React I

Introducción a React, JSX

Javier Ribal del Río

2026-01-16

Table of contents

1 Sintaxis JSX **2**

1.1 ¿Qué es la sintaxis JSX? . . . 2

1.2 ¿Qué nos ofrece la sintaxis JSX? . . . 2

1.2.1 Almacenar una etiqueta como una variable . . . 2

1.2.2 Renderizar etiquetas dentro de etiquetas . . . 2

1.2.3 Renderizar con el map . . . 3

1.2.4 Renderizar con el map . . . 3

2 React **3**

2.1 Motivación . . . 3

2.2 Bases de React . . . 3

2.3 Componente React . . . 3

2.4 Props en React . . . 4

2.4.1 Ejemplo 1: uso de `props` (forma explícita) . . . 4

2.4.2 Ejemplo 2: props desestructuradas (sintaxis JavaScript) . . . 4

2.5 Eventos en React . . . 4

2.5.1 Ejemplo básico de evento . . . 5

2.5.2 Paso de funciones como eventos . . . 5

2.5.3 Eventos más habituales en React . . . 5

2.6 Ciclo de vida de los componentes y renderizado en React . . . 6

2.6.1 El árbol de componentes . . . 6

2.6.2 Proceso de renderizado . . . 7

2.6.3 Ciclo de vida (visión simplificada) . . . 7

2.6.4 Relación con el DOM . . . 7

3 Creación de una ReactJS app **7**

3.1 ¿Para qué sirve un framework? 8

3.2 Framework y React 8

3.3 Creación de la aplicación 8

Contenido

- Sintaxis JSX
- Introducción a React
  - Modulo React
  - Componente React
  - Interacción React - Árbol del DOM
- Framework
  - Concepto de Framework
  - Uso de react a través de vite
  - Perder el miedo a los frameworks

1 Sintaxis JSX

1.1 ¿Qué es la sintaxis JSX?

- **JSX (JavaScript XML)** es una extensión de JavaScript utilizada en **React**.
- Permite escribir estructuras similares a HTML directamente dentro del código JavaScript.
- Facilita la creación y lectura de interfaces de usuario, integrando lógica y presentación.
- Aunque su apariencia es HTML, JSX se transpila a JavaScript puro antes de ejecutarse en el navegador.
- *Forma cómoda de almacenar etiquetas html como variables de JS*

1.2 ¿Qué nos ofrece la sintaxis JSX?

1.2.1 Almacenar una etiqueta como una variable

```
const title = <h1>Web del TC de SW</h1>;

const driveIndication = (drives)? <p>Puede conducir</p> : <p>No puede conducir</p>;
```

De esta forma la variable `title` se corresponde con una etiqueta que contiene el título. Sin embargo, como podemos podemos *utilizar* está etiqueta una vez la hemos definido como una variable.

1.2.2 Renderizar etiquetas dentro de etiquetas

```
const title = <h1>Web del TC de SW</h1>;

const page = <div>{title}</div>
```

De esta forma estámos encapsulando la etiqueta `h1` dentro del `div`.

1.2.3 Renderizar con el map

```
const profeAsi= [["Rosa", "Macro I"], ["Gabriela", "FSO"],["Yolanda","LTP"], ["Ferrán", "EM"], ["Pepe",

// Array de li Asignatura: nombre profesor
const itemsProfAsi =  profeAsi.map(x => <li>{x[1]}: {x[0]}</li>);

const listaProfesoresAsignatura = <ul>{itemsProfAsi}</ul>;
```

1.2.4 Renderizar con el map

```
const profeAsi= [["Rosa", "Macro I"], ["Gabriela", "FSO"], ["Yolanda","LTP"], ["Ferrán", "EM"], ["Pepe",

const listaProfesoresAsignatura = <ul>
{profeAsi.map((x,i) => <li key={i}>{x[1]}: {x[0]}</li>)}
</ul>;
```

2 React

2.1 Motivación

Hasta el momento hemos trabajado únicamente con HTML5, el cual a pesar de ser muy eficaz, presenta el inconveniente de que no implementa ninguna clase de lógica. Es decir, si queremos repetir alguna estructura previamente definida deberemos copiarla y pegarla. Otro problema, de HTML puro es la complejidad que requiere el manejo del DOM, como ya se vio en la semana anterior.

Bajo este pretexto, introducimos el *framework* React diseñado por Meta.

2.2 Bases de React

React es un componente basado en la sintaxis JS y en la programación funcional introducida en ES6.

2.3 Componente React

Los componentes de React los definimos como unidades reutilizables y modulares que se implementan de esta forma: una función que devuelve un objeto de JSX

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

Podemos renderizar nuestro componente `Saludo` asumiendo que se comporta como una etiqueta html. Recordemos que no lo es.

```
<main>
  <Saludo />
</main>
```

Dentro de este componente podemos incluir lógica propia de JavaScript, de forma que al definir componente podemos dotarlo de lógica interna.

```
function DivNúmeroAleatorio() {

  let n = Math.random();
  return <div className="n-random">{n}</div>;
}
```

**Nótese que aunque JS utilice camelCase para el nombre de las funciones, con React utilizaremos PascalCase para definir los componentes puesto que a pesar de que se trata de funciones, trabajaremos con ellas como si fuesen clases.*

2.4 Props en React

Hasta ahora, los componentes definidos siempre devuelven el mismo contenido. Sin embargo, para que un componente sea realmente reutilizable, necesitamos poder pasarle información desde el exterior.

En React, esta información se transmite mediante **props** (*properties*).

Las props se definen como los **parámetros de entrada** de un componente y permiten personalizar su comportamiento o su contenido sin modificar su definición interna.

2.4.1 Ejemplo 1: uso de `props` (forma explícita)

```
function Saludo(props) {
  return <h1>Hola, {props.nombre}</h1>;
}
```

Uso del componente:

```
<Saludo nombre="Rosa" />
<Saludo nombre="Gabriela" />
```

2.4.2 Ejemplo 2: props desestructuradas (sintaxis JavaScript)

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

Uso del componente (idéntico al anterior):

```
<Saludo nombre="Rosa" />
<Saludo nombre="Gabriela" />
```

2.5 Eventos en React

En React, el manejo de eventos se basa en los eventos del DOM, pero utilizando una sintaxis propia de JSX y siguiendo el estilo de JavaScript.

2.5.1 Ejemplo básico de evento

```
function BotonSaludo() {

  function manejarClick() {
    alert("Hola desde React");
  }

  return (
    <button onClick={manejarClick}>
      Saludar
    </button>
  );
}
```

2.5.2 Paso de funciones como eventos

La función asociada al evento **no se ejecuta al renderizar el componente**, sino únicamente cuando ocurre el evento; es decir la pasamos como un ciudadano de primera clase.

```
<button onClick={manejarClick}>Saludar</button>
```

Incorrecto:

```
<button onClick={manejarClick()}>Saludar</button>
```

2.5.3 Eventos más habituales en React

React soporta la mayoría de los eventos del DOM, adaptados a la sintaxis JSX. A continuación se listan los **eventos más utilizados**, clasificados por tipo.

2.5.3.1 Eventos de ratón

- **onClick**
- onDoubleClick
- onMouseEnter
- onMouseLeave
- **onMouseMove**
- onMouseDown
- onMouseUp

2.5.3.2 Eventos de teclado

- onKeyDown
- onKeyUp
- **onKeyPress**

2.5.3.3 Eventos de formularios

- `onChange`
- `onSubmit`
- `onFocus`
- `onBlur`
- `onInput`
- `onReset`

2.5.3.4 Eventos de carga y ciclo del elemento

- `onLoad`
- `onError`

2.5.3.5 Eventos de selección y portapapeles

- `onSelect`
- `onCopy`
- `onCut`
- `onPaste`

2.5.3.6 Eventos táctiles (dispositivos móviles)

- `onTouchStart`
- `onTouchMove`
- `onTouchEnd`
- `onTouchCancel`

2.6 Ciclo de vida de los componentes y renderizado en React

En React, los componentes **no se renderizan directamente sobre el DOM real**, sino que siguen un proceso intermedio que permite optimizar las actualizaciones de la interfaz.

La idea clave es que **la interfaz se organiza como un árbol de componentes**, análogo al **árbol del DOM**, pero a un nivel de abstracción superior.

2.6.1 El árbol de componentes

Cuando una aplicación React se ejecuta:

- Cada componente genera otros componentes o elementos JSX.
- React construye un **árbol de componentes**, donde:
  - El componente raíz suele ser `App`
  - Cada componente hijo ocupa una rama del árbol

Este árbol de componentes **se corresponde conceptualmente** con el **árbol del DOM**, aunque no es el DOM real.

2.6.2 Proceso de renderizado

El proceso general es el siguiente:

1. React ejecuta un componente (función).
2. El componente devuelve JSX.
3. El JSX se transforma en una representación interna del árbol.
4. React compara este árbol con el anterior.
5. Solo se actualizan en el DOM real los nodos que han cambiado.

De esta forma, React **minimiza el acceso directo al DOM**, que es una operación costosa.

2.6.3 Ciclo de vida (visión simplificada)

Desde un punto de vista conceptual, un componente pasa por tres fases:

1. **Montaje**
   - El componente se crea y se inserta en el DOM.
2. **Actualización**
   - El componente se vuelve a renderizar cuando cambian sus datos.
3. **Desmontaje**
   - El componente se elimina del DOM.

En componentes funcionales modernos, este ciclo se gestiona de forma implícita mediante el renderizado y, más adelante, mediante *hooks*.

2.6.4 Relación con el DOM

- En JavaScript puro:
  - Se modifica el DOM directamente.
- En React:
  - Se describe cómo debe ser la interfaz.
  - React decide el cuándo y el cómo modificar el DOM.

El desarrollador no recorre ni manipula el árbol del DOM, sino que trabaja sobre el **árbol de componentes**.

3 Creación de una ReactJS app

El desarrollo de aplicaciones web modernas requiere coordinar múltiples elementos: estructura del proyecto, gestión de dependencias, compilación del código y ejecución en el navegador.

Un framework de desarrollo proporciona un entorno de trabajo preconfigurado que organiza estos elementos y establece una forma estándar de trabajar.

3.1 ¿Para qué sirve un framework?

Un framework permite al desarrollador:

- Evitar configuraciones manuales complejas
- Seguir una estructura común de proyecto
- Automatizar tareas repetitivas
- Centrarse en la lógica de la aplicación y no en la infraestructura

En lugar de partir de un proyecto vacío, el framework ofrece una base funcional sobre la que desarrollar.

3.2 Framework y React

React se encarga de definir cómo se construye la interfaz, pero no gestiona por sí mismo:

- La estructura de archivos
- El arranque de la aplicación
- La compilación del código
- El servidor de desarrollo

Por ello, React se apoya en un framework que orquesta el entorno de ejecución.

3.3 Creación de la aplicación

Para crear una nueva aplicación React utilizando un framework, se ejecuta el siguiente comando en la terminal:

```
npm create vite@latest
```

Este comando genera:

- Una estructura inicial del proyecto
- La configuración necesaria para trabajar con React
- Un entorno listo para el desarrollo