React4teachers

2022 edition

1. React4teachers: From Zero 2 Hero

2. Contenido del curso

- 1.- Introducció
- 2.- De VanillaJs a React
- 3.- React framework: Motivos de exito
- 4.- React framework: Class components vs Function components
- 5.- React framework: State management - Control de el estado de la aplication
- 6.- React framework: Routing
- 7.- Storybook
- 8.- Monorepo
- 9.- TailwindCSS y StyledComponents
- 10.-Despliegue de aplicaciones

3. Chapter 3

3.- React framework: Motivos de exito

4. Casos de exito

React fue creado por los desarrolladores de Facebook e Instagram para crear componentes interactivos dentro de una interfaz de usuario. La librería React se hizo de código abierto en 2013 y se ha hecho muy popular desde entonces. Mientras Facebook e Instagram siguen utilizando React, otros productos conocidos, como Netflix, también han empezado a implementarlo en su interfaz de usuario.

Otras aplicaciones conocidas:

- Yahoo Mail
- Vivaldi Web Browser
- Khan Academy
- AirBnB
- Discord
- WhatsApp (web)

5. Aspectos tecnicos

- Permite la generacion de componentes independientes
- Permite la comunicacion entre dichos componentes
- Se basa en estructuras nativas del lenguaje, sin curva de aprendizaje elevada (pasaras mas tiempo aprendiendo vanillaJS que React)
- Es sencillo y permite resultado rapido
- Permite su escalabilidad
- Es la base de otras plataformas de alto rendimiento como Next.js o Gatsby

6. Elementos basicos de ECMA 6

- Clases
- Arrow functions
- Variables
- Recorrido de colecciones

- Destructuring
- Spread Operator
- Modulos
- Operador ternario

Fuente[1]

7. Clases

Las primeras versiones de React se basan en clases. Las clases tienen un comportamiento complejo arquitectonicamente hablando y una serie de fases en su ciclo de vida. Sin embargo, se basan en ECMA6.

```
class Car {
  constructor(name) {
    this.brand = name;
  }
  present() {
    return 'I have a ' + this.brand;
  }
}
class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
  }
  show() {
      return this.present() + ', it is a ' + this.model
  }
}
const mycar = new Model("Ford", "Mustang");
mycar.show();
```

8. Arrow functions

Las arrow functions han tenido un gran auge con la inclusion de los denominados react Hooks. React Hooks son funcionalidades que puedes incorporar a tu aplicacion en calidad de HoC (high order component), o las antiguas primitivas.

[1] https://www.w3schools.com/react/react_es6.asp

Esto ha facilitado y simplificado la sintaxis, aunque para entornos complejos puede que sigas usando clases.

```
hello = () => {
  return "Hello World!";
}
```

9. Variables

Esta prohibido usar "var".

- Su scope es global y puede hacer que pierdas el control del flujo de la aplicacion.
- let y const tienen unas casuisticas muy determinadas que cumplen con todos los requerimientos que vayas a necesitar.
- Si necesitas pasar informacion de parent>child utiliza estados

10. Recorrido de colecciones

Vas a utilizar map y forEach, mucho, mucho, mucho

- En el caso de map, deberas devolver algo
- En el caso de forEach, no

Si utilizas un render iterando una coleccion, deberas indicar una **key**

```
const myArray = ['apple', 'banana', 'orange'];
const myList = myArray.map((item) => <p>{item}</p>)
```

11. Destructuring

La desestructuracion ha sido muy expandida desde que se empiezan a utilizar con react hooks. Sin embargo es una caracteristica nativa

```
const vehicles = ['mustang', 'f-150', 'expedition'];
// old way
const car = vehicles[0];
const truck = vehicles[1];
```

```
const suv = vehicles[2];
// ahora
const [car, truck, suv] = vehicles;
```

Tambien sirve para obtener elementos de segundo orden en un json

12. Spread Operator

La generacion de formas combinadas de colecciones o extender un elemento json anadiendo nuevos elementos es muy util y rapida, y aunque consume mas memoria, en utilizacion con let es muy rapida

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne,...numbersTwo];
//junto a desestructuracion
const numbers = [1, 2, 3, 4, 5, 6];
const [one, two, ...rest] = numbers;
```

13. Modulos

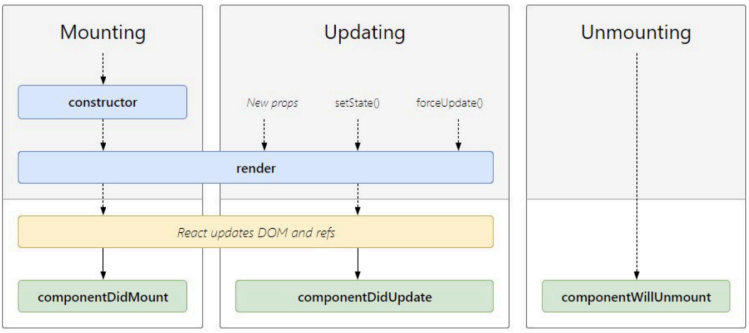
```
//WITHOUT DEFAULT
const name = "Jesse"
const age = 40
export { name, age }

import { name, age } from "./person.js";
//WITH DEFAULT
const message = () => {
  const name = "Jesse";
  const age = 40;
  return name + ' is ' + age + 'years old.';
};
export default message;

import message from "./message.js";
```

14. Operador ternario

Las bicondicionales permiten simplificar la sintaxis y en muchos casos lo utilizaremos

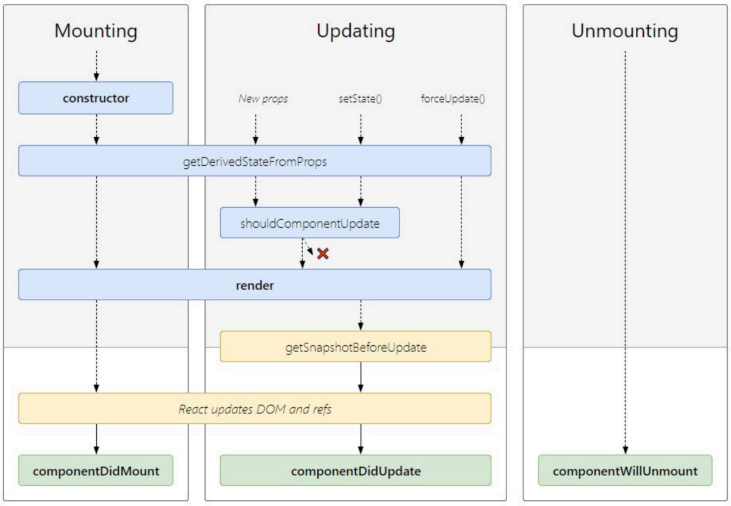
```
authenticated ? renderApp() : renderLogin();
```

15. React lifecycle



Common lifecycles

16. React lifecycle



All lifecycle

17. React mount

Mounting significa poner elementos en el DOM. React utiliza el DOM virtual para poner todos los elementos en la memoria. Tiene cuatro métodos incorporados para montar un componente:

- constructor()
- static getDerivedStateFromProps()
- render()
- componentDidMount()

```
import React, { Component } from 'react'

export default class App extends Component {
  constructor(props){
    super(props)
    this.state = {
      name: 'Constructor Method'
    }
  }
  render() {
    return (
      <div>
       <p> This is a {this.state.name}</p>
      </div>
    )
  }
}
```

18. React mount: getDerivedStateFromProps

Llamado justo antes de renderizar el elemento en nuestro DOM. Toma props y state como argumento y devuelve un objeto con los cambios en el estado.

```
mport React, { Component } from 'react'

export class ChildComponent extends Component {
    constructor(props){
        super(props)
        this.state = {
          name: 'Constructor Method'
        }
      }

    static getDerivedStateFromProps(props, state) {
        return {name: props.nameFromParent}
      }
```

```
    render() {
        return (
            <div>
                This is a {this.state.name}
            </div>
        )
    }
}
```

19. React mount: render

Este es el único método obligatorio requerido por React. Este método es el responsable de convertir nuestro JSX en DOM

```
import React, { Component } from 'react'

export default class renderMethod extends Component {
    render() {
        return (
            <>
                 <p>This is a render method</p>
            </>
        )
    }
}
```

20. React mount: componentDidMount

El método del ciclo de vida más común y ampliamente utilizado es componentDidMount. Este método es llamado después de que el componente sea renderizado. También puede utilizar este método para llamar a datos externos desde la API.

```
import React, { Component } from 'react'

export default class componentDidMountMethod extends Component {
  constructor(props){
    super(props)
    this.state = {
      name: 'This name will change in 5 sec'
    }
  }
```

```
  componentDidMount() {
    setTimeout(() => {
      this.setState({name: "This is a componentDidMount Method"})
    }, 5000)

  }
  render() {
    return (
      <div>
       <p>{this.state.name}</p>
      </div>
    )
  }
}
```

21. React mount: componentDidMount + fetch

```
import React, { Component } from 'react'

export default class componentDidMountMethod extends Component {
  constructor(props){
    super(props)
    this.state = {
      data: []
    }
  }

  componentDidMount() {
    fetch('https://jsonplaceholder.typicode.com/users').then(
        (response) => response.json()
    ).then(data => this.setState({data: data}))

  }

  render() {
    return (
      <div>
       <p>This will print all the name available in API users data</p>
    {this.state.data.map(d=> <h6 key={d.id}>{d.name}</h6>)}
      </div>
    )
  }
}
```

22. React: Updating

Esta es la segunda fase del ciclo de vida de React. Un componente se actualiza cuando hay un cambio de estado y props React tiene básicamente cinco métodos incorporados que se llaman mientras se actualizan los componentes.

- getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

Ya hemos mencionado render y getDerivedStateFromProps

23. shouldComponentUpdate()

Este método del ciclo de vida se utiliza cuando se desea que el estado o los props se actualicen o no. Este método devuelve un valor booleano que especifica si la renderización debe hacerse o no. El valor por defecto es true.

```
import React, { Component } from 'react'

export default class shouldComponentUpdateMethod extends Component {
  constructor(props){
    super(props)
    this.state = {
      name: 'shouldComponentUpdate Method'
    }
  }
  shouldComponentUpdate() {
    return false; //Change to true for state to update
  }

  componentDidMount(){
    setTimeout(() => {
      this.setState({name: "componentDidMount Method"})
    }, 5000)
  }
  render() {
    return (
```

```
      <div>
       <p>This is a {this.state.name}</p>
      </div>
    )
  }
}
```

nombre debería cambiar de "shouldComponentUpdate Method" a "componentDidMount Method" después de 5 segundos, pero no cambia porque shouldComponentUpdate es false, si lo cambias a true el estado se actualizará.

24. getSnapshotBeforeUpdate

Este método se llama justo antes de actualizar el DOM. Tiene acceso a las props y al estado antes de la actualización. Aquí puedes comprobar cuál era el valor de tus props o state antes de su actualización

Usar este metodo te fuerza a utilizar componentDidUpdate()

```
import React, { Component } from 'react'

export default class getSnapshotBeforeUpdateMethod extends Component {
    constructor(props){
        super(props)
        this.state = {
          name: 'constructor Method'
        }
      }

      componentDidMount(){
        setTimeout(() => {
          this.setState({name: "componentDidMount Method"})
        }, 5000)
      }
      getSnapshotBeforeUpdate(prevProps, prevState) {
        document.getElementById('previous-state').innerHTML = "The
 previous state was " + prevState.name
      }
      componentDidUpdate() {
        document.getElementById('current-state').innerHTML = "The
 current state is " + this.state.name
      }
    render() {
```

```
        return (
            <>
               <h5>This is a {this.state.name}</h5>
                <p id="current-state"></p>
                <p id="previous-state"></p>
            </>
        )
    }
}
```

25. componentDidUpdate()

El método componentDidUpdate es llamado después de que el componente sea actualizado en el DOM. Este es el mejor lugar en la actualización del DOM en respuesta al cambio de props y estado.

```
import React, { Component } from 'react'

export default class componentDidUpdateMethod extends Component {
    constructor(props){
        super(props)
        this.state = {
            name: 'from previous state'
        }
    }
    componentDidMount(){
        setTimeout(() => {
            this.setState({name: "to current state"})
          }, 5000)
    }
    componentDidUpdate(prevState){
        if(prevState.name !== this.state.name){
            document.getElementById('statechange').innerHTML = "Yes the
 state is changed"
        }
    }
    render() {
        return (
            <div>
                State was changed {this.state.name}
                <p id="statechange"></p>
            </div>
        )
    }
```

```
}
```

Nota: También puedes llamar a setState dentro de componentDidUpdate pero debes envolverlo en una condición como en el ejemplo anterior o causará un bucle infinito.

26. Unmounting

El último o el final del ciclo de vida de React. Se utiliza cuando un componente se elimina del DOM. React sólo tiene un método incorporado que se llama cuando un componente es desmontado:

- componentWillUnmount()

Si hay alguna acción de limpieza como la cancelación de llamadas a la API o la limpieza de caché, puedes realizarla en el método componentWillUnmount. No se puede utilizar setState dentro de este método, ya que el componente nunca se volverá a renderizar.

27. componentWillUnmount()

```
import React, { Component } from 'react'

export default class componentWillUnmount extends Component {
    constructor(props){
        super(props)
            this.state = {
                show: true,
            }
    }
    render() {
        return (
            <>
              <p>{this.state.show ? <Child/> : null}</p>
               <button onClick={() => {this.setState({show: !
this.state.show})}}>Click me to toggle</button>
            </>
        )
    }
}
```

```
export class Child extends Component{
    componentWillUnmount(){
        alert('This will unmount')
    }
    render(){
        return(
            <>
            I am a child component
            </>
        )
    }
}
```

28. Resumen del ciclo de vida

- El método constructor() es el mejor lugar para inicializar nuestro estado
- El método getDerivedStateFromProps() es un método del ciclo de vida poco utilizado y es el mejor lugar para establecer el objeto de estado basado en los props iniciales.
- El método shouldComponentUpdate() especifica si React debe continuar con el renderizado o no. El método render() es el método del ciclo de vida más utilizado y obligatorio.
- El método getSnapshotBeforeUpdate() tiene acceso a los props y al estado incluso después de la actualización.
- El método componentDidMount() es el método del ciclo de vida más común y utilizado y se llama después de que el componente se haya renderizado. También se puede utilizar este método para llamar a datos externos desde la API.
- El método componentDidUpdate() es llamado después de que el componente sea actualizado en el DOM y es el mejor lugar en la actualización del DOM en respuesta al cambio de props y estado.
- El método componentWillUnmount() ocurre justo antes de que el componente se desmonte y se destruya y se utiliza para acciones de limpieza como la cancelación de llamadas a la API.

29. Resumen

30. Ejercicio

Teniendo de base nuestro layout con un header, un panel de contenidos con Hero, vamos a realizar que cada boton tenga una accion

- El ejercicio 1, renderizara Hero
- El ejercicio 2, renderizara un componente que quieras
- Ejercicio 3, llamara a la API https://jsonplaceholder.typicode.com/users

Como resolverlo:

- Primero, renderiza los componentes nuevos, usando React class.
- Cambia App.js a React Class
- Cuando tengas todos los componentes, uno detras de otro, vas a usar un estado en el padre y un metodo en padre para actualizar el estado. Ademas, cada elemento hijo tendra un callback para cambiar el estado del padre.

31. Progresion del ejercicio

- Desevolucionar a React Class el fichero App.js

```
import {Hero} from "./components/Hero"
import Header from "./layout/Header"
```

```
function App() {
    return (
        <body className="text-gray-700 bg-white">
            <Header/>
            <Hero/>
        </body>
    );
}

export default App;

// a

import {Hero} from "./components/Hero"
import Header from "./layout/Header"
import React from 'react'

class App extends React.Component
{
    constructor(props) {
        super(props);
    }

    render() {
        return  <body className="text-gray-700 bg-white">
            <Header/>
            <Hero/>
        </body>
    }
}

export default App;
```

32. Progresion del ejercicio

Ahora generemos los componentes y los anadimos

```
import {Hero} from "./components/Hero"
import Header from "./layout/Header"
import React from 'react'
import Exercise2Image from "./components/Exercise2Image"
import Exercise3Data from "./components/Exercise3Data";
```

```
class App extends React.Component
{
    constructor(props) {
        super(props);
    }

    render() {
        return  <body className="text-gray-700 bg-white">
            <Header/>
            <Hero/>
            <Exercise2Image/>
            <Exercise3Data/>
        </body>
    }
}
export default App;
```

33. Progresion del ejercicio

- En App, pondremos un estado general, que sera la opcion Hero por defecto
- Pasaremos una funcion para cambiar el estado cada 5 segundos y pondremos que cambie a otro componente segun el estado. Para ello genera una funcion adicional y invocala con un timeout

34. Progresion del ejercicio

Este es un ejemplo de como cambiar el estado de la aplicacion y modificar el renderizaje.

```
import {Hero} from "./components/Hero"
import Header from "./layout/Header"
import React from 'react'
import Exercise2Image from "./components/Exercise2Image"
import Exercise3Data from "./components/Exercise3Data";

class App extends React.Component
{
    constructor(props) {
        super(props);
        this.state = {
            option: 1
        }
    }
```

```
    componentDidMount(){
        setTimeout(() => {
            this.setState({option: 2})
        }, 10000)
    }
    render() {
        let renderingPart ="";
        console.log(`Current state value is ${this.state.option}`)
        if (this.state.option===2){
            renderingPart = <Exercise2Image/>
        }else if(this.state.option===3){
            renderingPart = <Exercise3Data/>
        }else {
            renderingPart = <Hero/>
        }
        return  <body className="text-gray-700 bg-white">
            <Header/>
            {renderingPart}
        </body>
    }
}

export default App;
```

35. Ejercicio

Una vez modificado el layout, podemos generar una funcion que pasaremos como propiedad. Esta funcion cambiara el estado y actuara como callback en el momento del click

Usar las funciones como callback necesita hacer un BIND para no perder el estado de this. En React Class el binding se hace en el constructor.

```
import {Hero} from "./components/Hero"
import Header from "./layout/Header"
import React from 'react'
import Exercise2Image from "./components/Exercise2Image"
import Exercise3Data from "./components/Exercise3Data";

class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            option: 1
```

```
        }
        this.setActiveOption = this.setActiveOption.bind(this);
    }

    setActiveOption(val) {
        console.log(`Callback - changing state value to ${val}`)
        this.setState({option: val})
    }

    render() {
        let renderingPart = "";
        console.log(`Current state value is ${this.state.option}`)
        if (this.state.option === 2) {
            renderingPart = <Exercise2Image/>
        } else if (this.state.option === 3) {
            renderingPart = <Exercise3Data/>
        } else {
            renderingPart = <Hero/>
        }
        return <body className="text-gray-700 bg-white">
        <Header onClick={this.setActiveOption}/>
        {renderingPart}
        </body>
    }
}

export default App;
```

36. Ejercicio: resolucion

Para poder desarrollar el cambio de estado en el nodo hijo, pasamos al menu el metodo para cambiar la funcion activa. Esta funcion se pasa por propiedad del componente, en una propiedad que hemos llamado onClick.

> Es muy importante que observer como se define una funcion on click, ya que el evento se asigna por arrow function.

```
export default function Header({onClick}) {
    return (
        <nav>
            <div className="container mx-auto px-6 py-2 flex justify-
between items-center">
                <a className="font-bold text-2xl lg:text-4xl
 alternative-font" href="#">
```

```
                    React 4 teachers
                </a>
                <div className="block lg:hidden">
                    <button
                        className="flex items-center px-3 py-2 border
 rounded text-gray-500 border-gray-600 hover:text-gray-800 hover:border-
teal-500 appearance-none focus:outline-none">
                        <svg className="fill-current h-3 w-3" viewBox="0
 0 20 20" xmlns="http://www.w3.org/2000/svg">
                            <title>React 4 teachers</title>
                            <path d="M0 3h20v2H0V3zm0 6h20v2H0V9zm0
 6h20v2H0v-2z"/>
                        </svg>
                    </button>
                </div>
                <div className="hidden lg:block">
                    <ul className="inline-flex">
                        <li><a className="px-4 font-bold"
 href="/">Home</a></li>
                        <li><a className="px-4 hover:text-gray-800"
 href="#" onClick={()=>onClick(1)}>Ejercicio 1</a></li>
                        <li><a className="px-4 hover:text-gray-800"
 href="#" onClick={()=>onClick(2)}>Ejercicio 2</a></li>
                        <li><a className="px-4 hover:text-gray-800"
 href="#" onClick={()=>onClick(3)}>Ejercicio 3</a></li>
                    </ul>
                </div>
            </div>
        </nav>)
}
```

37. React intro

No te preocupes, es solo una introduccion! Pero te empezaran a salir muchas dudas, como son:

- Como controlo el estado de una aplicacion completa?
- Como debo ajustar las relaciones y propiedades padre-hijo?
- No es excesivamente complejo?
- Que diferencia hay entre React class y las funciones?


NOT SURE IF CLOCKS ARE SLOW
OR I'M JUST IMPATIENT