CURSO DESAROLLO WEB CON MEAN

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
germanux@gmail.com

Angular 2

INDICE

1) Comienzo

2) Arquitectura

3) Introducción al código

4) Módulos en Angular 2

5) Componentes en Angular 2

Comienzo


ANGULAR CLI
> npm install -g angular-cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
Eloquent WebApp

Comienzo

- Crear el esqueleto de una aplicación Angular 2
  - Uno de los comandos que puedes lanzar con Angular CLI es el de creación de un nuevo proyecto Angular 2. Este comando se ejecuta mediante "new", seguido del nombre del proyecto que queramos crear.
    - ng new mi-nuevo-proyecto-angular2
  - Lanzado este comando se creará una carpeta igual que el nombre del proyecto indicado y dentro de ella se generarán una serie de subcarpetas y archivos

Comienzo

- Se instalarán y se configurarán en el proyecto una gran cantidad de herramientas útiles para la etapa del desarrollo front-end.
- De hecho, gran cantidad de los directorios y archivos generados al crear un nuevo proyecto son necesarios para que estas herramientas funcionen.

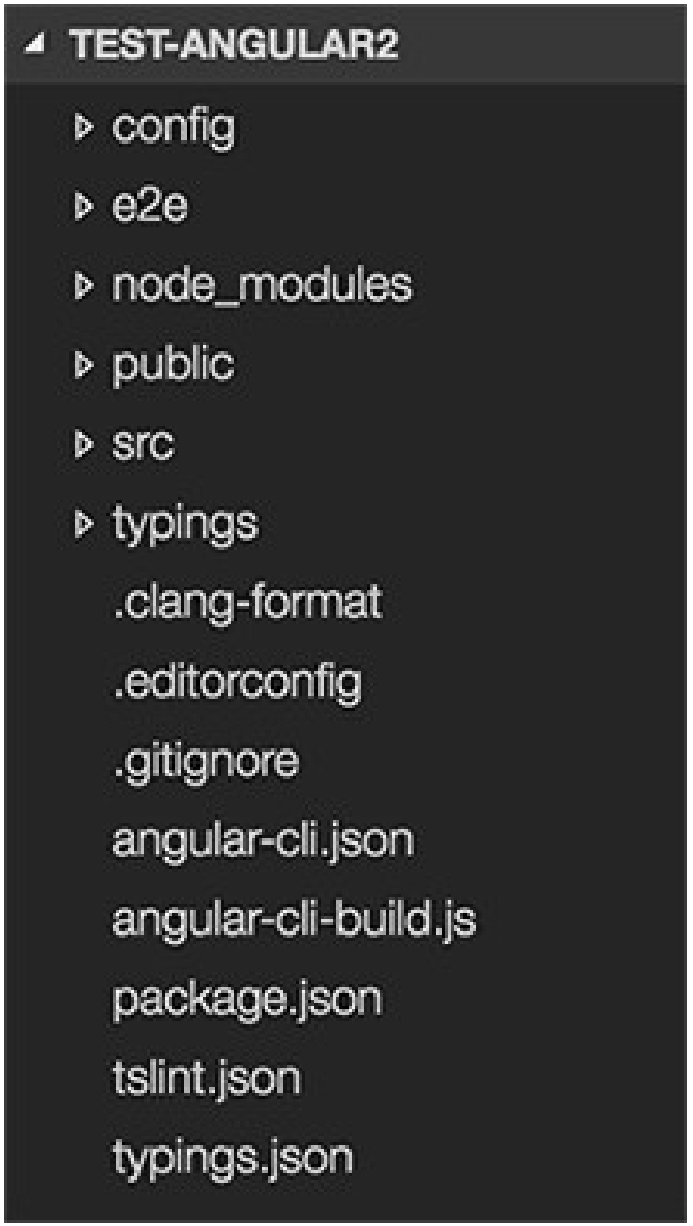
Comienzo

- Entre otras cosas tendremos:
  - Un servidor para servir el proyecto por HTTP
  - Un sistema de live-reload, para que cuando cambiamos archivos de la aplicación se refresque el navegador
  - Herramientas para testing
  - Herramientas para despliegue del proyecto
  - Etc.

Comienzo

- Una vez creado el proyecto inicial podemos entrar en la carpeta con el comando cd.
  - cd mi-nuevo-proyecto-angular2
- Una vez dentro de esa carpeta encontrarás un listado de archivos y carpetas similar a este:

Comienzo


TEST-ANGULAR2
config
e2e
node_modules
public
src
typings
.clang-format
.editorconfig
.gitignore
angular-cli.json
angular-cli-build.js
package.json
tslint.json
typings.json

Comienzo

Servir el proyecto desde un web server

- Angular CLI lleva integrado un servidor web, lo que quiere decir que podemos visualizar y usar el proyecto sin necesidad de cualquier otro software.
- Para servir la aplicación lanzamos el comando "serve".
  - ng serve
- Eso lanzará el servidor web y lo pondrá en marcha. Además, en el terminal verás como salida del comando la ruta donde el servidor está funcionando. Generalmente será algo como esto (pero te sugerimos verificar el puerto en la salida de tu terminal):
  - http://localhost:4200/

Comienzo

```
mcMiguel:test-angular2 midesweb$ ng serve
Could not start watchman; falling back to NodeWatcher for file system events.
Visit http://ember-cli.com/user-guide/#watchman for more info.
Livereload server on http://localhost:49152
Serving on http://localhost:4200/

Build successful - 831ms.

Slowest Trees                                 | Total
----------------------------------------------+---------------------
BroccoliTypeScriptCompiler                    | 405ms
vendor                                        | 337ms

Slowest Trees (cumulative)                    | Total (avg)
----------------------------------------------+---------------------
BroccoliTypeScriptCompiler (1)                | 405ms
vendor (1)                                    | 337ms
```

Podrías modificar el puerto perfectamente si lo deseas, simplemente indicando el puerto deseado con la opción --port:

ng serve --port 4201

Comienzo

ng new

The Angular2 CLI makes it easy to create an application that already works, right out of the box. It already follows our best practices!

ng generate

Generate components, routes, services and pipes with a simple command. The CLI will also create simple test shells for all of these.

ng serve

Easily put your application in production

Test, Lint, Format

Make your code really shine. Run your unittests or your end-to-end tests with the breeze of a command. Execute the official Angular2 linter and run clang format.

Comienzo

Generating and serving an Angular project via a development server

```
ng new PROJECT_NAME
cd PROJECT_NAME
ng serve
```

Navigate to `http://localhost:4200/`. The app will automatically reload if you change any of the source files.

You can configure the default HTTP port and the one used by the LiveReload server with two command-line options :

```
ng serve --host 0.0.0.0 --port 4201 --live-reload-port 49153
```

Comienzo

Generating Components, Directives, Pipes and Services

You can use the `ng generate` (or just `ng g`) command to generate Angular components:

```
ng generate component my-new-component
ng g component my-new-component # using the alias

# components support relative path generation
# if in the directory src/app/feature/ and you run
ng g component new-cmp
# your component will be generated in src/app/feature/new-cmp
# but if you were to run
ng g component ../newer-cmp
# your component will be generated in src/app/newer-cmp
```

Comienzo

You can find all possible blueprints in the table below:

Scaffold	Usage
Component	`ng g component my-new-component`
Directive	`ng g directive my-new-directive`
Pipe	`ng g pipe my-new-pipe`
Service	`ng g service my-new-service`
Class	`ng g class my-new-class`
Interface	`ng g interface my-new-interface`
Enum	`ng g enum my-new-enum`
Module	`ng g module my-module`

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

Arquitectura de Angular 2

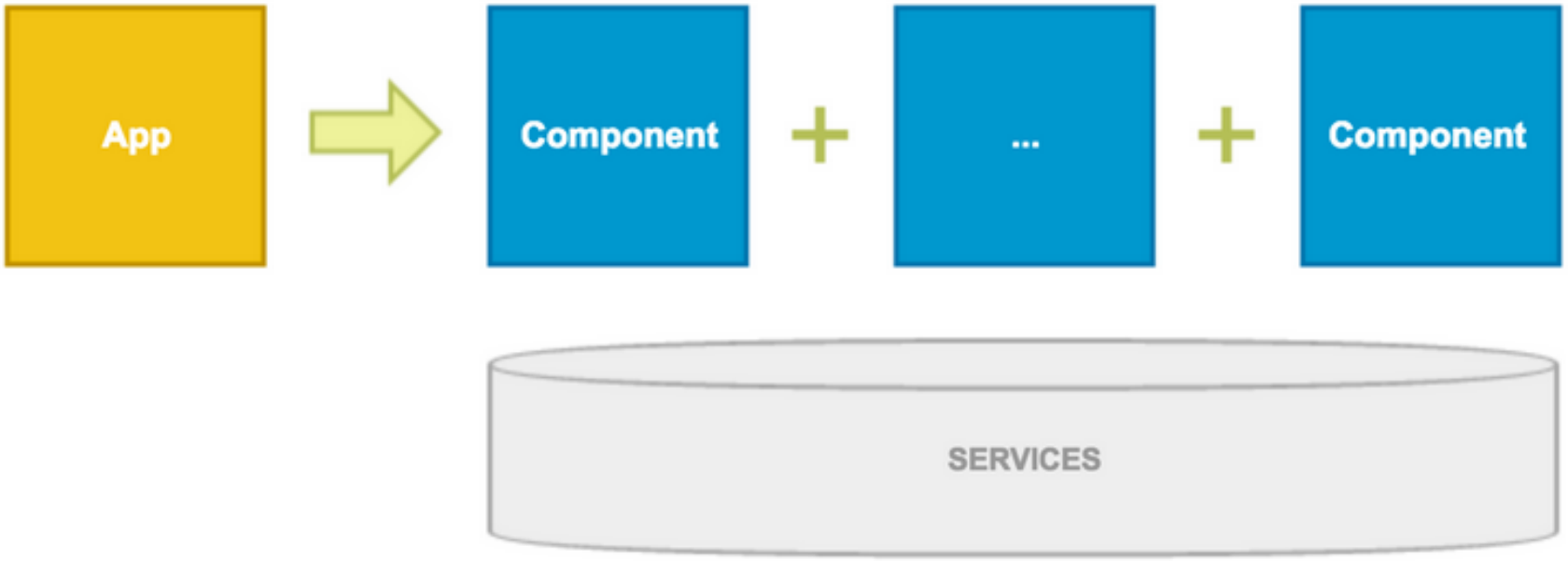
Angular 2 es un framework completo para construir aplicaciones **en cliente** con HTML y Javascript, es decir, con el objetivo de que el peso de la lógica y el renderizado lo lleve el propio navegador, en lugar del servidor.

A groso modo, para crear apps:

- Componemos plantillas HTML (***templates***) con el *markup* de Angular 2
- Escribimos **Componentes** para gestionar esas plantillas
- Encapsulamos la lógica de la aplicación en **Servicios**
- Entregamos el componente raíz de la app al sistema de arranque de Angular 2 (***bootstrap***).

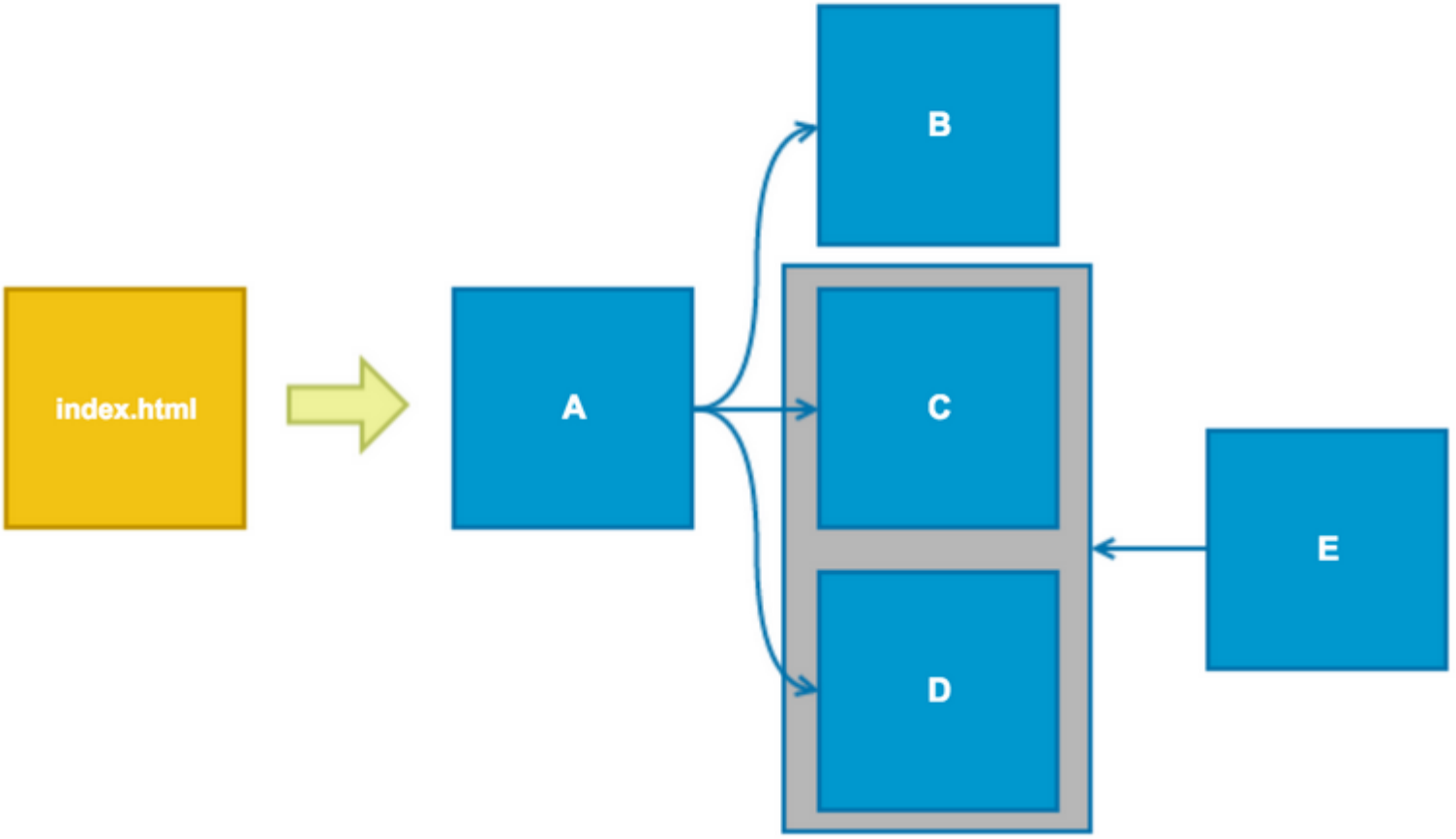
Arquitectura de Angular 2

Una app de *Angular 2* está basada en *components*



Arquitectura de Angular 2

Una arquitectura de ejemplo de una app con *Angular 2*



Arquitectura de Angular 2

TypeScript

¿Por qué TypeScript?

- Es un lenguaje Open Source
- Es un superset de JavaScript
- Transpila a JavaScript nativo
- Fuertemente tipado
- Orientación a objetos basado en clases (Java, C#, C++...)

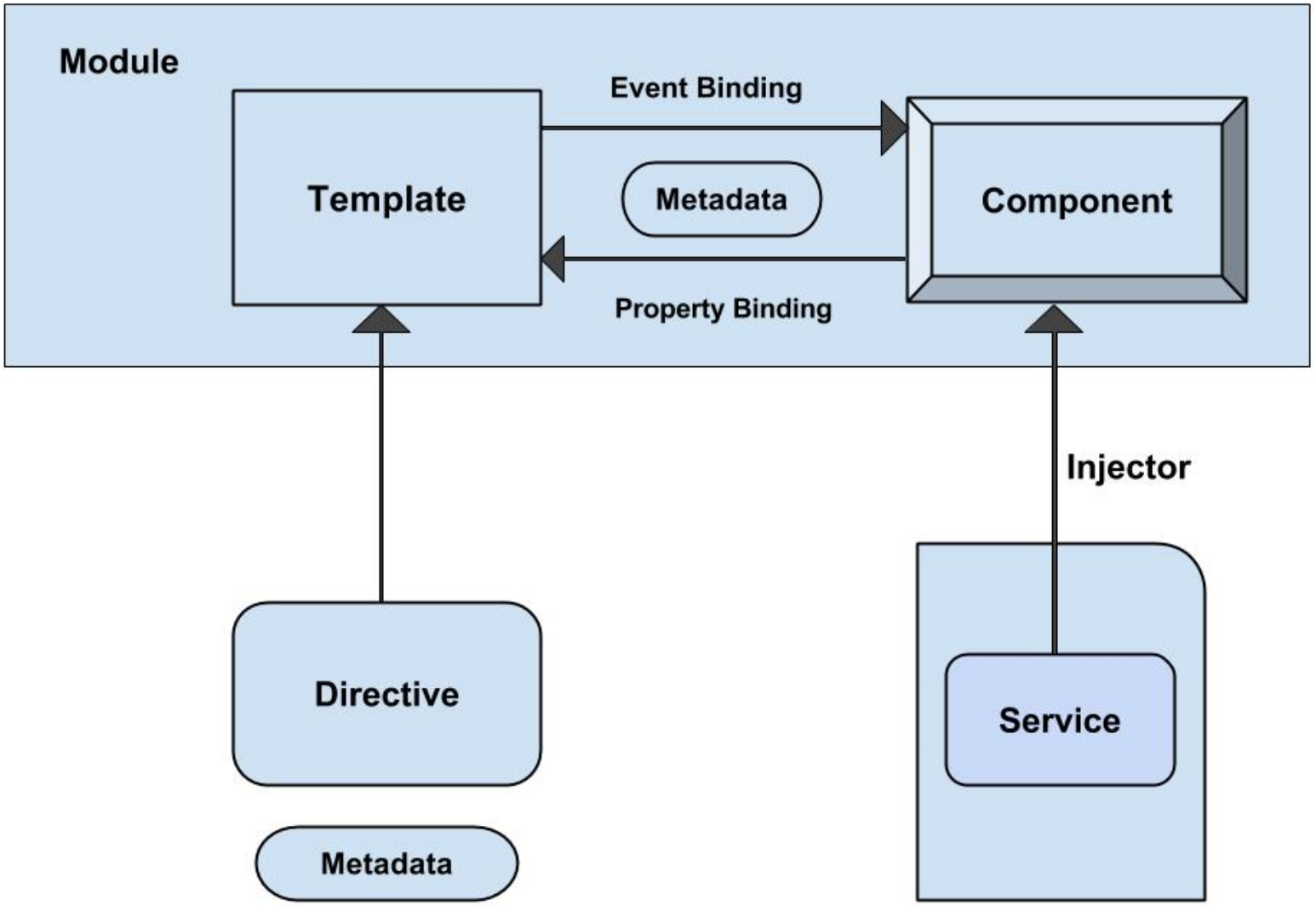
Editores

- Visual Studio
- Visual Studio Code
- WebStorm
- Atom + package
- Eclipse

Características:

- Interfaces
- Inheritance
- Modules

Arquitectura



Introducción al código

El mecanismo de *bootstrap* de Angular 2 sirve para cargar la aplicación y el ejemplo más básico sería así:

```
 //app/main.ts
import {bootstrap}     from '@angular/platform-browser-
dynamic';
import {AppComponent} from './app.component'; //main
component


bootstrap(AppComponent);
```

Introducción al código

En ese momento, Angular se hace cargo de la app, presentando nuestro contenido en el navegador, y respondiendo a las interacciones del usuario en base a las instrucciones que le hemos dado.

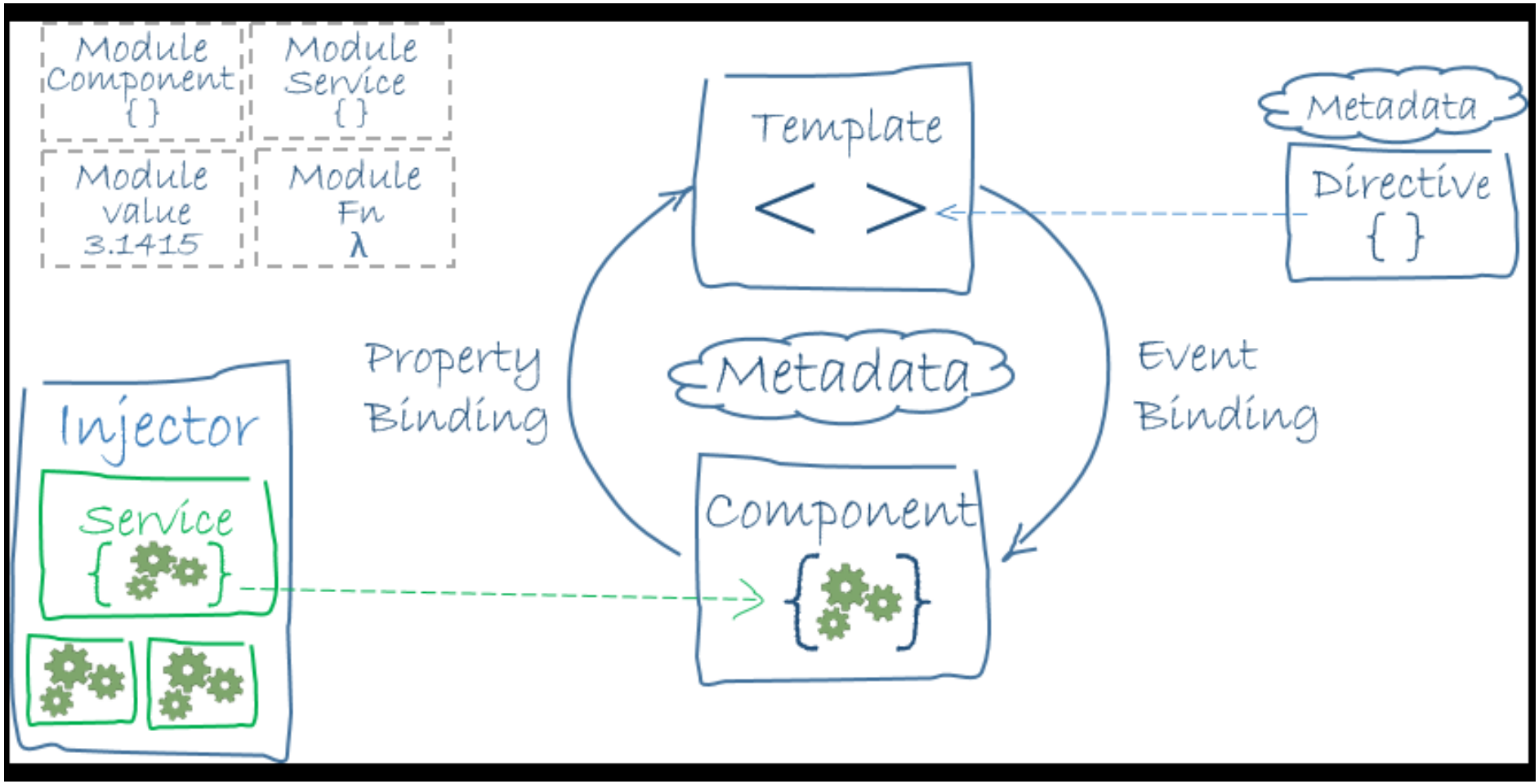
Veamos como se relacionan estos elementos en el diagrama de arquitectura típico, sacado de la web de Angular 2:

Introducción al código

Podemos identificar los 8 bloques principales de una app con Angular 2:

- Modulo
- Componente
- *Template*
- Metadatos
- Data Binding
- Directiva
- Servicio
- *Dependency Injection*

Introducción al código



Módulos en Angular 2

Módulo

Igual que con su predecesor, **las apps de Angular 2 son modulares**, aunque ahora no hace falta una sintaxis específica de Angular para definir módulos, sino que se aprovecha el estándar ECMAScript 2015.

Un módulo, típicamente es un conjunto de código dedicado a cumplir un único objetivo. El módulo exporta algo representativo de ese código, típicamente una única cosa como una clase.

> **Angular 2** utiliza el **sistema de módulos** que define el estándar **ECMAScript 2015**, tienes más detalles aquí http://blog.enriqueoriol.com/2016/03/intro-a-es6-javascript-moderno.html

Introducción al código

Exportar / importar un módulo

Pongamos que queremos exportar un nuevo componente ***AppComponent*** que tenemos definido en el archivo **app.component.js**. Lo haríamos del siguiente modo, con la palabra reservada ***export***:

```
//app/app.component.js
export class AppComponent {
    //aquí va la definición del componente
}
```

Para importarlo en otro lado, por ejemplo en **main.js**, utilizaríamos la palabra reservada **import**, junto con nombre del objeto a importar y el path del archivo:

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

Módulos en Angular 2

Módulos librería

Hay módulos que son librerías de conjuntos de módulos. Angular 2, sin ir más lejos, está empaquetado como una colección de librerías vinculadas a distintos paquetes npm, de modo que solo tengamos que importar aquellos servicios o módulos que necesitemos.

Las librerías principales de Angular 2 son:

- @angular/core
- @angular/common
- @angular/router
- @angular/http

Módulos en Angular 2

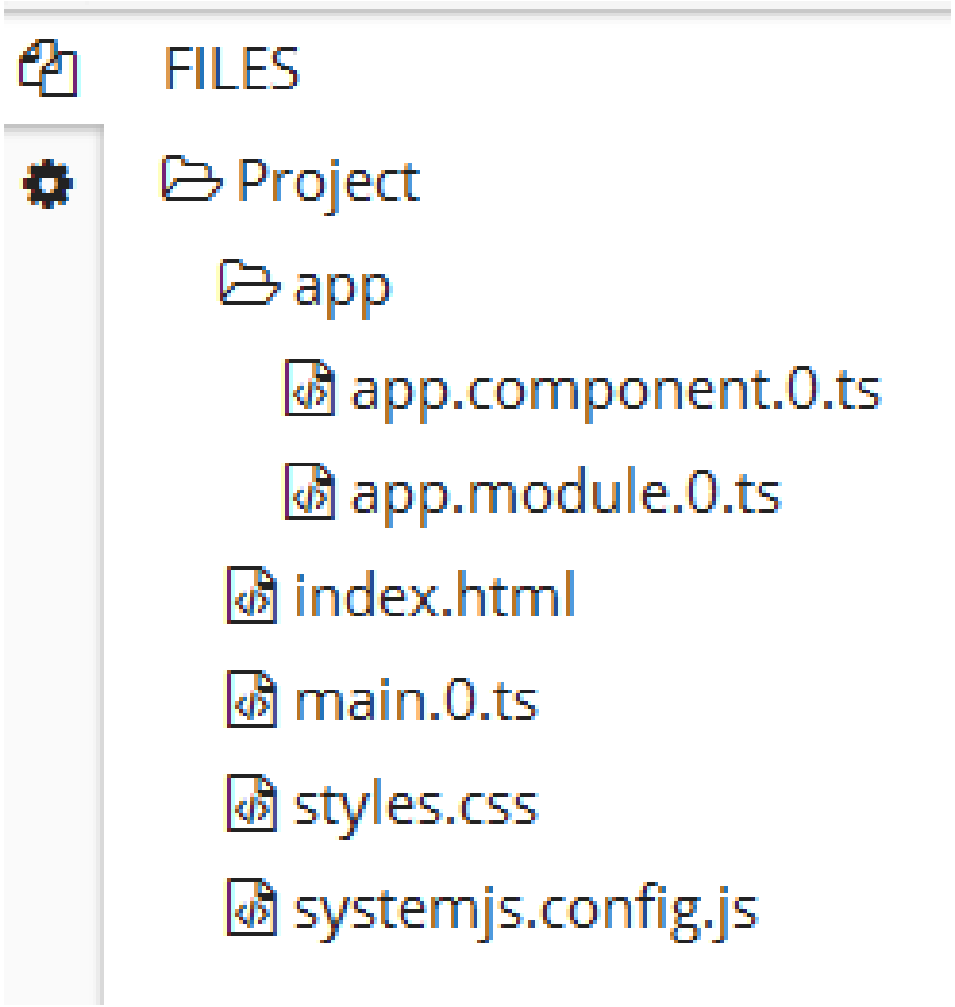
Para importar los elementos ***Component*** y ***Directive*** de ***@angular/core***, lo haríamos del siguiente modo:

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

Módulos en Angular 2

Minimal NgModule

https://embed.plnkr.co/?show=preview



Módulos en Angular 2

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <script>document.write('<base href="' + document.location + '" />');</script>
    <title>NgModule Minimal</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">

     <!-- Polyfills -->
    <script src="https://unpkg.com/core-js/client/shim.min.js"></script>

    <script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>
    <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>
    <script src="systemjs.config.js"></script>
    <script>
      System.import('main.0.js').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

Módulos en Angular 2

main.0.ts

```ts
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule }              from './app/app.module.0';

platformBrowserDynamic().bootstrapModule(AppModule);


/*
Copyright 2017 Google Inc. All Rights Reserved.
Use of this source code is governed by an MIT-style license that
can be found in the LICENSE file at http://angular.io/license
*/
```

@angular/platform-browser-dynamic public

Angular - library for using Angular in a web browser with JIT compilation

Módulos en Angular 2

app/app.module.0.ts

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

import
       { AppComponent }  from './app.component.0';
/*
       { AppComponent }  from './app.component';
*/

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

Módulos en Angular 2

SystemJS

Configurable module loader enabling dynamic ES module workflows in browsers and NodeJS.

SystemJS 0.20 release notes

- Loads any module format when running the ~15KB development build.
- Loads ES modules compiled into the `System.register` module format for production with exact circular reference and binding support
- Supports RequireJS-style map, paths, and bundles configuration.

Built with the ES Module Loader project, which is based on principles and APIs from the WhatWG Loader specification, modules in HTML and NodeJS.

Supports IE9+ provided a promises polyfill is available in the environment.

Módulos en Angular 2

systemjs.config.js

```
/**
 * WEB ANGULAR VERSION
 * (based on systemjs.config.js in angular.io)
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    // DEMO ONLY! REAL CODE SHOULD NOT TRANSPILE IN THE BROWSER
    transpiler: 'ts',
    typescriptOptions: {
      // Copy of compiler options in standard tsconfig.json
      "target": "es5",
      "module": "commonjs",
      "moduleResolution": "node",
      "sourceMap": true,
      "emitDecoratorMetadata": true,
      "experimentalDecorators": true,
      "lib": ["es2015", "dom"],
      "noImplicitAny": true,
      "suppressImplicitAnyIndexErrors": true
    },
    meta: {
```

Módulos en Angular 2

```
},
paths: {
  // paths serve as alias
  'npm:': 'https://unpkg.com/'
},
// map tells the System loader where to look for things
map: {
  // our app is within the app folder
  app: 'app',

  // angular bundles
  '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
  '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
  '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
  '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',
  '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.
  '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
  '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
  '@angular/router/upgrade': 'npm:@angular/router/bundles/router-upgrade.umd.js',
  '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
  '@angular/upgrade': 'npm:@angular/upgrade/bundles/upgrade.umd.js',
  '@angular/upgrade/static': 'npm:@angular/upgrade/bundles/upgrade-static.umd.js',

  // other libraries
  'rxjs':                      'npm:rxjs@5.0.1',
  'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js',
  'ts':                        'npm:plugin-typescript@5.2.7/lib/plugin.js',
  'typescript':                'npm:typescript@2.0.10/lib/typescript.js',

},
// packages tells the System loader how to load when no filename and/or no extension
packages: {
```

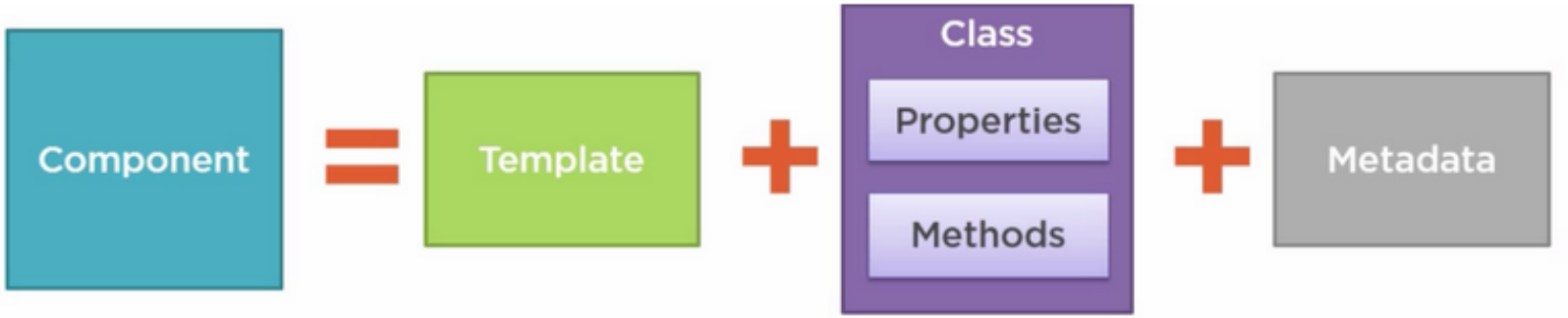
Módulos en Angular 2

```
    'typescript':                  'npm:typescript@2.0.10/lib/typescript.js',

  },
  // packages tells the System loader how to load when no filename and/or no extension
  packages: {
    app: {
      main: './main.ts',
      defaultExtension: 'ts'
    },
    rxjs: {
      defaultExtension: 'js'
    }
  }
});

})(this);
```

Componentes

Components

Angular 2 está basado en componentes. Su estructura básica es la siguiente



Componentes

app/app.component.0.ts

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

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>',
})
export class AppComponent {
  title = 'Minimal NgModule';
}
```

Componentes

styles.css

```css
/* Master Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
a {
  cursor: pointer;
  cursor: hand;
}
button {
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #aaa;
  cursor: auto;
}

/* Navigation link styles */
nav a {
  padding: 5px 10px;
  text-decoration: none;
  margin-right: 10px;
  margin-top: 10px;
  display: inline-block;
  background-color: #eee;
  border-radius: 4px;
}
nav a:visited, a:link {
  color: #607D8B;
}
nav a:hover {
  color: #039be5;
  background-color: #CFD8DC;
}
nav a.active {
  color: #039be5;
}
```

Componentes: Ejemplo

app.component.css

app.component.html

app.component.spec.ts

app.component.ts

Componentes: Ejemplo

app.component.html

```
1    <h1>
2      {{title}}
3    </h1>
```

app.component.css

```
1   /* Clases
2    Selectores
3    Etiquetas
4    y Estilos CSS
5    */
```

Componentes: Ejemplo

app.component.ts ✖

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'app works!';
10  }
```

Componentes: Ejemplo

app.component.spec.ts

```
1   /* tslint:disable:no-unused-variable */
2
3   import { TestBed, async } from '@angular/core/testing';
4   import { AppComponent } from './app.component';
5
6   describe('AppComponent', () => {
7     beforeEach(() => {
8       TestBed.configureTestingModule({
9         declarations: [
10          AppComponent
11        ],
12      });
13      TestBed.compileComponents();
14    });
```

Componentes: Ejemplo

app.component.spec.ts

```
16      it('should create the app', async(() => {
17        const fixture = TestBed.createComponent(AppComponent);
18        const app = fixture.debugElement.componentInstance;
19        expect(app).toBeTruthy();
20      }));
21
22      it(`should have as title 'app works!'`, async(() => {
23        const fixture = TestBed.createComponent(AppComponent);
24        const app = fixture.debugElement.componentInstance;
25        expect(app.title).toEqual('app works!');
26      }));
27
28      it('should render title in a h1 tag', async(() => {
29        const fixture = TestBed.createComponent(AppComponent);
30        fixture.detectChanges();
31        const compiled = fixture.debugElement.nativeElement;
32        expect(compiled.querySelector('h1').textContent).toContain('app works!');
33      }));
34    });
```

Arquitectura

Lifecycle

El ciclo de vida de un componente