React4teachers

2022 edition

1. React4teachers

From Zero 2 Hero

Conselleria d'educacio de les Illes Balears

By Alberto Soto

UD1-Ecma6

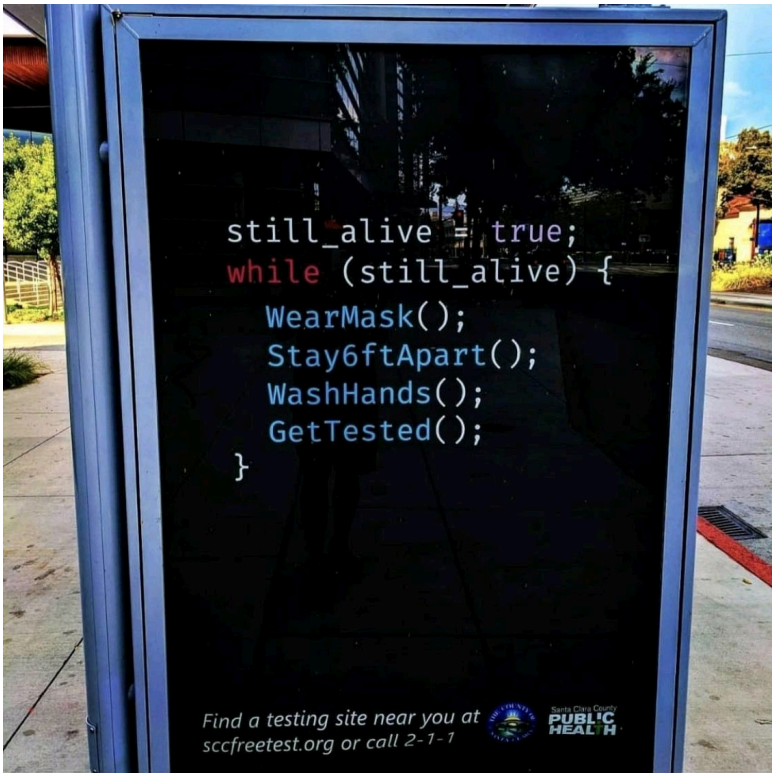
2022

2. Contenido del curso

- 1.- Introducción
- 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. Programming?



4. Introduccion a react

La mejor introduccion que podemos ofrecer para ReactJs es el propio javascript: Conocer nuestro punto de partida real es basico para saber que capacidades comporta.

Tradicionalmente al uso de javascript plano, sin intermediacion de librerias lo denominaremos VanillaJS.

React es una framework/libreria JS que extiende la funcionalidad de vanillaJS al infinito.

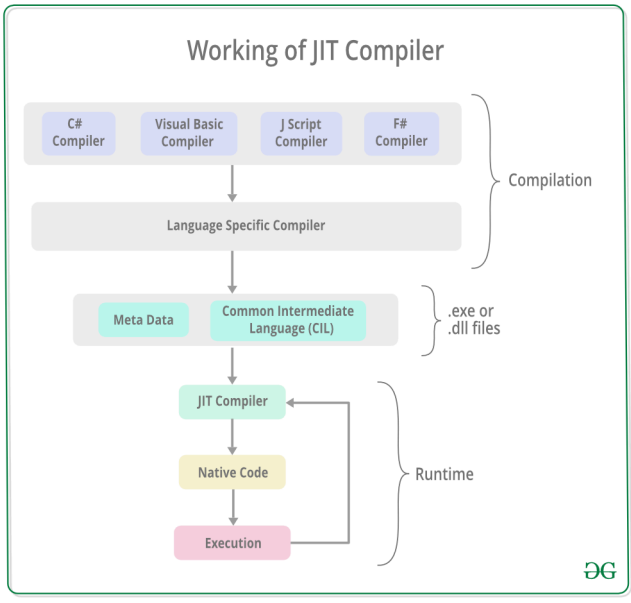
5. Lenguaje compilado

- PROS: Velocidad. IoT *. Especifidad
- CONS: Depende arq. Patrones obsoletos en control de datos.
- Ejemplos: C, C ++, Visual Basic, Fortran, Pascal

6. Lenguajes interpretados

- PROS: Facil instalacion, abstraccion.
- CONS: Lentitud, Consumo memoria
- Examples: Php, js, python

7. Semi-compiled languages



8. Pregunta:

Que tipo de lenguaje es Javascript? compilado, interpretado o semi compilado?

9. Pregunta II

Y RUST? Y WS?

Repasemos la web!

10. WS?

No, no son web services

WebAssembly es empaquetado en nativo Como la JVM, pero para cualquier lenguaje! JS tb puede ser WebAssembly

Ejemplo de js a WebAssembly[1]

11. La web tradicional

- Tradicionalmente el servidor ha generado la salida del html que el navegador procesa.
- Aproximaciones basicas y manuales a ECMA5 para JS garantizaban compatibilidad

12. Traditional JS Structure

Client side

```
<!DOCTYPE html>
<html>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
```

[1] https://engineering.q42.nl/webassembly/

```
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

13. Web 2.0

- El auge de la computacion + UX ha favorecido el desarrollo del entorno cliente
- Se genero el concepto de social web / web 2.0

14. React Class

```
import React from "react";
import { connect } from "react-redux";

const mapStateToProps = state => {
    return { articles: state.articles };
}
const ConnectedList = ({ articles }) => (
    <ul>
        { articles.map(article => (
            <li className="item" key={article.id}>{ article.title }</li>
        )) }
    </ul>
);
const List = connect(mapStateToProps)(ConnectedList);
class FirstComponent extends React.Component {
    render() {
        return (
            <div>
                <div className="Hello">Hello ankit this i</div>
                <List></List>
            </div>
        );
    }
}
export default FirstComponent;
```

15. React hooks

```
import React, { useState } from "react";
import ReactDOM from "react-dom";
import "./styles.css";
function App() {
  const [count, setCount] = useState(0);
  return (
    <div className="App">
      <h1>Hello CodeSandbox</h1>
      <h2>You clicked {count} times!</h2>

      <button onClick={() => setCount(count - 1)}>Decrement</button>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

source[2]

16. Web 3.0

- El antiguo concepto de web semantica se ha transformado a una web en auge a traves de smart-contracts y totalmente descentralizada.

Responsables:

- Ts, js, block-chain
- potencia de cliente
- Headless, Serverless, SSR, SPA, SSG

17. React + TS + solidity

```
import type { NextPage } from "next";
import Head from "next/head";
```

[2] https://codesandbox.io/s/7y6o4282lq?from-embed=&file=/src/index.js:0-521

```
import styles from "../styles/Home.module.css";
import {
  Navbar,
  Footer,
  Loader,
  Services,
  Transactions,
  Welcome,
} from "../components";

const Home: NextPage = () => {
  return (
    <div className="min-h-screen">
      <Head>
        <title>Web 3.0 Blockchain Solidity Example</title>
        <meta name="description" content="Web blockchain solidity
 example" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <main className="gradient-bg-welcome">
        <Navbar />
        <Welcome />
      </main>
      <Services />
      <Transactions />
      <Footer />
    </div>
  );
};

export default Home;
```

source[3]

18. Conceptos tecnologicos

- SSR: Server Side Rendering
- SSG: Static Site Generation
- SPA: Single Page Application

[3] https://github.com/dunapanta/Web-3.0-Blockchain-Solidity-App

19. SPA

Las SPA están estructuradas como una sola página HTML que no tiene contenido precargado. El contenido se carga a través de archivos Javascript para toda la aplicación y todo es una sola página HTML.

- PROs: UX, control
- Cons: SEO, deeplinking, payload

Actualmente: Backends/Social

React, AngularJS, Vue.js, Ember.JS y Svelte

20. SSG Static Site Generation

La pagina se crea en tiempo de compilacion de manera integra o progresiva.

- Metodologia: Gatsby, Nextjs
- Proveedores: CDN, Netlify, Vercel
- PRO: No hay nada mas rapido, SEO, escalable, economico.
- Cons: No encaja en todo modelo de negocio. Personalizacion, contenido dinamico

21. SSR

Actualmente se habla de SSR tradicional o SSR 2.0 Se puede hacer un renderizaje intermedio.

- Backend/microservicio
- Frontend Nextjs/SSR + Cache
- NextJS > SSG

Metodologia: Nextjs Proveedores: Cloud provider, Netlify, Vercel

22. Pregunto otra vez :)

Entonces… js es compilado o interpretado?

23. Java-script?

Developers survey[4]

24. La confusion: ECMA

Que es ECMA? Un subset de javascript.

ECMA 5 creado en 2009, acogido por completo en 2016.

Incluye:

- High-order iteration functions (map, reduce, filter, forEach);
- JSON support;
- Getters and setters;
- Better reflection and object properties;

25. La solucion: ECMA 6

ECMA 6 supuso un gran cambio en la percepcion de la web. Liberado en 2015. Incluye:

[4] https://www.jetbrains.com/lp/devecosystem-2020/javascript/

- Promises.
- Modules;
- Classes;
- Block-scoped variable declarations (let);
- Arrow functions;
- Template literal;
- Spread operator;
- De-structuring assignment;
- Parameter default values;
- Rest parameters;
- Symbols;
- Generator functions.

26. La reproduccion: ECMA 5.1

Es el estandard al que se han acogido los navegadores.

Jquery esta muerto! Ejemplo[5]

Motivos:

- Vanilla permite hacer TODO lo que permite jquery, pero no lo sabiamos.
- Nos ha condicionado la escritura del lenguaje
- Ha permitido el sobreuso de plugins y una web pesada por sobre abuso de ellos.

Soluciones de vanilla: template literals, fetch, querys

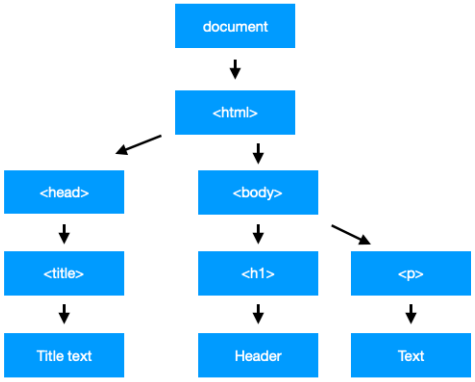
27. DOM, Shadow DOM, Virtual DOM

- Domain Object Model

[5] https://youmightnotneedjquery.com

La manipulación del DOM no es tan fácil y cómoda, y tiene muchos problemas de rendimiento. Hoy en día, hay dos conceptos esenciales de DOM: Shadow DOM y Virtual DOM

Live DOM[6]



28. DOM, Shadow DOM, Virtual DOM

- Shadow DOM

Es un arbol que reemplazara el DOM a posteriori, como si se tratara de un buffer. Es una copia en local de parte del arbol DOM original.

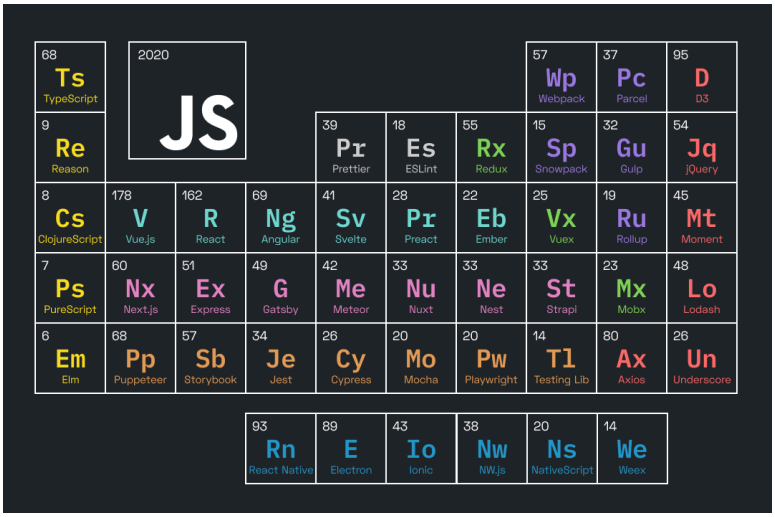
- Virtual DOM

Es una implementacion en memoria que integra React. Vue lo ha empeyado a usar tambien "recientemente". La copia es completa

Ambos solucionan problemas de rendimiento

[6] https://software.hixie.ch/utilities/js/live-dom-viewer/

29. El ecosistema actual



30. El ecosistema actual

- Package managers
- Bundlers
- Transpilers
- Precompilers

31. Gestores de paquetes

Dado que nodejs incluye npm, el gestor de paquetes mas habitual es npm.

Sin embargo, yarn, es una alternativa que permite:

- Mayor velocidad al tener una cache intermedia
- Permite el trabajo con monorepo (workspaces)

Por otro lado, tenemos los generadores de arquetipos, como Yeoman

32. Module Bundlers

- Webpack

- Vite
- Parcel
- Rollup

Los antiguos gestores de tarea como GULP o GRUNT han tenido que rendirse y desaparecer para permitir a evolucion de los bundlers, que son gestores de modulos. Los modulos se centran en tareas por funcionalidad.

Vite esta ganando adeptos, simplificando la configuracion de Rollup. Sin embargo, para grandes proyectos webpack es un estandard por su robustez.

33. Transpilers

El mismo codigo js debe ser convertido a una version de ECMA estandard que sea valido en todo navegador.

El transpiler por excelencia es Babel.

La intencion de Babel es modificar nuestro codigo para garantizar su compatibilidad > ECMA5

34. Practice time!

35. Vanilla JS topics

- The death of var
- Null is not null
- Falsy values

- The 3 ways of a function
- Everything is an object
- JS Classes from ECMA6
- Object destructuring
- Iterate me!
- Template literals
- From Jquery to vanillaJS

36. The death of var

```
let length = 16; // Number
const lastName = "Johnson"; // String
let cars = ["Saab", "Volvo", "BMW"]; // Array
let x = {firstName:"John", lastName:"Doe"}; // Object
console.log(typeof lastName);
```

37. Null is not null

```
typeof undefined // undefined
typeof null // object
null === undefined // false
null == undefined // true
```

38. Falsy values

- false
- undefined
- null
- 0
- NaN
- the empty string ("")

```
let b = new Boolean(false);
if (b)         // this condition evaluates to true
if (b == true) // this condition evaluates to false
```

39. The 3 ways of a function

```
function doLog(msg){
  console.log(msg);
}
let doAnotherLog = function(coolMsg){
  console.log(coolMsg);
}
const test = () => {
  console.log('hello world')
}
doLog('conventional function')
doAnotherLog('not so conventional function')
test()
```

40. Everything is an object

```
//Function type 2 (json object) (by expression)
let myFunctions = {
  /**
   * typical hello world
   * @param name parameter
   */
  sayHi: function (name) {
    console.log(`Aloha ${name}!`);
  }
};
myFunctions.sayHi("pepito");//<- execution
```

41. JS Classes from ECMA6

```
## ./modules/printer.js
export class Printer {
  constructor() {
    console.log('created a printer');
  }

  message(msg) {
    console.log(msg);
  }
}
```

```
## main.js
import { Printer } from './modules/printer';

const printerObj = new Printer();
printerObj.message('hello world');
```

42. Object destructuring

43. Object destructuring

```
const hero = {
  name: 'Batman',
  realName: 'Bruce Wayne'
};
const { name, realName } = hero;
name;     // => 'Batman',
realName; // => 'Bruce Wayne'
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// expected output: 10

console.log(b);
// expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// expected output: Array [30,40,50]
```

44. Template literals

```
let person = 'Mike';
let age = 28;

function myTag(strings, personExp, ageExp) {
  let str0 = strings[0]; // "That "
  let str1 = strings[1]; // " is a "
  let str2 = strings[2]; // "."

  let ageStr;
```

```
  if (ageExp > 99){
    ageStr = 'centenarian';
  } else {
    ageStr = 'youngster';
  }

  // We can even return a string built using a template literal
  return `${str0}${personExp}${str1}${ageStr}${str2}`;
}

let output = myTag`That ${ person } is a ${ age }.`;

console.log(output);
// That Mike is a youngster.
```

45. Iterate my soul

```
// Creating a map using Map object
let mp=new Map()

// Adding values to the map
mp.set("a",1);
mp.set("b",2);
mp.set("c",3);

// Logging map object to console
mp.forEach((values,keys)=>{
  document.write(values,keys+"<br>")
})
var el = document.getElementById('root');
var arr = [2, 5, 6, 3, 8, 9];
var newArr = arr.map(function(val, index){
    return {key:index, value:val*val};
})
console.log(newArr)
el.innerHTML = JSON.stringify(newArr);
```

More info[7]

Still more info[8]

[7] https://www.geeksforgeeks.org/most-useful-javascript-array-functions-part-2/
[8] https://masteringjs.io/tutorials/fundamentals/map-foreach

46. From Jquery to vanillaJS

Select elements:

```
// jQuery, select all instances of .box
$(".box");
// Instead, select the first instance of .box
document.querySelector(".box");
// …or select all instances of .box
document.querySelectorAll(".box");
```

Function on elements:

```
// with jQuery
// Hide all instances of .box
$(".box").hide();
// Without jQuery
// Iterate over the nodelist of elements to hide all instances of .box
document.querySelectorAll(".box").forEach(box => { box.style.display =
 "none" })
```

47. From Jquery to vanillaJS

Inner child

```
// With jQuery
// Select the first instance of .box within .container
var container = $(".container");
// Later...
container.find(".box");

// Without jQuery
// Select the first instance of .box within .container
var container = document.querySelector(".container");
// Later...
container.querySelector(".box");
```

Traversing

```
// with jQuery
// Return the next, previous, and parent element of .box
```

```
$(".box").next();
$(".box").prev();
$(".box").parent();

// Without jQuery
// Return the next, previous, and parent element of .box
var box = document.querySelector(".box");
box.nextElementSibling;
box.previousElementSibling;
box.parentElement;
```

48. From Jquery to vanillaJS

Events

```
// With jQuery
$(".button").click(function(e) { /* handle click event */ });
$(".button").mouseenter(function(e) {  /* handle click event */ });
$(document).keyup(function(e) {  /* handle key up event */  });

// Without jQuery
document.querySelector(".button").addEventListener("click", (e) => { /
* ... */ });
document.querySelector(".button").addEventListener("mouseenter", (e) =>
 { /* ... */ });
document.addEventListener("keyup", (e) => { /* ... */ });
```

Trigger events

```
// With jQuery
// Trigger myEvent on document and .box
$(document).trigger("myEvent");
$(".box").trigger("myEvent");

// Without jQuery
// Create and dispatch myEvent
document.dispatchEvent(new Event("myEvent"));
document.querySelector(".box").dispatchEvent(new Event("myEvent"));
```

49. From Jquery to vanillaJS

On ready

```
// With jQuery
$(document).ready(function() {
  /* Do things after DOM has fully loaded */
});

// Without jQuery
// Define a convenience method and use it
var ready = (callback) => {
  if (document.readyState != "loading") callback();
  else document.addEventListener("DOMContentLoaded", callback);
}

ready(() => {
  /* Do things after DOM has fully loaded */
});
```

Fetch

```
// With jQuery
$.ajax({
    url: "data.json"
  }).done(function(data) {
    // ...
  }).fail(function() {
    // Handle error
  });

// Without jQuery
fetch("data.json")
  .then(data => {
    // Handle data
  }).catch(error => {
    // Handle error
  });
```

Source[9]

50. Formato del curso

```
let isPresentacion = true;
```

[9] https://tobiasahlin.com/blog/move-from-jquery-to-vanilla-javascript/

```
const presentationContent=[{time:15,type:'presentacionPersonal'},
{time:15,type:'presentacionCurso'}]
const regularContent=[{time:30,type:'standup'}]
const inicio = isPresentacion?presentationContent:regularContent;
let doMasterClass = (doRecording) => {return 'knowledge'}
const principal = () => { doMasterClass(true); return
 {time:60,type:'learnNewTopics'}}
const prepareNextLesson = {time:30,type:'flushKnowledge'}
let result = [...inicio,principal(),prepareNextLesson]
const NUM_LESSONS = 5
const HOUR_VALUE = 60
let totalLessonTime = 0
result.forEach(aux=>{ totalLessonTime+= aux.time;})
console.log(`Total time per lesson is ${totalLessonTime} minutes`)
console.log(`Total in-person time is ${totalLessonTime*NUM_LESSONS}
 minutes (${(totalLessonTime*NUM_LESSONS)/HOUR_VALUE}h)`)
```

51. Ejercicio

Cuando el usuario haga en un boton, cargaras los usuarios de la siguiente direccion: http://dummy.restapiexample.com/api/v1/employees

Con los datos obtenidos, renderiza dinamicamente tu pagina para mostrarlos.

Puedes usar esta plantilla:

https://tailwindcomponents.com/component/testimonial-card Y una imagen de https://picsum.photos/