

# Teoría React I

## ¿Qué es la componentización?

La componentización consiste en "romper" la interfaz de usuario en elementos más pequeños y manejables (componentes). El proceso de componentización es iterativo y evolutivo. Conforme tu aplicación crezca y cambie, probablemente tendrás que volver a visitar y refinar tus componentes.

## ¿Cómo pasar información entre componentes?

Cada componente padre puede enviar información a sus componentes hijos mediante el uso de props. Las props son similares a los atributos HTML, pero permiten pasar cualquier valor de JavaScript a través de ellas, como objetos, arrays y funciones.

Los atributos de cualquier etiqueta HTML están predefinidas, sin embargo, puedes definir y pasar cualquier atributo o **prop** a tus componentes para personalizarlos.

## ¿Qué particularidades tienen los componentes?

La creación de componentes con el format **JSX/TSX** tiene en cuenta las siguientes particularidades: - Se usa **PascalCase** para nombrar los archivos y las funciones que definen componentes. - Tienes que cerrar etiquetas que no necesiten cierre como `<img />` y `<Componente />`. - Tienes que definir clases con el atributo **className** en lugar de **class**. - Como es una función, tu componente debe devolver un único elemento. En caso de necesitar devolver más de un elemento debes envolverlas en un padre compartido, como `<>...</>`. - El **export default** habilitará el uso de una etiqueta `<Componente />` para poder importarse y utilizarse donde sea necesario.

## ¿Cómo muestro datos en un componente?

Las llaves `{}` en JSX/TSX son un elemento poderoso que te permite "volver" a **JavaScript** desde el contexto del marcado (etiquetas).

Esta funcionalidad es especialmente útil para incrustar (o interpolar) variables y expresiones de JavaScript dentro del JSX, permitiéndote mostrar dinámicamente

datos y hasta incluso realizar comprobaciones y/o cálculos en la interfaz de usuario.

Unset

```
export default function Button() {  
  
  const clase = true;  
  
  const texto = "cadena de texto"  
  
  const condicion = clase ? 'color-green' : 'color-red'  
  
  return <button className={condicion}>{texto}</button>;  
  
}
```

## ¿Cómo realizo una renderización dinámica de un componente?

En React, frecuentemente nos encontramos con la necesidad de renderizar listas de datos de forma dinámica. Para lograr esto, utilizamos la función `map()` de JavaScript, que es una herramienta eficaz para iterar sobre una lista de datos y transformar cada elemento en un componente o elemento JSX/TSX.

Este enfoque es crucial porque, en React, cada elemento de la lista debe ser convertido en un elemento de JSX que luego se renderiza en la interfaz de usuario. La función `map()` facilita este proceso al permitirnos aplicar una función de transformación a cada elemento de la lista, devolviendo un nuevo array de elementos JSX.

Además, es importante recordar asignar un atributo `key` único a cada elemento resultante para ayudar a React a identificar qué elementos han cambiado, se han añadido o eliminado, lo que optimiza el proceso de renderizado.

Unset

```
export default function NavBar() {  
  
  const opciones = [{ ruta: "/", titulo: "" }, ...];  
  
  const opcionesTransformadas = opciones.map((elemento, index) => (  
  
    <a key={index} href={elemento.ruta}>  
  
      {elemento.titulo}
```

```

    </a>

  ));

  return <>{opcionesTransformadas}</>;
}

```

También se puede mapear directamente dentro del return.

```

Unset
export default function NavBar() {

  const opciones = [{ ruta: "/", titulo: "" }, ...];

  return <>{opciones.map((elemento, index) => (

    <a key={index} href={elemento.ruta}>

      {elemento.titulo}

    </a>

  ))}</>;

}

```


## ¿Cómo manejo eventos en React?

Como en cualquier programa, existen interacciones del usuario con la interfaz. Debemos identificar: - **¿Qué elemento activará el evento?** Un botón, un selector, un check, etc. - **¿De qué forma?** Un click, una tecla, un cambio, etc - **¿Que hará el evento?** Mostrar una lista desplegable, filtrar productos, modificar el color de un elemento, etc. Luego debemos crear el elemento y asignarle la función con el evento correspondiente.

Los eventos en React se asignan como atributos de la etiqueta/componente con `camelCase`, a modo de ejemplo, `onClick`, `onKeyUp`, `onChange`, etc.

Unset

```
export default function NavBar() {  
  
  const funcionFiltrado = ()=>{ ... }  
  
  const funcionOrdenamiento = ()=>{ ... }  
  
  return (<>  
  
    <input type="text" onKeyUp={funcionFiltrado} />  
  
    <input type="checkbox" onChange={funcionOrdenamiento} />  
  
  </>)  
  
}
```

💡 Como sugerencia, resulta beneficioso recurrir a fuentes oficiales de información al enfrentar dudas o al buscar una comprensión más profunda sobre algún tema. A continuación, te proporcionamos enlaces de utilidad para que puedas investigar a tu propio ritmo en el momento que lo consideres oportuno:  [React](#)