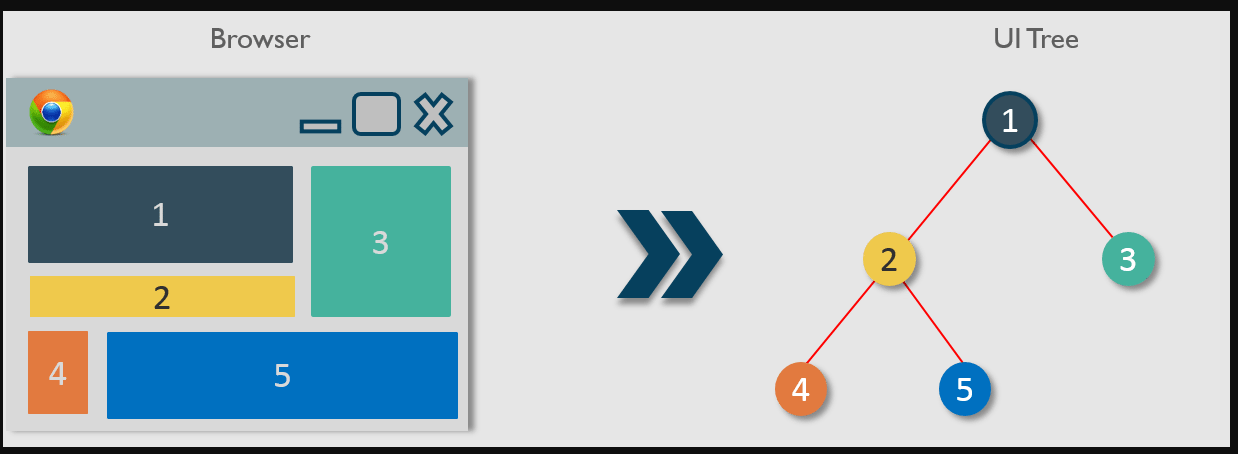
**1. Introducción**

**Componentes en React**

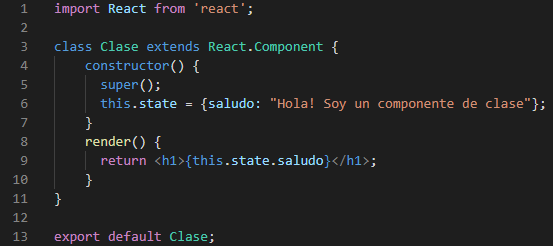
Las aplicaciones en React se construyen mediante componentes.

El potencial de este funcionamiento consiste en que podemos crear aplicaciones completas de una manera **modular** y de **fácil** **mantenimiento**, *a pesar de su complejidad.*

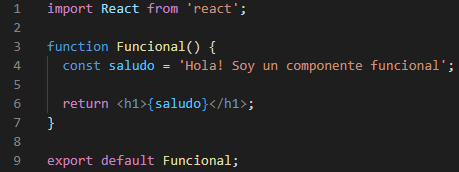


Pero en sí, para **JavaScript**, un componente no es más que una **función** que recibe un **objeto**, al que llamaremos “**props**”, y retorna un elemento de React que describe que va a aparecer en la pantalla.

React admite 2 tipos de componentes, los funcionales y los de clase, nosotros trabajaremos con componentes del tipo funcional, ya que en la actualidad se trabaja de esa forma.

**Componente de clase:**

**Componente funcional:**

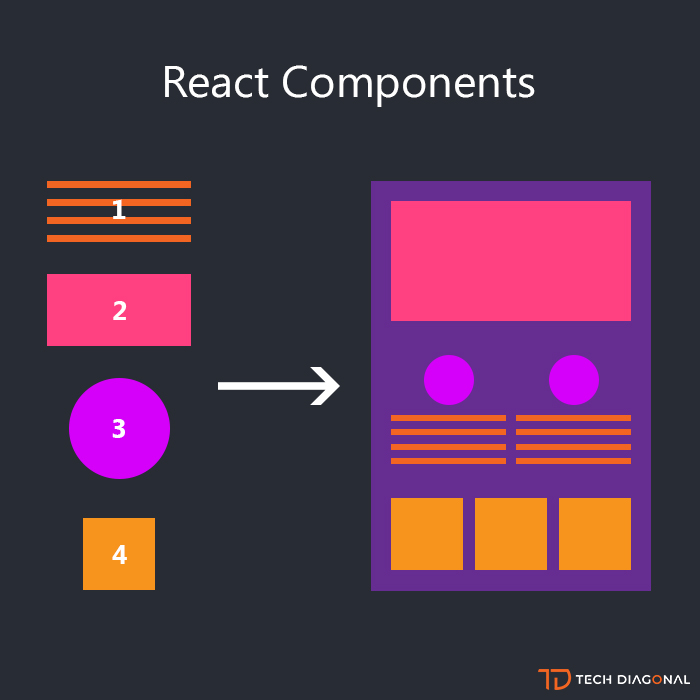
****

**2. Diseño Modular**

**Diseño modular**

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

Al desarrollar crearemos componentes para resolver pequeños distintos problemas, que son fáciles de visualizar y comprender durante el proceso de desarrollo, por lo cual nos ahorrar mucho tiempo.



Los distintos **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, etc.***

A menudo, las actualizaciones se convierten en un problema porque la aplicación tiene una ***lógica compleja*** y los cambios en un componente pueden afectar a otros. Para resolver el problema, **Facebook** ha complementado React con la capacidad de *reutilizar componentes del sistema*, y ​​los desarrolladores lo definen como **una de las mejores características de React js**.

La reutilización de activos es bien conocida entre los diseñadores, que suelen volver a emplear los mismos objetos digitales. Puede comenzar con los componentes más finos (casilla de verificación, botón, etc.), luego pasar a los componentes de envoltura compuestos por estos pequeños elementos y avanzar hasta el componente raíz principal. Todos los componentes tienen su lógica interna, lo que facilita su manipulación y definición. Este enfoque garantiza un aspecto uniforme de la aplicación y facilita el mantenimiento y el crecimiento de la base de código.

**Ventajas:**

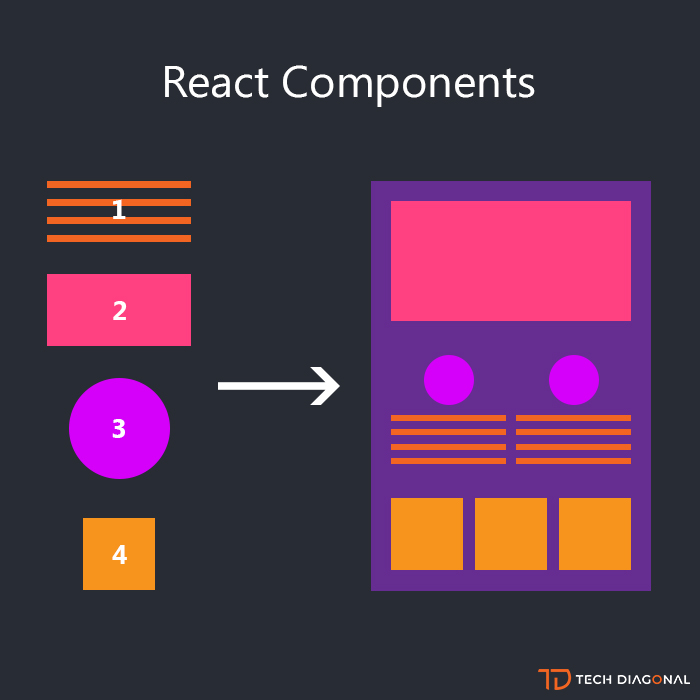
* *Favorece la separación de responsabilidades: cada componente debe tener una única tarea.*
* *Al tener la lógica de estado y los elementos visuales por separado, es más fácil reutilizar los componentes.*
* *Se simplifica la tarea de hacer pruebas unitarias.*
* *Puede mejorar el rendimiento de la aplicación.*
* *La aplicación es más fácil de entender.*

**2. Propiedades**

**Diseño modular**

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

Al desarrollar crearemos componentes para resolver pequeños distintos problemas, que son fáciles de visualizar y comprender durante el proceso de desarrollo, por lo cual nos ahorrar mucho tiempo.



Los distintos **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, etc.***

A menudo, las actualizaciones se convierten en un problema porque la aplicación tiene una ***lógica compleja*** y los cambios en un componente pueden afectar a otros. Para resolver el problema, **Facebook** ha complementado React con la capacidad de *reutilizar componentes del sistema*, y ​​los desarrolladores lo definen como **una de las mejores características de React js**.

La reutilización de activos es bien conocida entre los diseñadores, que suelen volver a emplear los mismos objetos digitales. Puede comenzar con los componentes más finos (casilla de verificación, botón, etc.), luego pasar a los componentes de envoltura compuestos por estos pequeños elementos y avanzar hasta el componente raíz principal. Todos los componentes tienen su lógica interna, lo que facilita su manipulación y definición. Este enfoque garantiza un aspecto uniforme de la aplicación y facilita el mantenimiento y el crecimiento de la base de código.

**Ventajas:**

* *Favorece la separación de responsabilidades: cada componente debe tener una única tarea.*
* *Al tener la lógica de estado y los elementos visuales por separado, es más fácil reutilizar los componentes.*
* *Se simplifica la tarea de hacer pruebas unitarias.*
* *Puede mejorar el rendimiento de la aplicación.*
* *La aplicación es más fácil de entender.*