Front End III

useEffect en acción

Imaginemos que queremos crear un componente funcional que represente el saludo de un determinado usuario. Para ello, queremos devolver un mensaje que contenga el nombre recibido a través de props y un efecto secundario que cambie el título de nuestra página y muestre el nombre del usuario en la parte superior de la página. Y queremos hacer esto cada vez que esta prop sufra algún cambio en su contenido y/o estado.

Ejemplo con el uso incorrecto de los efectos secundarios:

```
function Saludo({ nombre }) {
  const mensaje = `Hola, ${nombre}!`; // Armado del texto

  // Error!!👇 - No debemos generar efectos acá dentro
  document.title = `Saludo: ${nombre}`; // Efecto colateral

  return <div>{mensaje}</div>; // Valor de retorno
}
```

La representación del componente y la lógica del efecto secundario deben ser independientes. Sería un error realizar efectos secundarios directamente en el cuerpo del componente que se utiliza principalmente para calcular la salida del componente.

Ejemplo con el uso correcto de los efectos secundarios:

```
//👇 importamos el hook para su uso
import { useEffect } from 'react';

function Saludo({ nombre }) {
  const mensaje = `Hola, ${nombre}!`; //armamos el mensaje

  //👇 utilizamos el hook de useEffect
  useEffect(() => {

  //forma recomendada de realizar un efecto
    document.title = `Saludo: ${nombre}`;// Efeito colateral

  }, [nombre]);//De esta forma, se volverá a renderizar solo si hay un cambio en el prop de “nombre”

  return <div>{mensaje}</div>;// Estructura del componente
}
```

En el ejemplo anterior, la actualización del título del documento es el efecto secundario porque no se calcula directamente y no depende de la salida del componente. Por lo tanto, no queremos que el título de la página se actualice cada vez que el componente **Saludo** se renderice. Sin embargo, queremos cambiar el título de la página solo cuando el nombre recibido a través de prop cambia, por lo que proporcionamos el nombre como una dependencia a la matriz. Ejemplo: useEffect(callback, [nombre]).

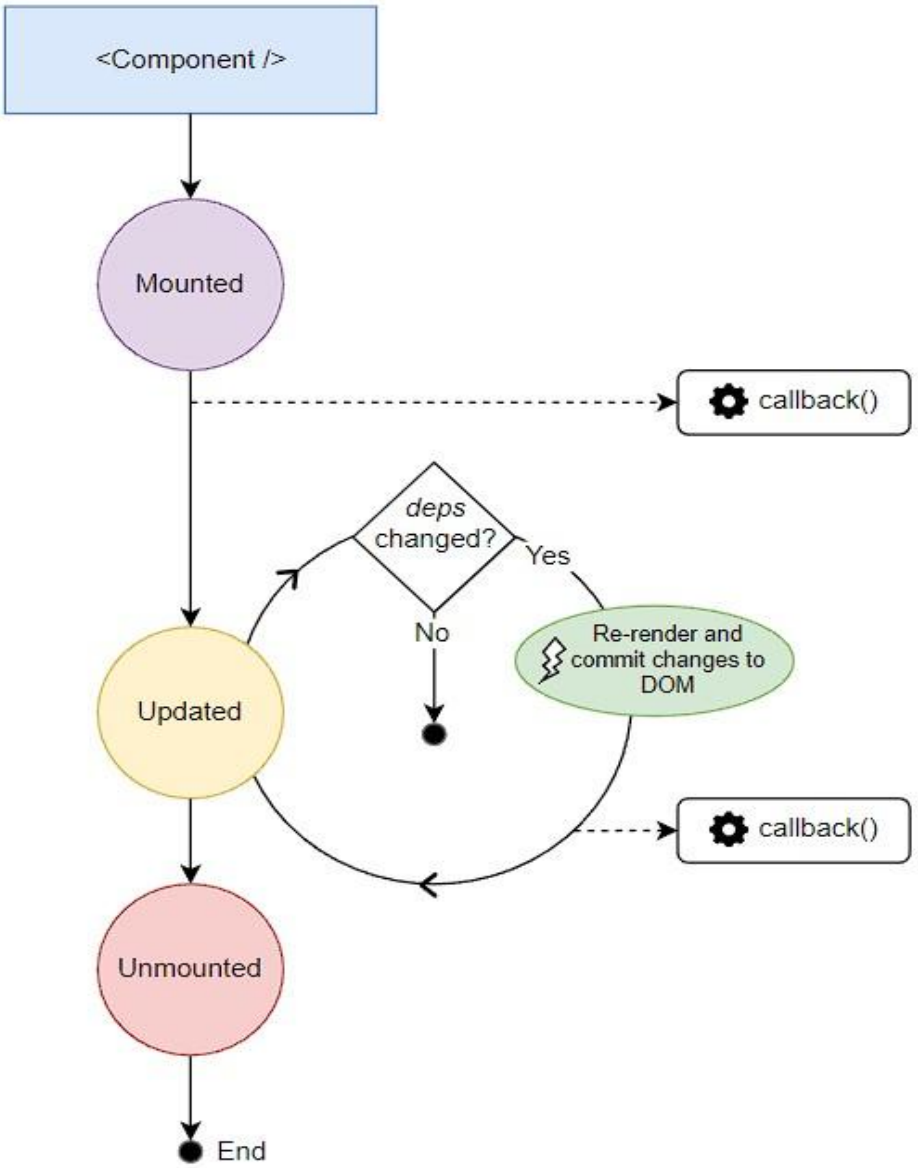
Ahora que entendemos un poco el concepto useEffect y vemos cómo debe ser utilizado correctamente en un componente, vamos a entender un poco mejor cómo utilizarlo.

El hook useEffect acepta 2 argumentos:

```
useEffect(callback, [array of dependencies]);
```

Por lo tanto, debemos poner la lógica de efectos secundarios en la función de devolución de llamada (primer parámetro) y utilizar la matriz de dependencia (segundo parámetro) para controlar cuándo queremos que se ejecute ese efecto secundario en particular.

La siguiente imagen muestra el flujo de ejecución del componente que utiliza el hook useEffect.



Formas de controlar el efecto secundario (a través de la matriz de dependencia)

- **Matriz no suministrada:**

El efecto secundario se produce después de cada renderización del componente.

```
function MyHookUseEffect() {

  useEffect(() => {
    //👇 Siempre se ejecuta después de cada renderizado
    // -- algún efecto secundario --
  }); //No se envia necesariamente el array de dependencias

}
```

- **Array vacío []:**

El efecto secundario se ejecuta solo una vez después de la renderización inicial del componente.

```
function MyHookUseEffect() {

  useEffect(() => {
    //👇 Siempre se ejecuta despues de cada renderizado
    // -- algún efecto secundario --
  }, []); //Acá si el array fue declarado, aunque vacío

}
```

- **Array controlado por props y/o states [props, states]:**

El efecto secundario se ejecuta una vez después de la renderización inicial, y se vuelve a renderizar cada vez que cambia alguno de los valores de la matriz de dependencia.

```
function MyHookUseEffect({ prop }) {
  const [state, setState] = useState("");

  useEffect(() => {
    //👇 Siempre se ejecuta después de cada renderizado
    // -- algún efecto secundario --

   }, [prop, state]); //Se vuelve a renderizar cada vez que cambia alguno de los
valores del array
}
```

Borrar los efectos secundarios con el hook useEffect

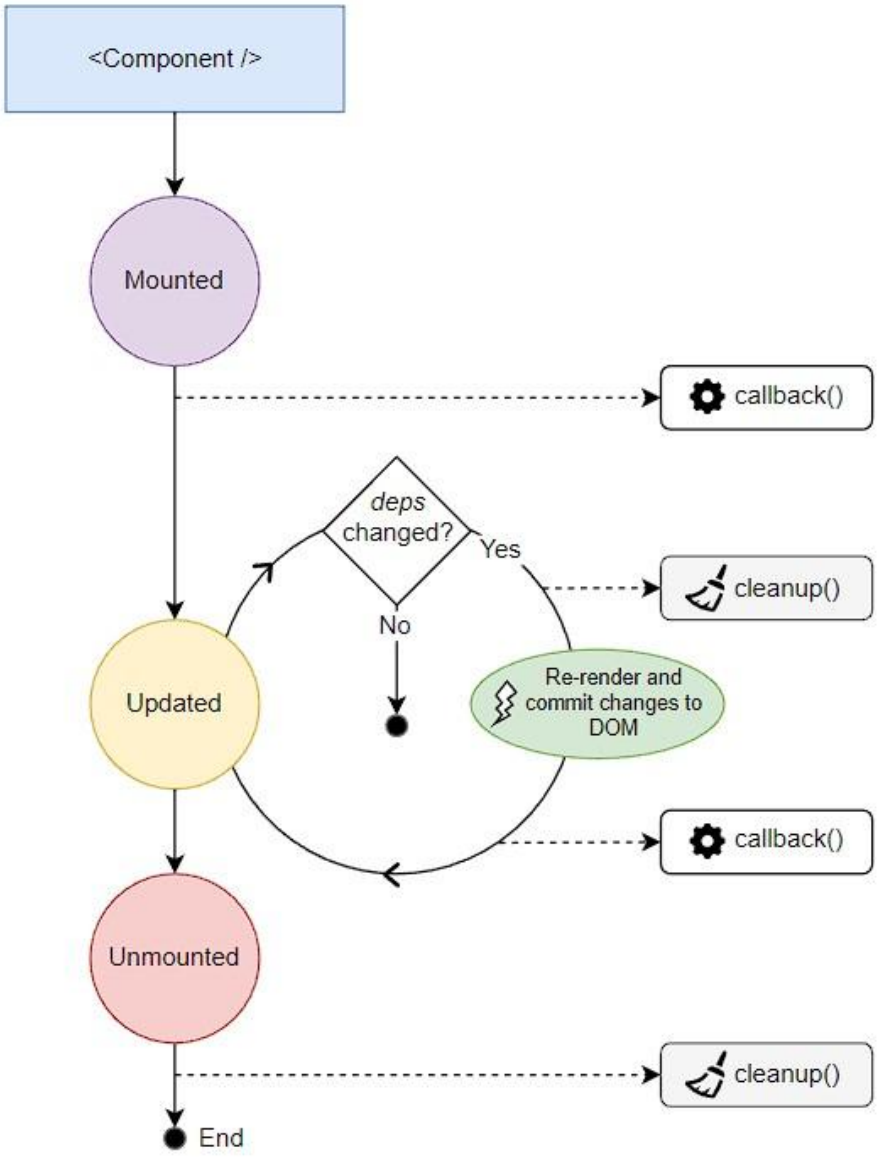
Los componentes que utilizan efectos secundarios pueden necesitar alguna limpieza/apagado. Ejemplos comunes son el cierre de conexiones abiertas y la finalización de temporizadores activos.

```
function MyHookUseEffectCleanUp() {
  useEffect(() => {
    //👇 Siempre se ejecuta después de cada renderizado
    // -- algún efecto secundario --

     //Se retorna una función que se encarga de la limpieza en el desmontaje del componente
    return function limpiar() {
      // Realizamos alguna acción de limpieza del efecto
    };
  }, [dependencias]);
}
```

Para el useEffect, si el callback principal (primer parámetro) devuelve alguna función, la considera como responsable de limpiar el efecto secundario.

La siguiente imagen muestra el flujo de ejecución del componente que utiliza el Hook useEffect y que además realiza el proceso de limpieza de efectos secundarios.


<Component />
Mounted
callback()
deps changed?
Yes
No
cleanup()
Updated
Re-render and commit changes to DOM
callback()
Unmounted
cleanup()
End