**Curso React.js**

**ReactDOM.render**

* React y ReactDOM trabajarán en conjunto.
  + React como análogo a createElement
  + ReactDOM a appendChild
* ReactDOM.render() toma dos argumentos: Qué queremos renderizar y dónde lo queremos renderizar.
* Siempre que escribas JSX es requisito importar React.

**JSX**

JSX es una extensión de JavaScript creada por Facebook para el uso con la biblioteca React. Sirve de preprocesador (como Sass o Stylus a CSS) y transforma el código generado con React a JavaScript.

JSX tiene su alternativa que es React.createElement pero es preferible JSX porque es mucho más legible y expresivo. Ambos tienen el mismo poder y la misma capacidad.

React.createElement recibe 3 argumentos:

* El tipo de elemento que estamos creando
* sus atributos o *props*
* y el *children* que es el contenido.

Ejemplo:  
React.createElement(‘a’, { href: ‘https://platzi.com’ }, ‘Ir a Platzi’);

En JSX se utilizan las llaves para introducir variables o expresiones de Javascript. Lo que sea que esté adentro se va a evaluar y su resultado se mostrará en pantalla.

Las expresiones pueden ser llamadas a otras funciones, cálculos matemáticos, etc. Si las expresiones son false, 0, null, undefined, entre otros, no se verán.

**¿Qué es un componente?**

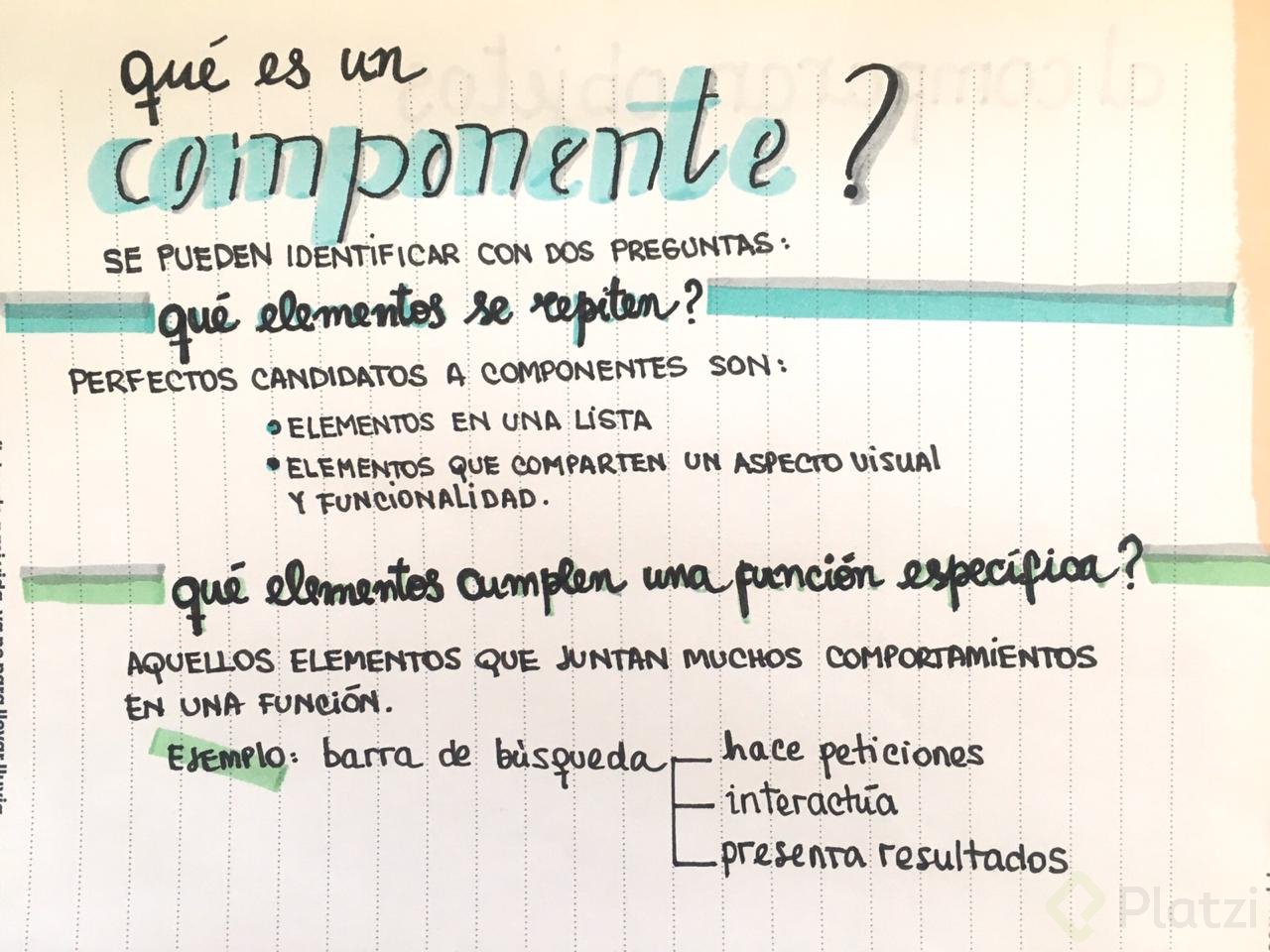
Los componentes en React son bloques de construcción.  
Las aplicaciones hechas con React son como figuras de Lego. Junta varias piezas (componentes) y puedes construir un website tan pequeño o tan grande como quieras.  
Los componentes serán barras de búsquedas, enlaces, encabezados, el *header*, etc.

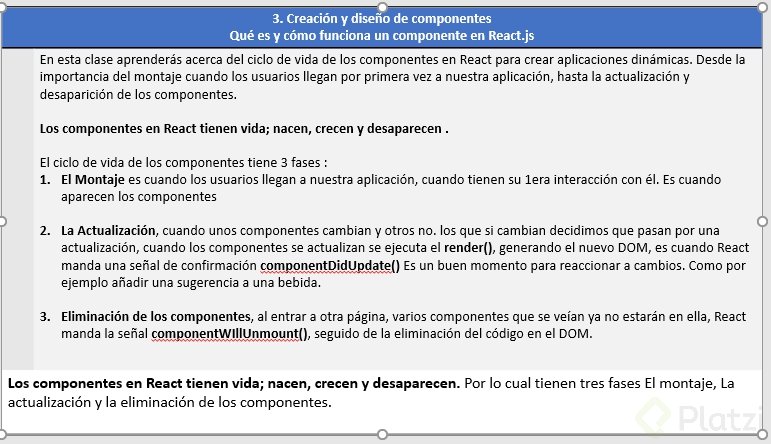
”Componente” vs “elemento  
Un elemento es a un objeto como un componente es a una clase. Si el elemento fuera una casa, el componente serían los planos para hacer esa casa.

Identificación de componentes  
Para identificarlos debes hacerte las siguientes preguntas:

* ¿Qué elementos se repiten? Estos son los elementos en una lista o los que comparten aspecto visual y su funcionalidad
* ¿Qué elementos cumplen una función muy específica? Estos sirven para encapsular la lógica y permiten juntar muchos comportamientos y aspectos visuales en un solo lugar.

Identificar componentes es una habilidad esencial para poder desarrollar aplicaciones de React.





**Nuestro primer componente**

Es una buena práctica que los componentes vivan en su propio archivo y para ello se les crea una carpeta.

* Todos los componentes requieren por lo menos el método render que define cuál será el resultado que aparecerá en pantalla.
* El *source* de las imágenes en React puede contener direcciones en la web o se le puede hacer una referencia directa importándola. Si se importa deben usarse llaves para que sea evaluado.

**Cómo aplicar estilos**

* Para los estilos crearemos una carpeta llamada **Styles** y allí vivirán todos los archivos de estilos que tienen que ver con los componentes.
* Para usar los estilos es necesario importarlos con *import*
* React funciona ligeramente diferente y para los atributos de clases no se utiliza *class* sino *className*
* Es posible utilizar **Bootstrap** con React, sólo debe ser instalado con npm install bootstrap y debe ser importado en el **index.js**
* Existen estilos que son usados de manera global o en varios componentes, así que deben ser importados en el index.js

**Props**

Los props que es la forma corta de *properties* son argumentos de una función y en este caso serán los atributos de nuestro componente como class, src, etc.

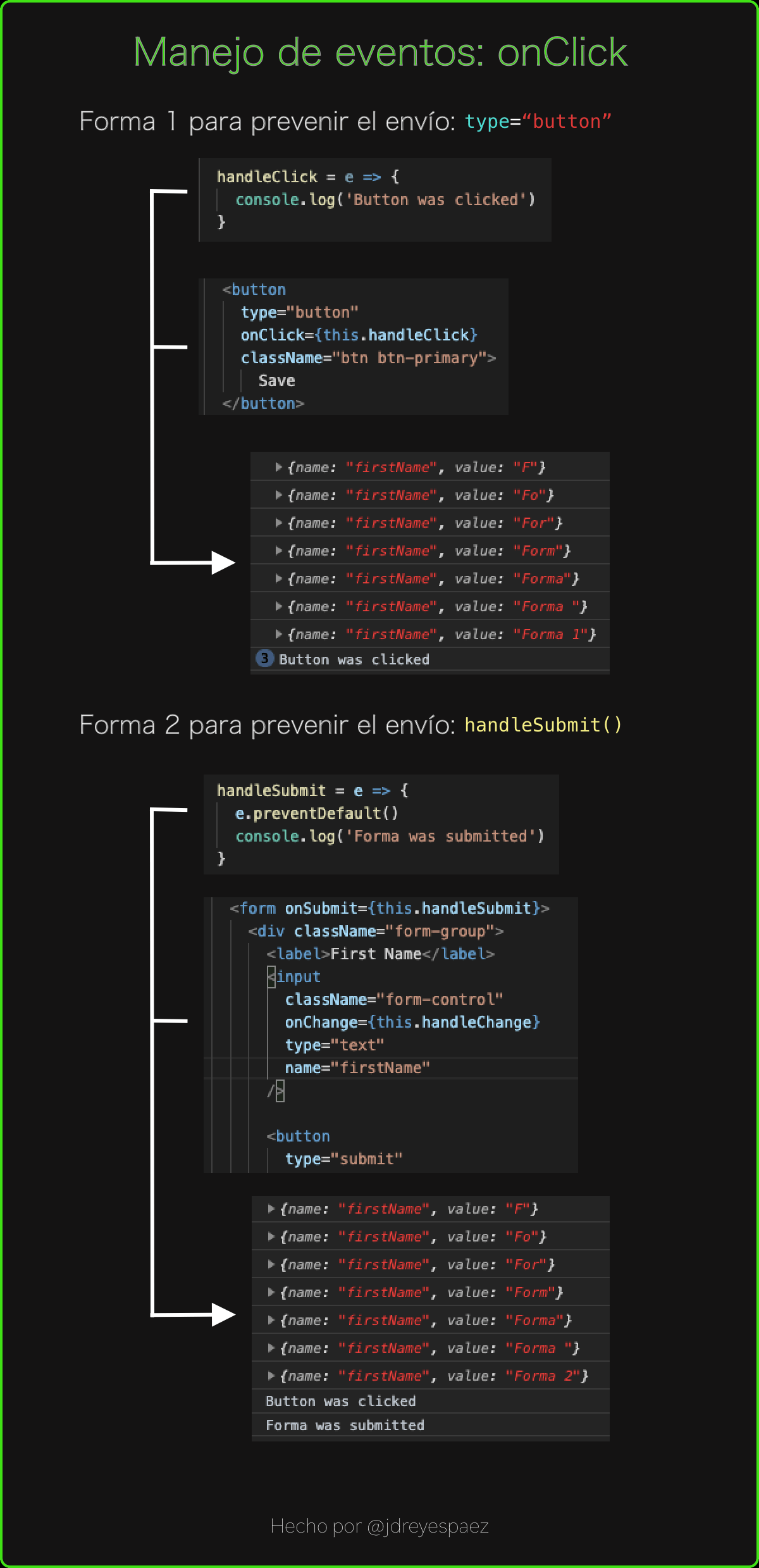
Estos props salen de una variable de la clase que se llama *this.props* y los valores son asignados directamente en el ReactDOM.render().

**Enlazando eventos**

React dispone de eventos. Cada vez que se recibe información en un input se obtiene un evento onChange y se maneja con un método de la clase *this.handleChange*

* Los elementos *button* también tienen un evento que es onClick.
* Cuando hay un botón dentro de un formulario, este automáticamente será de tipo submit. Si no queremos que pase así hay dos maneras de evitarlo: especificando que su valor es de tipo *button* o manejándolo desde el formulario cuando ocurre el evento onSubmit

**Aquí va mi resumen de las los formas de evitar que el comportamiento default del botón no sea enviar la información:**



**Manejo de estado**

Hasta esta clase todos los componentes han obtenido su información a través de *props* que vienen desde afuera (otros componentes) pero hay otra manera en la que los componentes pueden producir su propia información y guardarla para ser consumida o pasada a otros componentes a través de sus *props*. La clave está en que la información del state a otros componentes pasará en una sola dirección y podrá ser consumida pero no modificada.

* Para guardar la información en el estado se usa una función de la clase *component* llamada setState a la cual se le debe pasar un objeto con la información que se quiere guardar.
* Aunque no se ve, la información está siendo guardada en dos sitios. Cada *input* guarda su propio valor y al tiempo la está guardando en *setState*, lo cual no es ideal. Para solucionarlo hay que modificar los *inputs* de un estado de no controlados a controlados.

**Levantamiento del estado**

Levantar el estado es una técnica de React que pone el estado en una localización donde se le pueda pasar como *props* a los componentes. Lo ideal es poner el estado en el lugar más cercano a todos los componentes que quieren compartir esa información.

Algo interesante que le da el nombre a React es su parte de “reactivo” ya que cada vez que hay un cambio en el estado o en los *props* que recibe un componente se vuelve a renderizar todo el componente y todos sus descendientes.

**Introducción a React Router**

Las aplicaciones que se trabajan en React son llamadas single page apps. Esto es posible gracias a React Router que es una librería *Open Source*.

Multi Page Apps: Cada página implica una petición al servidor. La respuesta usualmente tiene todo el contenido de la página.

Single Page Apps (SPA): Aplicaciones que cargan una sola página de HTML y cualquier actualización la hacen re-escribiendo el HTML que ya tenían.

React Router (v4): Nos da las herramientas para poder hacer SPA fácilmente. Usaremos 4 componentes:

* BrowserRouter: es un componente que debe estar siempre lo más arriba de la aplicación. Todo lo que esté adentro funcionará como una SPA.
* Route: Cuando hay un match con el path, se hace render del component. El component va a recibir tres props: match, history, location.
* Switch: Dentro de Switch solamente van elementos de Route. Switch se asegura que solamente un Route se renderize.
* Link: Toma el lugar del elemento <a>, evita que se recargue la página completamente y actualiza la URL.

**Mejorando la User Interface con un Layout**

[**Aquí encuentras el archivo para hacer el reto que propone el profesor al final de la clase**](https://drive.google.com/file/d/11C4nqBCTqF6dJMrVYMIs06sgwI5Y1KsF/view?usp=sharing)

* React.Fragment es la herramienta que te ayudará a renderizar varios componentes y/o elementos sin necesidad de colocar un div o cualquier otro elemento de HTML para renderizar sus hijos. Al usar esta característica de React podremos renderizar un código más limpio y legible, ya que ``React.Fragment` no se renderiza en el navegador.
* El 404 es la ruta que se renderizará cuando ninguna otra coincida con la dirección ingresada.

Otra forma de hacer que todas tus URL’s que no existan sean redirigidas a tu componente de 404 sería de la siguiente forma:

import { Redirect, Route } from "react-router-dom";

<Route path="/404" component={MiComponente404} />

<Redirect from="\*" to="/404" />

Como podemos observar llamamos a nuestro componente 404 y luego utilizamos Redirect, el cual es un componente de React Router para hacer redirecciones; en este caso hacemos que todas las URL’s que no correspondan a alguna que hayamos declarado, sean redirigidas a MiComponente404.

**Introducción del ciclo de vida de un componente**

Cuando React renderiza los componentes decimos que entran en escena, cuando su estado cambia o recibe unos *props* diferentes se actualizan y cuando cambiamos de página se dice que se desmontan.

Montaje:

* Representa el momento donde se inserta el código del componente en el DOM.
* Se llaman tres métodos: *constructor*, *render*, *componentDidMount*.

Actualización:

* Ocurre cuando los *props* o el estado del componente cambian.
* Se llaman dos métodos: *render*, *componentDidUpdate*.

Desmontaje:

* Nos da la oportunidad de hacer limpieza de nuestro componente.
* Se llama un método: *componentWillUnmount*.

**Introducción llamadas a un API2**

Las llamadas a una API siguen un patrón similar siempre que las hacemos, cada llamada consta de tres estados:

* Loading: cuando la petición se envía y estamos esperando.
* Error: se debe dejar un mensaje para el usuario para arreglar el error o volver a intentarlo.
* Data: los datos nos pueden llegar de dos formas, o en error o con los datos requeridos.

<https://api-platzi-badges.vercel.app/api/badges>

**Enviando datos (POST)**

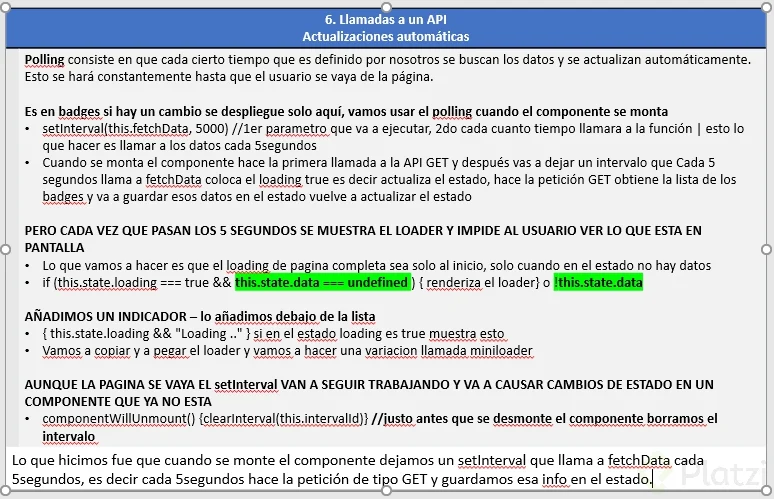
**MD5** es una pequeña librería a la cual se le da un texto y ella regresa un **hash**.

Podremos hacer pruebas para cifrar nuestros textos a md5 en el siguiente sitio [MD5 Online](https://md5online.es/cifrar-md5)

Hola Platzi = d3bfb9302fb1007c0f996b41cba2818c

**Actualizaciones automáticas**

**Polling** consiste en que cada cierto tiempo que es definido por nosotros se buscan los datos y se actualizan automáticamente. Esto se hará constantemente hasta que el usuario se vaya de la página.



**Los detalles de un Badge**

CRUD es el acrónimo de “Crear, Leer, Actualizar y Borrar” (del original en inglés: Create, Read, Update and Delete) Adicional todas las operaciones de tipo CRUD son aplicadas a la base de datos, es decir, creamos elementos de la base de dato, leemos y actualizamos. En el curso estamos haciendo uso de verbos HTTP para lograr las peticiones con la arquitectura restApi

**UI Components y Container Components**

En la programación es bueno separar las tareas en diferentes funciones y en React sucede lo mismo. Cuando un componente hace demasiado, probablemente es mejor dividirlo en dos.

Esta técnica de componentes presentacionales y componentes *container* es común, útil y hace parte de las buenas prácticas

**Portales**

Hay momentos en los que queremos renderizar un modal, un *tooltip*, etc. Esto puede volverse algo complicado ya sea por la presencia de un *z-index* o un *overflow hidden*.

En estos casos lo ideal será renderizar en un nodo completamente aparte y para esto React tiene una herramienta llamada **Portales** que funcionan parecido a ReactDOM.render; se les dice qué se desea renderizar y dónde, con la diferencia de que ese dónde puede ser fuera de la aplicación.

Recapitulando: **¿Qué es, para qué es y como podemos usar un portal modal?**

Un portal es un contenido HTML que está fuera del contenedor (el div id="app") de tu aplicación principal

Nos resulta especialmente relevante para crear Modales sin complicaciones extremas en el diseño ya que el nodo se encuentra a la misma altura que el de la App

**¿Cómo lo implemementamos?**

1. Crea el contenedor de tu modal en tu archivo index.html  
   <div id="modal"> </div>
2. En el componente donde lo vayas a utilizar importa el ReactDOM ya que a través de su método createPortal() invocaremos el contenedor externo que acabamos de crear
3. Crea el Portal al componente a través de ReactDOM.createPortal()
4. Pasa por parametro Qué renderizara y donde lo hará

ReactDOM.createPortal(

<h1> El titulo más bonito del mundo </h1> ,

document.getElementById("modal")

)

1. Supongo que si quieres importar un modal habrás creado en otra parte el componente Modal

ReactDOM.createPortal(

<Modal /> ,

document.getElementById("modal")

)

**Modales**

La técnica de usar componentes genéricos para crear uno nuevo especializado se llama **composición** y es una herramienta que todo buen programador debe saber utiliza

Los modales son ventanas flotante emergentes que pueden ser activadas por el usuario o por el mismo sitio web. Su función es mostrar contenido relevante que solamente será usado en el momento, por ejemplo:

* Formularios
* Petición de permisos
* Algún aviso
* etc

**Hooks**

Las funciones no tienen un estado propio que manejar como ciclos de vida a los que deben suscribirse, mientras tanto las clases sí cuentan con ello.

React tiene un feature llamado **Hooks** que permite que las funciones también tengan *features* que solamente tienen las clases.

**Hooks:** Permiten a los componentes funcionales tener características que solo las clases tienen:

* **useState:** Para manejo de estado.
* **useEffect:** Para suscribir el componente a su ciclo de vida.
* **useReducer:** Ejecutar un efecto basado en una acción.

**Custom Hooks:** Usamos los *hooks* fundamentales para crear nuevos *hooks custom*. Estos *hooks* irán en su propia función y su nombre comenzará con la palabra *use*. Otra de sus características es que no pueden ser ejecutados condicionalmente (*if*).

* *useState* regresa un arreglo de dos argumentos.