Contents

[1) DIFERENCIA ENTRE JQUERY Y REACT.JS 3](#_Toc43057112)

[2) Css -> 4](#_Toc43057113)

[ CSS tiene propiedades y selectores 4](#_Toc43057114)

[ Imagenes en CSS: 6](#_Toc43057115)

[3) CSS GAP: 6](#_Toc43057116)

[4) CSS AVANZADO: 6](#_Toc43057117)

[Note: 7](#_Toc43057118)

[ Algunas propiedades flexbox: 7](#_Toc43057119)

[ Extras flexbox: 7](#_Toc43057120)

[5) CSS AVANZADO: 7](#_Toc43057121)

[ CSS3: 7](#_Toc43057122)

[6) BOOTSTRAP 8](#_Toc43057123)

[7) MAILCHIMP 8](#_Toc43057124)

[8) CSS GRID and FLEXBOX 9](#_Toc43057125)

[ Media Queries for Standard Devices 10](#_Toc43057126)

[9) JAVASCRIPT 10](#_Toc43057127)

[10) DOM (document object model) 11](#_Toc43057128)

[11) DOM events 12](#_Toc43057129)

[12) DOM Selectors and JQUERY issues: 13](#_Toc43057130)

[go and check DOM -> selectors on ZTM folder in PUCMM cloud 13](#_Toc43057131)

[13) JAVASCRIPT ADVANCED: 14](#_Toc43057132)

[ CONTROL FLOW 14](#_Toc43057133)

[14) ES5 and ES6 14](#_Toc43057134)

[ Tambien tenemos los template strings( ` ` ) para agregar valores de variables dinamicamente en un string usando ${} e incluso hacer calculos y lo que quieres con ese valor 16](#_Toc43057135)

[15) JAVASCRIPT ADVANCED PART 2 (video 144 en Adelante) 17](#_Toc43057136)

[16) Pass by value vs pass by reference. 20](#_Toc43057137)

[ Como clonar objetos sin que los cambios del clon afecten al clonado: 20](#_Toc43057138)

[ Type Coercion = 22](#_Toc43057139)

[17) JAVASCRIPT ADVANCED (ES7) 22](#_Toc43057140)

[18) ES8: introduced in 2017 23](#_Toc43057141)

[19) ES10: introduced in 2019 (ES2019). Saltamos el ES9 por que este incluye JS asíncrono por lo cual es mejor tratarlo luego de cursar esta parte de JS 25](#_Toc43057142)

[20) JS Advanced Looping: there is for,while, dowhile, forEach but there are also FOR OF for iterable and FOR IN for numerable. 27](#_Toc43057143)

[21) Debugging, figure out what a function does and whether its working fine or not. 29](#_Toc43057144)

[22) How JS Really works? VIDEO 163 30](#_Toc43057145)

[23) What are JS Modules 32](#_Toc43057146)

[***-Inline script:*** 32](#_Toc43057147)

[**-script tags:** 33](#_Toc43057148)

[***-IIFE:*** 33](#_Toc43057149)

[***-Browserify:*** 33](#_Toc43057150)

[***-Webpack + ES6*** 34](#_Toc43057151)

[24) GITHUB – source control 35](#_Toc43057152)

[**Upload project to github repository:** 36](#_Toc43057153)

[**FORK UP TO DATE** 38](#_Toc43057154)

[25) NPM (NODE PACKAGE MANAGER) 39](#_Toc43057155)

[26. Node JS (AND NPM registry how works and commands) 39](#_Toc43057156)

[--INSTALLING LIVE SERVER AND LODASH 40](#_Toc43057157)

[--LIVE SERVER 40](#_Toc43057158)

[-- LODASH 40](#_Toc43057159)

[-- DEPENDENCY INSIDE PACKAGE.JSON FILE 41](#_Toc43057160)

[26) SCSS (It’s a file format for SAS, its kind of css but give extra features) 42](#_Toc43057161)

[27) REACT 42](#_Toc43057162)

[28) React (Install React but with create-react-app tool) 43](#_Toc43057163)

[-- Creating a React App before 44](#_Toc43057164)

[-- Creating a React App now 44](#_Toc43057165)

[when the app is created inside this files are: 44](#_Toc43057166)

[In SRC: 44](#_Toc43057167)

[29) JSX 44](#_Toc43057168)

[**Adding styles to JSX** 45](#_Toc43057169)

[30) TACHYONS 45](#_Toc43057170)

[31) ROBOFRIENDS APP 46](#_Toc43057171)

[32) React rules: 47](#_Toc43057172)

[33) Deploying project in github 50](#_Toc43057173)

[-add homepage 50](#_Toc43057174)

[- install github pages 51](#_Toc43057175)

[- add scripts to package.json: 51](#_Toc43057176)

[- npm run deploy 51](#_Toc43057177)

[- Go to settings inside repository and change to gh-pages branch 51](#_Toc43057178)

[34) REDUX 52](#_Toc43057179)

[That package needed for redux to connect to react 52](#_Toc43057180)

[35) middleware 54](#_Toc43057181)

[36) REDUX asincronous actions 54](#_Toc43057182)

[Whole explanation for redux with thunk and logger middlewares 55](#_Toc43057183)

[37) REACT UTILITIES 56](#_Toc43057184)

[**REACT ROUTER** 56](#_Toc43057185)

[RAMDA 56](#_Toc43057186)

[**LODASH** 57](#_Toc43057187)

[STYLING JS 57](#_Toc43057188)

[GATSBYJS.ORG 57](#_Toc43057189)

[Zeit.co for NEXT.JS 57](#_Toc43057190)

[**MATERIAL-UI.com** 57](#_Toc43057191)

[SEMANTIC-UI 57](#_Toc43057192)

[RESELECT 57](#_Toc43057193)

[THUNKS 58](#_Toc43057194)

[REDUX-SAGA 58](#_Toc43057195)

[IMMUTABLE JS 58](#_Toc43057196)

**ZTM 1**

# DIFERENCIA ENTRE JQUERY Y REACT.JS

A big difference between these two is that React works through the “virtual DOM”, whereas jQuery interacts with the DOM directly. The virtual DOM is a DOM implementation in memory that compares to the existing DOM elements and makes the necessary changes/updates. And that leads to much faster performance.

In React, A single component contains both the functional part of the View and the UI itself. That’s right! With React you code your UI elements in JSX, a syntax extension of HTML. Although it might sound counter-intuitive, it turns out it is much more efficient, and you have better control and maintenance over your code

* UI, Window and view:
* Hay 3 Cosas, ui, ventanas y views o vistas
* Entonces usted en el VS code tiene ej una vista del código html, luego decide abrir la vista del script.js
* Ahora tienes 2 views
* Cada uno tiene una ventana
* Entonce tienes 2 views contenidos en 2 ventanas
* Pero todo eso conforma el ui
* Tienes 2 views contenidos en 2 windows que conforman un ui
* En las pag pasa que todo es un window, un botón es un window, un texbox es un window donde lo que está dentro es el view de ese window
* Y al final todos los windows es tu ui

FAST LINKS ->

ZTM CAPTIONS FOR EACH SECTION : <https://github.com/zero-to-mastery/zero-to-mastery-captions/tree/master/Spanish/section11>

# Css ->

## CSS tiene propiedades y selectores

Los selectors son como,   
::placeholder o :hover como Mientras que propiedades pueden ser Align-content, background-color, etc.

-en CSS las cosas se siguen con orden.

CSS Cheat Sheet

Reference:

\*https://www.w3schools.com/cssref/css\_selectors.asp

\*https://css-tricks.com/almanac/

Cascading Style Sheets at the most basic level it indicates that the order of CSS rules matter.

**.class** ->>> si ponemos dentro de un <section>

<p class=”webtext”>Lorem ipsum.</p> esto es una forma de seleccionar un grupo de elementos y asegurarte que tendrán las mismas cosas

y luego en style.css declaramos la clase .webtext y le asignamos propiedades { border: 5px dashed green; )

**#id** --- (debemos poner el signo de numero al declarar una clase id que normalmente esta en un <div id=”div1”> cuando la declaramos en el style.css)

**\* --** con la estrella o asterisco se selecciona todo en el style.css

**Element** –-- un elemento reside en style.css ej body {property: value;} o h2{}

**element, element2** –-- normalito lo mismo pa los 2

**element1 element2** –-- aqui pasa que se aplican las propiedades a todos los elementos 2 que están dentro de los elementos 1 directamente

**element > element2** –-- igual que la anterior pero para todos los elementos 2 que tienen como padre a los elementos 1

**element + element2** --- se aplican las propiedades a todo elemento que este inmediatamente despues de elemento 1

**:hover ---** un selector que hace efecto hover cuando pasas el mouse coje las propiedades que le has dado

**:last-child ---** tomar el ultimo hijo en un montón de hijos

**:first-child –--** tomar el primer hijo

!**important (not recommended**)aplica una propiedad a toda clase con el mismo nombre sin importar que hayan otras ni en que orden

What seletors win out in the cascade depends on:

**-Specificity ---** entre mas detalles tenga la clase en style.css mas specificity. Ej : #outer a {bg-color: red;} es menos especifico que

#outer #inner a {bg-color: red;}

Inline styles always win over others in specificity: <section style=”color:red” id=”red”>

**-Importance ---** the first in order is more important than the next

**-Source Order --- same as before**

**That where selectors**

## Imagenes en CSS:

---= Float --- la propiedad float hace que el texto rodee una imagen

---= .Boxmodel {

border le da forma al borde

display: inline-block; le da forma de bloques a cada clase

padding: 5px 20px 5px 20px; le da un tamaño de x pixeles a top, right, bottom, left (espaciado adentro del bloque)

margin: toppx rightpx bottompx leftpx; o top&bottom Right&left; le da un espaciado a cada bloque (espaciado afuera del bloque)

--- PX pixels size

--- EM relative to father pixels size

--- REM size in relation to the root element (html element = H1, or Hx tag as father)

Ejercicios prácticos con CSS (practicamente todo lo que se puede hacer con css con ejemplos prácticos)

<https://www.w3schools.com/css/exercise.asp?filename=exercise_boxmodel2>

# CSS GAP:

<https://developer.mozilla.org/en-US/docs/Web/CSS/gap>

# CSS AVANZADO:

* Algo a tomar en cuenta es el hecho de que la pagina tiene que cargar cada cosa, html, css, etc y para reducir el tiempo de carga y ancho de banda se puede reducir el código, para CSS existe css minify lo cual comprime todo el código css en una línea. <https://www.cleancss.com/css-minify/>

-= FLEXBOX es la forma de solucionar el problema en CSS de tener las imágenes en el lugar que las deseamos.

Note: Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the [Grid](https://css-tricks.com/snippets/css/complete-guide-grid/) layout is intended for larger scale layouts.

-CSS tricks: todo sobre flexbox: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

**-**- CSS tricks: todo sobre Grid: <https://css-tricks.com/snippets/css/complete-guide-grid/>

## Algunas propiedades flexbox:

==display: flex; te lo pone todo en una línea y puedes deslizar

==flex-wrap: wrap; te lo pone todo de modo que quepa en la pagina sin tener que deslizar y hace que el diseño sea responsive.

==justify-content: center; para que todo se quede centrado siempre

## Extras flexbox:

Practica flexbox <http://flexboxfroggy.com/#es>

FLEXBOX CHEAT SHEET <https://darekkay.com/dev/flexbox-cheatsheet.html>

# CSS AVANZADO:

## CSS3:

Siendo lo mas Avanzado de CSS hasta ahora en general con esto se agregan nuevas propiedades para hacer las paginas web aun mas dinámicas. Con esto se puede hacer cosas como que las imágenes se muevan cuando pasas el mouse (when you hover over it) para esto en las propiedades de la imagen se agrega transition: all 1s; por ejemplo y se crea una clase img:hover {

Transform: scale(1.1)}; para que aumente en 1.1 veces el tamaño de la imagen. Y con lo anterior este aumento lo hará al cabo de 1 segundo con el mouse encima.

---Mas propiedades con efecto de transición <https://thoughtbot.com/blog/transitions-and-transforms>

--- para saber si el browser soporta las propiedades que haz agregado <https://www.w3schools.com/cssref/css3_browsersupport.asp>

<https://caniuse.com/>

--- para asegurarse de que el browser soporte las propiedades que haz agregado <https://autoprefixer.github.io/>

--- practica CSS <https://www.freecodecamp.org/learn/responsive-web-design/basic-css/use-rgb-values-to-color-elements>

# BOOTSTRAP

Libreria de CSS <https://getbootstrap.com/>

<https://expo.getbootstrap.com/>

hay 2 formas de usar bootstrap o lo descargas y lo usas local o copias la cdn lo cual se refiere a los datos que ellos alojan en la nube para que los recursos puedan ser accesados remotamente. Se apoya de javascript con los codigos para hacer cosas como un panel desplegable al momento de darle a un botón (modal button)

Bootstrap grid es un sistema de cuadriculas que se aplica como <div class=”col col-sm-4 col-md6”> haciendo que la pagina se vea bien sin importar el size

---- meta tags en las paginas son cosas útiles, van dentro del head tag y por ej meta charset=”utf-8” el cual es una forma de encoding to write text on our pages.

Meta name=”viewport” te hace la pagina responsive.

Bootstrap usa un sistema de cuadriculas o grid donde ej text-center col-12 te centra todo en la pagina horizontalmente.

---luego para la orientación vertical en Bootstrap se usa flexbox habilitándolo con d-flex y luego align-items-center y como todo estará dentro de un container debes hacer h-100 para que el container ocupe todo el espacio de la pagina y el centrado se vea bien.

<https://getbootstrap.com/docs/4.3/utilities/flex/> --- flexbox boostrap

# MAILCHIMP

Para tener email subscribe form correos de clientes y enviarles mensajes todo en la pagina y en internet <https://mailchimp.com/>

PARA HACER CUAL ANIMACION CON CSS

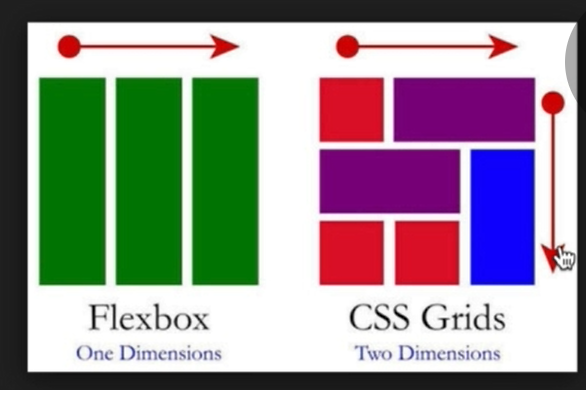
<https://daneden.github.io/animate.css/>

PLANTILLAS SITIOS WEB <https://zerotomastery.io/resources/>

PLANTILLAS PARA CSS <https://www.creative-tim.com/>

PLANTILLAS html5/CSS3 <http://www.mashup-template.com/templates.html>

# CSS GRID and FLEXBOX

with it you won’t even need bootstrap 4

Para usarlo en el container se pone display: grid; y se pone el num de columnas a usar con grid-template-columns: 300px 300px por ej haciendo 2 columnas siempre de 300px. También se usa grid-gap: 20px por ej para el espacio entre cuadriculas.

SIN EMBARGO USANDO FR EN VEZ DE PX estamos utilizando fracciones de la pagina y esto hace que se vea todo proporcional y perfecto.

Todo de CSS GRID <http://grid.malven.co/>

Practica CSS GRID <http://cssgridgarden.com/#es>

## Media Queries for Standard Devices

Para que la pagina se adapte a dispositivos específicos.

<https://css-tricks.com/snippets/css/media-queries-for-standard-devices>

PRETTIFY: free design resources <https://zerotomastery.io/resources/>

# JAVASCRIPT

Reference js charcodes-> key codes<https://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes>

fue creado en 1995 para Netscape web browser. Fue creado para habilitar acciones en los sitios web como darle a play a un video. Para que todos los sitios webs funcionaran igual sobre la versión de cada browser se creo ECMAscript( ES# **European Computer Manufacturers Association Script** es una especificación de lenguaje o estándar de programación publicada por ECMA International. El desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript propuesto como estándar por Netscape Communications Corporation. Actualmente está aceptado como el estándar ISO/IEC 22275:2018)

Ojo un fragmento de **código escrito que produce un valor** es una expresión que se finaliza con ;

Ojjo es mejor que el script se llame al final de los codigos de html para que la pagina cargue y no parezca que hay delay por estar cargando los scripts de js

* Lo que esta dentro de una función son los argumentos alert(“arguments”)
* Parámetros a,b y argumentos 5,6
* Las funciones son también variables y deben retornar un valor donde la expresión return cerrara la función
* JS array references <https://www.w3schools.com/jsref/jsref_obj_array.asp>
* Objects are collections of properties ej:

Var user = {

Name: “john”,

Age: 34

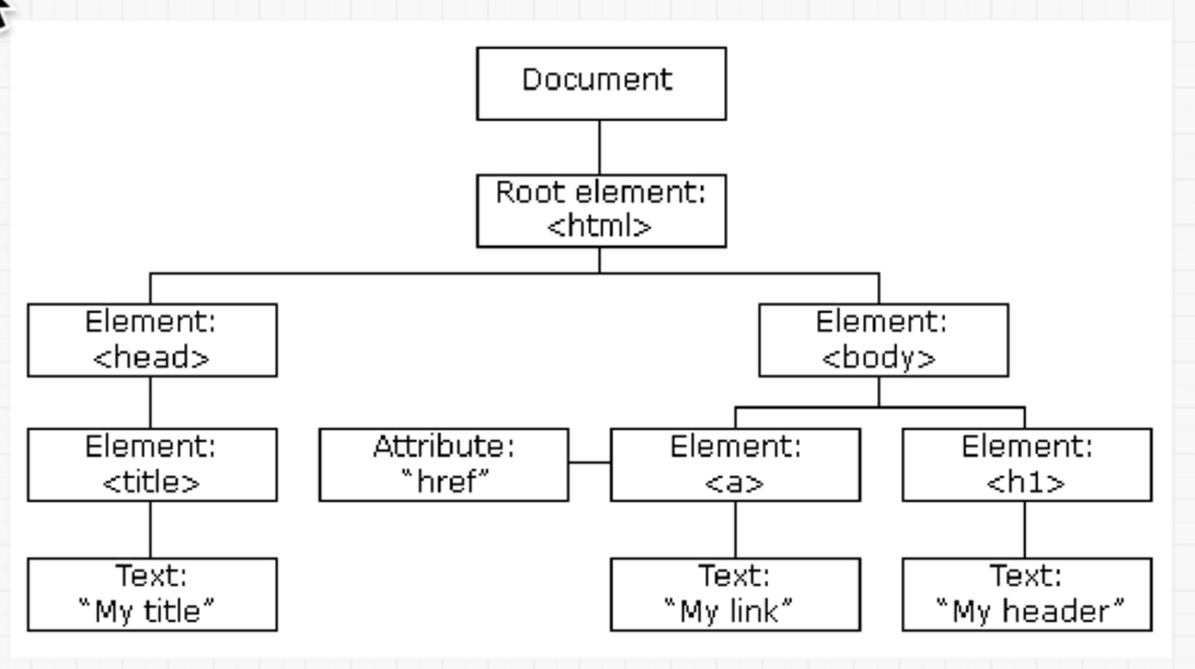
};

\*\*\*para acceder a las propiedades utilizamos el punto: use.age retorna 34

* A function inside an object or inside an array is a METHOD!

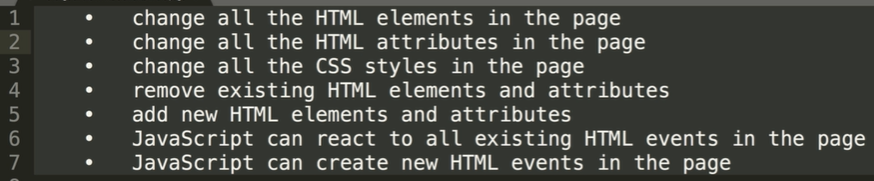
# DOM (document object model)

Con js Podemos evitar tener que volver a llamar el documento html y el documento css. JS puede hacer los cambios mientras el usuario navega por la pagina. Para entender esto hay que entender 2 cosas DOM y el JS engine (para Chrome es el V8 engine, para Edge es chalkerCore, para safari es Nitro y para Mozilla es spiderMonkey y estos son los que miran el archivo de javascript y lo leen línea por línea para ejecutar el codigo). Cuando una pagina carga el browser crea dicho DOM



Se necesita como una forma de obtener, agregar o eliminar HTML elementos. O sea es algo que el browser crea para permitirnos cambiar el html y css

With the object model, JavaScript gets all the power it needs to create dynamic HTML:



Para acceder al DOM lo hacemos mediante el objeto “document”. El cual a su vez tiene un parent object, es decir el objeto document esta contenido dentro de “window” el cual es también un objeto con un montón de propiedades. Window es el big parent de todo, es lo que describe la ventana del browser y por eso hasta hacer window.alert(“test”); funciona. Pero no puedes hacer window.write(“test”); por que no existe como propiedad de window sino solo de su hijo el objecto propiedad de window object.write(“test”); si funcionara. Aunque parezca html, el DOM es un Javascript object.

# DOM events

Dom events reference: <https://developer.mozilla.org/en-US/docs/Web/Events>

Example: aqui se improme en console click!!!! Cada vez que sacas el mouse del boton

var button – document.getElementsByTagName(“button”)[0]

Button.addEventListener(“mouseleave”, function() {

Console.log(“click!!!”);

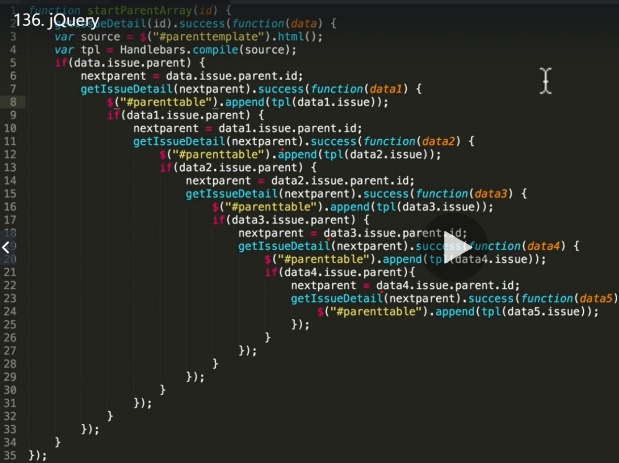
})

Mouseleave es un mouse event, Tambien hay Keyboard Events, estos dos paquetes son los mas utilizados.

# DOM Selectors and JQUERY issues:

# go and check DOM -> selectors on ZTM folder in PUCMM cloud

JQUERY: is an outdated (by REACT.js as its declarative instead of imperative) javascript library que se usaba para manipular el DOM mas facilmente, outdated because of its imperative way of doing things (tienes que decirle explicitamente cada cosa que quieres que haga) llevandote a una pyramid of doom cuando el codigo es muy extenso



First it was html then -> css -> javascript -> jquery to make js easier - > then React.js to solve jquery problems

# JAVASCRIPT ADVANCED:

Scope: por definicion el scope es el acceso que se tiene a las variables cuando el código está corriendo. Por defecto en javascript estas en el root scope que es el window object (que contiene al DOM). Si corres una función preguntara por la variable que busque al parent mas cercano y luego al siguiente parent y así sucesivamente. Ej:

var Fun = 5;

function funFunction () {

var Fun = “klk”;

console.log(Fun);

}

Console.log(Fun);

//aquí pasa que no se imprime el Fun = klk sino que se imprime el Fun = 5 por que es el que esta en el root scope o en el window. Si se ejecutara la función antes del console log entonces se imprimiría fun klk 2 veces. Si se hace después se imprime el Fun = 5 y luego el klk.

## CONTROL FLOW

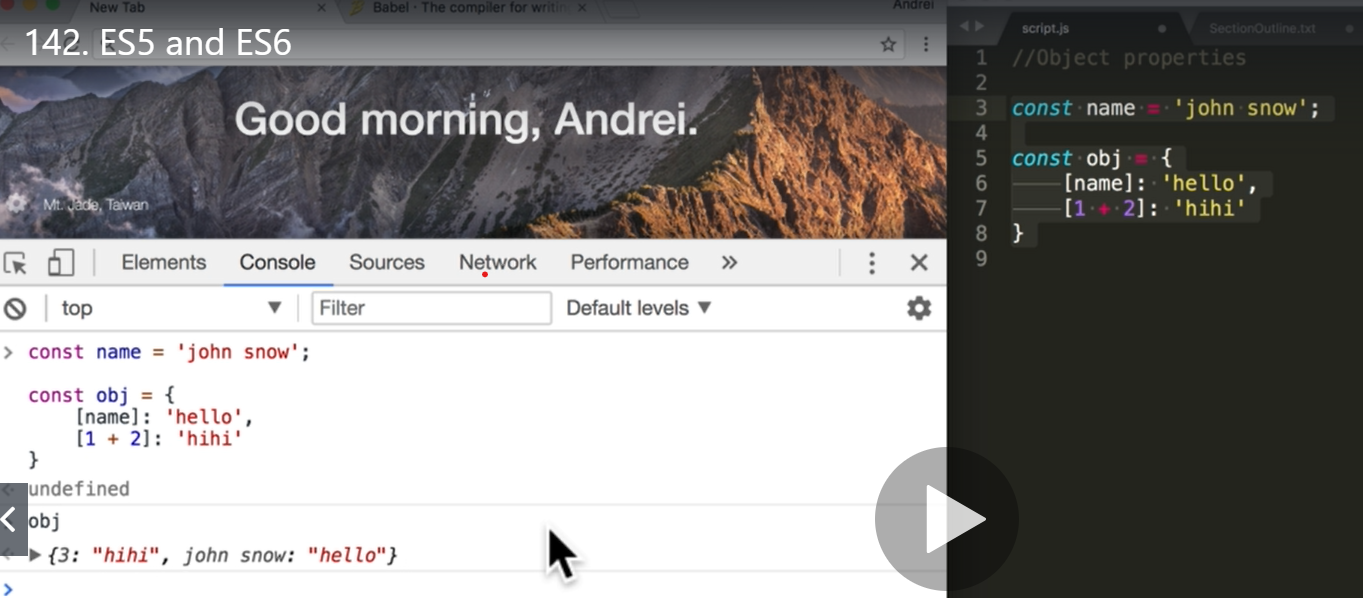
-aquí incluimos js conditionals: if, else, else if y Tambien Ternary operations and switch/case functions

Condition ? expr1 : expr2; eso es un operador ternario.

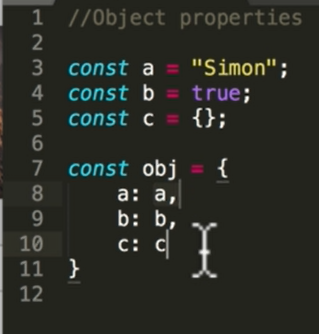
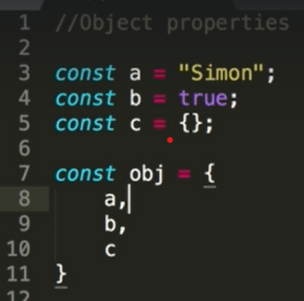
# ES5 and ES6

Esto es lo que pide la industria, siendo ECMAscript una forma estándar de escribir javascript, con BABEL puedes escribir ES6 y BABEL lo traduce a ES1 para que cualquier browser lo entienda.

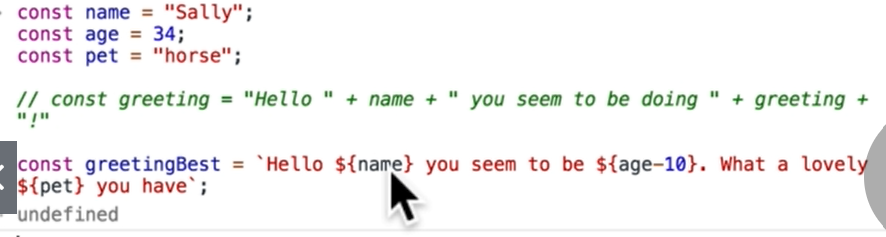
En ES6 tenemos las variables de JS: LET y CONST que hace totalmente innecesario volver a usar var. Mientras que la variable var cambia a la ultima vista o llamada, LET se mantiene por scope, pudiendo tener el mismo nombre pero valores diferentes para cada scope. Mientras que CONST no puedes volver a reasignarla en ningún scope por que te da un error. Sin embargo aunque no puedes reasignar la variable completa, con CONST puedes reasignar los valores de las propiedades dentro de la variable.

Con const se pueden hacer cosas como (en ES6) : 

también con ES6 podemos mejorar en vez de a:a -> a cuando necesitas igualar las propiedades y valores de un objeto

## Tambien tenemos los template strings( ` ` ) para agregar valores de variables dinamicamente en un string usando ${} e incluso hacer calculos y lo que quieres con ese valor



También tenemos los ternary operators ej:

var experiencePoints = winBattle() ? 10 : 1;

También el switch/case

También destructuring ej: const { player, experience } = obj;

# JAVASCRIPT ADVANCED PART 2 (video 144 en Adelante)

//--------------------////   word function into const arrow functions   //////////----------------

//Word function:

*function* first() {

*var* greet = 'Hi';

*function* second() {

        alert(greet);

    }

    return second;

}

*var* newFunc = first();

newFunc();

//but we dont use word functions (with var) anymore. now we use

//const for static unmodificable function value name and arrow functions:

*const* first = () *=>* {

*const* greet = 'Hi';

*const* second = () *=>* {

        alert(greet);

    }

    return second;

}

*const* newFunc = first();

newFunc();

//this was to make sure every time it runs is a new slate. all states starts clean!. to minimize bugs and errors.

//--------------------////   diferent topic comes now  //////////----------------

// CLOSURES

//its a javascript concept where

//A function runs, it gets executed and never executes again

//BUT its going to remember there are references to those variables

//So the childs scope always has access to parent scope

//ej:

*const* first = () *=>* {

*const* value1 = 'Hi';

    alert(value2);

*const* second = () *=>* {

*const* value2 = 'hi2';

        alert(value1);

    }

    return second;

}

*const* newFunc = first();

newFunc();

//aqui el child variable second tiene acceso al parent variable first y por ende a la variable value1 por ende puede proceder hacer alert

//sin embargo el parent variable first no tiene acceso a las variables del child variable second, por ende se muestra como que nadie la esta usando y no puede proceder a hacer el alert de value1

//--------------------////   diferent topic comes now  //////////----------------

//CURRYING  = currificacion

//En la ciencia de la computación, currificar es la técnica inventada por Moses Schönfinkel y Gottlob Frege

//consiste en transformar una función que utiliza múltiples parametros en una secuencia de funciones que utilizan un único parametro.

//ej currying:

*const* multiply = (*a*, *b*) *=>* *a* \* *b*; //una funcion que utiliza multiples parametros

*const* curriedMultiply = (*a*) *=>* (*b*) *=>* *a* \* *b*; //secuencia de funciones que utilizan un solo parametro

curriedMultiply(3); //esto hace que const curriedMultiply = (3) => (b) => 3 \* b; and returns (b) => a \* b

curriedMultiply(3)(4); // === 12

*const* multiplyBy5 = curriedMultiply(5);

multiplyBy5(6); // === 30 porque al correr la primera const multiplyBy5 = curriedMultiply(5); b se hizo igual a 5. por ende ahora es a quien cambia (a)=6 y (b)=5

//--------------------////   diferent topic comes now  //////////----------------

//composing or composition

//its the act of converting to functions into a single function

//you can do it in secuence, where the output of the first is the input of the second but one needs to finish to start the other

//or you can do it in parallel where both can be running at same time to return one value or one third function

//ej:

*const* compose = (*f*, *g*) *=>* (*a*) *=>* *f*(*g*(*a*)); //it reads take functions f and g, take parameter a, then take the value of function f wich needs value of function g

*const* sum = (*num*) *=>* *num* + 1; //sum 1 to the value num wich is what returns compose wich is 5

compose(sum, sum)(5); //it reads function sum will be function f and function sum will be function g, value of 5 will be parameter a.

//o sea basicamente las 2 funciones son sum (f y g) por ende al correr g con el valor 5 siendo sum, esto es 6

//luego corre f con el valor a devuelto por g siendo 6, al correr f =sum esto es 6+1

//el valor devuelto por compose es el valor de f por ende es 6+1 = 7.

//// funciones avanzadas, arreglos avanzados y objetos avanzados ver en js file

# Pass by value vs pass by reference.

Cuando pasamos por valor nos referimos a los valores primitivos es decir, los creados por el mismo js, los diseñados internamente por los que crearon js. Estos son ej números, true, false, etc. Cuando pasamos por valores lo que se hace es asignarle a dicha variable un valor o sea creamos otra cajita (otro espacio de memoria) pero con el mismo contenido y se la asignamos por esto el === lo determina como que son iguales pero en verdad no es el mismo espacio de memoria como pasa con pass by reference.

Cuando hacemos pass by reference lo cual se hace con los objetos, pues es simplemente que creamos una cajita a la cual ambos objetos están apuntando es decir, ese espacio de memoria lo reservamos nosotros y lo que se hace es que ambos objetos apuntan a dicho espacio de memoria.

## Como clonar objetos sin que los cambios del clon afecten al clonado:

* Ej 1 para los arreglos (puesto que los arreglos son también objetos.. estructuras pero sin una longitud predefinida, por esto no son un tipo de dato)

var c = [1,22,3,4,5];

var d = [].concat(c);

d.push(12321342134);

console.log(d) // esto arrojara [1,2,3,4,5, 12321342134] para el arreglo d. pero el arreglo c seguirá intacto.

* Ej 2 para los objetos

let obj = {a: ‘a’, b:’b’, c:’c’}; // objeto principal

let clone = Object.assign( {}, obj); //object.assign no permite que cambie el obj principal

let clone 2 = {…obj} // igual que objet.assign pero mas compacto. Crea nuevo arreglo

obj.c = 5;

console.log(obj);

console.log(clone);

console.log(clone2);

* Ej 3. Shallow clone (clonado superficial) es cuando solo puedes copiar el primer nivel o capa. Para clonar más capas o Deep clone se utiliza (JSON.stringify(obj)) lo cual hace convertir todo dentro del objeto a un string y luego le hacemos parse (analizar gramáticamente) y lo metemos a un objeto nuevamente y así puede mantenerlo todo sin cambios. Esto tiene el inconveniente de que en un archivo masivo hacer deep cloning toma muchos recursos y tiempo.

let obj = {

a: 'a',

b:'b',

c: {

deep: 'try and copy me'

}

}; // objeto principal

let clone = Object.assign( {}, obj); //object.assign no permite que cambie el obj principal

let clone2 = {...obj}; // igual que objet.assign pero mas compacto. Crea nuevo arreglo

let superClone = JSON.parse(JSON.stringify(obj));

obj.c.deep = 'hahah';

console.log(obj);

console.log(clone);

console.log(clone2);

console.log(superClone);

* Type Coercion = the language converting a certain type to another type 1==’1’ it converts ‘1’ string to number and === true

All language have type coercion of any kind (from whatever to 1 and 0 that is what machine knows basically (assembly)). JS have an specially heavy type coercion due to its nature for being a dynamically type language. In the if statement JS coerces 1 into true.

<https://dorey.github.io/JavaScript-Equality-Table/> TYPE COERCION TABLES FOR JS

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness> Object.is(NaN, NaN) in that case util but the page shows whole table.

<https://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3> full bored explanation

# JAVASCRIPT ADVANCED (ES7)

In ES7 they tried tro avoid problem in ES6 where there were so many changes that people had a hard time trying to learn everything. In ES7 we just have 2 new features: .include and \*\* para representar potencias.

Addition 1:

‘helloooooo’.includes(‘o’); //esto retornara true

const pets = [‘cat’, ‘dog’, ‘bat’];

pets.includes(‘dog’); //retronara true también pq lo incluye

addition #2:

const square = (x) => x\*\*2; //exponential 2

const cube = (y) => y\*\*3; //exponential 3

cube (3); //retornara 27 por que 3\*3\*3 = 27

# ES8: introduced in 2017

1)

.padstart ejemplo: ‘Turtle’.padStart(10); === “ Tutle”

.padend ejemplo: ‘Turtle’.padEnd(10); === “Tutle ”

1. Trailing commas

const fun = (a,b,c,d,) => { console.log(a); }

fun(1,2,3,4,)

//cuando son muchos la última coma hace más fácil agregar mas parámetros. También en GitHub lo hace ver mejor por los colores verde y rojo.

1. Object.values, Object.entries, (before this we had Object.keys) in order to be able to use map, filter and reduce en los objetos (parecido a los arreglos)

Ejemplo:

let obj = {

username0: ‘Santa’,

username1: ‘Rudolf’,

username2: ‘Mr.Grinch’

}

Before:

Object.keys(obj).forEach((key, index) => {

console.log(key, obj[key]) //grabbing the object with the key username. imprime cada //cosa completa, username0 Santa etc

})

Now:

Object.values(obj).forEach(value => {

console.log(value);

})

//you get only the values of each object: Santa, Rudolf, Mr. Grinch

Object.entries(obj).forEach(value => {

console.log(value);

})

//here there is a lot of control because you get all data in arrays containing the property and the //value or the key and value and you can do whatever you want, use map, filter, reduce all those //array functions

Ejercicio, tomar de una del arreglo solo los valores con el numero de índice:

Object.entries(obj).map(value => {

return value[1] + value[0].replace(‘username’, “ ”);

/\*esto es que tomar el segundo valor (value 1) por que queremos tomar los valores y no las propiedades (santa,Rudolf,mr. Grinch) luego concatenamos el primer valor (value 0) para aplicarle la funcion replace y quitarle lo de username y quedarnos solo con el numero (username0 sera solo 0)\*/

//esto arrojara Santa0, Rudolf1, Mr.Grinch2 yea

})

# ES10: introduced in 2019 (ES2019). Saltamos el ES9 por que este incluye JS asíncrono por lo cual es mejor tratarlo luego de cursar esta parte de JS

flat()

const nestedArray = [1,2,[3,4,,,,,,,,,[5]]];

nestedArray.flat(3); //it eliminates 3 nest where all will be in one clean array [1,2,3,4,5]

flatMap() as it name suggest it is flat() function plus map() function or method both at the same time

const nestedArray = [1,2,[3,4,,,,,,,,,[5]]];

const nestedArrayChaos = nestedArray.flatMap(nest => nest + ‘5’) //here all nest will be flattened //but containing each one the number 5 also. (here ,,, doesn’t clean)

trimStart() and trimEnd() to delete blank spaces

const userEmail = ‘ [eddyklk@gmail.com](mailto:eddyklk@gmail.com)’

const userEmail2 = ‘ [luisklk@gmail.com](mailto:luisklk@gmail.com)’

console.log(userEmail.trimStart() + userEmail2.trimEnd())

//no blank space but trim() delete blank space from both sides

fromEntries() its just the opposite of entries(). fromEntries() convert an array to an object with properties and values. Entries() just convert that object to an array.

Example: userProfiles = [[‘commanderTom’, 23], [‘derekzlander’, 40]];

const obj = Object.fromEntries(userProfiles); // returns

//{commanderTom: 23, derekzlander: 40}

Object.entries(obj); //returns 0: ["commanderTom", 23] 1: ["derekzlander", 40]

try catch:

before:

try {

bob + ‘hi’

} catch (error) {

Console.log(‘you messed up’ + error) //if no something passed by //parameter as error then code will give you syntax error. This returns you //messed up ReferenceError: bob is not defined.

}

now:

try {

bob + ‘hi’

} catch () {

Console.log(‘you messed up’) //now you don’t are forced to pass //something by parameter and code wont give you syntax error. This //returns you messed up

}

# JS Advanced Looping: there is for,while, dowhile, forEach but there are also FOR OF for iterable and FOR IN for numerable.

We are going to work with array basket and object detailedBasket:

const basket = [‘apples’, ‘oranges’, ‘grapes’];

const detailedBasket = {

apples: 5,

oranges: 10,

grapes: 1000

}

Before:

//for

For (let i = 0; I < basket.length; i++) {

console.log(basket[i]);

}

//forEach

basket.forEach(item => {

console.log(item);

})

//NOW

//for OF

//iterating – array, strings

for (item of basket) {

console.log(item);

}

//for IN – object properties is enumerable is let us see the properties

//enumerating – objects

for (item in detailedbasket) {

console.log(item);

}

//you cant iterate within objects!. But you can enumerate arrays.

//so FOR IN works in array but FOR OF doesn’t work in objects

//you can think arrays as objects.

# Debugging, figure out what a function does and whether its working fine or not.

Steps:

1. Just read it and understand functions in use
2. Desglosar esas funciones en trozos más entendibles
3. Probar cada trozo por separado para ver que está haciendo
4. En vez de probar estos trozos con consolo.log e imprimir, podemos usar debugger

const flattened = [[0, 1], [2, 3], [4, 5]].reduce(

(a, b) => a.concat(b), [ ]);

1. Read the function: estamos asignando una variable como constant flattened tiene un arreglo anidado constituido por 3 arreglos, usando la función reduce tenemos que concatenamos b con a en un arreglo nuevo.

Como reduce traba con accumulator, array o sea tienes la parte que va sumando o acumulando y el arreglo al cual le estamos trabajando

const flattened = [[0, 1], [2, 3], [4, 5]].reduce(

(accumulator, array) => accumulator.concat(array), [ ]);

--entonces tengo el accumulator, el array y el acc comienza con un arreglo vacio al cual le vamos concatenando los arreglos anidados.

1. Y 3. Desglozar en trozos:

const flattened = [[0, 1], [2, 3], [4, 5]].reduce(

(accumulator, array) => {

console.log(‘array’, array);

console.log(‘accumulator’, accumulator);

return accumulator.concat(array)

}, [ ]);

4. con debugger:

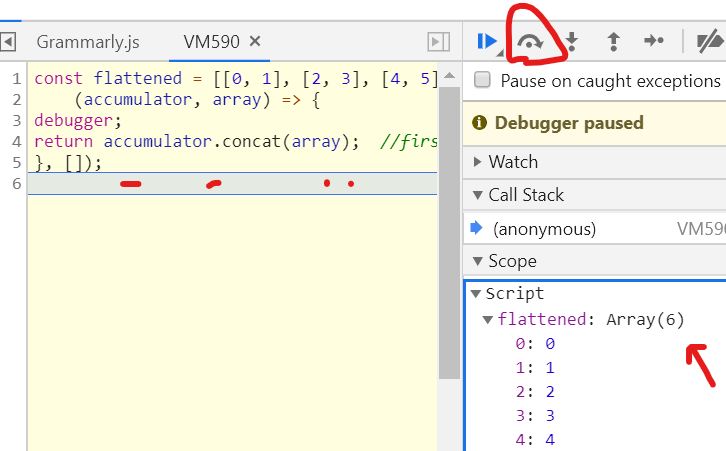
const flattened = [[0, 1], [2, 3], [4, 5]].reduce(

(accumulator, array) => {

debugger;

return accumulator.concat(array); //first step will be [ ].concat((0,1))

}, [ ]);



# How JS Really works? VIDEO 163

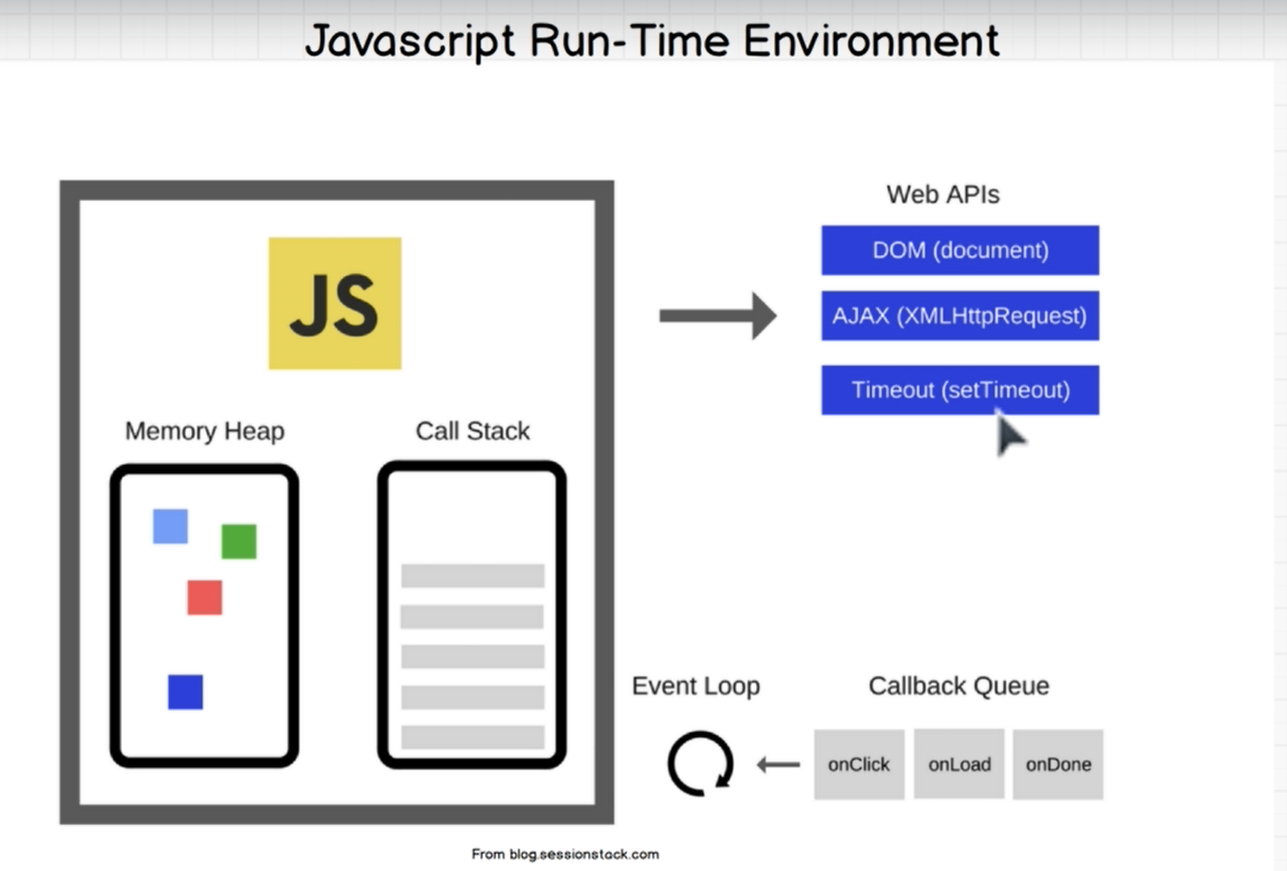
What is a program? A program does 2 simple things

-allocate memory (to have variables or a file in computer..)

-parse and execute (to read and to run code)

The V8 engine in chrome takes the code that we write and changes into machine executable instructions for the browser. The engine consist in two parts Memory heap (memory allocation happens here) and a call stack (to read and execute. If you don’t clean global variables it overflows the memory heap.

-JS is a single threaded language that can be non-blocking. Esto es que es single threaded pero hay cosas que no son parte de JS en si. Como Timeout el cual al JS ver esto pone un timer (incluso si le pones 0ms) de modo que ejecutara todo el otro código y pasado ese tiempo luego de ejecutar todo lo otro entonces ejecutara ese dentro del timeout haciendo asincrónicamente. VER VIDEO 163



We need more than just the javascript engine, we need what we call a javascript runtime environment. And Javascript runtime environment is again part of the browser, its included in the browsers. They have extra things, on top of the engine, they have something called the web APIs, callback queue and an event loop. And as you can see here, 'setTimeout' is part of the web API. Its not technically part of javascript. Its what the browsers give us to use, so we can do asynchronous program.

Javascript is a single threaded language that can be nonblocking. It has one call stack and it does one thing at a time. In order to not block the single thread, it can be asynchronous with callback functions and these callback functions gets run in the background, through the callback queue and then the event loop, to bring it back to the call stack.

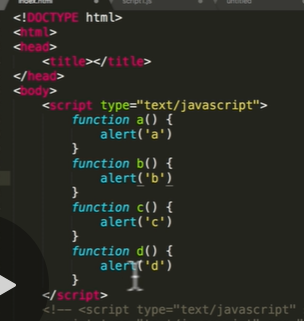
# What are JS Modules



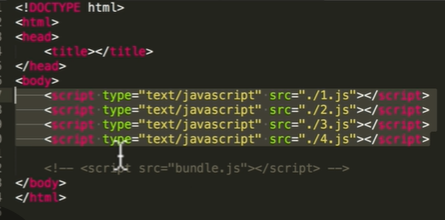
Mountain of modules.

## ***-Inline script:***

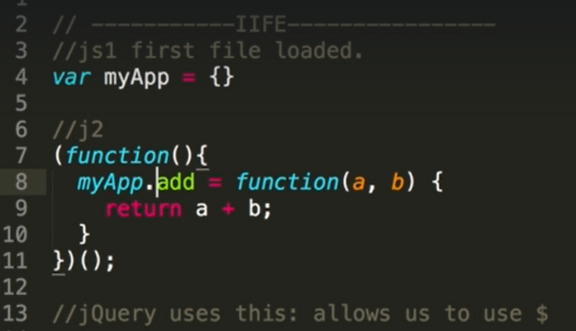
monton de scripts en el html en columna (problema **lack of code reusability-**tener que copier todo denuevo en la otra parte, **pollution of global namespaces**-hace posible tener el mismo nombre en 2 o mas variables)



**-script tags:** Bootstrap and Jquery everything outside html (problem we still need to copy and paste those script tags, also **lack of dependency resolution** you need to be sure scripts added in proper order, and also still pollution global namespaces problem)



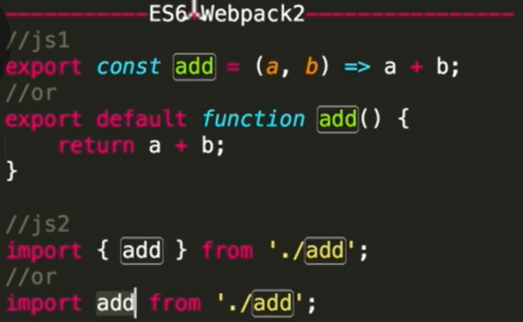
***-IIFE:*** Inmediately invoked function expression its to wrap a function in brackets and then last brackets () executes the function trying to make its own universe and avoid polluting namespaces. And order of the files stills important LACK OF DEPENDENCY RESOLUTION.



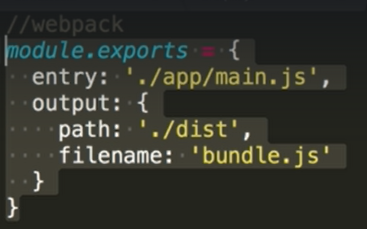
***-Browserify:*** browserify did use common.JS which allowed us to use a certain syntax. If I wanted to have on my first JavaScript file a function add, I can say module.dot exports and now in the second JavaScript file I could import that file. So Browserify is a module bundler it means it runs before you put the website online and all those JS files goes bundled so all our scripts will just be on one massive file for production were you don’t mind it is a chaos you still have it pretty in development mode.



***-Webpack + ES6*** : with ES6 what we have is for example in JS1 you can export a lot of functions in one file and then using destructuring take the function you want, or in you can also with export default just export one function with that file and import that function without the need of destructuring. By the other side webpack is a module bundler used to make ES6 available to all browsers.



That’s the way webpack works. Creating a bundle.js



More about modules : <https://hacks.mozilla.org/2018/03/es-modules-a-cartoon-deep-dive/>

MORE JAVASCRIPT

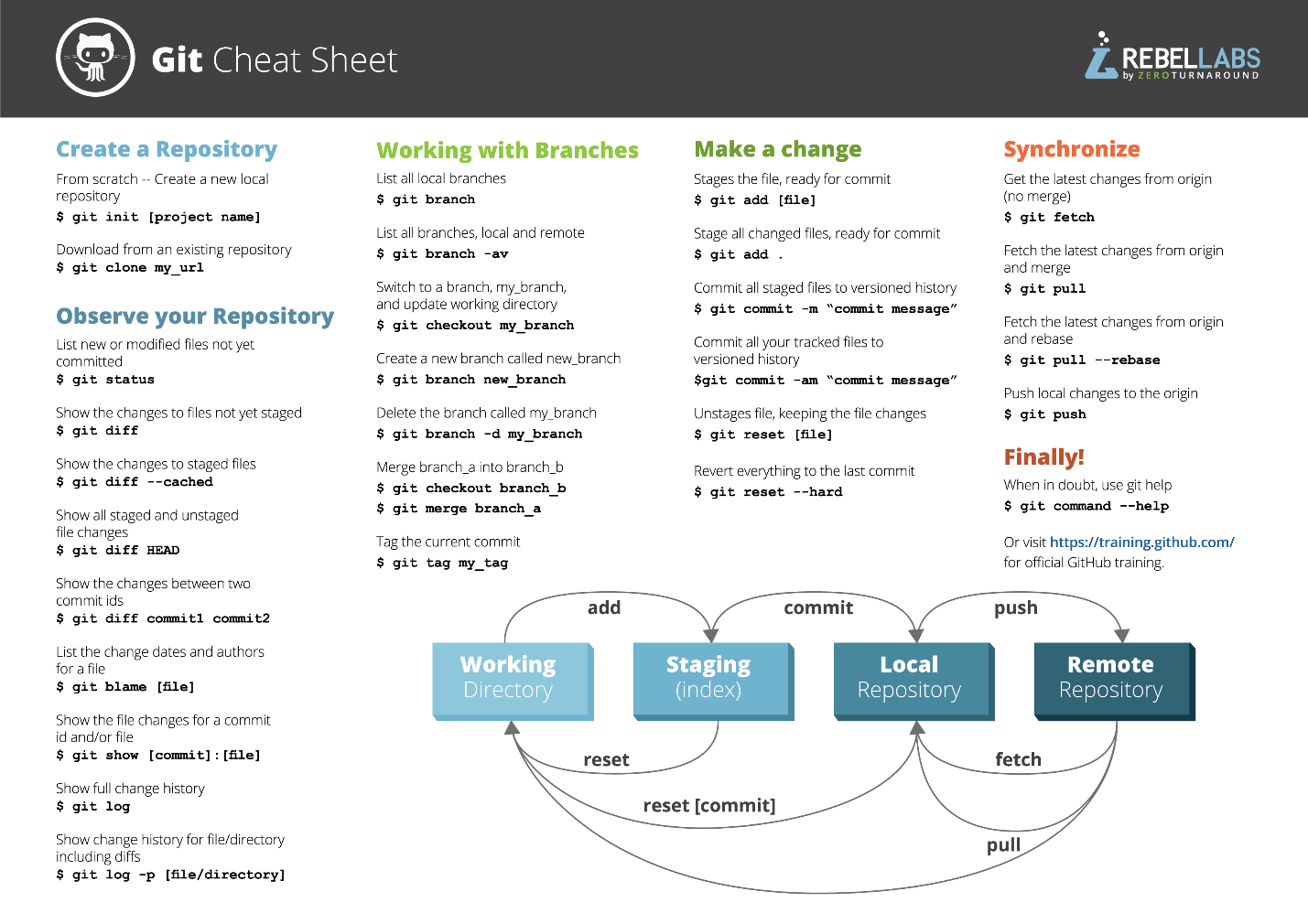
1. <https://github.com/getify/You-Dont-Know-JS>  
2. <http://javascript.info/>  
3. <http://dmitrysoshnikov.com/ecmascript/javascript-the-core-2nd-edition/>  
  
or... you can scroll all the way to the bottom of this course to **Extras: Bonus Part 2**to see some more Advanced JS that I teach :)

MAKE JS EXERCISES VIDEO 167

# GITHUB – source control



<https://www.jrebel.com/blog/git-cheat-sheet>



Upload project to github repository:

-Git clone repositoryAddress

-Git status

-Git add file or git add . (to add all changes local)

-Git commit -m “ add a comment “ (commit the changes)

-always to git pull to have lates changes of branch master before pushing something.

-Git push (push those changes to github.com)

-Git pull (retrieves all changes and shows it local)

Branchs (to avoid damaging production environment in master branch)

-Git Pull (Get the latest github changes)

-Git Branch (shows branch where you are working on)

-Git branch nameOfNewBranch (create a new branch)

-Git Checkout nameOfNewBranch (change branch to new created branch)

In this new branch you can do all before steps to upload all changes to github

You can review the changes comment and approve WITH MERGE PULL REQUEST in github and you can just delete the branch after this.

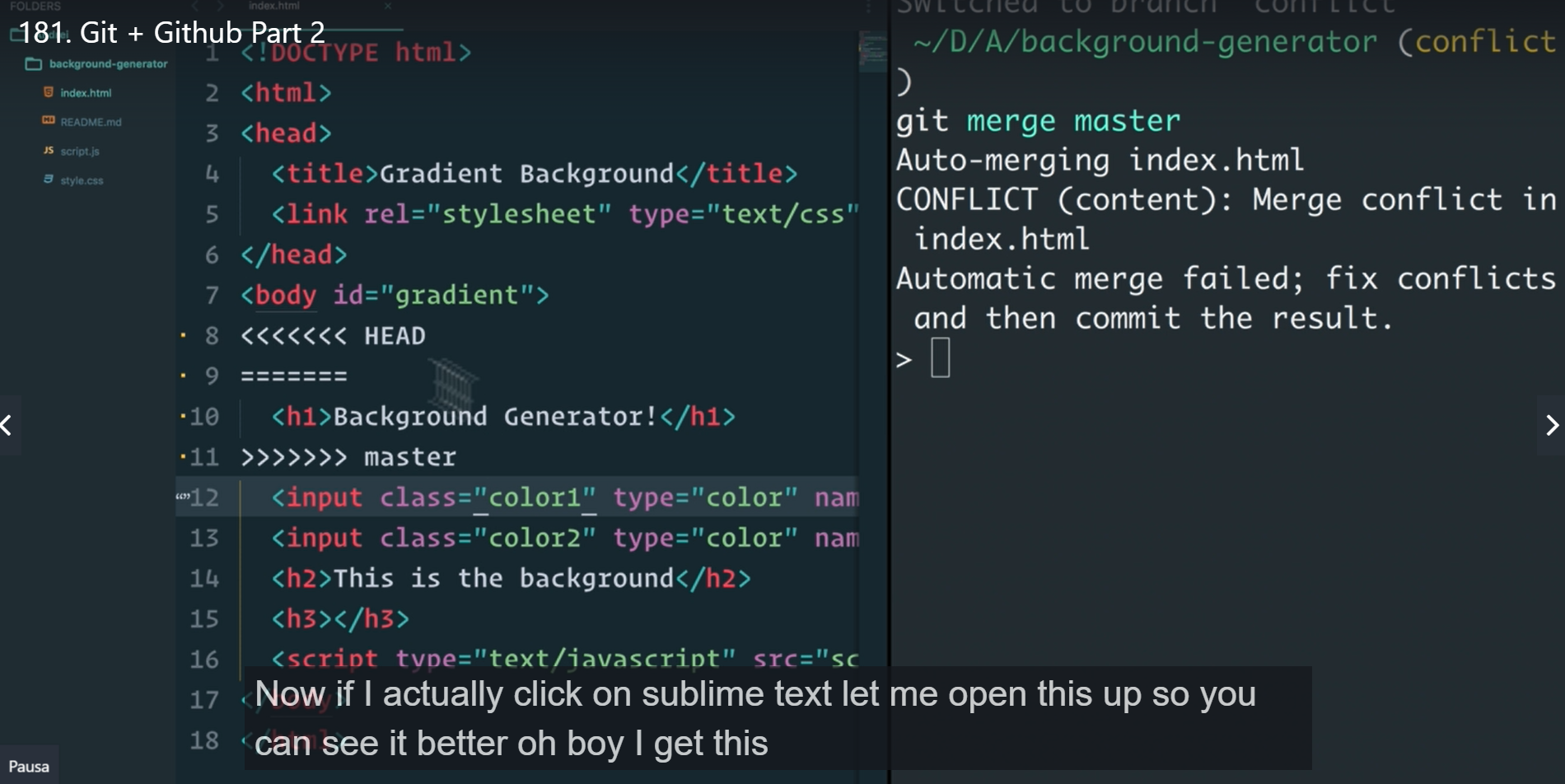
-Git merge master (to have all master branch latest changes in your new branch) if you did work on something in the new branch that is different in the master branch it will shows as follow:

<<<<<< HEAD

( between here start your thing in conflict in new branch )

======

( between Here will be the other thing in master that’s making conflict with new branch )

>>>>>>>

Then you ask other person to agree. And solve the conflict. Maybe just deleting the conflict syntax and keep going. Whatever both decide, then you can go as before with git add . -> git commit -m -> git push

-git push origin newCreatedBranchName (to puch new branches after conflict solved)

We can have a whole repository and own it copying to our github account by click (FORK) in the repository.

-Inside forked you can create a new branch git checkout -b newBranchName

-[@LuisPrz](https://github.com/LuisPrz)

Its mark down syntax than make possible clickable link inside github repository

-Git diff shows what are the changes made

-Git push origin master (push all changes to the original repository, not the forked one!)

-Git push origin whateverBranchNameIs (push all changes to whatever your own repository is)

-then you go to github.com and add a comment and click on create a pull request so when the admin see it they can merge your changes to the master branch. That pull request will be in the origin repository and if you see some problem with changes from other people you just need to delete the error marks.

<<<<<< HEAD

( between here start your thing in conflict in new branch )

======

( between Here will be the other thing in master that’s making conflict with new branch )

>>>>>>>

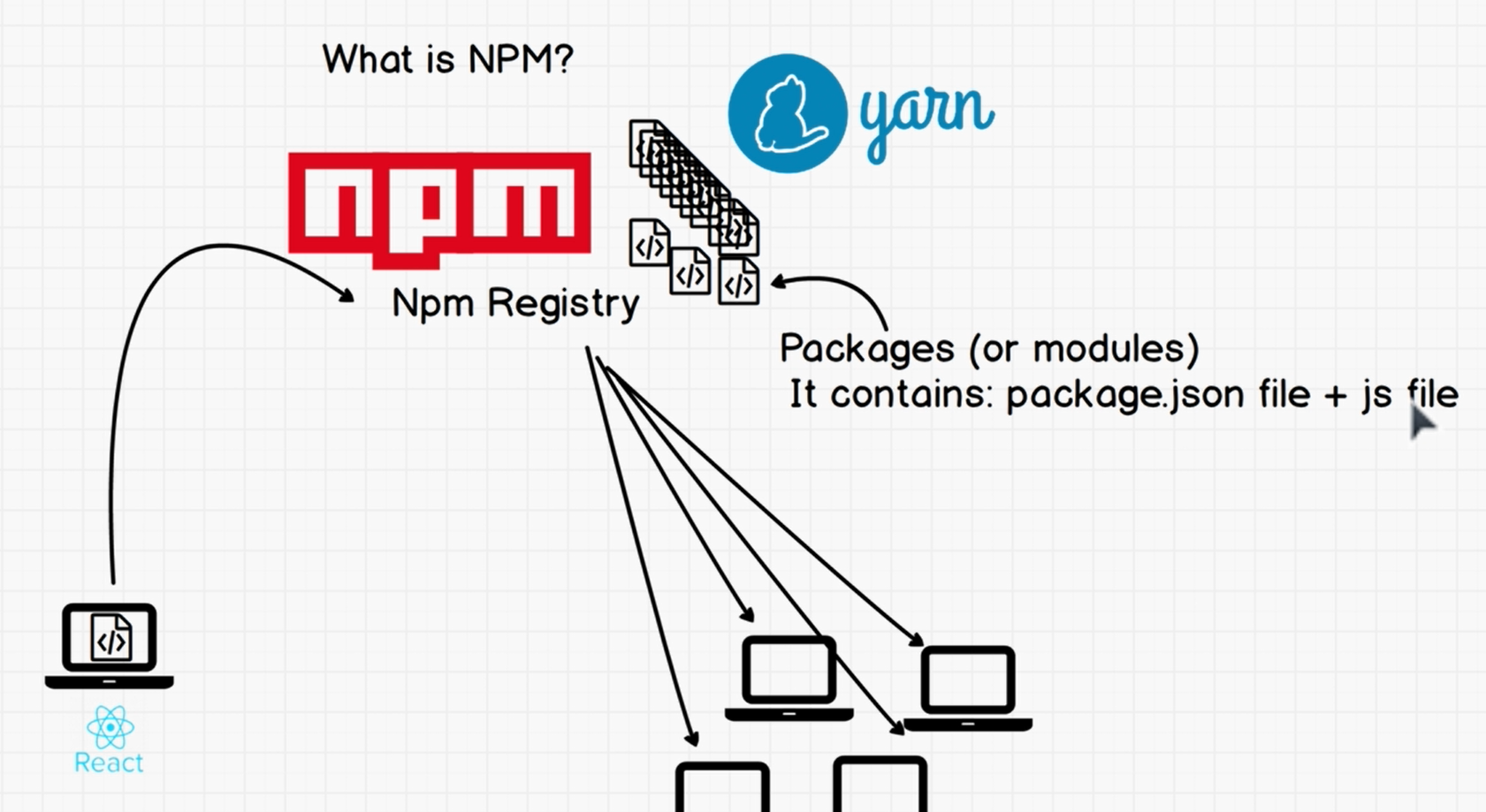
FORK UP TO DATE

git remote -v // shows your repository and the upstream remote repository

git remote add upstream // to add the repository where you pull the updates

git pull upstream master // from master branch it will sync the forks

# NPM (NODE PACKAGE MANAGER)



NPM is a registry or a collection of files modules or packages that you can search where those files or scripts are good at doing something (initially builded for Node.js but it is so good that can be use also for front end). Those packages or modules contains 2 things the package.json (metafile that describes the package and manage the files) file and the JS file. There are 3 types of packages front end, for terminal, and for backend(on node.js). NPM is uncluded in nodejs installation.

# 26. Node JS (AND NPM registry how works and commands)

It was created using the V8 Engine from chrome and it allows us to run JS outside the browser

Node -v //to check version / lastest version you need to download from web

npm install npm@latest -g //install the latest npm version globally (without -g //in is just local installation (inside project) )

React is actually a package created by facebook that you can download from npm registry

-NPM init is the command you need to use in order to create the package.json file in the directory wich allows you to use NPM In the project and grab whatever package we want from npm.

PACKAGE.JSON HAVE ALL PACKAGE INSTALLED FROM NPM SO THERE YOU CAN SEE WHAT YOU ARE USING

JSON is just a file format that looks like JS but it have double commas for properties example: *"name"*: "background-generator",

## --INSTALLING LIVE SERVER AND LODASH

Npm install packageName

npm install -g packageName //globally

## --LIVE SERVER

Npm install -g live server //with admin permission it’s a package inside NPM //that allows you to display changes live. You don’t need to refresh page.

## -- LODASH

Its like an extension of JS. It have a lot of functions that we can use with arrays. You want it to run locally inside project

npm install lodash

It also installs the node\_modules a folder that contains everything for lodash but that can be deleted when uploading to github or something because when NPM install packageName it sees that inside package.json dependency there is Lodash. And it installs automatically the node\_modules file locally.

To use Lodash you just need to apply the function without:

Import { without } from ‘lodash’;

console.log(without); //but as It uses the require syntax not all browsers supports it and that’s why you need browserify to convert it.

To use browserify we can install it globally

-npm install -g browserify /\* its just the module bundler where you give it a script.js file and it outputs another script.js bundled file that can be used in the browser!. although that was before. Now we can use webpack to use all those ES6 functions that were only for nodejs but now applies to JS!.\*/

-to create that bundle.js file we do:

browserify script.js > bundle.js

// the problem with all of this is that package of lodash is very big and also browserify work with all of this without matter that you use something or not it makes your files bigger. Also as now you have that dependency that you need to have this version of the lodash in order to work its extra difficulty layer!. But you can see it in package that json the dependency and the version ex 4.14.3 its called semver(semantic versioning) its ->> major(a whole new version that can break sites working of before version).medium(new features).smaller(bugs) changes

## -- DEPENDENCY INSIDE PACKAGE.JSON FILE

There are normal dependencies like the one that shows lodash and the semver version. But there are also dev dependency wich doesn’t display in github folder they are taken out automatically when the project is released.

-npm run scriptName // you can also run whatever script you want inside package.json file inside sccripts with this command

You can even do a build script to makes the script bundle to run automatically with every change.

<https://semver.npmjs.com/> for npm package version control and installations

# SCSS (It’s a file format for SAS, its kind of css but give extra features)

With SAS you can do kind variables using the dolar sign notation like $primary-color: #234523 so it will replace the color wherever you put $primary-color.

# REACT

React is just a library (inside npm registry) created by facebook to be able to create that massive app in a nice and predictable way avoiding inconvenients that happened with jquery, specially pyramid of doom. It allows multiple developers to work inside project at the same time because of it declarative way and enclosed universe for each function that makes it pure functions when you always know what is coming out from it.

-React works in a way that its like a bread machine, where you throw the ingredients and react do its magic underneath the hood and creates our website. The virtual DOM makes the Dom to change in predictable ways what makes it easy to scale and easy to manage. It can be use for mobile apps, VR, desktop apps, etc.

-Principles of react: thinking on components (instead of just a JS file, CSS file, HTML file) think on then as components like Lego blocks that you can combine to create your website.

Components:

-Atoms (icons and little small indivisible elements on your website such as images, tags)

-Molecules (combine atoms that have similar functionality such as the navigation bar)

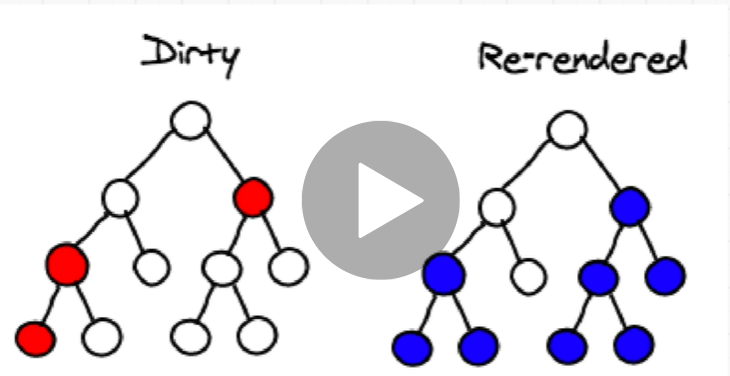
-Organisms (molecules that combines to a bigger scale functionality)

-Templates (the combination of organisms to give you an idea of how your app or website should look)!

-Pages (the page of your app!)

So we can reuse whatever of those components and that’s what makes React amazing

Another important concept of react is that data online flows in one way from top to bottom



On top there will be parents. On bottom there will be children. Only the children knows about the change. If red changes only blues cares.

-the top parent also would be Template, then organisms, then molecules, and the last bottom ones would be Atoms!

Before we were the paints. Now react comes with that robot called virtual DOM, where virtual DOM is just a copy of the DOM that’s in a JS object and the React underneath the hood just changes the view. And that’s way we can reuse react outside of browsers.

RECAP:

there is one way data flow, that is, everything flows from state to props and you can have multiple children but the only people that can change state is a container component. In our case, we only have one container component, but you can have multiple. And data just flows down one way and any time a child component changes, they just trigger an event on the parent. And you saw how awesome that was. We didn't think about how we should render things, we were doing things deterministically. That is: here's the state of the app, render it accordingly and React took care of it for us. We also learned that - React worries about painting and making the website

# React (Install React but with create-react-app tool)

create-react-app is a tool that allows us to create a react app with all out files already put into place!.

It’s a tool or package that you install globally from NPM and it creates a starter project that contains webpack, babel so we can use the latest javascript feature without the struggles of browserify and etc.

npm install -g create-react-app // to install the tool that installs react, webpack, babel and a lot of things

## -- Creating a React App before

create-react-app appName // to create a React app. It installs react and all dependency we need like node\_modules from lodash, and the npm package.json all of this included inside dependencies react, react-dom and react scripts

## -- Creating a React App now

npx create-react-app projectName // NPX is a package runner tool that comes with npm 5.2

## when the app is created inside this files are:

-the README.md // that displays in github. Markdown es un lenguaje de marcado ligero creado por John Gruber que trata de conseguir la máxima legibilidad

-package.json with scripts, dependencias, the scripts using react scripts that avoid the struggle we had with browserify for ES6

-package-lock.json //its automatically generated by package.json and it makes sure that the version numbers of dependencies are locked to make sure it will work no matter who you sent the app

-gitignore // to exclude whatever you don’t need to push into github for example

-Public //is the folder that contains index.html, manifiest.json(to download a shortcut to the website and have an icon), favicon.ico(the icon that displays in the page tab)

-SRC // the source folder, The one that you need to take care about! Where all react magic happens.

## In SRC:

-index.js //the main script file. Imports different scripts then grab an element of id root that is inside index.html!, and we are saying ReactDOM render app (the app component) with JSX syntax.

-You need to change IDE font to babel JS to detect all syntax

//SERVICE WORKING ITS to make app able to work offline (progressive apps) or downloable apps

//Entonces React consiste en componentes que están formados por funciones (aunque la ultima versión ya son clases pero en principio funcionan como funciones e incluso son intercambiables)

Que reciben props o propiedades de cualesquiera parámetros(seteados en el index.js y los llamas en el componente.js) y eso lo rendea

# JSX

Part of react is that it allows you to write this HTML like syntax (JSX) in your javascript. But doesn’t it break rule separation of concerns? And How can we add HTML to js FILE?

- Separation of concerns in react: its better to have functionality and styles per components so that each component is its own universe!. And you just add these components. It changes the paradigm but stills separating concerns by separating things in small universes!

- How can we add HTML to js FILE?. JSX lets us use a HTML like syntax but it isn’t actually HTML tags. React uses that JSX to create the virtual DOM, a Fake copy of the DOM that it builts based on what we give it. So it compares the DOM’s and updates the real DOM. That’s why react is so fast. Here we cant use class as like in HTML because it’s a JS reserved keyword so we need to use className!

The curly braces are a special syntax to let the JSX parser know that it needs to interpret the contents in between them as JavaScript instead of a string.

You need them when you want to use a JavaScript expression like a variable or a reference inside JSX. Because if you use the standard double quote syntax like so:

var css = { color: red }

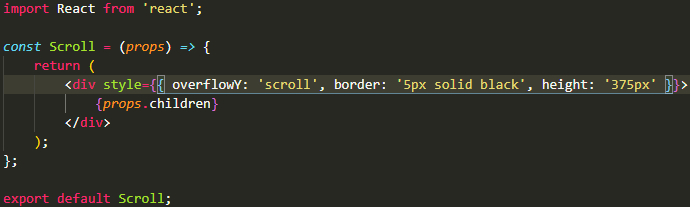
<h1 style="css">Hello world</h1>

JSX doesn't know you meant to use the variable css in the style attribute instead of the string. And by placing the curly braces around the variable css, you are telling the parser "take the contents of the variable css and put them here". (Technically its evaluating the content)

This process is generally referred to as "interpolation".

## **Adding styles to JSX**

We can add styles to JSX just like html tags, with style in the example we have to use double curly brackets. That would be 1 curly bracket(external) = JS expression, 2 curly bracket(internal) = an object that can have css styles and/or properties like overflow property.

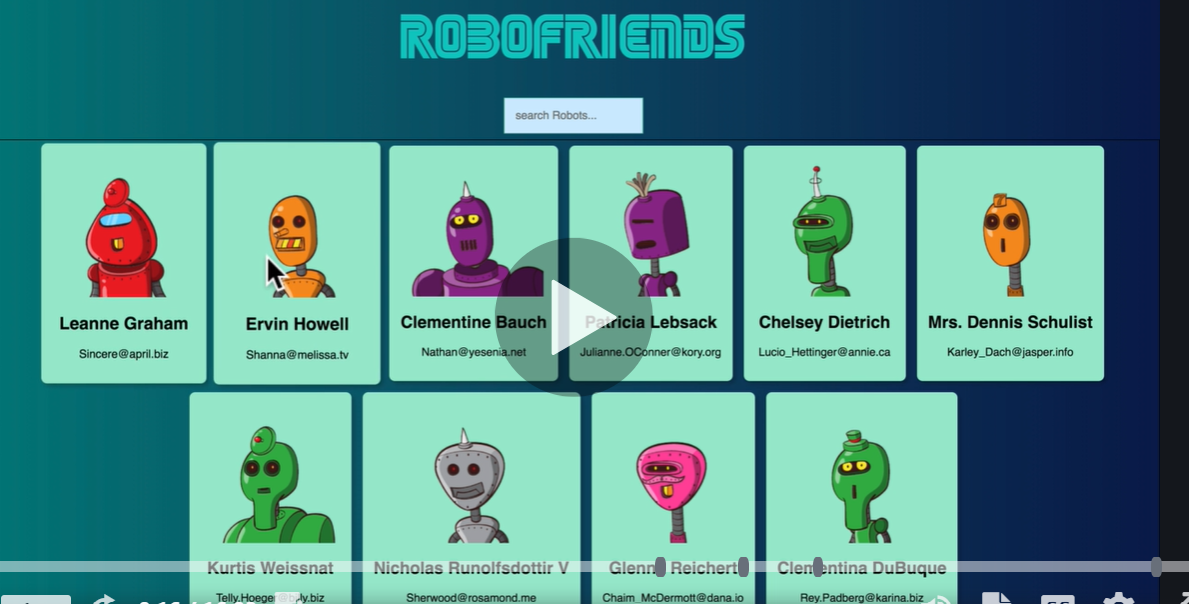


# TACHYONS

- npm install tachyons

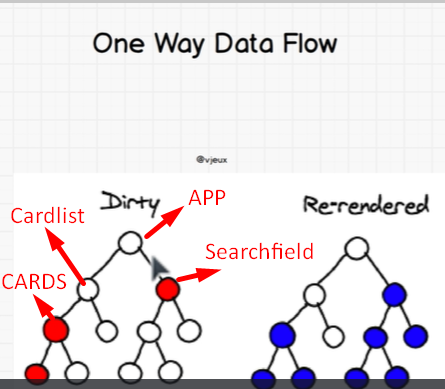
You install TACHYONS as npm install tachyons and import it inside index.js (import 'tachyons'; ) for faster css writing and templates

# ROBOFRIENDS APP



In that app the first you need Is to divide, the smaller thing you need to do is the card as you can see in diagram it will be the app that’s modifies the states then cardlist and then cards. Those childs of app communicates with props. But there is also children, another kind of props inside every component. So if you can build 1 card then you can copy all the components and add it to the page and make all other cards.

react robofriends app as in container and seach box and card list communicate using states that receive the app.

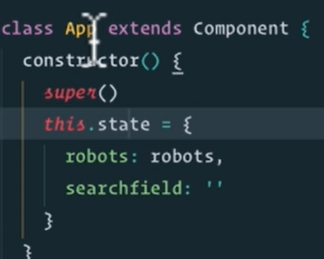


- npx create-react-app robofriends // create the react proyect app with all folders

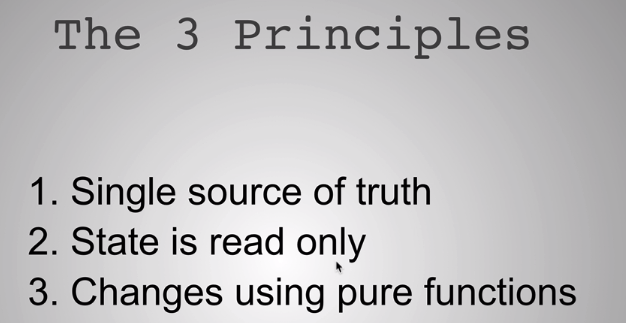
- npm install tachyons //install tachyons for css templates one row easy

Something you need to know: props never change and props is what pure functions or dumb components needs to communicate downside but to communicate upside they do it by states its some kind of memory that updates when there are changes. STATES is simply an object that describes the your application. A parent feeds state to a child component and child components receives as props that wont change. BUT in order for us to use and create the states inside the app we need to use class not functions anymore.

To create the state you create things that needs to communicate. And then inside app class you create the constructor and super constructor also as it extends from components



# React rules:



Principles:

1-big object that describes the app. The redux store

2- state cant be modified we can only create a new object with modifications

3- changes only use pure functions. Something gets in and we will have the same output always.

0- use arrow function to avoid event been taken from where it changes instead of App (the father)

1- event.target.value to see event changes

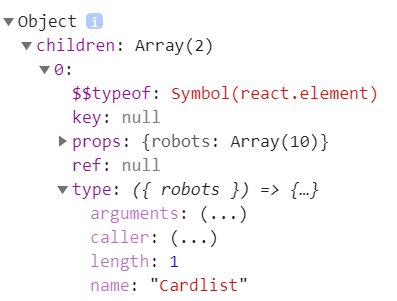
2- this.setState ({ searchField: event.target.value }) //para pasarle los valores al state y que este los pase al componente en este caso cardList

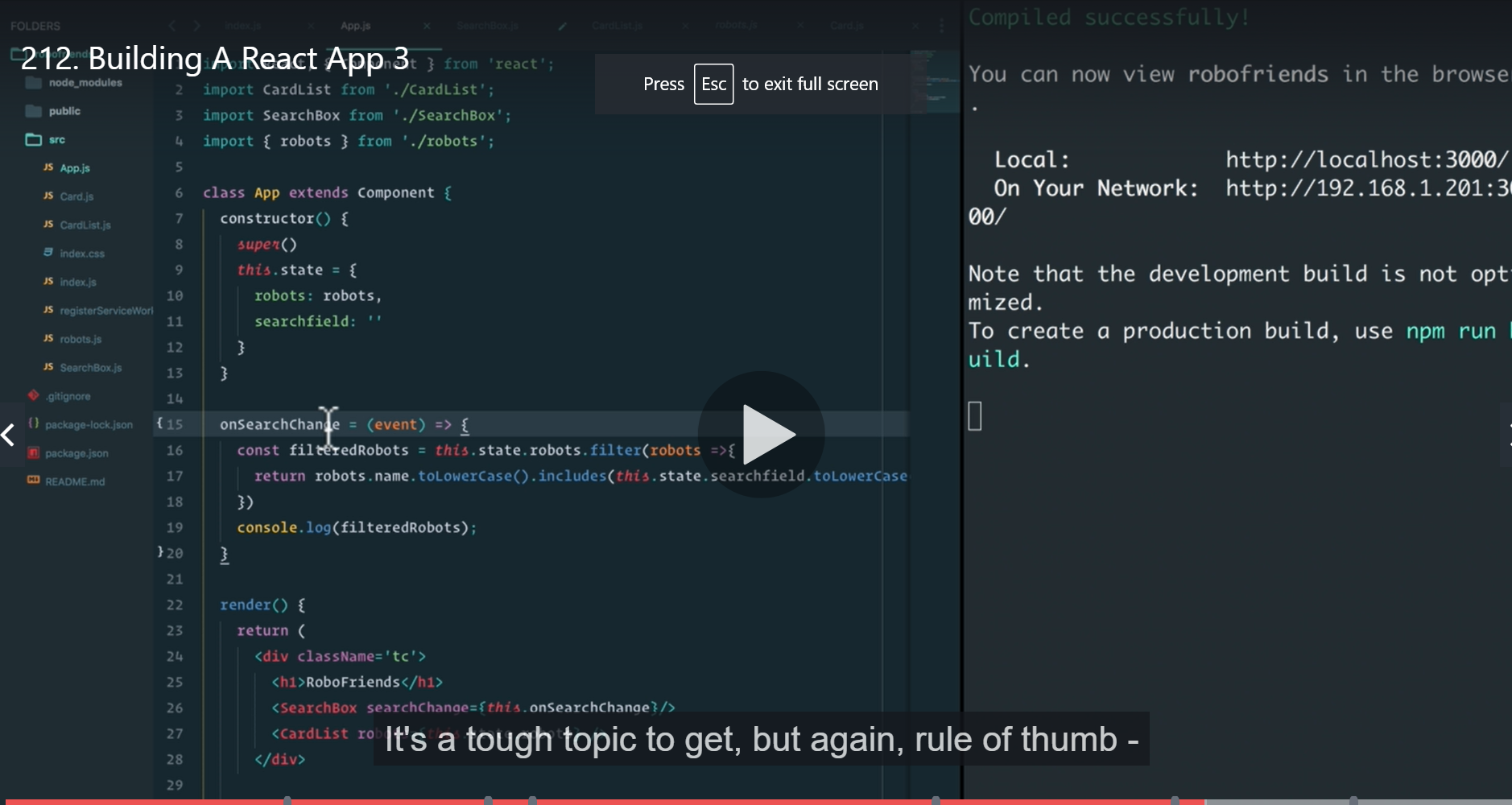
3- there is state, props, and also children

States is what defines the app, is what changes, while props is for dummies component or pure components it doesn’t change. Pure components talks to father triggering an event. And that makes them know to change the state and to send props.

Props.children can tell a component to render whatever is inside of this wraping component. The children is whatever is inside the wrapping component. But the component.js file needs to receive the props from the father that will be for example app.js.

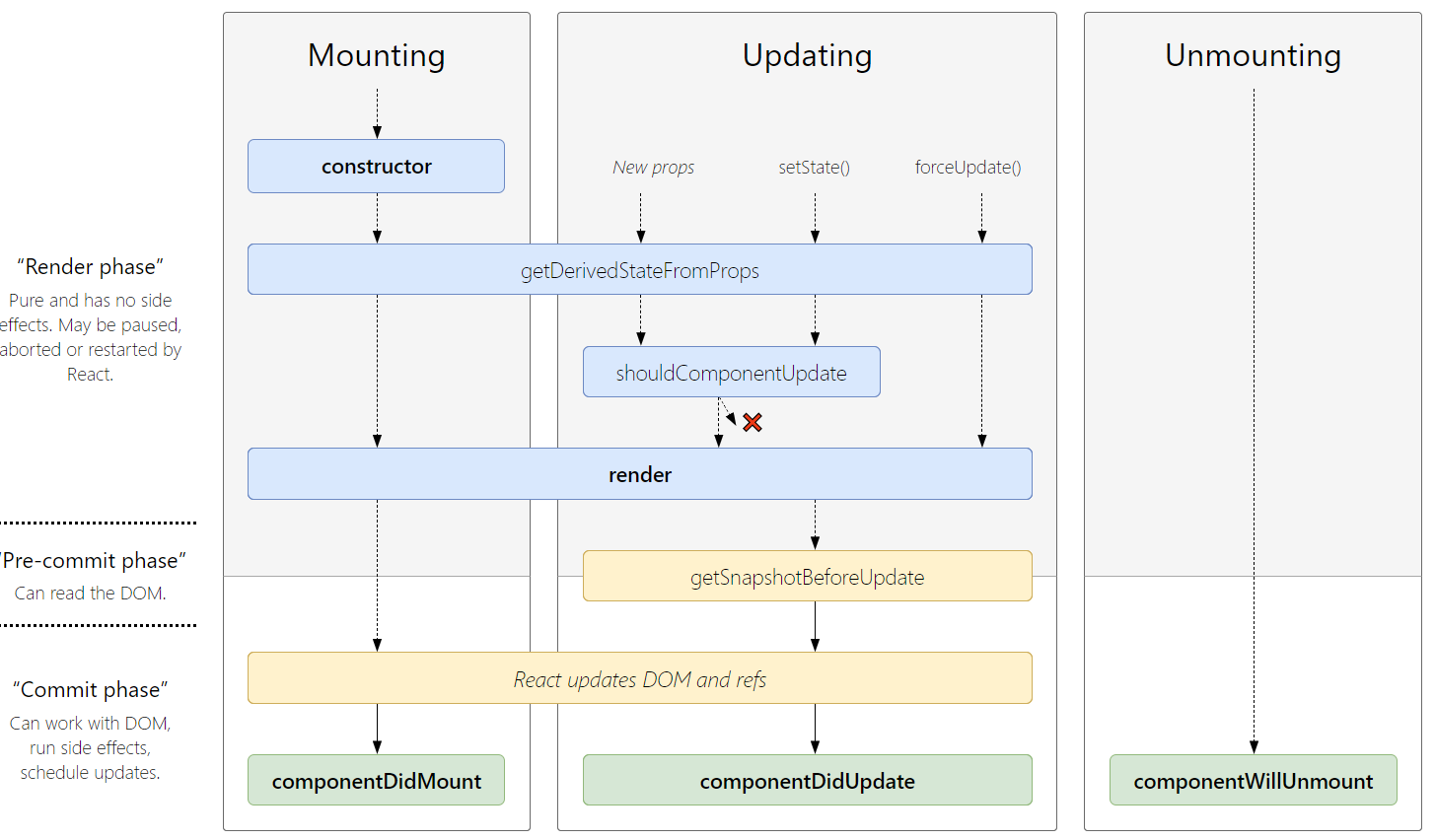
Every component have children, the children inside scroll component is cardlist:



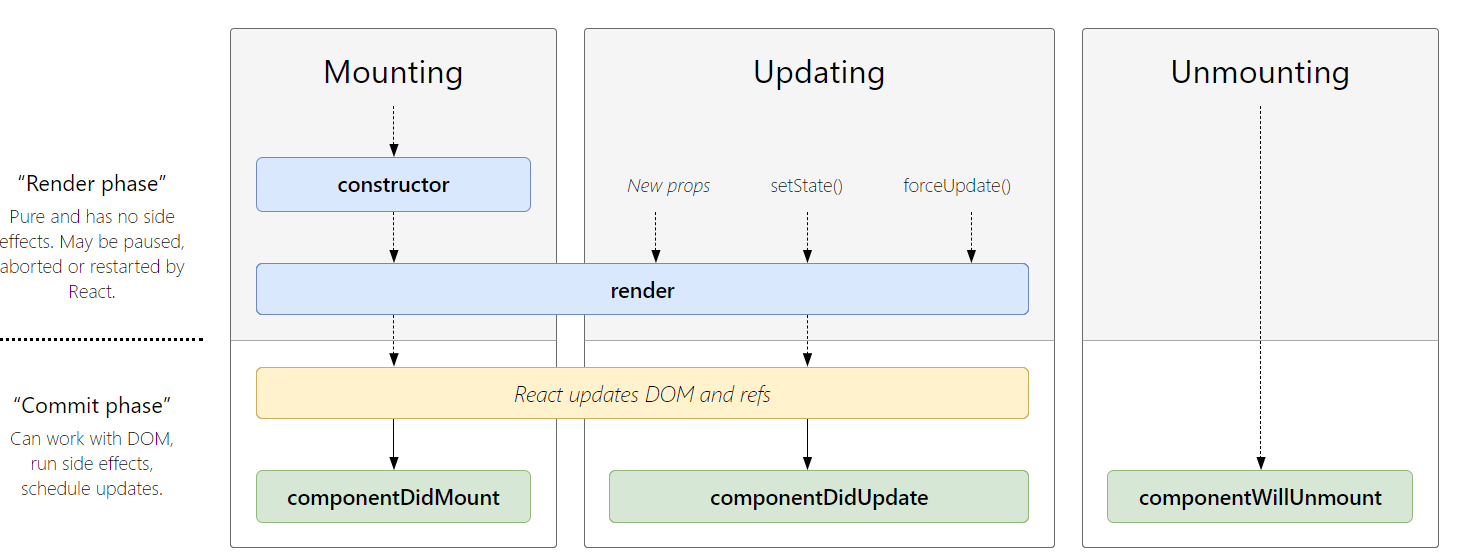


React lifecycle methods diagram

<https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>



With all methods.



Most used methods.

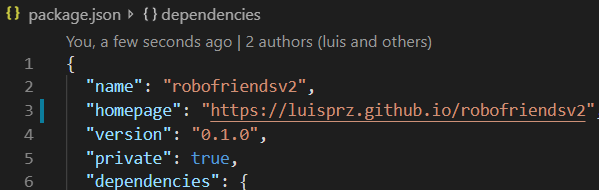
thods.

# Deploying project in github

You can do this using github pages: <https://create-react-app.dev/docs/deployment/#github-pages-https-pagesgithubcom>

Steps:

-add homepage to package.json ->



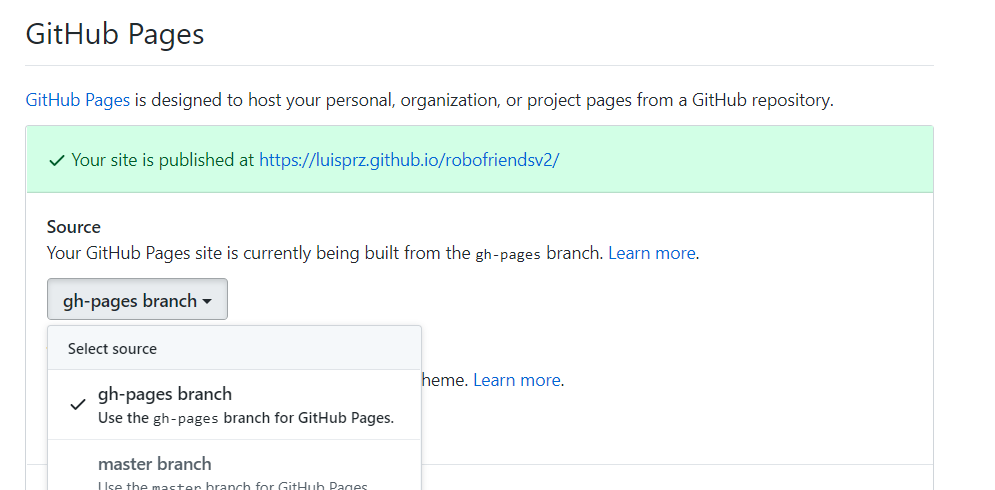
- install github pages -> npm install --save gh-pages

## - add scripts to package.json:

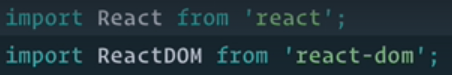
+ "predeploy": "npm run build", //create production version of project

+ "deploy": "gh-pages -d build", //use github pages and build folder

## - npm run deploy

- Go to settings inside repository and change to gh-pages branch, copy the link and add to description so everybody can see project running

# REDUX

Similar to react that needs to be installed and then to be connected to the dom it needs react DOM.

REDUX also needs to be connected to react because redux can be connected to whatever other packages.

That package needed for redux to connect to react

can be installed using: npm install react-redux.

That package works connecting only the containers. For example app.js to the redux store that’s the big JS object that manage all states and describes the state of our app. The container communicate with this redux store and viceversa. And the others dumb components wont know that redux exists!

REDUX actions and reducers



In the diagram remember that there's actions taken by a reducer and there is something called a store in redux which is the source of all truth. That is it's a big object, javascript object, that describes the state of our app so that react can render. it make changes and display it to the user.

So we've created an action we've created and reduce a reducer and we also need a store. Actions are things like to change the search field and just object that we return. Reducer is a big function that reads the action and spits out what we call state. Reducers receive the actions and change the states and the store catch all those states and manage it by us to make changes!.

To have the store we just import it: import { createStore } from ‘redux’;

And then you will have const store = createStore(rootReducer) so you can see states from reducers

npm install react-redux comes with easy api consisting on 2 things: provider and connect. Those you import with import { provider, connect } from ‘react-redux’

Provider wraps the container and will take care of pass down the store to all the components down the components tree

Connect is the part of the api that makes components smart and aware that redux exists. And this smart components will be containers. Connect is also a high order function that is that is will return another function. For ex it will return App after. That’s why we can put it like this

export default connect(mapStateToProps, mapDispatchToProps)(App);

connect accepts 2 parameters that can be named whatever but the standard is mapStateToProps, mapDispatchToProps in order to subscribe this smart component to any state changes from the store!

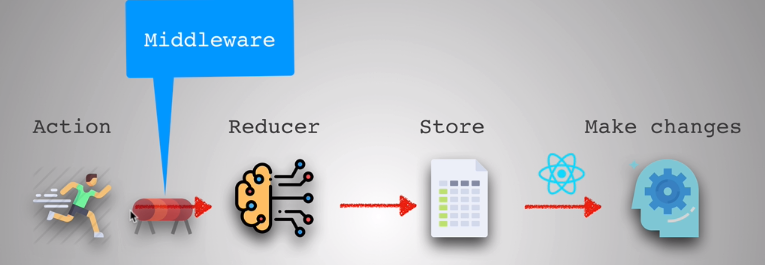
\*\*dispatch is what triggers the action, where an action is just a created object like set searchField and it will send the action to the reducer!

mapStateToProps is telling me what state, what piece state I need to listen to, and send down as props and

mapDispatchToProps says hey tell me what props I should listen to that are actions that need to get dispatched.

So connect is going to run the first part of the function (mapStateToProps) and say Okay I'm listening to this part of the state and I'm interested in these actions (mapDispatchToProps) and then it's going to give those props to the app.

# middleware

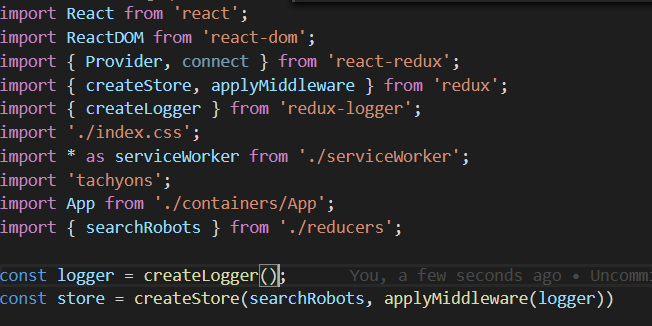


The middleware Is quite simple and works similarly to how middleware in express.js works. It simply listens for actions and it's a tunnel that actions go through and depending on what the middleware is, it can modify the action or trigger another action. So they're like triggers that actions go through and something happens within this blue box, before it hits the reducer. So why might that be useful for us.

we can install something that is a middleware and it's called redux logger:

npm install redux-logger //its to debug the app showing logs in the console

with applyMiddleware you can use the logger middleware with the store



# REDUX asincronous actions

For this we need to install thunk:

Npm install redux-thunk

Well redux-thunk is a middleware that waits and sees if any actions return a function instead of an object. What does that mean? If we go to actions we see that this is returning an object, but right here we're not returning an object

export const requestRobots = () => (dispatch) => { //high order function now to use or provide the dispatch function to use in everything //we calll request robots action

dispatch({ type: REQUEST\_ROBOTS\_PENDING}); //no payload

fetch('https://jsonplaceholder.typicode.com/users') //then we fetch or receive the users

.then(response => response.json())

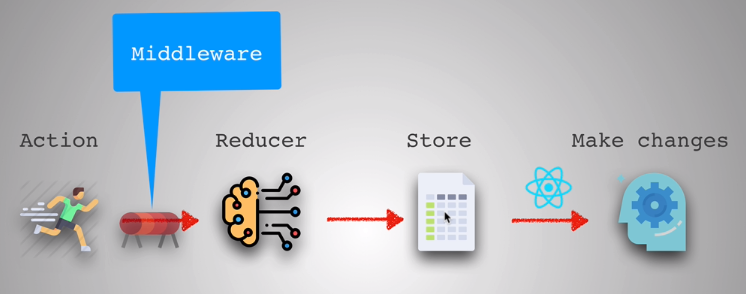
.then(data => dispatch({ type: REQUEST\_ROBOTS\_SUCCESS, payload: data })) //we have success with paload of users

.catch(error => dispatch({ type: REQUEST\_ROBOTS\_FAILED, payload: error })) // or a failed with the payload of the error

}

Remember this requestRobots inside actions.js, how it has this two functions. The requestRobot, right now redux, out of the box, wouldn't understand this, because we're not returning an object, as it expects for an action. We're returning a function and this function, isn't going to mean anything to it. By adding redux-thunk middleware, we're now listening to actions. And any time the requestRobots action gets triggered it's going to return a function and trigger redux-thunk and redux-thunk is going to say - oh this is a function I'm going to give you, here is the dispatch so you can actually call some actions, and we can finally run our actions like this.

## Whole explanation for redux with thunk and logger middlewares



So if we go back to our diagram we've created a system where the actions are being triggered. They go through any middleware. If it's a search term change, it's just going to go straight to the reducer, run through a nice function, update the store and make changes to our view. If we request robots it's going to notice that it's a function (it knows because instead of {} that is what redux knows, it will see () a function that is what thunks middleware cares about!. it's going to go into the middleware, and redux-thunk is gonna say. All right first just dispatch pending to the reducer, and I'll let you know when I'm done with the promise and I'll let you know if I got any robots. And when it returns it's going to dispatch the success, go through the reducer, update the store and make changes. Phew! That was a lot. But if you got this far and you understand it you've pretty much gotten redux. That's it. That's the whole redux library. It's very very simple.

# REACT UTILITIES

## **REACT ROUTER**

We first have 'REACT ROUTER' and that is for routing. So changing pages and changing routes in our url, it's pretty much the standard, when it comes to any React and Redux project. <https://reacttraining.com/react-router/>

## RAMDA

If you wanna really keep the functional programming ideology, a really really good library. <https://ramdajs.com/>

## **LODASH**

also the most popular 'Lo dash', which again gives you extra tools that you can use with javascript.

## STYLING JS

-Glamorous.rocks

-styled-components.com

-css-modules

## GATSBYJS.ORG

gatsby dot js(gatsbyjs)'. And, this is really really good for static web sites in React. So that is simple text based websites. Actually, if you go to the React documentation, their documentation is actually served with 'gatsby dot js(gatsbyjs)', because well they're just simple text based pages. That works really really well.

## Zeit.co for NEXT.JS

extremely popular for server side rendered apps. And that's something that we'll talk about when we get to the server side section. And we'll actually build a simple 'Next.js' app. But very popular and I highly recommend if you're building anything server side.

## **MATERIAL-UI.com**

React is really really good because we're able to build reusable components. So, if I build a really cool calendar component, I can put it online and other people can just grab that component, and use it in their projects as well. So some of the bigger ones are 'material', 'material ui' components, that again you can just use.

## SEMANTIC-UI

And there's also the 'semantic ui' library that has a ton of React components that you can use.

## RESELECT

And then finally there's tools that make Redux better, more efficient. You have things like, 'reselect', that you'll see in most Redux projects that help with something called selectors, and improve the performance of your apps.

## THUNKS

What we already know

## REDUX-SAGA

You also have things like 'Redux-Saga' which, which handles asynchronous actions in Redux. So, think of it as redux-thunk but supercharged. So as things get more and more complex, you have more and more asynchronous actions, you might wanna start looking at things such as 'Redux Saga'.

## IMMUTABLE JS

And then, by facebook, you also have something like 'IMMUTABLE' js, that you can use as a library to make sure that your state remains immutable and enforce that among your developer teams.

Becareful, Because adding any library to our project, adds extra weight, adds extra javascript that we need to deliver to the client.