ThePower business school

🇪🇸 UseEffect ReactJS[1]

Introducción

Este Hook nos permite "escuchar" o "engancharnos" a los eventos en el tiempo y poder ejecutar código de forma dinámica, vamos a comenzar por algo sencillo como ejecutar un código cuando el componente se monta en el DOM.

Gestión de la asincronia

Vamos crear un componente llamado `EffectOnLoad.jsx`.

```
import { useState } from "react";

export const EffectOnLoad = () => {
  const [myName, setMyName] = useState("David");

  return (
    <>
      <h4>{myName}</h4>
      <input
        type="text"
        value={myName}
        onChange={(e) => setMyName(e.target.value)}
      />
    </>
  );
};
```

Hay muchas operaciones que realizamos justo cuando se carga el DOM del navegador como por ejemplo cargar el perfil de un usuario con la información provista de una API. También pueden ser operaciones que queremos ejecutar cuando cambie un valor o después de cada render que realiza React bajo nuestra demanda.

Pero ¿qué sucede si esas operaciones que queremos realizar no son síncronas? Pues que tendremos que usar el hook useEffect porque si lo realizamos en un componente funcional no obtendremos el resultado deseado, ya que estos se crean y se destruyen, y con este hook podremos gestionar los “side effects” de nuestras aplicaciones.

Vamos a ver cómo funciona con dos ejemplos sencillos, el primero será cambiar un nombre cuando se pinta el componente, y luego haremos otro simulando una llamada asíncrona con un `setTimeout`. Vamos con la lógica de nuestro componente 🚀.

En primer lugar queremos añadir un valor por defecto a nuestro state pero solamente cuando el componente se haya renderizado en el DOM y no de inicio, para ello usaremos useEffect.

```
useEffect(() => {
  setMyName("Ziggy Stardust");
}, []);
```

Os recuerdo que el `[]` vacío hace referencia a que no hay ninguna condición para re-ejecutar este efecto, por lo que solo ocurrirá una vez en la vida de nuestro componente.

De este modo nuestro componente queda `EffectOnLoad.jsx` queda.

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

export const EffectOnLoad: React.FC = () => {
  const [myName, setMyName] = useState("David");

  useEffect(() => {
    setMyName("Ziggy Stardust");
  }, []);

  return (
    <>
      <h4>{myName}</h4>
      <input
        type="text"
        value={myName}
```

```
        onChange={(e) => setMyName(e.target.value)}
      />
    </>
  );
};
```

Vamos a ver como funciona un poco el useEffect, es hook que tiene **dos parámetros:**

- El primero es obligatorio siendo un código que se ejecuta cuando es llamado (suele ser una llamada a un servidor, un timeout...) en forma de callback, que no debe ser una función async/await, pero si que pueden serlo las funciones declaradas en su interior.
- El segundo vamos a verlo con ejemplos a continuación, ya que consiste en un array de dependencias que queremos escuchar...

El hook useEffect se ejecuta **siempre** en el primer render de nuestro componente, y a partir de aquí tendremos control sobre cuando relanzarlo. Para los que vengan de componentes de clase de React, se podría decir que esto se parece al método componentDidMount que se usaba antes de los componentes funcionales, aunque no son exactamente lo mismo.

```
useEffect(() => {
  setMyName("Ziggy Stardust");
}, []);
```

Cuando añadimos un valor al array de dependencias, la función del useEffect se ejecutará también cada vez que cambia dicho valor, en este caso `myName`, y en el caso de nuestro componente del ejemplo anterior, cada vez que cambie el state.

```
useEffect(() => {
  setMyName("Ziggy Stardust");
}, [myName]);
```

Este ejemplo representa código un poco inadecuado porque dentro del useEffect estamos modificando el state que ya hemos cambiado previamente, pero más adelante veremos ejemplos en los que tiene sentido hacer algo parecido.

Si no pasamos el segundo parámetro a useEffect, se ejecutará también después de cada render y tendremos una ejecución continua en cualquier interacción de nuestra aplicación.

```
useEffect(() => {
  setMyName("Ziggy Stardust");
});
```

Para cerrar este ejemplo vamos a simular una llamada asíncrona con un setTimeout y veremos cómo trabaja el useEffect.

```
useEffect(() => {
  // Imagina que esto fuese la respuesta de una API
  setTimeout(() => {
    setMyName("Ziggy Stardust");
  }, 1500);
}, []);
```

Nuestro resultado es el siguiente:

Al recargar tarda 1,5s en ejecutar nuestro código del useEffect ... ⏰