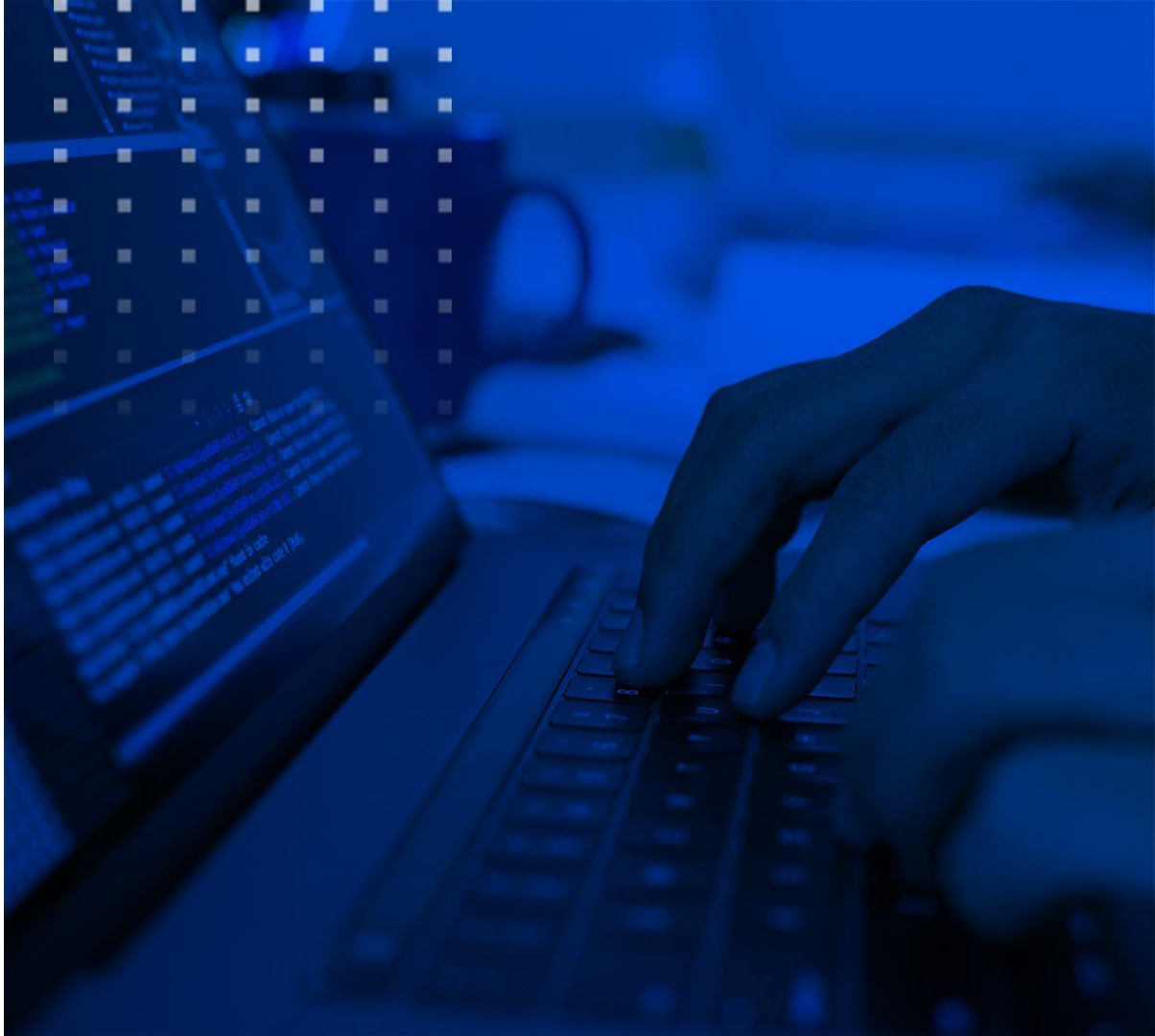
incluyeme.com

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

CURSO

FULL STACK DEVELOPER
NIVEL INICIAL



UNIDAD 21
JavaScript
Hooks

ESTADO Y CICLO DE VIDA

COMPONENTES
Estados

Los componentes **sin estado** no guardan ninguna información y por ello no necesitan de datos locales. Un componente así se podría usar importandolo y luego colocando la etiqueta correspondiente, como venimos haciendo.

COMPONENTES
Estados

Los componentes **con estado** son aquellos que almacenan datos de manera local al componente. Estos datos pueden variar a lo largo del tiempo bajo diversas circunstancias, por ejemplo por la interacción del usuario con el componente.

COMPONENTES
Estados

Los estados nos permiten **controlar los valores y datos** que maneja un componente. Permiten que este sea capaz de **cambiar la interfaz sin renderizar todo el DOM**. Podemos decir que es una manera de tener variables que podemos cambiar de manera dinámica.

COMPONENTES
Estados

Un ejemplo podría ser un componente que tenga un contador o que se **conecte** a una **API Rest**. A este componente le podemos pasar como propiedad la URL del API y una serie de parámetros para realizar la solicitud al servidor. Una vez que recibamos los datos lo común será almacenarlos como estado del componente, para usarlos como sea necesario.

COMPONENTES
Estados

Entonces, el estado es **similar a las props**, pero es **privado** y está controlado por el componente.

COMPONENTES
Estados

El componente podrá además **reaccionar al cambio de estado**, de modo que **actualice su vista** cuando sea necesario.

COMPONENTES
Ciclos de vida

El ciclo de vida no es más que una **serie de estados** por los cuales pasa todo componente a lo largo de su existencia. Esos estados tienen correspondencia en diversos métodos, que nosotros podemos implementar para realizar acciones cuando se van produciendo.

COMPONENTES
Ciclos de vida

Estas son las tres etapas de los estados dentro de un ciclo de vida del componente:

- **Montaje**: se produce la primera vez que un componente va a generarse, incluyéndose en el DOM.
- **Actualización**: se produce cuando el componente ya generado se está actualizando.
- **Desmontaje:** se produce cuando el componente se elimina del DOM.

COMPONENTES
Hooks

Para trabajar con estados de los componentes basados en funciones vamos a usar dos **Hooks**: **useEffect** y **useState**

HOOKS

HOOKS

Los hooks (enganches) son **funciones** que nos permiten **conectarnos** al **estado** de React y a las características del **ciclo de vida** de los componentes de la función.

Los hooks **no funcionan dentro de las clases**.

HOOKS

Repasando, los **estados** en React nos permiten **controlar los valores y datos que maneja un componente**.
Permiten que este sea capaz de cambiar la interfaz sin renderizar todo el DOM. Podemos decir que es una manera de tener variables que podemos cambiar de manera dinámica.

USESTATE

USESTATE

UseState es un **Hook** que permite agregar el **estado** React a los componentes de la función.

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

USESTATE

Es similar a una variable: después de importarlo dentro de nuestro archivo, **podemos asignarle un valor**.

```
const [nombre, setNombre] = useState('Hermione');
```

```
<p>{nombre}</p>
```

USESTATE

Tenemos un **valor** y un **set**. Con el set podemos actualizar el valor de estado de nuestro componente cuando lo queramos.

```
const [nombre, setNombre] = useState('Hermione');
```

```
<p>{nombre}</p>
```

En este momento el valor de nombre es Hermione

¡A Practicar!

Usando **useState** crear un botón en el componente contacto que al hacerle click muestre tu información de contacto.

USEEFFECT

USEEFFECT

Este hook recibe como parámetro una **función que se ejecutará cada vez que nuestro componente se renderice**, ya sea por un **cambio de estado**, por recibir **props nuevas** o porque es la primera vez que se monta.

USEEFFECT

Para usar este hook, primero debemos importarlo desde la librería de React:

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

USEEFFECT

Para realizar el cambio de estado de un componente con el set necesitamos de **un evento que active o llame la función**.

```
<p>{nombre}</p>
<Button onClick={() => setNombre('Harry')}>Cambiar nombre</Button>
```

Ahora el valor de **nombre** es **Harry**

USEEFFECT

Ahora añadimos un efecto que **se ejecutará cada vez que nuestro componente se renderice**. Para eso, ejecutaremos el método **useEffect** dentro de nuestra función y le pasamos **como parámetro la función** que queremos que ejecute al renderizar el componente.

USEEFFECT

Esto hace que se muestre en consola el mensaje *Hola!* después que el componente se renderice por primera vez.

```
useEffect(() =>
  console.log('Hola!')
);
```

USESTATE Y USEEFFECT

USEEFFECT

Cada vez que hacemos click en el componente. Cuando el state cambia, esto dispara un nuevo renderizado y, al renderizarse de nuevo, se vuelve a ejecutar la función que le hemos pasado a useEffect.

USESTATE Y USEEFFECT

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

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

  useEffect(() => {
    document.title = `Has hecho clic ${count} veces`
  })

  return (
    <div>
      <span>El contador está a {count}</span>
      <button onClick={() => setCount(count + 1)}>Incrementar contador</button>
    </div>
  )
}
```

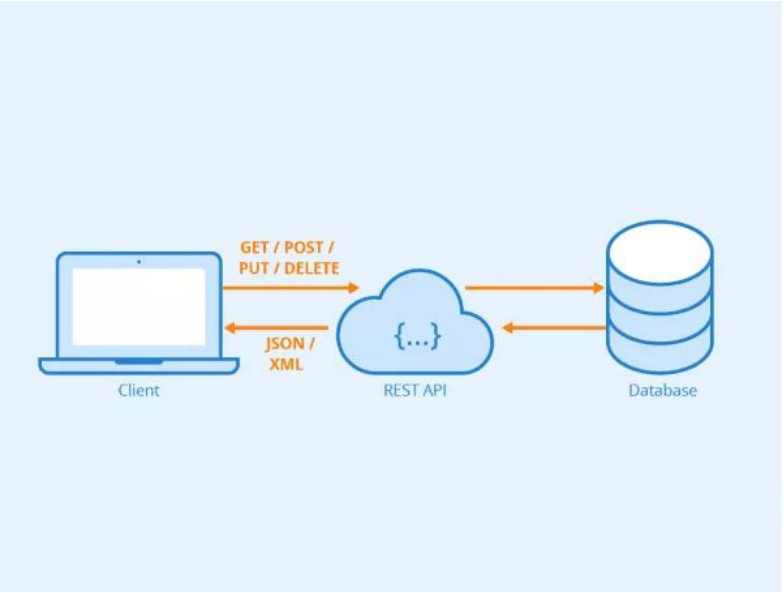
USEEFFECT

En el ejemplo, el título se actualiza:

- ★ Apenas ingresamos a la página: ya que se ejecuta useEffect al montarse nuestro componente.
- ★ Cada vez que hacemos click en el componente: cuando el state cambia, esto dispara un nuevo renderizado y se vuelve a ejecutar la función que le hemos pasado a useEffect.

CONSUMIR APIS

EJEMPLOS



- ★ Harry Potter API
- ★ Star Wars API
- ★ JSON Placeholder
- ★ Pokeapi
- ★ Ghibli Api

CONSUMIR APIS

Podemos hacer un **request** de manera simple utilizando **Fetch**.

```
fetch('https://hp-api.herokuapp.com/api/characters')
    .then(response =>
        response.json()
    )
    .then(personajes =>
        setPersonajes(personajes)
    );
```

CONSUMIR APIS

Y como queremos que se ejecute al montarse el componente, lo vamos a incluir dentro del **useEffect**

```
useEffect(() => {
    fetch('https://hp-api.herokuapp.com/api/characters')
        .then(response =>
            response.json()
        )
        .then(personajes =>
            setPersonajes(personajes)
        );
})
```

CONSUMIR APIS

Ahora, ¿Cómo usamos esto?

```
useEffect(() => {
    fetch('https://hp-api.herokuapp.com/api/characters')
        .then(response =>
            response.json()
        )
        .then(personajes =>
            setPersonajes(personajes)
        );
})
```

CONSUMIR APIS

Con un estado

```
const [personajes, setPersonajes] = useState([]);

useEffect(() => {
    fetch('https://hp-api.herokuapp.com/api/characters')
        .then(response =>
            response.json()
        )
        .then(personajes =>
            setPersonajes(personajes)
        );
})
```

CONSUMIR APIS

Vamos a iniciar el estado de personajes como un array vacío y cuando lleguen los datos los vamos a asignar a personajes.

```
const [personajes, setPersonajes] = useState([]);

useEffect(() => {
    fetch('https://hp-api.herokuapp.com/api/characters')
        .then(response =>
            response.json()
        )
        .then(personajes =>
            setPersonajes(personajes)
        );
})
```

MAP

MAP

Ahora, cómo **incluimos** éstos personajes en el **JSX**?

```
const [personajes, setPersonajes] = useState([]);

useEffect(() => {
    fetch('https://hp-api.herokuapp.com/api/characters')
        .then(response =>
            response.json()
        )
        .then(personajes =>
            setPersonajes(personajes)
        );
})
```

MAP

Sí, ¡vamos a usar el **map** que conocemos!

```
return(
    <div className="home">
        {personajes.map(personaje =>
            <p>{personaje.name}</p>
        )}
    </div>
);
```

MAP

Sí, ¡vamos a usar el **map** que conocemos!

```
return(
    <div className="home">
        {personajes.map(personaje =>
            <p>{personaje.name}</p>
        )}
    </div>
);
```

MAP

Idealmente hay que incluir en cada elemento la **propiedad key**. Esto ayudará a react a optimizar el renderizado ante cambios en el array.

```
return(
    <div className="home">
        {personajes.map((personaje, index) =>
            <p key={index}>{personaje.name}</p>
        )}
    </div>
);
```

MAP

De no tenerla podemos auto-generarla con el index provisto por el segundo parámetro de map.

```
return(
    <div className="home">
        {personajes.map((personaje, index) =>
            <p key={index}>{personaje.name}</p>
        )}
    </div>
);
```

¡A Practicar!


Utilizá la API que quieras y añadí a un componente una llamada utilizando useEffect, useState y map.
Utilizar el componente de Bootstrap Card:
https://react-bootstrap.github.io/components/cards/


[i]ncluyeme
.com


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES