CURSO DESAROLLO WEB CON MEAN

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
germanux@gmail.com

Angular 2

INDICE

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.

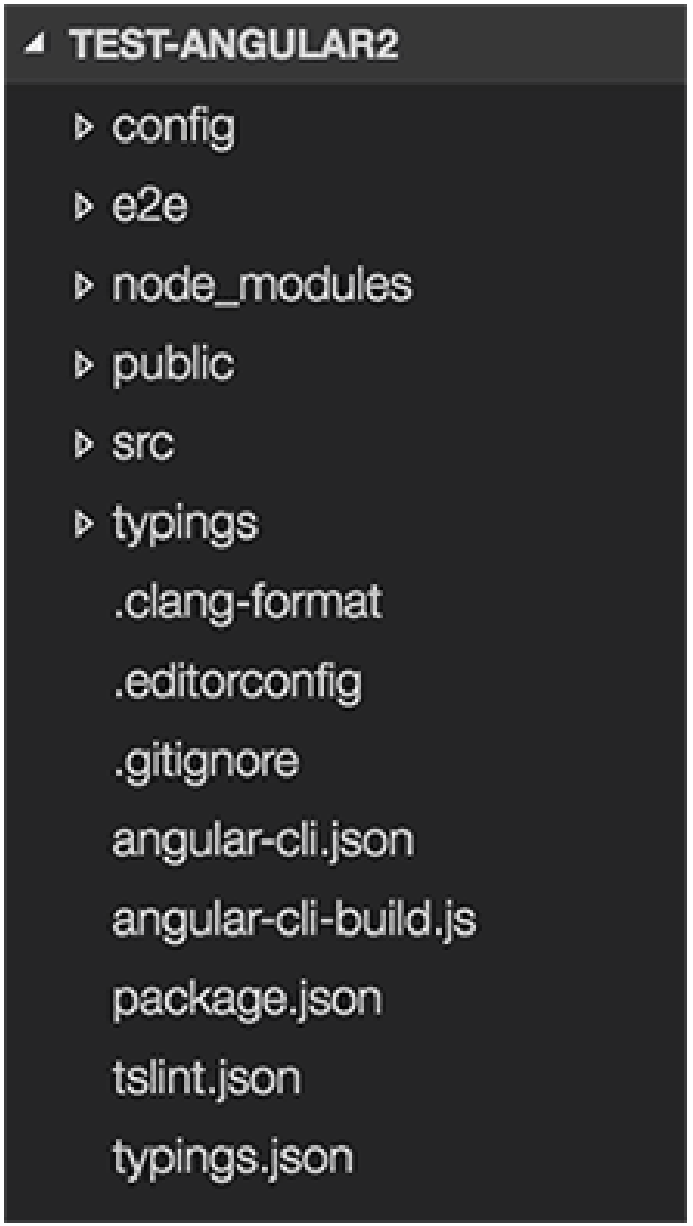
Comienzo

- 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

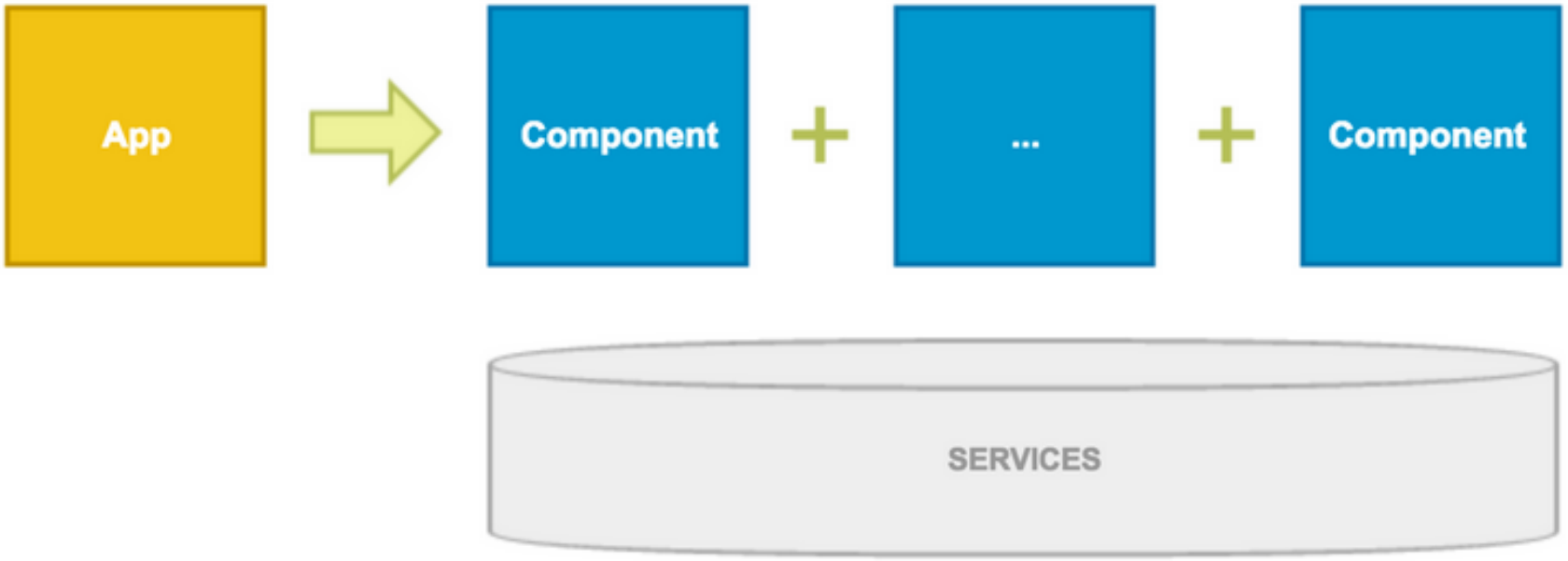
Angular 2 es un framework completo para construir aplicaciones **en cliente** con HTML y Javascript, es decir, con el objetivo de que el peso de la lógica y el renderizado lo lleve el propio navegador, en lugar del servidor.

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***).

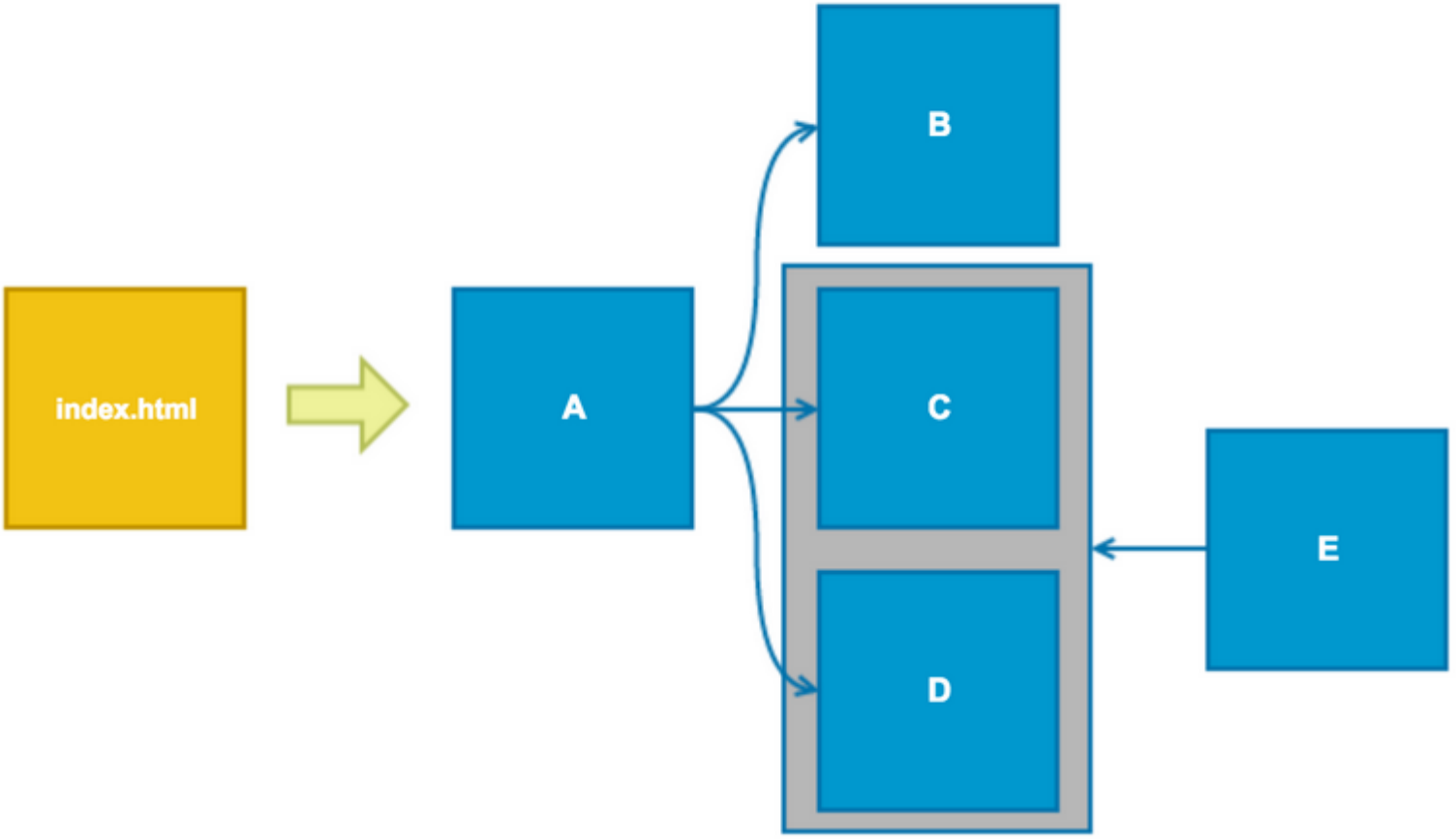
Arquitectura de Angular 2

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++...)

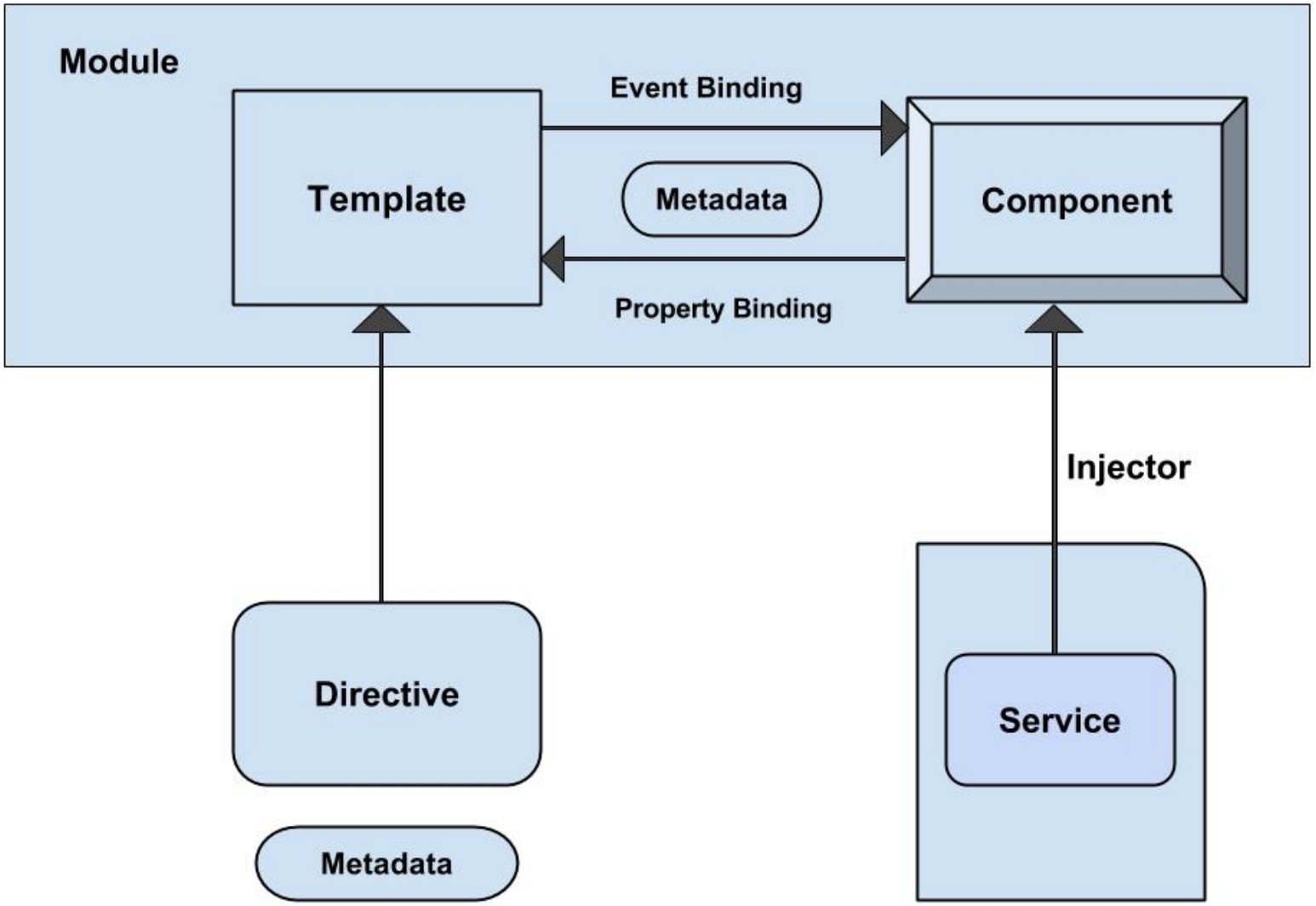
Editores

- 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.

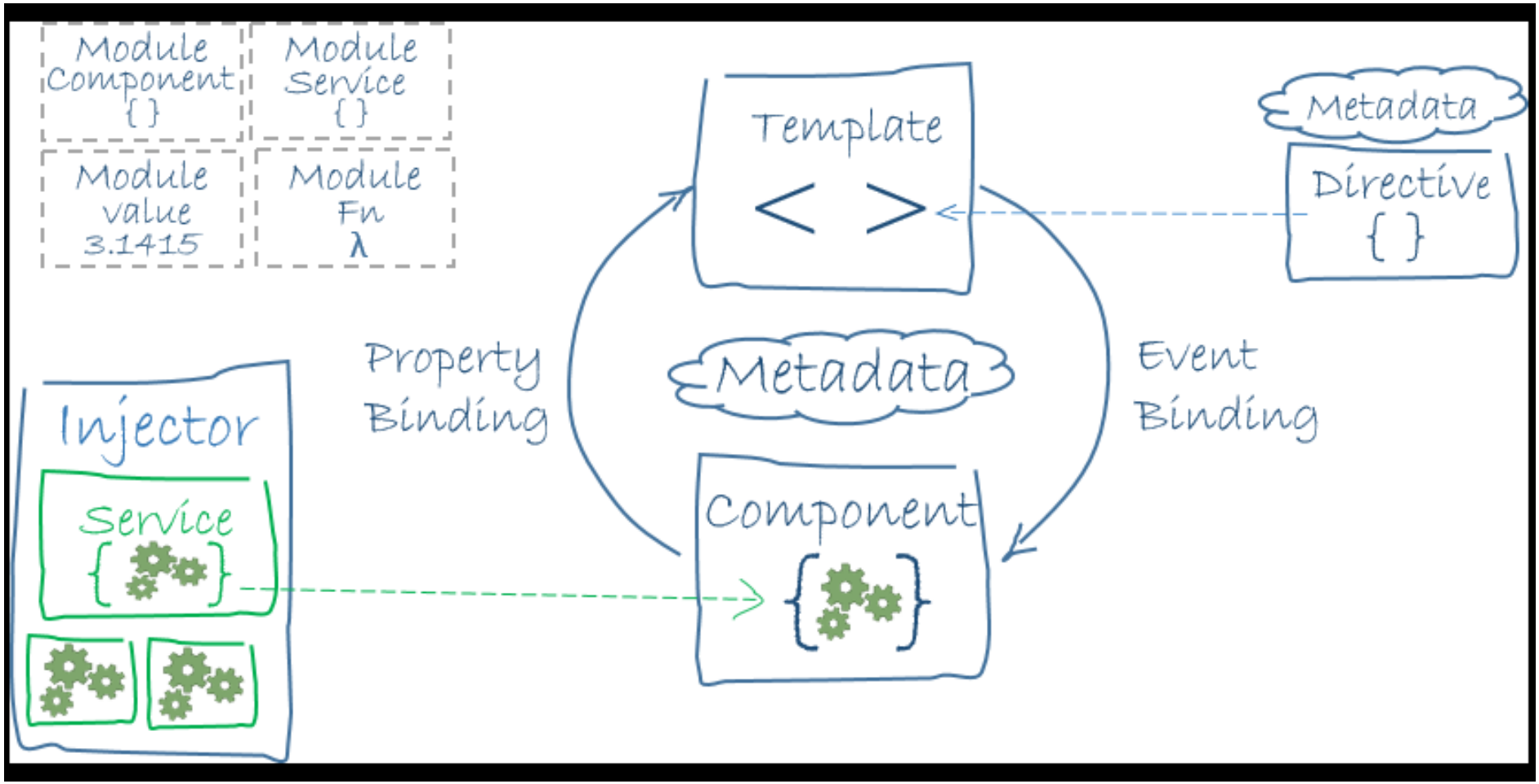
Veamos como se relacionan estos elementos en el diagrama de arquitectura típico, sacado de la web de Angular 2:

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

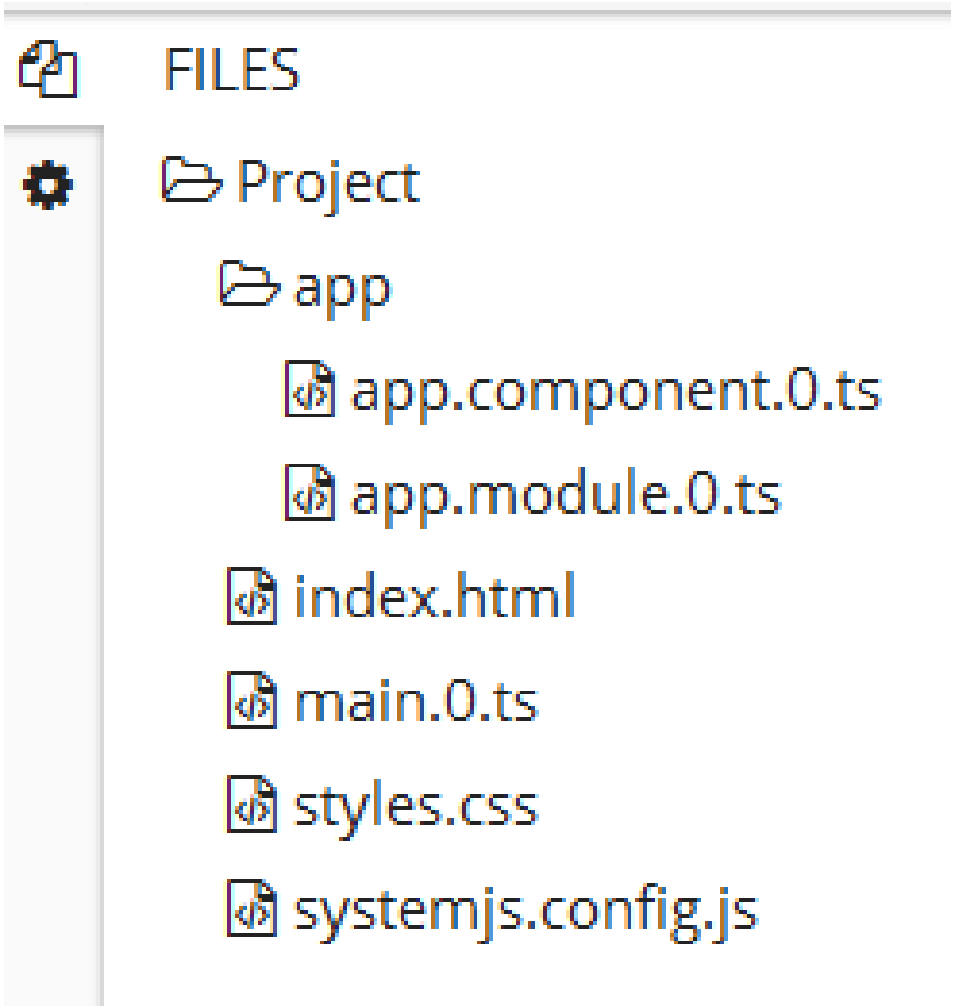
Para importar los elementos ***Component*** y ***Directive*** de ***@angular/core***, lo haríamos del siguiente modo:

```
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

```
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: {
```

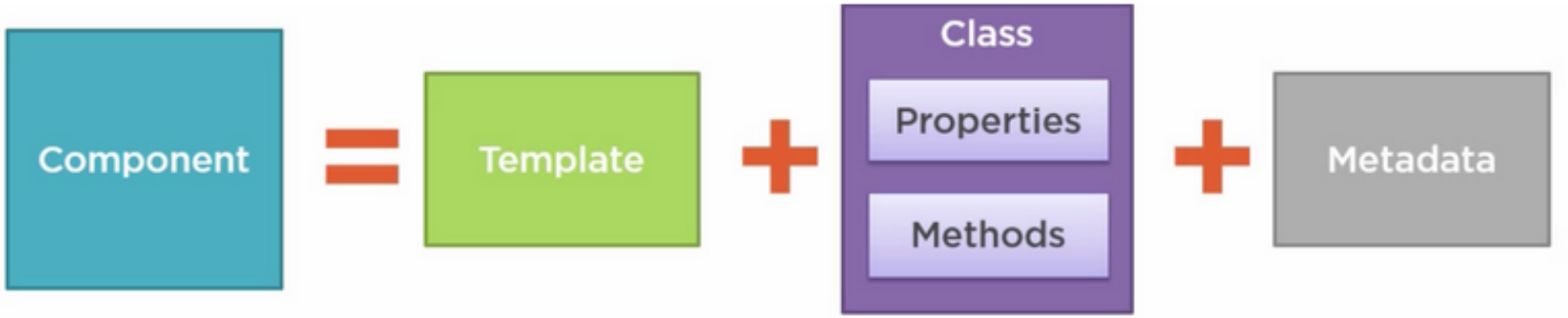
Módulos en Angular 2

```
    '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';
}
```

- Cada componente comienza con una función de @Component **decoradora** que toma un objeto de metadatos .
- El objeto de metadatos describe cómo funcionan la plantilla HTML y la clase de componente.

Componentes

Index.html (dentro de <body>)

```
<My-app> Cargando contenido de AppComponent aquí ... </ my-app>
```

- La propiedad del **selector** indica a Angular que muestre el componente dentro de una etiqueta index.html <my-app> en el index.html.
- La template define un mensaje dentro de un <h1> .
- El mensaje comienza con "Hola" y termina con {{name}} , que es una expresión de enlace de interpolación Angular.
- En tiempo de ejecución, Angular reemplaza {{name}} por el valor de la propiedad name del componente.
- El enlace de interpolación es una de las muchas características de angular.

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    });
```

Componentes: Otro ejemplo

Cadenas de plantilla de varias líneas

```
template:`
  <h1>{{title}}</h1>
  <h2>{{hero.name}} details!</h2>
  <div><label>id: </label>{{hero.id}}</div>
  <div>
    <label>name: </label>
    <input value="{{hero.name}}" placeholder="name">
  </div>
  `
```

```
export class Hero {
  id: number;
  name: string;
}
```

Componentes: Otro ejemplo

- Antes de poder utilizar el enlace de datos FormsModule para las FormsModule , FormsModule importar el paquete FormsModule en nuestro módulo Angular.
- Lo añadimos a la NgModule importaciones del decorador de NgModule.
- Esta matriz contiene la lista de módulos externos utilizados por nuestra aplicación.
- Ahora hemos incluido el paquete de ngModel que incluye ngModel .

Componentes: Otro ejemplo

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

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

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

```
<input [(ngModel)]="hero.name" placeholder="name">
```

Componentes: Otro ejemplo

```
const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynama' },
  { id: 18, name: 'Dr IQ' },
  { id: 19, name: 'Magma' },
  { id: 20, name: 'Tornado' }
];
```

Componentes: Otro ejemplo

Mostrar héroes en una plantilla

Nuestro componente tiene `heroes` . Vamos a crear una lista desordenada en nuestra plantilla para mostrarlos. Insertaremos el siguiente fragmento de HTML debajo del título y encima de los detalles del héroe.

Src / app / app.component.ts (plantilla de héroes)

```
1.    <H2> Mis Héroes </ h2> <ul class = "héroes"> <li> <! - cada héroe
va aquí -> </ li> </ ul>
```

Ahora tenemos una plantilla que podemos llenar con nuestros héroes.

Componentes: Otro ejemplo

Podríamos haber definido la lista de héroes aquí en esta clase de componentes. Pero sabemos que en última instancia conseguiremos a los héroes de un servicio de datos. Debido a que sabemos a dónde nos dirigimos, tiene sentido separar los datos del héroe de la implementación de clase desde el principio.

Componentes: Otro ejemplo

Mostrar héroes en una plantilla

Nuestro componente tiene `heroes` . Vamos a crear una lista desordenada en nuestra plantilla para mostrarlos. Insertaremos el siguiente fragmento de HTML debajo del título y encima de los detalles del héroe.

Src / app / app.component.ts (plantilla de héroes)

```
1.    <H2> Mis Héroes </ h2> <ul class = "héroes"> <li> <! - cada héroe
va aquí -> </ li> </ ul>
```

Ahora tenemos una plantilla que podemos llenar con nuestros héroes.

Componentes: Otro ejemplo

Listado de héroes con ngFor

Queremos vincular la matriz de `heroes` en nuestro componente a nuestra plantilla, iterar sobre ellos y mostrarlos individualmente. Necesitaremos ayuda de Angular para hacer esto. Hagamos esto paso a paso.

Primero modifica la etiqueta `<li>` añadiendo la directiva incorporada `*ngFor` .

src/app/app.component.ts (ngFor)

```
<li *ngFor="let hero of heroes">
```

El `ngFor` (`*`) delante de `ngFor` es una parte crítica de esta sintaxis.

Componentes: Otro ejemplo

El prefijo (`*`) a `ngFor` indica que el elemento `<li>` y sus hijos constituyen una plantilla maestra.

La `ngFor` ngFor itera sobre el array de `heroes` devuelto por la `AppComponent.heroes` y sella las instancias de esta plantilla.

El texto citado asignado a `ngFor` " *tomar cada héroe en la matriz de* `heroes` *, almacenarlo en la variable de* `hero` *local y ponerlo a disposición de la instancia de plantilla correspondiente* ".

La palabra clave `let` antes de "héroe" identifica al `hero` como una variable de entrada de plantilla. Podemos hacer referencia a esta variable dentro de la plantilla para acceder a las propiedades de un héroe.

Componentes: Otro ejemplo

Micro sintaxis

La microescritura angular le permite configurar una directiva en una cadena compacta y amigable. El analizador de microsyntax traduce esa cadena en atributos en la `<template>` plantilla `<template>` :

- La palabra clave `let` indica una *variable de entrada de* plantilla a la que se hace referencia dentro de la plantilla. Las variables de entrada en este ejemplo son `hero` , `i` y `odd` . El parser traduce `let hero` , `let i` y `let odd` en variables nombradas, `let-hero` , `let-i` y `let-odd` .

- El analizador de microsyntax toma `of` y `trackby` , title-cases ellos (`of` -> `Of` , `trackBy` -> `TrackBy`), y prefijos con el nombre de atributo de la directiva (`ngFor`), dando los nombres `ngForOf` y `ngForTrackBy` . Estos son los nombres de dos `NgFor` *entrada NgFor* . Así es como la directiva aprende que la lista es `heroes` y la función track-by es `trackById` .

- A medida que la `NgFor` NgFor recorre la lista, establece y restablece las propiedades de su propio objeto de *contexto* . Estas propiedades incluyen `index` y `odd` y una propiedad especial denominada `$implicit` .

- Las variables `let-i` y `let-odd` definieron como `let i=index` y `let odd=odd` . Angular los establece en el valor actual del `index` del contexto y las propiedades `odd` .

Componentes: Otro ejemplo

src/app/app.component.ts (ngFor template)

```
<li *ngFor="let hero of heroes">
  <span class="badge">{{hero.id}}</span> {{hero.name}}
</li>
```

Componentes: Otro ejemplo

Elegir a nuestros héroes

Nuestra lista de héroes se ve bastante sosa. Queremos hacer que sea visualmente obvio para un usuario sobre cuál héroe estamos flotando y qué héroe está seleccionado.

Añadamos algunos estilos a nuestro componente `@Component` la `@Component` styles en el `@Component` @Component en las siguientes clases CSS:

Componentes: Otro ejemplo

src/app/app.component.ts (styles)

```
styles: [`
  .selected {
    background-color: #CFD8DC !important;
    color: white;
  }
  .heroes {
    margin: 0 0 2em 0;
    list-style-type: none;
    padding: 0;
    width: 15em;
  }
  .heroes li {
    cursor: pointer;
    position: relative;
    left: 0;
    background-color: #EEE;
    margin: .5em;
    padding: .3em 0;
    height: 1.6em;
    border-radius: 4px;
  }
```

```
.heroes li.selected:hover {
  background-color: #BBD8DC !important;
  color: white;
}
.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}
.heroes .text {
  position: relative;
  top: -3px;
}
.heroes .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607D8B;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
```

Componentes: Otro ejemplo

Observe que usamos nuevamente la notación back-tick para cadenas de varias líneas.

Eso es un montón de estilos! Podemos ponerlos en línea como se muestra aquí, o podemos moverlos a su propio archivo que hará más fácil el código de nuestro componente. Lo haremos en un capítulo posterior. Por ahora vamos a seguir rodando.

Cuando asignamos estilos a un componente, éstos tienen un alcance para ese componente específico. Nuestros estilos solo se `AppComponent` a nuestro `AppComponent` y no "se `AppComponent` " al HTML externo.

Componentes: Otro ejemplo

Selección de un héroe

Tenemos una lista de héroes y tenemos un solo héroe que se muestra en nuestra aplicación. La lista y el solo héroe no están conectados de ninguna manera. Queremos que el usuario seleccione un héroe de nuestra lista y que el héroe seleccionado aparezca en la vista de detalles. Este patrón de interfaz de usuario es ampliamente conocido como "master-detail". En nuestro caso, el maestro es la lista de héroes y el detalle es el héroe seleccionado.

Vamos a conectar el maestro al detalle a través de una `selectedHero` componente selectedHero enlazada a un evento de clic.

Componentes: Otro ejemplo

Evento de clic

Modificamos el `<li>` insertando un evento Angular vinculante a su evento click.

src/app/app.component.ts (template excerpt)

```
<li *ngFor="let hero of heroes" (click)="onSelect(hero)">
  <span class="badge">{{hero.id}}</span> {{hero.name}}
</li>
```

Enfoque en el evento vinculante

```
(Click) = "onSelect (héroe)"
```

Componentes: Otro ejemplo

Los paréntesis identifican el evento de `click` del elemento `<li>` como destino. La expresión a la derecha del signo de `AppComponent` llama al método `onSelect()` , `onSelect()` , pasando el `hero` variable de entrada de `onSelect()` . Esa es la misma variable de `hero` que definimos anteriormente en el `ngFor` .

Agregar el controlador de clics

Nuestra `onSelect` eventos se refiere a un método `onSelect` que aún no existe. Vamos a añadir ese método a nuestro componente ahora.

¿Qué debe hacer ese método? Debe establecer el héroe seleccionado del componente para el héroe que el usuario ha hecho clic.

Nuestro componente aún no tiene un "héroe seleccionado". Empezaremos allí.

Componentes: Otro ejemplo

Exponer el héroe seleccionado

Ya no `AppComponent` la `AppComponent` static `hero` de `AppComponent` . `selectedHero` por esta sencilla `selectedHero` :

src/app/app.component.ts (selectedHero)

```
  selectedHero: Hero;
```

Hemos decidido que ninguno de los héroes debe ser seleccionado antes de que el usuario elija un héroe por lo que no se inicializará la `selectedHero` como `selectedHero` haciendo con el `hero` .

A **`onSelect`** agregue un método **`onSelect`** que `selectedHero` la `selectedHero` al `hero` que haya hecho clic en el usuario.

src/app/app.component.ts (onSelect)

```
1.  onSelect(hero: Hero): void {
2.    this.selectedHero = hero;
3.  }
```

Componentes: Otro ejemplo

Mostraremos los detalles del héroe seleccionado en nuestra plantilla. Por el momento, todavía se refiere a la propiedad del antiguo `hero` . Vamos a arreglar la `selectedHero` para vincular a la nueva `selectedHero` .

src/app/app.component.ts (template excerpt)

```
<h2>{{selectedHero.name}} details!</h2>
<div><label>id: </label>{{selectedHero.id}}</div>
<div>
    <label>name: </label>
    <input [(ngModel)]="selectedHero.name" placeholder="name"/>
</div>
```

Componentes: Otro ejemplo

Ocultar el detalle vacío con ngIf

Cuando nuestra aplicación carga vemos una lista de héroes, pero un héroe no está seleccionado. El `selectedHero` no está `undefined` . Es por eso que veremos el siguiente error en la consola del navegador:

```
EXCEPTION: TypeError: No se puede leer la propiedad 'name' de undefined en [null]
```

Recuerda que estamos mostrando `selectedHero.name` en la `selectedHero.name` . Esta `selectedHero` nombre no existe porque `selectedHero` no está definido.

Abordaremos este problema guardando el detalle del héroe del DOM hasta que haya un héroe seleccionado.

Envolvemos el contenido del detalle del héroe HTML de nuestra plantilla con un `<div>` . A `ngIf` la `ngIf` ngIf y la `selectedHero` en la `selectedHero` de nuestro componente.

Componentes: Otro ejemplo

src/app/app.component.ts (ngIf)

```
<div *ngIf="selectedHero">
  <h2>{{selectedHero.name}} details!</h2>
  <div><label>id: </label>{{selectedHero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="selectedHero.name" placeholder="name"/>
  </div>
</div>
```

Remember that the leading asterisk (`*`) in front of `ngIf` is a critical part of this syntax.

Componentes: Otro ejemplo

Cuando no hay `selectedHero` , la `ngIf` ngIf elimina el HTML del detalle del héroe del DOM. No habrá elementos de detalle de héroe ni enlaces para preocuparse.

Cuando el usuario `selectedHero` un héroe, `selectedHero` se `selectedHero` en "truthy" y `ngIf` pone el `ngIf` del detalle del héro en el DOM y evalúa los enlaces anidados.

> `ngIf` y `ngFor` se `ngFor` "directivas estructurales" porque pueden cambiar la estructura de porciones del DOM. En otras palabras, dan estructura a la forma en que Angular muestra contenido en el DOM.
>
> Obtenga más `ngIf` sobre `ngIf` , `ngFor` y otras directivas estructurales en los capítulos de directivas estructurales y sintaxis de plantillas .

El navegador se actualiza y vemos la lista de héroes pero no el detalle del héroe seleccionado. El `ngIf` lo mantiene fuera del DOM mientras el `selectedHero` no esté definido. Cuando hacemos clic en un héroe en la lista, el héroe seleccionado aparece en los detalles del héroe. Todo está funcionando como esperamos.

Componentes

Lifecycle

El ciclo de vida de un componente