Módulo 4 – Hooks

# ¿Qué son?

Slide 2

ReactJS es la librería más usada para el desarrollo de aplicaciones web. Y esto lo ha logrado con la adopción de patrones, tecnologías claves para que el desarrollo sea lo más eficiente y versátil. Alguno de esos conceptos son los Hooks.

Previamente ReactJS se manejaba solo con clases. Cada clase tenía dos principales propiedades “props” y “state”. Estas permiten recibir valores desde un componente padre y manejar el estado de valores ene l componente respectivamente.

Sin embargo, como hemos visto, con los componentes funciones no podemos mantener el estado de variables declarándolas con “let” ya que estas se reinicializan en cada render con el valor que escribimos en código. Es por esto que nacieron los Hooks, su traducción es gancho, y es porque permiten “enganchar” la funcionalidad del estado, y como lo veremos más adelante, administrar el Ciclo de vida del componente en Componentes funcionales y muchas más funcionalidades y características que podemos encapsular en Hooks. También podemos crear los nuestros para ahorrarnos cierto trabajo y reutilizarlo en diferentes lugares.

# ¿Cómo se realizaba antes?

Slide 3 - 4

En los componentes Clase la administración del ciclo de vida y el manejo de estado se obtenía por herencia de la clase Component de ReactJS. Estas nos daban funciones tales como componentDidUpdate, componentWillUnmount, setState y demás que permitían la modificación del estado y administración del ciclo de vida del componente.

Visualizar comparativa con las capturas del código

# Hooks principales

Visitar la página  
<https://react-hooks-cheatsheet.com>

* useState, administración de estados, pueden ser primitivos, JSON, clases, etc.
* useEffect, admintración del life cycle del componente, lo veremos más adelante del curso. Dato, este siempre se ejecuta después de que el componente de haya renderizado por algún cambio.
* useContext, usado para obtener el estado global cuando usamos React Context como Global Store
* useLayoutEffect, se invoca muy similar que el useEffect, con la diferencia de que se invoca antes del render. Puede afectar al rendimiento según su implementación.
* useReducer, usado principalmente como alternativa del useState, para simplificar la invocación al querer actualizar el estado.
* useCallback, usado principalmente para evitar que funciones se redefinan en cada render. Limitando su re-definicion según una lista de dependencias. Así logramos que por ejemplo componentes que reciben estas funciones o valores se vuelvan a renderizar a pesar de que los valores que usan no hayan cambiado. Este se usa en funciones no en Componentes.
* useMemo, muy similar a useCallback, pero esta enfocado a memorizar VALORES, estos también pueden ser Componentes. Por lo que se puede usar para que un componente solo sea re-renderizado según la lista de dependencias que se declare.
* useRef, usado para almacenar valores en toda la vida de un componente, puede parecerse mucho a useState, pero este no retorna una función para actualizar el estado. Su uso principal es para guardar referencias de Componentes para así invicar funciones de ellos de manera programáticas.

# Usando Hooks

Crear un nuevo URL “users/form/:id?” para crear un formulario donde vamos a usar los Hooks básicos de React. Crearemos el formulario con los cambios

* Name, String.
* Age, Number.
* City, String.
* Verified, boolean.

Usaremos los Hooks:

* useReducer, para adminitrar el type “Form” que tendrá los datos del formulario.
* useLayout, para precargar los datos cuando se tiene un ID en el URL.
* useMemo, para hacer que un componente custom cambie solo cuando verified cambia.
* useRef, para hacer un focus manual a un input al entrar
* useEffect, para invocar la función de focus del valor de hook ref solo cuando la página se monta por primera vez.