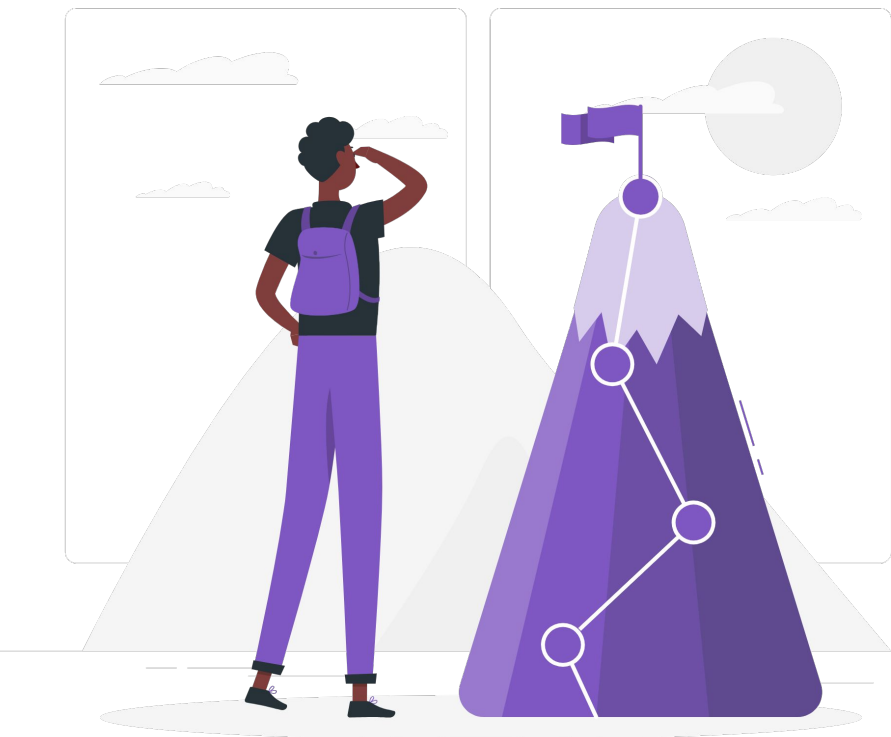


React Funcional

DEV.F
DESARROLLAMOS(PERSONAS);



Objetivos

- Conocer la evolución de la forma en que se escriben el código en React.
- Identificar las diferencias entre React basado en clases y el basado en funciones.
- Aprender acerca del concepto de programación funcional en React.
- Manejo de estados en React funcional.
- Conocer los hooks de React.

Un poco de historia y la relevancia de las funciones en React

```
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>
    )
  }
}
```

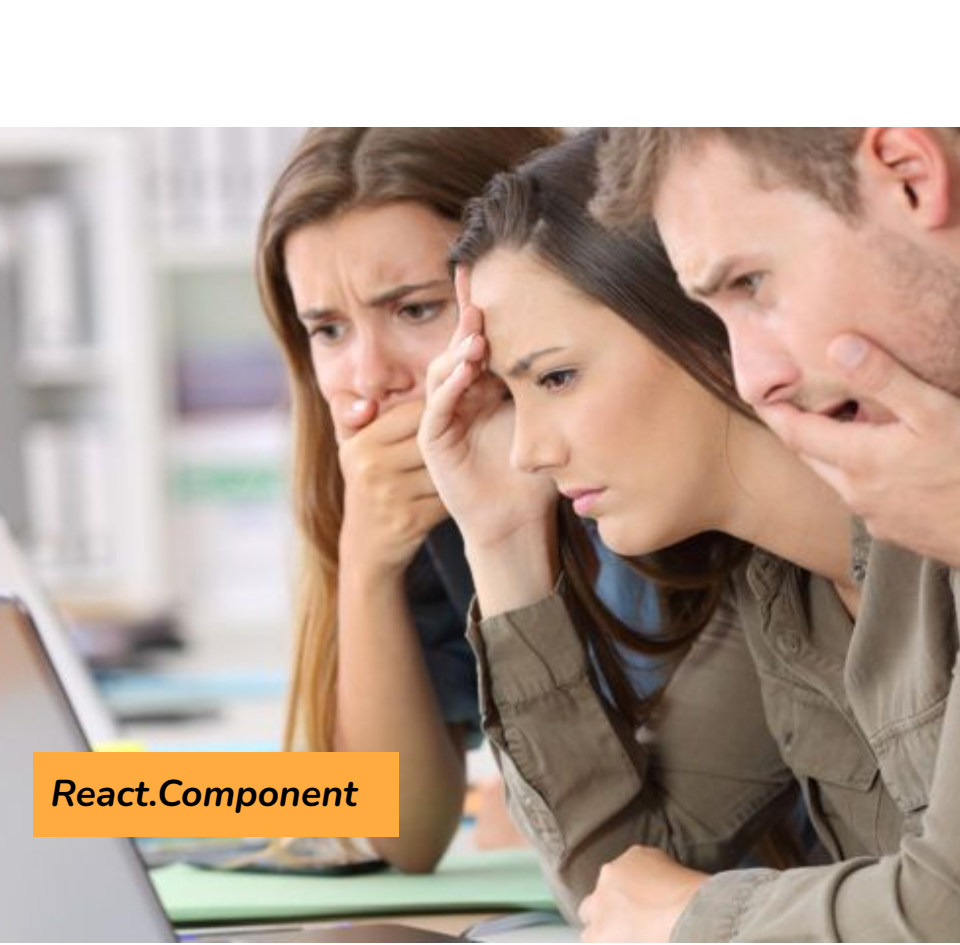
Un poco de historia: React y componentes de clases

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** nos permitieron guardar su estado y controlar lo que ocurre durante su ciclo de vida del componente (componentWillMount, render, componentDidMount, etc.).

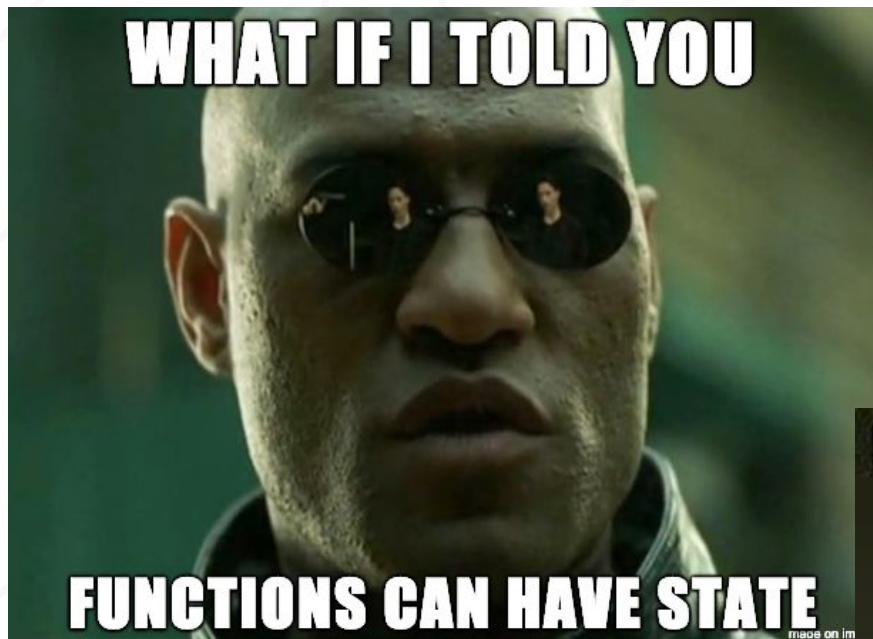


React.Component

Un poco de historia: Problemas de class component

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

- 1. Las clases confunden a las personas... 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. Los componentes complejos eran difíciles de entender:** Para respetar el ciclo de vida a veces teníamos que agrupar componentes no relacionados.



```
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>

Un poco de historia: React y Componentes funcionales

En febrero de 2019, 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 🐢.

```
1 import React, {PureComponent} from 'react'
2 import PropTypes from 'prop-types'
3 import {Link} from 'react-router'
4 import {fetchIndexPage/FetchIndexPage} from 'fetchIndexPage/FetchIndexPage'
5 import {fetchIndexPage/FetchIndexPage} from 'fetchIndexPage/FetchIndexPage'
6 const COM = 'ComponentAddress'
7
8 export default class RealStateDesktop extends PureComponent {
9   static propTypes = {
10     searchQuery: PropTypes.string
11   }
12   static defaultProps = {
13     link: PropTypes.object
14   }
15
16   render() {
17     const {link} = this.props
18     const {searchQuery} = this.props
19
20     return (
21       <div className="real-state-desktop">
22         <div className="real-state-desktop-link" title={link} />
23         {searchQuery}
24         <div className="real-state-desktop-link" title={link} />
25       </div>
26     )
27   }
28 }
29
30 // Babel output
31 'use strict'
32
33 Object.defineProperty(exports, '__esModule', {
34   value: true
35 })
36
37 var _createClass = function () { function defineProperties(target, props) {
38   for (var i = 0; i < props.length; i++) { var descriptor = props[i];
39     descriptor.enumerable = descriptor.enumerable || false;
40     descriptor.configurable = true; if ("value" in descriptor)
41       descriptor.writable = true; Object.defineProperty(target, descriptor.key,
42       descriptor); } } return function (constructor, protoProps, staticProps) {
43   if (protoProps) defineProperties(constructor.prototype, protoProps); if
44   (staticProps) defineProperties(constructor, staticProps); return
45   constructor; } }();
46
47 _createClass(RealStateDesktop, [{
48   key: 'render',
49   value: function render() {
50     const {link} = this.props;
51     const {searchQuery} = this.props;
52
53     return (
54       <div className="real-state-desktop">
55         <div className="real-state-desktop-link" title={link} />
56         {searchQuery}
57         <div className="real-state-desktop-link" title={link} />
58       </div>
59     );
60   }
61 }], [{
62   key: 'propTypes',
63   value: {
64     searchQuery: PropTypes.string
65   }
66 }], [{
67   key: 'defaultProps',
68   value: {
69     link: {}
70   }
71 }]);
72
73 exports.default = RealStateDesktop;
74
75 // Babel output
76 'use strict'
77
78 Object.defineProperty(exports, '__esModule', {
79   value: true
80 })
81
82 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
83 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
84 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
85 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
86 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
87 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
88 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
89 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
90 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
91 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
92 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
93 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
94 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
95 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
96 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
97 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
98 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
99 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
100 var _react__reactRouterLink = _interopRequireDefault(require('react-router-link'));
```

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.

¿Qué pasará con los Class Components?

React ha sido, y seguirá siendo en los próximos años, famoso por una API estable.

Por lo que **las clases no van a desaparecer** en el corto ni medio ni, seguramente, largo plazo.

Lo que va a ocurrir es que **los componentes funcionales, junto con los hooks, van a ser la forma “oficial” de crear componentes**, pero se va a seguir manteniendo compatibilidad con las clases por un largo tiempo.



Hooks

DEV.F
DESARROLLAMOS(PERSONAS);



Hooks

Los **Hooks** (ganchos) son una nueva **API** de la librería de **React** que nos permite tener estado, y otras características de **React**, en los componentes creados con una **function**.

Esto, antes, no era posible y nos obligaba a crear un componente con **class** para poder acceder a todas las posibilidades de **React**.

```
import React, { useState } from 'react';

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

State con useState

A diferencia de las **class component**, que incluían el manejo de estado al extender de **React.Component**, el manejo de estados en **functional components** se realiza por medio del **hook useState**.

Cada vez que se ejecuta el método **set** de un **useState** se reflejan los cambios en la interfaz de usuario.

Para poder usarlo es importante importarlo:

import React, { useState } from 'react';

Consejos Adicionales

DEV.F
DESARROLLAMOS(PERSONAS);

dev

```
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.

DESESTRUCTURANDO PROPS

Una práctica común consiste en desestructurar las props al recibirlas en la función, para evitarnos tener que escribir **`{props.nombre}`**, así solo escribiríamos **`{nombre}`** para poder usarla.


```
function myList() {  
  return (  
    <React.Fragment>  
      <ChildA />  
      <ChildB />  
      <ChildC />  
    </React.Fragment>  
  );  
}
```

Sintaxis completa

```
function myList() {  
  return (  
    ◇  
    <ChildA />  
    <ChildB />  
    <ChildC />  
    ◇  
  );  
}
```

Sintaxis corta

Fragmentos de React

Un patrón común en React es que un componente devuelva múltiples elementos. **Los Fragmentos te permiten agrupar una lista de hijos sin agregar nodos extra al DOM.**

`<React.Fragment>`

...

`</React.Fragment>`

Hay una sintaxis nueva, más corta que puedes usar para declarar fragmentos. Parecen etiquetas vacías:

`<>`

...

`</>`

Ciclo de Vida de Hooks

DEV.F
DESARROLLAMOS(PERSONAS);

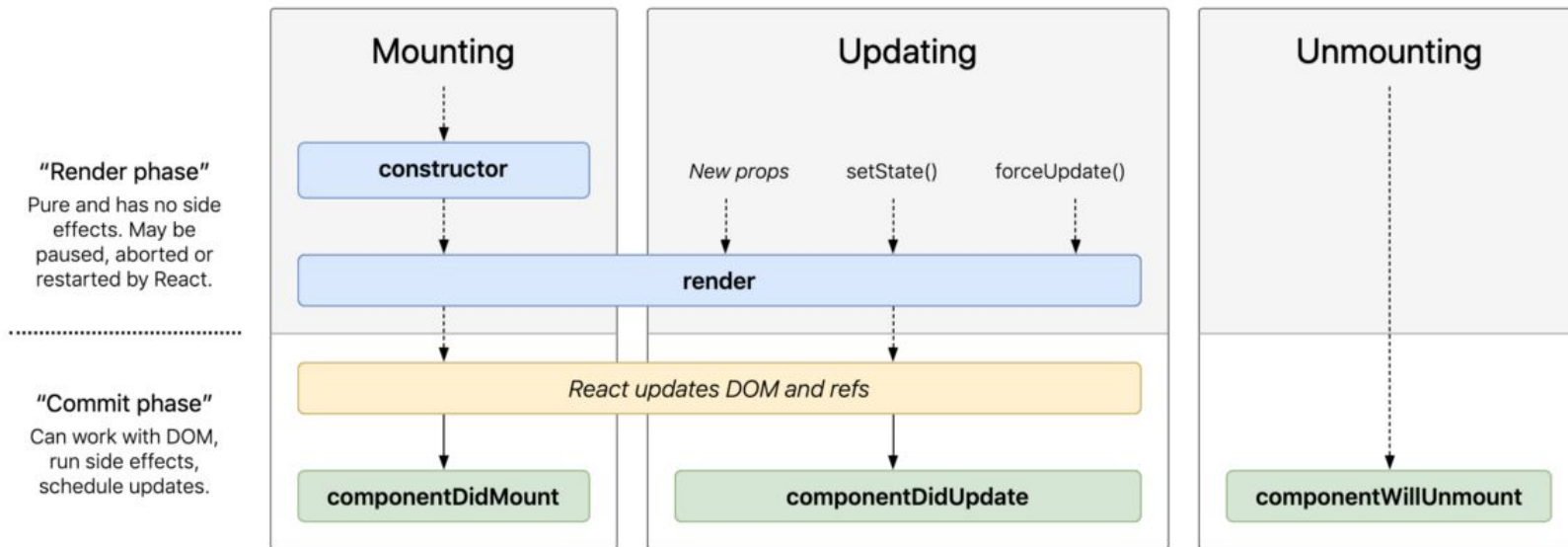
dev

Ciclo de Vida: Class Component

☐ Show less common lifecycles

React version ^16.4

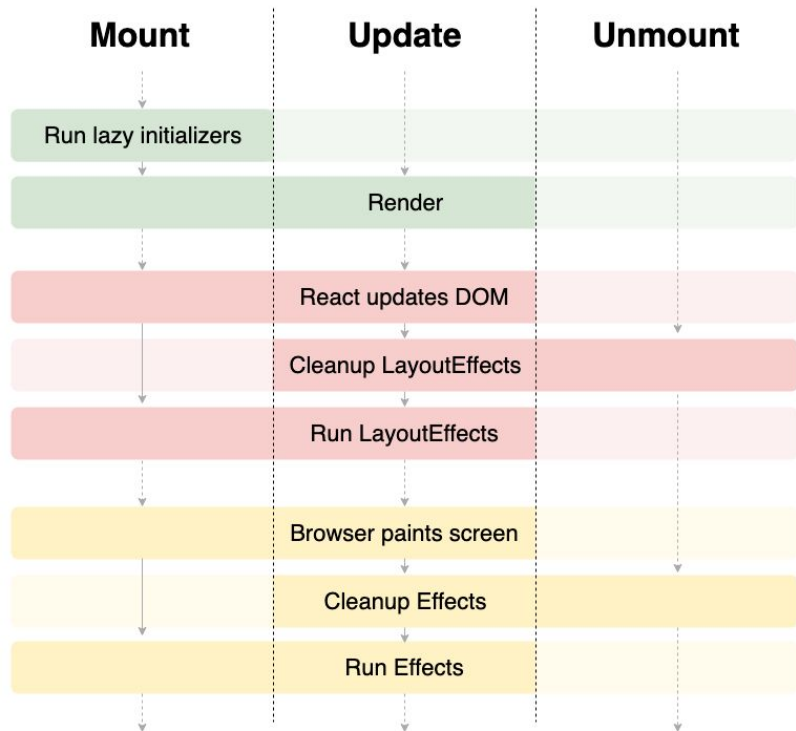
Language en-US



See project on [GitHub](#)

React Hook Flow Diagram

v1.3.1 github.com/donavon/hook-flow



Notes:

1. Updates are caused by a parent re-render, state change, or context change.
2. Lazy initializers are functions passed to `useState` and `useReducer`.

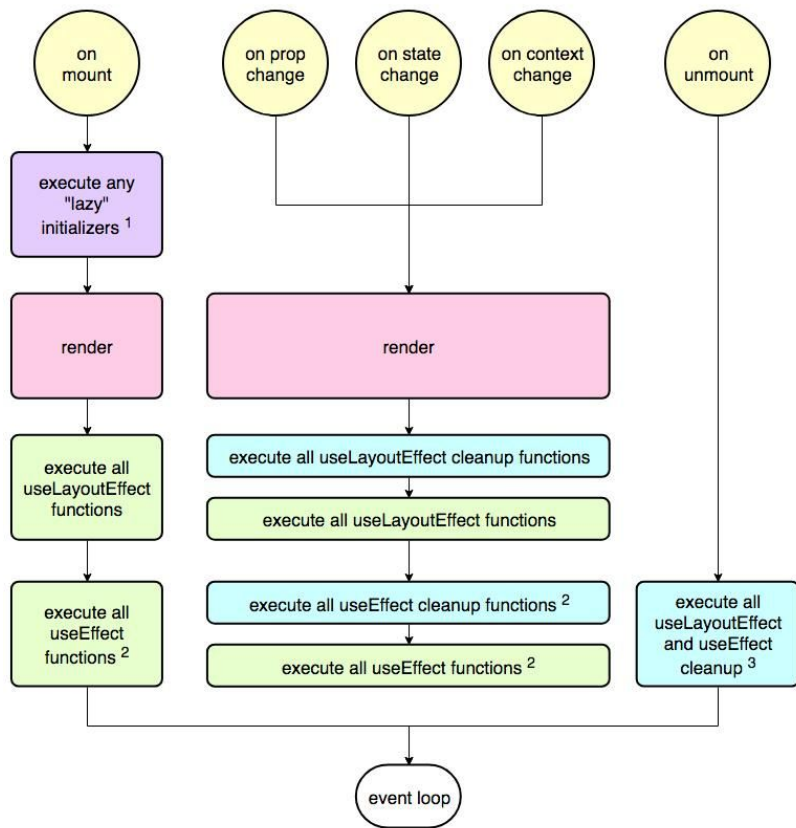
Flujo de Hooks en Functional Components

El ciclo de vida de los componentes en React permitía en nuestros *class components* ejecutar código en diferentes fases: montaje, actualización y desmontaje.

Con los *hooks* también podremos acceder a ese ciclo de vida en nuestros *functional components* usando *useEffect*.

Hook Flow Diagram

github.com/donavon/hook-flow



1. Lazy initializers are functions passed to `useState` and `useReducer`.

2. Execution is deferred until after the browser has painted.

3. Unmount cleanup happens in order of execution, without regard for type (i.e. `LayoutEffect/Effect`).

Flujo de Hooks en Functional Components



```
import React, { useEffect } from 'react';

function Ejemplo() {
  useEffect(function () {
    console.log('render!')
  }, [])

  return <span>Ejemplo de useEffect</span>
}
```

useEffect

Es un **hook** que **recibe como parámetro una función que se ejecutará cada vez que nuestro componente se renderice**, ya sea por un cambio de estado, por recibir props nuevas o, y esto es importante, porque es la primera vez que se monta.

Nota: Para usarlo es importante importarlo:
`import React, { useEffect } from 'react';`



```
import { useEffect } from 'react';

function Saludar({ nombre }) {

  const mensaje = `Hola, ${nombre}!`; // Calculates output
  useEffect(() => {
    document.title = `Saludos a ${nombre}`; // Side-effect!
  }, [nombre]);

  return <div>{mensaje}</div>; // Calculates output
}
```

useEffect

Proviene de side-effects

Un **functional component** de React utiliza **props** y/o **state** para calcular la salida. Si el **functional component** realiza cálculos que no tienen como objetivo el valor de salida, estos cálculos se denominan **side-effects**.

Ejemplos de **side-effects** son las llamadas a API, la manipulación directa del DOM, el uso de funciones de temporización como `setTimeout()`, etc.

Nota: La renderización del componente y la lógica de los side-effects son independientes. Es decir, no debemos realizarlos directamente en el cuerpo del componente, que se utiliza principalmente para calcular la salida.

Referencias

Miguel Durán (2019). *React Hooks, saca todo el potencial de React sin escribir clases.*

Recuperado de internet de:

<https://midu.dev/react-hooks-introduccion-saca-todo-el-potencial-sin-class/>

ReactJS (2019). *React v16.8: The One With Hooks.* Recuperado de internet de:

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

Adan Palacios (2019). *Tipos de componentes en React Js.* Recuperado de internet de:

<https://dev.to/ajpalacios/tipos-de-componentes-en-react-js-4epq>

Franco Cuarterolo (2021). *¿Componentes de clase o funcionales?* Recuperado de internet

de: <https://dev.to/cuarte4/componentes-de-clase-o-funcionales-4c1c>

React (sf). *Fragmentos de React.* Recuperado de internet de:

<https://es.reactjs.org/docs/fragments.html>

Referencias

Dmitri Pavlutin (2021). *A Simple Explanation of React.useEffect()*. Recuperado de internet de: <https://dmitripavlutin.com/react-useeffect-explanation/>

Miguel Durán (2019). *React Hooks, useEffect. Añadiendo funcionalidad en el ciclo de vida de nuestro componente - III.* Recuperado de internet de: <https://midu.dev/react-hooks-use-effect-funcionalidad-en-el-ciclo-vida-componentes/>