

Consultoría y Formación en  
Desarrollo Software

**Contacta con nosotros para cursos  
presenciales, online, in company**

# ¿Quién soy?



@micael\_gallego

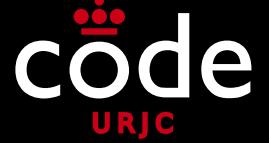


micael.gallego@urjc.es



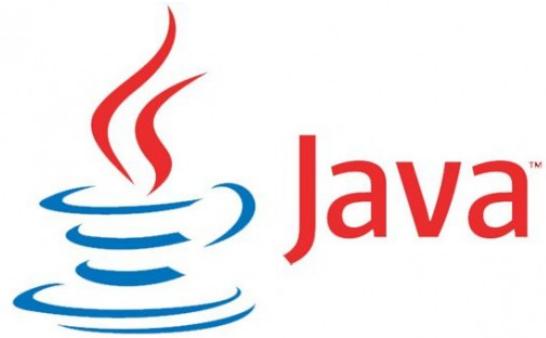
@micaelgallego

# ¿Quién soy?



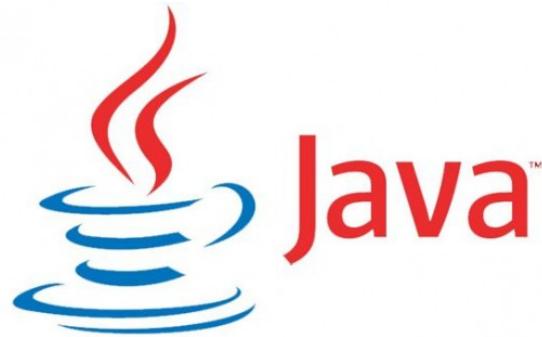
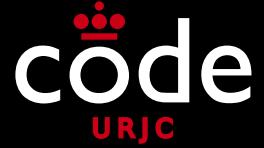
# developer

# ¿Quién soy?



# developer

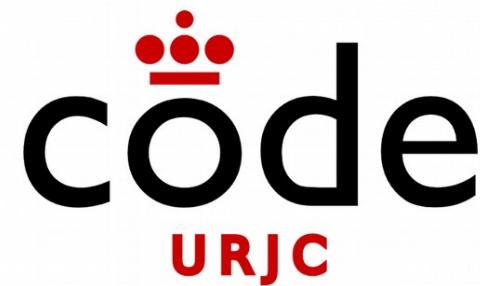
# ¿Quién soy?



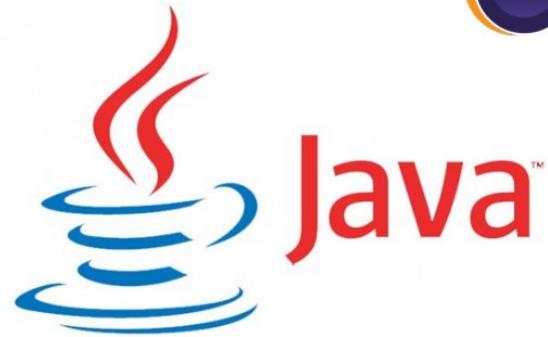
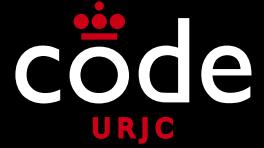
# developer



Universidad  
Rey Juan Carlos



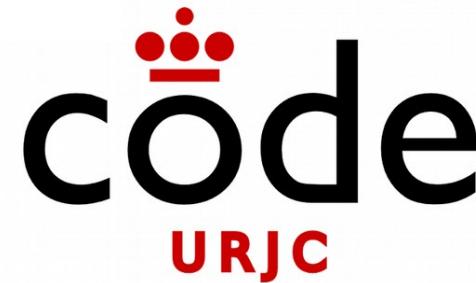
# ¿Quién soy?



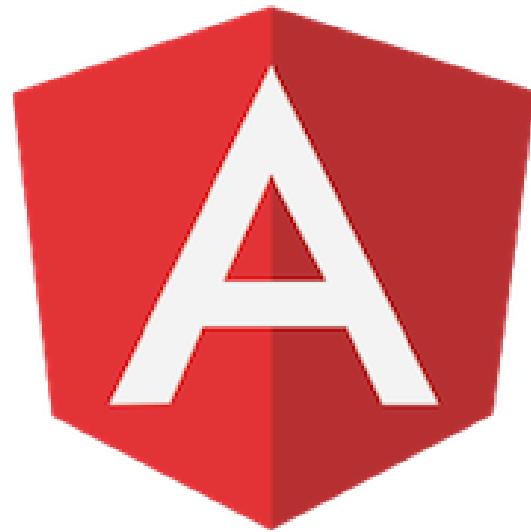
# developer



Universidad  
Rey Juan Carlos



# Curso



# Angular

# Temario

- Tema 1: Introducción a Angular: TypeScript y herramientas
- Tema 2: Componentes
- Tema 3: REST y Servicios
- Tema 4: Aplicaciones multipágina: Router
- Tema 5: Librerías de componentes
- Tema 6: Publicación

# Recursos

- Presentaciones y ejemplos de código
- Internet
  - Páginas oficiales
  - Tutoriales, ejemplos y blogs realizados por la comunidad
  - Aplicaciones web de código abierto
  - Y todo lo que Google encuentre :)
  - **Mucho cuidado con las versiones y la fecha de publicación!!!**
  - **Material actualizado a Angular 9**

# Tema 1

# Introducción a

# Angular: TypeScript y

# herramientas

Micael Gallego  
@micael\_gallego

# Taxonomía de las webs

## Existen varios tipos de webs

Dependiendo de cómo usan las tecnologías  
de cliente y servidor

### Página web

Servidor estático

El servidor web sirve  
contenido guardado en  
el disco duro

### Aplicación web

Servidor dinámico

El servidor web sirve  
contenido generado  
mediante código

## Página web

### Servidor estático

- Página web estática
- Página web interactiva

## Aplicación web

### Servidor dinámico

- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

## Página web

### Servidor estático

- Página web estática

- Página web interactiva

## Aplicación web

### Servidor dinámico

- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

java.util.concurrent (Java I X)

download.java.net/jdk8/docs/api/

Java™ Platform Standard Ed. 8 DRAFT ea-b123

All Classes All Profiles Packages

java.applet  
java.awt  
java.awt.color

AnnotationValueVisitor  
Any  
AnyHolder  
AnySeqHelper  
AnySeqHelper  
AnySeqHolder  
AppConfigurationEntry  
AppConfigurationEntry.LoginMo  
Appendable  
Applet  
AppletContext  
AppletInitializer  
AppletStub  
ApplicationException  
Arc2D  
Arc2D.Double  
Arc2D.Float  
Area  
AreaAveragingScaleFilter  
ARG\_IN  
ARG\_INOUT  
ARG\_OUT  
ArithmaticException  
Array  
Array  
ArrayBlockingQueue

Please note that the specifications and other information contained herein are not final and are subject to change. The information is being made available to you solely for purpose of evaluation.

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8 DRAFT ea-b123

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

## Package java.util.concurrent

Utility classes commonly useful in concurrent programming.

See: Description

### Interface Summary

Interface	Description
<a href="#">BlockingDeque&lt;E&gt;</a>	A Deque that additionally supports blocking operations that wait for the deque to become non-empty when retrieving an element, and wait for space to become available in the deque when storing an element.
<a href="#">BlockingQueue&lt;E&gt;</a>	A queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.
<a href="#">Callable&lt;V&gt;</a>	A task that returns a result and may throw an exception.
<a href="#">CompletableFuture.AsynchronousCompletionTask</a>	A marker interface identifying asynchronous tasks produced by <code>async</code> methods.
<a href="#">CompletionService&lt;V&gt;</a>	A service that decouples the production of new asynchronous tasks from the consumption of the results of completed tasks.

Maven – Maven in 5 Minu x

maven.apache.org/guides/getting-started/maven-in-five-minutes.html

# Apache Maven Project

http:// maven.apache.org

Apache > Maven > Maven in 5 Minutes

Last Published: 2015-01-11

## Maven in 5 Minutes

### Prerequisites

You must have an understanding of how to install software on your computer. If you do not know how to do this, please ask someone at your office, school, etc or pay someone to explain this to you. The Maven mailing lists are not the best place to ask for this advice.

### Installation

*Maven is a Java tool, so you must have Java installed in order to proceed.*

First, download Maven and follow the installation instructions. After that, type the following in a terminal or in a command prompt:

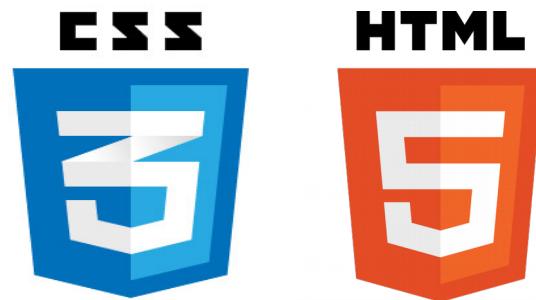
```
mvn --version
```

It should print out your installed version of Maven, for example:

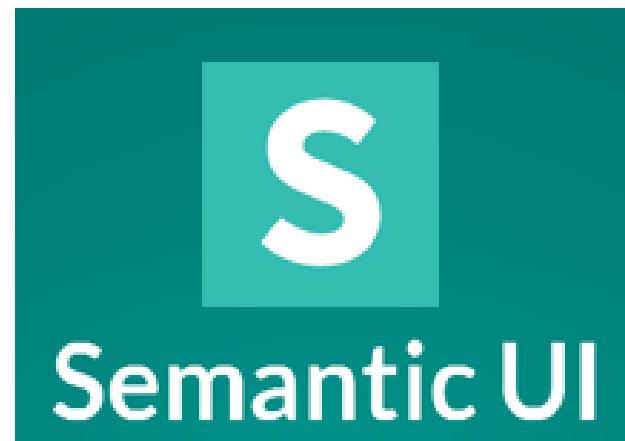
```
Apache Maven 3.0.5 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 2013-02-19 14:51:28+0100)
Maven home: D:\apache-maven-3.0.5\bin\..
Java version: 1.6.0_25, vendor: Sun Microsystems Inc.
Java home: C:\Program Files\Java\jdk1.6.0_25\jre
Default locale: en_NL, platform encoding: Cp1252
```

# Taxonomía de las webs

- Página web estática



- Se utilizan librerías de componentes CSS



# Buttons and navigation

Use these vector based elements to mockup a Bootstrap website with form elements.

All elements are full customizable.

Buttons ▾    Navigation ▾    Labels    Badges    Typography    Thumbnails    Alerts    Progress bars    Miscellaneous

Default    Primary    Info    Success    Danger    Warning    Inverse    1  
Default    Primary    Info    Success    Danger    Warning    Inverse    2  
Default    Primary    Info    Success    Danger    Warning    Inverse    4  
Large action    Large action  
1    2    3    4    Left    Middle    Right    Home    Profile    Messages    Home    Profile    Messages  
Home / Library / Data    «    1    2    3    4    »    ← Older    Newer →  
Seconday link  
Something else here

Home  
Profile  
Messages

Home  
Profile  
Messages

Home  
Library  
Applications

# Taxonomía de las webs

## Página web

### Servidor estático

- Página web estática
- Página web interactiva

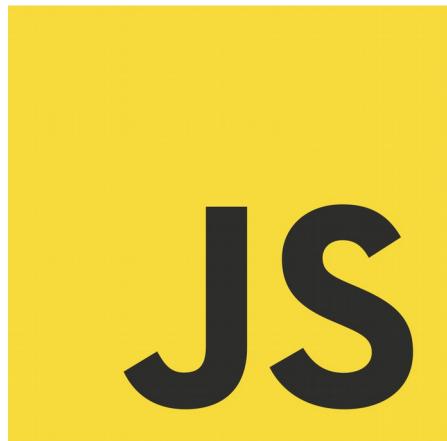
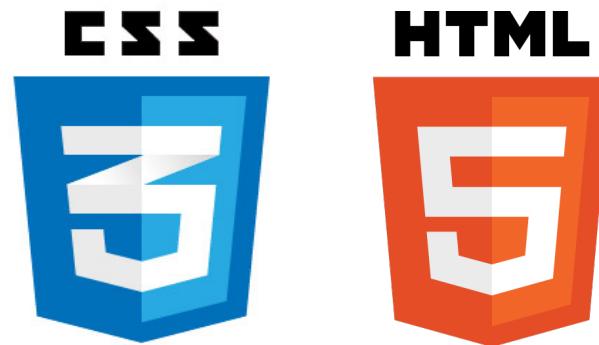
## Aplicación web

### Servidor dinámico

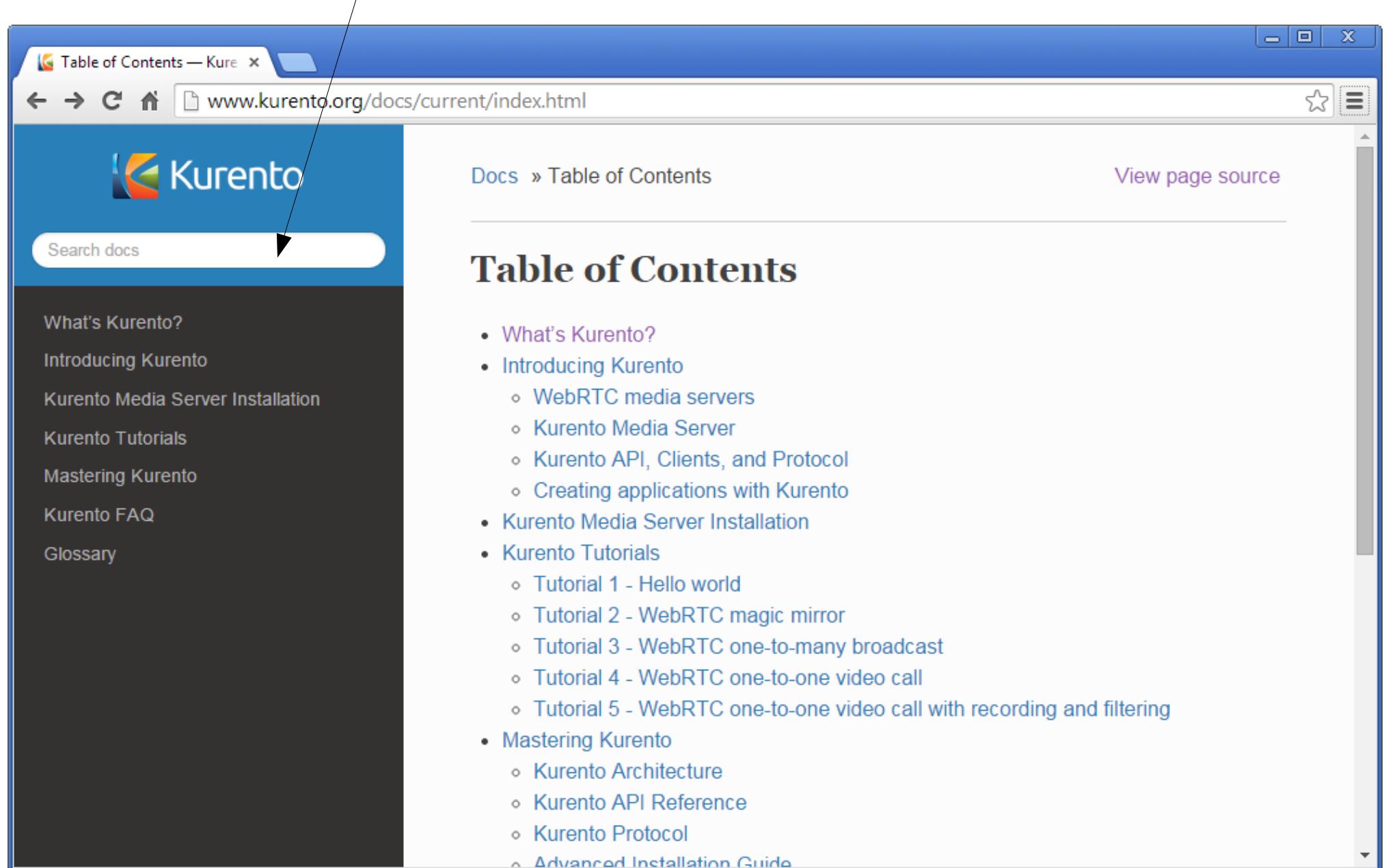
- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

# Taxonomía de las webs

- Página web interactiva



# Buscador implementado en JavaScript



The screenshot shows a web browser window with the title "Table of Contents — Kure". The address bar displays the URL [www.kurento.org/docs/current/index.html](http://www.kurento.org/docs/current/index.html). The page content is the "Table of Contents" for the Kurento documentation. The sidebar on the left lists several categories: "What's Kurento?", "Introducing Kurento", "Kurento Media Server Installation", "Kurento Tutorials", "Mastering Kurento", "Kurento FAQ", and "Glossary". A search bar labeled "Search docs" is located at the top of the sidebar. The main content area shows a hierarchical list of topics under "Table of Contents".

Docs » Table of Contents View page source

---

## Table of Contents

- [What's Kurento?](#)
- [Introducing Kurento](#)
  - [WebRTC media servers](#)
  - [Kurento Media Server](#)
  - [Kurento API, Clients, and Protocol](#)
  - [Creating applications with Kurento](#)
- [Kurento Media Server Installation](#)
- [Kurento Tutorials](#)
  - [Tutorial 1 - Hello world](#)
  - [Tutorial 2 - WebRTC magic mirror](#)
  - [Tutorial 3 - WebRTC one-to-many broadcast](#)
  - [Tutorial 4 - WebRTC one-to-one video call](#)
  - [Tutorial 5 - WebRTC one-to-one video call with recording and filtering](#)
- [Mastering Kurento](#)
  - [Kurento Architecture](#)
  - [Kurento API Reference](#)
  - [Kurento Protocol](#)
  - [Advanced Installation Guide](#)

## Página web

### Servidor estático

- Página web estática
- Página web interactiva

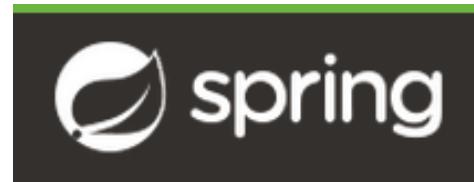
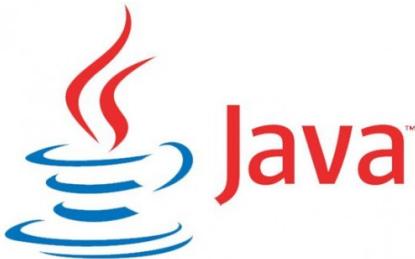
## Aplicación web

### Servidor dinámico

- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

# Evolución de la web

## Servidor dinámico



express



symfony



Portal de servicios Micael

miportal.urjc.es/portal/page/portal/portal\_inicio/

Inicio | Mis favoritos | Desconectar

 Universidad  
Rey Juan Carlos

# Portal de Servicios

Bienvenido/a: Micael Gallego Carrillo

**Servicios Alumno**

- [Mi Correo de Alumno - \(Ayuda\)](#)
- [Mis datos personales](#)
- [Mis notas](#)
- [Mis encuestas](#)
- [Mis traslados](#)
- [Aula Virtual](#)
- Expediente**
- [Mi expediente](#)
- [Mi nota media del expediente](#)
- [Mi progreso académico](#)

**Alumnos**

 **Centro de atención telefónica al alumno (C.A.T.A.)**

Fecha de Actualización: 06-FEB-2014

El C.A.T.A es un servicio de la Universidad Rey Juan Carlos cuyo objetivo es asistir a los alumnos y futuros alumnos en su búsqueda de información, al tiempo que asesora y orienta en las posibles cuestiones que puedan surgir en su relación con nuestra institución.

Horario del servicio:

Horario del servicio: de lunes a viernes de 9:00 a 20:00 horas

Teléfono: 91 488 93 93

**Enlaces Generales**

[Acceso DreamSpark Premium MSDN](#)

**Enlaces Biblioteca**

[Recordar pin](#)

[Acceso remoto a los recursos electrónicos de la biblioteca](#)

**Enlaces Alumnos**

[Centro de Atención Telefónica para el Alumno \(CATA\)](#)

[Apple on Campus](#)

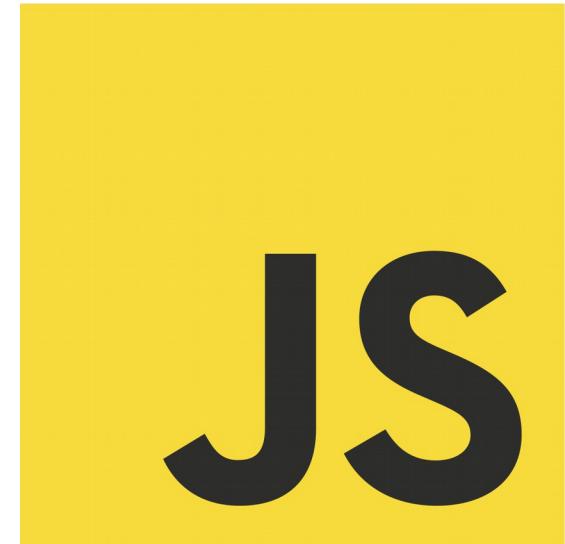
[CONVIVE - PROGRAMA](#)



# Taxonomía de las webs

- **Aplicaciones web con JavaScript**

- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA



## Página web

### Servidor estático

- Página web estática
- Página web interactiva

## Aplicación web

### Servidor dinámico

- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

[Sign Up](#)[Log In](#)

## create a new account

**Your Email**

Please use a valid email address.

**Confirm Email****Country****Zip Code**

Please provide a valid Zip Code

**Password**

Your password must be between 6-16 characters long.

**Confirm Password**

This does not match the password entered above.

Yes, I agree to the [Terms of Use](#)

[Sign Up](#)

### Why you'll love it

- See all your accounts in one place
- Set a budget and pay down your debt
- Find the best ways to grow your money
- Stay safe and secure

## Página web

### Servidor estático

- Página web estática
- Página web interactiva

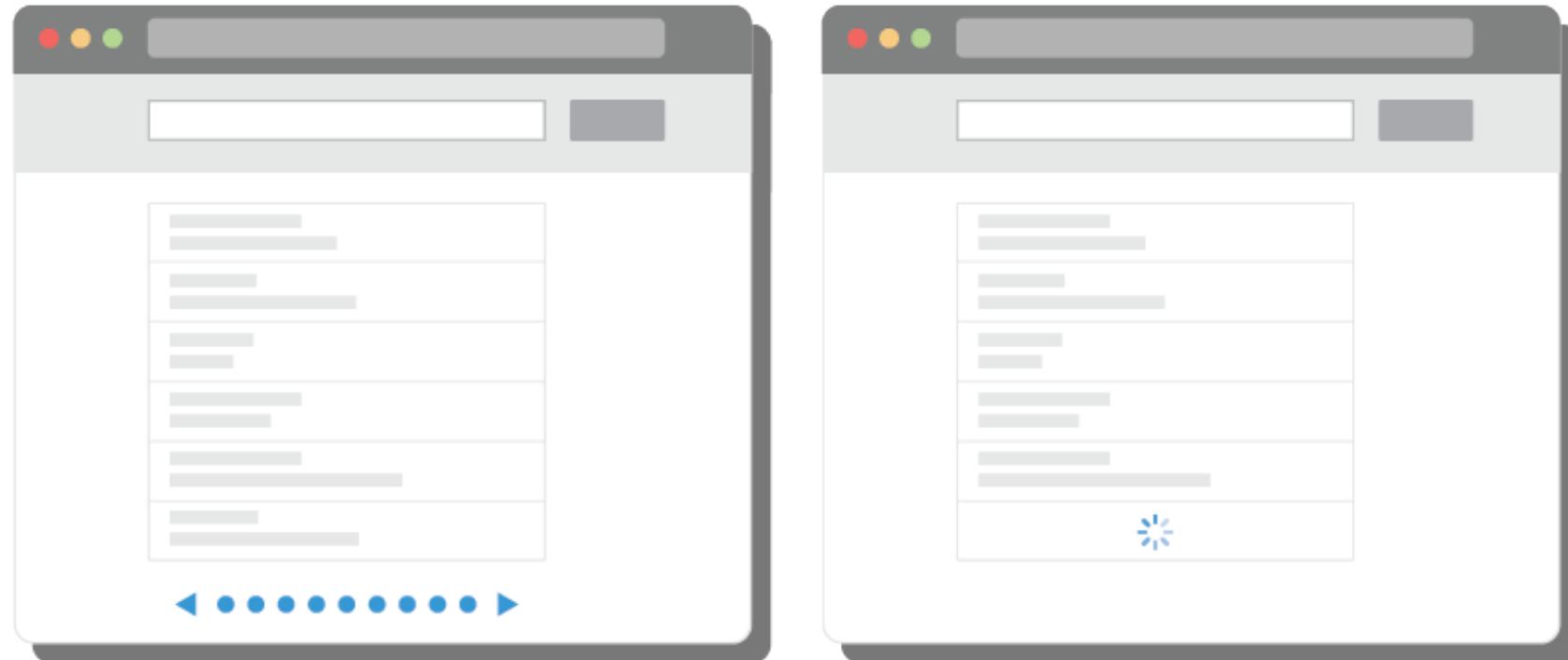
## Aplicación web

### Servidor dinámico

- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

# Taxonomía de las webs

- Aplicación web con AJAX



- Aplicación web con AJAX



**AJAX Username Verification**

NOTE: Please type an username and continue filling the other fields. You'll see the result immediately.

Already existing members in this demo: **john, michael, terry, steve**

Username:  ✓

Password:

Confirm Password:

## Página web

### Servidor estático

- Página web estática
- Página web interactiva

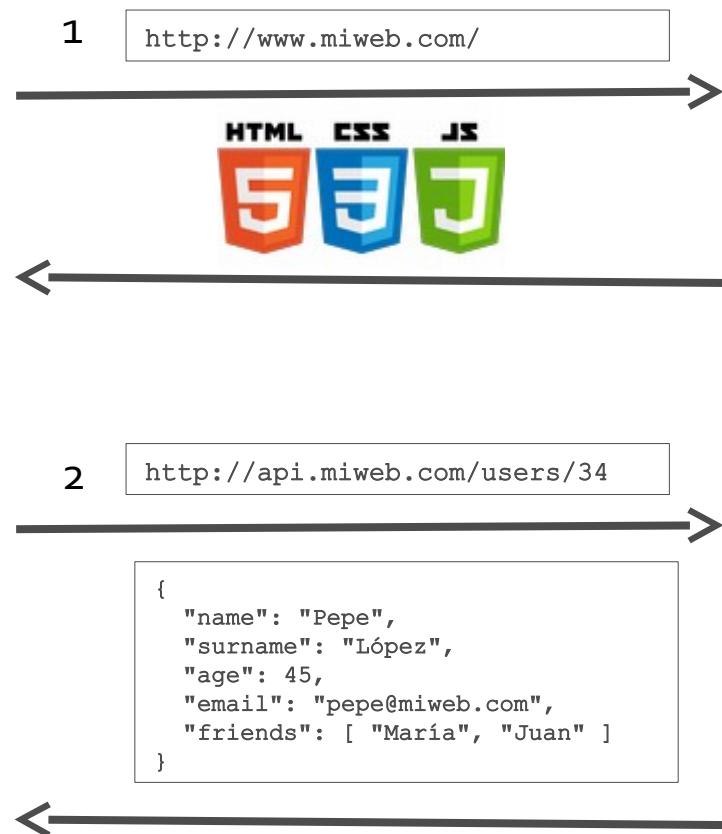
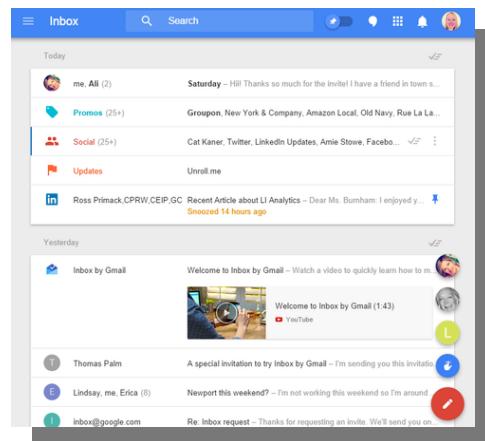
## Aplicación web

### Servidor dinámico

- Aplicación web con cliente estático
- Aplicación web interactiva
- Aplicación web con AJAX
- Aplicación web SPA

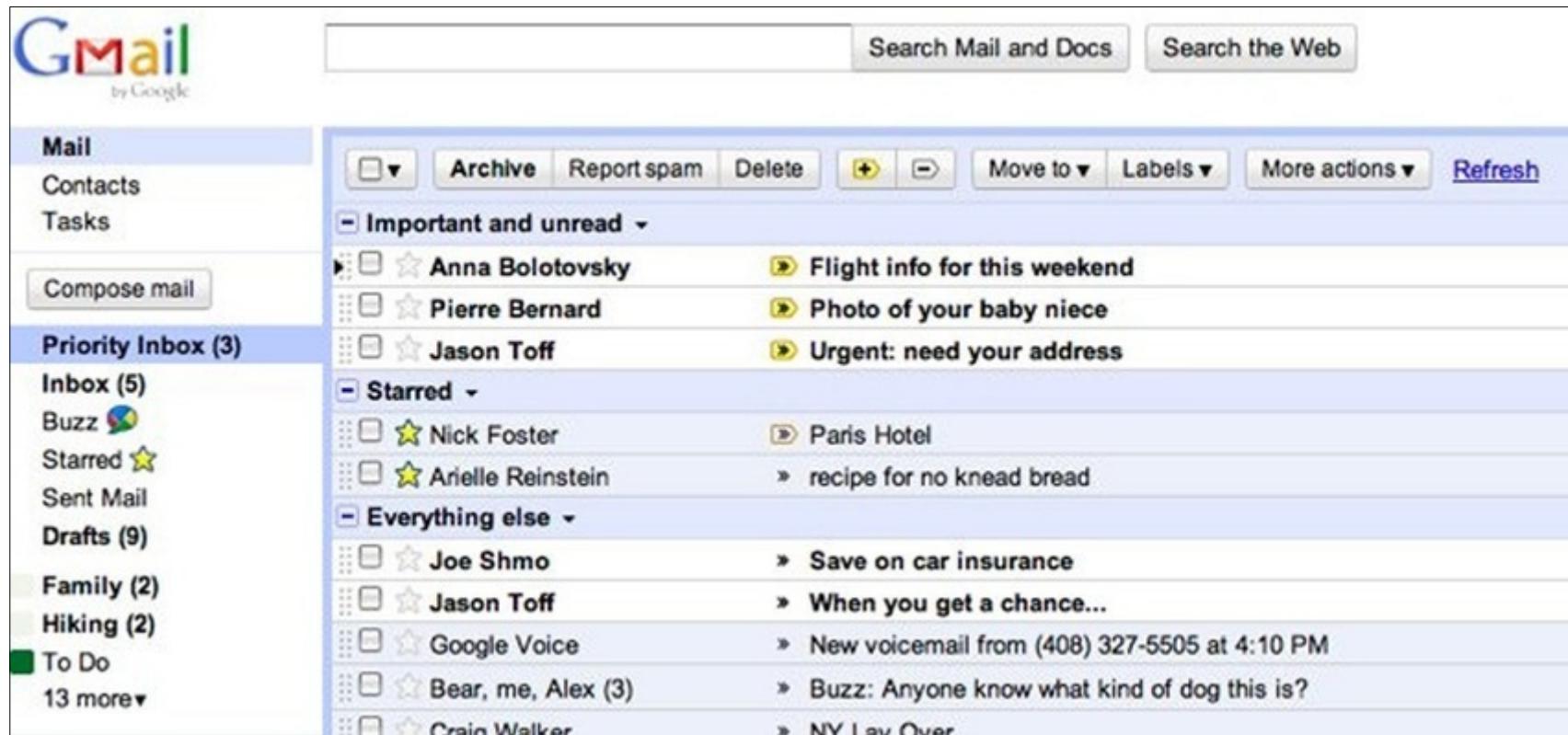
# Taxonomía de las webs

- Aplicación web SPA



# Taxonomía de las webs

- Aplicación web SPA



# Google Maps

Universidad Rey Juan Carlos - Google Maps - Mozilla Firefox

Universidad Rey Juan Car... +

<https://www.google.es/maps/place/Universidad+Rey+Juan+Carlos/@40.3347481,-3.8755547,15z>

Buscar

Escuela Superior De Ciencias...

Micael

1

Universidad Rey Juan Carlos, Calle Tulipán, Móstoles

Universidad Rey Juan Carlos

3,8 ★★★★☆ 66 reseñas

Cómo llegar

Universidad

GUARDAR LUGARES CERCANOS ENVIAR A TU TELÉFONO COMPARTIR

Calle Tulipán, s/n, 28933 Móstoles, Madrid

urjc.es

916 65 50 60

Cerrado hoy

Escuela Oficial de Idiomas de Móstoles

Instituto de Educación Secundaria Benjamín Rúa

IES Los Rosales

Universidad Rey Juan Carlos

CAFETERIA HARVARD CAFÉ

Av. del Alcalde de Móstoles

Calle Alonso Cano

Av. de los Abogados de Atocha

Calle Gran Capitán

Ceil Alonso Cano

Tierra

Datos del mapa ©2016 Google, Inst. Geogr. Nacional, Privacidad, Condiciones, Colegio Público de Madrid, Danos tu opinión, 100 m

# Soundcloud

That's How I Feel by FIXED FETISH - Mozilla Firefox

Thats How I Feel by Fl...

<https://soundcloud.com/groups/speedsound> | Buscar

SOUNDCLOUD Charts Search for artists, bands, tracks, podcasts Sign in or Create account Upload ...

  
SPEEDSOUND



FIXED FETISH Thats How I Feel 2 hours #Electronic  
0:07 3:39

Write a comment

Like Share Embed 7



TriWALKER Zen Party (LISTEN. 1 month #EDM  
MFers) (FREE DL)  
5:49

Buy 132 2

Moderators [View all](#)

 Speedsound REC.  
 28K 

 BLUE EYES KILL  
 19.9K [Follow](#)

164K members [View all](#)

Legal - Privacy - Cookies - Imprint - Popular searches  
Language: English (US)

Playing from :: Electronic Music Lovers Network (...  
Thats How I Feel

0:07 3:39

Back Next

Slides: Edit - Mozilla Firefox

Slides: Edit

https://slides.com/micaelgallego/deck-2/edit

Buscar

Shape

Line

Iframe

Table

Code

Math

Upgrade

+

Universidad  
Rey Juan Carlos

grafo  
RESEARCH

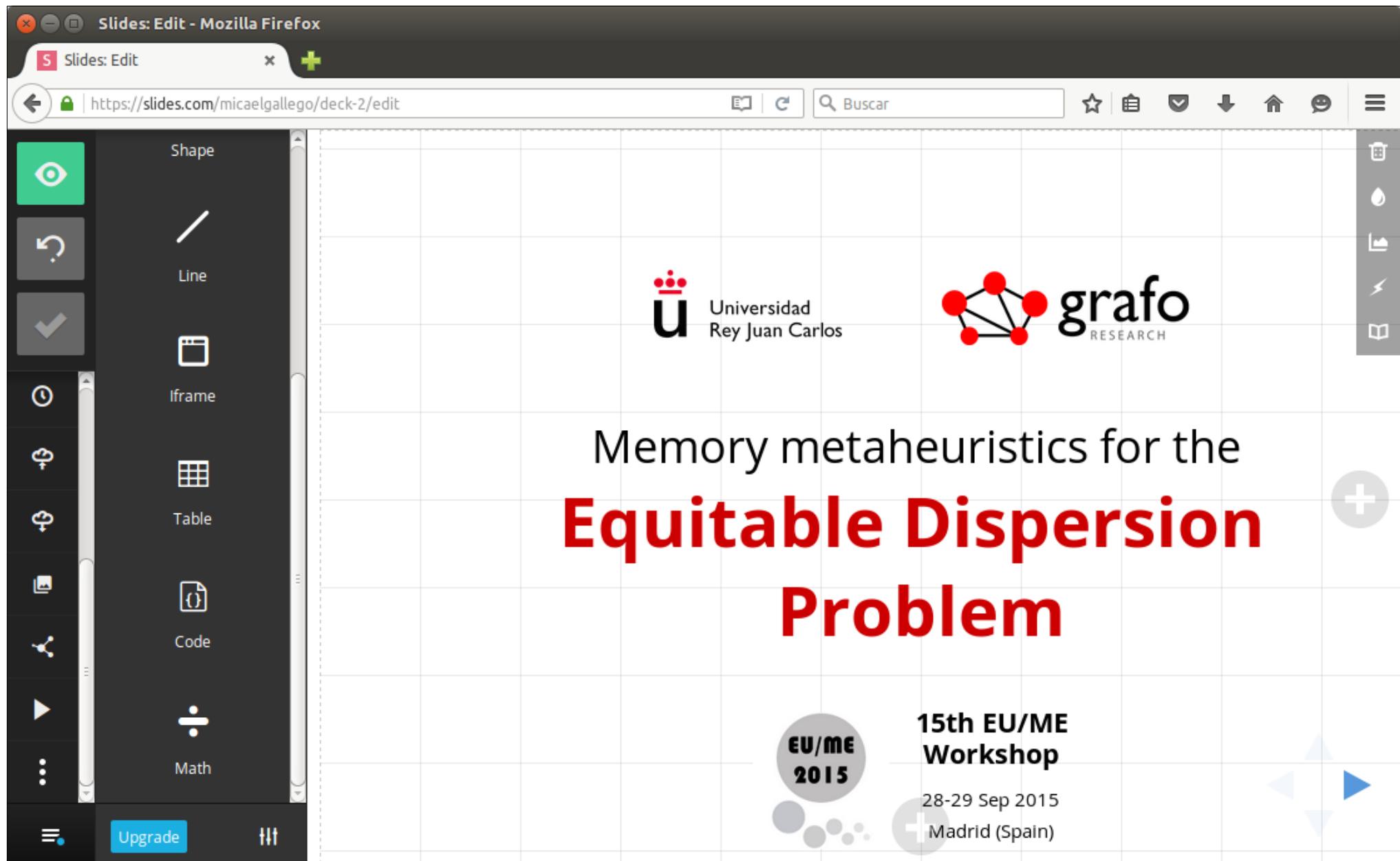
Memory metaheuristics for the  
**Equitable Dispersion  
Problem**

EU/ME  
2015

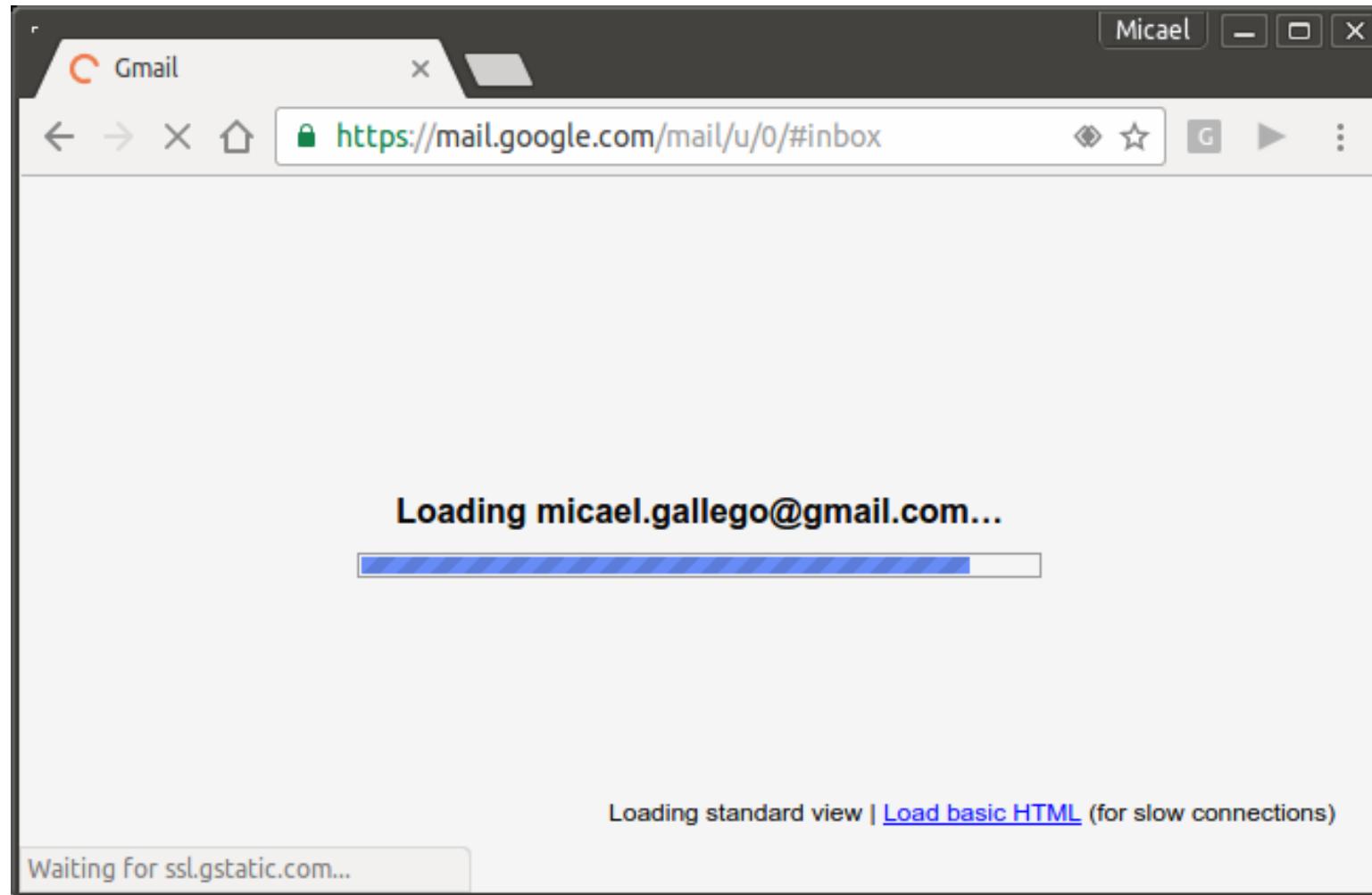
15th EU/ME  
Workshop

28-29 Sep 2015

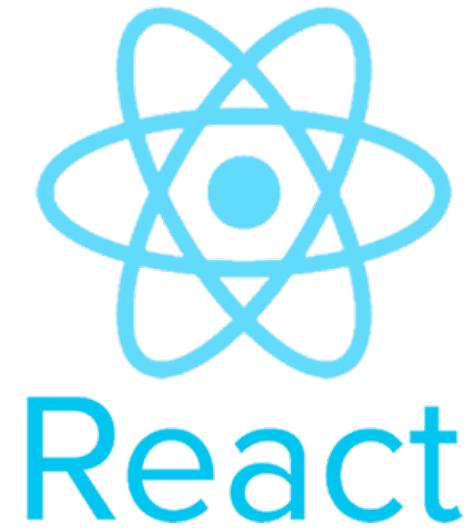
Madrid (Spain)



# Taxonomía de las webs



## Frameworks / librerías SPA





# Angular

## Características



- Angular es un framework para desarrollo **SPA**
- Permite extender el **HTML con etiquetas propias**
  - Con contenido personalizado (HTML)
  - Con comportamiento personalizado (JavaScript)
- Interfaz basado en **componentes** (no en páginas)
- Se recomienda usar con **TypeScript** (aunque se puede con ES5 y ES6)

<https://angular.io/>

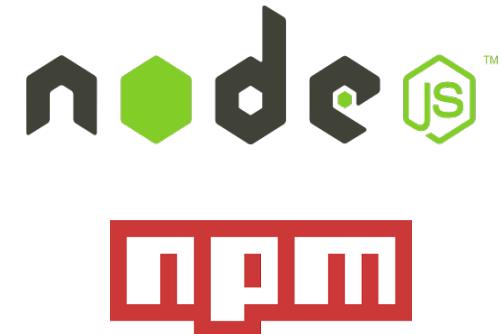
Google

## Funcionalidades

- Composición de componentes
- Inyección de dependencias
- Servicios (sin vista)
- Navegación por la app (Router)
- Cliente http (APIs REST)
- Animaciones
- Internacionalización
- Soporte para tests unitarios y e2e
- Librerías de componentes: Material Design
- Renderizado en el servidor (SEO friendly)
- ...

# Herramientas

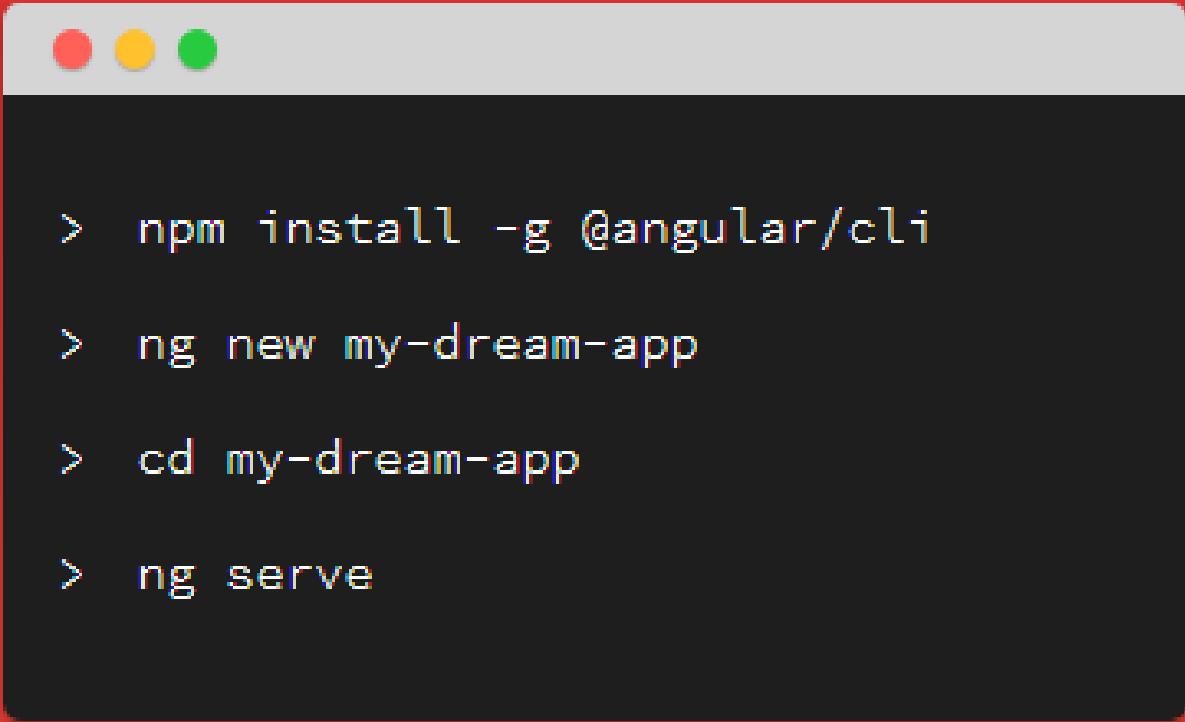
# Angular 2



- Generación del proyecto **inicial**
- Generación de partes **posteriormente**
- Compilado automático de **TypeScript** y actualización del **navegador**
- Construcción del proyecto para distribución (*build*)

<https://github.com/angular/angular-cli>

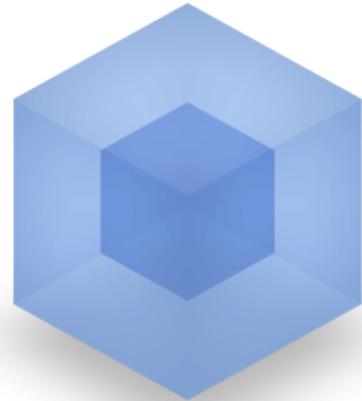
# Angular



```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Un proyecto ocupa unos **124Mb** en disco (se puede reutilizar las librerías entre varios proyectos)

# Angular



**webpack**  
MODULE BUNDLER

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

**Construcción  
del proyecto**

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

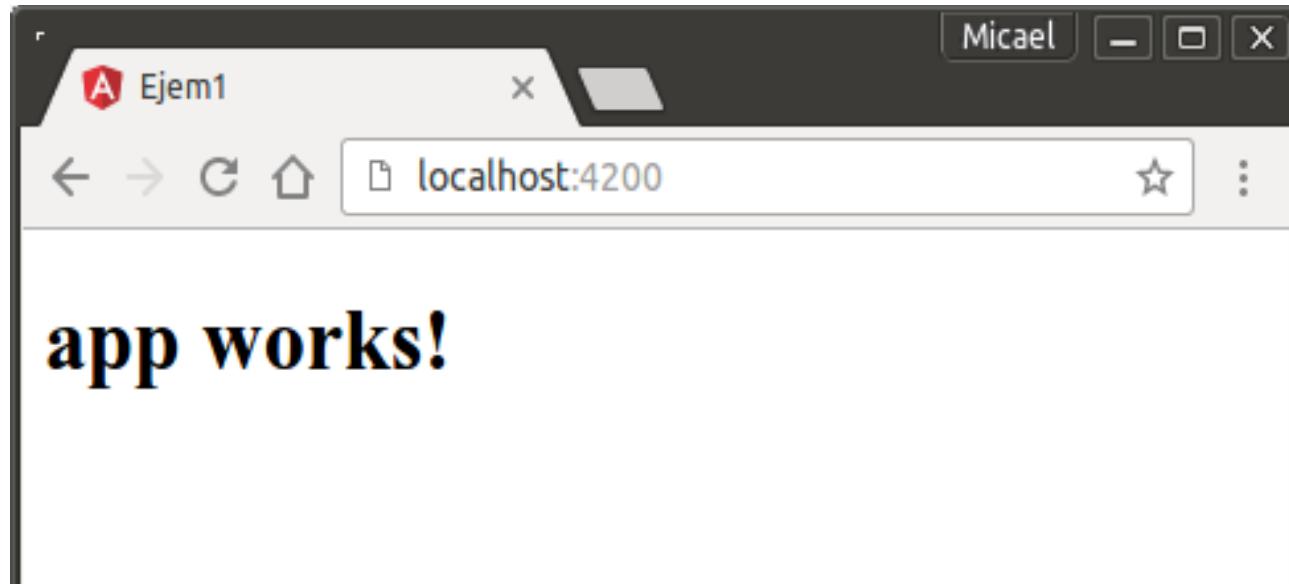
 Jasmine  
<http://jasmine.github.io/>

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

**Herramientas de testing**

# Angular

- **Ejecutar servidor web en modo desarrollo**
  - Se iniciará un servidor web en <http://localhost:4200>
  - Hay que abrir el navegador para ejecutar la app
  - Al guardar un fichero fuente, la aplicación se **recargará automáticamente**



# Angular

```
▽ ejem1
  > e2e
  > node_modules
  > src
  ⚙ .editorconfig
  ♦ .gitignore
  { angular.json
  ≡ browserslist
  K karma.conf.js
  { package-lock.json
  { package.json
  i README.md
  { tsconfig.app.json
  TS tsconfig.json
  { tsconfig.spec.json
  { tslint.json
```

- **Ficheros/Carpetas generadas**
  - **src:** Fuentes de la aplicación
  - **node\_modules:** Librerías y herramientas descargadas
  - **e2e:** Testing end to end
  - **package.json:** Configuración de librerías y herramientas
  - **angular.json:** Configuración de angular-cli
  - **tsconfig.json:** Configuración del compilador TS

# Angular 2

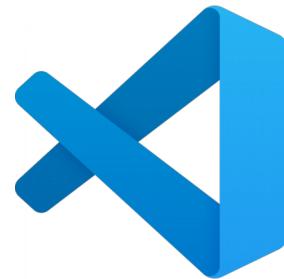
```
✓ src
  ✓ app
    ◁ app.component.html
    TS app.component.ts
    TS app.module.ts
  ✓ assets
    ♦ .gitkeep
    > environments
    ★ favicon.ico
    ◁ index.html
    TS main.ts
    TS polyfills.ts
    # styles.css
    TS test.ts
```

- **Ficheros/Carpetas generadas**
  - **app:**
    - Carpeta que contiene los ficheros **fuentes principales** de la aplicación.
    - **Borraremos su contenido** y le sustituiremos por los **ejemplos**
  - **assets:** Ficheros que se incluyen en el bundle cuando la aplicación se empaqueta para producción
  - **main.ts:** Fichero principal de la aplicación. No es necesario modificarle
  - **favicon.ico:** Icono de la aplicación
  - **index.html:** Página principal. No es necesario modificarle
  - **styles.css:** Estilos globales de la app

## Editores / IDEs



Sublime Text



Visual Studio  
Code



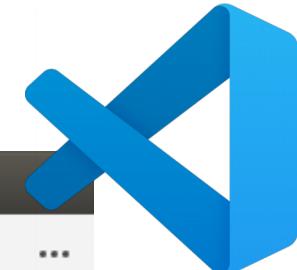
WebStorm



NetBeans



# Visual Studio Code



The screenshot shows the Visual Studio Code interface with the title bar "script2.ts - ejem2 - Visual Studio Code". The left sidebar displays the file tree under "OPEN EDITORS" and "EJEM2". The main editor pane contains the following TypeScript code:

```
1  export function process(){
2
3      let num = 3;
4
5      num = "ss";
6
7      return num;
8
9 }
10
11
12
13
14
```

A status bar at the bottom indicates "1 ERROR" and provides file navigation information: "Ln 9, Col 2".

The error list shows one error for "script2.ts":

- [ts] Type 'string' is not assignable to type 'number'. (5, 5)

<https://code.visualstudio.com>

- Actualización del mensaje de la app

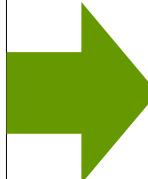
app.component.html

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

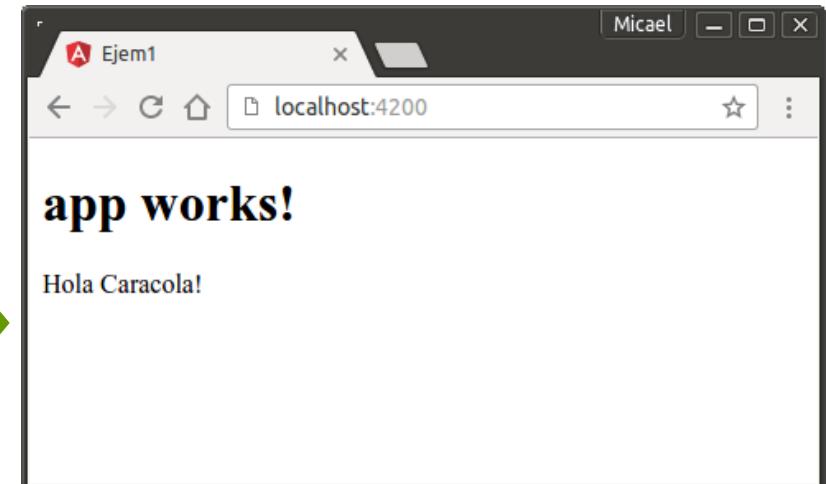
app.component.html

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

Hola caracola!



Al guardar un fichero el navegador se recarga de forma automática y vemos los cambios



# Angular

- **Descargar los ejemplos**
  - Descargar de github, descargar librerías y ejecutar un ejemplo

```
git clone https://github.com/codeurjc/curso-angular  
cd curso-angular
```

- Ejecutar un ejemplo

```
cd ejem1  
npm install  
ng serve
```

# Angular 2

## • Mover librerías

- Descargar una carpeta **node\_modules** por cada ejemplo puede suponer un gasto importante de disco duro y de tiempo de ejecución
- En la mayoría de los ejemplos podemos mover la carpeta la carpeta node\_modules de un ejemplo a otro

```
$ mv curso-angular/ejem1/node_modules \
curso-angular/ejem2/node_modules
```

## Lenguaje programación Angular

- Angular 2 tiene soporte oficial para desarrollo de apps con JavaScript (**ES5** y **ES6**) y **TypeScript**
- Se recomienda usar **TypeScript**





# TypeScript

# TypeScript

## Características

- JavaScript ES6 con tipos opcionales
- Las herramientas
  - Avisan de potenciales **errores**
  - Generan código JavaScript ES5 compatible con los browsers
- Integrado en todos los **editores / IDEs**

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

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



## Ventajas frente a JavaScript

- Con los **tipos**, las herramientas pueden avisar de potenciales problemas mucho mejor que un **linter**
- Los programas grandes son **menos propensos a errores**
- Los **IDEs** y **editores** pueden: Autocompletar, Refactorizar, Navegar a la definición
- Se puede programar muy parecido a **Java o C#** (pero no estás forzado a ello)

# TypeScript

## Facilidad de adopción para JavaScripters

- Existe mucha inferencia de tipos (evita tener que escribir los tipos constantemente)
- No tiene que cambiarse la forma de programar (no te obliga a programar con clases)
- Un mismo proyecto puede **combinar JS y TS**, lo que facilita migrar un proyecto existente

# TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string, salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
            ", Salario:"+this.salario;  
    }  
}
```

TypeScript

## Clases TypeScript vs JavaScript ES5

### Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
               salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

### Simulación de clase en JS ES5 con prototipos

```
function Empleado(nombre, salario){  
  
    this.nombre = nombre;  
    this.salario = salario;  
}  
  
Empleado.prototype.getNombre = function(){  
    return this.nombre;  
};  
  
Empleado.prototype.toString = function(){  
    return "Nombre:"+this.nombre+,  
           Salario:"+this.salario;  
};
```

## Clases Java vs TypeScript

### Clase en Java

```
public class Empleado {  
  
    private String nombre;  
    private double salario;  
  
    public Empleado(String nombre,  
                    double salario){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
  
    public String toString(){  
        return "Nombre:"+nombre+  
               ", Salario:"+salario;  
    }  
}
```

### Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
                salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

## Clases Java vs TypeScript

Clase en Java

```
public class Empleado {  
  
    private String nombre;  
    private double salario;  
  
    public Empleado(String nombre,  
                    double salario){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
  
    public String toString(){  
        return "Nombre:"+nombre+  
               ", Salario:"+salario;  
    }  
}
```

Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
                salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

En Java las clases son públicas. En TypeScript las clases se exportan

## Clases Java vs TypeScript

### Clase en Java

```
public class Empleado {  
  
    private String nombre;  
    private double salario;  
  
    public Empleado(String nombre,  
                    double salario){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
  
    public String toString(){  
        return "Nombre:"+nombre+  
               ", Salario:"+salario;  
    }  
}
```

### Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
               salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

En TS los tipos se ponen tras el nombre con : (dos puntos)

## Clases Java vs TypeScript

### Clase en Java

```
public class Empleado {  
  
    private String nombre;  
    private double salario;  
  
    public Empleado(String nombre,  
                    double salario){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
  
    public String toString(){  
        return "Nombre:"+nombre+  
               ", Salario:"+salario;  
    }  
}
```

### Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
                salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

En TS el constructor es “constructor”

## Clases Java vs TypeScript

### Clase en Java

```
public class Empleado {  
  
    private String nombre;  
    private double salario;  
  
    public Empleado(String nombre,  
                    double salario){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
  
    public String toString(){  
        return "Nombre:"+nombre+  
               ", Salario:"+salario;  
    }  
}
```

### Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
                salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

En TS los tipos se infieren (si quieras), incluso en el tipo que devuelven los métodos

## Clases Java vs TypeScript

### Clase en Java

```
public class Empleado {  
  
    private String nombre;  
    private double salario;  
  
    public Empleado(String nombre,  
                    double salario){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
  
    public String toString(){  
        return "Nombre:"+nombre+  
               ", Salario:"+salario;  
    }  
}
```

### Clase en TypeScript

```
export class Empleado {  
  
    private nombre:string;  
    private salario:number;  
  
    constructor(nombre:string,  
                salario:number){  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    getNombre(){  
        return this.nombre;  
    }  
  
    toString(){  
        return "Nombre:"+this.nombre+  
               ", Salario:"+this.salario;  
    }  
}
```

En TS siempre que quieras acceder a un elemento de la clase tendrás que usar this.

# TypeScript

## TypeScript

```
import { Empleado } from "./Empleado";

let emps = new Array<Empleado>();

emps.push(new Empleado('Pepe', 500));
emps.push(new Empleado('Juan', 200));

for(let emp of emps){
    console.log(emp.getNombre());
}

empleados.forEach(emp => {
    console.log(emp);
});
```

# TypeScript

## Imports / Listas / foreach / lambdas

### Java

```
List<Empleado> emps = new ArrayList<>();  
  
emps.add(new Empleado('Pepe', 500));  
emps.add(new Empleado('Juan', 200));  
  
for(Empleado emp : emps){  
    System.out.println(emp.getNombre());  
}  
  
empleados.forEach(emp -> {  
    System.out.println(emp);  
});
```

### TypeScript

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

# TypeScript

## Imports / Listas / foreach / lambdas

Java

```
List<Empleado> emps = new ArrayList<>();
emps.add(new Empleado('Pepe', 500));
emps.add(new Empleado('Juan', 200));

for(Empleado emp : emps){
    System.out.println(emp.getNombre());
}

empleados.forEach(emp -> {
    System.out.println(emp);
});
```

TypeScript

```
import { Empleado } from "./Empleado";

let emps = new Array<Empleado>();
emps.push(new Empleado('Pepe', 500));
emps.push(new Empleado('Juan', 200));

for(let emp of emps){
    console.log(emp.getNombre());
}

empleados.forEach(emp => {
    console.log(emp);
});
```

En Java las clases de la misma carpeta son del mismo paquete.

En TS cada fichero es un módulo, por eso hay que importar desde otros ficheros

# TypeScript

## Imports / Listas / foreach / lambdas

### Java

```
List<Empleado> emps = new ArrayList<>();

emps.add(new Empleado('Pepe', 500));
emps.add(new Empleado('Juan', 200));

for(Empleado emp : emps){
    System.out.println(emp.getNombre());
}

empleados.forEach(emp -> {
    System.out.println(emp);
});
```

### TypeScript

```
import { Empleado } from "./Empleado";

let emps = new Array<Empleado>();

emps.push(new Empleado('Pepe', 500));
emps.push(new Empleado('Juan', 200));

for(let emp of emps){
    console.log(emp.getNombre());
}

empleados.forEach(emp => {
    console.log(emp);
});
```

En TS las variables se declaran con let y tienen ámbito de bloque y no se pueden declarar dos veces. Se podría usar var (como JS), pero el ámbito sería la función y se podrían redeclarar

## Imports / Listas / foreach / lambdas

Java

```
List<Empleado> emps = new ArrayList<>();  
  
emps.add(new Empleado('Pepe', 500));  
emps.add(new Empleado('Juan', 200));  
  
for(Empleado emp : emps){  
    System.out.println(emp.getNombre());  
}  
  
empleados.forEach(emp -> {  
    System.out.println(emp);  
});
```

TypeScript

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

En TS las variables se pueden declarar con tipo (después de :), pero es opcional porque el tipo se infiere de la inicialización. En Java no lo veremos hasta Java 9 o Java 10

## Imports / Listas / foreach / lambdas

Java

```
List<Empleado> emps = new ArrayList<>();  
  
emps.add(new Empleado('Pepe', 500));  
emps.add(new Empleado('Juan', 200));  
  
for(Empleado emp : emps){  
    System.out.println(emp.getNombre());  
}  
  
empleados.forEach(emp -> {  
    System.out.println(emp);  
});
```

TypeScript

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

En Java usamos List y ArrayList generificados del API.  
En TS usamos el Array nativo de JS generificado por TS.

## Imports / Listas / foreach / lambdas

Java

```
List<Empleado> emps = new ArrayList<>();  
  
emps.add(new Empleado('Pepe', 500));  
emps.add(new Empleado('Juan', 200));  
  
for(Empleado emp : emps){  
    System.out.println(emp.getNombre());  
}  
  
empleados.forEach(emp -> {  
    System.out.println(emp);  
});
```

TypeScript

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

En Java List el método es “add”  
En el array de JS el método es “push”

## Imports / Listas / foreach / lambdas

### Java

```
List<Empleado> emps = new ArrayList<>();  
  
emps.add(new Empleado('Pepe', 500));  
emps.add(new Empleado('Juan', 200));  
  
for(Empleado emp : emps){  
    System.out.println(emp.getNombre());  
}  
  
empleados.forEach(emp -> {  
    System.out.println(emp);  
});
```

### TypeScript

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

La sintaxis del foreach es muy parecida en Java y TS.  
En TS está basado en iteradores (como en Java)

## Imports / Listas / foreach / lambdas

### Java

```
List<Empleado> emps = new ArrayList<>();  
  
emps.add(new Empleado('Pepe', 500));  
emps.add(new Empleado('Juan', 200));  
  
for(Empleado emp : emps){  
    System.out.println(emp.getNombre());  
}  
  
empleados.forEach(emp -> {  
    System.out.println(emp);  
});
```

### TypeScript

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

Las expresiones lambda de Java se llaman arrow function en TS.  
Se diferencian en la “flecha” con – o con =

# TypeScript

## Lambdas / Arrow functions

Java

```
int num = 0;  
  
empleados.forEach(emp -> num++);  
  
System.out.println(num);
```

ERROR

TypeScript

```
let num = 0;  
  
empleados.forEach(emp => num++);  
  
console.log(num);
```

En Java no se puede acceder a una variable que no sea final (declarada o efectiva) desde una lambda. En TS (como en JS) incluso se puede cambiar el valor de la variable desde la propia arrow function

# TypeScript

## Uso de `this` con la arrow function

### JavaScript

```
function Empleado(nombre, sueldo){
    this.nombre = nombre;
    this.sueldo = sueldo;
}

Empleado.prototype.alerta(button){
    var that = this;
    button.onclick = function(e){
        alert(that.nombre);
    }
}
```

### TypeScript

```
export class Empleado {
    private nombre:string,
    private sueldo:number}{}

    alerta(button:HTMLButtonElement){
        button.onclick = e => {
            alert(this.nombre);
        }
    }
}
```

En TS una arrow function permite usar **this** y siempre apunta al objeto (como Java). En JS, dentro de una función this puede cambiar de valor (y es necesario usar **that** para referirse al objeto)

# TypeScript

## Anotaciones

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

@Component({
  selector: 'app',
  templateUrl: 'app.component.html'
})
export class AppComponent {

}
```

# TypeScript

## Definición de atributos en el constructor

```
class Animal {  
    private name:string;  
    constructor(name: string) {  
        this.name = name;  
    }  
}
```

# TypeScript

## Definición de atributos en el constructor

```
class Animal {  
    private name:string;  
    constructor(name: string) {  
        this.name = name;  
    }  
}
```

```
class Animal {  
    constructor(private name: string) {  
    }  
}
```

# TypeScript

## Definición de atributos en el constructor

```
class Animal {  
    private name:string;  
    constructor(name: string) {  
        this.name = name;  
    }  
}
```

```
class Animal {  
    constructor(private name: string) {  
    }  
}
```

En TS se puede declarar un atributo e inicializar su valor desde el constructor declarando ese atributo como parámetro del constructor y usando el modificador de visibilidad

# TypeScript

## Getter / Setter con sintaxis de atributo

```
class Foo {  
  
    get bar() {  
        return ...;  
    }  
  
    set bar(bar: boolean) {  
        ...  
    }  
}
```

```
let foo = new Foo();  
  
if(foo.bar){  
    foo.bar = false;  
}
```

# TypeScript

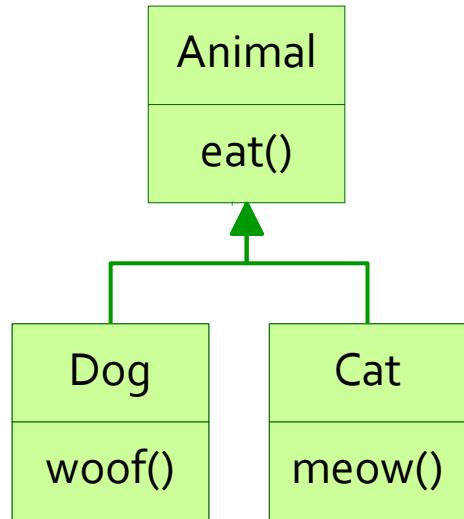
## Getter / Setter con sintaxis de atributo

```
class Foo {  
    get bar() {  
        return ...;  
    }  
    set bar(bar: boolean) {  
        ...  
    }  
}
```

```
let foo = new Foo();  
if(foo.bar){  
    foo.bar = false;  
}
```

# TypeScript

## Type guards Instanceof / typeof



```
class Animal { eat() { } }
class Dog extends Animal { woof() { } }
class Cat extends Animal { meow() { } }

let pet: Animal = ...;
if (pet instanceof Dog) {
    pet.woof();
} else if (pet instanceof Cat) {
    pet.meow();
} else {
    pet.eat();
}
```

The code demonstrates type guards. The variable `pet` is declared as type `Animal`. The `instanceof` operator is used to check if `pet` is an instance of `Dog`. If true, the `woof()` method is called. If false, it checks if `pet` is an instance of `Cat`. If true, the `meow()` method is called. In both of these conditional blocks, the `pet` variable is annotated with a green oval. An arrow points from the `instanceof` check in the first block to the `pet` annotation, highlighting how the type guard narrows the type of `pet`.

# TypeScript

## Objetos literales “tipados”

```
interface SquareConfig {  
    color: string;  
    width?: number;  
}
```

```
let config: SquareConfig;  
  
config = {color: "black"};  
config = {color: "black", width: 20};
```

# TypeScript

## Objetos literales “tipados”

```
interface SquareConfig {  
    color: string;  
    width?: number;  
}
```

```
let config: SquareConfig;
```

```
config = {color: "black"};
```

```
config = {color: "black", width: 20};
```

# TypeScript

## Tipos unión

```
let id: string | number;  
  
id = 3;  
...  
Id = "3";  
  
if(typeof id === "string"){  
    ...  
}
```

En JS es habitual que ciertas variables puedan tener la misma información aunque se represente con varios “tipos”. TS permite definir variables con varios tipos

# TypeScript

## Compatibilidad de tipos estructural

```
interface User {  
    name: string;  
}  
  
class Profile {  
    constructor(public name:string){}  
}  
  
let u: User;  
u = { name: "Pepe" };  
u = new Profile("Pepe");
```

# Compatibilidad de tipos estructural

```
interface User {  
    name: string;  
}  
  
class Profile {  
    constructor(public name:string){}  
}  
  
let u: User;  
u = { name: "Pepe" };  
u = new Profile("Pepe");
```

Un objeto **Profile** puede asignarse a una variable de tipo **User** porque tiene un atributo **name**

# TypeScript

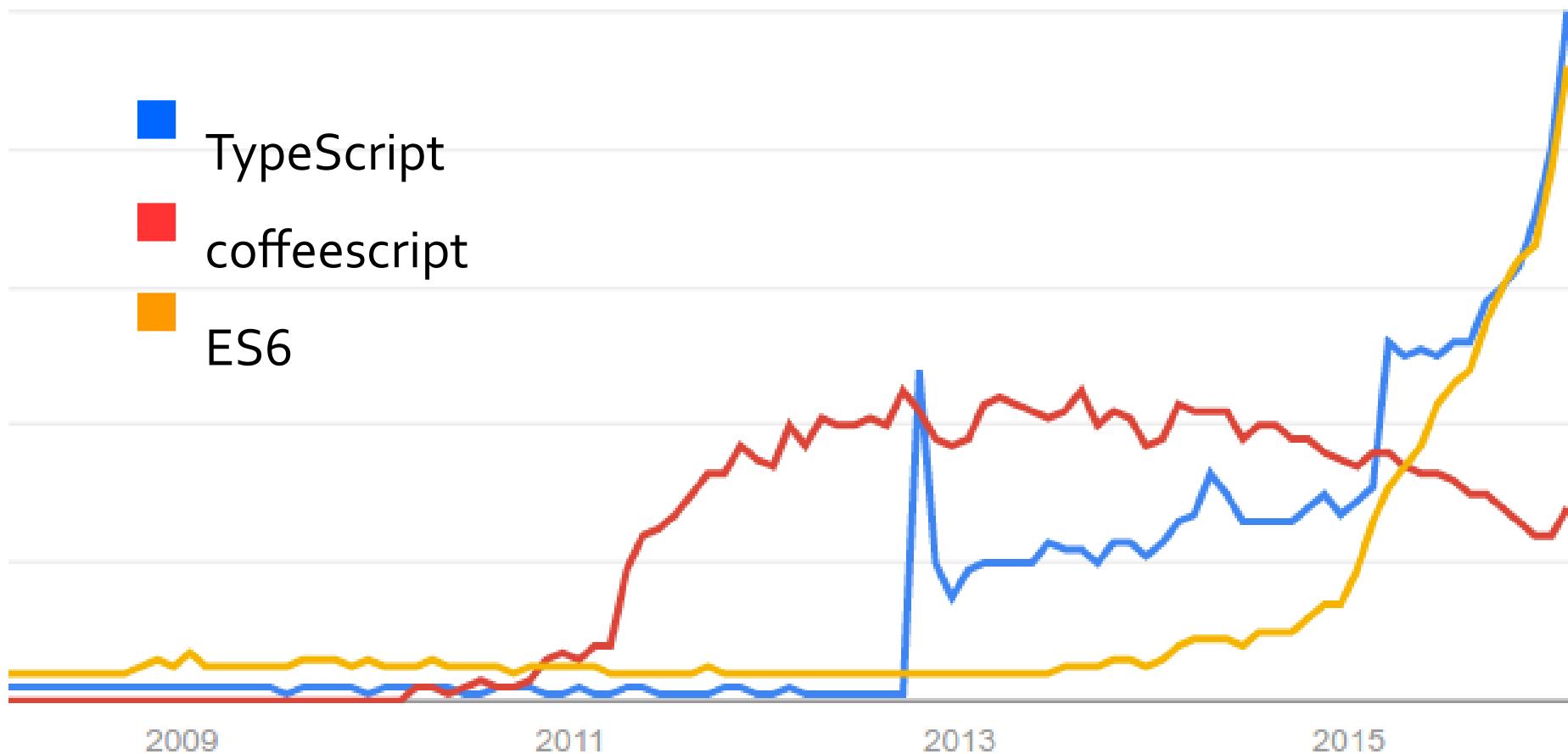
## Sobrecarga de métodos “especial”

```
class TestClass {  
  
    someMethod(p1: string): void;  
    someMethod(p1: number, p2: string): void;  
  
    someMethod(p1: string | number, p2?: string): void {  
  
        if (typeof p1 === "string"){  
            console.log("p1");  
        } else if(p2){  
            console.log("p2");  
        }  
    }  
}
```

Dos métodos con el mismo nombre deben tener una única implementación que detecte cuál ha sido llamado.  
La implementación tiene que cubrir todas las cabeceras

# TypeScript

## Popularidad de TypeScript

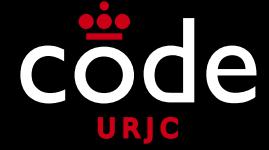


# Tema 2

# Componentes

Micael Gallego  
@micael\_gallego

# Angular



# Componentes

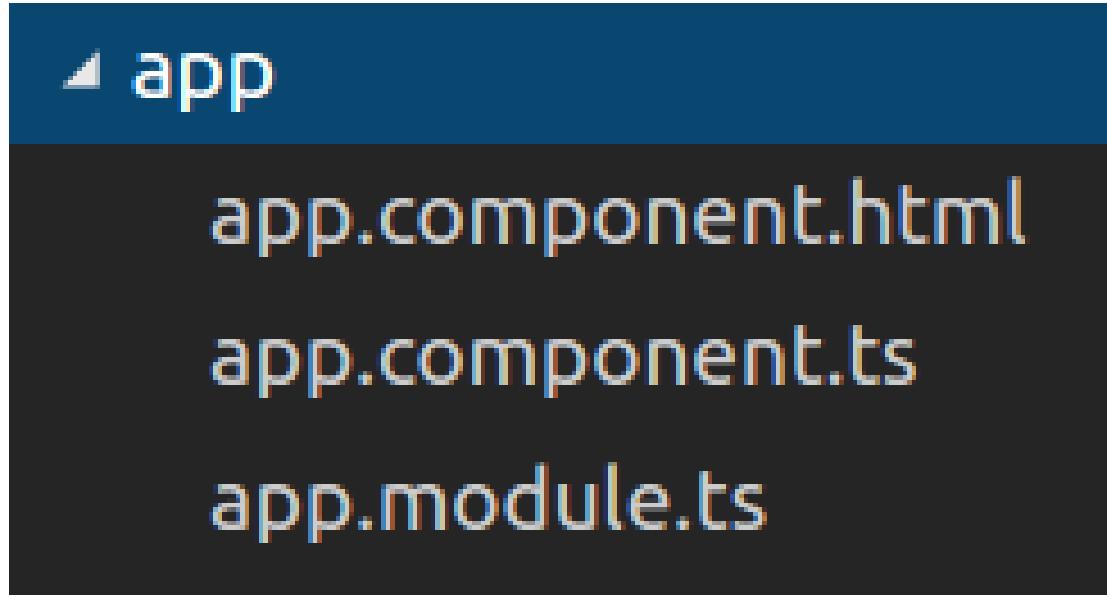
## Componentes en Angular

- Un componente es una **nueva etiqueta HTML** con una **vista** y una **lógica** definidas por el desarrollador
- La **vista** es una plantilla (*template*) en HTML con elementos especiales
- La **lógica** es una clase TypeScript vinculada a la vista

# Componentes

## Componentes en Angular

- Copiamos el contenido de la carpeta src/app del ejem1



# Componentes

ejem1

## Componentes en Angular

app.component.ts

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

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

app.component.html

```
<h1>My First Angular 2 App</h1>
```

Lógica

Vista

# Componentes

ejem1

## Componentes en Angular

app.component.ts

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

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

app.component.html

```
<h1>My First Angular 2 App</h1>
```

Lógica

Vista

# Componentes

ejem1

## Componentes en Angular

app.component.ts

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

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



Este componente no  
tiene ninguna lógica

app.component.html

```
<h1>My First Angular 2 App</h1>
```

Lógica

Vista

# Componentes

app.component.ts

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

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

**ejem1**

app.component.html

```
<h1>My First Angular 2 App</h1>
```

src/index.html

```
<html>
<head>...</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Para usar el componente se incluye en el **index.html** un **elemento HTML** con el nombre indicado en el selector (en este caso **app-root**)

# Componentes

ejem1

## Componentes en Angular

Toda app tiene un módulo que define los componentes de la app, el componente principal y qué otros módulos necesita

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { NgModule } from '@angular/core';

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

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

Componentes declarados

Módulos importados

Componente principal

# Componentes

ejem1



# Componentes

ejem1

## Componentes en Angular

app.component.ts

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

@Component({
  selector: 'app-root',
  template: `
    <h1>
      My First Angular 2 App
    </h1>
  `
})
export class AppComponent {
```

Se puede incluir la **vista** (HTML del template) directamente en la **clase**. Si se usa la tildes invertidas ( ` ) (grave accent), se puede escribir HTML multilínea

# Componentes

ejem2

## Visualización de una variable

La vista del componente (HTML) se genera en función de su estado (**atributos de la clase**)

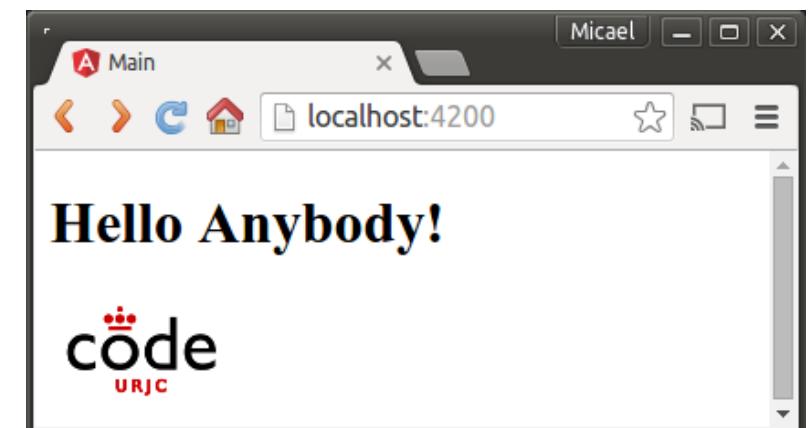
app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  name = 'Anybody';
  imgUrl = "assets/img.png";
}
```

app.component.html

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



# Componentes

ejem2

## Visualización de una variable

La vista del componente (HTML) se genera en función de su estado (atributos de la clase)

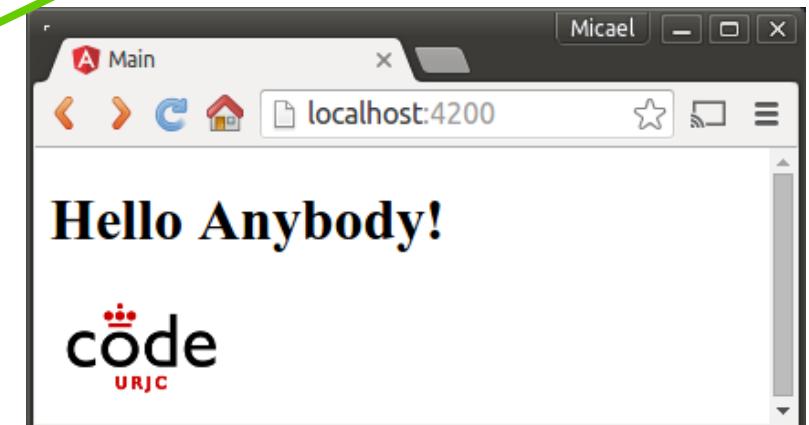
app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  name = 'Anybody';
  imgUrl = "assets/img.png";
}
```

app.component.html

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

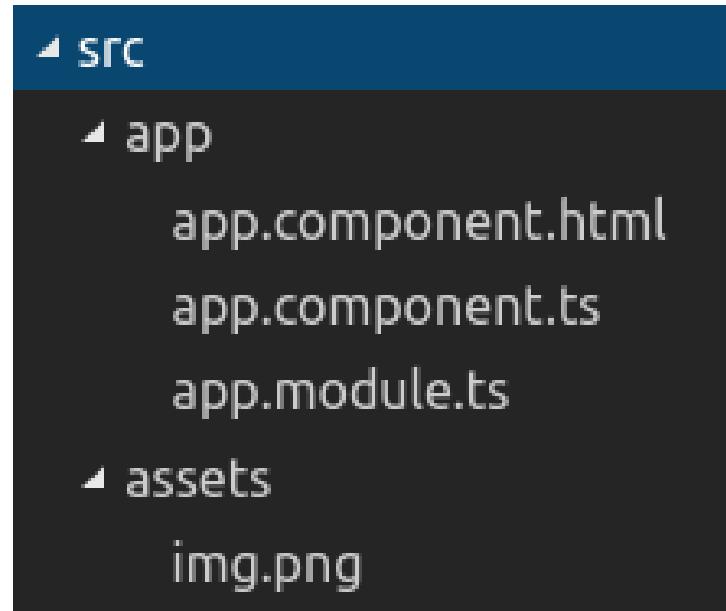


# Componentes

ejem2

## Recursos de la app

Los recursos (imágenes, fonts..) deben colocarse en una carpeta **src/assets** para que estén accesibles en desarrollo y cuando se genera el paquete de producción



# Componentes

ejem3

## Ejecución de lógica

Se puede ejecutar un método ante un evento producido en la vista del componente

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: './app.component.html'
})
export class AppComponent {

  name = 'Anybody';

  setName(name:string){
    this.name = name;
  }
}
```

app.component.html

```
<h1>Hello {{name}}!</h1>

<button (click)="setName('John')">
  Hello John
</button>
```

# Componentes

ejem3

## Ejecución de lógica

Se puede ejecutar un método ante un evento producido en la vista del componente

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: './app.component.html'
})
export class AppComponent {

  name = 'Anybody';

  setName(name:string){
    this.name = name;
  }
}
```

app.component.html

```
<h1>Hello {{name}}!</h1>

<button (click)="setName('John')">
  Hello John
</button>
```

# Componentes

ejem3

## Ejecución de lógica

Se puede ejecutar un método ante un evento producido en la vista del componente

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: './app.component.html'
})
export class AppComponent {

  name = 'Anybody';

  setName(name:string){
    this.name = name;
  }
}
```

app.component.html

```
<h1>Hello {{name}}!</h1>

<button (click)="setName('John')">
  Hello John
</button>
```

Se puede definir cualquier **evento** disponible en el **DOM** para ese elemento

# Componentes

ejem3

## Ejecución de lógica

Se puede ejecutar un método ante un evento producido en la vista del componente



ejem3

## Sintaxis de los templates

```
<h1>Hello {{name}}!</h1>  
  
<img [src]=" imgUrl "/>  
  
<button (click)="setName('John')">  
    Hello John  
</button>
```

`{{ attr }}`

Valor del atributo de la clase en el texto

`[prop] = "attr"`

Valor del atributo de la clase en la propiedad del elemento

`(event) = "metodo()"`

Se llama al método cuando se produce el evento

# Componentes

ejem4

## Datos enlazados (*data binding*)

Un campo de texto se puede “enlazar” a un atributo  
Atributo y campo de texto están sincronizados

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: './app.component.html'
})
export class AppComponent {
  name = 'Anybody';
  setName(name:string){
    this.name = name;
  }
}
```

app.component.html

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

<h1>Hello {{name}}!</h1>

<button (click)="setName('John')">
  Hello John
</button>
```

# Componentes

ejem4

## Datos enlazados (*data binding*)

Un campo de texto se puede “enlazar” a un atributo  
Atributo y campo de texto están sincronizados

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: './app.component.html'
})
export class AppComponent {
  name = 'Anybody';
  setName(name:string){
    this.name = name;
  }
}
```

app.component.html

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

<h1>Hello {{name}}!</h1>

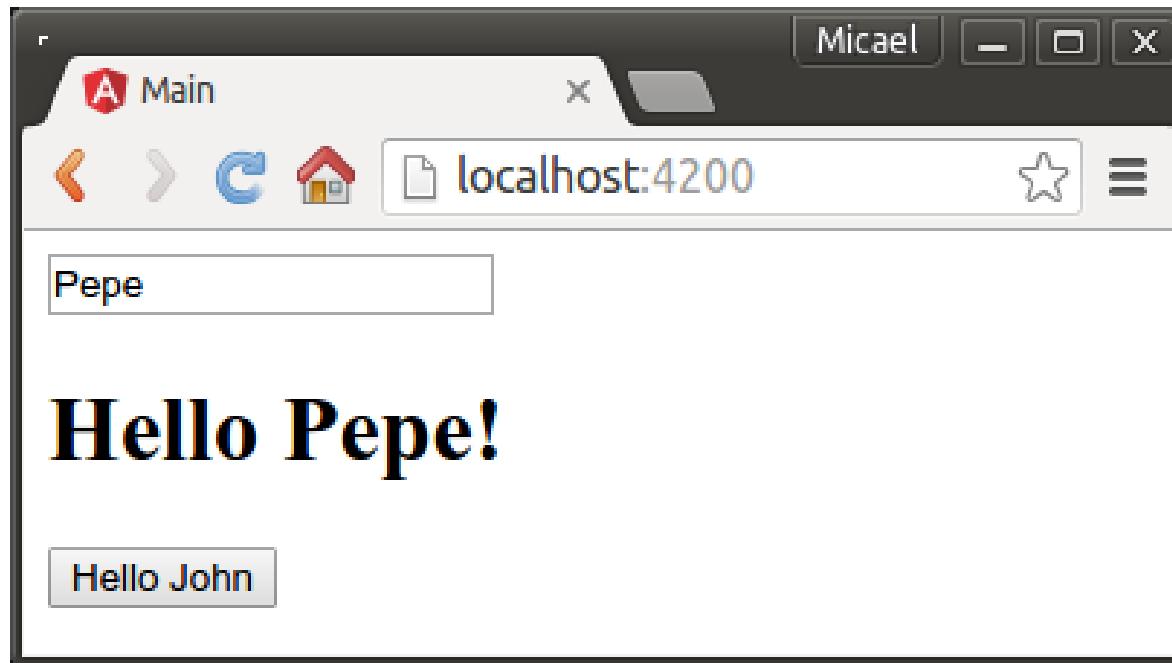
<button (click)="setName('John')">
  Hello John
</button>
```

# Componentes

ejem4

## Datos enlazados (*data binding*)

Un campo de texto se puede “enlazar” a un atributo  
Atributo y componente están sincronizados



# Templates

# Templates

- Los **templates** permiten definir la vista en función de la información del componente
  - Visualización condicional
  - Repetición de elementos
  - *Safe navigation operator*
  - Estilos
  - Formularios

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

# Templates

ejem5

- ## Visualización condicional

- 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**
- Por ejemplo dependiendo del **valor del atributo booleano visible**

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

- También se puede usar una **expresión**

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

# Templates

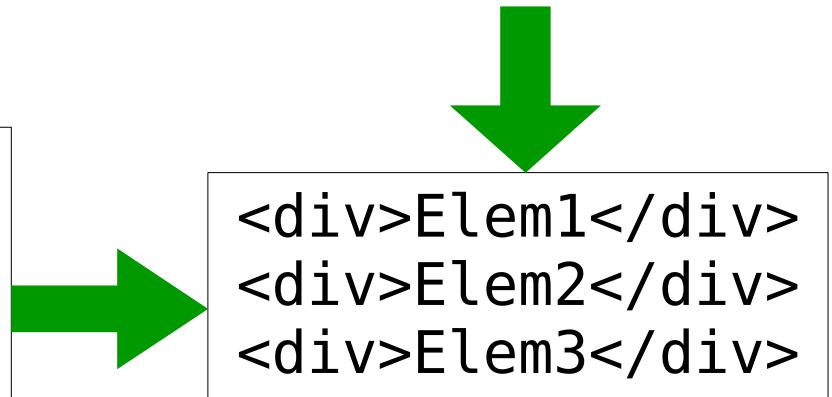
ejem5

- Repetición de elementos

- Es posible visualizar el contenido de un array con la directiva **ngFor**
- Se define cómo se visualizará cada elemento

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

```
items = [  
  { desc: 'Elem1', check: true },  
  { desc: 'Elem2', check: true },  
  { desc: 'Elem3', check: false }  
]
```



# Templates

ejem5

- **Directivas**

- Las **directivas** modifican a los elementos en los que se incluyen
- Existen muchas **directivas predefinidas** y podemos programar nuestras propias directivas
- Las directivas **estructurales** empiezan por \* y **modifican el DOM** del documento (\*ngIf, \*ngFor, \*ngSwitch)
- El \* es **azúcar sintáctico** para la definición del template

# Templates

ejem5

- ## Directivas

- 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>
```

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

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

# Templates

ejem5

- ***Safe Navigation Operator***

- Si el atributo **user** tiene valor undefined, se produce un error y no se muestra el componente

```
User's name: {{user.name}}
```

- Para evitarlo existe el *safe navigator operator*. Undefined se representa como cadena vacía

```
User's name: {{user?.name}}
```



# Ejercicio 1

- Implementa una aplicación con un **campo de texto** y un **botón de Añadir**
- Cada vez que se pulse el **botón**, el **contenido** del campo de texto se **añadirá al documento**

## Estilos CSS

# Estilos CSS

- Existen varias formas de definir un CSS en Angular 2
  - Globalmente asociado al index.html
  - Local al componente:
    - En la propiedad styles o styleUrls de @Component
    - En el template

# Estilos CSS

ejem5

- **Definir CSS en Angular**
  - Globalmente asociado al index.html
    - Si creamos un fichero **src/styles.css** se incluirá de forma automática en el **index.html**
    - Podemos añadir más ficheros .css a la carpeta assets o descargados de NPM. Hay que añadir esos ficheros en el fichero **angular.json**, entrada "styles"

<https://github.com/angular/angular-cli#global-styles>

# Estilos CSS

ejem5

- **Definir CSS en Angular**
  - Globalmente asociado al index.html
    - Para añadir Bootstrap CSS
    - Instalamos la dependencia NPM
      - Añadimos la entrada en el package.json

```
npm install --save bootstrap
```

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

# Estilos CSS

ejem5

- **Definir CSS en Angular**

- En la propiedad styles o styleUrls de @Component

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styles: [`  
    .red { color: red; }  
    .blue { color: blue; }  
  `]  
})  
export class AppComponent {  
  ...  
}
```

Se suelen usar los strings multilínea con tildes invertidas

# Estilos CSS

ejem5

- **Definir CSS en Angular**
  - En la propiedad styles o styleUrls de @Component

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



Con angular-cli no se puede usar styles y styleUrls a la misma vez en un component por las optimizaciones

# Estilos CSS

ejem5

- Definir CSS en Angular
  - En el template



```
<style>
  .orange {
    color: orange;
  }
</style>

<h1 [class]="className">Hello {{name}}!</h1>

<button (click)="setClass('blue')">Blue</button>
...
```

- Controlar la clase o estilo de un elemento
  - Hay muchas formas de controlar los estilos de los elementos
    - 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 mapa de string a boolean
    - Asociar un **estilo concreto** de un elemento a un atributo

ejem5

- Asociar la clase de un elemento a un atributo string
  - Cambiando el valor del atributo se cambia la clase del elemento
  - Por ejemplo, la clase del elemento h1 se cambia modificando el atributo **className** del componente

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

# Estilos CSS

ejem5

- Activar una clase concreta con un atributo boolean
  - Activa o desactiva una clase red con el valor del atributo booleano **redActive**

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

- Se puede usar para varias clases

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

ejem5

- Asociar la clase de un elemento a un mapa
  - Para gestionar varias clases es mejor usar un mapa de string (nombre de la clase) a boolean (activa o no)



```
<p [ngClass]="pClasses">Text</p>
```

```
pClasses = {  
  "red": false,  
  "bold": true  
}
```

```
changeParagraph(){  
  this.pClasses.bold = true;  
}
```

# Estilos CSS

ejem5

- **Asociar un estilo concreto a un atributo**
  - En algunos casos es mejor cambiar el estilo directamente en el elemento

```
<p [style.backgroundColor]="pColor">Text</p>
```

- Con unidades

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

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

ejem5

- Asociar un estilo concreto a un atributo
  - Usando mapas de propiedad a valor

```
<p [ngStyle]="getStyles()">Text</p>
```

```
getStyles(){
  return {
    'font-style':this.canSave? 'italic':'normal',
    'font-weight':!this.isUnchanged? 'bold':'normal',
    'font-size':this.isSpecial? '24px':'8px',
  }
}
```

# Formularios

# Formularios

- Existen diversas formas de controlar formularios en Angular
  - Vincular un control del formulario a un atributo del componente (*data binding*)
  - Acceso a los controles desde el **código** para leer y modificar su estado
  - Mecanismos **avanzados** con validación con **ngControl** (no los veremos en el curso)

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

# Formularios

ejem6

- ***Data binding en campo de texto***

- Se vincula el control a un atributo del componente con `[ (ngModel) ]`
- Cualquier cambio en el control se refleja en el valor del atributo (y viceversa)

```
<input type="text" [(ngModel)]="name">
<p>{{name}}</p>
```

name: string

# Formularios

## **ejem6**

- *Data binding* en checkbox (boolean)

- Cada control se asocia con [ (ngModel) ] a un atributo booleano y su valor depende de si está “checked”

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



angular:boolean  
javascript:boolean

# Formularios

ejem6

- *Data binding en checkbox (objetos)*

- Cada control se asocia con `[ (ngModel) ]` a un atributo booleano de un objeto de un array

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

```
items = [
  {value:'Item1', selected:false},
  {value:'Item2', selected:false}
]
```

ejem6

- *Data binding* en botones de radio

- Todos los botones del mismo grupo se asocian al mismo atributo con [(ngModel)]
- El valor del atributo es el “value” del control

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

gender string

ejem6

- **Acceso a los controles desde el código**

- *Template reference variables*
- Un elemento del template puede asociarse a una variable
- Podemos usar esa variable en el código del template para manejar ese elemento

```
<input #cityInput type="text">
```

```
<button (click)="update(cityInput.value); cityInput.value=''">  
    Update city  
</button>
```

Este control input puede referenciarse con el nombre **cityInput** en el template

Podemos usar la propiedades y métodos del elemento definidos en su API DOM

# Formularios

ejem6

- **Acceso a los controles desde el código**
  - También podemos acceder al elemento desde el código del componente
  - Creamos un atributo en el componente de tipo **ElementRef**
  - Anotamos ese atributo con **@ViewChild('refName')**
  - El atributo apuntará al elemento con ese **refName**

# Formularios

ejem6

- **Acceso a los controles desde el código**

```
<input #titleInput type="text">
<button (click)="updateTitle()">Update Title</button>
<p>Title: {{title}}</p>
```

```
export class AppComponent {

    @ViewChild('titleInput') titleInput: ElementRef;
    title: string;

    updateTitle() {
        this.title = this.titleInput.nativeElement.value;
        this.titleInput.nativeElement.value = '';
    }
}
```

ejem6

- **Acceso a los controles desde el código**

```
<input #titleInput type="text">
<button (click)="updateTitle()">Update Title</button>
<p>Title: {{title}}</p>
```

```
export class AppComponent {
    @ViewChild('titleInput') titleInput: ElementRef;
    title: string;

    updateTitle() {
        this.title = this.titleInput.nativeElement.value;
        this.titleInput.nativeElement.value = '';
    }
}
```

# Formularios

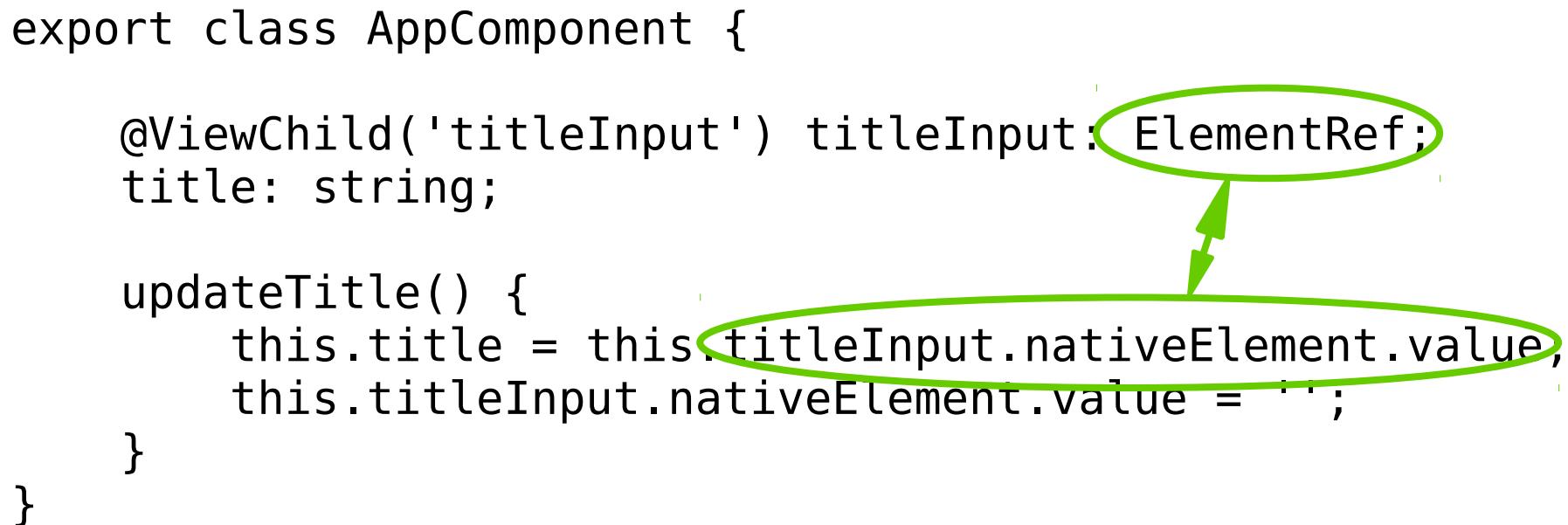
ejem6

- Acceso a los controles desde el código

```
<input #titleInput type="text">
<button (click)="updateTitle()">Update Title</button>
<p>Title: {{title}}</p>
```

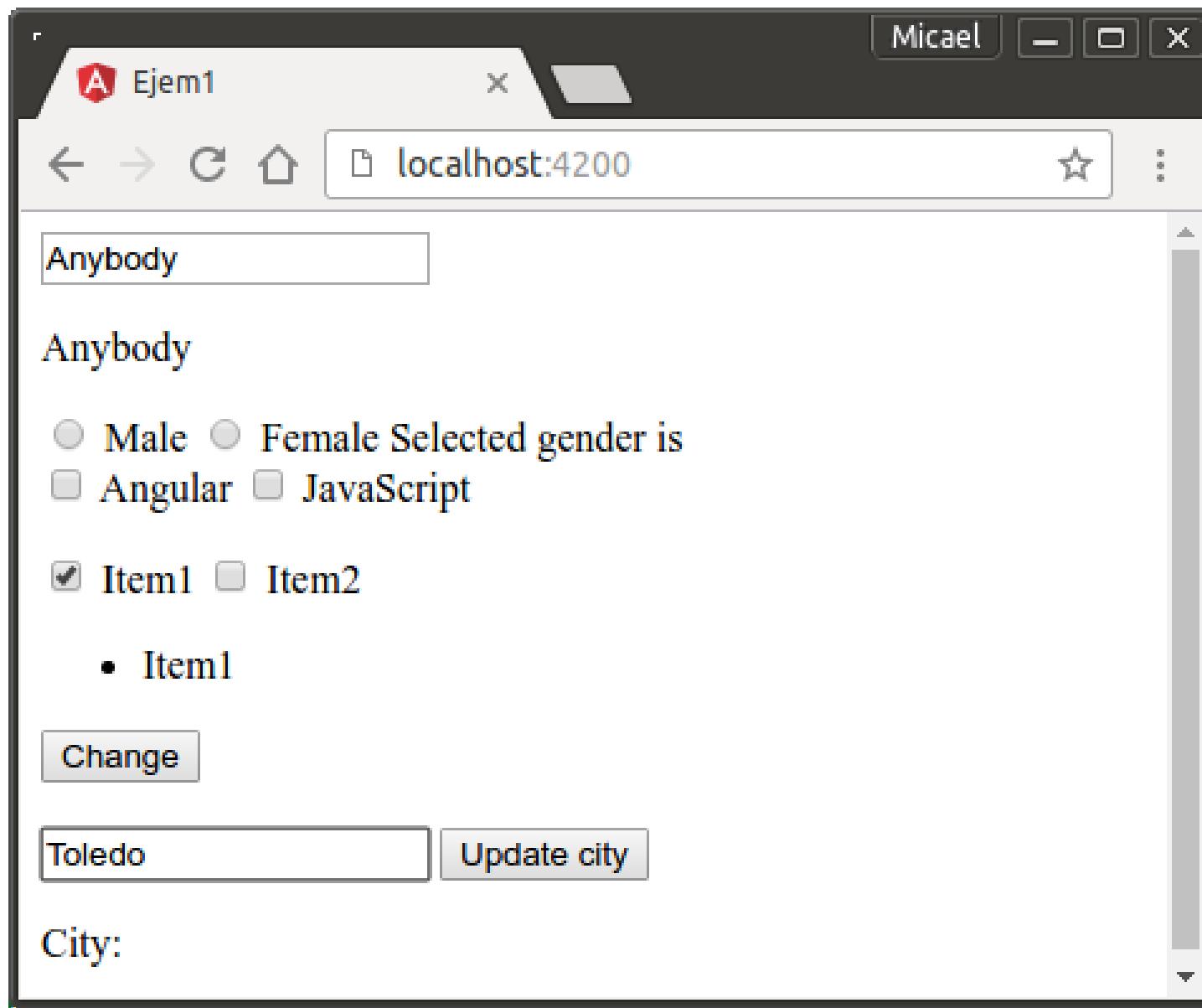
```
export class AppComponent {
    @ViewChild('titleInput') titleInput: ElementRef;
    title: string;

    updateTitle() {
        this.title = this.titleInput.nativeElement.value,
        this.titleInput.nativeElement.value = '';
    }
}
```



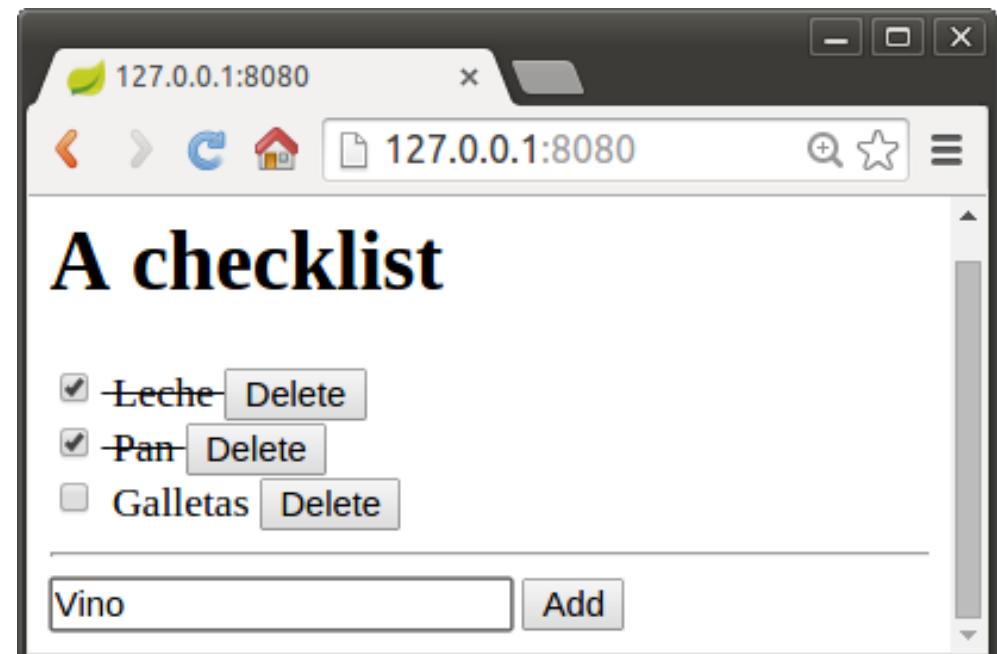
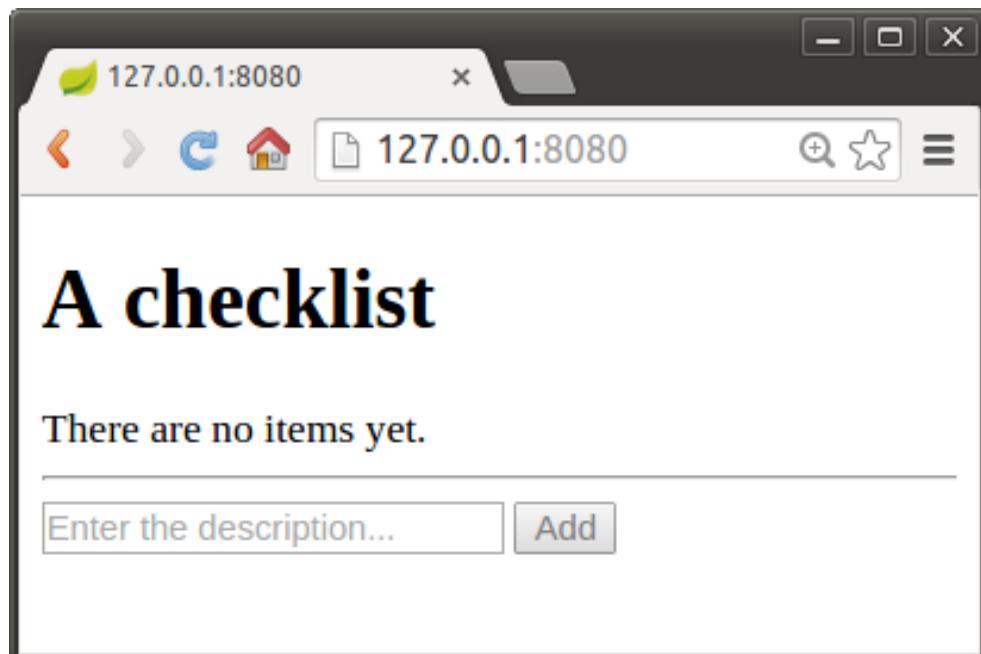
# Formularios

ejem6



# Ejercicio 2

- Implementa una aplicación de gestión de tareas
- Las tareas se mantendrán en memoria

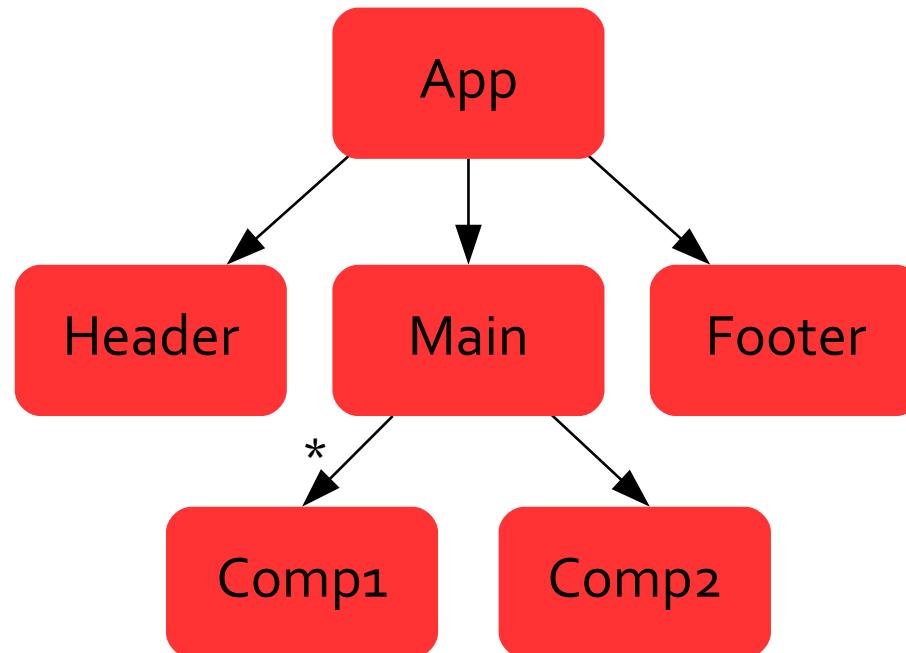


# Composición de componentes

# Composición de componentes

## Árboles de componentes

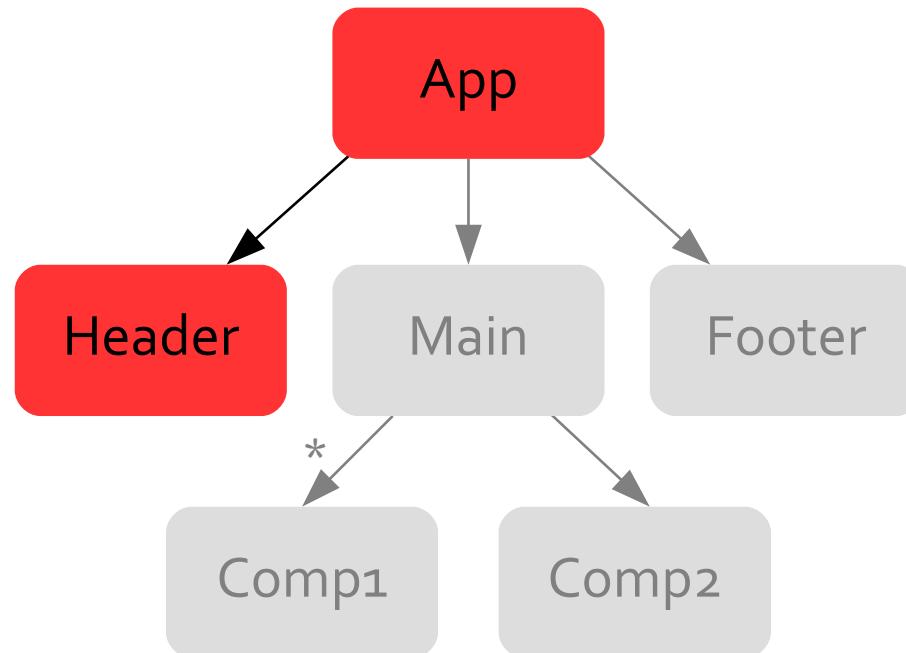
En Angular 2 un componente puede estar formado por más componentes formando un árbol



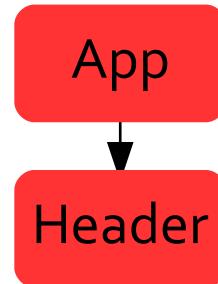
# Composición de componentes

## Árboles de componentes

En Angular 2 un componente puede estar formado por más componentes formando un árbol

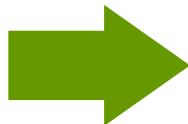


# Composición de componentes



## Árboles de componentes

```
<h1>Title</h1>
<p>Main content</p>
```

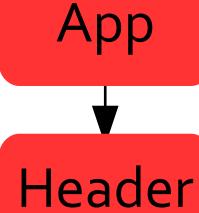


```
<header></header>
<p>Main content</p>
```

```
<header>
```

```
<h1>Title</h1>
```

# Composición de componentes



## Árboles de componentes

ejem7

app.component.ts

```

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

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

app.component.html

```

<header></header>
<p>Main content</p> 
```

header.component.ts

```

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

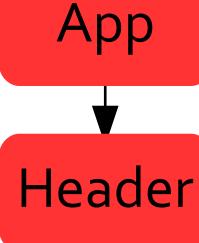
@Component({
  selector: 'header',
  templateUrl:
    './header.component.html'
})
export class HeaderComponent {} 
```

header.component.html

```

<h1>Title</h1> 
```

# Composición de componentes



app.component.ts

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

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

app.component.html

```
<header></header>
<p>Main content</p>
```

Para incluir un componente se usa su **selector**

ejem7

## Árboles de componentes

header.component.ts

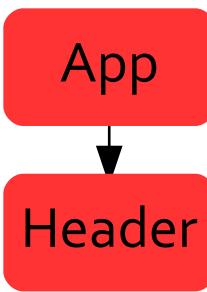
```
import {Component} from
  '@angular/core';

@Component({
  selector: 'header',
  templateUrl:
    './header.component.html'
})
export class HeaderComponent {}
```

header.component.html

```
<h1>Title</h1>
```

# Composición de componentes



app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { HeaderComponent } from "./header.component";

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

ejem7

## Árboles de componentes

Todos los componentes de la app deben declararse en el **@NgModule**

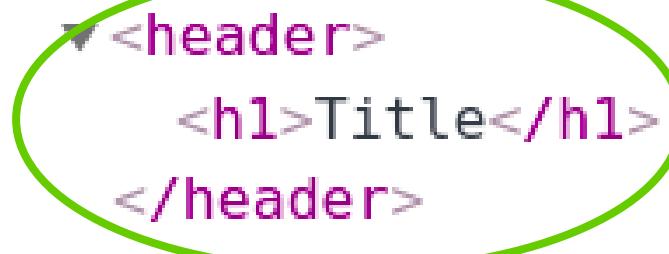
# Composición de componentes

## Árboles de componentes

ejem7

- Al cargar la app en el navegador, en el árbol **DOM** cada componente incluye en su **elemento** el contenido de la **vista** (HTML)

```
▼<app-root ng-version="9.0.7">
  ▼<header>
    <h1>Title</h1>
    </header>
    <p>Main content</p>
  </app-root>
```



# Composición de componentes

- Comunicación entre un **componente padre** y un **componente hijo**
  - 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)

# Composición de componentes

ejem8

## Configuración de propiedades (Padre → Hijo)

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

El título de <header> será el valor del atributo appTitle

```
<header [title]='appTitle'></header>
<p>Main content</p>
```

# Composición de componentes

ejem8

## Configuración de propiedades (Padre → Hijo)

app.component.ts

```
...
export class AppComponent {
  appTitle = 'Main Title';
}
```

header.component.ts

```
import {Component, Input} from
  '@angular/core';
...
export class HeaderComponent {
  @Input()
  title: string;
}
```

app.component.html

```
<header [title]='appTitle'></header>
<p>Main content</p>
```

header.component.html

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

# Composición de componentes

ejem8

## Configuración de propiedades (Padre → Hijo)

app.component.ts

```
...
export class AppComponent {
  appTitle = 'Main Title';
}
```

app.component.html

```
<header [title]='appTitle'></header>
<p>Main content</p>
```

header.component.ts

```
import {Component, Input} from
  '@angular/core';
...
export class HeaderComponent {
  @Input()
  title: string;
}
```

header.component.html

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

# Composición de componentes

ejem8

## Configuración de propiedades (Padre → Hijo)

app.component.ts

```
...
export class AppComponent {
  appTitle = 'Main Title';
}
```

header.component.ts

```
import {Component, Input} from
  '@angular/core';
...
export class HeaderComponent {
  @Input()
  title: string;
}
```

app.component.html

```
<header [title]='appTitle'></header>
<p>Main content</p>
```

header.component.html

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

# Composición de componentes

ejem9

## Envío de eventos (Hijo → Padre)

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

La variable \$event apunta al evento generado

```
<header (hidden)='hiddenTitle($event)'></header>
<p>Main content</p>
```

# Composición de componentes

ejem9

## Envío de eventos (Hijo → Padre)

app.component.ts

```
...
export class AppComponent {
  hiddenTitle(hidden: boolean){
    console.log("Hidden:"+hidden)
  }
}
```

app.component.html

```
<header (hidden)='hiddenTitle($event)'></header>
<p>Main content</p>
```

# Composición de componentes

ejem9

## Envío de eventos (Hijo → Padre)

```
...
export class AppComponent {
  hiddenTitle(hidden: boolean){
    console.log("Hidden:"+hidden)
  }
}
```

```
<header (hidden)='hiddenTitle($event)'></header>
<p>Main content</p>
```

# Composición de componentes

ejem9

## Envío de eventos (Hijo → Padre)

```
...
export class AppComponent {
  hiddenTitle(hidden: boolean){
    console.log("Hidden:" +hidden)
  }
}
```

Los eventos pueden tener valores que se capturan con \$event

```
<header (hidden)='hiddenTitle($event)'></header>
<p>Main content</p>
```

# Composición de componentes

ejem9

## Envío de eventos (Hijo → Padre)

header.component.ts

```
import {Component, Output, EventEmitter} from '@angular/core';
...
export class HeaderComponent {

    @Output()
    hidden = new EventEmitter<boolean>();

    visible = true;

    click(){
        this.visible = !this.visible;
        this.hidden.next(this.visible);
    }
}
```

header.component.html

```
<h1 *ngIf="visible">Title</h1>
<button (click)='click()'>Hide/Show</button>
```

# Composición de componentes

ejem9

## Envío de eventos (Hijo → Padre)

header.component.ts

```
import {Component, Output, EventEmitter} from '@angular/core';
...
export class HeaderComponent {
    @Output()
    hidden = new EventEmitter<boolean>();
    visible = true;
    click(){
        this.visible = !this.visible;
        this.hidden.emit(this.visible);
    }
}
```

Se declara un atributo de tipo **EventEmitter** con la anotación **@Output**

Para **lanzar un evento** se invoca el método **emit(valor)**

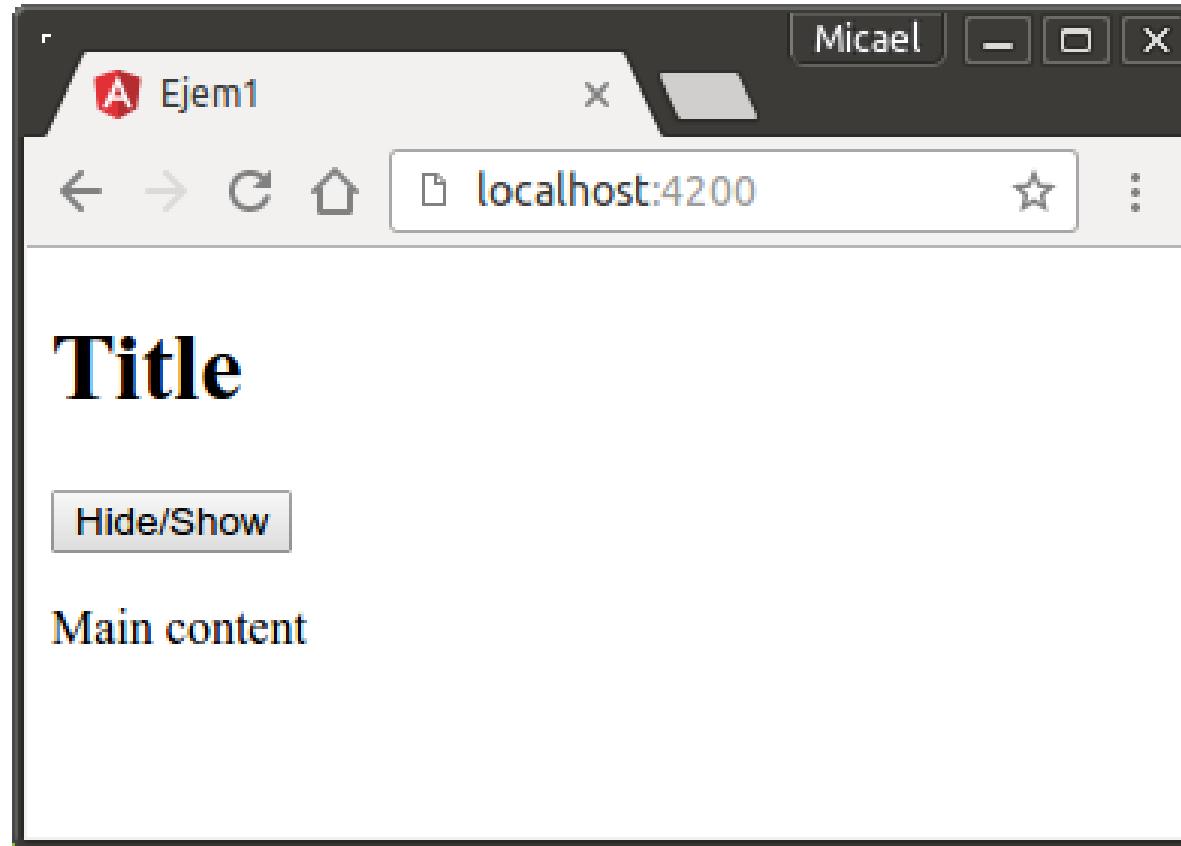
header.component.html

```
<h1 *ngIf="visible">Title</h1>
<button (click)='click()'>Hide/Show</button>
```

# Composición de componentes

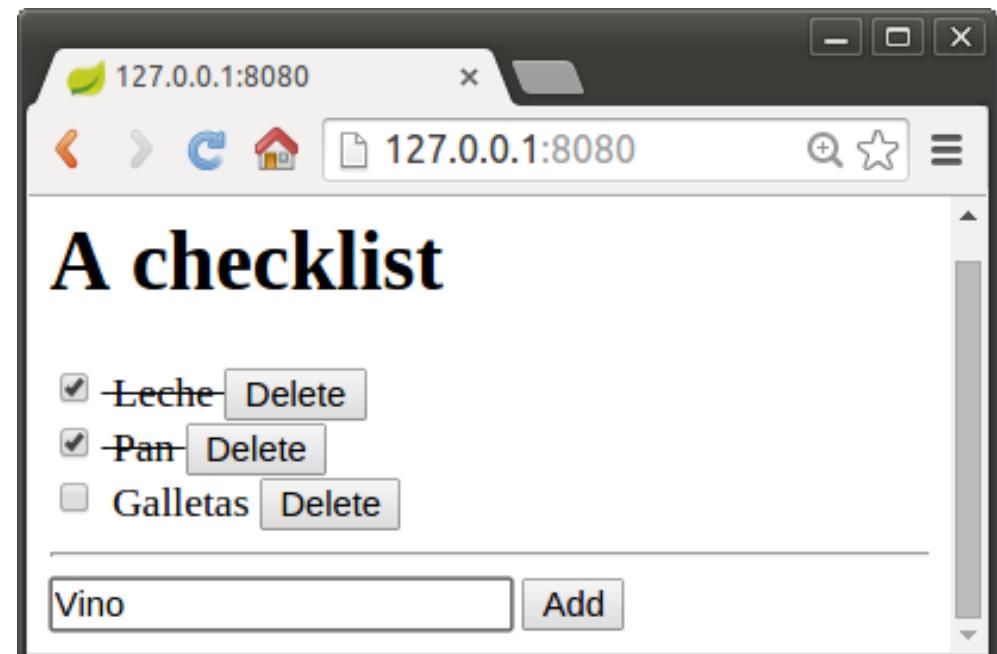
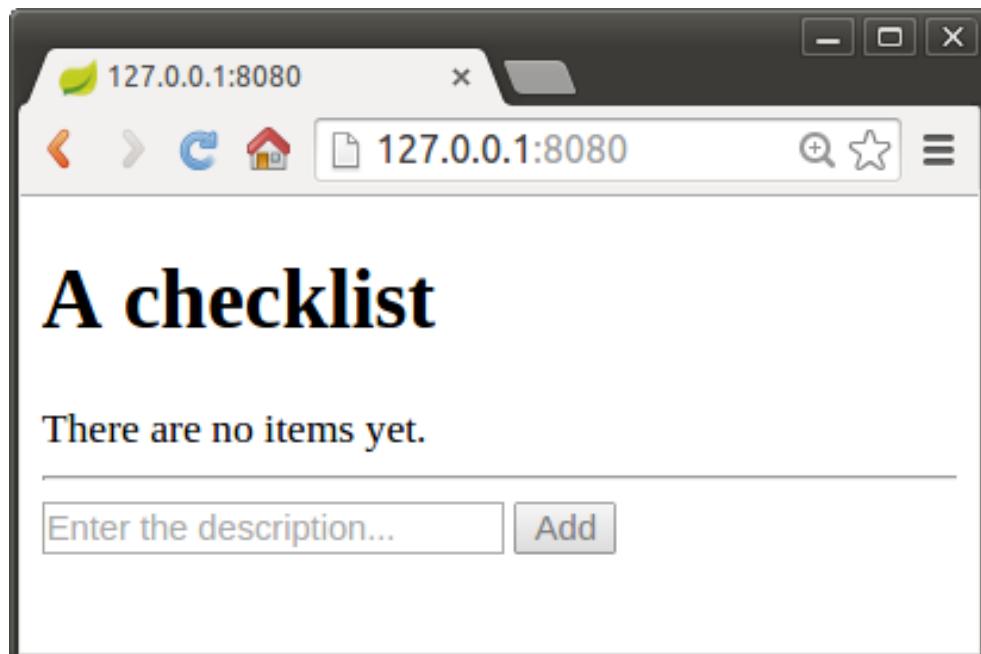
ejem9

## Envío de eventos (Hijo → Padre)



# Ejercicio 2

- Refactoriza la aplicación de gestión de tareas para que cada tarea sea un componente



# Composición de componentes

- **¿Cuándo crear un nuevo componente?**
  - El ejercicio y los ejemplos son **excesivamente sencillos** para que compense la creación de un nuevo componente **hijo**
  - En casos reales se crearían nuevos componentes:
    - Cuando la lógica y/o el *template* sean suficientemente **complejos**
    - Cuando los componentes hijos puedan **reutilizarse** en varios contextos

# Tema 3

# Rest y Servicios

Micael Gallego  
@micael\_gallego

# Cliente REST e inyección de dependencias

# Cliente REST e inyección de dependencias

- Angular 2 dispone de su propio **cliente de API REST**
- Es un objeto de la clase **Http**



```
Http http = ...  
  
http.get(url).subscribe(  
  response => console.log(response.json()),  
  error => console.error(error)  
);
```

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

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

# Cliente REST e inyección de dependencias

- Angular 2 dispone de su propio **cliente de API REST**
- Es un objeto de la clase **Http**

```
Http http = ...  
  
http.get(url).subscribe(  
  response => console.log(response.json()),  
  error => console.error(error)  
);
```

El método **subscribe** recibe dos parámetros:  
**1)** La función que se ejecutará cuando la petición sea **correcta**  
**2)** La función que se ejecutará cuando la petición sea **errónea**

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

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

# Cliente REST e inyección de dependencias

- Angular 2 dispone de su propio **cliente de API REST**
- Es un objeto de la clase **Http**

```
Http http = ...  
  
http.get(url).subscribe(  
  response => console.log(response.json()),  
  error => console.error(error)  
)
```

Para obtener la respuesta del servidor usamos el método **json()** del objeto **response**

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

- ¿Cómo podemos acceder al objeto http?
  - ¿Creamos el objeto con new?
    - Podríamos crear un objeto nuevo cada vez que nos haga falta
    - Esta opción **dificulta hacer tests automáticos** unitarios porque necesitaríamos que el servidor REST estuviese disponible y sería más difícil probar diferentes situaciones

- **¿Cómo podemos acceder al objeto http?**
  - Pidiendo al framework que proporcione el objeto
    - Cuando la **aplicación** se ejecute, el objeto http sería un **objeto real** que hace peticiones al servidor REST
    - Cuando ejecutemos los **tests**, el objeto http puede ser un **sustituto (mock)** que se comporte como queramos en el test pero no haga peticiones reales

- **Inyección de dependencias**

- Las **dependencias** que un módulo necesita son **inyectadas** por el sistema
- Esta técnica de solicitar dependencias que sean inyectadas por el **framework** se denomina **inyección de dependencias**
- Esta técnica se ha hecho muy popular en el desarrollo de *back-end* en frameworks como **Spring** o **Java EE**

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

# Cliente REST e inyección de dependencias

- Para usar un objeto **http** tenemos que usar la **inyección de dependencias**

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

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

  constructor(private httpClient: HttpClient) { }

  search(title: string) {
    ....
  }
}
```

# Cliente REST e inyección de dependencias

- Para usar un objeto **http** tenemos que usar la **inyección de dependencias**

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

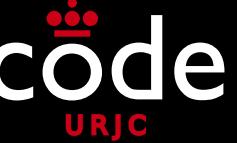
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  constructor(private httpClient: HttpClient) { }

  search(title: string) {
    ....
  }
}
```

Definimos un **parámetro Http** en el **constructor** de un componente

Cuando Angular construya el componente, **inyectará** el objeto http solicitado

# Cliente REST e inyección de dependencias



- Para usar un objeto **http** tenemos que usar la **inyección de dependencias**

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

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

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

# Cliente REST e inyección de dependencias

- Para usar un objeto **http** tenemos que usar la **inyección de dependencias**

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

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

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

Ponemos una referencia al módulo necesario para peticiones REST:  
**HttpClientModule**

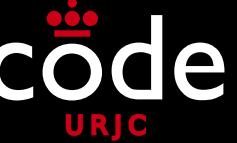
# Cliente REST e inyección de dependencias

ejem10

- Ejemplo de **buscador libros** en Google Books



# Cliente REST e inyección de dependencias



ejem10

- Ejemplo de **buscador libros** en Google Books

app.component.html

```
<h1>Google 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.ts

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

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

  private books: string[] = [];

  constructor(private http: Http) { }

  search(title: string) {

    this.books = [];
    let url = "https://www.googleapis.com/books/v1/volumes?q=intitle:"+title;
    this.http.get(url).subscribe(
      response => {
        let data: any = response;
        for (var i = 0; i < data.items.length; i++) {
          let bookTitle = data.items[i].volumeInfo.title;
          this.books.push(bookTitle);
        }
      },
      error => console.error(error)
    );
  }
}
```

**ejem10**

## app.component.ts

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

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

  private books: string[] = [];

  constructor(private http: Http) { }

  search(title: string) {

    this.books = [];
    let url = "https://www.googleapis.com/books/v1/volumes?q=intitle:"+title;
    this.http.get(url).subscribe(
      response => {
        let data: any = response;
        for (var i = 0; i < data.items.length; i++) {
          let bookTitle = data.items[i].volumeInfo.title;
          this.books.push(bookTitle);
        }
      },
      error => console.error(error)
    );
  }
}
```

ejem10

# Cliente REST e inyección de dependencias

- Peticiones POST

```
let data = { ... }
this.http.post(url, data).subscribe(
  response => console.log(response),
  error => console.error(error)
);
```

- Peticiones PUT

```
let data = { ... }
this.http.put(url, data).subscribe(
  response => console.log(response),
  error => console.error(error)
);
```

- Peticiones POST

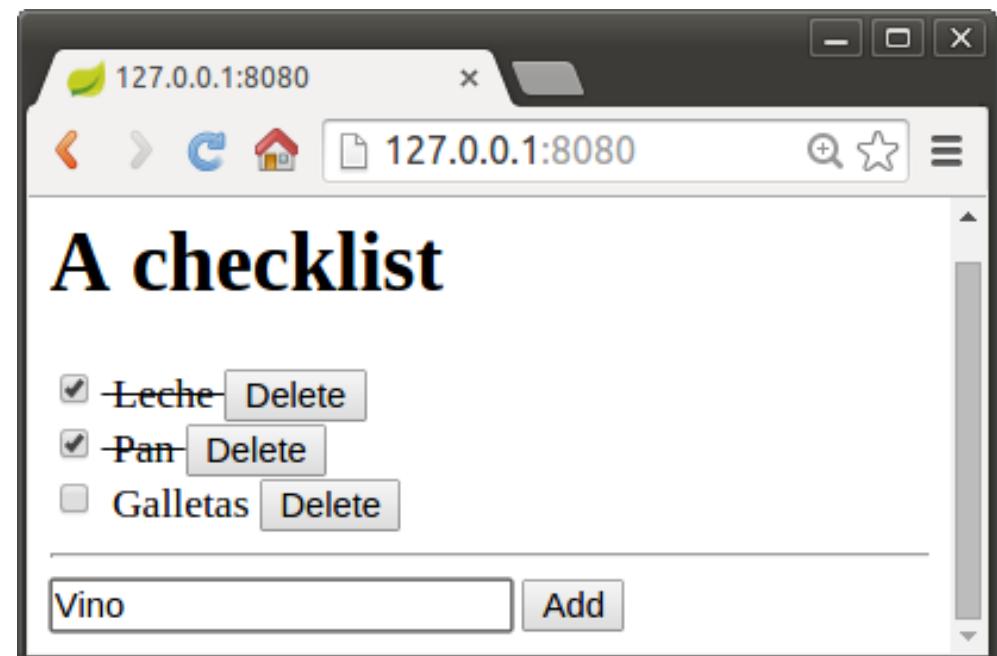
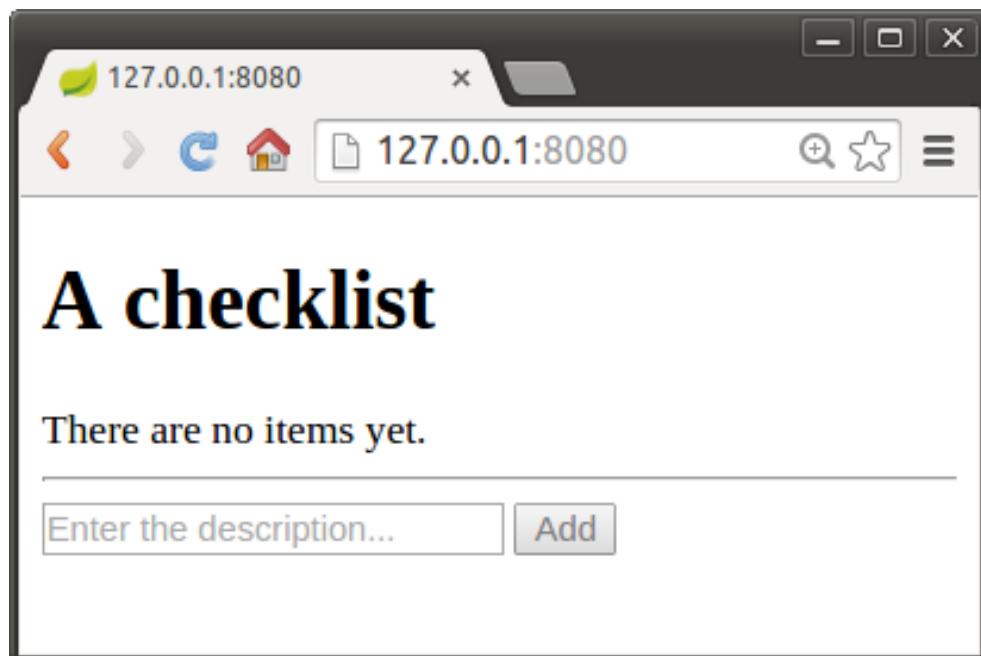
```
let data = { ... }
this.http.post(url, data).subscribe(
  response => console.log(response),
  error => console.error(error)
);
```

- Peticiones PUT

```
let data = { ... }
this.http.put(url, data).subscribe(
  response => console.log(response),
  error => console.error(error)
);
```

# Ejercicio 4

- Amplía el **servicio de gestión de items** para que utilice una **API REST** para gestionar los items



# Ejercicio 4

- Para ello se usará una aplicación web para el servidor (**backend**) que ofrece una API REST
- Es necesario **Java 8** para que se pueda ejecutar
- Se distribuye como un fichero **.jar**
- Ejecución:

```
java -jar items-backend.jar
```

- La API REST se podrá usar cuando aparece

```
Tomcat started on port(s): 8080 (http)
Started Application in 6.766 seconds (JVM running for 7.315)
```

# Ejercicio 3

- API REST Items

- Creación de items
  - Method: POST
  - URL: `http://127.0.0.1:8080/items/`
  - Headers: Content-Type: application/json
  - Body:

```
{ "description" : "Leche", "checked": false }
```

- Result:

```
{ "id": 1, "description" : "Leche", "checked": false }
```

- Status code: 201 (Created)

# Ejercicio 3

- API REST Items

- Consulta de items
  - Method: GET
  - URL: `http://127.0.0.1:8080/items/`
  - Result:

```
[  
  { "id": 1, "description": "Leche", "checked": false },  
  { "id": 2, "description": "Pan", "checked": true }  
]
```

- Status code: 200 (OK)

# Ejercicio 3

- API REST Items

- Modificación de items
  - Method: PUT
  - URL: `http://127.0.0.1:8080/items/1`
  - Headers: Content-Type: application/json
  - Body:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Status code: 200 (OK) / 404 (Not Found)

# Ejercicio 3

- API REST Items

- Modificación de items
  - Method: DELETE
  - URL: `http://127.0.0.1:8080/items/1`
  - Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Status code: 200 (OK) / 404 (Not Found)

# Servicios

# Servicios

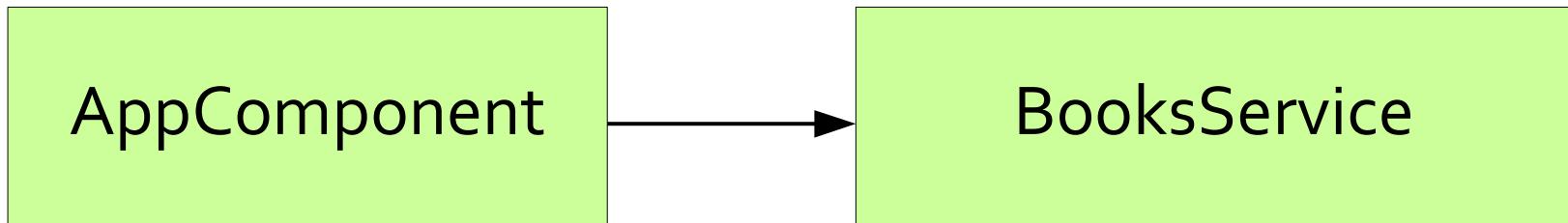
- Acoplar en el componente la lógica de las peticiones http **no es una buena práctica**
- El componente podría llegar a ser muy **complejo** y **difícil de ampliar / modificar**
- Es mucho más **difícil implementar tests unitarios** si el componente tiene muchas responsabilidades
- Es mucho mejor **modularizar** la aplicación en **elementos** que tengan una única **responsabilidad**
  - Componente: Interfaz de usuario
  - Otro elemento: Peticiones http

# Servicios

- A los **elementos** de la aplicación que no se encargan del interfaz de usuario se les conoce como **servicios**
- Angular 2 ofrece muchos **servicios predefinidos**
- El objeto **http** se considera un servicio de acceso a APIs REST, pero existen más
- El desarrollador puede **implementar** sus propios **servicios** en la aplicación
- Para que sea más sencillo implementar tests, los servicios se **inyectan** en los **componentes**

# Servicios

- **¿Cómo se implementa un servicio?**
  - Se crea una nueva **clase** para el servicio
  - Se anota esa 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



# Servicios

ejem11

## ¿Cómo se implementa un servicio?

Ejemplo de buscador de libros con información en memoria



ejem11

## ¿Cómo se implementa un servicio?

books.service.ts

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

@Injectable({ providedIn: 'root' })
export class BooksService {

    getBooks(title: string){
        return [
            'Aprende Java en 2 días',
            'Java para torpes',
            'Java para expertos'
        ];
    }
}
```

ejem11

## ¿Cómo se implementa un servicio?

books.service.ts

```
Import { Injectable } from '@angular/core';  
  
@Injectable({ providedIn: 'root' })  
export class BooksService {  
  
    getBooks(title: string){  
        return [  
            'Aprende Java en 2 días',  
            'Java para torpes',  
            'Java para expertos'  
        ];  
    }  
}
```

# Servicios

ejem11

## ¿Cómo se implementa un servicio?

app.component.ts

```
import { Component } from '@angular/core';
import { BooksService } from './books.service';

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

  books: string[] = [];

  constructor(private booksService: BooksService){}

  search(title: string){
    this.books = this.booksService.getBooks(title);
  }
}
```

# Servicios

ejem11

## ¿Cómo se implementa un servicio?

app.component.ts

```
import { Component } from '@angular/core';
import { BooksService } from './books.service';

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

  books: string[] = [];

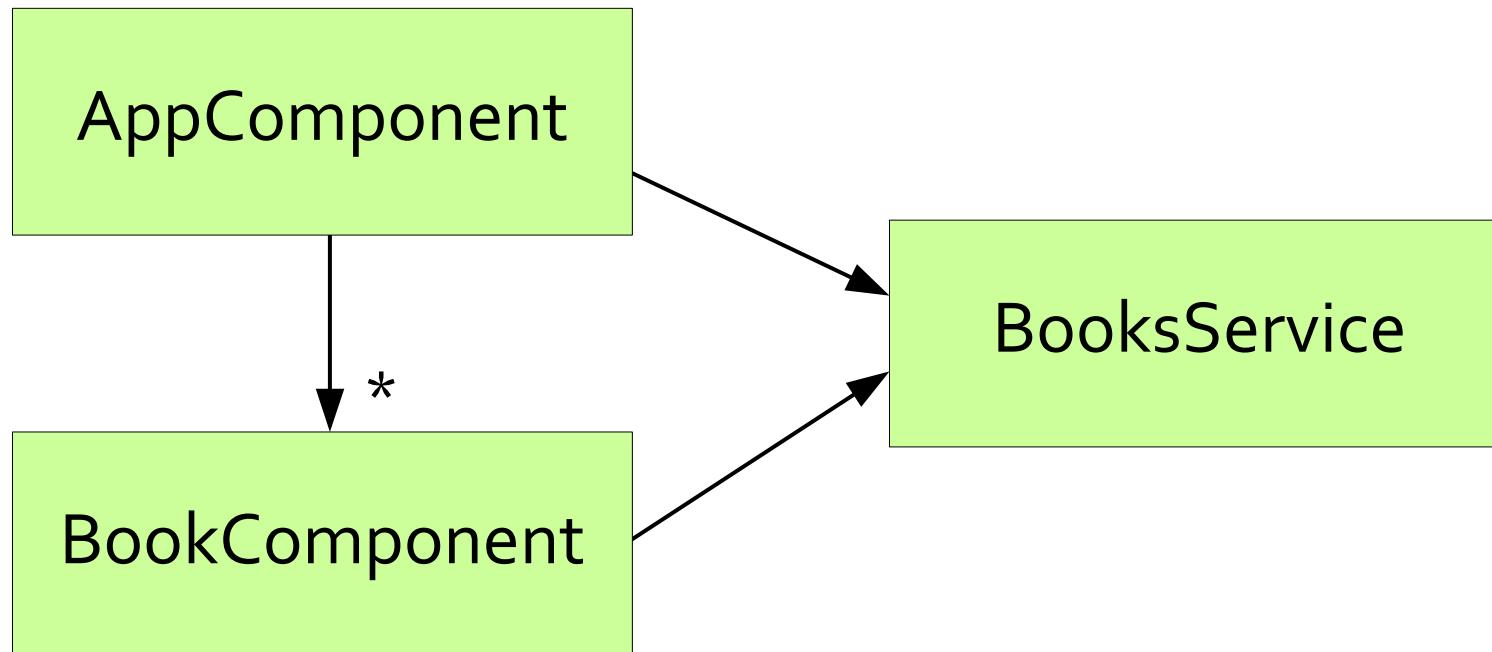
  constructor(private booksService: BooksService){}

  search(title: string){
    this.books = this.booksService.getBooks(title);
  }
}
```

## Compartir servicios entre componentes

- Es habitual que haya un **único objeto** de cada **servicio** en la aplicación (*singleton*)
- Es decir, todos los componentes **comparten** “el mismo” servicio
- De esa forma los **servicios** mantienen el **estado** de la aplicación y los **componentes** ofrecen el **interfaz** de usuario

## Compartir servicios entre componentes



## Servicio exclusivo para componente

- Se puede hacer que servicio **no sea compartido** entre todos los componentes de la aplicación (**no sea singleton**)
- Se puede crear un **servicio exclusivo** para un **componente y sus hijos**
- Se declara en el atributo **providers** del **@Component** y se quita del **@Injectable** el **providedIn**
- Puede ser compartido por el componente (**padre**) y por sus componentes **hijos** (incluidos en él)

## Servicio exclusivo para componente

app.component.ts

```
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) {}

  ...
}
```

## Peticiones http en un servicio

- No es buena práctica hacer peticiones http desde un componente
- Es mejor **encapsular** el acceso al backend con API REST en un **servicio**
- **Ventajas**
  - Varios componentes pueden acceder al mismo backend **compartiendo** el mismo servicio (*singleton*)
  - Es más fácil de **testear**
  - Es una **buenas prácticas** (más fácil de **entender** por otros desarrolladores)

## Peticiones http en un servicio

- ¿Cómo se **implementan** los métodos de ese servicio?
  - No pueden devolver información de forma **inmediata**
  - Sólo pueden devolver información cuando llega la **respuesta del servidor**
  - En JavaScript los métodos **no se pueden bloquear** esperando la respuesta
  - Son **asíncronos / reactivos**

# Servicios

## Peticiones http en un servicio

- ¿Cómo se implementan los métodos de ese servicio?

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



Un servicio que hace peticiones a una API REST **NO PUEDE** implementarse de forma **síncrona (bloqueante)** en JavaScript

## Peticiones http en un servicio

- Existen principalmente 3 formas de implementar un servicio con **operaciones asíncronas** en JavaScript
  - Callbacks
  - Promesas
  - Observables

## Peticiones http en un servicio

- **Callbacks:** Se pasa como parámetro una función (de *callback*) que será ejecutada cuando llegue el resultado. Esta función recibe como primer parámetro el **error** (si ha habido)

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

# Servicios

## Peticiones http en un servicio

- **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

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

## Peticiones http en un servicio

- **Observables:** Similares a las promesas pero con más funcionalidad. 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)  
);
```

## Peticiones http en un servicio

- Implementación de métodos asíncronos
  - **Callbacks:** Hay muchas librerías implementadas así. Ya no se recomienda este enfoque porque es más limitado
  - **Promesas:** La forma estándar en ES6. La forma recomendada si la funcionalidad es suficiente
  - **Observables:** Implementados en la librería RxJS. Es la forma recomendada por Angular por ser la más completa (aunque más compleja)

## Servicio con Observables de RxJS

- RxJS: Extensiones **reactivas** para JavaScript
- La librería RxJS está incluida en Angular
- Es mucho más **potente** que las **promesas** (estándar de ES6)
- Nos vamos a centrar únicamente en los aspectos que nos permitan implementar **servicios** con llamadas a una **API REST**



<https://github.com/ReactiveX/RxJS>

## Servicio con Observables de RxJS ejem12

- Tenemos que ofrecer **objetos de alto nivel** a los clientes del servicio (p.e. **array de titles**)
- Pero al hacer una petición REST con http obtenemos un objeto **Response**
- El objetivo es **transformar** el objeto Response en array de titles cuando llegue la respuesta

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

Objeto de  
alto nivel

## Servicio con Observables de RxJS ejem12

```
...
import { map } from 'rxjs/operators';
import { Observable } from 'rxjs';

@Injectable(...)
export class BooksService {
  ...
  getBooks(title: string): Observable<string[]> {
    let url = ...
    return this.httpClient.get(url).pipe(map(
      response => this.extractTitles(response as any))
    )
  }
  private extractTitles(response) { ... }
}

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

Con el método **map** se indica la **transformación** que hacemos a la respuesta para obtener el objeto de **alto nivel**

El cliente del servicio accede al **array de títulos** en vez de a la response

# Servicios

app.component.ts

```

import { Component } from '@angular/core';
import { BooksService } from './books.service';

@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).subscribe(
      books => this.books = books,
      error => console.error(error)
    );
  }
}

```

**ejem12**

Cuando llega la respuesta se actualiza el array de books



ejem13

## Servicio con Observables de RxJS

- Al igual que transformamos el resultado cuando la petición es correcta, también podemos **transformar el error** para que sea de más alto nivel
- Usamos el método **catch** para gestionar el error. Podemos **devolver un nuevo error** o simular una respuesta correcta (con un valor por defecto)

```
getBooks(title: string) {  
  let url = ...  
  return this.httpClient.get(url).pipe(  
    map(response => this.extractTitles(response as any)),  
    catchError(error => Observable.throw('Server error')));  
}
```

Lanzamos un  
nuevo error

## Estado en los servicios http

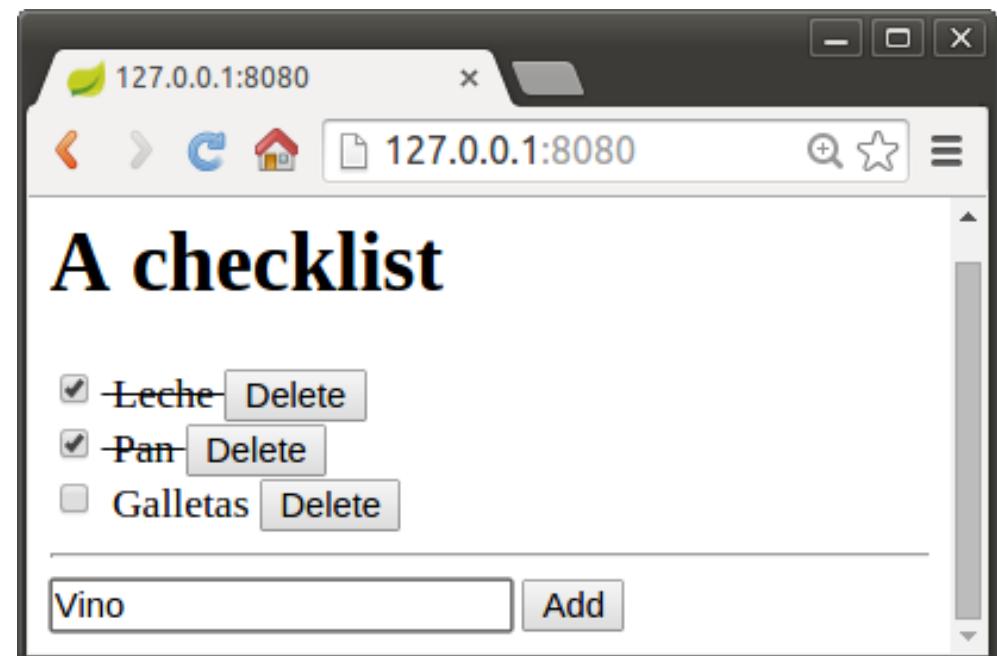
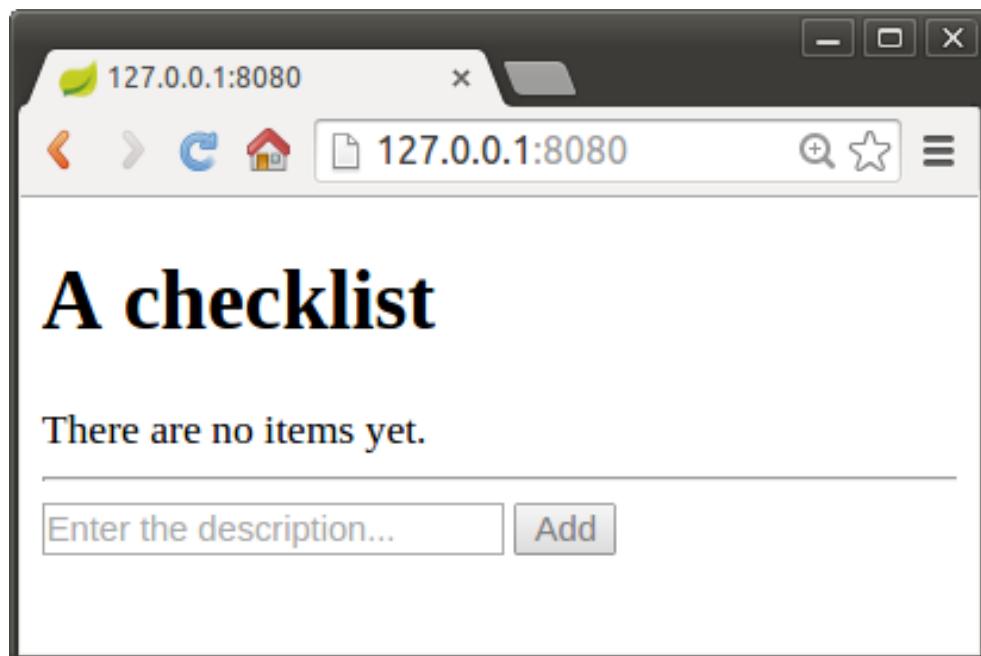
- **Servicios stateless (sin estado)**
  - No guardan información
  - Sus métodos devuelven valores, pero no cambian el estado del servicio
  - Ejemplo: **BooksService con llamadas a Google**
- **Servicios statefull (con estado)**
  - Mantienen estado, guardan información
  - Al ejecutar sus métodos cambian su estado interno, y también pueden devolver valores
  - Ejemplo: **BooksService con información en memoria**

## Estado en los servicios http

- ¿**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

# Ejercicio 4

- Refactoriza el **ejercicio anterior** para que las llamadas a la API REST estén en un servicio stateless **ItemsService**



Tema 4

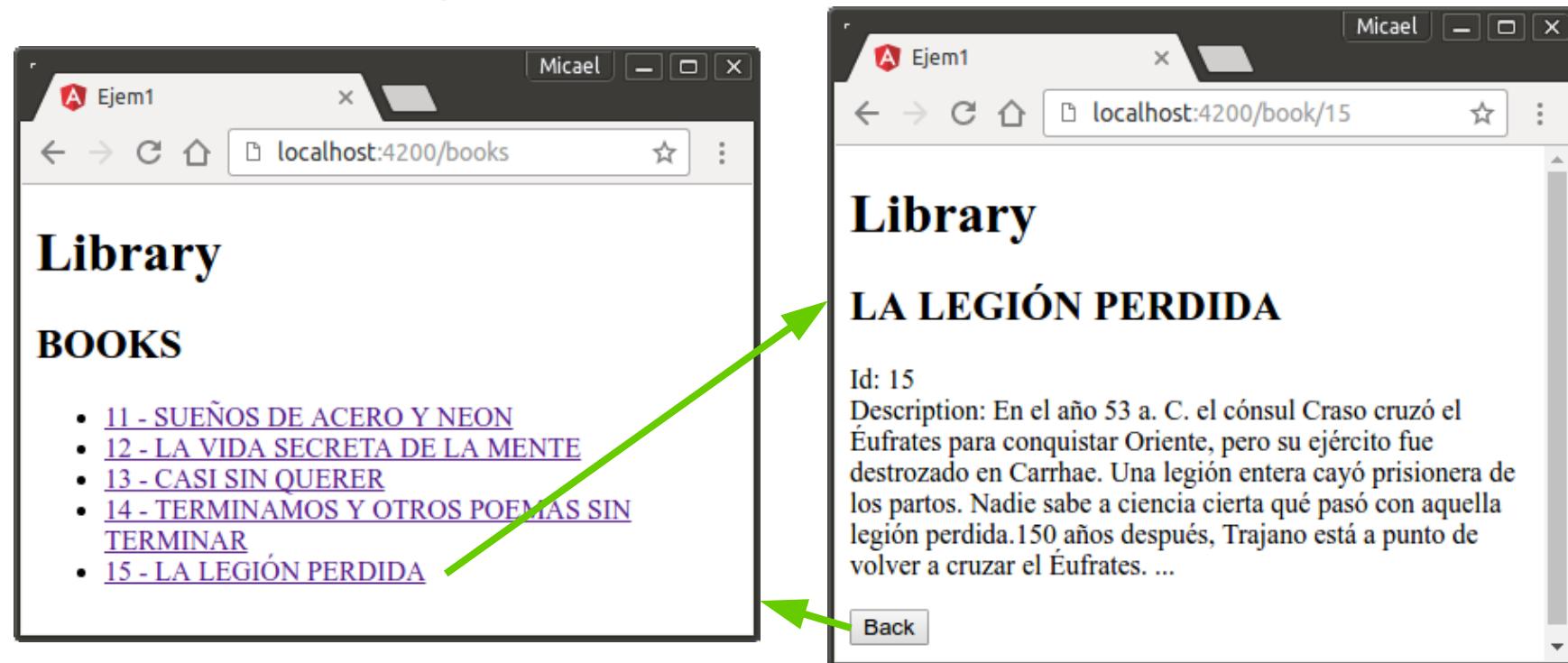
# Aplicaciones Multipágina - Router

Micael Gallego  
@micael\_gallego

# Aplicaciones Multipágina Router

# Aplicaciones multipágina: Router

- 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>

# Aplicaciones multipágina: Router

- El **componente principal** de la aplicación (`app-root`) tiene una parte fija (**cabecera, footer**) y una parte cuyo **contenido** depende de la **URL** (`<router-outlet>`)
- En `app.routing.ts` se define qué **componente** se muestra para cada **URL**
- Existen **links especiales** para navegar dentro de la aplicación web (`[ routerLink ]`)
- Desde el código se puede navegar de forma automática (**Router**)

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

ejem14

## Componente principal

app.component.ts

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

@Component({
  selector: 'app-root',
  template: `
    <h1 class="title">Library</h1>
    <router-outlet></router-outlet>
  `
})
export class AppComponent { }
```

ejem14

## Componente principal

app.component.ts

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

@Component({
  selector: 'app-root',
  template:
    `<h1 class="title">Library</h1>
     <router-outlet></router-outlet>
    `,
})
export class AppComponent { }
```

Zona que cambia en función de la URL

ejem14

## Configuración de las rutas

app.routing.ts

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

import { BookListComponent } from './book-list.component';
import { BookDetailComponent } from './book-detail.component';

const appRoutes = [
  { path: 'book/:id', component: BookDetailComponent, },
  { path: 'books', component: BookListComponent },
  { path: '', redirectTo: 'books', pathMatch: 'full' }
]

export const routing = RouterModule.forRoot(appRoutes);
```

ejem14

## Configuración de las rutas

app.routing.ts

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

import { BookListComponent } from './book-list.component';
import { BookDetailComponent } from './book-detail.component';

const appRoutes = [
  { path: 'book/:id', component: BookDetailComponent },
  { path: 'books', component: BookListComponent },
  { path: '', redirectTo: 'books', pathMatch: 'full' }
]

export const routes: Routes = appRoutes;
```

Para cada URL se indica el  
**componente** que será  
visualizado en el **router-outlet**

ejem14

## Configuración de las rutas

app.module.ts

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

@NgModule({
  declarations: [AppComponent,
    BookDetailComponent, BookListComponent],
  imports: [BrowserModule, FormsModule,
    HttpClientModule, routing],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

# Aplicaciones multipágina: Router

ejem14

## Configuración de las rutas

app.module.ts

```
...
import { routing } from './app.routing';
@NgModule({
  declarations: [AppComponent,
    BookDetailComponent, BookListComponent],
  imports: [BrowserModule, FormsModule,
    HttpClientModule, routing],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

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

# Aplicaciones multipágina: Router

book-list.component.ts

## BookListComponent

ejem14

```
...
@Component({
  template: `
    <h2>BOOKS</h2>
    <ul>
      <li *ngFor="let book of books">
        <a [routerLink]=["'/book',book.id]">
          {{book.id}}-{{book.title}}
        </a>
      </li>
    </ul>`})
export class BookListComponent {
  books: Book[];
  constructor(service: BookService) {
    this.books = service.getBooks();
  }
}
```

# Aplicaciones multipágina: Router

book-list.component.ts

## BookListComponent

ejem14

```
...
@Component({
  template:
    <h2>BOOKS</h2>
    <ul>
      <li *ngFor="let book of books">
        <a [routerLink]=["/book",book.id]>
          {{book.id}} - {{book.title}}
        </a>
      </li>
    </ul>
})
export class BookListComponent {
  books: Book[];
  constructor(service: BookService) {
    this.books = service.getBooks();
  }
}
```

En vez de **href**, los links usan **[routerLink]**. La URL se puede indicar como un string (**completa**) o como un **array** de strings si hay **parámetros**

# Aplicaciones multipágina: Router

book-detail.component.ts

## BookDetailComponent

ejem14

```
...
import { Router, ActivatedRoute } from '@angular/router';

@Component({
  template: `<h2>{{book.title}}</h2>
<div><label>Id: </label>{{book.id}}</div>
<div><label>Description: </label>{{book.description}}</div>
<p><button (click)="gotoBooks()">Back</button></p>`})
export class BookDetailComponent {
  book: Book;

  constructor(private router: Router,
    activatedRoute: ActivatedRoute, service: BookService) {

    let id = activatedRoute.snapshot.params['id'];
    this.book = service.getBook(id);
  }
  gotoBooks() { this.router.navigate(['/books']); }
}
```

# Aplicaciones multipágina: Router

book-detail.component.ts

## BookDetailComponent

ejem14

```
...
import { Router, ActivatedRoute } from '@angular/router';

@Component({
  template: `<h2>{{book.title}}</h2>
<div><label>Id: </label>{{book.id}}</div>
<div><label>Description: </label>{{book.description}}</div>
<p><button (click)="gotoBooks()">Back</button></p>`})
export class BookDetailComponent {
  book: Book;

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

Para acceder a los parámetros desde el componente usamos el servicio **ActivatedRoute**

# Aplicaciones multipágina: Router

book-detail.component.ts

## BookDetailComponent

ejem14

```
...
import { Router, ActivatedRoute } from '@angular/router';

@Component({
  template: `<h2>{{book.title}}</h2>
<div><label>Id: </label>{{book.id}}</div>
<div><label>Description: </label>{{book.description}}</div>
<p><button (click)="gotoBooks()">Back</button></p>`})
export class BookDetailComponent {
  book: Book;

  constructor(private router: Router,
    activatedRoute: ActivatedRoute, service: BookService) {

    let id = activatedRoute.snapshot.params['id'];
    this.book = service.getBook(id);
  }
  gotoBooks() { this.router.navigate(['/books']); }
}
```

Obtenemos un **snapshot** de los parámetros y accedemos al parámetro **id**

# Aplicaciones multipágina: Router

book-detail.component.ts

## BookDetailComponent

ejem14

```
...
import { Router, ActivatedRoute } from '@angular/router';

@Component({
  template: `<h2>{{book.title}}</h2>
<div><label>Id: </label>{{book.id}}</div>
<div><label>Description: </label>{{book.description}}</div>
<p><button (click)="gotoBooks()">Back</button></p>`})
export class BookDetailComponent {
  book: Book;

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

Para navegar desde código usamos el servicio **Router** y el método **navigate**

# Aplicaciones multipágina: Router

## • Funcionalidades avanzadas

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

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

# Ejercicio 6

- Implementa una **aplicación CRUD** de gestión de **libros**
- Funcionalidades (**varias pantallas**)
  - Listado de todos los libros (títulos)
  - Formulario de nuevo libro
  - Vista de detalle de un libro
  - Modificación de libro
  - Borrado de un libro
- Se proporciona una **API REST** en Java 8 (similar a la de los items).
- Cada libro tiene las propiedades: id, title, description

The screenshot shows a browser window titled "Ejem1" with the URL "localhost:4200/books". The page has a header "Library" and a section titled "Books" containing a list of book titles:

- [SUEÑOS DE ACERO Y NEON](#)
- [LA VIDA SECRETA DE LA MENTE](#)
- [CASI SIN QUERER](#)
- [TERMINAMOS Y OTROS POEMAS SIN TERMINAR](#)
- [LA LEGIÓN PERDIDA](#)

A "New book" button is located at the bottom left.

The screenshot shows a browser window titled "Ejem1" with the URL "localhost:4200/books/1". The page displays the details of the book "SUEÑOS DE ACERO Y NEON":

## Book "SUEÑOS DE ACERO Y NEON"

Los personajes que protagonizan este relato sobreviven en una sociedad en decadencia a la que, no obstante, lograrán devolver la posibilidad de un futuro. En un mundo dominado por las grandes corporaciones, solo un hombre, Thompson, detective privado deslenguado y vividor, pero de gran talento y sentido d...

Buttons: Remove, Edit, Back.

The screenshot shows a browser window titled "Ejem1" with the URL "localhost:4200/books/edit/1". The page displays the edit form for the book "SUEÑOS DE ACERO Y NEON":

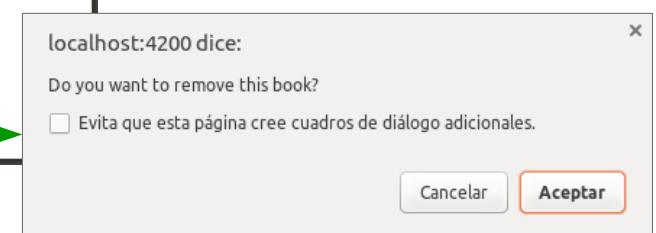
## Library

### Book "SUEÑOS DE ACERO Y NEON"

**Id:** 1  
**Title:** SUEÑOS DE ACERO Y N  
[Text area: Los personajes que protagonizan este]

**Description:** Description: Los personajes que protagonizan este

Buttons: Cancel, Save.



The screenshot shows a browser window titled "Ejem1" with the URL "localhost:4200/books/new". The page displays a new book creation form:

## Library

### Book ""

**Title:** title [Text area: description]

**Description:** Description: Description

Buttons: Cancel, Save.

# Tema 5

# Librerías de

# Componentes

Micael Gallego  
@micael\_gallego

# Librerías de componentes

- Angular 2 **no proporciona** componentes de alto nivel, permite usar HTML y CSS
- Pero existen **diferentes librerías** que facilitan el diseño de páginas web:
  - Frameworks CSS
  - Librerías específicas de Angular
    - Librerías con múltiples componentes
    - Librerías de charts (Gráficas)
    - Librerías de subida de ficheros
    - ...

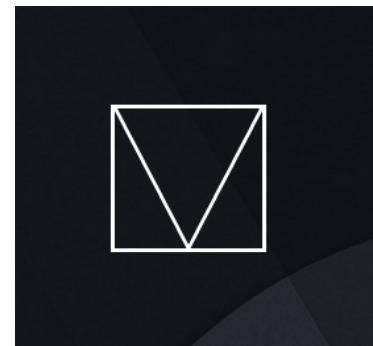
# Librerías de componentes

- **Frameworks CSS**

- Se puede usar cualquier framework de componentes que **sólo tenga CSS**



<http://getbootstrap.com/>



<http://www.getmdl.io/>



<http://semantic-ui.com/>

# Librerías de componentes

ejem15

- **Usar Bootstrap CSS**

- Instalamos la dependencia de Bootstrap 4 con NPM

```
npm install --save bootstrap
```

- Añadimos el .css en **angular.json** para que lo incluya en el **index.html**

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

# Librerías de componentes

- Usar Bootstrap CSS

ejem15

```
<h1>My First Angular 2 App</h1>  
  
<button type="button" class="btn btn-primary">Primary</button>
```



NOTA: No funciona con la carpeta node\_modules enlazada con un enlace simbólico

# Librerías de componentes

- **Librerías de componentes JavaScript**
  - No se recomienda usar directamente librerías gráficas JavaScript con Angular.
  - Para que un programa Angular pueda funcionar correctamente, tiene que tener **control total sobre el DOM**
  - Además, aunque alguna librería **funcione**, puede ser muy **ineficiente** porque Angular modifica partes del browser

# Librerías de componentes

- **Librerías de componentes JavaScript**
  - **JQuery**: Es mejor modificar el DOM con plantillas u otros mecanismos avanzados de Angular
  - **JavaScript de Bootstrap**: Está basado en jQuery. Es mejor usar una versión específica de Angular que veremos más adelante.
  - **Otras librerías**: Es mejor usar aquellas con diseñadas específicamente para Angular o que disponen de “adaptadores” para Angular. Esto evita problemas de rendimiento y funcionamiento en ciertos contextos

# Librerías de componentes

- **ng-bootstrap**

- Reimplementación de la parte JavaScript de Bootstrap 4 basada en Angular



## Bootstrap widgets

The angular way

Angular widgets built from the ground up using only Bootstrap 4 CSS with APIs designed for the Angular ecosystem.

No dependencies on 3rd party JavaScript.

[Demo](#)

[Installation](#)

Currently at v6.0.1

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

## • Instalación

- Instalar CSS de Bootstrap (como hemos visto antes)
- Instalar dependencia NPM

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

```
ng add @angular/localize
```

- Alta en el módulo (sección imports)

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

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

- Están implementados la mayoría de los componentes Bootstrap
  - Alert
  - Buttons
  - Carousel
  - Collapse
  - Datepicker
  - Dropdown
  - Modal
  - Pagination
  - Popover
  - Progressbar
  - Rating
  - Tabs
  - Accordion
  - Timepicker
  - Tooltip

ejem16

- Ejemplo



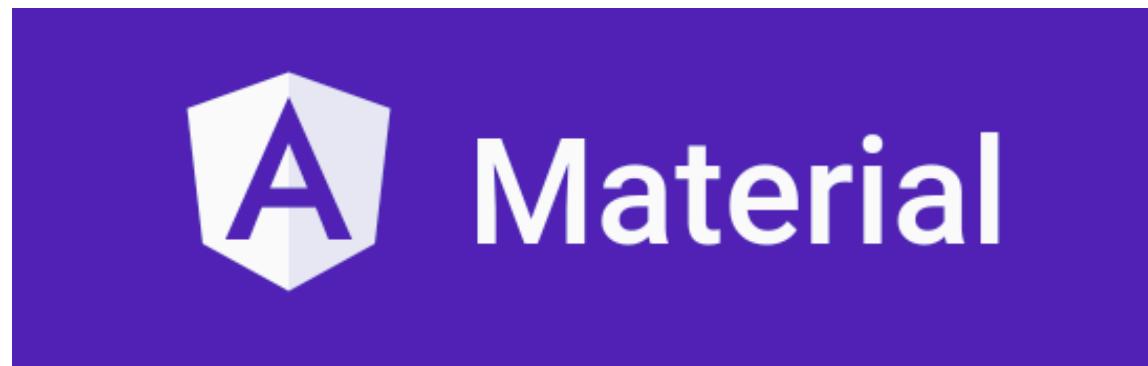
- Ejemplo (tabs)

```
<ngb-tabset>
  <ngb-tab title="Tab 1">
    <ng-template ngbTabContent>
      <p>Contenido de la tab 1</p>
      <button>Un botón</button>
    </ng-template>
  </ngb-tab>
  <ngb-tab>
    <ng-template ngbTabTitle><b>Tab</b> 2</ng-template>
    <ng-template ngbTabContent>
      <p>Contenido de la tab 2</p>
    </ng-template>
  </ngb-tab>
  <ngb-tab title="Tab 3" [disabled]="tab3Disabled">
    <ng-template ngbTabContent>
      <p>Contenido de la tab 3</p>
    </ng-template>
  </ngb-tab>
</ngb-tabset>
```

# Librerías de componentes

- **Angular Material**

- Librería de componentes con el estilo Material de Google para Angular
- Implementada por el equipo de Angular



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

# Angular Material

## Form Controls

Radio buttons, checkboxes, input fields, sliders, slide toggles, selects



## Navigation

Sidenav, toolbars, menus



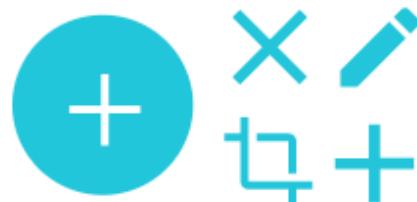
## Layout

Lists, grid-lists, cards



## Buttons, Indicators & Icons

buttons, button toggles, icons, progress spinners, progress bars



## Popups & Modals

Dialogs, tooltips, snackbars



Aplicación de demo: <https://material2-app.firebaseio.com/>

# Angular Material

- **Instalación**

- Instalar dependencia NPM

```
ng add @angular/material
```

- Alta en el módulo (sección imports)

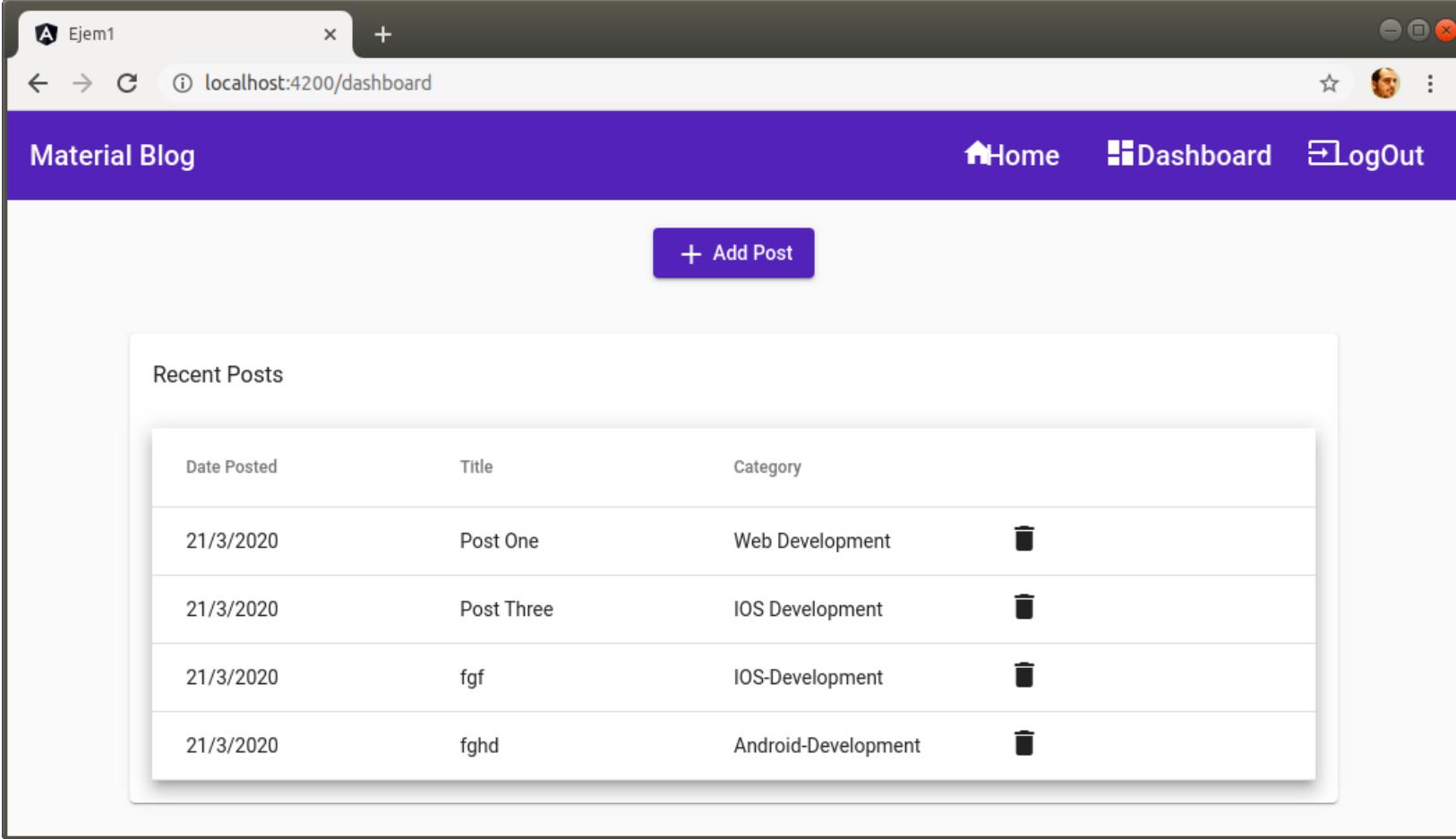
```
import { MaterialModule } from
'@angular/material';

@NgModule({
  ...
  imports: [MaterialModule.forRoot(), ...],
})
export class AppModule {}
```

# Angular Material

ejem17

- Ejemplo



The screenshot shows a web browser window titled "Ejem1" displaying a dashboard application at "localhost:4200/dashboard". The header is purple and contains the title "Material Blog", navigation links for "Home", "Dashboard", and "LogOut", and a user profile icon. A central button labeled "+ Add Post" is visible. Below this, a card titled "Recent Posts" displays a table of five posts:

Date Posted	Title	Category	Action
21/3/2020	Post One	Web Development	trash
21/3/2020	Post Three	IOS Development	trash
21/3/2020	fjf	IOS-Development	trash
21/3/2020	fghd	Android-Development	trash

# Angular Material

ejem17

```

<mat-card class="card" >
  <mat-card-title fxLayout.gt-xs="row" fxLayout.xs="column">
    <h3>Recent Posts</h3>
  </mat-card-title>
  <mat-card-content>
    <div class="example-container mat-elevation-z8">
      <mat-table #table [dataSource]="dataSource">
        <ng-container matColumnDef="date_posted">
          <mat-header-cell *matHeaderCellDef> Date Posted </mat-header-cell>
          <mat-cell *matCellDef="let element"> {{element.date_posted | date: 'd/M/y'}} </mat-cell>
        </ng-container>
        <ng-container matColumnDef="title">
          <mat-header-cell *matHeaderCellDef> Title </mat-header-cell>
          <mat-cell *matCellDef="let element"> {{element.title}} </mat-cell>
        </ng-container>
        <ng-container matColumnDef="category">
          <mat-header-cell *matHeaderCellDef> Category </mat-header-cell>
          <mat-cell *matCellDef="let element"> {{element.category}} </mat-cell>
        </ng-container>
        <ng-container matColumnDef="delete">
          <mat-header-cell *matHeaderCellDef></mat-header-cell>
          <mat-cell *matCellDef="let element">
            <a (click)="deletePost(element.position)" type="button">
              <mat-icon class="icon">delete</mat-icon>
            </a>
          </mat-cell>
        </ng-container>
        <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
        <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>
      </mat-table>
    </div>
  </mat-card-content>
</mat-card>

```

# Librerías de componentes

- **Teradata Covalent**

- Librería que añade más componentes a Angular Material
- Implementada por el equipo de Teradata
- Está en desarrollo (beta.1)



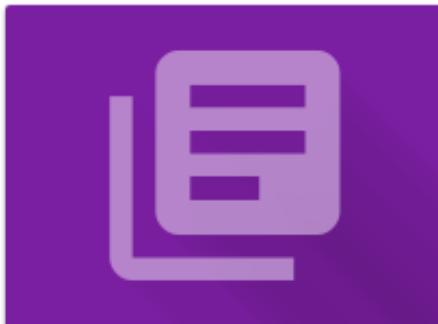
<https://teradata.github.io/covalent/>

# Teradata Covalent



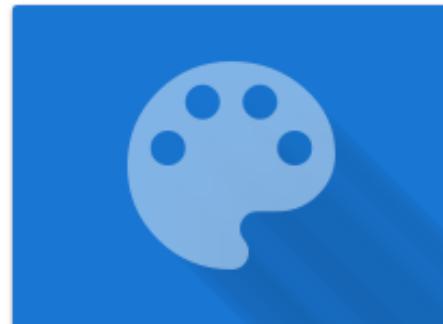
## Teradata Covalent UI Platform

a UI platform built on Angular 2 + Material Design



### Documentation

Your guide to start using the UI platform in your app!



### Style Guide

Teradata brand logo usage, color palettes and more



### Layouts

Several different material design layout options for your apps

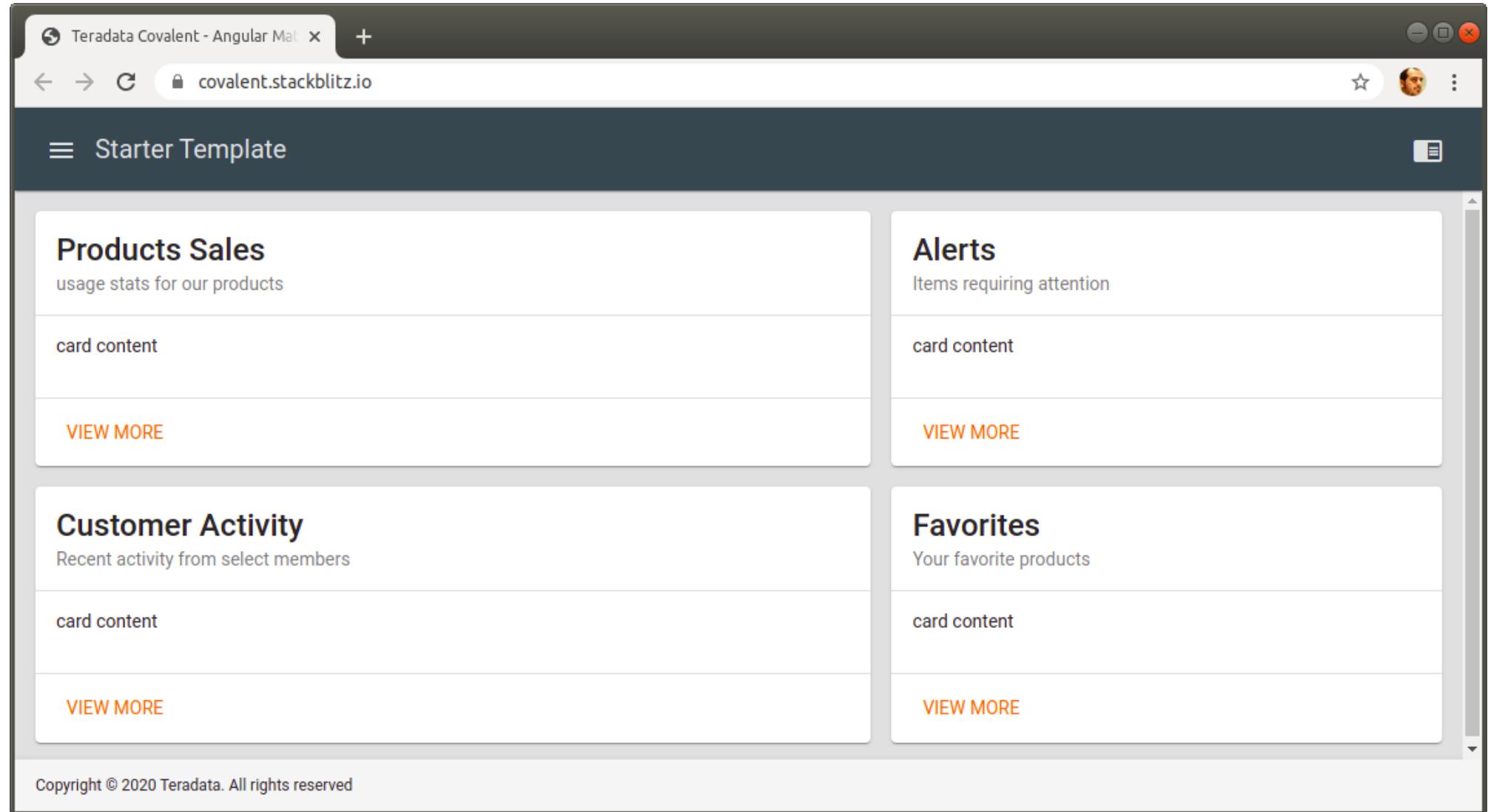


### Components & Addons

Covalent Components, Directives, Pipes & Services

<https://teradata.github.io/covalent/>

# Teradata Covalent



A screenshot of a web browser displaying the "Teradata Covalent - Angular Mat" application at [covalent.stackblitz.io](https://covalent.stackblitz.io). The page has a dark blue header with the title "Starter Template". The main content area is divided into four sections arranged in a 2x2 grid:

- Products Sales**: Subtitle "usage stats for our products". Content "card content". Call-to-action button "VIEW MORE".
- Alerts**: Subtitle "Items requiring attention". Content "card content". Call-to-action button "VIEW MORE".
- Customer Activity**: Subtitle "Recent activity from select members". Content "card content". Call-to-action button "VIEW MORE".
- Favorites**: Subtitle "Your favorite products". Content "card content". Call-to-action button "VIEW MORE".

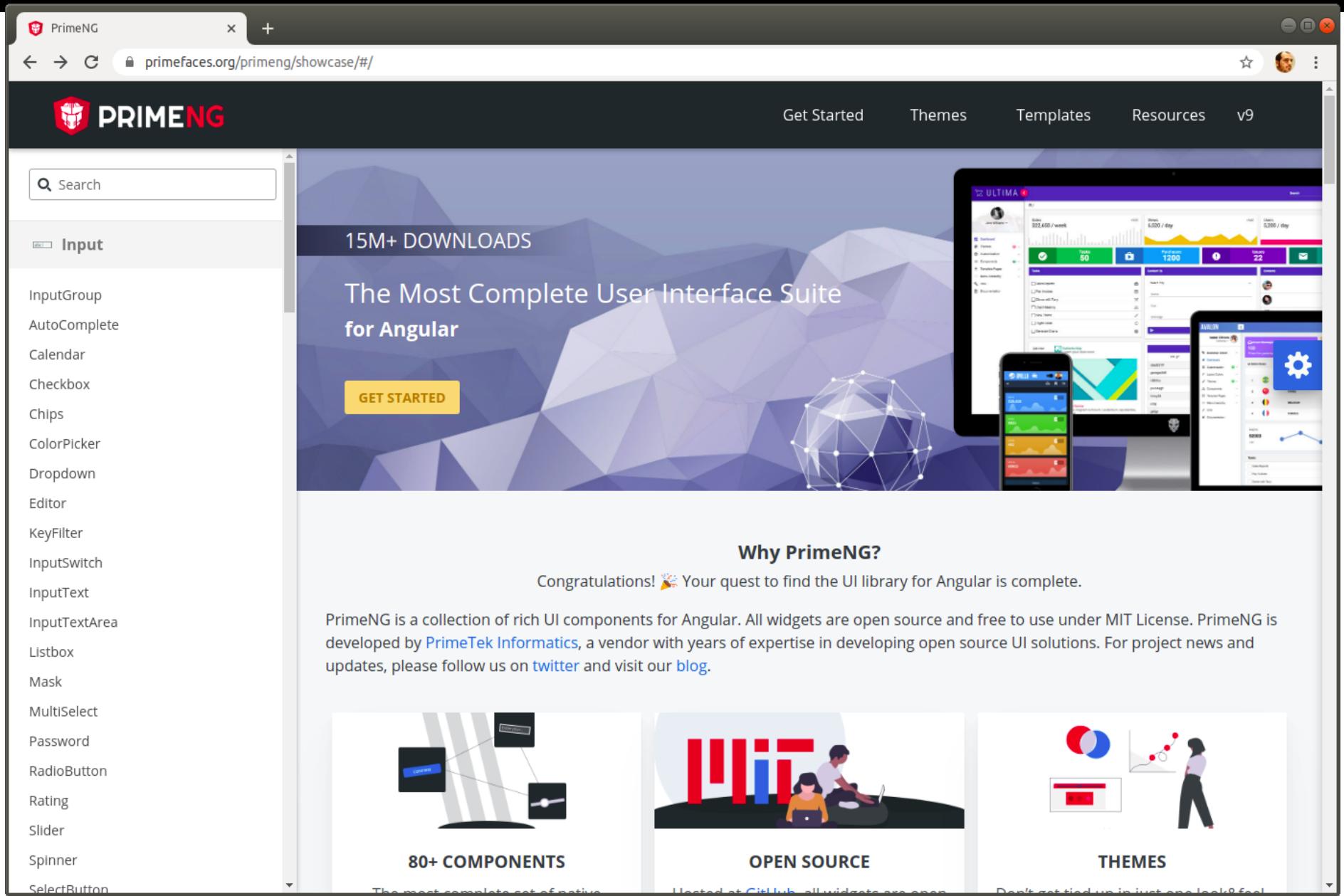
The bottom left corner of the page contains the copyright notice "Copyright © 2020 Teradata. All rights reserved".

# Librerías de componentes

- PrimeNG

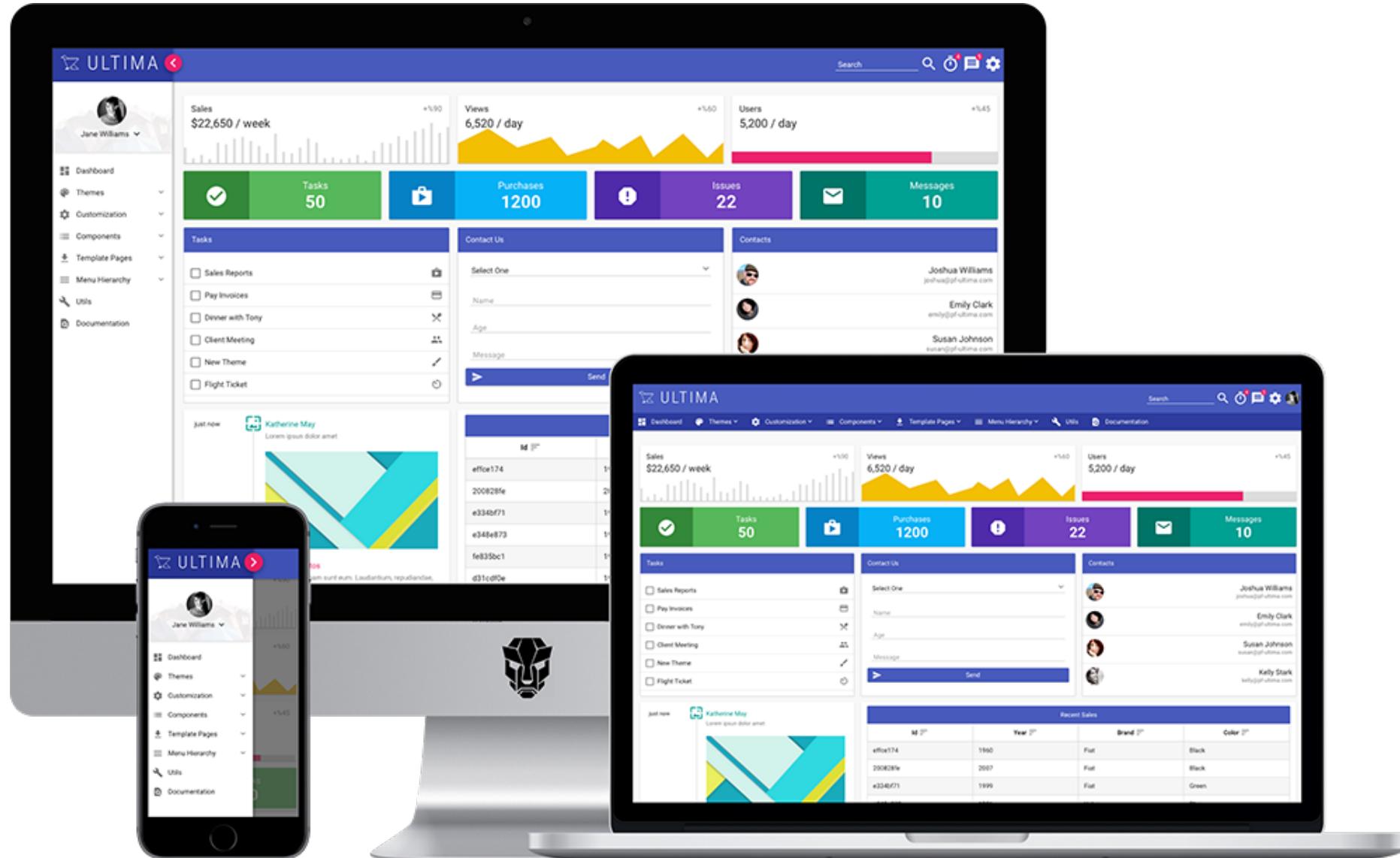


<http://www.primefaces.org/primeng/>



The screenshot shows the PrimeNG website homepage. The header features the PrimeNG logo and a search bar. The main navigation includes links for "Get Started", "Themes", "Templates", "Resources", and "v9". A sidebar on the left lists various UI components under the "Input" category, such as InputGroup, AutoComplete, Calendar, Checkbox, Chips, ColorPicker, Dropdown, Editor, KeyFilter, InputSwitch, InputText, InputTextArea, Listbox, Mask, MultiSelect, Password, RadioButton, Rating, Slider, Spinner, and SelectButton. The main content area has a purple geometric background. It displays the text "15M+ DOWNLOADS" and "The Most Complete User Interface Suite for Angular". A "GET STARTED" button is present. To the right, there are images of a tablet and a smartphone displaying different PrimeNG UI components. Below this, a section titled "Why PrimeNG?" includes the text "Congratulations! 🎉 Your quest to find the UI library for Angular is complete." and a paragraph about PrimeNG being a collection of rich UI components for Angular, developed by PrimeTek Informatics under MIT License. It encourages users to follow on Twitter and visit the blog. At the bottom, there are three cards: "80+ COMPONENTS" (illustrated with a diagram of interconnected components), "OPEN SOURCE" (illustrated with people working on a laptop), and "THEMES" (illustrated with a person looking at a screen). The footer contains the text "The most complete set of reactive" and "Hosted at GitHub all widgets are open source".

# PrimeNG



# Librerías de componentes

- Existen librerías que proporcionan un componente concreto (o de la misma familia)
  - ag-grid: Tabla con controles avanzados
  - ng-charts



<https://www.ag-grid.com/>

# Librerías de componentes

Demo of ag-Grid: Datagrid with [Data](#) [Rows](#) [Columns](#) [Pivot Mode](#) [Performance](#) [Filter](#) [Search](#)

ag-Grid

Data Size: 100 Rows, 22 Cols | Theme: Alpine | Filter: Filter any column... | Take a video tour

Participant	Game of Choice	Performance	Monthly Breakd	Pivot Mode					
Name	Language	Country	Game Name	Bought	Bank Balance	Rating	Total Winnings	Jan	Feb
Tony Smith	English	Ireland	Chess	✓	\$2,397	★★	\$569,571	\$38,031	
Andrew Connell	Swedish	Sweden	Bul	✓	\$12,749	★★★	\$481,734	\$17,697	
Kevin Flanagan	Spanish	Uruguay	Rithmomachy	✗	\$95,078		\$747,956	\$54,892	
Bricker McGee	French	France	Kalah	✗	\$65,506		\$605,384	\$81,081	
Dimple Unalkat	Portuguese	Portugal	Game of the Generals	✓	\$85,310	★★	\$600,036	\$73,947	
Gil Lopes	Spanish	Colombia	Hare and Hounds	✓	\$75,701	★★	\$574,681	\$20,479	
Sophie Beckham	English	Ireland	Sugoroku	✗	\$66,706		\$651,234	\$44,446	
Isabelle Black	French	France	Nine Men's Morris	✗	\$15,749	★★★	\$497,221	\$32,411	
Emily Braxton	Maltese	Malta	Blockade	✓	\$4,057	★★★★	\$622,755	\$82,948	
Olivia Brennan	French	France	Patolli	✓	\$32,835	★	\$727,405	\$93,514	
Lily Brock	Italian	Italy	YINSH	✗	\$7,440	★★★★	\$563,168	\$34,466	
Chloe Bryson	Greek	Greece	Downfall	✗	\$65,717	★	\$544,903	\$48,057	
Isabella Cadwell	English	Ireland	Gipf	✓	\$53,143	★★★★	\$598,959	\$59,018	
Amelia Cage	English	Ireland	Shogi	✓	\$28,394	★★★	\$616,276	\$50,571	
Jessica Carson	Swedish	Sweden	Mad Gab	✗	\$76,329	★★★	\$793,660	\$84,256	
Sophia Chandler	French	France	Agon	✗	\$81,286	★★	\$507,707	\$48,063	

Rows: 100

Columns

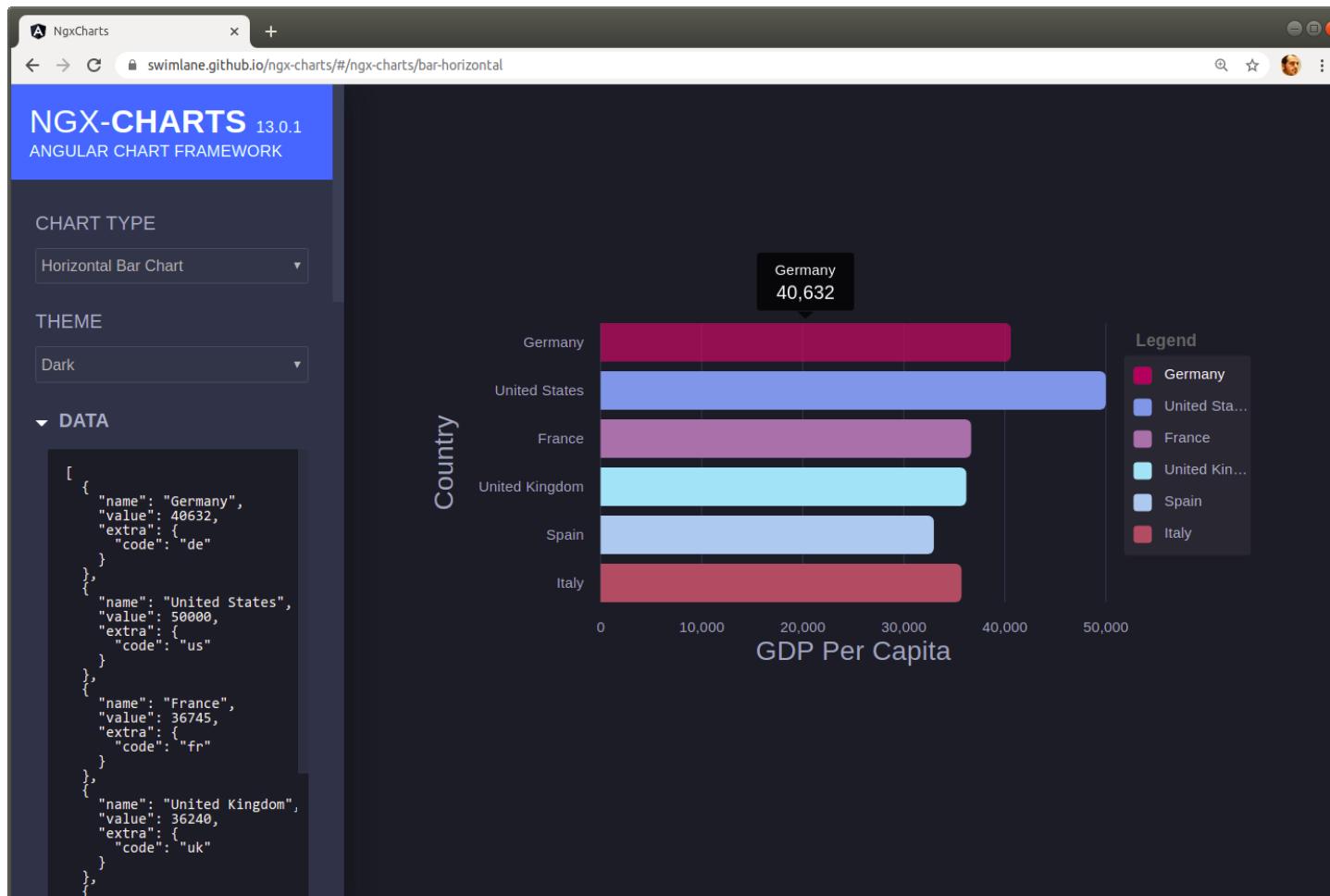
Filters

Row Groups

Values

# Librerías de componentes

- **ngx-charts:** Gráficos de barras, líneas, etc...



<https://swimlane.github.io/ngx-charts/>

# Librerías de componentes

- **Varias librerías de Valor Software**

- ng2-bootstrap
- ng2-charts
- ng2-file-upload
- ng2-table
- ng2-tree
- ng2-dragula: Drag and drop
- ng2-select



<https://github.com/valor-software?q=ng2->

# Librerías de componentes

- Varias librerías de Valor Software

Angular2 File Upload

Select files

Base drop zone

Another drop zone

Multiple

Choose Files No file chosen

Single

Choose File No file chosen

Upload queue

Queue length: 0

Name	Size	Progress	Status	Actions
------	------	----------	--------	---------

Queue progress:

⌚ Upload all ⚡ Cancel all 🗑 Remove all

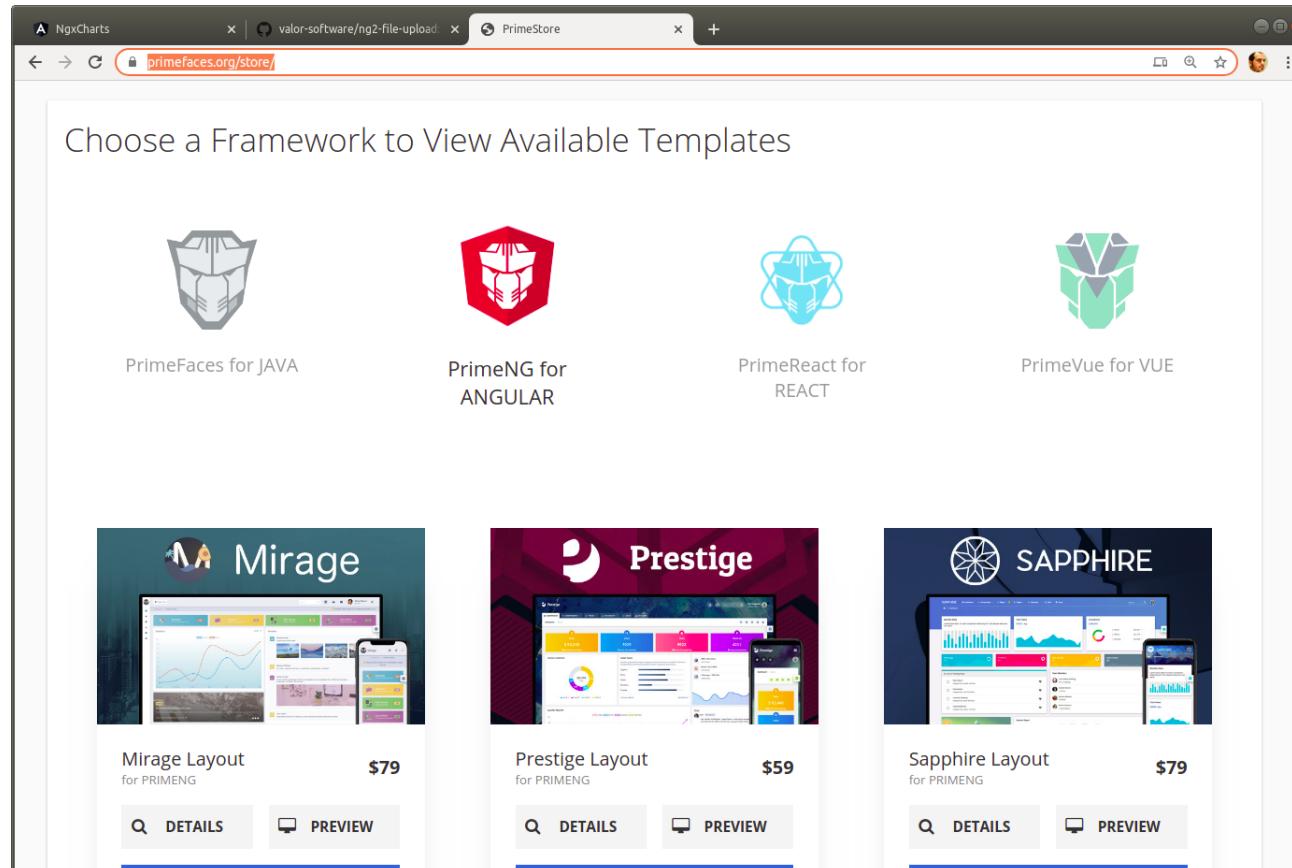
<https://github.com/valor-software?q=ng2->

# Librerías de componentes

- **Plantillas de aplicación**
  - Como las **plantillas web**, pero para Angular
  - Lo más habitual son los “paneles de **administración**”
  - Los de mayor calidad suelen tener un **coste económico**
  - Al seleccionar una plantilla hay que tener en cuenta:
    - Compatible con AOT (optimización del código)
    - Usa jQuery? No funcionará con *server side rendering*
    - Está basado en **angular-cli**? Qué versión. Algunos están basados en otras estructuras de proyecto

# Librerías de componentes

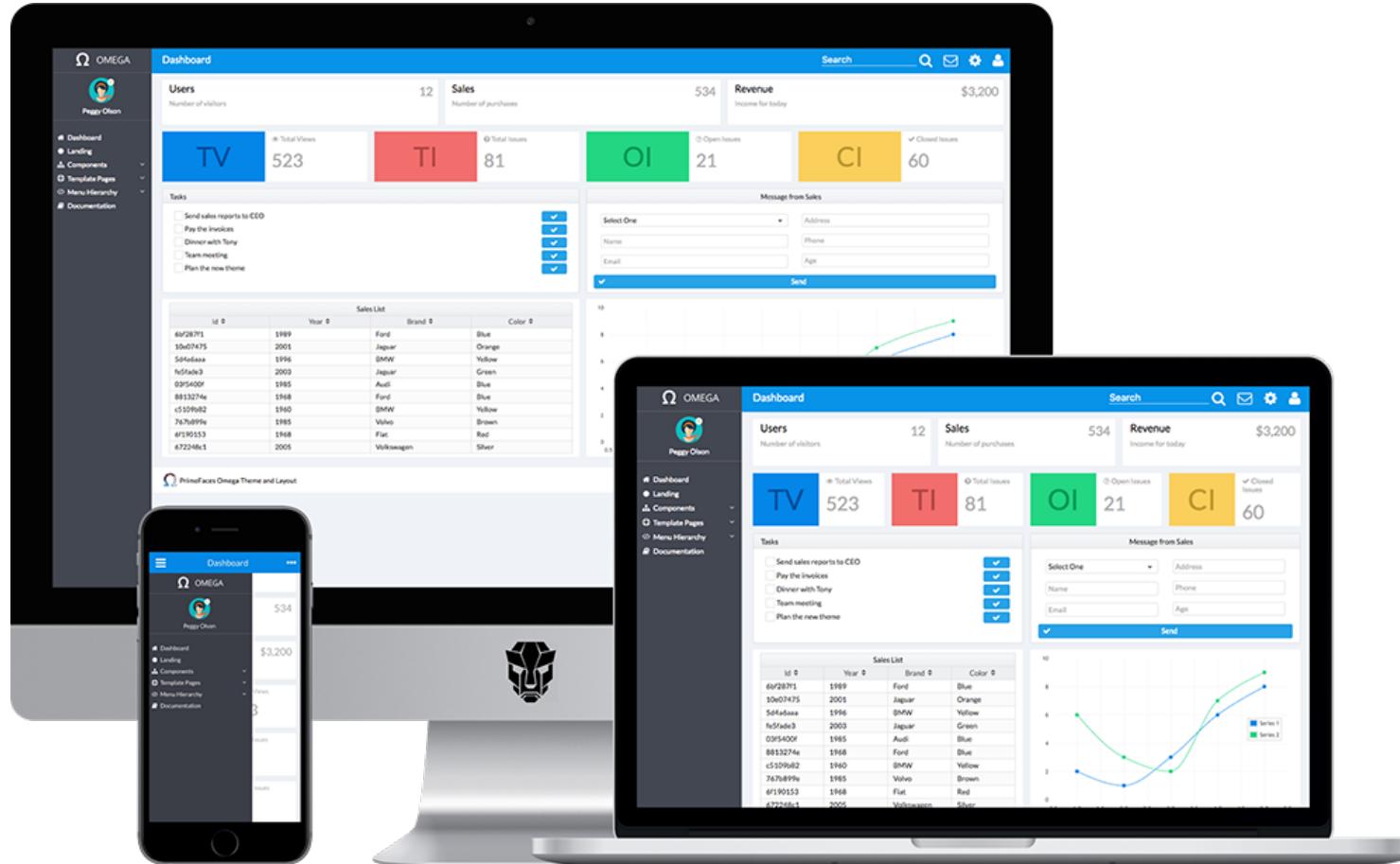
- **PrimeNG Templates**



<https://www.primefaces.org/store/>

# Librerías de componentes

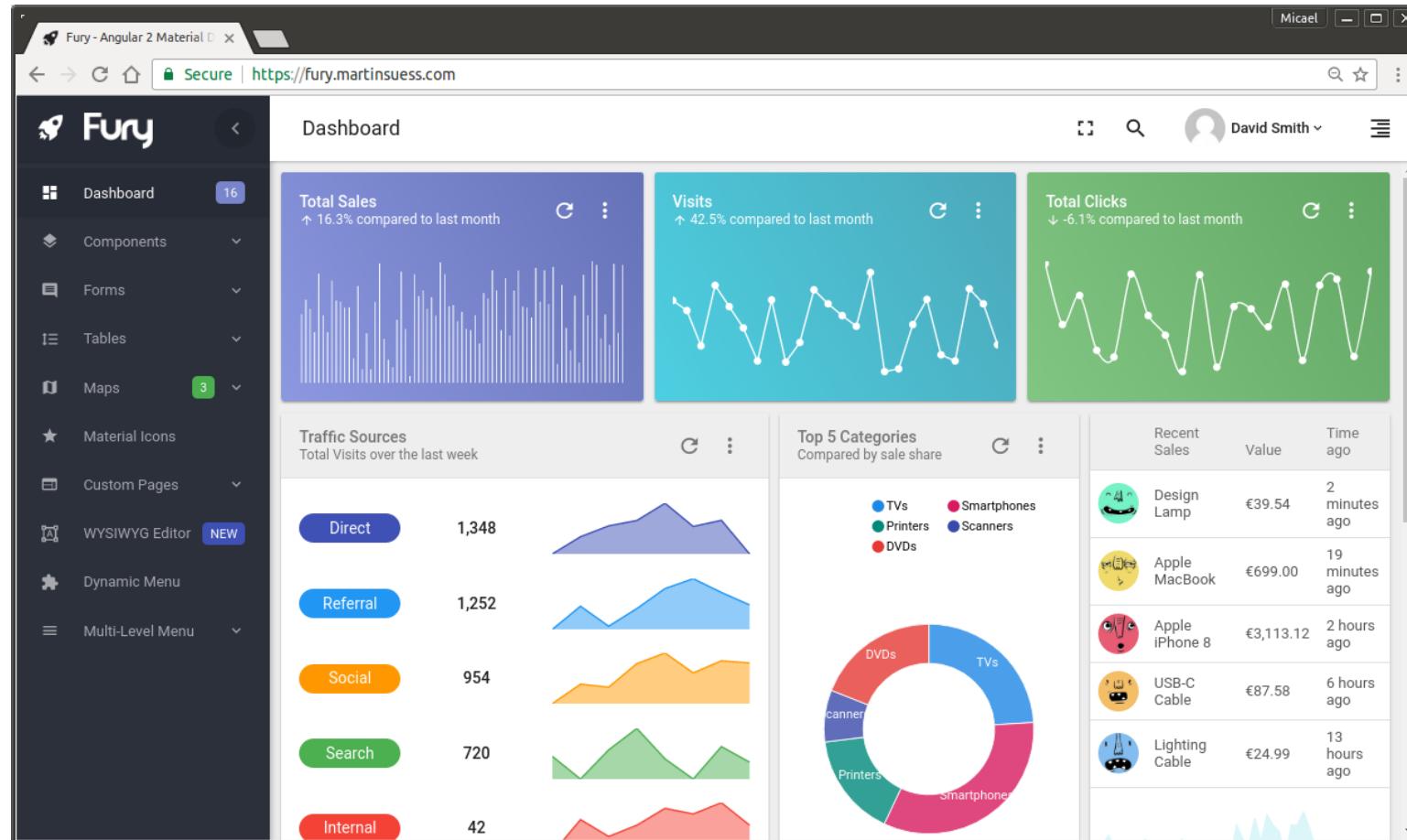
- PrimeNG Omega



<http://www.primefaces.org/layouts/omega-ng#>

# Librerías de componentes

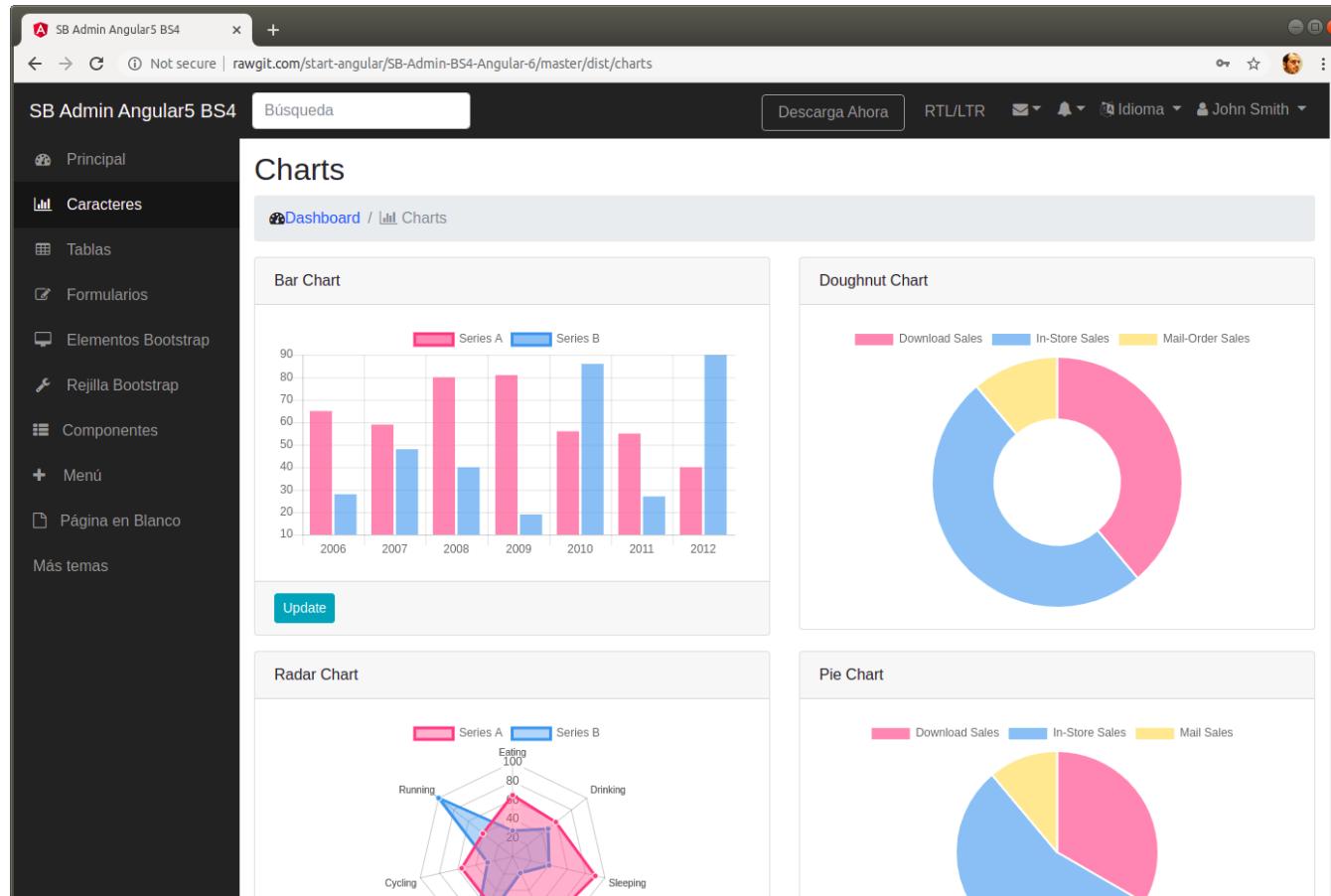
- Fury - Angular 2 Material Design Admin Template



<https://themeforest.net/item/fury-angular-2-material-design-admin-template/19325966>

# Librerías de componentes

- SB Admin Angular 9 and Bootstrap 4



<https://github.com/start-angular/SB-Admin-BS4-Angular-8>

# Ejercicio

- **Selecciona una librería de componentes o un template**
- **Mejora el aspecto de la aplicación de gestión de libros**
  - Haz que los formularios tengan “estilo”
  - Que los cuadros de diálogo no sean los nativos del navegador
  - En el listado principal incluye más información además del título del libro (forma de tabla)
  - Pon una gráfica de “descargas” (con datos simulados)

# Tema 6

# Publicación

Micael Gallego  
@micael\_gallego

# Publicación

- **Angular en modo desarrollo**

- Hasta ahora hemos visto cómo usar **angular-cli** para desarrollar aplicaciones
- LiveReload: Guardar fichero y actualización del navegador
- Código optimizado para depuración
- Tiempo de procesamiento bajo

```
ng serve
```

# Publicación

- **Publicación de aplicaciones SPA**

- Las aplicaciones SPA tienen parte **frontend** y parte **backend**
- El **frontend** se implementa con Angular (o con cualquier otro framework)
- El **backend** se implementa con cualquier tecnología como Java, Groovy, PHP, Python

# Publicación

- **Publicación de backend SPA**
  - Se necesita un servidor con la tecnología necesaria: Java, PHP, Python, Go, Node...
  - Ofrece un interfaz de comunicación con el SPA: REST, WebSocket, SSE (basado en http)
  - Puede hospedar los recursos de frontend o no (si están hospedados en otro sitio)

# Publicación

- **Publicación de frontend SPA**
  - Se generan los ficheros de la app SPA para producción
  - Estos ficheros se publican en un servidor HTTP
  - Pueden estar incluso en una red de distribución de contenidos (CDN)
  - Pueden estar en un servidor diferente al backend (y habitualmente están)

# Publicación

- Construcción para publicación
  - Comando

```
$ ng build --target=production --environment=prod
```

- El resultado está disponible en /build

<https://www.sitepoint.com/ultimate-angular-cli-reference/>

# Publicación

- **Publicación de frontend SPA**

- En un servidor web local

```
$ sudo npm install -g http-serve  
$ cd dist  
$ http-serve .
```

- Existen servicios gratuitos para publicar la parte de front
  - GitHub Pages
  - Firebase tiene capa gratuita limitado en uso

# Publicación

- Publicación en github
  - Crear un proyecto
  - Subir el contenido de dist
  - La página está disponible en

`http://username.github.io/repository`

`https://pages.github.com/`

# Publicación

- **Publicación en github**
  - En ciertos casos la app SPA tiene que “conocer” la URL en la que se va a desplegar

```
$ ng build --target=production -environment=prod\  
--base-href="http://username.github.io/repository"
```

<https://pages.github.com/>

- **Ahead of Time (AOT) compilation**
  - Por defecto en Angular los templates se analizan y procesan en el navegador
  - El análisis se realiza al cargar la aplicación
  - El analizador es un módulo importante (ocupa decenas de Kbytes)
  - Compilador Just In Time (JIT) porque analiza los templates cuando se usan

# Publicación

- **Ahead of Time (AOT) compilation**
  - Es posible analizar los templates en el momento de la construcción de la aplicación
  - Se reduce el tiempo de inicio de la aplicación (no hay que analizar templates)
  - En general se reduce el tamaño de la app (al no incluir analizador)
  - En la última versión de angular-cli se usa al general la aplicación para publicación