Mientras que useState nos da "memoria" para nuestros componentes, useEffect nos permite ejecutar "efectos secundarios". Un efecto secundario es cualquier lógica que necesite interactuar con el mundo exterior al componente, como:

- **Cargar datos de una API.** (¡Esto es lo que haremos!)
- Configurar suscripciones (como a un chat en tiempo real).
- Manipular el DOM directamente (aunque se hace con menos frecuencia en React).

En este tutorial, construiremos un **Directorio de Usuarios** que carga datos desde una API pública en internet.

Tutorial: Carga de Datos desde una API con useEffect

Objetivo del Proyecto:

Crearemos un componente que, al cargarse por primera vez, hará una petición a una API para obtener una lista de usuarios y la mostrará en pantalla. También manejaremos los estados de "cargando" y "error" para que la experiencia del usuario sea más clara.

Paso 1: Preparar el Componente

Siguiendo la estructura de nuestro proyecto anterior:

1. Dentro de la carpeta src/components, crea una nueva carpeta llamada UserDirectory.
2. Dentro de UserDirectory, crea dos archivos: UserDirectory.js y UserDirectory.css.

Paso 2: Definir los Estados del Componente

Para cargar datos, no solo necesitamos un lugar donde guardarlos. También debemos saber si la carga está en proceso o si ha ocurrido un error.

1. Abre src/components/UserDirectory/UserDirectory.js y añade el siguiente código inicial:

JavaScript

```
import React, { useState, useEffect } from 'react';
import './UserDirectory.css';


const UserDirectory = () => {
  // Estado para guardar la lista de usuarios
  const [users, setUsers] = useState([]);
```

```
  // Estado para saber si los datos están cargando
  const [loading, setLoading] = useState(true);
  // Estado para guardar un posible error
  const [error, setError] = useState(null);

  // Aquí es donde usaremos useEffect para cargar los datos

  // Aquí mostraremos la UI basada en los estados de loading, error y users

  return (
    <div className="user-directory">
      <h2>Directorio de Usuarios</h2>
      {/* El contenido dinámico irá aquí */}
    </div>
  );
};

export default UserDirectory;
```

🔍 **Desglose del Estado:**

- **users**: Un arreglo vacío que llenaremos con los datos de la API.
- **loading**: Un booleano que empieza en true. Lo usaremos para mostrar un mensaje de "Cargando..." mientras esperamos la respuesta de la API.
- **error**: Inicialmente null. Si algo sale mal con la petición, guardaremos el mensaje de error aquí.

Paso 3: Usar useEffect para Cargar los Datos

Este es el paso más importante. Usaremos el hook useEffect para realizar la petición a la API justo después de que el componente se monte por primera vez en la pantalla.

1. Modifica UserDirectory.js para añadir el useEffect:

JavaScript

```
import React, { useState, useEffect } from 'react';
import './UserDirectory.css';

const UserDirectory = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // useEffect para realizar efectos secundarios
  useEffect(() => {
    // Usamos la API 'fetch' del navegador para hacer la petición
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        if (!response.ok) {
          throw new Error('La respuesta de la red no fue satisfactoria');
        }
        return response.json();
      })
      .then(data => {
        setUsers(data); // Guardamos los usuarios en el estado
        setError(null); // Limpiamos cualquier error previo
      })
      .catch(error => {
        setError(error.message); // Guardamos el mensaje de error
        setUsers([]); // Limpiamos los datos de usuarios
      })
      .finally(() => {
        setLoading(false); // La carga ha terminado (sea con éxito o error)
      });
```

```
}, []); // <-- El arreglo de dependencias vacío es MUY importante


// ... el resto del componente
```

Desglose del useEffect:

- **useEffect(() => { ... }, []);**: El hook useEffect recibe dos argumentos:
    1. Una **función** (el "efecto") que contiene la lógica que queremos ejecutar. En nuestro caso, la petición fetch.
    2. Un **arreglo de dependencias**. Este arreglo le dice a React *cuándo* debe volver a ejecutar el efecto.
- **[] (Arreglo de Dependencias Vacío):** Cuando pasamos un arreglo vacío, le estamos diciendo a React: **"Ejecuta este efecto UNA SOLA VEZ, justo después del primer renderizado"**. Esto es perfecto para cargar datos iniciales. Si no pusiéramos este arreglo, el fetch se ejecutaría en un bucle infinito.

Paso 4: Renderizado Condicional

Ahora que tenemos los estados loading y error, podemos mostrar diferentes cosas en la pantalla según el estado actual de la carga de datos.

1. Completa la sección del return en UserDirectory.js:

JavaScript

```
// ... (importaciones, estados y useEffect)


return (
  <div className="user-directory">
    <h2>Directorio de Usuarios</h2>


    {/* 1. Si está cargando, muestra un mensaje */}
    {loading && <p>Cargando usuarios...</p>}


    {/* 2. Si hay un error, muestra el error */}
    {error && <p className="error-message">Error: {error}</p>}
```

```
      {/* 3. Si no hay error y no está cargando, muestra la lista */}
      {!loading && !error && (
        <ul>
          {users.map(user => (
            <li key={user.id} className="user-card">
              <h3>{user.name}</h3>
              <p>📧 {user.email}</p>
              <p>🌐 {user.website}</p>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default UserDirectory;
```

Este patrón de renderizado condicional es extremadamente común y poderoso en React.

Paso 5: Estilos e Integración Final

1. Añade algunos estilos a src/components/UserDirectory/UserDirectory.css para que se vea bien:

CSS

```
.user-directory {
  max-width: 800px;
  margin: 20px auto;
  padding: 20px;
}
```

```
.error-message {
  color: #D8000C;
  background-color: #FFD2D2;
  padding: 10px;
  border-radius: 4px;
}

.user-directory ul {
  list-style: none;
  padding: 0;
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 20px;
}

.user-card {
  background-color: #f9f9f9;
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 15px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
  transition: transform 0.2s ease-in-out;
}

.user-card:hover {
  transform: translateY(-5px);
}

.user-card h3 {
```

```
  margin-top: 0;
  color: #333;
}

.user-card p {
  margin: 5px 0;
  font-size: 0.9em;
  color: #666;
}
```

2. Finalmente, usa tu nuevo componente en src/App.js:

JavaScript

```
import React from 'react';
import './App.css';
import Header from './components/Header/Header';
import UserDirectory from './components/UserDirectory/UserDirectory'; // Importamos

function App() {
  return (
    <div className="App">
      <Header />
      <main>
        <UserDirectory /> {/* Lo usamos aquí */}
      </main>
    </div>
  );
}

export default App;
```

¡Resultado Final!

Ejecuta npm start. Al cargar la página, verás brevemente el mensaje "Cargando usuarios...", y luego aparecerá una lista de usuarios con sus datos, traídos directamente desde internet.

Resumen de Conceptos Clave

- **useEffect**: El hook para manejar efectos secundarios como peticiones a APIs.
- **Arreglo de Dependencias []**: La clave para que un efecto se ejecute solo una vez al montar el componente.
- **Estados de Carga y Error**: La práctica profesional de manejar los diferentes estados de una petición de datos para informar al usuario.
- **Renderizado Condicional**: Usar operadores lógicos (&&, !) en JSX para mostrar diferentes elementos de la UI según el estado de la aplicación.