



PARTE III: Estados

INTRODUCCIÓN

En este documento veremos:

- Uso del evento **onChange**
- Capturar valores de formularios
- Uso de hook **useState**

INTRODUCCIÓN

Creación del proyecto

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

`create-react-app estados`

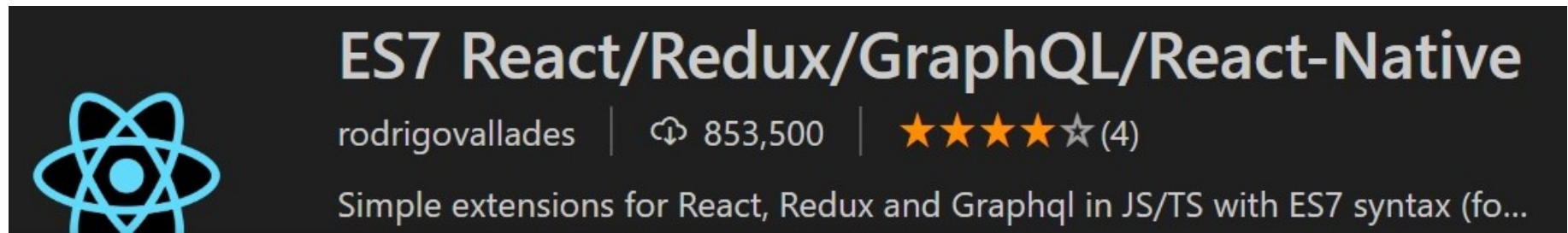
Preparar el entorno

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()**

ES7 REACT REDUX SNIPPET

Vamos a instalar un snippet para VSC que nos va a permitir confeccionar estructuras de código predeterminadas introduciendo únicamente un comando en el editor



Una vez instalado podemos editar el fichero **src/App.js** y tecleamos:

rafce

Y nos creará la estructura básica de nuestro componente

NOTA: podemos consultar la propia documentación del snippet para ver todas sus posibilidades y atajos

EL PROBLEMA DE LAS VARIABLES

1. Uso de variables dentro del documento JSX

Vamos a crear una variable con un valor y mostrar el contenido de esta variable en el documento JSX.

```
let nombre = "Mia Wallace";
```

```
<h1>Me llamo {nombre}</h1>
```

Me llamo Mia Wallace

NOTA: Si queremos asignar valores a más de una variable podemos utilizar este atajo JS:

```
let [nombre, edad] = ['Mia Wallace', 40];
```

Vemos como aparece el nombre asignado a la variable sin problemas

USO DE VARIABLES EN EL DOCUMENTO HTML

Ahora vamos a añadir un botón para cambiar el nombre cuando pulsemos sobre él:

```
const cambiarNombre = () => {  
  nombre = 'Vincent Vega';  
  console.log(nombre)  
}
```

```
<h1>Me llamo {nombre}</h1>  
<button onClick={cambiarNombre}>Cambiar</button>
```

Si pulsamos sobre el botón, vemos que no cambia el nombre en el navegador pero si vemos como en la consola el contenido de la variable **nombre** si que muestra el valor correcto.

Me llamo Mia Wallace

Vincent Vega

USO DE VARIABLES EN EL DOCUMENTO HTML

¿Cuál es el problema?

Pues que React actualiza el DOM solo cuando se ha ejecutado todo el código JS de forma que, aunque en la consola veamos el valor correcto, en el navegador vemos el valor que tenía la variable por defecto

Este comportamiento es realmente desconcertante y se soluciona gracias al uso de hook **useState**:

HOOK USESTATE

Vamos a implementar este hook para intentar solucionar el problema:

- Primero importamos el hook de la librería de react

```
import {useState} from 'react'
```

- Con este hook crearemos una constante con dos elementos:

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

nombre → es el nombre de nuestra variable

setNombre → es el nombre del método que utilizaremos para modificar el valor de nuestra variable nombre

useState → el método que modificará el valor de la variable a partir del método anterior (fijaos que, por defecto le asignamos el valor que teníamos antes en la variable **let** y que ya podremos eliminar)

HOOK USESTATE

- Y ahora modificaremos la función cambiarNombre para utilizar el hook para asignar el nuevo nombre en vez de la asignación directa que teníamos en la variable:

```
const cambiarNombre = () => {  
  nombre = 'Vincent Vega';  
  setNombre('Vincent Vega');  
  console.log(nombre)  
}
```

- Vemos como ahora el nuevo valor lo vemos reflejado en el navegador pero en la consola seguimos viendo el valor antiguo

Me llamo Vincent Vega

Mia Wallace

HOOK USESTATE

- Esto es debido a que la actualización de la variable se realiza una vez ha finalizado la ejecución de la función **cambiarNombre**.
- Si ponemos un **console.log** a continuación de la función veremos como si que en la consola aparece el valor correcto

Vincent Vega

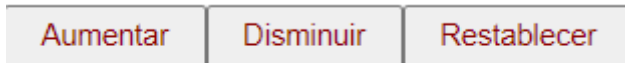
- Una vez que vemos que funciona podemos simplificar un poco el código y, en vez de utilizar la función **cambiarNombre**, podemos realizar al asignación del valor a la variable **nombre** directamente en el evento **onClick**:

```
<button onClick={() => setNombre("Vincent Vega")}>Cambiar</button>
```

EJERCICIO CONTADOR

Vamos a realizar otro ejemplo para profundizar un poco más en el uso del hook **useState**.

Vamos a realizar un contador de forma que, partiendo de cero, cada vez que se pulse un botón incrementaremos en uno el contador, si pulsamos un segundo botón restaremos uno y, con un tercer botón, lo restableceremos a cero



0

```
<button onClick={aumentar}>Aumentar</button>  
<button onClick={dismunuir}>Disminuir</button>  
<button onClick={restablecer}>Restablecer</button>  
<h1>{contador}</h1>
```

EJERCICIO CONTADOR

Creamos la variable **contador** y las tres funciones, inicialmente vacías:

```
const [contador, setContador] = useState(0)
const aumentar = () => { ... }
const disminuir = () => { ... }
const restablecer = () => { ... }
```

Para aumentar el contador ya hemos visto en el ejemplo anterior que no podemos cambiar directamente el valor de la variable, es decir, no podemos asignar directamente **contador++**. En su lugar tendremos que utilizar el método **setContador**:

```
setContador(contador+1);
```

Para disminuir el contador:

```
setContador(contador-1);
```

Y para restablecerlo:

```
setContador(0);
```

EJERCICIO CONTADOR

Para simplificar, podemos suprimir las tres funciones y realizar la operativa de incrementar, decrementar o restablecer dentro del propio evento **onClick**:

```
<button onClick={() => setContador(contador + 1)}>Aumentar</button>
```

```
<button onClick={() => setContador(contador - 1)}>Disminuir</button>
```

```
<button onClick={() => setContador(0)}>Restablecer</button>
```

NOTA: no podemos utilizar `onClick={setContador(contador + 1)}` porque entonces la función `setContador` se ejecutaría en un bucle infinito

EJERCICIO CONTADOR

¿Y si probamos, por ejemplo en incrementar, a utilizar?

```
<button onClick={() => setContador(contador++)}>Aumentar</button>
```

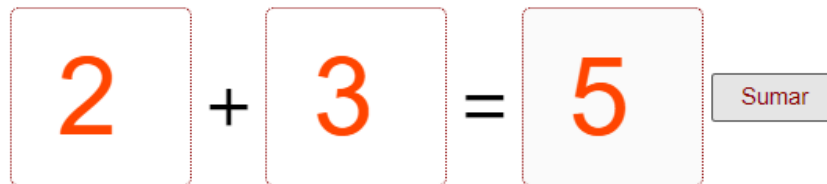
Veremos como el incremento no se realiza con la primera pulsación del botón sino a partir de la segunda

El problema es que hemos puesto el operador unario **++** después de **contador**, para solucionarlo lo utilizaremos antes de contador de forma que, primero se incremente en uno y, posteriormente, se asigna el nuevo valor:

```
<button onClick={() => setContador(++contador)}>Aumentar</button>
```

EJERCICIO CON INPUT

Y vamos a ver un tercer ejemplo para ver como utilizar `input` para solicitar valores al usuario y poder utilizarlos para calcular una suma.



```
<div className='caja'>
```

```
  <input type='number' value={numero1}/>+
```

```
  <input type='number' value={numero2}/>=
```

```
  <input type='number' value={resultado} disabled/>
```

```
  <button onClick={sumar}>Sumar</button>
```

```
</div>
```

```
input {color:orangered; width: 100px; line-height: 100px;margin:5px;text-align:
center;border-radius: 5px;border:1px dotted darkred;font-size: 4rem;}
```

```
.caja {font-size: 3rem;display: flex;align-items: center;}
```

```
button {font-size: rem; padding: 5px 15px;color: darkred}
```

NOTA: recordad importar el archivo css con `import './App.css';`

EJERCICIO CON INPUT

Vamos a definir las constantes para los tres valores:

```
const [numero1, setNumero1] = useState();  
const [numero2, setNumero2] = useState();  
const [resultado, setResultado] = useState();
```

Y la función que realizará la suma de los valores:

```
const sumar = () => {  
    setResultado(Number(numero1) + Number(numero2))  
}
```

NOTA: Recordad que tenemos que convertir a numéricos los datos que capturamos de un formulario ya que éstos siempre son de tipo texto

Obviamente si probamos ahora no funcionará la suma porque nos falta asignar los valores que capturemos del formulario a las variables **numero1** y **numero2** utilizando el hook **useState**

EJERCICIO CON INPUT

Añadimos el evento **onChange** para detectar cuando se informa un dato en los input y le asociamos la función donde asignaremos el valor a cada una de las variables:

```
<input type='number' value={numero1} onChange={asignarValor1}/>  
<input type='number' value={numero2} onChange={asignarValor2}/>
```

Y definimos las dos funciones con el hook **useState**:

```
const asignarValor1 = (e) => {  
  setNumero1(e.target.value);  
}
```

```
const asignarValor2 = (e) => {  
  setNumero2(e.target.value);  
}
```

Fijaos como recuperemos el valor del formulario utilizando el atributo **value** del objeto **e.target** donde **e** es el objeto que se crea automáticamente cuando se activa el evento y que contiene infinidad de información

EJERCICIO CON INPUT

Otra opción para ahorrarnos las funciones **asignarValor** es utilizar de nuevo el setter de cada uno de los valores dentro del evento **onChange**:

```
<input type='number' value={numero1} onChange={(e) =>
{setNumero1(e.target.value)}}/>
```

```
<input type='number' value={numero2} onChange={(e) =>
{setNumero2(e.target.value)}}/>
```

ACTIVIDAD 3

ACTIVIDAD

Para consolidar lo que hemos visto vamos a realizar una actividad

Actividad 3.1

A partir de un array de objetos con los títulos de tres películas y sus correspondientes directores vamos a confeccionar una caja con un botón de forma que, al pulsarlo, se irá mostrando título y dirección de cada una de las películas de forma sucesiva:

Ver Siguiente

Ver Siguiente

La dirección de la película **Apocalypto** corresponde a **Mel Gibson**

Ver Siguiente

La dirección de la película **La Milla Verde** corresponde a **Frank Darabont**

ACTIVIDAD

Actividad 3.1

- Necesitamos el array con los datos de las películas

```
const peliculas = [  
  {  
    titulo: 'Apocalypto',  
    direccion: 'Mel Gibson'  
  },  
  {  
    titulo: 'La Milla Verde',  
    direccion: 'Frank Darabont'  
  },  
  {  
    titulo: 'El resplandor',  
    direccion: 'Stanley Kubrick'  
  }  
]
```

ACTIVIDAD

Actividad 3.1

- Y la caja donde mostraremos los datos de cada película con un poco de css:

```
<div className="actividades peliculas">  
  <button onClick={datosPelicula}>Ver Siguiente</button>  
  <div>{mensaje}</div>  
</div>
```

```
@import url('https://fonts.googleapis.com/css2?  
family=Montserrat:wght@100;500&display=swap');  
* {font-family: 'Montserrat', sans-serif;}  
.actividades {width: 50%; margin:auto; border: 2px solid grey;padding: 20px;}  
.titulo {color:red}  
.direccion {color: green}
```

ACTIVIDAD

Actividad 3.1

- Vamos a crear la correspondiente variable para el mensaje para que utilice el hook **useState**

```
const [mensaje, setMensaje] = useState()
```

- La idea es que, cuando vayamos pulsando al botón '*siguiente*' se muestre la película que ocupa la posición 0 en el array de películas, la película de la posición 1 cuando pulsemos por segunda vez y así sucesivamente hasta llegar al final del array y, entonces, volver a mostrar de nuevo la primera película.

Para ello nos crearemos también una variable **contador** que inicializamos a cero:

```
const [contador, setContador] = useState(0)
```

NOTA: No necesitamos utilizar **useState()** con el array **peliculas** porque éste es inmutable, pero no así el contador para acceder a cada una de las películas ya que este si que es mutable

ACTIVIDAD

Actividad 3.1

- Ahora vamos a crear la función que se encarga de mostrar los datos de la película cada vez que pulsamos el botón:

```
const datosPelicula = () => { ... }
```

- Y, dentro de la función realizamos las siguientes tareas:

1.- Confeccionamos el texto que queremos mostrar en la caja con el título y la dirección de la película

```
let texto = <h2>La dirección de la pelicula <span  
className='titulo'>{peliculas[contador].titulo}</span> corresponde a  
<span className='direccion'>{peliculas[contador].direccion}</span></h2>;
```

NOTA: hemos asignado un `className` a los `span` de título y dirección para que se muestren en color rojo y verde respectivamente

2.- Asignamos el texto a la variable `mensaje` que estamos utilizando en la caja del documento JSX

```
setMensaje(texto);
```


ACTIVIDAD

Actividad 3.1

3.- Incrementamos en uno el **contador** (recordad que el incremento se realiza realmente cuando se acaba de ejecutar la función) solo cuando el incremento del contador no supere la longitud del array de películas.

En caso contrario lo inicializaremos a cero (ya que tendríamos un error al intentar acceder a un elemento más allá del número de elementos del array)

```
if (contador >= peliculas.length-1) {  
    setContador(0)  
} else {  
    setContador(contador+1)  
}
```

NOTA: Otra forma más elegante sería utilizando el operador aritmético módulo:

```
setContador((contador) => (contador + 1) % peliculas.length)
```

Aviso Legal

Los derechos de propiedad intelectual sobre el presente documento son titularidad de David Alcolea Martinez Administrador, propietario y responsable de www.alcyon-it.com El ejercicio exclusivo de los derechos de reproducción, distribución, comunicación pública y transformación pertenecen a la citada persona.

Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro).

La ley que ampara los derechos de autor establece que: “La introducción de una obra en una base de datos o en una página web accesible a través de Internet constituye un acto de comunicación pública y precisa la autorización expresa del autor”. El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multa, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujesen, plagiaran, distribuyeren o comunicaren públicamente, en todo o en parte, o su transformación, interpretación o ejecución fijada en cualquier tipo de soporte o comunicada a través de cualquier medio.

El usuario que acceda a este documento no puede copiar, modificar, distribuir, transmitir, reproducir, publicar, ceder, vender los elementos anteriormente mencionados o un extracto de los mismos o crear nuevos productos o servicios derivados de la información que contiene.

Cualquier reproducción, transmisión, adaptación, traducción, modificación, comunicación al público, o cualquier otra explotación de todo o parte del contenido de este documento, efectuada de cualquier forma o por cualquier medio, electrónico, mecánico u otro, están estrictamente prohibidos salvo autorización previa por escrito de David Alcolea. El autor de la presente obra podría autorizar a que se reproduzcan sus contenidos en otro sitio web u otro soporte (libro, revista, e-book, etc.) siempre y cuando se produzcan dos condiciones:

1. Se solicite previamente por escrito mediante email o mediante correo ordinario.
2. En caso de aceptación, no se modifiquen los textos y se cite la fuente con absoluta claridad.

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

