

React - Hook

Los hooks en React son funciones especiales que te permiten "enganchar" o conectar el estado de React y el ciclo de vida de los componentes funcionales. Antes de los hooks, los componentes funcionales no podían tener estado ni acceder al ciclo de vida de los componentes de manera directa. Los hooks cambiaron esto al proporcionar un conjunto de funciones, como `useState`, `useEffect`, `useContext`, entre otros, que permiten a los componentes funcionales tener estado interno, realizar efectos secundarios, suscribirse a contextos y más. Los hooks hacen que los componentes funcionales sean más potentes y reutilizables, y son una parte fundamental de la programación en React moderno.

React - Hook `{useState}`

`useState` Permite a los componentes funcionales tener un estado interno. Al llamar a `useState`, se devuelve una matriz con dos elementos: el primer elemento es el valor actual del estado y el segundo es una función que permite actualizar ese valor. Esta función se utiliza para modificar el estado y, cuando se actualiza, React vuelve a renderizar el componente con el nuevo valor del estado. El uso de `useState` simplifica la gestión del estado en los componentes funcionales, lo que los hace más simples y fáciles de entender en comparación con los componentes de clase. Además, `useState` es una de las características clave que permite a los desarrolladores escribir componentes más legibles y mantener la lógica de negocio separada de la interfaz de usuario en aplicaciones React.

Vamos a ver un ejemplo, crea un nuevo componente en tu proyecto de **React** llamado

`Contador`

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

const Contador = () => {

  //Usando useState
  const [number, setNumber] = useState(0)
  // Función para sumar
  const sumar = () => {
    setNumber(number + 1)
    console.log(number)
  }
  // Función para restar
  const restar = () => {
    setNumber(number - 1)
    console.log(number)
  }

  return(
    <div className="container bg-dark rounded text-light text-center">
      <h1>Contador</h1>
      <h2>{number}</h2>
      <button className="btn btn-warning" onClick={restar}>
        <i class="fa-solid fa-minus"></i>
      </button>
      <button className="btn btn-warning" onClick={sumar}>
        <i class="fa-solid fa-plus"></i>
      </button>

    </div>
  )
}


export default Contador
```

NOTA

Lo primero que hacemos es importar `{ useState }` y ahora solo creamos la función. Dentro de la funciones definimos una `const` y allí usamos a `useState`. Recordar que `useState` recibe dos parámetros, el primero la variable y el segundo una función. Adicional a ello, definimos dos funciones más, una para `sumar` y la otra para `restar`. Finalmente devolvemos la ejecución de esas funciones y estilizamos un poco el diseño con `Bootstrap` y agregamos algunos iconos.

Recuerda llamar a tu componente en tu archivo principal `App.js`

App.js

 Copiar código

```
import './App.css';
import Contador from './Componentes/Contador';
```

```
function App() {  
  return (  
    <div className="App">  
      <Contador />  
    </div>  
  );  
}  
  
export default App;
```

NOTA

Comprobando el ejercicio...

Contador

0



Contador

1



Contador

0




React - Hook {useEffect}

`useEffect` Permite realizar efectos secundarios en componentes funcionales. Se utiliza para ejecutar código adicional después de que el componente haya sido renderizado en el DOM. Puede ser utilizado para suscripciones a eventos, peticiones HTTP, actualizaciones de estado, entre otros. `useEffect` acepta una función como argumento, que se ejecutará después de cada renderizado del componente. También puede recibir un arreglo de dependencias como segundo argumento, lo que le permite controlar cuándo se debe ejecutar la función de efecto. Esto es especialmente útil para evitar ciclos de renderizado innecesarios o para limpiar efectos cuando el componente se desmonta. En resumen, `useEffect` es esencial para manejar lógica asíncrona y efectos secundarios en componentes funcionales de React.

Vamos a ver un ejemplo, crea un nuevo componente en tu proyecto de **React** llamado

`MiuseEffect`

Componentes/MiuseEffect.js

 Copiar código

```
const MiuseEffect = () =>{
  const [count, setCount] = useState(0)

  useEffect(() =>{
    document.title = `Ciclaste en ${count} `
  })

  return(
    <div className="container bg-dark mt-4 pt-4 rounded text-light text-center">
      <h1>Contador usando <code className="badge bg-secondary ">useEffect</code></h1>
      <h2> Clicaste en {count}</h2>
      <button className="btn btn-warning" onClick={() => setCount(count + 1)}>Clic aquí</button>
    </div>
  )
}

export default MiuseEffect
```


NOTA

Este **Hook** en el ejemplo hace exactamete lo mismo que el `{useState}` solo que añade una nueva funcionalidad, y es que al finalizar el `useState` y mostrar la variable `count` sumar 1 por cada clic, no solo mostrara el contenido en el `h2` donde se estableció el `count` inicialmente sino al rendererizar el DOM,

este modificara la etiqueta html `title` y mostrara lo que se ha definido en el `useEffect`. 25/04/2024 23:3-M2

Recuerda llamar a tu componente en tu archivo principal `App.js`

App.js

 Copiar código

```
import './App.css';
import MiuseEffect from './Componentes/MiuseEffect';

function App() {
  return (
    <div className="App">
      <MiuseEffect />
    </div>
  );
}

export default App;
```

NOTA

Comprobando el ejercicio...



Ciclaste en 0



localhost:3000

Contador usando `useEffect`

Clicaste en 0

Clic aquí



Ciclaste en 1



localhost:3000

Contador usando **useEffect**

Ciclaste en 1

Clic aquí

Segundo ejemplo de **useEffect**

Veamos otro ejemplo, crea un nuevo componente en tu proyecto de **React** llamado **Efecto**

js

Copiar código

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

const Efecto = () => {
  const [texto, setTexto] = useState('')

  const mostrarTexto = (e) => {
    setTexto(e.target.value)
    console.log(texto)
  }

  useEffect(() =>{
    console.log('Montado')
```

```

    return console.log('Desmontado')
  }, [])

  useEffect(() =>{
    console.log('Texto modificado')
  }, [texto])

  return(
    <div className="container bg-dark mt-4 pt-4 rounded text-light text-center">
      <h1>Ejemplo 2 -<code className='badge bg-secondary'>useEffect</code></h1>
      <input className='m-3' type="text" onChange={mostrarTexto}/>
      <p className='p-4'>{texto}</p>
    </div>
  )
}

export default Efecto


```

NOTA

Este código muestra un componente de React llamado Efecto, que utiliza el hook `useState` para manejar un estado llamado `texto`. Cuando se modifica un campo de entrada en la interfaz, se actualiza el estado `texto`. Además, emplea el hook `useEffect` para imprimir mensajes en la consola cuando el componente se monta y cuando el estado `texto` se modifica. El primer `useEffect` se ejecuta solo una vez al montar el componente y muestra un mensaje de "Montado", mientras que el segundo `useEffect` se activa cada vez que `texto` cambia y muestra un mensaje de "Texto modificado".

Recuerda llamar a tu componente en tu archivo principal `App.js`

App.js

 Copiar código

```

import './App.css';
import Efecto from './Componentes/Efecto';

function App() {
  return (
    <div className="App">
      <Efecto />
    </div>
  );
}

export default App;

```

NOTA

Comprobando el ejercicio...

Ejemplo 2 - useEffect



El contenido del `input` se actualiza en el `<p>`



Ejemplo 2 - useEffect

Ejemplo de useEffect