I. useEffect

By using React’s useEffect Hook, we can tell what the component needs to do after render.

useEffect is a function and it has 2 arguments:

* The first argument is a function where the effect occurs
* The second arguments is a dependency array of things that will trigger our effect.

**Note**

By default, useEffect runs after every re-render.

* If we leave out the second argument, the side-effect runs after each rendering
* If we leave it as an empty array [], the side-effect runs once after the component renders for the first time.
* If the array has any values [value1, value2]: the side-effect runs only when any value in the dependencies change.

useEffect(() => {

function()

}, [])

**Example**: We change the color each time the count value changes

import React, { useState, useEffect } from 'react'

import randomColor from 'randomcolor'

export default function Playground() {

const [count, setCount] = useState(0)

const [color, setColor] = useState(null)

useEffect(() => {

setColor(randomColor())

}, [count])

return (

<div style={{ borderTop: `10px solid ${color}`}}>

{count}

<button onClick={() => setCount(currentCount => currentCount - 1)}>-</button>

<button onClick={() => setCount(currentCount => currentCount + 1)}>+</button>

</div>

)

}

II. Fetch data with useEffect

The most common usage of useEffect is to fetch data.

For example, we would like to fetch API from Github with the following url

const url = 'https:*//api.github.com/users';*

**Step 1**: Set up the app

import React, { useState, useEffect } from 'react';

const url = 'https://api.github.com/users';

const App = () => {

const [users, setUsers] = useState([]);

return (

<div>

<h3>Github Users</h3>

<ul className='users'>

{users.map((user) => {

const { id, login, avatar\_url, html\_url } = user;

return (

<li key={id}>

<img src={avatar\_url} alt={login} />

<div>

<h4>{login}</h4>

<a href={html\_url}>profile</a>

</div>

</li>

);

})}

</ul>

</div>

);

};

export default App;

**Step 2**: Use Fetch with Async/await

We create a function named getUsers

const getUsers = async () => {

const response = await fetch(url);

const users = await response.json();

setUsers(users);

};

**Step 3**: Create useEffect function that only runs once

useEffect(() => {

getUsers();

}, []);

**Putting it all together**

import React, { useState, useEffect } from 'react';

const url = 'https://api.github.com/users';

const App = () => {

const [users, setUsers] = useState([]);

const getUsers = async () => {

const response = await fetch(url);

const users = await response.json();

setUsers(users);

};

useEffect(() => {

getUsers();

}, []);

return (

<div>

<h3>Github Users</h3>

<ul className='users'>

{users.map((user) => {

const { id, login, avatar\_url, html\_url } = user;

return (

<li key={id}>

<img src={avatar\_url} alt={login} />

<div>

<h4>{login}</h4>

<a href={html\_url}>profile</a>

</div>

</li>

);

})}

</ul>

</div>

);

};

export default App;

III. Clean up function in useEffect Hook

Clean up function is very useful because it can remove unnecessary behavior or prevent memory leaking issues.

useEffect() invokes the clean up function we return from the callback function

useEffect(() => {

*// Function*

return function cleanup() {

*// Side-effect cleanup*

};

}, dependencies);

Example

We log a message to console every 2 seconds:

import React, { useEffect, useState } from "react";

function Message({ message }) {

useEffect(() => {

setInterval(() => {

console.log(message);

}, 2000);

}, [message]);

return <div className="message">Logging to console "{message}"</div>;

}

const App = () => {

const [message, setMessage] = useState("Hello World!");

return (

<div className="App">

<h3>Type your message</h3>

<input

type="text"

value={message}

onChange={(e) => setMessage(e.target.value)}

/>

<Message message={message} />

</div>

);

}

export default App;

In the example, the console logs every 2 seconds each message typed.

To stop the logging of previous messages, we cleanup the side-effect by canceling the previous log when starting a new one.

We add the return function so only the latest message is logged in the console.

function RepeatMessage({ message }) {

useEffect(() => {

const id = setInterval(() => {

console.log(message);

}, 2000);

return () => {

clearInterval(id);

};

}, [message]);

return <div className="message">Logging to console "{message}"</div>;

}