

ReactJS

DEV.FX
DESARROLLAMOS(PERSONAS);

dev

¿Qué conocimientos previos debería tener?



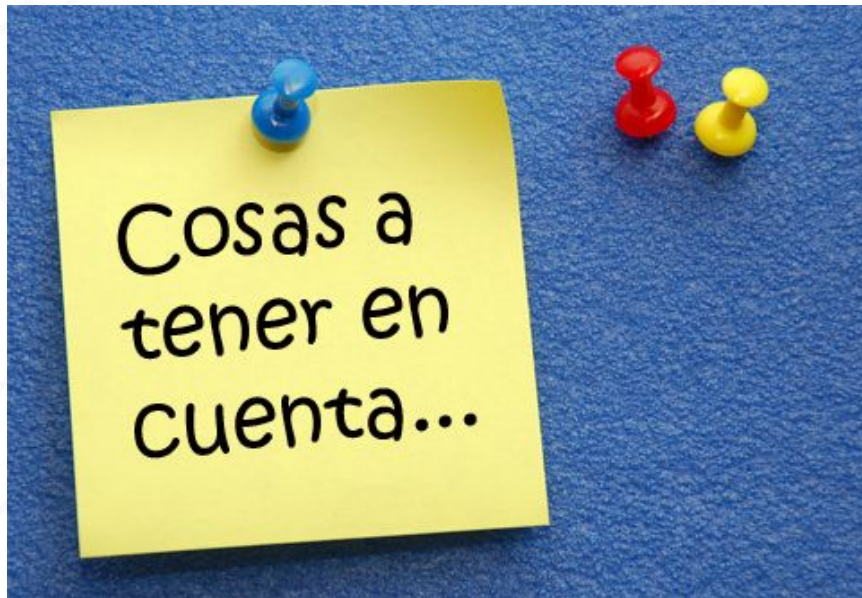
Conocimientos previos

- Fundamentos de **HTML**:
 - DOM.
 - Atributos y valores.
 - Tipos de etiquetas.
- **CSS**:
 - Modelo de caja.
 - Selectores y reglas.
 - Custom properties.
 - Cómo maquetar.
- **JavaScript**:
 - Funciones/métodos y parámetros.
 - Asincronía y llamado de APIs.

Conocimientos previos

- **EcmaScript:**
 - Ciclos (map, forEach).
 - Destructuring.
 - Condicionales (Operador ternario).
- **POO:**
 - 6 Principios de POO.
 - Prototype.
 - Sugar Sintaxis (class).
- **WebAPI:**
 - Manipulación del DOM.
 - Eventos.

Intro



- ¿Qué es React?
- ¿Por qué es necesario React?
- ¿Qué es un componente?
- Descomponentización.
- Ideología de react:
 - JSX.
 - Single Page Application (SPA).
 - Es declarativo.
- Funcionamiento:
 - CLI.
- Scaffold.

Intro

DEV.F
DESARROLLAMOS(PERSONAS);

dev

¿Qué es React?

Librería para construir interfaces web facilitando la creación y composición de componentes (principalmente declarativamente).

Una web y sus elementos son semejantes a las piezas que entran y salen en un escenario.

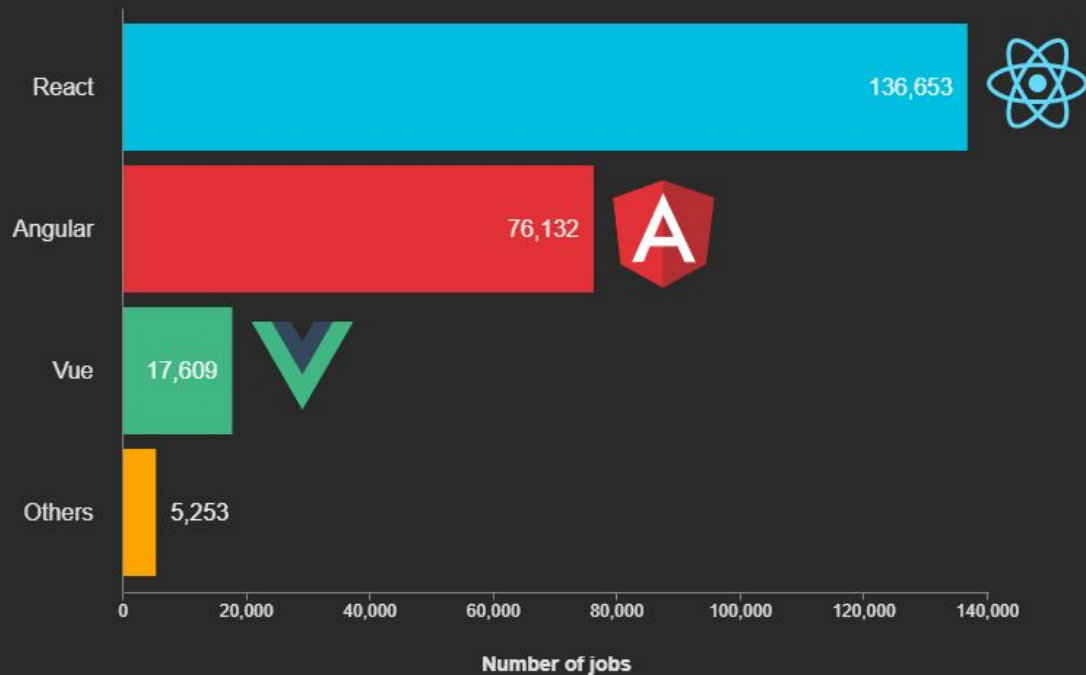


¿Por qué es necesario React?

- “Framework” front end más utilizado.
- Tecnología front end de las más demandadas y mejor pagadas.
- Desarrollar sitios web altamente escalables y de forma sencilla.
- Aprende una vez y úsalo en todos lados.
- Aprovecha al máximo el uso de JS.
- Comunidad de desarrollo.
- Recursos.

Number of jobs by frontend framework

From 01-Oct-2021 to 31-Mar-2022






Componentes y descomponentización


DEV.F
DESARROLLAMOS(PERSONAS);


dev


Piensa en componentes


  Mx


 Principal


 Shorts


 Suscripciones


 Originals


 YouTube Music


 Biblioteca


 Historial

 Tus videos


 Ver más tarde


 Descargas


 Recharge






 Mostrar más

Suscripciones


 Marca Claro (+)

 Dreamy Lofi (+)


 Learn English wit... (+)

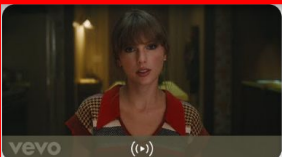
Todos Pokémon GO Mixes Videojuegos Ash Ketchum Liga de Campeones de la UEFA Música Película de superhéroes E >




¡EXPLOTA HALLOWEEN EN POKÉMON GO! NO TE...
Neludia ✓
14,545 vistas • hace 3 horas




Marvel Studios' Ant-Man and The Wasp: Quantumania [...]
Marvel Entertainment ✓
16 M de vistas • hace 1 día




Mix: Taylor Swift - Anti-Hero
Taylor Swift, Ariana Grande, Bruno Mars y muchos más



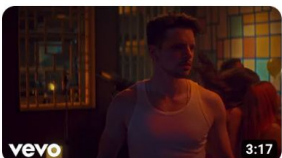
MI RUTA COMO: FRONTEND DEVELOPER
dayan developer
27,073 vistas • hace 3 meses




HOUSE OF THE DRAGON (Episodio 10)Final...
MR. MIKO
11 vistas • hace 4 horas



¿EL CAMBIO MÁS GRANDE EN REACT?
Vida MRR - Programacion w... ✓
1,892 vistas • hace 5 horas



Lasso - Ojos Marrones (Parte II)
LassoMusica ✓
1.3 M de vistas • hace 5 días



Los Caballeros del Zodiaco La Saga Hades : La Historia...
El FedeWolf ✓
5.7 M de vistas • hace 1 año

 Shorts



DEV.F

¿Qué es un componente?


- Elementos que se repiten.
- Elementos que cumplen una función específica.

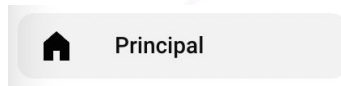
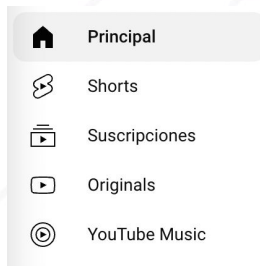
Ejemplos

- Un botón, una barra de menú, un card, un reproductor, un input, iconos, tablas, utilerías.

La **descomponentización** consiste en **dividir una pantalla en pequeños componentes que sean reutilizables y escalables**.



 Ant-Man and The Wasp: Quantumania | Tráiler Oficial...
Marvel Latinoamérica Oficial
1.2 M de vistas · hace 1 día



Ideología de React

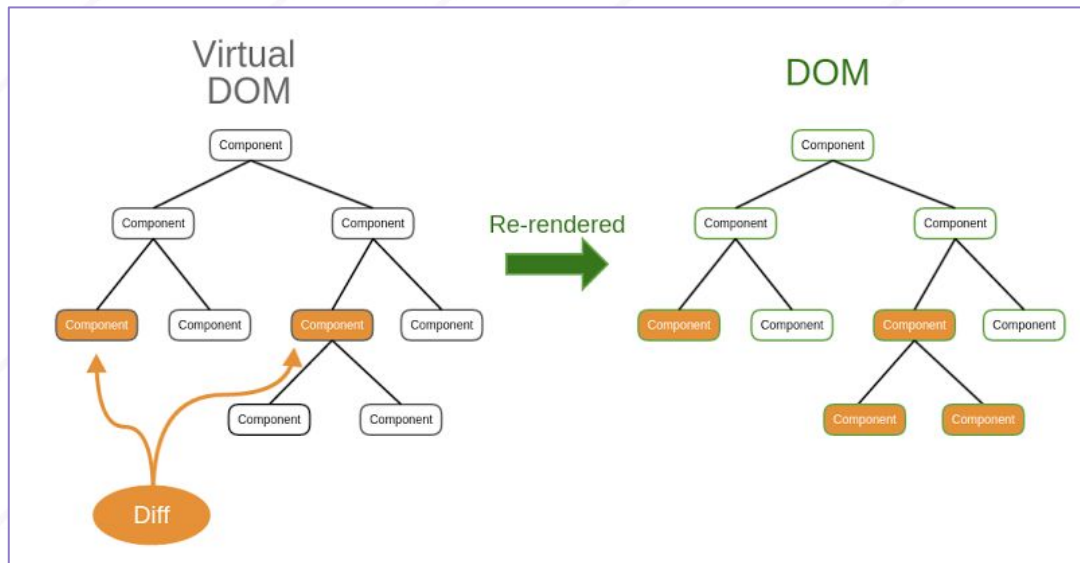
- JSX.
- Virtual DOM.
- Render y re render.
- Declarativo.
- Basado en una sola página.
- [JSConf](#).

```
import React from "react";

function CuadroHijo(props) {
  const character = {
    border: "15px solid salmon",
    background: "cyan",
    height: "100px",
    width: "100px",
    color: "black"
  };

  return(
    <div style={character}>
      {props.info}
    </div>
  );
}

export default CuadroHijo;
```



Funcionamiento de React

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Formas de usar React

- Agregando el CDN a nuestra página.
- Crear un scaffold desde 0 con webpack.
- Mediante una CLI (create-react-app o Vite).

CLI de React

Comando	Descripción
<code>npm i -g create-react-app</code>	<i>Instalar globalmente create-react-app.</i>
<code>npm create-react-app my-project</code>	<i>Crear un proyecto de react.</i>
<code>npx create-react-app my-project</code>	<i>Crear un proyecto de react sin instalar su CLI.</i>
<code>npm run start</code>	<i>Levantar el proyecto.</i>

Scaffold

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Scaffold

A decorative background pattern consisting of light purple lines and circles, resembling a circuit board or a network diagram. The lines are of varying thickness and form a complex, interconnected web across the slide. Some circles are solid, while others are hollow. The overall aesthetic is clean and modern, with a focus on geometric shapes and a monochromatic color palette of purples and greys.

Es la estructura de archivos y carpetas de un proyecto, también se le conoce como arquitectura del proyecto.

React

DEV.F
DESARROLLAMOS(PERSONAS);

dev

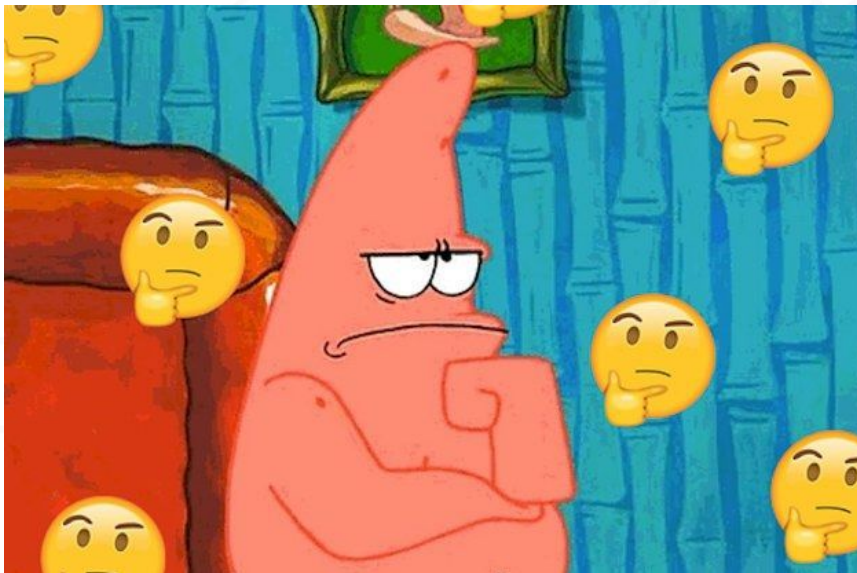
React

- **Conceptos clave.**
- **Historia de los componentes.**
- **Componentes funcionales.**
- Componentes de clase.
- Comparativa.
- Ciclo de vida.
- Variables y funciones.
- Estilos.
- **Router.**
- **Props y state.**
- **Eventos.**
- Manejo de formularios.
- **Arrays de componentes y condicionales.**
- **Hooks** (useState y useEffect).
- **Consumo de APIs.**



¿Y cómo diantres vamos a ver eso?

- Teoría.
- Ejercicio práctico que resuelve problemas del mundo real.
- Investigando y resolviendo errores.





Conceptos clave

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Conceptos clave

- Web component.
- Custom Tag `<Card/>`.
- **JSX.**
- **Virtual DOM.**
- Atributo vs propiedad `<p class="title-page">` vs `<Card className="title-page"/>`.
- Imports ES6 (export y export default)
- **React.fragment.**

Conceptos propios de React

Historia de los componentes

DEV.F
DESARROLLAMOS(PERSONAS);

dev

```
class Sensei extends React.Component {
  constructor(props){
    super(props)
    this.state = {
      nombre: "César Guerra"
      generacion: props.generacion
    }
  }

  componentDidMount(){
    this.startClass()
  }

  componentWillUnmount(){
    this.finishClass()
  }

  render(){
    return(
      <div>
        Master-Code G{this.state.generacion}, Sensei:
        {this.state.nombre}
      </div>
    )
  }
}
```

React y componentes de clases

Desde sus inicios en **2011** la forma de escribir React era usando **class components**.

Un **class component** es una clase de javascript que extiende la clase Component de React.

class Senseis extends React.Component

Los **class component** permiten guardar su estado y controlar lo que ocurre durante su ciclo de vida del componente (componentWillMount, render, componentDidMount, etc.).



Problemas de class component

Tras varios años de experiencia con esta aproximación, fueron surgiendo varios inconvenientes:

- 1. Las clases confunden a los devs JS y a las máquinas:** La orientación a objetos y el uso de **this** (especialmente con **bind** a eventos) puede resultar complejo para principiantes.
- 2. Era difícil reutilizar la lógica de los componentes:** Si esa era la intención de React, en la práctica era algo complejo.
- 3. Alta cohesión:** Componentes no relacionados requerían ser agrupados para controlar su ciclo de vida.

Nota: El ciclo de vida se explicará más adelante.



```
const Sensei = (props) => {  
  const [sensei, setSensei] = React.useState("César Guerra");  
  const [generation, setGeneration] = React.useState(9);  
  const [students, setStudents] = React.useState(props.numberOfStudents);  
  
  React.useEffect(() => {  
    startAssignment()  
    return () => {  
      finishAssignment()  
    }  
  });  
  
  return(  
    <div>  
      <h2>Master Code G{generation}: {sensei}</h2>  
    </div>  
  )  
}
```

React v16.8: The One With Hooks

<https://reactjs.org/blog/2019/02/06/react-v16.8.0.html>

React y Componentes funcionales

En febrero de 2018, se publica **React v16.8** que añade poder a la programación funcional por medio de los llamados **Hooks**.

Un componente funcional es una función que recibe el objeto Props y retorna un `ReactNode` (un `ReactNode` puede ser un elemento html, un string, un booleano, etc.).

```
const Sensei = (props) => { return(<ReactNode />) }
```

No hace uso explícito de **render**. Estas funciones solo reciben (**props**) y retornan, por eso tienen que utilizar **React Hooks**.


Nota: React Hooks se explicará más adelante.

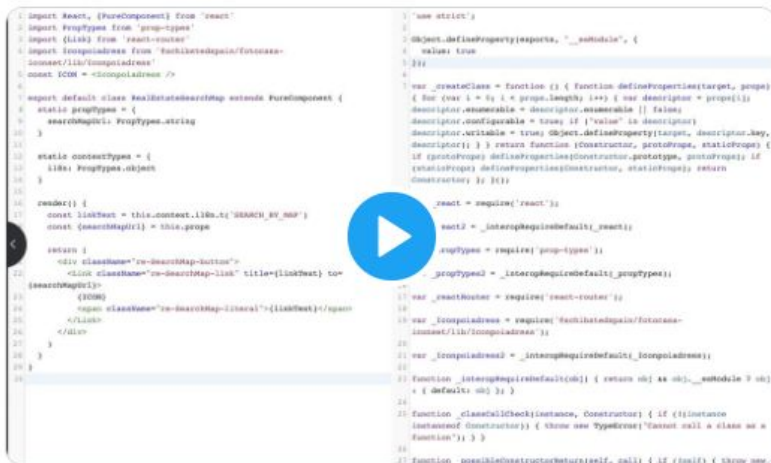


Miguel Ángel Durán

@midudev



Con la llegada de los Hooks a **#React**  vamos a empezar a ver más componentes en funciones. Pero, ¿sabes por qué deberías empezar ya a hacerlo cuando sea posible? 🤔 El output de Babel de una clase puede ser hasta un 40% más 🚀 y su ejecución es más lenta en el navegador 🐢.



2:03 a. m. · 22 nov. 2018



👍 72 🗨 3 🔄 Compartir este Tweet

Twittea tu respuesta

¿Por qué usar funciones por encima de clases?

- El frontend está experimentando una fuerte influencia de los **lenguajes de programación funcionales**.
- **Ayuda a unificar criterios**, donde todos los componentes tienen la misma estructura.
- **Nos ahorra entender el concepto de clases en Javascript**, aligerando la curva de aprendizaje.
- Hacer testing de un componente funcional suele ser más sencillo.
- Suelen requerir **menos líneas de código**, haciéndolo más fácil de entender.
- Un componente funcional es más ligero y rápido que su versión en clases.
- Un componente funcional solo tiene lo que debería tener y no más que eso.

¿Qué pasará con los Class Components?

- React ha sido, y seguirá siendo en los próximos años, famoso por una API estable.
- Las clases no van a desaparecer en el corto ni medio ni, seguramente, a largo plazo. Los componentes funcionales y uso de hooks, van a ser la forma “oficial” de crear componentes, pero se va a seguir manteniendo compatibilidad con las clases.

Componentes funcionales y de clase

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Componente de classe

```
class Click extends React.Component {  
  state = { clicks: 0 };  
  render() {  
    return (  
      <div>  
        <p>Clicks: {this.state.clicks}</p>  
        <button  
          onClick={() => this.setState(  
            ({clicks}) => ({clicks:clicks+1})  
          )  
        }  
        >Click</p>  
      </div>  
    );  
  }  
}
```


Componente funcional

```
function Click() {  
  const [clicks, setClicks] = React.useState(0);  
  
  return (  
    <div>  
      <p>Clicks: {clicks}</p>  
      <button  
        onClick={  
          () => setClicks(count + 1)  
        }  
      >Click me</button>  
    </div>  
  );  
}
```

Comparativa

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Diferencias

Functional Components

A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element (JSX).

There is no render method used in functional components.

Functional component run from top to bottom and once the function is returned it can't be kept alive.

Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI.

React lifecycle methods (for example, `componentDidMount`) cannot be used in functional components.

Hooks can be easily used in functional components to make them Stateful.

example: `const [name, setName] = React.useState('')`

Constructors are not used.

Class Components

A class component requires you to extend from `React.Component` and create a render function which returns a React element.

It must have the `render()` method returning JSX (which is syntactically similar to HTML)

Class component is instantiated and different life cycle method is kept alive and being run and invoked depending on phase of class component.

Also known as Stateful components because they implement logic and state.

React lifecycle methods can be used inside class components (for example, `componentDidMount`).

It requires different syntax inside a class component to implement hooks.

example:

```
constructor(props) {  
  super(props);  
  this.state = { name: '' }  
}
```

Constructors are used as it needs to store state.

Ciclo de vida

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Ciclo de vida

El ciclo de vida de un componente nos permite realizar distintas acciones dependiendo de en qué momento queremos que pasen cosas.

El **ciclo de vida** se puede dividir en **3 fases**:

- Montado.
- Actualización.
- Desmontado del componente.

A su vez, estas **fases se dividen en varios métodos** que puede tener el componente.

Métodos montado

Fase	Método
Montado	constructor(props)
	componentWillMount()
	render()
	componentDidMount()
Actualización	componentWillReceiveProps(nextProps)
	shouldComponentUpdate(nextProps, nextState)
	componentWillUpdate(nextProps, nextState)
	render()
	componentDidUpdate(lastProps, lastState)
Desmontado	componentWillUnmount()

Variables y funciones

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Variables y funciones

Se utilizan dentro de la función del componente pero fuera del render.

```
function CuadroHijo(props) {  
  const character = { ...  
}  
  
  return(  
    <div style={character}>  
      {props.info}  
    </div>  
  )  
}  
  
export default CuadroHijo;
```

```
function SearchFilters({searchedData, setSearchData}) {  
  
  const handleSubmitForm = (event) => {  
  }  
  
  return (  
    <  
      <form onSubmit={handleSubmitForm}>  
    </form>  
    </>  
  );  
}  
  
export default SearchFilters;
```


Estilos

DEV.F
DESARROLLAMOS(PERSONAS);

Estilos

- Estilos de línea (como objeto y con atributos en mayúsculas).
- Importación de hoja de estilos externa y uso de className.
- CSS Modular.
- CSS en el JS.
- Preprocesadores.
- FrameworksUI (MaterialUI, ReactBootstrap, Tailwind)

```
import React from 'react'
import './styles.css'

const Card = () => (
  <div className="card">
    <div className="card--header">
      <h1>Title</h1>
    </div>
    <div className="card--body">
      <p>
        Description
      </p>
    </div>
    <div className="card--footer">
      <p>All rights reserved</p>
    </div>
  </div>
)
```

Router

DEV.F
DESARROLLAMOS(PERSONAS);

dev

The background of the slide is a light gray abstract pattern resembling a circuit board. It consists of numerous thin, light gray lines that zigzag and intersect, forming a complex network. Small, light gray circles are placed at various points along these lines, suggesting nodes or components in a circuit. The pattern is more dense in the upper left and fades slightly towards the bottom right.

Router

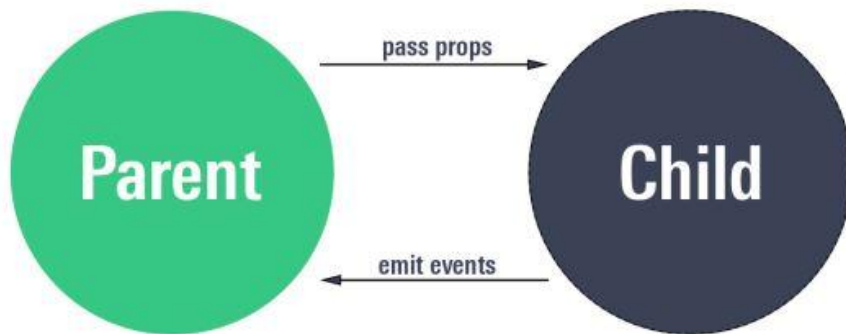
Props y state

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Props

- Son como los atributos de HTML.
- Las props son **entradas de datos** para los componentes funcionales. Y **son de solo lectura**.
- Se **utilizan** para **pasar información de padres a hijos PERO NO SE PUEDEN ACTUALIZAR**.
- Hay una **prop especial** llamada **children**.



```
App.js x HomeClass.js Home.js
04.function-components > src > App.js > ...
You, 3 minutes ago | 1 author (You)
1 import './App.css';
2 import Home from './components/Home'
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <Home saludo="Hola por props" />
9       </header>
10    </div>
11  );
12 }
13
14 export default App;
15
```

```
04.function-components > src > components > Home.js > [e] default
1 import React from 'react'
2
3 function Home(props){
4   return (
5     <React.Fragment>
6       <h1>Este es el Home en Función</h1>
7       <p>{props.saludo}</p>
8     </React.Fragment>
9   );
10 }
11
12 export default Home;
```

Este es el Home en Función

Hola por props

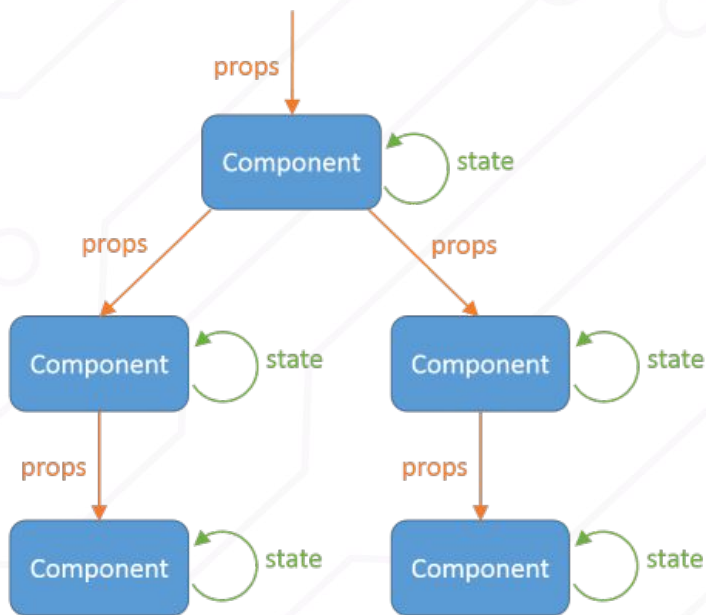
Props

Cuando escribíamos React con **class components**, los **props** se recibían por medio del **constructor** de la clase.

En un **function component** los **props** se reciben como parámetro de la función.

State

El estado es el **medio** que utiliza react para guardar los valores en tiempo de **ejecución**. A diferencia de las props, el estado **es actualizable**.



State



Data in the State control what you see in the View

```
const data = [  
  {  
    "name": "AFC Bournemouth",  
    "logo": "",  
    "manager": "Eddie Howe",  
    "stadium": "Dean Court",  
    "capacity": 11360  
  }  
]
```

EPL Teams

1. AFC Bournemouth
2. Arsenal
3. Brighton & Hove Albion
4. Burnley
5. Chelsea
6. Crystal Palace
7. Everton

Levantamiento de estado

El levantamiento de estado es una técnica de React que **pone el estado en una localización donde se pueda pasar como props a los componentes.**

Lo ideal es **poner el estado en el lugar más cercano a todos los componentes que quieren compartir esa información**, así todos nuestros componentes tendrán el mismo estado y cuando este cambie sólo re-renderizará lo necesario.

Eventos

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Listado de eventos

- El nombrado en camelCase
- Considere la WEB_API y su clases Event (e, e.target, etc.) y formData.
- Listado de eventos:
 - [Html.](#)
 - [JS.](#)
 - [React.](#)

Manejo de formularios

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Manejo de formularios



Arrays y condicionales

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Arrays y condiciones

- `&&` and `/ then`
- `&&` y `||`
- `key` y `map`

Hooks

DEV.F
DESARROLLAMOS(PERSONAS);

dev

¿Qué es un Hook?

Son **funcionalidades extra que podemos enganchar a nuestros componentes** funcionales. Anteriormente para usar esas funcionalidades forzosamente usábamos la sintaxis de clases.



Hooks útiles

Surge como una solución a la necesidad del manejo de estado de los componentes funcionales.

- *useState*.
- *useEffect*.
- *useContext*.



useEffect y useState

DEV.F
DESARROLLAMOS(PERSONAS);

dev

useEffect y useState

- **useState** ofrece una propiedad get y un setter para la actualización de cualquier variable que lo requiera.
- **useEffect** se ejecuta cada vez que se se actualiza el render.
 - Se puede condicionar.
 - `,[] =>` Solo se actualiza la primera vez.
 - `,[total] =>` se ejecuta cuando cambia la variable de estado total.

Consumo de API's

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Consumo de API's



Clasificación de componentes

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Clasificación de componentes

- De clase.
- Funcionales.
- Statefull.
- Stateless.
- Containers.
- Pages.
- Utils.