

***RECUERDA PONER A GRABAR LA
CLASE***





¿DUDAS DEL ON-BOARDING?

MIRALO AQUI



Clase 03. REACT JS

JSX Y WEBPACK



OBJETIVOS DE LA CLASE

- Entender las aristas del sugar syntax como proceso evolutivo de los lenguajes.
- Expandir nuestra sintaxis avanzada de JavaScript.
- Conocer el rol de webpack y babel en el bundling/retrocompatibilidad.
- Desarrollar código en JSX.

GLOSARIO:

Clase 2

Virtual DOM: Es un patrón de comportamiento y React lo implementa con una tecnología llamada “Fiber”. En sí, resulta ser todo lo que React sabe de tu aplicación y cada nodo o fibra.

Node.js: es un entorno de ejecución de javascript que le permite al código en js ser ejecutado en nuestra computadora.

NPM (Node Package Manager): cuando usamos Node.js, rápidamente tenemos que instalar módulos nuevos (librerías), ya que al ser un sistema fuertemente modular viene prácticamente “vacío”. Por lo tanto, para utilizar una funcionalidad de alguna librería publicada, deberemos instalar módulos adicionales.

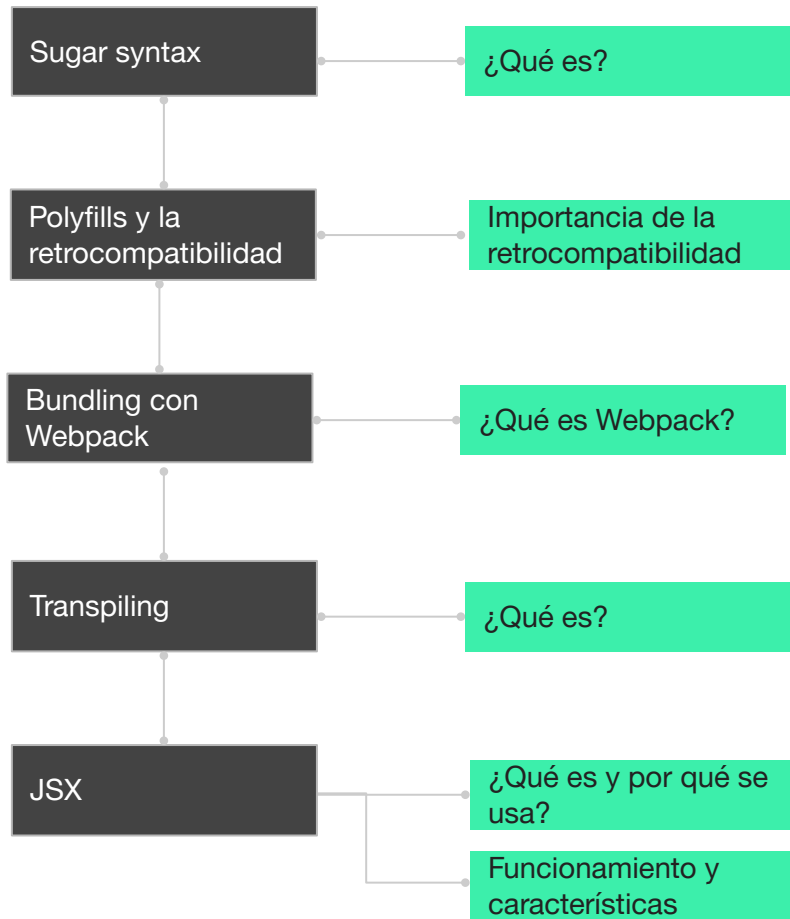
Esta operación se realiza de forma muy sencilla con esta herramienta.

CLI: la interfaz de línea de comandos o interfaz de línea de órdenes es un método que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 3

¡Para
recordar!



CRONOGRAMA DEL CURSO

Clase 2



Instalación y configuración del entorno



EJEMPLO EN VIVO



CREAR LA APP UTILIZANDO EL CLI

Clase 3



JSX y Webpack



EJEMPLOS EN VIVO



MENÚ E-COMMERCE

Clase 4



Componentes I



EJEMPLOS EN VIVO



ESTILOS Y HOME

SUGAR SYNTAX

¿POR QUÉ EXISTE EL SUGAR-SYNTAX?

RECORDEMOS:

Sugar Syntax refiere a la sintaxis agregada a un lenguaje de programación con el objetivo de hacer más fácil y eficiente su utilización. Favorece su escritura, lectura y comprensión.

```
i = i + 1 → i++
```

¿POR QUÉ EXISTE EL SUGAR-SYNTAX?

Los lenguajes tienen una tendencia natural a evolucionar en la manera de ser escritos.

Causas principales:

- Críticas de la comunidad.
- Grado de adoptabilidad.
- Dificultad de implementar patrones de diseño comunes en otros lenguajes.

C#

```
//[modificadores de acceso] - [class] - [identificadores]
public class Customer
{
    // Fields, properties, methods and events go here...
}
```

¿POR QUÉ EXISTE EL SUGAR-SYNTAX?

```
class Animal {  
  constructor(type) {  
    this.type = 'Cat';  
  }  
  
  talk() {  
    console.log('meow');  
  }  
}
```

Para lograr esto, JS **ES5** necesitaría implementar un código parecido al siguiente.

```
"use strict";  
  
function _instanceof(left, right) { if (right != null && typeof Symbol !==  
"undefined" && right[Symbol.hasInstance]) { return !!right[Symbol.hasInstance](left);  
} else { return left instanceof right; } }  
  
function _classCallCheck(instance, Constructor) { if (!_instanceof(instance,  
Constructor)) { throw new TypeError("Cannot call a class as a function"); } }  
  
function _defineProperties(target, props) { for (var i = 0; i < props.length; i++) {  
var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false;  
descriptor.configurable = true; if ("value" in descriptor) descriptor.writable =  
true; Object.defineProperty(target, descriptor.key, descriptor); } }  
  
function _createClass(Constructor, protoProps, staticProps) { if (protoProps)  
_defineProperties(Constructor.prototype, protoProps); if (staticProps)  
_defineProperties(Constructor, staticProps); return Constructor; }  
  
var Animal = /*#__PURE__*/function () {  
  function Animal(type) {  
    _classCallCheck(this, Animal);  
  
    this.type = 'Cat';  
  }  
  
  _createClass(Antimal, [{  
    key: "talk",  
    value: function talk() {  
      console.log('meow');  
    }  
  }]);  
  
  return Animal;  
}();
```

EJEMPLOS DE SUGAR-SYNTAX

Implementando los patrones y sugars adecuados podemos **mejorar la legibilidad y pragmatismo** de nuestro código:

```
> const condition = true;
let result = null;
if (condition) {
  result = 'correct';
} else {
  result = 'incorrect';
}

console.log(`This is ${result}`);
This is correct                                VM697:9
```

Con **ternary operator**:

```
> const condition = true;
console.log(`This is ${condition ? 'correct' : 'incorrect'}`);
This is correct
```

VI

OTROS EJEMPLOS

- Spread operator

`[a, ...arr]`

- Propiedades dinámicas

`{ foo: "bar", ["baz" + id]: 42 }`

- Deep matching

`var { a: val } = { a : 2 }`

- Asignación en desestructuración

`var [a = 1, b = 2, c = 3, d] = [4, 5]`

VAMOS AL CÓDIGO



CODER HOUSE

POLYFILLS Y LA RETROCOMPATIBILIDAD

***¿POR QUÉ NECESITO SER
RETROCOMPATIBLE?***

Cuando desarrollemos y pensemos la **experiencia** de nuestras aplicaciones, es importante tener en cuenta qué distribución tiene hoy el mundo, así como nuestro **target** de usuarios.



Can I use

fetch

?



Settings

50 results found

Fetch - LS

Usage

% of all users

?

Global

95.43% + 0.07% = 95.5%

A modern replacement for XMLHttpRequest.

Current aligned

Usage relative

Date relative

Apply filters

Show all

?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android
		2-33	4-39		10-26						
		^{1 4} 34-38	² 40		² 27						
	12-13	⁴ 39	^{2 3} 41	3.1-10	^{2 3} 28	3.2-10.2					
6-10	14-83	40-78	42-83	10.1-13	29-68	10.3-13.3		2.1-4.4.4	12-12.1		
11	84	79	84	13.1	69	13.5	all	81	46	84	68
		80-81	85-87	14-TP		14.0					

***UNA HISTORIA DE RETROCOMPATIBILIDAD:
EL MUNDO DE LOS “SUMADORES” ES
INVADIDO POR LOS “MULTIPLICADORES”***

ÉRASE UNA VEZ...

El **antiguo mundo** que siempre fue conocido por saber la existencia de los números y su capacidad para sumarlos...

CONSIDEREMOS LA SITUACIÓN...

Al llegar los multiplicadores, empezaron a colonizar y establecieron la multiplicación como **nuevo método** de operar.

CONSIDEREMOS LA SITUACIÓN...

Obligados por la fuerte falta de inclusión que generó esto, apareció 'Francis **Polyfill**' con una gran idea...

CONSIDEREMOS LA SITUACIÓN...

*“¡Sabemos que ustedes no pueden multiplicar,
pero no os preocupéis!”*

Si les cuesta multiplicar... **pueden tratar lo
siguiente:** *“¡simplemente sumen!”*

CONSIDEREMOS LA SITUACIÓN...

Si nosotros hacemos **10** x **2**, y ustedes no saben hacerlo, simplemente hagan lo que hicieron siempre: ¡sumen!

$$2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 = 20$$

RESULTADO

Entonces se entendieron y se integraron.

Crearon una **manera de resolver un problema nuevo** con los recursos que ya tenían, y nadie quedó excluido de multiplicar.

CONCLUSIÓN

Los **polyfills** nos permiten hacer nuestra aplicación compatible con navegadores antiguos, que no admiten de forma nativa alguna nueva funcionalidad

¿CÓMO SE INTEGRA UN POLYFILL?

Ejemplo: core-js

[zloirock/core-js: Standard Library](https://github.com/zloirock/core-js)

```
npm install --save core-js@3.6.5
```

```
import 'core-js'; // <- at the top of your entry point

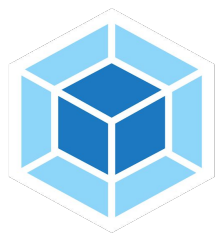
Array.from(new Set([1, 2, 3, 2, 1]));           // => [1, 2, 3]
[1, [2, 3], [4, [5]]].flat(2);                  // => [1, 2, 3, 4, 5]
Promise.resolve(32).then(x => console.log(x)); // => 32
```



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

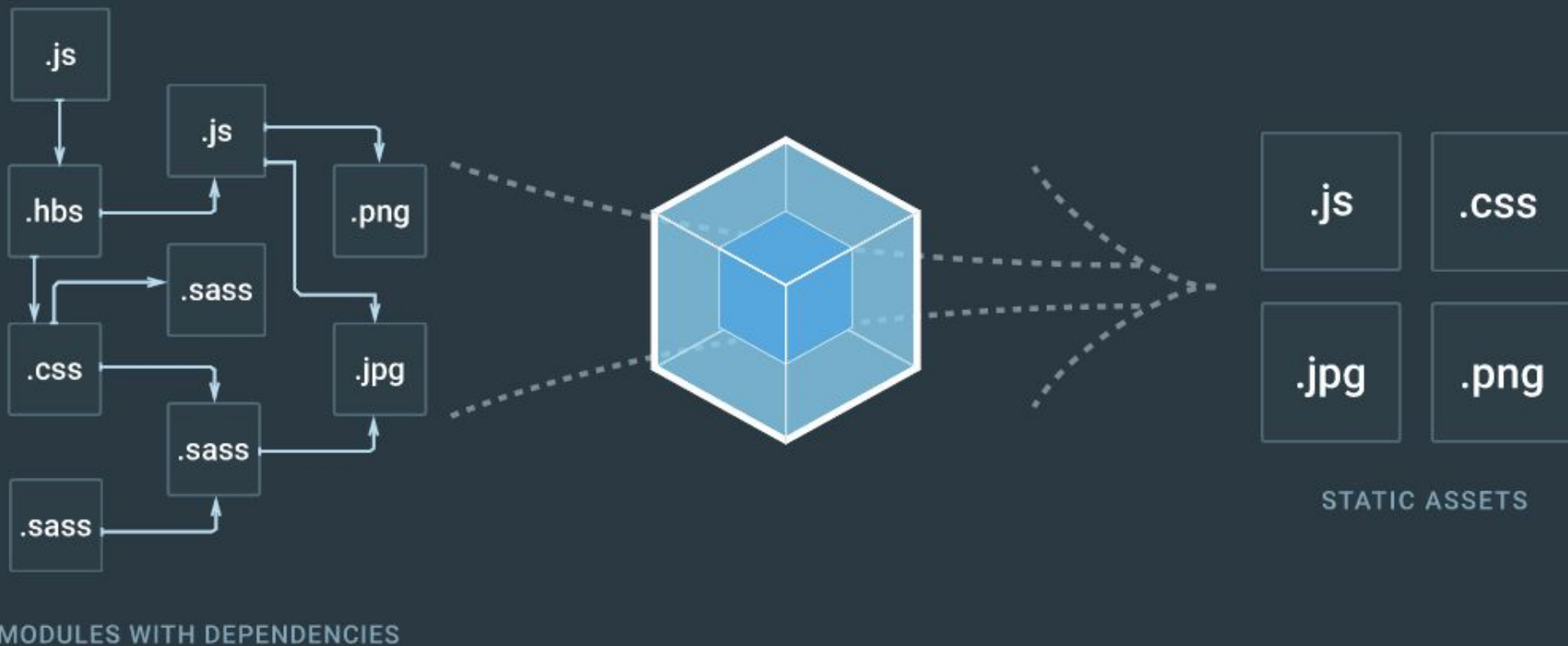
BUNDLING CON WEBPACK



webpack

Webpack **es un *module bundler* o empaquetador de módulos** que nació a finales de 2012, y en la actualidad es utilizado por miles de proyectos de desarrollo web Front-End.

Incluido desde React o Angular hasta en el desarrollo de aplicaciones conocidas como Twitter, Instagram, PayPal, o la versión web de Whatsapp.



Transformación de los módulos en Webpack.

¿CÓMO FUNCIONA?

Podemos tener, por ejemplo, un módulo JS que vaya a depender de otros módulos .js, con imágenes en diferentes formatos como JPG o PNG. O estar utilizando algún preprocesador de CSS, como puede ser SASS, Less y Stylus.

Webpack recoge todos estos módulos y los transforma a assets que puede entender el navegador, como por ejemplo archivos JS, CSS, imágenes, videos, etc.

¿CÓMO NOS AFECTA EN NUESTRO DESARROLLO?

Internamente está incluido en la aplicación generada por
create-react-app.

Importante: **el equipo de react es quien se encarga de mantener estas configuraciones actualizadas.**

Podemos modificarlas, pero para eso necesitamos realizar un **eject.**

```
> react-scripts eject
```

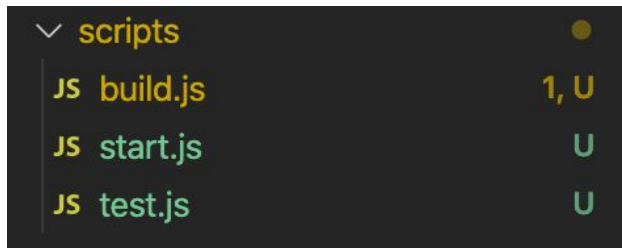
```
NOTE: Create React App 2+ supports TypeScript, Sass, CSS Modules and
```

```
? Are you sure you want to eject? This action is permanent. (y/N) █
```

¿EJECT?

Es una acción **permanente**, que permite tener un control más específico del bundling, a costa de que de ahora en adelante tendremos que encargarnos de mantenerlo.

```
npm run eject
```



COSTO Y ALTERNATIVAS

En algunas oportunidades, cuando tengamos más experiencia, nos puede dar **más flexibilidad**, pero **no siempre** es el caso. Hay algunos proyectos que dan alternativas, como **rewired**, pero el resumen es:



Dan Abramov
@dan_abramov

Replying to @AdamRackis

"Stuff can break" — Dan Abramov

7:57 PM · Sep 28, 2018 · [Twitter Web App](#)

*“Las cosas se
pueden romper...”*

TRANSPILING

¿QUÉ ES EL TRANSPILING?

TRANSPILING

Es el proceso de **convertir código** escrito en un lenguaje, **a su representación en otro lenguaje.**

Usualmente extienden o simplifican la escritura del lenguaje, o representación original.

- Implementan un proceso similar conceptualmente al **pollyfilling.**
- Logran niveles de simetricidad y simbiosis con el lenguaje original.



JSX

***¿QUÉ ES Y POR QUÉ LO
USAMOS?***

CODER HOUSE

JSX

Javascript Syntax Extension



javascript **x**ml

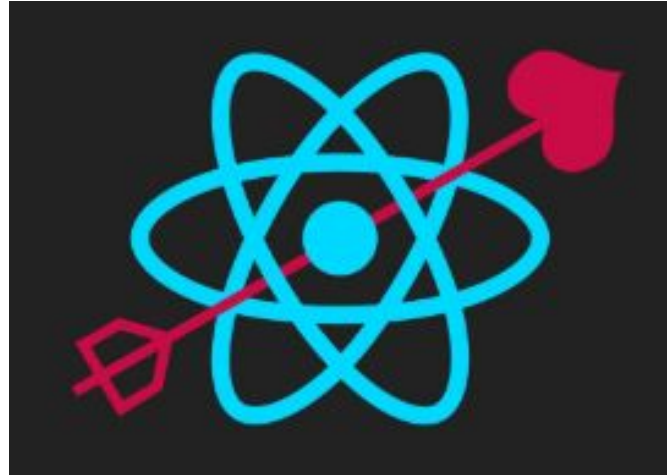
JSX es una extensión de sintaxis de Javascript que se parece a HTML

Oficialmente, es una **extensión que permite hacer llamadas a funciones y a construcción de objetos**. No es ni una cadena de caracteres, ni HTML.

Una vez compilado son archivos JS con funciones y objetos normales

JSX es una extensión de Javascript,
no de React.

Esto significa que **no hay obligación de utilizarlo**, pero **es recomendado** en el sitio web oficial de React.



FUNCIONAMIENTO Y CARACTERÍSTICAS

¿CÓMO FUNCIONA?

JSX se transforma en código
JavaScript.

```
<div className="active">Hola Coders</div>  
  
React.createElement('div', { className: 'active'}, 'Hola Coders');
```

Esto nos da algunas ventajas, como ver errores en tiempo de compilación, asignar variables, retornar métodos, etc.

STYLING EN JSX

Es posible definir y utilizar estilos inline en JSX, solo necesitamos convertirlos por convención:

border-color => borderColor
padding-top => paddingTop

'10px' => 10 (no es necesario el px)

```
let styles = {  
  borderColor: '#999'  
};  
  
const jsx = (  
  <div style={styles}>  
    Hola Coders  
  </div>  
)
```


INLINE STYLES EN JSX

Los mismos estilos se pueden configurar **inline** en JSX, solo necesitamos usar doble llave `{{ }}`,

- La **primera** llave para avisar que se agregará **un objeto** en **js**.
- La **segunda** llave para empezar a escribir el objeto en sí.

```
const Salute = () => <p style={{ marginLeft: 15}}>Hello</p>
```

REGLAS GENERALES

- **Los elementos deben ser balanceados.** Por cada apertura debe haber un cierre.

```
<div src=""> Mal  
<div src=""></div>
```

- **Las etiquetas de solo apertura como <link> o <input> deberán finalizar con /> (auto-cerrado)**

```
<img src="" /> Ideal
```

REGLAS GENERALES

- **Cada componente/función debe tener un return**
- **En este return debe haber máximo un solo elemento en el nivel máximo**

REGLAS GENERALES

Class es palabra reservada, en su lugar usar className.



Mal

`` Ok

```
return (
  <h1 className='large'>Hello World</h1>
);
```



VAMOS AL CÓDIGO



CODER HOUSE

¡NO OLVIDEMOS!

En JSX se utilizan tanto los estilos como los eventos estándar del DOM, como

onclick, **onchange**,

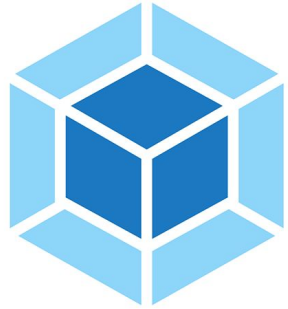
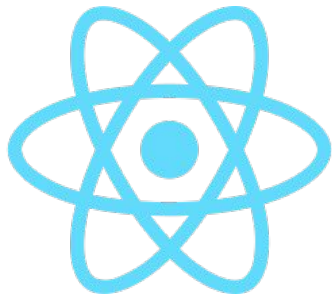
onkeydown, etc. pero

utilizando **camelCase**:

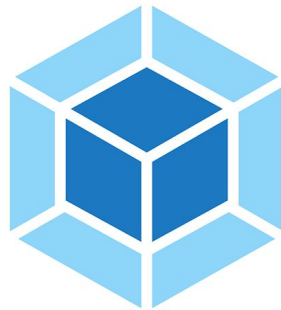
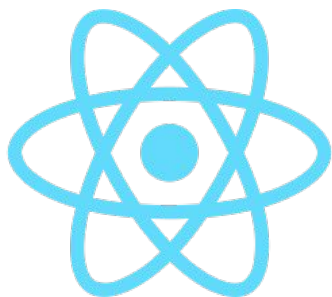
onClick, onChange,
onKeyDown / marginTop,
paddingBottom, etc.

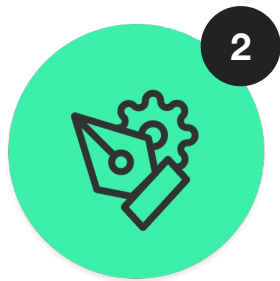
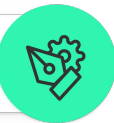


A medida que nuestra aplicación va creciendo y tenemos componentes más grandes que manejan distintos eventos, JSX nos va a ayudar mucho a agilizar y organizar nuestro desarrollo de componentes.



Para poder utilizar JSX en nuestra aplicación, debemos tener instalado **Webpack** que en nuestro caso viene incluido con **create-react-app**





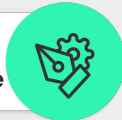
MENÚ E-COMMERCE

Crea una carpeta dentro de src llamada “components”, que contenga a NavBar.js, para crear una barra de menú simple.

MENÚ E-COMMERCE

Formato: link al último commit de tu repositorio en Github. Debe tener el nombre "Idea+Apellido".

Desafío
entregable



>> Consigna: en el directorio de tu proyecto, crea una carpeta dentro de src llamada "components", que contenga a NavBar.js para crear una barra de menú simple.

>>Aspectos a incluir en el entregable:

Crea una carpeta dentro de src llamada components que contenga a NavBar.js para crear una barra de menú simple, que tenga:

Brand (título/nombre de la tienda)

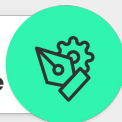
Un listado de categorías clickeables

Incorpora alguna librería de estilos con bootstrap/materialize u otro de tu preferencia (opcional).

MENÚ E-COMMERCE

Formato: link al último commit de tu repositorio en GitHub.

Desafío
entregable



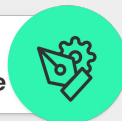
>> Ejemplo:

```
function App() {  
  return <>  
    // Desarrolla tu implementación de un navbar  
    dentro del componente NavBar.js  
    <NavBar />  
    <h2>Las ofertas de la semana</h2>  
  </>;  
}
```

MENÚ E-COMMERCE

Formato: link al último commit de tu repositorio en GitHub.

Desafío
entregable



>> Ejemplo:



Inicio

Vehiculos

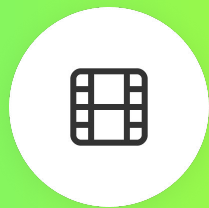
Electrónica

Libros

Login

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***

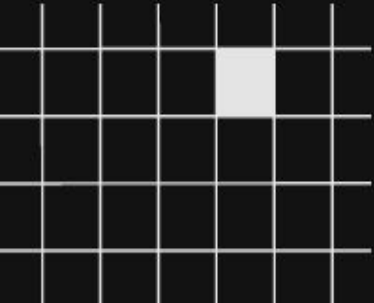


- React.js Essentials (1 ed.). EEUU, Packt. | ***Fedosejev, A. (2015)***
- ReactJS by Example (1 ed.). EEUU, Packt. | ***Amler (2016)***
- ReactJS Cookbook (1 ed.). EEUU, Packt. | ***Stein, J. (2016)***
- <https://caniuse.com> | ***Can I use***



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Sugar syntax.
 - Retrocompatibilidad.
 - Webpack.
 - JSX.
- 



OPINA Y VALORA ESTA CLASE