

Agenda

domingo, 4 de febrero de 2018 19:26

- Introducción a Angular
 - Conceptos y versiones
 - Entorno y Herramientas de Desarrollo
 - Editores y navegadores. Plugins
 - Angular CLI. Proyecto inicial.
 - Bootstrap y Font Awesome
 - Introducción a TypeScript
 - Arquitectura de las SPA
 - Arquitectura. Módulos y componentes
 - Enrutamiento y navegación. Lazy Loading
 - Entorno de Testing
 - Componentes, Directivas y Pipes
 - Componentes. Plantillas
 - Directivas
 - Pipes. (Pipes propios)
 - Binding y Formularios (Template Driven)
 - Aplicaciones SPA
 - Servicios. Inyectores y providers
 - Acceso al servidor. HttpClient: promesas
 - Enrutamiento y parámetros. Subrutinas
-
- Bibliotecas de componentes
 - ng-Bootstrap
 - Gráficos
 - Traducción
 - Programación reactiva
 - Observable / Subscribe
 - Operadores más comunes
 - Comunicación entre componentes
 - HttpClient
 - Formulario reactivos (Model Driven)



Proyecto

Lunes, 19 de febrero de 2018

11:40

Modificación del **prototipo de sitio Web** desarrollado previamente sobre el tema elegido

- Responsivo, incluyendo un menú replegado en la versión móvil
- Que valide correctamente el HTML5

Junto con las 2 incluidas previamente se añadirá al menos una tercera página (se pueden añadir libremente cualquier otro elemento que se considere apropiado)

- Home
- Blog
- About

Desde el punto de vista de diseño, puede mantenerse el CSS empleado previamente, distribuyéndolo adecuadamente en los componentes cuando sea necesario, o puede utilizarse un *framework* como Bootstrap.

Para las páginas previas se mantendrán los requisitos anteriores

Las páginas mostrarán los elementos comunes propios del sitio Web

- Header
- Nav
- Footer
- Se diferenciarán en el resto del contenido, que puede englobarse en un Main

La Home incluirá

- al menos 2 artículos (o el inicio de ellos) con texto y elementos gráficos (imágenes, vídeos...)
- algún tipo de información extra tangencial al contenido del sitio

La página About incluirá

- breve reseña y fotografía del autor / autores
- un formulario de contacto para enviar sugerencias en el que será obligatorio indicar un nombre y correo

- Todo el contenido será convertido en **componentes** Angular, **modularizado** de forma adecuada al diseño de la aplicación. Para todos los elementos se comprobará que se superan los test básicos creados por el sistema, tanto los test unitarios como e2e.
- Se creará una **SPA** de modo que el contenido específico de cada página sea cargado dinámicamente dentro de una zona de la única página realmente existente. La **carga** de estos componentes se realizará de forma **diferida**, sólo en el momento en el que son requeridos por la interacción del usuario.
- En el formulario incluido en la página About será obligatorio indicar un nombre y correo, este último con el formato adecuado, presentándose en caso necesario los correspondientes mensajes, basándose la **validación** en los elementos adecuados de Angular. Igualmente se solicitará algo de información de las preferencias del usuario respecto a la temática del sitio. Una vez terminado el formulario se mostrarán en pantalla los datos y se almacenarán en algún tipo de servidor.
- El **Blog** incluirá los siguientes elementos, separados en distintos componentes, de acuerdo con el modelo **controlador/presentadores**:

- un formulario para incluir nuevas entradas
 - una lista de las entradas disponibles, con algunos de sus campos
 - la posibilidad de acceder a la información completa de una entrada, cargando una nueva "página"
-
- Los datos del **Blog** se almacenarán en algún tipo de **servidor** basado en un **API REST** de modo que sean posibles las operaciones estándar de esta arquitectura: **get-post-put-delete**. Para esto puede usarse json-server, Firebase o cualquier otra opción
 - Sobre estos elementos básicos puede añadirse cualquier otra funcionalidad, e.g. nuevas páginas, que se consideren adecuadas

Ξ Angular. Primera parte

sábado, 9 de septiembre de 2017 13:22

- Introducción a Angular
 - *Frameworks en JS*
 - Presentación de AngularJS y Angular
- Tecnologías implicadas
 - Entorno de trabajo
 - Instalación e inicio. Angular cli
 - ES6
 - *TypeScript*

Navegadores.
Editores de código
Gestión de versiones. GIT
NodeJS y npm. Empaquetado

Formas de creación de proyectos
Arquitectura del proyecto (*Scaffolding*). Angular-cli

Frameworks en JS

sábado, 9 de septiembre de 2017 13:32

Bibliotecas o frameworks

- facilitan el desarrollo
- automatizan procesos
- aumentan la eficacia
- mejoran el producto

*jQuery
Underscore.js
MooTools
Prototype
Google Web Toolkit (de Java a JS)
YUI*

*Ext JS
Vue.js
SAP - OpenUI5*

AngularJS / Angular
BackboneJS
Ember.js
React.js

*AccDC
Ample SDK
Atoms.js
DHTMLX
Dojo
Echo3
Enyo
Handlebars
Kendo UI
Knockout.js
D3.js - Kinetic.js*

*Meteor
PhoneJS
Pyjamas
qooxdoo
Rialto
SmartClient & SmartGWT
Socket.IO
SproutCore
Wakanda
ZK
Webix*



- *framework estructural*
(i.e. orientado a la lógica)
- de JavaScript
- para el Desarrollo Web *Front End*



BackboneJS

<http://backbonejs.org/>



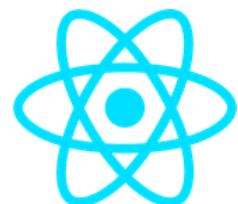
Ember.js

<http://emberjs.com/>



AngularJS

<https://angularjs.org>



React.js

<http://emberjs.com/>

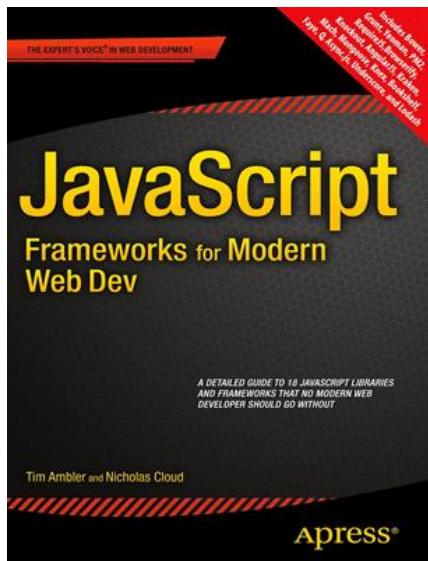
La elección de uno de ellos depende de los objetivos de cada proyecto



<http://todomvc.com/>



Frameworks interrelacionados



JavaScript Frameworks for Modern Web Dev
Tim Ambler & Nicholas Cloud
Apress, 2015

Contents at a Glance

About the Authors.....	.xx
About the Technical Reviewerxxi
Acknowledgmentsxxiii
Introductionxxv
■ Chapter 1: Bower	1
■ Chapter 2: Grunt	11
■ Chapter 3: Yeoman	37
■ Chapter 4: PM2	53
■ Chapter 5: RequireJS	73
■ Chapter 6: Browserify	101
■ Chapter 7: Knockout	121
■ Chapter 8: AngularJS	155
■ Chapter 9: Kraken	191
■ Chapter 10: Mach	251
■ Chapter 11: Mongoose	297
■ Chapter 12: Knex and Bookshelf	345
■ Chapter 13: Faye	381



Angular

Introducción a Angular

sábado, 9 de septiembre de 2017 16:48

The screenshot shows the AngularJS homepage. It features the AngularJS logo with the text "ANGULARJS by Google". Below the logo is the tagline "HTML enhanced for web apps!". There are two main calls-to-action: "Download AngularJS 1" (version 1.6.9) and "Try the new Angular 2". Below these are links to "View on GitHub" and "Design Docs & Notes". Social media links for GitHub, Facebook, and Twitter are also present. A button at the bottom says "Learn Angular in your browser for free!".

<https://angularjs.org/>

The screenshot shows the Angular website at https://angular.io/. It features the Angular logo and the tagline "One framework. Mobile & desktop.". A "GET STARTED" button is visible. At the bottom, there's a banner for "Google Developer Day Beijing & Shanghai 12/2016" with a "REGISTER NOW" button.

<https://angular.io/>

Orígenes y desarrollo

Misko Hevery

Adam Abrons



- 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**

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)**

The screenshot shows the MEAN.js website. The header includes the MEAN.js logo and navigation links for Home, Documentation, Packages, Release Notes, Support, Blog, and Contact. The main content features the tagline "The Friendly & Fun Javascript Fullstack for your next web application". Below this is a description: "MEAN is an opinionated fullstack javascript framework - which simplifies and accelerates web application development." A section titled "Get MEAN by running..." provides the command "\$ sudo npm install -g mean-cli \$ mean init yourNewApp.". At the bottom, it shows the latest release (v0.8.8), latest commit (Nov 23, 2016), and forks (2996).

Ejemplos de uso

made with NGULAR

<https://www.madewithangular.com/#/>

Communication



Education





AngularJS 1.x

- extiende directamente las funcionalidades **HTML**
 - utiliza como patrón arquitectónico **MVC** (Modelo, Vista, Controlador) o similares (estrictamente sería MV* o MVW (*Model-View-Whatever*))
 - está especialmente orientado a la creación de aplicaciones **SPA** (*Single-Page Applications*).
 - es muy eficiente: promueve el uso **patrones** de diseño de software
 - permite crear **tests unitarios** y *End-to-End* de forma sencilla empleando *Jasmine* y *Karma*
 - al estar exclusivamente orientado a la lógica, es un *framework* muy liviano: no incluye elementos gráficos ni CSS.
- Se complementa muy bien con *Bootstrap* o *Material Design*

Aplicaciones cada vez más ambiciosas

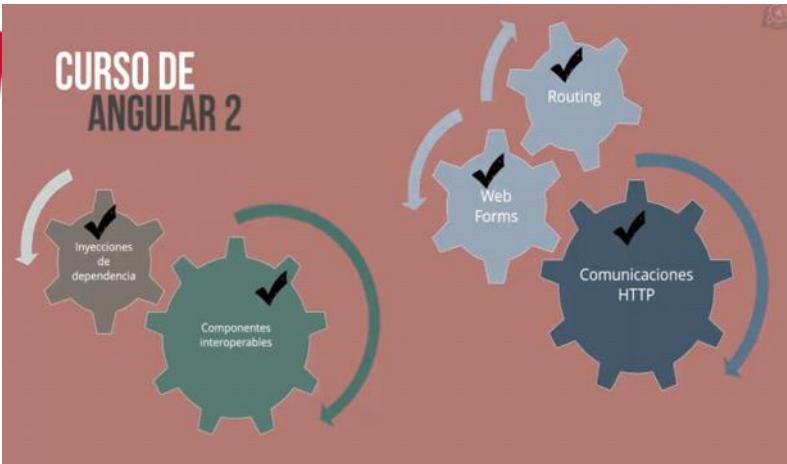


Angular 2.x
Angular 4.x
Angular 5.x

- amplia el modelo de **extender** las funcionalidades **HTML** empleando **componentes**
 - Árboles de componentes Web
 - Interconexiones entre ellos
 - Cada uno su propia interface I/O
 - Inyección de dependencias totalmente renovado
- con ello se modifica la forma de emplear el patrón arquitectónico **MVC** (Modelo, Vista, Controlador) o similares (estrictamente sería MV* o MVW (*Model-View-Whatever*))
- sigue estando especialmente orientado a la creación de aplicaciones **SPA** (*Single-Page Applications*).



CURSO DE ANGULAR 2



- Angular 2 Versión final: Septiembre 2016
Angular 4 : Abril 2017
- Está implementado desde cero no como una evolución de AngularJS
- Angular 2 no es compatible con AngularJS: no existe el `$scope`
- La documentación de AngularJS no sirve para Angular

Tabula Rasa



Funcionalidades en Angular

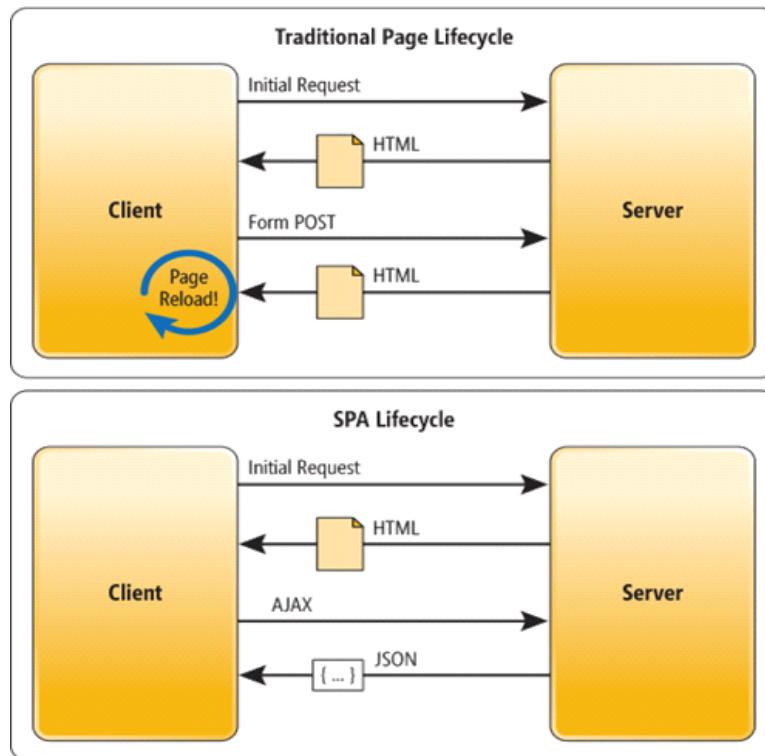
- Inyección de dependencias
- Servicios
- Cliente http (APIs REST)
- Navegación por la app (*Router*)
- Animaciones
- Internacionalización
- Soporte para tests unitarios y e2e
- Librerías de componentes
- Renderizado en el servidor (Angular Universal)



SPA: *Single-Page Applications*

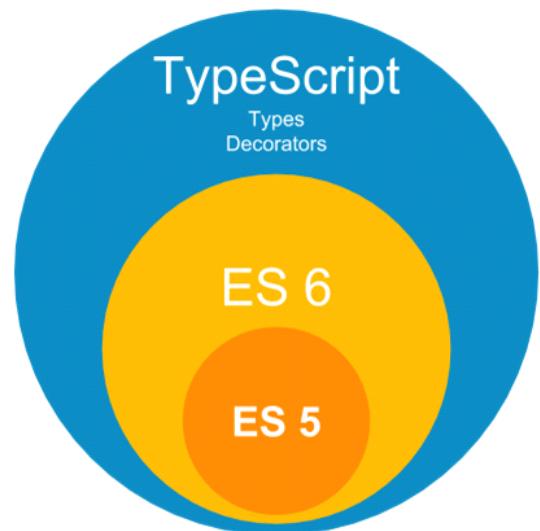
sábado, 9 de septiembre de 2017 17:13

- carga completa en **una sola página**
- **Asincronicidad**
- limitada dependencia del **servidor**
- aproximación a las **aplicaciones de escritorio**



Programación en ES6 + Typescript

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



TypeScript is a javascript superset

Transpilación

resultados en ES5 / ES6
(compatibilidad)



Características: MVC

Patrón arquitectónico (según otros autores **patrón de diseño**)
separación del código de los programas dependiendo de su responsabilidad

Se basa en las ideas claves
en el desarrollo de la
ingeniería del software

- **reutilización de código**
(*code reuse*, [Douglas McIlroy](#), 1968)
- **separación de conceptos**
(*separation of concerns*,
[Edsger W. Dijkstra](#), 1974)

Fue introducido por el científico
noruego [Trygve Reenskaug](#)
cuando trabajaba con Smalltalk-76 en el
Xerox Palo Alto Research Center (PARC)

Más tarde fue re implementado
en Smalltalk-80

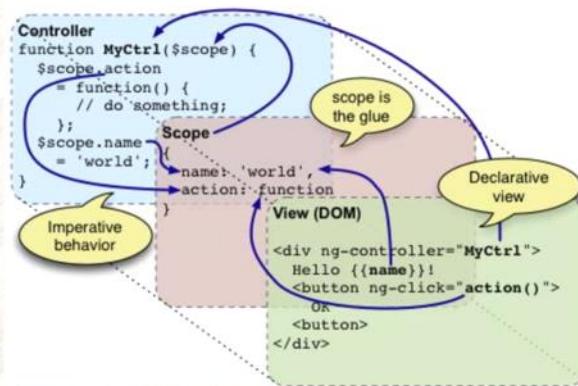


Model – View - Whatever



- **Vistas:** Será la representación de los datos o la información, es decir, el código del interfaz de usuario, básicamente el DOM/HTML/CSS .
- **Controladores:** Se encargarán de la lógica de la aplicación, incluyendo "Factorías" y "Servicios" para mover datos contra servidores o memoria local en HTML5.
- **Modelo o Modelo de la vista,** según se emplee la variante del patrón MVC o MVVC.

El modelo es la estructura lógica que subyace a los datos. Asociado a él se define el **scope**, responsable de detectar los cambios en el modelo y proporciona el contexto a las plantillas.



Futuro de HTML: Web Components

Web Components

conjunto de estándares que, permiten crear y utilizar elementos HTML personalizados.

Se puede así ampliar el “vocabulario” de HTML con elementos propios

En ellos está trabajando la W3C. Ya se soportan en algunos navegadores (Chrome) y está disponible un *polyfile* para que puedan usarse en otros.

Constan de cuatro especificaciones:

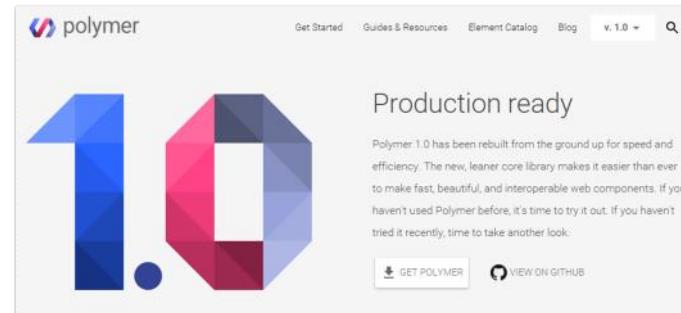


- **Custom elements**: Nos permite definir nuevos elementos HTML.
- **Templates**: Sistema de plantillas nativas en el navegador.
- **Shadow DOM**: DOM scope independiente en cada componente.
- **HTML Imports**: Carga de documentos HTML.



<http://webcomponents.org/>

<https://www.polymer-project.org/1.0/>



Web Components 1

domingo, 10 de diciembre de 2017 21:07

Adopta la sintaxis de ES6

Custom elements

```
class AppSample extends HTMLElement {  
    constructor () {  
        super()  
        console.log("Creado el componente")  
    }  
  
    customElements.define('app-sample', AppSample)
```

Shadow DOM

```
class AppSample extends HTMLElement {  
    constructor () {  
        super()  
        console.log("Creado el componente")  
        this.attachShadow({mode: 'open'}).innerHTML  
            = "<h1>Componente Web</h1>"  
    }  
}  
customElements.define('app-sample', AppSample)
```

Templates

```
<template id="temp1">  
    <style>  
        h1 { color: orange; }  
    </style>  
    <h1>Hola Mundo</h1>  
    <p>  
        Lorem ipsum dolor sit,  
        amet consectetur adipisicing elit.  
        Reiciendis, facilis magnam. Beatae, maxime itaque?  
        Id unde corrupti vero, eligendi obcaecati ullam placeat  
        ducimus sint, iste cumque neque non iure consequatur.  
    </p>  
</template>
```

HTML Imports

```
<link rel="import" href="./templete.html">
```

Resultado final

```
class AppSample extends HTMLElement {  
    constructor () {  
        super()  
  
        const oImport =  
            document.querySelector('link[rel="import"]').import;  
        const oElem = oImport.querySelector('#temp1')  
  
        this.attachShadow({mode: 'open'}).innerHTML = oElem.innerHTML  
    }  
}
```

Referencia al *template* importado para almacenarlo como un objeto de tipo elemento del DOM

```
}
```

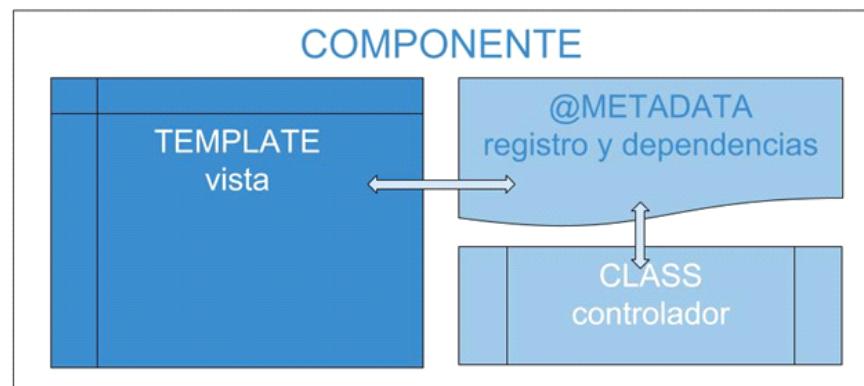
```
customElements.define('app-sample', AppSample)
```

Componentes en Angular

domingo, 1 de octubre de 2017 11:55

Componentes en Angular

- **Elemento personalizado:** Nos permite definir nuevos elementos HTML.
- Cada uno de ellos con su **template:** Sistema de plantillas nativas en el navegador.
- **Shadow DOM:** contenedor no visible
- **ciclo de vida** bien definido
- evolución de los componente definidos en AngularJS 1.5



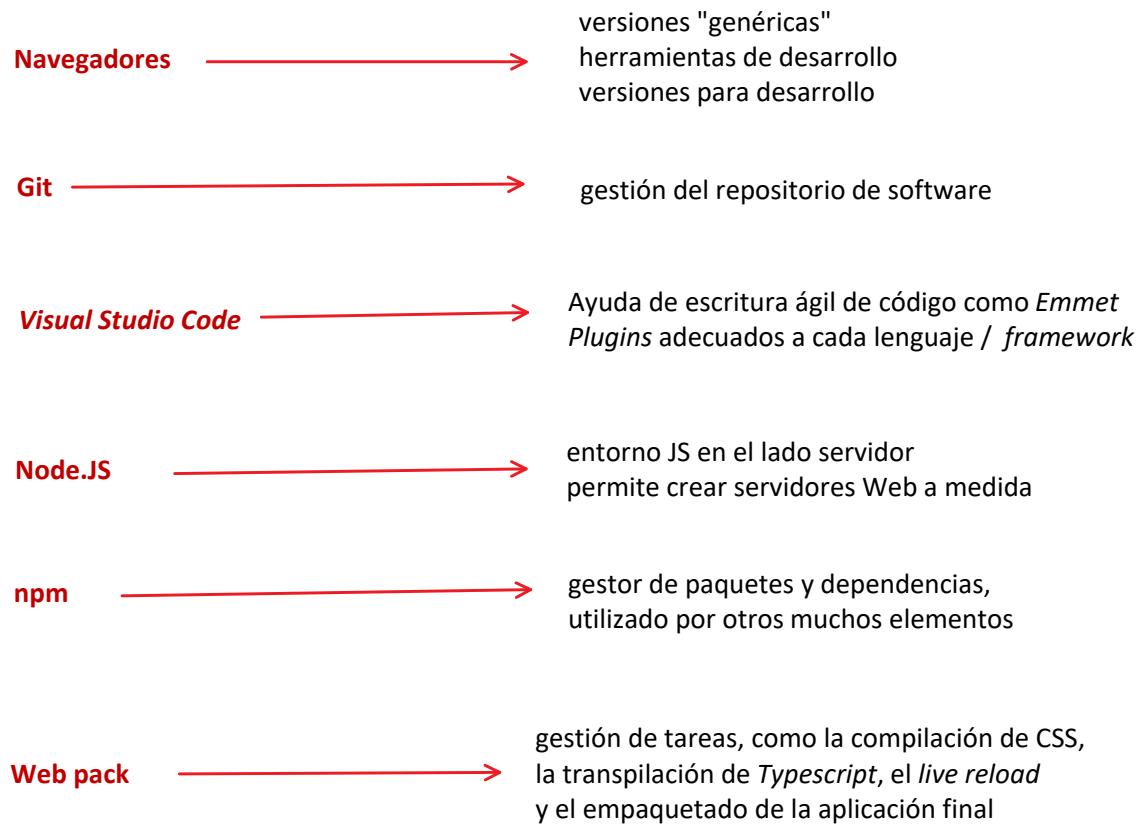
Árbol de componentes



Entorno de trabajo

sábado, 9 de septiembre de 2017 13:33

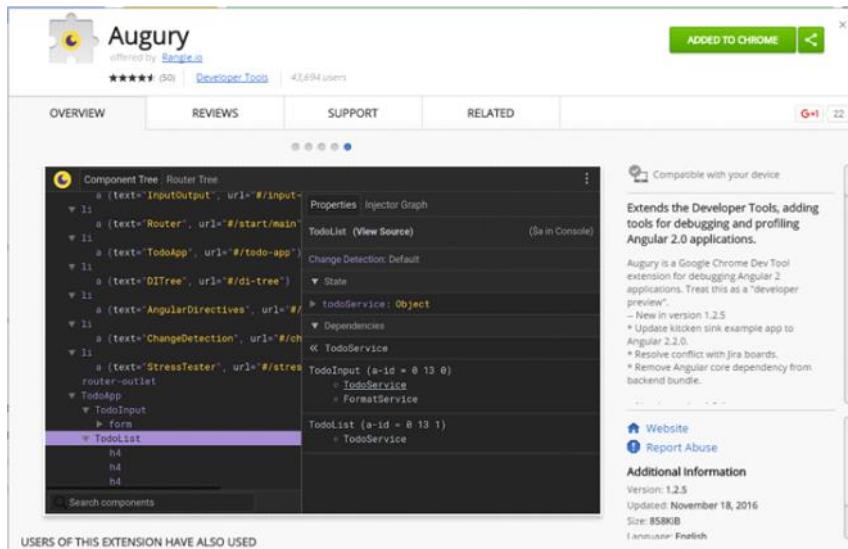
Navegadores.
Editores de código
Gestión de versiones. GIT
NodeJS y npm. Empaquetado



Navegadores

sábado, 9 de septiembre de 2017 18:20

Plugin en Chrome



<https://chrome.google.com/webstore/detail/augury/elgalmkoelokbchhhacckoklkejnhcd>

Editores de código

sábado, 9 de septiembre de 2017 18:21

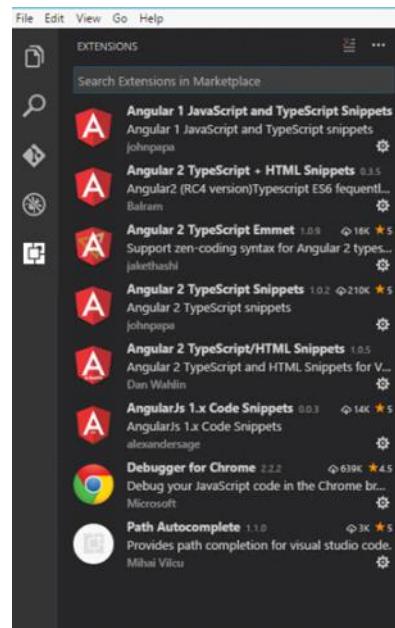


Extensiones de VSC

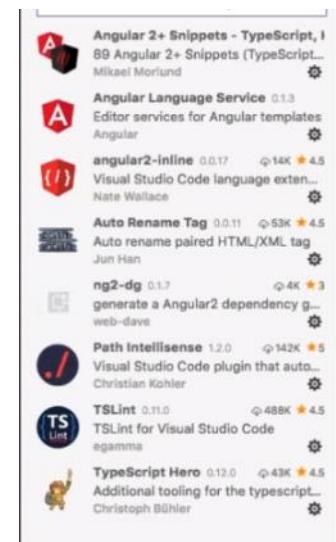
- Angular 2 Typescript
- Angular 2 Typescript Emmet
- Angular 4 and TypeScript/HTML VS Code Snippets (Dan Wahlin)
- **Angular 4 Typescript Snippets ***** (John Papa)**
- **Angular Languaje Service**
- **angular2-inline** (Nate Wallace)

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

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



A.Basalo, Angular 4



<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

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/>



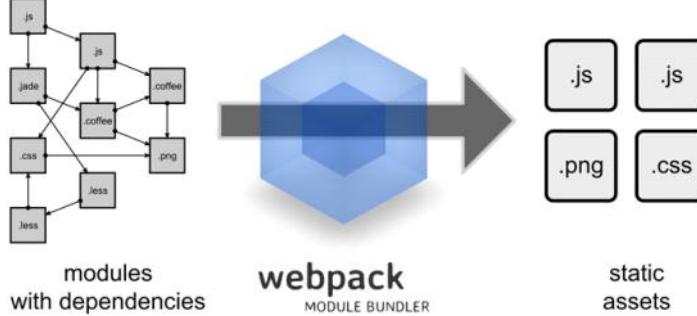
<http://gulpjs.com/>



<http://broccolijs.com/>



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



finalmente incluido en **Angular-cli**

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



webpack
MODULE BUNDLER

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

Construcción del proyecto

 ARMA
<https://karma-runner.github.io>

 Jasmine
<http://jasmine.github.io/>

 Protractor
end to end testing for AngularJS
<http://www.protractortest.org/>

Herramientas de testing

Instalación y uso

sábado, 14 de octubre de 2017 20:11

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

The screenshot shows a Windows command prompt window titled 'cmd C:\WINDOWS\system32\cmd.exe'. The command 'D:\>ng version' is run, displaying the following information:

```
D:\>ng version
angular-cli: 1.0.0-beta.19-3
node: 8.4.0
os: win32 x64
@angular/cli: 1.4.1
node: 8.4.0
os: win32 x64
C:\Users\alce6>
```

The text 'Angular CLI' is overlaid in large red letters across the bottom of the window.

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

```
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*
- *recargara automáticamente al guardar un fichero fuente*

En nuestro caso, el comando de creación del proyecto será:

```
ng new curso-app -p acc --routing true
```

<nombre
proyecto>

<prefijo>

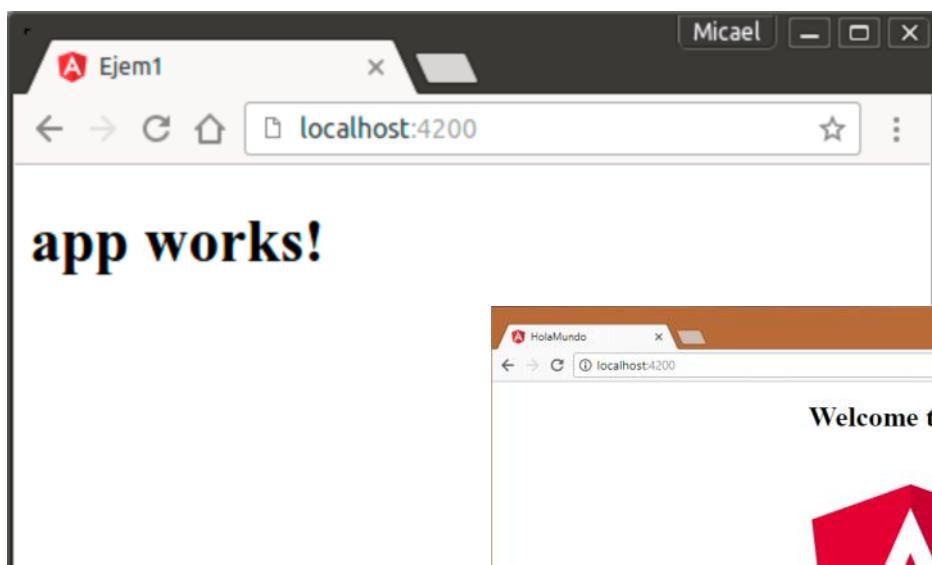
Ejecución: ng serve

domingo, 4 de febrero de 2018 21:21

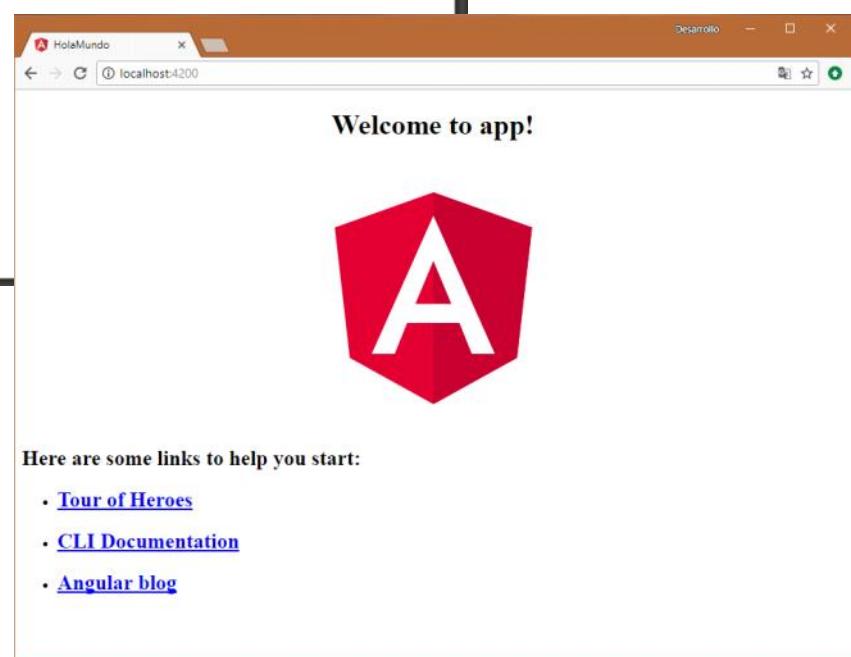
```
D:\Desarrollo\Angular2\Curso_CAS\curso-app>ng serve
** NG Live Development Server is running on http://localhost:4200. **
43871ms building modules
47ms sealing
0ms optimizing
0ms basic module optimization
180ms module optimization
4ms advanced module optimization
18ms basic chunk optimization
15ms 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: 52002ms
          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



En Angular 5, las mejoras del proceso de compilación aot lo hacen utilizable incluso en

fase de desarrollo por lo que se recomienda añadirlo al comando serve

```
ng serve --aot -o
```

Lo habitual es guardar el comando con sus modificadores como script npm



```
  "scripts": {  
    "ng": "ng",  
    "start": "ng serve --aot -o",  
    "build": "ng build --prod",  
    "test": "ng test",  
    "lint": "ng lint",  
    "e2e": "ng e2e"  
  },
```

Al ser un comando estándar npm se ejecuta directamente (sin anteponer run)

```
npm start
```

Hola Mundo

sábado, 9 de septiembre de 2017 23:57

Instalación de VSC y sus extensiones

Instalación de *git*
configuración del proxy

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

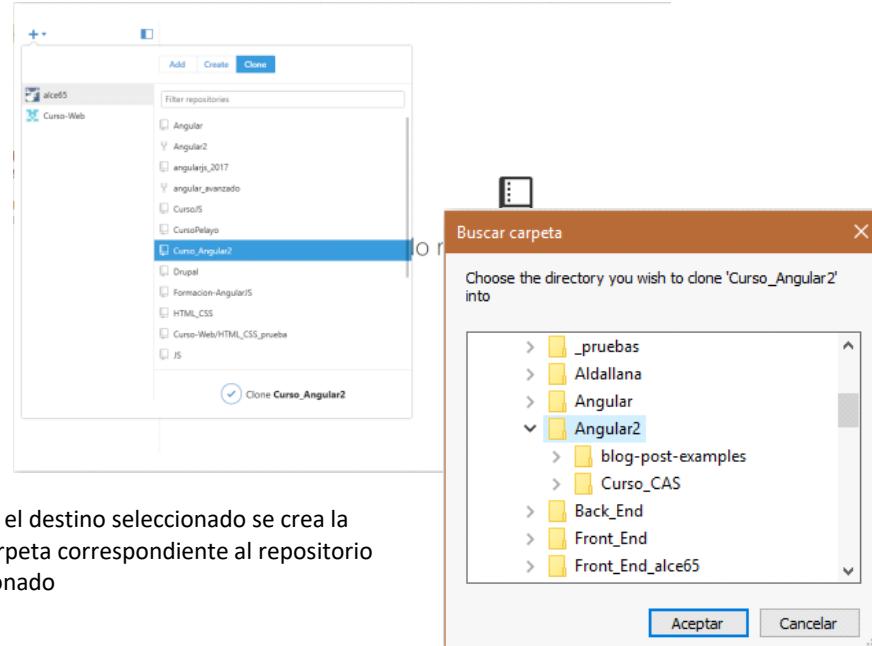
Instalación de *node/npm*
configuración del proxy

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

Instalación de Angular-cli:

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

Creación de la carpeta para el curso,
como repositorio *git* vinculado a GitHub



En el destino seleccionado se crea la
carpeta correspondiente al repositorio
clonado

Creación del proyecto: `ng new hola-mundo`

```
D:\Desarrollo\Angular2\Curso_Angular2> ng new hola-mundo
D:\Desarrollo\Angular2\Curso_Angular2> ng new hola-mundo
Administrator: C:\Windows\system32\cmd.exe

D:\Desarrollo\Angular2\Curso_Angular2> ng new hola-mundo
Unable to find "@angular/cli" in devDependencies.

Please take the following steps to avoid issues:
"npm install --save-dev @angular/cli@latest"

create  hola-mundo/e2e/app.e2e-spec.ts (292 bytes)
create  hola-mundo/e2e/app.po.ts (208 bytes)
create  hola-mundo/e2e/tsconfig.e2e.json (235 bytes)
create  hola-mundo/karma.conf.js (923 bytes)
create  hola-mundo/package.json (1315 bytes)
create  hola-mundo/protractor.conf.js (722 bytes)
create  hola-mundo/README.md (1100 bytes)
create  hola-mundo/tsconfig.json (363 bytes)
create  hola-mundo/tslint.json (3040 bytes)
create  hola-mundo/.angular-cli.json (1128 bytes)
create  hola-mundo/.editorconfig (245 bytes)
create  hola-mundo/.gitignore (516 bytes)
create  hola-mundo/src/assets/.gitkeep (0 bytes)
create  hola-mundo/src/environments/environment.prod.ts (51 bytes)
create  hola-mundo/src/environments/environment.ts (387 bytes)
create  hola-mundo/src/favicon.ico (5430 bytes)
create  hola-mundo/src/index.html (296 bytes)
create  hola-mundo/src/main.ts (370 bytes)
create  hola-mundo/src/polyfills.ts (2480 bytes)
```

```
Administrator: C:\Windows\system32\cmd.exe
D:\Desarrollo\Angular2\Curso_Angular2> ng new hola-mundo
Unable to find "@angular/cli" in devDependencies.

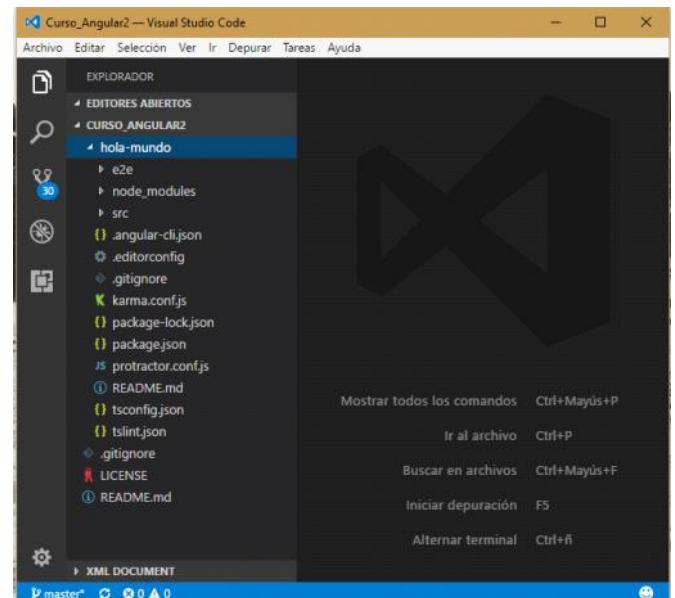
Please take the following steps to avoid issues:
"npm install --save-dev @angular/cli@latest"

  create  hola-mundo/e2e/app.e2e-spec.ts (292 bytes)
  create  hola-mundo/e2e/app.po.ts (208 bytes)
  create  hola-mundo/e2e/tsconfig.e2e.json (235 bytes)
  create  hola-mundo/karma.conf.js (923 bytes)
  create  hola-mundo/package.json (1315 bytes)
  create  hola-mundo/protractor.conf.js (722 bytes)
  create  hola-mundo/README.md (1180 bytes)
  create  hola-mundo/tsconfig.json (363 bytes)
  create  hola-mundo/tslint.json (3040 bytes)
  create  hola-mundo/.angular-cli.json (1128 bytes)
  create  hola-mundo/.editorconfig (245 bytes)
  create  hola-mundo/.gitignore (516 bytes)
  create  hola-mundo/src/assets/.gitkeep (0 bytes)
  create  hola-mundo/src/environments/environment.prod.ts (51 bytes)
  create  hola-mundo/src/environments/environment.ts (387 bytes)
  create  hola-mundo/src/favicon.ico (5430 bytes)
  create  hola-mundo/src/index.html (296 bytes)
  create  hola-mundo/src/main.ts (370 bytes)
  create  hola-mundo/src/polyfills.ts (2480 bytes)
  create  hola-mundo/src/styles.css (80 bytes)
  create  hola-mundo/src/test.ts (1085 bytes)
  create  hola-mundo/src/tsconfig.app.json (211 bytes)
  create  hola-mundo/src/tsconfig.spec.json (304 bytes)
  create  hola-mundo/src/typings.d.ts (104 bytes)
  create  hola-mundo/src/app/app.module.ts (314 bytes)
  create  hola-mundo/src/app/app.component.html (1075 bytes)
  create  hola-mundo/src/app/app.component.spec.ts (986 bytes)
  create  hola-mundo/src/app/app.component.ts (207 bytes)
  create  hola-mundo/src/app/app.component.css (0 bytes)

Installing packages for tooling via npm.
Installed packages for tooling via npm.
Directory is already under version control. Skipping initialization of git.
Project 'hola-mundo' successfully created.

D:\Desarrollo\Angular2\Curso_Angular2>
```

Accedemos al proyecto desde VSC



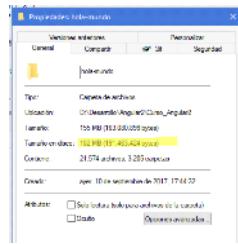
Resultado

domingo, 10 de septiembre de 2017 18:51

The image displays a vertical sequence of three screenshots illustrating the Angular 2 development workflow:

- Top Screenshot:** A screenshot of Visual Studio Code showing the file `app.component.html`. The code contains a link to the Angular tutorial: `Tour of Heroes`. A red arrow points from this code down to the terminal window.
- Middle Screenshot:** A screenshot of the integrated terminal in Visual Studio Code. It shows the command `ng serve` being run in a PowerShell window. The output indicates the server is listening on port 4200. Another red arrow points from this terminal window down to the browser window.
- Bottom Screenshot:** A screenshot of a web browser displaying the application. The page title is "HolMundo" and the URL is "localhost:4200". The content of the page includes the text "Welcome to app!" and the Angular logo. Below this, there is a section titled "Here are some links to help you start:" with three links: "Tour of Heroes", "CLI Documentation", and "Angular blog". To the right of this content, the terminal output from the previous screenshot is displayed again, showing the successful compilation of the Angular application.

Resultado: estructura



Angular 2.0

- ficheros de infraestructura**
- Infraestructura para **TypeScript**
 - Infraestructura de pruebas
 - Infraestructura de despliegue
- e2e**
- **E2e:** Testing end to end
 - **dist:** Recursos que hay que publicar en el servidor web
 - **node_modules:** Librerías y herramientas descargadas
 - **src:** Fuentes de la aplicación
- node_modules**
- src**
- .angular-cli.json**
- .editorconfig**
- .gitignore**
- karma.conf.js**
- package.json**
- package-lock.json**
- protractor.conf.js**
- README.md**
- tsconfig.json**
- tslint.json**

- **config:** configuración de environments y tests
- **E2e:** Testing end to end
- **dist:** Recursos que hay que publicar en el servidor web
- **node_modules:** Librerías y herramientas descargadas
- **public:**
- **src:** Fuentes de la aplicación
- **tmp:** temporal
- **typings:** tipos predefinidos para información al editor de código

package.json

- **.editconfig**
- **.gitignore: git**

- **package.json: librerías (dependencias)**
- **package_lock.json:** tipos

Ficheros: de configuración del editor y de git

```
{
  "name": "curso-app",
  "version": "0.0.0",
  "license": "MIT",
  "angular-cli": {},
  "scripts": {
    "start": "ng serve",
    "lint": "tslint \"src/**/*.ts\"",
    "test": "ng test",
    "e2e": "webdriver-manager update",
    "e2e": "protractor"
  },
  "private": true,
  "dependencies": {
    "@angular/common": "~2.1.0",
    "@angular/compiler": "~2.1.0",
    "@angular/core": "~2.1.0",
    "@angular/forms": "~2.1.0",
    "@angular/http": "~2.1.0",
    "@angular/platform-browser": "~2.1.0",
    "@angular/platform-browser-dynamic": "~2.1.0",
    "@angular/router": "~3.1.0",
    "core-js": "~2.4.1",
    "rxjs": "5.0.0-beta.12",
    "ts-helper": "~1.1.1",
    "zone.js": "~0.6.23"
  }
}
```

Scripts

Dependencias

Dependencias de desarrollo

- **karma.conf.js:** tests con karma
- **protractor.conf.js:** tests con protractor

- **config**
- **dist**
- **e2e**
- **node_modules**
- **public**
- **src**
- **tmp**
- **typings**
- **.clang-format**
- **.editorconfig**
- **.gitignore**
- **angular-cli.json**
- **angular-cli-build.js**
- **package.json**
- **tsconfig.json**
- **tslint.json**
- **typings.json**

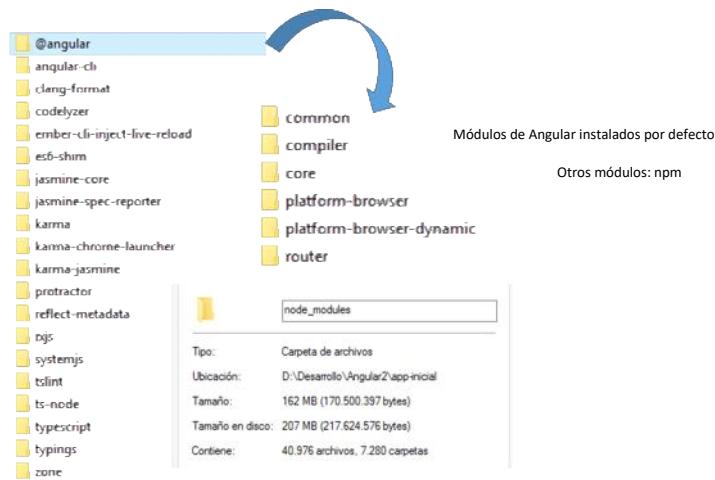
- **config**
- **dist**
- **e2e**
- **node_modules**
- **public**
- **src**
- **tmp**
- **typings**
- **.clang-format**
- **.editorconfig**
- **.gitignore**
- **angular-cli.json**
- **angular-cli-build.js**
- **package.json**
- **tsconfig.json**
- **tslint.json**
- **typings.json**

- **package.json: librerías (dependencias)**
- **angular-cli.json: angular-cli**
- **angular-cli-build.js: angular-cli**
- **tsconfig.json: typescript**
- **typings.json: tipos**

Node_Modules

domingo, 10 de septiembre de 2017 11:45

Resultado: node_modules



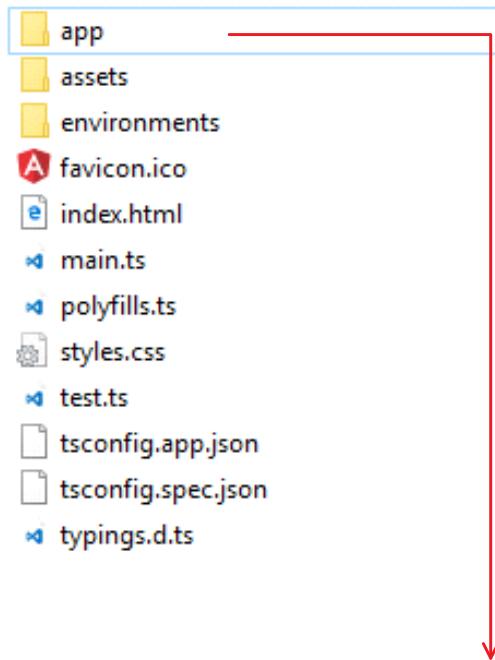
- No se incluye en la copia de los proyectos distribuida en repositorios git / GitHub
- Puede volver a generarse en cualquier momento, de acuerdo con lo indicado en `package.json`: `npm install`
- Desde el punto de vista de `npm/Node`, la carpeta `node_modules` puede reubicarse en niveles superiores para compartirla en diversos proyectos.

-
- npm busca la carpeta `node_modules` en la carpeta actual o en los niveles superiores
 - Puede reubicarse en niveles superiores para compartirlo en diversos proyectos
 - Puede ser necesario ajustar la configuración del editor de código, para indicarle donde se ubica el compilador de `typescript`

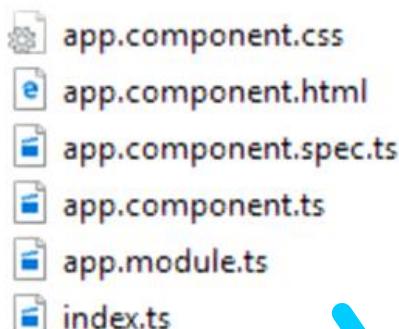
Sin embargo angular/cli necesita que se mantenga en su ubicación original

Sources: src

domingo, 10 de septiembre de 2017 11:49



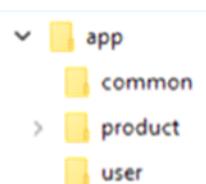
- **index.html:** Página principal.
Se editará para incluir CSS globales en la web
- favicon.ico: Icono de la aplicación
- **main.ts:** Fichero principal de la aplicación.
No es necesario modificarlo
- **tsconfig.app.json**
- **tsconfig.spec.json:** Configuración del compilador TS
 - style.css
 - polyfills.ts
 - test.ts
 - typings.d.ts



carpeta que contiene los ficheros fuente principales de la aplicación.

Distribución por características

se agrupan los elementos correspondientes a sus distintas características, e.g. las opciones del menú principal



Ficheros principales

domingo, 10 de septiembre de 2017 18:24

.\src\main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';
if (environment.production) {
  enableProdMode();
}
platformBrowserDynamic().bootstrapModule(AppModule)
.catch(err => console.log(err));
```

configura la forma en que se combina el código de angular y el HTML, dejando que lo haga el browser de forma similar a como hacia Angular 1

Importación del módulo principal con su nombre y ubicación por defecto (siempre sin extensión: será TS o JS en distintos momentos)

se pueden configurar los ambientes de desarrollo y producción

Loader (arranque) de la aplicación Angular, indicando el Módulo que se debe cargar

Sustituye al ng-app, el bootstrapping explícito y declarativo, típico de Angular 1.x, aunque opcionalmente podía usarse el formato implícito y programático (imperativo)

.\src\index.html

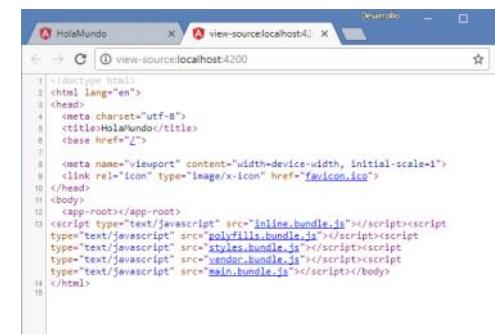
```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Hola Mundo</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Elementos de la cabecera (head) de HTML5

Llamada al COMPONENTE RAÍZ de la aplicación

Los componentes (aparecidos en Angular 1.5) suponen la ampliación del DTD de HTML con elementos del DOM diseñados a medida

Se echan en falta llamadas a ficheros externos (JS, CSS)
La función de Webpack es "empaquetar" toda esa información en bundles e injectar las correspondientes llamadas a ellos



Puede verse observando el código fuente tal como lo muestra el navegador

.\src\app\app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [
    AppComponent
  ]
})

export class AppModule {}
```

Definición de un **módulo**

- declarations
- imports
- Providers

En el módulo principal se añade el

- bootstrap

clase "decorada" por la función anterior

.\src\app\app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'app';
}
```

Definición de un **componente**

- selector
- templateUrl
- styleUrls

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
  
</div>
```

Ejercicio: Hola Mundo personalizado

viernes, 13 de octubre de 2017 22:23

Añadimos en la carpeta assets la imagen con el logotipo que nos interesa

Añadimos los estilos CSS necesarios

```
body {  
    background-color:#8D1919;  
}  
header {  
    text-align: center;  
    font-size: 1.8em;  
    color : papayawhip;  
}  
footer {  
    position: fixed;  
    bottom : 0;  
    width: 100%;  
    border-top: 1px papayawhip solid;  
    padding: 1em;  
}  
footer p {  
    text-align: center;  
    font-size: 0.9em;  
    color : papayawhip;  
    margin: 0.5em  
}  
div {  
    width : 40%;  
    margin : 1em auto  
}  
img {  
    width: 100%  
}
```

Creamos en el controlador del componente las siguientes variables

```
curso = 'Angular 4.x';  
formador = 'Alejandro Cerezo Lasne';  
empresa = 'Icono Training Consulting';  
fecha = '2017';
```

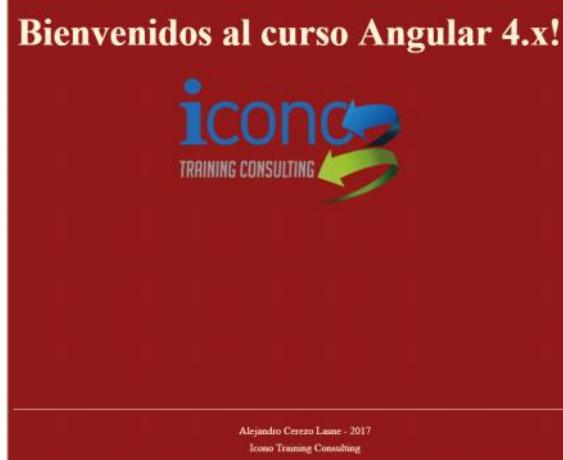
Actualizamos la vista de nuestro componente

.\src\app\app.component.html

```
<header>  
    <h1>Bienvenidos al curso {{curso}}!</h1>  
</header>  
<article>  
    <div>  
          
    </div>  
</article>  
<footer>  
    <p>{{formador}} - {{fecha}}</p>  
    <p>{{empresa}}</p>  
</footer>
```

Interpolamos las variables del componente

Incorporamos el fichero



Reubicación de node_modules

sábado, 11 de noviembre de 2017 22:48

Es necesario crear un enlace simbólico que simule la ubicación original

manualmente, en una consola de Administrador:

```
mklink /D <nombre_del_enlace> <path_real>
```

mediante una extensión del *shell* del Explorador de Windows



Link Shell Extension
Hermann Schinagl

14,6 MB
15/07/2017

<http://schinagl.priv.at/nt/hardlinkshellext/linkshellextension.html>

Para que no se produzcan avisos (*warnings*) será necesario añadir en `ng serve` un modificador

```
ng serve --preserve-symlinks
```

Es posible añadir este valor al comando `npm start` definido en `package.json`

Publicar en GitHub

domingo, 10 de septiembre de 2017 19:00

Desde el propio VSC podemos incorporar los proyectos al repositorio de **GitHub**

The screenshot shows the Visual Studio Code interface with a context menu open over a file named 'icono.png'. The menu has several options:

- Extraer de...
- Incorporación de cambios
- Incorporación de cambios (fusionar mediante cambio de base)
- Insertar
- Insertar en
- Sincronizar** (highlighted with a red box)
- Publicar rama
- Confirmar almacenados provisionalmente
- Confirmar almacenados provisionalmente (modificar)
- Confirmar por etapas (Aprobado)
- Confirmar todo
- Confirmar todo (aprobado)
- Confirmar todo (modificar)
- Deshacer última confirmación
- Cancelar almacenamiento provisional de todos los cambios
- Descartar todos los cambios
- Aplicar y quitar cambios guardados provisionalmente...
- Aplicar y quitar últimos cambios guardados provisionalmente...
- Guardar provisionalmente
- Mostrar salida de GIT
- Instalar proveedores adicionales de SCM...

A red arrow points from the 'git commit' text above to the 'Publicar rama' option in the menu. Another red arrow points from the 'git pull' and 'git push' text below to the 'Sincronizar' option. A large red box surrounds the entire context menu area.

IMPORTANTE

.gitignore

```
# Dependency directories
node_modules/
jspm_packages/
```

The GitHub repository page shows the following details:

- Repository name: alce65 / Curso_Angular2
- Last commit: 44efb97 26 seconds ago
- Branch: master
- Commits: 2
- Branches: 1
- Releases: 0
- Contributors: 1
- Licence: MIT

The commit history includes:

- holamundo (Initial commit, 26 seconds ago)
- holamundo2 (Initial commit, 8 hours ago)
- .gitignore (Initial commit, 8 hours ago)
- LICENSE (Initial commit, 8 hours ago)
- README.md (Initial commit, 8 hours ago)
- icono.png (Proyectos Hola Mundo, 26 seconds ago)

The README.md file content is:

Curso_Angular2

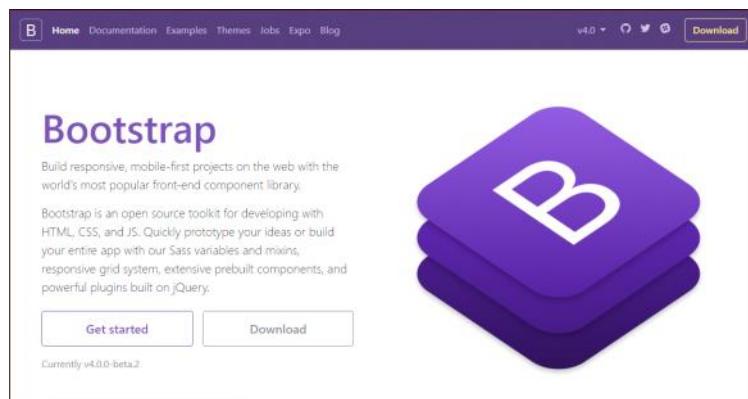
Curso de Angular2 en Icono Training Consulting

CSS: 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-beta.2
```

- 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",
```

Componentes ng-bootstrap

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

The screenshot shows the official GitHub page for ng-bootstrap. At the top, there's a navigation bar with links for 'ng-bootstrap', 'Getting Started', and 'Components'. On the right, there are social sharing icons for GitHub, Star, and Issues. Below the header, a large blue section features a white 3D letter 'B' icon. The text 'Bootstrap 4 components, powered by Angular' is displayed, along with a note 'Currently at v1.0.0-beta.5'. Two buttons, 'Demo' and 'Installation', are present. Below this, there are two main sections: 'Native' (represented by a CPU icon) and 'Widgets' (represented by a monitor icon). The 'Native' section describes it as 'Angular - specific widgets built from ground up using Bootstrap 4 CSS, APIs that makes sense in the Angular ecosystem. No dependencies on 3rd party JavaScript.' The 'Widgets' section describes it as 'All the Bootstrap widgets (ex. carousel, modal, popovers, tooltips, tabs, ...) and several additional goodies (datepicker, rating, timepicker, typeahead).'

Instalación

Se instalan mediante npm

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

Utilización

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

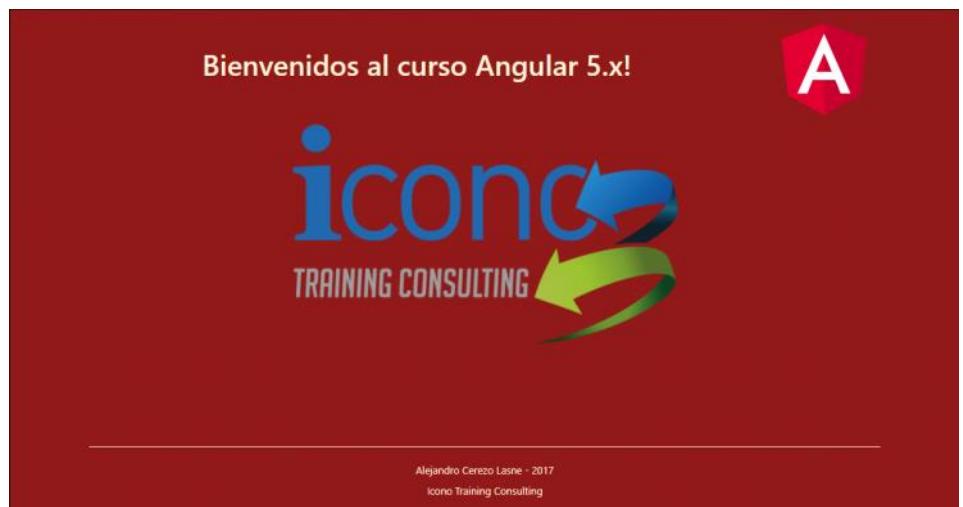
Lunes, 6 de noviembre de 2017 22:00

```
<div class= "container">
  <header class="row align-items-center">
    <div class="col-10 text-center">
      <h1>Bienvenidos al curso {{curso}}!</h1>
    </div>
    <div class="col-2 logo">
      <img [src]="logoAngular" alt="logotipo de Angular">
    </div>
  </header>
  <article class="row">
    <div class="col-6 offset-3 logo">
      <img class="img-fluid" [src]="iconoLogo" alt="logotipo de Icono">
    </div>
  </article>
  <div class="row">
    <footer class="col-10 offset-1 text-center">
      <p>{{formador}} - {{fecha}}</p>
      <p>{{empresa}}</p>
    </footer>
  </div>
</div> <!--Fin del container-->
```

CSS

```
header {
  color : papayawhip;
}
footer {
  position: fixed;
  bottom : 0;
  left:0;
  border-top: 1px papayawhip solid;
  padding: 1em;
  font-size: 0.9em;
  color : papayawhip;
}
footer p {
  margin: 0.5em
}
```

```
body {
  background-color:#8D1919;
}
```



Angular cli: despliegue

domingo, 10 de septiembre de 2017 10:40

Angular-cli incluye un comando para preparar el despliegue de la aplicación

Cuando queremos publicar la aplicación en **producción** tenemos que generar los archivos optimizados y publicarlos en un servidor web

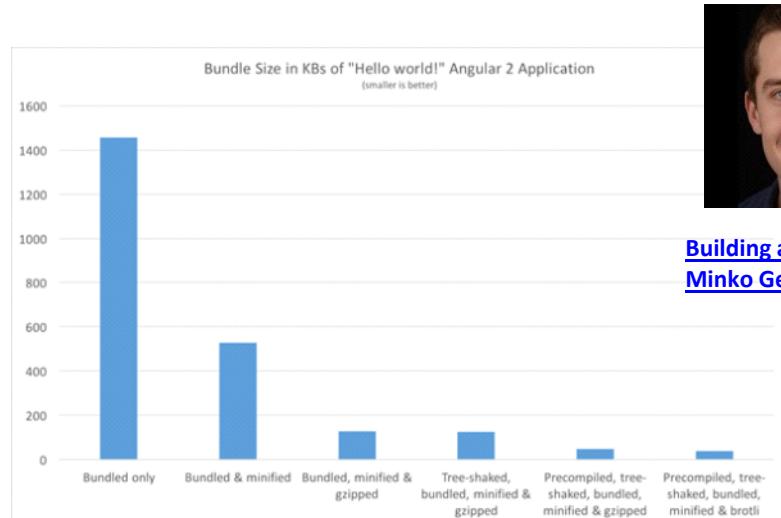
ng build

se generan en la carpeta *dist* los ficheros conocidos como *bundle*

Inicialmente no optimizaba el resultado (main.bundle.js ocupa 2,5 megas),

Sucesivas versiones de angular-cli han ido ajustando la configuración de *webpack* hasta llegar a un *bundle* de 50Kb

Mejoras del bundle



[Building an Angular 2 Application for Production – Minko Gechev's blog](#)

<http://blog.rangle.io/optimize-your-angular2-application-with-tree-shaking/>

Generamos la aplicación Hola-mundo para comparar los *bundles* con los que se generan temporalmente en desarrollo

Se optimiza con la opción *ng build -t production*
equivale a *ng build --target development / production*
o sus alias *ng build -dev / -prod*

Flag	--dev	--prod
--aot	false	true
--environment	dev	prod
--output-hashing	media	all
--sourcemaps	true	false
--extract-css	false	true

--named-chunks	true	false
--build-optimizer	false	true (with AOT and Angular 5)

Compilación

domingo, 4 de febrero de 2018 20:49

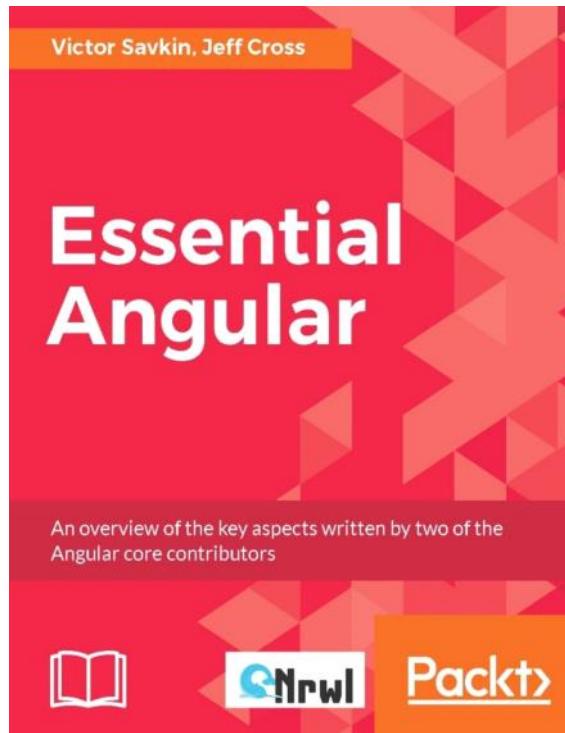
Angular incorpora un compilador de HTML (*HTML Compiler*) cuya función es

- recorrer el documento y localizar las directiva
- ejecutar los comportamientos asociados a esas directivas.

Este proceso puede tener lugar en 2 momentos diferentes

Just-in-time (JIT: durante la ejecución del código (*runtime*), cuando arranca (*bootstrapping*) la aplicación, como ocurre en AngularJS

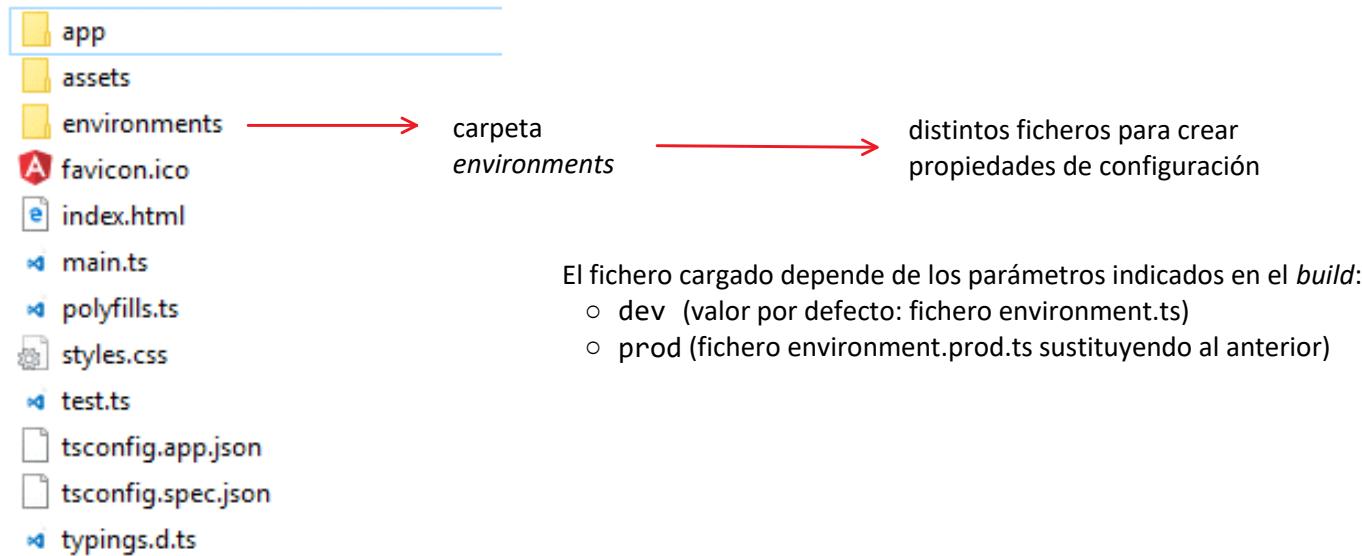
Ahead-of-time (AOT): durante el proceso de empaquetado y construcción (*build*) de la aplicación, con una considerable mejora del rendimiento



Essential Angular
Victor Savkin & Jeff Cross
Packt Publishing, 2017

Environments

domingo, 4 de febrero de 2018 21:09



TypeScript

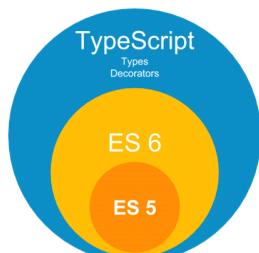
sábado, 9 de septiembre de 2017 13:34



<http://www.typescriptlang.org/>

Open source

Super Set de JavaScript ES6
(=> es JavaScript)



- Orientado a Objetos con clases
- Añade **tipos estáticos**
 - Inferencia de tipos
(no hay que declararlos en muchos sitios)
 - Tipos opcionales (si no quieres, no los usas)
- Anotaciones / Decoraciones
- Es *transpilado*: se genera código JavaScript ES5 (compatible con los navegadores web actuales)

Documentación on-line

The screenshot shows a GitBook page titled 'TypeScript Deep Dive' by 'basarat'. The page has a navigation bar with links to Pricing, Explore, About, and Blog. It includes a 'Sign In' and 'Sign Up' button. The main content area displays the book's introduction and first chapter. At the bottom right, there is a photo of Basarbat Ali Syed.

<https://www.gitbook.com/book/basarat/typescript/details>

Basarbat Ali Syed



Pruebas del código

<http://www.typescriptlang.org/play/>

The screenshot shows the TypeScript Playground interface. On the left, there is a code editor window containing a TypeScript file with code for creating a Greeter object and a button click event. On the right, there is a preview window showing the generated JavaScript code. The playground also features tabs for 'Run' and 'JavaScript'.

Clases

sábado, 29 de julio de 2017 15:33

```
clase // ejemplo de clase en TypeScript
      ↓
      export class Empleado {
propiedades    private nombre: string;
                private salario: number;
constructor     constructor(nombre: string, salario: number) {
                this.nombre = nombre;
                this.salario = salario;
}
métodos        getName() {
                return this.nombre;
}
toString() {
                return "Nombre:" + this.nombre +
                    ", Salario:" + this.salario;
}
}
```

- Al estándar de ES6 se le añaden
- Propiedades definidas fuera de los métodos
 - Modificadores de acceso (*private*, *protected*, *public*)
 - Interfaces

Herencia

```
class Animal {
    constructor(public name: string) { }
    move(distanceInMeters: number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}

class Snake extends Animal {
    constructor(name: string) { super(name); } ← Llamada al constructor de la clase padre
    move(distanceInMeters = 5) {
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}

class Horse extends Animal {
    constructor(name: string) { super(name); } ← Sobre escritura de los métodos de la clase padre
    move(distanceInMeters = 45) {
        console.log("Galloping...");
        super.move(distanceInMeters);
    }
}
```

Interfaces

Los interfaces son siempre públicos, por lo que no se utilizan modificadores de acceso

```
interface Usuario {
    id: number,
    name: string,
    dirección: {calle : string,
               num : number,
               zip: string}
    formación?: Array<string>
    saludar: Function;
    calcularPrecio(iva: number): void;
}
class Socio implements Usuario {
    id: number;
    name: string;
    dirección: {calle : string,
               num : number,
               zip: string}
    saludar() {
        console.log(`Hola, saludos de ${name}`);
    };
    calcularPrecio(iva) {
        ...
    }
}
```

elemento opcional, no tiene que estar en la implementación

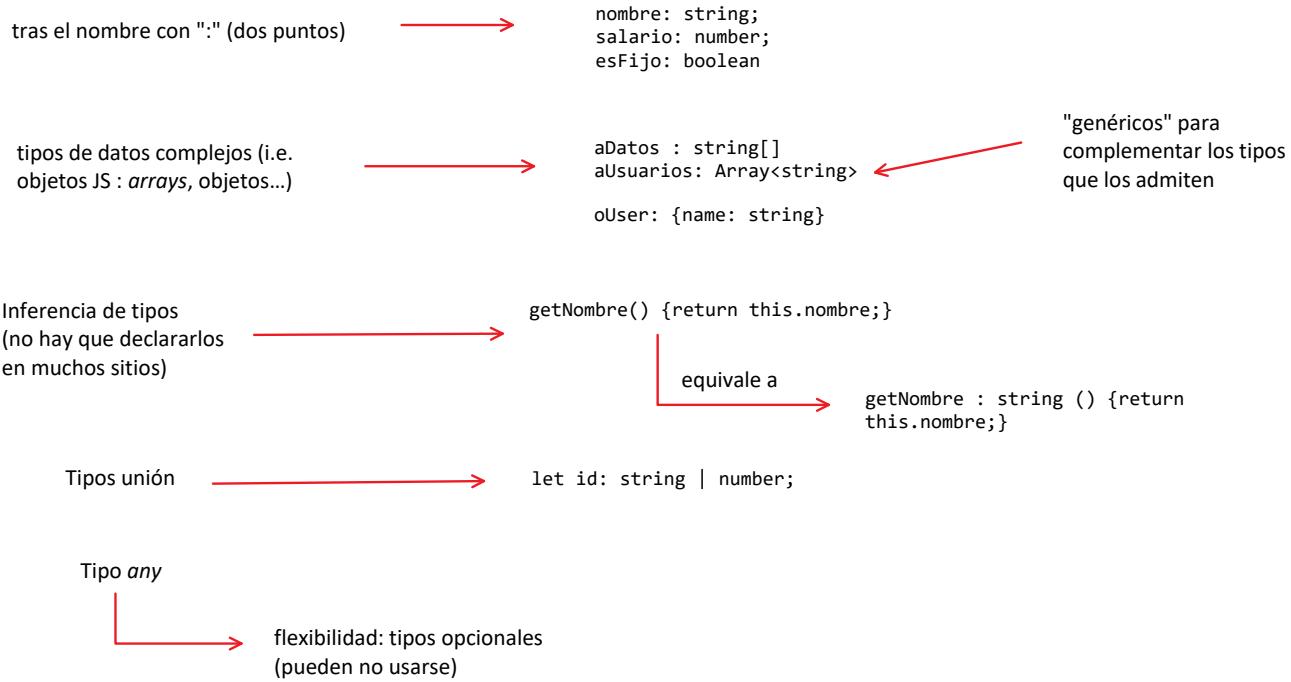
Método declarado en el interfaz y su correspondiente implementación

Los interfaces también se utilizan para definir tipos, como veremos a continuación

Tipos

domingo, 10 de septiembre de 2017 22:40

Tipos estáticos en tiempo de desarrollo;
evidentemente desaparecen tras la compilación (*traspilación*) a JS



Interfaces y tipos

Se pueden declarar interfaces exclusivamente con propiedades tipadas

```
interface Usuario {  
    id: number,  
    name: string,  
    dirección: {calle: string,  
               num: number,  
               zip: string}  
    formación: Array<string>  
}
```

Posteriormente, dentro de las clases se pueden usar esos interfaces, sin necesidad de implementarlos, para la declaración de tipos

```
class Empleado {  
    ...  
    public oUser: Usuario;  
    public aUsers: Array<Usuario>;
```

A partir de ahí se pueden crear objetos literales "tipados" por interfaces

Alias

Crea un nuevo nombre para un tipo o conjunto de tipos

```
type Name = string;  
type NameResolver = () => string;  
type NameOrResolver = Name | NameResolver;
```

- aumenta el contenido semántico de los tipos
- agrupa conjuntos de tipos

<https://www.typescriptlang.org/docs/handbook/advanced-types.html>

Módulos (y elementos de ES6)

domingo, 10 de septiembre de 2017 22:49

empleado.ts
fichero en el que se crea y
exporta una clase

```
export class Empleado {  
    nombre : string;  
    salario: number;  
  
    constructor (pNombre, pSalario)  
    {  
        this.nombre = pNombre;  
        this.salario = pSalario;  
    }  
}
```

sample.ts
fichero en el que se importa y utiliza
la clase anterior

```
import { Empleado } from "./empleado";  
  
let emps = new Array<Empleado>();  
  
emps.push(new Empleado('Pepe', 500));  
emps.push(new Empleado('Juan', 200));  
emps.push("Luis",400}  
  
for (let emp of emps) {  
    console.log(emp.getNombre());  
}  
  
emps.forEach(emp => {  
    console.log(emp);  
});
```

Importación desde otro módulo (fichero).
Se sobreentiende la extensión .ts

Tipo Array de objetos de la clase importada

código ES6 dentro de TypeScript

La ausencia de soporte del estándar ES6 en los navegadores o en Node hace necesaria la transpilación a ES5
cuando se utilizan módulos en TS
Desde TS se configura el uso de Node/CommonJS o sistemas de módulos alternativos.

Decoradores en Angular

domingo, 10 de septiembre de 2017 23:04

Importación de una clase desde otro módulo (fichero)

decorador de la clase a la que acompaña

exportación de la clase "decorada"

```
import {Component} from 'angular2/core';

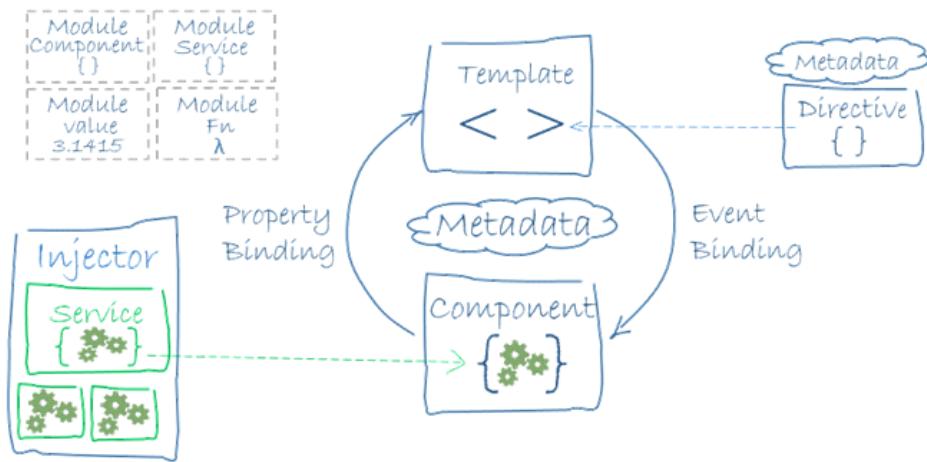
@Component({
  selector: 'app',
  templateUrl: 'app.component.html'
})
export class AppComponent {
  // código
  @input() nombre: string;
}
```

El decorador es un objeto JSON que da valor a una serie de METADATOS (propiedades) que esperan ser definidas y que pasarán a formar parte de la clase decorada

Ξ Angular. Segunda Parte

lunes, 11 de septiembre de 2017 22:13

Arquitectura Angular2



Elementos principales

*Modules
Components
Templates
Metadata
Data binding
Directives
Services
Dependency injection*

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

Guía de estilo

sábado, 14 de octubre de 2017 20:13

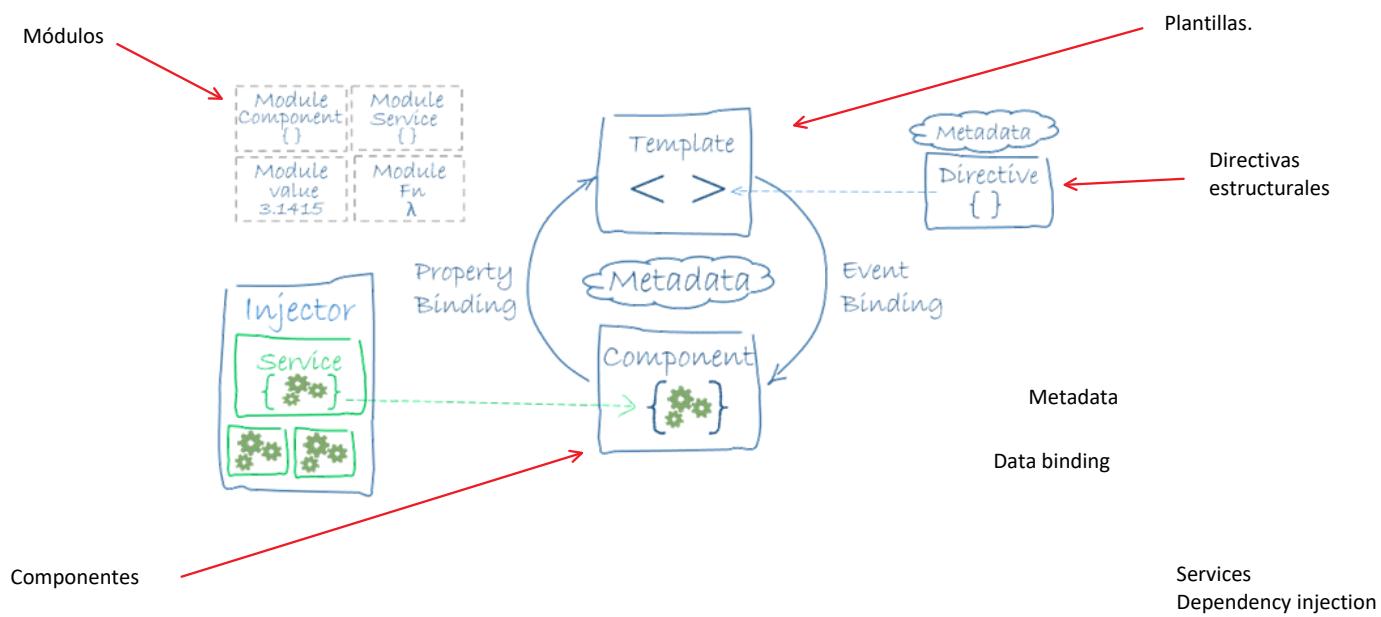
<https://angular.io/guide/styleguide>

The screenshot shows the Angular Style Guide page. The left sidebar has a navigation menu with sections like 'GETTING STARTED', 'TUTORIAL', 'FUNDAMENTALS', 'TECHNIQUES' (with 'Internationalization (i18n)', 'Security', 'Setup & Deployment', 'Upgrading', 'Visual Studio 2015 QuickStart', 'Style Guide', 'Glossary', and 'API'), and a note 'stable (v4.4.4)'. The main content area is titled 'Style Guide' and includes sections on 'Style vocabulary', 'Style conventions', and 'Why?'. The right sidebar lists topics such as 'Style vocabulary', 'File structure conventions', 'Single responsibility', 'Rule of One', 'Small functions', 'Naming', 'General Naming Guidelines', 'Separate file names with dots and dashes', 'Symbols and file names', 'Service names', 'Bootstrapping', 'Directive selectors', 'Custom prefix for components', 'Custom prefix for directives', 'Pipe names', 'Unit test file names', 'End-to-End (E2E) test file names', 'Angular NgModule names', 'Coding conventions', 'Application structure and NgModules', 'Components', 'Directives', 'Services', 'Data Services', 'Lifecycle hooks', and 'Appendix'.

- **Style vocabulary**
- **File structure conventions**
- **Single responsibility**
 - *Rule of One*
 - *Small functions*
- **Naming**
 - *General Naming Guidelines*
 - *Separate file names with dots and dashes*
 - *Symbols and file names*
 - *Service names*
 - *Bootstrapping*
 - *Directive selectors*
 - *Custom prefix for components*
 - *Custom prefix for directives*
 - *Pipe names*
 - *Unit test file names*
 - *End-to-End (E2E) test file names*
 - *Angular NgModule names*
- **Coding conventions**
 - *Classes*
 - *Constants*
 - *Interfaces*
 - *Properties and methods*
 - *Import line spacing*
- **Application structure and NgModules**
 - *LIFT*
 - *Locate*
 - *Identify*
 - *Flat*
 - *T-DRY (Try to be DRY)*
 - *Overall structural guidelines*
 - *Folders-by-feature structure*
 - *App root module*
 - *Feature modules*
 - *Shared feature module*
 - *Core feature module*
 - *Prevent re-import of the core module*
 - *Lazy Loaded folders*
 - *Never directly import lazy loaded folders*
- **Components**
 - *Component selector names*
 - *Components as elements*
 - *Extract templates and styles to their own files*
 - *Decorate input and output properties*
 - *Avoid aliasing inputs and outputs*
 - *Member sequence*
 - *Delegate complex component logic to services*
 - *Don't prefix output properties*
 - *Put presentation logic in the component class*
- **Directives**
 - *Use directives to enhance an element*
 - *HostListener/HostBinding decorators versus host metadata*
- **Services**
 - *Services are singletons*
 - *Single responsibility*
 - *Providing a service*
 - *Use the @Injectable() class decorator*
- **Data Services**
 - *Talk to the server through a service*
- **Lifecycle hooks**
 - *Implement lifecycle hook interfaces*
- **Appendix**
 - *Codelyzer*
 - *File templates and snippets*

Módulos. Componentes. Vistas

lunes, 11 de septiembre de 2017 22:17



Módulos

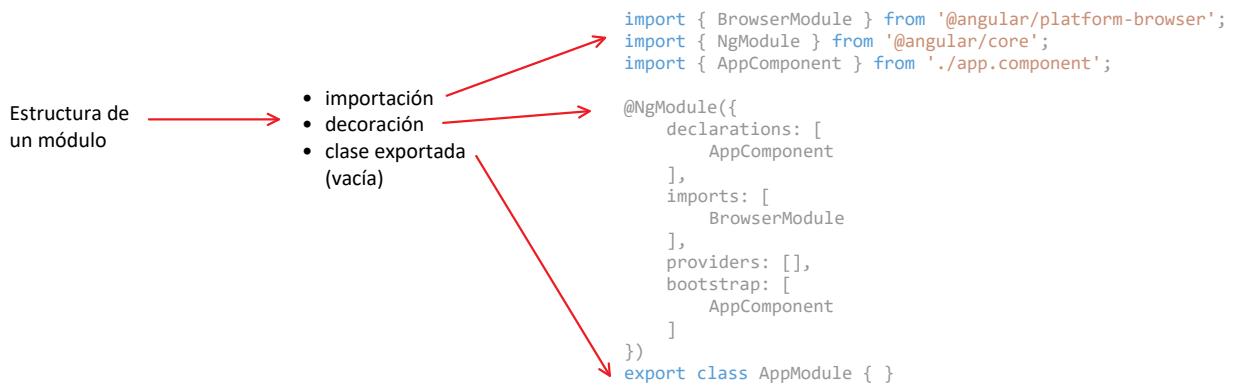
lunes, 11 de septiembre de 2017 22:23

Agrupación de componentes y otros elementos que son declarados en la decoración de una determinada clase

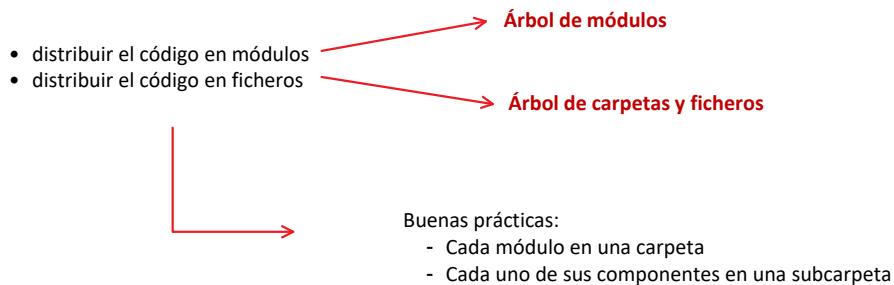
Toda app tiene

al menos un módulo que define los componentes de la app : AppModule

incluyendo al menos el componente principal
AppComponent -> selector: app-root

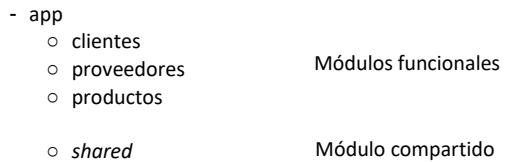


En cualquier aplicación hay que diferenciar 2 procesos



Independientemente de ambos, el uso de los componentes en el código HTML genera un **árbol de componentes**

Ejemplo: Módulos



Importación

lunes, 11 de septiembre de 2017 22:32

Todo los ficheros de *typescript/JS* que tengan que utilizarse dentro del módulo tienen que ser importados



Desaparecen los <script> en el HTML durante el desarrollo

No se indica la extensión:

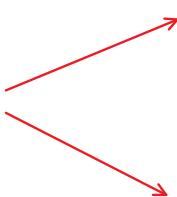
- en tiempo de desarrollo será TS
- en ejecución será JS, una vez *transpilado* y se reagrupará en el bundle.js

El comando *import* siempre implica dos elementos

```
import{ nombre de la clase }from 'donde esta la clase';
```

Si se omite el nombre de la clase {*} -> se importaran todas las que existan en el sitio indicado
(No es recomendable, por cómo funciona el *tree shaking*.)

Dos posibles "orígenes" a la hora de importar elementos



por **nombre simbólico**, desde *node_modules*, los elementos de Angular y otras librerías

```
import{ NgModule }from '@angular/core';
```

por **ruta**, desde un *path*, relativo a nuestra "base", los elementos de la aplicación
(esa "base" se define en los metadatos de index.html)

```
import { AppComponent }from './app.component';
```

Un módulo también puede ser importado desde otro, dando lugar a un árbol de módulos

Importación: index.ts



En TypeScript se pueden definir ficheros index.ts que exportan todos los ficheros de una carpeta.

- simplifica la importación desde otros ficheros
- desacopla los tipos del fichero en el que se declaran

index.ts

```
export * from './app.component';
export * from './app.module';
```

Fichero que lista todos los ficheros TS de una carpeta

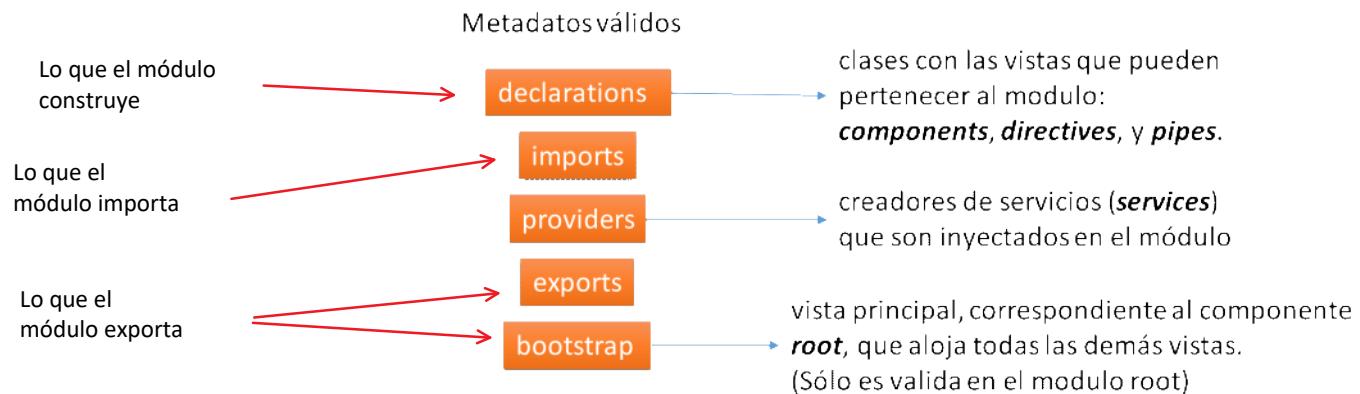
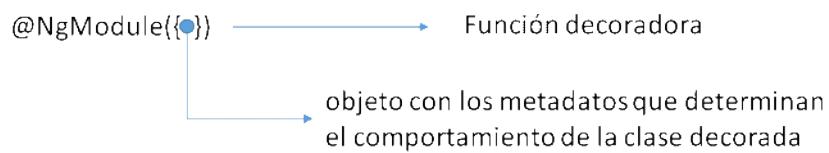
src/main.ts

```
import { AppModule } from './app/';
```

Al importar ese fichero se puede referenciar cualquier clase de la carpeta (similar a paquetes Java)

Decoración / Anotación

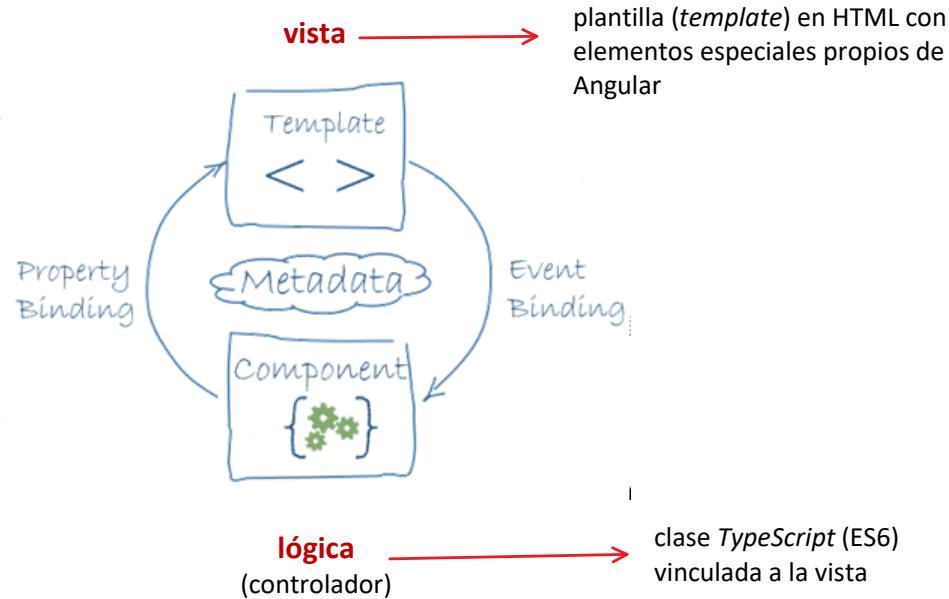
Lunes, 11 de septiembre de 2017 22:49



Componentes

lunes, 11 de septiembre de 2017 22:52

Un componente supone una nueva **etiqueta HTML** con una vista y una lógica definidas por el desarrollador



Estructura de un componente

sábado, 14 de octubre de 2017 20:30

Estructura de un componente

- importación
- decoración
 - selector
 - template / templateUrl
 - ...
- clase exportada

Importamos al menos la clase *Component* de Angular

```
import { Component } from '@angular/core';
```

Dicha clase puede usarse en forma de decorador

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
)  
export class AppComponent {  
  ...  
}
```

Nombre mediante el que podrá ser importada para que forme parte de un módulo
(suele coincidir con el nombre del fichero)

Este decorador concreto admite determinados metadatos

```
@Component({  
  selector  
  template  
  templateUrl  
  styles  
  styleUrls  
  encapsulation  
  animations  
})
```

Otros metadatos

```
changeDetection  
viewProviders  
moduleId  
interpolation  
entryComponents  
preserveWhitespaces  
  
// inherited from core/Directive  
host  
providers  
exportAs  
queries
```

Ya no se utilizan los metadatos

```
directives  
pipes  
inputs  
outputs
```

<https://angular.io/api/core/Component>

Vistas HTML

lunes, 11 de septiembre de 2017 22:54

La **vista** del componente (**HTML**) se genera en función de dos elementos

- su estado, definido por el valor de los **atributos de la clase** en un determinado momento
- la plantilla o *template* que tiene asociado el componente, donde además de HTML puede haber referencia a dichos atributos

```
export class AppComponent {  
    name = 'Curso de Angular';  
    imgUrl = "assets/logo.jpg";  
}
```

Propiedades de
la clase

Esa relación se
denomina "*binding*"

```
<h1>Hola {{name}}!</h1>  
<img [src]="imgUrl"/>
```

Uso de esas
propiedades
desde la vista

Recursos de la aplicación

lunes, 11 de septiembre de 2017 23:19

Los recursos (imágenes, fonts..) deben colocarse en una carpeta **src/assets** para que se copien en el **build**



Creación de componentes

Lunes, 11 de septiembre de 2017 23:21

Utilizamos angular-cli para generar la base de un nuevo componente

```
ng g(enerate) c(omponent) "nombre"
```

Se habrá añadido directamente en el módulo

- como import
- como declaration
- Dispondremos de una carpeta con la estructura de archivos.

Inicialmente crearemos un componente "dumpty":
stateless, incluso sin lógica ninguna.
Lo incorporaremos al componente principal

Cuando utilizamos angular-cli para generar la base de un nuevo componente,
la clase se crea con una estructura de partida, que es la recomendada
en todos los componentes

```
export class <nombre> implements OnInit {
    constructor() {} : inyección de dependencias
    ngOnInit() {} : inicialización de valores

}

export class FeatureComponent implements OnInit {
    constructor() {}

    ngOnInit() []
}
```

Ejemplo: creación de 1 componente

lunes, 11 de septiembre de 2017 23:22

app-root

Hola Mundo 2 componentes

app-pie

Creamos una nueva aplicación

ng new hola-componentes

Modificamos el componente principal como en casos anteriores

Creamos un nuevo componente

ng g c pie

En el módulo principal se añade

```
import { PieComponent } from './pie/pie.component';
@NgModule({
  declarations: [
    AppComponent,
    PieComponent
  ],
  ...
})
```

Cambiamos la lógica (*controller*) de nuestro componente

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-pie',
  templateUrl: './pie.component.html',
  styleUrls: ['./pie.component.css']
})
export class PieComponent implements OnInit {
  public formador: string
  public empresa: string
  public fecha: string

  constructor() {}

  ngOnInit() {
    this.formador = "Alejandro Cerezo Lasne"
    this.empresa = "Icono Training Consulting"
    this.fecha = "2017"
  }
}
```

Cambiamos el contenido de la vista de nuestro componente

```
<footer>
  <p>{{formador}} - {{fecha}}</p>
  <p>{{empresa}}</p>
</footer>
```

Cambiamos el CSS específico de nuestro componente

```
footer {
  position: fixed;
  bottom : 0;
  width: 100%;
  border-top: 1px papayawhip solid
}
p {
  text-align: center;
  font-size: 1.3em;
  color : papayawhip;
}
```

Consumimos el componente <app-pie> desde la vista del componente principal

```
<header style="text-align:center">
  <h1>
    Bienvenidos al curso {{curso}}!
  </h1>
  
</header>
<app-pie></app-pie>
```

Bienvenidos al curso Angular 2.x!



Versión Angular2 del tradicional "Hola Mundo" con un pie creado como componente independiente

Alejandro Cerezo Lasue - 2017
Icono Training Consulting

Ejercicio: Módulos y componentes

sábado, 23 de septiembre de 2017 20:39

Módulos y componentes

- App -> app-main

► Shared ->cabeza, pie



app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { SharedModule } from './shared/shared.module';

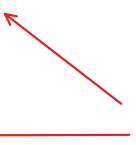
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    SharedModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



El módulo *shared* es incluido / vinculado en el módulo principal

shared.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { CabezaComponent } from './cabeza/cabeza.component';
import { PieComponent } from './pie/pie.component';
@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [
    CabezaComponent,
    PieComponent],
  exports: [
    CabezaComponent,
    PieComponent
  ]
})
export class SharedModule { }
```



Los componentes del módulo *shared* son declarados y referenciados como exportables

Ciclo de vida de los componentes

miércoles, 13 de septiembre de 2017 22:20

I'm Todd, a Developer Advocate @Telerik. Founder of @UltimateAngular Creator of the ngMigrate. JavaScript, Angular, React, conference speaker. Developer Expert at Google.

ULTIMATE ANGULAR

Master Angular 1.x and Angular 2 with me online

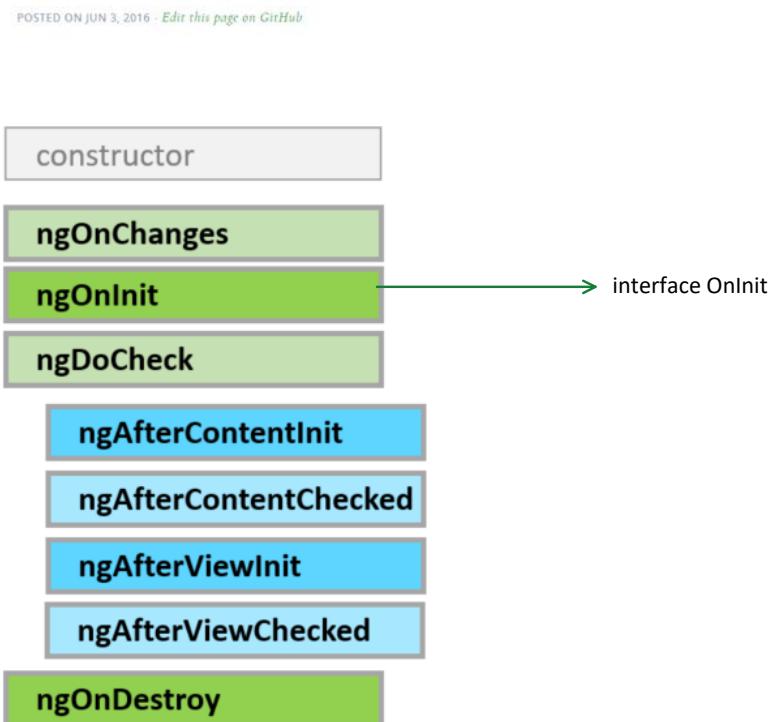
Limited Angular 2 preorders now available.
Master the latest Angular 1.5 components, or preorder the most in-depth Angular 2 courses.

See the courses >

Implementado en Angular 1.5

Lifecycle hooks in Angular 1.5

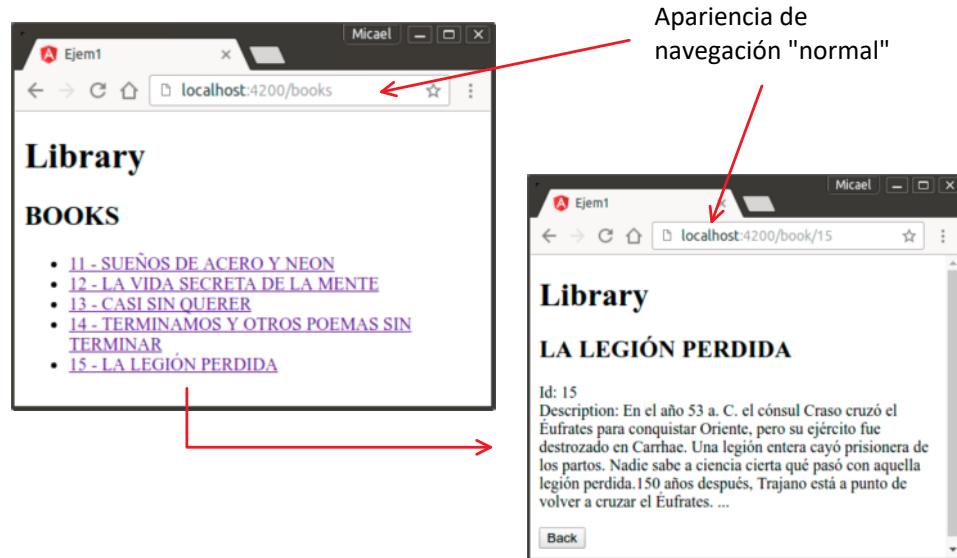
<https://toddmotto.com/angular-1-5-lifecycle-hooks>



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>



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)`

Procedimiento (1)

miércoles, 27 de septiembre de 2017 20:36

Definición de las Rutas

```
app.routing.ts
```

Para cada URL se indica un nombre y el componente que será visualizado

valor por defecto si no se indica ninguna ruta

valor si se indica cualquier ruta distinta de las anteriores

```
import { AboutComponent } from './about/about.component';
import { EnlacesComponent } from './enlaces/enlaces.component';
import { AutoresComponent } from './autores/autores.component';
import { CatalogoComponent } from './catalogo/catalogo.component';
import { HomeComponent } from './home/home.component';
import { RouterModule, Routes } from '@angular/router';
// cada ruta se identifica por su path y su componente
// en este ejemplo inicio, catalogo, autores, enlaces, about
const routes: Routes = [
  { path: 'inicio', component: HomeComponent },
  { path: 'catalogo', component: CatalogoComponent },
  { path: 'autores', component: AutoresComponent },
  { path: 'enlaces', component: EnlacesComponent },
  { path: 'about', component: AboutComponent },
  { path: '', pathMatch: 'full', redirectTo: 'inicio' },
  { path: '**', redirectTo: 'inicio' }
];
export const appRouting = RouterModule.forRoot(routes);
```

"Configuración"

La definición de rutas puede hacerse en

- un fichero de rutas incorporado al módulo principal
(después de crearlo puede completarse con el snippet Routes)
- un módulo de enrutamiento que define las rutas
(es así como lo hace angular cli, cuando se utiliza ng new --routing)

Configuración del módulo

```
app.module.ts
```

```
// Modulos de Angular
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { RouterModule } from '@angular/router';
import { NgModule } from '@angular/core';
// Modulos propios
import { SharedModule } from './shared/shared.module';
import { appRouting } from './app.routing';
;
// Componentes
...
@NgModule({
  declarations: [
    ... componentes ...
  ],
  imports: [
    BrowserModule,
    FormsModule,
    appRouting,
    SharedModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Las rutas de consideran un módulo que debe importarse en la aplicación

Procedimiento (2)

jueves, 14 de septiembre de 2017 21:27

Componente principal

La vista (*template*) del componente principal define la posición de la "salida" del enrutado "router outlet"

app.component.ts

```
<header>
  <h1 class="title">Library</h1>
</header>
<router-outlet></router-outlet>
<footer>
  <p>...</p>
</footer>
```

Se pueden refactorizar como componentes

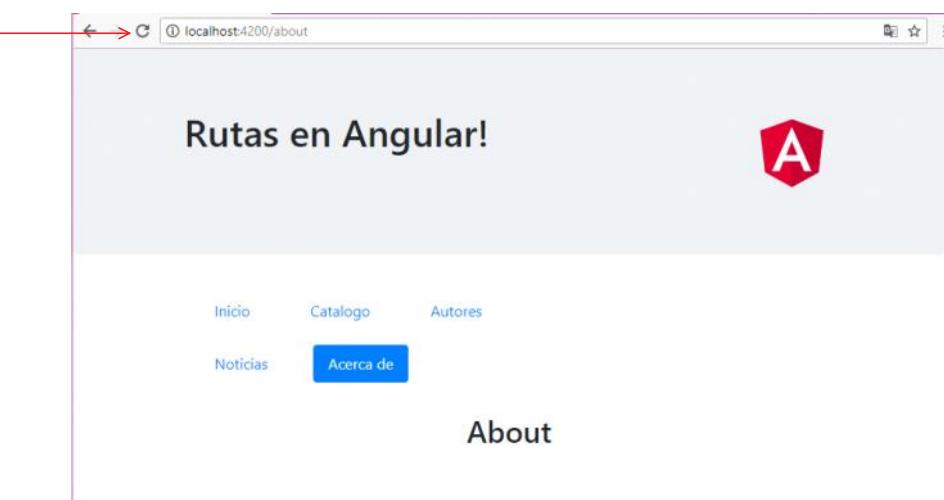
Enlaces a las rutas

app.component.ts

```
<nav>
  <ul>
    <li><a routerLink="inicio" routerLinkActive="active">Inicio</a></li>
    <li><a routerLink="catalogo" routerLinkActive="active">Catálogo</a></li>
    <li><a routerLink="autores" routerLinkActive="active">Autores</a></li>
    <li><a routerLink="enlaces" routerLinkActive="active">Enlaces</a></li>
    <li><a routerLink="about" routerLinkActive="active">Acerca de</a></li>
  </ul>
</nav>
```

Aplica a la ruta activa la clase active para que resalte frente a las otras rutas

Además, indicando el nombre de la ruta en la barra del navegador también es posible acceder al destino



Ejemplo

miércoles, 27 de septiembre de 2017 20:57



[routerLinkActive]="['active']"

Desde <<https://stackoverflow.com/questions/35422526/how-to-set-bootstrap-navbar-active-class-in-angular-2>>

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 mencionan 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 Modulo
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    appRouting,
    SharedModule
  ],
})
```

No se mencionan los módulos enrutados "perezosamente"

Entorno de Testing

viernes, 26 de enero de 2018 17:46

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 entorno 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



→ ejecutador de pruebas unitarias
<https://karma-runner.github.io/2.0/index.html>



→ El framework para la declaración (programación) de las pruebas en JS
<https://jasmine.github.io/>



→ ejecutador de pruebas de integración (end-to-end, e2e)
<http://www.protractortest.org/#/>

Basado en *SeleniumHQ*
<http://www.seleniumhq.org/>

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**

Pruebas unitarias

viernes, 26 de enero de 2018 18:13

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("expresión")` 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

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';  
import { PieComponent } from './pie.component';  
  
Suite de pruebas → describe('PieComponent', () => {  
    let component: PieComponent;  
    let fixture: ComponentFixture<PieComponent>;  
  
    beforeEach(async(() => {  
        TestBed.configureTestingModule({  
            declarations: [ PieComponent ]  
        }).compileComponents();  
    }));  
  
    beforeEach(() => {  
        fixture = TestBed.createComponent(PieComponent);  
        component = fixture.componentInstance;  
        fixture.detectChanges();  
    });  
  
    it('should create', () => {  
        expect(component).toBeTruthy();  
    });  
});  
  
Preparación → Preparación del "lecho" en el que se instanciará el componente  
Configuración del módulo de pruebas → Configuración del componente  
Pruebas it de diversas funcionalidades → Pruebas it de diversas funcionalidades
```

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

TestBend -> 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-configuration 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 *TestBend* un componente de la clase que estamos probando

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';
```

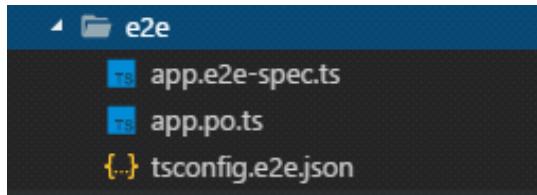
```
let elem: DebugElement;
elem = fixture.debugElement.query(By.css('#test'));
```

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();
  }
}
```

Plantillas (Templates)

Las plantillas (*templates*) permiten definir la vista en función de la información del componente

En todas las directivas es posible añadir el prefijo data- para que las directivas no supongan problema de validación

Desarrollo declarativo

expansión de las características del HTML, añadiéndole funcionalidades sin necesidad de escribir código JavaScript

Se puede ver como una forma de agregar valor semántico al HTML.

Lenguaje de plantillas

```
 {{user.name}}  
 href = {{miUrl}}  
 [href] = "miUrl"  
 (click)="usarBoton()"
```

[] a cualquier atributo HTML se le asigna una variable del componente

() a cualquier evento HTML se le asigna un método del componente

{{}} se interpola una variable

[()] two way binding

se declara una variable local en la vista

- Expresiones
- Directivas estructurales
 - Visualización condicional
 - Repetición de elementos
- Directivas y Estilos CSS

lenguaje de plantillas mediante {{}}

atributos *ng-específicos de angular que se pueden asignar a etiquetas HTML.

atributos ng que gestionan dinámicamente los estilos y clases CSS

Formularios

<https://angular.io/docs/ts/latest/guide/template-syntax.html>

Eventos

domingo, 24 de septiembre de 2017 11:57

Otro elemento clave para entender el funcionamiento de las vistas son los eventos del sistema

- proporcionados por el navegador, como interfaz con el S.O. y
- definidos en HTML5

Eventos del sistema (1)



Evento	Descripción	Elementos para los que está definido
blur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
change	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
click	Pinchar y soltar el ratón	Todos los elementos
dblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
focus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
keydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
keypress	Pulsar una tecla	Elementos de formulario y <body>
keyup	Soltar una tecla pulsada	Elementos de formulario y <body>
load	La página se ha cargado completamente	<body>

Eventos del sistema (2)

Evento	Descripción	Elementos para los que está definido
mousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
mousemove	Mover el ratón	Todos los elementos
mouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
mouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
mouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
reset	Inicializar el formulario (borrar todos sus datos)	<form>
resize	Se ha modificado el tamaño de la ventana del navegador	<body>
select	Seleccionar un texto	<input>, <textarea>
submit	Enviar el formulario	<form>
unload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Nuevos eventos en HTML5

Window

onafterprint
onbeforeprint
onbeforeunload
onerror
onhaschange
onmessage
onoffline
ononline
onpagehide
onpageshow
onpopstate
onredo
onresize
onstorage
onundo

Form

oncontextmenu
onformchange
onforminput
oninput
oninvalid

Mouse

ondrag
ondragend
ondragenter
ondragleave
ondragover
ondragstart
ondrop
onmousewheel
onscroll

oncanplay

oncanplaythrough
ondurationchange
onemptied
onended
onerror
onloadeddata
onloadedmetadata
onloadstart
onpause
onplay
onplaying
onprogress

Media Events

onratechange
onreadystatechange
onseeked
onseeking
onstalled
onsuspend
ontimeupdate
onvolumechange
onwaiting

Gestión de eventos

domingo, 15 de octubre de 2017 9:36

El operador () indicando su nombre dentro de los paréntesis, permite definir el manejador de cualquier evento estándar de un determinado elemento del DOM.

El objeto **\$event**, muy similar al utilizado en JQuery, se envía como parámetro al manejador de evento que se defina

```
(click) = "btnResponder($event)"  
          ↓  
  btnResponder (oEvent) {  
    ... // respuesta al evento  
    console.log(oEvent) →  
  }
```

- aunque no es una práctica recomendada, puede ser una expresión asociada al evento
`(click) = "++nCount"`
- una llamada a una función definida en la clase que constituye el componente
`(click) = "setContador(2)"`

```
▼ MouseEvent {isTrusted: true, screenX: 2232, screenY: 592  
  altKey: false  
  bubbles: true  
  button: 0  
  buttons: 0  
  cancelBubble: false  
  cancelable: true  
  clientX: 493  
  clientY: 401  
  composed: true  
  ctrlKey: false  
  currentTarget: null  
  defaultPrevented: false  
  detail: 1  
  eventPhase: 0  
  fromElement: null  
  isTrusted: true  
  layerX: 493  
  layerY: 401  
  metaKey: false  
  movementX: 0  
  movementY: 0  
  offsetX: 24  
  offsetY: 8  
  pageX: 493  
  pageY: 401  
  ▶ path: (9) [button, form.ng-valid.ng-dirty.ng-touched,  
  relatedTarget: null  
  returnValue: true  
  screenX: 2232  
  screenY: 592  
  shiftKey: false  
  ▶ sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: true}  
  ▶ srcElement: button  
  ▶ target: button  
  timeStamp: 196737.7700000002  
  ▶ toElement: button}
```

Propiedades y Directivas

lunes, 2 de octubre de 2017 21:59

Angular permite manipular las propiedades del DOM, aunque aparentemente haga referencia a los atributos HTML.

Hay que recordar que los atributos HTML suelen reflejarse en las correspondiente propiedades del DOM, pero en algunos casos esta relación no es tan directa.

En AngularJS las propiedades de los elementos del DOM se manipulaban mediante directivas



Existía un gran número de directivas

A partir de Angular 2 se utiliza el operador [] que asocia cualquier propiedad de un elemento con una variable del modelo



existen muy pocas directivas

Ejemplos

Directivas y referencias



[src] indica la *url* del fichero que actúa como fuente para una etiqueta **img**, sustituyendo el habitual atributo *src*

En lugar de `

Respuesta a eventos (*Event Binding*)
`<button (click)="setName('Pepe')">`

Datos enlazados (*two-way data binding*)
`<input type="text" [(ngModel)]="name">
{name}`

Ejemplo

```
<input type="text" id="nombre" name="nombre"
[value]= "sNombre">

<button (click)="sNombre=''">Borrar</button>

<p>Hola {{sNombre}}</p>
```

En el controlador

```
this.nombre = 'Pepe';
```

Usando Bindings básicos

Dime tu nombre

Hola Pepe

El valor inicial de la propiedad se establece accediendo a la variable *sNombre* del controlador

En respuesta al evento clic se modifica el valor de la variable *sNombre*.

Una expresión refleja el valor de la variable *sNombre*.

Usando Bindings básicos

Dime tu nombre

Hola

Al modificar la variable *sNombre* se refleja en la vista

Usando Bindings básicos

Dime tu nombre

Hola Pepe

Al modificar la vista NO se refleja en la variable del modelo/controlador

en la vista

variable del modelo/controlador

Se trata de un binding en una sola dirección

ngModel y doble binding

domingo, 24 de septiembre de 2017 11:11

Directiva ngModel



relaciona elementos del DOM con modelos de datos, informando al compilador HTML para que tome una variable del modelo.

- Se utiliza en los controles de formulario, como INPUT, SELECT, TEXTAREA o controles personalizados.
- la notación [] enlaza (*binding*) una **propiedad** de la clase que define al componente con el correspondiente control de formulario de la vista

```
<input type="text" [ngModel]="name">
```

además de su funcionamiento normal, en una dirección, cuando se invoca como propiedad, este "enlace" puede funcionar en las dos, cuando se combina con el uso de eventos ()

En realidad, la directiva *ngModel* está agrupando 2 operaciones

```
<input type="text" id="nombre"
       [value] = "sNombre"
       (input) = 'sNombre = $event.target.value'>
```

Doble binding



combinación de

- la directiva *ngModel*
- el evento de Angular *ngModelChange*

```
<input type="text" id="nombre"
       [ngModel] = "sNombre"
       (ngModelChange) = 'sNombre = $event'>
```

Al no ser un evento del sistema, cambia el valor de \$event

Forma abreviada de escribirlo [...]



"banana in a box"

```
<input type="text" id="nombre"
       [(ngModel)] = "sNombre">
```

Sirve de base al doble binding

<https://angular.io/guide/template-syntax>

Ejemplo

```
<div>
  <h2>Usando "[()]"</h2>
  <form>
    <label for="nombre">Dime tu nombre </label>
    <input type="text" id="nombre" name="nombre"
           [(ngModel)] = "nombre">
    <button (click)='btnBorrar($event)'>Borrar</button>
  </form>
  <p>Hola {{nombre}}</p>
</div>
```

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

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

Usando "[0]"

Dime tu nombre

Hola Pepe

Usando "[0]"

Dime tu nombre

Hola Luis

```

    selector: 'app-formulario-bnb',
    templateUrl: './formulario.component.html',
    styleUrls: ['./formulario.component.css']
})
export class FormularioBnbComponent implements OnInit {
  public nombre: string;
  constructor() { }
  ngOnInit() {
    this.nombre = '';
  }
  btnBorrar (evento) {
    this.nombre = '';
    console.log(evento);
  }
}

```

The screenshot shows a user interface with a red border. At the top, there is a text input field labeled "Dime tu nombre" containing the text "Luis". To the right of the input field is a button labeled "Borrar". Below the input field, the text "Hola Luis" is displayed in a larger font. A red arrow points from the text "Hola Luis" back towards the code snippet above, indicating that the change in the view (the output "Hola Luis") reflects the change in the model (the variable "nombre" being set to "Luis").

En este caso, al modificar la vista SI se refleja en la variable del modelo/controlador



Expresiones

- Lógica limitada: no se pueden incluir en ellas condicionales (excepto el operador ternario), bucles o excepciones
- Se les puede dar formato mediante filtros

Referencias a modelos

`{{Dato}}`

Operaciones
aritméticas básicas

`{{Dato + 4}}`

Empleo de los métodos de
los objetos envolventes de
JS (e.g. String)

`>{"Beginning AngularJS".toUpperCase()}`
`{"ABCDEFG".indexOf('D')}`

Uso del operador ternario

`{{Dato == 1 ? "Red" : "Blue"}}`



Ejemplos de expresiones

Ejemplos de Expresiones

Operaciones aritméticas básicas

$6 + 4 = 10$

El resultado se obtiene con la expresión: `{(6 + 4)}`

Empleo de los métodos de los objetos envolventes de JS (e.g. String)

BEGINNING ANGULARJS

Resultado de la expresión: `{"Beginning AngularJS".toUpperCase()}`

3

Resultado de la expresión: `{"ABCDEFG".indexOf('D')}`

Uso del operador ternario

Red

Resultado de la expresión: `{(1==1 ? "Red" : "Blue")}`

Alejandro L. Cerezo - Madrid 2015

Referencias locales en plantillas

lunes, 2 de octubre de 2017 22:29

Son variables que a nivel de la plantilla hacen referencia a un elemento del DOM, sea un estándar HTML o un componente, permitiendo manipular su valor desde la propia plantilla en respuesta a determinados eventos.

```
<div>
  <h2>Usando #</h2>
  <form>
    <label for="nombre">Dime tu nombre (y pulsa enter)</label>
    <input type="text" id="nombre" name="nombre" #nombre>
    <button (click)="nombre.value = ''">Borrar</button>
  </form>
  <p>Hola {{nombre.value}}</p>
</div>
```

referencia local el elemento input

Las referencias locales pueden ser accedidas desde la vista gracias al decorador @ViewChild

```
@ViewChild("<referencia>") <variable>: ElementRef;
```

El tipo corresponde a la referencia general a cualquier elemento del DOM

Este decorador suele utilizarse en la gestión de los formularios, como veremos.

Existen los decoradores
@ViewChild / @ViewChildren
@ContentChild / @ContentChildren

Los segundos están relacionados con el acceso a elementos procedentes de la proyección de contenidos (transclusión de AngularJS)

Encapsulación de la vista

domingo, 15 de octubre de 2017 10:27

```
@Component({  
  selector:  
  templateUrl:  
  ...  
  encapsulation:  
  ...  
})
```

Metadato que define el tipo de encapsulación de la vista que se utilizará

ViewEncapsulation.Native
ViewEncapsulation.Emulated
ViewEncapsulation.None.

ViewEncapsulation.Native

Utiliza el Shadow DOM definido en el nuevo estándar de *Web Components*

- los estilos del componente son totalmente independiente
- los estilos del componente no son visibles fuera de el
- los estilos externos NO se aplican en el componente

Puede no ser reconocido por todos los navegadores

Es interesante en componentes que se van a reutilizar y que contienen todas sus reglas de estilo

ViewEncapsulation.Emulated

Emula en cierto modo el anterior aplicando CSS estándar

- es el valor por defecto
- los estilos del componente no son visibles fuera de el
- los estilos externos SI se aplican en el componente, siempre que no sean sobreescritos por estilos internos

ViewEncapsulation.None.

No hay ninguna encapsulación

Comunicación entre componentes

miércoles, 13 de septiembre de 2017 19:22

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

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

Configuración de propiedades

miércoles, 13 de septiembre de 2017 19:54

(Padre → Hijo)

El componente padre puede especificar propiedades en el componente hijo como si fuera un elemento nativo HTML

vista
(padre) → <hijo [title]='appTitle'></hijo>

El valor de title en el hijo corresponderá a la propiedad appTitle en el padre

class controller:
(hijo)

@Input()
private title: string;

vista
(hijo)

<h1>{{title}}</h1>

Envío de eventos

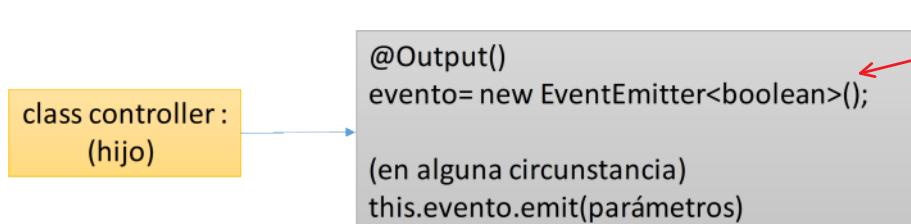
miércoles, 13 de septiembre de 2017 19:56

(Hijo → Padre)

El componente hijo puede generar eventos que son atendidos por el padre como si fuera un elemento nativo HTML

El padre se suscribe al evento : le asigna una función manejadora.
La variable \$event apunta al evento generado

vista
(padre) → <header (evento)='hiddenTitle(\$event)'></header>



Ejemplo de comunicación entre componentes



Cuándo crear un componente

miércoles, 13 de septiembre de 2017 22:18

- Cuando la lógica y/o el *template* sean suficientemente complejos
- Cuando los componentes hijos puedan reutilizarse en varios contextos

Los ejemplos del curso (y otros similares) suelen ser demasiado sencillos para que compense la creación de componentes hijos

Directivas de Angular

domingo, 15 de octubre de 2017 10:27

son atributos específicos de angularJS que se pueden asignar a cualquier etiqueta HTML.

- El atributo o etiqueta se denomina `ng-nombre`;
- el método asociado `ngNombre`

un elemento del DOM queda "marcado" para que Angular le asigne un determinado comportamiento, que puede suponer incluso una transformación de ese elemento del DOM o alguno de sus hijos



Si modifican la estructura del DOM se denominan **directivas estructurales**.
Su nombre comienza por *

Directivas estructurales

**NgFor
*NgIf
NgSwitch

Otras directivas

*NgStyle
NgClass
NgNonBindable*

Iteraciones



*ngFor

Directiva estructural que genera el recorrido a una colección (un *array* o un objeto del modelo) indicándole la variable local a su ámbito donde se almacenará el elemento actual de cada iteración.

en la clase aElementos = [.....]

Existe un *array* en el modelo

```
<tag_html *ngFor="(let) elemento of aElementos | filtros | ordenación">  
  ... {{elemento}} ...  
</tag_html>
```

Similitud con el bucle **for ... in** de JS

Más adelante veremos las opciones de filtrado y ordenación

<https://angular.io/docs/ts/latest/api/common/index/NgFor-directive.html>

Ámbito de las iteraciones



Técnicamente, el código HTML iterado tendrá un ámbito (*scope*) local

- en él se pueden definir propiedades específicas de este ámbito, que se inicializan mediante let
- además existen una serie de propiedades ya predefinidas que toman valor dinámicamente conforme se realizan las sucesivas iteraciones y que pueden ser recogidas en propiedades del controlador

La más importante es **index** que devuelve en cada momento el índice en el recorrido del elemento actual sobre el que se está iterando

```
*ngFor="let elemento of aElementos; let i = index">
```

Scope de las iteraciones



El conjunto de las variables que almacena automáticamente el ámbito (*scope*) de una iteración es el siguiente

index	Number	Iterator offset of the repeated element (0..length-1)
first	Boolean	True, if the repeated element is first in the iterator
middle	Boolean	True, if the repeated element is between first and last in the iterator
last	Boolean	True, if the repeated element is last in the iterator
even	Boolean	True, if the iterator position \$index is even (otherwise, false)
odd	Boolean	True, if the iterator position \$index is odd (otherwise, false)

Pensamientos: ejemplo de iteraciones

Conforme añadimos ideas, creamos un *array* que mostramos, iterando sobre el, en la parte interior

Junto con la iteración, vemos de nuevo como funciona el doble *binding*.

Pensamientos

Indica tu nombre

Comparte una idea

Hola Pepe.

Pensamientos que has tenido

- Tu pensamiento numero 1 fue: "Una idea"
- Tu pensamiento numero 2 fue: "Segunda idea"
- Tu pensamiento numero 3 fue: "No se me ocurre nada"
- Tu pensamiento numero 4 fue: "Ya termine"

Alejandro L. Cerezo - Madrid 2015

Condiciones: *ngIf*



Se puede controlar si un elemento aparece o no en la página dependiendo del valor de un atributo de la clase usando la directiva `ngIf`

- dependiendo del valor del atributo booleano `visible`

```
<p *ngIf="visible">Text</p>
```

- dependiendo de una **expresión**

```
<p *ngIf="num == 3">Num 3</p>
```

Combinando ngFor / ngIf



No se pueden incluir dos directivas estructurales (de tipo *) en el mismo elemento

```
<li *ngFor="let elem of elems" *ngIf="elem.check">  
  {{elem.desc}}  
</li>
```

La solución más simple es anidar elementos HTML (divs), cada uno con su directiva

También es posible usar la versión de las directivas sin el azúcar sintáctico (*), en su versión extendida con el elemento *template* (que no aparece en el DOM)

```
<template ngFor let-elem [ngForOf]="elems">  
  <li *ngIf="elem.check">{{elem.desc}}</li>  
</template>
```

<https://github.com/angular/angular/issues/4792>

Angular 4

miércoles, 04 de octubre de 2017 9:05

```
<div *ngIf="aldeas.length > 0; then ideasTemplate else vacioTemplate"></div>

<ng-template #ideasTemplate class="lista">
  <h2>Lista de ideas</h2>
  <ul><li *ngFor="let idea of aldeas">{{idea}}</li></ul>
</ng-template>

<ng-template #vacioTemplate>
  <p>Escribe alguna idea</p>
</ng-template>
```

Muestrario: mostrar y ocultar (1)

Uso de la directiva

***ngIf**

- asociándola al evento clic en un botón
- asociándola al paso del ratón sobre una imagen

Separamos el *footer* del fichero principal y lo incorporamos con otro componente

Muestrario

Ratones inalambricos disponibles en los siguientes colores



Ocultar Colores Disponibles

Rojo

Verde

Azul

ngSwitch / *ngSwitchCase

martes, 17 de octubre de 2017 22:37

```
<ul *ngFor="let person of usuarios"
    [ngSwitch]="person.country">
    <li *ngSwitchCase="'UK'"
        class="text-success">>{{ person.name }} ({{ person.country }})
    </li>
    <li *ngSwitchCase="'USA'"
        class="text-primary">>{{ person.name }} ({{ person.country }})
    </li>
    <li *ngSwitchCase="'SP'"
        class="text-danger">>{{ person.name }} ({{ person.country }})
    </li>
    <li *ngSwitchDefault
        class="text-warning">>{{ person.name }} ({{ person.country }})
    </li>
</ul>
```

ngSwitch define la variable
que determinara los casos

ngSwitchCase define
cada uno de los casos

Ejercicio

lunes, 16 de octubre de 2017 22:51

Entrada de datos:

- Autor →
- Titulo

Creación de una lista

- ngFor
- ngIf

The screenshot shows a web application interface. At the top right is the logo for "Icono TRAINING CONSULTING". The main title is "Libros!". Below it, there's a section titled "Datos del libro" with fields for "Titulo" and "Autor", each with an associated input field. Underneath are two buttons: "Añadir" and "Borrar". To the right of this is a section titled "Lista de Libros" containing a list of books: "Neuromante (William Gibson)" and "Fundación (Isaac Asimov)".

Alejandro Cerezo Lasne - 2017
Icono Training Consulting

Estilos iniciales



Existen varias formas de definir inicialmente un CSS en Angular 2

- **Globalmente** en el **fichero css** asociado al index.html
- Local al componente:
 - En la propiedad styles de @Component
 - En el **fichero css** definido en styleUrls de @Component
 - En el template

Gestión de estilos



Directamente

- Asociar un estilo concreto de un elemento a un atributo

Mediante clases (Buena práctica)

- Asociar la clase de un elemento a un atributo de tipo string
- Activar una clase concreta con un atributo boolean
- Asociar la clase de un elemento a un atributo de tipo objeto (mapa de string a boolean)



Style (1)

[style.<nombre>]

permite asignar a un elemento del DOM una propiedades de estilo almacenadas como un string en el modelo de la aplicación

```
<p [style.color] ="estiloColor">
```

Se pueden indicar fácilmente las unidades

```
<p [style.fontSize.em] ="pSizeEm">Text</p>
```

```
<p [style.fontSize.%] ="pSizePerc">Text</p>
```



Style(2)

[ngStyle]

permite asignar a un elemento del DOM una o varias propiedades de estilo almacenadas como un objeto en el modelo de la aplicación

```
<p [ngStyle] ="estiloColor">
```

En el controller:

```
estilos = {"bachgrouun-color" : "green",  
          "color": "silver"}
```

Como en cualquier otro contexto CSS, no es una buena práctica la aplicación directa de estilos, siendo más recomendable la aplicación de las clases adecuadas a cada caso

class (1)



[class] permite asignar a un elemento del DOM una clase mediante expresiones que hacen referencia al modelo. De esa forma al modificar el modelo se aplican clases diferentes y se modifica el aspecto del elemento

La propiedad del modelo puede tener varios formatos:

- una **cadena de caracteres** con uno o más nombres de clases

```
<h1 [class]="nombreClase">Title!</h1>
```

class controller : → nombreClase =..."

El cambio del valor de la variable se refleja en la aplicación de diferentes clases dinámicamente

class (2)



[class.<nombre>] = "boolean"

Es posible activar una clase concreta con un atributo boolean y se puede hacer con diversas clases

```
<h1      [class.red]="redActive"
           [class.yellow]="yellowActive">
    Title!
</h1>
```

class controller :

redActive = "true"
yellowActive = "false"

class (3)



[ngClass] permite asignar a un elemento del DOM una clase mediante expresiones más complejas

[ngClass] = "array"

Es posible aplicar diversas clases cuyos nombres se almacenan en un array

```
<h1      [ngClass]="['class1', 'class2', 'class3']"  
Title!</h1>
```

[ngClass] = "objeto"

- las claves permiten especificar nombres de clases y
- sus valores corresponden a expresiones que deben cumplirse para que éstas se apliquen.

```
<p [ngClass]="{positivo: total>=0, negativo: total<0}">
```

The screenshot shows a presentation slide with a yellow textured background. In the top left corner is the Angular logo (a red hexagon with a white stylized letter 'A'). To its right, the title 'Acumulador con clases' is displayed in a large, bold, blue font. Below the title is a dark red button with the text 'Acumulador con clases' in white.

On the left side of the slide, there is a section titled 'Ejemplos de clases que se aplican dinámicamente' (Dynamic class application examples). It contains a bulleted list:

- en respuesta a una selección por el usuario
- en función del valor de un dato del modelo

To the right of this list is a demonstration area. It starts with a heading 'Acumulador' and a dropdown menu 'Elige en tamaño de letra del titular' set to 'Grande'. Below this is a section titled 'Control de operación:' containing a numeric input field with the value '10' and two buttons labeled '+' and '-'. Underneath is a section titled 'Totales:' with the text 'En el acumulador llevamos 10'.

At the bottom right of the slide, there is a footer with the text 'Alejandro L. Cerezo', 'CLE Formación', and 'Madrid - 2015'.



Lista de Tareas / Componentes

A screenshot of a web browser window titled "127.0.0.1:8080". The page has a header "A checklist" and a message "There are no items yet.". Below is a form with a text input "Enter the description..." and a "Add" button.

Cada tarea será
un componente

Refactorizamos la
aplicación de
gestión de tareas

A screenshot of a web browser window titled "127.0.0.1:8080". The page has a header "A checklist" and displays a list of tasks: "Leche" (checked), "Pan" (checked), and "Galletas" (unchecked). Each task has a "Delete" button next to it. Below the list is a form with a text input "Vino" and a "Add" button.

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.concretewebpage.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 injectado en 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

jueves, 14 de septiembre de 2017 13:41

Filter/Pipe	Angular 1.x	Angular 2	
<code>currency</code>	✓	✓	definen el aspecto de los valores monetarios
<code>date</code>	✓	✓	definen en diversos detalles el formato de una fecha; permiten utilizar una serie de formatos ya predefinidos
<code>lowercase</code>	✓	✓	
<code>uppercase</code>	✓	✓	
<code>titlecase</code>		✓	definen el uso de mayúsculas y minúsculas
<code>json</code>	✓	✓	
<code>limitTo</code>	✓		formatea la salida de un objeto completo
<code>slice</code>		✓	
<code>number</code>	✓		presenta parte de un array
<code>decimal</code>		✓	
<code>percent</code>		✓	
<code>i18nSelect</code>		✓	definen el número de decimales de un dato
<code>i18nPlural</code>		✓	
<code>async</code>		✓	
<code>orderBy</code>	✓		
<code>filter</code>	✓		

Formatos de fechas

<code>yyyy</code>	Four-digit representation of year (for example, AD 1 => 0001, AD 2010 => 2010)
<code>yy</code>	Two-digit representation of year, padded (00–99) (for example, AD 2001 => 01, AD 2010 => 10)
<code>y</code>	One-digit representation of year (for example, AD 1 => 1, AD 199 => 199)
<code>MMMM</code>	Month in year (January–December)
<code>MMM</code>	Month in year (Jan-Dec)
<code>MM</code>	Month in year, padded (01–12)
<code>M</code>	Month in year (1–12)
<code>dd</code>	Day in month, padded (01–31)
<code>d</code>	Day in month (1–31)
<code>EEEE</code>	Day in week (Sunday–Saturday)
<code>EEE</code>	Day in week (Sun–Sat)

<code>HH</code>	Hour in day, padded (00–23)
<code>H</code>	Hour in day (0–23)
<code>hh</code>	Hour in AM/PM, padded (01–12)
<code>h</code>	Hour in AM/PM, (1–12)
<code>mm</code>	Minute in hour, padded (00–59)
<code>m</code>	Minute in hour (0–59)
<code>ss</code>	Second in minute, padded (00–59)
<code>s</code>	Second in minute (0–59)
<code>.sss</code>	Millisecond in second, padded (000–999)
<code>o,sss</code>	
<code>a</code>	AM/PM marker
<code>Z</code>	Four-digit (+sign) representation of the time zone offset (-1200 – +1200)
<code>ww</code>	ISO 8601 week of year (00–53)
<code>w</code>	ISO 8601 week of year (0–53)

Atajos

Parámetro	Descripción	Ejemplo
<code>medium</code>	equivalent to 'MMM d, y h:mm:ss a' for en_US locale	Sep 3, 2010 12:05:08 PM
<code>short</code>	equivalent to 'M/d/yy h:mm a' for en_US locale	9/3/10 12:05PM
<code>fullDate</code>	equivalent to 'EEEE, MMMM d, y' for en_US locale	Friday, September 3, 2010
<code>longDate</code>	equivalent to 'MMMM d, y' for en_US locale	September 3, 2010
<code>mediumDate</code>	equivalent to 'MMM d, y' for en_US locale	Sep 3, 2010
<code>shortDate</code>	equivalent to 'M/d/yy' for en_US locale	9/3/10
<code>mediumTime</code>	equivalent to 'h:mm:ss a' for en_US locale	12:05:08 PM
<code>shortTime</code>	equivalent to 'h:mm a' for en_US locale	12:05 PM

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

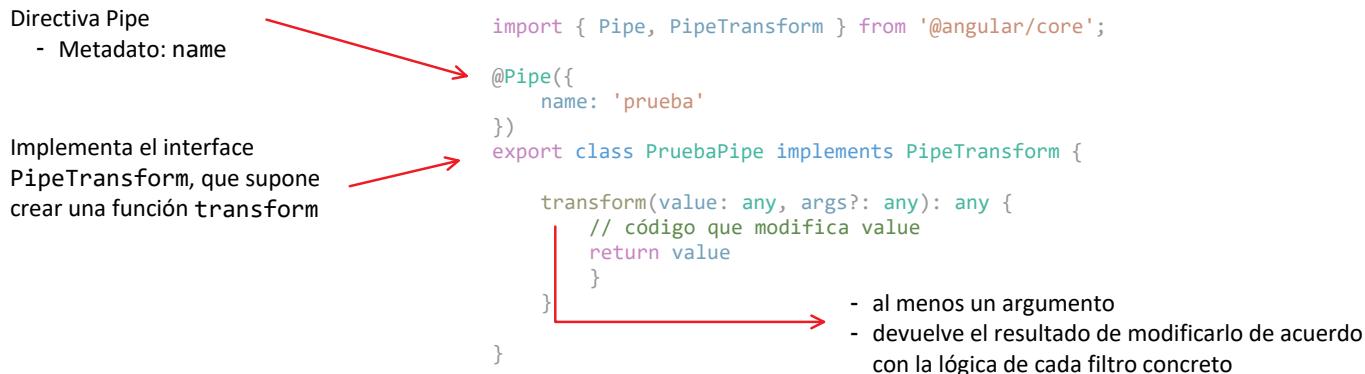
registerLocaleData(localeEs);

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

Pipes i18N

Pipes propios

Lunes, 27 de noviembre de 2017 22:53



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 determinan como se realiza la transformación

Pipe puro

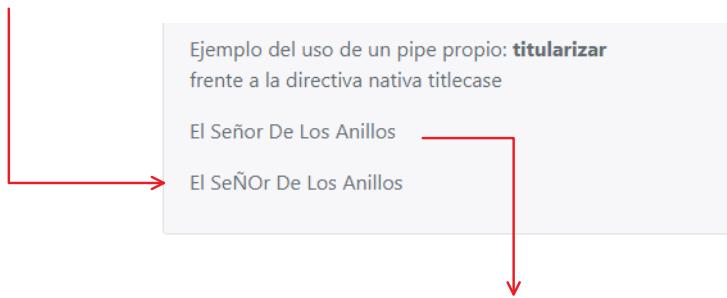
Pipes "con atributos"

↑
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



```
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++) {
      // Si es fin de 'palabra'
      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

Animaciones

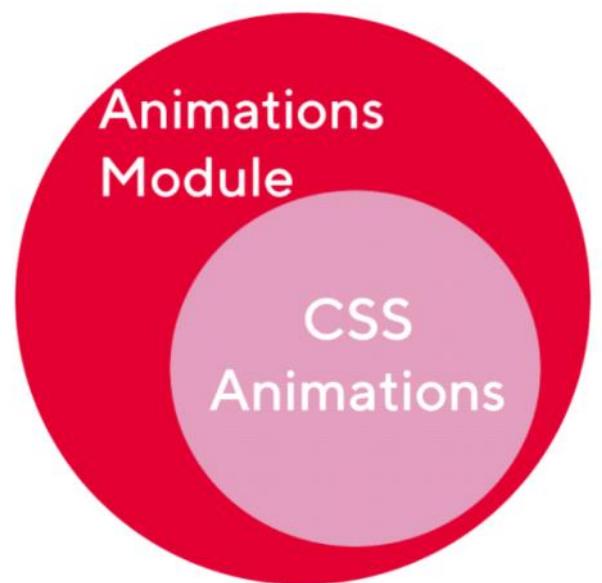
lunes, 29 de enero de 2018 23:01

Desde angular es posible generar directamente las animaciones CSS (en realidad correspondientes al estándar *Web Animations API*)

Las animaciones CSS incluyen

- transiciones
- transformaciones
- animaciones complejas (*keyframes*)

el paso de un conjunto de propiedades CSS a otro siguiendo un determinado patrón a lo largo de un intervalo de tiempo



En Angular se basan en utilizar, en el decorador del componente, el metadato `animations` con las propiedades

trigger,
state,
style,
animate,
transition,
keyframes

funciones temporizadoras
(*timing functions*)

Valor	Descripción
<code>ease</code>	<i>Default value. Specifies a transition effect with a slow start, then fast, then end slowly (equivalent to cubic-bezier(0.25,0.1,0.25,1))</i>
<code>linear</code>	<i>Specifies a transition effect with the same speed from start to end (equivalent to cubic-bezier(0,0,1,1))</i>
<code>ease-in</code>	<i>Specifies a transition effect with a slow start (equivalent to cubic-bezier(0.42,0,1,1))</i>
<code>ease-out</code>	<i>Specifies a transition effect with a slow end (equivalent to cubic-bezier(0,0,0.58,1))</i>
<code>ease-in-out</code>	<i>Specifies a transition effect with a slow start and end (equivalent to cubic-bezier(0.42,0,0.58,1))</i>
<code>step-start</code>	<i>Equivalent to steps(1, start)</i>
<code>step-end</code>	<i>Equivalent to steps(1, end)</i>
<code>steps(int,start/end)</code>	<i>Specifies a stepping function, with two parameters. The first parameter specifies the number of intervals in the function. It must be a positive integer (greater than 0). The second parameter, which is optional, is either the value "start" or "end", and specifies the point at which the change of values occur within the interval. If the second parameter is omitted, it is given the value "end"</i>
<code>cubic-bezier(n,n,n,n)</code>	<i>Define your own values in the cubic-bezier function. Possible values are numeric values from 0 to 1</i>

Procedimiento

lunes, 29 de enero de 2018 21:28

A nivel del módulo principal, registramos el módulo de Angular responsable de las animaciones en el navegador

```
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
...
imports: [
  ...
  BrowserAnimationsModule,
  ...
]
```

A nivel del componente en el que tendrá lugar la animación

```
import {
  Component, OnInit,
  trigger, state, style, transition, animate
} from '@angular/core';
```

Elementos de angular /core relacionados con la animación

Los parámetros de la animación se definen mediante metadatos del decorador del componente

```
@Component({
  ...
  animations: [
    trigger('botonState', [
      state('inactive', style({
        backgroundColor: '#eee',
        transform: 'scale(1)'
      })),
      state('active', style({
        backgroundColor: '#cfcd8dc',
        transform: 'scale(1.2)'
      })),
      transition('inactive=>active', animate('500ms ease-in')),
      transition('active=>inactive', animate('500ms ease-out'))
    ])
  ]
})
```

trigger: permite definir diversas animaciones y asignar a cada una de ellas un nombre que la lance

state: define los distintos estados de la animación

style: cada estado corresponde a un conjunto de valores CSS

transition: define los pasos de un estado a otro

animate: para cada transición, determina los parámetros del proceso de aplicación

En un elemento del DOM se dispara la animación

```
<button class="btn btn-primary"
[@botonState]="boton.state" (click)="toggleState()">
```

se utiliza una propiedad correspondiente al *trigger* de la animación

el valor de la propiedad corresponde a uno de los estados de la animación

en este caso hay un manejador de evento responsable de los cambios en el valor del estado

animación



```
toggleState() {  
    this.boton.state =  
        this.boton.state === 'active' ? 'inactive' : 'active';  
    this.boton.label =  
        this.boton.label === 'Activar' ? 'Desactivar' : 'Activar';  
}
```

Formularios

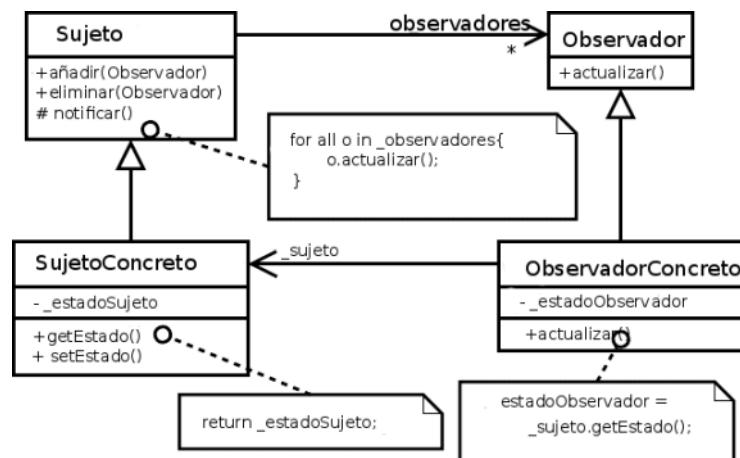
domingo, 8 de octubre de 2017 8:00

Basados en la Vista
(*Template Driven*)

- Similares a los utilizados en AngularJS
- La detección del cambio sigue el patrón *Push*
- La clave está en el doble *binding*
- Se necesita importar el módulo *FormsModule*

Basados en el Modelo
(*Model Driven*)

- Emplean programación reactiva
- La detección del cambio sigue el patrón *Observer (Pull)*
- Se necesita importar el módulo *ReactiveFormsModule*
- Son la opción recomendable en Angular



Elementos de HTML y Bootstrap

martes, 7 de noviembre de 2017 20:56

Elementos de un formulario:

- `input type text...`
 - `text`
 - `password`
 - `email | tel | url`
 - `search`
 - `number | range`
 - `color`
 - `submit`
 - `date | datetime | datetime-local`
 - `month | week | time`
- `textbox`
- `input type checkbox`
- `input type radio`
- `select/options`

Formato básico en Bootstrap

Directivas y formularios

A

input
textarea
select

[(ngModel)]: indica la propiedad del modelo.
a la que se asocia el elemento

```
<label for="nif">NIF:</label>  
<input id="nif" type="text" id = "nif" name="nif" [(ngModel)]="oSeguro.nif" />
```

Para poder prescindir del atributo `name` dentro de un `form`, hay que modificar las propiedades del formulario

```
<label for="casado">Casado:</label>  
<input id="casado" type="checkbox" id="casado" name="casado" [(ngModel)]="oSeguro.casado" />
```

Se suele decir:

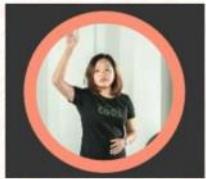
“Si el valor de una directiva ng-model no incluye un punto es que está mal.”

Formularios: más información

A

[https://scotch.io/tutorials/
how-to-deal-with-different-
form-controls-in-angular-2](https://scotch.io/tutorials/how-to-deal-with-different-form-controls-in-angular-2)

Jecelyn Yeen



Angular 2 Form Controls

Code Demo angular2 angularJS javascript typescript

How to Deal with Different Form Controls in Angular 2

How to Deal with Different Form Controls in Angular 2 (with latest forms module)

 jecelyn.yeen · jul 18, 2016 · Tutorials · Comments · 0

RadioButtons y Checkboxes



ngModel: como en cualquier control de formulario, indica la propiedad del modelo asociada al elemento

Radio-buttons → cada conjunto de ellos comparte el mismo valor de ngModel

Checkboxes → Si la propiedad tiene tipo, debe ser boolean

```
<input type="checkbox" [(ngModel)]="angular"/>
<label>Angular</label>
<input type="checkbox" [(ngModel)]="javascript"/>
<label>JavaScipt</label>
```

class controller : → angular:boolean
javascript:boolean

Checkboxes y arrays



*ngFor: permite utilizar un array de objetos asociado a un conjunto de controles

```
<span *ngFor="let item of items">
    <input type="checkbox"
        [(ngModel)]="item.selected"/> {{item.value}}
</span>
```

```
class controller : items: Array<Object> = [
    {value:'Item1', selected:false},
    {value:'Item2',selected:false}
]
```

Checkboxes en Angular 1.x



Checkboxes

ngTrueValue: permite asignar un valor personalizado al elemento cuando el campo *checkbox* está marcado.

ngFalseValue: es lo mismo que ngTrueValue, pero en este caso con el valor asignado cuando el campo no está marcado.

ngChange: sirve para indicar operaciones a realizar cuando se produce un evento de cambio en el elemento. Se dispara cuando cambia el estado del campo, marcado a no marcado y viceversa. El valor puede ser una expresión o una llamada a una función del *scope*.

Radio-Buttons



ngModel: se asocia a una misma propiedad del controller en todos los RB de un grupo.

su valor vera el correspondiente al atributo value del RB seleccionado

```
<input type="radio" name="gender"
[(ngModel)]="genero" value="Male"><label>Male</label>
<input type="radio" name="gender"
[(ngModel)]="genero" value="Female"><label>Female</label>
```

class controller : → genero: string

Formularios: select / options



En HTML, cada "option" puede tener 2 componentes

```
<option value="valor opcional">Etiqueta</option>
```

Angular únicamente añade la directiva **ngModel** para recoger el valor (si existe) o la etiqueta de la opción seleccionada

```
<select name="country" ng-model="user.country">
  <option value="">Please select an option</option>
  <option value="US">United States</option>
  <option value="GB">United Kingdom</option>
  <option value="AU">Australia</option>
</select>
```

Angular añade una opción en blanco a no ser que exista una con valor ""

Select / options desde el modelo



***ngFor
ngValue**

permite **crear automáticamente** un select/options a partir de un conjunto de datos, procesando un array de objetos (altems)

```
<select id="select" [(ngModel)]="resultado">  
<option *ngFor="let elem of aElementos" [ngValue]="elem">  
{{elem.nombre}} </option>
```

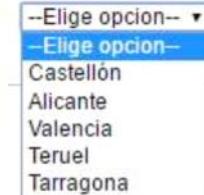
ngValue es el responsable de recoger los valores de los elementos (en este caso objetos completos) y de que el seleccionado se almacene en la variable asociada a ngModel

corresponde al ngOptions de Angular 1.x



Ejemplo de select/options

```
provincias=[  
    {idProvincia:2, nombre:"Castellón"},  
    {idProvincia:3, nombre:"Alicante"},  
    {idProvincia:1, nombre:"Valencia"},  
    {idProvincia:7, nombre:"Teruel"},  
    {dProvincia:5, nombre:"Tarragona"}  
];  
provinciaSeleccionada=null
```



```
<select [(ngModel)] = "provinciaSeleccionada"  
    <option *ngFor = "let provincia of provincias"  
        [ngValue] = "provincia"> {{provincia.nombre}}  
    </option>  
</select>
```



Formulario: Selección de opciones

Ejemplo de formulario con 2 de los mecanismos habituales de selección de opciones: *check box* y *select / options*

Formulario

Selección de opciones

- Imprimir resultado
- Tono claro

Provincia —Elige opción-- ▾

Resultado

- Opción print seleccionada: false
- Opción claro seleccionada: oscuro
- Provincia elegida:

Alejandro L. Cerezo - Madrid 2015

Lista de tareas (1): formulario

- Añadimos un formulario que permita recoger la descripción de una tarea y que incluya un botón añadir tarea. Mediante ngSubmit vinculamos el botón a una función manejadora.
- A continuación de cada item añadimos un botón que nos permita eliminarlo

Tareas

Describe una tarea	Añadir
Preparar curso de Angular	X
Preparar curso de Web Components	X

♥ CLE Formación - Curso de Angular



Lista de compras

A checklist

There are no items yet.

Enter the description...

Add

Ejemplo de "formulario" para gestionar una lista de tareas

A checklist

- Leche
- Pan
- Galletas

Vino Add

Métodos de arrays:
array.push()
array.indexOf()
array.splice()

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 llenar 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()">
```

Anulamos la validación estándar HTML5

```
<input type="text" id="firstname" name="firstname" [(ngModel)]="user.firstname" required="true" minlength="2" #firstnameState="ngModel">
```

input con doble binding

requerimientos de validación

referencia local

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

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

Para cada directiva de validación existe una propiedad errors que tomará 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

Ejemplo

domingo, 8 de octubre de 2017 8:38

The slide has a yellow textured background. At the top right, there is a large orange bar with the word 'Formularios: validación' in white. Below this, on the left, is a text block describing the requirements for a form. On the right, there is a screenshot of a web application interface.

Realizamos un formulario con:

- los campos Nombre y Apellido obligatorios y de un mínimo de 2 caracteres
- el campo Teléfono de exactamente 9 caracteres exclusivamente numéricos:
ng-pattern = "/^\\d{9}\$/"

Datos personales

Nombre: El nombre debe tener un mínimo de 2 caracteres

Apellidos:

Teléfono:

Resultado

- Nombre:
- Apellido:
- Teléfono:

Alejandro L. Cerezo - Madrid 2015

Requerimientos de validación

- required
- minlength
- maxlength
- patern, e.g. ^\d{9}\$

Mensajes de error en la validación

Mensajes simples, cuando sólo existe un requerimiento de validación

```
<div [hidden]="item.valid || item.unouched"
      class="error-message">
    El item es obligatorio
</div>
```

Mensajes complejos, cuando existen varios requerimientos de validación

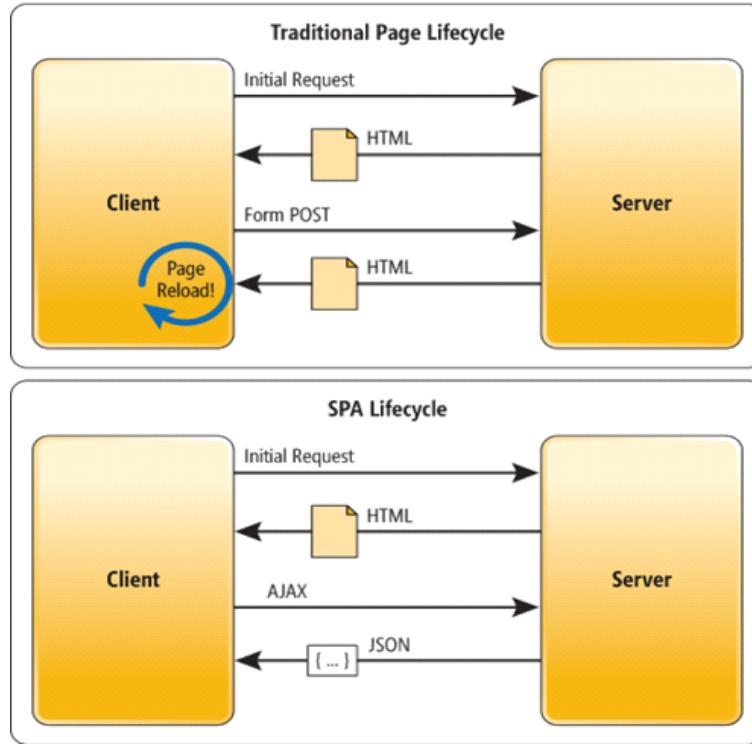
```
<div [hidden]="item.valid || item.unouched ">
  <div class="error-message"
       [hidden] ="!item.errors?.required">
    El item es obligatorio
  </div>
  <div class="error-message"
       [hidden] = "!
       item.errors?.minlength">
    El item debe tener un mínimo de ...
    caracteres
  </div>
</div>
```

Ξ Angular. Tercera parte

jueves, 14 de septiembre de 2017 20:31

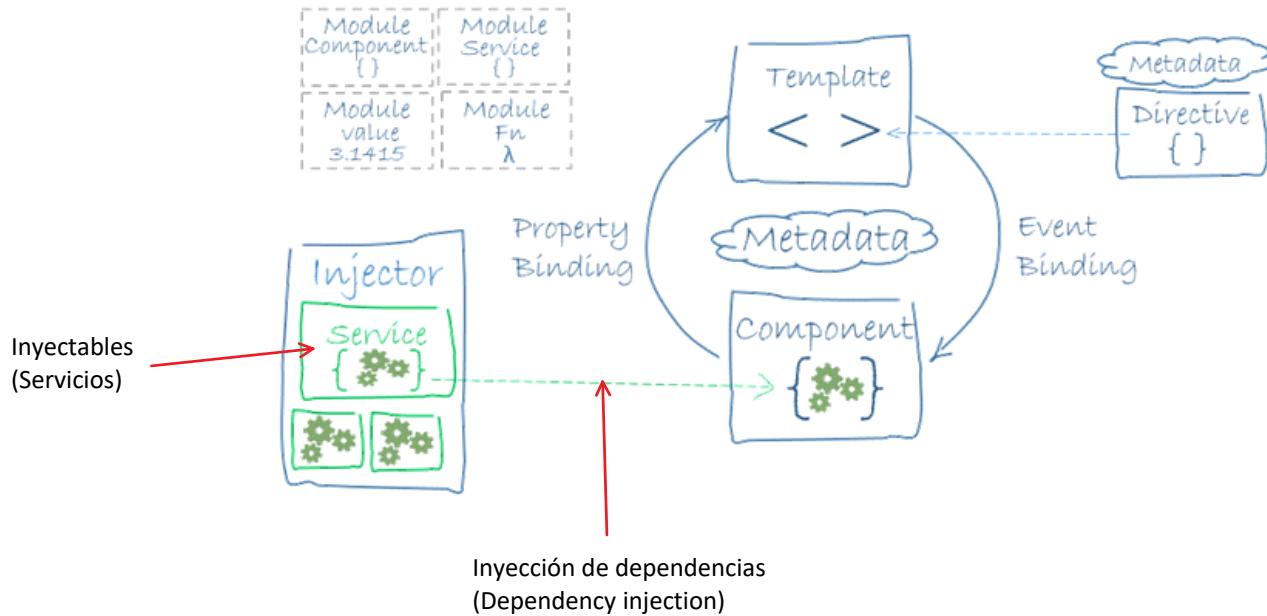
Aplicaciones SPA

- Enrutamiento y navegación
- Acceso al servidor (Comunicaciones HTTP con APIs REST)



Inyectables (Servicios)

miércoles, 13 de septiembre de 2017 22:25



Servicios

Elementos de la aplicación que no se encargan del interfaz de usuario

- Son clave para modularizar la aplicación en elementos que tengan una única responsabilidad
 - Componente: Interfaz de usuario
 - Servicios (e.g. Peticiones http)
- Permiten la buena práctica de NO acoplar en el componente la lógica de negocio, e.g. las peticiones http
- Permiten reducir la complejidad de los componentes y facilitar que estos sean ampliados / modificados
- Facilitan la implementar tests unitarios al reducir el número de responsabilidades que tiene el componente

Servicios: características técnicas

- En principio se instancian según el **patrón singleton**: permiten compartir información entre componentes
- Los servicios mantienen el estado de la aplicación y los componentes ofrecen el interfaz de usuario
- Están anotados como *injectable* para que puedan ser inyectados en los componentes que necesitan utilizarlos
- Angular 2 ofrece muchos servicios predefinidos como la clase http utilizada para el acceso asíncrono (AJAX) a las APIs REST
- Lo habitual es que en cada proyecto de aplicación se implementen los servicios propios necesarios

Inyección de dependencias

miércoles, 13 de septiembre de 2017 22:38

- La técnica que permite solicitar objetos al *framework* se denomina inyección de dependencias
- Las dependencias que un módulo necesita son inyectadas por el sistema
- Técnica muy popular en el desarrollo de *back-end* en *frameworks* como Spring o Java EE
- En Angular 2 se realiza mediante el constructor de la clase controladora del componente

<https://angular.io/docs/ts/latest/guide/dependency-injection.html>

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.

Injector

viernes, 19 de enero de 2018 18:11

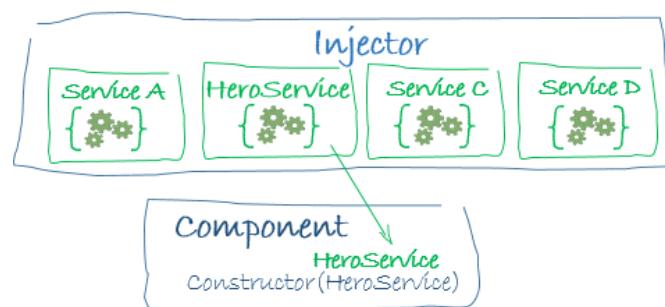
El Injector es el principal mecanismo detrás de la DI.

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

- Si una instancia no está en el contenedor, el injector **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 injector lo instancia** y lo guarda en un contenedor. Cuando inyectamos un servicio, antes de nada el injector 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 injector**, sin olvidar que podemos tener injectores a distintos niveles.



Cuando el injector 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 qué nivel será *singleton* la instanciaación

- 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

- Se puede hacer que servicio no sea compartido entre todos los componentes de la aplicación (no *singleton*)
- Se puede crear un servicio exclusivo para un componente y sus hijos

En vez de declarar el servicio en el atributo providers del @NgModule se declara en el @Component

```
import { Component } from '@angular/core';
import { BooksService } from './books.service';
@Component({
  selector: 'app-algo',
  templateUrl: './app.component.html',
  providers: [BooksService]
})
export class AlgoComponent {
  constructor(private booksService: BooksService){}
...
}
```

Servicios para 1 componente

jueves, 14 de septiembre de 2017 18:38

- Se puede hacer que servicio no sea compartido entre todos los componentes de la aplicación (*no singleton*)
- Se puede crear un servicio exclusivo para un componente y sus hijos

En vez de declarar el servicio en el atributo providers del @NgModule se declara en el @Component

```
import { Component } from '@angular/core';
import { BooksService } from './books.service';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  providers: 'BooksService'
})
export class AppComponent {
  constructor(private booksService: BooksService){}
  ...
}
```

Registro en módulos

viernes, 19 de enero de 2018 18:33

Procedimiento

jueves, 14 de septiembre de 2017 18:41

Implementación de un servicio

- Se importa de `@angular/core` la clase `injectable`
- Se crea una nueva clase para el servicio
- Se anota nuestra clase con `@Injectable`
- Se indica esa clase en la lista de `providers` del `NgModule`
- Se pone como parámetro en el constructor del componente que usa el servicio

Se puede automatizar con angular-cli

`ng g s <nombre>`

Implementación del servicio en su fichero `<nombre>.service.ts`

Se importa de `@angular/core` la clase `injectable`

clase correspondiente al servicio anotada con `@Injectable`

Funcionalidad del servicio, normalmente mediante métodos que retornan determinados valores

```
import { Injectable } from '@angular/core';
@Injectable()
export class AlgunService {
  unMetodo() {
    return ...
  }
}
```

Incorporación del servicio en un módulo, en este caso el módulo principal, `app.module`

Importación del servicio, a partir del fichero que lo contiene

Declaración de que se trata de un servicio, incluyéndolo en la lista de `providers`

```
import ...
import { AlgunService } from './algun.service';

@NgModule({
  declarations: [AppComponent],
  imports: [...],
  bootstrap: [AppComponent],
  providers: [AlgunService]
})
export class AppModule { }
```

Consumo del servicio en un componente, en este caso el componente principal `app-root`

Inyección del servicio en el constructor

Uso de los métodos del objeto correspondiente al servicio, instanciado de forma `singleton` en la aplicación

```
import { AlgunService } from './algun.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  constructor(private algunService: AlgunService) { }

  // En algún sitio
  ... this.algunService.unMetodo() ...
}
```

Ejemplo

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

books.service.ts
@Injectable()
export class BooksService {
  getBooks(title: string) {
    return [
      'Aprende Angular 2 en 2 días',
      'Angular 2 para torpes',
      'Angular 2 para expertos'
    ];
  }
}

import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { NgModule } from '@angular/core';
import { HttpClientModule, JsonpModule } from '@angular/http';
import { AppComponent } from './app.component';
import { BooksService } from './books.service';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, FormsModule, HttpClientModule,
    JsonpModule],
  bootstrap: [AppComponent],
  providers: [BooksService]
})
export class AppModule { }

app.component.ts
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  private books: string[] = [];
  constructor(private booksService: BooksService) { }
  search(title: string) {
    this.books = this.booksService.getBooks(title);
  }
}

<h1>Memory Books</h1>
<input #title type="text">
<button (click)="search(title.value); title.value=''">
  Buscar
</button>

<p *ngFor="let book of books">{{book}}</p>

app.component.html
```

Asincronia

jueves, 14 de septiembre de 2017 20:03

Con frecuencia los servicios encapsulan el acceso al backend con API REST (**buenas prácticas**),

No pueden devolver información de forma inmediata,
teniendo que esperar para devolver información cuando llega la respuesta del servidor

En JavaScript los métodos no se pueden bloquear esperando la respuesta. Son

asíncronos
reactivos

```
private service: BooksService = ...  
let books =  
this.booksService.getBooks(title);  
console.log(books);
```

- Callbacks
- Promesas
- Observables

NO se puede hacer

Callbacks

Al consumir el servicio, se le pasa como parámetro una función (de *callback*), frecuentemente en forma de función anónima.

Esta función

- será ejecutada cuando llegue el resultado.
- recibe como primer parámetro el error (si ha habido)

```
service.getBooks(title, (error, books) =>  
{  
    if(error) {  
        return console.error(error);  
    }  
    console.log(books);  
});
```

Promesas

El método devuelve un objeto *Promise*.

- Con el método *then* de ese objeto se define la función que se ejecutará cuando llegue el resultado.
- Con el método *catch* de ese objeto se define la función que se ejecutará si hay algún error

Creación de la promesa

```
getBooks(): Promise<Book[]> {  
    return Promise.resolve(aBooks);  
}
```

Consumo del servicio

```
service.getBooks(title)  
.then(books => console.log(books))  
.catch(error => console.error(error));
```

recomendada si la funcionalidad es suficiente

Observables

Similares a las promesas pero con más funcionalidad. (Más complejos de programar)
Con el método subscribe se definen las funciones que serán ejecutadas cuando llegue el resultado o si se produce un error

```
service.getBooks(title).subscribe(  
  books => console.log(books),  
  error => console.error(error)  
)
```

Es la forma recomendada por Angular 2 por ser la más completa (aunque más compleja)

Ejemplo: "Maqueta" con promesas

sábado, 9 de diciembre de 2017 16:22

```
aLibros: Array<string>;
constructor() {
  this.aLibros = [
    'Angular básico',
    'Angular en 19 minutos',
    'Angular avanzado'
  ];
}

buscarLibrosAsync(clave: string) {
  return new Promise(
    (resolve, reject) => {
      setTimeout(
        () => { resolve(this.aLibros); }, 2000
      );
    }
}
```

Array de datos para simular el resultado de una búsqueda

Se instancia y se devuelve un objeto "promesa"

La promesa recibe dos funciones *callback* que serán responsables de que sea resuelta o rechazada

En este caso, después de un tiempo definido por `setTimeout` para generar asincronía, la promesa se resuelve siempre correctamente, devolviendo el array de datos que simula el resultado de la búsqueda

Consumo del servicio

```
btnBuscar() {
  this.aLibros = [];
  this.librosMockService.buscarLibrosAsync(this.sClave)
  .then(
    (response) => {this.aLibros = response;}, // función OK
    (error) => { console.log(error); } // función error
  );
}
```

El método `then` permite definir las dos funciones que se ejecutarán tanto si la promesa se resuelve como si es rechazada

Cliente REST (AJAX)

Angular utiliza como cliente de API REST un objeto correspondiente a un servicio (inyectable) que lleva a cabo las peticiones asíncronas al servidor

- vía el objeto XMLHttpRequest (nativo de JavaScript)
- vía JSONP.

Angular 2/4 utiliza objetos de la clase **Http**, que incluye diversos métodos alternativos o atajos (shortcuts)

`http()`
`http.get()`
`http.post()`
`http.put()`
`http.delete()`
`http.jsonp()`
`http.head()`
`http.patch()`

Angular 5 incorpora un nuevo servicio, **HttpClient**, de uso más sencillo

En Angular 1.x, devolvía una promesa (similar a JQ)
En Angular devuelve un observable, u opcionalmente, una promesa

<https://angular.io/docs/ts/latest/guide/server-communication.html>

<https://angular.io/docs/ts/latest/api/http/index/Http-class.html>

Servicios Http / HttpClient

sábado, 9 de diciembre de 2017 15:56

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...*)

```
http([<url> ], {<objeto con los parámetros>})
http.get([<url> ], {<objeto con los parámetros>});
```

↓ Ejemplo ↓

```
http( {method: 'POST',
url: 'memberservices/register',
data: theData} )
http.get('memberservices/register')
```

Procesamiento de promesas

sábado, 9 de diciembre de 2017 20:09

El modificador `asPromise()` permite transformar la respuesta en una promesa, como las utilizadas por el servicio `$http` en *AnngularJS*

```
http.get(url).toPromise()
```

Respuesta al servicio Http

```
this.http.get(url)
  .toPromise()
  .then(
    response => console.log(response.json()), // promesa resuelta
    error => console.error(error); // promesa rechazada
  );
```

ejecutaremos alguna de las funciones definidas según el resultado de la promesa

Si se ha resuelto correctamente la promesa, para obtener los datos enviados por el servidor usamos el método `json()` del objeto `response`

Respuesta al servicio HttpClient

```
this.http.get(url)
  .toPromise()
  .then(
    response => console.log(response), // promesa resuelta
    error => console.error(error); // promesa rechazada
  );
```

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

Ejemplo

jueves, 14 de septiembre de 2017 22:43

API de Google para la búsqueda de libros

<https://www.googleapis.com/books/v1/volumes?q=intitle:'clave'>

```
search(title: string) {  
    this.books = [];  
    let url ="https://www.googleapis.com/books/v1/volumes?q=intitle:+title";  
  
    this.http.get(url)  
        .map(response => response.json())  
        .subscribe(  
            response => {  
                const data = response.items;  
                data.forEach ((item) => {  
                    const bookTitle = item.volumeInfo.title;  
                    this.books.push(bookTitle);  
                })  
            },  
            error => console.error(error)  
        );  
}
```

Se procesa la información proporcionada por el API consultada, en este caso para crear un array únicamente con los títulos de los libros

Las operaciones concretas de esta etapa de procesamiento de la respuesta dependen siempre de la estructura concreta de los datos proporcionados por la API que es consultada

The screenshot shows a search interface for books. At the top, there is a search bar with the placeholder "Clave de búsqueda" containing the text "peces". Below the search bar are three buttons: "Buscar", "Buscar Rx", and "Buscar RxFull". The main area is titled "Libros encontrados" and displays a list of book titles found in the search results:

- The precautionary approach to fisheries with reference to straddling fish stocks and highly migratory fish stocks
- 100 Peces Argentinos
- Peces en bandeja
- El día que cambié a mi padre por dos peces de colores
- El Sueño de Los Peces
- David, peces, pingüinos ...
- Manuales del acuario. Peces rojos o carpas doradas
- Los Panes y los Peces
- Peces del llano
- El pan y los peces

API con datos geográficos de países de un continente

<http://restcountries.eu/rest/v1/region/> <continente> → africa
europe
americas...

Países: consumo de un servicio

Paises del Mundo

Selecciona un continente:



Alejandro L. Cerezo.
CLE Formación
Madrid - 2015

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 varis 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

```
getLibros(clave: string): any {
  this.aLibros = [];
  const url = this.urlBase + clave;
  return this.http.get(url).toPromise()
    .then(
      (response: any) => {
        response.items.forEach(element => {
          this.aLibros.push(element.volumeInfo.title);
        });
        return new Promise ((resolve, reject) => resolve(this.aLibros));
      },
    );
}
```

Estado de los servicios

jueves, 14 de septiembre de 2017 22:54

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

Enrutamiento. Parámetros

jueves, 14 de septiembre de 2017 21:45

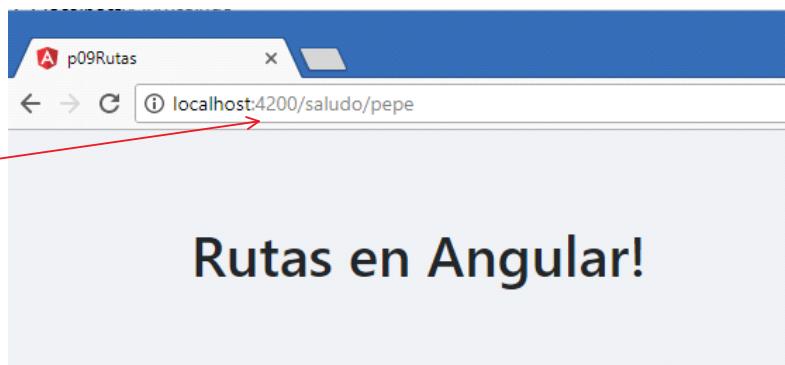
Una ruta puede incluir parámetros de forma estática
En la constante Routes aparecerá como

```
{  
  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 URLs

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

Ejemplo

jueves, 14 de septiembre de 2017 21:43

CRUD de libros

The diagram illustrates the CRUD operations for a book titled "SUEÑOS DE ACERO Y NEON".

- Read:** The top-left window shows the list of books, with "SUEÑOS DE ACERO Y NEON" selected. A green arrow points from this selection to the top-right window.
- Update:** The top-right window shows the details for "SUEÑOS DE ACERO Y NEON", including its description. A green arrow points from the "Edit" button in the top-left window to this window.
- Create:** The bottom-right window shows a new book form with the title "Book ***". A green arrow points from the "New book" button in the top-left window to this window.
- Delete:** The bottom-left window shows a confirmation dialog asking if the user wants to remove the book. A green arrow points from the "Remove" button in the top-left window to this dialog.

Testing

domingo, 11 de febrero de 2018 13:48

Angular proporciona ya configurados un conjunto de entorno que permiten la ejecución de los tests:

- *Karma* para los test unitarios
- *Protractor*, para los test e2e



Angular proporciona ya configurados un conjunto de herramientas de *testing* que permiten ejecutar el código en los anteriores entornos:

- Jasmine
- Las herramientas de test del propio Angular
(*Angular Unit Testing Framework*:
`@angular/core/testing`)



Angular genera ficheros de especificaciones para test unitarios (`.spec.ts`) para todos los elementos creados mediante `ng generate`, incluyendo:

- componentes
- directivas
- pipes

Comandos e informes

domingo, 11 de febrero de 2018 13:51

Modificadores del comando ng test

`ng test --single-run`

Ejecuta la batería completa de test una sola vez mostrando el resultado en la ventana de comandos

`ng test --code-coverage`

Crea un informe de cobertura, indicando que parte del código está cubierta por las pruebas

`ng test --single-run --code-coverage`

combina los dos anteriores

Informe de cobertura

Por defecto se genera en formato HTML, y siempre en una carpeta `coverage` añadida al proyecto

<code>app/</code>		79.41%
<code>app/dashboard-organizer/</code>		100%
<code>app/dashboard-sponzor/</code>		100%
<code>app/events-organizer/</code>		99.56%
<code>app/events-sponsor/</code>		84.21%
<code>app/services/</code>		100%
<code>app/sponsors-organizer/</code>		100%
<code>app/tasks-organizer/</code>		100%
<code>app/tasks-sponsor/</code>		29.52%
<code>app/users/</code>		100%
<code>app/widgets/</code>		54.55%

```
1 1x export class Calculator {
2
3 1x   multiply(numberA: number, numberB: number): number{
4 1x     return numberA * numberB;
5   }
6
7 1x   divide(numberA: number, numberB: number): number{
8     if(numberB === 0){
9       return null;
10    }
11   return numberA / numberB;
12 }
13 1x }
14
```

Una opción para mostrar el informe es tener instalado el servidor http-server, basado en *Node*
<https://www.npmjs.com/package/http-server>

El servidor se instala globalmente mediante

```
npm install http-server -g
```

Y se ejecuta en la línea de comandos indicándole la carpeta que será raíz del servidor web

```
http-server coverage
```

Configuración

domingo, 11 de febrero de 2018 13:56

El funcionamiento por defecto de los tests en Angular depende de los ficheros

- karma.conf.js
- protractor.conf.js

Configuración de Karma

```
Frameworks           module.exports = function (config) {  
  config.set({  
    basePath: '',  
    frameworks: ['jasmine', '@angular/cli'],  
    plugins: [  
      require('karma-jasmine'),  
      require('karma-chrome-launcher'),  
      require('karma-jasmine-html-reporter'),  
      require('karma-coverage-istanbul-reporter'),  
      require('@angular/cli/plugins/karma')  
    ],  
    client:{  
      clearContext: false  
      // leave Jasmine Spec Runner output visible in browser  
    },  
    coverageIstanbulReporter: {  
      reports: [ 'html', 'lcovonly' ],  
      fixWebpackSourcePaths: true  
    },  
    angularCli: {  
      environment: 'dev'  
    },  
    reporters: [ 'progress', 'kjhtml' ],  
    port: 9876,  
    colors: true,  
    logLevel: config.LOG_INFO,  
    autoWatch: true,  
    browsers: [ 'Chrome' ],  
    singleRun: false  
  });  
};  
  
Plugins incluidos por defecto  
  
Salida del informe de cobertura  
  
Salida de datos:  
- consola  
- browser
```

Plugins

karma-mocha-reporter

salida de datos por consola en Karma con el formato habitual de Mocha

<https://www.npmjs.com/package/karma-mocha-reporter>

Focus

domingo, 11 de febrero de 2018 14:51

El modificador `f (focus)` puede usarse dentro de un fichero de especificaciones (`.spec.ts`) para definir el set de pruebas o incluso la prueba concreta que serán lo único ejecutado por karma cuando se lance mediante el comando `ng test`



Solo se ejecutará este set de pruebas

```
fdescribe('Test for XComponent', () => {...});
```

```
fit("should make something", ()=>{...});
```



Solo se ejecutará esta prueba

Test de componentes

domingo, 11 de febrero de 2018 17:21

El entorno de ejecución refleja las complejidades del entorno real del componente

En este entorno es frecuente distribuir las tareas entre distintos componente, e.g. con el modelo controlador / presentador

En un componente "controlador", es necesario declarar los componentes "presentadores" como parte de su entorno de pruebas

```
describe('TareasComponent', () => {
  let component: TareasComponent;
  let fixture: ComponentFixture<TareasComponent>;
  let declarations: [TareasComponent, ItemComponent, ListaComponent, PipesComponent];
  let imports: [SharedModule, FormsModule];
  let TestBed: TestBed;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations,
      imports
    })
    .compileComponents();
  }));
});
```

Se declara cualquier componente consumido por el *template* del que se está probando

Se importa cualquier módulo que incluya funcionalidades usadas por el componente, nativas de angular o propias (e.g. directivas o pipes)

En un componente "presentador" es necesario inicializar las propiedades decoradas como inputs cuando no son tipos elementales

```
beforeEach(() => {
  fixture = TestBed.createComponent(ListaComponent);
  component = fixture.componentInstance;
  component.aItems = [];
  fixture.detectChanges();
});
```

Se declara e inicializa una propiedad input, que normalmente recibiría su valor desde el componente "controlador"

Test de directivas

domingo, 11 de febrero de 2018 17:51

Importaciones habituales en la mayoría de los tests

```
import { Component, DebugElement, ElementRef } from '@angular/core';
import { By } from '@angular/platform-browser';
import { TestBed, ComponentFixture } from '@angular/core/testing';
```

En el caso de las directivas se puede declarar en el propio test un componente "virtual", sobre el que se aplicará la directiva

```
@Component({
  template: `<p id="test" destacar></p>`
})
class TestComponent { }
```



```
describe('DestacarDirective', () => {

  let component: TestComponent;
  let fixture: ComponentFixture<TestComponent>;
  let elem: DebugElement;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        TestComponent,
        DestacarDirective
      ]
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(TestComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
    elem = fixture.debugElement.query(By.css('#test'));
  });

  it('should create an instance', () => {
    const directive = new DestacarDirective(elem);
    expect(directive).toBeTruthy();
  });
});
```

Importación de la directiva y del componente recién creado para probarla

El componente se trata igual que si fuera el objeto de las pruebas, instanciándolo dotado de su fixture

Una variable de tipo DebugElement representa al elemento del DOM correspondiente al componente

Al instanciar la Directiva es necesario pasarle al constructor el elemento del DOM sobre el que la directiva se aplicará

Test de pipes

domingo, 11 de febrero de 2018 18:07

No requieren apenas configuración para su versión inicial

```
describe('TitularizePipe', () => {  
  it('create an instance', () => {  
    const pipe = new TitularizePipe();  
    expect(pipe).toBeTruthy();  
  });  
});
```

La clase correspondiente al pipe se instancia directamente para realizar las pruebas

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/ng2-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>

The screenshot shows the official website for ng-Bootstrap. At the top, there's a navigation bar with links for "ng-bootstrap", "Getting Started", and "Components". On the right side of the header are "Star" and "Tweet" buttons. The main content area features a large blue hexagonal logo with a white letter "B". Below the logo, the text "Bootstrap 4 components, powered by Angular" is displayed. A note indicates "Currently at v1.0.0-beta.5". Below this, there are two buttons: "Demo" and "Installation". In the bottom left, there's a section titled "Native" with a CPU icon and a brief description. In the bottom right, there's a section titled "Widgets" with a window icon and a brief description.

Components

- Accordion
- Alert
- Buttons
- Carousel
- Collapse
- Datepicker
- Dropdown
- Modal
- Pagination
- Popover
- Progressbar
- Rating
- Tabs
- Timepicker
- Tooltip
- Typeahead

<https://www.primefaces.org/primeng/#/>

The screenshot shows the PrimeNG website homepage. On the left, there is a sidebar with a navigation menu containing the following items: Input, Button, Data, Panel, Overlay, File, Menu, Charts, Messages, Multimedia, DragDrop, and Misc. The 'DragDrop' item is currently selected, indicated by a blue background. The main content area features a large banner with the text "The Most Complete User Interface Suite for Angular" and a "GET STARTED" button. To the right of the banner is a large image of a smartphone displaying a complex user interface. Below the banner, the text "Why PrimeNG?" is followed by a congratulatory message: "Congratulations! 🎉 Your quest to find the UI library for Angular is complete." It also states that PrimeNG is a collection of rich UI components for Angular, developed by PrimeTek Informatics, and provides links to Twitter and the blog. The page is divided into several sections: "70+ COMPONENTS" (with a component icon), "OPEN SOURCE" (with an open source icon), and "PRODUCTIVITY" (with a productivity icon). Each section has a brief description.

PRIME NG

GET STARTED THEMES SUPPORT

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

OPEN SOURCE

PRODUCTIVITY

Angular Camp página 174

Material Design

sábado, 7 de octubre de 2017 19:54

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

The screenshot shows the Angular Material homepage. At the top, there's a navigation bar with links for 'Material', 'Components', and 'Guides'. On the right side of the nav bar are icons for a file, GitHub, and a person. The main title 'Angular Material' is centered at the top in a large white font, with the subtitle 'Material Design components for Angular' below it. A 'Get started' button is located just below the subtitle. To the left of the main content area, there's a graphic featuring a smartphone icon and several blue UI component icons like a switch, a button, and a list item. To the right of the graphic, the text 'Sprint from Zero to App' is displayed, followed by a brief description: 'Hit the ground running with comprehensive, modern UI components that work across the web, mobile and desktop.' Below this, a section titled 'Component Categories' lists six categories: 'Form Controls', 'Navigation', 'Layout', 'Buttons & Indicators', 'Popups & Modals', and 'Data table'.

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>
```

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.



Traducción

domingo, 28 de enero de 2018 22:06

i18n en Angular

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

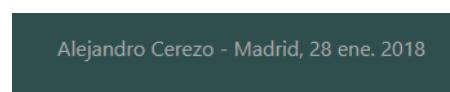
```
import { LOCALE_ID, NgModule } from '@angular/core';
import { registerLocaleData } from '@angular/common';
import localeEs from '@angular/common/locales/es';

registerLocaleData(localeEs);

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

### Ejemplo

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



## Librería de traducción

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

Love JavaScript? Your insights can make it even better. [Take t](#)

Librería para facilitar la traducción de la aplicación

### ★ ng2-translate public

An implementation of angular translate for Angular 2.

Simple example using ng2-translate: <http://plnkr.co/edit/btpW3l0jr5beJvJohy1Q?p=preview>

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

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

### Instalación

```
npm install ng2-translate
```

### Configuración

```
import { TranslateModule, TranslateLoader, TranslateStaticLoader } from 'ng2-translate';
import { HttpModule, 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

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

Tipo de acceso a los datos de traducción

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
```

# 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

# Programación reactiva

sábado, 9 de diciembre de 2017 16:39

# Conceptos

viernes, 19 de enero de 2018 11:48

- Flujo de datos
- Observables

# Flujo de datos

miércoles, 31 de enero de 2018 0:49

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



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

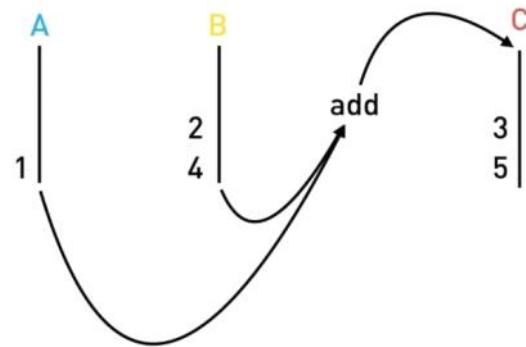
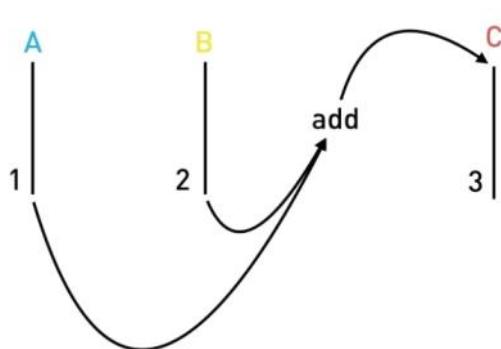
flujos de datos (*streams*):  
series de datos encadenados que  
pueden ser emitidos en el tiempo.



secuencias de valores a lo largo del 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

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



# Observables

miércoles, 31 de enero de 2018 0:50

## Observables

- son mecanismos creados para representar esos flujos de datos
- 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.

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.

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.

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.

<https://codekstudio.com/post-blog/conceptos-observables-rxjs-y-angular-2->

[javascript-reactivo-y-funcional/57d1e2840897131b5ec54b90](#)

Observables en ES6:  
Implementados  
en la [librería RxJS](#)  
(*Reactive Extensions  
for JavaScript*)



Extensiones reactivas  
para JavaScript incluidas  
en Angular



[ReactiveX](#)

Utilizada principalmente en

- Formularios reactivos
- Emisión de eventos
- Servicio Http

<http://reactivex.io/>



The screenshot shows the official website for ReactiveX. At the top, there's a dark navigation bar with the ReactiveX logo on the left and links for 'Introduction', 'Docs', 'Languages', 'Resources', and 'Community'. The main content area has a dark background with a subtle image of light trails. In the center, the ReactiveX logo is displayed next to the text 'ReactiveX'. Below that, it says 'An API for asynchronous programming with observable streams'. At the bottom left, there's a pink button with the text 'Choose your platform'.

## Uso de RxJS

viernes, 19 de enero de 2018 12:12

```
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 comunique 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 subscriba a él. Esto es importante porque de esta manera no se consumen recursos sin sentido.

Ahora, cualquier cambio, como es que se añade 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

```
import { Observable } from 'rxjs/Observable';
buscarLibrosAsyncRx(clave: string) {
 return new Observable(←
 (observer) => {
 setTimeout(() => {
 observer.next(this.aLibros);
 }, 2000);
 }
);
}
```

Importamos Observable desde rxjs/Observable.

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

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

1

El equivalente en  
ES6 puro, usando  
NodeJS, sería

```
let Rx = require('rx');
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

# Uso en Angular

viernes, 19 de enero de 2018 16:35

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.

# Programación funcional

viernes, 19 de enero de 2018 16:52

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” quieras 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.

# Comunicación entre componentes

miércoles, 13 de septiembre de 2017 19:22

## 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 injectables (servicios) son objetos *singleton* y por tanto compartidos entre los distintas clases que los instancian

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

# Inyección de servicios observables

miércoles, 31 de enero de 2018 0:08

Comunicación entre componentes completamente desacoplada  
"en medio hay algo", donde un componente puede escribir y otro puede enterarse de los cambios al estar suscrito a un *stream* de datos.

El problema es como un componente se entera de que un dato ha cambiado en otro componente.

Una alternativa al patrón *pull*, en el que se consulta cada cierto tiempo una variable para poder detectar los cambios.

Observable → permite la suscripción a cambios de un stream

Subject → se comporta como un observable y además permite la emisión de datos como un observable, que puede ser leído por los subscriptores pero no modificado

Se utiliza la librería **rxjs** de Microsoft

En Angular, la detección del cambio pasa a ser una tarea del programador  
En este modelo reactivo, los cambios corresponden a la emisión de un evento

# Ejemplo

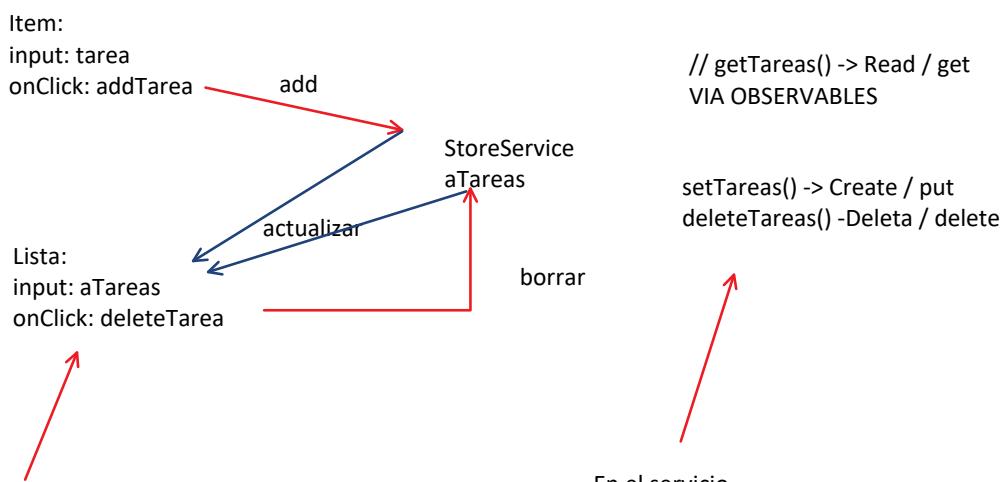
miércoles, 31 de enero de 2018 0:15

En una interpretación muy imprecisa del patrón REDUX

Componte 1 - entrada de datos

Componente 2 - presentación de los datos

Ambos usan el servicio que define los datos



En el componente lista

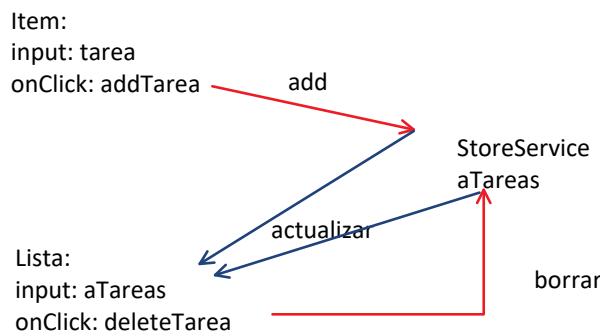
- Se crea un observable correspondiente al array datos
- El observable refleja el subject del servicio
- Así es posible suscribirse al servicio, que actúa como emisor de eventos
  - Las respuestas del servicio son similares a las de HttpClient cuando accede a un API REST
  - El método subscribe() permite procesar la respuesta, pasando los valores recibidos al array de datos

En el servicio

- Se crea un Subject correspondiente al store. Una variable capaz de informar de los cambios que se produzcan en la anterior
- Un método del servicio devuelve el Subject, de modo que desde fuera sea posible suscribirse a él
- Cada vez que el store cambia (se añaden o se eliminan items), se emite un evento mediante el método next() del subject
- De esa forma los elementos suscritos al subject son informados de los cambios

# Procedimiento

martes, 30 de enero de 2018 23:54



- Se declara un observable correspondiente al array datos:

```
aItems: Array<any>;
aItems$: Observable<any[]>;
```

- En el observable se recoge el Subject del servicio

```
this.aItems$ = this.datosService.getSubjectDatos();
```

En la suscripción Subject/Observable se establece la respuesta a los cambios

```
this.aItems$.subscribe(
 (data) => this.aItems = data
)
```

Como alternativa al procesamiento se puede enviar directamente el observable al template HTML usando el pipe async para su correcta visualización

getSubjectTareas() -> Read / get VIA OBSERVABLES

setTareas() -> Create / put  
deleteTareas() -> Delete / delete

- Se crea un Subject correspondiente al store

```
aDatos: Array<any>;
aDatos$: Subject<any[]>;
```

variable capaz de informar de los cambios que se produzcan en la anterior

- Se instancia en el constructor

```
this.aDatos$ = new Subject<any[]>();
```

Una función permite suscribirse al store

```
suscribeDatos() {
 return this.aDatos$;
}
```

Es posible devolver el Subject como un observable, para encapsular la forma en que se ha implementado el proceso

```
return this.aDatos$.asObservable
```

## Clase 5b - Observables (1:32:40)

viernes, 29 de septiembre de 2017 20:21

### En el servicio

- se declara una variable que almacenara información: un "almacén".  
Si existe un interfaz con el modelo de los datos, se declara como array de este tipo

```
// Almacén de movimientos en memoria
private movimientos: MovimientoModel[]
```

- se declara el emisor de eventos correspondiente; por convenio su nombre es el de la variable seguido de \$.  
Su tipo se crea mediante un **genérico** del objeto *Subject* específico del tipo del "almacén"  
Se instancia el objeto correspondiente

```
// Emisor de eventos relacionados con el almacén de movimientos
private movimientos$: Subject<MovimientoModel []> = new Subject<MovimientoModel []>();
```

- cualquier modificación en el almacén va acompañada por la **emisión de un evento** que genera un nuevo valor en el observable

```
this.movimientos. push (movimientos) ;
// se genera un nuevo valor en el observable
this.movimientos$.next (this.movimientos) ;
```

el objeto *Subject* emite un evento que concuerde con su tipo especificado

- definimos un método que devuelve el *Subject*

```
Devuelve un observable que notifica cambios en el
getMovimientos$(): Observable<MovimientoModel []> {
 // se comporta como un observable
 return this.movimientos$.asObservable() ;
}
```

### En el componente 2

- declaro la propiedad correspondiente al objeto Observable sólo para el tipo específico con el que estamos trabajando

```
movimientos$: Observable<MovimientoModel []>
```

- inyecto en el constructor el servicio

```
Este componente depende del objeto DatosService
constructor(private datosService: DatosService) { }
```

enlazo el componente con el "emisor de eventos" en el servicio

```
ngOnInit() {
 // Al iniciar el componente se enlaza con el almacén de datos
 this.movimientos$ = this.datosService.getMovimientos$();
 // si se quiere se puede suscribir programáticamente
 this.movimientos$.subscribe(
 d=>console.log("Dato recibido:")
 // como alternativa a esta última linea
 // se puede usar el pipe async en la vista
 }
}
```

En la vista del componente

El pipe async implica una suscripción a un observable

```
<!--el pipe async se subscribe a un observable-->
<!--los pipes se pueden entubar-->
&{ movimientos$ | async | json }
```

# Servicios Http / HttpClient

sábado, 9 de diciembre de 2017 15:56

## 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...*)

```
http([<url>], {<objeto con los parámetros>})
http.get([<url>], {<objeto con los parámetros>});
```

↓ Ejemplo ↓

```
http({method: 'POST',
url: 'memberservices/register',
data: theData})
http.get('memberservices/register')
```

## 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 onNext
);
```

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 onNext
);
```

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

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

## 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)

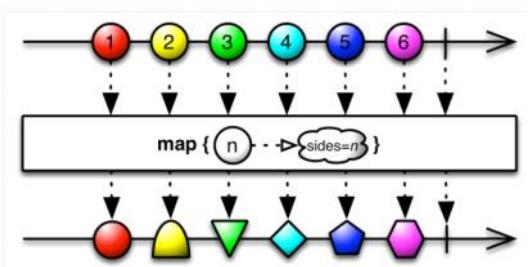
Nuestro propio servicio realizara varis operaciones

- 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



# Estado de los servicios

jueves, 14 de septiembre de 2017 22:54

## 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

# 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 injectado 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
 // p醙ole como par醟metro el objeto con la definici髇 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 llenar 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()">
```

Anulamos la validación estándar HTML5

```
<input type="text" id="firstname" name="firstname" [(ngModel)]="user.firstname" required="true" minlength="2" #firstnameState="ngModel">
```

input con doble binding

requerimientos de validación

referencia local

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

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

Para cada directiva de validación existe una propiedad errors que tomará 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

## Conclusiones

jueves, 14 de septiembre de 2017 23:03

### En resumen ...

- Un framework de desarrollo apps SPA
- Se recomienda TypeScript, más preparado para grandes aplicaciones (pero puede usar ES5, ES6)
- Orientado a componentes, con inyección de dependencias y templates
- No es compatible con Angular 1, pero comparte su filosofía
- Mucho mejor rendimiento que Angular 1
- Seguramente será uno de los frameworks más usados para desarrollo web en los próximos años



### 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
- Orientado a componentes, con inyección de dependencias y *templates*
- Mucho mejor rendimiento que Angular 1
- Seguramente será uno de los *frameworks* más usados para desarrollo web en los próximos años

### ... y más

- Validación de formularios (con NgForm y NgControl)
- Testing unitario y de integración (Jasmine, Karma, Protractor, Inyección de dependencias de testing...)
- Carga bajo demanda de componentes (para acelerar la carga inicial de la aplicación)
- Gestión del estado al estilo Redux (ngrx)
- Animaciones



### ... 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...)

### ... y aún más

- Aplicaciones con múltiples @NgModule (para estructurar componentes y dependencias)
- Optimización de la app en producción: Compilador de templates a TS (angular-compiler), eliminación de funcionalidades de la librería no usadas... (tree shaking)
- Angular Universal: Renderizado en el servidor para optimizar la descarga inicial



### ... y aún más

- Aplicaciones con múltiples @NgModule (para estructurar componentes y dependencias)
- Optimización de la app en producción: Compilador de templates a TS (angular-compiler), eliminación de funcionalidades de la librería no usadas... (tree shaking)
- Angular Universal: Renderizado en el servidor para optimizar la descarga inicial

## Ecosistema Angular



- **Angular2-electron:** Aplicaciones de escritorio multiplataforma con Angular2
- **Ionic2:** Aplicaciones móviles híbridas con Angular2
- **NativeScript:** Aplicaciones móviles con UI nativo con Angular2
- **Angular2-Meteor:** Framework JavaScript/TypeScript fullstack para desarrollo de apps web interactivas (comunicación websockets cliente servidor)
- **AngularFire2:** Cliente Angular 2 para el backend as a service Firebase de Google

## Ecosistema Angular

- **Angular2-electron:** Aplicaciones de escritorio multiplataforma con Angular2
- **Ionic2:** Aplicaciones móviles híbridas con Angular2
- **NativeScript:** Aplicaciones móviles con UI nativo con Angular2
- **Angular2-Meteor:** Framework JavaScript/TypeScript fullstack para desarrollo de apps web interactivas (comunicación *web sockets* cliente servidor)
- **AngularFire2:** Cliente Angular 2 para el *backend as a service* Firebase de Google