *React.Component* except that it handles the shouldComponentUpdate() method for you. When props or state changes, *PureComponent* will do a shallow comparison on both props and state. *Component* on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.

## **What are synthetic events in React?**

SyntheticEvent is a cross-browser wrapper around the browser's native event. It's API is same as the browser's native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers.

## **What is Virtual DOM?**

The *Virtual DOM* (VDOM) is an in-memory representation of *Real DOM*. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called *reconciliation*.

## **What are controlled components?**

A component that controls the input elements within the forms on subsequent user input is called **Controlled Component**, i.e, every state mutation will have an associated handler function.

For example, to write all the names in uppercase letters, we use handleChange as below,

handleChange(event) {

this.setState({value: event.target.value.toUpperCase()})

}

## **What is JSX?**

*JSX* is a XML-like syntax extension to ECMAScript (the acronym stands for *JavaScript XML*). Basically it just provides syntactic sugar for the React.createElement() function, giving us expressiveness of JavaScript along with HTML like template syntax.

In the example below text inside <h1> tag is returned as JavaScript function to the render function.

class App extends React.Component {

render() {

return(

<div>

<h1>{'Welcome to React world!'}</h1>

</div>

)

}

}

## **What is the difference between Element and Component?**

An *Element* is a plain object describing what you want to appear on the screen in terms of the DOM nodes or other components. *Elements* can contain other *Elements* in their props. Creating a React element is cheap. Once an element is created, it is never mutated. Whereas a **component** can be declared in several different ways. It can be a class with a render() method. Alternatively, in simple cases, it can be defined as a function.

## **When to use a Class Component over a Function Component?**

If the component needs *state or lifecycle methods* then use class component otherwise use function component. *However, from React 16.8 with the addition of Hooks, you could use state , lifecycle methods and other features that were only available in class component right in your function component.*

## **What is the difference between state and**

## **props?**Both *props* and *state* are plain JavaScript objects. While both of them hold information that influences the output of render, they are different in their functionality with respect to component. Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

## **What are inline conditional expressions?**

You can use either *if statements* or *ternary expressions* which are available from JS to conditionally render expressions. Apart from these approaches, you can also embed any expressions in JSX by wrapping them in curly braces and then followed by JS logical operator &&.

<h1>Hello!</h1>

{

messages.length > 0 && !isLogin?

<h2>

You have {messages.length} unread messages.

</h2>

:

<h2>

You don't have unread messages.

</h2>

}

## **What is the use of refs?**

The *ref* is used to return a reference to the element. They *should be avoided* in most cases, however, they can be useful when you need a direct access to the DOM element or an instance of a component.

## **What is the difference between Shadow DOM and Virtual DOM?**

The *Shadow DOM* is a browser technology designed primarily for scoping variables and CSS in *web components*. The *Virtual DOM* is a concept implemented by libraries in JavaScript on top of browser APIs.

## **What is React Fiber?**

Fiber is the new *reconciliation* engine or reimplementation of core algorithm in React v16. The goal of React Fiber is to increase its suitability for areas like animation, layout, gestures, ability to pause, abort, or reuse work and assign priority to different types of updates; and new concurrency primitives.

## **What is the difference between createElement and cloneElement?**

JSX elements will be transpiled to React.createElement() functions to create React elements which are going to be used for the object representation of UI. Whereas cloneElement is used to clone an element and pass it new props.

## **What is Lifting State Up in React?**

When several components need to share the same changing data then it is recommended to *lift the shared state up* to their closest common ancestor. That means if two child components share the same data from its parent, then move the state to parent instead of maintaining local state in both of the child components.

## **What are the lifecycle methods of React?**

1. Before React 16.3
   * **componentWillMount:** Executed before rendering and is used for App level configuration in your root component.
   * **componentDidMount:** Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.
   * **componentWillReceiveProps:** Executed when particular prop updates to trigger state transitions.
   * **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
   * **componentWillUpdate:** Executed before re-rendering the component when there are props & state changes confirmed by shouldComponentUpdate() which returns true.
   * **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes.
   * **componentWillUnmount:** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.
2. React 16.3+
   * **getDerivedStateFromProps:** Invoked right before calling render() and is invoked on *every* render. This exists for rare use cases where you need derived state. Worth reading [if you need derived state](https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html).
   * **componentDidMount:** Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.
   * **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
   * **getSnapshotBeforeUpdate:** Executed right before rendered output is committed to the DOM. Any value returned by this will be passed into componentDidUpdate(). This is useful to capture information from the DOM i.e. scroll position.
   * **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes. This will not fire if shouldComponentUpdate() returns false.
   * **componentWillUnmount** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

## **What are Higher-Order Components?**

A *higher-order component* (*HOC*) is a function that takes a component and returns a new component. Basically, it's a pattern that is derived from React's compositional nature.

We call them **pure components** because they can accept any dynamically provided child component but they won't modify or copy any behavior from their input components.

const EnhancedComponent = higherOrderComponent(WrappedComponent)

HOC can be used for many use cases:

1. Code reuse, logic and bootstrap abstraction.
2. Render hijacking.
3. State abstraction and manipulation.
4. Props manipulation.

## **What is context?**

*Context* provides a way to pass data through the component tree without having to pass props down manually at every level.

For example, authenticated user, locale preference, UI theme need to be accessed in the application by many components.

const {Provider, Consumer} = React.createContext(defaultValue)

## **What is reconciliation?**

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called *reconciliation*.

## **What are stateless components?**

If the behaviour is independent of its state then it can be a stateless component. You can use either a function or a class for creating stateless components. But unless you need to use a lifecycle hook in your components, you should go for function components. There are a lot of benefits if you decide to use function components here; they are easy to write, understand, and test, a little faster, and you can avoid the this keyword altogether.

## **What is the use of react-dom package?**

The react-dom package provides *DOM-specific methods* that can be used at the top level of your app. Most of the components are not required to use this module. Some of the methods of this package are:

1. render()
2. hydrate()
3. unmountComponentAtNode()
4. findDOMNode()
5. createPortal()

## **How you implement Server Side Rendering or SSR?**

React is already equipped to handle rendering on Node servers. A special version of the DOM renderer is available, which follows the same pattern as on the client side.  
import ReactDOMServer from 'react-dom/server'

import App from './App'

1. ReactDOMServer.renderToString(<App />)  
   This method will output the regular HTML as a string, which can be then placed inside a page body as part of the server response. On the client side, React detects the pre-rendered content and seamlessly picks up where it left off.