



JavaScript y React

Clase 12

KOICA

IGU HANDONG GLOBAL
UNIVERSITY



UNA

OBJETIVOS DE LA CLASE 12

- Manejar otras restricciones que tiene el formato JSX.
- Manejar el Evento: submit
- Operadores de igualdad y desigualdad (`==`, `===`, `!=`, `!==`).
- Implementar los códigos de ejemplos propuestos en clase.



Crear tu segundo proyecto React con Vite

— — —

Abrí la terminal o consola de tu sistema operativo y escribí:

```
npm create vite@latest
```

Te pedirá:

- Nombre del proyecto: proyecto002
- Framework: seleccioná React
- Variant: elegí JavaScript (más adelante podés probar TypeScript)

Realizar el comando

```
cd proyecto002
```

 ¿Qué hace?

- cd significa “cambiar de carpeta” (viene del inglés change directory).
- Te metés dentro de la carpeta de tu nuevo proyecto.

 Es como decirle a la compu:

“Ahora quiero trabajar dentro de la carpeta proyecto002”.

 Después de esto, estás “dentro” del proyecto.

Realizar el comando

```
npm install
```

 ¿Qué hace?

Le dice a Node.js:

“Instalá todos los archivos que este proyecto necesita para funcionar”.

Realizar el comando

```
npm run dev
```

 ¿Qué hace?

Le dice a Vite:

“Arrancá el servidor para que yo pueda ver mi proyecto en el navegador”.

 Vite va a mostrar la página en `http://localhost:5173` (puede variar).

Modificar tu segunda app

Abrí src/App.jsx y reemplazá el código por:

```
function App() {  
  return (  
    <div>  
      <h1>¡Hola Mundo desde React con Vite!</h1>  
      <p>Mi primera aplicación de React</p>  
    </div>  
  );  
}  
  
export default App;
```



Guardá y mirá el navegador: los cambios se actualizan solos sin recargar.

Modificar tu segunda app

- La función App no retorna ni HTML, ni Javascript puro, es un nuevo formato propuesto por los creadores de React que luego de ser compilado se genera Javascript puro que lo pueden entender los navegadores.
- Hay ciertas reglas que debe cumplir el formato JSX (JavaScript XML), presentaremos algunas de ellas en este segundo ejercicio.
- Modifiquemos valor devuelto por la función App:

Modificar tu segunda app

— — —

```
import './App.css';

function retornarAleatorio() {
  return Math.trunc(Math.random() * 10);
}

function App() {
  const siglo = 21
  const persona = {
    nombre: 'Ivan',
    edad: 34
  }
}
```

```
return (
  <div>
    <h1>Título nivel 1</h1>
    <hr />
    <p>Estamos en el siglo {siglo}</p>
    <h3>Acceso a un objeto</h3>
    <p>{persona.nombre} tiene {persona.edad} años</p>
    <h3>Llamada a un método</h3>
    <p>Un valor aleatorio llamando a un método.</p>
    {retornarAleatorio()}
    <h3>Calculo inmediato de expresiones</h3>
    3 + 3 = {3 + 3}
  </div>
);
}

export default App;
```

Formato JSX

- La función App tiene por objetivo retornar el elemento JSX que representa la interfaz visual de la componente 'App' (por el momento desarrollaremos toda nuestra aplicación en una única componente, luego veremos que un programa se descompone en muchas componentes)
- Como vemos dentro del bloque de JSX podemos disponer etiquetas HTML tal como conocemos:

```
<h1>Título nivel 1</h1>
```

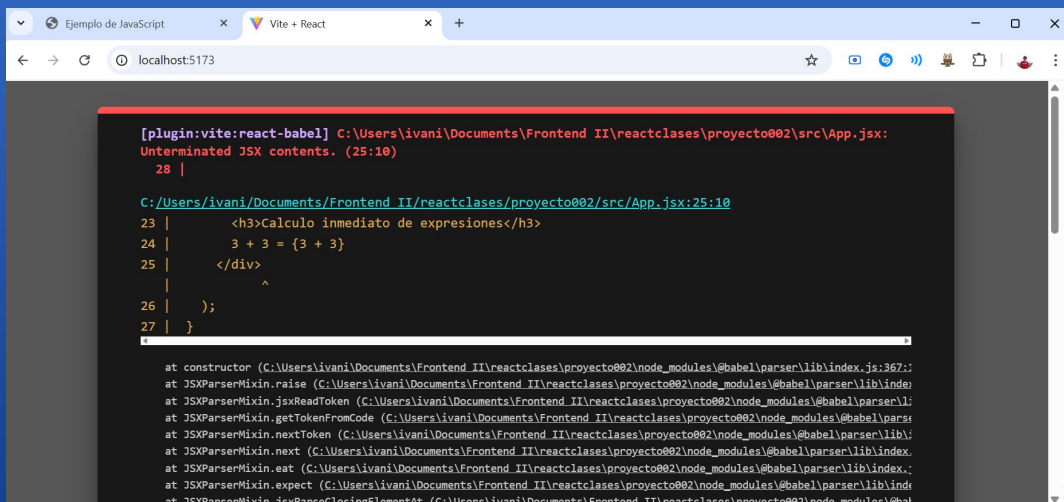
Formato JSX

- Una restricción de JSX es que siempre los elementos HTML deben tener su marca de comienzo y fin, y en el caso que solo tengan una etiqueta que es tanto de comienzo como fin debemos agregar el caracter '/':

```
<hr />
```

Formato JSX

- Si nos olvidamos de agregar la barra de cierre se genera un error cuando tratamos de compilar la aplicación:



```
[plugin:vite:react-babel] C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\src\App.jsx:
Unterminated JSX contents. (25:10)
 28 |
    |
    |
C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\src\App.jsx:25:10
 23 |     <h3>Calculo inmediato de expresiones</h3>
 24 |       3 + 3 = {3 + 3}
 25 |     </div>
    |           ^
 26 |   );
 27 | }

at constructor (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index.js:367:
at JSXParserMixin.raise (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index
at JSXParserMixin.jsxReadToken (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\li
at JSXParserMixin.getTokenFromCode (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\pars
at JSXParserMixin.nextToken (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\
at JSXParserMixin.next (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index
at JSXParserMixin.eat (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index.
at JSXParserMixin.expect (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\ind
at JSXParserMixin.jsxParseClosingElementAt (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@ba
```

Formato JSX

- Dentro del bloque JSX podemos acceder a variables o constantes indicando entre llaves dicha variable o constante:

```
<p>Estamos en el siglo {siglo}</p>
```

Formato JSX

— — —

- Luego cuando se compila en lugar de la expresión se muestra el contenido de la variable o constante:



Formato JSX

- En forma similar podemos disponer expresiones para acceder a propiedades de un objeto previamente definido:

```
<p>{persona.nombre} tiene {persona.edad} años</p>
```

- Otra posibilidad en una expresión es hacer la llamada a otras funciones:

```
{retornarAleatorio()}
```

- Podemos inclusive disponer una operación que será ejecutada previo a su visualización:

```
3 + 3 = {3 + 3}
```

Otras restricciones que tiene el formato JSX

- Se recomienda dividir el JSX en varias líneas para facilitar la lectura, también recomiendan envolverlo entre paréntesis para evitar los inconvenientes de la inserción automática de punto y coma.
- Para definir valores a las propiedades de un elementos HTML mediante expresiones no debemos disponer las comillas. Por ejemplo modifique el método render() y pruebe esto:


```
import './App.css';

function retornarAleatorio() {
  return Math.trunc(Math.random() * 10);
}

function App() {
  const buscadores =
['http://www.google.com' ,
  'http://www.bing.com' ,
  'https://www.yahoo.com' ];
  return (
    <div>
      <a href={buscadores[0]}>Google</a><br
/>
      <a href={buscadores[1]}>Bing</a><br />
      <a href={buscadores[2]}>Yahoo</a><br />
    </div>
  );
}

export default App;
```

Otras restricciones que tiene el formato JSX

- No deben ir las comillas en la asignación de la propiedad href si el valor se extrae de una expresión:

```
<a href={buscadores[0]}>Google</a><br />
```

- Si queremos darle el valor directamente si se requieren las comillas:

```
<a href="http://www.google.com">Google</a><br />
```

Otras restricciones que tiene el formato JSX

- Podemos plantear funciones que retornen trozos de JSX que luego se agregan al que retorna la función App, probar de modificar nuevamente el problema con:

```
import './App.css';

function mostrarTitulo(tit) {
  return (<h1>
    {tit}
  </h1>);
}
```

```
function App() {
  return (
    <div>
      {mostrarTitulo('Hola Mundo')}
      {mostrarTitulo('Fin')}
    </div>
  );
}
export default App;
```

Una restricción del JSX es que siempre

— — —

- Debe retornar un elemento HTML que puede tener en su interior otros elementos anidados, pero nunca dos elementos HTML hermanos, esto genera un error:

```
function App() {  
  return (  
    <div>  
      <h1>Titulo 1</h1>  
    </div>  
    <div>  
      <h1>Titulo 2</h1>  
    </div>  
  );  
}
```

Debemos disponer obligatoriamente solo un elemento div que envuelva todo:

```
function App() {  
  return (  
    <div>  
      <div>  
        <h1>Titulo 1</h1>  
      </div>  
      <div>  
        <h1>Titulo 2</h1>  
      </div>  
    </div>  
  );  
}
```

Se puede utilizar otros elementos HTML para envolver todo el JSX como por ejemplo 'span', 'section' o cualquier otro

- Siempre y cuando no haya elementos hermanos en la raíz (inclusive podemos encerrarlo con etiquetas vacías):

```
function App() {  
  return (  
    <><div>  
      <h1>Titulo 1</h1>  
    </div><div>  
      <h1>Titulo 2</h1>  
    </div>  
  </>  
);
```

Importante

- Otra cosa muy importante que hay que tener en cuenta que los nombres de las propiedades de los elementos HTML cambian en varias situaciones:
- Debemos utilizar la palabra 'className' en lugar de class (debido a que class es una palabra clave de Javascript):

```
<h1 className="recuadro">Titulo 1</h1>
```

Importante

- Si el nombre de la propiedad está formada por más de una palabra luego el primer caracter a partir de la segunda palabra debe ir en mayúsculas:

```
<input type="text" tabIndex="1" />
```


Captura de eventos

- Los nombres de eventos en React comienzan con "on" y luego el primer caracter de cada palabra en mayúsculas:

```
onClick  
onDoubleClick  
onMouseEnter  
onMouseLeave  
onMouseMove  
onKeyPress  
onKeyUp  
onSubmit  
etc.
```

Problema

- Disponer dos controles de formulario HTML `input="number"` y un botón. Al presionar el botón mostrar en un alert su suma.
- Crear con la aplicación `npm create vite@latest` el proyecto003.

```
npm install
```

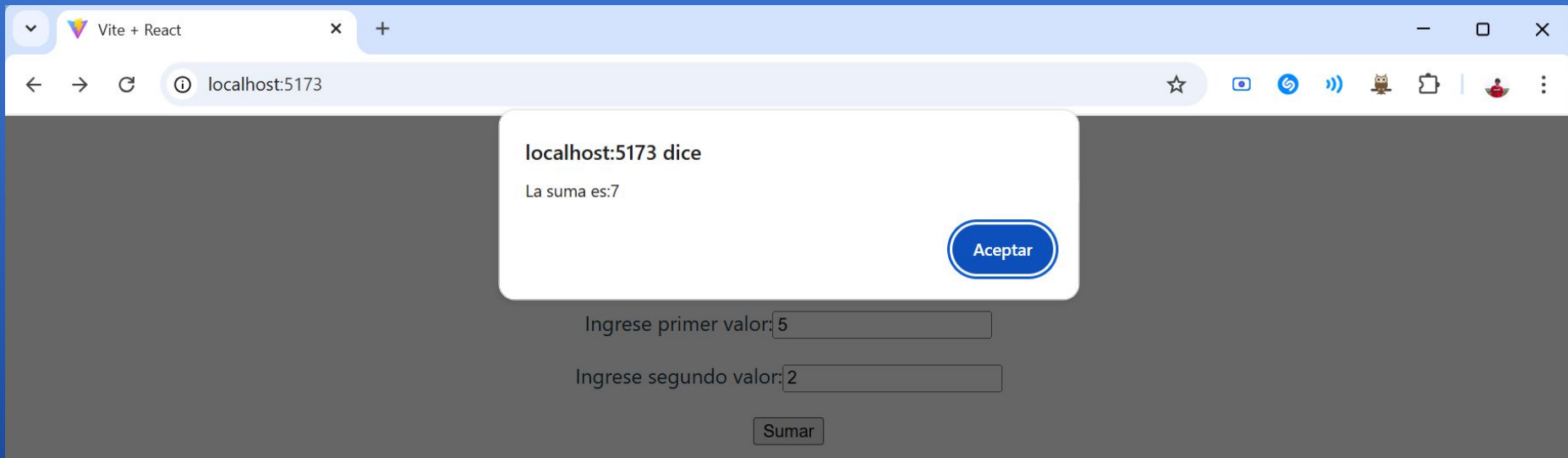
```
npm run dev
```

```
import './App.css';
```

```
function App() {  
  return (  
    <div>  
      <form onSubmit={presion}>  
        <p>Ingresa primer valor:  
          <input type="number" name="valor1" />  
        </p>  
        <p>Ingresa segundo valor:  
          <input type="number" name="valor2" />  
        </p>  
        <p>  
          <input type="submit" value="Sumar" />  
        </p>  
      </form>  
    </div>  
  );  
}
```

```
function presion(e) {  
  e.preventDefault();  
  const v1=parseInt(e.target.valor1.value, 10);  
  const v2=parseInt(e.target.valor2.value, 10);  
  const suma=v1+v2;  
  alert('La suma es:'+suma);  
}  
  
export default App;
```

Cuando ejecutamos la aplicación luego de cargar los dos enteros y presionar el botón 'submit' tenemos:



Captura de eventos

- Lo más importante en este problema es ver como enlazamos el evento onSubmit con la función presion.
- En la función App inicializamos el evento onSubmit con la referencia de la función 'presion':

```
<form onSubmit={presion}>
```

Captura de eventos

- La función `presion` propiamente dicha primero detiene el envío de datos al servidor llamando a `preventDefault`, luego recupera los dos valores ingresados por teclado y muestra su suma:

```
function presion(e) {  
  e.preventDefault();  
  const v1=parseInt(e.target.valor1.value, 10);  
  const v2=parseInt(e.target.valor2.value, 10);  
  const suma=v1+v2;  
  alert('La suma es:'+suma);  
}
```

Captura de eventos

- El objetivo de este concepto es ver un poco la sintaxis cómo enlazar un evento que dispara un control HTML y el método que lo captura.
- En conceptos futuros analizaremos cada uno de los controles de formulario y cómo procesarlos.

Evento: submit

- Todo formulario se le puede capturar el evento submit que se dispara previo a enviar los datos del formulario al servidor.
- Uno de los usos más extendidos es la de validar los datos ingresados al formulario y abortar el envío de los mismos al servidor (con esto liberamos sobrecargas del servidor)

Evento: submit

- El evento submit se dispara cuando presionamos un botón de tipo `type="submit"`.
- Para probar el funcionamiento del evento submit implementaremos un formulario que solicita la carga de una clave y la repetición de la misma. Luego cuando se presione un botón de tipo "submit" verificaremos que las dos claves ingresadas sean iguales.

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de JavaScript</title>
  <meta charset="UTF-8">
</head>
<body>
  <form method="post" action="procesar.php"
id="formulario1">
    Ingrese clave:
    <input type="password" id="clave1"
name="clave1" size="20" required>
    <br> Repita clave:
    <input type="password" id="clave2"
name="clave2" size="20" required>
    <br>
    <input type="submit" id="confirmar"
name="confirmar" value="Confirmar">
  </form>
```

```
<script>
document.getElementById("formulario1").addEventListener('submit', validar);

    function validar(evt) {

        let cla1 =
document.getElementById("clave1").value;

        let cla2 =
document.getElementById("clave2").value;

        if (cla1 != cla2) {

            alert('Las claves ingresadas son distintas');
            evt.preventDefault();

        }

    }
</script>
</body>
</html>
```

Tengamos en cuenta que la primera línea indica que se trata de una página de HTML5:

— — —

- `<!doctype html>`

Definimos un formulario que solicita la carga de dos claves y un botón submit para enviar los datos al servidor:

```
<form method="post" action="procesar.php" id="formulario1">  
Ingrese clave:  
<input type="password" id="clave1" name="clave1" size="20" required>  
<br>  
Repita clave:  
<input type="password" id="clave2" name="clave2" size="20" required>  
<br>  
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">  
</form>
```

Registramos el evento submit del formulario:

— — —

- `document.getElementById("formulario1").addEventListener('submit', validar);`

La función `validar` extrae los contenidos de los dos "password" y verificamos si tienen string distintos en cuyo caso llamando al método `preventDefault` del objeto que llega como parámetro, lo cual previene que los datos se envíen al servidor:

```
function validar(evt) {  
    let cla1 = document.getElementById("clave1").value;  
    let cla2 = document.getElementById("clave2").value;  
    if (cla1 !== cla2) {  
        alert('Las claves ingresadas son distintas');  
        evt.preventDefault();  
    }  
}
```

Operadores de igualdad y desigualdad (==, ===, !=, !==)

El operador == primero hace la conversión a un mismo tipo de dato para verificar si son iguales, en cambio el operador === llamado operador de igualdad estricta compara los valores sin hacer conversiones. Esto hace que cuando utilizamos el operador === retorne siempre false en caso que las variables que comparamos sean de distinto tipo.

Operadores de igualdad y desigualdad (==, ===, !=, !==)

El siguiente if se verifica verdadero ya que con el operador == primero se transforman los dos valores al mismo tipo de datos previos a verificar su igualdad:

```
let v1='55';  
  
if (v1==55)  
    document.write('son iguales');
```

Operadores de igualdad y desigualdad (==, ===, !=, !==)

En cambio el siguiente if utilizando el operador de igualdad estricta se verifica false ya que son variables o valores de distinto tipo:

```
let v1='55';  
  
if (v1===55)  
    document.write('son iguales');
```

Operadores de igualdad y desigualdad (==, ===, !=, !==)

Los operadores != y !== son los opuestos de == y ===.

El operador != retorna true si son distintos previo conversión a un mismo tipo de dato y el operador !== retorna true si los valores son distintos sin hacer una conversión previa, teniendo en cuenta que para tipos de datos distintos es true el valor retornado.

El siguiente if se verifica false:

```
let v1='55';  
  
if (v1!=55)  
  
    document.write('son distintos');
```


Operadores de igualdad y desigualdad (==, ===, !=, !==)

En cambio utilizando el operador !== se verifica verdadero ya que son tipos de datos desiguales:

```
let v1='55';  
  
if (v1!==55)  
  
    document.write('son distintos');
```

Operadores de igualdad y desigualdad (==, ===, !=, !==)

El operador de igualdad estricto === analiza los operandos de la siguiente forma:

- Si los dos valores son de diferente tipo luego el resultado es false.
- Si los dos valores almacenan null luego el resultado es true. Lo mismo si los dos valores almacenan undefined.
- Si uno de los valores almacena el valor NaN luego la condición se verifica como false.

Operadores de igualdad y desigualdad (==, ===, !=, !==)

El operador de igualdad == analiza los operandos de la siguiente forma:

- Si los dos operandos no son del mismo tipo el operador == primero verifica si uno de los valores es null y el otro undefined luego retorna true, si uno es un número y el otro un string convierte a string el número y luego compara, si uno de los valores es true convierte este a 1 y luego compara, si uno de los valores es false convierte este a 0 (cero) y luego compara.

Algunas comparaciones:

— — —

- `if (true==1) //verdadero`
- `if (true===1) //falso`
- `if (false==0) //verdadero`
- `if (false===0) //falso`

EJERCICIOS ADICIONALES PROPUESTOS

— — —





**¡MUCHAS
GRACIAS!**