**How do you decide when to split a component into subcomponents?**

The decision of splitting a component into subcomponents can be based on the following:

1. Since each component is designated to perform a specific task, we need to split a component into subcomponents when we have a component that has to do multiple tasks.
2. There might be situations when we may need to reuse some parts of a component. Thus, to re-use a component, we have to split it to subcomponents

**What is the difference between state and props?**

1. States are private attribute of components. Whereas props are read only attributes passed from parent to child
2. States are responsible for storing data and rendering component. Props are responsible for transferring data and functions from parent to child.
3. States are mutable. Which means values inside state can be updated. Props are immutable. They can’t be updated by child components

**How to trigger rerender in a component?**

Re-render can be triggered by changing the state of a component or when the props of a child component is changed in the parent component then the child component get re-rendered.

**Why do you like react over other front-end libraries and frameworks?**

1. Because React is Component based, it makes it easier to maintain, reuse a piece of code.
2. React’s virtual DOM optimizes the performance by re-rendering only the parts of the DOM which are changed
3. React has many libraries and tools for state management and for creating single page applications.
4. React has large community which is helpful in case of support needed.
5. React’s declarative approach makes the code more readable and makes it easier for debugging.

**What’s the difference between controlled components and uncontrolled components?**

1. Controlled Components are controlled by the component’s state, whereas Uncontrolled Components are controlled by the DOM
2. Internal state is not maintained in Controlled Components. Internal state is maintained in Uncontrolled Components.
3. Controlled Components are maintained by parent component. Uncontrolled Components are maintained by the DOM.
4. Controlled Components Have better control on the form data and values. Uncontrolled Components have limited control over form data and values.

**How to prevent components from unnecessary rerendering?**

We can prevent unnecessary rendering of components using the following:

1. React.memo: It prevents the components from re-rendering unless it’s props has changed.
2. useCallback: It memorizes functions so that they don’t get created again and again on every render
3. useMemo: It memorizes values so that they don’t get created again and again on every render

**Why are props needed to be immutable?**

Props are needed to be immutable in order to maintain consistent and predictable flow of data. If child components were able to mutate it’s prop, it could lead to unexpected behavior and make it difficult to track the source of changes. By keeping props immutable, React ensures that the prop in ChildComponent always reflects the state managed by ParentComponent.

**Explain the Virtual DOM and how React uses it to improve performance.**

Virtual DOM is basically a lightweight copy of the actual DOM used by React to improve performance. When a component’s state or props change React updates the virtual DOM first. Then it compares the updated virtual DOM with the previous copy of the virtual DOM to identify the differences. It uses an algorithm called “diffing” to perform the comparison. It then updates only the parts of the original DOM that have been changed. This minimizes direct manipulation of the real DOM and reduces costly operations.

**Can you explain the useMemo and useCallback hooks and provide examples of when you might use them?**

**useMemo:** useMemo is a React hook that is used for optimization purposes. It does the optimization by memoizing values. It recomputes the momoized value only if the dependency has been changed and returns the recomputed value. This can optimize the performance as we don’t have to do expensive computations on every render.

Consider the following code. Here we DON’T use useMemo and the slowFunction() is executed in every render. Whether we change the value input the input field or we toggle the background. However, what we want is that we want to execute slowFunction() only when the value we change the value inside input field.

import React from 'react'

import {useState, useMemo, useCallback} from 'react'

export default function MemoizationDemo(){

    const [number, setNumber] = useState(0)

    const [dark, setDark] = useState(false)

    const doubleNumber = slowFunction(number)

    const theme = {

        backgroundColor: dark ? 'black' : 'white',

        color: dark ? 'white' : 'black'

    }

    const toggleDark = ()=>{

        setDark(!dark)

    }

    const getInputNumber = (event)=>{

        setNumber(parseInt(event.target.value))

    }

    return(

        <div>

            <input type="number" onChange = {getInputNumber}/>

            <button onClick={toggleDark}>Toggle Background</button>

            <div style={theme}>

                <p>{doubleNumber}</p>

            </div>

        </div>

    )

}

// This function is a expensive function as it runs a for loop for long time

// This function called in every re-render. Whether we change the value inside

// input field or toggle the theme. However, we only want to execute it only when // the value inside the input field changes

const slowFunction = (num)=>{

    for (let i = 0; i < 1000000000; i++) {}

    console.log("INSIDE SLOW FUNCTION")

    return num \* 2

}

Now, consider the following code. Here we are using useMemo and the slowFunction() is executed only when “number” is changed as it is the dependency of useMemo.

import React from 'react'

import {useState, useMemo, useCallback} from 'react'

export default function MemoizationDemo(){

    const [number, setNumber] = useState(0)

    const [dark, setDark] = useState(false)

    const doubleNumber = useMemo(() => {

        return slowFunction(number)

    }, [number]) //only when "number" changes useMemo is executed

    const theme = {

        backgroundColor: dark ? 'black' : 'white',

        color: dark ? 'white' : 'black'

    }

    const toggleDark = ()=>{

        setDark(!dark)

    }

    const getInputNumber = (event)=>{

        setNumber(parseInt(event.target.value))

    }

    return(

        <div>

            <input type="number" onChange = {getInputNumber}/>

            <button onClick={toggleDark}>Toggle Background</button>

            <div style={theme}>

                <p>{doubleNumber}</p>

            </div>

        </div>

    )

}

// This function is a expensive function as it runs a for loop for long time

// This function called in every re-render. Whether we change the value inside input field

// or we toggle the theme. However, we only want to execute it only when the value inside the input field changes

const slowFunction = (num)=>{

    for (let i = 0; i < 1000000000; i++) {}

    console.log("INSIDE SLOW FUNCTION")

    return num \* 2

}

**useCallback:** useCallback is similar to useMemo except it returns a function instead of value.

**Explain the concept of Higher-Order Components (HOCs) and provide an example use case.**

A Higher-Order Component is a pattern in React that allows us to reuse component logic. These are basically functions that take a component as an argument and return a new component with enhanced functionality. They do not modify the original component, but instead, create a wrapper component that adds additional behavior or data.

Example use: Suppose we have multiple components in our application that need to fetch data from an API and display it. Instead of duplicating the data fetching logic in each

**Discuss the differences between React's class components and functional components. Which one do you prefer and why?**

1. Functional Components are plain JavaScript functions whereas Class Components are classes that needs to be extended from React
2. Functional Components don’t require render() function to render the elements. Class Components uses render() function to render the elements.
3. Functional components run from top to bottom and once the function is returned it can’t be kept alive. The class component is instantiated, and different life cycle method is kept alive and is run and invoked depending on the phase of the class component.
4. Functional Components are stateless as they simply accept data and display them in some form. Class Components are stateful as they implement logic and state
5. Functional Components don’t use constructors. Class Components uses constructors.

I would prefer functional components because they are simple and concise as well as they introduce hooks which allows functional components manage states effectively.

**How do you ensure your code is maintainable and scalable?**

I make sure that my code has multiple components where each component has specific responsibility, I make use of functional components instead of class components, keep the css file separate, use clear and descriptive names for components, variables and functions, perform unit testing and integration testing to ensure reliability, keep documentation for readability and maintainability.