Clase 4 - Componentes I

# Componentes I: Introducción

Para generalizar la definición de componente de UI podemos decir que un componente es un conjunto de elementos que cumplen una función específica.

En React un componente va a ser una unidad modular y reutilizable de una interfaz de usuario que se encarga de representar una parte específica de la interfaz de usuario.

Los componentes de React se utilizan para construir la jerarquía de elementos de una interfaz de usuario. Cada componente se encarga de representar una parte específica de la interfaz de usuario y puede contener otros componentes como hijos.

**Diseño modular**

Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada.

Al desarrollar crearemos componentes para resolver pequeños problemas, que son fáciles de visualizar y comprender.

Luego, unos componentes se apoyarán en otros para solucionar problemas mayores y al final la aplicación será un conjunto de componentes que trabajan entre sí.

Este modelo de trabajo tiene varias ventajas, como la facilidad de mantenimiento, depuración, escalabilidad, etcétera.

**Ventajas del enfoque**

* Favorece la separación de responsabilidades: cada componente debe tener una única tarea.
* La aplicación es más fácil de entender.
* Mejora el rendimiento de la aplicación.
* Se simplifica la tarea de hacer pruebas unitarias.

**Componentes I**

En React JS nuestras aplicaciones van a ser estructuradas con nuestros componentes, por eso es importante entender sus características.

Primero, deberíamos pensar a nuestra aplicación de React como una máquina, podemos traer de ejemplo un automóvil, teniendo en cuenta que es solo una analogía para entender estos conceptos abstractos. Así como un automóvil está compuesto por muchas piezas que trabajan juntas para cumplir una función específica, una aplicación de React está compuesta por muchos componentes que trabajan juntos para crear una experiencia de usuario coherente y funcional.

Ahora bien, es importante que hablemos de los estados porque es el concepto más importante hoy en día para entender el funcionamiento de React.

Volviendo al ejemplo del auto, podemos tomar una pieza como la caja de cambios que en su conjunto va a tener un estado, es decir, en un momento va a estar en punto muerto, en otro va a estar en primera, en otro en segunda y así. Desde el momento que enciendo el auto, hasta que llego a mi destino, la caja de cambios fue cambiando de marcha o lo que es lo mismo cambiando de estado. Los estados de un componente en React se utilizan para representar la información que puede cambiar durante la vida útil del componente y afectar su representación en la interfaz de usuario. Un estado puede incluir datos que provienen de la entrada del usuario, datos que se obtienen a través de una API o datos que cambian dinámicamente a medida que la aplicación evoluciona.

**Componentes basados en funciones**

Hoy en día, los componentes de React se crean a partir de funciones. Antes los componentes se creaban de forma distinta, con clases, pero a pesar de compartir muchas características el modelo mental es muy distinto. Utilizando solos funciones es mucho más fácil de escribir el código y entenderlo. Esto no quita que ambos tipos de componentes puedan convivir sin inconvenientes.

En React JS existen dos maneras de entender los componentes, que varían según desde dónde nos paremos para analizarlo.

Vamos a decir que existen tipos de componentes y patrones.

**Características principales de los componentes**

* Pueden recibir propiedades (props).
* Tienen la capacidad de hacer render de un único elemento. Aunque este elemento puede tener muchos elementos dentro.
* Pueden tener estados.

**Propiedades**

Las propiedades, llamadas props en react, son la forma que tiene React para pasar parámetros de un componente superior a sus hijos (children).

Las props en React son una forma de pasar datos de un componente padre a un componente hijo. Las props son objetos que contienen datos específicos que un componente necesita para renderizarse correctamente. Al pasar props a un componente, se pueden reutilizar los mismos componentes en diferentes partes de la aplicación con diferentes datos, lo que hace que el código sea más modular y escalable. Además, es la manera de implementar el flujo de datos unidireccional.

Un detalle importante es que si alguna prop es una función, el componente hijo puede llamarla para provocar efectos secundarios en el componente padre que le paso la función. Esta función pasada por props y ejecutada en el hijo, tiene acceso a los datos del componente padre gracias al closure. Un closure es un mecanismo que permite a una función tener acceso a variables y parámetros que están fuera de su ámbito o alcance (scope).

**Render de un único elemento**

En React, los componentes deben retornar un solo elemento o componente. Esto se debe a que React trabaja con un modelo de árbol de componentes en el que cada componente es un nodo del árbol, pero este elemento o componente si puede tener más de un elemento o componente hijo.

En caso de no necesitar que el elemento padre forme parte del árbol del DOM, se puede utilizar React.Fragment, o lo que es lo mismo utilizar un tag sin tipo <></>

**Estados**

Como mencionamos antes los estados de un componente en React se utilizan para representar la información que puede cambiar durante la vida útil del componente y afectar su representación en la interfaz de usuario.

Esto quiere decir varias cosas. Primero, que los estados son un concepto de React, no de JavaScript, por lo que para implementarlo vamos a necesitar de React. Segundo, que todo cambio de estado va inicializar el proceso de render a partir del nodo donde se produjo el cambio de estado y así se generará el nuevo Virtual DOM que luego será reconciliado. Esto quiere decir que todo dato que cambie y no este apuntado a un estado no generará cambios en la interfaz, implementando **React con programación funcional con hooks**, solamente un cambio de estado inicia todo el proceso de renderizado de React para la actualización.

Pero que son los hooks?

**Hooks**

Los hooks de React son una característica introducida en React 16.8 que permiten utilizar las características de React en los componentes funcionales, en lugar de tener que usar componentes basados en clases.

Los hooks son funciones que se pueden usar dentro de los componentes funcionales para agregar funcionalidad adicional de React. Son varias las funcionalidades, una de ellas son los estados, con el hook useState.

Cada uno de los hooks tiene características particulares con respecto a su funcionalidad, pero comparten características con respecto a su implementación:

Todos los hooks empiezan con el prefijo “use” para que React pueda identificarlos y permitir su uso en los componentes y realizar validaciones.

Solamente pueden utilizarse dentro de componentes funcionales.

El hook debe ejecutarse siempre, es decir, en cada renderizado y el orden de ejecución debe ser siempre el mismo. No puede estar dentro de if, for, etc. Deben ejecutarse siempre en el cuerpo de la función del componente, por lo que normalmente se los ejecuta al inicio.

spoiler: Cualquier función que se declare con el prefijo “use”, React la tratara como un hook realizando las validaciones correspondientes. Con esto podemos pensar que dentro de una función hook puedo llamar otros hooks, ya que si este hook padre cumple las validaciones, y dentro de él, los hooks cumplen, los requisitos correspondientes, todas las validaciones pasarán y podrá ejecutarse sin problemas. A estos hooks declarados por el desarrollador para encapsular una lógica de componente específica se les llama custom hooks.

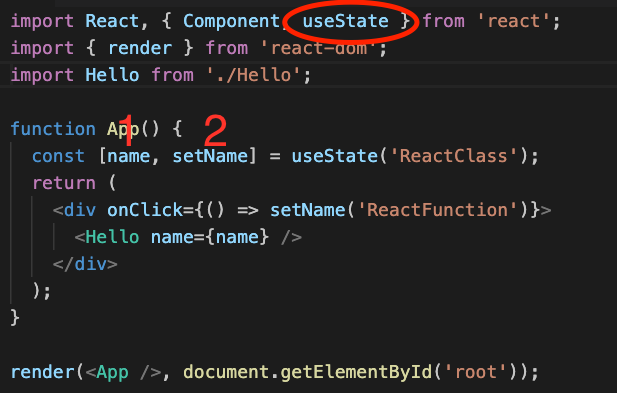
**useState**

useState es un hook de React que permite a los componentes funcionales tener estado. Este estado apunta a valores y estos pueden ir cambiando durante la vida útil del componente. Estos cambios inician el proceso de renderizado a partir de la fibra (nodo) dueña del estado hacia sus hijos.

La ejecución de la función useState retorna un array con dos elementos. El primer elemento es el valor actual del estado y el segundo es una función que se utiliza para actualizar el estado, al llamar esta función con un nuevo valor pasado como argumento, React re-renderiza el componente con el nuevo valor del estado. Es importante destacar que esta función no actualiza el valor del estado de manera síncrona, sino que React lo actualiza cuando inicia el próximo ciclo de renderizado. Por lo tanto, si se necesita acceder al valor actualizado del estado del estado inmediatamente después del llamado de esta función, se deben utilizar otras funcionalidades de React (Clase componentes II).

Para inicializar el estado, se debe proporcionar un valor inicial como argumento a useState.

Como esta función retorna un array de dos elementos podemos desestructurarlo para darle un nombre más apropiado a nuestro estado y a su función de actualización. El nombre debe ser representativo del contenido y la función debe ser el mismo nombre con el prefijo “set”



**Patrones**

**Componentes de presentación**

Son aquellos que simplemente se limitan a mostrar datos y tienen poca o nula lógica asociada a manipulación del estado (por eso son también llamados stateless components).

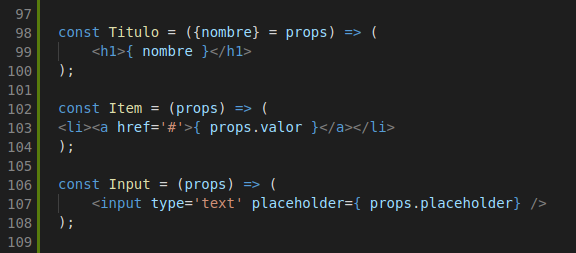
**Características**

* Orientados al aspecto visual.
* No tienen dependencia con fuentes de datos (ej. Llamadas a APIs, Redux).
* Normalmente no tienen estado.

Los componentes de presentación usualmente no tienen estado, en caso de tenerlos serán para trabajar características visuales del componente pero no para almacenar datos de la aplicación, sino que reciben datos de aplicación por props.

La ventaja más evidente de estos componentes es la posibilidad de reutilizarlos siempre que queramos, sin tener que recurrir a escribir el mismo código una y otra vez, ya que

Usando esta sintaxis, las propiedades se reciben como parámetros de la función, y podemos obtener las variables que nos interesan por separado.

****

**Composición de componentes**

**Componentes contenedores**

Tienen como propósito encapsular a otros componentes

y proporcionarles las propiedades que necesitan.

Además se encargan de modificar el estado de la aplicación para que el usuario vea el cambio en los datos (por eso son también llamados state components).

Componentes contenedores.

**Características**

* Orientados al funcionamiento de la aplicación.
* Contienen componentes de presentación y/u otros contenedores.
* Se comunican con las fuentes de datos.
* Tienen estado para representar el cambio en los datos.