Argentina programa 4.0

UNIVERSIDAD TECNOLOGICA NACIONAL

# Desarrollo en React JS

## Componentes parte 2 - HOOKS

***Desarrollador Web con React Js***

# Hooks

Argentina programa 4.0

Hooks son una nueva característica en React 16.8. Estos te permiten usar el estado y otras características de React sin escribir una clase.

**useState** es un hook en react, y el mismo lo utilizaremos para el manejo de estados.

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

function Example() {
  // Declara una nueva variable de estado, la cual llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

# Hooks de estados

Argentina programa 4.0

Dentro de un componente funcional llamamos al método useState y de esta forma agregamos un estado local al mismo. React mantendrá este estado entre re-renderizados.

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

function Example() {
  // Declara una nueva variable de estado, que llamaremos "count".
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Este ejemplo renderiza un contador.
Cuando haces click en el botón, incrementa el valor

**useState** devuelve un par: el valor de estado actual y una función que le permite actualizarlo.
Puedes llamar a esta función desde un controlador de eventos o desde otro lugar.
El único argumento para useState es el estado inicial. El argumento de estado inicial solo se usa durante el primer renderizado.

# Hooks de efectos

El Hook de efecto, **useEffect,** agrega la capacidad de realizar efectos secundarios desde un componente funcional. Tiene el mismo propósito que **componentDidMount**, **componentDidUpdate** y **componentWillUnmount** en las clases React, pero unificadas en una sola API.

Declaramos la variable de estado count y le indicamos a React que necesitamos usar un efecto. Le pasamos una función al Hook useEffect. Dentro de nuestro efecto actualizamos el título del documento usando la API del navegador document.title.
Cuando React renderiza nuestro componente, recordará este efecto y lo ejecutará después de actualizar el DOM. Esto sucede en cada renderizado, incluyendo el primero.

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

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

# useReducer

Argentina programa 4.0

Acepta un reducer de tipo (state, action) => newState y devuelve el estado actual emparejado con un método dispatch

**useReducer** a menudo es preferible a useState cuando se tiene una lógica compleja que involucra múltiples subvalores o cuando el próximo estado depende del anterior. useReducer además te permite optimizar el rendimiento para componentes que activan actualizaciones profundas, porque puedes pasar hacia abajo dispatch en lugar de callbacks.

```jsx
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:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

Argentina programa 4.0

En el ejemplo vemos un componente llamado **Counter.**

El mismo hace uso de **useReducer**

- en el primer parámetro se pasa la función **reducer**
- en el segundo parámetro el estado inicial (count=0) useReducer nos retorna un **state** y una función **dispatch**

Vemos que cada vez que hacemos click en el botón “**incrementar**” o “**decrementar**” se realiza el **dispacth** y se envía como parámetro el tipo de operación (es el action de la función **reducer** y se puede enviar cualquier objeto que queramos como parámetro)

En la función **reducer** tenemos la lógica para determinar qué acción tomar con el estado en base al action que tomamos como parámetro.

Esta parte es totalmente programable.

# Algunas consideraciones extra

Argentina programa 4.0

## ¿Tengo que nombrar mis Hooks personalizados comenzando con ”use“?

Por favor, hazlo. Esta convención es muy importante. Sin esta, no podríamos comprobar automáticamente violaciones de las reglas de los Hooks porque no podríamos decir si una cierta función contiene llamados a Hooks dentro de la misma.

Argentina programa 4.0

# ¿Dos componentes usando el mismo Hook comparten estado?

No. Los Hooks personalizados son un mecanismo para reutilizar lógica de estado (como configurar una suscripción y recordar el valor actual), pero cada vez que usas un Hook personalizado, todo estado y efecto dentro de este son aislados completamente.

# Gracias!