
ICARO
Nuestro compromiso es con el futuro.

React

Clase 4

React: Hooks

Conceptos básicos:

- **Re-renderizado**
- **Características**
- **Reglas**
- **Estado**
- **Efectos**

React

Hooks: Re-renderizado

Hooks: Re-renderizado

Re-renderizado:

- *En React llamamos “re-renderizado” a la carga completa de un componente.*

Hooks: Re-renderizado

Re-renderizado:

- *En React llamamos “re-renderizado” a la carga completa de un componente.*

- *Al generarse esta recarga, todos los métodos que están dentro del componente son ejecutados.*

Hooks: Re-renderizado

Re-renderizado:

- *En React llamamos “re-renderizado” a la carga completa de un componente.*
- *Al generarse esta recarga, todos los métodos que están dentro del componente son ejecutados.*
- *Estas recargas son generadas por algún evento que dispara el usuario de nuestra UI.*

Hooks: Re-renderizado

Algunas formas de activar el re-renderizado:

- Cambio en las props desde el componente padre.

Hooks: Re-renderizado

Algunas formas de activar el re-renderizado:

- Cambio en las props desde el componente padre.

- Seteo de un nuevo estado con la función “seteadora” del hook useState().

Hooks: Re-renderizado

Algunas formas de activar el re-renderizado:

- Cambio en las props desde el componente padre.
- Seteo de un nuevo estado con la función “seteadora” del hook useState().
- Invocar la acción de algún manejador de estado global como puede ser “redux” o “context”.

Hooks: Re-renderizado

Existen otras formas de “disparar” o “triguerear” el re-renderizado, pero son poco utilizadas hoy en día.

Hooks: Re-renderizado

Existen otras formas de “disparar” o “triguerear” el re-renderizado, pero son poco utilizadas hoy en día.

Para investigar más sobre ese tema se puede buscar información sobre “forceUpdate()”, “componentDidUpdate()” o “shouldComponentUpdate()”.

Hooks: Re-renderizado

Existen otras formas de “disparar” o “triguerear” el re-renderizado, pero son poco utilizadas hoy en día.

Para investigar más sobre ese tema se puede buscar información sobre “forceUpdate()”, “componentDidUpdate()” o “shouldComponentUpdate()”.

*Son métodos un poco antiguos y de poco uso en la actualidad.

React

Hooks: Características

Hooks: Características

Algunas características de los hooks…

Hooks: Características

Algunas características de los hooks…

- Manejo de estados (useState)

Hooks: Características

Algunas características de los hooks…

- Manejo de estados (useState)
- Sincronizan efectos secundarios (useEffect)

Hooks: Características

Algunas características de los hooks…

- Manejo de estados (useState)
- Sincronizan efectos secundarios (useEffect)
- No son únicos de React

Hooks: Características

Algunas características de los hooks…

- Manejo de estados (useState)
- Sincronizan efectos secundarios (useEffect)
- No son únicos de React
- Sólo sirven en **componentes funcionales**

Hooks: Características

Algunas características de los hooks…

- Manejo de estados (useState)
- Sincronizan efectos secundarios (useEffect)
- No son únicos de React
- Sólo sirven en **componentes funcionales**
- Inspiración en el paradigma funcional

Hooks: Características

Algunas características de los hooks…

- Manejo de estados (useState)
- Sincronizan efectos secundarios (useEffect)
- No son únicos de React
- Sólo sirven en **componentes funcionales**
- Inspiración en el paradigma funcional
- Inspiración en el patrón modular

Hooks: Características

Mentalidad…

La palabra “hook” en inglés significa “gancho”, “anzuelo”, “enganchar”, etc…

Hooks: Características

Mentalidad…

La palabra “hook” en inglés significa “gancho”, “anzuelo”, “enganchar”, etc…

Una forma útil para pensar los “hooks” en React sería:

“Son anzuelos que ‘pican’ cada vez que la función .render() es llamada”

Hooks: Características

Mentalidad…

La palabra “hook” en inglés significa “gancho”, “anzuelo”, “enganchar”, etc…

Una forma útil para pensar los “hooks” en React sería:

“Son anzuelos que ‘pican’ cada vez que la función .render() es llamada”

* La ejecución de la función ‘.render()’ se conoce coloquialmente como ‘re-renderizado’
* El término ‘pican’ alude a la acción de un pez cuando muerde un anzuelo.

Hooks: Características

Algunas Consideraciones…

Hooks: Características

Algunas Consideraciones…

- *En cada “re-renderizado” del componente se ejecutan todos los hooks de ese mismo componente.*

Hooks: Características

Algunas Consideraciones…

- *En cada “re-renderizado” del componente se ejecutan todos los hooks de ese mismo componente.*

- *Los hooks respetan el orden en el que son llamados dentro de la función.*

React

Hooks: Reglas

Hooks: Reglas

- *Las reglas nos permiten prevenir errores y al mismo tiempo explotar por completo las funcionalidades de la librería de React.*

Hooks: Reglas

- *Las reglas nos permiten prevenir errores y al mismo tiempo explotar por completo las funcionalidades de la librería de React.*

- *Por otro lado las convenciones ayudan a la universalidad sobre “la forma de hacer las cosas en React”.*

Hooks: Reglas

Algunas pautas que veremos…

- Regla de los hooks
- Nombramiento
- Convenciones

Hooks: Reglas

Regla de los hooks:

- *El orden en el que son creados los hooks es de vital importancia para el funcionamiento de React.*

Hooks: Reglas

Regla de los hooks:

- *El orden en el que son creados los hooks es de vital importancia para el funcionamiento de React.*

- *Está totalmente prohibido su uso dentro de condicionales u otra forma que altere el orden de llamado de los mismos.*

Hooks: Reglas

Regla de los hooks:

```
const Alumno = props ⇒ {
 if (props.nombre) {
        const [nombre, setNombre] = useState("")
    }…
```

Hooks: Reglas

Regla de los hooks:

¡Prohibido!

```
const Alumno = props ⇒ {
 if (props.nombre) { // ← ¡No!
        const [nombre, setNombre] = useState("")
    }…
```

Hooks: Reglas

Regla de los hooks:

- *Sólo pueden utilizarse en componentes funcionales.*

Hooks: Reglas

Regla de los hooks:

- *Sólo pueden utilizarse en componentes funcionales.*
- *Los hooks requieren de una “recarga” o “re-renderizado” para funcionar.*

Hooks: Reglas

Regla de los hooks:

```
import { useState } from "react"

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

  return "No soy un componente, soy un simple string"
}
```

Hooks: Reglas

Regla de los hooks:

¡Prohibido!

```
import { useState } from "react"

const contador = () ⇒ {  // ← Es una función de JS que no retorna un componente.
  const [count, setCount] = useState(0)

  return "No soy un componente, soy un simple string"
}
```

Hooks: Reglas

Regla de los hooks:

¡Prohibido!

```
import { useState } from "react"

const contador = () ⇒ {
  const [count, setCount] = useState(0)

  return "No soy un componente, soy un simple string" // ← Retorna un string.
}
```

Hooks: Reglas

Regla de los hooks:

- *Muchas veces nos podemos ver tentados de usar hooks en funciones que no son componentes de React.*

Hooks: Reglas

Regla de los hooks:

- *Muchas veces nos podemos ver tentados de usar hooks en funciones que no son componentes de React.*

- *Si bien las funciones pueden ser llamadas igual, no obtendremos un resultado positivo en ese caso.*

Hooks: Reglas

Algunas convenciones de los hooks…

Hooks como “useState” tiene su propia forma para nombrar la función actualizadora.

Hooks: Reglas

Algunas convenciones de los hooks…

Hooks como “useState” tiene su propia forma para nombrar la función actualizadora.

Ejemplo: setXxxx

```
const Alumno = props ⇒ {
    const [nombre, setNombre] = useState("")
```

Hooks: Reglas

Nombramiento de los hooks:

Todos los nombres de los hoooks son escritos en camelCase y deben comenzar con la palabra “use”:

Hooks: Reglas

Nombramiento de los hooks:

Todos los nombres de los hoooks son escritos en camelCase y deben comenzar con la palabra “use”:

Ejemplo: useXxxx

Hooks: Reglas

Nombramiento de los hooks:

Todos los nombres de los hoooks son escritos en camelCase y deben comenzar con la palabra “use”:

Ejemplo: useXxxx

** Esto aplica también para los custom hooks*

Hooks: Reglas

Algunas convenciones de los hooks…

Otra convención es la utilización de la desestructuración en “useState”.

Hooks: Reglas

Algunas convenciones de los hooks…

Otra convención es la utilización de la desestructuración en “useState”.

Ejemplo: [] = useState(“”)

```
const Alumno = props ⇒ {
    const [nombre, setNombre] = useState("")
```

React

Hooks: Estado

Hooks: Estado

Algunas ideas sobre el estado…

Hooks: Estado

Algunas ideas sobre el estado…

- Es un concepto universal de la programación.

Hooks: Estado

Algunas ideas sobre el estado…

- Es un concepto universal de la programación.
- Permite personalizar la experiencia de usuario.

Hooks: Estado

Algunas ideas sobre el estado…

- Es un concepto universal de la programación.
- Permite personalizar la experiencia de usuario.
- Las “malas prácticas” dificultan su mantenibilidad.

Hooks: Estado

Algunas ideas sobre el estado…

- Es un concepto universal de la programación.
- Permite personalizar la experiencia de usuario.
- Las “malas prácticas” dificultan su mantenibilidad.
- Cada paradigma lo trata de manera diferente.

Hooks: Estado

Definición de estado:

Hooks: Estado

Definición de estado:

- *En React el “estado” es una estructura de datos actualizable que es utilizada para contener información dentro de los componentes.*

Hooks: Estado

Definición de estado:

- *En React el “estado” es una estructura de datos actualizable que es utilizada para contener información dentro de los componentes.*

- *El estado en un componente puede cambiar a lo largo del tiempo.*

Hooks: Estado

Algunas Consideraciones…

Hooks: Estado

Algunas Consideraciones…

- *En cada “re-renderizado” los hooks apuntan al mismo estado.*

Hooks: Estado

Algunas Consideraciones…

- *En cada “re-renderizado” los hooks apuntan al mismo estado.*

- *A menos que se conserve una referencia, el estado de un hook se cambia únicamente a través de los métodos que el mismo trae.*

Hooks: Estado

Algunas Consideraciones…

- *En cada “re-renderizado” los hooks apuntan al mismo estado.*

- *A menos que se conserve una referencia, el estado de un hook se cambia únicamente a través de los métodos que el mismo trae.*

* Uniendo estos puntos podemos afirmar que es seguro mutar un objeto y actualizar el estado sólo a través del método provisto por el hook.

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useState”.*

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useState”.*

- *Los hooks de “useContext” y los de librerías como redux o zustand, son parecidos a useState.*

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useState”.*

- *Los hooks de “useContext” y los de librerías como redux o zustand, son parecidos a useState.*

* También podemos mencionar a “useRef” en esta categoría, aunque su comportamiento difiere en que no “re-renderiza”, ni tiene función “seteadora”.

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState } from "react"
2.
3.  const Counter = props => {
4.      const [count, setCount] = useState(0)
5.
6.      return (
7.          <div>
8.              <p>El contador actual es: {count}</p>
9.
10.             <button onClick={() => setCount(count + 1)}>+ 1</button>
11.
12.             <button onClick={() => setCount(count - 1)}>- 1</button>
13.
14.             <button onClick={() => setCount(0)}>Reset</button>
15.         </div>
16.     )
17. }
```

Hooks: Estado

Ejemplo con useState:

```jsx
import React, { useState } from "react" // Importamos el hook desde la librería de react.

const Counter = props => {
    const [count, setCount] = useState(0)

    return (
        <div>
            <p>El contador actual es: {count}</p>

            <button onClick={() => setCount(count + 1)}>+ 1</button>

            <button onClick={() => setCount(count - 1)}>- 1</button>

            <button onClick={() => setCount(0)}>Reset</button>
        </div>
    )
}
```

Hooks: Estado

Ejemplo con useState:

```jsx
import React, { useState } from "react"

const Counter = props => {
    const [count, setCount] = useState(0) // Creamos el valor del contador con el valor inicial 0 y su respectivo setter.

    return (
        <div>
            <p>El contador actual es: {count}</p>

            <button onClick={() => setCount(count + 1)}>+ 1</button>

            <button onClick={() => setCount(count - 1)}>- 1</button>

            <button onClick={() => setCount(0)}>Reset</button>
        </div>
    )
}
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState } from "react"
2.
3.  const Counter = props => {
4.      const [count, setCount] = useState(0)
5.
6.      return (
7.          <div> // Consumimos el valor del contador en la interfaz de usuario luego de cada re-renderizado.
8.              <p>El contador actual es: {count}</p>
9.
10.             <button onClick={() => setCount(count + 1)}>+ 1</button>
11.
12.             <button onClick={() => setCount(count - 1)}>- 1</button>
13.
14.             <button onClick={() => setCount(0)}>Reset</button>
15.         </div>
16.     )
17. }
```

Hooks: Estado

Ejemplo con useState:

```jsx
import React, { useState } from "react"

const Counter = props => {
    const [count, setCount] = useState(0)

    return (
        <div>
            <p>El contador actual es: {count}</p>
            // Utilizamos el valor actual y sumamos el valor 1, al utilizar "setCount" se dispara el re-renderizado.
            <button onClick={() => setCount(count + 1)}>+ 1</button>

            <button onClick={() => setCount(count - 1)}>- 1</button>

            <button onClick={() => setCount(0)}>Reset</button>
        </div>
    )
}
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState } from "react"
2.
3.  const Counter = props => {
4.      const [count, setCount] = useState(0)
5.
6.      return (
7.          <div>
8.              <p>El contador actual es: {count}</p>
9.
10.             <button onClick={() => setCount(count + 1)}>+ 1</button>
11.             // Utilizamos el valor actual y restamos el valor 1, al utilizar "setCount" se dispara el re-renderizado.
12.             <button onClick={() => setCount(count - 1)}>- 1</button>
13.
14.             <button onClick={() => setCount(0)}>Reset</button>
15.         </div>
16.     )
17. }
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState } from "react"
2.
3.  const Counter = props => {
4.      const [count, setCount] = useState(0)
5.
6.      return (
7.          <div>
8.              <p>El contador actual es: {count}</p>
9.
10.             <button onClick={() => setCount(count + 1)}>+ 1</button>
11.
12.             <button onClick={() => setCount(count - 1)}>- 1</button>
13.             // Seteamos el valor a 0, al utilizar "setCount" se dispara el re-renderizado.
14.             <button onClick={() => setCount(0)}>Reset</button>
15.         </div>
16.     )
17. }
```

React

Hooks: Efectos

Hooks: Efectos

Algunas ideas sobre efectos…

Hooks: Efectos

Algunas ideas sobre efectos…

- Muchas veces se le llama “efectos secundarios”.

Hooks: Efectos

Algunas ideas sobre efectos…

- Muchas veces se le llama “efectos secundarios”.
- Controla los estados a medida que cambian.

Hooks: Efectos

Algunas ideas sobre efectos…

- Muchas veces se le llama “efectos secundarios”.
- Controla los estados a medida que cambian.
- Existe una similitud con los “eventos”.

Hooks: Efectos

Algunas ideas sobre efectos…

- Muchas veces se le llama “efectos secundarios”.
- Controla los estados a medida que cambian.
- Existe una similitud con los “eventos”.
- Su principal rol deber ser sincronizar asincronías.

Hooks: Efectos

Algunas ideas sobre efectos…

- Muchas veces se le llama “efectos secundarios”.
- Controla los estados a medida que cambian.
- Existe una similitud con los “eventos”.
- Su principal rol deber ser sincronizar asincronías.
- Proviene del paradigma de la programación funcional.

Hooks: Estado

Definición de efectos:

Hooks: Estado

Definición de efectos:

- *En React el concepto de “efectos” se referiere a las transformaciones de estado que recibe nuestra aplicación de forma no programada.*

Hooks: Estado

Definición de efectos:

- *En React el concepto de “efectos” se referiere a las transformaciones de estado que recibe nuestra aplicación de forma no programada.*

- *Una forma de llamar a esta problemática puede ser el término: “manejo de efectos”.*

Hooks: Estado

Ejemplos de efectos:

Hooks: Estado

Ejemplos de efectos:

- *Un usuario modifica un campo de nuestra UI.*

Hooks: Estado

Ejemplos de efectos:

- *Un usuario modifica un campo de nuestra UI.*
- *Una API nos devuelve un response.*

Hooks: Estado

Ejemplos de efectos:

- *Un usuario modifica un campo de nuestra UI.*
- *Una API nos devuelve un response.*

* Estos son los ejemplos más comunes en el día a día de “manejo de efectos”.

Hooks: Estado

Algunas Consideraciones…

Hooks: Estado

Algunas Consideraciones…

- *Existe una similitud muy fuerte entre el “manejo de efectos” y “manejo de eventos”.*

Hooks: Estado

Algunas Consideraciones…

- *Existe una similitud muy fuerte entre el “manejo de efectos” y “manejo de eventos”.*
- *Hoy en día se prefiere trabajar con “eventos” en la mayoría de los casos, y los efectos reservarlos para la sincronización únicamente.*

Hooks: Estado

Algunas Consideraciones…

- *Existe una similitud muy fuerte entre el “manejo de efectos” y “manejo de eventos”.*

- *Hoy en día se prefiere trabajar con “eventos” en la mayoría de los casos, y los efectos reservarlos para la sincronización únicamente.*

* En este último caso nos referimos puntualmente al hook “useEffect”.

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useEffect”.*

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useEffect”.*
- *Los hooks de “useCallback”, “useMemo” y los de librerías como ReactQuery, son parecidos a useEffect.*

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useEffect”.*

- *Los hooks de “useCallback”, “useMemo” y los de librerías como ReactQuery, son parecidos a useEffect.*

*ReactQuery es una librería de manejo de llamados a APIs.

Hooks: Estado

- *El ejemplo más normal de ver es la utilización de “useEffect”.*

- *Los hooks de “useCallback”, “useMemo” y los de librerías como ReactQuery, son parecidos a useEffect.*

*ReactQuery es una librería de manejo de llamados a APIs.

*ReactQuery implementa por debajo “useEffect” también existen otras librerías como redux-toolkit-query.

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react"
2.  const Counter = props => {
3.      const [count, setCount] = useState(0)
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json())
10.                 .then(response => setMessage(response.message))
11.                 .catch(() => setMessage("Hubo un error en el llamado."))
12.         }
13.     }, [count])
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react" // Importamos el hook de useEffect.
2.  const Counter = props => {
3.      const [count, setCount] = useState(0)
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json())
10.                 .then(response => setMessage(response.message))
11.                 .catch(() => setMessage("Hubo un error en el llamado."))
12.         }
13.     }, [count])
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react"
2.  const Counter = props => {
3.      const [count, setCount] = useState(0) // Estos useState son los que gobiernan el estado de la UI.
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json())
10.                 .then(response => setMessage(response.message))
11.                 .catch(() => setMessage("Hubo un error en el llamado."))
12.         }
13.     }, [count])
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Hooks: Estado

Ejemplo con useState:

```jsx
import React, { useState, useEffect } from "react"
const Counter = props => {
    const [count, setCount] = useState(0)
    const [message, setMessage] = useState("")
    const MAX_COUNT = 10
    useEffect(() => { // En el primer parámetro del useEffect definimos el callback que va a ser ejecutado cada vez que cambie el array de dependencias.
        if (count === MAX_COUNT) {
            fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
                .then(data => data.json())
                .then(response => setMessage(response.message))
                .catch(() => setMessage("Hubo un error en el llamado."))
        }
    }, [count])
    return (
        <div>
            <p> El contador actual es: {count} {message} </p>
            <button onClick={() => setCount(count + 1)}>+ 1</button>
        </div>
    )
}
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react"
2.  const Counter = props => {
3.      const [count, setCount] = useState(0)
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json()) // Fetch es una promesa, por lo tanto su utilización está más que juistificada para se considerada un efecto.
10.                 .then(response => setMessage(response.message))
11.                 .catch(() => setMessage("Hubo un error en el llamado."))
12.         }
13.     }, [count])
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react"
2.  const Counter = props => {
3.      const [count, setCount] = useState(0)
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json())
10.                 .then(response => setMessage(response.message)) // Acá “seteamos” nuestra UI en el caso exitoso.
11.                 .catch(() => setMessage("Hubo un error en el llamado."))
12.         }
13.     }, [count])
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react"
2.  const Counter = props => {
3.      const [count, setCount] = useState(0)
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json())
10.                 .then(response => setMessage(response.message))
11.                 .catch(() => setMessage("Hubo un error en el llamado.")) // Acá “seteamos” nuestra UI cuando el llamado a la API falla.
12.         }
13.     }, [count])
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Hooks: Estado

Ejemplo con useState:

```
1.  import React, { useState, useEffect } from "react"
2.  const Counter = props => {
3.      const [count, setCount] = useState(0)
4.      const [message, setMessage] = useState("")
5.      const MAX_COUNT = 10
6.      useEffect(() => {
7.          if (count === MAX_COUNT) {
8.              fetch(`https://soy-el-ejemplo.de.una/api/que/se-dispara/?al-llegar-a${MAX_COUNT}`)
9.                  .then(data => data.json())
10.                 .then(response => setMessage(response.message))
11.                 .catch(() => setMessage("Hubo un error en el llamado."))
12.         }
13.     }, [count]) // Este es el array de dependencias, cada vez que cambie alguno de estos estados, se ejecutará el callback del primer argumento de useEffect.
14.     return (
15.         <div>
16.             <p> El contador actual es: {count} {message} </p>
17.             <button onClick={() => setCount(count + 1)}>+ 1</button>
18.         </div>
19.     )
20. }
```

Entorno Productivo

Si es que hay tiempo para ir al código…

Entorno Productivo

Si es que hay tiempo para ir al código…

Traten de “jugar” con los ejemplos dados.

Entorno Productivo

Si es que hay tiempo para ir al código…

Traten de "jugar" con los ejemplos dados.

Son ejemplos bastante "potentes" y estándar dentro de la industria.

¡Vamos al código!

Clase 4: Hooks

¡Muchas gracias!

ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar