Historia de los componentes

DEV.F::
DESARROLLAMOS(PERSONAS);

Versiones significativas

DEV.F::
DESARROLLAMOS(PERSONAS);

Versiones significativas

- **React 14 (Octubre de 2015):** Esta versión introdujo los componentes de función sin estado lo que permitió definir componentes simples como funciones en lugar de clases, lo que mejoró la legibilidad del código y fomentó la composición de componentes.

- **React 16.3 (Marzo de 2018):** Introdujó los "Context API" mejorados, que facilitaron la propagación de datos a través de la jerarquía de componentes sin necesidad de pasar props manualmente a través de múltiples niveles. Además, se presentaron los "Componentes de React con Hooks", lo que permitió a los componentes funcionales tener estado y usar características previamente reservadas para los componentes de clase.

Versiones significativas

- **React 16.8 (Febrero de 2019):** Esta versión fue revolucionaria debido al uso de "Hooks", sin escribir clases. Esto simplificó en gran medida el código y fomentó la reutilización de la lógica del componente.
- **React 17 (Octubre de 2020):** Aunque no introdujo cambios drásticos en la API de React, esta versión se centró en mejoras internas, como la mejora de la detección de errores en la renderización y el soporte para múltiples versiones de React que se ejecutan en la misma página web.

React
con clases

DEV.F:
DESARROLLAMOS(PERSONAS);

```
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.).

DEV.F::

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***.

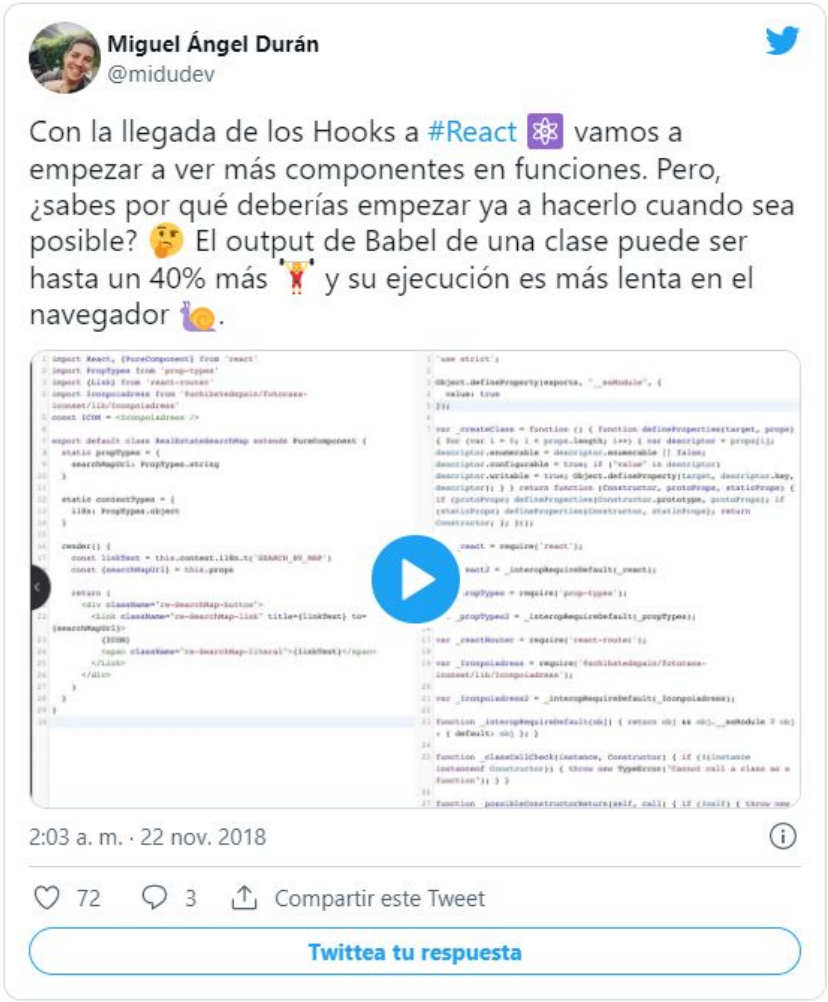
Un componente funcional es una función que recibe el objeto Props y retorna un ReactNode (un ReactNode puede ser un elemento html, un string, un booleano, etc.).

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.

DEV.F::



¿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.

DEV.F::

¿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);

Componente de clase

```
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);

Diferencias

Functional Components	Class Components
A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).	A class component requires you to extend from React. Component and create a render function which returns a React element.
There is no render method used in functional components.	It must have the render() method returning JSX (which is syntactically similar to HTML)
Functional component run from top to bottom and once the function is returned it cant be kept alive.	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 Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).

Hooks can be easily used in functional components to make them Stateful.
example: const [name,SetName]= React.useState(‘’)
	It requires different syntax inside a class component to implement hooks.
	example: constructor(props) {
	super(props);
	this.state = {name: ‘’}
	}

Constructors are not used.	Constructor are used as it needs to store state.

Ciclo de vida

DEV.F::
DESARROLLAMOS(PERSONAS);

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()