JavaScript para Desarrollo FrontEnd

Contenidos

1. Introducción

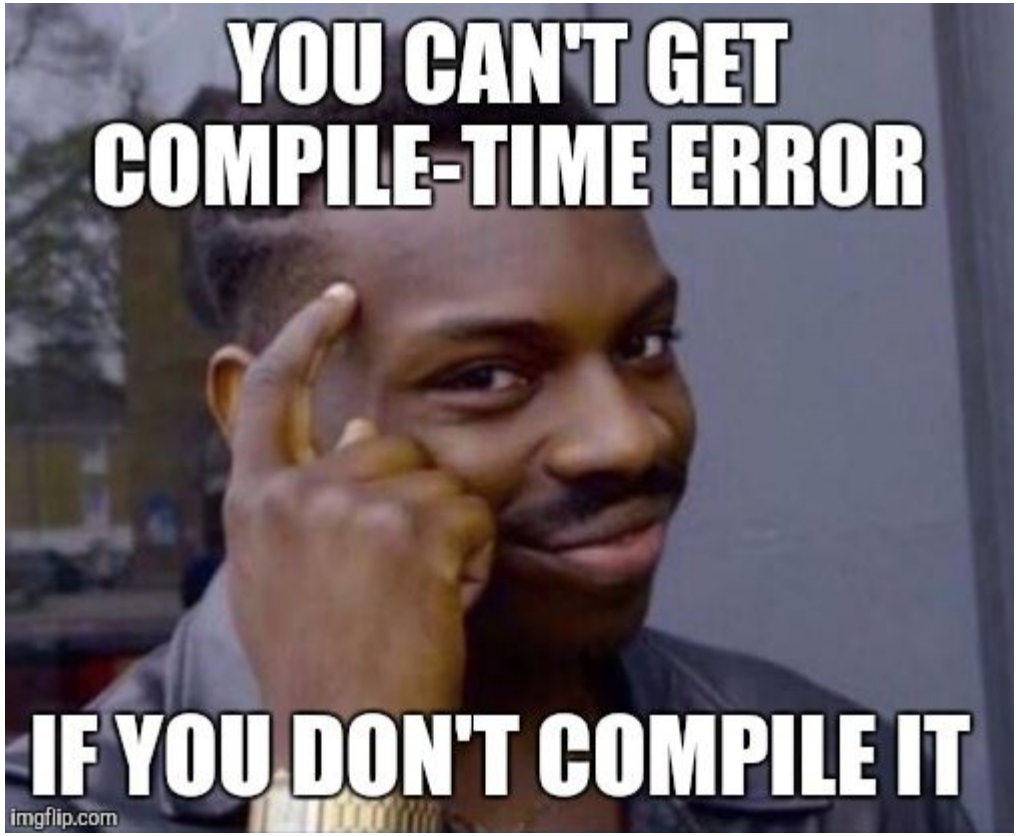
1. Introducción

Junto con HTML y CSS, JavaScript está entre las principales tecnologías para el desarrollo web.

Entre sus características están:

- Alto nivel
- Dinámico
- Débilmente tipado
- Interpretado en el cliente

1. Introducción



ECMAScript
Es el estándar en que se basan los motores de JS

Edition	Date published	Changes from prior edition	Editor
1	June 1997	First edition	Guy L. Steele Jr.
2	June 1998	Editorial changes to keep the specification fully aligned with ISO/IEC 16262 international standard	Mike Cowlishaw
3	December 1999	Added regular expressions, better string handling, new control statements, try/catch exception handling, tighter definition of errors, formatting for numeric output and other enhancements	Mike Cowlishaw
4	Abandoned	Fourth Edition was abandoned, due to political differences concerning language complexity. Many features proposed for the Fourth Edition have been completely dropped; some are proposed for ECMAScript Harmony.	
5	December 2009	Adds "strict mode," a subset intended to provide more thorough error checking and avoid error-prone constructs. Clarifies many ambiguities in the 3rd edition specification, and accommodates behaviour of real-world implementations that differed consistently from that specification. Adds some new features, such as getters and setters, library support for JSON, and more complete reflection on object properties.[11]	Pratap Lakshman, Allen Wirfs-Brock
5.1	June 2011	This edition 5.1 of the ECMAScript Standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.	Pratap Lakshman, Allen Wirfs-Brock
6	June 2015[12]	The Sixth Edition, initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015 (ES2015)[12] adds significant new syntax for writing complex applications, including classes and modules, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and `for/of` loops, Python-style generators and generator expressions, arrow functions, binary data, typed arrays, collections (maps, sets and weak maps), promises, number and math enhancements, reflection, and proxies (metaprogramming for virtual objects and wrappers). As the first "ECMAScript Harmony" specification, it is also known as "ES6 Harmony."	Allen Wirfs-Brock
7	June 2016[13]	The Seventh Edition, also known as ECMAScript 2016,[13] intended to continue the themes of language reform, code isolation, control of effects and library/tool enabling from ES2015, includes two new features: the exponentiation operator (**) and Array.prototype.includes.	Brian Terlson

2. Funciones y variables

2.1 Funciones

- En Javascript las funciones son consideradas objetos de primera clase. Pueden tener propiedades, métodos y se pueden pasar como parámetros a otras funciones.
- Se pueden declarar de las siguientes maneras:

```
var f = function(){alert('hi')}

function f(){alert('hi')}
```

2.2 Variables

Declaradas con

- **VAR**: Alcance global
- **LET**: Alcance de bloque
- **CONS**: Constantes

```
function varTest() {
  var x = 31;
  if (true) {
    var x = 71;  //misma variable!
    console.log(x);  // 71
  }
  console.log(x);  // 71
}

function letTest() {
  let x = 31;
  if (true) {
    let x = 71;  //variable diferente
    console.log(x);  // 71
  }
  console.log(x);  // 31
}
```

3. ECMA6

Una de las principales adicciones de la versión ECMA6 corresponde a las **funciones flecha**

Sintaxis

```
(param1, param2, paramN) => {declaraciones}
```

Es equivalente a

```
(function (param1, param2, paramN) {

    declaraciones;

});
```

Una de las principales adicciones de la versión ECMA6 corresponde a las **funciones flecha**

Sintaxis

```
(param1, param2, paramN) => expresion
```

Es equivalente a

```
(function (param1, param2, paramN) {

    return expresion;

});
```

4. Programación funcional

Programación funcional

*“**Functional programming** (often abbreviated FP) is the process of building software by composing pure functions, avoiding shared state, mutable data, and side-effects. Functional programming is **declarative** rather than **imperative**, and application state flows through pure functions. Contrast with object oriented programming, where application state is usually shared and colocated with methods in objects.”*

Javascript imperativo

```
const doubleMap = function(numbers) {
 const doubled = [];
 for (let i = 0; i < numbers.length; i++) {
   doubled.push(numbers[i] * 2);
 }
 return doubled;
};


console.log(doubleMap([2, 3, 4])); // [4, 6, 8]
```

Javascript declarativo

```
const doubleMap = numbers => numbers.map(n => n * 2);

console.log(doubleMap([2, 3, 4])); // [4, 6, 8]
```

Funciones relevantes de arreglos **map()**

The **`map()`** method creates a new array with the results of calling a provided function on every element in the calling array.

```
var numbers = [1, 5, 10, 15];
var doubles = numbers.map(function(x) {
   return x * 2;
});
// doubles is now [2, 10, 20, 30]
// numbers is still [1, 5, 10, 15]
```

Funciones relevantes de arreglos **reduce()**

El método **reduce()** aplica una función a un acumulador y a cada valor de un array (de izquierda a derecha) para reducirlo a un único valor.

```
var total = [0, 1, 2, 3].reduce(function(sum, value)
{
  return sum + value;
}, 0);
// total is 6
```

Funciones relevantes de arreglos **filter()**

El método **filter()** crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

```
var words = ["spray", "limit", "elite",
"exuberant", "destruction", "present"];

var longWords =
words.filter(function(word){
  return word.length > 6;
})

// Filtered array longWords is
["exuberant", "destruction", "present"]
```

5. JavaScript Asíncrono

Promesas

El objeto **Promise** (Promesa) es usado para computaciones asíncronas. Una promesa representa un valor que puede estar disponible ahora, en el futuro, o nunca.

```
let myFirstPromise = new Promise((resolve, reject) => {
  // We call resolve(...) when what we were doing made async
successful, and reject(...) when it failed.
  // In this example, we use setTimeout(...) to simulate async code.
  // In reality, you will probably be using something like XHR or an
HTML5 API.
  setTimeout(function(){
    resolve("Success!"); // Yay! Everything went well!
  }, 250);
});

myFirstPromise.then((successMessage) => {
  // successMessage is whatever we passed in the resolve(...)
function above.
  // It doesn't have to be a string, but if it is only a succeed
message, it probably will be.
  console.log("Yay! " + successMessage);
});
```

Callbacks

Callbacks are just the name of a **convention** for using JavaScript functions. There isn't a special thing called a 'callback' in the JavaScript language, it's just a convention. Instead of immediately returning some result like most functions, **functions that use callbacks take some time to produce a result**.

The word 'asynchronous', aka 'async' just means 'takes some time' or 'happens in the future, not right now'. Usually callbacks are only used when doing I/O, e.g. **downloading things, reading files, talking to databases,** etc.

Callbacks

Llamada a función normal:

```
var result = multiplyTwoNumbers(5, 10)
console.log(result)
// 50 gets printed out
```

Llamada a función asíncrona

```
var photo =
downloadPhoto('http://coolcats.com/cat.gif')
// photo is 'undefined'!
```

Callbacks

```
downloadPhoto('http://coolcats.com/cat.gif',
handlePhoto);

function handlePhoto (error, photo) {
  if (error) console.error('Download error!',
error)
  else console.log('Download finished', photo)
}

console.log('Download started');
```

Control de flujo en callbacks

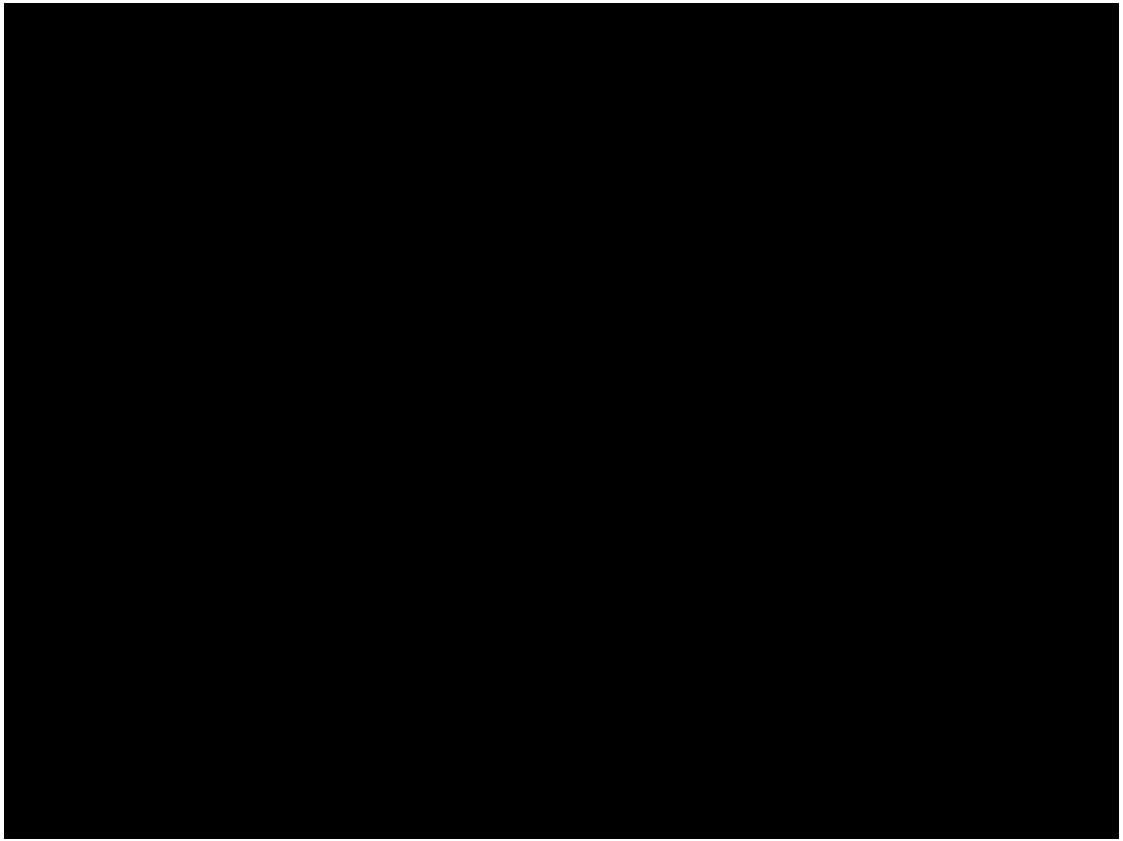
```
/**
 * Show a list of songs for the artist with
 * the given name.
 */
function showSongs (artistName) {
     var artist = artists.getByName(artistName);
     var albums = albums.getByArtist(artist);
     var songs = songs.getByAlbum(albums);
     songView.render(songs);
}
```

Control de flujo en callbacks

```
/**
 * Show a list of songs for the artist with
 * the given name and execute the callback when
 * complete.
 */
function showSongs (artistName, callback)  {
    // And we're not even handling errors!
    artists.getByName(name, function (artist) {
        albums.getByArtist(artist, function (albums) {
            songs.getByAlbum(albums, function (songs) {
                songView.render(songs);
                return callback();
            });
        });
    });
}
```

6. Automatización y empaquetamiento

Gestión de dependencias

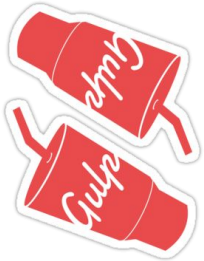


Automatización y empaquetamiento

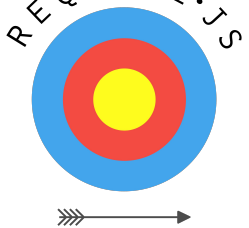
En el empaquetamiento de aplicaciones JS se han utilizado herramientas como **Grunt**, **Gulp**. La herramienta que utilizaremos en los proyectos es **Webpack**.

Otras herramientas de automatización de tareas

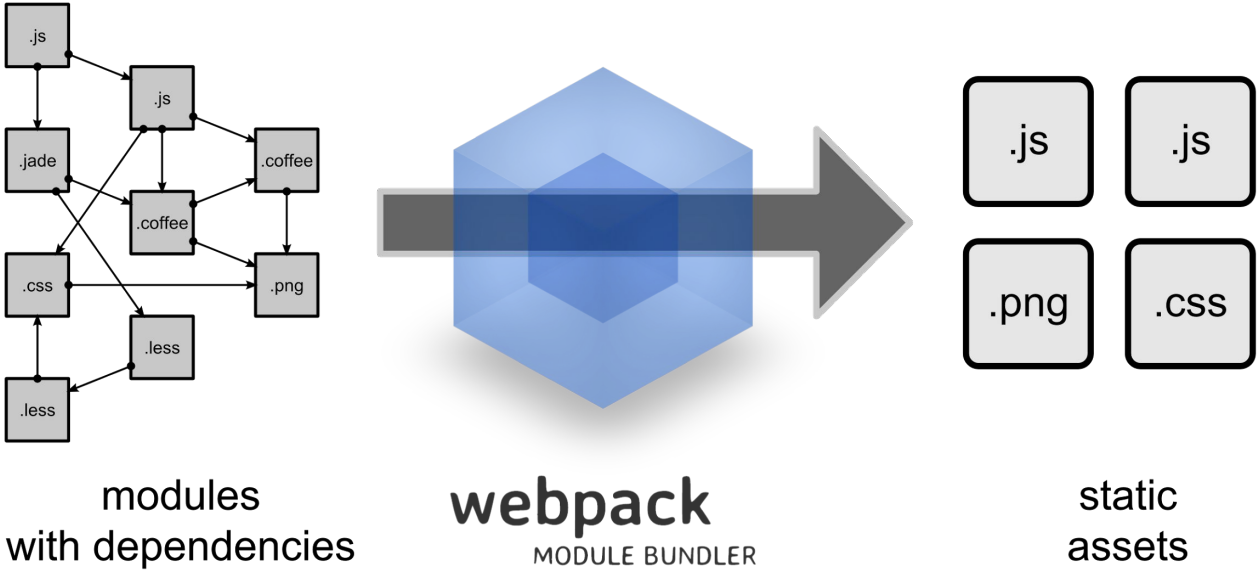
Automatización de tareas



Gestión de Módulos JS



Webpack


.js
.js
.jade
.coffee
.coffee
.css
.png
.less
.less
modules
with dependencies
webpack
MODULE BUNDLER
.js
.js
.png
.css
static
assets

7. Frameworks y librerías de JS

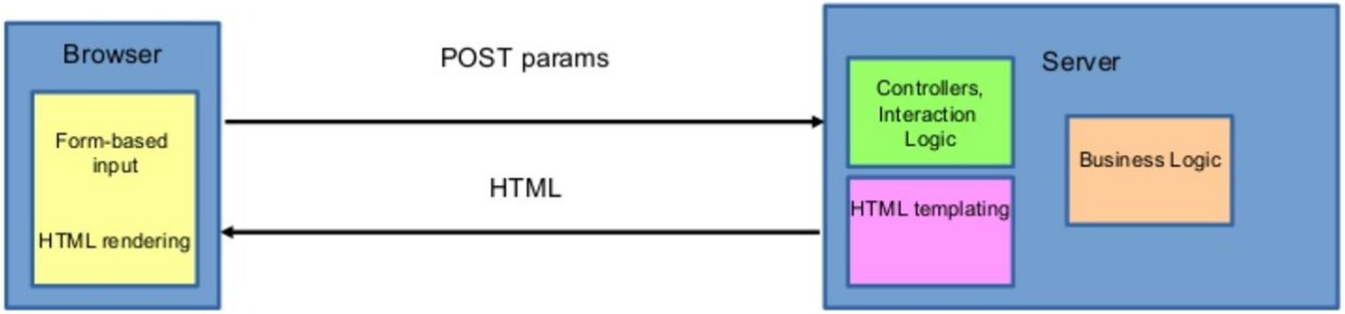
7.1 Aplicaciones Single Page SPA

A diferencia de aplicaciones tradicionales de páginas de servidor:

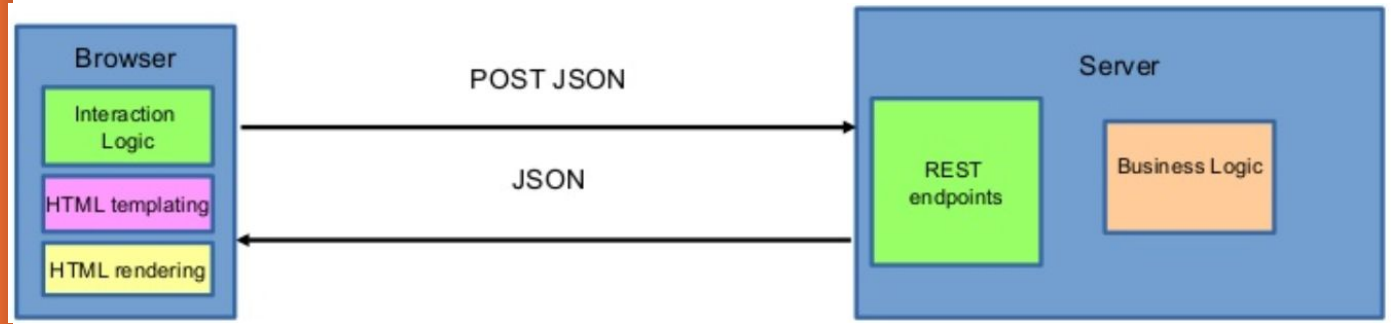
- Consisten en páginas html **estáticas** en el servidor
- Se conectan a datos por servicios REST y Websockets
- Se ejecutan de manera **dinámica** en el cliente

Son más poderosas e interactivas que las aplicaciones web tradicionales.

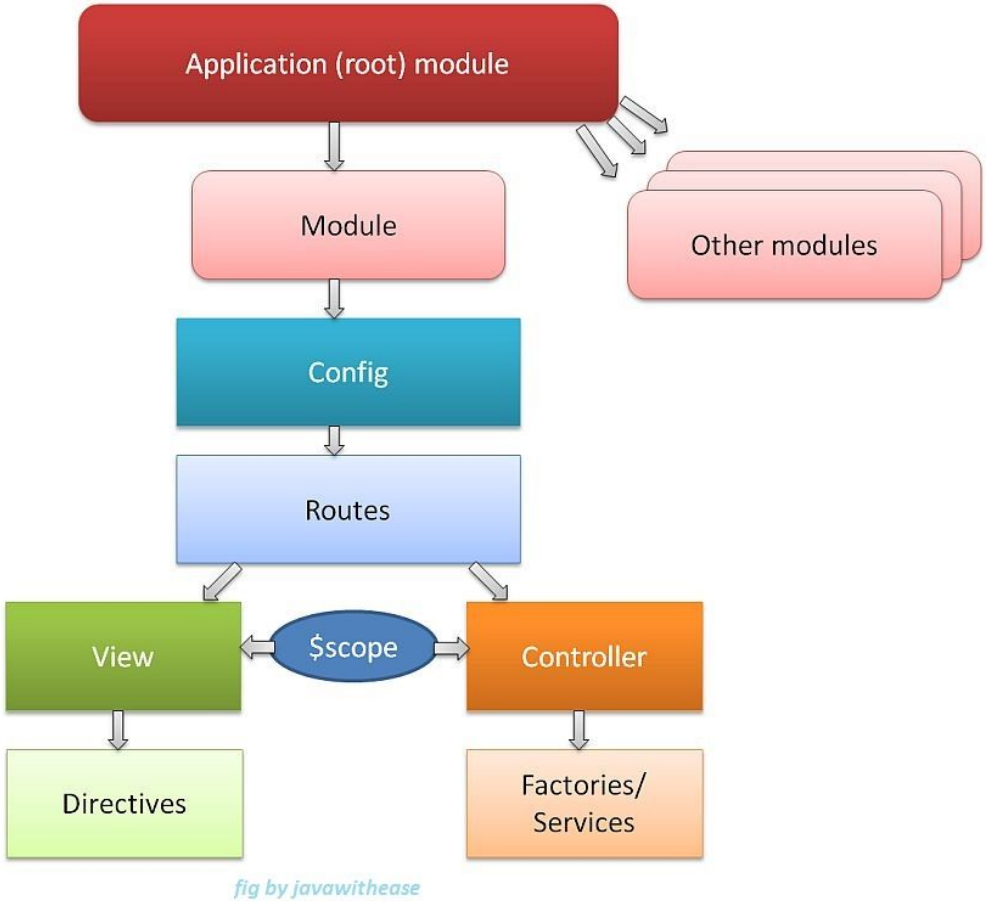
Arquitectura tradicional



Arquitectura de SPA



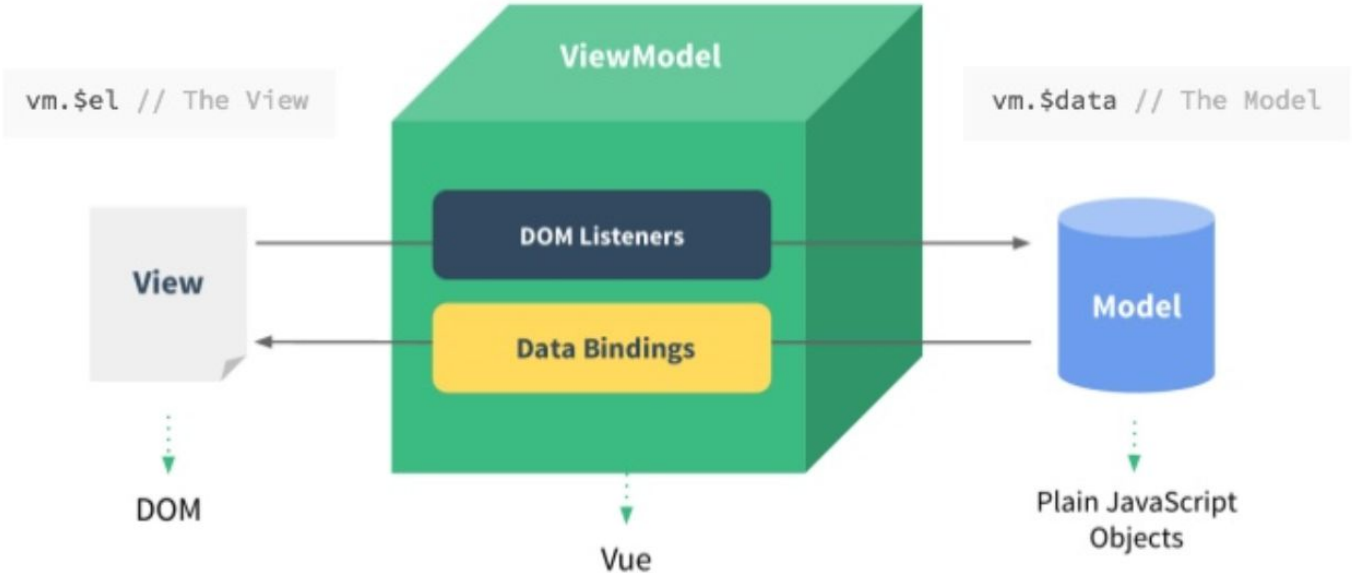
Ejemplo de arquitectura para Angular 1.x



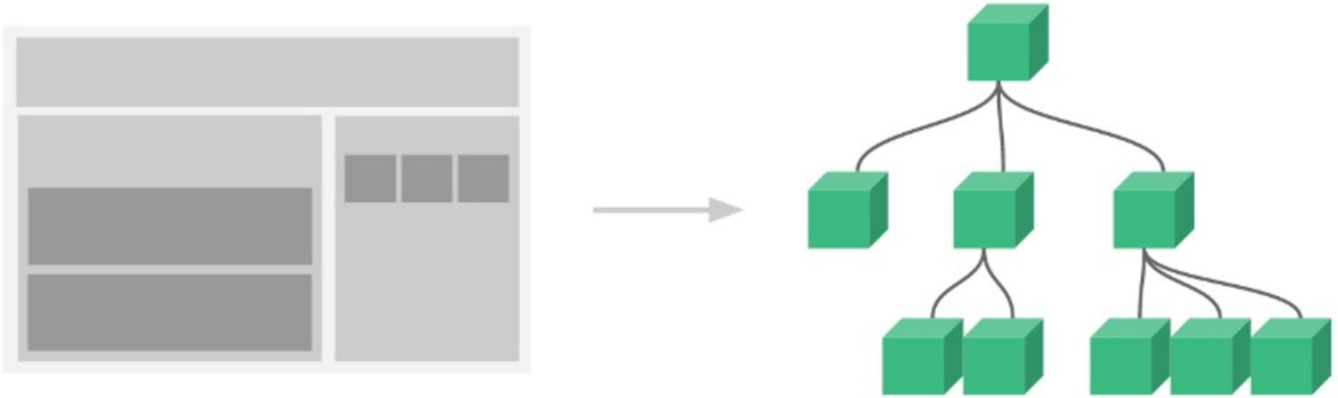
Vue. JS

Arquitectura de Vue.js

```
var vm = new Vue({ /* options */ })
```



Una aplicación es un árbol de componentes reutilizables

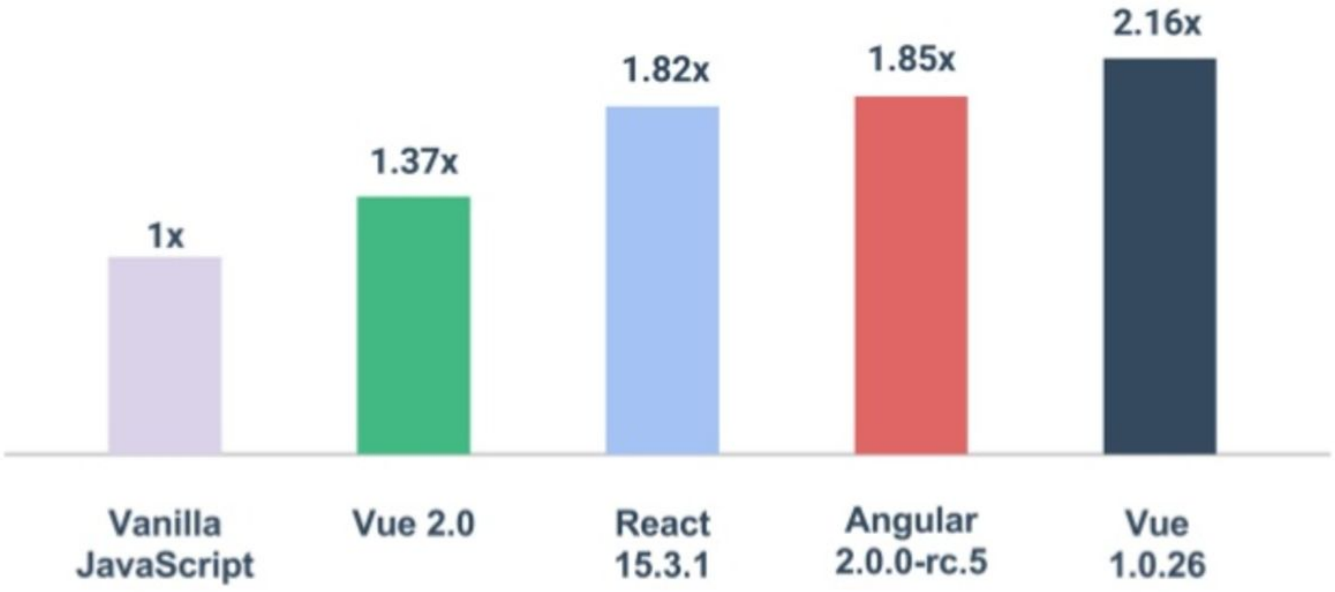


Instancia de Vue.js

```
<div id="app">
  {{ message }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

Desempeño


1x
1.37x
1.82x
1.85x
2.16x
Vanilla JavaScript
Vue 2.0
React 15.3.1
Angular 2.0.0-rc.5
Vue 1.0.26

Gracias por su atención

Consultas a:

- diego.avila@usach.cl