React: Formularios

Vamos a poner un input en el que poder escribir algo. Pero queremos que lo que se muestre en esa caja de texto no sea un valor fijo, si no que sea algo que podamos modificar como, por ejemplo, el estado.

Importamos **useState** y declaramos la variable de estado:

```
App.js
let [nombre, setNombre] = useState('Peio');
```

En el return del componente, ponemos un **p**árrafo con el valor de la variable de estado:

```
App.js
<p>{nombre}</p>
```

En ese párrafo nos aparecerá el valor de la variable **nombre**. Si queremos que el usuario pueda escribir/introducir información, tenemos que crear un input de tipo texto. Pero nosotros no vamos a tener que estar pendiente de lo que ha escrito el usuario, sino que es React quien encarga de conectar nuestro código JavaScript con la página. Queremos que de algún modo la caja de texto y la variable **nombre** estén conectados, de modo que lo que escriba el usuario se guarde en el estado.

Creamos la etiqueta **<input>** a la que le damos un atributo **value** por defecto:

```
App.js
<input type="text" value="Jon" />
```

SI abrimos la página, veremos que en el input aparece el nombre Jon pero **NO** vamos a poder modificarlo. Cuando React renderiza el componente, pinta los elementos con los valores que tienen en ese momento, por lo que **value** será un valor inmutable.

Podríamos, entonces, asignar al atributo **value** la variable de estado, de modo que cuando cambie esa variable también cambie el valor del **input**.

```
App.js
<input type="text" value={nombre} />
```

Ahora veremos que lo que aparece escrito en **input** es lo que tenemos almacenado en el estado, pero seguimos sin poder cambiar ese valor.

Para ello tendremos que definir un evento **onChange** dentro de la caja de texto para que, cuando se produzca un cambio en ella, React ejecute una función:

```
App.js
<input type="text" value={nombre} onChange={cambiarNombre} />
```

Y definimos ahora la función **cambiarNombre**:

```
function cambiarNombre(){
  setNombre('el valor que tenemos en la caja')
}
```

Lo que queremos que haga React es que actualice el estado con el valor que hemos escrito en la caja. Pero de momento no sabemos cómo hacer eso, ya que utilizaremos los **eventos** del navegador.

Un evento es cualquier interacción del usuario con el navegador. Al hacer click en un botón, al pasar el ratón por encima de un elemento, rellenar una caja de texto...se emiten eventos que el navegador recoge en un objeto. En nuestro caso será React quien intercepte ese evento y nos lo pase para poder utilizarlo.

En cada evento **onAlgo**, en la función que definimos para que React ejecute, recibimos **siempre** como primer parámetro ese evento que ha capturado el navegador y que React ha interceptado:

```
App.js
function cambiarNombre(event){
    console.log(event)
  }
```

Cuando se produzca algún cambio en la caja de texto (al escribir algo), veremos en la consola el objeto del **evento** que se ha producido: tendremos información sobre el elemento en el que se ha producido el evento dentro del atributo **target**. Para acceder al **value** de ese elemento podremos poner **event.target.value**

```
App.js
function cambiarNombre(event){
    console.log(event.target.value)
  }
```

Ahora, en la consola, saldrá lo que tenemos en la caja de texto seguido de la tecla que que pulsemos en cada momento. Entonces, si queremos modificar la variable de estado con lo que escribimos en la caja de texto, tendremos que escribir, en la función **setNombre()**, el valor de lo escrito: **event.target.value**.

```
App.js
```

```
function cambiarNombre(event){
    setNombre(event.target.value)
  }
```

Podemos hacer la comprobación y veremos que podemos escribir y borrar texto y, además, aparecerá lo que hemos escrito en el **p**árrafo que hemos creado antes.

Podemos crear ahora un botón para borrar el nombre escrito en el input y almacenado en el estado:

App.js
```
<button onClick={borrarNombre}>Borrar nombre</button>
```

App.js
```
function borrarNombre(){
    setNombre('');
  }
```

App.js
```
function App() {
  let [nombre, setNombre] = useState('Peio');

  function cambiarNombre(event) {
    setNombre(event.target.value)
  }

  function borrarNombre(){
    setNombre('');
  }


  return (
    <>
      <p>{nombre}</p>
      <input type="text" value={nombre} onChange={cambiarNombre} />
      <button onClick={borrarNombre}>Borrar nombre</button>
    </>
  );
}
```

Cambiar estado de React cuando es un array

Hasta ahora, cuando teníamos una cadena de texto o un número almacenados en el estado de React, cuando ejecutamos **setEstado()**, le pasamos un nuevo valor y el estado se actualiza correctamente.

Cuando en la variable de estado tenemos un array (o un objeto), la cosa se complica. Para saber por qué tenemos que saber qué hace React cuando ejecutamos la función **setEstado()**: se compara el valor que ya teníamos almacenado con el valor nuevo; si hay diferencia se actualiza y, si no, no se actualiza el estado.

Cuando declaramos un array, tenemos una variable que está apuntando a los valores que tenemos almacenados en ese array. Si asignamos un array a una variable de estado, estamos guardando los valores que tenemos almacenados en el array. Entonces, al modificar el array original (usando **.push**, por ejemplo) y guardarlo en el estado mediante **setEstado()**, a React le estamos pasando el mismo array (aunque tenga valores modificados), por lo que no actualizará el HTML.

Para que React actualice el HTML al cambiar algún valor del array que tenemos en el estado, tendremos que crear un **nuevo array** en el que copiaremos lo que tenemos en el array original y pasar en **setEstado()** el **nuevo array** modificado.