**What is useEffect? What are the different behaviors of useEffect? What is a dependency array?**

useEffect is a React Hook that allows us to perform side effects in our function components. Side effects could include data fetching, subscriptions, or manually changing the DOM.

The behavior of of useEffect completely depends on the dependency array provided to it’s second argument. The behavior depends on the following states of the dependency array:

1. If we omit the dependency array, the effect will run after every render and re-render.
2. If we provide empty dependency array, useEffect will run only once during initial rendering
3. If we provide an array with dependencies, the effect will run only when one of the dependencies has changed since the last render.

The dependency array in useEffect is an array of values that the effect depends on. The effect will only re-run if one of these values changes between renders. This allows for precise control over when side effects are executed and when the cleanup function is called.

**What is useRef and when do you want to use it?**

useRef is a React hook that allows us to persist values without causing re-rendering when the value changes. It contains a mutable “ref” object with a “current” property that can be used to store any value.

We use useRef in follow scenarios:

1. Accessing DOM Elements:

It can be used for interacting with DOM elements. Similar to how document.getElementById or document.querySelector is used in vanilla JavaScript.

1. Persisting Values Without Re-rendering:

useRef can be used to store mutable values that don't trigger a re-render when they change. It is useful for keeping track of a value or state for later use without changing the UI

How to reuse hook logic in React?

The reuse hook logic can be implemented using custom hooks. It allows us to create reusable functions that can be used by different components. Custom hooks can use React’s built-in hooks.

Example:

Let's say we have multiple components that need to fetch data from an API. Instead of duplicating the fetching logic in each component, we can create a custom hook to handle it.

import { useState, useEffect } from 'react';

function useFetch(url) {

const [data, setData] = useState(null);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

useEffect(() => {

const fetchData = async () => {

try {

const response = await fetch(url);

if (!response.ok) {

throw new Error('Network response was not ok');

}

const result = await response.json();

setData(result);

setLoading(false);

} catch (error) {

setError(error);

setLoading(false);

}

};

fetchData();

}, [url]);

return { data, loading, error };

}

export default useFetch;