JSX y ReactDOM

JSX trabaja en conjunto con React para crear interfaces de usuario dinámicas y reactivas. JSX es una extensión de sintaxis que nos permite escribir código HTML en nuestros archivos JavaScript. Con JSX, podemos definir componentes de React utilizando una sintaxis similar a HTML.

Cuando escribimos código JSX en nuestros archivos JavaScript, React se encarga de compilar ese código en llamadas a funciones de JavaScript. Estas llamadas a funciones representan los elementos de la interfaz de usuario que se mostrarán en el navegador.

En resumen, JSX nos permite escribir código HTML en nuestros archivos JavaScript, y React se encarga de compilar ese código en llamadas a funciones y de manejar la actualización eficiente de la interfaz de usuario utilizando el Virtual DOM.

JSX y ReactDOM

Vamos a implementar JSX en nuestra primera aplicación que creamos anteriormente:

index.js

react_app > src > index.js > ...

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const JSX = <h1>¡Hola Mundo!</h1>;
5  const root = ReactDOM.createRoot(document.getElementById('root'));
6  root.render(JSX);
```

Todo se ve exactamente igual, pero la ventaja es que dentro de JSX podemos trabajar con expresiones de JavaScript y de esa forma estaríamos creando una plantilla y después podemos insertar la misma en el DOM de nuestra página.

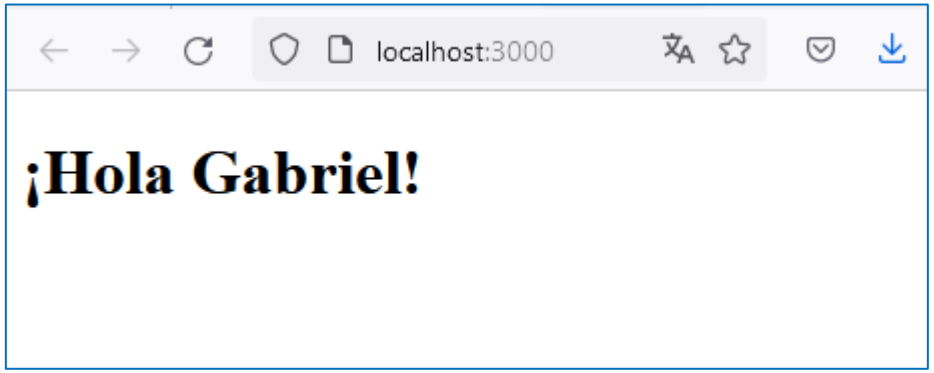
JSX y ReactDOM

De esta forma, si creamos una variable nombre y guardamos un valor en ella, podemos imprimirla en pantalla gracias a JSX, es una forma más dinámica de trabajar.

index.js

react_app > src > index.js > ...

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const nombre = "Gabriel";
5  const JSX = <h1>¡Hola {nombre}!</h1>;
6  const root = ReactDOM.createRoot(document.getElementById('root'));
7  root.render(JSX);
```



JSX y ReactDOM

Sin embargo, para trabajar con JSX hay que seguir ciertas reglas:

1. **Sintaxis de etiquetas:** Deben tener una etiqueta de apertura y una etiqueta de cierre, o pueden ser etiquetas auto-cerradas si no tienen contenido.
2. **Expresiones en llaves:** Puedes incluir expresiones JavaScript dentro de llaves {} en el cuerpo de las etiquetas JSX y mostrar variables, realizar cálculos o llamar a funciones.
3. **Clases y atributos:** Para agregar clases CSS a un elemento JSX, se utiliza el atributo "className" en lugar de "class". Los atributos se especifican utilizando la sintaxis de atributo HTML, como "id" o "style".
4. **Estilos:** Para agregar estilos debemos colocar el atributo style igualado a dos (2) llaves de apertura y dos (2) llaves de cierre “{{}}"
5. **Trabajo con varias etiquetas JSX:** Para trabajar con varias sentencias JSX debemos encerrarlas dentro de un contenedor.

JSX y ReactDOM

A continuación, aplicaremos JSX con variables y estilos básicos para poder visualizar las reglas anteriormente expuestas.

index.js

react_app > src > index.js > ...

```
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3
4   const nombre = "Clase";
5   const color = "blue"
6   const JSX = (
7     <>
8       <h1 style={{color:color}}>¡Hola {nombre}!</h1>
9       <p>Te doy la Bienvenida a esta Experiencia de Aprendizaje</p>
10    </>
11  );
12  const root = ReactDOM.createRoot(document.getElementById('root'));
13  root.render(JSX);
```

Condicionales en JSX

Para entender los condicionales en JSX haremos la simulación de un inicio de sesión en función de una constante llamada "sesion", dependiendo del valor de la misma, se mostrará un contenido diferente en pantalla.

index.js

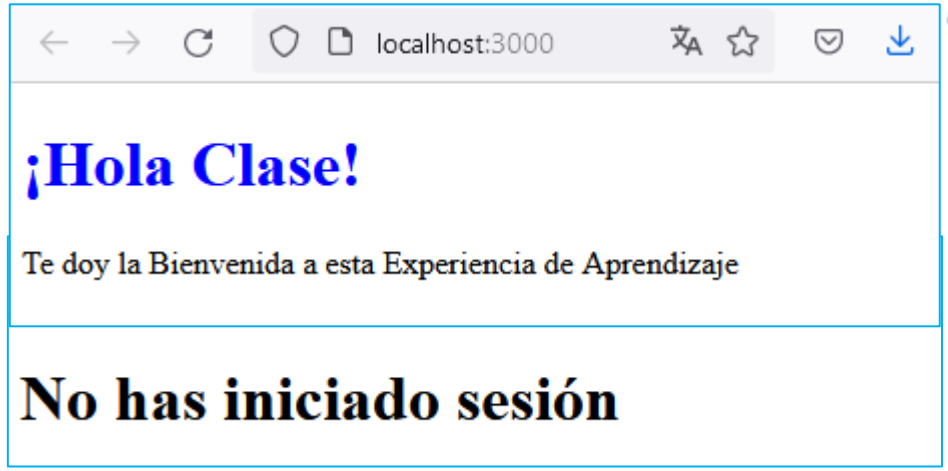
C: > Users > Gabriel Paredes > Desktop > diplomado_react > react_app > src > index.js > ...

```
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3
4   const nombre = "Clase";
5   const color = "blue"
6   const sesion = true;
7
8   const JSX = (
9     <>
10      <h1 style={{color:color}}>¡Hola {nombre}!</h1>
11      <p>Te doy la Bienvenida a esta Experiencia de Aprendizaje</p>
12    </>
13  );
14
15  const root = ReactDOM.createRoot(document.getElementById('root'));
16  root.render(JSX);
```

Condicionales en JSX

Condicionales en JSX con Funciónes Tipo Flecha: Haremos uso de una función tipo flecha llamada "verificarSesion" en la que preguntaremos por el valor de la constante "sesion". Si la misma vale verdadero, retornaremos la constante "JSX" que guarda el contenido del sitio, si no, retornamos un mensaje indicando que no se ha iniciado sesión. Para finalizar, en la línea 24, enviaremos a la pantalla el resultado de la función flecha, es decir, lo que la función retorne.

```
15  const verificarSesion = (sesion) => {
16    if(sesion === true){
17      return JSX;
18    } else {
19      return <h1>No has iniciado sesión</h1>
20    }
21  }
22
23  const root = ReactDOM.createRoot(document.getElementById('root'));
24  root.render(verificarSesion(sesion));
```

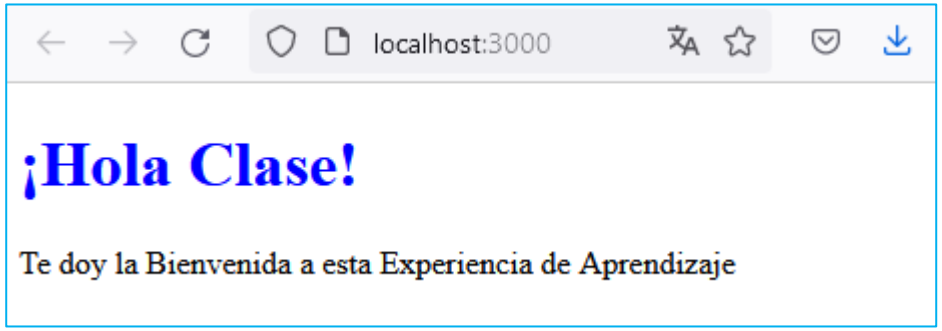


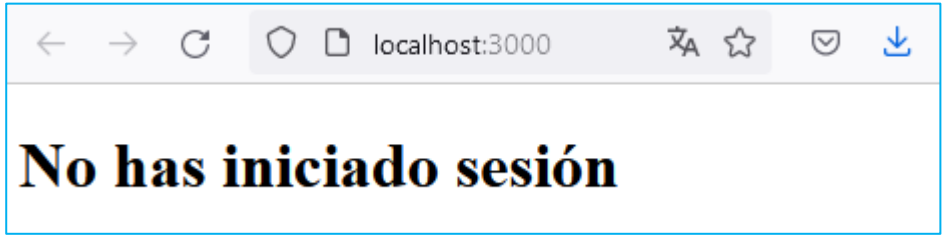
*En las funciones flecha, se hace la declaración con el nombre de la función, luego se pone un signo de igual (=), seguido de unos paréntesis que contendrán los parámetros necesarios para que la función trabaje, luego se coloca la flecha, se abren llaves y dentro de ellas colocamos el contenido propio de la función.

Condicionales en JSX

Condicionales en JSX con Operadores Ternarios: Dentro del contenido de la constante JSX colocaremos un operador ternario que mostrará el contenido de la web en caso de ser verdadero o mostrará el mensaje indicativo de que no se ha iniciado sesión en caso de ser falso. Para finalizar, enviaremos a renderizar el contenido de la variable JSX en la línea 22:

```
8  const JSX = (
9    <>
10     {sesion === true ?
11     <>
12       <h1 style={{color:color}}>¡Hola {nombre}!</h1>
13       <p>Te doy la Bienvenida a esta Experiencia de Aprendizaje</p>
14     </>
15     :
16     <h1>No has iniciado sesión</h1>
17     }
18   </>
19 );
20
21 const root = ReactDOM.createRoot(document.getElementById('root'));
22 root.render(JSX);
```





Condicionales en JSX

Condicionales en JSX en base a la definición de constantes: Crearemos la constante "clase" (línea 7) y luego dentro del contenido de la constante JSX colocaremos un condicional que funcionará preguntando si la constante "clase" está definida: En caso de estar definida, mostrará un texto y de lo contrario no mostrará nada (línea 15).

```
7   const clase = "Diplomado de Programación";
8
9   const JSX = (
10    <>
11      {sesion === true ?
12      <>
13        <h1 style={{color:color}}>¡Hola {nombre}!</h1>
14        <p>Te doy la Bienvenida a esta Experiencia de Aprendizaje</p>
15        {clase && <p>Estamos en clase del {clase}</p>}
16      </>
17      :
18      <h1>No has iniciado sesión</h1>
19      }
20    </>
21  );
22
23  const root = ReactDOM.createRoot(document.getElementById('root'));
24  root.render(JSX);
```

Arreglos en JSX

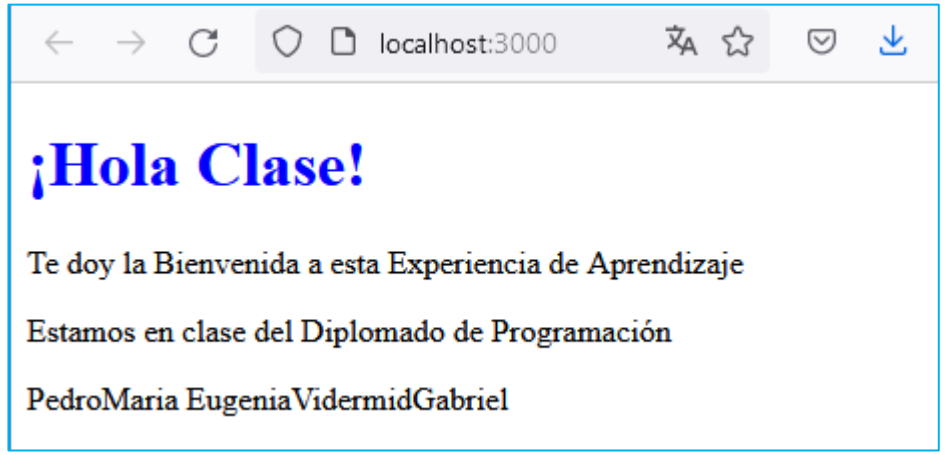
Para ver cómo trabajar con Arreglos en JSX, mostraremos en pantalla el nombre de los instructores del Diplomado. Por ello crearemos el arreglo "instructores" en la línea 8:

```
7   const clase = "Diplomado de Programación";
8   const instructores = ["Pedro", "Maria Eugenia", "Vidermid", "Gabriel"];
```

Asimismo, en la línea 18 colocaremos entre llaves el arreglo creado anteriormente y de esa forma ya podemos imprimirlo en pantalla:

```
17          {clase && <p>Estamos en clase del {clase}</p>}
18          {instructores}
19        </>
```

El detalle es que el resultado que tenemos, no está ordenado de la mejor forma, salen los nombres, pero uno al lado del otro y sin espaciado. Lo arreglaremos a continuación.



Arreglos en JSX

Para conseguir un resultado visual más agradable, crearemos una encabezado <h4> con el texto "Estos son los profesores", seguidamente haremos una lista <ul> y dentro de ella, usaremos el arreglo "instructores" con el método ".map", así ingresamos a cada una de las posiciones del arreglo en cuestión y por cada una de ellas ejecutaremos una función tipo flecha que imprime un elemento de lista <li> que muestre el nombre de cada instructor:

```
18        <h4>Estos son los profesores:</h4>
19        <ul>
20        {instructores.map((instructor, index) => {
21          return <li key={index}>{instructor}</li>
22        })}
23        </ul>
24      </>
```

"instructor" e "index" en la línea 20, son parámetros que recibe la función flecha en cada iteración: "instructor" es el que usaremos para mostrar el contenido de cada posición del arreglo e "index" es el que usaremos para mostrar la posición del arreglo en cada item del listado, en el valor del atributo key para evitar avisos en la consola del navegador.