**M2S15: React Hooks (useState, useEffect)**

**Hola, ¿cómo estás?, espero que te sientas muy bien. Antes de iniciar con este contenido me gustaría que revises el siguiente Podcast que te brindará unas recomendaciones para cuidar tu vista mientras trabajas en dispositivos eléctrónicos, si deseas escucharlo [presiona aquí](https://open.spotify.com/episode/2vbfVnZbfuaOL4Ik8PKzUD?si=xC-r_Hs1SLWi2lR22lSd5w&dl_branch=1)**

**Si no tienes mucho tiempo para escuchar el podcast no te preocupes, guárdalo y escúchalo cuando tengas tiempo, vayas en camino, o cuando hagas ejercicio, recuerda que no es obligatorio, solo quería compartirlo contigo.**

**Ahora bien sigamos adelante con el contenido de la semana...**

**ÍNDICE**

* [Hooks](https://github.com/U-Camp/BOOT-M2-SEM15#hooks)
* [useState](https://github.com/U-Camp/BOOT-M2-SEM15#usestate)
* [useEffect](https://github.com/U-Camp/BOOT-M2-SEM15#useeffect)

**Hooks**

Como se mencionó antes, los hooks son una nueva manera de implementar el estado y los métodos del ciclo de vida de un componente. Anteriormente los componentes funcionales eran completamente stateless, por lo que no tenían un estado asociado al mismo ni tenían acceso a los métodos del ciclo de vida del componente, sólo los componentes de clase tenían esas características, pero a partir de la versión 16.8 de React, se presentaron los hooks, los cuales proveen de estado y acceso al ciclo de vida a los componentes funcionales, aunque de una manera especial y distinta a como normalmente se hace con los componentes de clase.

Normalmente un hook se define con el prefijo use y el nombre del hook al que se hace referencia; los hooks siempre reciben un sólo parámetro con el valor inicial de la variable a enlazar y a su vez regresan un arreglo de dos posiciones, la primera posición (posición 0) es la variable que se está enlazando reactivamente en el hook y la segunda posición (posición 1) es un método que permite actualizar esa variable.

**useState**

Para manejar el estado se usa el hook useState, lo cual es similar a inicializar el estado con this.state en un componente de clase.

Puede ser usado múltiples veces, por lo que permite un "estado múltiple".

Observa el siguiente ejemplo:

import { useState } from 'react'

function Ejemplo(){

const [data, setData] = useState("palabra 1")

manejarCambios(){

return setData("palabra 2")

}

return(

<>

<p>{data}</p>

<button onClick={(e) => manejarCambios(e)}>

Cambiar palabra

</button>

</>

)

}

* Importa el hook useState
* Crearemos dos variables a partir de la desestructuración de arreglos:
  + data. Será una variable en un componente local.
  + setData. Será una función que afecte esa variable en ese componente local.
* Asignaremos los valores a través del hook useState. El argumento será el valor inicial.
* Generaremos una función que permitirá manejar los cambios de React. Dentro, aplicaremos la función setData. Cada vez que esa función se ejecute, cambiará el dato.
* En el retorno, mostraremos el dato y crearemos un botón que, al hacerse clic, me cambie el valor.

**useEffect**

Para controlar el ciclo de vida del componente, se usa el hook useEffect, el cual, como su nombre lo dice, permite enlazar un side effect que vaya a ocurrir en el componente y éste de cierta manera reemplaza los métodos del ciclo de vida componentDidMount, componentDidUpdate y componentWillUnmount, pero unificados en una sola API.

Cuando se llama el hook useEffect se le está indicando a React que ejecute la función de efecto después de haber renderizado el componente en el DOM (replicando de cierta manera el método componentDidMount), así mismo si el efecto regresa una función cualquiera, react ejecutará la función de retorno antes de actualizar el renderizado, como si fuera un componentWillUnmount y finalmente si se le pasa un arreglo de dependencias al efecto (esto quiere decir las variables/constantes que son accesibles dentro del efecto) el efecto sólo se ejecutará si ésta variable/constante cambia (replicando componentDidUpdate).

Todo esto permite organizar efectos secundarios en un componente según las partes que están relacionadas (como agregar o eliminar un dato), en lugar de forzar una división basada en métodos del ciclo de vida, por lo que al igual que cualquier otro hook se pueden usar tantos useEffect en un mismo componente como se requiera.

Los Hooks son funciones de Javascript, pero imponen dos reglas adicionales:

Sólo llamar hooks en el nivel superior. No es recomendable llamar hooks dentro de loops, condiciones o funciones anidadas.

Sólo llamar hooks desde componentes funcionales de React. No es recomendable llamar hooks desde las funciones regulares de Javascript.

Solo hay otro lugar válido para llamar hooks: los hooks personalizados, los cuales son hooks que el desarrollador mismo puede crear para compartir funcionalidad de estado entre componentes, esto se verá más adelante, así como los hooks de contexto (useContext) y el hook reducer (useReducer) los cuales son muy útiles bajo el patrón front-controller que se verá cuando se llegue al submódulo de donde se consuman APIs.

La manera en que ha evolucionado React y la dirección en la que el equipo de desarrollo quiere llevar la librería, es que eventualmente los componentes de clase ya no sean soportados y todo componente tenga que ser forzosamente un componente funcional y hacer uso exclusivo de los hooks para lograr la funcionalidad del componente, y aunque esa no es una meta a corto plazo ya que van a pasar al menos otras 2 versiones mayores antes de que deprequen los componentes de clase, es necesario repensar cómo se desarrolla en React. De tal manera que nuevas implementaciones sean desarrolladas bajo éste nuevo approach para evitar que las aplicaciones se rompan en un futuro.

const { useState, useEffect } = React;

// Componente funcional con hooks de estado y efecto

function Contador(props) {

// Arreglo con variable de estado - método de actualización

// Por convención el método debe llevar el prefijo `set`

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

// De forma similar a componentDidMount y componentDidUpdate

useEffect(() => {

// Actualiza el título del documento usando la API del navegador

document.title = `Has dado click en el botón ${contador} veces`;

});

// El botón usa el método `setContador` para actualizar el estado

return (

<div>

<h1>Hola, {props.nombre}</h1>

<p>Has dado click en el botón {contador} veces</p>

<button onClick={() => setContador(contador + 1)}>

Pícale aquí

</button>

</div>

);

}

ReactDOM.render(

<Contador \*nombre\*="Armando" />,

document.getElementById('root')

);

*Ejemplo de componente funcional con hooks*

**Hemos llegado al final del contenido de esta semana, recuerda que puedes hacer consultas a tus coaches o comunicarte con tus compañeros de clase.**

**Nos vemos en la siguiente clase. Saludos!!**