FullStack Developer

React – Estado de los Componentes y Custom Hooks

Estado de los Componentes (Component State)

El estado en un componente de React es un objeto que determina cómo se renderiza el componente y cómo reacciona a las interacciones del usuario. Cuando el estado de un componente cambia, React vuelve a renderizar ese componente para reflejar los cambios en la interfaz de usuario.

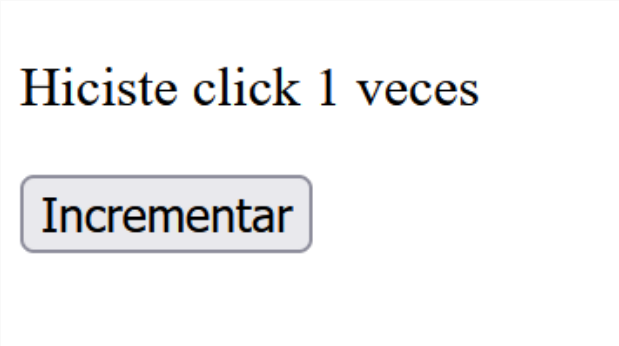
Estado Local vs Global: El estado puede ser local a un componente o compartido (global) a través de varios componentes. Para el estado global, se suele utilizar un gestor de estado como Redux o el Context API de React. (Tema que veremos más adelante pero que vale la pena ir mencionando)

```jsx
import { useState } from 'react'
import './App.css'

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Hiciste click {count} veces</p>
      <button onClick={() => setCount(count + 1)}>
        Incrementar
      </button>
    </div>
  );
}

export default App
```



Renderizado Condicional

Supongamos que tenemos un componente que muestra un mensaje de "Disponible" o "No Disponible" basado en el estado de disponibilidad de un producto.

```
function Disponibilidad() {
  const [estaDisponible, setEstaDisponible] = useState(true);

  return (
    <div>
      <h1 className={estaDisponible ? 'disponible' : 'no-disponible'}>
        {estaDisponible ? 'Producto Disponible' : 'Producto No Disponible'}
      </h1>
      <button onClick={() => setEstaDisponible(!estaDisponible)}>
        {estaDisponible ? 'Marcar como No Disponible' : 'Marcar como Disponible'}
      </button>
    </div>
  );
}
```

Importamos el componente en App()

```jsx
function Disponibilidad() {
  const [estaDisponible, setEstaDisponible] = useState(true);

  return (
    <div>
      <h1 className={estaDisponible ? 'disponible' : 'no-disponible'}>
        {estaDisponible ? 'Producto Disponible' : 'Producto No Disponible'}
      </h1>
      <button onClick={() => setEstaDisponible(!estaDisponible)}>
        {estaDisponible ? 'Marcar como No Disponible' : 'Marcar como Disponible'}
      </button>
    </div>
  );
}
```

Consumiendo una API y Manejando el Estado

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

function ListaDePaises() {
  const [paises, setPaises] = useState([]); // Estado para almacenar los países
  const [cargando, setCargando] = useState(true); // Estado para manejar la carga
  const [error, setError] = useState(null); // Estado para manejar errores

  useEffect(() => {
    // Realiza la solicitud a la API cuando el componente se monta
    fetch('https://restcountries.com/v3.1/all')
      .then((response) => {
        if (!response.ok) {
          throw new Error('Error al obtener los datos');
        }
        return response.json();
      })
      .then((data) => {
        setPaises(data); // Actualiza el estado con los datos obtenidos
        setCargando(false); // Cambia el estado de carga a falso
      })
      .catch((error) => {
        setError(error.message); // Captura y almacena cualquier error
        setCargando(false); // Cambia el estado de carga a falso incluso si hay un error
      });
  }, []); // El array vacío significa que el efecto solo se ejecuta una vez, al montar el componente

  if (cargando) {
    return <p>Cargando países...</p>; // Muestra un mensaje de carga mientras se obtienen los datos
  }
```

```
  if (error) {
    return <p>Error: {error}</p>; // Muestra un mensaje de error si la solicitud falla
  }

  return (
    <div>
      <h1>Lista de Países</h1>
      <ul>
        {paises.map((pais) => (
          <li key={pais.cca3}>
            {pais.name.common} - Capital: {pais.capital ? pais.capital[0] : 'N/A'}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ListaDePaises;
```

Importamos el componente en App() (vamos a borrar lo anterior para que no se superpongan los conceptos).

Y obtenemos esto

Explicación:

1.Estado Inicial:

- •paises: Es un array que se inicializa vacío y que se llenará con los datos obtenidos de la API de países.
- •cargando: Es un booleano que indica si la solicitud a la API está en progreso.
- •error: Es una cadena que almacenará cualquier mensaje de error si la solicitud falla.

2.useEffect para Consumir la API:

- •useEffect se utiliza para realizar la solicitud a la API cuando el componente se monta.
- •Se usa fetch para obtener los datos de la API. Si la respuesta es exitosa, los datos se almacenan en el estado paises y cargando se establece en false.
- •Si hay un error en la solicitud, se captura y se almacena en error, y cargando también se establece en false.

3.Renderizado Condicional:

- •Mientras cargando es true, se muestra un mensaje de "Cargando países...".
- •Si error contiene un mensaje, se muestra ese mensaje en lugar de la lista de países.
- •Si la solicitud fue exitosa y cargando es false, se muestra una lista de países utilizando el estado paises.
- •Cada país se muestra con su nombre común y su capital. Si un país no tiene una capital definida, se muestra "N/A".

Resumen:

Este ejemplo te muestra cómo manejar el estado en un componente React que consume una API de países. La estructura del código es similar al ejemplo anterior, pero ahora trabajamos con datos más específicos como nombres de países y sus capitales. Este patrón es muy útil para construir aplicaciones que interactúan con APIs externas y manejan datos dinámicos.

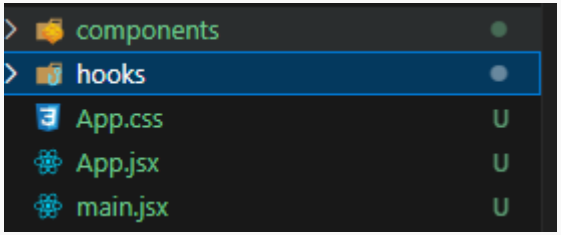
Custom Hooks

Un **custom hook** en React es una función que te permite extraer y reutilizar lógica de estado y efectos en componentes funcionales. Los custom hooks te permiten encapsular lógica que puede ser compartida entre varios componentes sin necesidad de repetir código.

Características clave de los Custom Hooks:
•**Reutilización de lógica:** Podemos utilizar el mismo hook personalizado en diferentes componentes.
•**Composición:** Podemos combinar varios hooks (como useState, useEffect, etc.) dentro de un custom hook para manejar lógica más compleja.
•**Convención de nombres:** Los custom hooks siguen la convención de nombres de empezar con "use", como useFetch, useToggle, etc. Esto es importante porque React usa esta convención para identificar que la función es un hook.

Cuándo usar Custom Hooks:
•Cuando tenemos lógica que se repite en varios componentes.
•Cuando queremos separar la lógica del estado del código de presentación (UI).
•Para mejorar la organización y legibilidad del código.

Vamos a hacer una carpeta llamada hooks para colocarlos ahí y la carpeta components donde usaremos estos hooks



Custom Hook: useToggle

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

function useToggle(initialValue = false) {
  const [value, setValue] = useState(initialValue);

  const toggle = () => {
    setValue((prevValue) => !prevValue);
  };

  return [value, toggle];
}

export default useToggle;
```

Uso:

```jsx
import useToggle from './useToggle';

function ToggleButton() {
  const [isToggled, toggle] = useToggle();

  return (
    <button onClick={toggle}>
      {isToggled ? 'ON' : 'OFF'}
    </button>
  );
}

export default ToggleButton
```

Explicación: useToggle es un hook simple que devuelve el valor actual (value) y una función (toggle) para alternarlo entre true y false.

Custom Hook: useFetch

Este custom hook realiza una solicitud fetch a una API y maneja el estado de los datos, la carga y los errores.

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

function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch(url)
      .then((response) => {
        if (!response.ok) {
          throw new Error('Error en la solicitud');
        }
        return response.json();
      })
      .then((data) => {
        setData(data);
        setLoading(false);
      })
      .catch((error) => {
        setError(error.message);
        setLoading(false);
      });
  }, [url]);

  return { data, loading, error };
}

export default useFetch;
```

Uso

```jsx
import useFetch from '../hooks/useFetch';

function ListaDePaises() {
  const { data: paises, loading, error } = useFetch('https://restcountries.com/v3.1/all');

  if (loading) return <p>Cargando países...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
    <div>
      <h1>Lista de Países</h1>
      <ul>
        {paises.map((pais) => (
          <li key={pais.cca3}>
            {pais.name.common} - Capital: {pais.capital ? pais.capital[0] : 'N/A'}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ListaDePaises;
```

Explicación:

1.Custom Hook useFetch:

- **URL Dinámica:** El hook recibe una URL como parámetro y realiza la solicitud cuando el componente se monta.
- **Manejo de Estado:** El hook gestiona el estado de los datos (data), el estado de carga (loading), y el manejo de errores (error).
- **fetchData:** Es una función asíncrona que realiza la solicitud a la API, maneja errores, y actualiza los estados correspondientes.

2.Componente ListaDePaises:

- **Uso del Hook:** Se utiliza el custom hook useFetch para hacer la solicitud a la API de países.
- **Renderizado Condicional:**
  - Si los datos están cargando, se muestra un mensaje de carga.
  - Si ocurre un error, se muestra un mensaje de error.
  - Si la solicitud es exitosa, se renderiza una lista de países, mostrando su nombre común y su capital (si está disponible).

Este patrón de uso del hook useFetch con la API de países es flexible y reutilizable, permitiéndote consumir cualquier API simplemente cambiando la URL proporcionada al hook.

```
const { data: paises, loading, error } = useFetch('https://restcountries.com/v3.1/all');
```

1. Llamada al Custom Hook useFetch:
•useFetch('https://restcountries.com/v3.1/all') invoca el custom hook con la URL de la API de países. Este hook realiza la solicitud a la API y devuelve un objeto que contiene los estados data, loading y error.

2. Desestructuración de Objetos:
•La desestructuración es una característica de JavaScript que permite extraer propiedades de un objeto y asignarlas a variables.
•En este caso, el objeto retornado por useFetch podría ser algo como esto:
javascript
{ data: [/* Array de países obtenidos de la API */], loading: true/false, // Estado de carga error: null/Message, // Mensaje de error si ocurre algún problema }

3. Alias para la Propiedad data:

•data: paises es una forma de desestructuración que le asigna a la propiedad data un nuevo nombre de variable, en este caso, paises.

- **•Original:** La propiedad se llama data dentro del objeto retornado por useFetch.
- **•Alias:** Se asigna a una nueva variable llamada paises para hacer el código más semántico en el contexto de la aplicación, ya que data representa una lista de países.

4. Asignación de las Variables:

•**paises:** Contendrá el valor de data, que es el array de países retornado por la API.

•**loading:** Contendrá el estado de carga (true o false) que indica si los datos aún se están cargando.

•**error:** Contendrá un mensaje de error si la solicitud falla o null si no hubo ningún error.

Resumen:

Esta línea de código:

•Llama al hook useFetch con una URL específica.

•Extrae tres propiedades del objeto retornado (data, loading, error).

•Renombra data a paises para que sea más claro que se está manejando una lista de países.

Es una forma concisa y clara de manejar el resultado de una solicitud a la API, facilitando el uso de los datos y el manejo de estados en el componente.

Ejercicios:

Replicar la app países con custom hooks y además aplicar algunos estilos de css.