Implementando el hook useState

La meta de esta sección es finalizar el desarrollo de la aplicación de tipo contador: **counterApp**, se busca implementar el hook **useState** que permita incrementar el valor del contador y a su vez visualizar el cambio en la aplicación web.

Continuando el proyecto

Sigue estos pasos:

Paso 7: Implementar el hook

1. Procede a importar el hook **useState** desde la biblioteca **React**. Puedes escribir **useState** y pulsar la tecla **Enter** para importar automáticamente. Este hook va a permitir agregar el estado al componente.

import { useState } from "react";

export const CounterApp = () => {

    const [] = useState()
    //let counter = 0;

    const counterIncrement = () => {
        counter = counter + 1
        console.log("click " + counter);
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={() => {
            counterIncrement()
        }}>Incrementar contador +1</button>
    </>

}

1. Utiliza **useState** para crear una variable de estado **counter**, el cual va a almacenar el valor del contador. Este hook devuelve un arreglo desestructurado, donde el primer elemento es el estado actual (**counter**) y el segundo elemento es una función (**setCounter**) que permite cambiar o actualizar el estado.

import { useState } from "react";

export const CounterApp = () => {

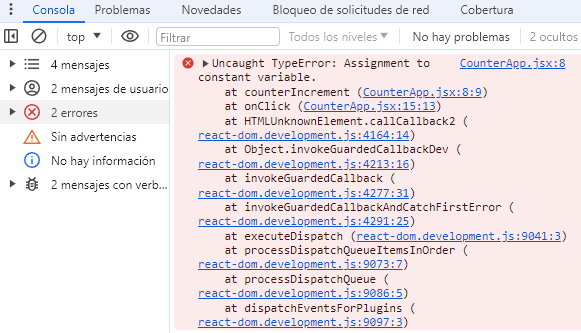
    const [counter, setCounter] = useState();
    //let counter = 0;

    const counterIncrement = () => {
        counter = counter + 1
        console.log("click " + counter);
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={() => {
            counterIncrement()
        }}>Incrementar contador +1</button>
    </>

}

**Nota:** En el caso de pulsar el botón en el navegador, consecutivamente va a mostrar un error en la consola, el cual indica que se está asignando un valor a una constante y no se puede alterar ese valor.



1. En la función **counterIncrement**, en lugar de actualizar directamente la variable **counter**, utiliza la función **setCounter** para incrementar el valor del contador. Esto asegura que el estado se actualice de manera correcta y segura.

import { useState } from "react";

export const CounterApp = () => {

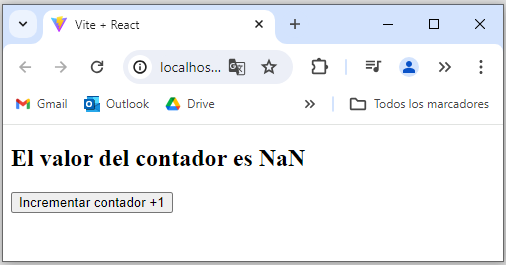
    const [counter, setCounter] = useState();
    //let counter = 0;

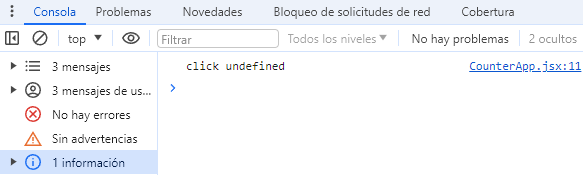
    const counterIncrement = () => {
        // counter = counter + 1
        setCounter(counter + 1)
        console.log("click " + counter);
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={() => {
            counterIncrement()
        }}>Incrementar contador +1</button>
    </>

}


1. Al pulsar el botón en el navegador, inicialmente puede mostrar un valor **NaN** o **undefined** en el título y en la consola. Esto se debe a que el estado inicial de **counter** está indefinido. En el próximo paso, se solucionara este problema al inicializar el estado con un valor predeterminado.





Paso 8: Establecer un valor por defecto en useState

1. **useState** es un hook que recibe un argumento que representa el valor por defecto o inicial del estado. En este caso, si el contador va empezar en **0**, escribe el valor **0** como argumento a **useState**.

import { useState } from "react";

export const CounterApp = () => {

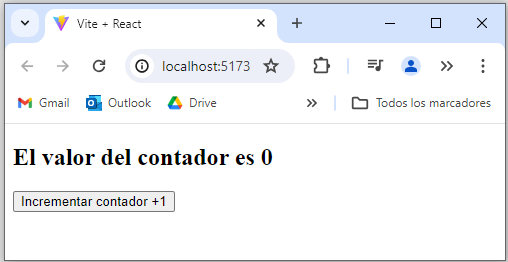
    const [counter, setCounter] = useState(0);
    //let counter = 0;

    const counterIncrement = () => {
        // counter = counter + 1
        setCounter(counter + 1)
        console.log("click " + counter);
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={() => {
            counterIncrement()
        }}>Incrementar contador +1</button>
    </>

}


1. Al guardar los cambios y cargar la página en el navegador, se puede notar que el contador comienza en **0**.



1. Si el contador va a empezar en un valor diferente, como **5**, simplemente se pasa ese valor como argumento a **useState**.

import { useState } from "react";

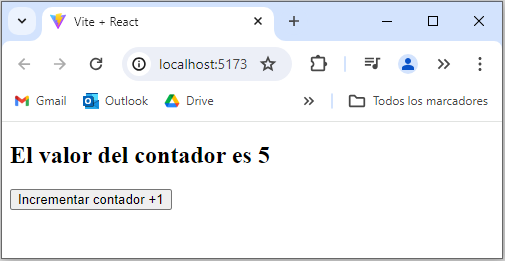
export const CounterApp = () => {

    const [counter, setCounter] = useState(5);
    //let counter = 0;

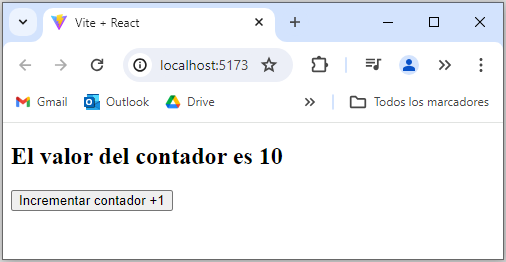
    const counterIncrement = () => {
        // counter = counter + 1
        setCounter(counter + 1)
        console.log("click " + counter);
    }

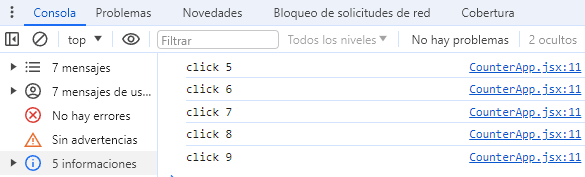
    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={() => {
            counterIncrement()
        }}>Incrementar contador +1</button>
    </>

}



1. Al hacer clic en el botón para incrementar el contador, se puede observar que el valor del contador aumenta correctamente en el navegador y se mantiene su estado. El hook **useState** se encarga de mantener el estado del componente y renderizar solo la parte relevante del HTML, lo que hace que la aplicación sea más eficiente.





**Nota:** Es importante tener en cuenta que el orden del código puede afectar la salida en la consola. Por ejemplo, en el caso de imprimir el valor del contador en la consola después de llamar a **setCounter**, es posible que se obtenga el valor anterior antes de que se incremente y se renderice. Esto se debe a que **setCounter** es una función asíncrona y el valor de counter no se actualiza inmediatamente después de llamarla.

Paso 9: Simplificar el código

1. En lugar de pasar directamente el valor al llamar a **setCounter**, se puede utilizar un callback o función de flecha en **setCounter**, el cual va a tomar el valor actual del estado y devolverá el nuevo valor. Esto garantiza que siempre se esté trabajando con el valor más actualizado del estado.

import { useState } from "react";

export const CounterApp = () => {

    const [counter, setCounter] = useState(5);
    //let counter = 0;

    const counterIncrement = () => {
        // counter = counter + 1
        setCounter(c => c + 1)
        console.log("click " + counter);
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={() => {
            counterIncrement()
        }}>Incrementar contador +1</button>
    </>

}


1. Procede a simplificar el código del botón para que llame directamente a la función **counterIncrement** eliminando los paréntesis y la flecha.

import { useState } from "react";

export const CounterApp = () => {

    const [counter, setCounter] = useState(5);
    //let counter = 0;

    const counterIncrement = () => {
        // counter = counter + 1
        setCounter(c => c + 1)
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={counterIncrement}>Incrementar contador +1</button>
    </>

}

import { useState } from "react";

export const CounterApp = () => {

    const [counter, setCounter] = useState(5);
    //let counter = 0;

    const counterIncrement = () => {
        // counter = counter + 1
        setCounter(c => c + 1)
    }

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={counterIncrement}>Incrementar contador +1</button>
    </>

}

1. Si ya no se necesita el mensaje de consola, se puede eliminar el **console.log** y reducir la función **counterIncrement** a una sola línea.

import { useState } from "react";

export const CounterApp = () => {

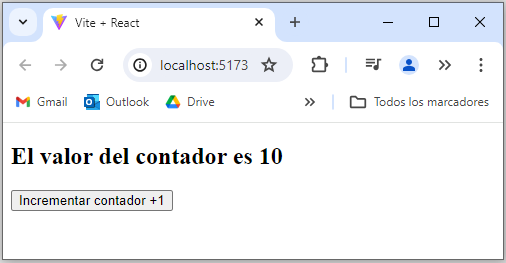
    const [counter, setCounter] = useState(5);
    //let counter = 0;

    const counterIncrement = () => setCounter(c => c + 1)

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={counterIncrement}>Incrementar contador +1</button>
    </>

}

1. Con estas optimizaciones, el código se vuelve más limpio y conciso, manteniendo la misma funcionalidad.



Paso 10: Pasar el valor por defecto desde el componente padre

1. Modifica la definición del componente **CounterApp** para que reciba el valor inicial como una propdesestructurada llamada **value**. Luego utiliza este valor en el estado inicial del hook **useState**.

import { useState } from "react";

export const CounterApp = ({value}) => {

    const [counter, setCounter] = useState(value);
    //let counter = 0;

    const counterIncrement = () => setCounter(c => c + 1)

    return <>
        <h2>El valor del contador es {counter}</h2>
        <button onClick={counterIncrement}>Incrementar contador +1</button>
    </>

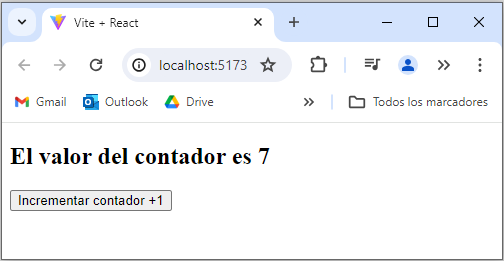
}


1. En el componente padre (**main.jsx**), asignale un valor a la prop **value** del componente **CounterApp**. Ten en cuenta que el valor se asignara al momento de renderizar el componente.

import React from 'react'
import ReactDOM from 'react-dom/client'
import { CounterApp } from './CounterApp.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
    <React.StrictMode>
        <CounterApp value={7} />
    </React.StrictMode>,
)


1. El valor inicial del contador se establece según el valor que se pasa desde el componente padre. En este caso, se establece en **7**.



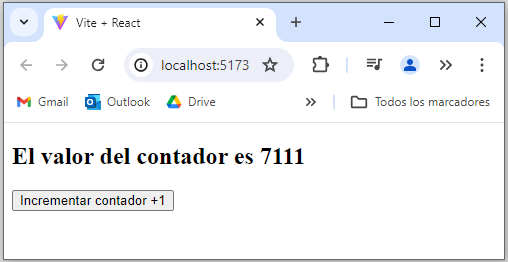
Paso 11: Concatenar varios "unos"

1. Si se proporciona el valor inicial como un String en lugar de un número, el contador empezará a concatenar "unos" en lugar de sumar números. Por ejemplo, al pasar el valor **"7"** como un String, el contador tratará el valor como una cadena y realizará la concatenación en lugar de la suma.

import React from 'react'
import ReactDOM from 'react-dom/client'
import { CounterApp } from './CounterApp.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
    <React.StrictMode>
        <CounterApp value="7" />
    </React.StrictMode>,
)


1. Al pulsar el botón, el contador va a concatenar números uno al valor existente en lugar de sumarlos.



Paso 12: Realizar la prueba final

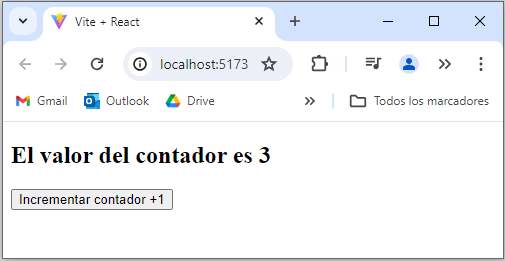
Para asegurar que el contador funcione correctamente y realice operaciones de suma en lugar de concatenación, procede a restaurar el valor inicial como un número en lugar de una cadena.

import React from 'react'
import ReactDOM from 'react-dom/client'
import { CounterApp } from './CounterApp.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
    <React.StrictMode>
        <CounterApp value={0} />
    </React.StrictMode>,
)


import React from 'react'
import ReactDOM from 'react-dom/client'
import { CounterApp } from './CounterApp.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
    <React.StrictMode>
        <CounterApp value={0} />
    </React.StrictMode>,
)

Hasta aquí se ha culminado el desarrollo de la aplicación **CounterApp**.