

ReactJS Developer

Módulo 1

Componentes

Componentes

Cuando desarrollamos una interfaz, necesitamos gestionar **cuatro aspectos clave: el código HTML, la funcionalidad JavaScript, los estilos CSS y la información mostrada.**

De estos cuatro aspectos, salen **tres principios** del desarrollo en ReactJS.

1. **Pensar en componentes (HTML y JS):** un **componente es una pieza de código reutilizable que representa una porción de la interfaz.** Podemos pensar en los componentes como *“HTML en función de un objeto JavaScript”*.

2. **Gestionar la apariencia (CSS):** al fin y al cabo, estamos desarrollando front-end y es imprescindible **cuidar todo detalle de la experiencia del usuario.**
3. **Gestionar el estado (información):** una interfaz React es una interfaz que cambia **cuando cambia la información contenida en ella sin recargar la página.**



Cómo dijimos antes, un componente es **un código *HTML* vinculado a un código *JS* que sirve para representar una porción de la interfaz.**

Que sea un código *HTML* vinculado a un código *JS* significa que el pedazo de código *HTML* de un componente cambiará **automáticamente y sin recargar la página cuando cambie su objeto *JavaScript* asociado.** Es decir, **reaccionará.**

De esta forma, puedes hacer cambios en la interfaz muy fácil: si cambias un dato, **de forma automática cambias la interfaz.** Eso se llama reactividad y los componentes nos sirven **para implementar esa característica.**

Para implementar esta reactividad es necesario introducir el concepto de **variable reactiva.** En React tenemos variables normales y también tenemos variables reactivas.

Una variable reactiva es como una variable común con la salvedad de que **al cambiar su valor, cambian las porciones de interfaz que usen dicho valor, automáticamente.**



Las variables reactivas se gestionan únicamente a través de componentes. Existen **dos objetos JavaScript** a los que reacciona todo componente React:

- El objeto llamado **Props**: son datos reactivos **externos** al componente, es decir, **que sirven para la comunicación con otros componentes**.
- El objeto llamado **State**: son datos reactivos **internos** al componente, es decir, que solo sirven para reactividad y funcionalidades **dentro de este componente**.

La forma de gestionar las props y el state varía **según la estructura del componente**.



Podemos agrupar los componentes de dos formas diferentes:

Según su propósito

- **Contenedores:** un contenedor es un **componente que no influye en la interfaz**, sino que sirve **para dar características a otros componentes**.
- **De presentación:** un componente de presentación es de lo que venimos hablando, **una porción de tu interfaz *HTML* que va a reaccionar a los cambios de un objeto *JavaScript* específico**.

Según su estructura

- **Componente funcional:** se crea **con una función**. Las props son su **argumento** y el state se **gestiona mediante *Hooks*** (lo veremos más adelante).
- **Componente basado en clases:** se crea **con una clase**. Las props son el **argumento al constructor** y el state se lee e **inicializa con `this.state`** y se cambia con **`this.setState({})`**

Componente funcional

Un componente funcional es un componente que se crea **usando una función de JavaScript**. Veamos el paso a paso para crearlo:

1. Crea un **nuevo archivo** .jsx dentro de la carpeta /src.
2. Dentro de ese archivo, escribe **una función normal**.
3. El nombre de la función será el **nombre del componente**. Los componentes se deben escribir en **CamelCase**.
4. Recibe **un solo parámetro** llamado **props**.
5. Escribe en la sentencia **return el código JSX que reaccionará a los cambios**. Recuerda siempre que debes retornar **un solo nodo raíz**.
6. Una vez creado tu componente, debes **exportarlo**. Coloca al final del archivo **export default NombreDeTuComponente**.
7. Ya puedes incluir ese archivo y usar tu componente en **JSX** como si fuese **una nueva etiqueta HTML** (incluso pasarle atributos que serán recibidos a través de las *props*).

Item.jsx U X

src > Item.jsx > ...

```
1  function Item(props)
2  {
3      return (
4          <div>
5              <span>{ props.nombre }</span>
6              <span>{ props.precio }</span>
7          </div>
8      );
9  }
10
11  export default Item;
12
```

Item.jsx U

App.js M X

src > App.js > ...

```
1  import Item from './Item';
2  import './App.css';
3
4  function App() {
5      return (
6          <div className="App">
7              <Item
8                  nombre="Celular"
9                  precio="2500"
10              ></Item>
11          </div>
12      );
13  }
14
15  export default App;
16
```


Componente basado en clases

Un componente basado en clases **se crea mediante clases de JavaScript ES6.**

Veamos el paso a paso para crearlo:

1. Crea un **nuevo archivo .jsx** dentro de la carpeta **/src**.
2. Dentro de ese archivo, escribe ahora una **clase vacía**.
3. El nombre de la clase será el **nombre del componente**. Los componentes se deben escribir en **CamelCase**. La clase debe heredar de **React.Component**. Necesitarás importar el paquete React.
4. Crea el método **constructor** y recibe allí **un solo parámetro llamado props**. Llama al constructor de la clase padre ahí mismo con **super**, pasado las props como parámetro.

5. Escribe el método **render**. El retorno de ese método será **el código JSX que reaccionará a los cambios**. Recuerda siempre que debes retornar **un solo nodo raíz**.
6. Una vez creado tu componente, tienes que exportarlo. Coloca al final del archivo **export default NombreDeTuComponente**.
7. Ya puedes incluir ese archivo y usar tu componente en *JSX*, **como si fuese una nueva etiqueta HTML** (incluso pasarle atributos, que serán recibidos a través de las props).



```
3 import React from "react";
4
5 class Item extends React.Component
6 {
7   constructor(props)
8   {
9     super(props);
10  }
11
12  render()
13  {
14    return (
15      <div>
16        <span>{ this.props.nombre }</span>
17        <span>{ this.props.precio }</span>
18      </div>
19    );
20  }
21 }
22
23 export default Item;
24
```

```
Item.jsx U App.js M X
src > App.js > ...
1 import Item from './Item';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <Item
8         nombre="Celular"
9         precio="2500"
10      ></Item>
11     </div>
12   );
13 }
14
15 export default App;
16
```

**¡Sigamos
trabajando!**

