

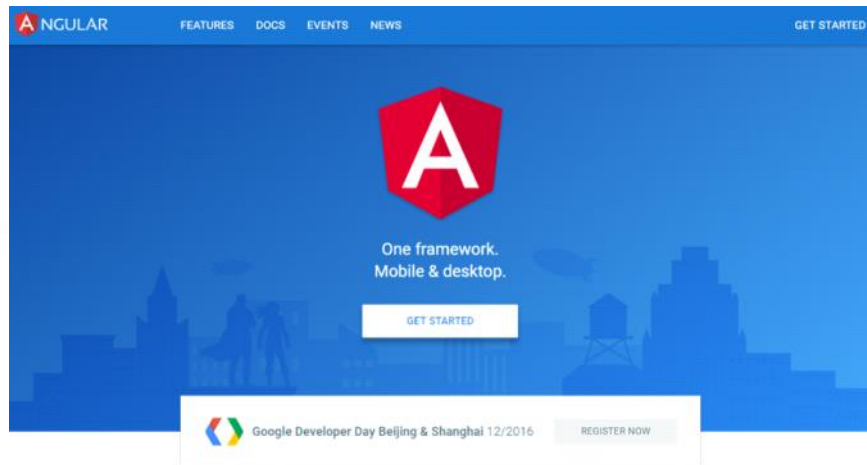


# Versiones de Angular

sábado, 9 de septiembre de 2017 16:43



<https://angularjs.org/>



<https://angular.io/>

Angular 2 - Angular 4 - Angular 5

## Orígenes y desarrollo

- proyecto de **código abierto**, realizado íntegramente en JavaScript
- creado en **2009**, por Misko Hevery de *Brat Tech LLC* y Adam Abrons
- está mantenido por **Google** y junto con una amplia y creciente **comunidad**.
- puede coexistir con **otros frameworks** (e.g JQuery, Bootstrap, *Material Design*)
- se ha hecho muy popular desde finales de 2012 hasta ahora,
- especialmente en asociación con otras tecnologías, dando lugar a **MEAN**

Misko Hevery



Adam Abrons



MongoDB  
ExpressJS  
AngularJS  
NodeJS



<http://mean.io/#/>

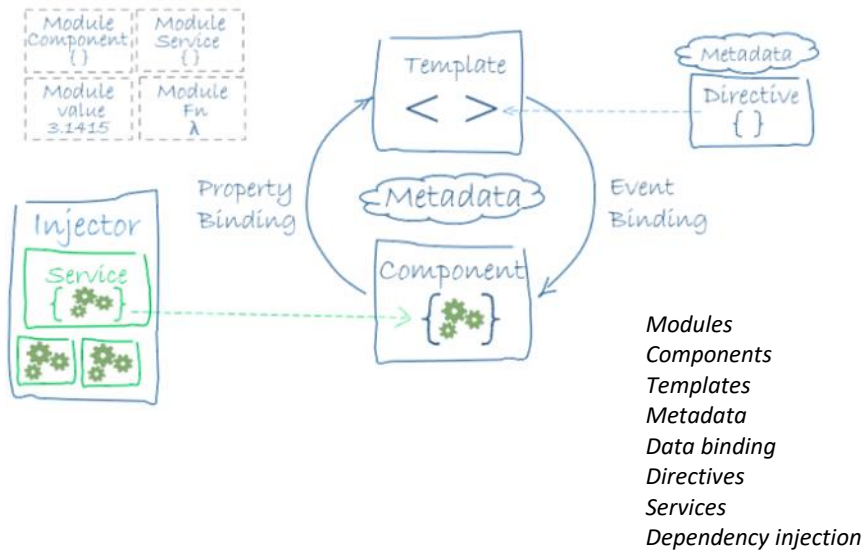
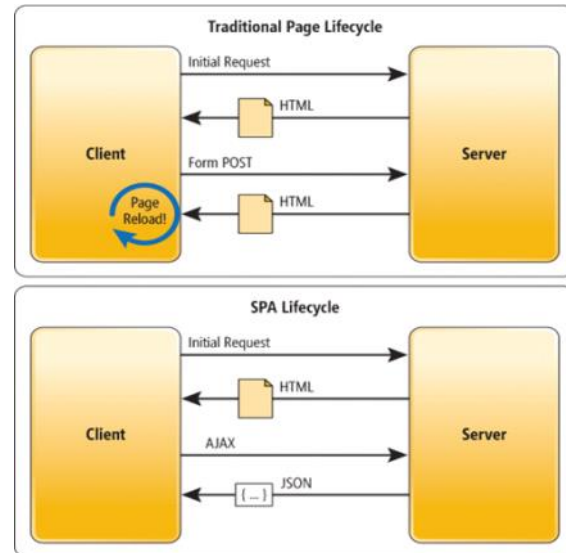
Se habla de una nueva *technology fullstack* como antes era *xAMP* (Apache + MySQL + PHP)

Se traduce a aplicaciones **JavaScript de principio a fin** (*End-to-End*)



## El curso anterior, en resumen ...

- Un framework de desarrollo apps **SPA**
- Se recomienda **TypeScript**, más preparado para grandes aplicaciones (pero puede usar ES5, ES6)
- No es compatible con Angular 1, pero comparte su filosofía
- Mucho mejor rendimiento que Angular 1
- Orientado a **componentes**, con templates e inyección de **dependencias**
- Sobre los templates estáticos se inyectan dinámicamente los datos procedentes de servidores (e. g. API REST)
- Seguramente será uno de los frameworks más usados para desarrollo web en los próximos años



Routing  
HttpClient

### ... y más

- Formularios reactivos y su validación (*NgForm* y *NgControl*)
- Carga bajo demanda de módulos (para acelerar la carga inicial de la aplicación)
- Gestión del estado al estilo *Redux* (*ngRx*)
- Animaciones
- *Testing* unitario y de integración  
(*Jasmine*, *Karma*, *Protractor*, Inyección de dependencias de *testing*...)

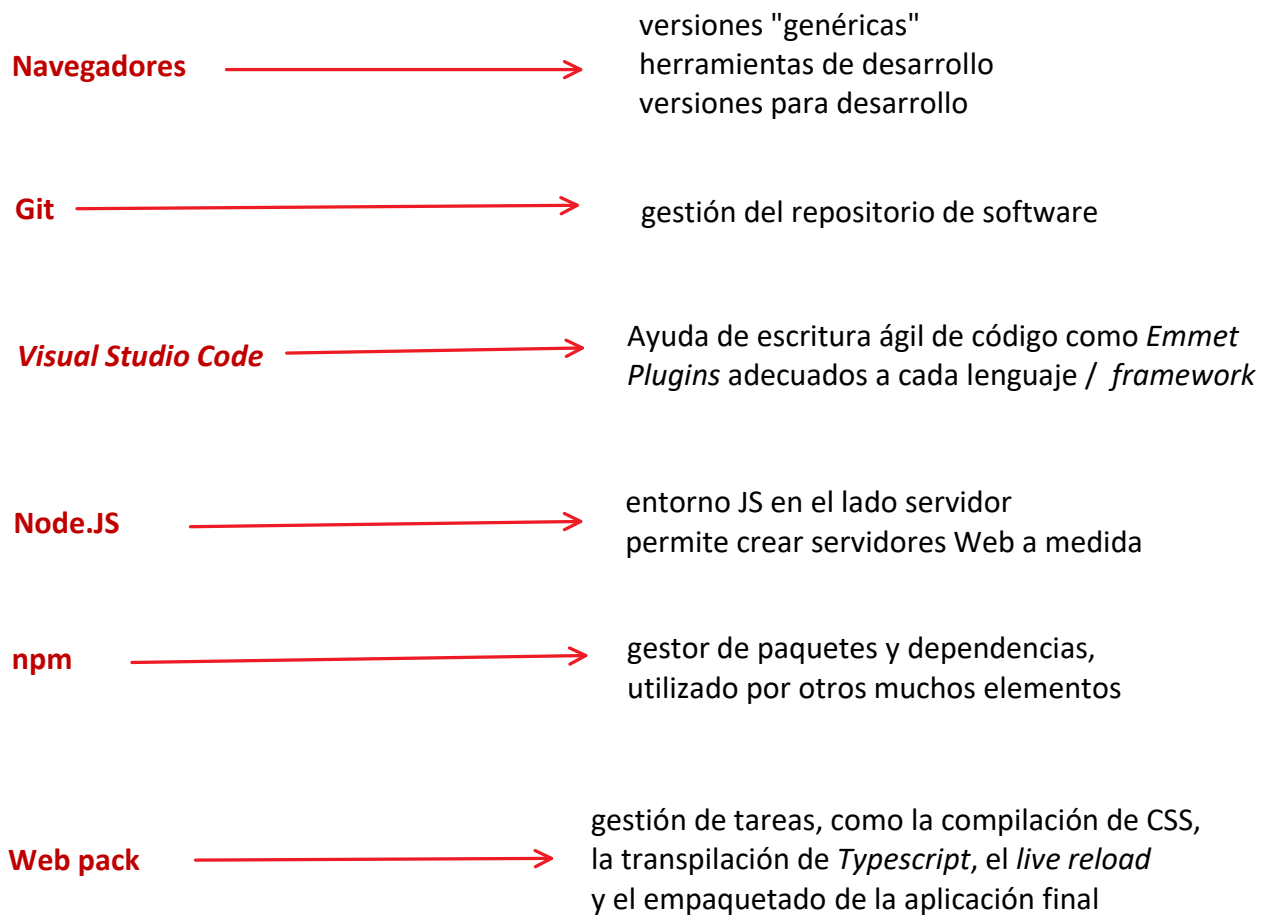


### Agenda del curso

- 1. ES6 / Typescript avanzado**
  - a. Retomando ES6
  - b. TypeScript: Tipos (Union types, Alias...)
  - c. Decoradores
- 2. Testing**
  - a. Plataforma: Karma + jasmine
  - b. Testing Unitario
  - c. Test end-to-end
- 3. Bibliotecas de componentes**
  - a. Angular y Bootstrap (Responsiveness)
  - b. Angular Material (Librería components UI)
  - c. Gráficos, Traducción
- 4. Más allá de los componentes**
  - a. Crear directivas propias
  - b. Crear pipes propia
  - c. Animaciones
- 5. DI: Inyectores & Providers**
  - a. Árbol de providers
- 6. Angular Avanzado. Routing**
  - a. Como leer parámetros
  - b. Lazy loading
  - c. Animaciones
- 7. Reactive Programming**
  - a. Observable / Subscribe
  - b. Operadores más comunes
  - c. Comunicación entre componentes
  - d. HttpClient
- 8. Formularios reactivos**
  - a. Formulario reactivos...
  - b. Validación
- 9. Arquitectura de proyecto avanzado**
  - a. Estructura de directorios
  - b. Npm, Package.json, Angular-cli, Webpack
  - c. REDUX
- 10. Angular y otros frameworks**
  - a. Angular Environment
  - b. Angular vs React
  - c. Como pasar de AngularJS a Angular
  - d. Ionic- Introducción

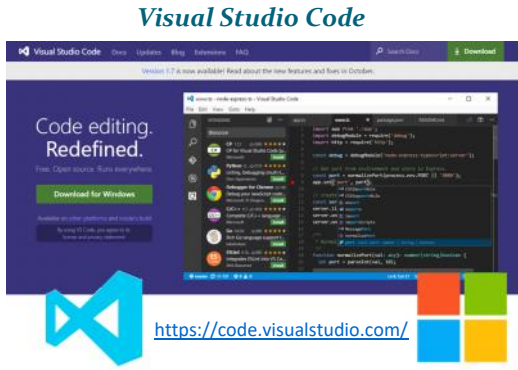
# Entorno de trabajo

domingo, 28 de enero de 2018 19:20



# Visual Studio Code

sábado, 9 de septiembre de 2017 18:21



## Visual Studio Code

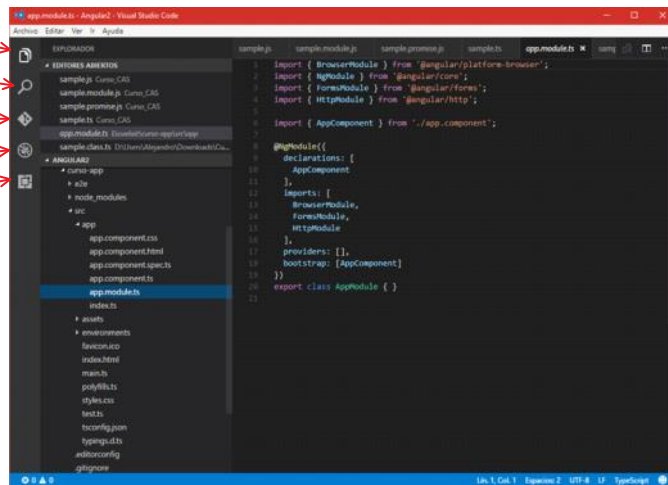
Carpetas y ficheros

Búsquedas en el proyecto

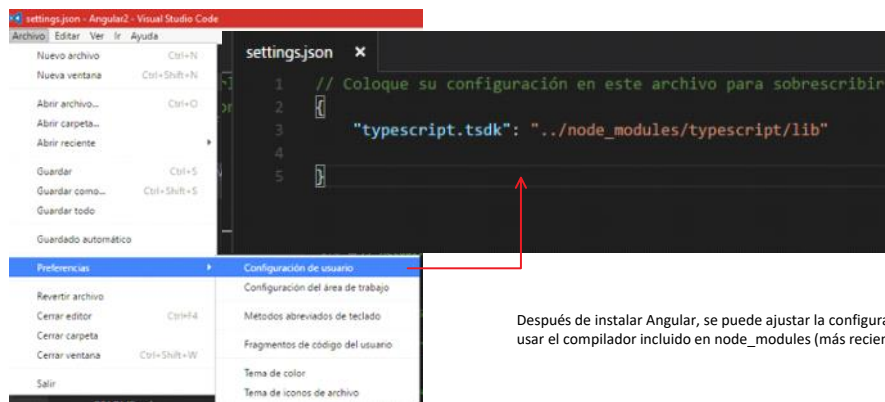
Git - GitHub integrado

Depurador

Extensiones



## Configuración VSC

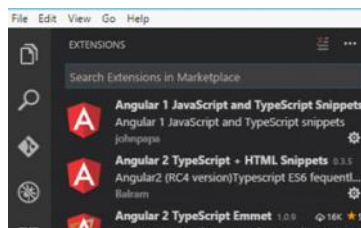


Después de instalar Angular, se puede ajustar la configuración para usar el compilador incluido en node\_modules (más reciente)

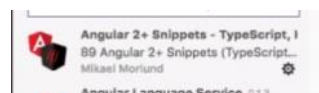
<https://code.visualstudio.com/Docs/languages/typescript>

## Extensiones de VSC

- Angular 2 TypeScript
- Angular 2 TypeScript Emmet
- Angular 4 and TypeScript/HTML VS Code Snippets (Dan Wahlin)
- **Angular 4 TypeScript Snippets \*\*\*\*\* (John Papa)**
- **Angular Language Service**



A.Basalo, Angular 4



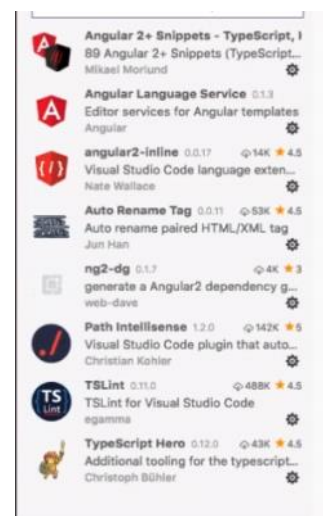
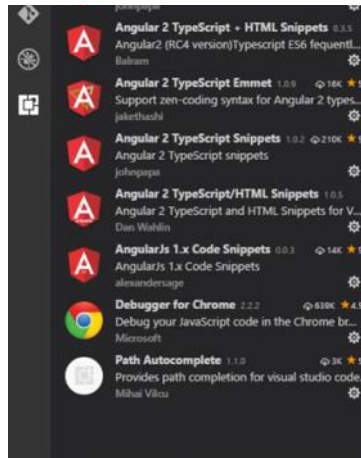
#### HTML VS Code Snippets

(Dan Wahlin)

- **Angular 4 Typescript Snippets \*\*\*\*\***  
(John Papa)
- **Angular Language Service**
- **angular2-inline**  
(Nate Wallace)

- **TSLint \*\*\*\*\***  
(egamma)
- **Auto Import (steoates) /**  
TypeScript Hero (Christoph Bühler)
- **AutoRenameTag**

- **Debugger para Chrome \*\*\*\*\***  
(Microsoft)
- **npm \*\*\***  
(egamma)
- **Path Intellisense \*\*\***  
(Christian Kohler)



<https://marketplace.visualstudio.com/items?itemName=johnpapa.angular-essentials>

Pack con las extensiones de Angular, según recomendación de John Papa

- Angular v5 Snippets
- Angular Language Service
- Angular Inline
- Path Intellisense
- tslint
- Chrome Debugger

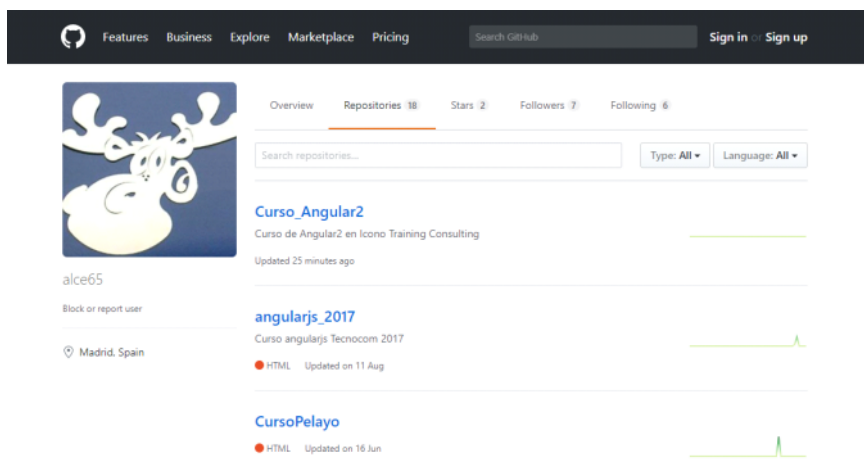
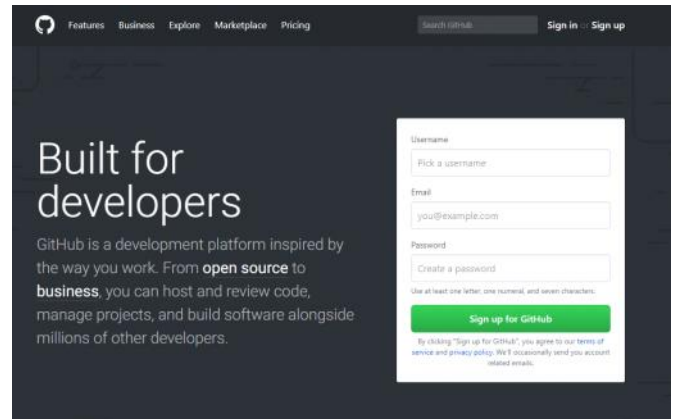
- Editor Config
- PrettierVS Code
- Winter is Coming theme

# Git. GitHub

domingo, 10 de septiembre de 2017 12:28

- Git
- GitHub

```
$ git config --global user.name "Pepe Perez"
$ git config --global user.email pperez@example.com
$ git config --global core.editor "code --wait"
```



<https://github.com/alce65>



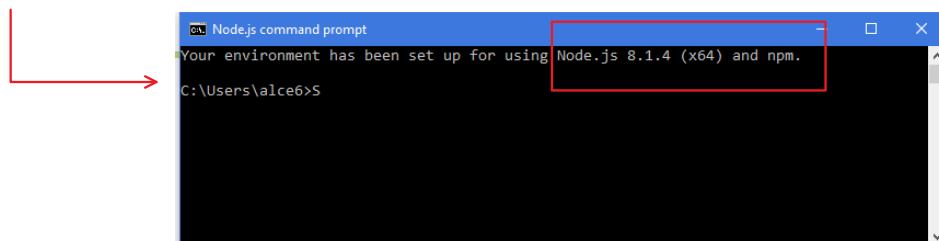
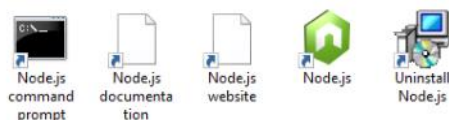
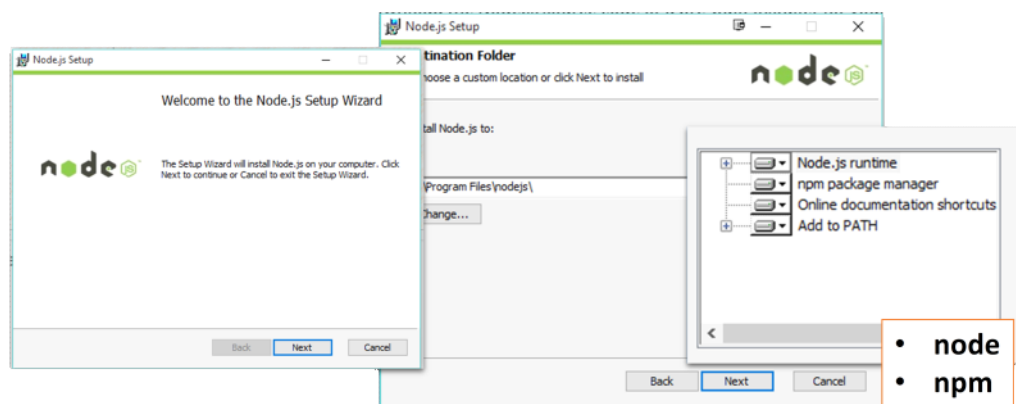
# Node.js & npm

sábado, 9 de septiembre de 2017 20:35

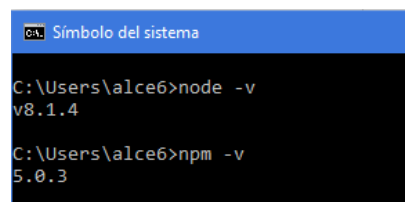
v8.4.0 Current

Latest Features

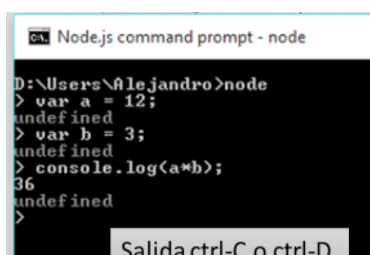
node-v8.4.0-x64.msi



- Accesible desde cualquier CLI (*cmd*, PowerShell...)
- Incluido en el *path*
- Se puede comprobar consultando la versión de *Node* y de *npm*



Con el comando "*node*" entramos en la consola de *Node*, un entorno REPL (*Read-Eval-Print-Loop*) similar a la consola de JS en los navegadores



El mismo comando *node* <fichero.js>, permite la ejecución de un fichero

## Construcción de proyectos / empaquetado

herramientas para procesar  
los fuentes de la aplicación

- Reducción del tiempo de descarga
- Preprocesadores CSS
- Optimización del código, CSS, HTML
- Cumplimiento de estilos y
- Generación de JavaScript (transpilación)



<http://gruntjs.com/>



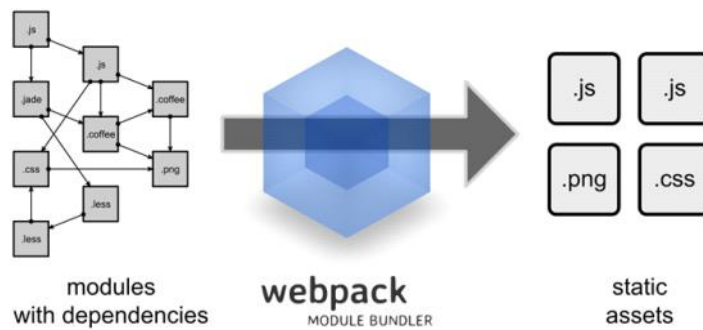
<http://gulpjs.com/>



<http://broccolijs.com/>



<https://webpack.github.io/>



finalmente incluido en **Angular-cli**

# Configuración de *proxies*

lunes, 7 de agosto de 2017 21:38

## Configuración *git*

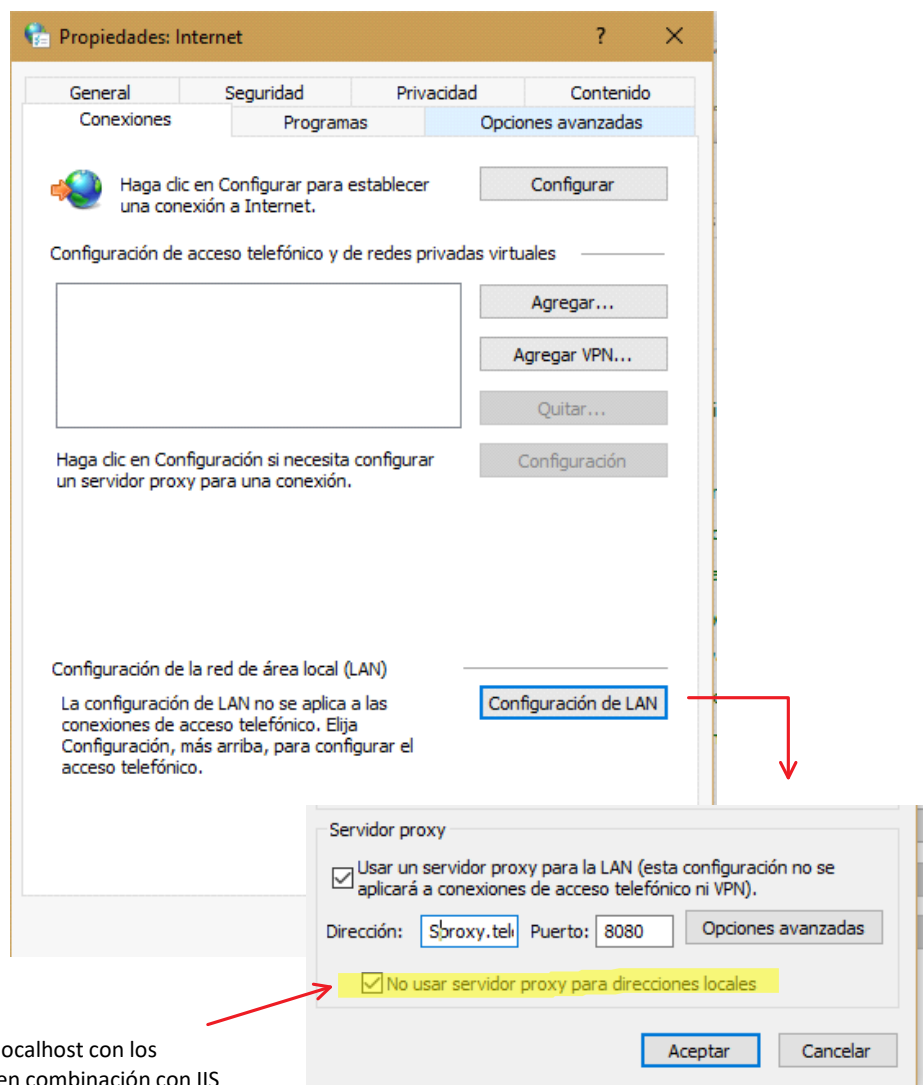
```
git config --global http.proxy http://user:passw@proxy.empresa.es:8080
```

## Configuración *npm*

```
$>npm config set proxy http://user:passw@proxy.empresa.es:8080
```

```
$>npm config set https-proxy http://user:passw@proxy.empresa.es:8080
```

## Propiedades de internet



Permite usar localhost con los navegadores en combinación con IIS para probar el código desarrollado

# Angular-cli

sábado, 9 de septiembre de 2017 18:51

Angular command line interface

Herramienta oficial de gestión de proyectos

<https://cli.angular.io>

Ofrece comandos para todo el ciclo de desarrollo:

- Generación (*bootstrapping*) del proyecto inicial
- Herramientas de generación de código
- Modo desarrollo con
  - compilado automático de *TypeScript*
  - arranque de un servidor Web
  - y actualización del navegador (*Live Reloading*.)
- *Testing*
- Construcción del proyecto para distribución (*build*)



Herramientas de terceros  
incluidas en Angular-cli



Herramientas de testing

## Instalación

Desde una ventana de terminal "como administrador"

```
npm install -g @angular/cli
```

instala globalmente la herramienta angular cli

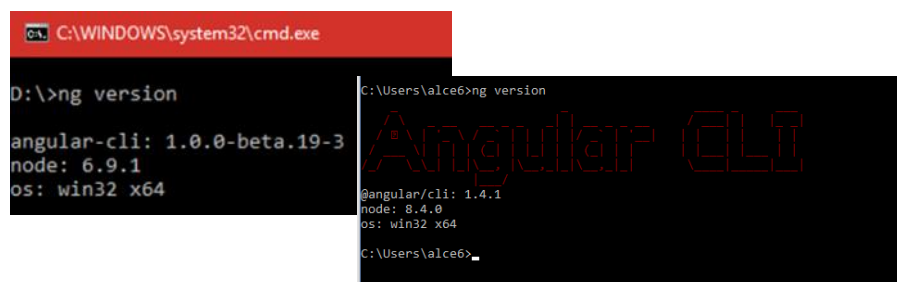
```
npm update -g @angular/cli
```

actualiza la instalación global de angular cli

Destino de la instalación

"<user>\AppData\Roaming\npm\node\_modules"

Resultado de la instalación



# Proyecto básico

domingo, 28 de enero de 2018 19:21

## Generación de un proyecto

Desde la carpeta donde queremos que resida nuestro proyecto

```
ng new my-app
```

crea una carpeta en la que descarga hasta 250MB. configura y organiza el conjunto de herramientas de una forma prefijada

Modificadores

-sg: no crea un nuevo repositorio Git para el proyecto  
--routing: crea el fichero de rutas como módulo

```
cd my-app
```

desde el directorio del proyecto, recién creado se levanta un servidor *Node* que mostrará la aplicación en localhost:4200

```
ng serve
```

- *transpilará Typescript*
- recargará automáticamente al guardar un fichero fuente

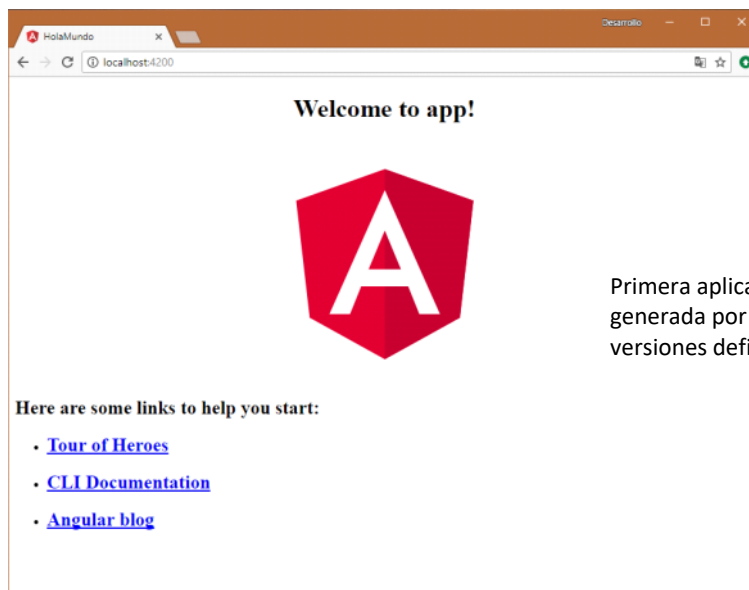
## Resultado: ejecución

```
D:\Desarrollo\Angular2\Curso_CAS\curso-app>ng serve
** NG Live Development Server is running on http://localhost:4200. **
4387ms building modules
47ms sealing
0ms optimizing
0ms basic module optimization
180ms module optimization
4ms advanced module optimization
18ms basic chunk optimization
15ms chunk optimization
0ms advanced chunk optimization
16ms module and chunk tree optimization
266ms module reviving
15ms module order optimization
16ms module id optimization
16ms chunk reviving
0ms chunk order optimization
31ms chunk id optimization
109ms hashing
16ms module assets processing
250ms chunk assets processing
15ms additional chunk assets processing
0ms recording
0ms additional asset processing
4069ms chunk asset optimization
2923ms asset optimization
78ms emitting
Hash: 541eb735658c9449ad60
Version: webpack 2.1.0-beta.25
Time: 5200ms

   Asset      Size  Chunks             Chunk Names
main.bundle.js  2.71 MB    0, 2  [emitted]  main
styles.bundle.js 10.2 kB    1, 2  [emitted]  styles
  inline.js     5.53 kB    2  [emitted]  inline
  main.map     2.82 MB    0, 2  [emitted]  main
  styles.map   14.2 kB    1, 2  [emitted]  styles
  inline.map    5.59 kB    2  [emitted]  inline
  index.html   475 bytes  [emitted]
Child html-webpack-plugin for "index.html":
   Asset      Size  Chunks             Chunk Names
  index.html  2.81 kB    0
webpack: bundle is now VALID.
```

Ventana de la consola en la que *webpack*

- levanta un servidor Web basado en *Node* en la máquina local y el puerto 4200
- con la aplicación empaquetada de forma temporal en modo adecuado para el desarrollo,
- capaz de actualizarse automáticamente ante los cambios en los ficheros fuente



Primera aplicación en Angular2  
generada por angular-cli en sus  
versiones definitivas

Here are some links to help you start:

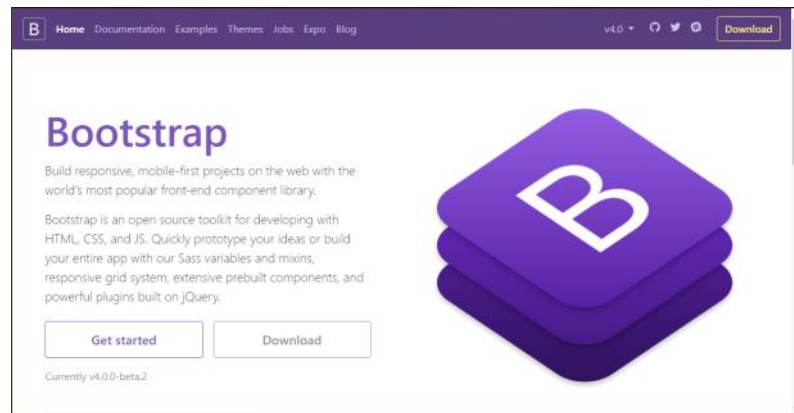
- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

# Bootstrap

lunes, 6 de noviembre de 2017 20:37

Probablemente la más conocida entre las alternativas para construir el UI en las aplicaciones Angular

Actualmente en su versión 4.0



## Instalación

```
npm install bootstrap@4.0.0-alpha.6
```

- añade la carpeta Bootstrap en *node\_modules*
- actualiza las dependencias en *package.json*

Más información

<https://medium.com/codingthesmartway-com-blog/using-bootstrap-with-angular-c83c3cee3f4a>

## Utilización

Incorporamos el CSS en el fichero de configuración de angular cli: *.angular-cli.json*

```
"styles": [  
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "styles.css"  
],
```

Existe un problema en angular/cli para acceder a los estilos cuando *node\_modules* es un link simbólico. En ese caso hay que añadir Bootstrap desde styles.css empleando un import

```
styles.css:  
@import "../node_modules/bootstrap/dist/css/bootstrap.min.css",
```

Los elementos JS/JQuery incluidos en Bootstrap **NO SE UTILIZAN** en Angular

En su lugar



## Componentes ng-bootstrap

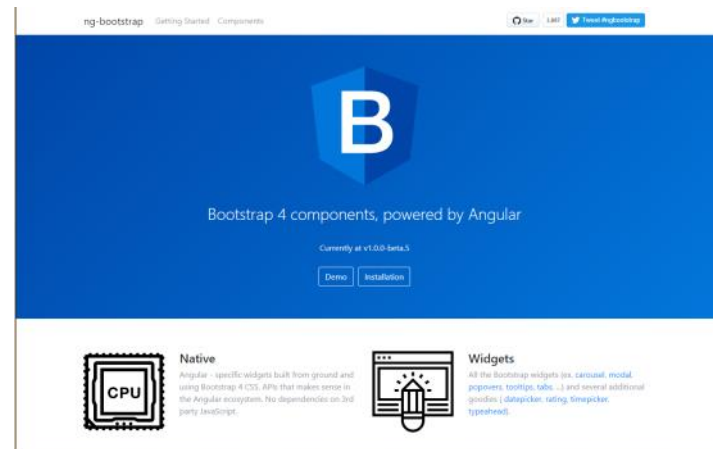
Los componentes de Bootstrap han sido migrados a Angular como parte de <https://ng-bootstrap.github.io/#/home>





## Componentes ng-bootstrap

Los componentes de Bootstrap han sido migrados a Angular como parte de <https://ng-bootstrap.github.io/#/home>

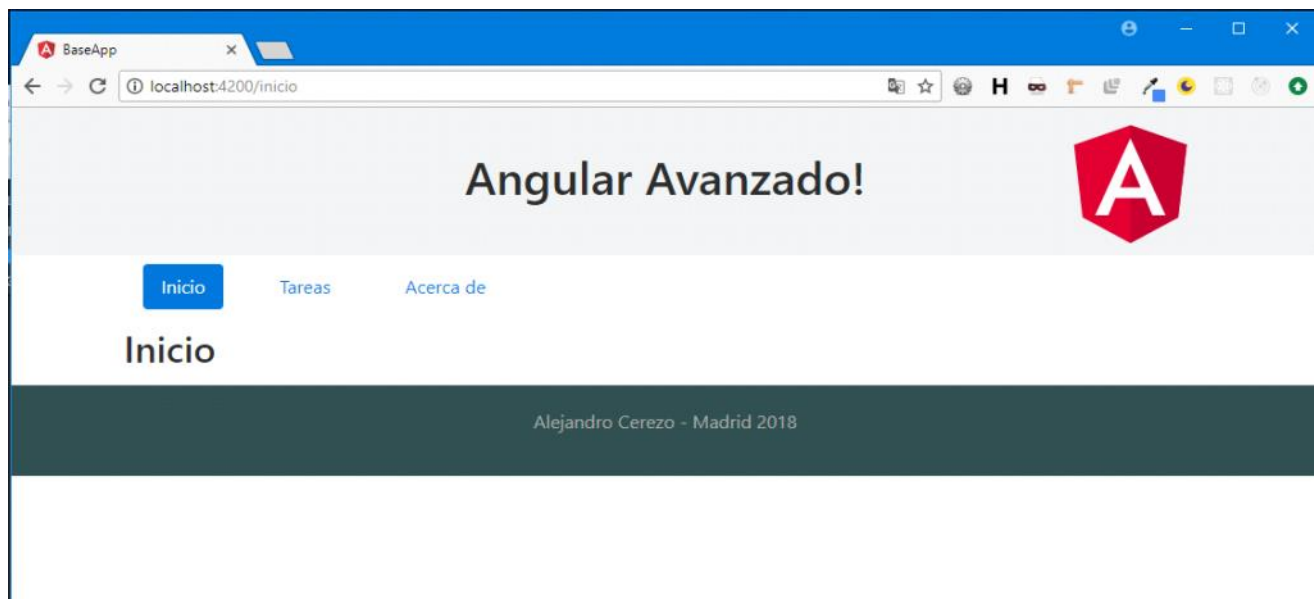




# Implementación

domingo, 28 de enero de 2018 21:11

- Módulo Shared
  - o Componentes Cabeza, Pie, Menu
  - o Uso del *Grid Responsive* de Bootstrap
  - o Creación del menú basado en Bootstrap
  - o Uso de `assets` para los elementos estáticos: imagen del logo
- Enrutamiento
  - o Módulos con un componente plano (flat): Inicio, Tareas, About creados como `ng g c --flat -is -it`
  - o Definición de las rutas correspondientes



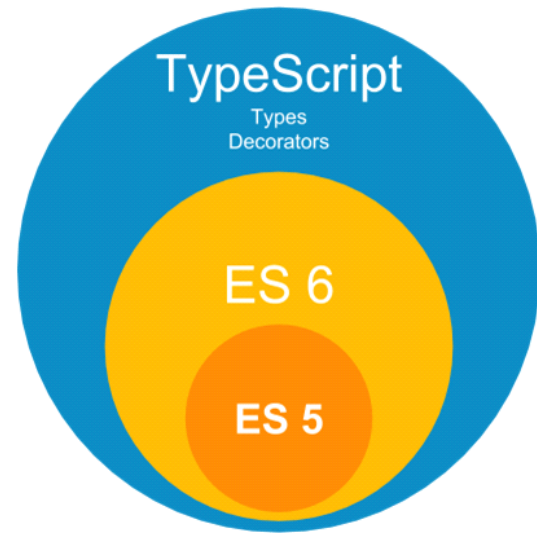
## Programación en ES6 + Typescript

- ES6
  - let (variables con ámbito)
  - clases (class)
  - módulos (import y export)
  - funciones arrow
- TypeScript
  - tipos
  - anotaciones

### Transpilación



resultados en ES5 / ES6  
(compatibilidad)



TypeScript is a javascript superset

# ES6

sábado, 30 de diciembre de 2017 19:39

# Variables y constantes

sábado, 30 de diciembre de 2017 12:47

la declaración con `var` siempre se "eleva" al inicio del código, independientemente de que acompañe o no a una inicialización

```
(function prueba_var () {  
  console.log(x)  
  var x = 20  
})();
```

Equivale a

```
var x  
(function prueba_var () {  
  console.log(x)  
  x = 20  
})();
```

devuelve undefined

la declaración con `let` no se eleva

```
(function prueba_let () {  
  console.log(x)  
  let x = 20  
})();
```

ReferenceError x is not defined

Además su ámbito de existencia está limitado al bloque en que se declara y los bloques contenidos en el

```
(function bloques () {  
  let x = 0  
  let y = 0  
  {  
    x = 20  
    let y = "Modificada"  
    console.log(x)  
    console.log(y)  
    let z = 25  
  }  
  console.log(x)  
  console.log(y)  
  console.log(z)  
})();
```

20  
Modificada  
20  
0  
ReferenceError z is not defined

`const` declara como constantes los tipos elementales o las referencias a los objetos, pero nunca el contenido de los objetos. Para ello disponemos del método de ES5 `Object.freeze()`

```
(function constantes () {  
  const MES = "Enero"
```

```


const DIAS = 31
const USER = {
  name : "",
  apellido : "",
  puesto : ""
}

USER.name = "Pepe"
USER.apellido = "Perez"
USER.edad = 25
delete USER.puesto


console.log(USER)
USER = {}
})();

```

En un objeto "constante" podemos modificar, añadir o eliminar propiedades.

 { name: 'Pepe', apellido: 'Perez', edad: 25 }

No podemos reasignar el objeto  
 TypeError: Assignment to constant variable.



# Nuevas sintaxis

sábado, 30 de diciembre de 2017

19:39

- Clases
- *arrow functions*
- *template strings*
- *Map*
- Bucles *for...of*

# Operador de propagación

domingo, 28 de enero de 2018 23:51

El operador de propagación *spread operator* permite que una expresión sea expandida en situaciones donde se esperan múltiples argumentos (llamadas a funciones) o múltiples elementos (*arrays* literales).

Llamadas a funciones:

```
f(...iterableObj);
```

*Arrays* literales:

```
[...iterableObj, 4, 5, 6]
```

Desestructuración *destructuring*:

```
[a, b, ...iterableObj] = [1, 2, 3, 4, 5];
```

Se define una función que espera múltiples argumentos

```
function f(x, y, z) { }
```

La función es invocada pasándole un array con el operador de propagación, que será expandido a los múltiples argumentos que espera la función

```
var args = [0, 1, 2];  
f(...args);
```

<http://www.etnassoft.com/2014/06/03/el-operador-de-propagacion-en-javascript-ecmascript-6-y-polyfill/>

# Desestructuración

lunes, 29 de enero de 2018 0:02

La desestructuración no es un concepto nuevo en programación. De hecho, eso es algo que se ha tenido muy en cuenta a la hora de fijar el estándar: incorporar de forma progresiva al lenguaje Javascript lo mejor de otros.

- en Python o en OCaml, tendríamos las tuplas
- en PHP las listas
- sus correspondientes en Perl y Clojure...


una expresión que permite asignar valores a nombres conforme a una estructura de tabla dada

## Desestructuración de ARRAYS

```
const aNumbers = [1, 2, 3, 4]

const [uno, dos, tres] = aNumbers
console.log(uno, dos, tres)
```

Desestructuración de un array declarando las variables




## Desestructuración de un objeto

```
{
  const oNumbers = {
    uno: 1,
    dos : 2,
    tres : 3,
    cuatro : 4
  }

  const {uno , cuatro, tres, dos} = oNumbers

  console.log(uno, dos, tres, cuatro)
}
```

Cada una de las variables recibe el valor de uno de los miembros del objeto



Ejemplos en el fichero basicos.4.destructuring.js

<http://www.etnassoft.com/2016/07/04/desestructuracion-en-javascript-parte-1/>



# Valores por defecto

lunes, 29 de enero de 2018 0:11

ES6: parámetros por defecto y  
desestructuración del paso de parámetros

```
function drawCircleES6( {radius = 30,  
                        coords = { x: 0, y: 0}  
                        } = {}) {  
    console.log(radius, coords);  
};
```

Desestructuración:

```
{radius = 30,  
 coords = { x: 0, y: 0}} = {}
```

valores por defecto  
desestructurados:  
- radius  
- coords

parámetro real  
recibido como objeto

Diversas llamadas a la función

```
drawCircleES6(); // radius: 30, coords.x: 0, coords.y: 0 }  
drawCircleES6({radius: 10}); // radius: 10, coords.x: 0, coords.y: 0 }  
drawCircleES6({coords: {y: 10, x: 30}, radius: 10}); // radius: 10, coords.x: 30, coords.y: 10 }
```

Ejemplos en el fichero basicos.4.default.js

Una promesa representa el resultado eventual de una operación.  
Se utiliza para especificar que se hará cuando esa eventual operación de un resultado de éxito o fracaso.

## Promesas

JS

Un objeto promesa representa un valor que todavía no está disponible pero que lo estará en algún momento en el futuro

Permiten escribir código asíncrono de forma más similar a como se escribe el código síncrono:

La función asíncrona retorna inmediatamente y ese retorno se trata como un proxy cuyo valor se obtendrá en el futuro

El API de las promesas en Angular corresponde al **servicio \$q**

la biblioteca Q desarrollada por **Kris Kowal**

<https://github.com/krisowal/q>



## Promesas: \$q

JS

```
function getPromise()
```

```
    var deferred=$q.defer();
```

crea una promesa

```
        deferred.resolve()  
        deferred.reject()
```

resuelve la promesa en un sentido u otro al cabo del tiempo

```
    return deferred.promise
```

devuelve la promesa

```
var promise = getPromise();
```

```
    promise.then(successCallback,failureCallback,notifyCallback);
```

```
    promise.catch(errorCallback)
```

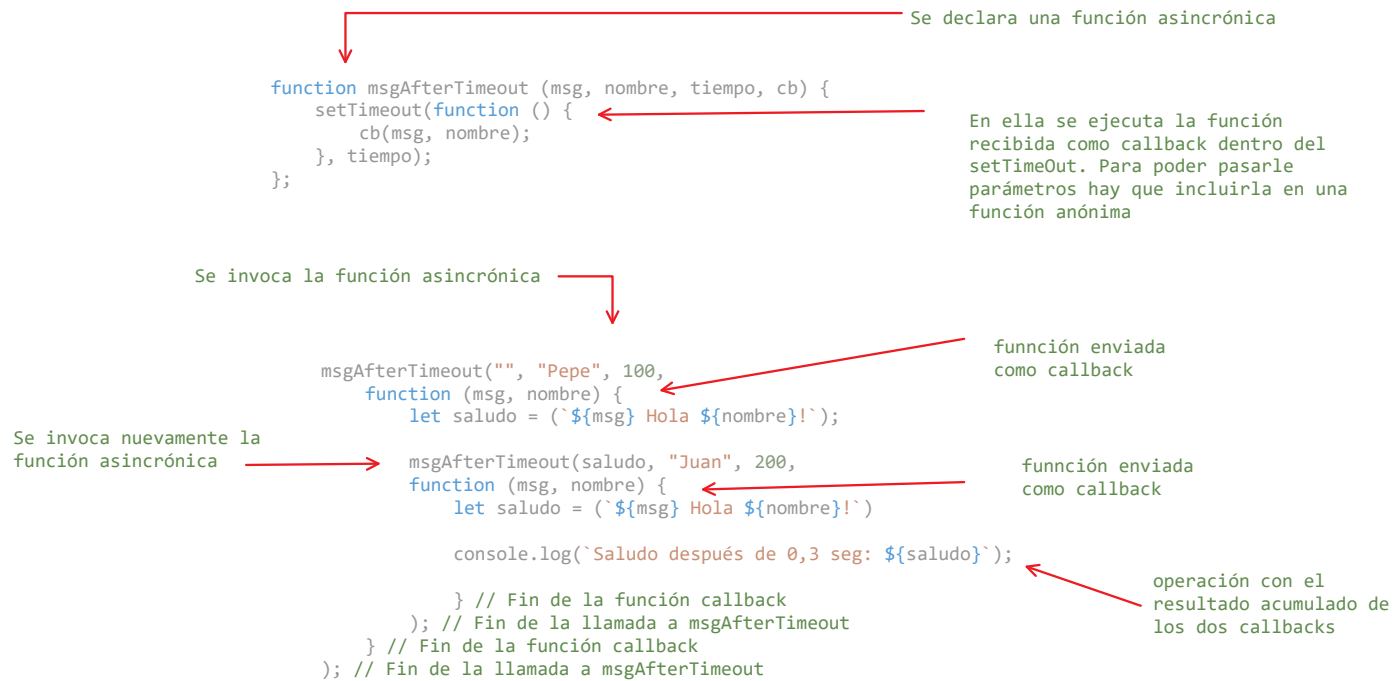
```
    promise.finally(callback)
```

```
promise.catch(errorCallback)  
promise.finally(callback)
```



## Callbacks anidados

sábado, 14 de octubre de 2017 13:52



## Implementación new Promise

el objeto promesa recibe como parámetros dos funciones:

- La función "*resolve*": se ejecutará cuando queramos finalizar la promesa con éxito.
- La función "*reject*": se ejecutará cuando queramos finalizar una promesa informando de un caso de fracaso.

```
function hacerAlgoPromesa (){  
  return new Promise (function (resolve, reject) {  
    console.log ( 'hacer algo que ocupa un tiempo...');  
    setTimeout (resolve(), 1000);  
  })  
}
```

En este caso la promesa siempre se resuelve correctamente; la función admite como parámetro los datos que la promesa deba retornar

## Utilización

a la función que retorna el objeto promesa se le encadenan el método `then` con dos funciones como parámetros,

- la función que se ejecutará cuando la promesa haya finalizado con éxito.
- la función que se ejecutara cuando la promesa haya finalizado informando de un caso de fracaso.

```
hacerAlgoPromesa()  
  .then (  
    function (){ console.log (' la promesa terminó.');},  
    function (){ console.log (' la promesa fracasó.');}  
  )
```

Alternativamente puede utilizarse el método `catch` para declarar la función que se ejecutara cuando la promesa haya finalizado informando de un caso de fracaso.

```
hacerAlgoPromesa()  
  .then (function (){ console.log (' la promesa terminó.');})  
  .catch(function (){ console.log (' la promesa fracasó.');})
```

[https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/Promise](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise)

## Ejemplo de promesas en ES6

sábado, 14 de octubre de 2017 14:43

```
function msgAfterTimeout (msg, nombre, tiempo) {  
  return new Promise((resolve, reject) => {  
    setTimeout(  
      () => resolve(`${msg} Hola ${nombre}!`),  
      tiempo)  
    })  
  })  
}
```

Función que crea y devuelve un **objeto promesa**

En este caso, la promesa siempre se resuelve correctamente, creando un mensaje de saludo a un usuario

### Utilización de las promesas, encadenando las llamadas a ellas

*msg* almacena el resultado del primer proceso asincrónico

*msg* almacena los sucesivos resultados de los procesos asincrónicos

```
msgAfterTimeout("", "Pepe", 100)  
  .then((msg) =>  
    msgAfterTimeout(msg, "Juan", 200))  
  .then((msg) => {  
    console.log(`Saludo después de 0,3 seg: ${msg}`)  
  })
```

operamos finalmente con *msg*

```
MENSAJES  
Saludo despues de 0,3 seg:  Hola Pepe! Hola Juan!  
PS D:\Desarrollo\Front_End_alce65\Angular\angular_4_2017\02_tecnologias\ES6>
```

# TypeScript

viernes, 19 de enero de 2018 18:40

## Decoradores o anotaciones

sábado, 14 de octubre de 2017 19:13

Ejemplo de función que define un *decorator* sencillo, sin argumentos

```
function course(target) {  
  Object.defineProperty(  
    target.prototype,  
    'course',  
    {value: () => "Angular 2"}  
  )  
}
```

Uso del anterior *decorator* para modificar una clase

```
@course  
class Person {  
  firstName;  
  lastName;  
  constructor(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
  }  
}
```

Instanciación de un objeto de esta clase

```
let oPersona = new Person("Pepe", "Pérez");  
console.log(oPersona.course()); // Angular 2
```

Ejemplo de función que define un *decorator* con argumentos

```
function Student(config) {  
  return function (target) {  
    Object.defineProperty(  
      target.prototype,  
      'course',  
      {value: () => config.course}  
    )  
  }  
}
```

La función recibe como parámetro el objeto de configuración

Uso del anterior *decorator* para modificar una clase, pasándole un argumento en forma de objeto

```
@Student({  
  course: "Angular 2"  
})  
class Persona {  
  firstName;  
  lastName;  
  constructor(firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
  }  
}
```

al definir el *decorator* utiliza la clave correspondiente del objeto recibido

Instanciación de un objeto de esta clase

```
let oEstudiante = new Persona("Pepe", "Pérez");  
console.log(oEstudiante.course()); // Angular 2
```



# Otras características

domingo, 10 de septiembre de 2017 23:02

- *Getter / Setter* con sintaxis de atributo
- *Type guards Instanceof / typeof*
- Compatibilidad de tipos estructural
- Sobrecarga de métodos “especial”

Cuando se definen las pruebas

- antes del desarrollo (TDD - *Test Driven Development*)  
conceptualmente mucho más complejo  
una de las propuestas ligadas a las metodologías ágiles, junto con el *refactoring*
- después del desarrollo

**Pruebas unitarias**  
- (en desarrollo *front*)

Se critica que la mejora obtenida en la calidad no se compensa con el esfuerzo de usar y mantener un extorno de pruebas (*testing*)

De alguna manera Angular facilita las pruebas para desmentir esta crítica, proporcionando un entorno de pruebas ya pre-configurado: **Karma**, un *ejecutador* de pruebas, que se programan en **Jasmine**

**Pruebas e2e**

Son de gran utilidad:  
permiten simular de forma automatizada las interacciones del usuario

## Herramientas de terceros incluidas en Angular-cli



Del arranque de *Karma* y *Protractor* se ocupan los comandos (scripts) de npm

- *Karma*: se ocupa el comando de **npm test**  
(se ejecuta directamente igual que npm start)
- *Protractor* se inicia con el comando e2e,  
que al no ser estándar se inicia con **npm run e2e**

## Definición: Aspectos generales de Jasmine

Ficheros `.spec` creados automáticamente por el cli para los componentes y servicios utilizando la sintaxis de Jasmine

```
describe ("Descripción de la suite de pruebas", function ()
{
    it ("Descripción de la prueba 1, function(): boolean {}")
    it ("Descripción de la prueba 2, function(): boolean {}")
})
```

En las funciones `it` se utiliza el método `expect("expresion")` que define las expectativas de la prueba sobre una determinada expresión o variable

al resultado puede concatenársele la definición de la evaluación a realizar gracias al conjunto de métodos soportados en el *framework* Jasmine

```
expect(variable).toEqual(valorEsperado)
```

Previamente al `it`, para preparar la prueba se utiliza el método `beforeEach`, que recibe una función que se encargará de preparar las pruebas

```
beforeEach(function() {})
```

### Ejemplo



Pruebas *it* incluida por defecto en todos los componentes  
se limita a comprobar que el componente existe

# Elementos específicos de Angular

viernes, 26 de enero de 2018 18:36

**TestBed** -> Lecho o "camilla" de la prueba: un módulo de pruebas, equivalente a cualquier módulo, pero específico para las pruebas

En un *BeforeEach* se incluye la configuración de este módulo, gracias al método *configureTestingModule()*

Para completar la pre-configuración de Angular suele ser necesario que el módulo de pruebas sea muy similar al módulo real del componente

En un fichero *spec* de un componente existirán dos variables:

- *component* del tipo del componente asociado
- **fixture** (accesorios) del tipo *ComponentFixture* y el genérico (subtipo) del componente asociado

```
let component: MiComponent;  
let fixture: ComponentFixture<MiComponent>;
```

En un segundo *BeforeEach* se instancian

```
beforeEach(() => {  
  fixture = TestBed.createComponent(TestComponent);  
  component = fixture.componentInstance;  
  fixture.detectChanges();  
});
```

la fixture, creando en el *TestBed* un componente de la clase que estamos provando

el componente en sí mismo a partir de la fixture

Estas fixture son una forma de manipular la creación de un componente, por ejemplo dando valor a las propiedades de tipo input del componente

# Creación de pruebas específicas

domingo, 28 de enero de 2018 17:01

## Acceso a un elemento del DOM

Existe un componente con la siguiente plantilla

```
template: `<p id="test" destacar></p>`
```

Gracias a la propiedades de la fixture se puede acceder al elemento del DOM que será renderizado partir de la plantilla

```
import { By } from '@angular/platform-browser';
import { DebugElement } from '@angular/core';

let elem: DebugElement;
elem = fixture.debugElement.query(By.css('#test'));
```

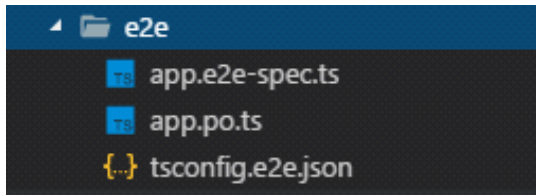
Existen varias opciones para *By* siendo la más común utilizar selectores CSS:

- id
- selector de etiqueta
- selector de atributo
- pseudoclases de posición...

# Pruebas e2e

domingo, 28 de enero de 2018

20:53



Se agrupan todas en la carpeta e2e en la raíz del proyecto

- fichero de expectativas
- fichero de pruebas
- fichero de configuración

Muy similar al que utiliza Karma:

- suite de pruebas
- preparación: define la página a probar
- funciones it: en cada caso de invoca una función definida en el fichero de pruebas

Define las pruebas concretas

- accede al DOM
- simula la interacción con el usuario

## app.e2e-spec.ts

```
import { AppPage } from './app.po';
describe('base-app App', () => {

  let page: AppPage;

  beforeEach(() => {
    page = new AppPage();
  });

  it('should display app title', () => {
    page.navigateTo();
    expect(page.getTitle()).toEqual('Angular Avanzado!');
  });

  it('should display app footer', () => {
    page.navigateTo();
    expect(page.getFooter()).toBeTruthy();
  });
});
```

## app.po.ts

```
import { browser, by, element } from 'protractor';
export class AppPage {

  navigateTo() {
    return browser.get('/');
  }

  getTitle() {
    return element(by.css('app-root h1')).getText();
  }

  getFooter() {
    return element(by.css('app-root footer')).getText();
  }

}
```



## Librerías de componentes: UI

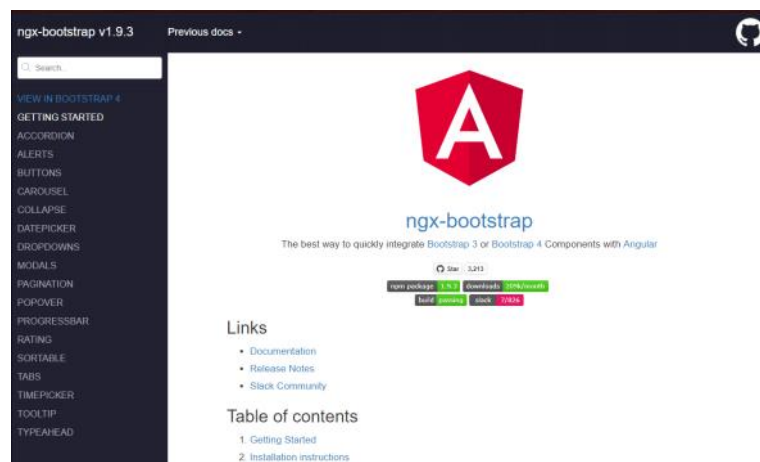
jueves, 14 de septiembre de 2017 23:00

- Múltiples ejemplos
- Distintos *widgets* hechos en JS convertidos en componentes de Angular

Entre las primeras en aparecer, la versión de Bootstrap realizada por Valor Software

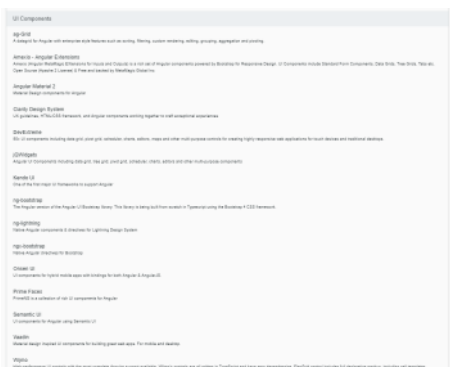


<https://valor-software.com/ngx-bootstrap/#/>



Más información en la Web oficial de Angular

<https://angular.io/resources>



## UI Components

### ag-Grid

A datagrid for Angular with enterprise style features such as sorting, filtering, custom rendering, editing, grouping, aggregation and pivoting.

### Amexio - Angular Extensions

Amexio (Angular MetaMagic EXtensions for Inputs and Outputs) is a rich set of Angular components powered by Bootstrap for Responsive Design. UI Components include Standard Form Components, Data Grids, Tree Grids, Tabs etc. Open Source (Apache 2 License) & Free and backed by MetaMagic Global Inc

### Angular Material 2

Material Design components for Angular

### Clarity Design System

UX guidelines, HTML/CSS framework, and Angular components working together to craft exceptional experiences

### DevExtreme

50+ UI components including data grid, pivot grid, scheduler, charts, editors, maps and other multi-purpose controls for creating highly responsive web

applications for touch devices and traditional desktops.

#### **jqWidgets**

Angular UI Components including data grid, tree grid, pivot grid, scheduler, charts, editors and other multi-purpose components

#### **Kendo UI**

One of the first major UI frameworks to support Angular

#### **ng-bootstrap**

The Angular version of the Angular UI Bootstrap library. This library is being built from scratch in Typescript using the Bootstrap 4 CSS framework.

#### **ng-lightning**

Native Angular components & directives for Lightning Design System

#### **ngx-bootstrap**

Native Angular directives for Bootstrap

#### **Onsen UI**

UI components for hybrid mobile apps with bindings for both Angular & AngularJS.

#### **Prime Faces**

PrimeNG is a collection of rich UI components for Angular

#### **Semantic UI**

UI components for Angular using Semantic UI

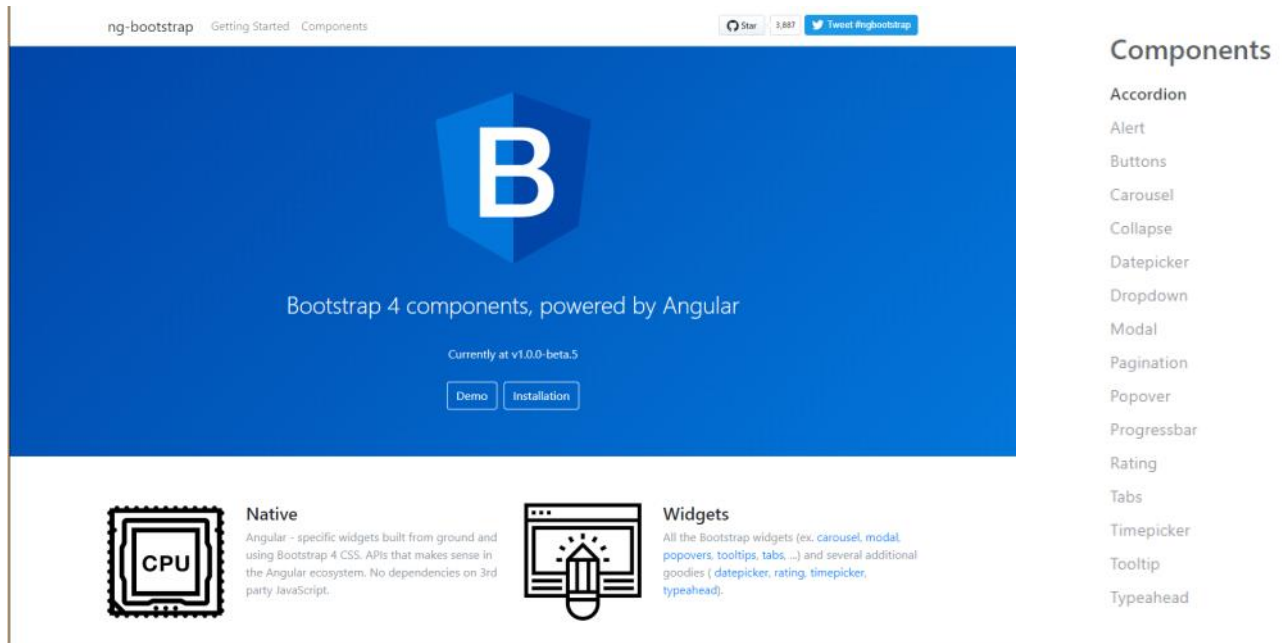
#### **Vaadin**

Material design inspired UI components for building great web apps. For mobile and desktop.

#### **Wijmo**

High-performance UI controls with the most complete Angular support available. Wijmo's controls are all written in TypeScript and have zero dependencies. FlexGrid control includes full declarative markup, including cell templates.

<https://ng-bootstrap.github.io/#/home>



## Instalación

Se instalan mediante npm

```
npm install @ng-bootstrap/ng-bootstrap
```

## Configuración

Para utilizarlo, se importa en el módulo principal, ejecutando el método `forRoot()`:

```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
```

```
@NgModule({
  ...
  imports: [NgbModule.forRoot(), ...],
```

Y se importa normalmente en otros módulos que tengan que utilizar los componentes

```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
```

```
@NgModule({
  ...
  imports: [NgbModule, ...],
```

```
...
```

## Ejemplo de Utilización

En el Módulo `about`, se añade un componente `info`, que utiliza a su vez el componente "acordeón" de `ng-bootstrap`

Componentes

- `ngb-accordion`
- `ngb-panel`

```
<ngb-accordion #acc="ngbAccordion" activeIds="ngb-panel-0">
  <ngb-panel title="Información">
    <ng-template ngbPanelContent>
      Anim pariatur cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. 3 wolf moon officia aute, non cupidatat skateboard dolor brunch. Food truck quinoa nesciunt laborum eiusmod. Brunch 3 wolf moon tempor.
```

ngb-accordion

- ngb-panel

Componente de Angular

- ng-template

Ejemplo del uso de font awesome

```

<ng-template ngbPanelContent>
  Anim pariatum cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. 3 wolf moon officia aute, non cupidatat skateboard dolor brunch. Food truck quinoa nesciunt laborum eiusmod. Brunch 3 wolf moon tempor, sunt aliqua put a bird on it squid single-origin coffee nulla assumenda shoreditch et. Nihil anim keffiyeh helvetica, craft beer labore wes anderson cred nesciunt sapiente ea proident. Ad vegan excepteur butcher vice lomo. Leggings occaecat craft beer farm-to-table, raw denim aesthetic synth nesciunt you probably haven't heard of them accusamus labore sustainable VHS.
</ng-template>
</ngb-panel>

<ngb-panel>
  <ng-template ngbPanelTitle>
    <span><i class="fa fa-star" aria-hidden="true"></i>
    <b>Lo más destacado</b>
    <i class="fa fa-star" aria-hidden="true"></i>
    </span>
  </ng-template>
  <ng-template ngbPanelContent>
    Anim pariatum cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. 3 wolf moon officia aute, non cupidatat skateboard dolor brunch. Food truck quinoa nesciunt laborum eiusmod. Brunch 3 wolf moon tempor, sunt aliqua put a bird on it squid single-origin coffee nulla assumenda shoreditch et. Nihil anim keffiyeh helvetica, craft beer labore wes anderson cred nesciunt sapiente ea proident. Ad vegan excepteur butcher vice lomo. Leggings occaecat craft beer farm-to-table, raw denim aesthetic synth nesciunt you probably haven't heard of them accusamus labore sustainable VHS.
  </ng-template>
</ngb-panel>
<ngb-panel title="Más información">
  <ng-template ngbPanelContent>
    Anim pariatum cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. 3 wolf moon officia aute, non cupidatat skateboard dolor brunch. Food truck quinoa nesciunt laborum eiusmod. Brunch 3 wolf moon tempor, sunt aliqua put a bird on it squid single-origin coffee nulla assumenda shoreditch et. Nihil anim keffiyeh helvetica, craft beer labore wes anderson cred nesciunt sapiente ea proident. Ad vegan excepteur butcher vice lomo. Leggings occaecat craft beer farm-to-table, raw denim aesthetic synth nesciunt you probably haven't heard of them accusamus labore sustainable VHS.
  </ng-template>
</ngb-panel>
</ngb-accordion>

```

Angular Avanzado!



[Inicio](#)
[Tareas](#)
[Acerca de](#)

## Acerca de

Información

Anim pariatum cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. 3 wolf moon officia aute, non cupidatat skateboard dolor brunch. Food truck quinoa nesciunt laborum eiusmod. Brunch 3 wolf moon tempor, sunt aliqua put a bird on it squid single-origin coffee nulla assumenda shoreditch et. Nihil anim keffiyeh helvetica, craft beer labore wes anderson cred nesciunt sapiente ea proident. Ad vegan excepteur butcher vice lomo. Leggings occaecat craft beer farm-to-table, raw denim aesthetic synth nesciunt you probably haven't heard of them accusamus labore sustainable VHS.

★ Lo más destacado ★

Más información

Alejandro Cerezo - Madrid, 28 ene. 2018

# Font Awesome

domingo, 28 de enero de 2018 22:07

## Instalación

npm install font-awesome

## Utilización

Ejemplo librería que proporciona elementos de interfaz basándose únicamente en la incorporación de CSS



Se añaden como CSS en `[styles]` o como `import` (igual que los CSS de Bootstrap)

```
@import "../node_modules/font-awesome/css/font-awesome.min.css";
```

Se insertan los iconos como elementos HTML vacíos, e.g. `<i></i>` a los que se aplica la clase adecuada

```
<i class="fa fa-star" aria-hidden="true"></i>
```

[GET STARTED](#)
[THEMES](#)
[SUPPORT](#)

- Input
- Button
- Data
- Panel
- Overlay
- File
- Menu
- Charts
- Messages
- Multimedia
- DragDrop
- Misc

## The Most Complete User Interface Suite for Angular

GET STARTED

### Why PrimeNG?

Congratulations! 🎉 Your quest to find the UI library for Angular is complete.

PrimeNG is a collection of rich UI components for Angular. All widgets are open source and free to use under MIT License. PrimeNG is developed by [PrimeTek Informatics](#), a vendor with years of expertise in developing open source UI solutions. For project news and updates, please follow us on [twitter](#) and visit our [blog](#).

#### 70+ COMPONENTS

The most complete set of native widgets featuring 70+ easy to use components for all your UI requirements.

#### OPEN SOURCE

Hosted at [GitHub](#), all widgets are open source and free to use under MIT license. Feel the power of open source.

#### PRODUCTIVITY

Allocate your valuable time on business logic rather than dealing with the complex user interface requirements.

<https://material.angular.io/>



Sprint from Zero to App

Hit the ground running with comprehensive, modern UI components that work across the web, mobile and desktop.

Component Categories

Form Controls	Navigation	Layout
Buttons & Indicators	Popups & Modals	Data table

# Gráficos: ng2-chart

domingo, 28 de enero de 2018 21:59

<https://valor-software.com/ng2-charts/>

Ejemplo de librería que proporciona una serie de directivas, que aplicadas a la etiqueta *canvas* permiten generar diversos tipos de gráficos

## Instalación

npm install ng2-charts

## Configuración

La dependencia con "chart.js" obliga a incluir una referencia explícita a este. Para ello se utiliza [scripts] en angular-cli.json

```
"scripts": [  
  "../node_modules/chart.js/dist/Chart.bundle.min.js"  
],
```

## Utilización.

Como ejemplo, se crea un componente en el módulo inicio para que muestre un gráfico.





## i18n en Angular

Configuración de los parámetros de i18n para usar correctamente pipes como *Date* (o *Currency*)

```
import { LOCALE_ID, NgModule } from '@angular/core';
import { registerLocaleData } from '@angular/common';
import localeEs from '@angular/common/locales/es';

registerLocaleData(localeEs);

providers: [ { provide: LOCALE_ID, useValue: 'es' } ],
...

```

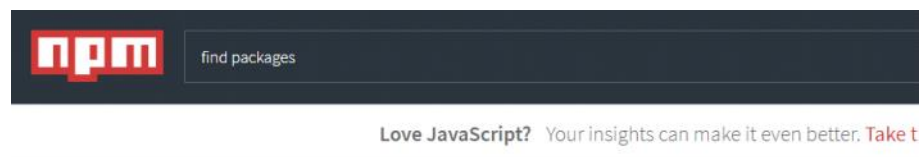
### Ejemplo

Modificamos el *footer* para que utilice una variable de tipo *Date()* con el formato adecuado

Alejandro Cerezo - Madrid, 28 ene. 2018

## Librería de traducción

<https://www.npmjs.com/package/ng2-translate>



Librería para facilitar la traducción de la aplicación

### Instalación

```
npm install ng2-translate
```

### ★ ng2-translate public

An implementation of angular translate for Angular 2.

Simple example using ng2-translate: <http://plnkr.co/edit/btpW310jr5beJVjohy1Q?p=preview>

Get the complete changelog here: <https://github.com/ocombe/ng2-translate/releases>

- Installation
- Usage
- API
- FAQ
- Plugins
- Additional Framework Support

### Configuración

```
import { TranslateModule, TranslateLoader, TranslateStaticLoader } from 'ng2-translate';
import { HttpClientModule, Http } from '@angular/http';

@NgModule({
  ...
  imports: [
    ...
    TranslateModule.forRoot(
      {
        provide: TranslateLoader,
        useFactory: tralationFactory,
        deps: [Http]
      }
    ),
  ],
})

```

Declara como *provider* un servicio incluido en la librería

Función que se declara en el módulo con los datos de la configuración

```

provide: TranslateLoader,
useFactory: tralationFactory,
deps: [Http]
}),
...

```

con los datos de la configuración

Tipo de acceso a los  
datos de traducción

```

export function tralationFactory(http: Http) {
  return new TranslateStaticLoader(http, '/assets/i18n', '.json');
}

```

Establece la localización de la carpeta en la que se podrá  
acceder via Http a los ficheros *json* con el diccionario de  
traducción específico de cada idioma

Fichero con los datos para un idioma, en esta caso en .json

```

{
  "Inicio": "Home" ,
  "Tareas": "ToDo",
  "Acerca de": "About"
}

```

El servicio `TranslateService` se inyecta en el componente principal y se utiliza  
para definir el idioma en que se renderizará la aplicación

```

import { TranslateService } from 'ng2-translate';

constructor(public translate: TranslateService) {
  this.translate.use('en');
}

```

Directiva responsable de indicar que elementos de la aplicación deben traducirse,  
en esta caso las opciones del menú

```

<a class="nav-link" [routerLinkActive]="['active']" routerLink="inicio"
  translate>Inicio</a>

```

# Selección de Idioma

domingo, 28 de enero de 2018 22:34

Añadimos la opción de seleccionar dinámicamente el idioma

```
import { TranslateService } from 'ng2-translate';

constructor(public translate: TranslateService) {
  this.aIdiomas = [
    {name: 'Español', code: 'es'},
    {name: 'Inglés', code: 'en'},
    {name: 'Francés', code: 'fr'}
  ]
  this.selectIdioma = {name: 'Español', code: 'es'};
  this.translate.use(this.selectIdioma.code);
}
```

Idiomas posibles

Idioma establecido a partir de uno de los posibles

Método manejador del evento de cambio en el select/options

```
seleccionarIdioma() {
  this.translate.use(this.selectIdioma.code);
}
```

Idioma establecido a partir de la selección del usuario

HTML del select/options

```
<div class="form-group">
  <label for=""></label>
  <select class="form-control" name="idioma" id="idioma"
    [(ngModel)] = "selectIdioma" (change)="seleccionarIdioma()">
    <option *ngFor="let idioma of aIdiomas" [ngValue]="idioma">{{idioma.name}}</option>
  </select>
</div>
```

# Refactorización

domingo, 28 de enero de 2018

23:03

Modelo de datos maestro: fichero maestros.models.ts

- Interface
- Clase

Datos correspondientes al modelo: fichero maestros.data.ts

Componente idioma con el template del HTML

- Idioma seleccionado como @output
- Manejador del evento incluido en el componente

# Más allá de los componentes

lunes, 27 de noviembre de 2017 22:53

## Tipos de directivas

Estructurales: alteran la estructura del DOM.

De atributo: alteran la apariencia o comportamiento de un elemento de Componentes

El decorador `@Component` hereda del decorador `@Directive`, ampliando su funcionalidad

# Directivas propias

lunes, 27 de noviembre de 2017 22:53

<https://angular.io/guide/attribute-directives>

<https://www.codementor.io/christiannwamba/build-custom-directives-in-angular-2-jlqrk7dpw>

<https://www.concretepage.com/angular-2/angular-2-custom-directives-example>

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appPrueba]',
})
export class PruebaDirectiva {
  constructor(private eTarget: ElementRef) {}
}
```

El selector tiene el formato de atributo, para que la directiva sea utilizada como tal

el parámetro eTarget es el elemento al que se aplica la directiva que es inyectado el aquella. Equivale a \$('selector') sobre el elemento. Al ser de tipo ElementRef, incluye la propiedad nativeElement que corresponde al elemento en si

Una vez creada, la directiva se utiliza como cualquier otra directiva de Angular

```
<p appPrueba>Contenido del párrafo</p>
```

## Atributos

Se definen como en cualquier componente, decorando una propiedad de la clase con el decorador @Input. Lo habitual es que la propiedad corresponda al propio nombre de la directiva

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appPrueba]',
})
export class PruebaDirectiva {

  @Input() appPrueba: string;

  constructor(private eTarget: ElementRef) {}
}
```

## Eventos

El decorador @HostListener (<nombre de evento>) se puede aplicar a un método para convertirlo en manejador del mencionado evento

# Pipes propios

lunes, 27 de noviembre de 2017 22:53

Directiva Pipe  
- Metadato: name

Implementa el interface  
PipeTransform, que supone  
crear una función transform

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'prueba'
})
export class PruebaPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    // código que modifica value
    return value
  }
}
```

- al menos un argumento
- devuelve el resultado de modificarlo de acuerdo con la lógica de cada filtro concreto

Pipes "puros": la función transform SOLO recibe como argumento el valor que tiene que modificar

Pipes "con atributos": la función transform recibe más argumentos, que de alguna forma determinansn como se realiza la transformación

## Pipe puro

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'capitalizar'
})
export class CapitalizarPipe implements PipeTransform {

  transform(text: string): string | null {
    if (text != null) {
      return text.substring(0, 1).toUpperCase() +
        text.substring(1);
    }
    return text;
  }
}
```

## Pipes "con atributos"

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'truncar'
})
export class TruncarPipe implements PipeTransform {
  transform(value: string, limit: number = 10): any {
    return (value.length > limit) ? value.substr(0, limit) + '...' : value;
  }
}
```

Argumento secundario (limit), en este caso con un valor por defecto definido en la forma de ES6

## Alternativas a Pipes nativos

domingo, 28 de enero de 2018 23:20

Existe un pipe en Angular, `titleCase`, cuyo funcionamiento en castellano es incorrecto

Ejemplo del uso de un pipe propio: **titularizar**  
frente a la directiva nativa `titlecase`

El Señor De Los Anillos  
El SeÑOr De Los Anillos

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'titularizar'
})
```

```
export class TitularizarPipe implements PipeTransform {
```

```
  transform(pTexto: string): any {
    if (pTexto.length === 0) {
      return pTexto;
    }
```

```
    const aCaracteres = pTexto.split('');
```

```
    aCaracteres[0] = aCaracteres[0].toUpperCase();
```

```
    for (let i = 0; i < aCaracteres.length - 2; i++) {
```

```
      //
      if (aCaracteres[i] === ' ' ||
          aCaracteres[i] === '.' ||
          aCaracteres[i] === ',')
```

```
      {
        aCaracteres[i + 1] = aCaracteres[i + 1].toUpperCase();
```

```
      }
    }
    return aCaracteres.join('');
```

```
  }
```

si no tenemos realmente un  
string no continuamos

mediante el método `split` del objeto  
wrapper `String`, obtenemos un array de  
caracteres correspondiente a la cadena

Mediante el uso del método `toUpperCase` del  
objeto wrapper `String`, podremos obtener el  
carácter en mayúscula del primer elemento

Analizamos el resto de la cadena,  
recorriendo todo el array el `-2` es para  
evitar una excepción al salirnos del array

Si es fin de 'palabra'

Reemplazamos el siguiente  
elemento por su mayúscula

Finalmente, retornamos el string creado a partir  
del array con el método `join` del objeto `Array`



# Inyección de dependencias

martes, 5 de diciembre de 2017 13:42

La Inyección de Dependencias (DI) es un mecanismo para proporcionar nuevas instancias de una clase con todas aquellas dependencias que requiere plenamente formadas.

La mayoría de dependencias son servicios, y Angular usa la DI para proporcionar nuevos componentes con los servicios que necesitan.

En cada componente, se pueden indicar en el constructor todos aquellos servicios que necesita. A nivel interno, cuando Angular crea un componente, antes de crearlo obtiene de un *Injector* esos servicios de los que depende el componente.

# Inyector

viernes, 19 de enero de 2018 18:11

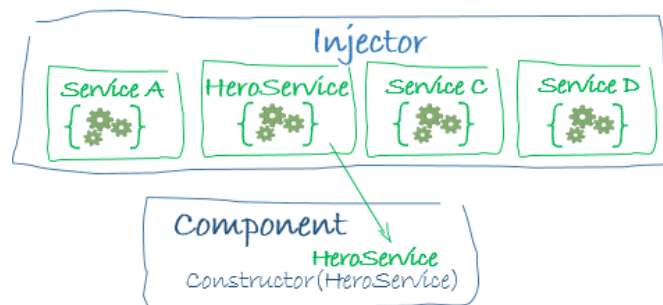
El Inyector es el principal mecanismo detrás de la DI.

A nivel interno, un inyector dispone de un **contenedor** con las **instancias de servicios** que crea él mismo.

- Si una instancia no está en el contenedor, el inyector **crea una nueva** y la añade al contenedor antes de devolver el servicio a Angular.
- Cuando todos los servicios de los que depende el contenedor se han resuelto, Angular puede llamar al constructor del componente, al que le pasa las instancias de esos servicios como argumento.

Dicho de otro modo, **la primera vez** que se inyecta un servicio, **el inyector lo instancia** y lo guarda en un contenedor. Cuando inyectamos un servicio, antes de nada el inyector busca en su contenedor para ver si ya existe una instancia.

Ese es el motivo por el que en Angular los servicios son **singletons**, pero solo dentro del **ámbito de su inyector**, sin olvidar que podemos tener inyectores a distintos niveles.



Cuando el inyector no tiene el servicio que se le pide, sabe cómo instanciar uno gracias a su **Provider**.

# Provider

viernes, 19 de enero de 2018 18:22

En Angular el ***provider*** es la propia clase que define el servicio.

Los *providers* pueden registrarse en cualquier nivel del árbol de componentes de la aplicación a través de los metadatos de componentes, a nivel de un módulo completo, en los metadatos de *NgModule*, o a nivel raíz, en el módulo principal de la aplicación, .

Como el inyector utiliza el *provider* cuando necesita instanciar un servicio, el nivel en el que se registra el *provider* determina a que nivel será *singleton* la instanciación

- Al registrar un *provider* en el *NgModule* del módulo principal , éste estará disponible para toda la aplicación instanciándose de forma *singleton* en toda ella.
- Si un servicio que se necesita declarar solo afecta a una pequeña parte de la aplicación, como puede ser un componente o un componente y sus hijos, tiene más sentido declararlo a nivel de componente.

# Registro en componentes

viernes, 19 de enero de 2018 18:33

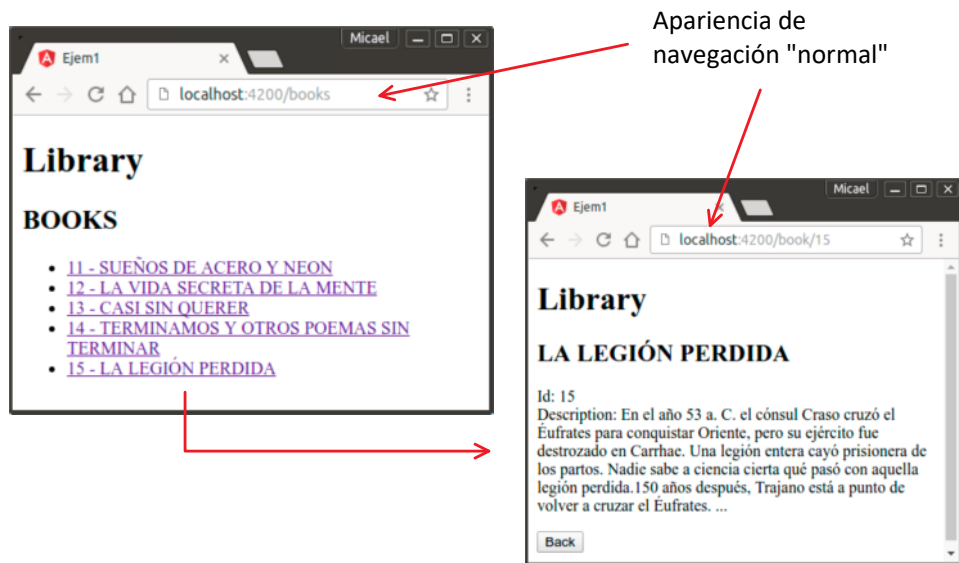
# Registro en módulos

viernes, 19 de enero de 2018 18:33

# Enrutamiento

jueves, 14 de septiembre de 2017 20:36

Las webs SPA (*single page application*) pueden tener varias pantallas simulando la navegación por diferentes páginas



<https://angular.io/docs/ts/latest/guide/router.html>



<https://vsavkin.com/>

## Principios generales

- El componente principal de la aplicación (*app-root*) tiene una parte fija (cabecera, footer) y una parte cuyo contenido depende de la URL "salida" (*<router-outlet>*)
  - En *app.routing.ts* se define qué componente se muestra para cada URL, es decir las "rutas"

- Existen varias formas de recorrer la aplicación (navegar) :
  - Desde la URL indicada al navegador, escribiendo la ruta correcta
  - Desde los links específicos para navegar dentro de la aplicación web (*[routerLink]*)
  - Desde el código, de forma programática, gracias al método (*Router.navigate*)

# Parámetros

jueves, 14 de septiembre de 2017 21:45

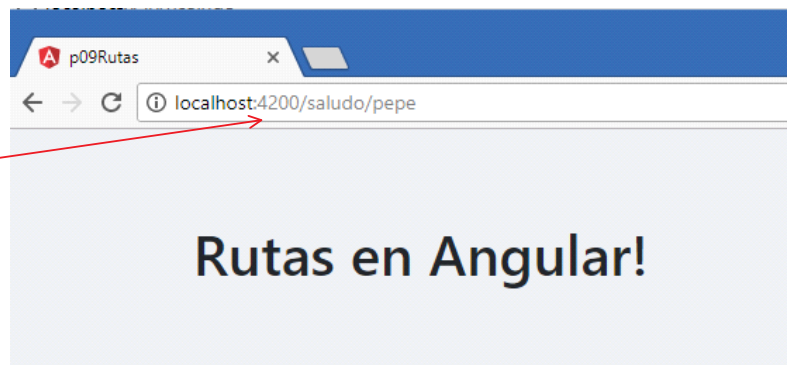
Una ruta puede incluir parámetros de forma estática  
En la constante Routes aparecerá como

```
{  
  path: 'saludo/:amigo',  
  component: SaludoComponent  
},  
}
```

El path incluye junto a si nombre la referencia a una parte variable

El componente correspondiente tendrá que ser capaz de recoger esa parte variable, como los parámetros que acompañan a la ruta

La URL para acceder a ella incluirá el valor asignado al parámetro de entrada



El componente indicado en la ruta accede al parámetro a través del servicio `ActivatedRoute`. donde existe una propiedad **params** de tipo **Observable**, que puede ser accedida de dos maneras

mediante una instantánea del estado del servicio, denominada **snapshot** que incluye el objeto (array asociativo) con cada uno de los parámetros

```
const user = this.activatedRoute.snapshot.params['amigo'];
```

declarando un observable que corresponde a la propiedad **params** y suscribiéndose a él para recoger los valores de cada parámetro concreto

```
let user;  
const user$: Observable<any> = this.activatedRoute.params;  
user$.subscribe ((parametros) => {  
  user = parametros['amigo'] || 'amigo';  
});
```

## Parámetros dinámicos en las URL

En vez de href, los links usan `[routerLink]`.

La URL se puede indicar

- como un *string* (completa)
- como un array de *strings* si hay parámetros

```
<a [routerLink]="['/enlaces', enlace.id]">
```



En el siguiente ejemplo se generan en un `*ngFor` enlaces específicos a cada uno de los libros incluidos en un array, donde cada ítem incluyen el id y el título de un libro

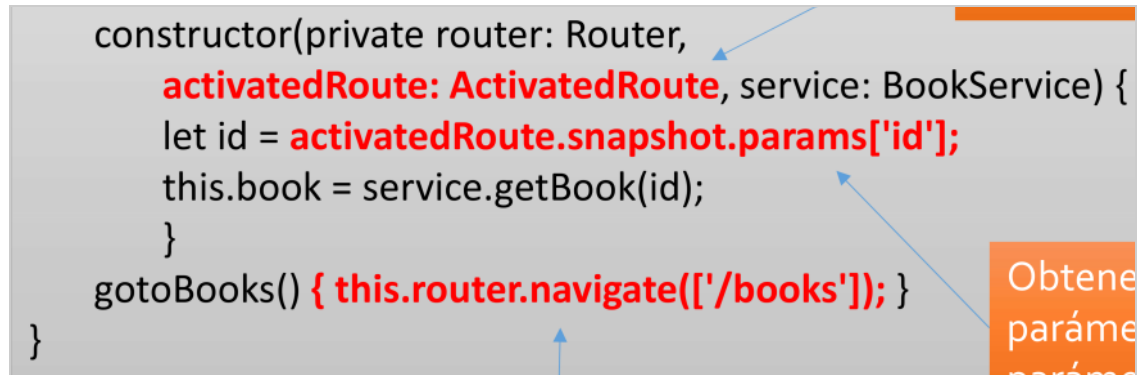
```
<a [routerLink]="['/book', book.id]">
  {{book.id}}-{{book.title}}
</a>
```

# Navegar desde el código

jueves, 7 de diciembre de 2017 10:45

Para navegar de forma programática o imperativa (desde código) usamos la dependencia Router y el método navigate.

```
constructor(private router: Router,
  activatedRoute: ActivatedRoute, service: BookService) {
  let id = activatedRoute.snapshot.params['id'];
  this.book = service.getBook(id);
}
gotoBooks() { this.router.navigate(['/books']); }
```



Obtene  
paráme  
paráme

## Carga perezosa de módulos: *lazy loading*

miércoles, 27 de septiembre de 2017 20:32

Cambia la definición de rutas en el módulo principal

```
import { RouterModule, Routes } from '@angular/router/';
```

```
const routes: Routes = [  
  {  
    path: '',  
    loadChildren: '../home/home.module#HomeModule'  
  },  
  {  
    path: 'about',  
    loadChildren: '../about/about.module#AboutModule'  
  }  
];
```

```
@NgModule({  
  imports: [  
    ...  
    RouterModule.forRoot(routes),  
  ],  
})
```

Cada objeto definidor de una ruta sustituye la propiedad `component` por `loadChildren`, que tiene como valor un *string* (por tanto no carga inicialmente el módulo al que hace referencia) con el formato: `<path del módulo>#<nombre del módulo>`

Se ejecuta el método `forRoot` de `RouterModule`

Cada módulo tiene su propio enrutador, que será realmente el que se encargue de cargar el componente

```
import { RouterModule, Routes } from '@angular/router/';
```

```
const routes: Routes = [  
  {  
    path: '',  
    component: AboutComponent  
  }  
];
```

```
@NgModule({  
  imports: [  
    ...  
    RouterModule.forChild(routes)  
  ],  
})
```

El objeto definidor de la ruta raíz del módulo hace referencia al componente principal que habría sido referenciado en el router "padre"

Se ejecuta el método `forChild` de `RouterModule`

En el módulo principal NO se referencian los módulos enrutados, de forma que no carguen al principio sino cuando son utilizados por primera vez.

```
// No puede existir referencia a aquellos módulos que deben cargarse de forma Lazy  
// import { HomeModule } from '../home/home.module';  
// import { AboutModule } from '../about/about.module';
```

```
// Componentes del Módulo  
import { AppComponent } from './app.component';  
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
    SharedModule  
  ],  
})
```

No se referencian los módulos enrutados "perezosamente"

# Funciones avanzadas

jueves, 14 de septiembre de 2017 21:48

- Rutas para un componente concreto (no para toda la app)
- Ejecutar código al salir de una pantalla
  - Si el usuario navega a otra página “sin guardar” se le puede preguntar si realmente desea descartar los cambios o abortar la navegación
- Verificar si se puede ir a una nueva pantalla
  - Generalmente se comprueba si el usuario tiene permisos para hacerlo : *Guardians*
- Animaciones

## Guardias

Son servicios que implementan alguno de los interfaces soportados por las guardias

*CanActivate to mediate navigation to a route.*

*CanActivateChild to mediate navigation to a child route.*

*CanDeactivate to mediate navigation away from the current route.*

*Resolve to perform route data retrieval before route activation.*

*CanLoad to mediate navigation to a feature module loaded asynchronously.*

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';

@Injectable()
export class AuthGuard implements CanActivate {
  canActivate() {
    console.log('La guardia AuthGuard#canActivate ha sido invocada');
    return true;
  }
}
```

Estos servicios se asocian a una ruta en la constante Routes utilizando la propiedad correspondiente al interfaz implementada, e.g. `canActivate`.

```
const routes: Routes = [
  { path: 'inicio', component: MainComponent, canActivate: [AuthGuard] },
  ...
];
```



Array con las posibles guardias asignadas

# Programación reactiva

martes, 5 de diciembre de 2017 13:40

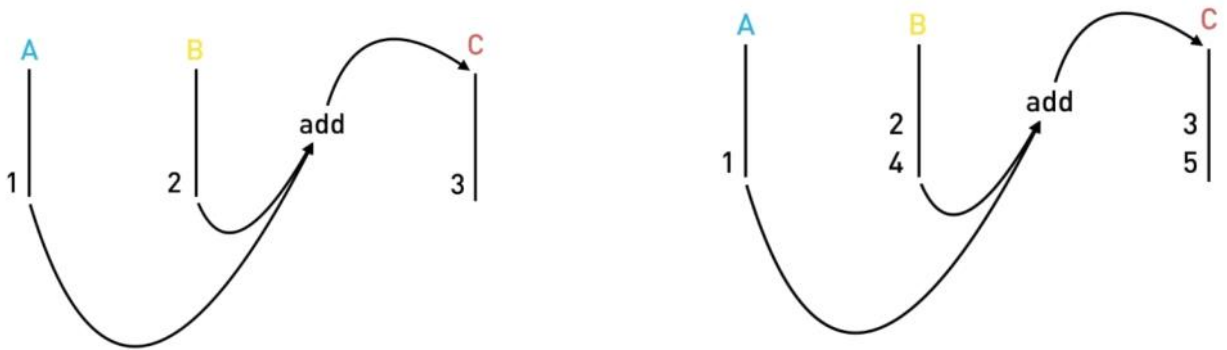
# Programación reactiva

sábado, 9 de diciembre de 2017 16:39

Se basa en el uso de **streams** ("corrientes o flujos de datos")  
asíncronicos

→ secuencias de valores a lo largo del tiempo

los **operadores** son las funciones que permiten realizar operaciones sobre estos streams



**Observables** son un nuevo tipo primitivo, que actúan como un plano (*blueprint*) o representación de cómo podemos crear *streams*, suscribirnos a ellos, responder a nuevos valores o combinar varios *streams* para construir uno nuevo.

Actualmente se está valorando la posibilidad de incorporar este nuevo tipo en el estándar ES7. Entre tanto, la librería **RxJS** (*Reactive Extensions for JavaScript*) proporciona su implementación en ES6.

Implementados en la [librería RxJS](#)

→ Extensiones reactivas para JavaScript incluidas en Angular



[ReactiveX](#)

Utilizada principalmente en

- Formularios reactivos
- Emisión de eventos
- Servicio Http

<https://codekstudio.com/post-blog/conceptos-observables-rxjs-y-angular-2-javascript-reactivo-y-funcional/57d1e2840897131b5ec54b90>

Los Observables se basan en dos patrones de programación

- 
- es el patrón "Observer"
  - el patrón "Iterator"

De esta manera no debemos pensar en arrays, eventos de ratón, llamadas http al servidor... separados, sino en algo que los agrupa a todos, el Observable. De alguna manera, cuando quieras hacer programación reactiva con un array, habrá un método para poder transformar el array en Observable y poder trabajar con él de esta manera.

la **programación reactiva** se basa en programar con **flujos de datos** (*streams*) **asíncronicos**.



proporciona una alternativa a otras formas de gestionar la asincronía, como *callbacks* o promesas

flujos de datos (*streams*):  
series de datos encadenados que pueden ser emitidos en el tiempo.

- el registro del movimiento del ratón
- los datos enviados y recibidos de una base de datos
- los *arrays* en general son también flujos de datos

## Observables

- son mecanismos creados para representar esos flujos de datos se basan en los patrones de programación "*Observer*" e "*Iterator*"

Aplicación del patrón *observer*:



Tratar todo tipo de información como un *stream* observable de entrada y de salida, al cual se le pueden agregar operaciones que procesan los flujos de datos.

- un proceso como la comunicación mediante http a un servidor es un flujo de datos que puede ser representado por un **Observable**. De esta forma toda comunicación será "vigilada" por éste.
- Se puede definir un **Observador**, es decir un elemento capaz de "mirar" a un Observable y reaccionar a los cambios que se produzcan en aquel.
- Para ello un Observador se **Suscribe** a un determinado Observable como el mencionado, para que reaccione en concreto a aquellos cambios en la comunicación con el servidor.

```
import * as Rx from 'rxjs/Rx';
```

De esta forma, *Rx* es el objeto que representa la librería RxJS ya importada en la aplicación.

Gracias a ello es posible crear Observables o transformar datos existentes, como *arrays* a Observables,

Teniendo un Observable, *RxJS* proporciona numerosas herramientas (operadores) para poder manejar ese flujo de datos, entre los que destaca el operador *map*

```
Rx.Observable  
  .from(array)  
  .map(function(element) {  
    return element + 2;  
  })  
  .filter(function(e) {  
    return e > 10;  
  });
```

Aumentamos en 2 cada elemento del flujo de datos y filtramos solo aquellos valores que son mayores que 10

Los datos son inmutables, por lo que al modificarlos se crean copias de los mismos

Al crear un observador (*Observer*), se pueden definir las respuestas a diferentes eventos del observable, como son que cambie (*onNext*), que emita un error (*onError*) o que se complete el flujo y termine su emisión (*onCompleted*).

```
const observador = Rx.Observer.create(  
  function onNext(x) { console.log('Next: ' + x); },  
  function onError(err) { console.log('Error: ' + err); },  
  function onCompleted() { console.log('Completed'); }  
);
```

*create()* es el método por defecto, por lo que puede obviarse, indicando directamente *Rx:Observer(...)*.

Por último suscribimos a nuestro Observador a nuestro Observable y de esta forma el Observable comunica al Observador sus cambios.

```
observable.subscribe(observador);
```

El array no es emitido ni manejado de ninguna manera por el Observable hasta que un Observador como mínimo se suscriba a él. Esto es importante porque de esta manera no se consumen recursos sin sentido.

Ahora, cualquier cambio, como es que se añada al array un nuevo miembro le será notificado al observador que responderá con la función "*onNext*", en la que podrá definirse la reacción adecuada en cada aplicación.



# Observables

sábado, 9 de diciembre de 2017 17:16

Importamos Observable desde rxjs/Observable.

```
import { Observable } from 'rxjs/Observable';

buscarLibrosAsyncRx(clave: string) {
  return new Observable(
    (observer) => {
      setTimeout(() => {
        observer.next(this.aLibros);
      }, 2000);
    }
  );
}
```

Instanciamos un nuevo Observable con el método Observable, que encapsula Rx.Observable.create, definiendo la función que será ejecutada cuando se produzca alguna suscripción, representada por el parámetro observer

create(obs => { obs.next(1); })

1

El equivalente en ES6 puro, usando NodeJS, sería

```
let Rx = require('rxjs');
let observable = Rx.Observable.create(
  (observer) => {
    observer
      .interval(2000)
      .next(this.aLibros);
  }
);
```

El método next() es el responsable de emitir un evento, con los datos indicados, que será recogido por el observador, que habrá sido definido con el método subscribe(). Eventos de otro tipo son emitidos mediante error() y complete()

## Observador y suscripción

```
btnBuscarRx() {
  this.aLibros = [];
  this.librosMockService.buscarLibrosAsyncRx(this.sClave)
    .map(response => response)
    .subscribe(
      (response) => {
        console.dir(response);
        this.aLibros = response;
      }
    );
}
```

método del servicio que devuelve un observable

ejemplo de los operadores que permiten manipular los observables

suscripción al observable

Una vez suscritos ejecutaremos alguna de las tres funciones definidas en función de los estados del observable:

- onNext
- onError
- onComplete

## Angular utiliza Observables en diversas situaciones

Como base de la emisión y vigilancia de **eventos**, uno de los patrones básicos en la comunicación **entre componentes**. Cuando se necesita que el resto de la aplicación conozca un cambio en un componente y reacciones al mismo, se hace uso de **EventEmitter**, que no es más que una clase de Angular que envuelve métodos de *RxJS*.

En los formularios basados en el modelo (*ModelDriven*)

En la comunicación con el servidor

```
http.get('/api/usuario')  
  .map(res: Response => res.txt)  
  .subscribe(res => this.user = res);
```

la respuesta a una solicitud http al servidor es un Observable.

Puede ser procesada mediante operadores

En lugar de crear explícitamente un Observador, se encadena directamente “subscribe” a la cadena de operadores pasándole una función.

Este modo rápido supone realmente la **creación de un Observador** en el que la primera (y única) función indicada se le asigna al evento “*onNext*”, que es el primero de los que pueden definirse.

La programación funcional pretende básicamente actuar de forma **declarativa** en lugar de imperativo, es decir indicar que es lo que quiere hacer en cada momento y no como hacerlo

cuando hemos aplicado la función “map” al array estamos diciendo “quiero crear un nuevo array pero sumando 2 a cada número”. En un modo tradicional, en Javascript tendrías que hacer un bucle para recorrer el array y luego crear un nuevo array para introducir los nuevos valores, estarías diciendo “como” hacerlo, y no “que” quieres solamente.

La programación funcional en realidad trata todo como **funciones matemáticas**. Quiero hacer esto y lo otro mediante esta y la otra función, las combino, las resto... pero además no cambia el estado ni los datos del programa, cada función opera de manera aislada y no utiliza sentencias sino declaraciones. Por ejemplo, los bucles no son buenos amigos de la programación funcional. La **inmutabilidad** de la que antes hemos hablado un poco, es un concepto principal en PF. Nunca se modifican los datos. ¿Como se consigue?, básicamente realizando copias de los mismos cuando se deben alterar.

Además los datos de salida de una función solo dependen de los argumentos que son introducidos en la función, y solo de estos, asegurando que una función siempre produce los mismos resultados **eliminado efectos colaterales**, que son efectos producidos en el exterior de una función pero producidos por ella (cambios de estado). Por ejemplo cuando una función cambia un valor de una variable global exterior. Esto resulta en algo impredecible puesto que cualquiera puede “tocar” esa variable exterior, cambiando el estado.

## Aproximaciones en el desarrollo Web

Pues bien, todo esto es para decir que realmente ni Angular 2, ni RxJS ni si quiera React ni otras muchas librerías y Frameworks que hay por ahí son programación funcional. Lo son **Hope, Haskell, Erlang o F#**, que son mayoritariamente usados en **ámbitos académicos**. Todo este rollo entonces ¿para qué?, pues porque este paradigma de programación es algo a lo que muchos nuevos lenguajes y frameworks intenta emular puesto que las ventajas son bastante importantes.

Cualquier frameworks (casi) usado hoy en día para el desarrollo web está de una manera u otra basado en programación de objetos, por lo que no tiene cabida la programación funcional. No obstante veamos el caso de Angular. Ha pasado de tener el “two way data binding” (vínculos de dos vías entre datos y vista) como piedra angular (o una de ellas), para pasar en Angular 2 a proclamar la vía única (“one way”) como santo grial, algo que React lleva en la sangre desde siempre. Es decir ahora los datos fluyen en un solo sentido y no se pueden modificar (o deben) desde diferentes lados porque no se podría razonar bien sobre ellos, perderíamos el control más fácilmente.

No obstante, dicho esto, sí que debemos de tomarnos en serio este giro hacia la programación funcional y reactiva, puesto que todo apunta a un futuro prometedor. Mi recomendación es que si ves artículos que hablen de estos temas, te los leas para ver que se cuece y entender mejor estos apasionantes conceptos que te hemos presentado.

## Formas de comunicación

Comunicación entre un componente padre (contenedor) y un componente hijo (incluido en el anterior)

- Configuración de propiedades (Padre → Hijo)
- Envío de eventos (Hijo → Padre)

- Invocación de métodos (Padre → Hijo)
  - Con variable *template*
  - Inyectando hijo con *@ViewChild*
- Compartiendo el mismo servicio (Padre ↔ Hijo)

Los inyectables (servicios) son objetos *singleton* y por tanto compartidos entre las distintas clases que los instancian

<https://angular.io/docs/ts/latest/cookbook/component-communication.html>

# Formularios reactivos

miércoles, 4 de octubre de 2017 6:54

- en lugar de `FormsModule`, se utiliza `ReactiveFormsModule`, también incluido en `@angular/forms`
- el desarrollo declarativo (en la vista es mínimo)
  - el atributo `[formGroup]` en el elemento `form`
  - el atributo `formControlName` para identificar a cada uno de los controles, en cierto modo en lugar del `[(ngModel)]`
- la gestión del formulario se traslada al controlador, donde se crea un objeto de la clase `FormGroup` para que se ocupe de ello invocándolo desde el correspondiente atributo de la vista
- Existen 2 posibilidades para instanciar ese objeto
  - Crear el objeto directamente, instanciando cada uno de sus componentes como `FormControl`
  - utiliza el método `group` del servicio `FormBuilder`, que tiene que ser inyectado como cualquier otro servicio
    - este método tiene como parámetro un objeto en el que se definen cada uno los `formControlName` de cada uno de los controles del formulario, de acuerdo con los valores asignados en la vista. Si es necesario, se puede indicar el valor inicial de los controles

```
<form [formGroup]="formLibros" (ngSubmit)="enviarFormLibros()">
  <label for="titulo">Titulo</label>
  <input type="text" id="titulo" formControlName="titulo">
  <label for="autor">Autor</label>
  <input type="text" id="autor" formControlName="autor">
  <label for="editorial">Editorial</label>
  <input type="text" id="editorial" formControlName="editorial">
  <label for="fecha">Fecha (Año)</label>
  <input type="text" id="fecha" formControlName="fecha">
  <label></label>
  <button type="submit">Enviar</button>
</form>
```

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-formulario',
  templateUrl: './formulario.component.html',
  styleUrls: ['./formulario.component.css']
})
export class FormularioComponent implements OnInit {

  // propiedad de tipo FormGroup (grupo de controles)
  // que se asociara a un formulario o subformulario (en casos complejos)
  formLibros: FormGroup;

  // Se inyecta FormBuilder para instanciar el FormGroup
  // correspondiente a la propiedad que se acaba de definir
  constructor(private FormBuilder: FormBuilder) { }
```

```

ngOnInit() {
  // Gracias al servicio FormBuilder, se instancia un FormGroup
  // pándole como parámetro el objeto con la definición del formulario
  // con los formControlNames asignados en la vista
  // forControlName="titulo"
  // forControlName="autor"
  // forControlName="editorial"
  // forControlName="fecha">
  this.formLibros = this.formBuilder.group({
    titulo: [],
    autor: [],
    editorial: [],
    fecha: ['2017']
  });
} // Fin del ngOnInit

enviarFormLibros () {}
}

```

# Validación

miércoles, 13 de septiembre de 2017 0:16

**form** → la propia etiqueta HTML está ligada a la directiva Angular **ngForm** (i.e. es su selector), por lo que se instancia automáticamente el correspondiente objeto, que permitirá conocer en todo momento el estado del formulario y de cualquiera de sus controles.

Esta instancia es oculta, pero puede ser accedida en la propia **vista** mediante una **referencia local**

```
<form novalidate (ngSubmit)="enviar()" #myform= "ngForm">
```

← referencia local que acceda a la instancia del formulario

El acceso desde el **modelo/controlador** se consigue gracias al decorador **@ViewChild**

```
@ViewChild('myform') form: any;  
...  
console.log(this.form);
```

```
▼ NgForm {_submitted: false, ngSubmit: EventEmitter, form: FormGroup} ⓘ  
  control: (...)  
  controls: (...)  
  dirty: (...)  
  disabled: (...)  
  enabled: (...)  
  errors: (...)  
  form: FormGroup {validator: null, asyncValidator: null, _onCollectionChange: f,  
    formDirective: (...)  
    invalid: (...)  
  }  
  ngSubmit: EventEmitter {_isScalar: false, observers: Array(1), closed: false, is  
    path: (...)  
    pending: (...)  
    pristine: (...)  
    statusChanges: (...)  
    submitted: (...)  
    touched: (...)  
    untouched: (...)  
    valid: (...)  
    value: (...)  
    valueChanges: (...)  
    _submitted: false  
  }  
  __proto__: ControlContainer
```

## Requerimientos y estado

Los requerimientos de validación se establecen directamente con los nuevos atributos incorporados en HTML5

- **required**: valor booleano: cuando es true marca un campo como obligatorio.
- **max**: indica el número máximo de caracteres permitidos en un campo.
- **min**: indica el número mínimo de caracteres permitidos en un campo.
- **pattern**: Valida un campo frente a una expresión regular (regex).

El estado del formulario y de cada control viene definido por el valor de una serie de propiedades

- **Untouched**: When true, the control has not been interacted with the user
- **Touched**: When true, the control has been interacted with the user
- **Pristine**: The control and its underlying model has not been changed
- **Dirty**: The control and its underlying model has been changed

Estas propiedades permiten no mostrar mensajes de validación hasta que el usuario ha comenzado a rellenar el formulario

- **Valid**: The inner model is valid
- **Invalid**: The inner model is not valid

Estas propiedades permiten determinar la validez de cualquier control para hacer visibles o no los correspondientes mensajes, e.g utilizando el atributo **hidden**.

Cuando se renderiza el HTML, aquellas propiedades que valgan true darán lugar a la aplicación de las correspondientes clases de CSS.

Estas propiedades son accesibles desde la referencia local del formulario

`myform.form.controls.firstname` ← *name* asignado a cada uno de los controles

Pero es más sencillo crear referencias locales específicas para cada control

```
<input name="firstname" ... #firstnameState="ngModel">
```

## Información al usuario

Si no se cumplen los requerimientos de validación, el navegador responderá en la forma que tenga predefinida en función de la validación HTML5. Para evitar estos mensajes se utiliza el atributo **novalidate** en la etiqueta `form`.

El siguiente paso es crear los mensajes específicos de cada situación y ocultarlos o mostrarlos en función del valor de las propiedades antes citadas. Para acceder a ellas se definen referencias locales (#) en cada uno de los controles

```
<form name="myform" novalidate (ngSubmit)="enviar()">

  <input type="text" id="firstname"
    name="firstname"
    [(ngModel)]="user.firstname"

    required="true"
    minlength="2"

    #firstnameState="ngModel">

  <!--Mensajes de validación-->
  <div class="error-message"
    [hidden]="firstnameState.valid || firstnameState.pristine">
    El nombre es obligatorio</div>
```

Anulamos la validación estándar HTML5

input con doble binding

requerimientos de validación

referencia local

condiciones en las que se oculta el mensaje de error de validación

Para cada directiva de validación existe una propiedad `errors` que tomara un valor según las circunstancias, creándose un objeto correspondiente al primero de los errores que se esté produciendo

```
{{firstnameState.errors?.required}}
{{firstnameState.errors?.minlength}}
```

Para mostrarlos se utiliza el **operador Elvis**, para que solo se intente llamar la propiedad de la derecha si la de la izquierda no es nula



## Servicio Http

### Inyección del servicio en el componente

```
import { Http } from '@angular/http';

@Component({...})
export class NameComponent implements OnInit {
  constructor(public http: Http) { }
```

Con frecuencia el proceso es algo más complejo, al estar mediado por un servicio propio del usuario que a su vez hace uso del servicio Http.

## Servicio HttpClient

Aparece en Angular 4.3, en el módulo `HttpClientModule` incluido en `@angular/common/http`, como una completa reimplementación de `HttpModule`, que en la versión 5 pasa a estar deprecado.

En el nuevo servicio se accede directamente a la respuesta en formato JSON, sin necesidad de manipulaciones previas para obtener dicho formato

### Inyección del servicio en el componente

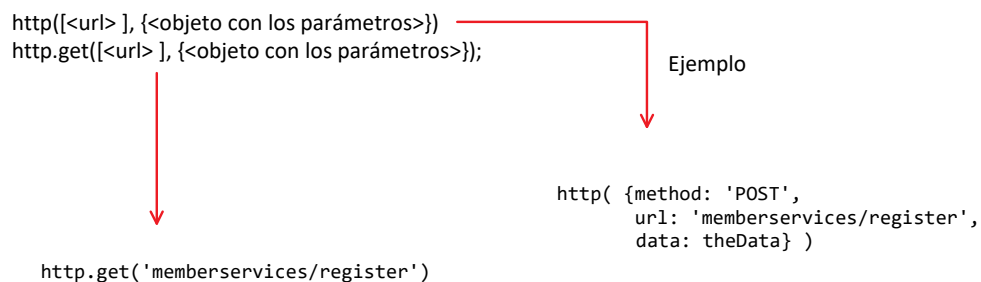
```
import { HttpClient } from '@angular/common/http';

@Component({...})
export class NameComponent implements OnInit {
  constructor(public http: HttpClient) { }
```

<https://medium.com/codingthesmartway-com-blog/angular-4-3-httpclient-accessing-rest-web-services-with-angular-2305b8fd654b>

## Consumo (uso) del servicio

Se utiliza el propio servicio o alguno de los métodos que proporciona (*get*, *post*...)



## Procesamiento de observables

jueves, 7 de diciembre de 2017 20:07

La respuesta por defecto a una petición http en cualquiera de los servicios Angular (`Http` o `HttpClient`) es un **observable**. Por tanto necesitamos **suscribirnos** a él para gestionar la respuesta.

### Respuesta al servicio `Http`

```
this.http.get(url)
  .subscribe(
    response => console.log(response.json()), // fin del metodo onNext
    error => console.error(error); // fin del metodo onError
  );
```

Una vez suscritos ejecutaremos alguna de las funciones definidas en función de los estados del observable

Si todo ha sido correcto, para obtener los datos enviados por el servidor usamos el método `json()` del objeto `response`

Siendo más correcto en el uso de la **sintaxis reactiva**

```
this.http.get(url)
  .map(response => response.json())
  .subscribe(
    response => console.log(response), // fin del metodo onNext
    error => console.error(error); // fin del metodo onError
  );
```

Utilizamos el operador `map` para aplicar una transformación al observable antes de suscribirnos a él.

### Respuesta al servicio `HttpClient`

```
this.http.get(url)
  // .map(response => response.json())
  .subscribe(
    response => console.log(response), // fin del metodo onNext
    error => console.error(error); // fin del metodo onError
  );
```

En el nuevo servicio se accede directamente a la respuesta en formato JSON, sin necesidad de manipulaciones previas para obtener dicho formato

## Servicios propios

jueves, 14 de septiembre de 2017 22:52

Al hacer una petición REST con `Http` / `HttpClient` obtenemos un objeto `Response` que debe ser procesado de acuerdo con las características del API concreto del que proceden los datos

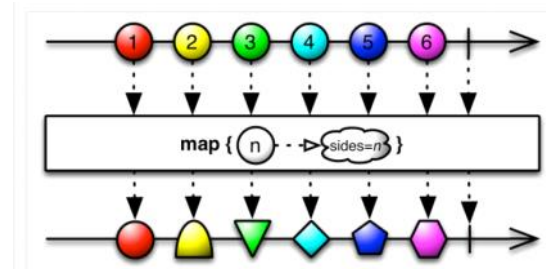
En lugar de utilizar directamente dichos servicios conviene encapsularlos en otros, capaces de ofrecer objetos de alto nivel a los clientes del servicio (e.g. el array de títulos ya procesado en el ejemplo que hemos visto)

Nuestro propio servicio realizara varias operaciones

- La consulta al API via `Http` / `HttpClient`
- La transformación del objeto `Response` en el conjunto de datos adecuado (e.g. el array de títulos) cuando llegue la respuesta
- El envío de los datos ya transformados con el mismo formato de llegada, para conservar la asincronía del proceso: un `Observable` o una `Promesa`, como los que proporcionan los servicios nativos

```
buscarRx (clave: string) {  
  const url = this.sURL + clave;  
  return this.http.get(url)  
    .map(response => this.extractTitles(response));  
}  
  
private extractTitles(response: any) {  
  if (response.items) {  
    return response.items.map(book => book.volumeInfo.title);  
  } else {  
    return response;  
  }  
}
```

el operador map



## Servicios *stateless* (sin estado)

- No guardan información
- Sus métodos devuelven valores, pero no cambian el estado del servicio
- Ejemplo: BooksService con llamadas a Google

## Servicios *statefull* (con estado)

- Mantienen estado, guardan información
- Al ejecutar sus métodos cambian su estado interno, y también pueden devolver valores
- Ejemplo: LoginService con información en memoria

## ¿*Stateless* vs *statefull*?

- Los servicios *stateless* son más fáciles de implementar porque básicamente encapsulan las peticiones REST al *backend*
- Pero la aplicación es menos eficiente porque cada vez que se visualiza un componente se tiene que pedir de nuevo la información
- Los servicios *statefull* son más complejos de implementar porque hay que definir una política de sincronización entre *frontend* y *backend*
- Pero la aplicación podría ser más eficiente porque no se consulta al *backend* constantemente

# Arquitectura de proyecto

viernes, 19 de enero de 2018 18:52

## Gestión de proyectos

Estructura de directorios  
Npm Avanzado  
Npm >= 5 - Lockfile  
Package.json  
Angular-cli  
Webpack

### 1. Extras

- a. Angular y Bootstrap (Responsiveness)
- b. Angular Material (Libreria components UI)
- c. Angular Environment
- d. ngTranslate
- e. Reusar JQuery plugin

# Patrón REDUX

viernes, 19 de enero de 2018 18:50

# Testing

viernes, 19 de enero de 2018 18:53

Testing Unitario  
Test end-to-end

# Servicios "helper"

lunes, 29 de enero de 2018 0:26