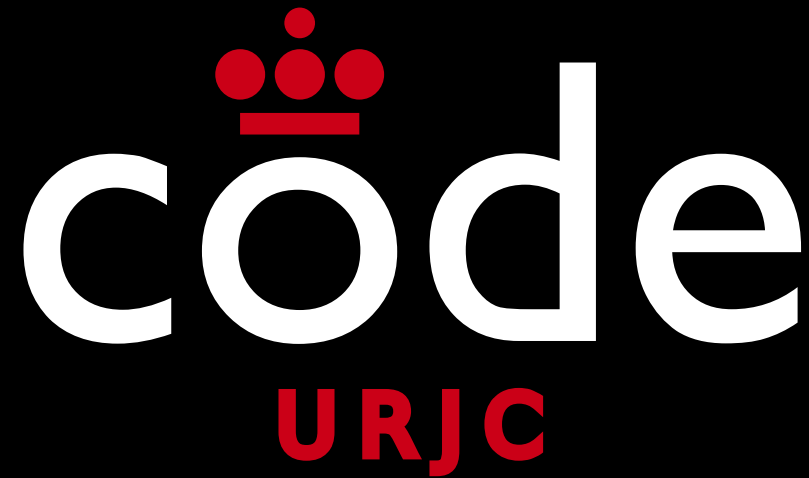


# TypeScript Angular 2 Ionic 2

**Micael Gallego**  
micael.gallego@urjc.es  
@micael\_gallego



# Consultoría y Formación en Desarrollo Software

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



Universidad  
Rey Juan Carlos

<http://codeurjc.github.io>

**Micael Gallego**

[micael.gallego@urjc.es](mailto:micael.gallego@urjc.es)

[@micael\\_gallego](https://twitter.com/micael_gallego)

# SPA con TypeScript y Angular 2

**Micael Gallego**  
micael.gallego@urjc.es  
@micael\_gallego

- **Introducción a Angular 2**
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

- La **última tendencia** en el desarrollo web es la implementación de aplicaciones **web SPA**
- Las **web SPA** son clientes completos implementados con **HTML, CSS y JavaScript** que se comunican con el **servidor web con API REST**
- Existen **frameworks** especialmente diseñados para implementar **webs SPA**
- Uno de los frameworks más usados es **Angular**
- **Angular 2** se encuentra en desarrollo (beta)

## Frameworks / librerías SPA



BACKBONE.JS



React



ANGULARJS

by Google

## Frameworks / librerías SPA



BACKBONE.JS



React



ANGULARJS

by Google

## Angular 2

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

<https://angular.io/>





## Angular 2 vs Angular 1

- Está en **beta** (Marzo 2016)
- Está implementado desde cero, no como una evolución de Angular 1
- Angular 2 **no es compatible con Angular 1**
- Cuidado, la **documentación de Angular 1 no sirve** para Angular 2

~~\$scope~~

## Lenguaje programación Angular 2

- Angular 2 tiene soporte oficial para desarrollo de apps con JavaScript (**ES5** y **ES6**) y **TypeScript**
- Se puede usar cualquier lenguaje que transpile a JavaScript



## Lenguaje programación Angular 2

- Angular 2 tiene soporte oficial para desarrollo de apps con JavaScript (**ES5** y **ES6**) y **TypeScript**
- Se puede usar cualquier lenguaje que transpile a JavaScript



## Funcionalidades de Angular 2

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

- Introducción a Angular 2
- **TypeScript**
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

## Características

- Añade **tipos estáticos** a JavaScript **ES6**
  - Inferencia de tipos
  - Tipos opcionales
- El **compilador** genera código JavaScript **ES5** (compatible con los navegadores web actuales)
- Orientado a Objetos con clases (no como ES5)
- Anotaciones

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

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



## Ventajas frente a JavaScript

- Con el **tipado estático** el compilador puede verificar la **corrección** de muchas más cosas que con el tipado dinámico
- Los programas grandes son **menos propensos a errores**
- Los **IDEs** y **editores** pueden: Autocompletar, Refactorizar, Navegar a la definición
- **Muy parecido a Java y C#**

## Facilidad de adopción para JavaScripters

- Los tipos son **opcionales**
- La **inferencia de tipos** permite no tener que escribir los tipos constantemente
- En realidad es **JavaScript con más cosas**, así que todo lo conocido se puede aplicar
- Un mismo proyecto puede **combinar JS y TS**, lo que facilita migrar un proyecto existente



```
export class Empleado {                                     TypeScript

    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 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;  
    }  
}
```

## Classes 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;  
    }  
}
```

## Classes 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;  
    }  
}
```

## Classes 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;  
    }  
}
```



## 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);  
});
```

## 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);
});
```

## 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 del mismo paquete (carpeta) se pueden usar sin importar  
En TypeScript se tienen que importar porque cada fichero es un módulo diferente

## 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 del SDK.  
En TypeScript usamos el Array nativo de JavaScript

## 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 TypeScript Array 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);
});
```

## 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 lamda de Java  
se llaman **arrow function** en TypeScript (ES6)

## 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 {  
  
    constructor(  
        private nombre:string,  
        private sueldo:number) {}  
  
    alerta(button:HTMLButtonElement) {  
        button.onclick = e => {  
            alert(this.nombre);  
        }  
    }  
}
```

En TypeScript una arrow function permite usar **this** y siempre apunta al objeto, no al evento (no es necesario usar **that**)



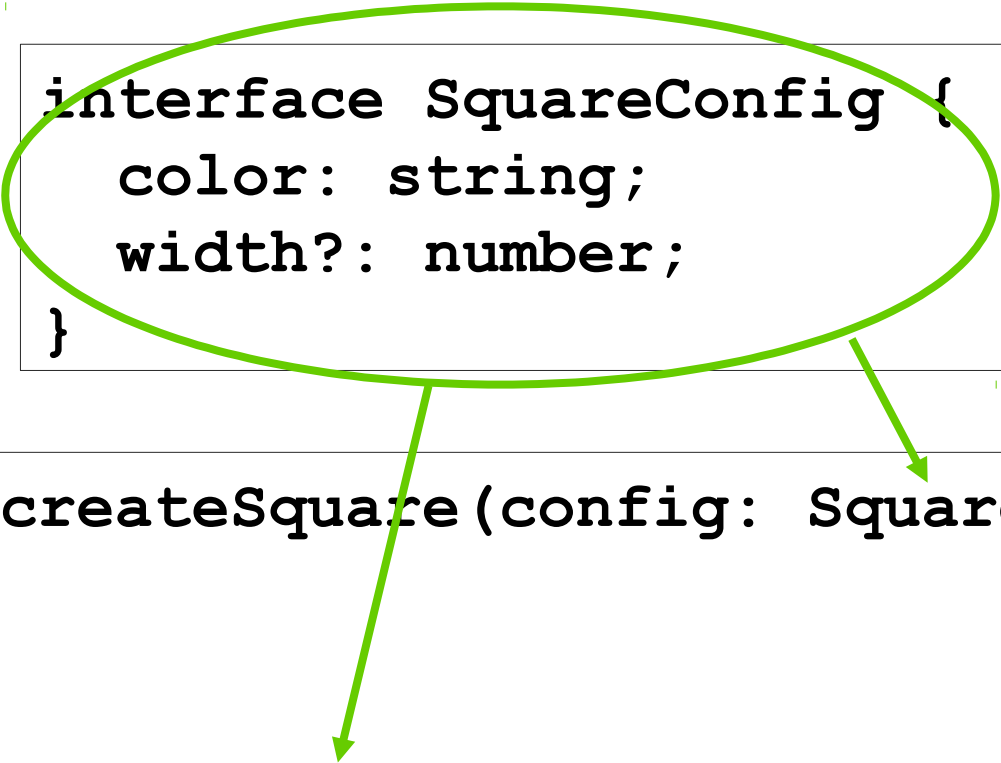
## Objetos literales en TypeScript

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

```
function createSquare(config: SquareConfig) {  
    ...  
}  
  
createSquare({color: "black"});  
createSquare({color: "black", width: 20});
```

## Objetos literales en TypeScript

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



```
function createSquare(config: SquareConfig) {  
  ...  
}
```

```
createSquare({color: "black"});  
createSquare({color: "black", width: 20});
```

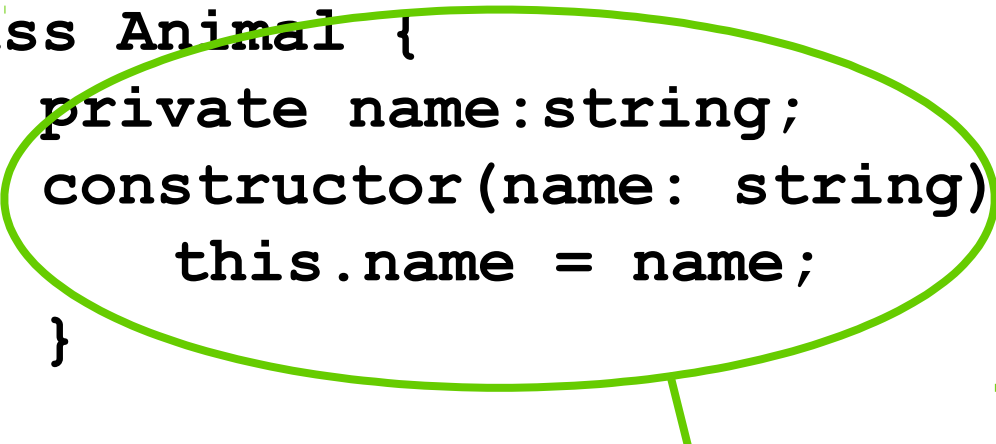
## Atributos inicializados en el constructor

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

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

## Atributos inicializados en el constructor

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



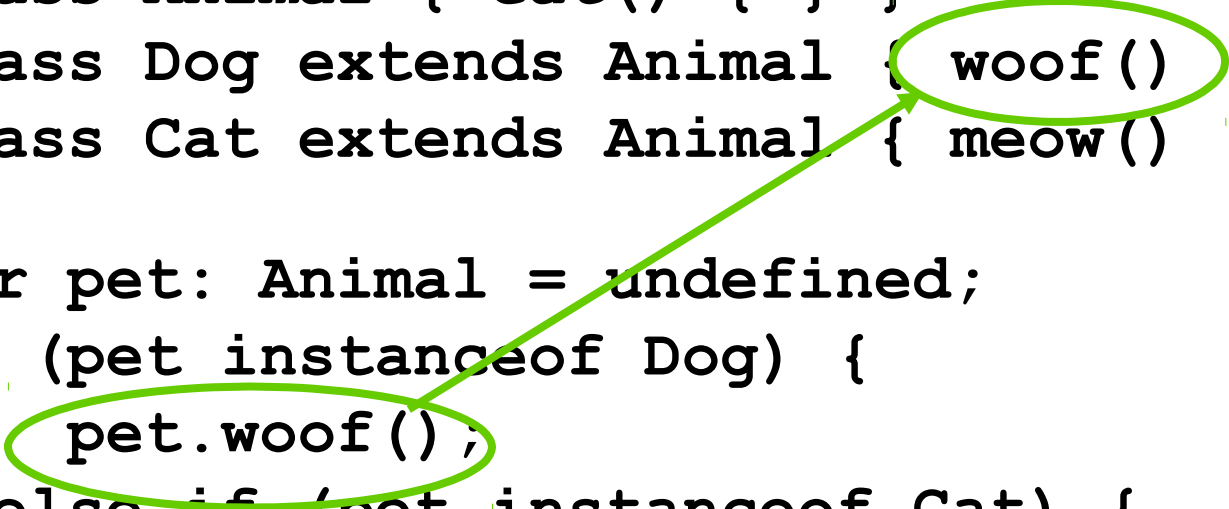
```
class Animal {  
    constructor(private name: string) {  
    }  
}
```

## Instanceof sin casting

```
class Animal { eat() { } }  
class Dog extends Animal { woof() { } }  
class Cat extends Animal { meow() { } }  
  
var pet: Animal = undefined;  
if (pet instanceof Dog) {  
    pet.woof();  
} else if (pet instanceof Cat) {  
    pet.meow();  
} else {  
    pet.eat();  
}
```

## Instanceof sin casting

```
class Animal { eat() { } }  
class Dog extends Animal { woof() { } }  
class Cat extends Animal { meow() { } }  
  
var pet: Animal = undefined;  
if (pet instanceof Dog) {  
    pet.woof();  
} else if (pet instanceof Cat) {  
    pet.meow();  
} else {  
    pet.eat();  
}
```

A green oval highlights the `woof()` method in the `Dog` class definition. Another green oval highlights the `pet.woof();` line in the `if` block. A green arrow points from the `pet.woof();` line to the `woof()` method in the `Dog` class, illustrating the runtime casting process.

## Compatibilidad de tipos estructural

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

## Compatibilidad de tipos estructural

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

Una variable de tipo User sólo necesita un objeto que tenga un atributo name. Un objeto de la clase Profile lo cumple.



## Editores / IDEs

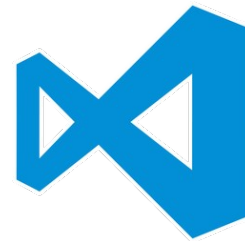
Hay plugins para la mayoría de los editores / IDEs



ATOM



Sublime Text



Visual Studio  
Code



WebStorm



**NetBeans**



Brackets



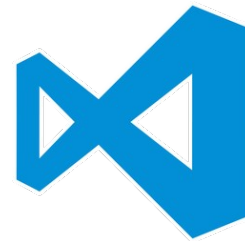
**eclipse**

## Editores / IDEs

Hay plugins para la mayoría de los editores / IDEs



Sublime Text



Visual Studio  
Code



WebStorm



**NetBeans**



Brackets



**eclipse**

# WebStorm 11



File Edit View Navigate Code Refactor Run Tools VCS Window Help

typescriptdemo Empleado.ts

Project

- typescriptdemo [typescript]
- build
- Empleado.js
- Empleado.ts
- tsconfig.json
- External Libraries

Structure

```
1 export class Empleado {
2
3     private nombre: string;
4     private salario: number;
5
6     constructor(nombre: string, salario: number){
7         this.nombre = nombre;
8         this.salario = salario;
9     }
10
11     getNombre(){
12         return this.nombre;
13     }
14
15     toString(){
16         return "Nombre:" + this.nombre +
17             ", Salario:" + this.salario;
18     }
19 }
20
21 let emp = new Empleado("Pepe", 200);
22
```

TypeScript Compiler: Current Errors Project Errors Console

No Errors

2: Favorites

Terminal TypeScript Compiler 6: TODO Event Log

TypeScript Compiler: Discovered tsconfig.json: null (today 11:28) 21:37 LF UTF-8

## WebStorm 11



```
11  getNombre(){
12      return this.nombre;
13  }
14
15  toString(){
16      return "Nombre:"+this.nombre+
17          ", Salario:"+this.salarario;
18  }
19  }
20
21  let emp = new Empleado("Pepe", 200);
22  emp.
```

Autocomplete dropdown menu showing suggestions for the `emp.` property access:

- `getNombre()`
- `toString()`
- `constructor Object (lib.d.ts)`
- `hasOwnProperty([string] v)` boolean
- `isPrototypeOf([Object] v)` boolean
- `propertyIsEnumerable([string] v)` boolean
- `toLocaleString()` string
- `valueOf()` Object
- `else` if (!expr)
- `if` if (expr)
- `it in` for(var obj in expr)

Ctrl+Abajo and Ctrl+Arriba will move caret down and up in the editor >>



```
20  
21 let emp = new Empleado("Pepe", 200);  
22  
23 emp.getSuelto();  
24
```

mpiler: Current Errors Project Errors Console

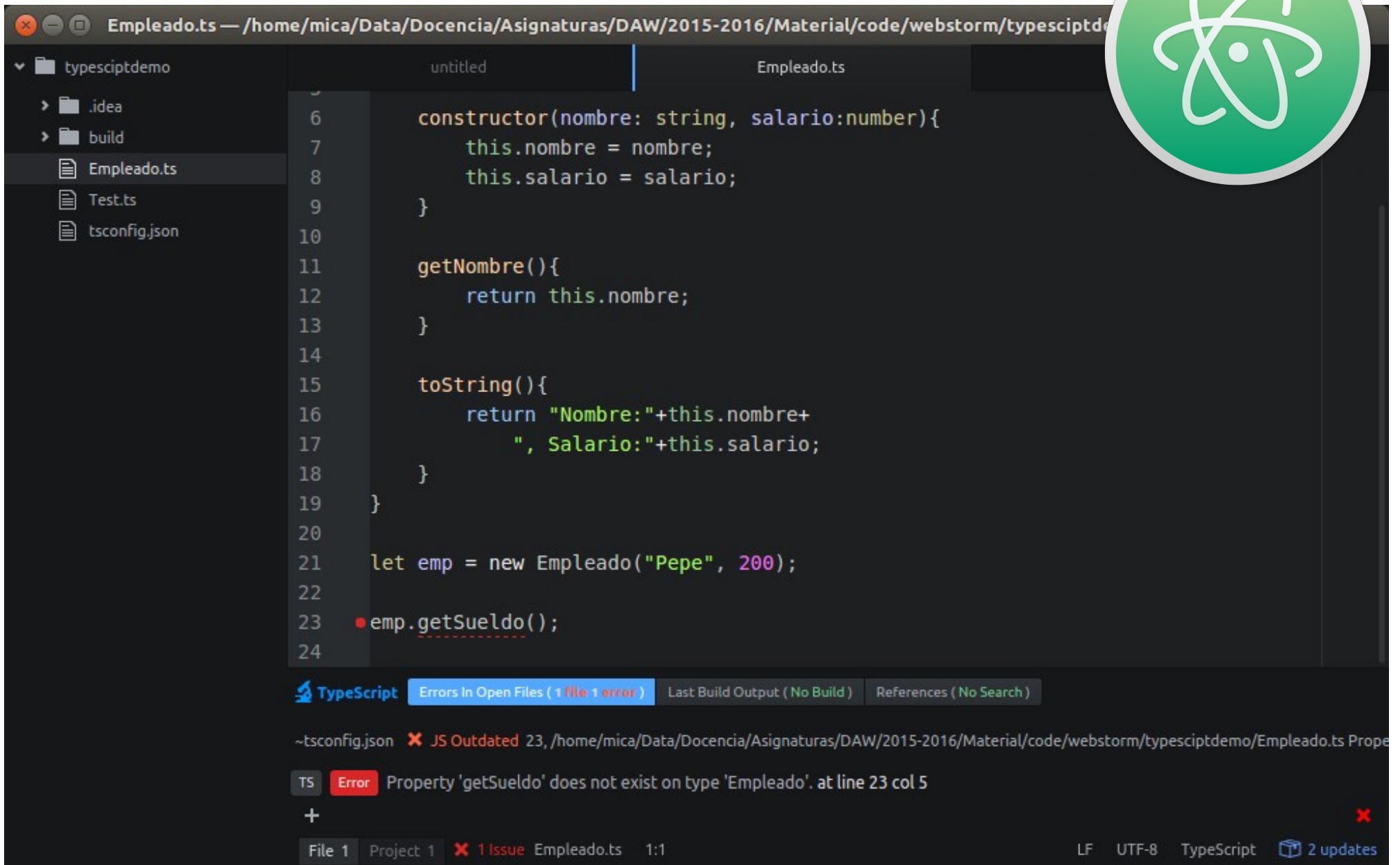


/home/mica/Data/Docencia/Asignaturas/DAW/2015-2016/Material/code/webstorm  
ts



Error:(23, 5) TS2339: Property 'getSuelto' does not exist on type 'Empleado'.

# Atom / atom-typescript

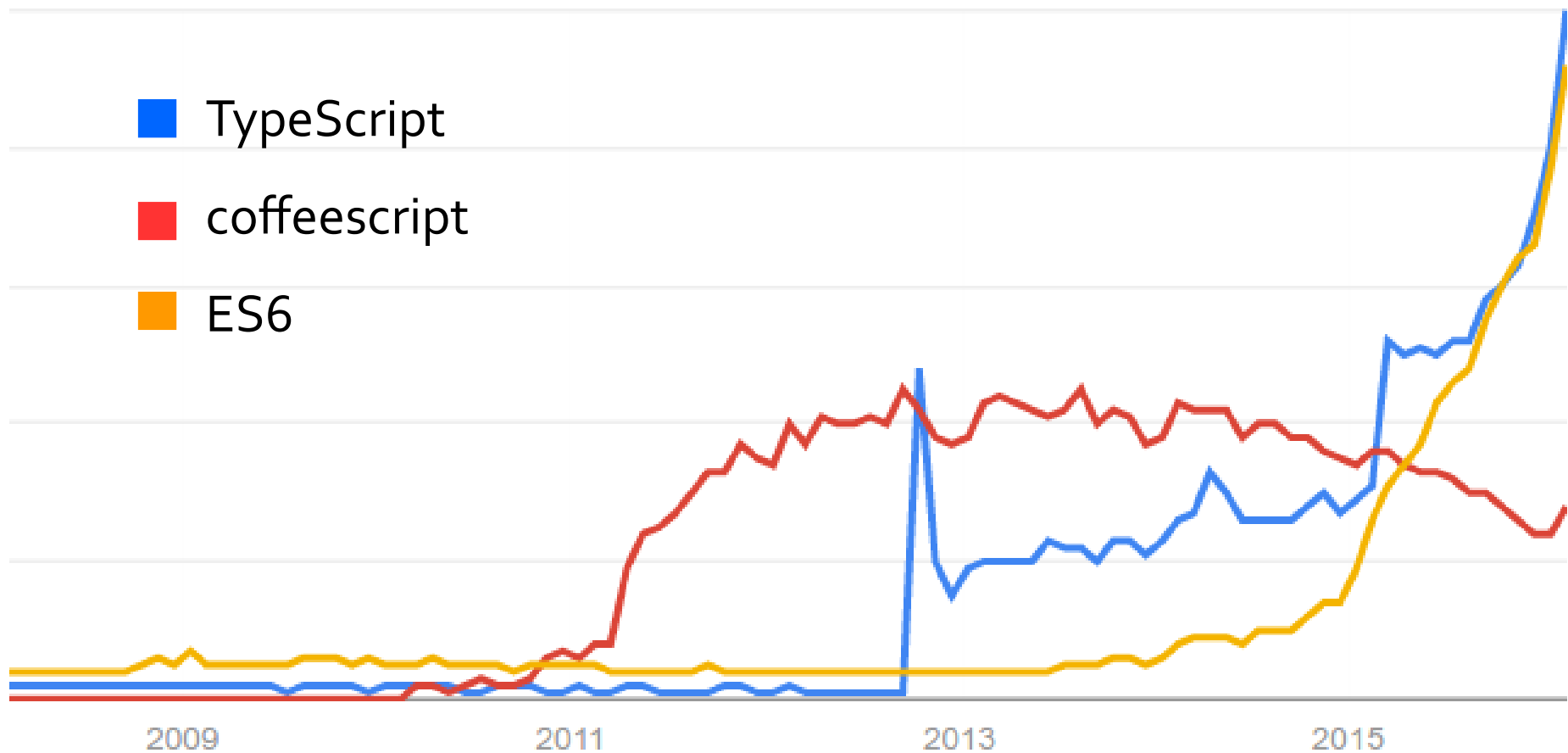
A screenshot of the Atom text editor interface. The top bar shows the file path: Empleado.ts — /home/mica/Data/Docencia/Asignaturas/DAW/2015-2016/Material/code/webstorm/typescriptdemo. The left sidebar shows a file tree with folders .idea and build, and files Empleado.ts, Test.ts, and tsconfig.json. The main editor area shows the code for Empleado.ts. The code defines a class Empleado with a constructor and two methods: getNombre() and toString(). Below the class definition, there is a variable declaration and an attempt to call a method that does not exist.

```
6      constructor(nombre: string, salario:number){
7          this.nombre = nombre;
8          this.salario = salario;
9      }
10
11     getNombre(){
12         return this.nombre;
13     }
14
15     toString(){
16         return "Nombre:"+this.nombre+
17             ", Salario:"+this.salario;
18     }
19 }
20
21 let emp = new Empleado("Pepe", 200);
22
23 emp.getSueldo();
24
```

The bottom status bar shows the TypeScript error: "Property 'getSueldo' does not exist on type 'Empleado'. at line 23 col 5". The error message is displayed in a red box. The status bar also shows "File 1", "Project 1", "1 Issue", "Empleado.ts", "1:1", "LF", "UTF-8", "TypeScript", and "2 updates".

<https://atom.io/packages/atom-typescript>

## Popularidad de TypeScript



- Introducción a Angular 2
- TypeScript
- **Herramientas de desarrollo**
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones



## Angular2 con brackets

- Es posible desarrollar aplicaciones **angular 2** únicamente con un **editor** y un servidor **web** (p.e. **brackets**)
- **Inconvenientes**
  - El código tarda más en cargarse y en ejecutarse
  - Notificación de errores mucho más limitada
  - Sin autocompletar ni navegación entre los elementos del código

## Plataforma y gestión de paquetes



Plataforma para ejecutar aplicaciones JS fuera del navegador



Gestor de herramientas de desarrollo y librerías JavaScript (integrado con node.js)

## Instalación node.js y npm

- Para usar las herramientas de desarrollo de Angular 2 es necesario instalar **node.js 4** o superior
- Instalación windows y mac

<https://nodejs.org/en/download/stable/>

- Instalación linux

<https://nodejs.org/en/download/package-manager/>

- Ubuntu

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -  
sudo apt-get install -y nodejs  
sudo apt-get install -y nodejs-legacy
```

## Construcción de proyectos / empaquetado

- Existen muchas **herramientas** para procesar los fuentes de la aplicación
- **Objetivos:**
  - Reducción del tiempo de descarga
  - Preprocesadores CSS
  - Optimización del código, CSS, HTML
  - Cumplimiento de estilos y generación de JavaScript

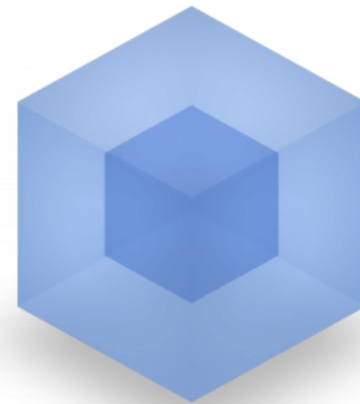
## Construcción de proyectos / empaquetado



<http://gruntjs.com/>



<http://gulpjs.com/>



**webpack**  
MODULE BUNDLER

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



<http://broccolijs.com/>

## Generación de código esqueleto

- No se suele crear un proyecto **desde cero** porque hay muchos **ficheros y carpetas** que son muy **parecidos** en todos los proyectos
- Existen **muchos enfoques** para conseguir el esqueleto inicial de una web SPA
- Nos centraremos en los específicos para **Angular 2**

## Generación de código esqueleto

- Un proyecto Angular 2 se puede construir con **gulp, grunt, webpack, broccoli, ...**
- Existen **varios enfoques** para la carga de la página, la optimización de recursos, el proceso de desarrollo...
- Cada una de las **técnicas** que veremos usan **enfoques diferentes** para todos estos aspectos

## Generación de código esqueleto

- Generadores basados en Yeoman
- Proyectos semilla (*seed*) disponibles en github
- **Herramienta oficial** de gestión de proyectos



YEOMAN





## Generación de código esqueleto

- Generadores de código Angular 2 no oficiales basados en Yeoman
  - <https://www.npmjs.com/package/generator-modern-web-dev>
  - <https://www.npmjs.com/package/generator-angular2>
  - <https://www.npmjs.com/package/generator-gulp-angular2>
  - <https://github.com/joshuacaron/generator-angular2-gulp-webpack>
  - <https://www.npmjs.com/package/slush-angular2>

**NOTA:** Es conveniente verificar si están actualizados a la última versión de Angular 2. Pueden estar desactualizados



## Generación de código esqueleto

- Proyectos semilla (*seed*) disponibles en github
  - <http://mgechev.github.io/angular2-seed/>
  - <https://github.com/ghpabs/angular2-seed-project>
  - <https://github.com/cureon/angular2-sass-gulp-boilerplate>
  - <https://angularclass.github.io/angular2-webpack-starter/>
  - <https://github.com/LuxDie/angular2-seed-jade>
  - <https://github.com/justindujardin/angular2-seed>

**NOTA:** Es conveniente verificar si están actualizados a la última versión de Angular 2. Pueden estar desactualizados



## Generación de código esqueleto

- Herramienta oficial de gestión de proyectos
- <https://github.com/angular/angular-cli>
- Ofrece **comandos** para:
  - Generación del proyecto **inicial**
  - Generación de partes **posteriormente**
  - Modo desarrollo con compilado automático de **TypeScript** y actualización del **navegador**
  - Construcción del proyecto para distribución (**build**)



- **Herramienta oficial angular-cli**
  - **Instalación**
    - Pueden ser necesarios permisos de administrador

```
npm install -g angular-cli@0.0.24
```

- **Generación del proyecto**
  - Se descargarán 250Mb de Internet y se configurarán las herramientas. Puede tardar bastante tiempo.

```
ng new main  
cd main
```

- Herramienta oficial angular-cli



<http://broccolijs.com/>

**Construcción  
del proyecto**



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



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



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

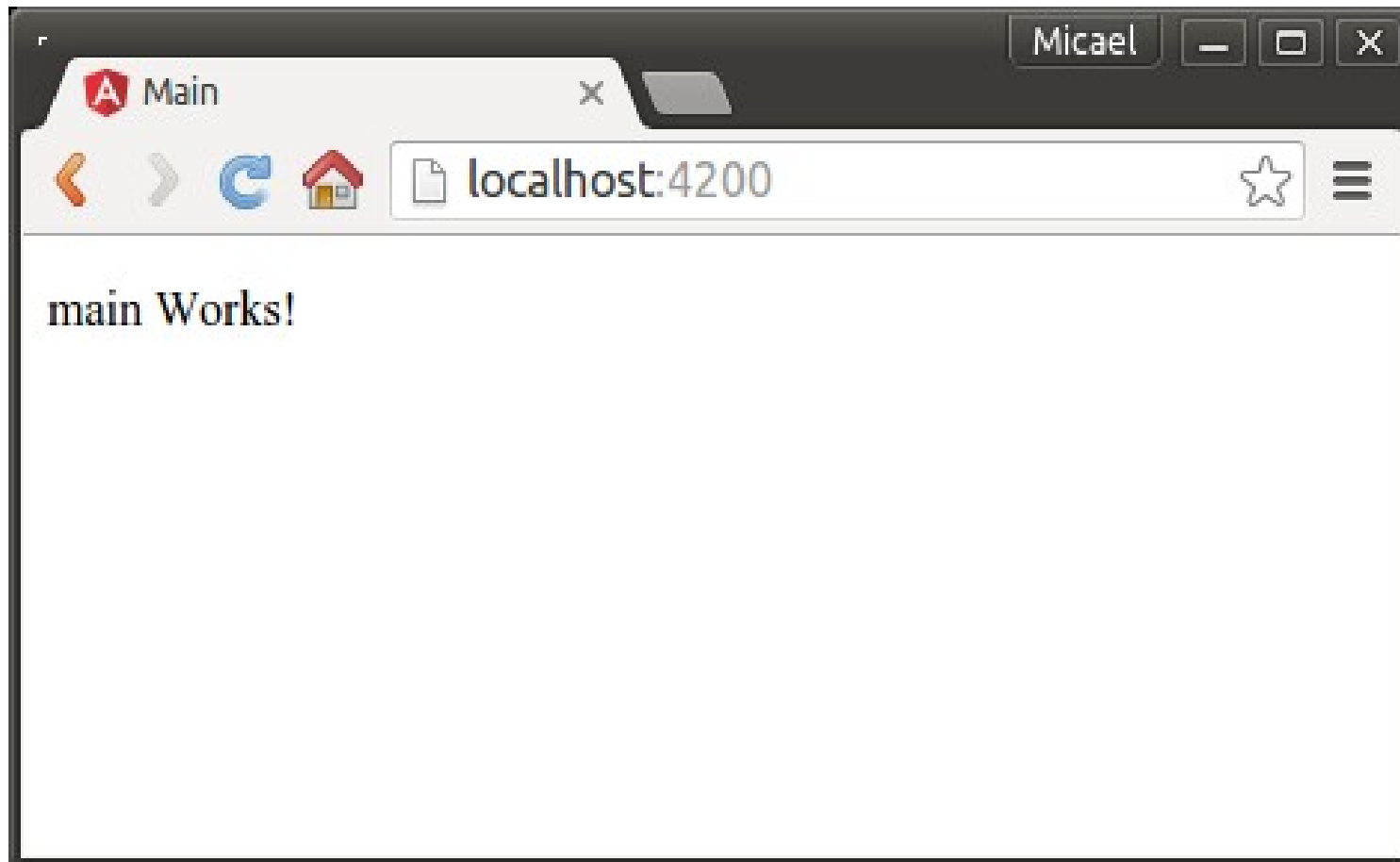
**Herramientas de testing**

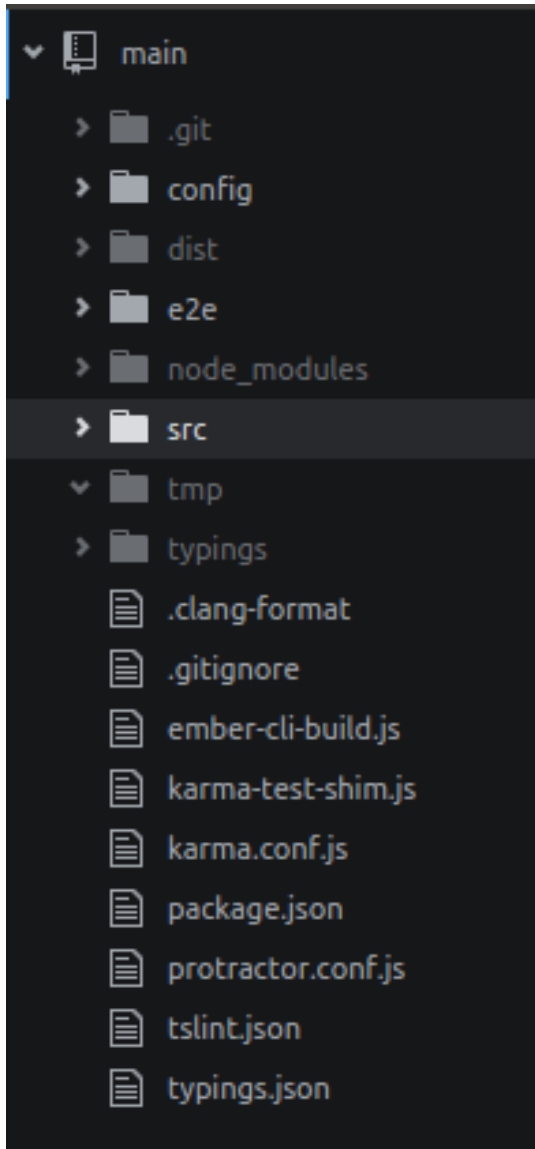
- **Herramienta oficial angular-cli**
  - Ejecutar servidor web en modo desarrollo
    - Se iniciará un servidor web en `http://localhost:4200`
    - Hay que abrir el navegador para ejecutar la app
    - Al guardar un fichero fuente, la aplicación se recargará automáticamente
    - El código TypeScript que transpilará a JavaScript automáticamente

```
ng serve
```

- En Windows es mejor ejecutar el comando como administrador para que vaya más rápido

- Herramienta oficial angular-cli





- **Ficheros/Carpetas generadas**
  - **dist:** Recursos que hay que publicar en el servidor web
  - **node\_modules:** Librerías y herramientas descargadas
  - **src:** Fuentes de la aplicación
  - **package.json:** Configuración de librerías y herramientas
  - **typings.json:** Más configuración de librerías



- **Ficheros/Carpetas generadas**

- **src/app:**

- Carpeta que contiene los ficheros **fuentes principales** de la aplicación.

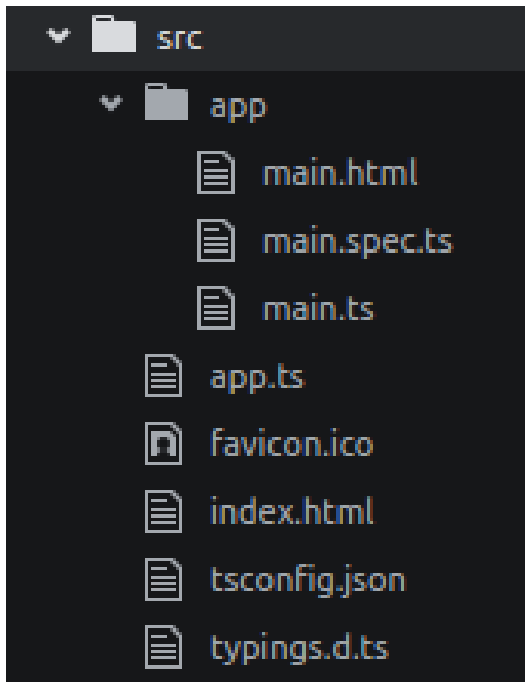
- **Borraremos su contenido** y le sustituiremos por los ejemplos

- **app.ts:** Fichero de configuración de la aplicación

- **favicon.ico:** Icono de la aplicación

- **index.html:** Página principal. Se editará para incluir CSS globales en la web (bootstrap)

- **tsconfig.json:** Configuración del compilador TypeScript

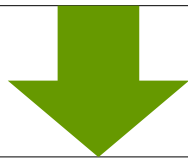


- Herramienta oficial angular-cli

main.html

```
<p>
  main Works!
</p>

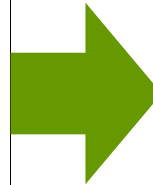
<router-outlet></router-outlet>
```



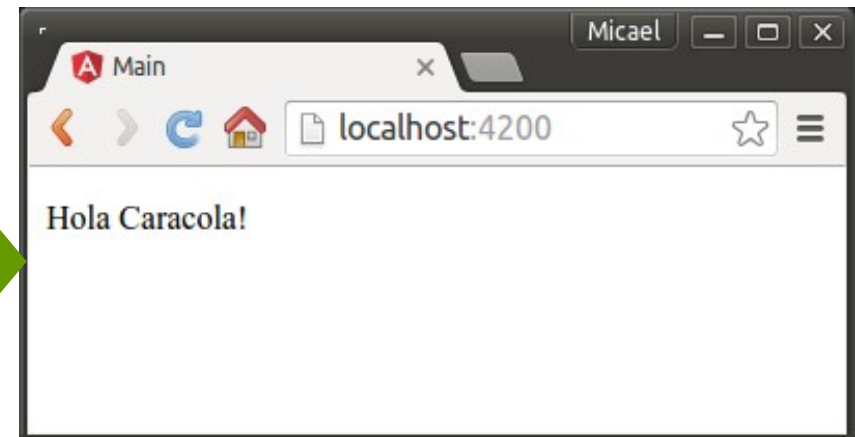
main.html

```
<p>
  Hola Caracola!
</p>

<router-outlet></router-outlet>
```



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



- **Herramienta oficial angular-cli**
  - **Generar el contenido para publicar en producción**
    - Cuando queremos publicar la aplicación en producción tenemos que generar los archivos optimizados y publicarlos en un **servidor web**
    - Usamos el comando

```
ng build
```

- Que genera los ficheros en la carpeta **dist**
- Actualmente los ficheros no están muy optimizados, pero se espera que si lo estén cuando se publique la versión final de Angular 2

- Edición en Atom – TypeScript
  - Configuramos atom para que no genere los ficheros JavaScript (porque los genera angular-cli)

```
{
  "compilerOptions": {
    "declaration": false,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "mapRoot": "",
    "module": "system",
    "moduleResolution": "node",
    "noEmitOnError": true,
    "noImplicitAny": false,
    "outDir": "../dist/",
    "rootDir": ".",
    "sourceMap": true,
    "sourceRoot": "/",
    "target": "es5"
  }
}
```



```
{
  "compilerOptions": {
    "declaration": false,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "mapRoot": "",
    "module": "system",
    "moduleResolution": "node",
    "noEmitOnError": true,
    "noImplicitAny": false,
    "outDir": "../dist/",
    "rootDir": ".",
    "sourceMap": true,
    "sourceRoot": "/",
    "target": "es5"
  },
  "compileOnSave": false
}
```

- ## Optimización de espacio en disco

- Para tener varias aplicaciones, podemos **copiar la carpeta main** completa y cambiar los ficheros de **src\app**
- La carpeta **node\_modules** ocupa unos 250Mb, pero es la misma para todas las aplicaciones
- La podemos compartir usando **enlaces simbólicos**

<http://www.howtogeek.com/howto/16226/complete-guide-to-symbolic-links-symlinks-on-windows-or-linux/>

- También la podemos borrar. Se puede regenerar descargando desde internet:

```
npm install
```

- Si compartimos el proyecto en git, esa carpeta se ignora. Tenemos que usar **npm install** al clonar el proyecto.

- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- **Componentes**
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

## Componentes en Angular 2

- 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 en Angular 2

app.component.ts

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

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

}
```

app.component.html

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

Lógica

Vista



## Componentes en Angular 2

app.component.ts

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

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

app.component.html

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

Lógica

Vista

## Componentes en Angular 2

app.component.ts

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

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

Este componente no  
tiene ninguna lógica

Lógica

app.component.html

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

Vista

app.component.ts

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

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

app.component.html

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

index.html

```
<html>
  <head>...</head>
  <body>

    <app>Loading...</app>

    <!-- Scripts and libs -->
  </body>
</html>
```

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

ejem1



## Componentes en Angular 2

app.component.ts

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

@Component({
  selector: 'app',
  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

## Visualización de una variable

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

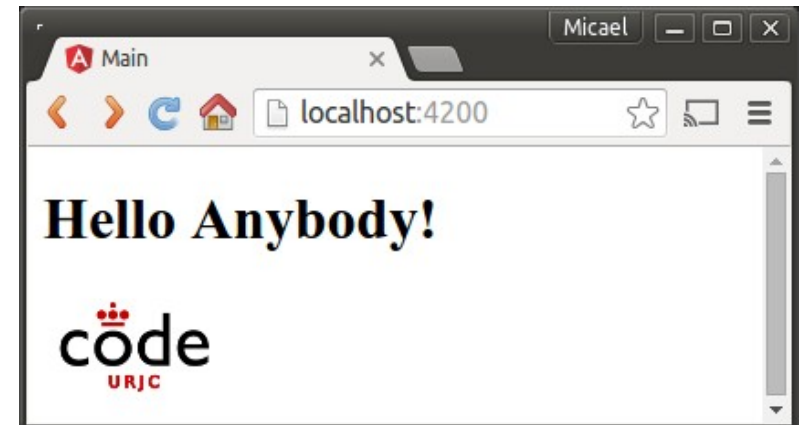
app.component.ts

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

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

app.component.html

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



## Visualización de una variable

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

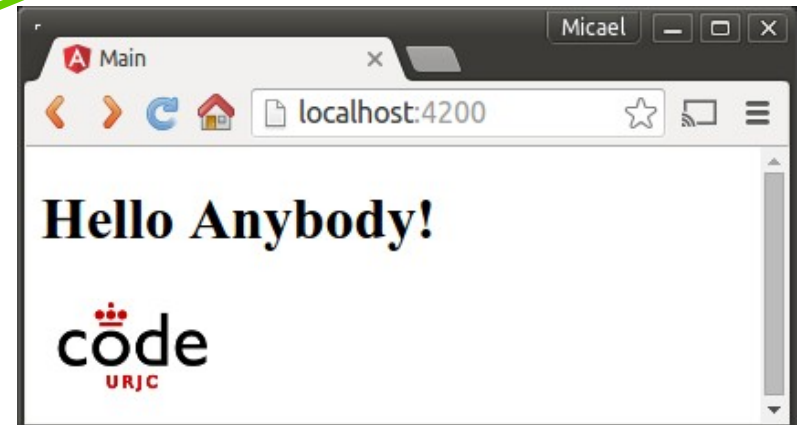
app.component.ts

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

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

app.component.html

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



## Ejecución de lógica

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

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: 'app/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>
```



## Ejecución de lógica

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

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: 'app/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>
```

## Ejecución de lógica

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

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: 'app/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

## Ejecución de lógica

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



## Datos enlazados (*data binding*)

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

app.component.ts

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

@Component({
  selector: 'app',
  templateUrl: 'app/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>
```

## Datos enlazados (*data binding*)

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

app.component.ts

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

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

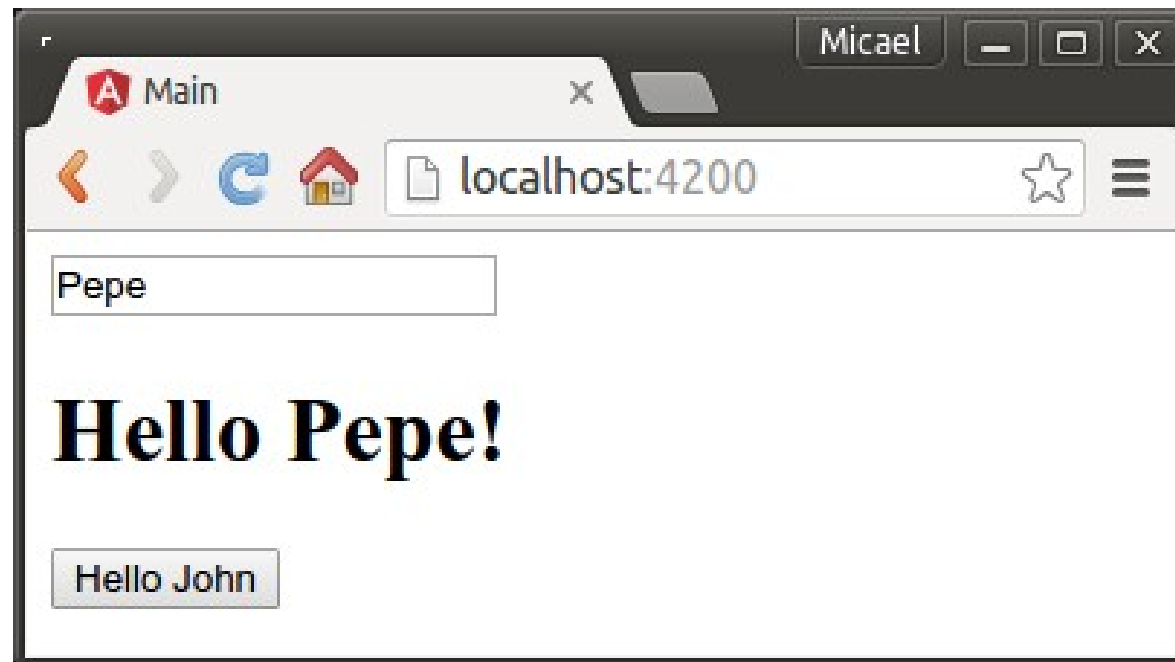
  setName(name:string) {
    this.name = name;
  }
}
```

app.component.html

```
<input type="text" [(ngModel)]="name">
<h1>Hello {{name}}!</h1>
<button (click)="setName('John')">
  Hello John
</button>
```

## Datos enlazados (*data binding*)

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



- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- Componentes
- **Templates**
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

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

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



- **Visualización condicional**

- Se puede controlar si un elemento aparece o no en la página dependiendo del valor de un atributo

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

- También se puede usar una expresión

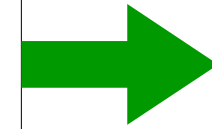
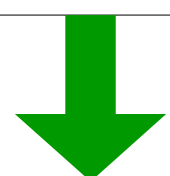
```
<p *ngIf="num == 3">Num 3</p>
```

- Repetición de elementos

- Es posible visualizar el contenido de un array
- Se define cómo se visualizará cada elemento del array

```
<div *ngFor="#e of elems">{{e.desc}} </div>
```

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



```
<div>Elem1</div>  
<div>Elem2</div>  
<div>Elem3</div>
```

- **Estilos**

- Hay muchas formas de controlar los estilos de los elementos
  - Asociar la **clase** de un elemento a un atributo de tipo string
  - Asociar una **clase concreta** a 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

- Asociar la clase de un elemento a un atributo string <sup>ejem5</sup>
  - Cambiando el valor del atributo se cambia la clase
  - El estilo se define en el CSS asociado a la clase

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

- El CSS se puede asociar al fichero index.html (como cualquier HTML)
- Esos estilos son globales y se aplican a todo el documento

- Asociar la clase de un elemento a un atributo string<sup>ejem5</sup>
  - Se pueden definir estilos CSS en el propio componente
  - Se puede usar el atributo **styles** o **styleUrls** de la anotación **@Component**
  - O se pueden incluir en la **plantilla**
  - De esa forma los estilos son locales y no afectan a otros\* componentes del documento


\* Hasta ahora hemos visto un único componente en una página, pero veremos cómo se pueden incluir más

- Asociar la clase de un elemento a un atributo string<sup>ejem5</sup>
  - Atributo styles

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


Se suelen usar los strings  
multilínea con tildes  
invertidas

- Asociar la clase de un elemento a un atributo string <sup>ejem5</sup>
  - Atributo styleUrls



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

- Asociar la clase de un elemento a un atributo string<sup>ejem5</sup>
  - Incluir CSS en el template



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

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

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



- Asociar una clase concreta a un atributo boolean<sup>ejem5</sup>
  - Activa o desactiva una clase en un elemento

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

- Se puede usar para varias clases

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

- 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;  
}
```

- 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>
```

- 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',  
  }  
}
```

- Los **templates** permiten manipular de forma básica los elementos de un formulario
  - Asociar a una variable cada componente
  - Leer y modificar su valor
- Existen mecanismos más **avanzados** usando la clase `ngControl`

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

- **Campo de texto**
  - Se vincula el control a un atributo
  - Cualquier cambio en el control se refleja en la variable (y viceversa)

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

```
name:string
```

- **Checkbox basado en booleanos**
  - Se vincula el control a un atributo
  - Cualquier cambio en el control se refleja en la variable (y viceversa)

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

```
angular:boolean  
javascript:boolean
```

- **Checkbox basado en array**

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

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



- Botones de radio
- Se manipula el componente directamente en el template

```
<input #male name="gender" type="radio"  
value="Male" (click)="gender = male.value"  
[checked]="gender == 'Male'"/> Male
```

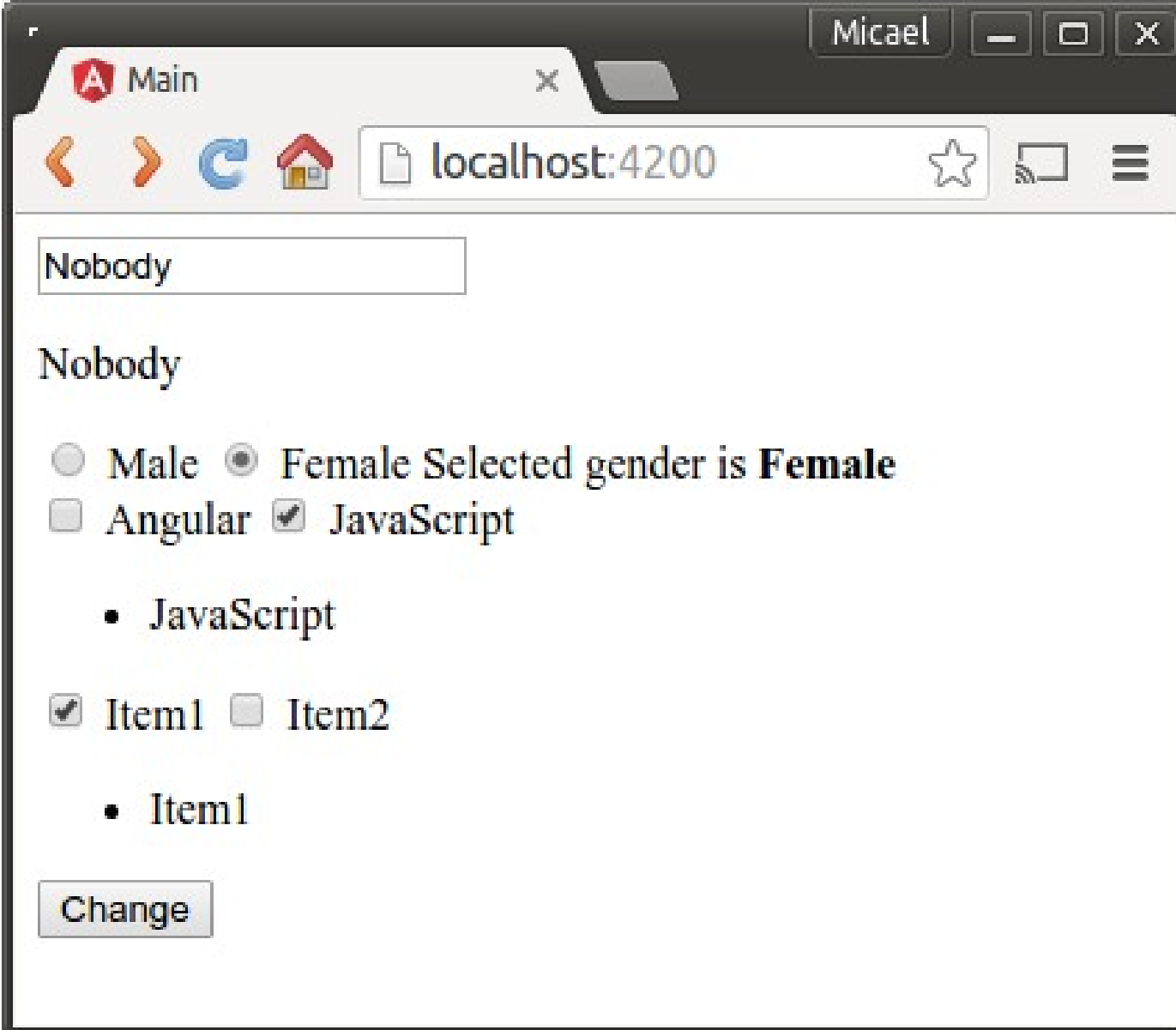
```
<input #female name="gender" type="radio"  
value="Female" (click)="gender = female.value"  
[checked]="gender == 'Female'"/> Female
```

```
gender:string
```

**NOTA:** En el momento de escribir este material(beta7) esta es la mejor forma de manipular un grupo de botones de radio. Es posible que se simplifique en versiones posteriores

# Controles de formulario

ejem6



A screenshot of a web browser window. The address bar shows 'localhost:4200'. The page content includes a text input field with 'Nobody', a label 'Nobody', a radio button group with 'Male' and 'Female' (the latter is selected and followed by the text 'Selected gender is Female'), a checkbox group with 'Angular' and 'JavaScript' (the latter is checked), a bulleted list with 'JavaScript', another checkbox group with 'Item1' (checked) and 'Item2', a third bulleted list with 'Item1', and a 'Change' button at the bottom.

Nobody

Nobody

☐ Male ☒ Female Selected gender is **Female**

☐ Angular ☒ JavaScript

- JavaScript

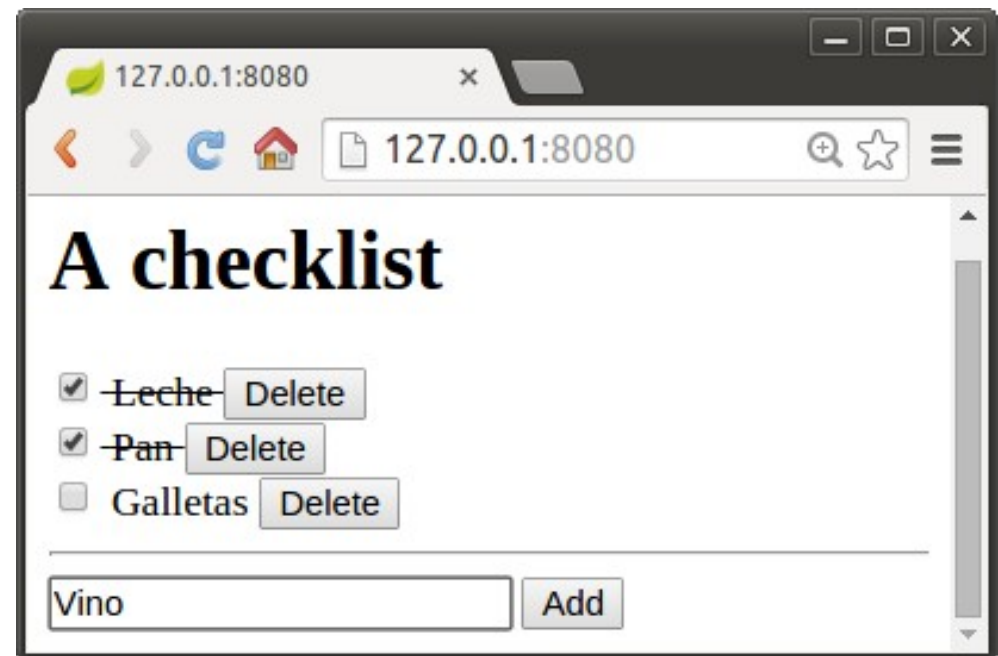
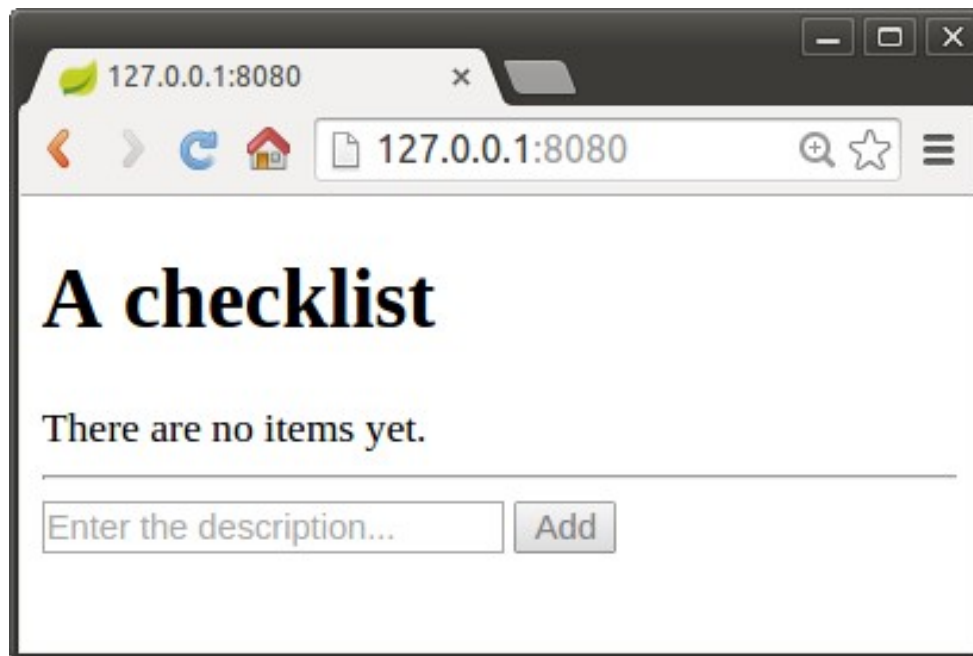
☒ Item1 ☐ Item2

- Item1

Change

# Ejercicio 1

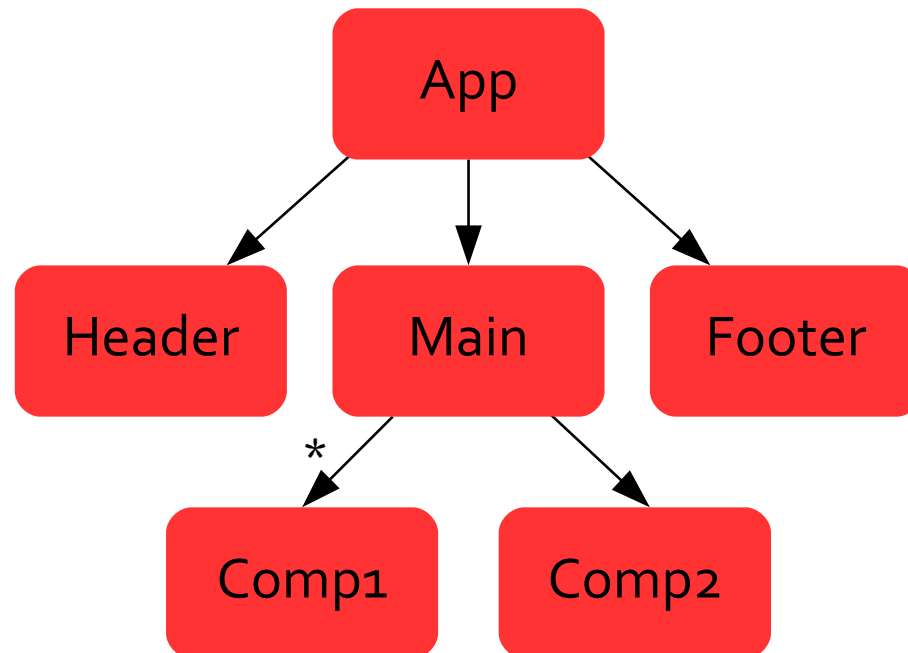
- Implementa una aplicación de **gestión de tareas**
- Las tareas se mantendrán en **memoria**



- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- **Composición de componentes**
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

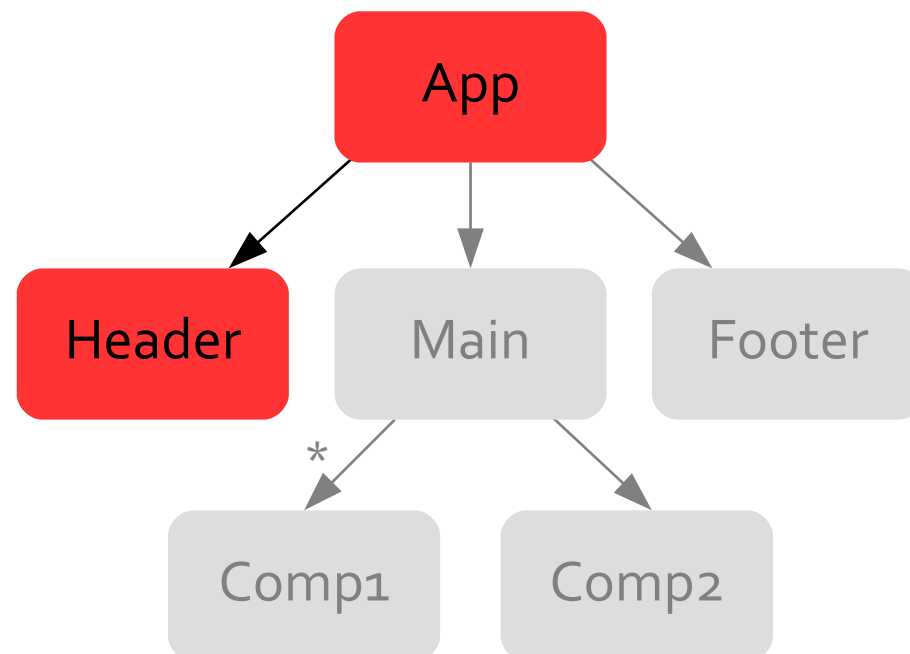
## Árboles de componentes

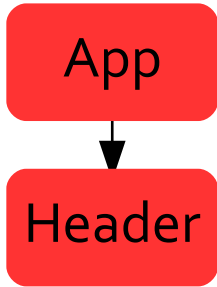
En Angular 2 un componente puede estar formado por más componentes formando un árbol



## Árboles de componentes

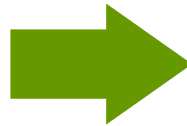
En Angular 2 un componente puede estar formado por más componentes formando un árbol





## Árboles de componentes

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



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

```
<header>
```

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

## Árboles de componentes

App



Header

app.component.ts

```
import {Component} from 'angular2/core';
import {HeaderComponent} from
  './header.component';

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

app.component.html

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

header.component.ts

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

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

header.component.html

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



# Composición de componentes

ejem7

## Árboles de componentes

App

Header

app.component.ts

```
import {Component} from 'angular2/core';
import {HeaderComponent} from
  './header.component';

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

header.component.ts

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

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

app.component.html

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

header.component.html

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

Hay que indicar los  
componentes que se  
usan en el template

# Composición de componentes

## Árboles de componentes

App

Header

app.component.ts

```
import {Component} from 'angular2/core';
import {HeaderComponent} from
  './header.component';

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

header.component.ts

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

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

app.component.html

En TypeScript es necesario importar las clases de otro Módulo (aunque esté en la misma carpeta)

```
<header>
<p>M
```

header.component.html

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

## Árboles de componentes

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

```
▼ <body>
  ▼ <app>
    ▼ <header>
      <h1>Title</h1>
    </header>
    ▼ <p>
      "Main content"
    </p>
  </app>
```

- 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)

## Configuración de propiedades (Padre → Hijo)

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

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

## Configuración de propiedades (Padre → Hijo)

app.component.ts

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

header.component.ts

```
import {Component, Input} from  
  'angular2/core';  
...  
export class HeaderComponent {  
  
  @Input()  
  private title: string;  
}
```

app.component.html

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

header.component.html

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

## 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  
  'angular2/core';  
...  
export class HeaderComponent {  
  @Input()  
  private title: string;  
}
```

header.component.html

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

## Configuración de propiedades (Padre → Hijo)

app.component.ts

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

header.component.ts

```
import {Component, Input} from  
  'angular2/core';  
...  
export class HeaderComponent {  
  
  @Input()  
  private title: string;  
}
```

app.component.html

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

header.component.html

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



## Envío de eventos (Hijo → Padre)

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

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

## Envío de eventos (Hijo → Padre)

`app.component.ts`

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

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

## 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>
```

## Envío de eventos (Hijo → Padre)

app.component.ts

```
...  
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>
```

## Envío de eventos (Hijo → Padre)

header.component.ts

```
import {Component, Output, EventEmitter} from 'angular2/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>
```

## Envío de eventos (Hijo → Padre)

header.component.ts

```
import {Component, Output, EventEmitter} from 'angular2/core';  
...  
export class HeaderComponent {  
  @Output()  
  hidden = new EventEmitter<boolean>();  
  
  visible = true;  
  
  click() {  
    this.visible = !this.visible;  
    this.hidden.next(this.visible);  
  }  
}
```

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

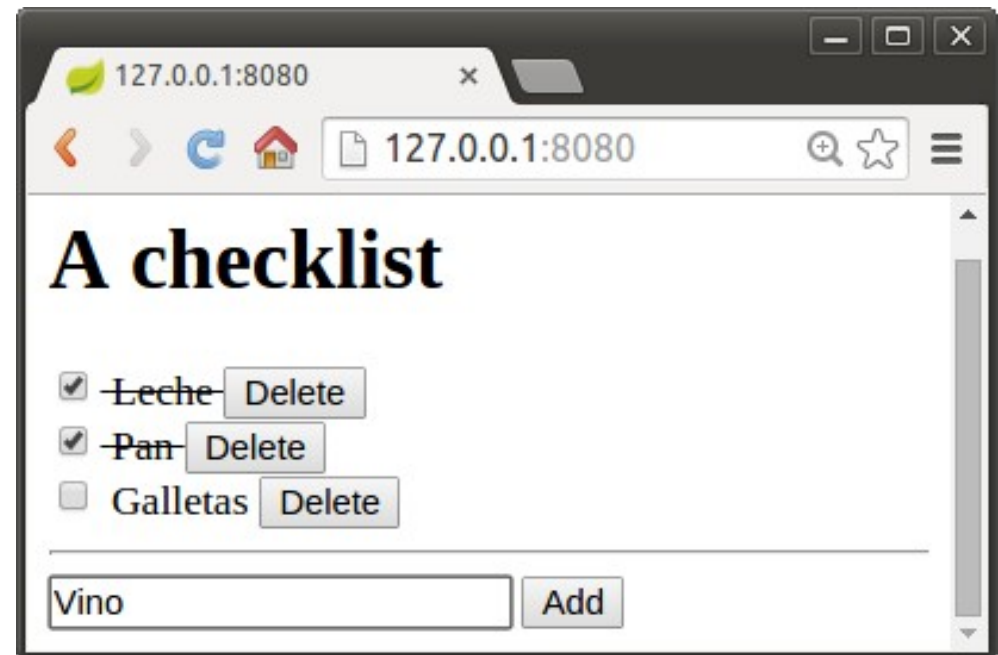
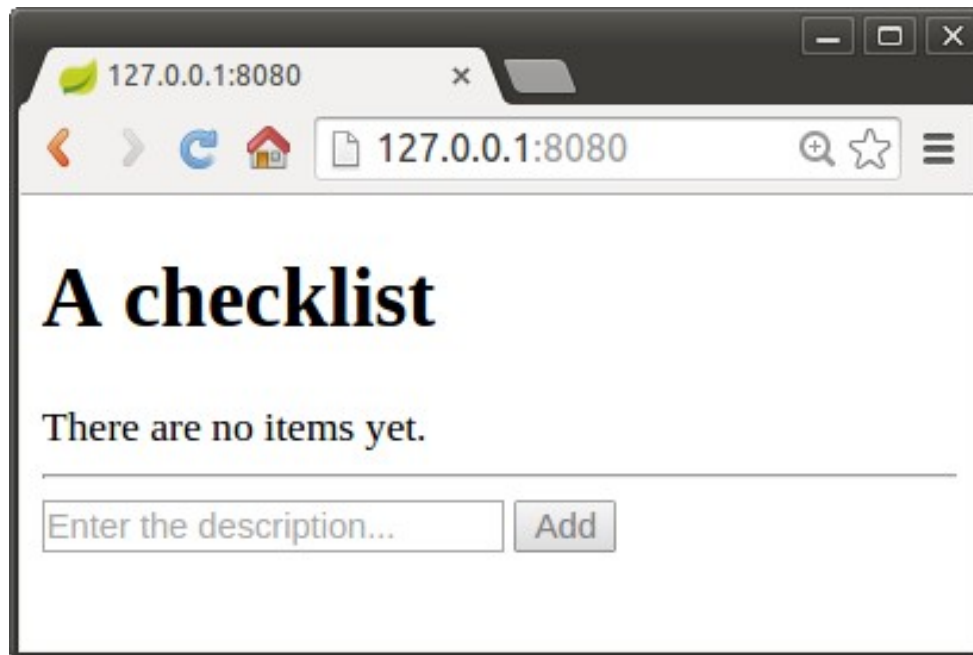
Para lanzar un evento se invoca el método **next(valor)**

header.component.html

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

# Ejercicio 2

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



- ¿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



- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- **Inyección de dependencias y servicios**
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

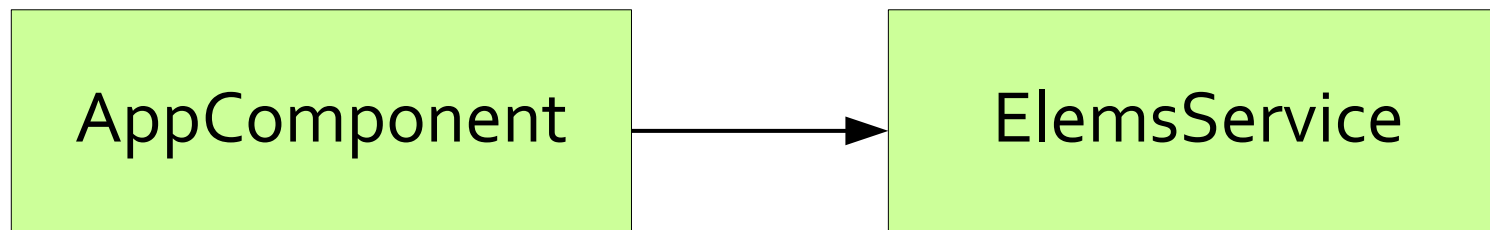
# Inyección de dependencias y servicios

- La inyección de dependencias es una técnica muy usada para **estructurar una aplicación**
- Los componentes definen sus **dependencias** y el framework se encarga de instanciar esas **dependencias** e **inyectarlas** donde se necesiten
- Esta técnica se ha hecho muy popular en el desarrollo de *back-end* en frameworks como **Spring** o **Java EE**

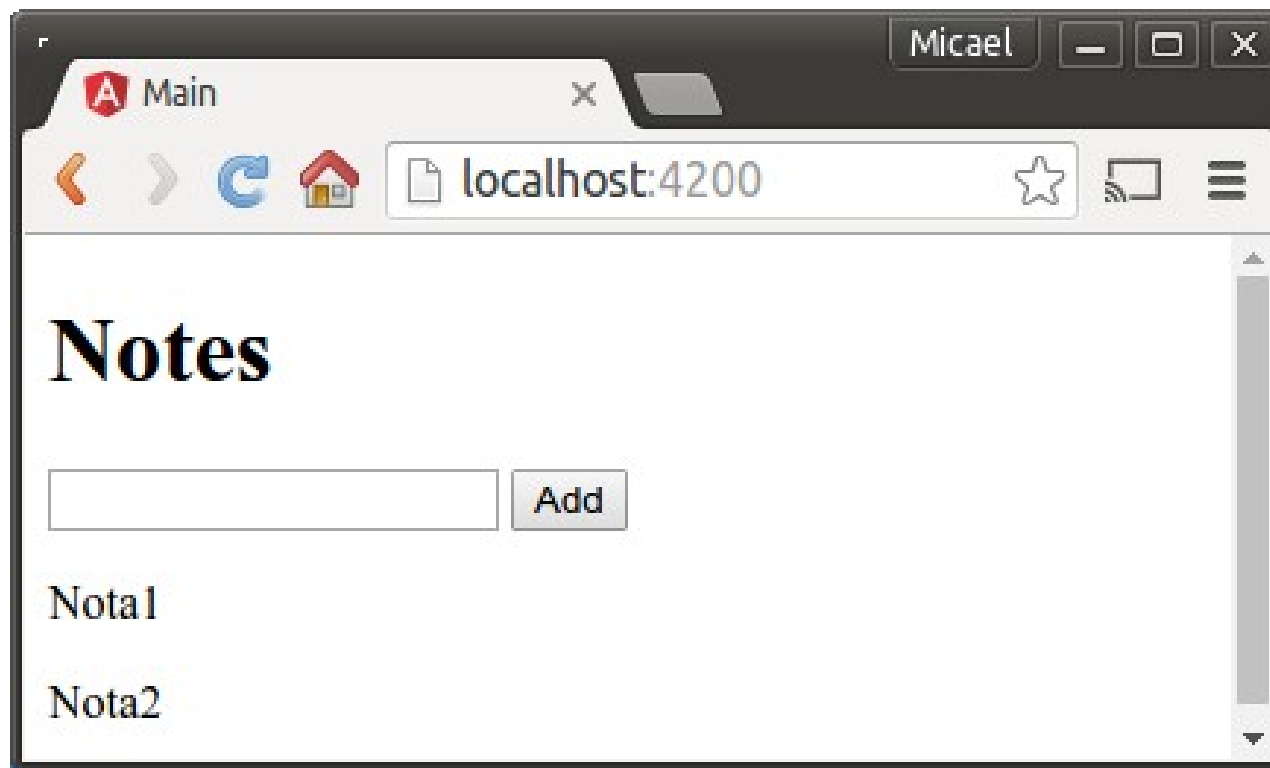
<https://angular.io/docs/ts/latest/guide/dependency-injection.html>

# Inyección de dependencias y servicios

- En **Angular2** se pueden inyectar servicios proporcionados por el *framework* o **desarrollados en la aplicación**
- Es una buena práctica que los **componentes** gestionen los datos usando un **servicio**
- La inyección de dependencias favorece la **modularidad** de la aplicación y el **testing**



## Gestor de notas sin usar servicios



## Gestor de notas sin usar servicios

app.component.html

```
<h1>Notes</h1>

<input #text type="text">
<button (click)="add(text.value); text.value = ' '>Add
</button>

<p *ngFor="#elem of elems">{{elem}}</p>
```

app.component.ts

```
...
export class AppComponent {

  elems: string[] = []

  add(elem: string){ this.elems.push(elem); }
}
```

## ¿Cómo se implementa un servicio?

elems.service.ts

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

@Injectable()
export class ElemsService {

  public elems: string[] = []

  add(elem: string) {
    this.elems.push(elem);
  }
}
```

## ¿Cómo se implementa un servicio?

elems.service.ts

```
import {Injectable} from 'angular2/core';  
  
@Injectable()  
export class ElemsService {  
  
    public elems: string[] = []  
  
    add(elem: string) {  
        this.elems.push(elem);  
    }  
}
```

## ¿Cómo se implementa un servicio?

app.component.ts

```
import {Component} from 'angular2/core';
import {ElemsService} from './elems.service';

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

  private elems: string[];

  constructor(private elemsService : ElemsService){
    this.elems = elemsService.elems;
  }
  add(elem: string){
    this.elemsService.add(elem);
  }
}
```



## ¿Cómo se implementa un servicio?

app.component.ts

```
import {Component} from 'angular2/core';
import {ElemsService} from './elems.service';

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

  private elems: string[];

  constructor(private elemsService : ElemsService){
    this.elems = elemsService.elems;
  }
  add(elem: string){
    this.elemsService.add(elem);
  }
}
```

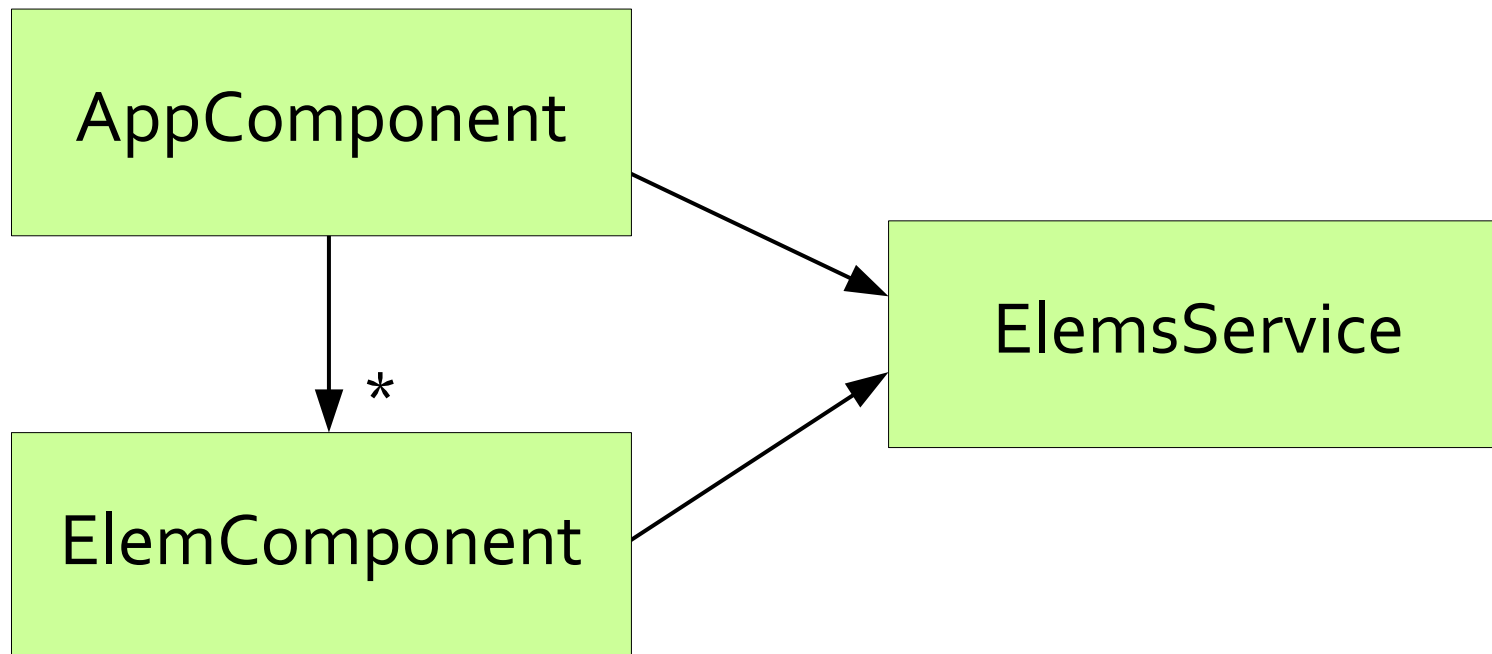
## ¿Cómo se implementa un servicio?

- Implementar la clase del servicio:  
**ElmsService**
- Anotar la clase con: **@Injectable**
- Dar de alta el servicio en el **@Component** en el array de **providers**
- Poner un parámetro en el constructor del componente para que se inyecte el servicio

## Compartir servicios entre componentes

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

## Compartir servicios entre componentes



## Compartir servicios entre componentes

- Los servicios definidos en un componente (**providers**) pueden **usarse** en los componentes hijos
- Para que un componente **hijo comparta el mismo servicio** que el padre basta con que lo declare en el **constructor**
- El componente hijo **no** tiene que incluir el servicio en los **providers** para que utilice el del padre

## Compartir servicios entre componentes

elem.component.ts

```
import {Component, Input} from 'angular2/core';
import {ElemsService} from './elems.service';

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

  @Input() elem:string;

  constructor(private elemsService : ElemsService){}

  remove() {
    this.elemsService.remove(this.elem);
  }
}
```

## Compartir servicios entre componentes

elem.component.ts

```
import {Component, Input} from 'angular2/core';
import {ElemsService} from './elems.service';

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

  @Input() elem:string;

  constructor(private elemsService : ElemsService){}

  remove() {
    this.elemsService.remove(this.elem);
  }
}
```

## Compartir servicios entre componentes

app.component.ts

```
import {Component} from 'angular2/core';
import {ElemsService} from './elems.service';
import {ElemComponent} from './elem.component';

@Component({
  selector: 'app',
  templateUrl: 'app/app.component.html',
  providers: [ElemsService],
  directives: [ElemComponent]
})
export class AppComponent {
  private elems: string[];

  constructor(private elemsService : ElemsService){
    this.elems = elemsService.elems;
  }
  add(elem: string){
    this.elemsService.add(elem);
  }
}
```



## Compartir servicios entre componentes

app.component.ts

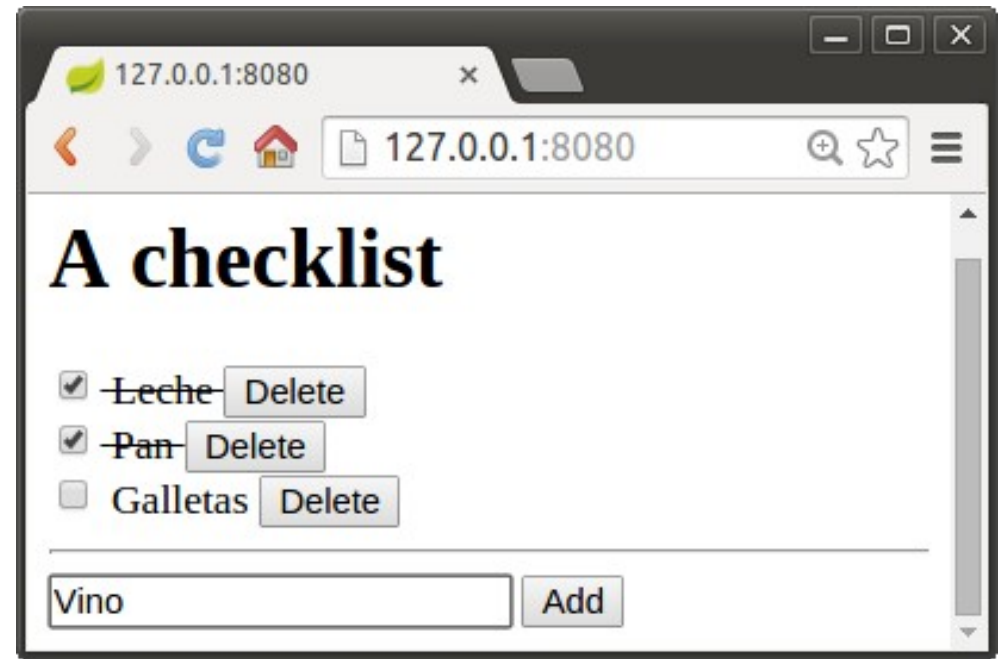
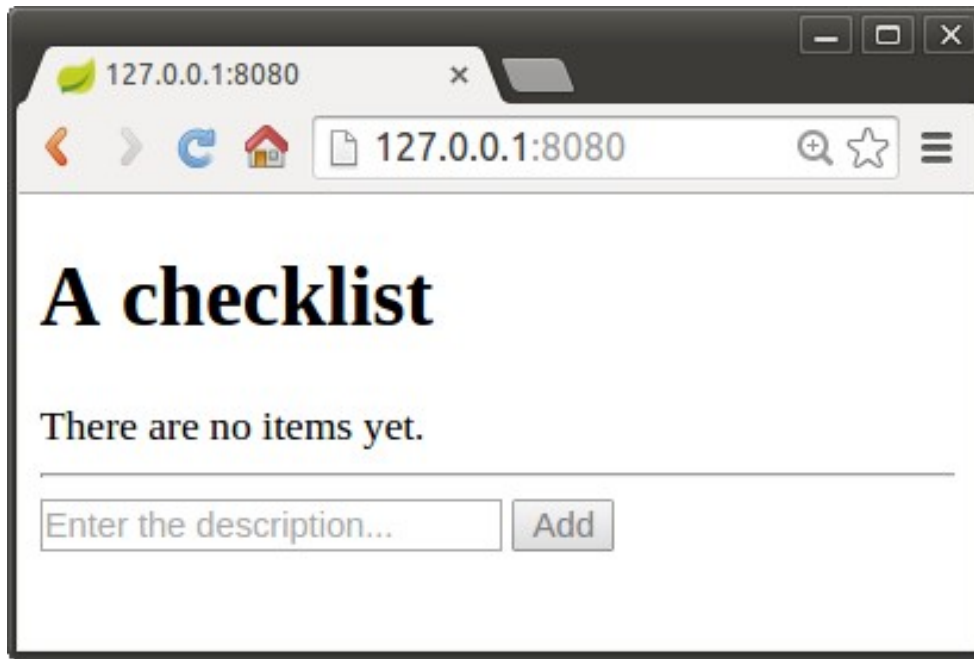
```
import {Component} from 'angular2/core';
import {ElmsService} from './elems.service';
import {ElemComponent} from './elem.component';

@Component({
  selector: 'app',
  templateUrl: 'app/app.component.html',
  providers: [ElmsService],
  directives: [ElemComponent]
})
export class AppComponent {
  private elems: string[];

  constructor(private elemsService : ElmsService){
    this.elems = elemsService.elems;
  }
  add(elem: string){
    this.elemsService.add(elem);
  }
}
```

# Ejercicio 3

- Refactoriza la aplicación de **gestión de tareas** para que las tareas sean gestionadas en un **servicio inyectado** en los componentes



- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- **Cliente REST**
- Aplicaciones multipágina: Router
- Librerías de componentes
- Conclusiones

- 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/Http-class.html>

- 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 respuesta 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/Http-class.html>

- 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** en el objeto **response**

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

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

```
import {Component} from 'angular2/core';
import {HTTP_PROVIDERS, Http} from 'angular2/http';

@Component({
  ...
  providers: [HTTP_PROVIDERS]
})
export class AppComponent {

  constructor(private http: Http) {}

  search(title: string) {
    //Usamos el objeto http
  }
}
```

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

```
import {Component} from 'angular2/core';
import {HTTP_PROVIDERS, Http} from 'angular2/http';

@Component({
  ...
  providers: [HTTP_PROVIDERS]
})
export class AppComponent {

  constructor(private http: Http) {}

  search(title: string) {
    //Usamos el objeto http
  }
}
```

**HTTP\_PROVIDERS** es un array con varios providers (entre ellos **Http**)



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



- 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="#book of books">{{book}}</p>
```

```
import {Component} from 'angular2/core';
import {HTTP_PROVIDERS, Http} from 'angular2/http';

@Component({
  selector: 'app',
  templateUrl: 'app/app.component.html',
  providers: [HTTP_PROVIDERS]
})
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 = response.json();
        for (var i = 0; i < data.items.length; i++) {
          let bookTitle = data.items[i].volumeInfo.title;
          this.books.push(bookTitle);
        }
      },
      error => console.error(error)
    );
  }
}
```

- Peticiones **POST**

```
let data = ...
let url = ...

let body = JSON.stringify(data);
let headers = new Headers({
    'Content-Type': 'application/json'
});
let options = new RequestOptions({headers});

this.http.post(url, body, options).subscribe(
    response => console.log(response),
    error => console.error(error)
);
```

## 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 servicio
  - Es más fácil de **testear**
  - Es una **buena práctica** (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**

## Peticiones http en un servicio

- ¿Cómo se **implementan** los métodos de ese servicio?

```
let service: GoogleBooksService = ...  
let books = service.getBooks(title) ;  
console.log(books) ;
```

Un servicio que hace peticiones de red **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

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

## Peticiones http en un servicio

- **Promesas:** El método devuelve un objeto **Promise**. Con el método **then** se definen las funciones que serán ejecutadas cuando llegue el resultado (correcto o erróneo)

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

```
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 2 por ser la más completa

## Servicio con Observables de RxJS

- **RxJS**: Extensiones **reactivas** para JavaScript
- La librería **RxJS** está incluida en Angular2
- 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



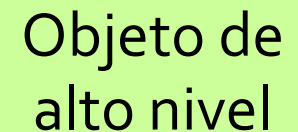
<https://github.com/ReactiveX/RxJS>

## Servicio con Observables de RxJS

- 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

```
...
import 'rxjs/Rx';

export class GoogleBooksService {
  ...
  getBooks(title: string) {
    let url = ...
    return this.http.get(url).map(
      response => this.extractTitles(response)
    )
  }
  private extractTitles(response: Response) {...}
}
```

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

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

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

## Servicio con Observables de RxJS

```
import 'rxjs/Rx';
```

```
export class GoogleBooksService {  
  ...  
  getBooks(title: string) {  
    let url = ...  
    return this.http.get(url).map(  
      response => this.extractTitles(response)  
    )  
  }  
  private extractTitles(response: Response) {...}  
}
```

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

Para poder usar el método **map** es necesario **importar** la librería **rxjs/Rx**



# Cliente REST


googleservice.service.ts

```
import {Injectable} from 'angular2/core';  
import {Http, Response} from 'angular2/http';  
import 'rxjs/Rx';
```

ejem14

```
@Injectable()  
export class GoogleBooksService {  
  
  constructor(private http: Http) { }  
  
  getBooks(title: string) {  
  
    let url = "https://www.googleapis.com/books/v1/volumes?q=intitle:" + title;  
  
    return this.http.get(url).map(  
      response => this.extractTitles(response)  
    )  
  }  
  
  private extractTitles(response: Response) {  
    return response.json().items.map( book => book.volumeInfo.title)  
  }  
}
```

Método que extrae los  
títulos de la respuesta a  
la API REST



# Cliente REST

app.component.ts

```
import {Component} from 'angular2/core';
import {HTTP_PROVIDERS, Http} from 'angular2/http';
import {GoogleBooksService} from '../googlebooks.service';

@Component({
  selector: 'app',
  templateUrl: 'app/app.component.html',
  providers: [HTTP_PROVIDERS, GoogleBooksService]
})
export class AppComponent {

  private books: string[] = [];

  constructor(private http: Http, private service: GoogleBooksService) {}

  search(title: string) {

    this.books = [];
    this.service.getBooks(title).subscribe(
      books => this.books = books,
      error => console.error(error)
    );
  }
}
```

ejem14


Cuando llega la respuesta se actualiza el array de books

## 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.http.get(url)  
    .map(response => this.extractTitles(response))  
    .catch(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: **GoogleBooksService**
- **Servicios statefull (con estado)**
  - Mantiene estado, guardan información
  - Sus métodos pueden devolver valores y cambian el estado interno del servicio
  - Ejemplo: **ElmsService**

## Estado en los servicios http

- **Dependiendo de cada caso es mejor uno u otro**
- **Cuándo usar servicios stateless**
  - El componente requiere un mayor control sobre las peticiones REST
  - El componente muestra información de carga, decide cuándo actualizar los datos desde el servidor, etc.
- **Cuándo usar servicio statefull**
  - El componente es más sencillo
  - El servicio se encarga de la gestión de datos y ofrece información de alto nivel para que sea visualizada por el componente (estado de carga, errores, etc)

## Estado en los servicios http

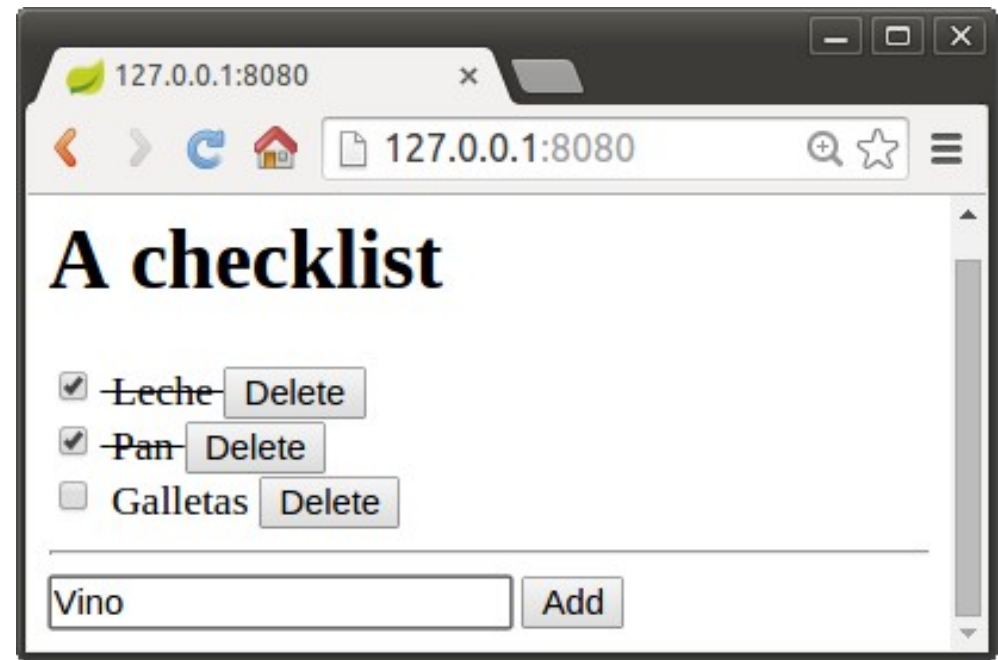
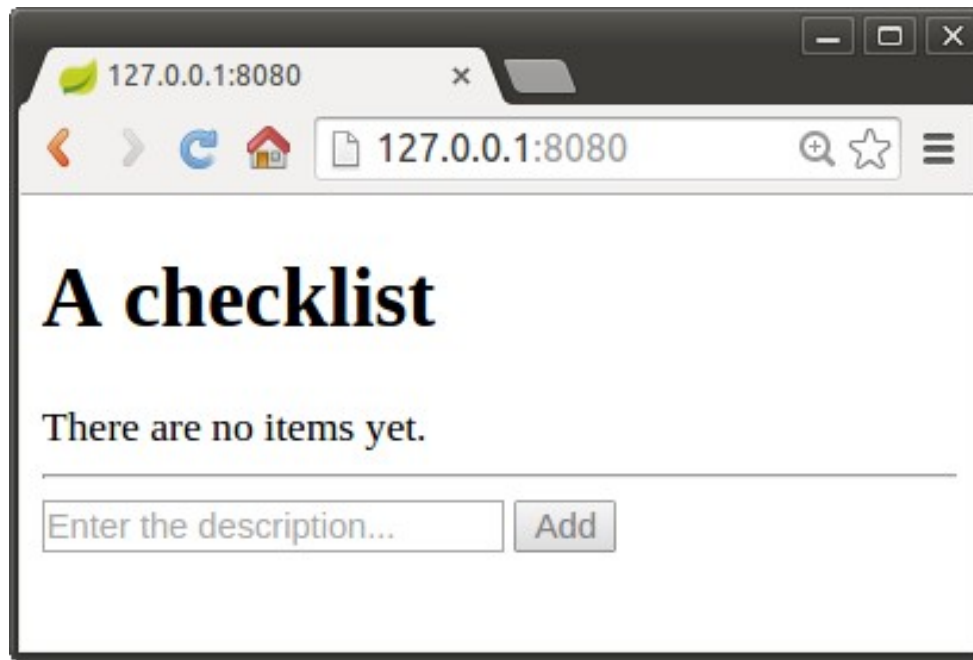
- Existen **muchas formas** de implementar un servicio con **estado** que se basa en una API REST
- **Una estrategia básica:**
  - Un método en el servicio permite al componente obtener un **array con los elementos del servidor**
  - El componente asigna ese array a un **atributo** y lo **muestra** usando la vista (*template*)
  - Los métodos del servicio que invocan una operación REST **mantienen actualizado** ese array (no crean un nuevo) para que la vista se actualice de forma **automática**

## Estado en los servicios http

- **Limitaciones** de la estrategia básica:
  - El cliente del servicio no sabe cuándo acaba la carga (cuándo llega el resultado de la petición)
  - No puede poner un indicador de carga mientras tanto
  - Tampoco puede detectar cuándo se ha producido un error
- **Mejoras** de la estrategia básica:
  - Cada método que hace una petición REST devuelve un observable para el control de la petición desde el cliente

# 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
- Está implementada en Java 8
- 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)
```

- **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)

- **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)

- **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)

- **API REST Items**

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

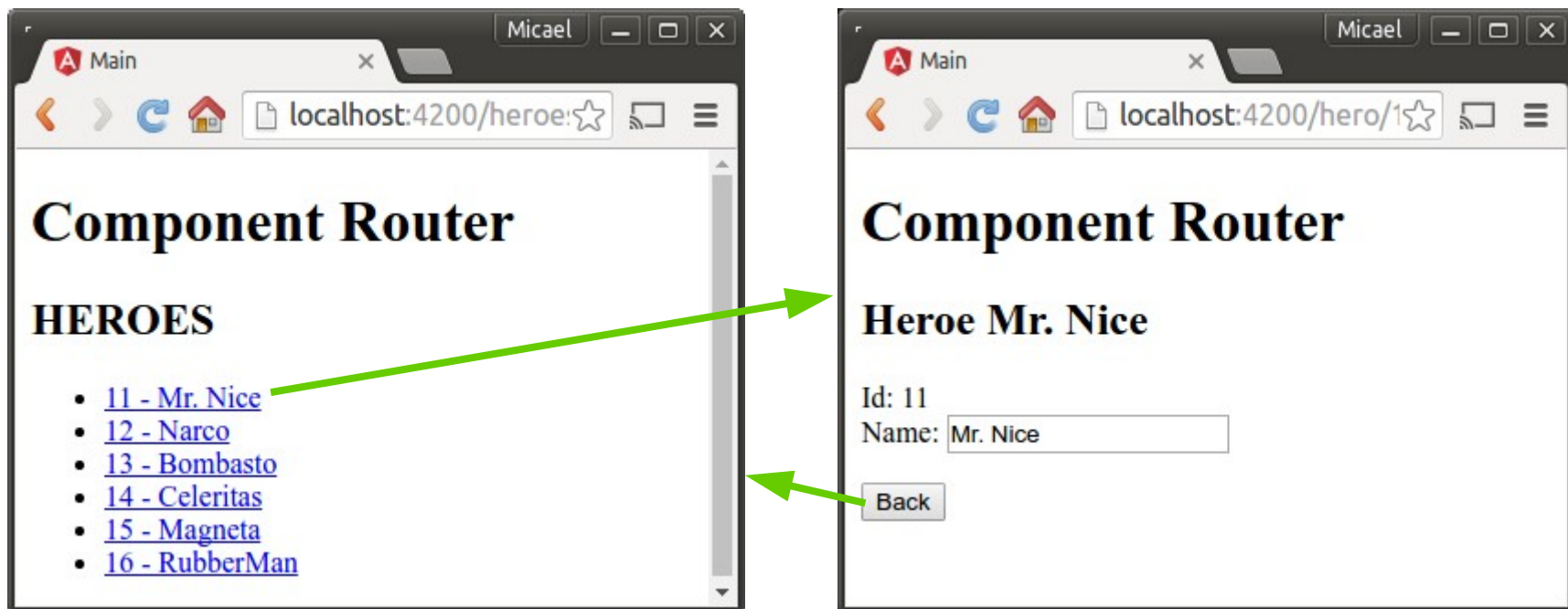
```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Status code: 200 (OK)

- **Introducción a Angular 2**
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- **Aplicaciones multipágina: Router**
- Librerías de componentes
- Conclusiones

# 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>

- El **componente principal** de la aplicación (**app**) puede tener una **zona** cuyo **contenido** depende de la **URL** (`<router-outlet>`)
- En el componente principal se define qué componente se muestra para cada URL (`@RouterConfig`)
- Existen **links especiales** para navegar dentro de web (`[routerLink]`)
- Desde el código se puede navegar (**Router**)

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



## Componente principal

app.component.ts

```
...
import {RouteConfig, ROUTER_DIRECTIVES} from 'angular2/router';

@Component({
  selector: 'app',
  template: `
    <h1 class="title">Component Router</h1>
    <router-outlet></router-outlet>
  `,
  providers: [HeroService],
  directives: [ROUTER_DIRECTIVES]
})

@RouteConfig([
  {path: '/heroes', name: 'Heroes', component: HeroListComponent, useAsDefault: true},
  {path: '/hero/:id', name: 'HeroDetail', component: HeroDetailComponent},
])
export class AppComponent {}
```

## Componente principal

app.component.ts

```
...
import {RouteConfig, ROUTER_DIRECTIVES} from 'angular2/router';

@Component({
  selector: 'app',
  template: `
    <h1 class="title">Component Router</h1>
    <router-outlet></router-outlet>
  `,
  providers: [HeroService],
  directives: [ROUTER_DIRECTIVES]
})

@RouteConfig([
  {path: '/heroes', name: 'Heroes', component: HeroListComponent, useAsDefault: true},
  {path: '/hero/:id', name: 'HeroDetail', component: HeroDetailComponent},
])
export class AppComponent {}
```

Zona que cambia en función de la URL

## Componente principal

app.component.ts

```
...
import {RouteConfig, ROUTER_DIRECTIVES} from 'angular2/router';

@Component({
  selector: 'app',
  template: `
    <h1 class="title">Component Router</h1>
    <router-outlet></router-outlet>
  `,
  providers: [HeroService],
  directives: [ROUTER_DIRECTIVES]
})

@RouteConfig([
  {path: '/heroes', name: 'Heroes', component: HeroListComponent, useAsDefault: true},
  {path: '/hero/:id', name: 'HeroDetail', component: HeroDetailComponent},
])
export class AppComponent {}
```

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

## Componente principal

app.component.ts

```
...  
import {RouteConfig, ROUTER_DIRECTIVES} from 'angular2/router';
```

```
@Component({  
  selector: 'app',  
  templateUrl: 'app.component.html',  
  providers: [Router],  
  directives: [ROUTER_DIRECTIVES],  
})
```

Hay rutas que pueden llevar **parámetros** (que podrán ser leídos por el componente)

Se puede indicar una **URL** por **defecto** (a la que se navega de forma **automática**)

```
@RouteConfig([  
  {path: '/heroes', name: 'Heroes', component: HeroListComponent, useAsDefault: true},  
  {path: '/hero/:id', name: 'HeroDetail', component: HeroDetailComponent},  
)  
export class AppComponent {}
```

# Aplicaciones multipágina: Router

## Componente HeroList

ejem16

hero-list.component.ts

```
...
import {ROUTER_DIRECTIVES} from 'angular2/router';

@Component({
  directives: [ROUTER_DIRECTIVES],
  template: `
    <h2>HEROES</h2>
    <ul class="items">
      <li *ngFor="#hero of heroes">
        <a [routerLink]="['HeroDetail',{id:hero.id}]">
          {{hero.id}} - {{hero.name}}</a>
        </li>
      </ul>`
})
export class HeroListComponent {
  heroes: Hero[];
  constructor(service: HeroService) {
    this.heroes = service.getHeroes();
  }
}
```

# Aplicaciones multipágina: Router

## Componente HeroList

ejem16

hero-list.component.ts

```
...
import {ROUTER_DIRECTIVES} from 'angular2/router';

@Component({
  directives: [ROUTER_DIRECTIVES],
  template: `
    <h2>HEROES</h2>
    <ul class="items">
      <li *ngFor="#hero of heroes">
        <a [routerLink]="['HeroDetail',{id:hero.id}]">
          {{hero.id}} - {{hero.name}}</a>
        </li>
      </ul>
    `
})
export class HeroListComponent {
  heroes: Hero[];
  constructor(service: HeroService) {
    this.heroes = service.getHeroes();
  }
}
```

En vez de href, los links usan **[routerLink]**. Se indica un array con la primera posición el **nombre** y la segunda un objeto con **parámetros**

# Aplicaciones multipágina: Router

## Componente HeroDetail

ejem16

hero-detail.component.ts

```
...
import {RouteParams, Router} from 'angular2/router';

@Component({
  template: `
    <h2>Hero {{hero.name}}</h2>
    ...
    <button (click)="gotoHeroes()">Back</button>`
})
export class HeroDetailComponent {
  hero: Hero;
  constructor(private _router:Router,
    routeParams:RouteParams, service: HeroService){
    let id = routeParams.get('id');
    this.hero = service.getHero(id);
  }
  gotoHeroes() {
    this._router.navigate(['Heroes']);
  }
}
```

# Aplicaciones multipágina: Router

## Componente HeroDetail

ejem16

hero-detail.component.ts

```
...
import {RouteParams, Router} from 'angular2/router';

@Component({
  template: `
    <h2>Hero {{hero.name}}</h2>
    ...
    <button (click)="gotoHeroes()">Ba
  })
export class HeroDetailComponent {
  hero: Hero;
  constructor(private _router:Router,
    routeParams:RouteParams, service: HeroService) {
    let id = routeParams.get('id');
    this.hero = service.getHero(id);
  }
  gotoHeroes() {
    this._router.navigate(['Heroes']);
  }
}
```

Para acceder a los parámetros desde el componente usamos la dependencia **RouteParams**



# Aplicaciones multipágina: Router

## Componente HeroDetail

ejem16

hero-detail.component.ts

```
...
import {RouteParams, Router} from 'angular2/router';

@Component({
  template: `
    <h2>Hero {{hero.name}}</h2>
    ...
    <button (click)="gotoHeroes()">Back</button>`
})
export class HeroDetailComponent {
  hero: Hero;
  constructor(private _router:Router,
    routeParams:RouteParams, service:
    let id = routeParams.get('id');
    this.hero = service.getHero(id);
  }
  gotoHeroes() {
    this._router.navigate(['Heroes']);
  }
}
```

Para navegar desde código usamos la dependencia **Router** y el método **navigate**

- **Funcionalidades avanzadas**
  - **Rutas en componentes hijos**
    - Un componente puede tener su propia configuración de rutas (@RouteConfig), no sólo la app
    - Ese componente (y sus rutas) son reutilizables en diferentes contextos
  - **Ejecutar código al cambiar de pantalla**
    - Si el usuario navega a otra página “sin guardar” se le puede preguntar si realmente desea descargar los cambios o abortar la navegación
  - **Redirecciones**
  - **Animaciones**

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

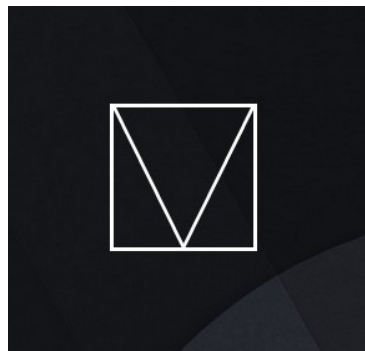
- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- **Librerías de componentes**
- Conclusiones

# Librerías de componentes

- Angular 2 **no proporciona** componentes de alto nivel, usa HTML y CSS
- Se pueden usar cualquier librería de componentes CSS: Bootstrap, Semantic ui, Google Material Design Lite...



<http://getbootstrap.com/>



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



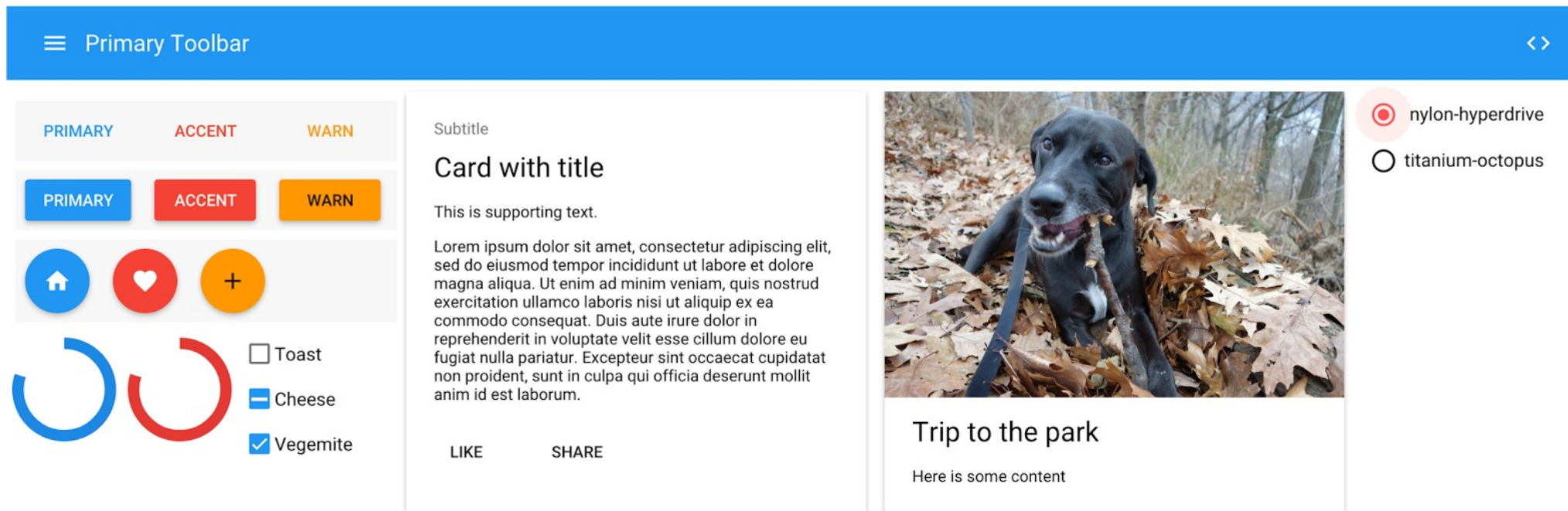
<http://semantic-ui.com/>

- **No se recomienda** usar directamente librerías gráficas JavaScript con Angular 2:
- **JQuery:** Es mejor modificar el DOM con plantillas u otros mecanismos avanzados de Angular2
- **JavaScript de Bootstrap:** No se recomienda usar directamente. Es mejor usar **ng2-bootstrap** [1], componentes bootstrap adaptados a Angular2
- **Otras librerías:** Es mejor usar aquellas con diseñadas para Angular 2 o con adaptadores para Angular2 para evitar problemas de rendimiento y en la construcción de la app

[1] <http://valor-software.com/ng2-bootstrap/>

# Librerías de componentes

- Material Design Angular 2: Librería de componentes



# Librerías de componentes

- ag-grid: Tabla con controles avanzados

Type text to filter...

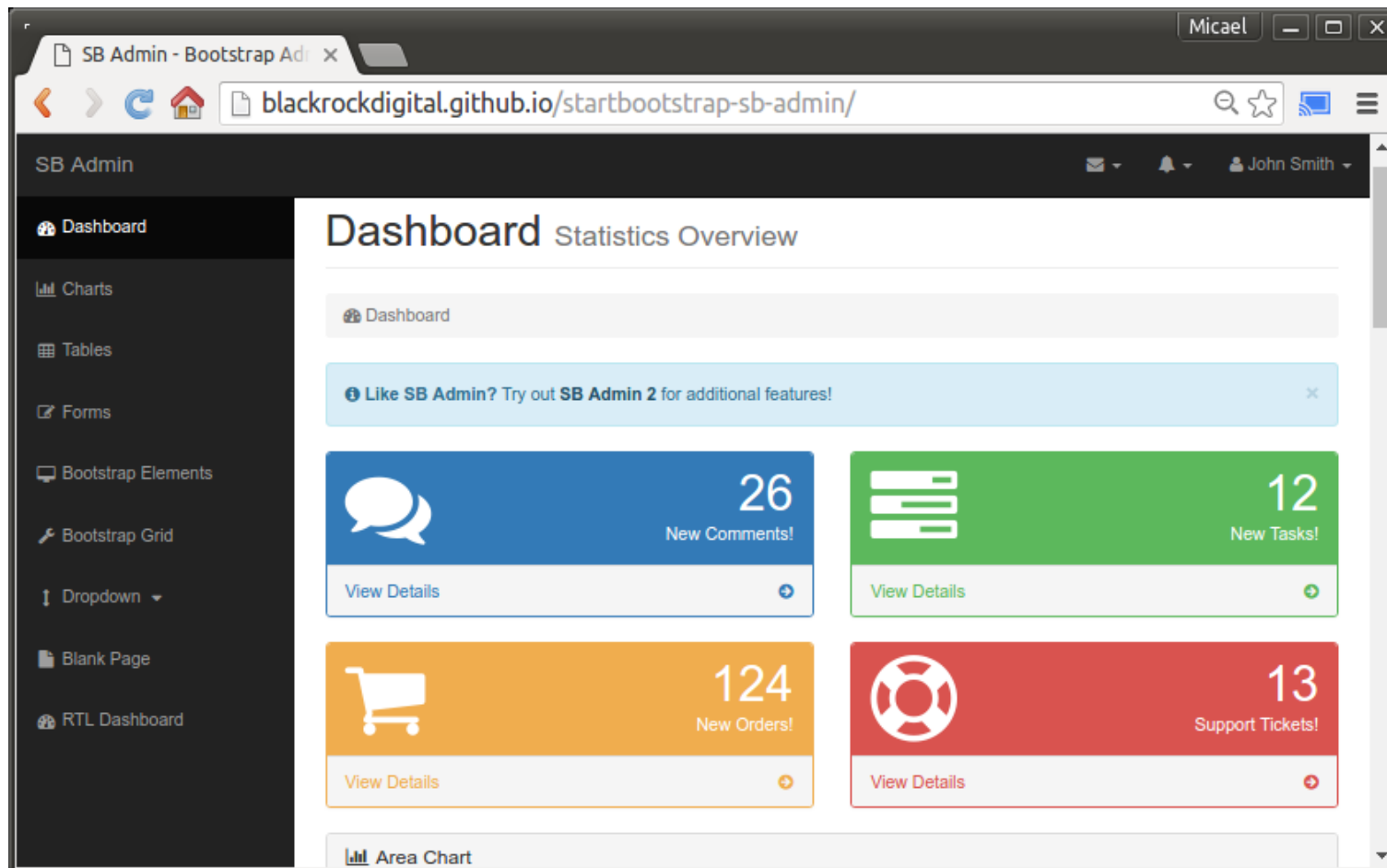
Example Dataset Showing 10.000 / 10.000 rows(s)

#	Employee		IT Skills		Contact		
	Name	▼ Country	Skills	Proficiency	Mobile	Land-line	
<input type="checkbox"/>	Jessica Kade	Uruguay		76%	+873 117 227 477	+141 627 522 455	4819 Honey Treasure Park, A
<input type="checkbox"/>	Chloe Brennan	Uruguay		86%	+333 840 805 1062	+464 546 279 888	5313 Clear Willow Route, Am
<input type="checkbox"/>	Mia Corbin	Uruguay		61%	+224 1092 327 055	+783 3410 5610 816	7390 Harvest Crest, Mosquito
<input checked="" type="checkbox"/>	Emily Hudson	Uruguay		99%	+886 659 691 561	+634 112 676 1059	6918 Cotton Pine Corner, Ken
<input type="checkbox"/>	Isabelle Donovan	Uruguay		78%	+241 221 394 845	+097 641 829 585	7619 Tawny Carrefour, Senlac
<input checked="" type="checkbox"/>	Isla Dalton	Uruguay		15%	+338 085 844 1010	+326 734 1102 156	7619 Tawny Carrefour, Senlac
<input checked="" type="checkbox"/>	Ava Cadwell	Uruguay		44%	+1102 171 194 147	+145 153 485 213	6686 Lazy Ledge, Two Rock,
<input checked="" type="checkbox"/>	Evie Griffin	Uruguay		16%	+6108 968 379 632	+799 044 511 914	3685 Rocky Glade, Showtuck
<input type="checkbox"/>	Poppy Jett	Uruguay		49%	+037 9104 647 7110	+243 9910 258 929	9218 Crystal Highway, Pickelv
<input type="checkbox"/>	Lily Jagger	Uruguay		10%	+413 527 778 358	+7910 465 373 918	7619 Tawny Carrefour, Senlac
<input type="checkbox"/>	Lucy Cole	Uruguay		90%	+641 937 319 379	+664 143 819 999	9218 Crystal Highway, Pickelv
<input type="checkbox"/>	Chloe Corbin	Uruguay		10%	+722 489 590 185	+536 566 115 237	6683 Colonial Street, Swan R
<input type="checkbox"/>	Poppy Chandler	Uruguay		17%	+557 530 1020 238	+974 945 987 804	7619 Tawny Carrefour, Senlac
<input type="checkbox"/>	Freya Cadwell	Uruguay		27%	+626 876 473 799	+809 203 617 347	6918 Cotton Pine Corner, Ken
<input type="checkbox"/>	Poppy Cadwell	Uruguay		73%	+1056 876 820 921	+242 674 779 705	2347 Indian Boulevard, Frisbe
<input type="checkbox"/>	Lucy Hudson	Uruguay					

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

# Librerías de componentes

- **SB Admin 2.0 ng2:** Tema completo para admin



<http://angularshowcase.github.io/ng2-bootstrap-sbadmin/>



- Introducción a Angular 2
- TypeScript
- Herramientas de desarrollo
- Componentes
- Templates
- Composición de componentes
- Inyección de dependencias y servicios
- Cliente REST
- Aplicaciones multipágina: Router
- Librerías de componentes
- **Conclusiones**

- **Introducción a Angular 2...**
  - Un framework de desarrollo apps **SPA**
  - Recomendación **TypeScript**, más preparado para grandes aplicaciones (pero puede usar ES5, ES6 y Dart)
  - **Orientado a componentes**, con **inyección de dependencias** y **templates**
  - **No es compatible** con Angular 1, pero comparte su arquitectura
  - Mucho mejor **rendimiento** que Angular 1
  - Está en **beta**, pero **pronto** se publicará la versión **estable**
  - Seguramente será uno de los frameworks **más usados** para desarrollo web en los **próximos años**

- **Angular 2 es mucho más..**
  - **Librerías de testing:** Jasmine y Protractor
  - **Carga bajo demanda**
  - **Animaciones**
  - **Angular Universal:** Renderizado en el servidor para optimizar la descarga inicial
  - **Angular2-electron:** Aplicaciones de escritorio con Angular2
  - **Ionic2:** Aplicaciones móviles híbridas con Angular2
  - **NativeScript:** Aplicaciones móviles con UI nativo con Angular2
  - **Angular2-Meteor:** Framework JavaScript fullstack para desarrollo de apps web interactivas

# Apps móviles con Ionic 2

**Micael Gallego**  
micael.gallego@urjc.es  
@micael\_gallego

- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- Creación de proyecto
- Páginas y componentes
- Navegación
- GitHub app
- Ionic app en el móvil
- Conclusiones

## Desarrollo apps móviles

- Plataformas principales: **Android, iOS, Windows Phone**
- Cada plataforma móvil tiene su **propia tecnología** de desarrollo
- **Diferentes** lenguajes, entornos, librerías
- **Alto coste** de desarrollo apps multiplataforma



# Apps móviles híbridas

## Apps móviles híbridas



## Apps móviles híbridas

- Aplicaciones implementadas con **tecnologías web** (HTML, CSS y JS)
- Se **instalan y actualizan** como una app normal
- Acceden a los recursos del dispositivo con ***plugins* nativos**
- No son necesarios conocimientos de cada **tecnología nativa**



## Apps nativas vs híbridas

- Las apps nativas...

- Se **integran** mejor con el sistema operativo
- Aprovechan mejor las **funcionalidades** de los **dispositivos**
- Pueden tener mejor **rendimiento** en terminales **antiguos**

- Las apps híbridas...

- Son mucho **más fáciles** de desarrollar (tecnologías web)
- Un **mismo** código, **múltiples** plataformas
- Pueden **compartir** parte del código con **webs SPA**

## Frameworks apps híbridas



PhoneGap



**Kendo UI**  
THE ART OF WEB DEVELOPMENT



*Onsen UI*



**Sencha**



APACHE  
CORDOVA™



titanium™

Intel® XDK

## Frameworks apps híbridas



PhoneGap



**Kendo UI**  
THE ART OF WEB DEVELOPMENT



*Onsen UI*



**Sencha**



APACHE  
CORDOVA™



titanium™

Intel® XDK

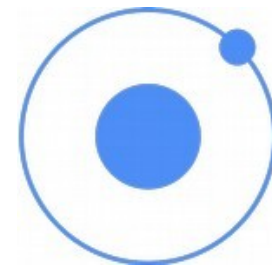
- Apps móviles híbridas
- **Ionic 2**
- Instalación de herramientas
- Creación de proyecto
- Páginas y componentes
- Navegación
- GitHub app
- Ionic app en el móvil
- Conclusiones

## ¿Qué es?

- Es un framework **libre** de desarrollo de apps móviles híbridas
- Con **tecnologías web** (HTML, CSS y JavaScript)
- Basado en **Angular 2** (por tanto, todavía en **beta**)
- La misma app se ejecuta en **Android** y **iOS** (**soporte alpha para Windows Phone**)
- Basado en **Apache Cordova**



APACHE  
CORDOVA™



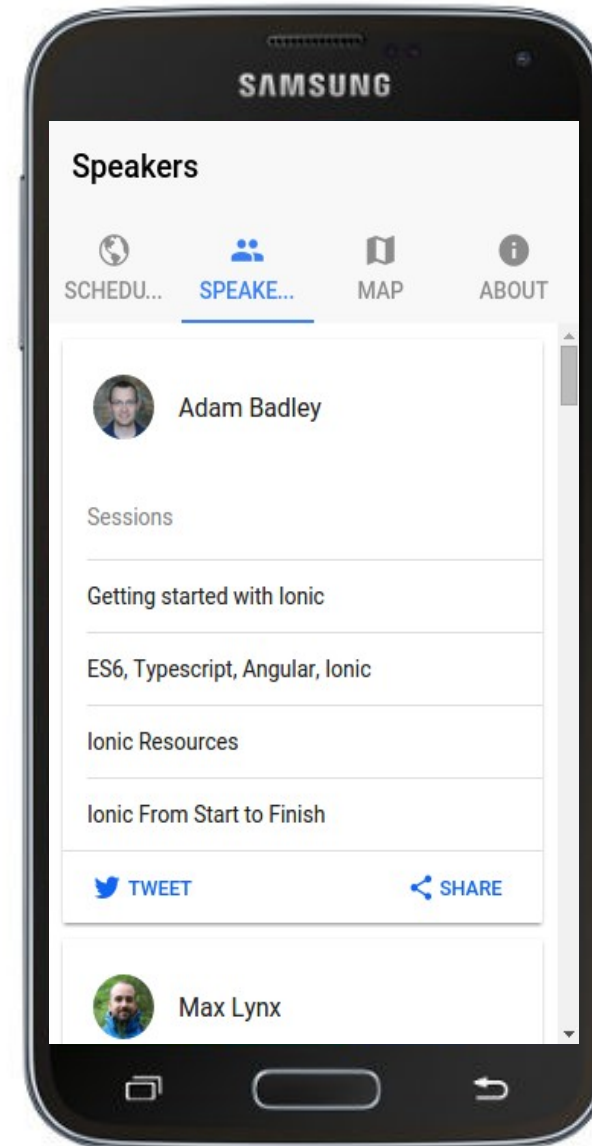
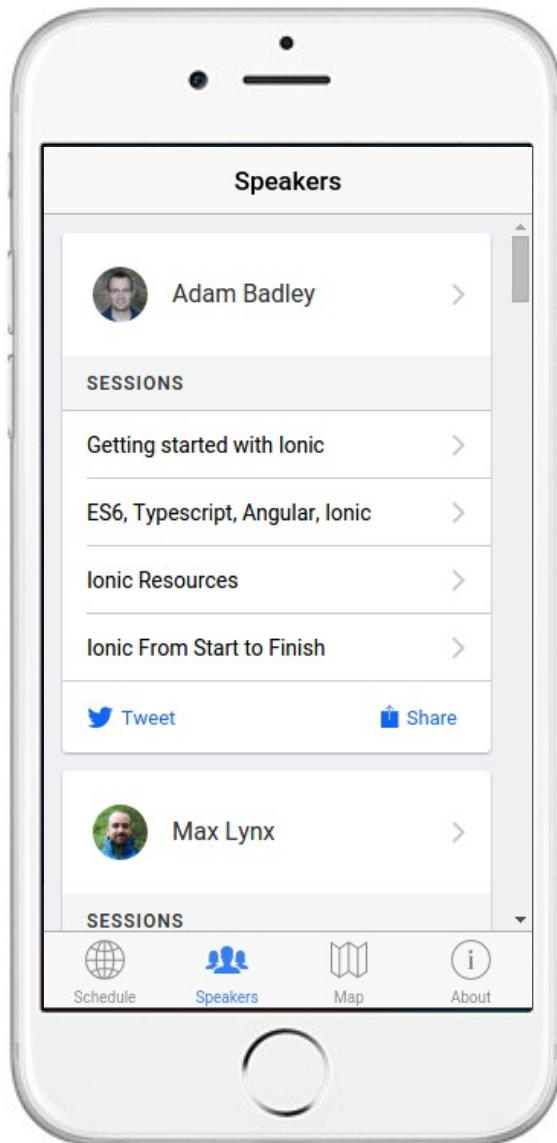
ionic

<http://ionicframework.com/>

## ¿Qué aporta ionic sobre apache cordova?

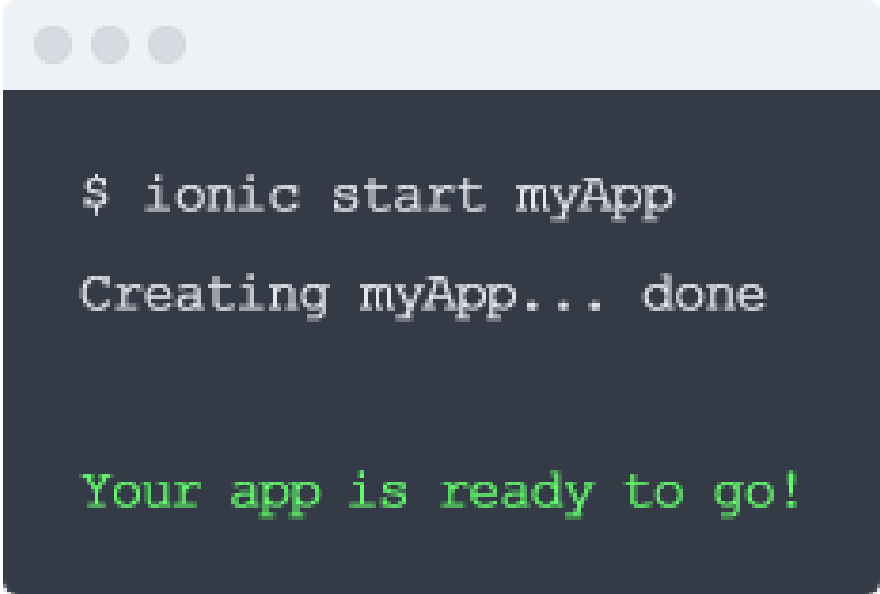
- Integración con Angular 2
- Iconos
- Librería de componentes
- La misma app se ve diferente en Android, iOS o Windows Phone (si queremos)
- Catálogo de plugins Apache Cordova probados y con soporte
- Herramientas y servicios

# Ionic 2



## Herramientas y servicios

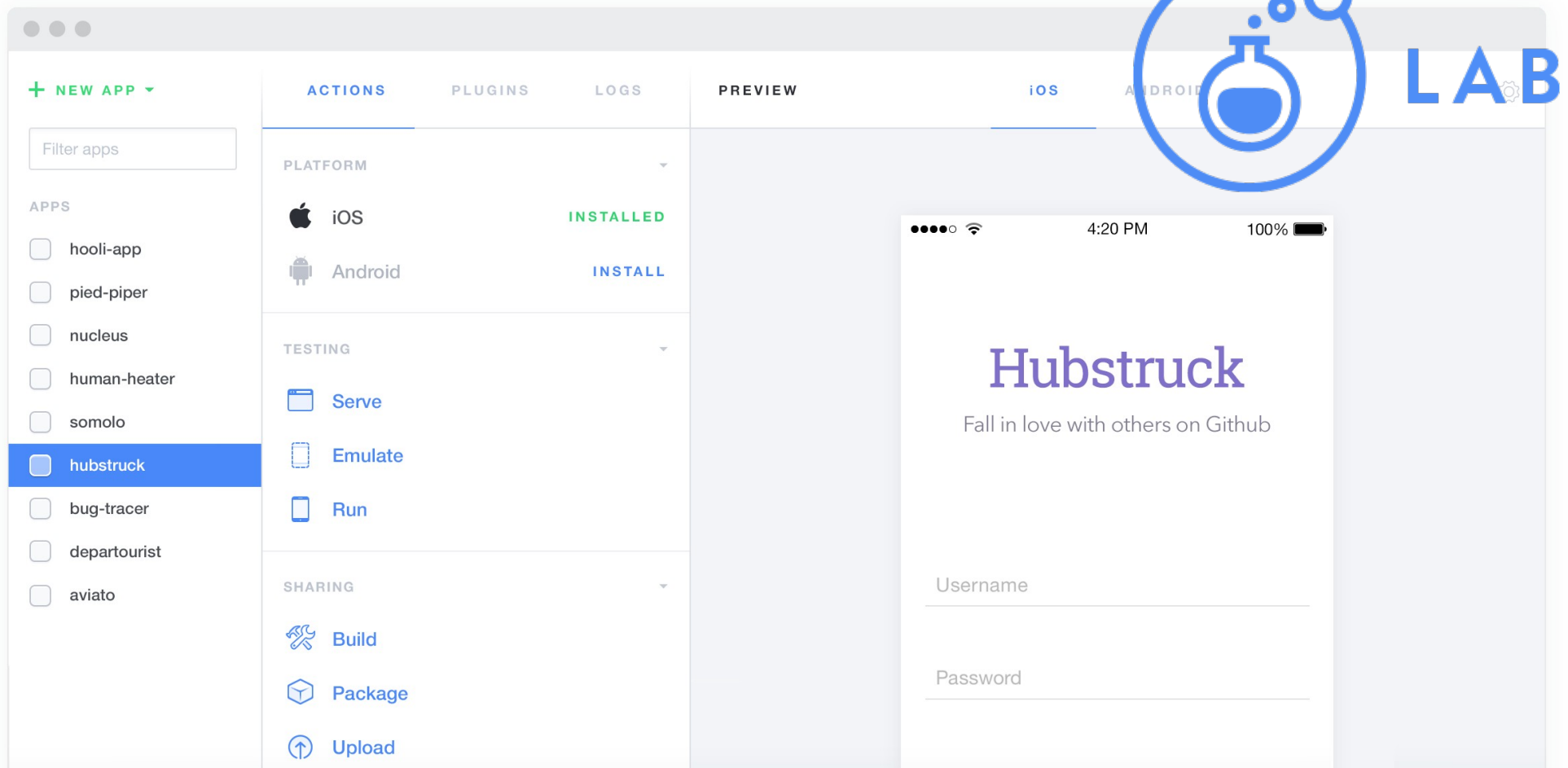
- **ionic-cli:** Herramienta de línea de comandos (cli) para gestionar proyectos: creación, compilación, publicación...



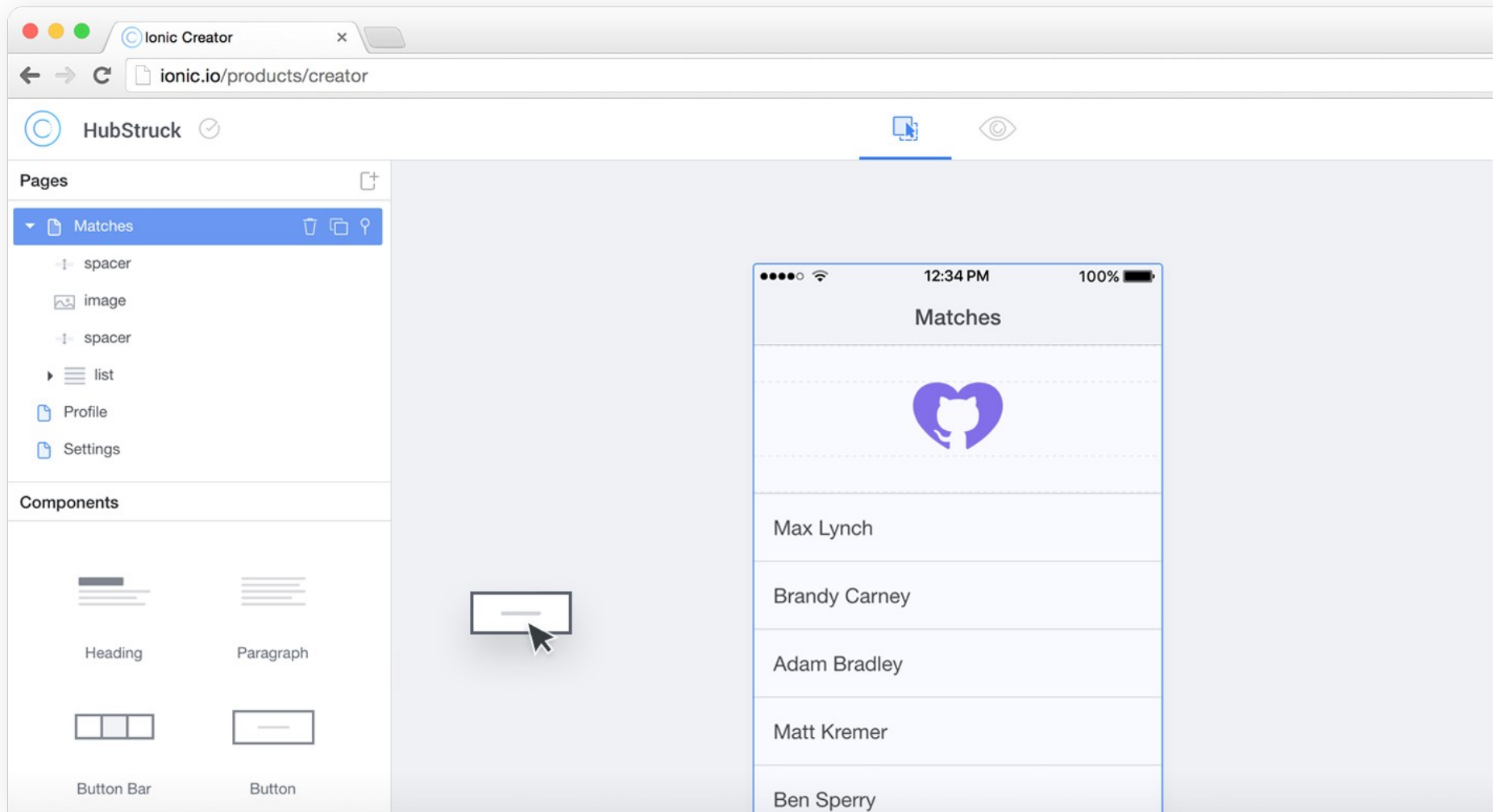
```
$ ionic start myApp  
Creating myApp... done  
  
Your app is ready to go!
```



## Ionic-lab: App interactiva para prototipos



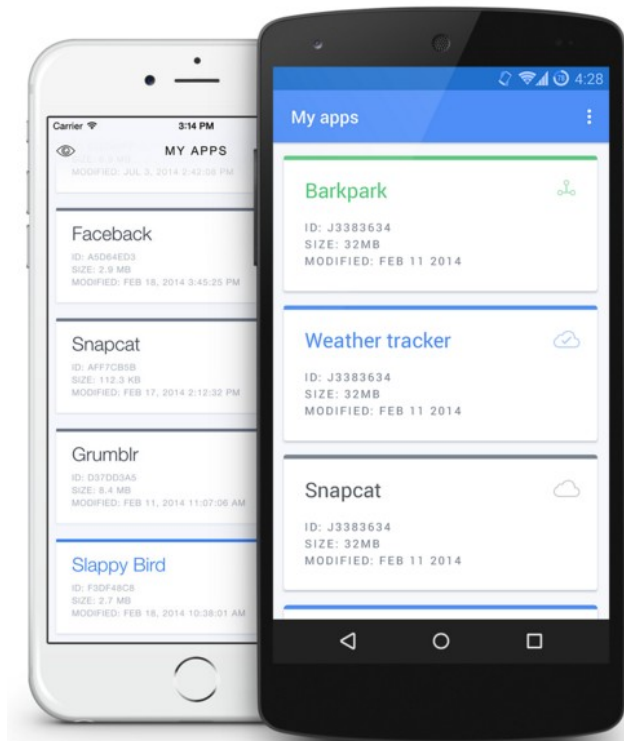
## Ionic-creator: Creación interactiva de apps



**Ionic-view:** Aplicación móvil que permite visualizar cualquier app ionic en desarrollo. Ideal para enseñar a clientes, probar en dispositivos...



ionicview



## Ionic-platform: Servicios en la nube\* para facilitar el desarrollo de apps



BUILD



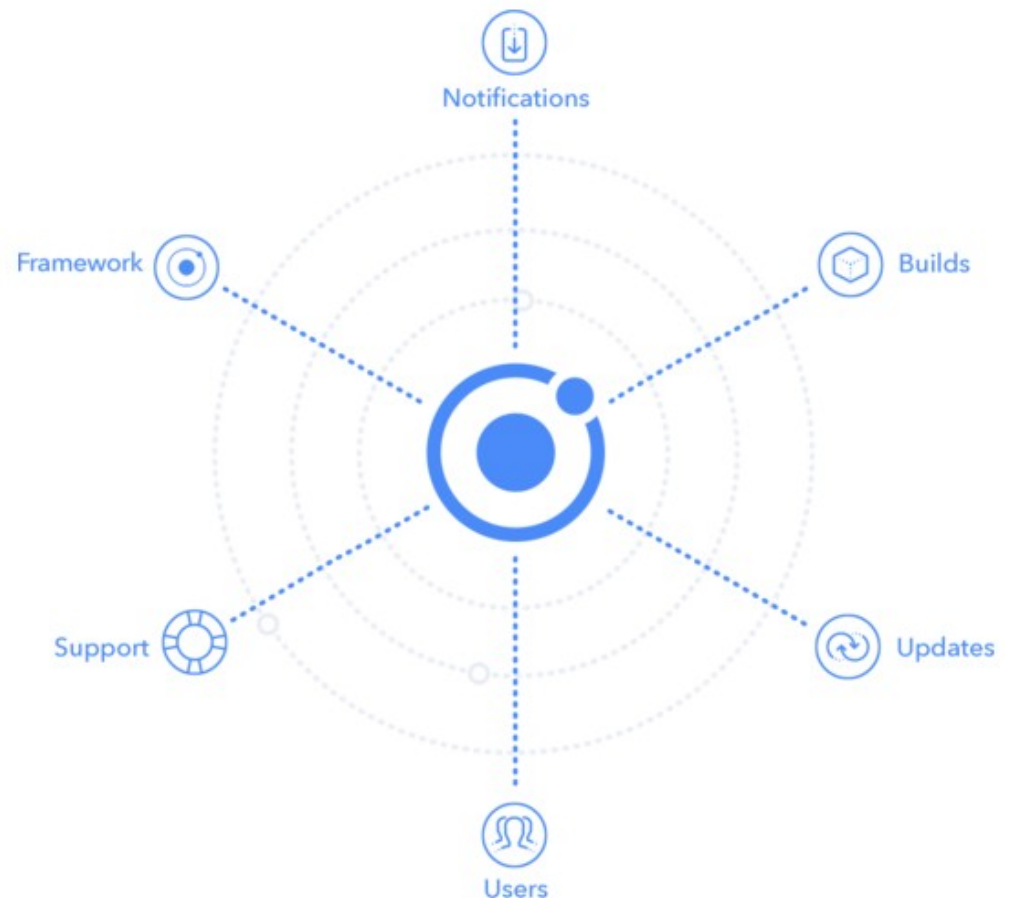
DEPLOY



AUTH



PUSH



\* Servicios de pago

- Apps móviles híbridas
- Ionic 2
- **Instalación de herramientas**
- Creación de proyecto
- Páginas y componentes
- Navegación
- GitHub app
- Ionic app en el móvil
- Conclusiones

- Vamos a utilizar la herramienta por línea de comandos: **ionic-cli**
- Tenemos que tener instalado **nodejs** (que incluye **npm**)
- Comando de **instalación** (ejecutar con permisos de **administrador**)

```
$ npm install -g ionic@beta
```

- Verificar instalación

```
$ ionic
```

- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- **Creación de proyecto**
- Páginas y componentes
- Navegación
- GitHub app
- Ionic app en el móvil
- Conclusiones

- Existen varias **plantillas** con proyectos de ejemplo
- Nosotros usaremos la plantilla **vacía**

```
$ ionic start ejem1 blank --v2 --ts
```

- Si nos pregunta, de momento no creamos la cuenta en la web de ionic para poder usar **ionic view**



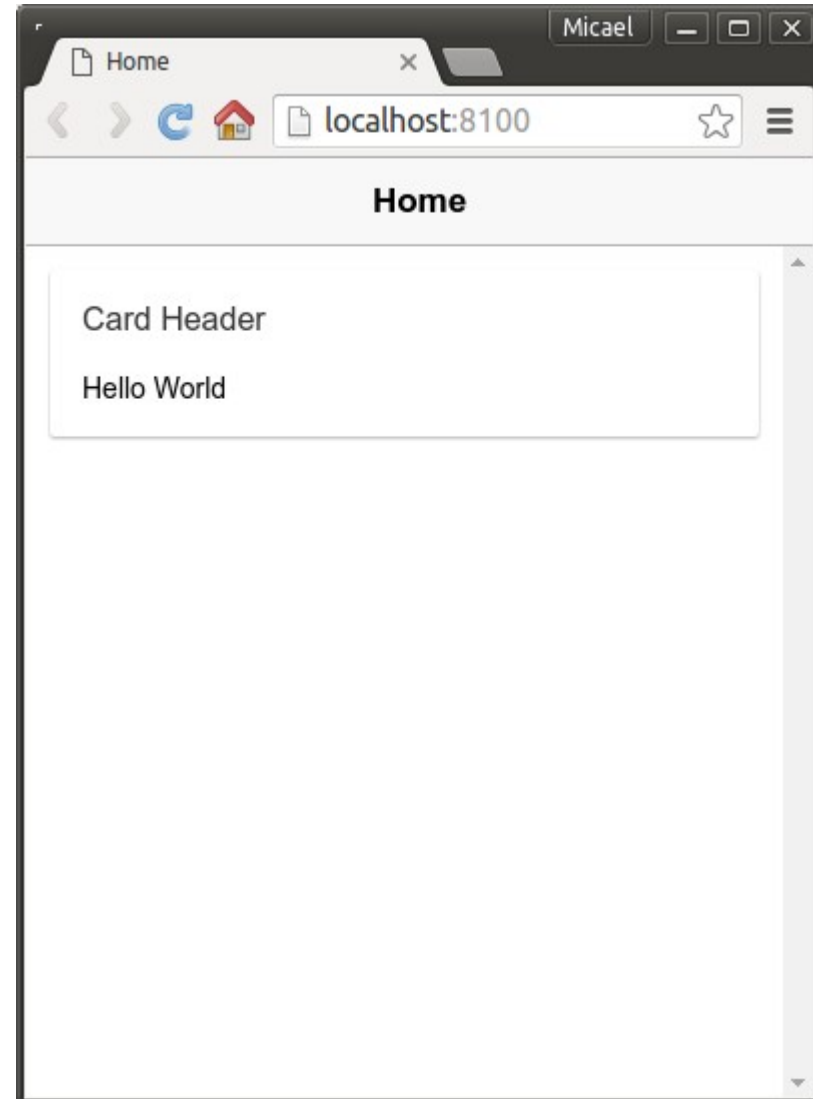
- Compilación con inicio automático del navegador\* y **autorearga** al guardar

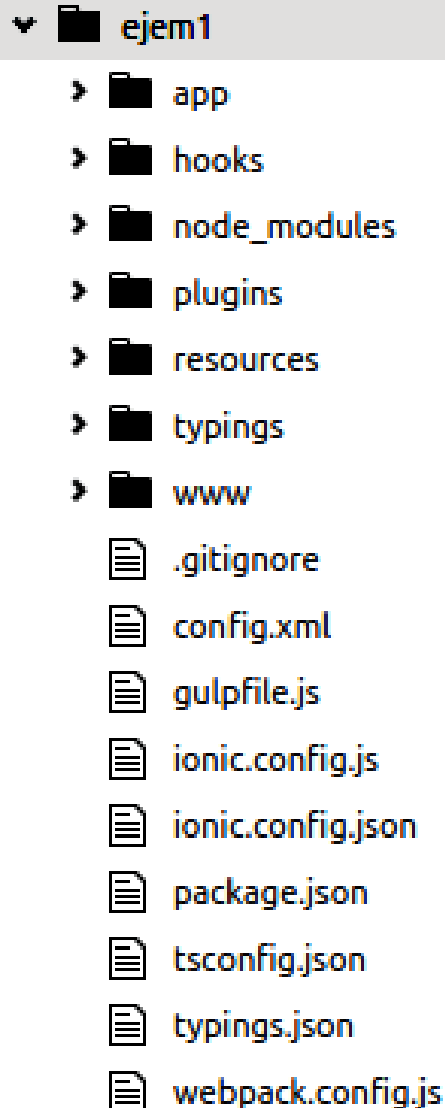
```
$ cd ejem1  
$ ionic serve
```

- Es posible que nos pregunte la **IP** en la que arrancar el **servidor**
- Si usamos la IP de la **wifi**, podemos acceder a la app con el navegador del **móvil**

# Creación de proyecto

- Mejor cambiar el **tamaño** del browser para que sea similar a un **móvil**
- Con ionic serve podemos ver la **consola del navegador** en el terminal





- **Estructura del proyecto**
  - **Configuración**
    - package.json: Librerías
    - typings.json: Librerías
    - tsconfig.json: Compilador
    - gulpfile.json: Construcción
  - **Librerías descargadas**
    - node\_modules
    - typings

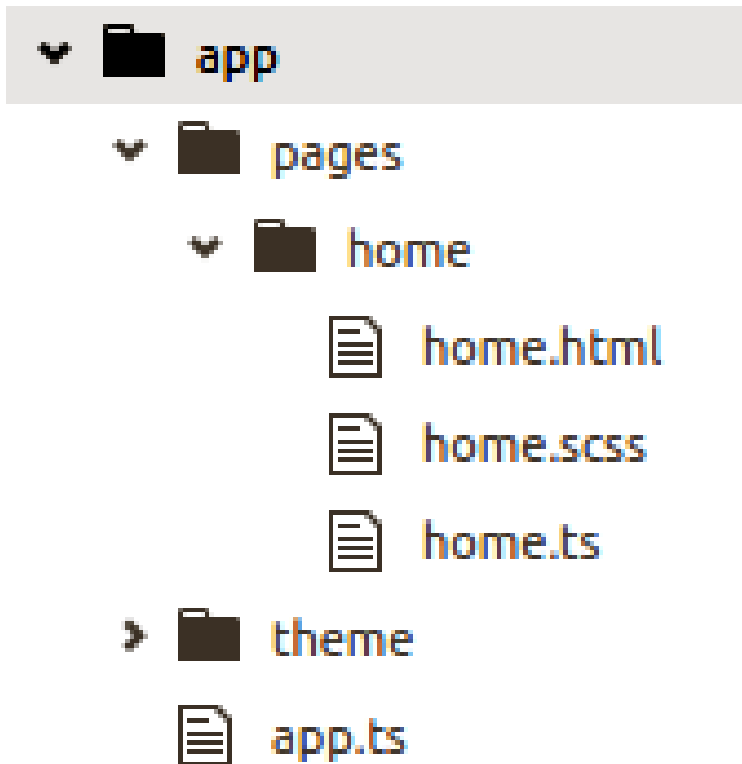
- Estructura del proyecto

- `app.ts`: Fichero inicial

- `pages`: Pantallas de la app

- `home.html`: Plantilla
- `home.ts`: Componente
- `home.scss`: CSS

- `theme`: CSSs globales de la app



- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- Creación de proyecto
- **Páginas y componentes**
- Navegación
- GitHub app
- Ionic app en el móvil
- Conclusiones

- **Páginas vs Componentes**
  - En **angular 2**, la página principal es un **@Component**
  - En **ionic 2**, cada pantalla de la app es una **@Page**

app/pages/home/home.ts

```
import {Page} from 'ionic-angular';

@Page({
  templateUrl: 'build/pages/home/home.html'
})
export class HomePage {}
```

- **Librería de componentes**
  - En **angular 2** no se ofrece ninguna librería de componentes, se usa HTML y CSS
  - En **ionic 2** existe una librería de componentes especialmente diseñados para móviles
  - Todos los componentes tienen la forma `<ion-comp>`
  - Esos componentes tienen un **diseño y comportamiento** adaptado a la **plataforma**

<http://ionicframework.com/docs/v2/components/#overview>

app/pages/home/home.html

```
<ion-navbar *navbar>
  <ion-title>
    Home
  </ion-title>
</ion-navbar>

<ion-content class="home">
  <ion-card>
    <ion-card-header>
      Card Header
    </ion-card-header>
    <ion-card-content>
      Hello World
    </ion-card-content>
  </ion-card>
</ion-content>
```



- Una app ionic 2 es una aplicación angular 2
- Podemos usar:
  - Servicios con inyección de dependencias
  - Componentes personalizados
  - Http
- No podemos usar:
  - Sistema de navegación entre pantalla con @RouteConfig, ionic 2 tiene su propio sistema adaptado al móvil

- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- Creación de proyecto
- Páginas y componentes
- **Navegación**
- GitHub app
- Ionic app en el móvil
- Conclusiones

- **Angular 2 vs ionic 2**
  - El modelo de **navegación** de una app móvil es **diferente** al de una web
  - En las apps móviles **no se ven las URLs** y el usuario no las puede cambiar
  - A diferencia de **angular 2**, no hay que configurar nada porque **no hay URLs (no hay @RouteConfig)**

<http://ionicframework.com/docs/v2/components/#navigation>  
<http://ionicframework.com/docs/v2/api/components/nav/NavController/>

- Ionic 2
  - En **ionic 2** la navegación se realiza usando los métodos del servicio **NavController**
  - Los botones tienen que gestionar el evento (**click**) y llamar al método **nav.push(...)** con la página a la que navegar

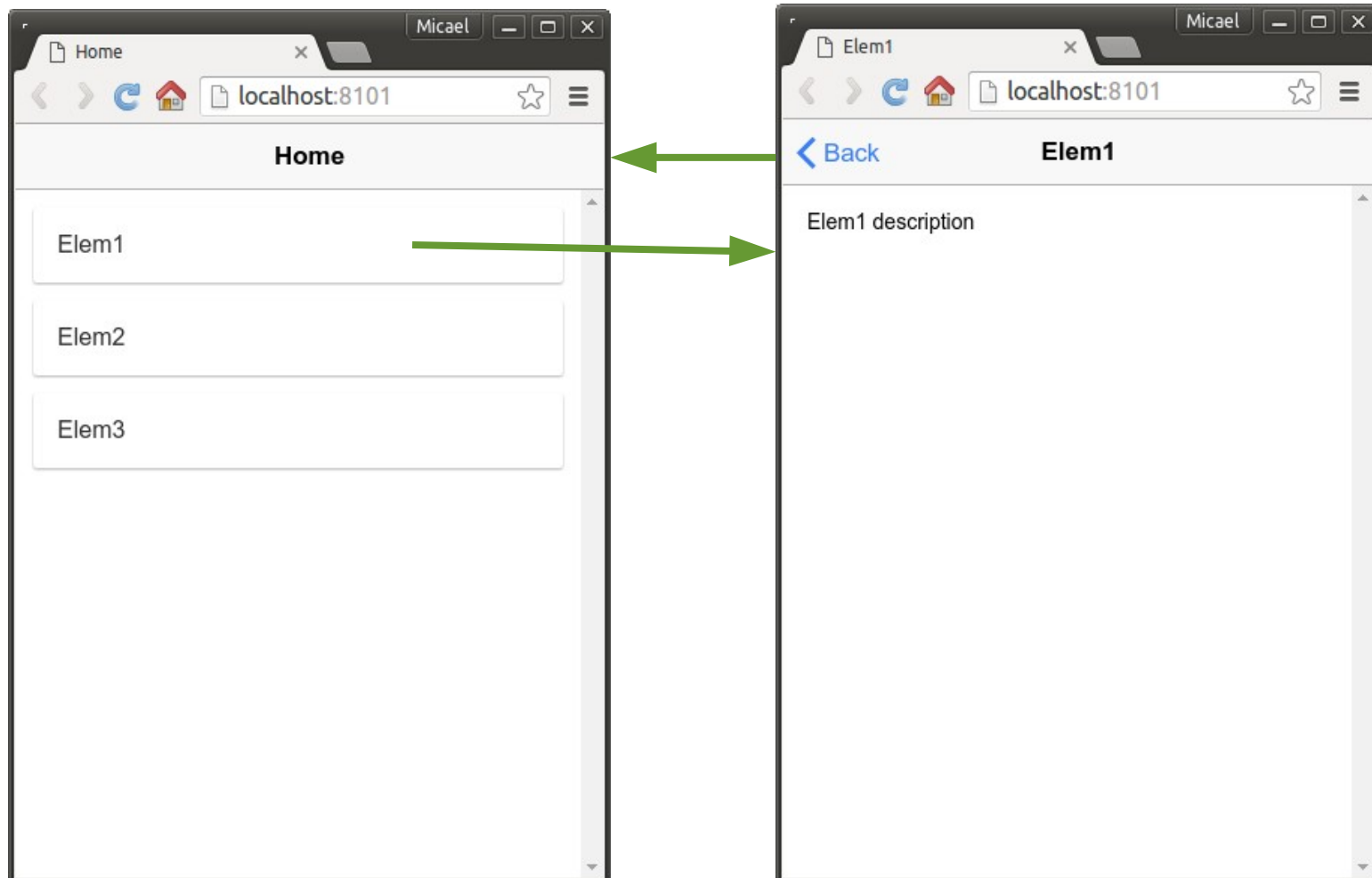
```
@Page({
  template: `
    <ion-navbar *navbar><ion-title>Login</ion-title></ion-navbar>
    <ion-content>
      <button (click)="goToOtherPage()">Go to OtherPage</button>
    </ion-content>`
})
export class StartPage {
  constructor(private nav: NavController){}
  goToOtherPage() { nav.push(OtherPage); }
}
```

- **Ionic 2**

- Si la página tiene `<ion-navbar>` aparecerá automáticamente un botón de volver atrás
- También se puede navegar hacia atrás con `nav.pop()`
- El método `nav.push()` puede recibir como segundo parámetro un objeto con parámetros a la página
- La nueva página puede acceder a esos parámetros inyectando el servicio **NavParams**

# Navegación

ejem2



app/pages/home/home.ts

ejem2

```
import {Page, NavController} from 'ionic-angular';
import {DetailsPage} from '../details/details';

@Page({
  templateUrl: 'build/pages/home/home.html'
})
export class HomePage {

  public elems = [
    {name:"Elem1", desc: "Elem1 description"},
    {name:"Elem2", desc: "Elem2 description"},
    {name:"Elem3", desc: "Elem3 description"}
  ];

  constructor(private nav: NavController) {}

  goToDetails(elem) {
    this.nav.push(DetailsPage, {elem: elem})
  }
}
```

app/pages/home/home.html

```
<ion-navbar *navbar>
  <ion-title>Home</ion-title>
</ion-navbar>

<ion-content>
  <ion-card *ngFor="#elem of elems"
    (click)="goToDetails(elem)">
    <ion-card-header>
      {{ elem.name }}
    </ion-card-header>
  </ion-card>
</ion-content>
```



app/pages/details/details.ts

```
import {Page, NavParams} from 'ionic-angular';

@Page({
  templateUrl: 'build/pages/details/details.html'
})
export class DetailsPage {

  elem: any;

  constructor(navParams: NavParams) {
    this.elem = navParams.get('elem');
  }
}
```

app/pages/details/details.html

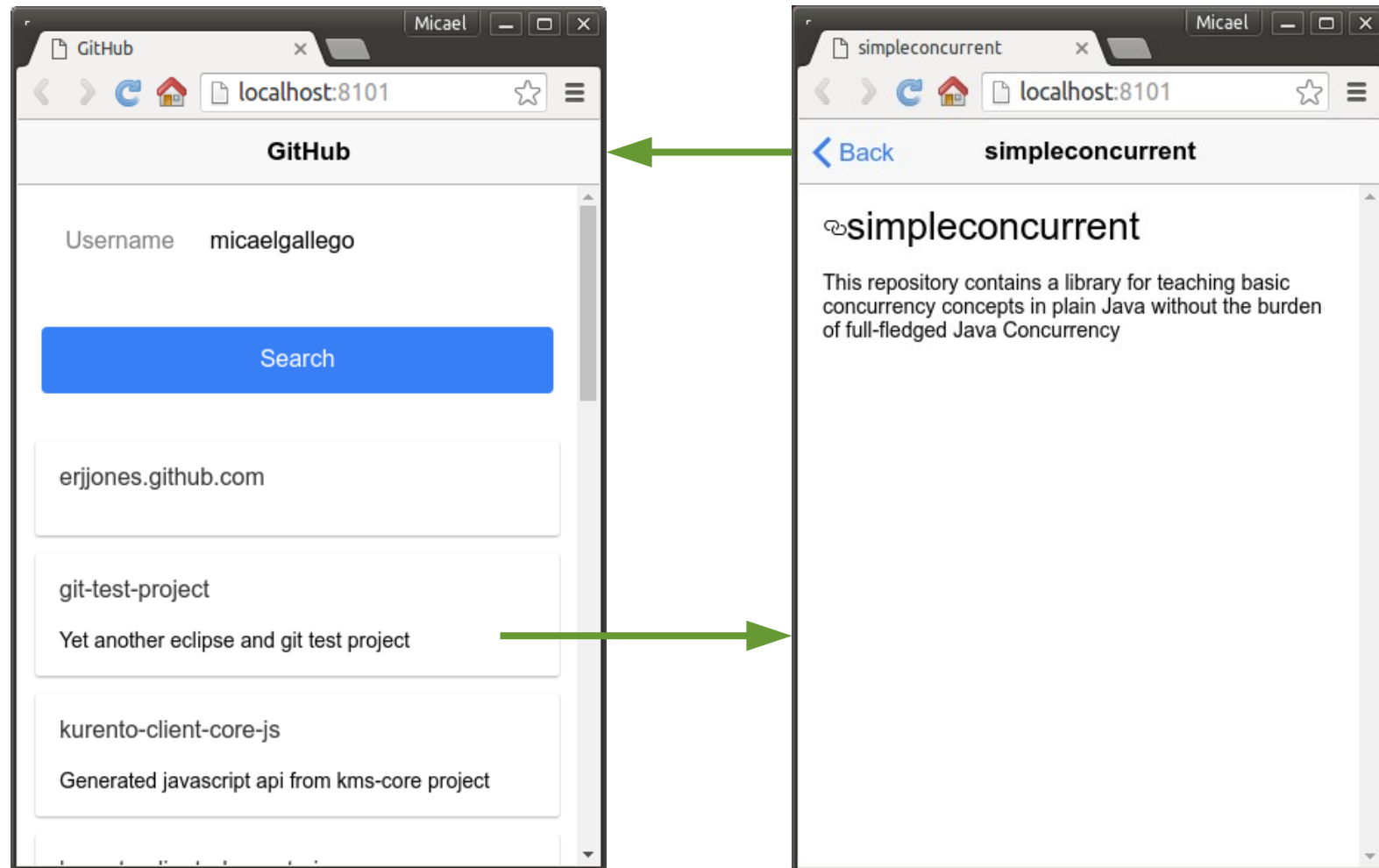
```
<ion-navbar *navbar>
  <ion-title>
    {{ elem.name }}
  </ion-title>
</ion-navbar>

<ion-content>
  {{ elem.desc }}
</ion-content>
```

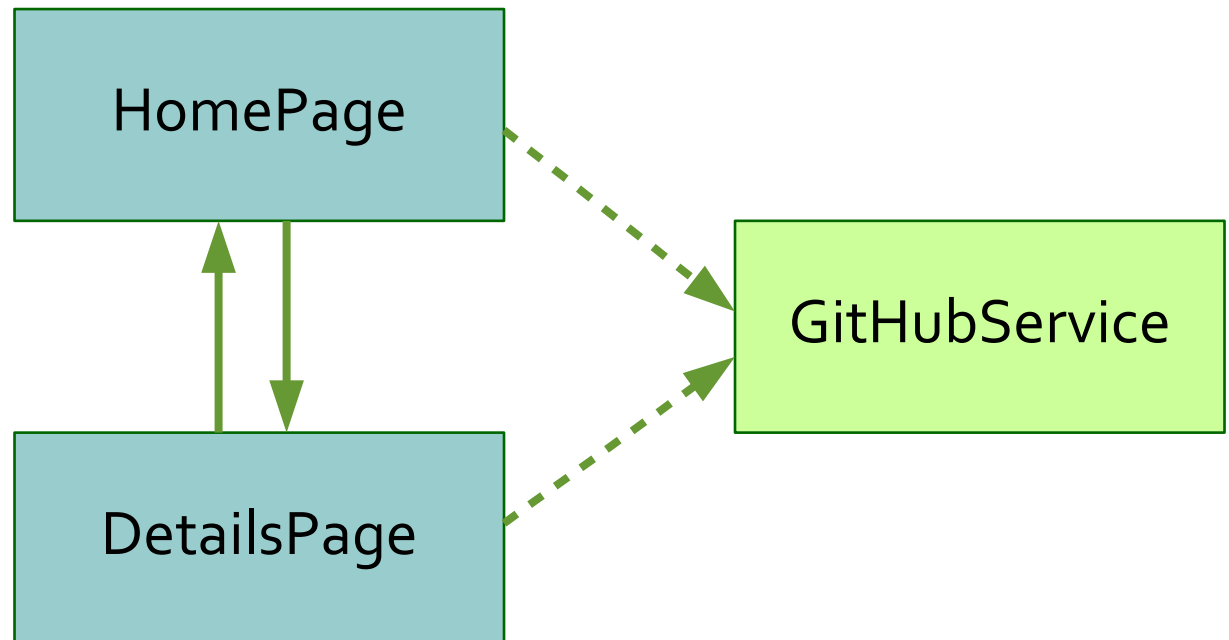
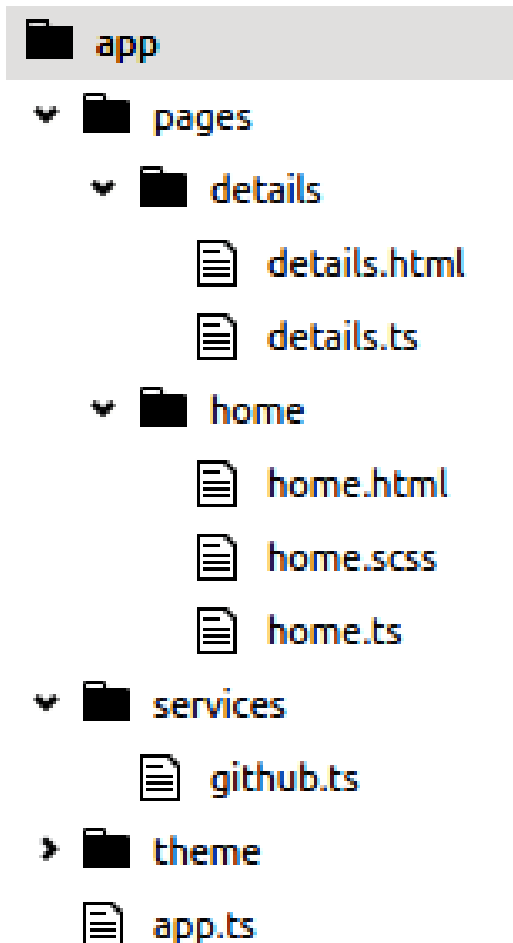
- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- Creación de proyecto
- Páginas y componentes
- Navegación
- **GitHub app**
- Ionic app en el móvil
- Conclusiones

# GitHub app

ejem3



- Arquitectura de la app



```
import {Page, NavController} from 'ionic-angular';
import {GitHubService} from '../services/github';
import {DetailsPage} from '../details/details';

@Page({
  templateUrl: 'build/pages/home/home.html',
  providers: [GitHubService]
})
export class HomePage {

  public foundRepos: any[];
  public username: string;

  constructor(private github: GitHubService,
               private nav: NavController) {}

  getRepos() {
    this.github.getRepos(this.username).subscribe(
      data => this.foundRepos = data.json(),
      err => console.error(err)
    );
  }

  goToDetails(repo) {
    this.nav.push(DetailsPage, { repo: repo });
  }
}
```

app/pages/home/home.html

```
<ion-content class="home">
  <ion-list inset>
    <ion-item>
      <ion-label>Username</ion-label>
      <ion-input [(ngModel)]="username" type="text">
      </ion-input>
    </ion-item>
  </ion-list>
  <div padding>
    <button block (click)="getRepos()">Search</button>
  </div>
  <ion-card *ngFor="#repo of foundRepos" (click)="goToDetails(repo)">
    <ion-card-header>
      {{ repo.name }}
    </ion-card-header>
    <ion-card-content>
      {{ repo.description }}
    </ion-card-content>
  </ion-card>
</ion-content>
```

```
import {Page, NavController, NavParams} from 'ionic-angular';
import {GitHubService} from '../../services/github';

@Page({
  templateUrl: 'build/pages/details/details.html',
  providers: [GitHubService]
})
export class DetailsPage {

  public readme = '';
  public repo;

  constructor(private github: GitHubService,
               private nav: NavController, private navParams: NavParams) {

    this.repo = navParams.get('repo');
    this.github.getDetails(this.repo).subscribe(
      data => this.readme = data.text(),
      err => {
        if (err.status == 404) {
          this.readme = 'This repo does not have a README. :(';
        } else {
          console.error(err);
        }
      },
      () => console.log('getDetails completed')
    );
  }
}
```



app/pages/details/details.html

```
<ion-navbar *navbar>
  <ion-title>
    {{ repo.name }}
  </ion-title>
</ion-navbar>

<ion-content>
  <div padding [innerHTML]="readme"></div>
</ion-content>
```

app/services/github.ts

```
import {Injectable} from 'angular2/core';
import {Http, Headers} from 'angular2/http';

@Injectable()
export class GitHubService {

  constructor(private http: Http) {}

  getRepos(username) {
    return this.http.get(
      `https://api.github.com/users/${username}/repos`);
  }

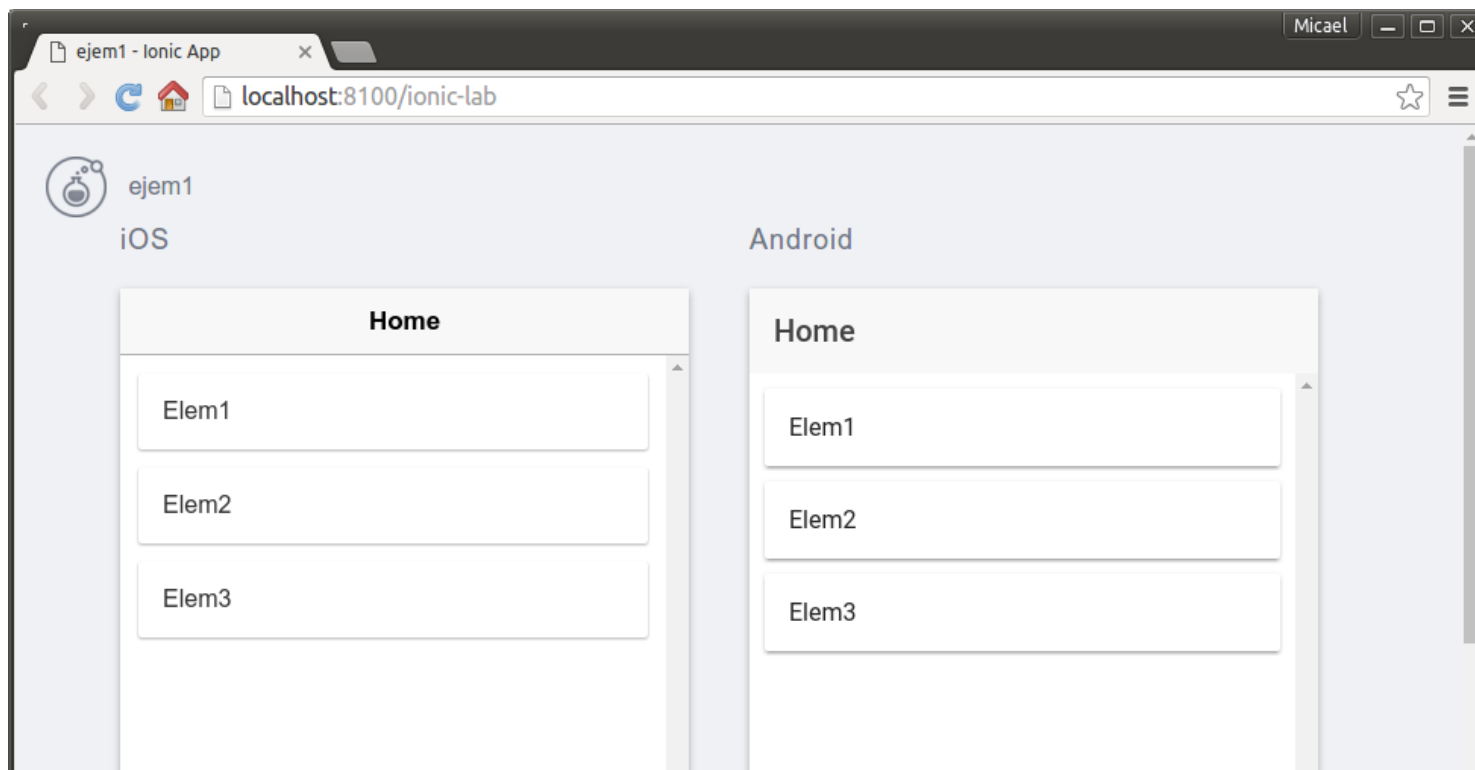
  getDetails(repo) {
    let headers = new Headers();
    headers.append(
      'Accept', 'application/vnd.github.VERSION.html');
    return this.http.get(
      `${repo.url}/readme`, { headers: headers });
  }
}
```

- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- Creación de proyecto
- Páginas y componentes
- Navegación
- GitHub app
- **Ionic app en el móvil**
- Conclusiones

# Ionic app en el móvil

- Para ver la app de forma más parecida a cómo se vería en el móvil podemos usar la opción **--lab**

```
$ ionic serve --lab
```



- La forma más sencilla de ejecutar una **app en un móvil real** es subirla a los servidores de ionic y ejecutarla con **ionic view**
- Creamos una cuenta en <https://apps.ionic.io/signup>
- Subimos la app

```
$ ionic upload
```

- Instalamos **app ionic view** en nuestro móvil (Android o iOS) y abrimos nuestra app

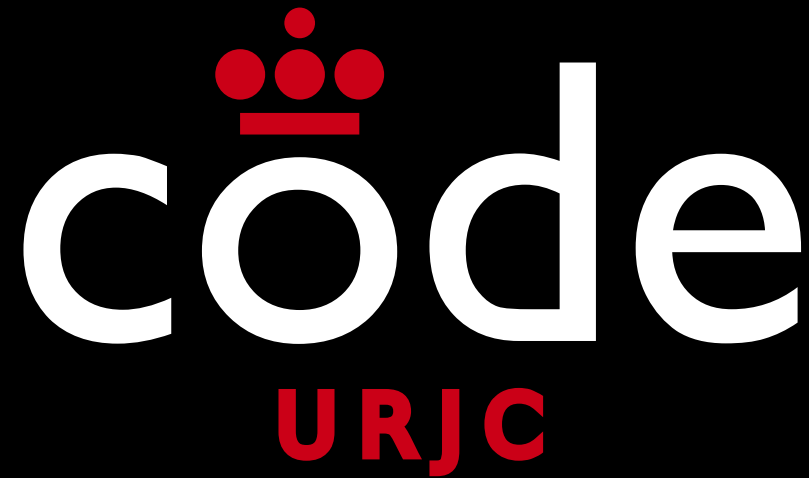
```
Successfully uploaded (2c900a49)
```

- También podemos desplegar las apps directamente en un **emulador** o en **nuestro terminal**
- Es la mejor forma de desarrollar aplicaciones con **plugins** que acceden a los servicios **nativos** del terminal
- Es necesario instalar software específico
  - **Android:** Emulador, herramientas de compilación
  - **iOS:** Emulador, herramientas... y un Mac!!

- Apps móviles híbridas
- Ionic 2
- Instalación de herramientas
- Creación de proyecto
- Páginas y componentes
- Navegación
- GitHub app
- Ionic app en el móvil
- **Conclusiones**

- **ionic** se ha hecho muy popular como plataforma híbrida de desarrollo de apps móviles
- **ionic 2** promete mejora en rendimiento y facilidad de desarrollo
- **ionic 2** es esencialmente **Angular 2**: Se pueden reutilizar los mismos **servicios** que en la **web**
- La **plataforma** en la nube de **ionic** permite **generar** las apps y publicarlas en los markets sin necesidad de herramientas **nativas**





# Consultoría y Formación en Desarrollo Software

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