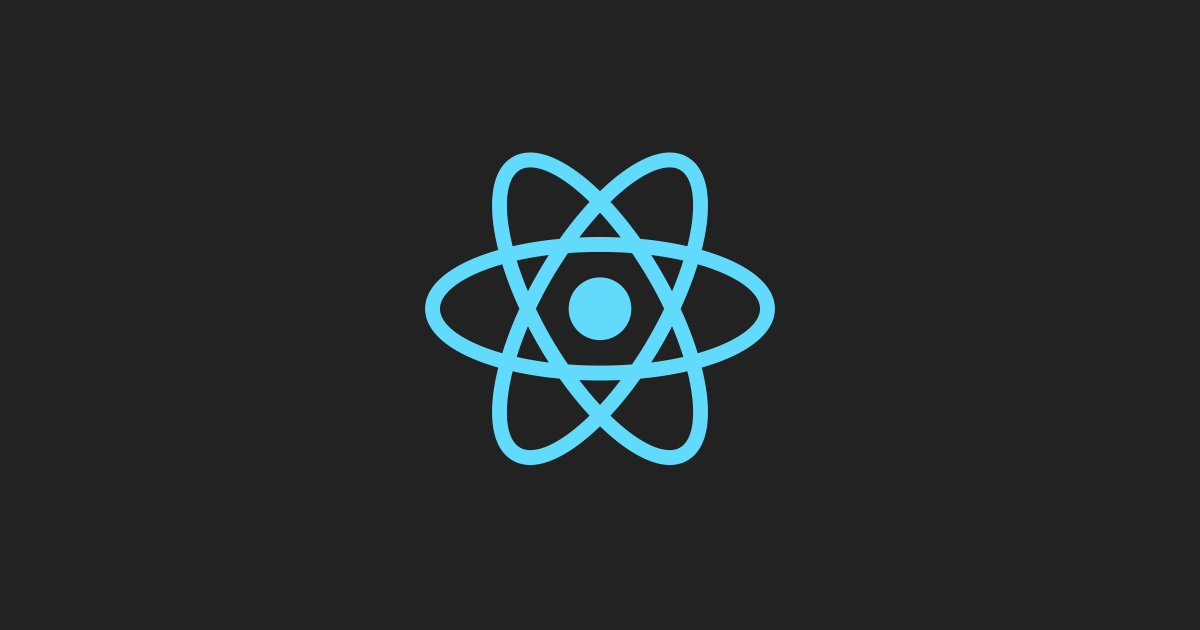
****

**REACT**

# **INTRODUCCIÓN**

*React* es una biblioteca *JavaScript* altamente eficiente y declarativa que se utiliza para crear interfaces de usuario interactivas.

Fue creado por el ingeniero de *Facebook* *Jordan Walke* y se lanzó en mayo de 2013.

En los últimos años, ha superado a sus rivales y ha establecido firmemente su dominio.

React te insita a crear **código declarativo** (indicas el qué, no el cómo) y **orientado a componentes**.

Un componente es un pieza de código que puede representar una parte de la interfaz de usuario o una funcionalidad en particular que se puede encapsular y reutilizar en diferentes partes de un desarrollo o incluso en diferentes aplicaciones.

Cada componente **reaccionará** a cambios internos (en su estado) o externos (nuevas propiedades recibidas) y se volvera a redibujar (**renderizado**) en la interfaz. Para ello *React* utiliza su ***Virtual DOM*** que es una copia del *DOM* original del navegador para sólo hacer los cambios necesarios en los nodos que hayan reaccionado y evitar redibujar todo el árbol del *DOM*.

Para poder interactuar con la interfaz *React* tiene **eventos sintéticos** que son una abstracción de los eventos nativos de los navegadores, para mejorar la compatibilidad y evitar el uso de librerías para ***crossbrowsing.***

## ¿Qué es una *API*?

Una **API** (Application Programming Interface), o Interfaz de Programación de Aplicaciones, es un conjunto de rutinas que provee acceso a funciones de un determinado software, para efectos de esta nota, acceso a las funciones de nuestros sitios y aplicaciones web.

Dentro de los navegadores web tenemos 2 tipos de APIs:

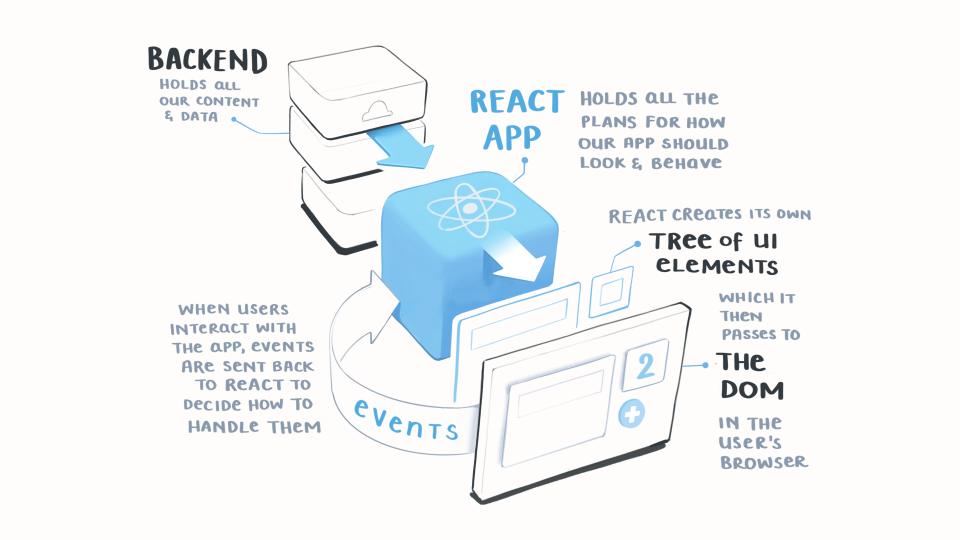
1. **APIs nativas**, como el DOM que es el API que nos permite acceder al código HTML de nuestros documentos a través de JavaScript o la de Geolocalización que no permite acceder a las coordendas terrestres de nuestra ubicación actual; y,
2. **APIs externas**, como la de Twitter que nos permite obtener los últimos tweets de un usuario, o la de GitHub que nos permite obtener la información de los repositorios de una cuenta o como la API de WordPress que nos permite acceder a la información de páginas y entradas de un sitio.

### **DOM**

El **DOM** es lo que el navegador genera para que el usuario vea.

Atrás esta mi app (**REACT APP**) administrando la lógica para mostrar en el dom.

Lo que está en el medio entre el DOM y REACT APP es el **VIRTUAL DOM** es como “una copia del DOM”.



### **¿Qué es el DOM virtual?**

El DOM virtual (VDOM) es un concepto de programación donde una representación ideal o “virtual” de la IU se mantiene en memoria y en sincronía con el DOM “real”, mediante una biblioteca como ReactDOM. Este proceso se conoce como [reconciliación](https://es.reactjs.org/docs/reconciliation.html).

Este enfoque hace posible la API declarativa de React: le dices a React en qué estado quieres que esté la IU, y se hará cargo de llevar el DOM a ese estado. Esto abstrae la manipulación de atributos, manejo de eventos y actualización manual del DOM que de otra manera tendrías que usar para construir tu aplicación.

Ya que “DOM virtual” es más un patrón que una tecnología específica, las personas a veces le dan significados diferentes. En el mundo de React, el término “DOM virtual” es normalmente asociado con [elementos de React](https://es.reactjs.org/docs/rendering-elements.html) ya que son objetos representando la interfaz de usuario. Sin embargo, React también usa objetos internos llamados “fibers” para mantener información adicional acerca del árbol de componentes. Éstos pueden ser también considerados como parte de la implementación de “DOM virtual” de React.

## REPASO JS

## Let: permite asignaciones, asigno un nuevo valor de la variable

## Const: Puede cambiar su valor, pero lo que no puede cambiar es su asignación. Puedo modificar el array pero no el const, es mas estricto.

# **FUNCIONAMIENTO DE REACT**

## React es una librería que nos permite manipular el DOM.

## React calcula el área de impacto y determina la menor cantidad de movimientos de modo heurístico, por lo que también sabe donde no pueden haber ocurrido cambios.

## Para que React pueda garantizar el funcionamiento nos va a demandar que este estructurada por jerarquías. Componentes y árbol de componentes reacción(bajan datos a los hijos)-acción(suben los eventos).

## 

## *VIRTUAL DOM*

## Es un patrón de comportamiento y React lo implementa con una tecnología llamada Fiber. Es todo lo que React sabe de tu app y cada nodo o fibra. Es una representación virtual de la UI (interfaz de usuario) que se mantiene en memoria y en sincronía “reconciliado” con el DOM “real”. Es una representación previa que React crea antes del DOM. Almacena datos, información.

## En primer lugar, React ejecutara un algoritmo de “diffing” que identifica lo que ha cambiado

## En segundo paso es la reconciliación donde se actualiza el dom real con los resultados de “diff”.

## 

## \*Renderizar es hacer visual algo, pintar la pantalla.

## *NODE.JS*

## Javascript es un lenguaje interpretado, una de las formas de interpretarlo es con NODE.JS.

## NODE.JS es un entorno de ejecución de JS que le permite al código js ser ejecutado en nuestra computadora. Podemos darle node a un archivo de js y este puede ejecutarlo y darles acceso a recursos de nuestra computadora.

## NODE PACKAGER MANAGER / NPM

## Cuando usamos Node.js tenemos que instalar módulos nuevos (librerías) ya que al ser un sistema modular viene “vacío”.

## Por lo tanto, para utilizar una funcionalidad de alguna librería publicada, debemos instalar módulos adicionales. Se hace con la herramienta npm.

## Se utiliza para dos cosas:

## Como repositorio ampliamente utilizado para publicación de proyectos Node.js de código abierto.

## Como una herramienta de línea de comandos, ayuda a instalar y desinstalar paquetes, gestionar versiones, gestionar dependencias necesarias para ejecutar un proyecto.

## 

## *INTERFAZ DE LÍNEA DE COMANDOS (CLI)*

## La interfaz de línea de comandos o interfaz de línea de ordenes es un método que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple.

## React tiene su propia CLI, pero actualmente solo admite la creación de una aplicación: créate-react-app.

## Pasos:

## Abrir consola

## Ubicarnos en el directorio especifico (carpeta) con el comando cd.

## Instalar React js: Debemos ejecutar el comando *npm install -g create-react-app*

## Debemos ejecutar: create-react-app nombre-de-la-app (va a ser el nombre de la carpeta que se va a crear)

## Para instalar y ejecutar a la vez se puede hacer todo desde un comando solo: *npx create-react-app nombre-de-la-app.*

## Para ejecutar la ampliación en el navegador hay que ejecutar *npm start.*

## 

# **JSX Y WEBPACK**

## *SUGAR-SYNTAX*

## Sintaxis agregada a un lenguaje de programación con el objetivo de hacer más fácil y eficiente su utilización. Favorece su escritura, lectura y comprensión.

## Ejemplo: i= i + 1 // i++

## *POLYFILLS Y LA RETROCOMPATIBILIDAD*

## Cuando desarrollemos y pensemos la experiencia de nuestras aplicaciones, es importante tener en cuenta qué distribución tiene hoy el mundo, asi como nuestro target de usuarios.

## 

## 

## Los polyfills nos permiten hacer nuestra aplicación compatible con navegadores antiguos, que no admiten de forma nativa alguna nueva funcionalidad. Hay herramientas que se encargan de traducir para que sea compatible con todos los navegadores, por ejemplo: core.js, babel (compilador de js) opera de fondo si usas React, no hay saber usarlo opera ya con React.

## *BUNDLING CON WEBPACK*

## Webpack es un module bundler o empaquetador de módulos que nació a finales de 2012 y en la actualidad es utilizado por miles de proyectos de desarrollo Front-End. Incluido desde React, angular, hasta en el desarrollo de aplicaciones como Twitter, Instagram, PayPal, o la versión web de WhatsApp.

## 

## Podemos tener, por ejemplo, un modulo JS que vaya a depender de otros módulos .js, con imágenes en diferentes formatos como JPG o PNG o estar utilizando algún procesador de CSS, como puede ser SASS, Less y Stylus.

## Webpack recoge todos estos módulos y los transforma a asset que puede entender el navegador, como por ejemplo archivos JS, CSS, imágenes, videos, etc.

## Webpack lo que hace es facilitar la interacción de todos los módulos, compila los módulos, y lo hace legible para que el navegador los leas. Por esto no se puede levantar desde live server y hay que hacer un host. Live server no funciona porque le navegador no entiende el mensaje. En cambio, levantando con *npm star* se activa el Webpack y queda legible para el navegador.

## EJECT

## Es una acción permanente, que permite tener un control mas específico del bundling, en muy pocos caso se usa el eject, compilar desde React es lo más habitual. Una vez que se ejecuta no hay vuelta atrás. Se ejecuta si queres cambiar la forma en que se da el bundling. Se ejecuta: *npm run eject.*

## *TRANSPILING*

## Es el proceso de convertir código escrito en un lenguaje, a su representación en otro lenguaje. Usualmente extienden o simplifican la escritura del lenguaje, o representación original.

## Implementa un proceso similar conceptualmente pollyfilling

## Logran niveles de simetricidad y simbiosis con el lenguaje original.

## *JSX*

## Javascript xml (JSX) es una extensión de sintaxis de Javascript que se parece a HTML. Oficialmente, es una extensión que permite hacer llamadas a funciones y a constructores de objetos. No es ni una cadena de caracteres ni HTML.

## JSX es una extensión de Javascript no de React. Esto significa que no hay obligación de utilizarlo, pero es recomendado en el sitio oficial de React.

## JSX se transforma en código de Javascript. Esto nos da algunas ventajas, como ver errores en tiempo de compilación, asignar variables, retornar métodos, etc.

## INLINE STYLES EN JSX

## Los mismos estilos se puede configurar inline en JSX, solo necesitamos usar doble llave {{}}.

## La primera llave para avisar que se le agregara un objeto en js.

## La segunda llave para empezar a escribir el objeto en sí.

## Si el elemento no tiene hijos debe ser idealmente auto-cerrado:

## <img src= “ ”/>

## Class es una palabra reservada, en su lugar usar className.

## En JSX se utilizan tanto los estilos como los eventos estándar del DOM usando camelCase. Por ejemplo: onClick, onChange, paddingBottom, etc.

## JSX viene incluido con la instalación de create-react-app.

## COMPONENTES

## Las aplicaciones en React se construyen mediante componentes. El potencial de este funcionamiento consiste en que podemos crear aplicaciones completas de una manera modular de fácil mantenimiento, a pesar de su complejidad.

## 

## 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, etc.

## En React existen dos maneras de entender los componentes, que varían según desde donde nos paremos para analizarlo. Vamos a decir que existen tipos de componentes y patrones.

## Los dividiremos en estas dos representaciones, que después servirán de base para implementar múltiples patrones.

## 

## Un componente es una función que retorna un bloque de código de jsx.

## Pueden recibir propiedades (props)

## Tienen la capacidad de hacer render de un único elemento.

## Los componentes van con mayúscula. Por lo general el archivo se llama igual que el componente y va con mayúscula también.

## PROPIEDADES

## Las propiedades son la forma que tiene React para pasar parámetros de un componente superior a sus children.

## Es la mejor manera de implementar el flujo de datos unidireccional.

## Si alguna prop es una función, el child component puede llamarla para provocar efectos secundarios en el parent.

## Son valores que pueden percibir de un elemento padre.

## 

## 

## This.props.nombre contendrá el valor pasado en la propiedad “nombre” y this.props.app el valor de la propiedad “app”.

## Nuestras propiedades se encuentran encerradas entres llaves {}.

## Las llaves son importantes, porque es la manera con la que se escapa un código JSX, permitiendo colocar dentro sentencias Javascript “nativo”.

## Aquello que devuelvan esas sentencias se volcará como contenido en la vista.

## PATRONES

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

## COMPONENTES DE PRESENTACIÓN

## Orientados al aspecto visual

## No tienen dependencia con fuentes de datos

## Reciben callbacks por medio de props.

## Pueden ser descritos como componentes funcionales.

## Normalmente no tienen estado.

## Los componentes de presentación usualmente no tienen estado, por eso hace mas sentido utilizar mas simplentente function based componentes. Todo componente puede recibir de padre (superior) props y children.

## 

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

## 

## 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 también llamados state components).

## Orientados al funcionamiento de la aplicación.

## Contienen componentes de presentación y/u otros contenedores.

## Se comunican con fuentes de datos

## Usualmente tienen estado para representar el cambio en los datos.

## 

## Este tipo de componentes será el encargado de realizar llamadas a las APIS externa y/o establecer la lógica a realizar, en función de las acciones que realice el usuario sobre la interfaz.

## CHILDREN

## Children es una manera que tiene React de permitirnos proyectar/traslucir uno o mas componentes dentro del otro.

## <Component>

## <coldComponent/>

## </Component>

## Es ideal cuando:

## Necesitamos que un elemento quede dentro de otro sin que sepan el uno del otro.

## Necesitamos implementar patrones más complejos.

## 