Estado de los componentes y eventos 

Objetivos

Conocer que es un estado en React

Manejar estado en los componentes con useState

Eventos en React

Recuperar datos desde formularios mediante eventos

Validar datos de un formulario

¿Qué son los estados en React?

Los estados en React se refieren a la representación actual de datos en un componente y cómo estos datos pueden cambiar a lo largo del tiempo debido a interacciones o eventos en la aplicación. Los estados permiten que los componentes sean dinámicos y reactivos, lo que es esencial para crear interfaces de usuario interactivas.

En React, los estados son administrados mediante el uso del hook useState. Los estados son importantes porque permiten que los componentes respondan a cambios y actualicen su contenido sin necesidad de recargar toda la página. Aquí hay algunas características clave de los estados en React:

Dinamismo:
Los estados permiten que los componentes sean dinámicos y cambien su contenido basado en eventos o interacciones del usuario.

Interactividad:
Los estados son fundamentales para crear componentes interactivos, como formularios, botones, contadores y más.

Actualización Eficiente:
React maneja la actualización de componentes de manera eficiente al comparar los estados anteriores y actuales, y solo actualiza los componentes que han cambiado.

Inmutabilidad:
Los estados en React deben ser tratados como inmutables. Esto significa que no se deben modificar directamente los estados, sino crear nuevos estados con los cambios necesarios.

Renderizado Reactivo:
Cuando un estado cambia, React vuelve a renderizar automáticamente el componente afectado y sus descendientes para reflejar el nuevo estado.

El hook useState

useState es uno de los hooks proporcionados por React, y es utilizado en componentes funcionales para agregar y gestionar estados locales. Los hooks son una característica introducida en React 16.8 que permite a los componentes funcionales tener características previamente reservadas para componentes de clase, como el manejo de estados y efectos.

La función useState se utiliza para declarar una variable de estado en un componente funcional y devuelve un par de valores: el valor actual del estado y una función para actualizar ese estado.

Ejemplo de sintaxis:

```
const [stateValue, setStateFunction] = useState(initialValue);
```

Donde:

stateValue:
Este es el valor actual del estado que estás gestionando. Es similar a un getter, ya que te permite acceder al valor actual del estado en tu componente.

setStateFunction:
Esta es la función que te permite actualizar el estado. Es similar a un setter, ya que al llamar a esta función con un nuevo valor, React actualizará el estado y volverá a renderizar el componente con el nuevo valor.

Ejemplo de codigo aplicando useState:

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

const Counter = () => {
  Usamos el gancho useState para inicializar el estado "count"
  con el valor 0
  const [count, setCount] = useState(0);

  Función que se ejecuta cuando se hace clic en el botón de incremento
  const increment = () => {
    setCount(count + 1); // Actualiza el estado "count"
  };

  Función que se ejecuta cuando se hace clic en el botón de reinicio
  const reset = () => {
    setCount(0); // Reinicia el estado "count"
  };

  return (
    <div>
    <p>Contador: {count}</p>
      <button onClick={increment}>Incrementar</h1>
      <button onClick={reset}>Reiniciar</h1>
    </div>
  );
  );
}
export default Counter;
```

Eventos en React

En React, los eventos son acciones que ocurren en la interfaz de usuario, como hacer clic en un botón, mover el mouse sobre un elemento o escribir en un campo de entrada. React proporciona una forma de manejar estos eventos de manera similar a cómo se manejan en JavaScript tradicional, pero con algunas diferencias y conveniencias para trabajar en el entorno de componentes.

En React, los eventos se manejan utilizando atributos de JSX, similar a cómo se haría en HTML. Sin embargo, en lugar de utilizar cadenas para nombrar los manejadores de eventos, se utilizan referencias a funciones. Por ejemplo, para manejar el evento de clic en un botón:

Ejemplo de sintaxis:

```
function handleClick() {
  console.log("Botón clickeado");
}

const element = <button onClick={handleClick}>Haz clic</button>;
```

Eventos más usados

A continuación te dejo una lista de los eventos más utilizados

onClick:
Este evento se dispara cuando se hace clic en un elemento, como un botón, un enlace o cualquier otro elemento con el que el usuario pueda interactuar.

onChange:
Se utiliza principalmente en elementos de entrada (<input>, <textarea>, <select>) y se activa cuando el valor del elemento cambia. Es útil para manejar la entrada del usuario, como la escritura en un campo de texto.

onSubmit:
Este evento se activa cuando se envía un formulario. Se utiliza para capturar y procesar los datos del formulario antes de que se envíen al servidor.

onFocus y onBlur:
Se activan cuando un elemento gana o pierde el foco, respectivamente. Son útiles para realizar acciones cuando un usuario interactúa con un elemento de entrada.

Procesando datos de un formulario

En React, los formularios se manejan de manera similar a como se manejarían en HTML. Sin embargo, en lugar de utilizar cadenas para nombrar los manejadores de eventos, se utilizan referencias a funciones.

Para manejar los datos de un formulario, se utiliza el atributo value para establecer el valor del elemento de entrada y el evento onChange para manejar los cambios en el valor del elemento de entrada.A continuación veremos un ejemplo de un formulario que captura un nombre

Ejemplo:

```
import { useState } from 'react';
const Formulario = () => {
  const [nombre, setNombre] = useState("")
  const validarInput = () => {
  e.preventDefault()
  console.log(nombre)
  }
  return (
    <>
      <form onSubmit={validarInput}>
        <h3>{nombre}</h3>
        <div className="form-group">
          <input
            className="form-control"
            type="text"
            onChange={(e) => setNombre(e.target.value)}/>
          <button className="btn btn-dark mt-3" type="submit">
          Enviar</button>
          </div>
      </form>
    </>
  )
}
```

Explicación del código anterior

const [nombre, setNombre] = useState("");
Declaración del estado nombre utilizando el gancho useState. nombre es el valor actual del estado, y setNombre es la función que se utilizará para actualizar ese estado. Inicialmente, nombre se establece en una cadena vacía.

const validarInput = () => {...}
Esta función será llamada cuando el formulario se envíe. En este caso, solo muestra un console.log para demostrar que se disparó el evento de envío del formulario.

{nombre}:
Muestra el valor actual del estado nombre en un encabezado <h3>. capturar y procesar los datos del formulario antes de que se envíen al servidor.

<input onChange={(e) => setNombre(e.target.value)}/>:
Un campo de entrada que está vinculado al estado nombre. Cuando el usuario escriba en el campo, la función setNombre se llamará para actualizar el estado nombre.

<button className="btn btn-dark mt-3" type="submit">Enviar</button>:
Un botón que envía el formulario. Cuando se hace clic en él, se ejecutará la función validarInput.

validando datos de un formulario

En el ejemplo anterior, el formulario se envía cuando se hace clic en el botón Enviar. Sin embargo, si el usuario no ha ingresado ningún valor para el nombre, el formulario se enviará de todos modos. Para evitar esto, podemos agregar una validación para asegurarnos de que el usuario haya ingresado un nombre antes de enviar el formulario.

Para hacer esto, podemos agregar una condición a la función validarInput para verificar si el estado nombre está vacío. Si está vacío, se mostrará una alerta y el formulario no se enviará. De lo contrario, el formulario se enviará y se mostrará el nombre ingresado.

Ejemplo:

```
import { useState } from 'react';
const Formulario = () => {
  const [nombre, setNombre] = useState("")
  const validarInput = () => {
    if (nombre.trim() === "") {
      alert("Por favor ingrese un nombre")
    } else {
      console.log(nombre)
    }
  }
  return (
    <>
      <form onSubmit={validarInput}>
        <h3>{nombre}</h3>
        <div className="form-group">
          <input
            className="form-control"
            type="text"
            onChange={(e) => setNombre(e.target.value)}/>
          <button className="btn btn-dark mt-3" type="submit">
          Enviar</button>
          </div>
```

```
        </form>
      </>
    )
  }
```

Manejando el error con useState

En el ejemplo anterior, se muestra una alerta cuando el usuario no ingresa un nombre. Sin embargo, esto no es ideal, ya que la alerta bloquea la interfaz de usuario y no se puede cerrar. Para solucionar esto, podemos utilizar el estado para mostrar u ocultar la alerta.

Para hacer esto, podemos agregar un nuevo estado llamado error y establecerlo en false inicialmente. Luego, cuando el usuario no ingrese un nombre, podemos establecer el estado error en true. Finalmente, podemos mostrar u ocultar la alerta dependiendo del estado error.

Ejemplo:

```
import { useState } from 'react';
const Formulario = () => {
  const [nombre, setNombre] = useState("")
  const [error, setError] = useState(false)
  const validarInput = () => {
    if (nombre.trim() === "") {
      setError(true)
    } else {
      console.log(nombre)
    }
    setError(false)
  }
  return (
    <>
      <form onSubmit={validarInput}>
        <h3>{nombre}</h3>
        <div className="form-group">
          <input
            className="form-control"
            type="text"
            onChange={(e) => setNombre(e.target.value)}/>
          <button className="btn btn-dark mt-3" type="submit">
          Enviar</button>
          </div>
      </form>
      {error && <div className="alert alert-danger">Por favor ingrese
      un nombre</div>}
```

```
        </>
      )
    }
```

Ahora que ya conocemos las bases para manejar formularios y estados, vamos a seguir practicando y aplicando estilos a nuestro formulario.