Material complementario - Manejo de efectos en React

¡Veamos el siguiente ejercicio!

Para desarrollar este ejercicio debemos crear una nueva aplicación en React, una vez hecho, limpiaremos la aplicación para trabajar con el archivo App.jsx.

- **Paso 1:** En nuestro App.jsx vamos a importar de react useState para trabajar con un formulario de dos input, uno para email y otro para password. Seguidamente, definiremos un estado para cada campo.

```
// Estado del formulario
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
```

- **Paso 2:** Definimos la conexión de nuestro formulario con los inputs de HTML haciendo uso de las propiedades value y name y además, el uso del método onChange para capturar lo que el usuario escriba y que esa información se almacene en el nuevo estado. Veamos cómo queda el return de nuestro componente App.jsx.

```
return (
    <>
      <h1>Manejo de efectos en react</h1>
      <form>
        <input
          type="email"
          placeholder="nombre"
          name={email}
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <input
          type="password"
```

```
          placeholder="password"
          name={password}
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
      </form>
    </>
  );
```

Comprueba la captura del nuevo estado para cada campo del formulario haciendo uso de la herramienta react developer tools.

- **Paso 3:** Ya que tenemos nuestro formulario configurado, haremos uso del useEffect. Debemos importarlo al inicio junto con nuestro useState, quedando de la siguiente manera:

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

Ya que tenemos la importación hecha, debemos revisar algunos aspectos teóricos asociados a este Hook de react y comprender su sintaxis.

```
useEffect(() => {
    //Primer nivel de montaje
})
```

El código anterior nos muestra la sintaxis de un useEffect en react. Es una función que recibe un callback y este callback puede ejecutar código en 3 niveles. En este momento vemos que tenemos un primer nivel de montaje del componente, el que corresponde a la fase de cuando nuestro componente es mostrado por primera vez.

Volvamos al ejercicio y ejecutemos alguna acción cuando el componente es montado la primera vez.

- **Paso 4:** mostrando un mensaje en consola cuando el componente App.jsx es montado la primera vez en el DOM.

```
//Efecto cuando el componente es cargado la primera vez
  useEffect(() => {
    console.log("Componente App cargado");
  });
```

Una vez realizado este paso, veremos en la consola del navegador el mensaje que hemos asignado de impresión.

Si realizas algunas pruebas, verás que al escribir en los campos del formulario, el mensaje que definimos en nuestro console.log() se ejecuta tantas veces como escribamos en cada input. Esto ocurre porque React está detectando cambios en el estado y está volviendo a hacer render del componente App.

Este es un comportamiento no tan lógico ni esperado, es decir, no queremos que por cada acción, incluso letra que el usuario escriba en el formulario, se realice un "re-render". Para evitar esto, useEffect nos permite especificar un arreglo de dependencias al finalizar su instrucción. Veamos dónde se ubica este arreglo de dependencias.

- **Paso 5:** Agregando control de render al componente App.jsx con el arreglo de dependencias de useEffect.

```
  //Efecto cuando el componente es cargado la primera vez
  useEffect(() => {
    console.log("Componente cargado");
  }, []);
```

Con este arreglo de dependencias le estamos diciendo que se ejecute el mensaje del console.log() solo una vez y es cuando el componente se carga en la primera ocasión. Si probamos ahora en nuestro formulario, al escribir en los campos no se ejecuta el "re-render".

- **Paso 6:** Este arreglo de dependencias también lo podemos utilizar para detectar cuando algún elemento de nuestro DOM cambia y una vez detectado realizar alguna acción. Entonces, en nuestro arreglo de dependencias le diremos ahora que imprima un console.log() cuando el campo email cambie. Veamos cómo queda esto en código.

```
 //Efecto cuando el campo email cambia en el formulario
  useEffect(() => {
```

```
    console.log("El email cambió su estado");
  }, [email]);
```

Podemos hacer lo mismo con el password

```
//Efecto cuando el campo password cambia en el formulario
  useEffect(() => {
    console.log("El password cambió su estado");
  }, [password]);
```

El hook useEffect tiene también dentro de su callback una fase de limpieza o desmontado, que nos permitirá también de manera controlada ejecutar alguna acción cuando un componente deje de mostrarse en el DOM. La sintaxis del useEffect con la fase de limpieza es la siguiente:

```
useEffect(() => {
    //Montaje inicial

    return () => {
      //Limpieza
    }
}, [third]) //Actualización
```

- **Paso 7:** Utilicemos la fase de limpieza a partir del siguiente planteamiento. Imaginemos que queremos mostrar un componente llamado Contenido una vez que el usuario llene el formulario y la contraseña sea igual a `"123456"`. Para lograrlo, debemos entonces primero crear el componente `Contenido.jsx` que de momento solo retornará un título.

```
//Contenido.jsx
const Contenido = () => {
  return <h1>Desde contenido</h1>;
};

export default Contenido;
```

- **Paso 8:** En nuestro App.jsx agregaremos condicionalmente la carga de este componente. Esta carga condicional la haremos después de nuestro formulario y el código quedaría de la siguiente manera.

```
{/* Componente cargado condicionalmente si el password es igual a 123456
*/}
{password == "123456" && <Contenido />}
```

Estamos diciendo de manera condicionada, si el password es igual a `"123456"`, entonces muestra el componente `Contenido.jsx`. Otra forma un poco más larga, pero que igual funciona es la siguiente, selecciona la forma que sea más entendible para ti.

```
{password == "123456" ? <Contenido /> : null}
```

- **Paso 9:** en el componente `Contenido.jsx` haremos uso del useEffect y la fase de desmontaje que va indicada dentro del return del callback. El código queda de la siguiente manera.

```
//Contenido.jsx
import { useEffect } from "react";

const Contenido = () => {
  useEffect(() => {
    console.log("Componente montado");
    return () => {
      console.log("Componente desmontado");
    };
  }, []);

  return <h1>Desde contenido</h1>;
};

export default Contenido;
```

Resumen

Con este ejercicio comprendimos el uso de un nuevo hook de react llamado useEffect, e intencionamos a través de varias etapas su uso y sintaxis para comprender qué sucede durante la fase de montaje, actualización y desmontaje.

Te invitamos a seguir practicando para consolidar los aprendizajes.

Nota: Podrás acceder al repositorio de este ejercicio en el LMS con el nombre **Repositorio Ejercicio - Manejo de efectos en react.**