



JavaScript y React

Clase 15

KOICA

IGU HANDONG GLOBAL
UNIVERSITY



UNA

OBJETIVOS DE LA CLASE 15

- Manejar Componentes: propiedades.
- Manejar Componentes: eventos generados por una componente
- Manejar Array: métodos push y pop
- Manejar ES6 - Funciones con parámetros Rest
- Manejar ES6 - operador Spread (operador de propagación)
- Implementar los códigos de ejemplos propuestos en clase.



Array: métodos push y pop

— — —

- Como los Array en Javascript son objetos, los mismos tienen una serie de métodos que nos facilitan trabajar con ellos.
- Para insertar elementos en un vector hemos visto que con solo asignar un valor al vector en un determinado índice el dato queda almacenado y eventualmente el atributo length modificado:

```
let vec=[];  
  
vec[0]=10;  
  
vec[1]=20;  
  
document.write(vec.length); //imprime 2
```

Array: métodos push y pop

— — —

- Esta sintaxis tenemos que tener cuidado como variamos el subíndice para no dejar componentes vacías si queremos implementar un array denso.
- Una variante para resolver este mismo problema es utilizar el método push del objeto Array. Este método añade el valor al final del vector:

```
let vec=[];  
  
vec[0]=10;  
  
vec[1]=20;  
  
document.write(vec.length); //imprime 2
```

Array: métodos push y pop

— — —

- Automáticamente cuando llamamos al método push el valor que le pasamos en el parámetro se almacena en el vector y se incrementa el atributo length.
- Podemos inclusive llamar al método push pasando más de 1 parámetro:

```
let vec=[];  
  
vec.push(10,20);  
  
document.write(vec.length); //imprime 2
```

Array: métodos push y pop

— — —

- El método inverso llamado pop extrae el último elemento del Array y decrementa en uno el atributo length:

```
let vec=[];  
  
vec.push(10,20,30,40);  
  
document.write(vec.length+'<br>'); //imprime 4  
  
vec.pop();  
  
document.write(vec.length+'<br>'); //imprime 3  
  
document.write(vec.pop()+'<br>'); //imprime un 30  
  
document.write(vec.length+'<br>'); //imprime 2
```

Array: métodos push y pop

- El método `pop()` además de eliminar el último elemento del vector retorna el valor almacenado en dicha componente.
- Si llamamos al método `pop` y el vector está vacío retorna el valor `undefined`.

ES6 - Funciones con parámetros Rest

- Otra funcionalidad en JavaScript con la actualización ECMAScript 2015 (también llamada ES6) es la posibilidad de pasar una lista indefinida de valores y que los reciba un vector.
- El concepto de parámetros Rest se logra antecediendo tres puntos al nombre del parámetro (el parámetro es un objeto de la clase Array con todas sus funcionalidades):

```
function sumar(...parametro)
```


ES6 - Funciones con parámetros Rest

- Luego en la llamada podemos pasar una lista indeterminada de datos:

```
sumar(4, 55, 33);
```

ES6 - Funciones con parámetros Rest

```
<script>

  function sumar(...valores) {
    let suma = 0;
    for (let x = 0; x < valores.length; x++)
      suma += valores[x];
    return suma;
  }

  document.write(sumar(10, 2, 44, 3));
  document.write('<br>');
  document.write(sumar(1, 2));
  document.write('<br>');
  document.write(sumar());
  document.write('<br>');

</script>
```

ES6 - Funciones con parámetros Rest

- El parámetro valores es en realidad un vector y como tal podemos disponer un for para recorrerlo y acceder a sus elementos mediante un subíndice:

```
function sumar(...valores) {  
    let suma = 0;  
    for (let x = 0; x < valores.length; x++)  
        suma += valores[x];  
    return suma;  
}
```

ES6 - Funciones con parámetros Rest

- Cuando llamamos a la función podemos pasar una cantidad variable de enteros para que sean sumados:

```
document.write(sumar(10, 2, 44, 3));
```

```
document.write(sumar(1, 2));
```

```
document.write(sumar());
```

- La función con un parámetro Rest puede tener otros parámetros pero se deben declarar antes:

```
function operar(tipo, ...valores)
```

ES6 - operador Spread (operador de propagación)

- El operador Spread permite descomponer una estructura de datos en elementos individuales. Es la operación inversa de los parámetros Rest.
- La sintaxis se aplica anteponiendo al nombre de la variable tres puntos:

```
<script>
    function sumar(x, y, z) {
        return x + y + z;
    }
    const vec = [10, 20, 30];
    const s = sumar(...vec);
    document.write(s);
</script>
```

ES6 - operador Spread (operador de propagación)

- Es decir, en la llamada a la función sumar le antecedemos tres puntos al nombre de la variable que vamos a pasar:

```
const s = sumar(...vec);
```

- Luego el parámetro x recibe la primer componente, el parámetro y la segunda componente y finalmente el parámetro z recibe la tercer componente.
- Esto anteriormente debíamos hacerlo indicando cada uno de los elementos del vector:

```
let s = sumar(vec[0], vec[1], vec[2]);
```

Componentes: propiedades

- El elemento fundamental que tenemos para comunicarnos con una componente son las propiedades. Mediante las propiedades podemos enviar datos a la componente para que los muestre.
- En el concepto anterior vimos cómo pasar un valor entero a la componente Dado mediante la sintaxis (empleando una variable de estado con el API de Hook de React):

```
<Dado valor={valor1} />
```

Componentes: propiedades

— — —

- Luego dentro de la componente 'Dado' accedemos al valor pasado mediante el parámetro 'props':

```
function Dado(props) {  
  
  return (  
  
    <div className="dado-recuadro">{props.valor}</div>  
  
  );  
  
}
```

- Podemos pasar cualquier tipo de datos a una componente, no solo tipos primitivos como enteros, reales, string.

Problema

- Confeccionar una aplicación que permita ingresar por teclado dos enteros y nos muestre la suma. Agregar a una lista todas las operaciones ejecutadas hasta este momento.
- Crear un nuevo proyecto llamado proyecto007
- Primero creamos el archivo 'ListadoResultados.jsx' en la carpeta 'src' donde definimos la componente con el mismo nombre:

```
function ListadoResultados(props) {  
  return (  
    <ul>  
      {props.resultados.map((elemento) =>  
        <li>La suma de {elemento.valor1} y {elemento.valor2} es {elemento.resultado}</li>  
      )}  
    </ul>  
  );  
}  
  
export default ListadoResultados;
```

Problema

— — —

- El parámetro 'props' dispone de una propiedad llamada 'resultados' que llegará desde la componente 'App'. La misma es un vector con componentes de tipo objeto que almacenan los dos valores y el resultado de su suma.
- Podemos imaginar que llega algo similar a esto:

```
[
  {
    valor1: 10,
    valor2: 12,
    resultado: 22
  },
  {
    valor1: 5,
    valor2: 20,
    resultado: 25
  },
  {
    valor1: 1,
    valor2: 1,
    resultado: 2
  }
]
```

```
import ListadoResultados from "../ListadoResultados";
import { useState } from "react";

function App() {

  const [operaciones, setOperacion] = useState([])

  function sumar(event) {
    event.preventDefault();
    const v1 = parseInt(event.target.valor1.value, 10)
    const v2 = parseInt(event.target.valor2.value, 10)
    const suma = v1 + v2
    const nuevo = {
      resultado: suma,
      valor1: v1,
      valor2: v2
    }
    setOperacion([nuevo, ...operaciones])
    event.target.valor1.value = ''
    event.target.valor2.value = ''
  }
}
```

```
return (
  <div>
    <form onSubmit={sumar}>
      <p>Ingrese primer valor:<input
type="text" name="valor1" /></p>
      <p>Ingrese segundo valor:<input
type="text" name="valor2" /></p>
      <input type="submit" value="Sumar"
/>
    </form>
    <ListadoResultados
resultados={operaciones} />
  </div>
);
}

export default App;
```

Problema

- Importamos el archivo que contiene la componente funcional ListadoResultados:

```
import ListadoResultados from "../ListadoResultados";
```

- En la función App definimos la variable de estado donde almacenaremos un arreglo con las distintas operaciones efectuadas:

```
const [operaciones, setOperacion] = useState([])
```

Problema

- En el bloque JSX además del formulario de entrada de datos propiamente dicho definimos un elemento de tipo 'ListadoResultados' y le pasamos la propiedad 'resultado' con el valor almacenado en la variable de estado 'operaciones':

```
return (  
  <div>  
    <form onSubmit={sumar}>  
      <p>Ingrese primer valor:<input type="text" name="valor1" /></p>  
      <p>Ingrese segundo valor:<input type="text" name="valor2" /></p>  
      <input type="submit" value="Sumar" />  
    </form>  
    <ListadoResultados resultados={operaciones} />  
  </div>  
) ;
```

Problema

- Cada vez que se presiona el botón sumar se ejecuta la función 'sumar' donde agregamos al estado un nuevo elemento al principio del vector, esto hará que se actualice la lista de resultados en la página sin tener que recargarla:

```
function sumar(event) {  
  event.preventDefault();  
  const v1 = parseInt(event.target.valor1.value, 10)  
  const v2 = parseInt(event.target.valor2.value, 10)  
  const suma = v1 + v2  
  const nuevo = {  
    resultado: suma,  
    valor1: v1,  
    valor2: v2  
  }  
}
```

```
setOperacion([nuevo, ...operaciones])  
event.target.valor1.value = ''  
event.target.valor2.value = ''  
}
```

Problema

— — —

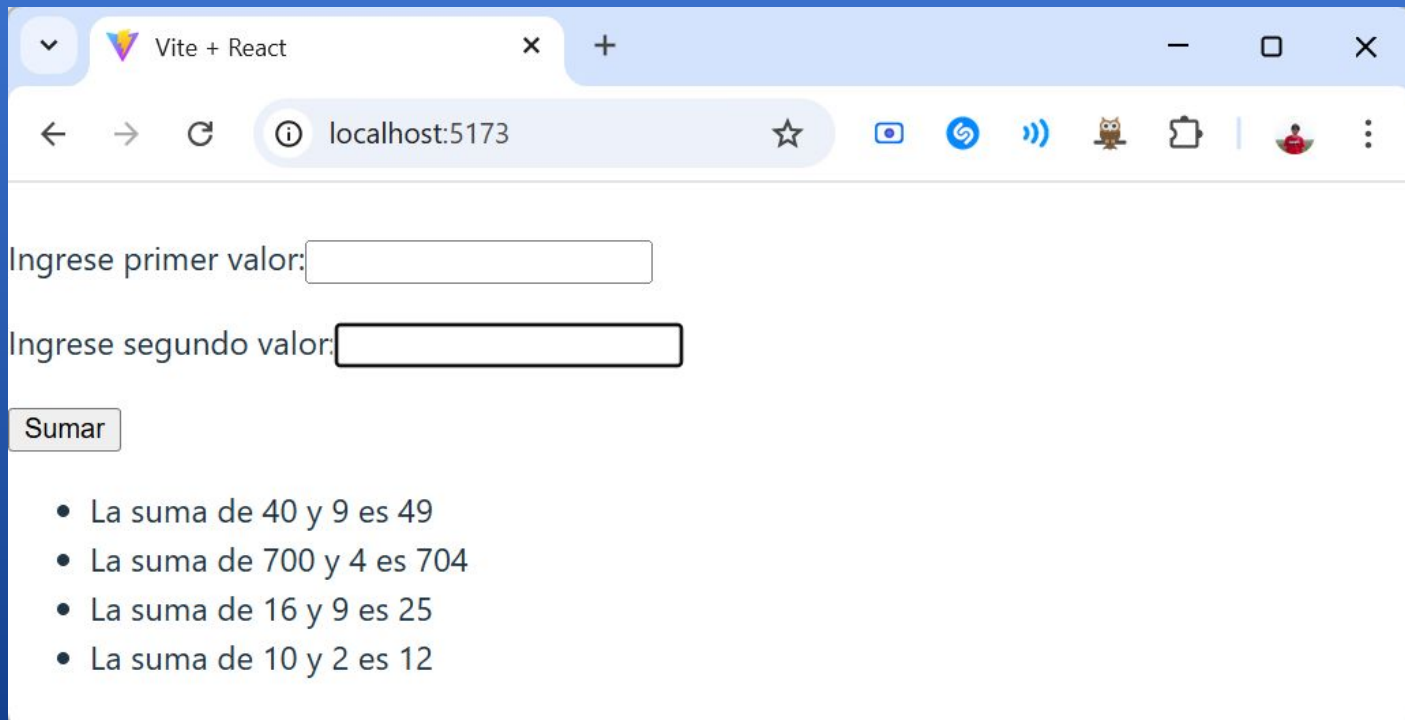
- Para actualizar el estado en la variable 'operaciones' de manera correcta, necesitamos pasar otro arreglo completo. Utilizamos el operador spread para descomponer el vector actual y agregar como primer elemento la nueva operación:

```
setOperacion([nuevo, ...operaciones])
```

- Si queremos agregar la operación al final del vector luego podemos implementar la siguiente sintaxis:

```
setOperacion([...operaciones, nuevo])
```


Problema



A screenshot of a web browser window. The tab is titled 'Vite + React'. The address bar shows 'localhost:5173'. The page content includes two input fields for numbers, a 'Sumar' button, and a list of four example calculations.

Ingrese primer valor:

Ingrese segundo valor:

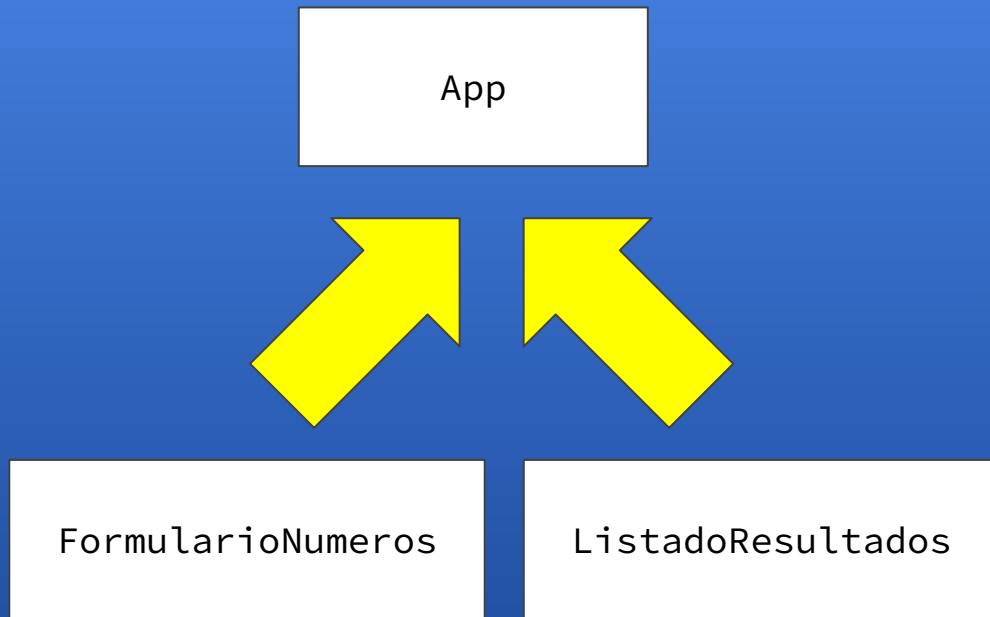
- La suma de 40 y 9 es 49
- La suma de 700 y 4 es 704
- La suma de 16 y 9 es 25
- La suma de 10 y 2 es 12

Componentes: eventos generados por una componente

- Una aplicación consta de una componente principal (utilizando la herramienta create-react-app se llama App), en esta definimos objetos de otras componentes y así sucesivamente cada componente puede estar construida en base a otras componentes.
- Veremos ahora que una componente puede emitir un evento para informar a la componente padre un suceso.

Problema

- Modificaremos el proyecto del concepto anterior para implementar el formulario de entrada de datos en otra componente llamada 'FormularioNumeros'. Las componentes que ahora tendrá la aplicación son:



Problema

- Debemos crear el archivo 'FormularioNumeros.jsx' en la carpeta 'src' con el siguiente contenido:

```
function FormularioNumeros(props) {  
  return (  
    <form onSubmit={props.onSumar}>  
      <p>Ingrese primer valor:<input type="text" name="valor1" /></p>  
      <p>Ingrese segundo valor:<input type="text" name="valor2" /></p>  
      <input type="submit" value="Sumar" />  
    </form>  
  );  
}  
  
export default FormularioNumeros;
```

Problema

— — —

- En el evento onSubmit le pasaremos la referencia a una función que llega como una propiedad en el parámetro 'props' y lo llama en la componente padre con el nombre 'onSumar:

```
<FormularioNumeros onSumar={sumar} />
```

Problema

— — —

- Ahora el código de la componente 'App' queda con la siguiente sintaxis:

```
import ListadoResultados from "../ListadoResultados";
import FormularioNumeros from "../FormularioNumeros";
import { useState } from "react";
function App() {
  const [operaciones, setOperacion] = useState([])
  function sumar(event) {
    event.preventDefault();
    const v1 = parseInt(event.target.valor1.value, 10)
    const v2 = parseInt(event.target.valor2.value, 10)
    const suma = v1 + v2
    const nuevo = {
      resultado: suma,
      valor1: v1,
      valor2: v2
    }
  }
}
```

Problema

— — —

- Ahora el código de la componente 'App' queda con la siguiente sintaxis:

```
    setOperacion([nuevo, ...operaciones])
    event.target.valor1.value = ''
    event.target.valor2.value = ''
  }

  return (
    <div>
      <FormularioNumeros onSumar={sumar} />
      <ListadoResultados resultados={operaciones} />
    </div>
  );
}

export default App;
```

Problema

- Debemos importar primero los dos archivos que contienen las componentes:

```
import ListadoResultados from "../ListadoResultados";  
import FormularioNumeros from "../FormularioNumeros";
```


Problema

- El bloque JSX ahora queda muy simple, hemos inicializado la propiedad 'onSumar' con la referencia de la función 'sumar':

```
return (  
  <div>  
    <FormularioNumeros onSumar={sumar} />  
    <ListadoResultados resultados={operaciones} />  
  </div>  
);
```

Problema

— — —

- Como vemos la función 'sumar' se ejecuta cuando la componente FormularioNumeros lo requiera.
- Finalmente el archivo 'ListadoResultados' no varía con respecto al problema anterior:

```
function ListadoResultados(props) {  
  return (  
    <ul>  
      {props.resultados.map((elemento) =>  
        <li>La suma de {elemento.valor1} y  
        {elemento.valor2} es {elemento.resultado}</li>  
      )}  
    </ul>  
  );  
}  
  
export default ListadoResultados;
```

EJERCICIOS ADICIONALES PROPUESTOS

— — —





**¡MUCHAS
GRACIAS!**