UD 2. Fundamentos de JSX

1. Primeros pasos

JSX (JavaScript XML) es una **extensión de sintaxis para JavaScript** que permite escribir estructuras similares a HTML directamente en el código. Aunque puede parecer "HTML en JS", en realidad JSX se compila a llamadas a React.createElement, y tiene reglas propias que conviene dominar.

Este punto se centra en aprender y dominar JSX con profundidad, comprendiendo no solo su sintaxis, sino también sus implicaciones, buenas prácticas y casos avanzados.

JSX permite describir qué se debe renderizar en React de forma declarativa, usando una sintaxis parecida a HTML incrustada en JavaScript.

```
const elemento = <h1>Hola, React</h1>;
```

Esto se transforma (tras el proceso de compilación con Babel) en:

```
const elemento = React.createElement('h1', null, 'Hola, React');
```

Ventajas:

- Mayor legibilidad.
- Facilita composición de componentes.
- Más expresivo y cercano al DOM real.

ATENCIÓN: JSX no es obligatorio en React, pero sí altamente recomendable por legibilidad y productividad.

2. Sintaxis y reglas

JSX tiene una sintaxis y algunas diferencias clave con respecto a HTML que debemos conocer.

- **Debe devolver un único elemento raíz**

```
// Incorrecto
return (
<h1>Hola</h1>
<p>Mundo</p>
);
```

```
// Correcto
return (
<> // También con div
<h1>Hola</h1>
<p>Mundo</p>
</>
);
```

Se pueden usar fragmentos (<>...</>) como envoltorio sin añadir elementos extra al DOM, o bien usar la etiqueta div.

- **Atributos en camelCase**

En JSX, algunos atributos HTML tienen nombres diferentes a sus equivalentes en el DOM debido a las convenciones de nomenclatura de JavaScript y a la forma en que React maneja los eventos y las propiedades de los elementos. La principal razón para esta transformación es evitar conflictos con palabras reservadas de JavaScript y mantener la coherencia con la sintaxis del lenguaje. Veamos cuáles:

- class → className
- for → htmlFor
- tabindex → tabIndex
- onclick → onClick

```
<input className="campo" htmlFor="nombre" />
<button onClick={handleClick}>Enviar</button>
```

- **Inserción de expresiones JS con { }**

```
const nombre = "Josemi";
return <h1>Hola, {nombre}</h1>;
```

Podemos usar cualquier expresión JS válida:

```
<h2>{nombres.join(", ")}</h2>
<h3>{5 * 3}</h3>
<h4>{usuario?.nombre || "Invitado"}</h4>
```

- **No se permite if directamente en el return en JSX (ni entre { })**

```
// Correcto con operador ternario
{logeado ? <p>Bienvenido</p> : <p>Inicia sesión</p>}
// También con &&
{logeado && <p>Contenido exclusivo</p>}
```

- **Elementos auto-cerrados**: JSX obliga a cerrar todos los elementos:

```
// Correcto
<img src="foto.jpg" alt="foto" />
<input type="text" />
```

3. JSX es Javascript: expresiones, no sentencias

JSX permite **expresiones de JavaScript** dentro de {}, pero **no sentencias** (if, for, while, etc.).

```
return (
<div>
{usuario ? <p>Hola</p> : <p>Invitado</p>}
</div>
);
```

Si necesitas lógica más compleja, prepárala antes del return:

```
let mensaje;
if (edad > 18) {
mensaje = <p>Eres mayor de edad</p>;
} else {
mensaje = <p>Eres menor</p>;
}
return <div>{mensaje}</div>;
```

Una de las mayores potencias de JSX es su **naturaleza programable.** Ejemplo para **mapear arrays:**

```
const alumnos = ["Ana", "Luis", "Marta"];
return (
<ul>
{alumnos.map((nombre, i) => (
<li key={i}>{nombre}</li>
```

```
))}
</ul>
);
```

Siempre que uses .map() en JSX, usa la key para identificar los elementos y evitar re-renderizados incorrectos. JSX distingue elementos HTML y componentes React según el nombre de la etiqueta:

- Minúsculas: se interpreta como HTML (, <p>, <img>).
- Mayúsculas: se interpreta como componente React (<MiBoton>, <Cabecera>).

Ejemplo:

```
function MiBoton() {
return <button>Click</button>;
}
function App() {
return (
<div>
<MiBoton /> {/* Componente */}
<button>Normal</button> {/* HTML */}
</div>
);
}
```

Recordatorio de buenas prácticas con JSX

- Usa fragmentos (<>...</>) para evitar innecesarios.
- Extrae lógica fuera del JSX cuando sea compleja.
- Usa map() y ternarios para condicionales sencillos, pero extrae componentes si se complican.
- Recuerda siempre las key al renderizar listas.

4. Componentes en React

React se basa en una arquitectura **componentizada**: la interfaz se divide en piezas reutilizables llamadas **componentes**, las cuales encapsulan lógica, estructura y comportamiento; es decir, pueden contener HTML, CSS y JS todo en uno.

A partir de React 16.8 y especialmente en React 18/19, el uso de **componentes funcionales** con **Hooks** se ha convertido en el enfoque recomendado y moderno, reemplazando en la práctica a los antiguos componentes de clase.

Un "componente funcional" es simplemente una **función** de JavaScript que recibe propiedades (props) como argumento, procesa y devuelve JSX (con CSS).

Estructura básica:

```
function NombreComponente(props) {
return <div>{/* JSX */}</div>;
}
```

O también con funciones flecha (muy habitual):

```
const NombreComponente = (props) => {
return <div>{/* JSX */}</div>;
};
```

Comencemos con un ejemplo sencillo:

Versión inicial

```
function Saludo() {
return <h1>Hola, React</h1>;
}
```

Y lo usamos desde App.jsx:

```
function App() {
return <Saludo />;
}
```

Recuerda que los componentes siempre deben empezar con **mayúscula**. Y que, si devolvemos más de una línea, todo el bloque irá entre paréntesis. Lo verás en el siguiente ejemplo.

4.1 Tu primer componente

Vamos a diseñar el **primer componente** para la aplicación de productos informáticos. Será un **listado de productos** que muestre:

- **Código**
- **Descripción**
- **Precio**
- **Disponible** (sí/no)

Que está ubicado en src/data/catalog.js

src/data/catalog.js

```
export const catalog = [
  { id: 'lap-001', nombre: 'Lenovo ThinkPad E14', precio:
799.99, enStock: true },
  { id: 'mon-101', nombre: "LG UltraGear 27'' 144Hz", precio:
279.0, enStock: true },
  { id: 'per-501', nombre: 'Logitech MX Keys', precio: 109.0,
enStock: false },
];
```

La ubicación del fichero será dentro de la carpeta **src/components/** del proyecto que generamos antes:

src/components/ProductList.js

```
import React from 'react';

export default function ProductList({ products }) {
  if (!products || products.length === 0) {
    return <p className="text-center text-muted mt-3">No
hay productos disponibles.</p>;
  }

  return (
    <div className="container mt-4">
      <h2 className="mb-3">Listado de Productos
Informáticos</h2>
      <table className="table table-striped table-bordered">
        <thead className="table-dark">
          <tr>
            <th>Código</th>
            <th>Descripción</th>
            <th>Precio (€)</th>
            <th>Disponible</th>
          </tr>
        </thead>
        <tbody>
          {products.map((p) => (
            <tr key={p.id}>
              <td>{p.id}</td>
              <td>{p.nombre}</td>
              <td>{p.precio.toFixed(2)}</td>
              <td>
                <span className={`badge ${p.enStock ? 'bg-
success' : 'bg-danger'}`}>
                  {p.enStock ? 'Sí' : 'No'}
                </span>
```

```
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
 );
}
```

Editamos el fichero App.js

src/App.js:

```
import React from 'react';
import ProductList from './components/ProductList';
import { catalog } from './data/catalog';

export default function App() {
  return (
    <div className="app">
      <ProductList products={catalog} />
    </div>
  );
}
```

4.2 Bootstrap

Para integrar Bootstrap en tu proyecto React realizaremos los siguientes pasos

En la raíz del proyecto, ejecuta:

```
C:\productos\npm install Bootstrap
```

Esto descargará la última versión de Bootstrap en node_modules.

Para importar Bootstrap en la aplicación editaremos el fichero src/index.js o src/main.jsx

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app,
pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more:
https://bit.ly/CRA-vitals
reportWebVitals();
```