Table of Contents

[React js 1](#_Toc134695502)

[react 1](#_Toc134695503)

[Componentes y elementos 2](#_Toc134695504)

[¿Cómo escuchamos los eventos en React? 3](#_Toc134695505)

[¿Qué es el Estado? 3](#_Toc134695506)

[Icons 3](#_Toc134695507)

[Local Storage con React.js 3](#_Toc134695508)

# **React js**

## react

* npm run Ejecuta nuestro proyecto en modo desarrollo
* npm build Genera versión del proyecto para desplegar a producción.
* ¿Dónde se renderiza (dibuja) nuestra aplicación? En el div con id="root" del index.html
* ¿De dónde viene lo que se renderiza? Del componente “App” que definimos en index.js
* ¿ Qué son los componentes ? piezas de código (generalmente en forma de funciones con nombre en mayúscula).
* ¿Qué retorna la función ? los elementos del componente (elementos de React), en un formato llamado JSX, el cual se parece a HTML.
* ¿ Qué utilidad tiene JSX ? combinar código HTML con JS para, por ejemplo, usar variables para dar valores a los atributos de los elementos
* ¿ Principal diferencia entre elementos y componentes ? Los elementos empiezan con minúsculas y los componentes con mayúsculas.
* ¿ Para qué sirven ? Para reutilizarlos y así no repetir código
* ¿ Cómo llamo a un componente? Escribiendo su nombre con la siguiente sintáxis < Componente1 />
* ¿ Cómo agrego dinamismo a los componentes para que cambien su contenido ? Recibiendo props por parámetros de la función
* Nota: Los componentes se pueden anidar, es decir, meter uno dentro de otro

## Componentes y elementos

Componentes => Reciben Propiedades.

Elementos => Reciben Atributos.

…

Hay que tener en cuenta que en JSX, la sintaxis entre atributos y propiedades es muy parecida, ya que escribimos el valor de las props en los componentes como si fueran atributos, pero estos no lo son…

…

Las props son variables que declaramos en nuestro componente como si fueran parámetros de una función, podemos declarar una unica (Prop) o declarar varias Props utilizando la {Desestructuración}.

Para luego posteriormente darles un valor como si fueran argumentos, pero utilizando otro tipo de sintaxis, que es muy parecida a la de los atributos de los elementos HTML, con la diferencia que el otorgamos el valor entre {llaves}.

…

En cuanto a la propiedad “props.children” esta hace referencia a que React transforma automáticamente a todo lo que se encuentre dentro del componente padre en un elemento “children” , es decir todo lo que se encuentre dentro de una etiqueta de apertura <Component> y una etiqueta de cierre </Component> para React será “Children”.

…

Por otro lado existe <React.Fragment> o su sintaxis más corta " <> </> " que es un contenedor invisible que tiene como objetivo envolver todos los componentes JSX dentro de un contenedor padre, esto es debido a que React necesita envolver esos componentes para renderizarlos, y asi no crear nodos innecesarios en el DOM.

…

Por ultimo, debemos tener en cuenta que podemos renderizar elementos a través de un Array, utilizando en este caso el metodo .map(), y devolviendo el componente “TodoItem” por cada uno de los objetos del array…

Es necesario que le pasemos las props “key={todo.text} text={todo.text}” al componente para que nos muestre un contenido distinto por cada objeto de este array.

…

La prop “Key” funciona como un Identificador para objeto del array y es necesaria cuando utilizamos el metodo .map()

## ¿Cómo escuchamos los eventos en React?

…

Una de las formas es añadiendo “onclick” a nuestro elemento, parecido a la forma que lo realizábamos convencionalmente en HTML y JS, con la diferencia que en React utilizaremos camelCase, ósea “onClick”.

Todo lo que empiece con “on” en React será considerado un Evento, es decir, todo lo que comience con "on" será transformado a un AddEventListener (Escuchador de Eventos), es importante tener en cuenta que el valor que recibirá nuestro Evento, debe ser encapsulado dentro de una función para que se ejecute correctamente.

## ¿Qué es el Estado?

…

El "Estado" es una de las principales herramientas que React nos proporciona para crear UI´s dinámicas y actualizadas en tiempo real.

Podríamos definir al Estado como “Variables que almacenan información dinámica de un componente y se actualizan a medida que el usuario interactúa con los Componentes”.

Aquella información utilizada en el estado puede ser utilizada por el componente para cambiar su apariencia y comportamiento en tiempo real.

El estado también nos permite mantener la información actualizada de un componente sin necesidad de renderizar todo el componente, esto gracias al Virtual DOM que compara los cambios que se hicieron y los actualiza.

## Icons

<https://react-icons.github.io/react-icons/>

## Local Storage con React.js

**LocalStorage** es una forma de almacenar datos en el navegador web, que persisten aún después de que el usuario haya cerrado la ventana o salido del sitio web. Es útil para guardar información del usuario, como preferencias, configuraciones, entre otros.

Los métodos más comunes de **LocalStorage** son:

* localStorage.setItem(key, value): Agrega un elemento con una clave y un valor al almacenamiento local.
* localStorage.getItem(key): Recupera el valor de la clave especificada en el almacenamiento local.
* localStorage.removeItem(key): Remueve un elemento del almacenamiento local según su clave.
* localStorage.clear(): Borra todos los elementos del almacenamiento local.