React Nivel Intermedio: Estado y Eventos 🚀

Introducción 📝

En este nivel, profundizaremos en el manejo del **estado** y los **eventos** en React. Estos conceptos son fundamentales para crear interfaces de usuario dinámicas e interactivas. El **estado** permite a los componentes almacenar y gestionar datos que cambian con el tiempo, mientras que los **eventos** permiten interactuar con el usuario, detectando acciones como clics o cambios en inputs.

Temas que Cubriremos 📚

1. **El Estado en React:** ¿Qué es y cómo manejarlo con hooks?
2. **Eventos en React:** Uso de eventos para crear interactividad.
3. **Manejo del Estado con `useState`:** Uso de `useState` para gestionar el estado local.
4. **Ejemplos Prácticos:** Ejercicios de manejo de estado y eventos.
5. **Buenas Prácticas:** Consejos para evitar errores comunes y escribir código más limpio.

---

1. El Estado en React 🛠️

El **estado** es un objeto que contiene datos dinámicos que pueden cambiar a lo largo del tiempo. A diferencia de las **props**, que son inmutables, el estado es mutable y controlado por el componente, lo que permite actualizar la UI automáticamente cuando el estado cambia.

Manejo del Estado con `useState`

En React moderno, usamos el hook `useState` para manejar el estado en componentes funcionales.

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

function Contador() {
  const [contador, setContador] = useState(0);

  return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}
```

En este ejemplo:

- `useState(0)` inicializa el estado `contador` con el valor `0`.
- `setContador` es la función que se usa para actualizar el estado.
- Cuando el botón es clicado, se incrementa el contador y se actualiza la UI automáticamente.

---

2. **Eventos en React** 🖱️

Los **eventos** en React funcionan de forma similar a los eventos en HTML, pero con una sintaxis propia y estandarizada. React usa camelCase para nombrar los eventos (`onClick`, `onChange`, etc.) y las funciones manejadoras se definen directamente en JavaScript.

Ejemplo de Manejo de Evento `onClick`

```
function BotonClick() {
  const manejarClick = () => {
    console.log("Botón clickeado");
  };

  return <button onClick={manejarClick}>Haz clic aquí</button>;
}
```

- El evento `onClick` activa la función `manejarClick` cuando el botón es clicado.
- Es una forma sencilla de agregar interactividad a los componentes.

---

3. **Ejercicios Prácticos** 📝

Ejercicio 1: Contador Básico con `useState`

En este ejercicio, crearemos un componente que muestra un contador y un botón para incrementarlo.

Código:

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

function ContadorIncremento() {
  const [contador, setContador] = useState(0);

  return (
    <div>
      <h2>Contador: {contador}</h2>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}

export default ContadorIncremento;
```

Ejercicio 2: Estado con Input de Texto

En este ejercicio, crearemos un input de texto y mostraremos el valor ingresado en tiempo real.

Código:

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

function InputTexto() {
  const [texto, setTexto] = useState("");

  const manejarCambio = (event) => {
```

```
    setTexto(event.target.value);
  };

  return (
    <div>
      <input type="text" value={texto} onChange={manejarCambio} />
      <p>Texto ingresado: {texto}</p>
    </div>
  );
}

export default InputTexto;
```

- El evento `onChange` actualiza el estado `texto` cada vez que el usuario escribe en el input.
- El valor ingresado se muestra en tiempo real debajo del input.

---

4. **Buenas Prácticas al Manejar Estado y Eventos** 🔍

- **No mutar el estado directamente:** Siempre usa la función de actualización del estado (`setState` o `setNombreEstado`).

  ❌ Incorrecto:

```
contador = contador + 1;
```

  ✅ Correcto:

```
setContador(contador + 1);
```

- **Definir funciones manejadoras fuera del JSX:** Esto mejora la legibilidad y evita crear funciones anónimas innecesarias.
- **Usar destructuración:** Para acceder a props o eventos de forma más clara.

```
const manejarCambio = ({ target }) => {
setTexto(target.value);
};
```

Conclusión 🎯

En este nivel, hemos aprendido a manejar el **estado** con `useState` y a utilizar **eventos** para interactuar con el usuario. Estos son conceptos esenciales para cualquier aplicación React, ya que permiten que la UI responda a los cambios en los datos y a las acciones del usuario. A medida que avances, podrás combinar estos conocimientos con hooks más avanzados como `useEffect` y manejar el estado global usando herramientas como **Context API** o **Redux**.

¡Sigue practicando y experimentando con diferentes estados y eventos para dominar React! 🚀💻