



FORMACIÓN PROFESIONAL  
MONTECASTELO

## React (I)

Desarrollo Web en Entorno Cliente

Ciclo Superior de Desarrollo de Aplicaciones Web

2023/2024

# Introducción a React

- React es una biblioteca de JavaScript utilizada para construir interfaces de usuario interactivas y eficientes.
- JavaScript estándar, se enfoca en manipular el DOM directamente. En cambio, React utiliza una abstracción llamada el "Virtual DOM".
  - Esto significa que React realiza cambios en una representación virtual de la interfaz de usuario en lugar de actualizar directamente el DOM del navegador, lo que hace que las actualizaciones sean más rápidas y eficientes.



# Configuración del entorno

- Para comenzar a trabajar con React, necesitamos configurar nuestro entorno de desarrollo.
- Asegúrate de tener Node.js y npm instalados en tu sistema.
- Luego, podemos usar la herramienta create-react-app para crear un nuevo proyecto de React de manera sencilla.



# Configuración del entorno

- Instalación Node.js en Linux:

```
curl -fsSL https://deb.nodesource.com/setup_1ts.x  
| sudo -E bash -  
sudo apt-get install -y nodejs npm
```

- Comprobamos instalación con:

```
node -v  
npm -v
```



# Configuración del entorno

- Instalación Node.js en Windows:
  - Navega a la web de Node.js, descarga y ejecuta el instalador LTS para Windows.

<https://nodejs.org/>

- Comprobamos instalación con:

```
node -v
```

```
npm -v
```



# Configuración del entorno

- Creación de aplicación de React:
- Abre una ventana de línea de comandos en la ubicación donde deseas crear tu proyecto y crea el proyecto con:
  - `npx create-react-app nombre-de-tu-app`
- Dirígete al directorio e inicia el servidor de desarrollo de React:
  - `cd nombre-de-tu-app`
  - `npm start`



# Configuración del entorno

- Los pasos anteriores inician el servidor de desarrollo y abre tu aplicación de React en el navegador.
- A partir de aquí, puedes comenzar a editar los archivos de tu aplicación y ver los cambios en tiempo real mientras desarrollas.



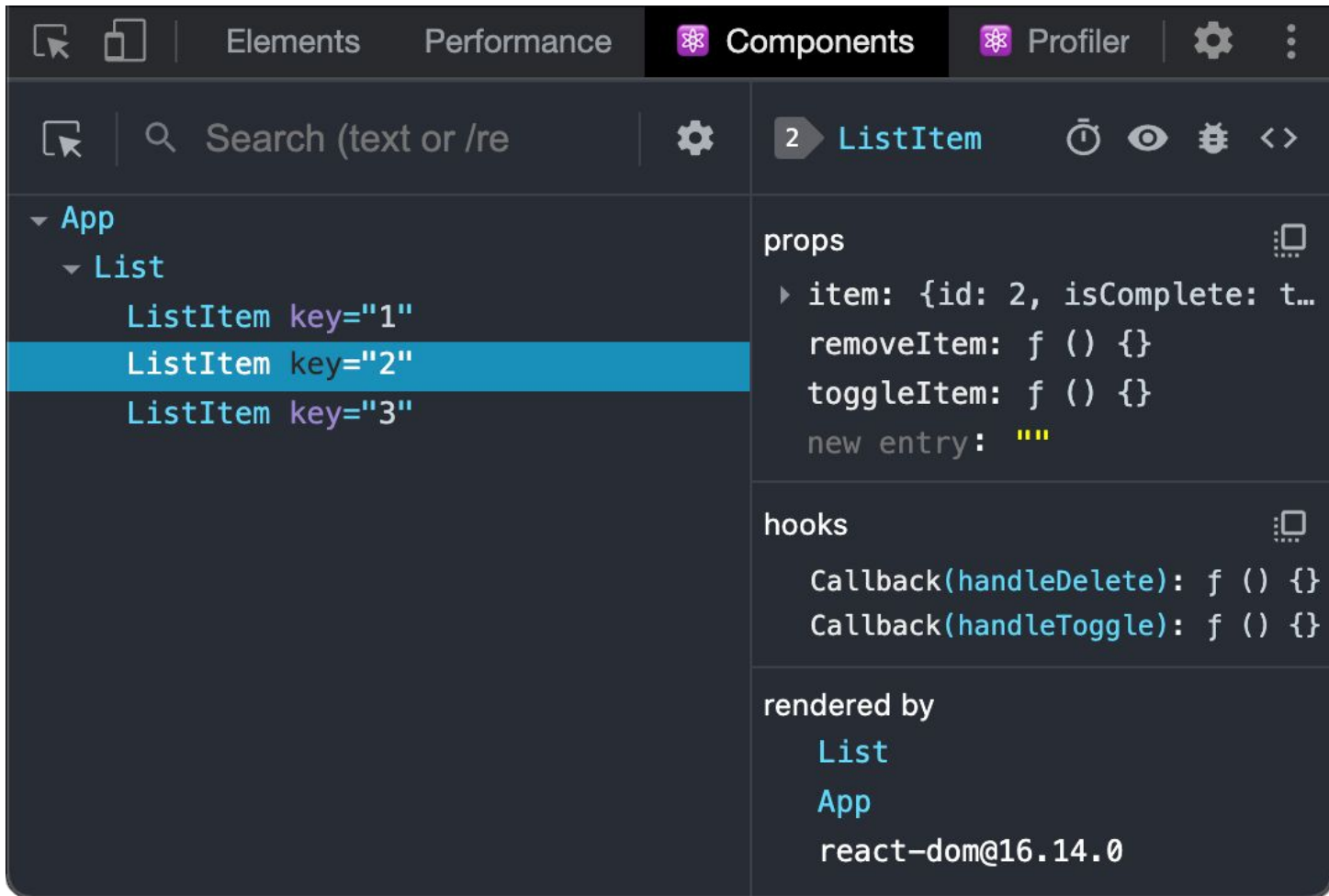
# Configuración del entorno

- Instalación de herramientas de desarrollo sobre el navegador (chrome, Firefox, Edge) como [complemento](#).
- Permite inspeccionar **componentes**, propiedades (**props**) y estados (**states**), así como identificar problemas de rendimiento.





# Configuración del entorno



# Componentes y JSX

- En React, todo es un componente. Los componentes son bloques de construcción reutilizables que representan partes de la interfaz de usuario.
- Pueden ser componentes simples, como un botón, o componentes más complejos, como una barra de navegación.



# Componentes y JSX

```
import React from 'react';  
function MyButton() {  
  return (  
    <button>Soy un botón</button>  
  );  
}  
export default MyButton;
```



# Componentes y JSX

- En el ejemplo anterior se define un componente de función llamado Mybutton que renderiza un elemento button con el texto "*Soy un boton*".
- Todas las aplicaciones de React están hechas a partir de componentes.
- Estos componentes pueden ser tan pequeños como un botón, o tan grande como una página.



# Componentes y JSX

- Los componentes se integran unos en otros de la siguiente manera:

```
function MyApp() {  
  return (  
    <div>  
      <h1>Bienvenido a mi aplicación</h1>  
      <MyButton />  
    </div>  
  );  
}
```



# Componentes y JSX

- React utiliza JSX (JavaScript XML) para definir la estructura de los componentes.
- JSX es una extensión de JavaScript que permite escribir código similar a HTML dentro de JavaScript.



# Componentes y JSX

- Los nombres de los componentes de React siempre deben comenzar con mayúscula, mientras las etiquetas HTML deben estar minúsculas.
- JSX es restrictivo para diferenciar el marcado de componentes y JS.




# Componentes y JSX

- En JSX, hay que cerrar las etiquetas simples. Ejemplo: `<br />`
- Un componente no puede devolver múltiples etiquetas sino que debe envolverlas en un sólo `<div>...</div>` o un envoltorio vacío `<>...</>`






# Componentes y JSX



```
function AboutPage() {  
  return (  
    <h1>Acerca de</h1>  
    <p>Hola.<br />¿Cómo vas?</p>  
  );  
}
```



```
function AboutPage() {  
  return (  
    <h1>Acerca de</h1>  
    <div>  
      <p>Hola.<br />¿Cómo vas?</p>  
    </div>  
  );  
}
```

```
function AboutPage() {  
  return (  
    <div>  
      <h1>Acerca de</h1>  
      <p>Hola.<br />¿Cómo vas?</p>  
    </div>  
  );  
}
```

```
function AboutPage() {  
  return (  
    <>  
      <h1>Acerca de</h1>  
      <p>Hola.<br />¿Cómo vas?</p>  
    </>  
  );  
}
```



# Componentes y JSX

- JSX te permite poner marcado dentro de JavaScript con llaves que permiten «escapar» hacia JS e incrustar por ejemplo una variable.

```
var user = {name: "Juan", edad: 21}  
  
...  
return (  
  <h1>  
    {user.name}  
  </h1>  
)
```



# Renderización de componentes

- El renderizado es el proceso actualización del frontend para que coincida con el estado actual de la aplicación.
- React representa la interfaz de usuario como un árbol de componentes y, cuando cambia el estado de la aplicación, React realiza una comparación virtual entre el árbol anterior y el nuevo para determinar qué partes deben actualizarse en el DOM real.



# Renderización de componentes

- React utiliza un algoritmo de "reconciliación" para minimizar la cantidad de cambios necesarios en el DOM, lo que hace que la actualización de la interfaz de usuario sea eficiente y rápida.



# Renderización de componentes

- Navegando hasta la raíz de una aplicación React (index.js), encontramos la función **ReactDOM.render()** que es el punto de entrada principal para renderizar un componente de React en el DOM.
- Tiene dos argumentos:
  - el elemento de React que deseas renderizar
  - el elemento del DOM en el que deseas que se renderice.



# Renderización de componentes

```
import React from 'react';
import ReactDOM from 'react-dom';
function MiComponente() {
  return <h1>Hola, Mundo</h1>;
}
const elementoDOM = document.getElementById('root');
ReactDOM.render(<MiComponente />, elementoDOM);

// El componente “MiComponente” se renderizará en el
elemento HTML con id=”root” (index.html)
```



# Renderización de componentes

- Se puede renderizar de manera condicional los componentes, mezclando React con JS:

```
let content;  
if (isLoggedIn) content = <AdminPanel />;  
else content = <LoginForm />;  
return (  
  <div>  
    {content}  
  </div>  
>);
```



# Renderización de componentes

- Cuando los **componentes contienen listas**, la renderización es un poco particular, ya que hay que hacer uso de la función JS `.map()` y añadir un atributo `key` a los elementos `<li>` con un valor único.
- Este atributo `key`, ha de ser una cadena o un número que identifique ese elemento de forma única entre sus hermanos.





# Renderización de componentes

- De esta manera React maneja dinámicamente la inserción, eliminación o reordenación ante un cambio de estado del componente:

```
const listItems = products.map(product =>
  <li key={product.id}>
    {product.title}
  </li>
);
return (
  <ul>{listItems}</ul>
);
```



# Props y State

- Los componentes pueden recibir datos externos llamados props. Esto permite que los datos fluyan de un componente padre a un componente hijo.

```
function Saludo(props) {  
    return <div>Hola, {props.nombre}</div>;  
}  
function App() {  
    return <Saludo nombre="Juan" />;  
}
```



# Props y State

- En el ejemplo anterior, el componente *Saludo* recibe la prop *nombre* y muestra un saludo personalizado.
- Además de **props**, los componentes pueden tener un **state**, que es un objeto que representa la información interna del componente.



# Props y State

- El state es la información que se desea que el componente “recuerde”.
- Para utilizarlo hay que importar useState de React: `import { useState } from 'react';`
- Después declarar una variable de estado dentro del componente que contendrá el estado inicial, el estado actual y la función que modifica el estado.



# Props y State

```
function MyButton() {  
  const [count, setCount] = useState(0);  
  // ...  
}
```

- Al declarar la variable de estado dentro del componente estamos declarando lo siguiente:
  - El estado actual: count
  - La función que permite actualizar el estado: setCount
  - El estado inicial: 0, en useState(0)



# Props y State

```
import { useState } from 'react';  
function MyButton() {  
  const [count, setCount] = useState(0);  
  
  function handleClick() {  
    setCount(count + 1);  
  }  
  
  return (  
    <button onClick={handleClick}>  
      Hiciste clic {count} veces  
    </button>  
  );  
}  
export default MyButton;
```



# Props y State

- Cuando el **estado cambia**, React **vuelve a renderizar** el componente.
- Esto es importante, porque si cambia un prop el componente no se renderizaría de nuevo.
- En el ejemplo anterior, sólo cuando cambia count (haciendo click en el botón), se “recargará” el componente.



# Declaración de componentes

- En React, puedes crear componentes como funciones o clases.
- Los componentes funcionales son más simples y se usan para casos sencillos, mientras que los componentes de clase ofrecen más funcionalidades.
- Los componentes pueden ser reutilizables. Por ejemplo, puedes tener un componente *Boton* que renderice un botón con estilos predefinidos y funcionalidad.





# Declaración de componentes

- Componente Funcional:

```
function Titulo(props) {  
  return <h1>{props.texto}</h1>;  
}
```



# Declaración de componentes

- Componente de Clase:

```
class Titulo extends React.Component {  
  render() {  
    return <h1>{this.props.texto}</h1>;  
  }  
}
```



# Declaración de componentes

- Se pueden declarar componentes en archivos diferentes e importarlos desde otros archivos.
- Siempre que se quiera importar o exportar componentes, la notación es diferente según el número de componentes:
  - Un solo componente se utiliza el componente literalmente.  
Ej: `import Componente from ...;`
  - Varios componentes, se incluyen entre llaves, separados por comas.  
Ej: `import { Componente1, Componente2 } from ...;`



# Declaración de componentes

- En la **exportación** se utiliza el comando `export`:

```
export { Componente1, Componente2 };
```

- Cuando se exporta sólo un módulo, se le puede añadir la palabra reservada `default` que indica el valor predeterminado para ese módulo:

```
export default Componente;
```



# Declaración de componentes

- En la **importación** se utiliza el comando `import`:

```
import { Componente1, Componente2 }  
from 'archivo';
```

- Cuando se importa sólo un módulo y este ha sido precedido de `default`, se permite que sea importado con cualquier otro nombre:

```
import Componente from 'archivo'; //mismo  
import OtraManera from 'archivo'; //diferente
```



# Referencias

- [React en web oficial React developers](#)
- [React en w3schools](#)





FORMACIÓN PROFESIONAL  
MONTECASTELO