ALCYON
IT Servicios

PARTE VI: useEFfect y Fetch API

ALCYON
IT Servicios

INTRODUCCIÓN

En este documento veremos:

- Función useEffect
- Comunicarnos con un servicio externo con API FETCH

ALCYON
IT Servicios

INTRODUCCIÓN

Creación del proyecto

Crearemos un nuevo proyecto desde el cmd o el terminal de VSC:

create-react-app api_fetch

Preparar el entorno

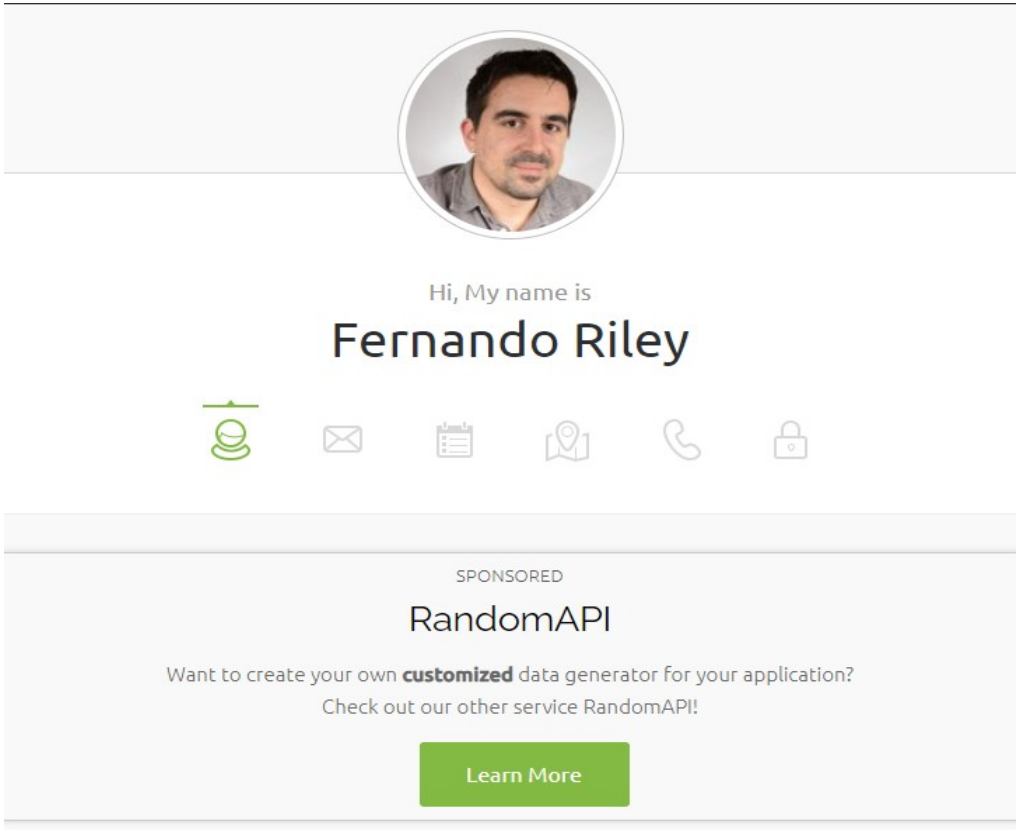
Vamos a editar el fichero **src/App.css** y borraremos el contenido

Editamos el fichero **src/App.js** y vaciamos el contenido del return de la función App()

INTRODUCCIÓN

API Externa

En los ejemplos en vez de utilizar un array interno como fuente de información, utilizaremos la API externa https:/randomuser.me



ALCYON
IT Servicios

INTRODUCCIÓN

API Externa

Si en la url del navegador introducimos https://randomuser.me/api/ veremos un JSON con multitud de datos de un usuario aleatorio


```
{"results":[{"gender":"female","name":{"title":"Miss","first":"Frida","last":"Olivares"},"location":{"stree
{"latitude":"23.2232","longitude":"32.2700"},"timezone":{"offset":"+4:00","description":"Abu Dhabi, Muscat,
73219c53a1dc","username":"tinyladybug272","password":"shelly","salt":"58OvQpYf","md5":"697417fbc8eed9cdcd73
{"date":"1955-01-15T03:07:16.221Z","age":68},"registered":{"date":"2004-07-30T03:36:07.617Z","age":19},"pho
{"large":"https://randomuser.me/api/portraits/women/59.jpg","medium":"https://randomuser.me/api/portraits/m
{"seed":"4d307015ee92827e","results":1,"page":1,"version":"1.4"}}
```


Si, desde la chrome web store instalamos esta herramienta podremos ver los datos sin compactar

ALCYON
IT Servicios

FETCH API

Vamos a acceder a la API comentada antes para traernos el nombre del usuario (atributo name) y su foto (atributo picture) y para ello utilizaremos la API Fetch de JavaScript

Componente principal App.js

Nos crearemos primero una constante para guardar la url de la api a utilizar:

```
const api = "https://randomuser.me/api";
```

Utilizaremos ahora la librería **fetch api** de JS para comunicar con el servicio anterior.

```
const peticion = fetch(api)
```

NOTA: la instrucción fetch tiene dos parámetros: el primero hace referencia a la url donde queremos enviar la petición y el segundo serian los datos de la petición. Puesto que haremos una petición de tipo GET el segundo parámetro no es necesario

ALCYON
IT Servicios

FETCH API

Con fetch realizamos una petición asíncrona al servidor y que puede llevar más o menos tiempo pero que, en cualquier caso y al ser asíncrona, el resto de instrucciones que tengamos a continuación se ejecutarán siempre antes de que el servidor (la api que estamos consumiendo) nos devuelva la respuesta.

Para recoger la respuesta tendremos que considerar dos escenarios:

- El servidor ha atendido la petición y nos devuelve el JSON con el usuario
- Ha ocurrido algún error que ha impedido que el servidor nos devuelva el usuario

Por tanto, tenemos que considerar las dos respuestas de la siguiente forma:

```
peticion
.then()     → respuesta correcta
.catch()    → respuesta con error (url no existe, etc...)
```

ALCYON
IT Servicios

FETCH API

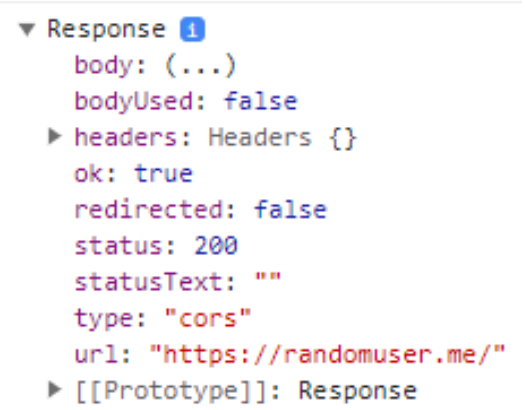
¿Qué recibe como argumento el método .then()?

Pues una función **callback**, es decir, una función que se va a ejecutar cuando se reciba la respuesta de servidor y que, como hemos comentado antes, puede ser que sea casi inmediata o demorarse en el tiempo (en función de la complejidad de la tarea a realizar o, incluso, del ancho de banda de la conexión)

.then((resp) => {console.log(resp)})

Vemos en la consola un objeto de respuesta en donde nos indica que la petición ha ido bien (status: 200 y ok: true), pero no vemos todavía el objeto con los datos del usuario

NOTA: Veréis que muestra dos respuestas (esto lo arreglaremos más adelante)



ALCYON
IT Servicios

FETCH API

¿Qué recibe como argumento el método .catch()?

Pues también una función callback que se va a ejecutar cuando se reciba una respuesta errónea del servidor

.catch((error) => {console.log(error)})

Por ejemplo, si escribimos mal la url de la api:

```
error: TypeError: Failed to fetch                                    Api
    at App (App.js:6:1)
    at renderWithHooks (react-dom.development.js:16305:1)
    at mountIndeterminateComponent (react-dom.development.js:20074:1)
    at beginWork (react-dom.development.js:21587:1)
    at beginWork$1 (react-dom.development.js:27426:1)
    at performUnitOfWork (react-dom.development.js:26557:1)
    at workLoopSync (react-dom.development.js:26466:1)
    at renderRootSync (react-dom.development.js:26434:1)
    at performConcurrentWorkOnRoot (react-dom.development.js:25738:1)
    at workLoop (scheduler.development.js:266:1)
```

ALCYON
IT Servicios

FETCH API

¿Cómo acceder a los datos del usuario si la petición ha ido bien?

Para acceder a los datos necesitamos un segundo .then() que recoja la respuesta recibida en el then anterior y nos extraiga los datos de ésta en el formato que necesitemos. Veamos el ejemplo:

```
peticion
.then((resp) => resp.json())
.then((usuario) => console.log(usuario))
.catch((error) => {console.log(error)})
```

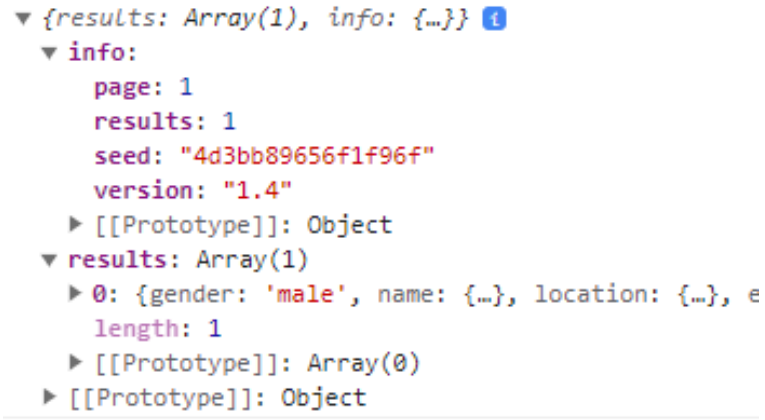
Vemos que en el primer then indicamos, en vez del console.log, que esperamos recibir una respuesta del servidor en formato json y que la propia **api fetch** de javascript se encargará de transformar automáticamente a formato objeto JS que recogeremos en el segundo then dentro del argumento usuario (que también es una función **callback**)

ALCYON
IT Servicios

FETCH API

¿Cómo acceder a los datos del usuario si la petición ha ido bien?

Si miramos la consola del segundo then veremos un objeto con los datos del usuario que nos ha enviado la api



NOTA: La Api Fetch admite tres tipos de respuestas:

resp.json() si la respuesta del servidor es un JSON

resp.text() si la respuesta del servidor es un texto

resp.blob() si la respuesta del servidor es un objeto binario (por ejemplo el mapa de bits de una imagen)

Los datos que necesitaríamos son:

usuario.results[0].name.first
usuario.results[0].name.last
usuario.results[0].picture.large

ALCYON
IT Servicios

FETCH API

Antes de mostrar en pantalla la ficha con los datos del usuario, vamos a entender un poco más que es el concepto de asincronía:

Vamos a añadir un console.log() inmediantemente después de la instrucción fetch

```
peticion
.then((resp) => resp.json())
.then((usuario) => console.log(usuario))
.catch((error) => {console.log(error)})
```

```
me ejecuto antes que el resto
▶ {results: Array(1), info: {…}}
```

console.log('me ejecuto antes que el resto')

Vemos como, efectivamente, el nuevo console.log se ha ejecutado antes del console.log con los datos del usuario. ¿Por qué, si esta situado después?.

Pues porque fetch es una función asíncrona y lo que hace JS es ejecutar todas las funciones no asíncronas del código antes de ejecutar fetch

ALCYON
IT Servicios

FETCH API

Y también veremos ahora como evitar que nuestra llamada a la api y, en general, el código que tenemos en el componente se ejecute dos veces.

Si miramos la consola veremos:

```
me ejecuto antes que el resto
me ejecuto antes que el resto
▶ {results: Array(1), info: {…}}
▶ {results: Array(1), info: {…}}
```

Tanto la llamada a la api como el console.log del mensaje se han ejecutado dos veces

¿Por qué aparece dos veces?

Pues porque **React** renderiza el componente cuando se inicia la aplicación y una segunda vez cuando el componente ya está montado (cuando se crea un componente y cuando el estado del componente se actualiza)

ALCYON
IT Servicios

USE EFFECT

Para evitar el doble renderizado utilizaremos otro hook de React llamado useEffect() que utilizaremos, en este caso, para evitar que se vuelva a renderizar el elemento cuando cambie su estado (todavía no tenemos estados en este ejercicio)

Primero tenemos que importar el hook

```
import {useEffect} from 'react';
```

En segundo lugar colocamos todo el código de la api fetch dentro de la función anónima que utiliza useEffect

```
useEffect(() => {
    const api = "https://randomuser.me/api";
    const peticion = fetch(api)
    … /...
})
```

ALCYON
IT Servicios

USE EFFECT

Si probamos la aplicación vemos que continua ejecutándose dos veces y esto es porque hemos obviado un segundo parámetro de la función useEffect que sirve para indicar cuando se tiene que ejecutar el código contenido en él.

Si solo queremos que se ejecute una vez indicamos como segundo parámetro un array vacio [ ]:

```
useEffect(() => {
    const api = "https://randomuser.me/api";
    const peticion = fetch(api)
    … /...
}, [ ])
```

NOTA: Más adelante veremos que podemos hacer con este segundo parámetro que ahora dejamos como array vacío

ALCYON
IT Servicios

USE EFFECT

Probamos la aplicación y ¡oh sorpresa! Vuelve a ejecutarse dos veces tal como vemos en la consola:

```
▶ {results: Array(1), info: {…}}
▶ {results: Array(1), info: {…}}
```

¿Por qué?

Pues porque React, por defecto (y esto depende de la versión que estemos utilizando) renderiza siempre un componente dos veces cuando estamos ejecutándolo en entorno de desarrollo.

Teóricamente lo hace así para facilitar la depuración de código ya que, recordad, React trabaja siempre con el virtual DOM y, después, traslada los cambios al DOM real. De esta forma podemos comprobar si lo que tenemos en el virtual DOM coincide con lo que vemos en el Dom real

ALCYON
IT Servicios

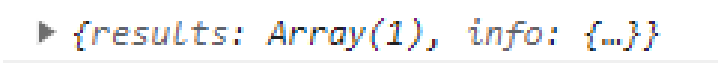
USE EFFECT

Para evitar el doble renderizado, incluso en el entorno de desarrollo, tendremos que editar el fichero **src/index.js** y eliminar la ejecución en modo estricto de React, simplemente eliminado las dos líneas de código JSX que se muestran a continuación:

```
const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Si ahora probamos nuestra aplicación veremos que solo se ejecuta una vez:



ALCYON
IT Servicios

USE EFFECT

Una vez solucionado el problemo (como decía Terminator) vamos a mostrar en pantalla los datos del usuario que nos entrega la API (en cada ejecución nos entrega un usuario aleatorio)

Para ello vamos a definir dos constantes para el nombre y la imagen del usuario

```
const [nombre, setNombre] = useState();
const [imagen, setImagen] = useState();
```

NOTA: Recordad importar el hook useState:

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

ALCYON
IT Servicios

USE EFFECT

Y en el segundo then donde recogemos el objeto usuario que nos entrega la api informamos las dos variables:

```
.then((usuario) => {
    setNombre(`${usuario.results[0].name.first} ${usuario.results[0].name.last}`)
    setImagen(usuario.results[0].picture.large)
})
```

Que informamos dentro del documento JSX

```
<div className="App">
    <h3>Empleado del mes:</h3>
    <p>Nombre: {nombre}</p>
    <img src={imagen} alt={nombre}/>
</div>
```

Empleado del mes:

Nombre: Melike Sadıklar

ALCYON
IT Servicios

USE EFFECT

Vamos a ver un poco más sobre este segundo parámetro que utilizamos en useEffect y que estamos informando como un array vacio:

```
peticion
    .then((resp) => resp.json())
    .then((usuario) => { … })
    .catch((error) => {…)
}, [ ])
```

La verdadera utilidad de este parámetro es incorporar una o más variables para que, en caso que alguna de ellas cambie su valor (su estado) se vuelva a ejecutar la función (la petición fetch a la api)

Probad a incorporar la variable nombre:

```
useEffect(() => { … peticion.then(...).then(...).catch(...)}, [nombre])
```

USE EFFECT

¿Qué está pasando aquí?. ¿React ha perdido la cabeza?

Empleado del mes:

Nombre: Séléna Masson

Empleado del mes:

Nombre: Chad Reed

Empleado del mes:

Nombre: Frederik Sørensen

....

Empleado del mes:

Nombre: Lucila Farias

y así hasta ∞

Pues no, React está haciendo lo que le hemos indicado, es decir, ejecutar la función fetch cada vez que cambia el contenido de la variable nombre.

Recordad que la API que estamos utilizando nos devuelve un usuario aleatorio de forma que, cada vez que la ejecutamos, está cambiando el valor de la variable nombre y, por tanto, se vuelve a ejecutar una y otra vez. Obviamente esta API es solo para probar y rara vez veremos este comportamiento en producción

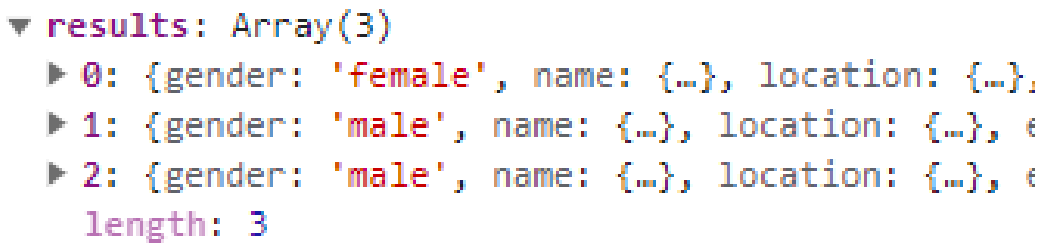
ALCYON
IT Servicios

USE EFFECT

Vamos a introducir una modificación en el comportamiento de nuestra aplicación para que, en vez de un empleado, muestre tres empleados del mes.

Para ello la API que estamos utilizando tiene una opción para que nos devuelva un número determinado de empleados:

```
const api = "https://randomuser.me/api/?results=3";
```



ALCYON
IT Servicios

USE EFFECT

En cuanto recibamos un array de más de un objeto ya no tiene sentido guardar en forma de variables los datos de nombre e imagen de un usuario y, en su lugar, crearemos una variable para guardar todos los datos de cualquier usuario (en nuestro caso 3 usuarios)

const [usuarios, setUsuarios] = useState([]);
~~const [nombre, setNombre] = useState();~~
~~const [imagen, setImagen] = useState();~~

Cuando recogemos la respuesta de la api informamos nuestra variable usuarios:

```
.then((usuarios) => {
    ~~setNombre(`${usuario.results[0].name.first} ${usuario.results[0].name.last}`)~~
    ~~setImagen(usuario.results[0].picture.large)~~
    setUsuarios(usuarios.results)
  })
```

ALCYON
IT Servicios

USE EFFECT

Y modificamos el código JSX para recorrer con map nuestra variable usuarios (que será un array con tres objetos) y mostrar los datos de cada uno de los usuarios obtenidos:

```
<div className="App">
    <h3>Empleados del mes:</h3>
    <p>Nombre: {nombre}</p>
    <img src={imagen} alt={nombre}/>
    {usuarios.map((usuario) =>
      <div key={usuario.email}>
        <p>Nombre: {usuario.name.first} {usuario.name.last}</p>
        <img src={usuario.picture.large} alt={usuario.name.first}/>
      </div>
    )}
</div>
```

NOTA: Recordad que necesitamos una clave única por iteración (hemos escogido el email)

ALCYON
IT Servicios

USE EFFECT

SI probamos la aplicación deberíamos ver los tres usuarios entregados por la API

Empleados del mes:

Nombre: Lummetje Van de Burgt

Nombre: Lauri Haapala

Nombre: Per De Veth

ALCYON
IT Servicios

ACTIVIDAD 6

ALCYON
IT Servicios

ACTIVIDAD

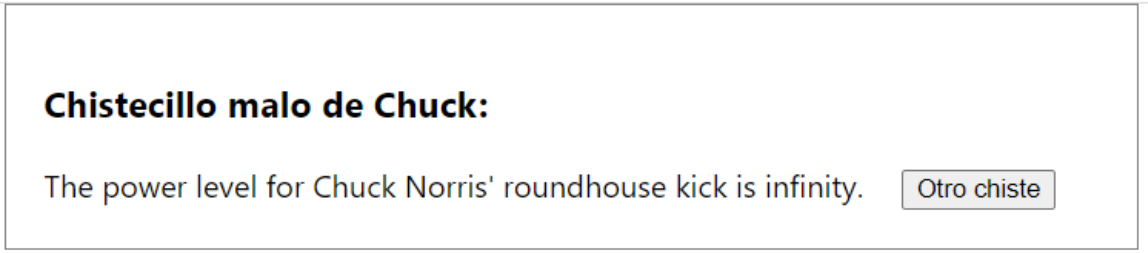
Actividad 6.1

Vamos a realizar una sencilla actividad para mostrar en pantalla un chiste aleatorio de Chuck Norris obtenido de la API:

https://api.chucknorris.io/jokes/random


```
{
    "categories": [],
    "created_at": "2020-01-05 13:42:25.099703",
    "icon_url": "https://assets.chucknorris.host/img/avatar/chuck-norris.png",
    "id": "Cpl6pEBzTuSqMPAuiprOcA",
    "updated_at": "2020-01-05 13:42:25.099703",
    "url": "https://api.chucknorris.io/jokes/Cpl6pEBzTuSqMPAuiprOcA",
    "value": "Chuck Norris doesn't use a phone to communicate his messages to a distant person, he just shouts it over."
}
```


Y, cuando pulsemos un botón, se mostrará otro diferente



ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.1

Componente principal App.js

Primero importaremos los hooks que necesitaremos y que serán el useEffect que hemos visto antes y el useState para la variable que utilizaremos para guardar el chiste:

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

En esta actividad tenemos que mostrar un chiste solo entrar en la página y, cada vez que pulsemos el botón de '*Otro chiste*' tendremos que volver a realizar una petición a la API para mostrar otro distinto

Por lo tanto lo que haremos es confeccionar una función que se encargue de realizar la llamada a la API:

```
const otroChiste = () => { … }
```

Y también crearemos la variable donde guardaremos el chiste:

```
const [chiste, setChiste] = useState();
```

ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.1

Componente principal App.js

Y dentro de esta función informaremos la url de la API a utilizar, realizaremos la petición asíncrona y recogemos el objeto que nos devuelve la API con el chiste aleatorio:

```
const otroChiste = () => {
    const api = "https://api.chucknorris.io/jokes/random";
    const peticion = fetch(api)
    peticion
    .then((resp) => resp.json())
    .then((chiste) => { setChiste(chiste.value) })
    .catch((error) => {console.log(error)})
}
```

NOTA: Podemos ver como con la respuesta de la API informamos la variable que hemos creado antes para guardar el chiste

ACTIVIDAD

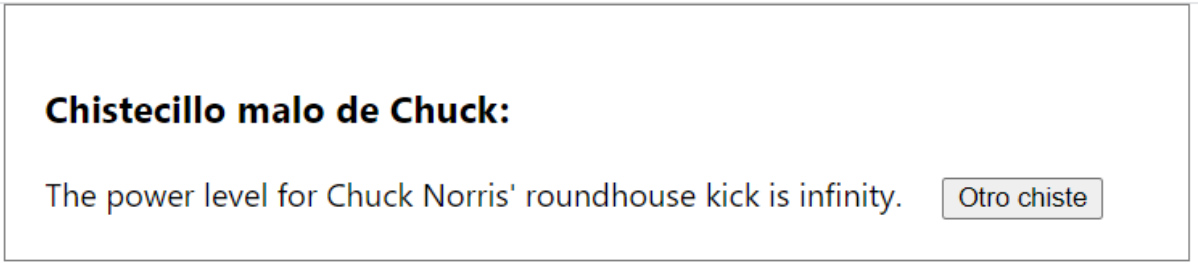
Actividad 6.1

Componente principal App.js

Por último, informamos en el documento JSX el chiste consultado junto con el botón al que le asociamos el evento que ejecutará de nuevo la función que hemos creado con la llamada a la API

```
<div className="App">
    <h3>Chistecillo malo de Chuck:</h3>
    <span>{chiste}</span>
    <button onClick={otroChiste}>Otro chiste</button>
</div>
```

Si pulsamos el botón veremos



ACTIVIDAD

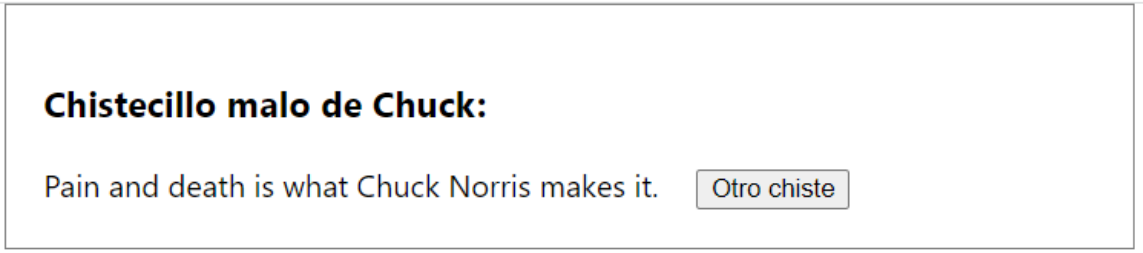
Actividad 6.1

Componente principal App.js

Pero si entramos a la aplicación por primera vez veremos que no se muestra ningún chiste:

Y esto es porque no estamos ejecutando la función otroChiste cuando se carga la aplicación. Para que se ejecute la función al cargar el componente necesitamos el hook useEffect:

```
useEffect(() => {
    otroChiste()
}, [])
```



ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.1

Componente principal App.css

Si queremos asociar un poco de css:

```
.App {
    width: 50%;margin:auto;padding: 20px;border: 1px solid grey;
}

button {margin-left: 20px;}
```

ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.2

Vamos a realizar otra sencilla actividad para mostrar en pantalla todos los chistes de Chuck Norris que contengan la palabra *fire* obtenidos de la API:

https://api.chucknorris.io/jokes/search?query=fire


```
{
  "total": 103,
  "result": [
    {
      "categories": [],
      "created_at": "2020-01-05 13:42:18.823766",
      "icon_url": "https://assets.chucknorris.host/img/avatar/chuck-norris.png",
      "id": "oBanalmvRZeyddEvcwl22Q",
      "updated_at": "2020-01-05 13:42:18.823766",
      "url": "https://api.chucknorris.io/jokes/oBanalmvRZeyddEvcwl22Q",
      "value": "Chuck Norris once put a forest fire out by spitting on it."
    },
    {
      "categories": [],
      "created_at": "2020-01-05 13:42:18.823766",
      "icon_url": "https://assets.chucknorris.host/img/avatar/chuck-norris.png",
      "id": "_HyMF0b6Rty9-5SXnnYG1Q",
      "updated_at": "2020-01-05 13:42:18.823766",
      "url": "https://api.chucknorris.io/jokes/_HyMF0b6Rty9-5SXnnYG1Q",
      "value": "Chuck Norris can cook fire."
    },
```


Chistecillos malos con FIRE:

1 - Chuck Norris once put a forest fire out by spitting on it.

2 - Chuck Norris can cook fire.

3 - Chuck Norris can eat fire and air.

ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.2

Componente principal App.js

Primero importaremos los hooks que necesitaremos y que serán el useEffect que hemos visto antes y el useState para la variable que utilizaremos para guardar los chistes:

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

Confeccionaremos una función que se encargue de realizar la llamada a la API de consulta de todos los chistes:

```
const chistesFire = () => { … }
```

Y también crearemos la variable donde guardaremos los chistes:

```
const [chistes, setChistes] = useState([]);
```

NOTA: inicializamos la variable como un array vacio

ACTIVIDAD

Actividad 6.2

Componente principal App.js

Dentro de esta función informaremos la url de la API a utilizar, realizaremos la petición asíncrona y recogemos el array de objetos que nos devuelve la API con los chistes solicitados:

```
const otroChiste = () => {
    const api = "https://api.chucknorris.io/jokes/search?query=fire";
    const peticion = fetch(api)
    peticion
    .then((resp) => resp.json())
    .then((chistes) => { setChistes(chistes.result) })
    .catch((error) => {console.log(error)})
}
```

NOTA: Podemos ver como con la respuesta de la API informamos la variable que hemos creado antes para guardar el chiste a partir del atributo result que nos devuelve la API

ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.2

Componente principal App.js

Por último, informamos en el documento JSX los chistes consultados utilizando la función map

```
<h3>Chistecillos malos con FIRE:</h3>
{chistes.map((chiste, index) =>
    <p key={chiste.id}>{index+1} - {chiste.value}</p>
)}
```

NOTA: Recordad informar un valor único para cada etiqueta <p>

Y puesto que los chistes se tienen que mostrar solo cargar la aplicación utilizaremos el hook useEffect

```
useEffect(() => {
    otroChiste();
    chistesFire();
}, [])
```

ALCYON
IT Servicios

ACTIVIDAD

Actividad 6.2

Componente principal App.js

Si probamos la aplicación deberíamos ver toda la lista de chistes a cada cual peor que el anterior

Chistecillos malos con FIRE:

1 - Chuck Norris once put a forest fire out by spitting on it.

2 - Chuck Norris can cook fire.

3 - Chuck Norris can eat fire and air.

4 - Chuck Norris once whent to pizzahut. He ordered an eggroll with some swiss cheese. A rotten cow corpse.A dead president.And a large box with Hellfire missiles. Pizza hut served it on a big tuna fish pizza. Chuck Norris Roundhousekicked the whole place to oblivion making everyone inside suffer terrible pain for eternity. Chuck Norris did not order any pizza.

Aviso Legal

David Alcolea
david-alcolea@alcyon-it.com
www.alcyon-it.com