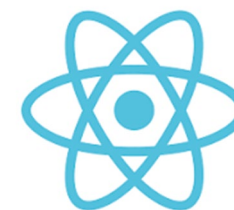
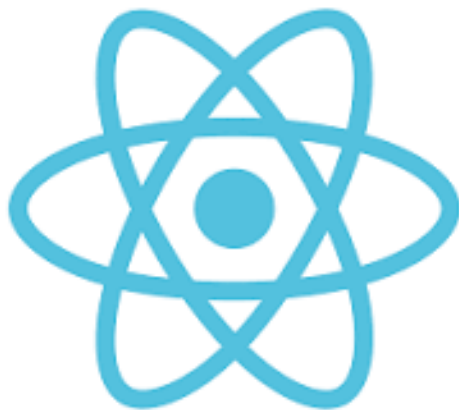


REACT

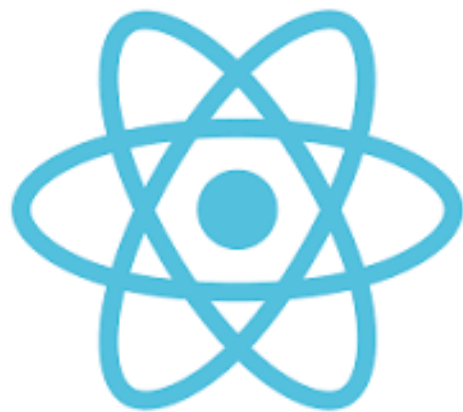
Programación Web Avanzada - UTN



- Sitios oficiales de React
- <https://reactjs.org/>
- <https://es.reactjs.org/>
- <https://es.reactjs.org/docs/hello-world.html>
- <https://es.reactjs.org/docs/create-a-new-react-app.html>



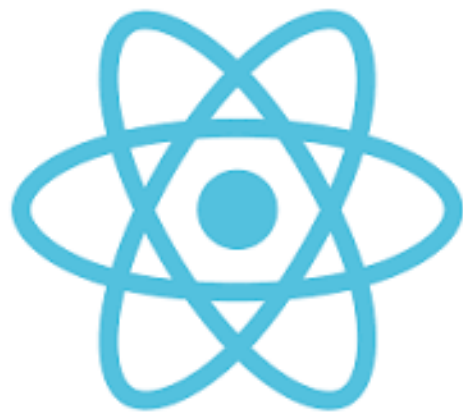
React JS fue creada por Jordan Walke, un ingeniero de software en Facebook, inspirado por los problemas que tenía la compañía con el mantenimiento del código de los anuncios dentro de su plataforma. React intenta ayudar a los desarrolladores a **construir aplicaciones que usan datos que cambian todo el tiempo**. Su objetivo es ser **sencilla, declarativa y fácil de combinar**.



¿Cómo llega React a la performance que tanta fama le trae?

Hablemos de tres conceptos:

Virtual DOM vs **React Fiber** y la **Reconciliación**



Primera premisa

El acceso indiscriminado al DOM **es caro**, entonces se requirió encontrar una manera de realizarlo de la manera **más óptima** posible.



Zona de
evento

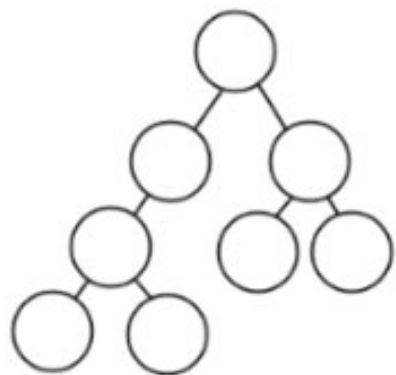
Primera premisa: Optimicemos los movimientos

V-DOM

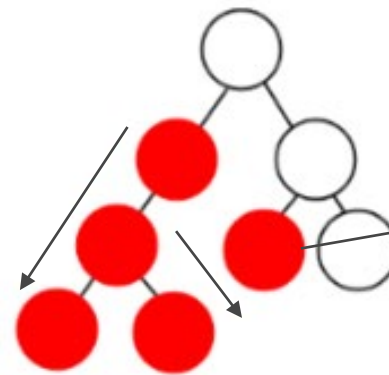
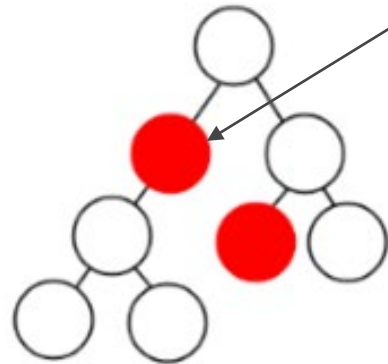
Nuevo V-DOM

Reconciliado

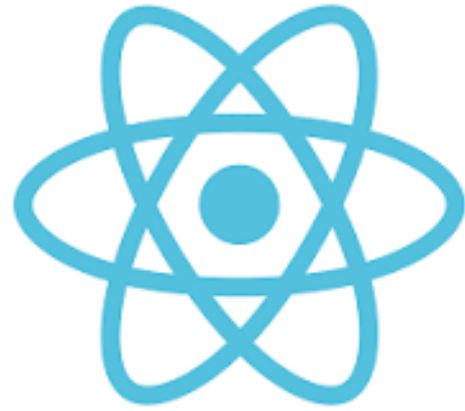
Fiber
tree



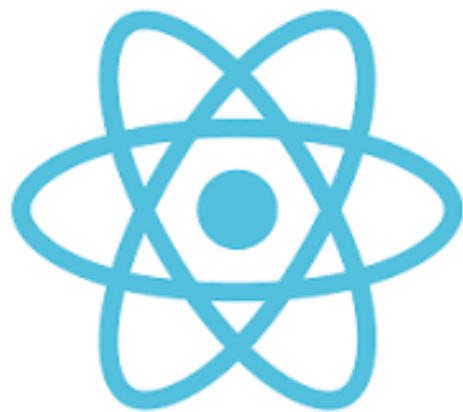
Estado
original



Cambio aislado



En vez de aplicar **uno a uno** los cambios en los cinco nodos, **React** procesa este resultado en una memoria. 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.

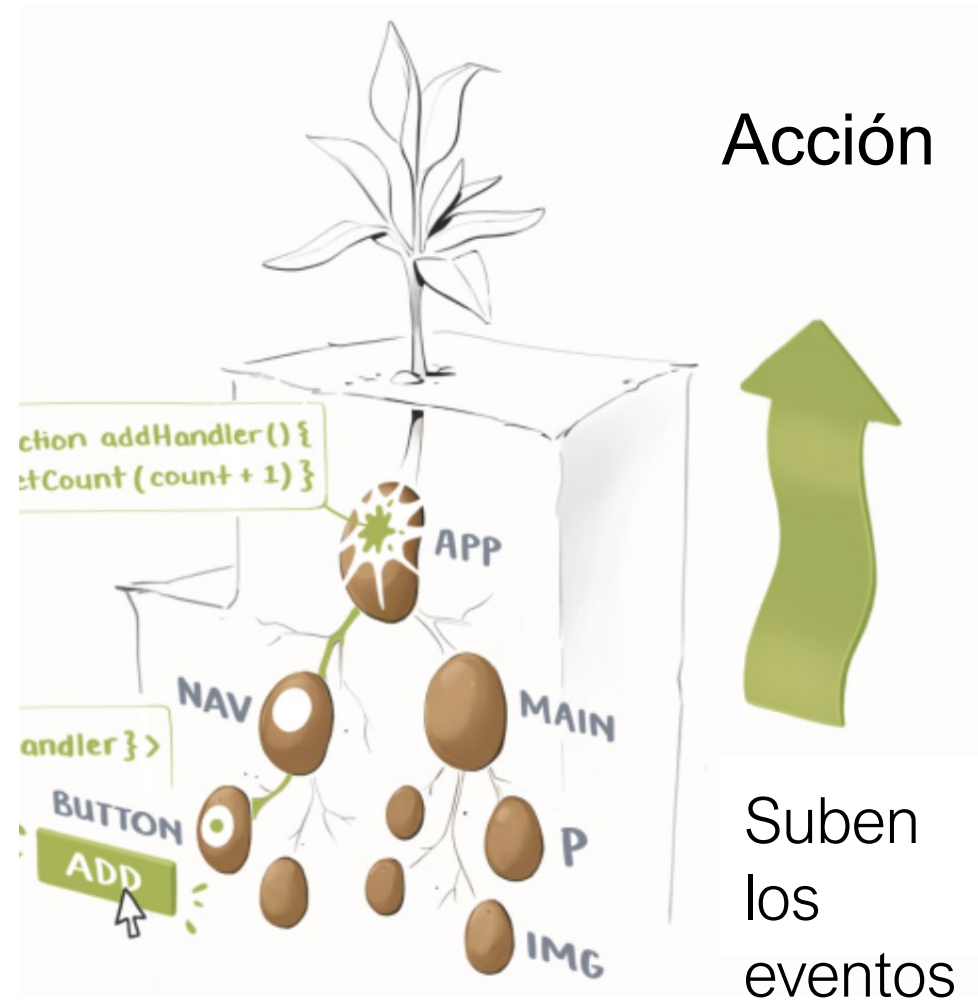
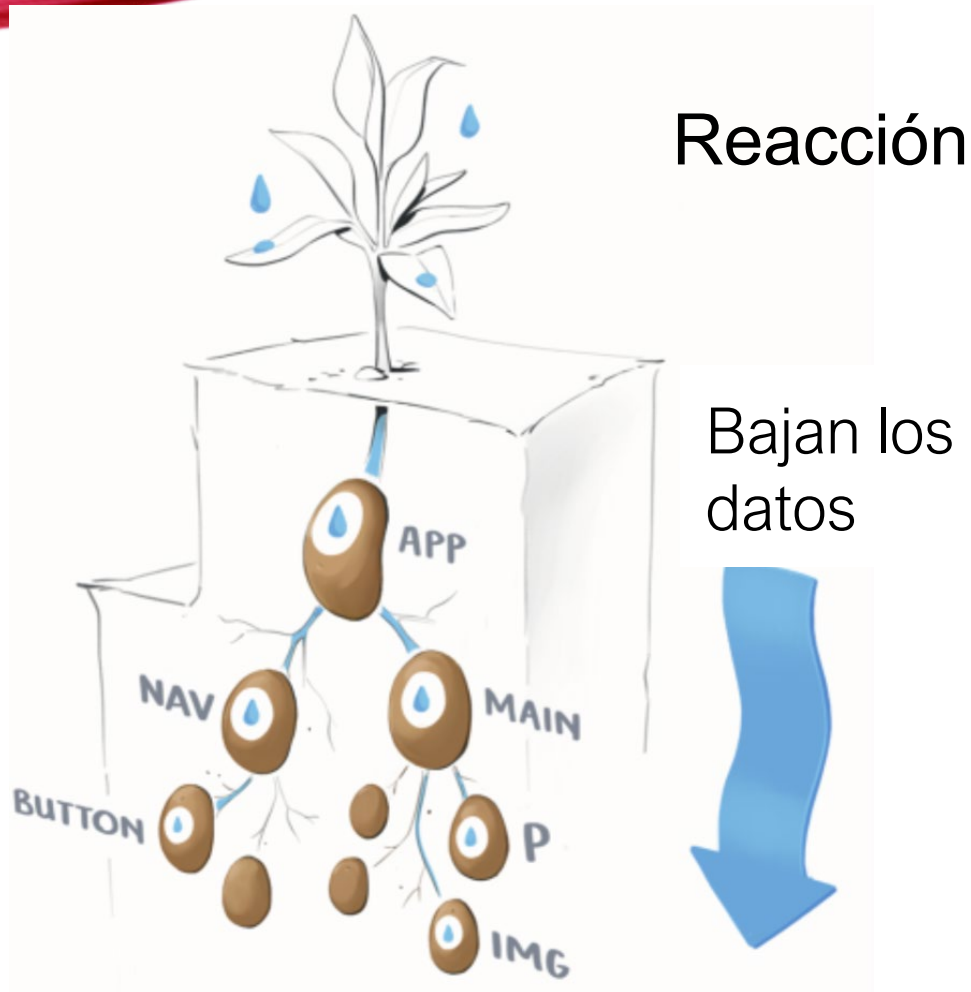


Segunda premisa: Flujo de datos unidireccional

Para establecer esa seguridad, requiere que los datos y los cambios idealmente se provoquen de una manera específica con dos características:

Unidireccionalidad / De arriba **hacia abajo**

Flujo de datos



Ilustradora: [Maggie Appleton](#) @ Woman of React 2020

Virtual DOM

Es un patrón de comportamiento y **React** lo implementa con una tecnología llamada “**Fiber**”.

En sí resulta ser todo lo que React sabe de tu aplicación y cada nodo o **fibra**.

Esto es básicamente lo que React hace con el Virtual DOM: **una representación virtual de la IU que se mantiene en memoria y en sincronía “reconciliado” con el DOM “real”**.

Virtual DOM

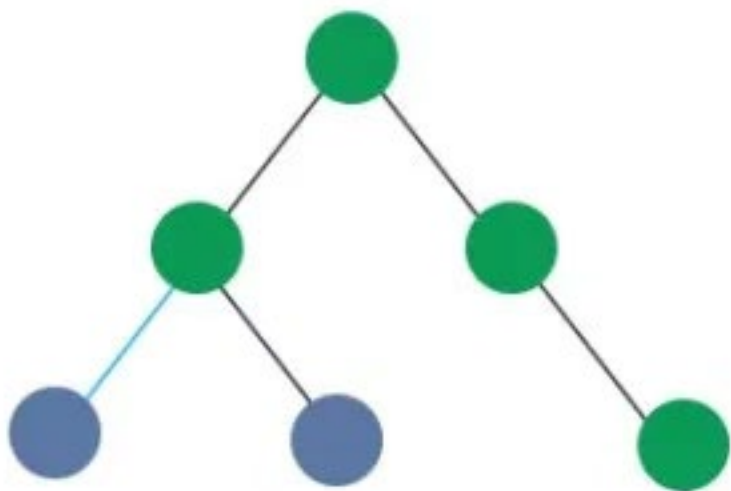
Resumiendo el proceso

- En primer lugar, React ejecuta un algoritmo de “**diffing**” que identifica lo que ha cambiado.
- El segundo paso es la **reconciliación**, donde se actualiza el **DOM** con los resultados de **diff**.

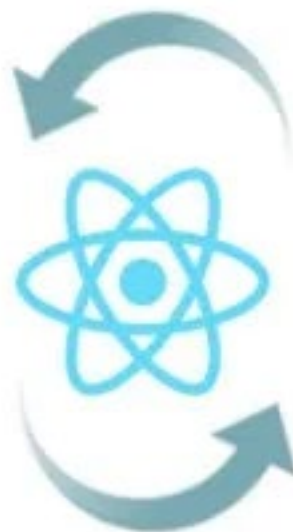
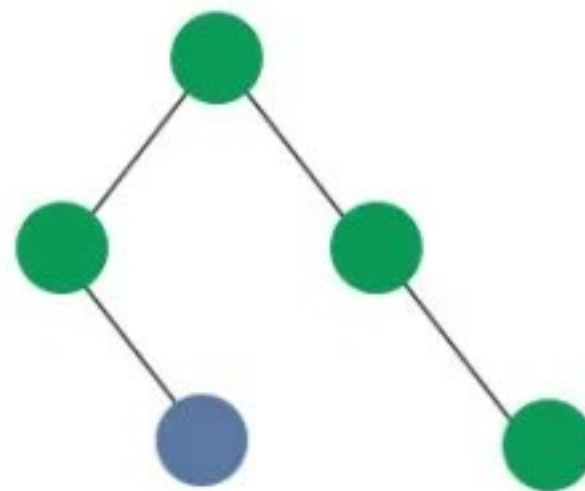
React se encarga de todo esto, nosotros solo aprenderemos a ayudarlo

Virtual DOM

Virtual DOM



Real DOM



Instalar React JS

Debemos ejecutar el comando `npm install -g create-react-app`

```
G:\sites\react>npm install -g create-react-app
```

Crear la aplicación

Debemos ejecutar `create-react-app my-app`

```
G:\sites\react>create-react-app my-app  
Creating a new React app in G:\sites\react\my-app.
```



NPX - TODO EN UN COMANDO - eXecute

```
npx create-react-app nombre-de-app  
cd nombre-de-app
```


Ejecutar aplicación en el navegador

```
G:\sites\react\utn>npm start

> utn@0.1.0 start G:\sites\react\utn
> react-scripts start
Starting the development server...
Compiled successfully!

You can now view utn in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.0.11:3000/

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Para ejecutar una aplicación y poder acceder desde el navegador debemos ejecutar (dentro del directorio de la aplicación creada) `npm start`

Ahora escribe la dirección obtenida (Ej:localhost:3000) en tu navegador y el resultado obtenido será el siguiente:



Welcome to React

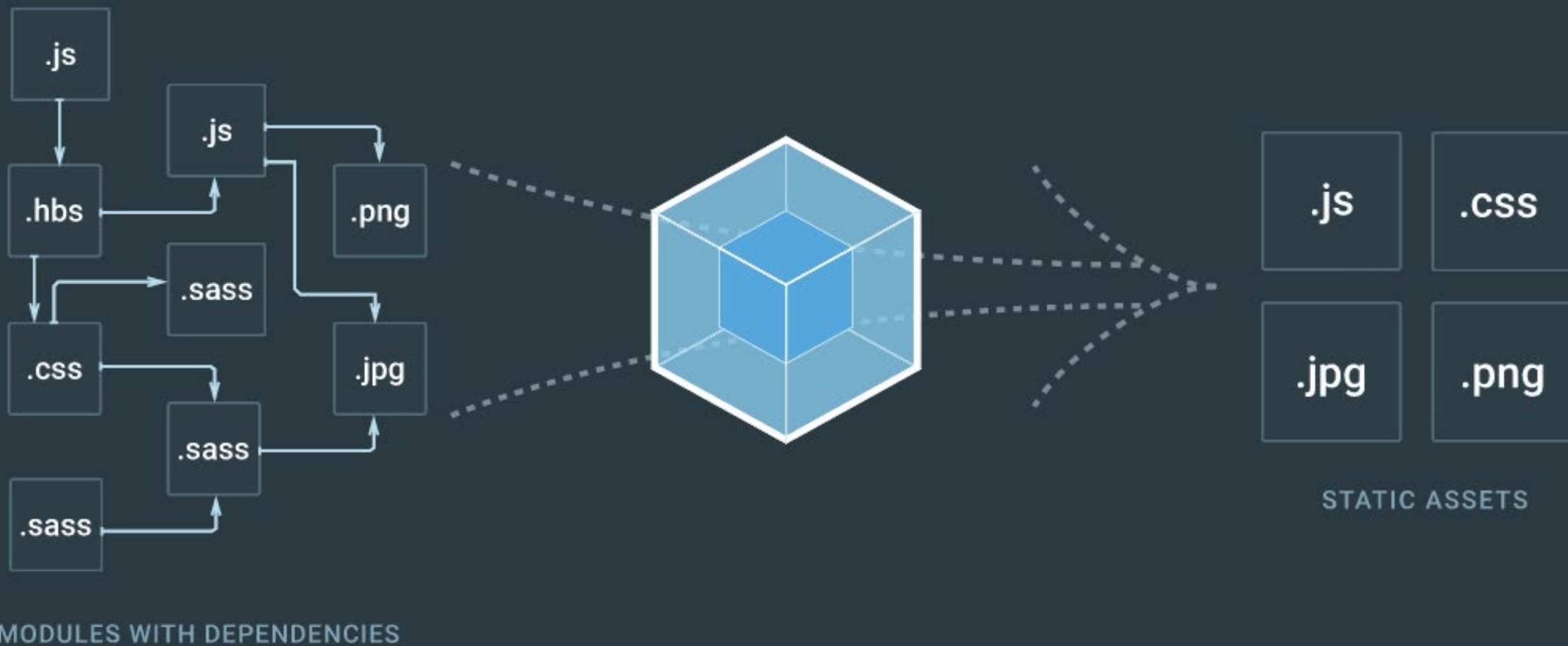
To get started, edit `src/App.js` and save to reload.



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 web **Front-End**.

Incluido desde React o Angular hasta en el desarrollo de aplicaciones conocidas como Twitter, Instagram, PayPal, o la versión web de Whatsapp.



Transformación de los módulos en Webpack

¿Cómo funciona?

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

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



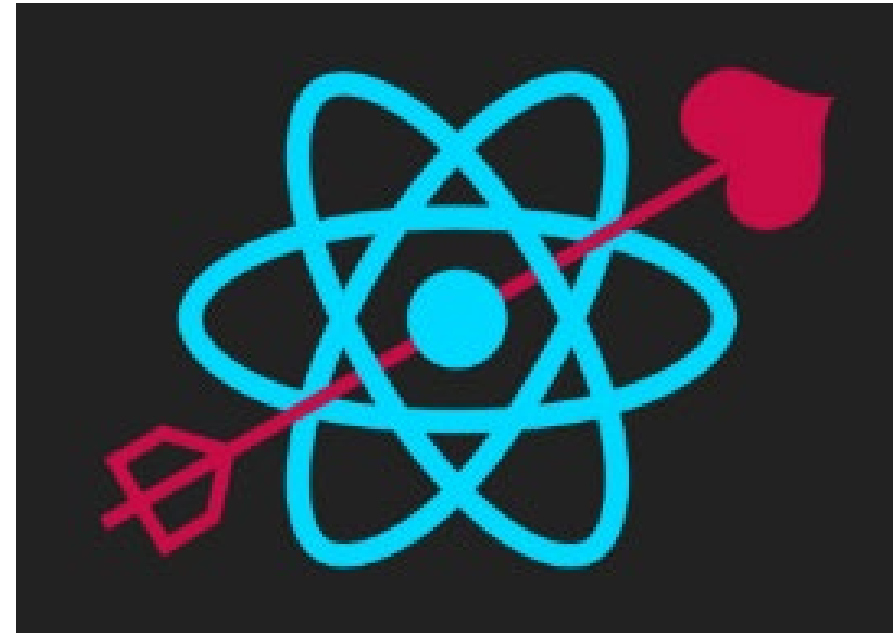
javascript **x**ml

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 construcción 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 web oficial de React.



¿Cómo funciona?

JSX se transforma en código JavaScript

```
07  
08  
09 <div className="active">Hola Coders</div>  
10  
11  
12 React.createElement('div', { className: 'active'}, 'Hola Coders');  
13  
14
```

Esto nos da algunas ventajas, como ver errores en tiempo de compilación, asignar variables, retornar métodos, etc.

Styling en JSX

Es posible definir y utilizar **estilos inline** en JSX, solo necesitamos convertirlos por convención:

border-color => borderColor
padding-top => paddingTop
'10px' => 10 (No es necesario el px)

```
104  
105  
106 let styles = {  
107   |   borderColor: '#999'  
108   | };  
109  
110 const jsx = (  
111   |   <div style={styles}>  
112   |     Hola Coders  
113   |   </div>  
114   | )
```

Inline styles en JSX

Los mismos estilos se pueden configurar **inline** en JSX, solo necesitamos usar doble llave {{ }},

- La **primera** llave para avisar que se agregará **un objeto** en **js**
- La **segunda** llave para empezar a escribir el objeto en sí.

```
const Salute = () => <p style={{ marginLeft: 15}}>Hello</p>
```

Reglas generales

Los elementos deben ser balanceados.

Por cada apertura debe haber un cierre.

`` Mal

`` Es mejorable

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

`` Ideal

Reglas generales

Class es palabra reservada, en su lugar usar className

`` Mal

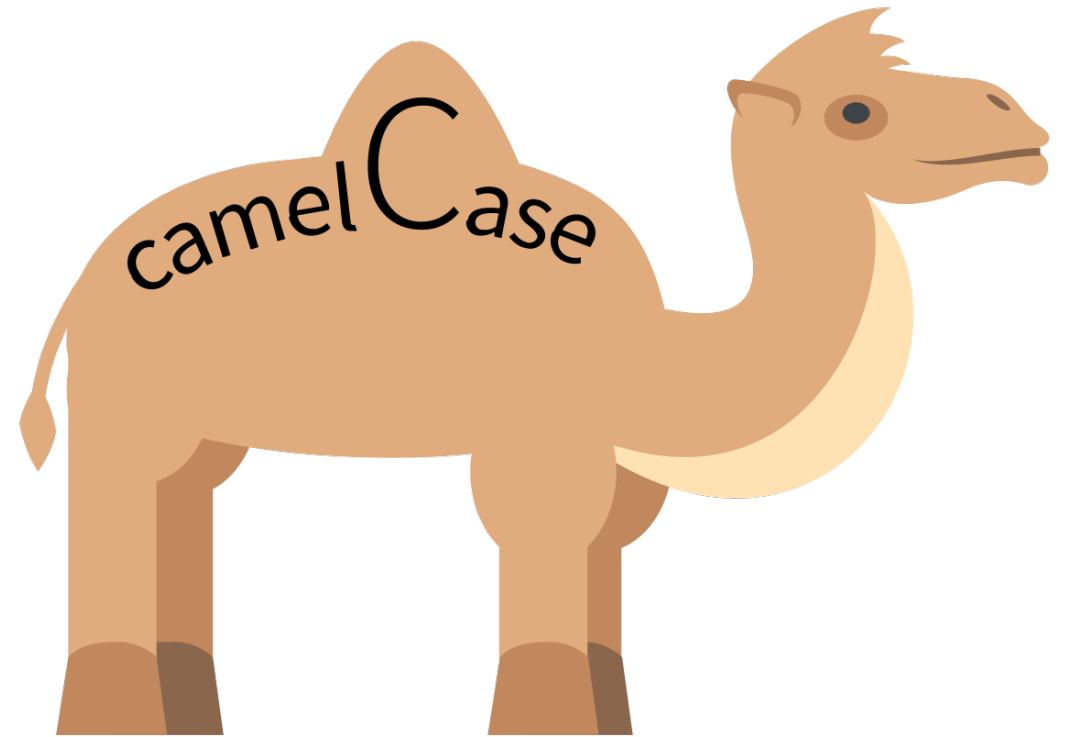
`` Ok

```
return (
  <h1 className='large'>Hello World</h1>
);
```

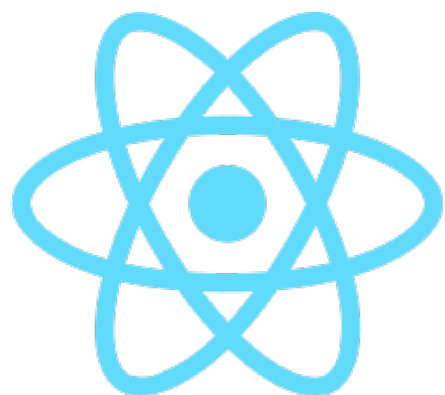


¡No olvidemos!

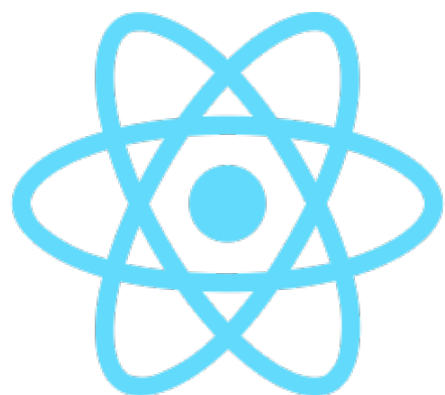
En JSX se utilizan tanto los estilos como los eventos estándar del DOM, como **onclick**, **onchange**, **onkeydown**, etc. pero utilizando **camelCase**:
onClick, onChange,
onKeyDown / marginTop,
paddingBottom, etc.



A medida que nuestra aplicación va creciendo y tenemos componentes más grandes que manejan distintos eventos, JSX nos va a ayudar mucho a agilizar y organizar nuestro desarrollo de componentes.

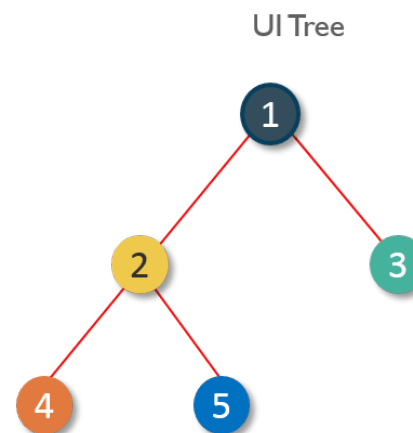
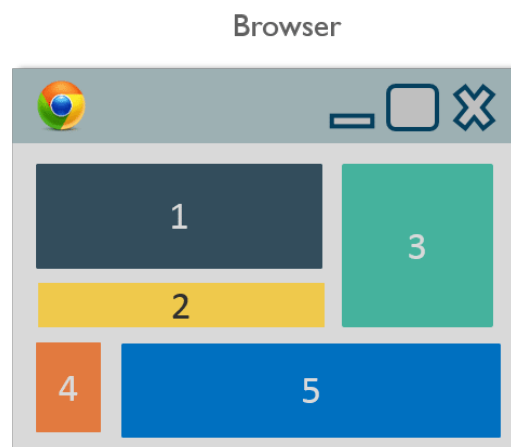


Para poder utilizar JSX en nuestra aplicación, debemos tener instalado **Webpack** que en nuestro caso viene incluido con **create-react-app**



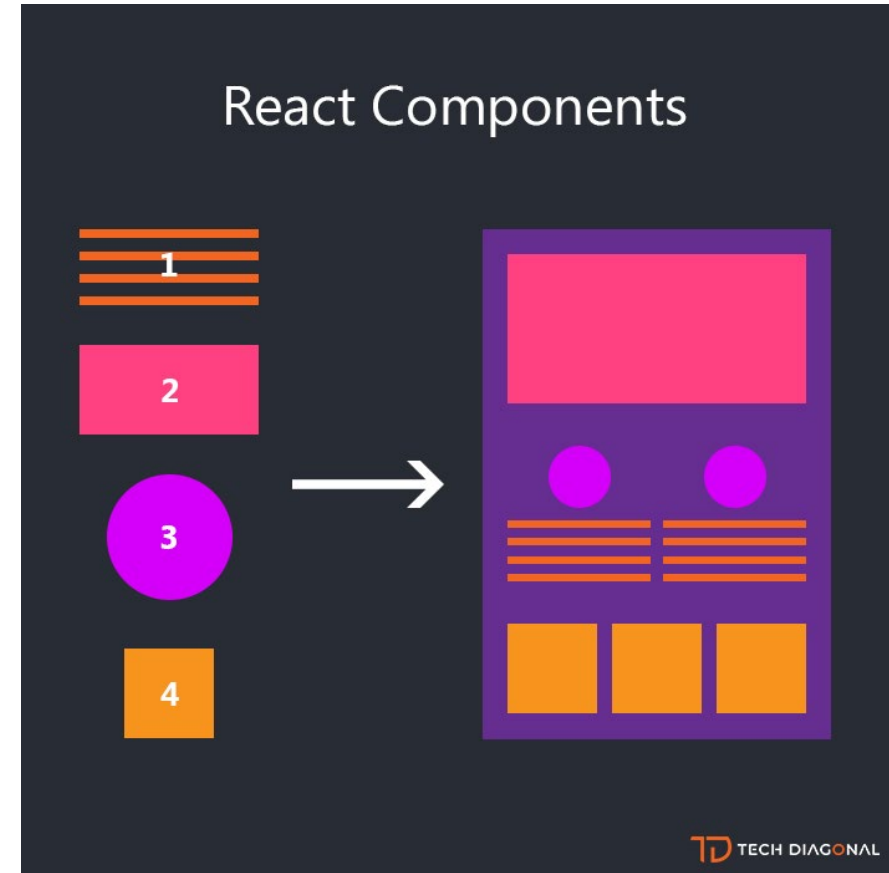
Las aplicaciones en React básicamente se construyen mediante **componentes**.

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



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 resolver, visualizar y comprender.

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

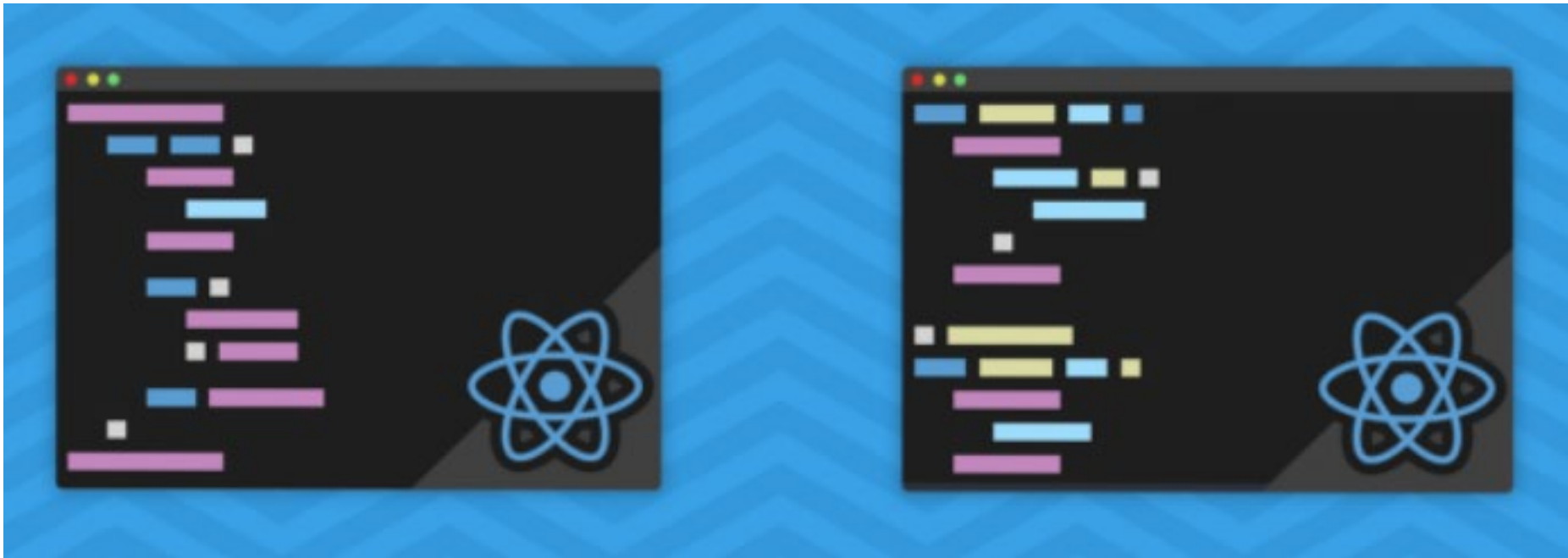
Este modelo de trabajo tiene varias ventajas, como la facilidad de mantenimiento, depuración, escalabilidad, etc.

Ventajas del enfoque

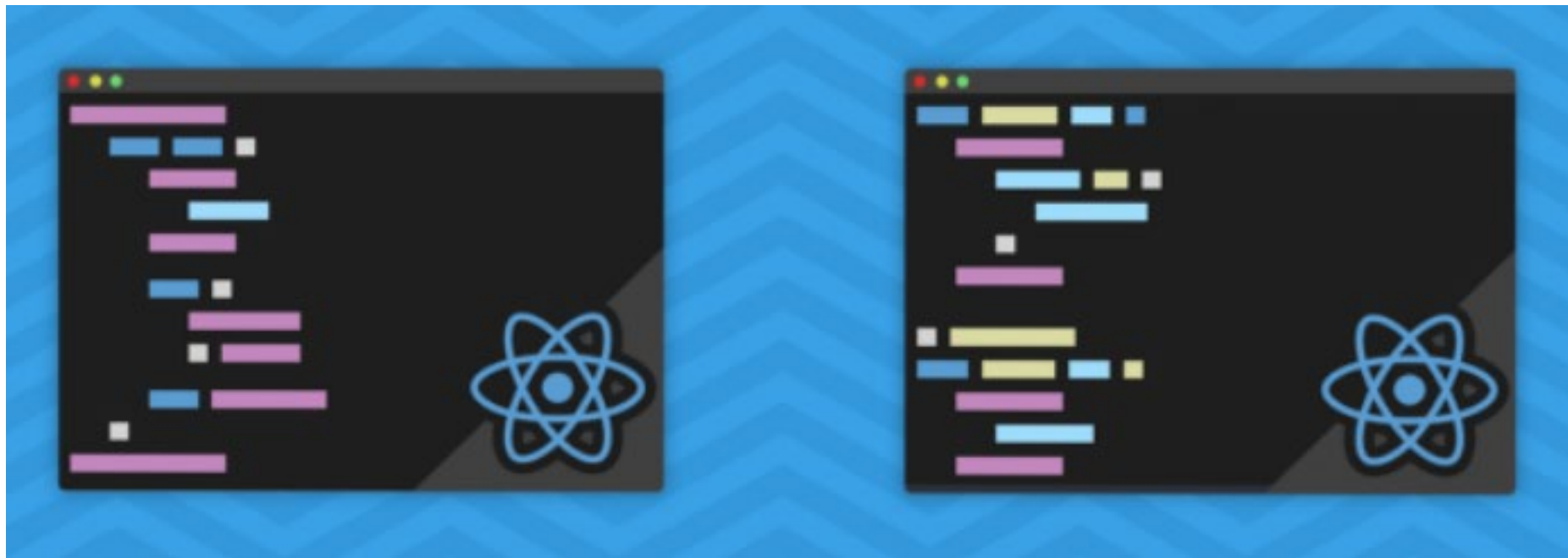
- Favorece la **separación de responsabilidades**: cada componente debe tener una única tarea.
- Al tener la lógica de estado y los elementos visuales por separado, es **más fácil reutilizar** los componentes.
- Se simplifica la tarea de hacer **pruebas unitarias**.
- Puede **mejorar el rendimiento**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

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**



La confusión se acentúa cuando no somos capaces de identificar las diferencias.



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

Class based components

componentes basados en clases

```
class App extends Component {  
  render() {  
    return (  
      <p>  
        Class based :)  
      </p>  
    );  
  }  
}  
  
render(<App />, document.getElementById('root'));
```

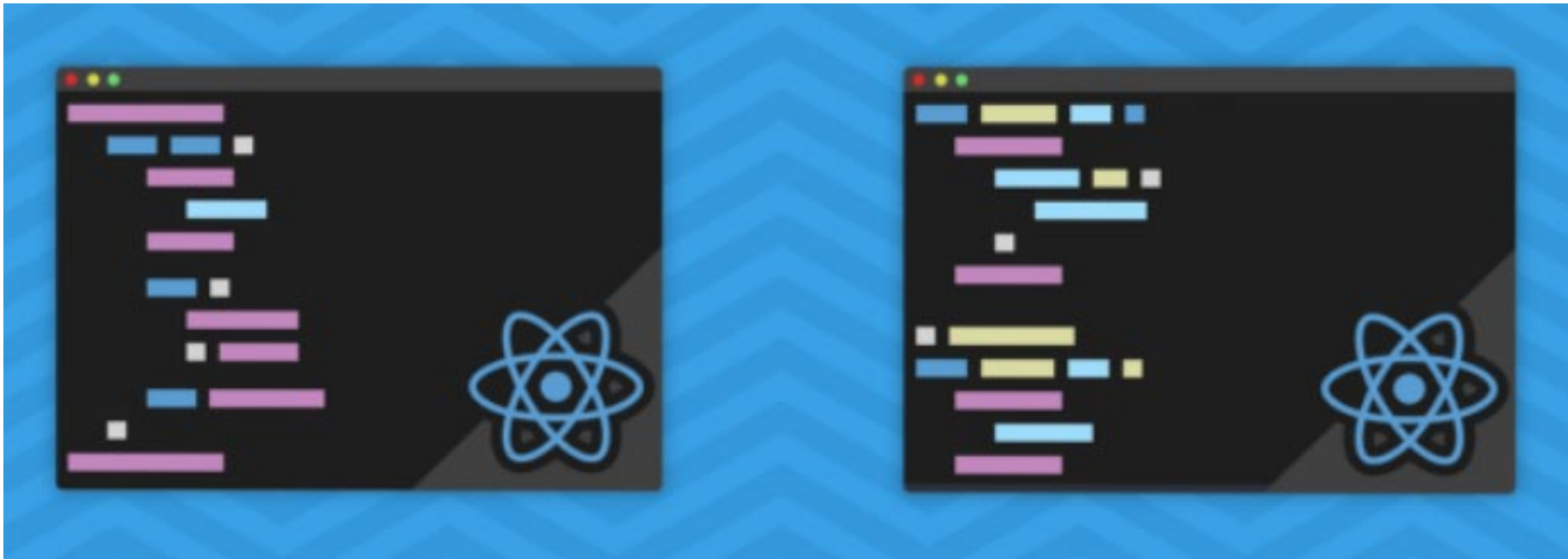
Function based components

componentes basados en funciones

```
const App = () => {  
  return (  
    <p>  
      Function based :)  
    </p>  
  );  
}  
  
render(<App />, document.getElementById('root'));
```


Puntos en común

- Pueden recibir propiedades (**props**)
- Tienen la capacidad de hacer **render** de un único elemento*



* aunque este elemento pueda tener muchos elementos dentro ;)

Propiedades

Class based components componentes basados en clases	Function based components componentes basados en funciones
--	--

Las propiedades enviadas al componente las recibiremos a través de **this.props** para acceder a un objeto en el cual tendremos todas las propiedades disponibles.

Simplemente se reciben como parámetro de la función

```
( { name } ) => <p>{name}</p>
```

Imaginemos que a nuestro componente le pasamos dos propiedades, llamadas **"nombre"** y **"app"**.

Entonces podremos usar esas propiedades de la siguiente manera:

```
112 import React, { Component } from 'react';
113
114
115
116 export default class FeedbackMessage extends Component {
117   render() {
118     return (
119       <div>
120         Bienvenido {this.props.nombre} a {this.props.app}
121       </div>
122     )
123   }
124 }
125
```

Propiedades

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 entre 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.

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***).

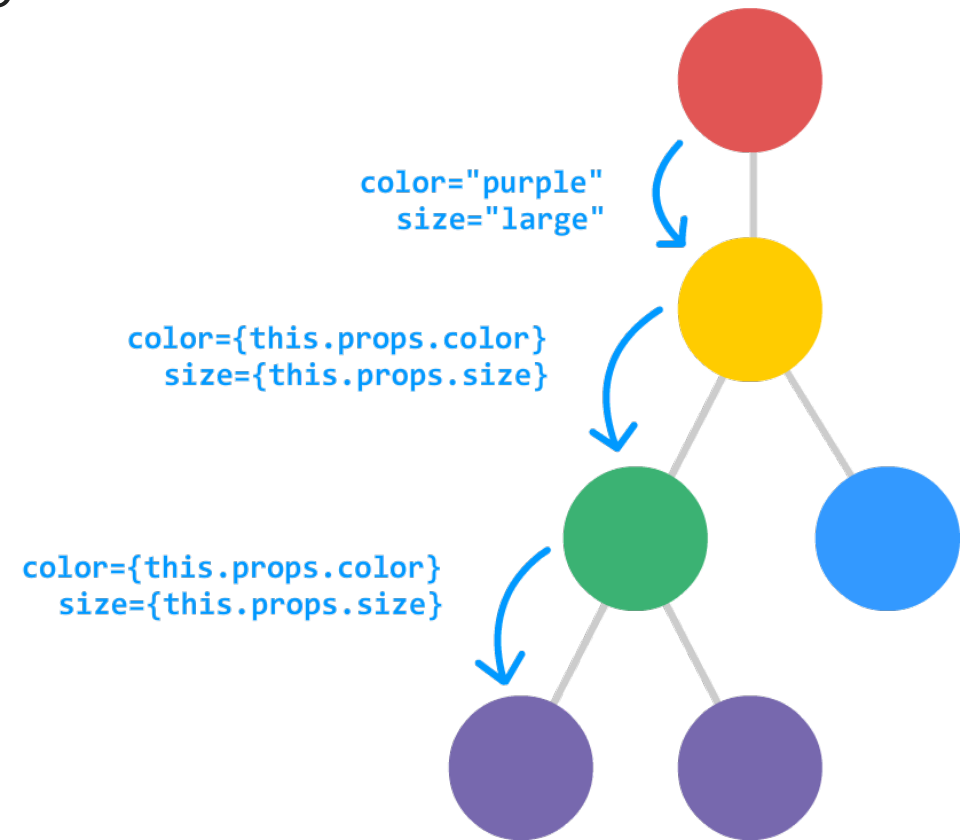
Componentes de presentación

- 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 más sentido utilizar más simplemente **function** based componentes.


Todo componente puede recibir de su **parent** (superior), **props** y **children**.

(Aunque no sea obligatorio)



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

```
97  
98   const Titulo = ({nombre} = props) => (  
99     <h1>{ nombre }</h1>  
100   );  
101  
102   const Item = (props) => (  
103     <li><a href='#'>{ props.valor }</a></li>  
104   );  
105  
106   const Input = (props) => (  
107     <input type='text' placeholder={ props.placeholder} />  
108   );  
109
```



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.