




REACT.js

Master Class



The image features abstract geometric shapes in the corners. On the left, there are overlapping shapes in green, blue, orange, and purple. On the right, there are overlapping shapes in green, blue, purple, and orange. The central text is in a dark blue, sans-serif font.

¿Qué es
React ?

Decorative geometric shapes in the top-left corner, including a green triangle, a blue triangle, and a red triangle, all pointing towards the center.

React.js es una **librería** Javascript *open source* desarrollada por Facebook, y focalizada en el desarrollo de **interfaces de usuario**.

Es la V del MVC.

Decorative geometric shapes in the top-right corner, including a green triangle, a blue triangle, and a red triangle, all pointing towards the center.

Ecosistema React

Al ser solo una librería deja de lado muchas otras soluciones que nos aportan los frameworks.

Sin embargo existe todo un ecosistema de herramientas, aplicaciones y librerías que al final equiparan React a un framework.

The logo for ES6 (ECMAScript 2015) features the text "ES6" in a bold, black, sans-serif font, centered within a solid yellow square.

ES6

The Babel logo consists of the word "BABEL" written in a stylized, yellow, hand-drawn script font with black outlines and shadows, giving it a three-dimensional appearance.

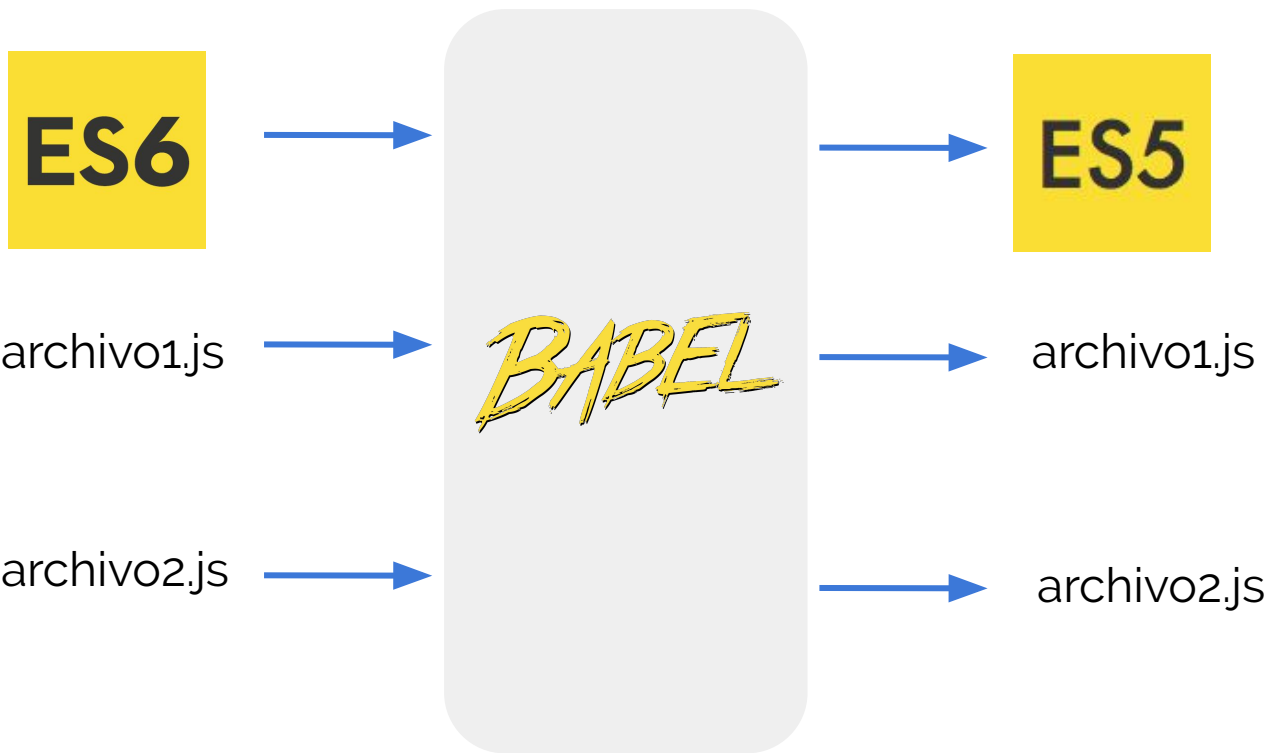
BABEL

The React Router logo features a stylized icon above the text "REACT ROUTER". The icon is composed of three black dots and a light blue shape resembling a three-lobed flower or a stylized 'R'. The text "REACT" is in black and "ROUTER" is in light blue, both in a bold, sans-serif font.

REACT ROUTER

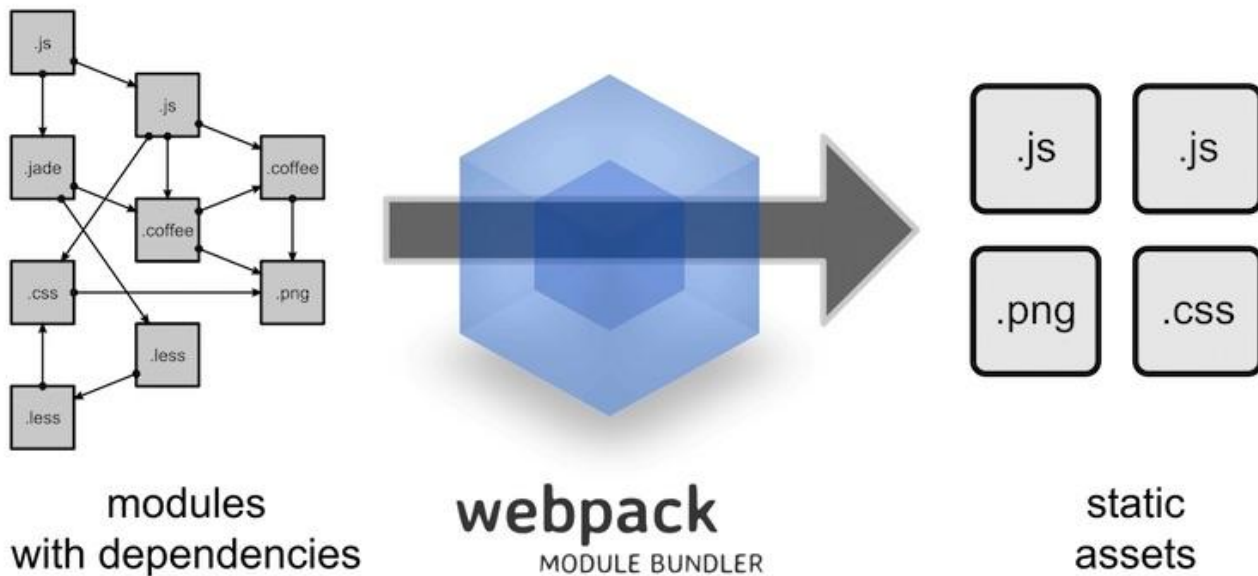
The Redux logo features a purple icon above the word "Redux". The icon is a stylized, continuous purple line forming a shape that resembles a three-lobed flower or a stylized 'R'. The word "Redux" is in a bold, black, sans-serif font.

Redux



Webpack

Es un empaquetador de módulos, te permite generar **UN SOLO** archivo con todos aquellos módulos que se necesitan.



BABEL



WEBPACK



JS



Abstract geometric shapes in the top-left corner, including a green triangle, a light blue parallelogram, a brown trapezoid, and a large purple parallelogram.

¿Cómo trabaja React?

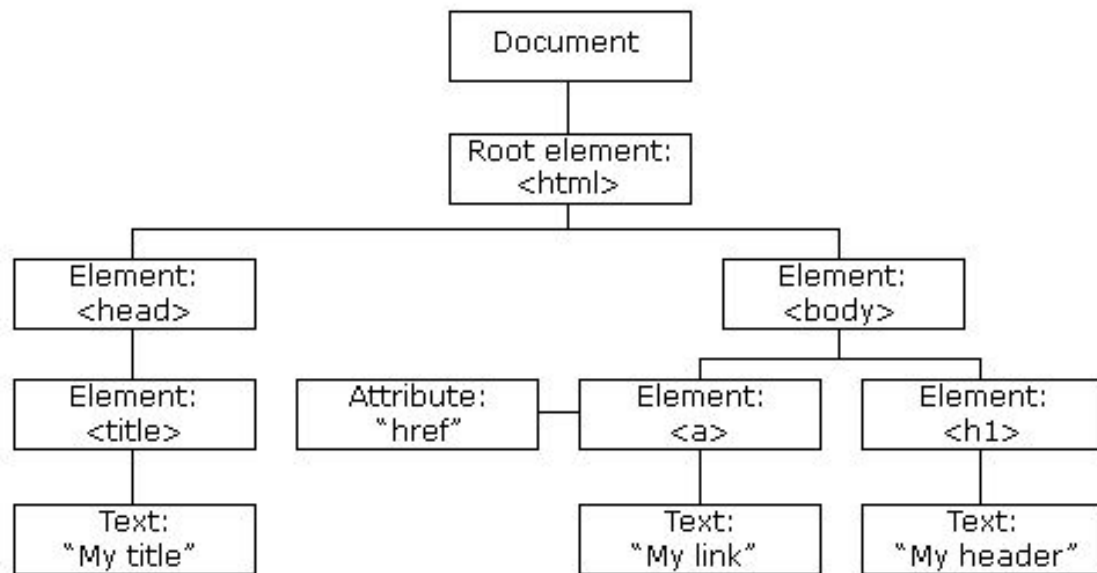
VIRTUAL DOM

Una abstracción del DOM.

Abstract geometric shapes in the top-right corner, including a green triangle, a light blue parallelogram, a purple parallelogram, and a red triangle.

DOM

Es una representación estructurada del documento HTML y define de qué manera los programas pueden acceder al fin de modificar, tanto su estructura, estilo y contenido.

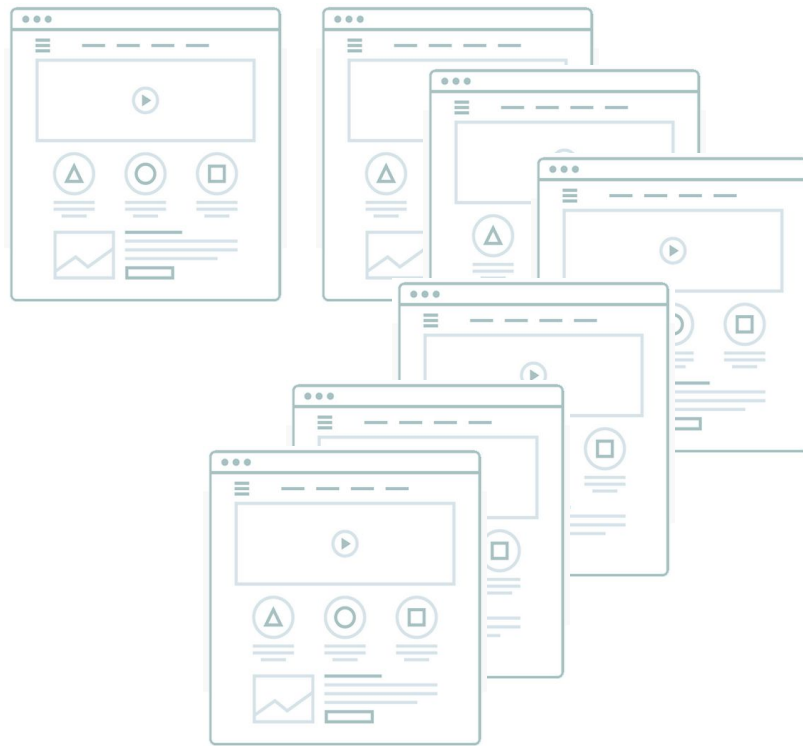


DOM



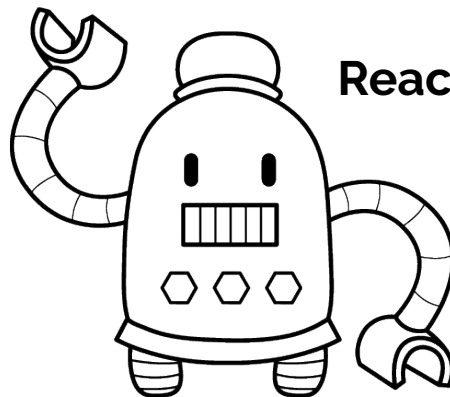
DOM

¿Qué sucede si se reconstruye el DOM cada vez que hay cambios?



React implementa **Virtual DOM**.

React crea una copia ligera del DOM y en cada cambio lo compara con el DOM Real. En lugar de renderizar el DOM completo en cada cambio, los aplica exclusivamente en las partes que varían.



ReactJS

NPM

<https://nodejs.org>

NPM (Node Package Manager) es un **gestor de paquetes** de **Javascript** de Node.js.

Por medio de esta herramienta podemos crear, compartir y reutilizar módulos en nuestras aplicaciones de forma sencilla.

NPM

<https://nodejs.org>

1. Instalar NPM (nodejs.org)
2. Instalar paquete create-react-app con NPM
3. Crear proyecto React con create-react-app

facebook ofrece un paquete para crear una aplicación rápida en React sin tener que preocuparnos de las configuraciones Webpack y todas las que involucran para hacerlo funcionar.

```
npm install -g create-react-app
```

facebook ofrece un paquete para crear una aplicación rápida en React sin tener que preocuparnos de las configuraciones Webpack y todas las que involucran para hacerlo funcionar.

```
npm install -g create-react-app  
create-react-app app-react
```


facebook ofrece un paquete para crear una aplicación rápida en React sin tener que preocuparnos de las configuraciones Webpack y todas las que involucran para hacerlo funcionar.

```
$ npm install -g create-react-app  
$ create-react-app app-react  
$ cd app-react
```

facebook ofrece un paquete para crear una aplicación rápida en React sin tener que preocuparnos de las configuraciones Webpack y todas las que involucran para hacerlo funcionar.

```
$ npm install -g create-react-app  
$ create-react-app app-react  
$ cd app-react  
$ npm start
```

\$ npm start

Importante: debemos estar dentro de la carpeta de nuestro proyecto, ya que la ejecución de **npm start** busca el archivo **package.json** y dentro de él, un script con la clave start y ejecuta el comando especificado, en nuestro caso, nos permite correr (levantar) nuestra aplicación.

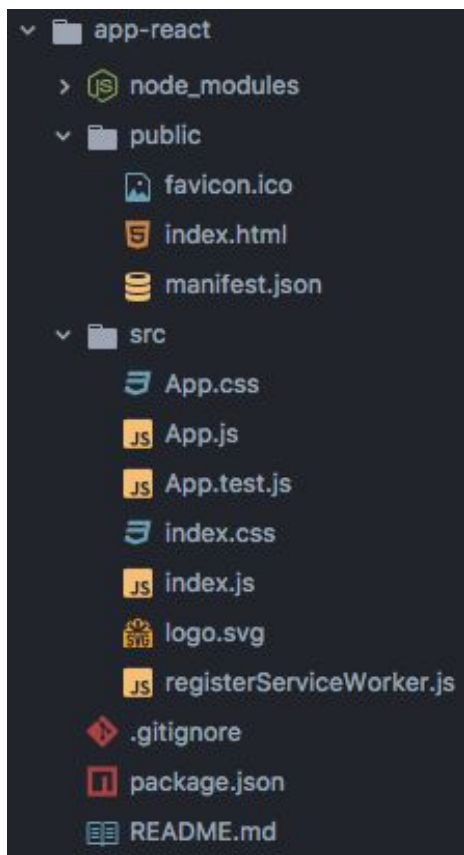
Y... ¿qué acabamos de hacer?

create-react-app nos descarga un conjunto de paquetes para comenzar rápidamente con una aplicación basada en **React**. Incluye:

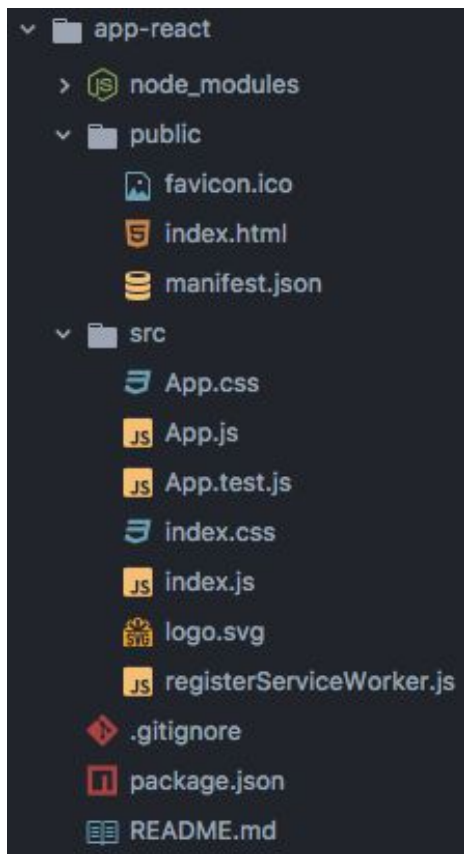
- Librerías de React
- Ecosistema Babel+Webpack configurado
- HMR (Hot Module Replacement) **¿WTF?**
- ¡Y varios módulos más!

Ecosistema de REACT

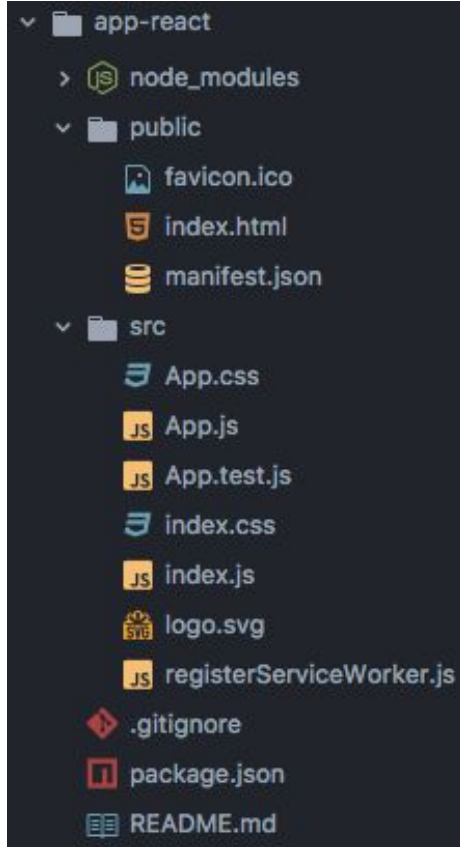




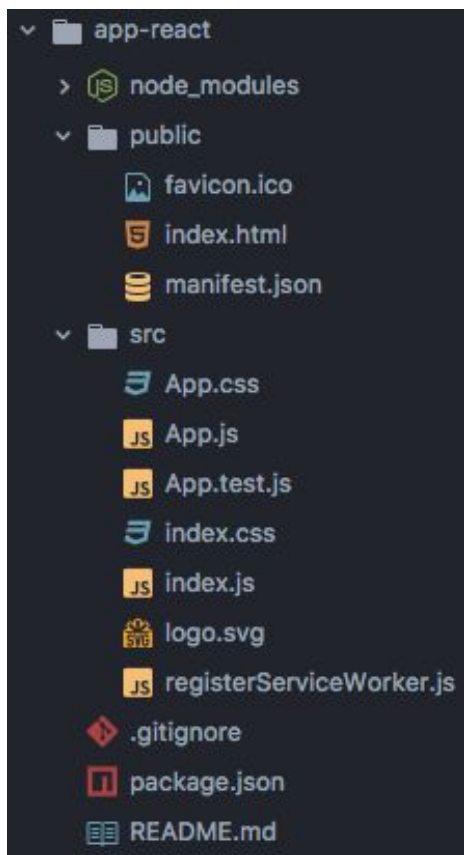
En **package.json** se especifican las dependencias y las versiones de los paquetes de las que depende el proyecto.



En **node_modules**
están todos los
paquetes de node.js
instalados para el
proyecto en React.



En la carpeta **public** se encuentra el archivo **index.html** que es el archivo html principal que se va cargar cuando el usuario ingresa a la url de nuestra aplicación.



En **src** están **todos** los archivos donde vamos a trabajar en nuestro proyecto en React.

Atentis al **index.js**

src/index.js



```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(<App />, document.getElementById('root'));
registerServiceWorker();
```

```
ReactDOM.render(element, document.getElementById('root'));
```

Y ¿ese **index.js** qué es?

index.js es nuestro punto de entrada a toda la aplicación. El mismo se encarga de *renderear* nuestro componente principal (**App.js**) y a su vez es quien carga con la responsabilidad de permitir que el VIRTUAL DOM visualice los cambios que se han presentado, para de esta manera generar un DOM real dentro del navegador.

src/App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

?

src/App.js

ES6

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

src/App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```



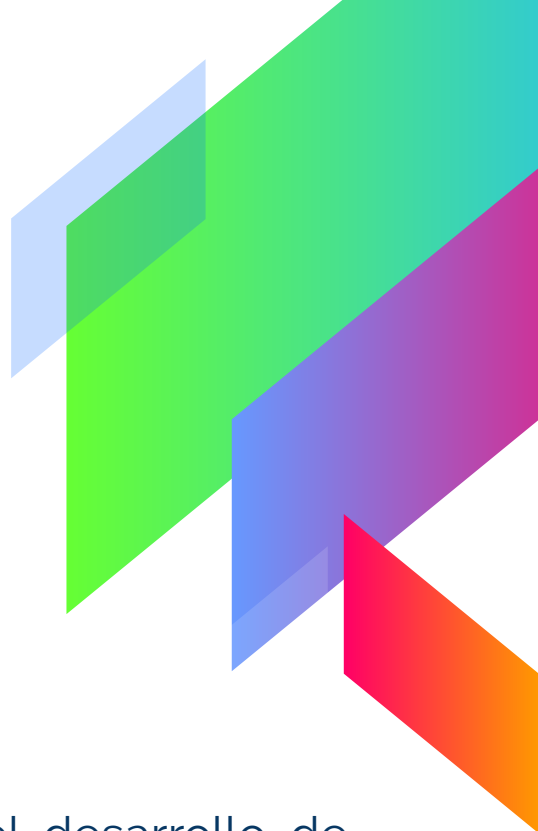
The slide features abstract geometric shapes in the corners. The top-left and bottom-left corners have overlapping shapes in shades of green, blue, orange, and purple. The top-right and bottom-right corners have overlapping shapes in shades of green, blue, purple, and orange. The main content is centered on a white background.

Componente

Nos permiten desglosar el desarrollo
de aplicaciones web en pequeños
contenedores reusables

A cluster of overlapping, semi-transparent geometric shapes in shades of green, blue, orange, and purple, arranged in a dynamic, angular pattern.

JSX

A cluster of overlapping, semi-transparent geometric shapes in shades of green, blue, purple, and orange, arranged in a dynamic, angular pattern.

JSX es una extensión de ECMAScript.
Es un pseudolenguaje que facilita el desarrollo de aplicaciones para crear elementos en el DOM gracias a su sintaxis muy parecida al XML.

JSX

```
var nav = <ul id="nav">  
  <li><a href="#">Home</a></li>  
  <li><a href="#">About</a></li>  
  <li><a href="#">Contact Us</a></li>  
</ul>
```

JSX

```
var nav = <ul id="nav">  
  <li><a href="#">Home</a></li>  
  <li><a href="#">About</a></li>  
  <li><a href="#">Contact Us</a></li>  
</ul>
```



Creamos componentes en React con las clases de ES6 que extienden de la clase **Component** (más adelante veremos que no todos los componentes son clases).

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

El componente tiene un método **render** que es el que se encarga de renderizar en el navegador el HTML correspondiente al componente.

En nuestro método render usamos **JSX** para facilitar el desarrollo y creación de elementos HTML.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

¡¡IMPORTANTE!

El método render
retorna un solo
elemento.

Por lo tanto, si
tenemos más de un
elemento debemos
meterlos en un
contenedor padre.

Entonces... supongamos que hago un segundo componente:

```
//Segundo.js
```

```
class Segundo extends Component {  
  render() {  
    return (  
      <marquee>Yeah!</marquee>  
    );  
  }  
}  
export default Segundo;
```

Y lo queremos incorporar en **App.js**

App.js se debería ver así:

```
import Segundo from './Segundo.js';

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          
          <h2>Welcome to React</h2>
          <Segundo/>
        </div>
      </div>
    );
  }
}
```

Atributos



Atributos

class -> className

```
<div className="username"></div>
```

for -> htmlFor

```
<label htmlFor="username">Usuario</label>
```


Atributos

¡Podemos usar
objetos como
atributos!

```
const divStyle = {  
  color: 'white',  
  backgroundColor: 'black',  
}  
  
class MiComponente extends Componentes {  
  render(){  
    return <div style={divStyle}> Hello World! </div>  
  }  
}
```

Atributos

```
const divStyle = {  
  color: 'white',  
  backgroundColor: 'black',  
}  
  
class MiComponente extends Componentes {  
  
  render(){  
    return <div style={divStyle}> Hello World! </div>  
  }  
}
```

Las propiedades de CSS separadas por un **guión**, como background-color, necesitan escribirse con la técnica **camelCase** para que puedan renderizarse de la manera correcta

Atributos

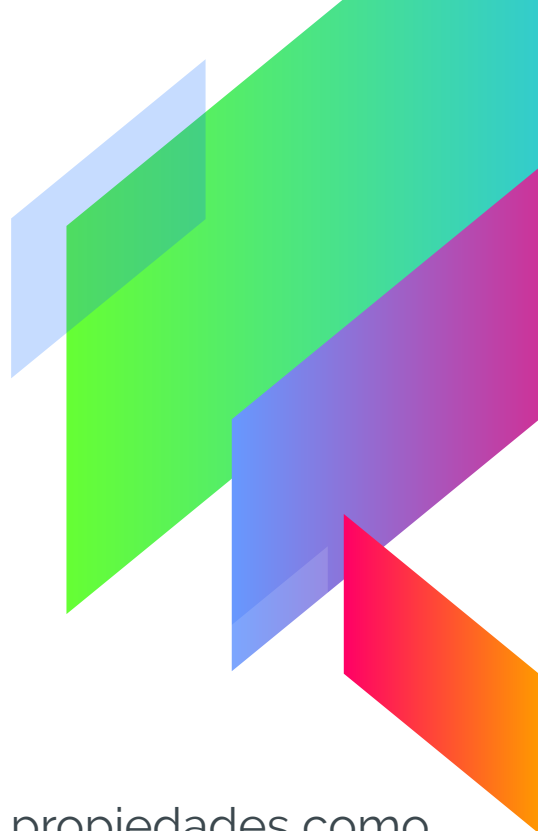
<https://reactjs.org/docs/dom-elements.html>

accept acceptCharset accessKey **action** allowFullScreen allowTransparency alt
async autoComplete autoFocus autoPlay capture cellPadding cellSpacing challenge
charSet checked cite classID **className** colSpan cols content contentEditable
contextMenu controls controlsList coords crossOrigin data dateTime default defer
dir disabled download draggable encType form formAction formEncType formMethod
formNoValidate formTarget frameBorder headers **height** hidden high **href** hrefLang
htmlFor httpEquiv icon **id** inputMode integrity is keyParams keyType kind label
lang list loop low manifest marginHeight marginWidth max maxLength media
mediaGroup method min minLength multiple muted name noValidate nonce open
optimum pattern placeholder poster preload profile radioGroup readOnly rel
required reversed role rowSpan rows sandbox scope scoped scrolling seamless
selected shape size sizes span spellCheck **src** srcDoc srcLang srcSet start step
style summary tabIndex target title **type** useMap value **width** wmode wrap

A cluster of overlapping, semi-transparent geometric shapes in the top-left corner, including a green parallelogram, a light blue parallelogram, a brown trapezoid, an orange parallelogram, a red parallelogram, and a purple parallelogram.

Props

Un componente en React puede recibir propiedades como parámetros para poder insertar valores y eventos en su HTML.

A cluster of overlapping, semi-transparent geometric shapes in the top-right corner, including a light blue parallelogram, a green parallelogram, a teal parallelogram, a purple parallelogram, and a red parallelogram.

Props (atributos)

```
<MiComponente titulo="Clase 3"/>
```

Props (atributos)

`<MiComponente titulo="Clase 2"/>`

```
class MiComponente extends Component{
  render(){
    return(
      <div>
        <h1> {this.props.titulo} </h1>
      </div>
    );
  }
}

export default MiComponente;
```

Props (atributos)

En el componente que importa a MiComponente:

```
<MiComponente titulo="Clase 2" texto="Elementos de un componente"/>
```

En el componente MiComponente.js:

```
class MiComponente extends Component{
  render(){
    return(
      <div>
        <h1> {this.props.titulo} </h1>
        <p> {this.props.texto} </p>
      </div>
    );
  }
}

export default MiComponente;
```


Pasando Data en Propiedades

```
const usuarios = ["Dario", "Javier", "Alejandro"];  
<MyList items={usuarios} />
```


Props

```
const usuarios = ["Dario", "Javier", "Alejandro"];  
<MyList items={usuarios} />
```

```
class MyList extends Component {  
  render(){  
    const { items } = this.props;  
    return (  
      <ul>{items.map(item => <li> {item} </li>)}</ul>  
    );  
  }  
}  
  
export default MyList;
```



⊗ ▶ Warning: Each child in an array or iterator should have a unique "key" prop.

```
class MyList extends Component {  
  render(){  
    const { items } = this.props;  
    return (  
      <ul>{items.map(item => <li> {item} </li>)}</ul>  
    );  
  }  
}  
  
export default MyList;
```

Key Props

Las key ayudan a React a identificar qué elementos han cambiado, agregado o eliminado.

Es decir, React por medio de las keys determina si es el mismo elemento o no.

- **Solo es necesario** agregar keys cuando devolvemos un array de elementos iguales.
- **La key debe ser única** entre elementos hermanos.
- Las keys no se muestran en el HTML final (si quisiéramos esto también deberíamos utilizar id)

Key Props

```
const usuarios = ["Dario","Javier","Alejandro"];  
<MyList items={usuarios} />
```

```
class MyList extends Component {  
  render(){  
    const { items } = this.props;  
    return (  
      <ul>{items.map(item => <li key={item}> {item} </li>)}</ul>  
    );  
  }  
}  
  
export default MyList;
```

The background features several overlapping, semi-transparent geometric shapes in various colors including light blue, green, teal, purple, magenta, red, orange, and yellow. These shapes are arranged in a way that creates a sense of depth and movement, with some shapes appearing to be layered on top of others. The overall composition is modern and vibrant.

The children prop



Se considera **“children”** a todo lo que se encuentre entre el tag de apertura y el tag de cierre de un componente

```
...  
<MiComponente>  
  <h1>Yo soy un hijo de MiComponente</h1>  
  <p>Y yo otro hijo</p>  
</MiComponente>  
...
```

En el ejemplo, “MiComponente” posee 2 hijos. El <h1> y el <p>

Un **componente de React puede tener varios hijos, un hijo, o ningún hijo**. Es decir, todo lo que se encuentre entre su tag de apertura y su tag de cierre, se considera hijo, y podremos acceder a ellos con **this.props.children**

```
...  
class MiComponente extends Component {  
  render() {  
    return (  
      <div className="component-wrapp">  
        {this.props.children}  
      </div>  
    )  
  }  
}  
...
```


Pasando **varios hijos** a “MiComponente”

```
...  
<MiComponente>  
  <h1>Yo soy un hijo de MiComponente</h1>  
  <p>Y yo otro hijo</p>  
  <OtroComponente algunaProp={10} />  
</MiComponente>  
...
```

Pasando **un sólo hijo** a “MiComponente”...

```
...  
<MiComponente>  
  <h1>Yo soy el único hijo de MiComponente</h1>  
</MiComponente>  
...
```

“MiComponente” **sin hijos**...

```
...  
<MiComponente />  
...
```

The slide features decorative geometric shapes in the corners. On the left, there are overlapping triangles in shades of green, blue, orange, and purple. On the right, there are larger, more complex geometric shapes in shades of green, blue, purple, and orange, some with a 3D effect.

PropTypes

Con React creamos componentes reusables.
Por lo tanto, es de buena práctica **definir qué props acepta el componente**.

Con PropTypes documentamos y validamos las propiedades.



Statefull Components

con estado - de clase



Stateful Components

Los componentes stateful o de clase permiten mantener **datos propios** a lo largo del tiempo e implementar distintos comportamientos durante su **ciclo de vida**.

Al conjunto de datos internos del componente se conocen como **estado «state»** y es una característica disponible **solo para los componentes definidos como clases**. Es similar a las props, pero es privado y está completamente controlado por el componente.

State

```
class Counter extends Component{  
  
  constructor(){  
    super();  
    this.state = {  
      count: 1  
    }  
  }  
  
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}  
  
export default Counter;
```

State

```
class Counter extends Component{  
  
  constructor(){  
    super();  
    this.state = {  
      count: 1  
    }  
  }  
  
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}  
  
export default Counter;
```

El método **constructor()** es necesario para poder definir la estructura del estado de un componente

State

```
class Counter extends Component{
```

```
  constructor(){  
    super();  
    this.state = {  
      count: 1  
    }  
  }  
}
```

```
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}
```

```
export default Counter;
```

La función **super()** en el constructor es necesario en React. Hereda de su clase padre

State

```
class Counter extends Component{
```

```
  constructor(){  
    super();  
    this.state = {  
      count: 1  
    }  
  }
```

```
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}
```

```
export default Counter;
```

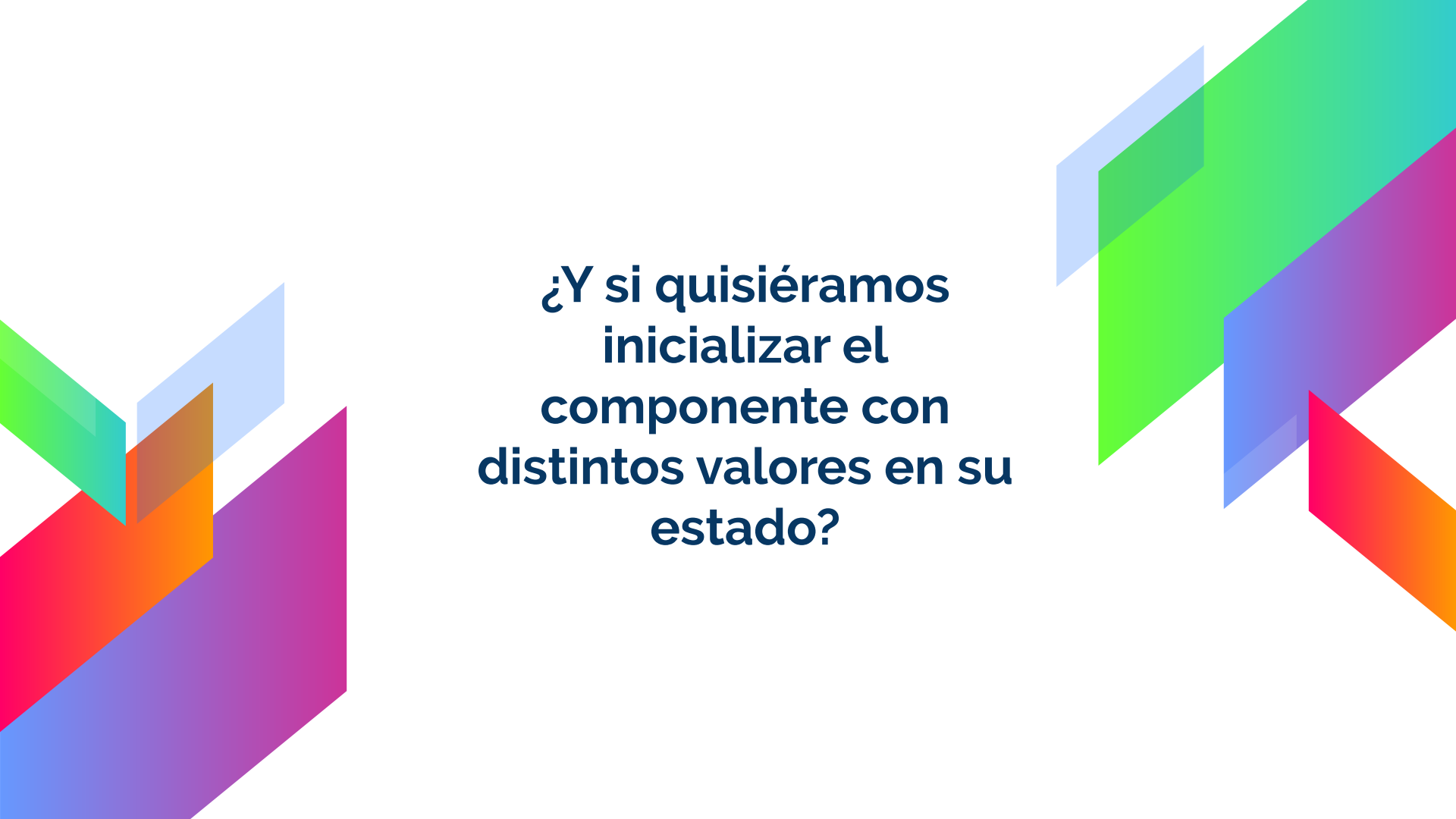
El **estado** de un componente será un **objeto literal de JS**

State


```
class Counter extends Component{  
  
  constructor(){  
    super();  
    this.state = {  
      count: 1  
    }  
  }  
  
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}  
  
export default Counter;
```

El **estado** de un componente será un **objeto literal de JS**

El cual luego puede ser utilizado en cualquier método de la clase tanto para lectura como para escritura.

The image features a white background with decorative geometric shapes in the corners. In the top-left and bottom-left corners, there are overlapping translucent shapes in shades of green, blue, orange, and purple. In the top-right and bottom-right corners, there are similar overlapping translucent shapes in shades of green, blue, purple, and orange. The text is centered in the middle of the image.

**¿Y si quisiéramos
inicializar el
componente con
distintos valores en su
estado?**

The slide features decorative geometric shapes on both the left and right sides. These shapes are composed of overlapping, semi-transparent polygons in various colors including green, blue, purple, orange, and red, creating a modern, abstract background. The central text is in a dark blue, sans-serif font.

**¿Y si quisiéramos
inicializar el
componente con
distintos valores en su
estado?**

iProps!

State

```
class Counter extends Component{  
  
  constructor(props){  
    super(props);  
    this.state = {  
      count: props.initialValue  
    }  
  }  
  
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}  
  
export default Counter;
```

¡Podemos recibir las props en el constructor para luego utilizarlas!

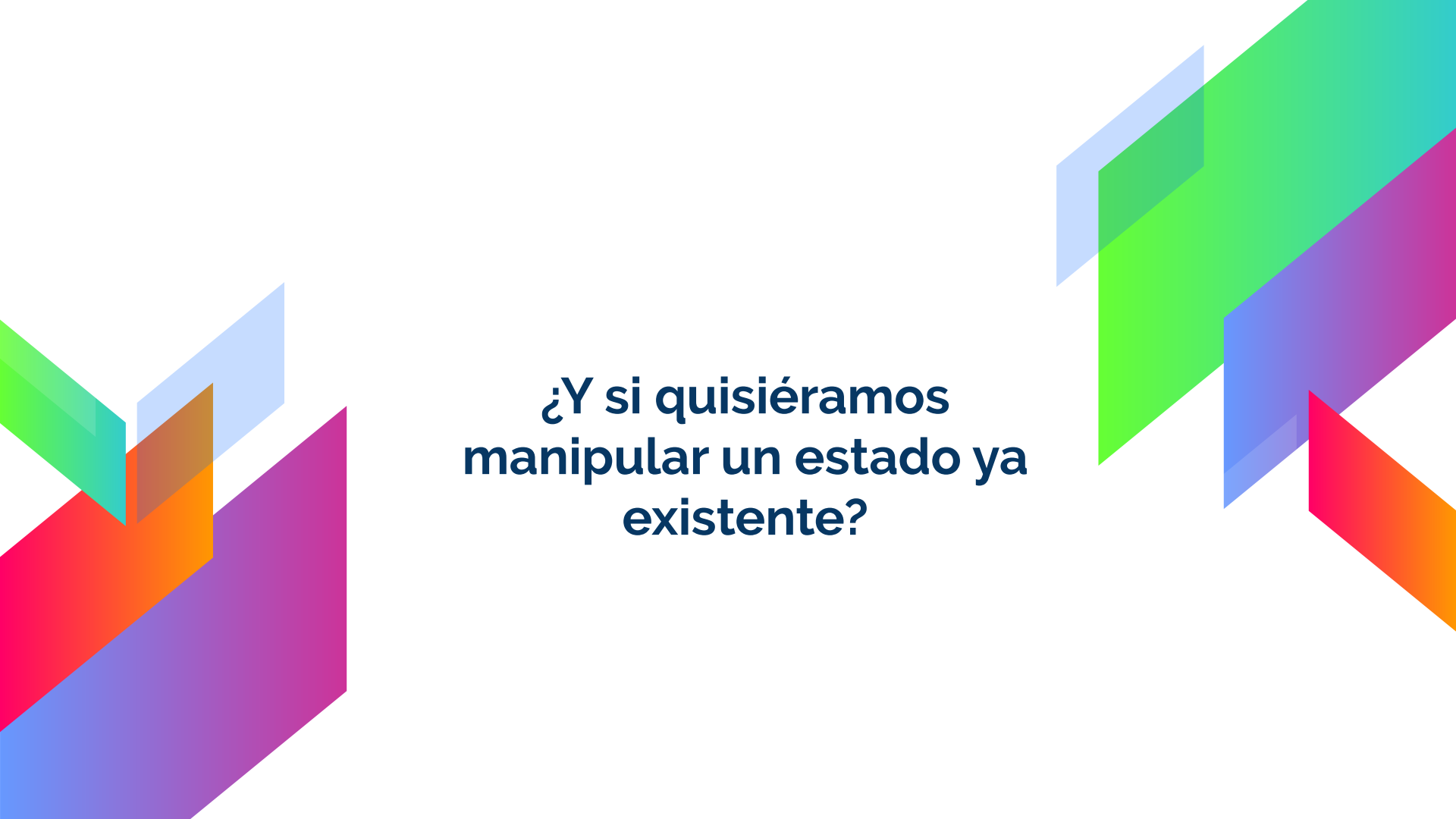
Es buena práctica utilizarlas al llamar `super()`

State

```
<Counter initialValue={10} />
```

```
class Counter extends Component{  
  constructor(props){  
    super(props);  
    this.state = {  
      count: props.initialValue  
    }  
  }  
  ...  
}  
  
export default Counter;
```

**¿Y si quisiéramos
manipular un estado ya
existente?**



setState

```
setState(nextState)
```

ó

```
setState(callback)
```

Es buena práctica cambiar el estado a través de **setState**

setState recibe un objeto literal con los atributos modificados del estado.

Ó **puede recibir un callback** que debe retornar un objeto literal

setState

```
class Counter extends Component{
  constructor(){
    super()
    this.state = {
      count: 1
    }
  }
  handleIncrement = () => {
    this.setState({count: this.state.count + 1});
  }

  render(){
    return(
      <div>
        <h1>{this.state.count}</h1>
        <button onClick={this.handleIncrement}>Incrementar</button>
      </div>
    );
  }
}

export default Counter;
```

onClick es un handle-event

Más adelante, explayaremos sobre **eventos**.

setState

```
class Counter extends Component{
  constructor(){
    super()
    this.state = {
      count: 1
    }
  }
  handleIncrement = () => {
    this.setState(state => ({count: state.count + 1}));
  }

  render(){
    return(
      <div>
        <h1>{this.state.count}</h1>
        <button onClick={this.handleIncrement}>Incrementar</button>
      <div>
    );
  }
}

export default Counter;
```

Handling Events



```
<button onclick="...">  
  Click me  
</button>
```



En React:

```
<button onClick={...}>  
  Click me  
</button>
```

```
class BotonSaludar extends Component{
  saludar = () => {
    alert('Hello world');
  }

  render(){
    return (
      <button onClick={this.saludar}> Haceme click </button>
    )
  }
}

export default BotonSaludar;
```

**Si retomamos uno de los
slides anteriores...**



```
class Counter extends Component{
  constructor(){
    super()
    this.state = {
      count: 1
    }
  }
  handleIncrement = () => {
    this.setState(state => ({count: state.count + 1}));
  }
  render(){
    return(
      <div>
        <h1>{this.state.count}</h1>
        <button onClick={this.handleIncrement}>Incrementar</button>
      </div>
    );
  }
}

export default Counter;
```

*...ahora tiene más sentido el
manejo de eventos*



setState - onChange de input

```
class Search extends Component{
  constructor() {
    super()
    this.state = {
      inputValue: '',
    }
  }

  handleOnChangeInput = event => {
    this.setState({
      inputValue: event.target.value
    })
  }

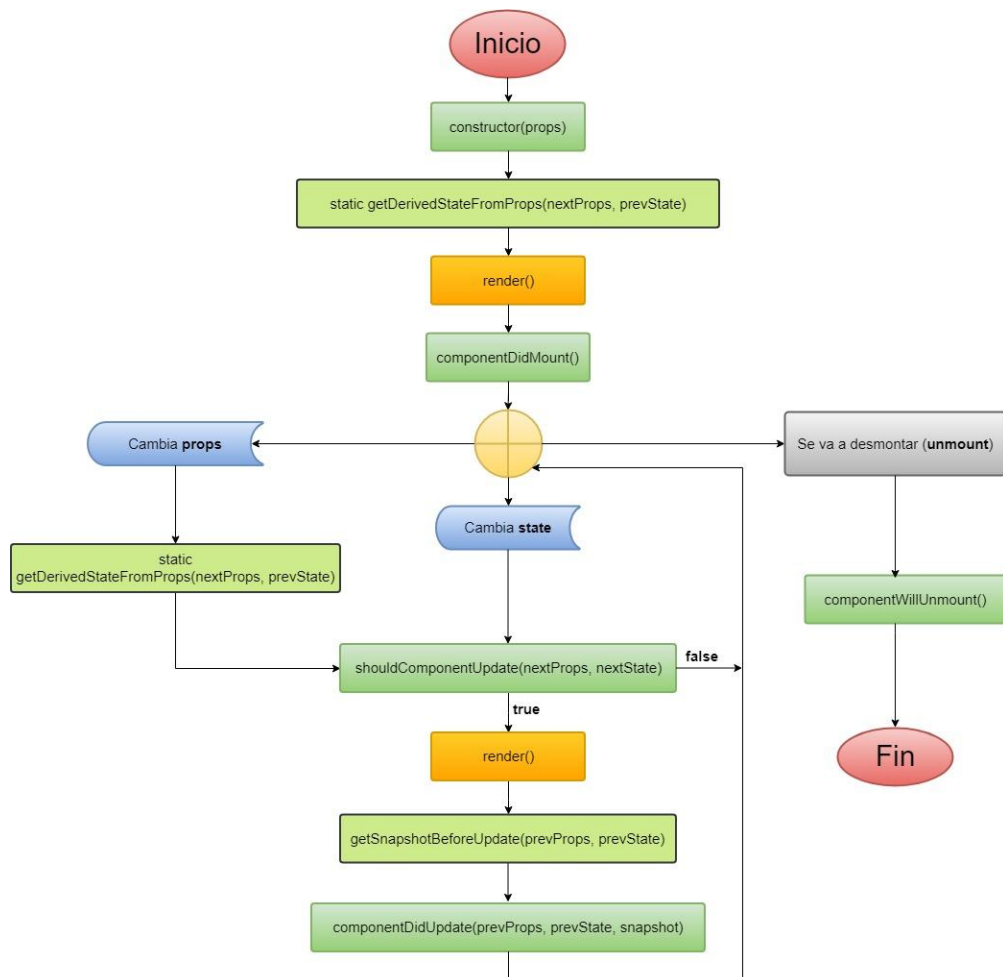
  render(){
    return (
      <input value={this.state.inputValue} onChange={this.handleOnChangeInput} />
    )
  }
}

export default Counter;
```

Ciclo de vida de un componente statefull



Ciclo de vida



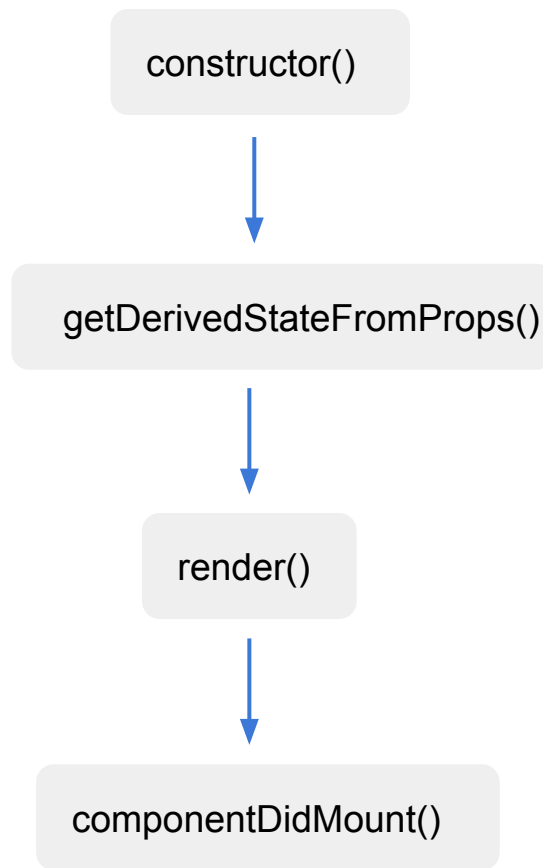
Fases del ciclo de vida

1. Montaje
2. Actualización
3. Desmontaje

Ciclo de vida

1. Montaje

2. Actualización
3. Desmontaje



Montado del componente

```
class Counter extends Component{  
  constructor(props){  
    super(props);  
    this.state = { count: 0 }  
  }  
  render(){  
    return(  
      <h1>{this.state.count}</h1>  
    );  
  }  
}  
  
export default Counter;
```

Pasamos las propiedades al constructor del componente de clase. En el constructor es donde inicializamos el estado del componente.

Montado del componente

```
static getDerivedStateFromProps(props, state){  
  //Code...  
}
```

El método `getDerivedStateFromProps()` se llama justo antes de ejecutar el método `render()`.

Se utiliza en casos muy extraños en los que el estado del componente depende de cambios en las propiedades. Por ejemplo en transiciones

Montado del componente

```
render(){  
  //Code...  
}
```

El método `render()` se ejecuta cada vez que cambia `this.props` o `this.state`. En la fase de montaje se ejecuta luego del `constructor()`

Montado del componente

```
componentDidMount(){  
  fetch(URL)  
    .then((r)=>console.log(r))  
    .catch((e)=>console.log(e));  
}
```

Se invoca esta función una vez ya ejecutado el método `render()`. Como el DOM ya es accesible podemos en este método realizar cualquier manipulación sobre él.

Se estila en este método hacer pedidos a endpoints via AJAX, inicializar timers o generar suscripciones a servicios.

Ciclo de vida

1. Montaje

2. Actualización

3. Desmontaje

`new props || setState() || forceUpdate()`



`getDerivedStateFromProps()`



`shouldComponentUpdate()`



`render()`



`getSnapshotBeforeUpdate()`



`componentDidUpdate()`

Actualización en el estado

```
shouldComponentUpdate(newProps, newState){  
  return newsProps.prop !== this.state.prop  
}
```

Este método se invoca antes de volver a renderizar cuando se reciben nuevos props o estados.

El método devuelve un valor booleano.

Por defecto retorna true. En caso de devolver false, no se llama a los métodos render(), componenteWillUpdate() y componentDidUpdate().

Actualización en el estado

```
componentDidUpdate(prevProps,  
prevState){  
  //code...  
}
```

Se llama justo después de render después que todos los cambios han sido hechos en el DOM. Puedes utilizar este componente para hacer alguna operación el DOM después que el componente se haya actualizado.

Ciclo de vida

1. Montaje
2. Actualización
- 3. Desmontaje**

```
componentWillUnmount()
```

Desmontado del componente

```
componentWillUnmount(){  
  //code...  
}
```

Es llamado justo antes de que el componente sea removido del DOM, es útil para hacer cualquier operación de limpieza, tales como invalidar timers, eliminar elementos que se hayan creados durante `componentDidMount` y cualquier otra operación pendiente.

Desmontado del componente

```
class Button extends Component {  
  componentWillUnmount() {  
    alert('El componente será desmontado')  
  }  
  
  render(){  
    return (  
      <button onClick={this.props.onClick}>Guardar cambios</button>  
    )  
  }  
}  
  
Button.propTypes = {  
  onClick: PropTypes.func.isRequired,  
}
```

Desmontado del componente

```
...
import Button from './Button'

class App extends Component {
  constructor() {
    super()
    this.state = {
      buttonVisible: true
    }
  }

  handleClickButton = () => {
    this.setState({
      buttonVisible: false
    })
  }

  render() {
    return (
      this.state.buttonVisible && <Button onClick={this.handleClickButton}/>
    )
  }
}
```

El desmontado se ejecuta cuando se quita un componente montado. Cuando cambiamos el estado "buttonVisible" a false, se desmonta el componente Button

Desmontado del componente

```
...
import Button from './Button'

class App extends Component {
  constructor() {
    super()
    this.state = {
      buttonVisible: true
    }
  }

  handleClickButton = () => {
    this.setState({
      buttonVisible: false
    })
  }

  render() {
    return (
      { this.state.buttonVisible
        ? <Button onClick={this.handleClickButton}/>
        : <p> Sin botón </p>
      }
    )
  }
}
```