**Q.1. What are controlled / uncontrolled components? Why do we use controlled**

**components?**

In a controlled component, the form data is handled by the state within the component. The state within the component serves as “the single source of truth” for the input elements that are rendered by the component.

Uncontrolled components act more like traditional HTML form elements. The data for each input element is stored in the DOM, not in the component. Instead of writing an event handler for all of your state updates, you use a ref to retrieve values from the DOM.

In most cases, we recommend using controlled components to implement forms. In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.

**Q.2. What are side effects? Write a note about them.**

Side effects are basically anything that affects something outside of the scope of the current function that's being executed. In our dashboard, this includes: API requests to our backend service. Calls to our authentication service.

Some examples of side effects in React components are:

• Making asynchronous API calls for data.

• Setting a subscription to an observable.

• Manually updating the DOM element.

• Updating global variables from inside a function.

**Q.3. What is the uni-directional data flow in React?**

Unidirectional data flow is a technique that is mainly found in functional reactive programming. It is also

known as one-way data flow, which means the data has one, and only one way to be transferred to other

parts of the application. In essence, this means child components are not able to update the data that is

coming from the parent component. In React, data coming from a parent is called props.

The major benefit of this approach is that data flows throughout your app in a single direction, giving you

better control over it.

In terms of React it means:

• state is passed to the view and to child components

• actions are triggered by the view

• actions can update the state

• the state change is passed to the view and to child components

**Q.4. List down and explain the 3 ways in which useEffect() hook can be used, with relevant**

**examples.**

useEffect hook is an extremely powerful a versatile tool, allowing you to even create your own, custom

hooks.

Hooks are available for functional components. Let’s introduce a simple useEffect to our functional

component:

import React, { useEffect } from "react";

function SimpleUseEffect() {

useEffect(() => {

alert("Component Rendered")

});

return (

<div>

<b>A Simple use of useEffect...</b>

</div>

)

}

In the code above, we used the useEffect hook. It takes a function as input, which is executed after the

initial rendering, as well as re-rendering, of the component. After each rendering, one the DOM has been

updated and the function passed to useEffect is invoked. In the above scenario, the component gives an

alert after the initial rendering of the component