

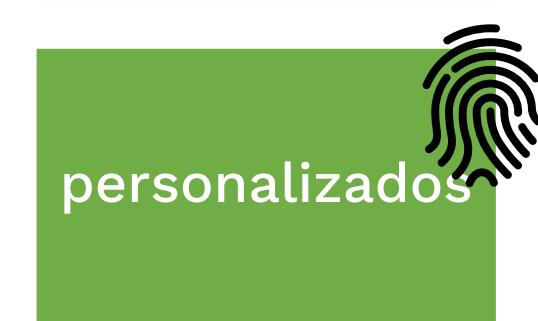
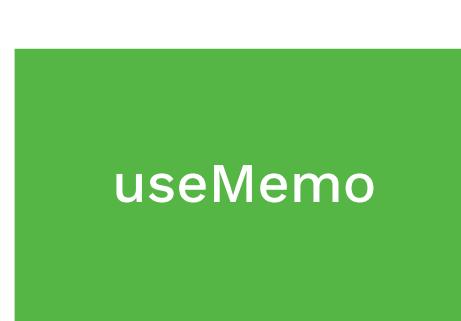
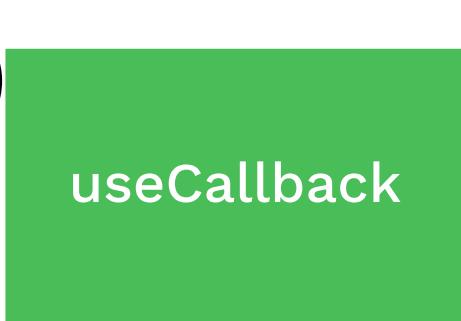


# Hooks en React

Centro de Servicios y Gestión Empresarial  
SENA Regional Antioquia

# Concepto - Hooks

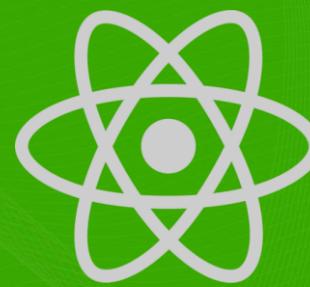
Hooks son funciones que permiten usar estado y otros aspectos de React sin escribir una clase. Los Hooks son un enlace que permiten enganchar los componentes funcionales a todas las características que ofrece React.



# Importancia - Hooks

Los hooks han transformado la manera en que se manejan el estado y los efectos secundarios en React, haciéndolos más intuitivos y fáciles de usar en componentes funcionales.

- Los componentes funcionales manejan su ciclo de vida a través de useEffect, en lugar de los métodos de ciclo de vida de clase.
- El Virtual DOM mejora el rendimiento al minimizar las actualizaciones directas al DOM real.
- Los hooks son funciones que permiten usar el estado y otros aspectos de React en componentes funcionales, facilitando el desarrollo y la reutilización del código.



# Tipos de Hooks

# 1. Hook - useState

Es el hook más básico y se usa para manejar el estado en componentes funcionales. Devuelve una variable de estado y una función para actualizarla.

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

// Sintaxis
const [state, setState] = useState(initialValue);
```

- **state:** La variable que representa el valor actual del estado.
- **setState:** La función que se usa para actualizar el valor de state.
- **initialValue:** El valor inicial del estado, puede ser un valor constante o una función que devuelva el valor inicial.

# Hook - useState

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

export function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);

  return (
    <div>
      <p>Valor: {count}</p>
      <button onClick={increment}>Aumentar</button>
    </div>
  );
}
```

## 2. Hook - useEffect

Permite realizar efectos secundarios en componentes, como hacer llamadas a una API, configurar un temporizador o suscribirse a eventos. Se ejecuta después de que el componente se renderiza.

```
import React, { useEffect } from 'react';
// Sintaxis
useEffect(() => {
  // Código a ejecutar después del renderizado
  return () => {
    // Código opcional de limpieza
  };
}, [dependencies]);
```

- **Array de dependencias ([dependencies]):** Controla cuándo se ejecuta el efecto. Si está vacío ([]), el efecto solo se ejecuta en el montaje y desmontaje. Si tiene dependencias, el efecto se ejecutará cada vez que alguna de esas dependencias cambie.

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

export function ItemList() {
  const [inputValue, setInputValue] = useState('');
  const [items, setItems] = useState([]);

  useEffect(() => {
    if (items.length > 0) {
      const lastItem = items[items.length - 1];
      alert(`Se agregó el ítem: ${lastItem}`);
    }
  }, [items]);

  const handleInputChange = (e) => { setInputValue(e.target.value); };

  const handleAddItem = () => {
    if (inputValue.trim() !== '') {
      setItems((prevItems) => [...prevItems, inputValue.trim()]);
      setInputValue('');
    }
  };

  return (...);
}
```

## Hook – useEffect

Parte 01

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

export function ItemList() {
  ...
  return (
    <div>
      <h2>Lista de Items</h2>
      <input
        type="text"
        value={inputValue}
        onChange={handleInputChange}
        placeholder="Escribe un item ..."/>
      &nbsp;
      <button onClick={handleAddItem}>Agregar</button>

      <ul>
        {items.map((item, index) => (<li key={index}>{item}</li>))}
      </ul>
    </div>
  );
}
```

## Hook – useEffect

Parte 02

### 3. Hook - useContext

Se utiliza para acceder a un contexto sin necesidad de recurrir a los componentes de proveedor (**Provider**) y consumidor (**Consumer**) de forma explícita.

Esta simplificación facilita el intercambio de datos a lo largo del árbol de componentes, permitiendo una gestión más eficiente del estado global de la aplicación.

```
import React, { createContext, useContext } from 'react';

// Sintaxis
const value = useContext(MyContext);
```

# Hook - useContext

Además del uso de **useContext**, es fundamental utilizar un **Provider**. Este componente especial se emplea junto con la API de Context para compartir datos (como temas, configuraciones o información de autenticación) entre componentes. Además, se evita la necesidad de pasar explícitamente los datos a través de las **props** en cada nivel de la jerarquía de componentes.

```
// Provider - Sintaxis
<MyContext.Provider value={/* valor a compartir */}>
  {/* componentes que pueden acceder al contexto */}
</MyContext.Provider>
```

# Hook - useContext

## Pasos a Seguir

- 1. Creación del Contexto:** Utiliza createContext para crear un nuevo contexto.
- 2. Uso del Provider:** Envuelve los componentes que necesitan acceder al contexto con el Provider y proporciona un valor que será accesible para ellos.
- 3. Consumo del Contexto:** Usa useContext dentro de los componentes hijos para acceder a los valores compartidos.

## Hook - useContext

```
import React, { createContext, useContext } from 'react';

export const ThemeContext = createContext('light');

export function DisplayTheme() {
  const theme = useContext(ThemeContext);
  const icon = theme === 'light' ? '☀️' : '🌙';
  return (
    <p>Tema seleccionado es: <strong>{theme} - {icon}</strong></p>
  );
}

export function App() {
  return (
    <div>
      <ThemeContext.Provider value="dark">
        <DisplayTheme />
      </ThemeContext.Provider>

      <DisplayTheme />
    </div>
  );
}
```

# Hook - useContext

## Ventajas de Usar un Provider

- **Simplificación del Paso de Datos:** Permite que cualquier componente acceda a ciertos datos sin tener que pasarlo manualmente a través de cada componente en el árbol, lo que mejora la eficiencia del código.
- **Reusabilidad:** Puedes cambiar el valor del contexto de manera dinámica. Los componentes que dependen de ese contexto se actualizarán automáticamente, lo que facilita la gestión del estado.
- **Mantenibilidad:** Ayuda a organizar y separar los datos de configuración o temas comunes, lo cual contribuye a que el código sea más legible y fácil de mantener. Además, esta estructura fomenta una mayor claridad en la lógica del negocio.

## 4. Hook - useReducer

Es una alternativa a **useState** cuando se necesita una lógica de estado más compleja o cuando el próximo estado depende del anterior. Es similar a reduce en JavaScript y funciona muy bien para manejar múltiples valores de estado.

```
import React, { useReducer } from 'react';
// Sintaxis
const [state, dispatch] = useReducer(reducer, initialState);
```

- **reducer:** Una función que recibe el estado actual y una acción, y devuelve el nuevo estado.
- **dispatch:** La función que dispara las acciones para modificar el estado.

## 4. Hook - useReducer

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

const initialState = { count: 0 };
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
}
```



```
export function CounterReducer() {
  const [state, dispatch] = useReducer(reducer, initialState);
  const increment = () => dispatch({ type: 'increment' });
  const decrement = () => dispatch({ type: 'decrement' });

  return (
    <div>
      <button onClick={increment}>+</button>
      <p> {state.count} </p>
      <button onClick={decrement}>-</button>
    </div>
  );
}
```

# 5. Hook - useRef

Crea una referencia mutable que persiste durante el ciclo de vida del componente.

Es útil para acceder directamente a elementos del DOM o para almacenar un valor que no necesita causar una nueva renderización cuando cambia.

```
// Sintaxis
import React, { useRef } from 'react';
const refContainer = useRef(initialValue);
```

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

export function FocusInput() {
  const inputRefEmail = useRef(null);
  const inputRefPass = useRef(null);
  const handleFocus = () => {
    inputRefEmail.current.value = '';
    inputRefPass.current.value = '';
    inputRefEmail.current.focus();
  };

  return (
    <div>
      <label>Usuario:</label>
      <input type="email" defaultValue="aperezn@sena.edu.co" ref={inputRefEmail} />
      <br />
      <label>Contraseña:</label>
      <input type="password" defaultValue="123456" ref={inputRefPass} />
      <button onClick={handleFocus}>Restablecer</button>
    </div>
  );
}
```

## 5. Hook - useRef



# G R A C I A S

Presentó: Alvaro Pérez Niño

Instructor Técnico

Correo: aperezn@misena.edu.co

<http://centrodesserviciosygestionempresarial.blogspot.com/>

Línea de atención al ciudadano: 01 8000 910270

Línea de atención al empresario: 01 8000 910682



[www.sena.edu.co](http://www.sena.edu.co)