useEffect Hook

## **What is useEffect?**

This Hook is used to perform side effects in your components. The useEffect hook contains a callback function that comes into effect when the page loads. It allows us to execute some functions when a component gets updated.

## **useEffect Dependency array**

The useEffect manages an array that contains the state variables or functions which are kept an eye for any changes. These changes then trigger the callback function.

### ****Empty array****

The most basic dependency array would be an empty array. The empty array indicates that the useEffect doesn’t have any dependencies on any state variables. Therefore, the callback function is only called once the page renders in this case. For example: Fetch data from an API endpoint when the page renders.

useEffect(() => {

fetch("https://pokeapi.co/api/v2/type/3")

}, []);

The fetch() is used to GET data from the API endpoint. In this example, the fetch() needs to be executed only once therefore we are using an empty dependency array.

### ****Array containing dependencies****

Single State variable: If the array contains a state variable, the useEffect callback function gets triggered on 2 occasions. First, when the page renders and whenever the state variable is updated.

Example:

useEffect(() => {

console.log("Counter value: ", counter);

}, [counter]);

In this example, the counter state is included in the dependency array, therefore whenever the counter state is updated i.e setCounter(counter + 1), the useEffect callback function is triggered and the value of the counter is displayed.

Multiple State variable: In case of multiple dependencies, the callback function is triggered when the page is first rendered and whenever any of the included states gets updated.

useEffect(() => {

console.log(num1 + num2);

}, [num1, num2]);

Here, we have 2 state variables num1 and num2 in the array dependency. We want to perform an addition operation of these two state variables. Therefore, whenever either of the 2 state variables gets updated, we perform an addition operation.

Infinite re-rendering problem: The useEffect hook is quite useful but if not used properly it can crash your entire application. This might occur due to infinite rerendering. Let’s take a look at the example below to understand this better.

useEffect(() => {

setCounter(counter + 1);

console.log('Counter value: ', counter);

}, [counter]);

Here, inside the callback function, we’re updating the counter and then printing the value of the counter state. When setCounter is called, the useEffect with counter state dependency is triggered and again counter value is updated. This process goes on and on and the page becomes unresponsive.

### ****Functions in a dependency array****

If we call functions inside the useEffect callback function, we would have to include the function name in the dependency array. This will result in a bug as printName() isn’t wrapped in the callback. That will lead to useEffect triggering on every render of React component! Let’s take a look.

import { useEffect } from 'react';

const ExampleComponent = () => {

const name = "Thomas"

const printName = (value) => {

console.log(value);

};

useEffect(() => {

printName(name);

}, [name, printName]);

return <p>UseEffect!</p>;

};

export default ExampleComponent;

We can prevent this by placing the function outside the component function.

import { useEffect } from 'react';

const name = "Thomas"

const printName = (value) => {

console.log(value);};

const ExampleComponent = () => {

useEffect(() => {

printName(name);

}, [name, printName]);

return <p>UseEffect!</p>;};

export default ExampleComponent;

Fetch API

There are many ways to fetch data from an external API in React.

## How to Fetch Data in React Using the Fetch API

* The most accessible way to fetch data with React is using the Fetch API. The Fetch API is a promise-based interface for fetching resources by making HTTP requests to servers from web browsers. It is similar to XML HTTP requests but better and more powerful.
* To do so, we make our request within the useEffect hook, and we make sure to provide an empty dependencies array as the second argument, so that our request is only made once (assuming it's not dependent on any other data in our component).
* The Fetch API provides an interface for fetching resources (including across the network).
* The fetch() method takes one mandatory argument, the path to the resource you want to fetch. It returns a [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) that resolves to the [Response](https://developer.mozilla.org/en-US/docs/Web/API/Response) to that request
* The Fetch API provides the fetch() method defined on a window object. This is used to perform requests. This method returns a Promise which can be further used to retrieve response of the request.
* fetch () method that allows you to fetch data from all sorts of different places and work with the data fetched. It allows you to make an HTTP request, i.e., either a GET request (for getting data) or POST request (for posting data).
* Calling fetch () starts a request and returns a promise. When the request completes, the promise resolves to the response object. From the response object you can extract data in the format you need: JSON, raw text, Blob.
* The basic fetch request can be explained by the following code:

fetch('url')

.then(response => {

//handle response

console.log(response);

})

.then(data => {

//handle data

console.log(data);

})

.catch(error => {

//handle error

});