



ever
FUTURE



Angular (2.0 y siguientes) con TypeScript

Orientado a profesionales que están empezando a desarrollar con el framework Angular basado en TypeScript en cualquiera de las versiones 2.0 o siguientes.

Angular (2.0 y siguientes) con TypeScript

ever
FUTURE



ÍNDICE:

1. Qué es Angular
2. Instalación en Windows
3. TypeScript
4. Módulos y Decoradores
5. Estructura de un proyecto Angular
6. Componentes
7. Agregando Bootstrap
8. Trabajando con Componentes



01

Qué es Angular

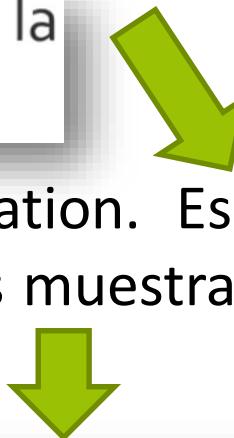




1. Qué es Angular

La **última tendencia** en el desarrollo web es la implementación de aplicaciones **web SPA**

SPA son las siglas de Single Page Application. Es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador.



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**

1. Qué es Angular

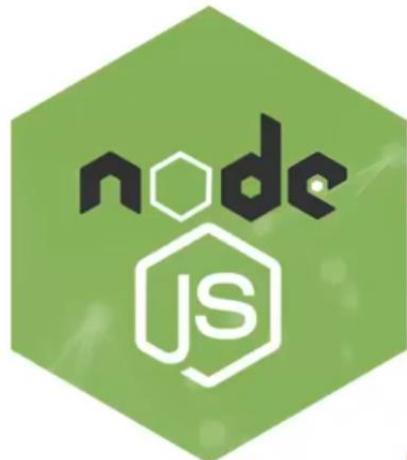
Pero, ¿qué es Angular?

- No es una biblioteca JavaScript. No hay funciones que podamos llamar y utilizar directamente.
- No es una biblioteca de manipulación DOM como jQuery. Pero utiliza el subconjunto de jQuery para la manipulación de DOM (llamado jqLite).
- AngularJS es un framework Javascript MVC creado por Google para crear aplicaciones web adecuadamente diseñadas y mantenibles de Tipo SPA.
- **Angular no tiene relación con AngularJS**, ya que tras su primera versión fue reescrito al completo.



1. Qué es Angular

¿Qué vamos a necesitar para trabajar?

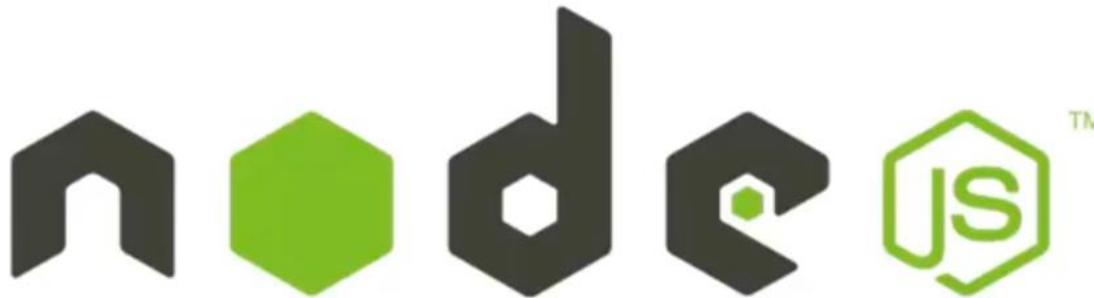


Node JS nos proporcionará la infraestructura. **Chrome** las herramientas de desarrollo/debug. **TypeScript** como lenguaje de desarrollo. Y el **cliente Angular** para el trabajo con la consola.



1. Qué es Angular

¿Qué vamos a necesitar para trabajar?



Plataforma para ejecutar aplicaciones JS fuera del navegador



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



02

Instalación en Windows

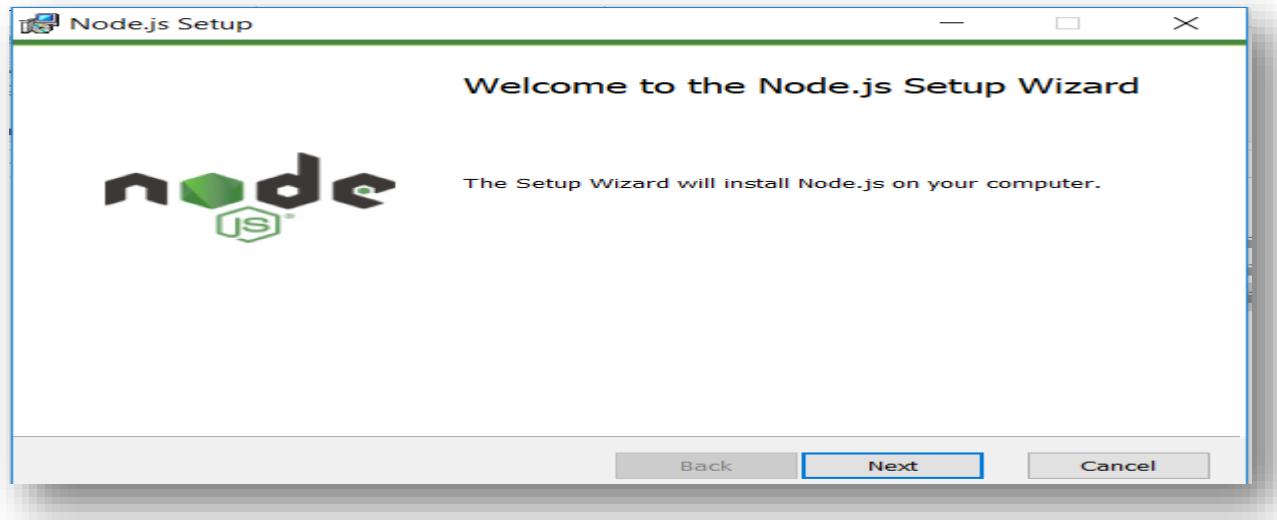


2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de ™

Accedemos a: <https://nodejs.org> y nos bajamos la ‘Recommended For Most Users’.

Tras descargarla, la instalaremos.



Pulsamos ‘siguiente’ en todas las ocasiones y por último ‘Instalar’.



2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de  node JS™

Una vez finalizada la instalación, vamos a comprobar que se ha instalado correctamente, para ello abriremos la consola y ejecutaremos 'node -v':

```
C:\ Símbolo del sistema
Microsoft Windows [Versión 10.0.16299.1686]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32> node -v
v12.16.1
```

Tras ello lanzamos las actualizaciones con 'npm install npm@latest -g'.

Y tras finalizar comprobamos la versión de npm instalada con 'npm -v':

```
+ npm@6.14.2
added 435 packages from 866 contributors in 54.125s

C:\WINDOWS\system32> npm -v
6.14.2
```



2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de [TypeScript](#)

TypeScript es el lenguaje que vamos a utilizar de base en nuestra programación, para generar un código **JavaScript** más limpio con una correcta orientación a objetos.

Para instalarlo escribiremos en consola ‘npm install –g typescript’.

```
C:\WINDOWS\system32> npm install -g typescript
C:\Users\jhernand\AppData\Roaming\npm\tsc -> C:\Users\jhernand\AppData\Roaming\npm\node_modules\typescript\bin\tsc
C:\Users\jhernand\AppData\Roaming\npm\tsserver -> C:\Users\jhernand\AppData\Roaming\npm\node_modules\typescript\bin\tsserver
+ typescript@3.8.3
added 1 package from 1 contributor in 13.753s
```

Comprobamos la versión con ‘tsc –v’

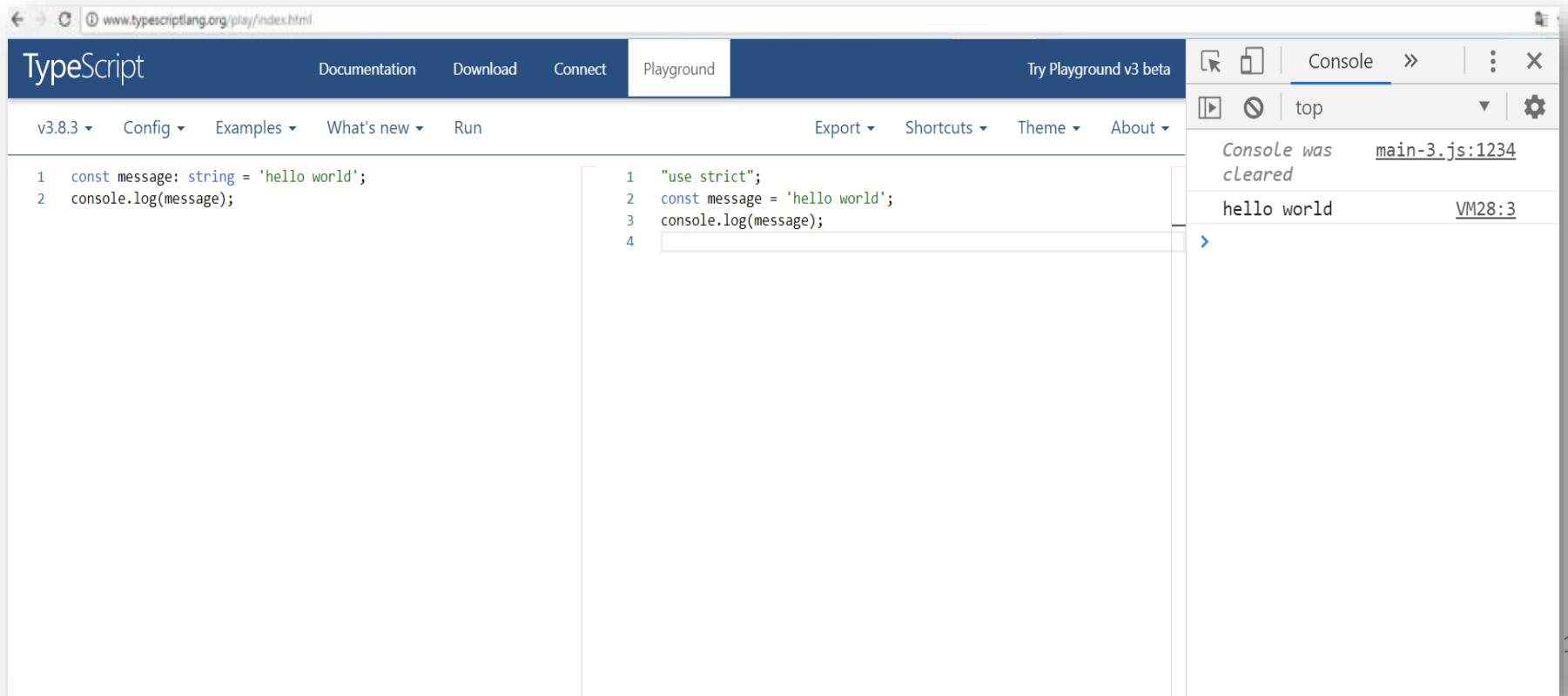


```
C:\WINDOWS\system32> tsc -v
Version 3.8.3
```

2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de **TypeScript**

Desde el enlace <https://www.typescriptlang.org/play/index.html> vamos a poder escribir y probar nuestro código **TypeScript**:



The screenshot shows the TypeScript Playground interface. The top navigation bar includes links for Documentation, Download, Connect, and Playground (which is currently selected). Below the navigation is a toolbar with version information (v3.8.3), configuration dropdowns, and links for Examples, What's new, Run, Export, Shortcuts, Theme, and About. The main workspace contains two blocks of TypeScript code:

```
1 const message: string = 'hello world';
2 console.log(message);
```

```
1 "use strict";
2 const message = 'hello world';
3 console.log(message);
4
```

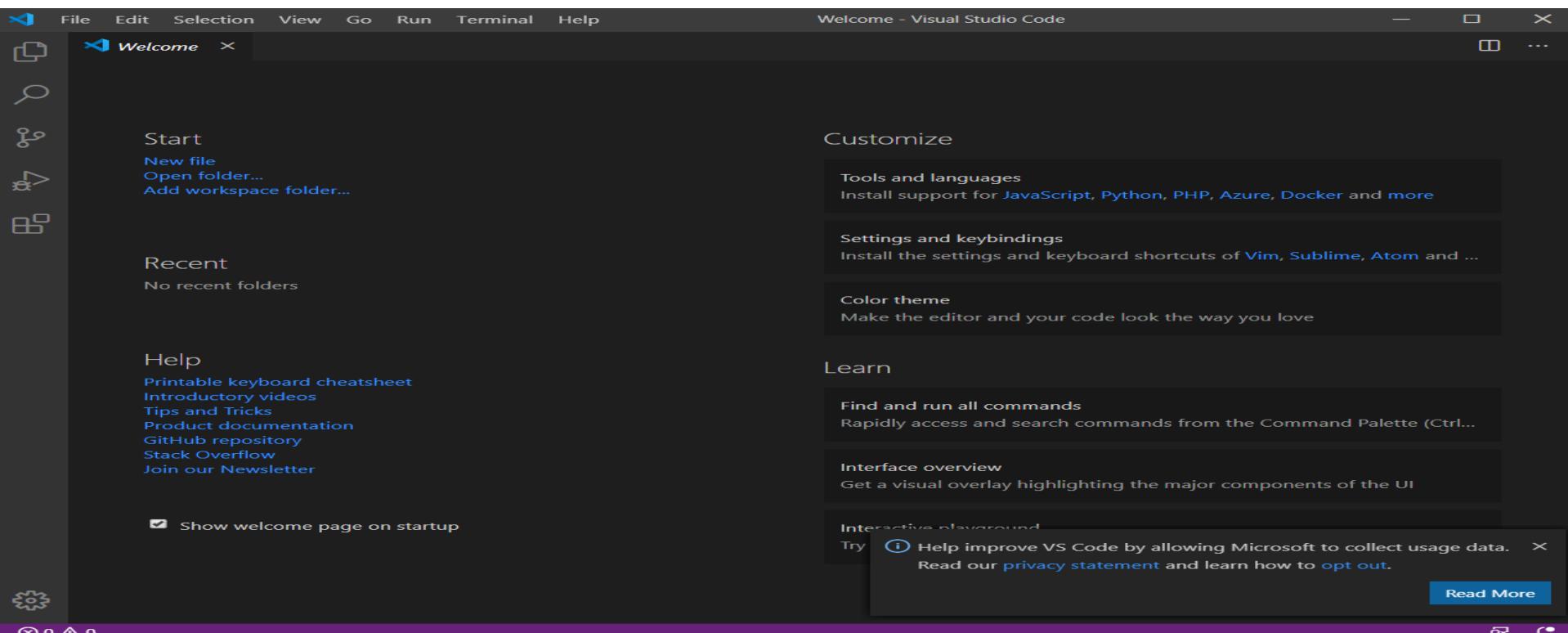
To the right of the code is a Console panel showing the output of the executed code:

```
Console was cleared
main-3.js:1234
hello world VM28:3
```

2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de Visual Studio

Desde el enlace <https://code.visualstudio.com/> nos descargaremos el IDE de desarrollo.

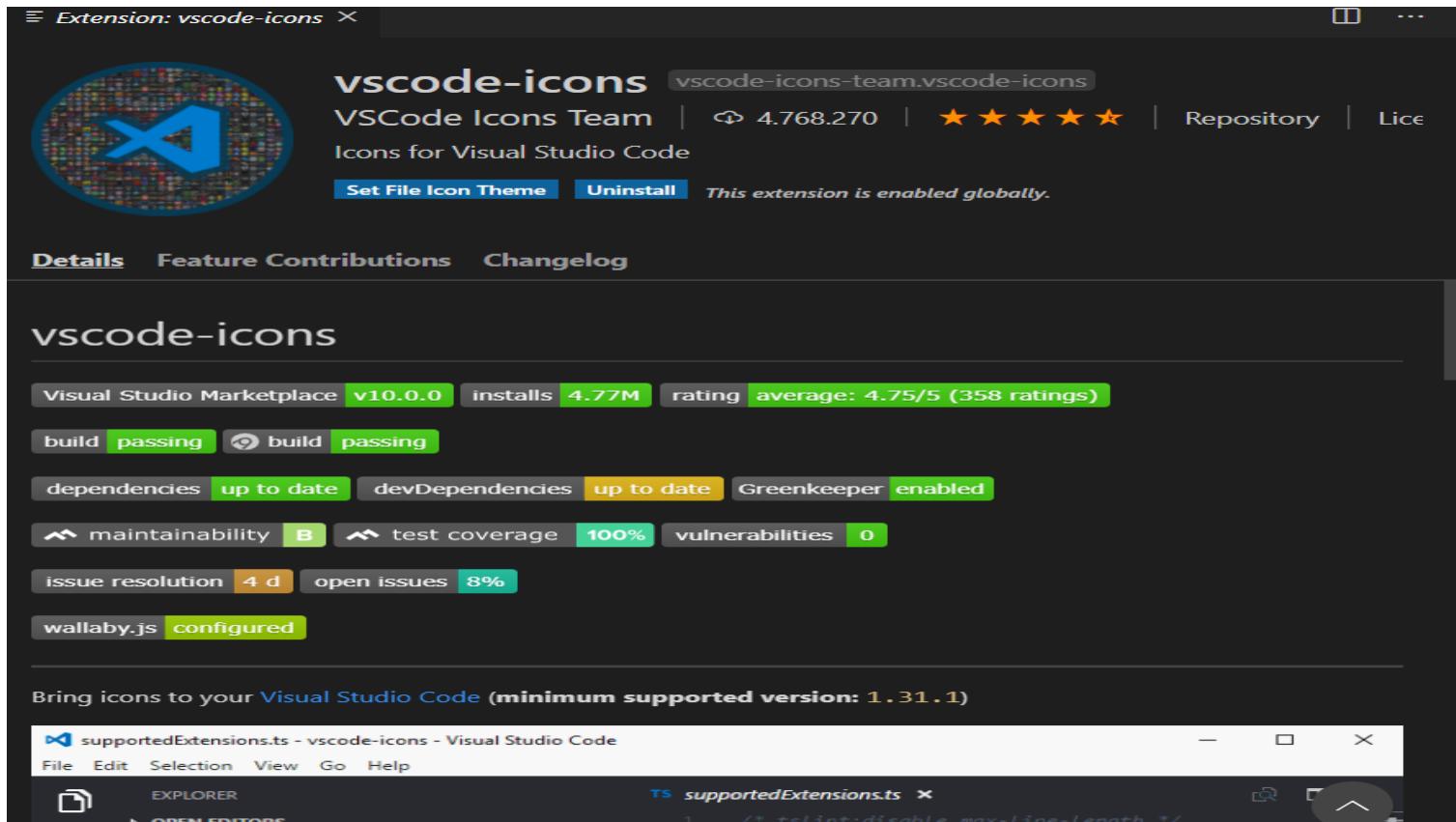


Para cambiar Visual Studio de idioma: Control+Shift+P → Configure Display Language

2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de Visual Studio

Posteriormente nos meteremos en la web y en la sección de extensiones y nos bajaremos la vscode-icons (también puede instalarse desde la sección de extensiones del IDE):



The screenshot shows the Visual Studio Marketplace page for the "vscode-icons" extension. The extension is developed by the "VSCode Icons Team" and has a version of 4.768.270. It has a rating of 4.75/5 based on 358 ratings. The extension is enabled globally. Below the main card, there's a summary of build status, dependencies, and other metrics like maintainability (B), test coverage (100%), and vulnerabilities (0). A note at the bottom encourages users to bring icons to their Visual Studio Code environment.

Extension: vscode-icons

vscode-icons vscode-icons-team.vscode-icons

VSCode Icons Team | 4.768.270 | ★★★★★ | Repository | License

Icons for Visual Studio Code

[Set File Icon Theme](#) [Uninstall](#) This extension is enabled globally.

[Details](#) [Feature Contributions](#) [Changelog](#)

vscode-icons

Visual Studio Marketplace v10.0.0 installs 4.77M rating average: 4.75/5 (358 ratings)

build passing build passing

dependencies up to date devDependencies up to date Greenkeeper enabled

maintainability B test coverage 100% vulnerabilities 0

issue resolution 4 d open issues 8%

wallaby.js configured

Bring icons to your Visual Studio Code (minimum supported version: 1.31.1)

supportedExtensions.ts - vscode-icons - Visual Studio Code

File Edit Selection View Go Help

EXPLORER OPEN EDITORS

TS supportedExtensions.ts

```
/* tslint:disable:max-line-length */
```

2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de  ANGULAR CLI

La página de referencia es la siguiente: <https://cli.angular.io/>

Primeramente desde consola vamos a ejecutar ‘**npm install -g @angular/cli**’

```
C:\WINDOWS\system32> npm install -g @angular/cli
npm WARN deprecated mkdirp@0.5.3: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note that t
he API surface has changed to use Promises in 1.x.)
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
C:\Users\jhernand\AppData\Roaming\npm\ng -> C:\Users\jhernand\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

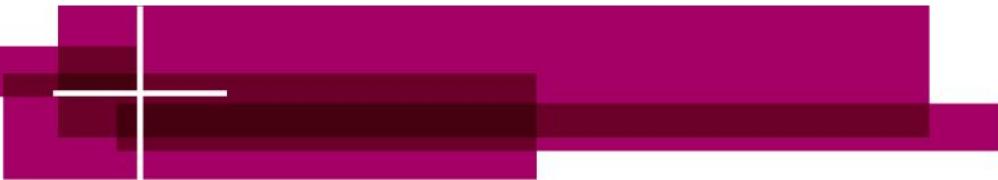
> @angular/cli@9.0.6 postinstall C:\Users\jhernand\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js

? Would you like to share anonymous usage data with the Angular Team at Google under
Google's Privacy Policy at https://policies.google.com/privacy? For more details and
how to change this setting, see http://angular.io/analytics. Yes

Thank you for sharing anonymous usage data. If you change your mind, the following
command will disable this feature entirely:

  ng analytics off

+ @angular/cli@9.0.6
added 260 packages from 205 contributors in 149.97s
```



2. Instalar en Windows NodeJS, TypeScript, Visual Studio code y Angular

Instalación de ANGULAR CLI

Tras ello validaremos la instalación con ‘ng v’:

```
C:\WINDOWS\system32>ng v

Angular CLI: 9.0.6
Node: 12.16.1
OS: win32 x64

Angular:
...
Ivy Workspace:

Package          Version
-----
@angular-devkit/architect    0.900.6
@angular-devkit/core         9.0.6
@angular-devkit/schematics   9.0.6
@schematics/angular          9.0.6
@schematics/update           0.900.6
rxjs                6.5.3
```



03 TypeScript



3. TypeScript

¿Qué es TypeScript?

TypeScript viene a solucionar muchos de los problemas que trae consigo los grandes desarrollos basados en **JavaScript**.

- TypeScript es un lenguaje que nos permite tener una orientación a objetos más limpia y potente en JavaScript, además añade un tipado fuerte y es el lenguaje que utilizamos para programar aplicaciones web con Angular que es uno de los frameworks más populares para desarrollar aplicaciones modernas y escalables en el lado del cliente.
- TypeScript es un lenguaje libre desarrollado por Microsoft y es un superconjunto de JavaScript que gracias a las ventajas que ofrece está siendo cada vez más utilizado.
- TypeScript es un lenguaje “compilado” → nosotros escribimos código TypeScript y el compilador (transpilador) lo traduce a código JavaScript que el navegador podrá interpretar.

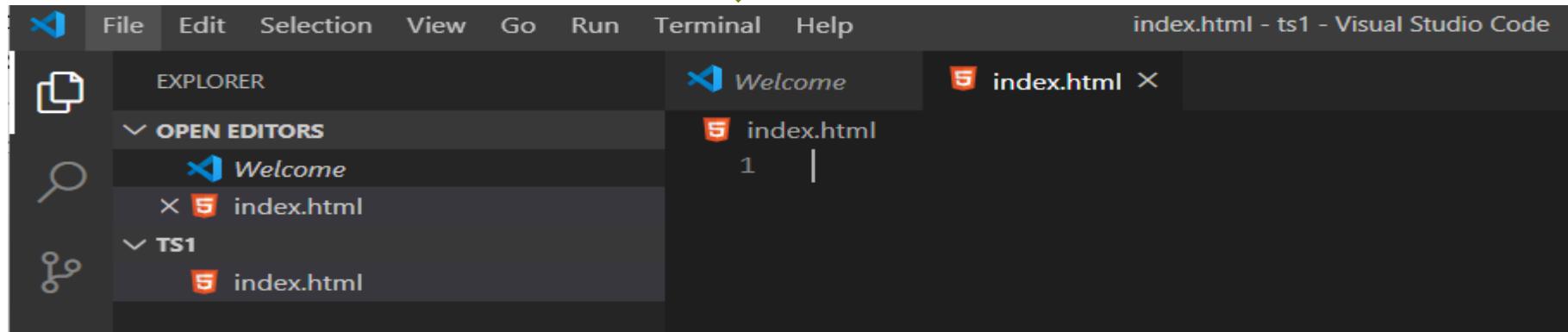
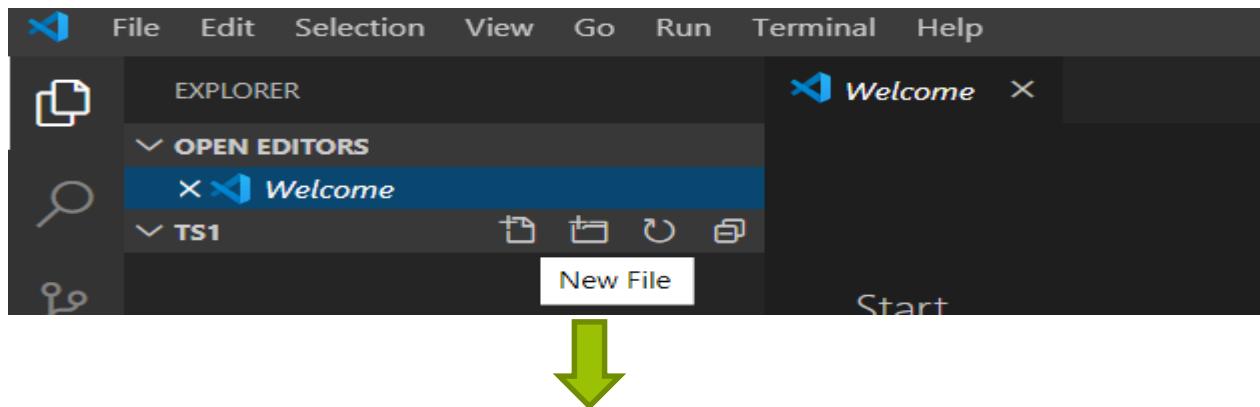
Transpilar → convertir código TypeScript en código JavaScript

3. TypeScript

Mi primer 'Hola mundo' con TypeScript

Primeramente vamos a crear un directorio de trabajo: `c:\dev\workspace\username\ts1`

Tras ello crearemos un fichero el fichero '`index.html`'





3. TypeScript

Mi primer 'Hola mundo' con TypeScript

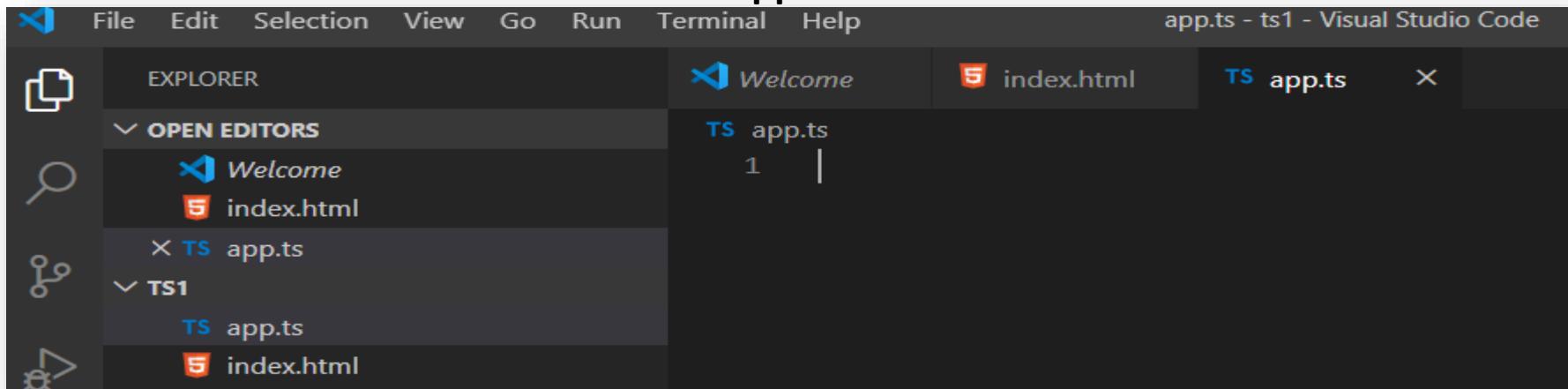
El contenido inicial que le daremos será el siguiente:

```
5 index.html > ...
1   <html>
2     <head>
3       <meta charset="UTF-8">
4       <title>Mi primer TypeScript</title>
5     </head>
6     <body>
7       <script src="app.js"></script>
8     </body>
9   </html>
```

3. TypeScript

Mi primer 'Hola mundo' con TypeScript

Tras ello crearemos un archivo llamado '**app.ts**':



The screenshot shows the Visual Studio Code interface. The title bar reads "app.ts - ts1 - Visual Studio Code". The left sidebar has icons for Explorer, Search, Symbols, and Terminal. The Explorer section shows "OPEN EDITORS" with files "Welcome", "index.html", and "app.ts" (which is currently selected). The main editor area shows the "Welcome" page with the file "app.ts" open, containing the code "1 console.log("Hola mundo TypeScript");".

El contenido será:

```
TS app.ts
1   console.log("Hola mundo TypeScript");|
```

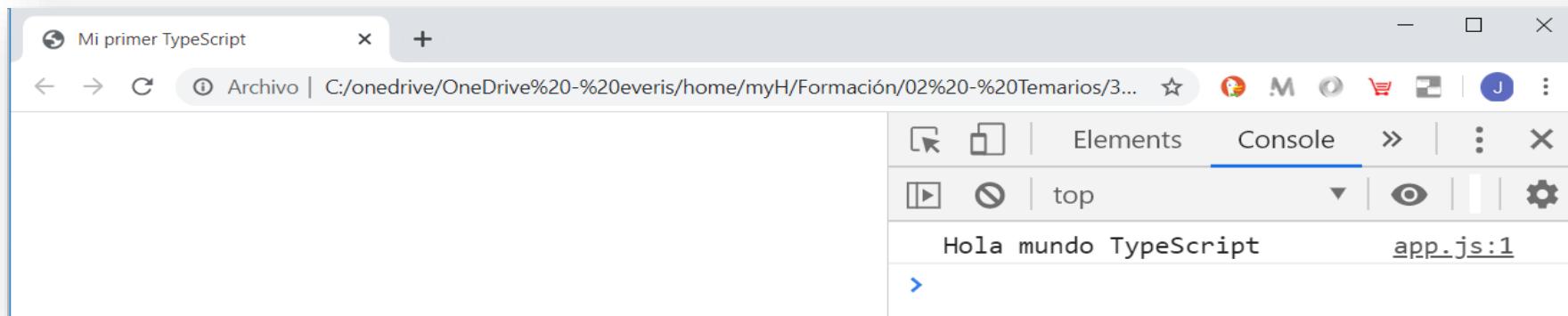
Tras ello habilitamos el terminal de comandos desde: **menú 'View – Terminal'**

Una vez abierto escribiremos: **tsc app.ts** para que se nos genere automáticamente el archivo **app.js**

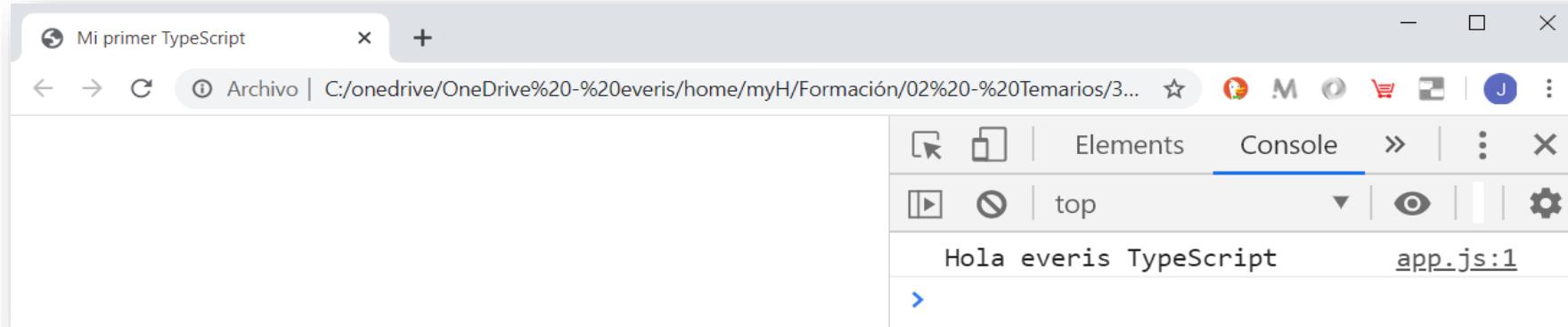
3. TypeScript

Mi primer 'Hola mundo' con TypeScript

Ahora si abrimos el fichero **index.html** con la opción de consola activada, veremos:



Si queremos cambiar por ejemplo el mensaje a '**Hola everis TypeScript**', necesitamos cambiarlo en el fichero **app.ts** y volvemos a ejecutar '**tsc app.ts**' y al recargar el html veremos:





3. TypeScript

Mi primer 'Hola mundo' con TypeScript

Para no tener que estar ejecutando 'tsc' por cada cambio que hagamos, vamos a dejar ahora un escuchador abierto para que coja cualquier cambio que hagamos. Para ello ejecutaremos en consola '**tsc -w app.ts**'

Y cualquier cambio que hagamos en nuestro fichero app.ts, nos generará automáticamente la versión correspondiente del app.js

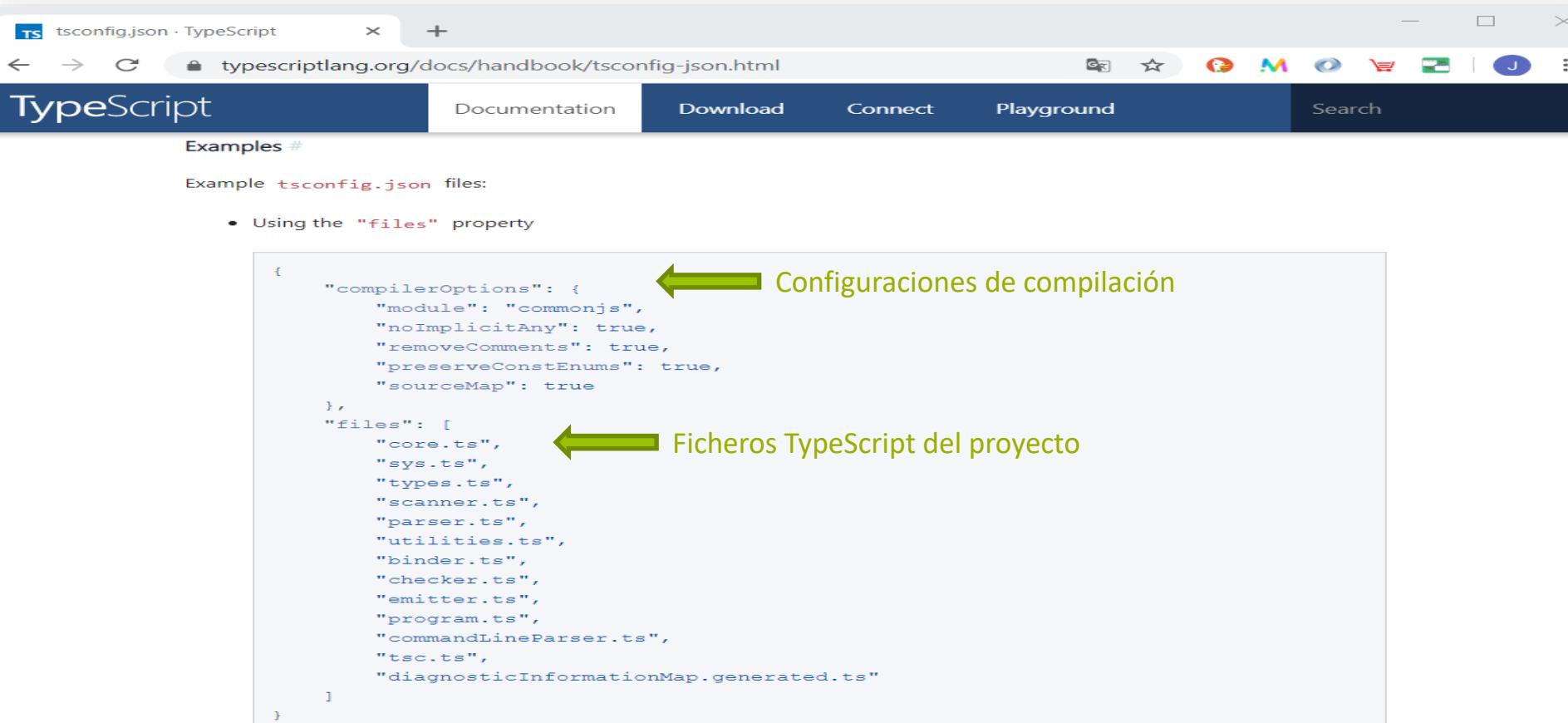
```
[13:30:10] File change detected. Starting incremental compilation...
[13:30:10] Found 0 errors. Watching for file changes.
```

3. TypeScript

El fichero tsconfig.json

Vamos a ver un fichero especial: **tsconfig.json**.

Es el fichero de configuración de un proyecto **TypeScript**.



tsconfig.json - TypeScript

typescriptlang.org/docs/handbook/tsconfig-json.html

TypeScript Documentation Download Connect Playground Search

Examples #

Example `tsconfig.json` files:

- Using the "files" property

```
{  
  "compilerOptions": {  
    "module": "commonjs",  
    "noImplicitAny": true,  
    "removeComments": true,  
    "preserveConstEnums": true,  
    "sourceMap": true  
  },  
  "files": [  
    "core.ts",  
    "sys.ts",  
    "types.ts",  
    "scanner.ts",  
    "parser.ts",  
    "utilities.ts",  
    "binder.ts",  
    "checker.ts",  
    "emitter.ts",  
    "program.ts",  
    "commandLineParser.ts",  
    "tsc.ts",  
    "diagnosticInformationMap.generated.ts"  
  ]  
}
```

Configuraciones de compilación

Ficheros TypeScript del proyecto

3. TypeScript

El fichero tsconfig.json

Ahora vamos a generar nuestro **tsconfig.json** ejecutando en el terminal '**tsc –init**':

```
TS tsconfig.json > ...
1  {
2    "compilerOptions": {
3      /* Basic Options */
4      // "incremental": true,
5      "target": "es5",
6      "module": "commonjs",
7      // "lib": [],
8      // "allowJs": true,
9      // "checkJs": true,
10     // "jsx": "preserve",
11     // "declaration": true,
12     // "declarationMap": true,
13     // "sourceMap": true,
14     // "outFile": "./",
15     // "outDir": "./",
16     // "rootDir": "./",
17     // "composite": true,
18     // "tsBuildInfoFile": "./",
19     // "removeComments": true,
20     // "noEmit": true,
21     // "importHelpers": true,
22     // "downlevelIteration": true,
23     // "isolatedModules": true,
```



3. TypeScript

let / var /const

En JavaScript podemos declarar las variables con **var** y con **let**. El primero tiene ámbito de bloque, el segundo no.

var

```
var foo = 123;
if (true) {
    var foo = 456;
}
console.log(foo); // 456
```

let

```
let foo = 123;
if (true) {
    let foo = 456;
}
console.log(foo); // 123
```



3. TypeScript

let / var /const

const nos va a permitir definir variables inmutables:

```
const foo = 123;
foo = 456; // NO permitido
```

Las constantes también admiten objetos literales como por ejemplo:

```
const foo = { bar: 123 };
foo = { bar: 456 }; // ERROR no se permite la modificación de objeto
```

Pero sí se puede modificar el contenido de las variables que contiene el objeto:

```
const foo = { bar: 123 };
foo.bar = 456; // Permitido
console.log(foo); // { bar: 456 }
```



3. TypeScript

Tipos de datos primitivos

Boolean

true o false

```
let isDone: boolean = false;
```

Number

Datos numéricos

```
let decimal: number = 6;
```

```
let hex: number = 0xf00d;
```

```
let binary: number = 0b1010;
```

```
let octal: number = 0o744
```

String

Cadenas de caracteres y/o textos

```
let color: string = "blue"; //  
color = 'red';
```

También se pueden utilizar "*Templates*" plantillas para concatenar strings como por ejemplo:

```
let fullName: string = 'Bob Bobbington';  
let age: number = 37;  
let sentence: string = `Hello, my name is ${ fullName }. I'll be ${ age + 1 } years old next month.`
```

Para poder utilizar esta sintaxis los string deben estar contenidos entre ` `.

Este tipo de sintaxis es el equivalente a:

```
let sentence: string = "Hello, my name is " + fullName + "." + "I'll be " + (age + 1)  
+ " years old next month."
```



3. TypeScript

Tipos de datos primitivos

Arrays, sino se les especifica tipo son **ANY**

ANY quiere decir que admite 'cualquier tipo' de dato

```
let list: number[] = [1, 2, 3];
```

Con esta sintaxis se puede especificar qué tipo de datos debe haber en el array

```
let list: Array<number> = [1, 2, 3];
```

Undefined

Es cuando un objeto o variable existe pero no tiene un valor. Si nuestro código interactúa con alguna API podemos recibir null como respuesta, para evaluar esas respuestas es mejor utilizar == en vez de ===

```
// ----- ejemplo.ts -----
console.log(undefined == undefined); // true
console.log(null == undefined); // true
console.log(0 == undefined); // false
console.log('' == undefined); // false
console.log(false == undefined); // false
```



3. TypeScript

Tipos de datos primitivos

Any

Puede ser cualquier tipo de objeto de javascript

```
let notSure: any = 4;
notSure = "maybe a string instead"; // typeof = string
notSure = false; // typeof = boolean
```

```
let notSure: any = 4;
notSure.toFixed(); // OK, toFixed existe, pero no es comprobado por el compilador
let prettySure: Object = 4;
prettySure.toFixed(); // Error: La propiedad 'toFixed' no existe en un 'Object'.
```

```
let list: any[] = [1, true, "free"];
list[1] = 100;
```



3. TypeScript

For in

For in es una característica que ya tenía javascript ,y no ha sido mejorada en TypeScript, mediante la cual puedes acceder y recorrer objetos y arrays y obtener tanto los índices

For in accediendo al valor de una variable dentro de un objeto:

TypeScript

```
let list = {a: 1, b: 2, c: 3};

for (let i in list) {
    console.log (i); // a, b, c
}
```



3. TypeScript

For Of

For of es una característica nueva de ES6 con la **cual puedes acceder y recorrer arrays y strings obteniendo su valor**, es decir, no puede recorrer objetos. Aunque se podrían recorrer objetos en el caso de que estos fueran creados por clases que implementen `Symbol.iterator`. `for ... of` también tiene un peor rendimiento en comparación con el `for...in` ya que al compilarlo a JS crea más variables y hace más comprobaciones.

For of accediendo al valor de una variable dentro de un array:

TypeScript

```
let list = ["a", "b", "c"];

for (let b of list) {
    console.log(b); // a, b, c
}
```



```
let list = {a: 1, b: 2, c: 3};

for (let i in list) {
    console.log(i); // a, b, c
}

let list2 = [1, 2, 3];

for (let i of list2) {
    console.log(i); // 1, 2, 3
}
```



3. TypeScript

Parámetros obligatorios por defecto y opcionales

Vamos a empezar definiéndonos una función en el app.ts:

```
index.html    TS app.ts    ×  
  
TS app.ts > ...  
1  function testFunTypeScript (param1: string, param2: string, param3: string) {  
2      console.log (" P1: " + param1);  
3      console.log (" P2: " + param2);  
4      console.log (" P3: " + param3);  
5  }  
6  
7  testFunTypeScript ("Esto es", "una formación", "de TypeScript");
```

Al actualizar el fichero 'index.html':

Mi primer TypeScript

Archivo | C:/onedrive/OneDrive%20-%20everis/home/myH/Formación/02%20-%20Tem...

Console

Message	Line Number
P1: Esto es	app.js:2
P2: una formación	app.js:3
P3: de TypeScript	app.js:4



3. TypeScript

Parámetros obligatorios por defecto y opcionales

Si intentamos llamar a nuestra función pasándole un único parámetro nos dará error:

```
testFunTypeScript ("Esto es");
```



```
app.ts:9:1 - error TS2554: Expected 3 arguments, but got 1.
```

```
9 testFunTypeScript ("Esto es");
```

```
app.ts:1:45
```

```
1 function testFunTypeScript (param1: String, param2: String, param3: String) {
```

```
    An argument for 'param2' was not provided.
```

```
[14:00:40] Found 1 error. Watching for file changes.
```

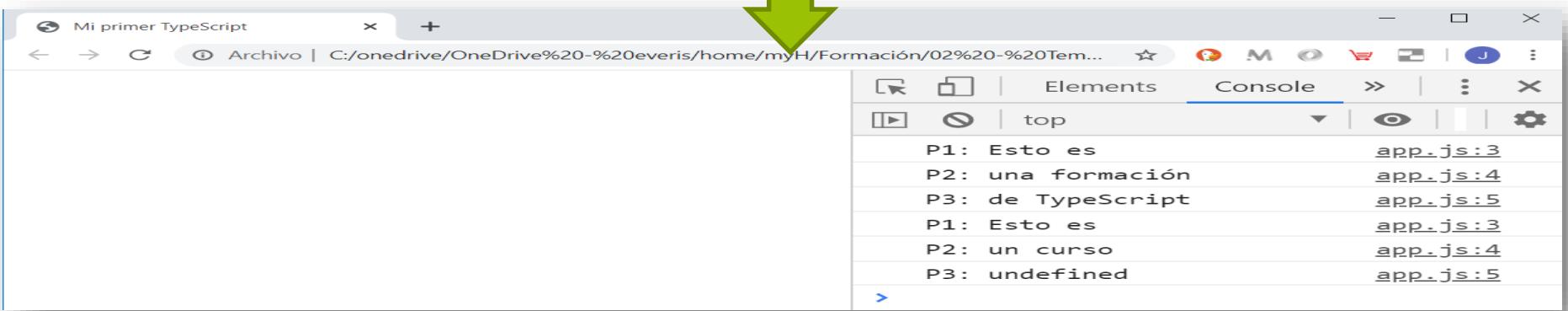


3. TypeScript

Parámetros obligatorios por defecto y opcionales

Ahora vamos a hacer la misma llamada pero antes vamos a modificar nuestra función indicando un valor por defecto para el segundo parámetro (en caso de no recibirla) y marcando el tercer parámetro como opcional:

```
function testFunTypeScript (param1: String, param2: String = "un curso", param3?: String) {  
    console.log (" P1: " + param1);  
    console.log (" P2: " + param2);  
    console.log (" P3: " + param3);  
}  
  
testFunTypeScript ("Esto es", "una formación", "de TypeScript");  
  
testFunTypeScript ("Esto es");
```



The screenshot shows a browser window titled "Mi primer TypeScript". The address bar indicates the file is located at "C:/onedrive/OneDrive%20-%20everis/home/myH/Formación/02%20-%20Tem...". The browser's developer tools are open, specifically the "Console" tab under the "Elements" section. A large green arrow points from the explanatory text above to this console tab. The console output displays the following log entries:

Log Entry	Line Number
P1: Esto es	app.js:3
P2: una formación	app.js:4
P3: de TypeScript	app.js:5
P1: Esto es	app.js:3
P2: un curso	app.js:4
P3: undefined	app.js:5



3. TypeScript

Arrow functions

```

nombre      parametros
  \        /
function add ( a, b ) {
    return a + b;    cuerpo/scope
}
                                |
retorno
  
```



```

nombre      parametros      flecha / scope
  \        |                /
var add = ( x,y ) => x + y;
                                |
retorno directo
  
```

```

function testSumaDosNumeros(param1: number, param2: number) {
    return param1 + param2;
}

console.log("1. Suma de 5 y 55 = "+ testSumaDosNumeros (5,55) ); // 60
  
```



```

var testSumaDosNumeros_v2 = (param1: number, param2: number) => param1 + param2;

console.log("2. Suma de 5 y 55 = "+ testSumaDosNumeros_v2 (5,55) ); // 60
  
```



3. TypeScript

Declaración de promesas en TypeScript

Desde **TypeScript** vamos a poder definir las **promesas** que tan imprescindibles se han vuelto en el contexto **JavaScript** con funciones dependientes de la ejecución de otras.
Lo único que nos cambiará respecto **JavaScript** tradicional será la nomenclatura en la definición de la función interna de la **promise**:

```
let mipromesa = new Promise(function (resolve: any, reject: any) {
    resolve();
});

mipromesa.then(function () {
    console.log('la operacion de la promesa a finalizado con exito');
}, function () {
    console.log('la operacion de la promesa a finalizado con error');
});
```

3. TypeScript

Programación Orientada a Objetos en TypeScript

Modificadores de acceso:

Private: Cuando un método o atributo (variable) es declarada como private, su uso queda restringido al interior de la misma clase, no siendo visible para el resto.

Protected: Un método o atributo definido como protected es visible para las clases que se encuentren en el mismo paquete y para cualquier subclase de esta aunque este en otro paquete. Este modificador es utilizado normalmente para Herencias, así que lo estudiaremos más a fondo cuando lleguemos a las Herencias.

Public: El modificador public ofrece la máxima visibilidad, una variable, método o clase con modificador public será visible desde cualquier clase, aunque estén en paquetes distintos

Por defecto las propiedades como modificador de acceso son públicas en typescript



3. TypeScript

Programación Orientada a Objetos en TypeScript

```
1 class Programa{  
2     public nombre: string;  
3     public version: number;  
4  
5     getNombre(){  
6         return this.nombre;  
7     }  
8  
9     setNombre(nombre:string){  
10        this.nombre = nombre;  
11    }  
12  
13    getVersion(){  
14        return this.version;  
15    }  
16  
17    setVersion(version:number){  
18        this.version = version;  
19    }  
20 }
```

← Definición de clase

← Métodos getters/setters

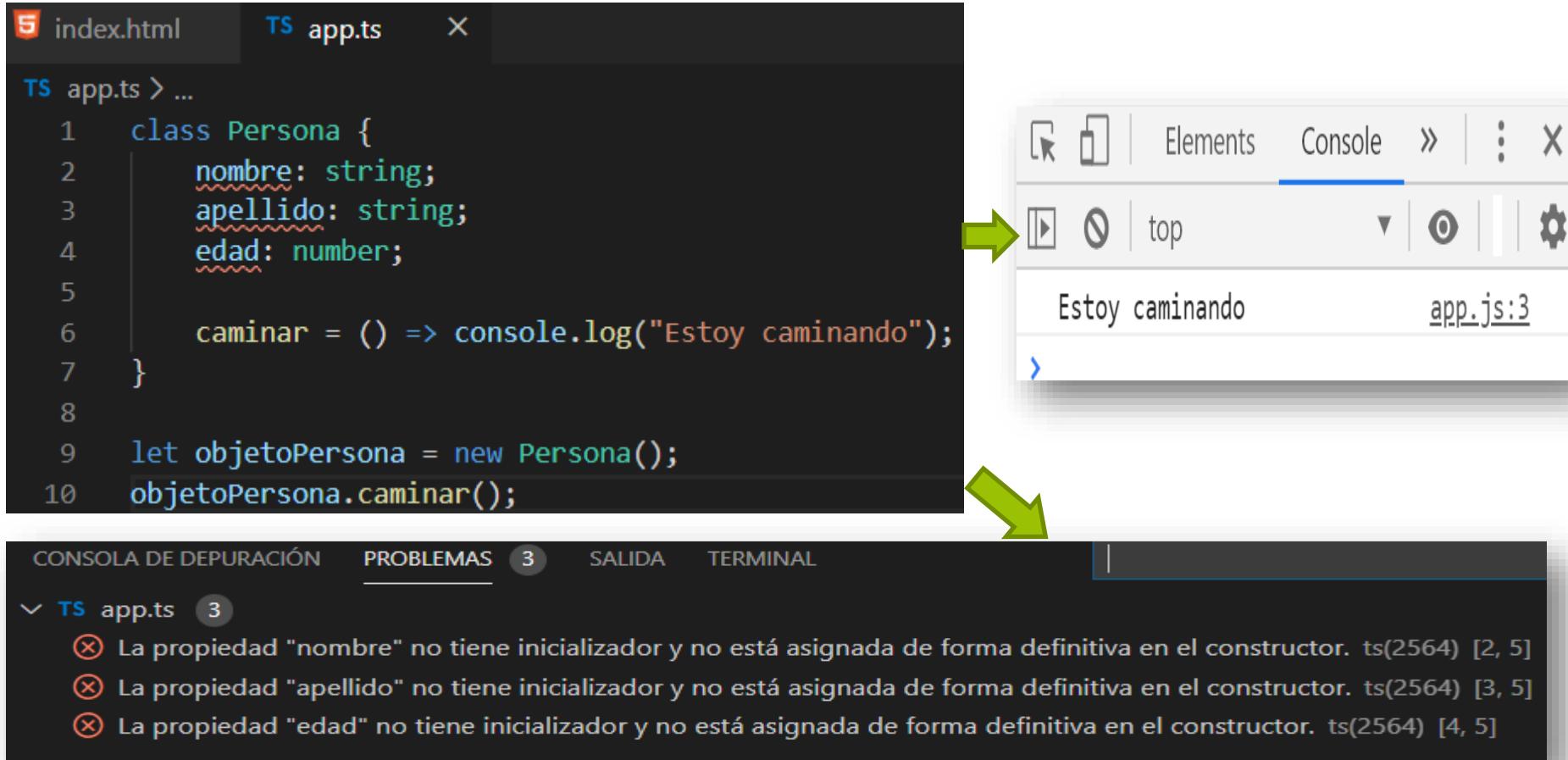
Herencia

```
21  
22 class EditorVideo extends Programa{  
23     public timeline:number;
```

3. TypeScript

Programación Orientada a Objetos en TypeScript

Actividad: Vamos a escribir el siguiente código en el **app.ts** y lo probamos:



The screenshot shows a TypeScript development environment with the following components:

- Editor Area:** Shows the **app.ts** file content:

```
TS app.ts > ...
1  class Persona {
2      nombre: string;
3      apellido: string;
4      edad: number;
5
6      caminar = () => console.log("Estoy caminando");
7  }
8
9  let objetoPersona = new Persona();
10 objetoPersona.caminar();
```
- Console Tab:** Shows the output of the code execution:

```
Elements Console
top
Estoy caminando
app.js:3
```
- Terminal Tab:** Shows the output of the code execution:

```
>
```
- PROBLEMAS Tab:** Shows three errors:
 - La propiedad "nombre" no tiene inicializador y no está asignada de forma definitiva en el constructor. ts(2564) [2, 5]
 - La propiedad "apellido" no tiene inicializador y no está asignada de forma definitiva en el constructor. ts(2564) [3, 5]
 - La propiedad "edad" no tiene inicializador y no está asignada de forma definitiva en el constructor. ts(2564) [4, 5]



3. TypeScript

Programación Orientada a Objetos en TypeScript

Actividad: Versionaremos nuestra clase de `app.ts` y la volvemos a probar:

```
class Persona {  
    nombre: string;  
    apellido: string;  
    edad: number;  
  
    constructor(name:string, surname:string, age:number) {  
        this.nombre = name;  
        this.apellido = surname;  
        this.edad = age;  
    }  
  
    caminar = () => console.log("Hola soy",this.nombre, "y estoy caminando");  
}  
  
let objetoPersona = new Persona("Rocío", "De la O", 23);  
objetoPersona.caminar();
```



Nota: para este ejemplo ignoraremos el error generado en consola.

```
PS C:\Dev\workspace\jhernand\ts1> tsc Persona.ts
PS C:\Dev\workspace\jhernand\ts1> tsc app.ts
app.ts:3:25 - error TS2304: Cannot find name 'Persona'.
3 let objetoPersona = new Persona("Rocío", "De la O", 23);
                                                 ^
Found 1 error.
```



3. TypeScript



Hola soy Rocío y estoy caminando

Programación Orientada a Objetos en TypeScript

Actividad: Ahora vamos a llevarnos la clase Persona a un nuevo fichero **Persona.ts**

```
TS Persona.ts > Persona
1 class Persona {
2     nombre: string;
3     apellido: string;
4     edad: number;
5
6     constructor(name:string, surname:string, age:number) {
7         this.nombre = name;
8         this.apellido = surname;
9         this.edad = age;
10    }
11
12    caminar = () => console.log("Hola soy",this.nombre, "y estoy caminando");
13 }
```

```
TS app.ts > ...
1 let objetoPersona = new Persona("Rocío", "De la O", 23);
2 objetoPersona.caminar();
```

index.html

```
5 index.html > ...
1 <html>
2     <head>
3         <meta charset="UTF-8">
4         <title>Mi primer TypeScript</title>
5     </head>
6     <body>
7         <script src="Persona.js"></script>
8         <script src="app.js"></script>
9     </body>
10    </html>
```



04

Módulos y Decoradores





4. Módulos y Decoradores

Módulos

TypeScript nos va a permitir la facilidad de **exportar** nuestras clases para ser **importadas** y usadas en cualquier otra parte. Esto son lo que llamamos **módulos**. Para ello en línea con el ejemplo anterior:

TS Persona.ts > ...

```

1  export class Persona {
2      nombre: string;
3      apellido: string;
4      edad: number;
5
6      constructor(name:string, surname:string, age:number) {
7          this.nombre = name;
8          this.apellido = surname;
9          this.edad = age;
10     }
11
12     caminar = () => console.log("Hola soy",this.nombre, "y estoy caminando");
13 }
```

TS app.ts > ...

```

1   import {Persona} from "./Persona";
2
3   let objetoPersona = new Persona("Rocío", "De la O", 23);
4   objetoPersona.caminar();
```

index.html > ...

```

1  <html>
2      <head>
3          <meta charset="UTF-8">
4          <title>Mi primer TypeScript</title>
5      </head>
6      <body>
7          <script src="app.js"></script>
8      </body>
9  </html>
```



4. Módulos y Decoradores

Decoradores

Básicamente es una implementación de un patrón de diseño de software que en sí sirve para extender una función mediante otra función, pero sin tocar aquella original, que se está extendiendo. El decorador recibe una función como argumento (aquella que se quiere decorar) y devuelve esa función con alguna funcionalidad adicional.

Las funciones decoradoras comienzan por una "@" y a continuación tienen un nombre. Ese nombre es el de aquello que queramos decorar, que ya tiene que existir previamente. Podríamos decorar una función, una propiedad de una clase, una clase, etc.

Podemos considerar un decorador como la forma de aumentar las funcionalidades a ciertos tipos.



4. Módulos y Decoradores

Decoradores

Se crean con el objetivo de mejorar, ampliar, validar, etc lo que ya existe.

Los **decoradores** pueden ser de: **clases, propiedades, parámetros y métodos**.

Estamos asignándole
dinámicamente el
método 'saludo'

```
function Bienvenida(target: Function): void {  
    target.prototype.saludo = function(): void {  
        console.log('¡Hola!');  
    }  
}  
  
@Bienvenida  
class Saludar {  
    constructor() {  
        // Implementación va aquí...  
    }  
}  
  
var miSaludo = new Saludar();  
miSaludo.saludo(); // Consola mostrará '¡Hola!'
```



05

Estructura de un proyecto Angular



5. Estructura de un proyecto Angular

Estructura

Vamos a crearnos por comandos un nuevo proyecto Angular en c:\dev\workspace\username escribiendo “**ng new proyecto1**”

```
C:\Dev\workspace\jhernand> ng new proyecto1
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
The file will have its original line endings in your working directory
Successfully initialized git.
```

```
C:\Dev\workspace\jhernand>cd proyecto1
C:\Dev\workspace\jhernand\proyecto1> dir
El volumen de la unidad C es OSDisk
El n mero de serie del volumen es: EA5A-6103

Directorio de C:\Dev\workspace\jhernand\proyecto1

21/03/2020  13:09    <DIR>          .
21/03/2020  13:09    <DIR>          ..
21/03/2020  12:56            246 .editorconfig
21/03/2020  12:56            631 .gitignore
21/03/2020  12:56            3.591 angular.json
21/03/2020  12:56            429 browserslist
21/03/2020  12:56            e2e
21/03/2020  12:56            1.021 karma.conf.js
21/03/2020  13:09    <DIR>          node_modules
21/03/2020  13:09            498.658 package-lock.json
21/03/2020  12:56            1.286 package.json
21/03/2020  12:56            1.026 README.md
21/03/2020  12:56    <DIR>          src
21/03/2020                210 tsconfig.app.json
21/03/2020                489 tsconfig.json
21/03/2020                270 tsconfig.spec.json
21/03/2020  12:56            1.953 tslint.json
                           12 archivos      509.810 bytes
                           5 dirs     87.566.315.520 bytes libres
```

Tras ello accederemos al proyecto:



5. Estructura de un proyecto Angular

Estructura

A continuación vamos a publicar el proyecto para verlo desde el navegador con ‘ng serve’

```
C:\Dev\workspace\jhernand\proyecto1> ng serve
? Would you like to share anonymous usage data about this project with the Angular Team at
Google under Google's Privacy Policy at https://policies.google.com/privacy? For more
details and how to change this setting, see http://angular.io/analytics. No
0% compiling
Compiling @angular/core : es2015 as esm2015

Compiling @angular/common : es2015 as esm2015

Compiling @angular/platform-browser : es2015 as esm2015

Compiling @angular/platform-browser-dynamic : es2015 as esm2015

Compiling @angular/router : es2015 as esm2015

chunk {main} main.js, main.js.map (main) 60.6 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 9.71 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.99 MB [initial] [rendered]
Date: 2020-03-21T12:32:56.692Z - Hash: 2f5dcdecb951fc3d1817 - Time: 28227ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
**
: Compiled successfully.

Date: 2020-03-21T12:32:58.532Z - Hash: 2f5dcdecb951fc3d1817
5 unchanged chunks

Time: 1170ms
: Compiled successfully.
```



5. Estructura de un proyecto Angular

Estructura

Ahora accedemos desde el navegador a <http://localhost:4200/> para ver nuestro proyecto.

Welcome

projecto1 app is running!

Resources

Here are some links to help you get started:

- Learn Angular >
- CLI Documentation >
- Angular Blog >

Next Steps

What do you want to do next with your app?

- + New Component
- + Angular Material
- + Add PWA Support
- + Add Dependency
- + Run and Watch Tests
- + Build for Production

ng generate component xyz

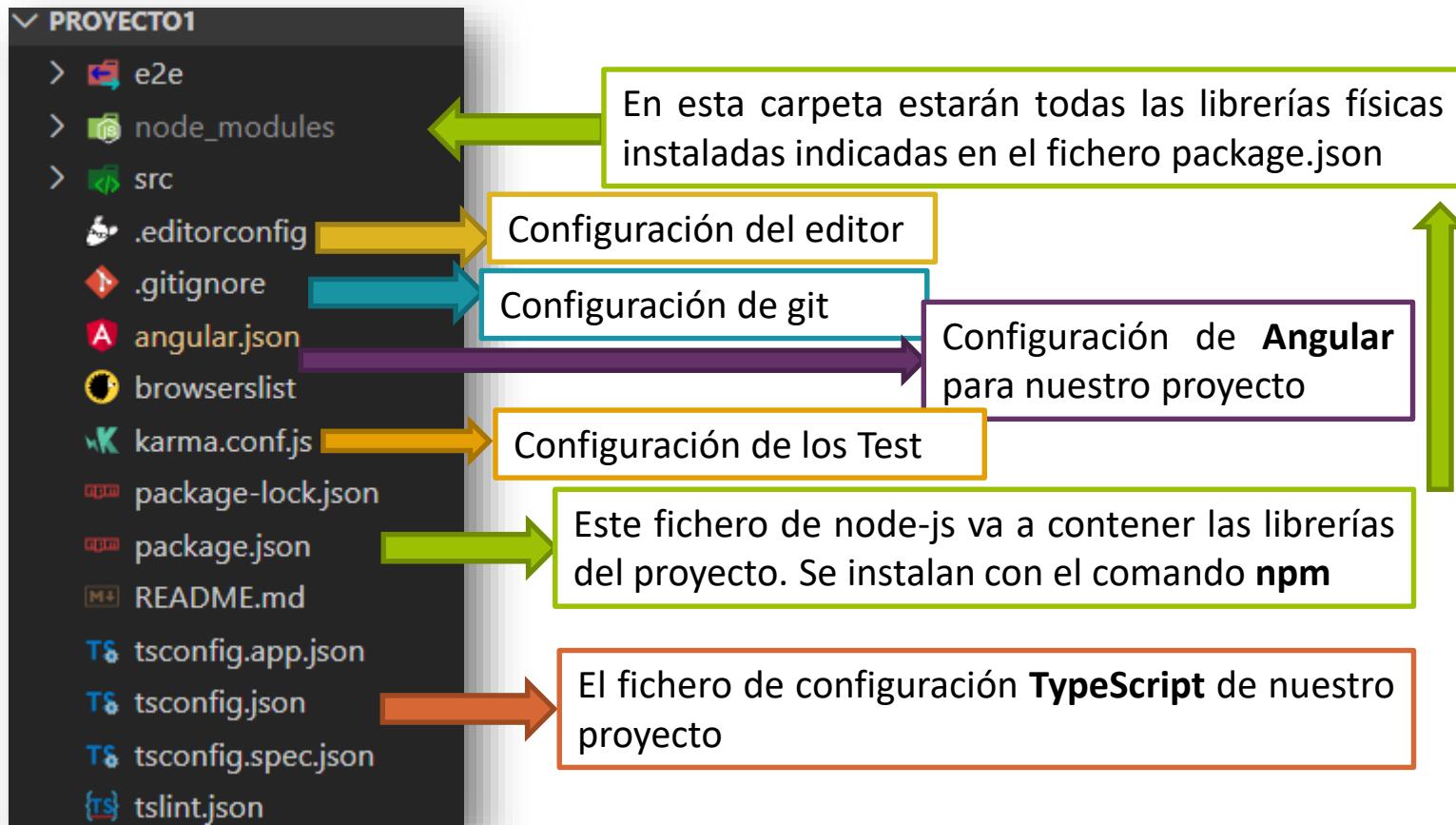
Love Angular? Give our repo a star. ★ Star >



5. Estructura de un proyecto Angular

Estructura

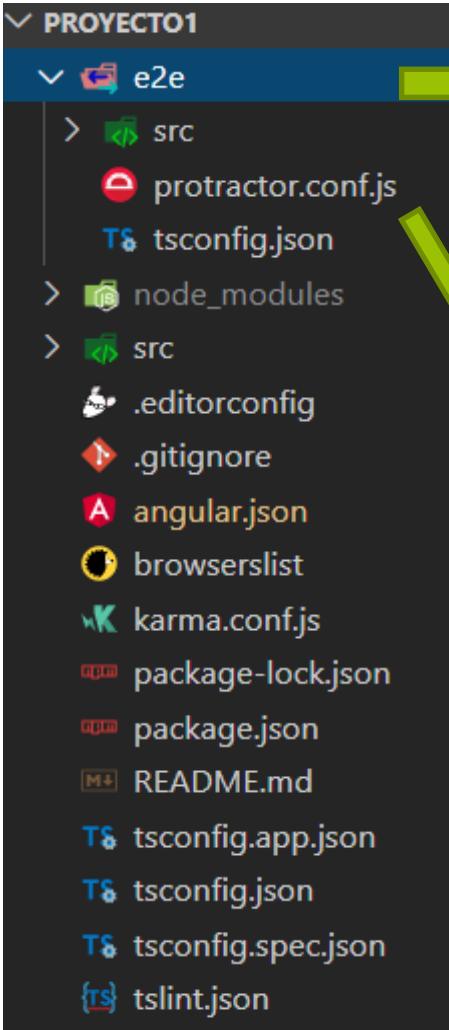
Ahora vamos a abrir el proyecto desde Visual Studio code: **Menú – Archivo – Abrir carpeta**





5. Estructura de un proyecto Angular

Estructura de e2e



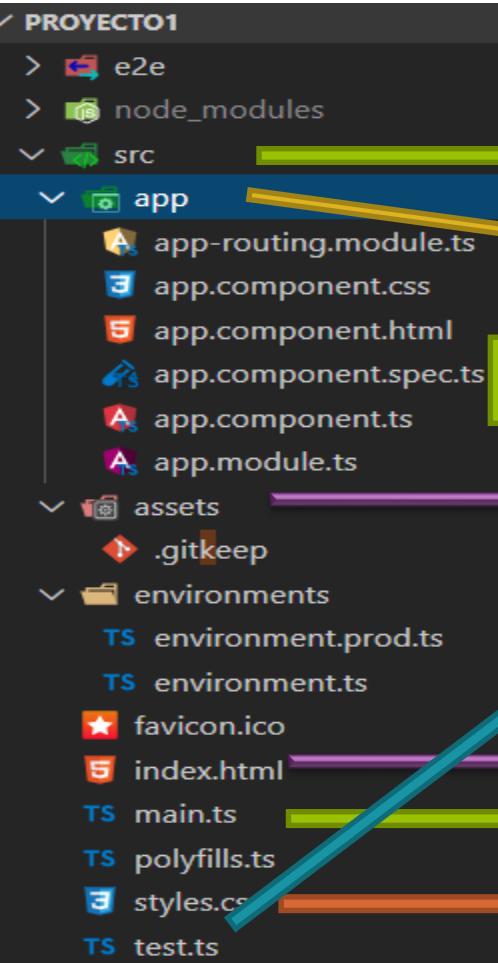
Carpeta asociada a metodología de pruebas end-to-end en la que el flujo entero de la aplicación es evaluado.

Para verificar que los distintos componentes de una aplicación funcionan correctamente entre sí.

Fichero de configuración del framework Protractor, que es el encargado de pruebas end-to-end

5. Estructura de un proyecto Angular

Estructura de src



Carpeta con el código de desarrollo, el que vamos a tocar principalmente.

Ficheros asociados al proyecto **Angular**

Componentes y módulos de nuestra aplicación

Recursos que use nuestra aplicación: imágenes, archivos físicos, etc

Archivo para pruebas del proyecto

Archivo de entrada inicial de nuestra aplicación

Archivo de entrada inicial para TypeScript

Plantilla css del proyecto



06 Componentes





6. Componentes

¿Qué es un componente?

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

Un **estilo** para la vista

El **spec** será el archivo usado para pruebas unitarias del componente.

Tienen un ciclo de vida que deben respetar.

6. Componentes

¿Qué es un componente?



The diagram illustrates a web page structure divided into several colored sections:

- Barra Lateral** (Purple vertical bar on the left)
- Menú de navegación** (Blue header bar)
- Demás páginas y sub páginas** (Yellow main content area)
- Pie de la página o aplicación** (Green footer bar)

A green arrow points from the yellow content area towards the bottom right, indicating the scope of a component.

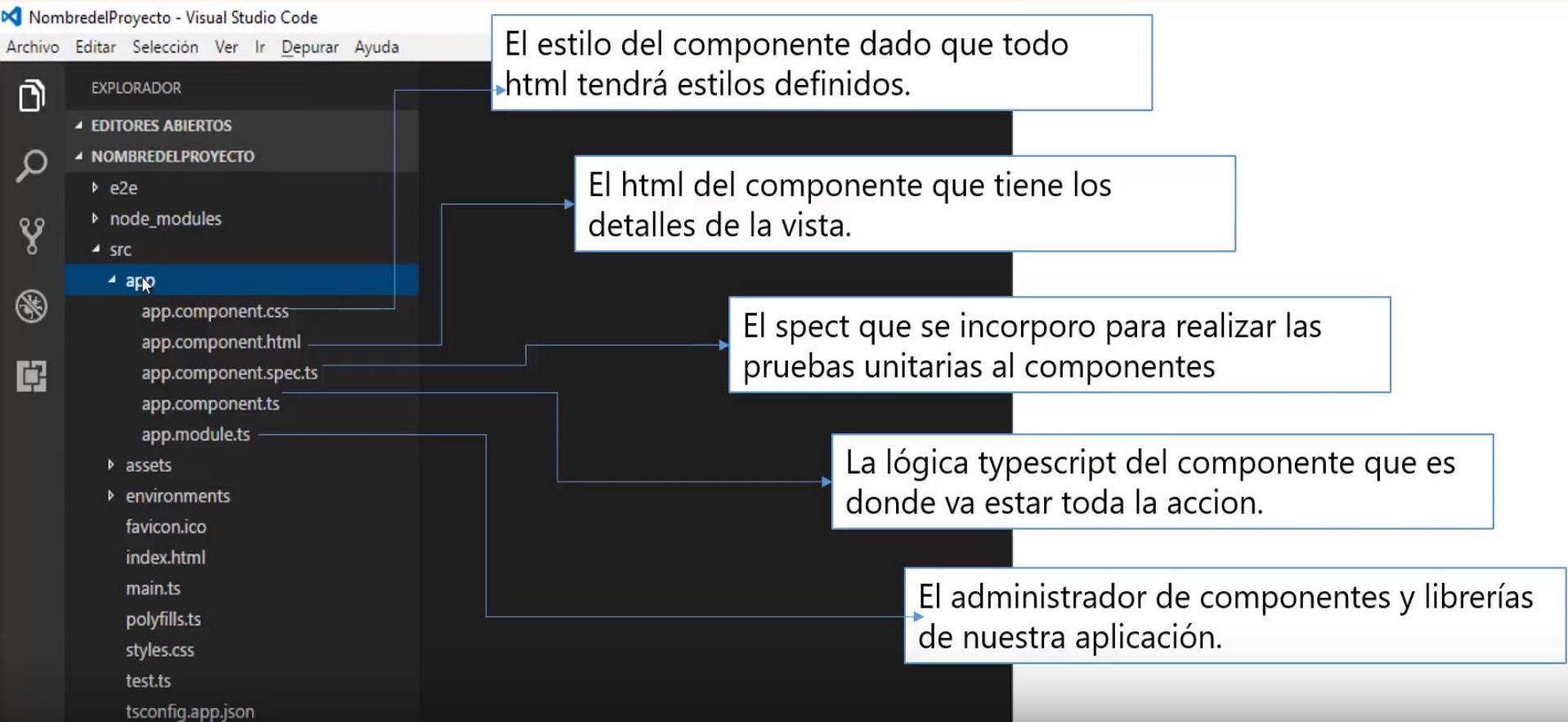
En la parte central (fondo amarillo) podría ser un componente con **etiquetas HTML, estilo en CSS, lógica y parte de test definidos en TypeScript**.



6. Componentes

Dentro de /app

Repasamos parte de la estructura de un componente:





6. Componentes

Importante

A tener en cuenta:

- Todo componente tiene un decorador que permite asociar los elementos que lo conforman
- Todo componente para formar parte de una vista tendrá un selector <app-root>
- Todo componente esta formado por una vista , una lógica typescript y un estilo de la vista.
- Todo componente Importa Librerias que usara
- Todo componente se exporta para que pueda formar parte del todo.



6. Componentes

```
src > app > componentes > footer > footer.component.ts > ...
1 import { Component, OnInit, Input } from '@angular/core';
2
3 @Component({
4   selector: 'app-footer',
5   templateUrl: './footer.component.html',
6   styleUrls: ['./footer.component.css']
7 })
8 export class FooterComponent implements OnInit {
9
10   @Input("usuario") usuarioAutenticado: any = {"nombre": "_", "id": "0", "username": ""};
11
12   constructor() { }
13
14   ngOnInit(): void {
15   }
16
17 }
```



```
app > app.module.ts > ...
    import { HeaderComponent } from './componentes/header/header.component';
    import { FooterComponent } from './componentes/footer/footer.component';
```

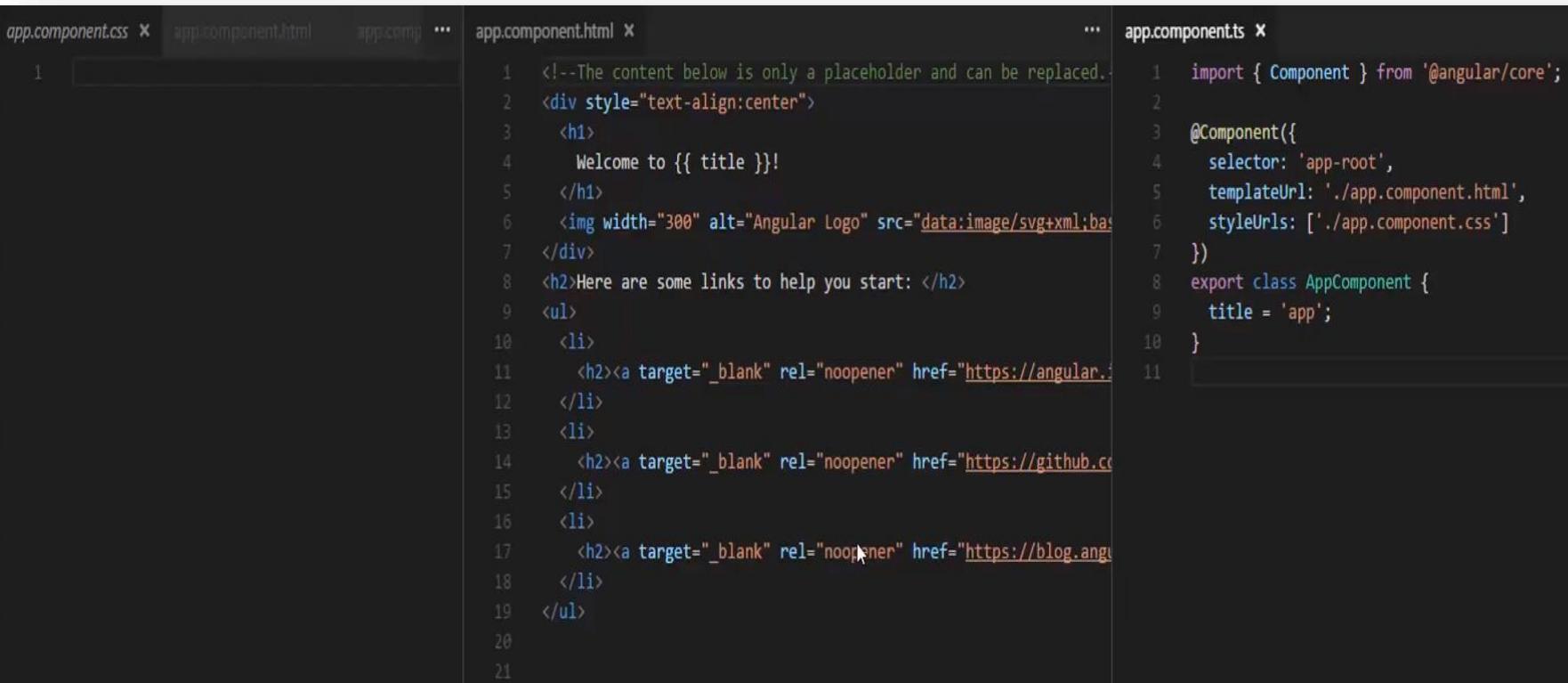


6. Componentes

Partes de un componente

Así un componente estará formado por:

- Un estilo
- Un código HTML que define la vista
- Y un código TypeScript que define comportamiento y estados entre vista y lógica.



The screenshot shows a code editor with three tabs open:

- app.component.css**: Contains some placeholder styles.
- app.component.html**: Contains the component's template code.
- app.component.ts**: Contains the component's class definition.

```
app.component.css x app.component.html app.component.ts ...
1  <!--The content below is only a placeholder and can be replaced.-->
2  <div style="text-align:center">
3    <h1>
4      Welcome to {{ title }}!
5    </h1>
6    
7  </div>
8  <h2>Here are some links to help you start: </h2>
9  <ul>
10   <li>
11     <h2><a target="_blank" rel="noopener" href="https://angular.io">Angular Documentation</a></h2>
12   </li>
13   <li>
14     <h2><a target="_blank" rel="noopener" href="https://github.com/angular/angular">Source Code</a></h2>
15   </li>
16   <li>
17     <h2><a target="_blank" rel="noopener" href="https://blog.angular.io">Blog</a></h2>
18   </li>
19 </ul>
```

```
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';
10 }
```



6. Componentes

index.html → app-root es el selector de componentes

app.component.ts → incluye la referencia a fichero html (la vista) y css (el estilo).

```
index.html ×
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Nombre del Proyecto</title>
6   <base href="/">
7
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
15
```

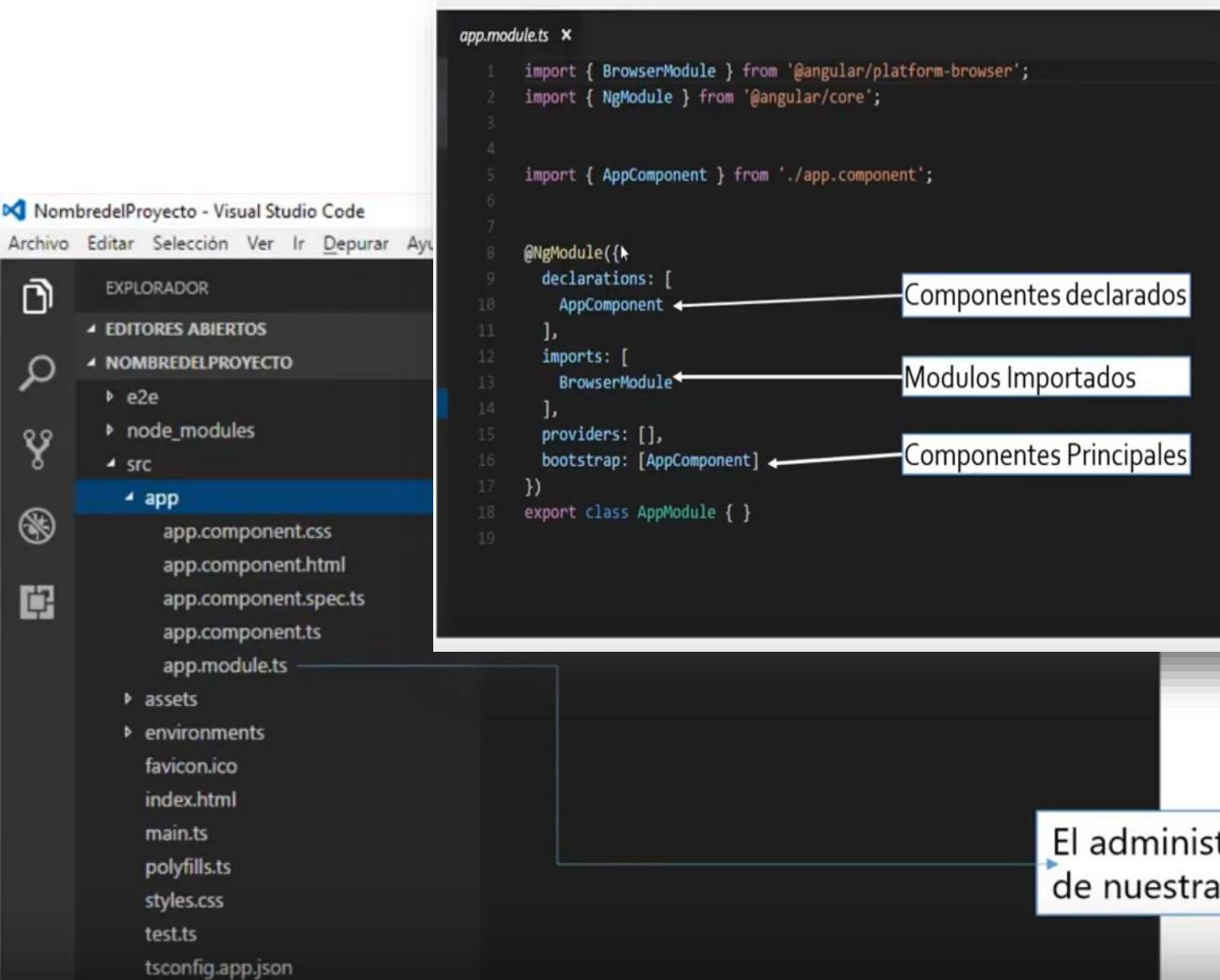
```
app.component.ts ×
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'app';
10 }
11
```

```
app.component.html ×
1 <!--The content below is only a placeholder and can be replaced.-->
2 <div style="text-align:center">
3   <h1>
4     Welcome to {{ title }}!
5   </h1>
6   
12       </a></h2>
13   </li>
14   <li>
15     <h2><a target="_blank" rel="noopener" href="https://github.com/angular/>
16   </li>
17   <li>
18     <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">
19   </li>
20 </ul>
```

```
app.component.css ×
1
```

6. Componentes

El administrador de componentes es a su vez un componente que forma parte de mi aplicación



```

app.module.ts ×
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4
5 import { AppComponent } from './app.component';
6
7
8 @NgModule({
9   declarations: [
10     AppComponent ← Componentes declarados
11   ],
12   imports: [
13     BrowserModule ← Modulos Importados
14   ],
15   providers: [],
16   bootstrap: [AppComponent] ← Componentes Principales
17 })
18 export class AppModule { }
19

```

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with the project structure:

- Archivo
- Editar
- Selección
- Ver
- Ir
- Depurar
- Ayud.
- EXPLORADOR**
- NOMBREDELPROYECTO**
 - e2e
 - node_modules
 - src**
 - app**
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
- assets
- environments
- favicon.ico
- index.html
- main.ts
- polyfills.ts
- styles.css
- test.ts
- tsconfig.app.json

Antes de importar cualquier módulo hay que definirlo. En Angular los módulos se declaran como clases de TypeScript, habitualmente vacías, decoradas con una función especial. Es la función `@NgModule()` que recibe un objeto como único argumento. En las propiedades de ese objeto es dónde se configura el módulo.

Este archivo es el encargado de entender qué componentes y dependencias tenemos en nuestra aplicación. Si el componente no fue declarado como por ejemplo `AppComponent` no puede ser utilizado.

El administrador de componentes y librerías de nuestra aplicación.

6. Componentes

El contexto de NgModule

NgModule es un decorador que recibe un objeto de metadatos que definen el módulo. Los metadatos más importantes de un **NgModule** son:

declarations: Las vistas que pertenecen a tu módulo. Hay 3 tipos de clases de tipo vista: componentes, directivas y pipes.

exports: Conjunto de declaraciones que deben ser accesibles para templates de componentes de otros módulos.

imports: Otros NgModules, cuyas clases exportadas son requeridas por templates de componentes de este módulo.

providers: Los servicios que necesita este módulo, y que estarán disponibles para toda la aplicación.

bootstrap: Define la vista raíz. Utilizado solo por el root module.



6. Componentes

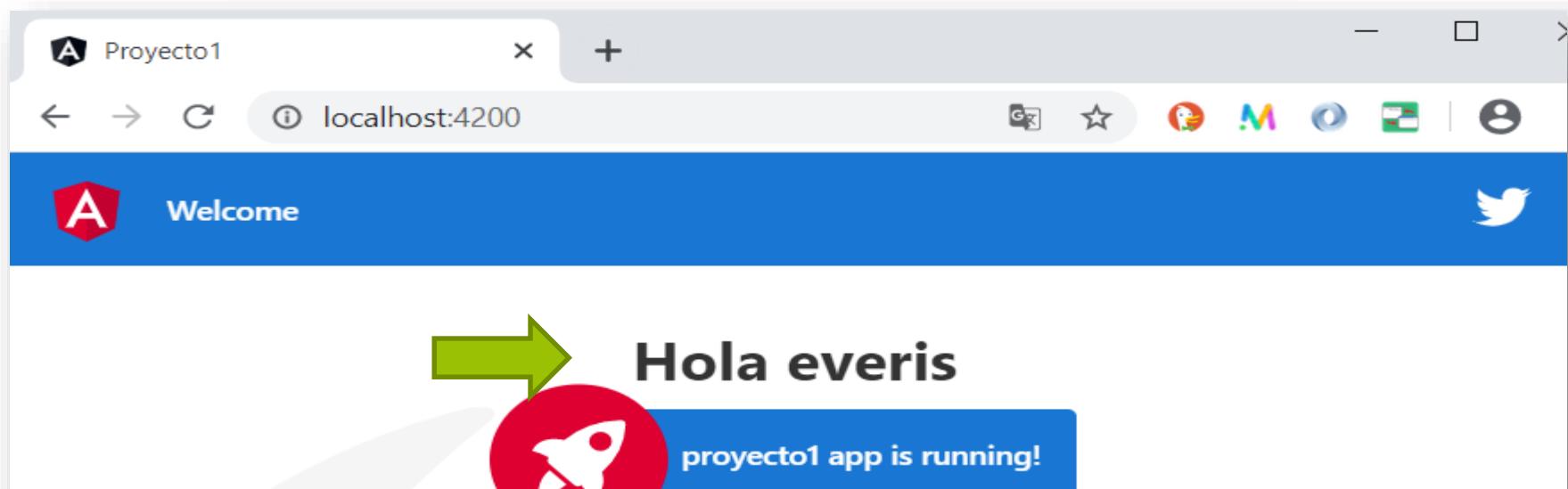
Refresco automático de nuestros cambios en código

Al haber levantado el servidor virtual con ‘**ng serve**’ se ha quedado escuchando a los cambios que metamos en nuestro código.

Así si modificamos el fichero ‘src/app/app.component.html’ y añadimos la línea **319**:

```
318  <div class="content" role="main">
319 | <h1>Hola everis</h1>
320 |   <!-- Highlight Card -->
321 >   <div class="card highlight-card card-small">
```

Al acceder de nuevo a nuestra web (<http://localhost:4200/>) sin necesidad de refrescar veremos:



A screenshot of a web browser window titled 'Proyecto1'. The address bar shows 'localhost:4200'. The page content is a blue header with a red 'A' icon and the word 'Welcome'. Below it is a green arrow pointing right towards the text 'Hola everis'. A red circular icon at the bottom left contains a white rocket ship, with the text 'proyecto1 app is running!' next to it. The browser interface includes standard controls like back, forward, and search.



6. Componentes

Creando un nuevo componente

Vamos a crearnos un componente ‘Principal’ para entender mejor cómo funcionan. Como primer paso escribiremos en consola ‘**ng g c principal**’:

```
PS C:\Dev\workspace\jhernand\proyecto1> ng g c principal
CREATE src/app/principal/principal.component.html (24 bytes)
CREATE src/app/principal/principal.component.spec.ts (649 bytes)
CREATE src/app/principal/principal.component.ts (287 bytes)
CREATE src/app/principal/principal.component.css (0 bytes)
UPDATE src/app/app.module.ts (487 bytes)
PS C:\Dev\workspace\jhernand\proyecto1>
```



File Explorer showing the project structure:

- app
 - principal
 - principal.component.css
 - principal.component.html
 - principal.component.spec.ts
 - principal.component.ts
 - app-routing.module.ts
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts

```
src > app > A app.module.ts > ...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { PrincipalComponent } from './principal/principal.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     PrincipalComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule {}
```



6. Componentes

Creando un nuevo componente

Primeramente modificamos la vista de nuestro componente ‘Principal’:

```
principal.component.html ×
src > app > principal > principal.component.html > ...
1   <p>Componente principal</p>
2   |
```

```
app.module.ts ×
src > app > app.module.ts > AppModule
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { PrincipalComponent } from './principal/principal.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     PrincipalComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule
16   ],
17   providers: [],
18   bootstrap: [PrincipalComponent] ←
19 })
20 export class AppModule { }
```

Tras ello procedemos a indicar, que ahora nuestro componente es el principal de la aplicación:



6. Componentes

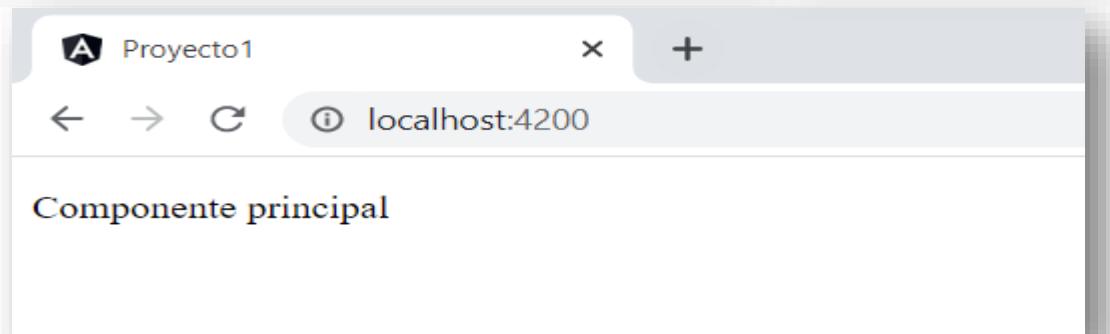
Creando un nuevo componente

El siguiente paso será cambiar en **index.html** para que llame al selector del componente '**Principal**':

```
src > 5 index.html > ...
1  <!doctype html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8">
5  <title>Proyecto1</title>
6  <base href="/">
7  <meta name="viewport" content="width=device-width, initial-scale=1">
8  <link rel="icon" type="image/x-icon" href="favicon.ico">
9 </head>
10 <body>
11 |  <app-principal></app-principal> ←
12 </body>
13 </html>
```

```
src > app > principal > A principal.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-principal', ←
5    templateUrl: './principal.component.html',
6    styleUrls: ['./principal.component.css']
7  })
8  export class PrincipalComponent implements OnInit {
9
10  constructor() { }
11
12  ngOnInit(): void {
13  }
14
15 }
```

Y al acceder ahora a nuestra web veremos:





07

Agregando Bootstrap





7. Bootstrap

Qué es Bootstrap

Bootstrap es un **framework** desarrollado y liberado por Twitter que tiene como objetivo facilitar el diseño web. Permite crear de forma sencilla webs de diseño adaptable, es decir, que se ajusten a cualquier dispositivo y tamaño de pantalla y siempre se vean bien. Es **open source**, por lo que podemos usarlo de forma gratuita y sin restricciones.

B Home Documentation Examples Icons Themes Expo Blog

Github Twitter GitHub Gitter Download

Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

Get started

Download



	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
.container	100%	540px	720px	960px	1140px
.container-sm	100%	540px	720px	960px	1140px
.container-md	100%	100%	720px	960px	1140px
.container-lg	100%	100%	100%	960px	1140px
.container-xl	100%	100%	100%	100%	1140px
.container-fluid	100%	100%	100%	100%	100%



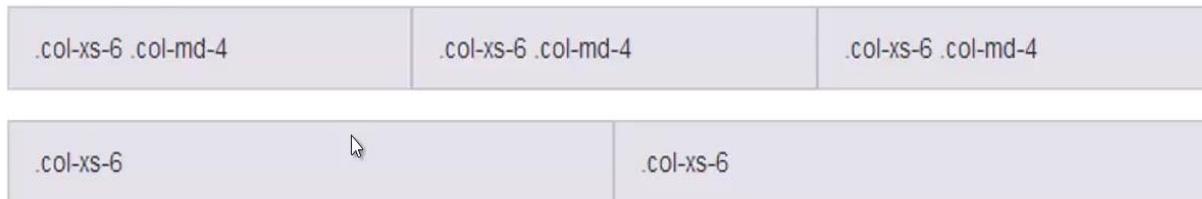
Qué es Bootstrap

Es multidispositivo

Bootstrap es una especie de **facilitador** para el **diseño de interfaces de aplicaciones web**.

Es soportado por todos los navegadores, excepto Internet Explorer 7 (IE8 funciona con respond.js). Contiene estilos para tablas, formularios personalizados, agrupación de etiquetas, etc.

El diseño de una pantalla se basa en que por debajo tiene un **grid** (cuadrícula) de 12 columnas:



```

<div class="container">
  <div class="row">
    <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
    <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
    <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  </div>
  <div class="row">
    <div class="col-xs-6">.col-xs-6</div>
    <div class="col-xs-6">.col-xs-6</div>
  </div>
</div>

```

Usaré 6 columnas para dimensión xs y 4 para md

7. Bootstrap

Incorporar Bootstrap dentro de Angular

Para incorporar **Bootstrap** a nuestro proyecto nos lo descargaremos desde <https://getbootstrap.com/> y tras pulsar en **Download**, accedemos a la parte de **BootstrapCDN** y copiamos el código que nos pone (ambos apartados):

BootstrapCDN

Skip the download with [BootstrapCDN](#) to deliver cached version of Bootstrap's compiled CSS and JS to your project.

```
min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5ToeNV6gBiFeWPGFN9MuhoF23Q9Ifjh" crossorigin="anonymous">
="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfkjtj0Uod8GCEx13og8ifwB6" crossorigin="anonymous"></script>
```

If you're using our compiled JavaScript, don't forget to include CDN versions of jQuery and Popper.js before it.

```
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7im" crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZfJoft+2mJbHaE" crossorigin="anonymous">
```

Lo pegamos en el **index.html** en la parte del **<head>** (**jquery.js** debe ir antes que **bootstrap.js**):

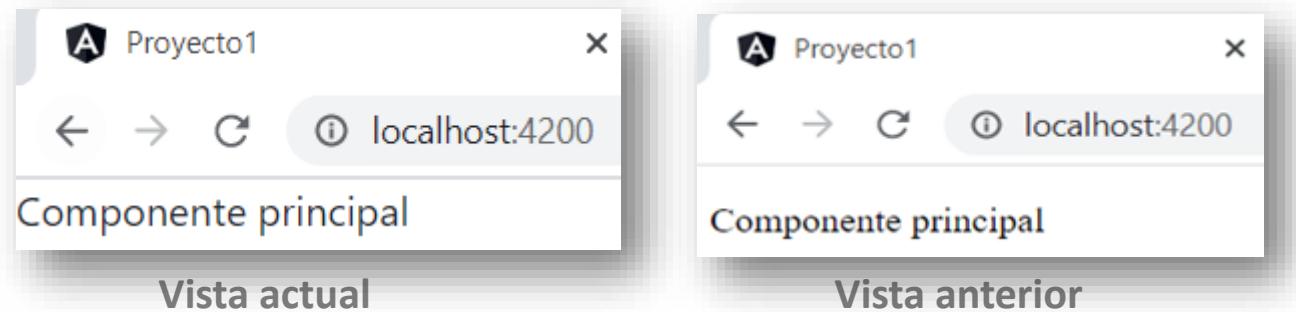
```
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Proyecto1</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5ToeNV6gBiFeWPGFN9MuhoF23Q9Ifjh" crossorigin="anonymous">
  <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7im" crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZfJoft+2mJbHaE" crossorigin="anonymous">
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfkjtj0Uod8GCEx13og8ifwB6" crossorigin="anonymous">
</head>
<body>
  <app-principal></app-principal>
</body>
</html>
```



7. Bootstrap

Incorporar Bootstrap dentro de Angular

Podemos ver si accedemos a nuestra web que ya se está teniendo en cuenta **Bootstrap**:

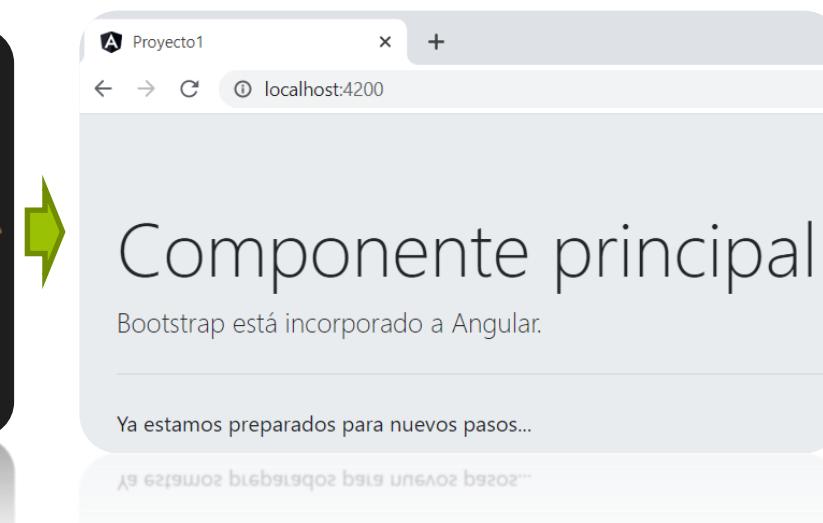


Vista actual

Vista anterior

A continuación modificaremos el contenido de **src/app/principal/principal.component.html**:

```
src > app > principal > 5 principal.component.html > ...
1 <div class="jumbotron">
2   <h1 class="display-4">Componente principal</h1>
3   <p class="lead">Bootstrap está incorporado a Angular.</p>
4   <hr class="my-4">
5   <p>Ya estamos preparados para nuevos pasos...</p>
6 </div>
7 <div>
8   <h2>Componente principal</h2>
9   <p>Bootstrap está incorporado a Angular.</p>
10  <hr class="my-4">
11  <p>Ya estamos preparados para nuevos pasos...</p>
12 </div>
```



Componente principal

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...



08

Trabajando con componentes

8. Trabajando con componentes

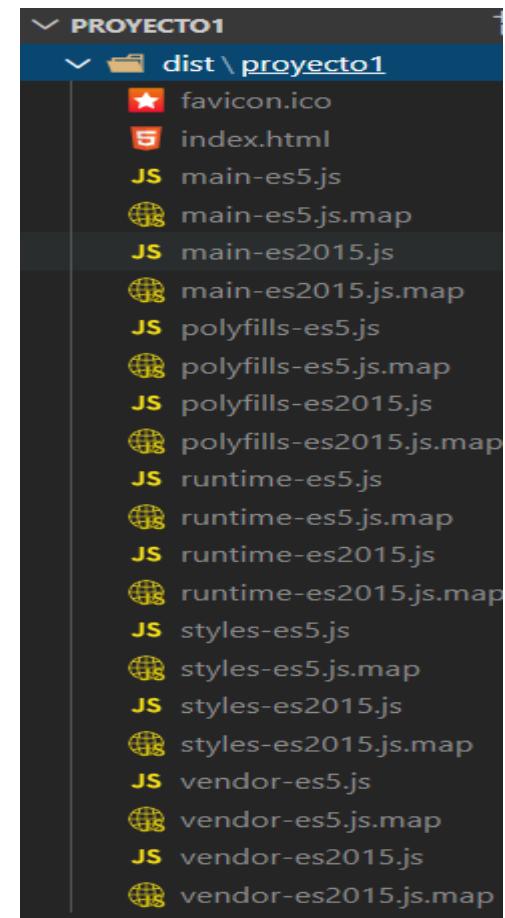
Publicando Angular

Antes de nada vamos a ver cómo podríamos enganchar nuestro código **Angular** en un servidor donde la parte **backend** se haya realizado en otro lenguaje: así escribiremos en consola '**ng build**' (con '**ng build --prod**' compactamos los ficheros) :

```
c:\Dev\workspace\jhernand\proyecto1> ng build
Generating ES5 bundles for differential loading...
ES5 bundle generation complete.

chunk {polyfills} polyfills-es2015.js, polyfills-es2015.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime-es2015.js, runtime-es2015.js.map (runtime) 6.16 kB [entry] [rendered]
chunk {runtime} runtime-es5.js, runtime-es5.js.map (runtime) 6.16 kB [entry] [rendered]
chunk {styles} styles-es2015.js, styles-es2015.js.map (styles) 9.72 kB [initial] [rendered]
chunk {styles} styles-es5.js, styles-es5.js.map (styles) 11 kB [initial] [rendered]
chunk {main} main-es2015.js, main-es2015.js.map (main) 62.7 kB [initial] [rendered]
chunk {main} main-es5.js, main-es5.js.map (main) 66.3 kB [initial] [rendered]
chunk {polyfills-es5} polyfills-es5.js, polyfills-es5.js.map (polyfills-es5) 656 kB [initial] [rendered]
chunk {vendor} vendor-es2015.js, vendor-es2015.js.map (vendor) 2.66 MB [initial] [rendered]
chunk {vendor} vendor-es5.js, vendor-es5.js.map (vendor) 3.11 MB [initial] [rendered]
Date: 2020-04-09T10:17:12.545Z - Hash: 3bce03bb2710eac12634 - Time: 51444ms
C:\Dev\workspace\jhernand\proyecto1>
```

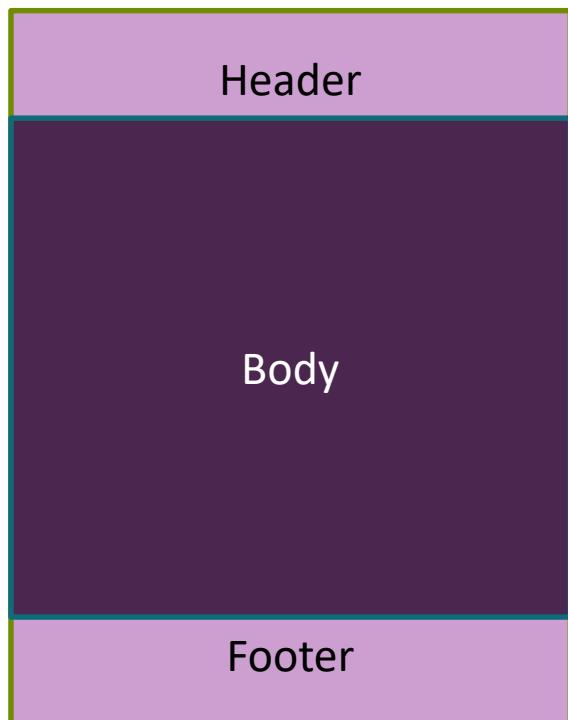
Así es como generamos una distribución que sería la parte **frontend** de nuestro proyecto web.



8. Trabajando con componentes

Desarrollo de componentes en Angular

Vamos a desarrollar una aplicación **SPA**:



La aplicación estará basada en **tres componentes**:

- Uno para la cabecera
- Otro para el cuerpo
- Y otro para el pie de página.

Como ya tenemos nuestro componente ‘Principal’, éste podría ser el del **body**. Pero nos quedarán crear los dos restantes:

ng g c header

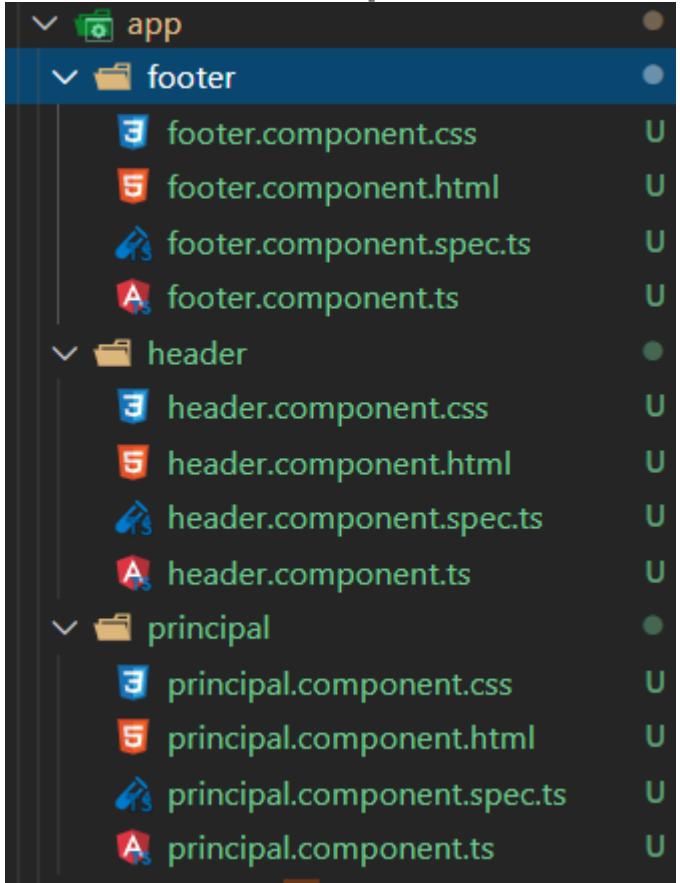
ng g c footer

```
PS C:\Dev\workspace\jhernand\proyecto1> ng g c header
CREATE src/app/header/header.component.html (21 bytes)
CREATE src/app/header/header.component.spec.ts (628 bytes)
CREATE src/app/header/header.component.ts (275 bytes)
CREATE src/app/header/header.component.css (0 bytes)
UPDATE src/app/app.module.ts (575 bytes)
PS C:\Dev\workspace\jhernand\proyecto1> ng g c footer
CREATE src/app/footer/footer.component.html (21 bytes)
CREATE src/app/footer/footer.component.spec.ts (628 bytes)
CREATE src/app/footer/footer.component.ts (275 bytes)
CREATE src/app/footer/footer.component.css (0 bytes)
UPDATE src/app/app.module.ts (657 bytes)
```



8. Trabajando con componentes

Desarrollo de componentes en Angular



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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { PrincipalComponent } from './principal/principal.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';

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

Como primer paso arrancaremos el servidor asociado para que escuche nuestros cambios:

ng serve

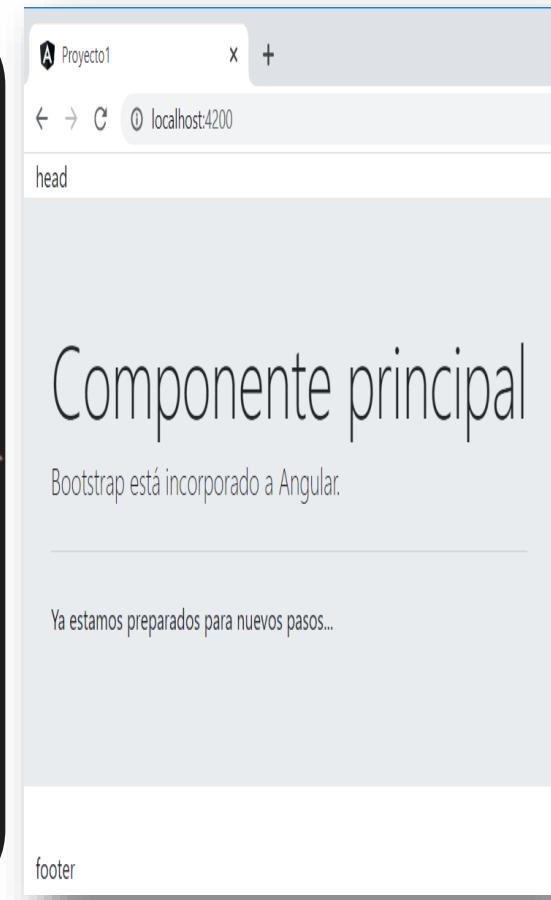
8. Trabajando con componentes

Desarrollo de componentes en Angular

Ahora vamos a modificar la vista de nuestro componente **Principal** para que se adapte a nuestro esquema visual usando **Bootstrap**:

```
src > app > principal > 5 principal.component.html > ...
1  <div class="container">
2    <div class="row">
3      <div class="col-12">head</div>
4    </div>
5
6    <div class="row">
7      <div class="jumbotron">
8        <h1 class="display-4">Componente principal</h1>
9        <p class="lead">Bootstrap está incorporado a Angular.</p>
10       <hr class="my-4">
11       <p>Ya estamos preparados para nuevos pasos...</p>
12     </div>
13   </div>
14
15   <div class="row">
16     <div class="col-12">footer</div>
17   </div>
18 </div>
```

Body



The screenshot shows a browser window titled 'Proyecto1' at 'localhost:4200'. The page content is as follows:

Componente principal

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...

footer

8. Trabajando con componentes

Desarrollo de componentes en Angular

Ahora vamos al **header.component.ts** y copiamos el nombre del **selector**, hacemos lo mismo para el **footer**.

Y lo ponemos en la vista de nuestro componente principal como si fueran etiquetas, reemplazando las palabras 'head' y 'footer' que habíamos puesto con anterioridad:

```
src > app > principal > 5 principal.component.html > ...
1   <div class="container">
2     <div class="row">
3       <div class="col-12">
4         <app-header></app-header>
5       </div>
6     </div>
7
8     <div class="row">
9       <div class="jumbotron">
10      <h1 class="display-4">Componente principal</h1>
11      <p class="lead">Bootstrap está incorporado a Angular.</p>
12      <hr class="my-4">
13      <p>Ya estamos preparados para nuevos pasos...</p>
14    </div>
15  </div>
16
17  <div class="row">
18    <div class="col-12">
19      <app-footer></app-footer>
20    </div>
21  </div>
22 </div>
```



8. Trabajando con componentes

Desarrollo de componentes en Angular

Ahora vamos a incorporar una barra de navegación. Para ello accedemos a <https://getbootstrap.com/docs/> y desde aquí a la sección de **Components** y luego en **Navbar** copiamos el código que está junto a la siguiente barra de navegación:

Navbar Home Features Pricing Disabled

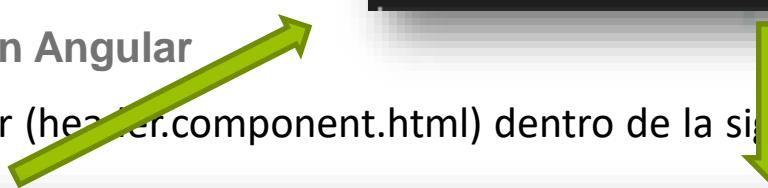
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Features</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Pricing</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
      </li>
    </ul>
  </div>
</nav>
```

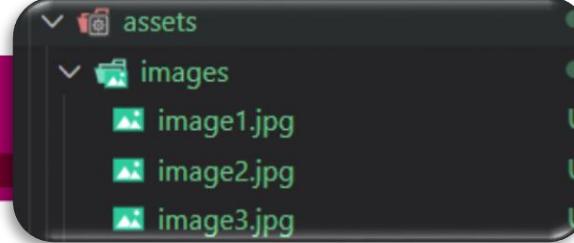
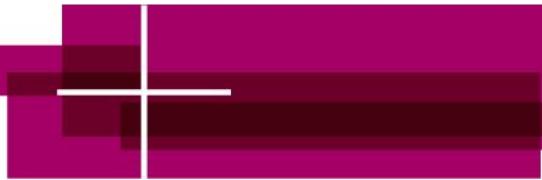
8. Trabajando con componentes

Desarrollo de componentes en Angular

Lo pegamos en la vista del header (header.component.html) dentro de la siguiente estructura en vez del texto “ **pegar aquí** ”:

```
<div class="row">
|   <div class="col-12">_pegar_aquí_</div>
</div>
<div class="row">
|   <div class="col-12">slide</div>
</div>
```





8. Trabajando con componentes

Desarrollo de componentes en Angular

En la sección de la cabecera donde indica ‘Slide’ vamos a poner el componente **Carousel** de **Bootstrap**, así copiaremos el código del ejemplo que indica etiquetado como ‘carouselExampleControls’, luego vamos a descargarnos 3 imágenes y las almacenaremos en ‘assets/images/’:

```
</div>
<div class="row">
  <div class="col-12">
    <div id="carouselExampleControls" class="carousel slide" data-ride="carousel">
      <div class="carousel-inner">
        <div class="carousel-item active">
          
        </div>
        <div class="carousel-item">
          
        </div>
        <div class="carousel-item">
          
        </div>
      </div>
      <a class="carousel-control-prev" href="#carouselExampleControls" role="button" data-slide="prev">
        <span class="carousel-control-prev-icon" aria-hidden="true"></span>
        <span class="sr-only">Previous</span>
      </a>
      <a class="carousel-control-next" href="#carouselExampleControls" role="button" data-slide="next">
        <span class="carousel-control-next-icon" aria-hidden="true"></span>
        <span class="sr-only">Next</span>
      </a>
    </div>
  </div>
</div>
```

8. Trabajando con componentes

Desarrollo de componentes en Angular



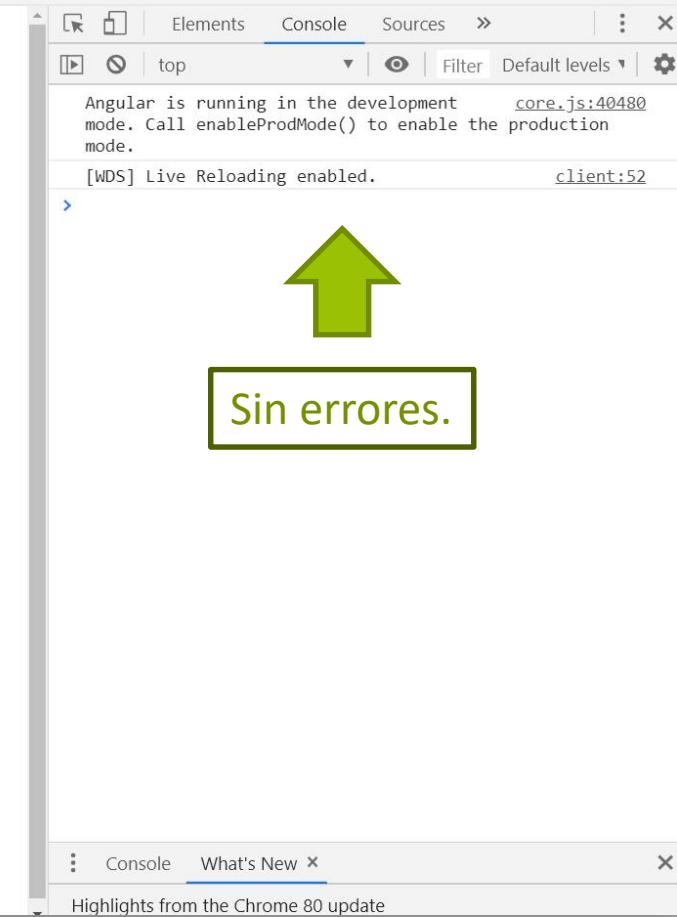
everFuture: Home Sección 1 Sección 2 Sección 3

Componente principal

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...

footer works!



Console

top

Angular is running in the development mode. Call enableProdMode() to enable the production mode.

[WDS] Live Reloading enabled.

client:52

 Sin errores.

Console What's New

Highlights from the Chrome 80 update



8. Trabajando con componentes

Desarrollo de componentes en Angular

Vamos a darle unos pequeños retoques para terminar:

A) En la vista del **header** (**header.component.html**) indicaremos la referencia de cada enlace del **navbar**.

```
<div class="collapse navbar-collapse" id="navbarNav">
  <ul class="navbar-nav">
    <li class="nav-item active">
      | <a class="nav-link" href="#home">Home <span class="sr-only">(current)</span></a>
    </li>
    <li class="nav-item">
      | <a class="nav-link" href="#uno">Sección 1</a>
    </li>
    <li class="nav-item">
      | <a class="nav-link" href="#dos">Sección 2</a>
    </li>
    <li class="nav-item">
      | <a class="nav-link" href="#tres">Sección 3</a>
    </li>
  </ul>
</div>
```



```
<div class="row">
  <section id="home">
    <div class="jumbotron">
      <h1 class="display-4">Componente principal</h1>
      <p class="lead">Bootstrap está incorporado a Angular.</p>
      <hr class="my-4">
      <p>Ya estamos preparados para nuevos pasos...</p>
    </div>
  </section>
  <section id="uno">Contenido de la sección 1</section>
  <section id="dos">Contenido de la sección 2</section>
  <section id="tres">Contenido de la sección 3</section>
</div>
```

B) Definiremos 4 secciones en la 'row' central de la vista Principal (**principal.component.html**):





8. Trabajando con componentes

Desarrollo de componentes en Angular

C) Ahora crearemos los estilos asociados a cada sección (en `principal.component.css`):

```
#home {
    width: 2000px;
}
#uno {
    height: 300px;
    width: 2000px;
    background-color: ■rgb(231, 245, 237);
}
#dos {
    height: 300px;
    width: 2000px;
    background-color: ■rgb(241, 242, 250);
}
#tres {
    height: 300px;
    width: 2000px;
    background-color: ■rgb(226, 227, 248);
```

D) Y por último vamos a poner una firma en el footer (`footer.component.html`), por ejemplo:

```
src > app > footer > 5 footer.component.html > ...
1   <p>Formación Angular de everFuture</p>
```



8. Trabajando con componentes

everFuture: [Home](#) [Sección 1](#) [Sección 2](#) [Sección 3](#)



Componente principal

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...

Contenido de la sección 1

Contenido de la sección 2

Contenido de la sección 3

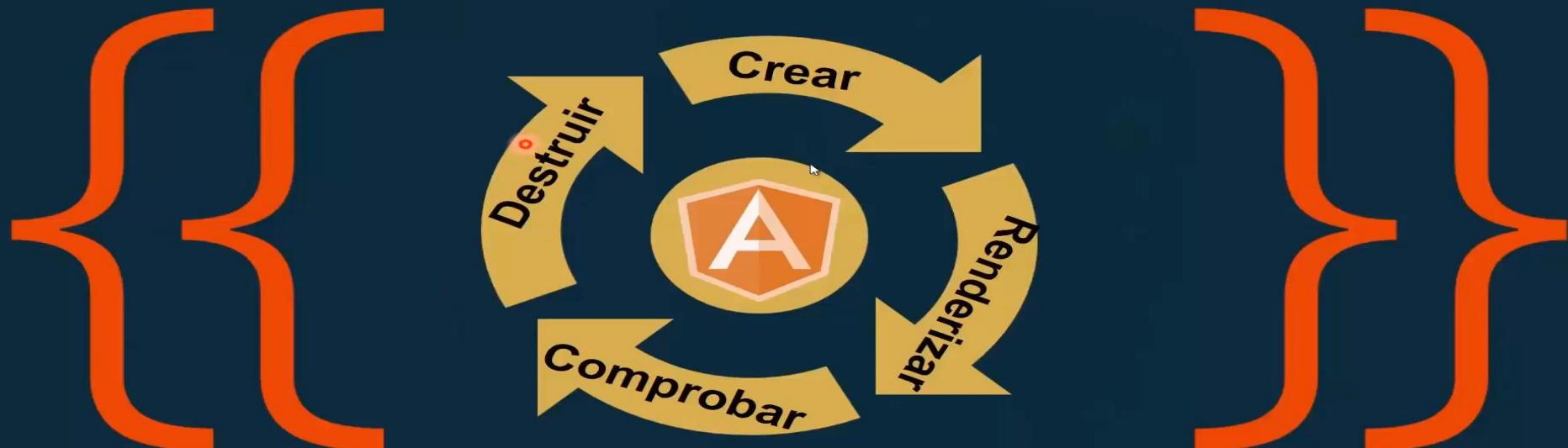
8. Trabajando con componentes

Ciclo de vida de un componente

El término **Hooking** abarca una gama de técnicas utilizadas para alterar o aumentar el comportamiento de un sistema operativo, aplicación o de otros componentes software interceptando llamadas de función, mensajes o eventos pasados entre componentes software.

El código que maneja tales llamadas de función, eventos o mensajes interceptados se llama un **Hoock**.

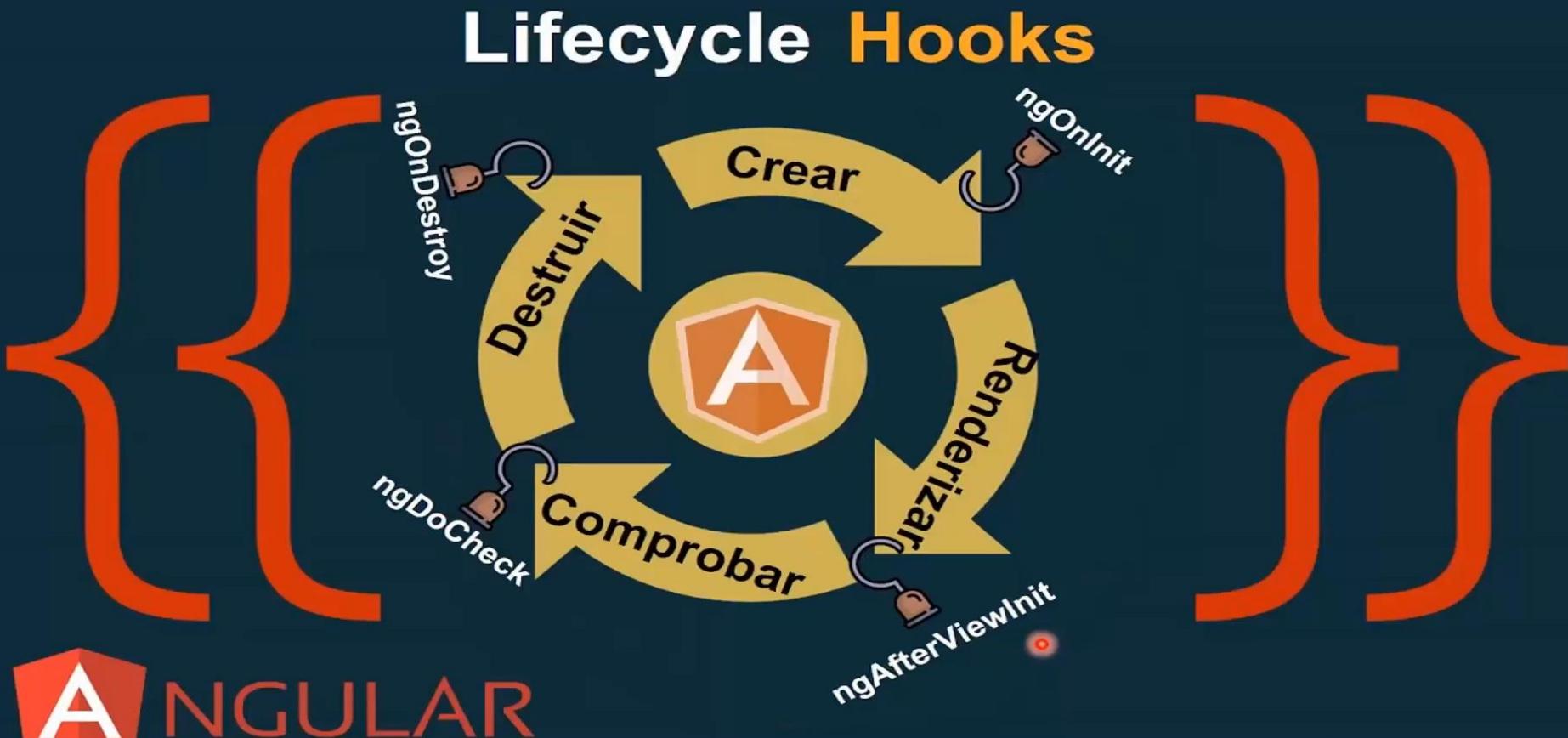
Lifecycle Hooks



8. Trabajando con componentes

Ciclo de vida de un componente

Los **eventos** generados para estos comportamientos son: `ngOnDestroy`, `ngOnInit`, `ngAfterViewInit` y `ngDoCheck`.





8. Trabajando con componentes

Ciclo de vida de un componente

Los **eventos** generados para estos comportamientos son: **ngOnDestroy**, **ngOnInit**, **ngAfterViewInit** y **ngDoCheck**.

```
constructor() {
  console.log("Constructor");
}

ngOnChanges() {
  console.log("ngOnChanges"); //Ejecuta Cambios en Inputs
}

ngOnInit() {
  console.log("ngOnInit"); //Inicializa el contenido del Componente, solo una vez
}

ngDoCheck(){
  console.log("ngDoCheck"); //Detecta cambio en el Componente
}

ngOnDestroy(){
  console.log("ngOnDestroy");
}
```

8. Trabajando con componentes

Ciclo de vida de un componente - **ngOnChanges**

Este método va a ser el primero que se va a ejecutar, se va a ejecutar el primero de todos y tambien se va a ejecutar cuando se produce algun cambio en los valores de las propiedades de nuestro componente.

Para usarlo hay que importar las clases OnChanges y SimpleChanges:

```
import { Component, OnChanges, SimpleChanges } from "@angular/core";
```

Después implementar la interfaz OnChanges en la clase del componente:

```
export class NuevoComponente implements OnChanges {
```

Y por ultimo definir el método **ngOnChanges** que se ejecutara al cargar el componente o hacer cambios en las propiedades.

```
ngOnChanges(changes: SimpleChanges): void {
  throw new Error("Method not implemented.");
}
```

El parámetro changes de tipo SimpleChanges es un objeto que incluye los cambios realizados en las propiedades.

8. Trabajando con componentes

Ciclo de vida de un componente - `ngOnInit`

El hook `ngOnInit` se ejecuta cuando cargamos la directiva de nuestro componente, es el primer método que se ejecuta después de lanzar el constructor de la clase del componente.

Para usarlo debemos importar primero `OnInit`

```
import { Component, OnChanges, SimpleChanges, OnInit } from "@angular/core";
```

Luego deberemos declarar la implementación en mi componente

```
export class NuevoComponente implements OnChanges, OnInit {
```

Finalmente solo debemos implementar el evento en mi componente

```
ngOnInit(): void {
  throw new Error("Method not implemented.");
}
```

8. Trabajando con componentes

Ciclo de vida de un componente - `ngDoCheck`

```
import { Component, OnChanges, SimpleChanges, OnInit, DoCheck } from "@angular/core";  
  
@Component({  
  selector: 'app-nuevo',  
  template:  
    <h1> Hola nuevo componente</h1>  
}  
  
)  
  
export class NuevoComponente implements OnChanges, OnInit, DoCheck {  
  
  ngDoCheck(): void {  
    throw new Error("Method not implemented.");  
  }  
  
  ngOnInit(): void {  
    throw new Error("Method not implemented.");  
  }  
  
  ngOnChanges(changes: SimpleChanges): void {  
    throw new Error("Method not implemented.");  
  }  
}
```

El método **DoCheck** se ejecuta cada vez que se produce algún cambio o evento en el componente o en la aplicación de Angular y es un método que se va a estar ejecutando frecuentemente.



8. Trabajando con componentes

Ciclo de vida de un componente - `ngDoCheck`

Dentro del ciclo de vida del evento `ngDoCheck`, existen una serie de eventos que se ejecutan de acuerdo a diferentes estados:

Lifecycle Hooks

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy



8. Trabajando con componentes

Ciclo de vida de un componente - **ngDoCheck**

Dentro del ciclo de vida del evento **ngDoCheck**, existen una serie de eventos que se ejecutan de acuerdo a diferentes estados (sólo existen para componentes, no para otras entidades como servicios):

```
ngAfterContentInit(){
  console.log("ngAfterContentInit"); //Ejecuta al proyectar contenido
}
ngAfterContentChecked(){
  console.log("ngAfterContentChecked"); //Ejecuta al confirmar el contenido proyectado
}
ngAfterViewInit(){
  console.log("ngAfterViewInit"); //Ejecuta al cargar las View y ViewChild
}
ngAfterViewChecked(){
  console.log("ngAfterViewChecked"); //Ejecuta al confirmar la carga de View y ViewChild
}
```



8. Trabajando con componentes

Data Binding

Uno de los principales valores de Angular es que nos abstrae de la lógica pull/push asociada a insertar y actualizar valores en el HTML y convertir las respuestas de usuario (inputs, clicks, etc) en acciones concretas.

Escribir toda esa lógica a mano (lo que típicamente se hacía con JQuery) es tedioso y propenso a errores, y Angular 2 lo resuelve por nosotros gracias al **Data Binding**.

```
<div>{{todo.subject}}</div>
<todo-detail [todo]="selectedTodo"></todo-detail>
<div (click)="selectTodo(todo)"></div>
```

Data Binding es la capacidad que tiene Angular de relacionar los datos de la capa **TypeScript** con la vista.



8. Trabajando con componentes

Data Binding

- **Interpolación:** (Hacia el DOM)

Al hacer `{{todo.subject}}` , Angular se encarga de insertar el valor de esa propiedad del componente entre las etiquetas `<div>` donde lo hemos definido. Es decir, evalúa `todo.subject` e introduce su resultado en el DOM.

- **Property binding:** (Hacia el DOM)

Al hacer `[todo]="selectedTodo"`, Angular está pasando el **objeto** `selectedTodo` del Componente padre a la propiedad `todo` del Componente hijo, en este caso de `TodoDetailComponent`. Recordemos de la **sección Componentes** que para esto el componente `TodoDetailComponent` habrá definido una propiedad `todo` con el decorador `@Input()`.



- **Event binding:** (Desde el DOM)

Al hacer `(click)="selectTodo(todo)"`, le indicamos a Angular que cuando se produzca un evento `click` sobre esa etiqueta `<div>`, llame al método `selectTodo` del Componente, pasando como atributo el objeto `todo` presente en ese contexto. (Aunque hemos simplificado el ejemplo, esto venía de un bucle que itera el array `todos` obteniendo la variable `todo`).

8. Trabajando con componentes

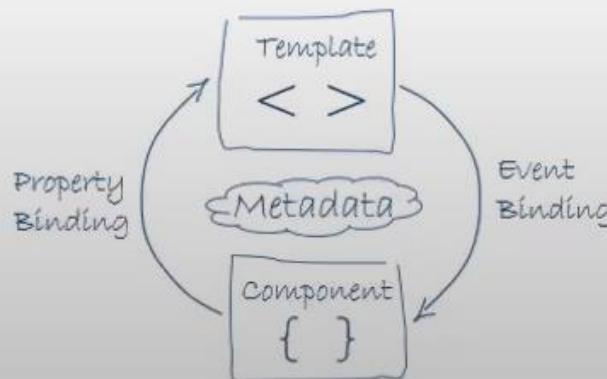
Data Binding

- **Two-way binding:** (Desde/Hacia el DOM)

Un caso importante que no hemos visto con los ejemplos anteriores es el *binding bi-direccional*, que combina *event binding* y *property binding*, como podemos ver en el siguiente ejemplo:

```
<input [(ngModel)]="todo.subject">
```

En este caso, el valor de la propiedad fluye a la caja de *input* como en el caso *property binding*, pero los cambios del usuario también fluyen de vuelta al componente, actualizando el valor de dicha propiedad.



Interpolación: (Hacia el DOM)

Al hacer `{{todo.subject}}` , Angular se encarga de insertar el valor de esa propiedad del componente entre las etiquetas `<div>` donde lo hemos definido. Es decir, evalúa `todo.subject` e introduce su resultado en el DOM.

ever
FUTURE
LOQUE



8. Trabajando con componentes

Data Binding - Interpolación

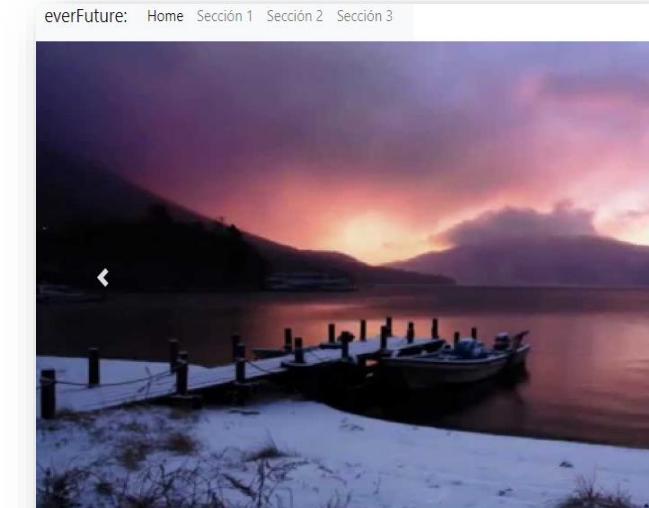
Vamos a crearnos una variable ‘`titulo`’ asociada a la lógica de nuestro componente **Principal** (`principal.component.ts`):

```
src > app > principal > 1 principal.component.ts > ...
...
8   export class PrincipalComponent implements OnInit {
9
10  10  titulo: any = "Angular con everFuture"
11
12  constructor() { }
```

Y tras ello modificaremos la vista asociada para incluirla:

```
src > app > principal > 5 principal.component.html > ...
...
9    <section id="home">
10      <div class="jumbotron">
11        <h1 class="display-4"> {{titulo}} </h1>
12        <p class="lead">Bootstrap está incorporado a Angular.</p>
13        <hr class="my-4">
14        <p>Ya estamos preparados para nuevos pasos...</p>
15      </div>
```

Esto es la interpolación



Angular con everFuture

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...

Al hacer `[todo]="selectedTodo"`, Angular está pasando el **objeto** `selectedTodo` del Componente padre a la propiedad `todo` del Componente hijo, en este caso de `TodoDetailComponent`. Recordemos de la **sección Componentes** que para esto el componente `TodoDetailComponent` habrá definido una propiedad `todo` con el decorador `@Input()`.



8. Trabajando con componentes

Data Binding – Property binding

Ahora vamos a crearnos un nuevo componente ‘hijoPr’ que va a ser hijo del componente **Principal**:

`ng g c hijoPr`

```
PS C:\Dev\workspace\jhernand\proyecto1> ng g c hijoPr
CREATE src/app/hijo-pr/hijo-pr.component.html (22 bytes)
CREATE src/app/hijo-pr/hijo-pr.component.spec.ts (629 bytes)
CREATE src/app/hijo-pr/hijo-pr.component.ts (278 bytes)
CREATE src/app/hijo-pr/hijo-pr.component.css (0 bytes)
UPDATE src/app/app.module.ts (741 bytes)
```

Ahora en la lógica del componente Principal definiremos un objeto ‘personaData’:

```
src > app > principal > principal.component.ts > ...
...
8   export class PrincipalComponent implements OnInit {
9
10  titulo: any = "Angular con everFuture"
11  personaData: any = {nombre: "María", apellidos: "De la O", edad: 27};
```

Como siguiente paso cogemos el ‘selector’ del componente ‘hijoPr’ y lo añadimos a la vista de **Principal**:

```
src > app > principal > principal.component.html > div.container > div.row > div.col-12
...
9    <section id="home">
10      <div class="jumbotron">
11        <h1 class="display-4"> {{titulo}} </h1>
12        <p class="lead">Bootstrap está incorporado a Angular.</p>
13        <hr class="my-4">
14        <p>Ya estamos preparados para nuevos pasos...</p>
15        <app-hijo-pr></app-hijo-pr>
16      </div>
17    </section>
18    <section id="uno">Contenido de la sección 1</section>
```

Al hacer `[todo]="selectedTodo"`, Angular está pasando el **objeto** `selectedTodo` del Componente padre a la propiedad `todo` del Componente hijo, en este caso de `TodoDetailComponent`. Recordemos de la **sección Componentes** que para esto el componente `TodoDetailComponent` habrá definido una propiedad `todo` con el decorador `@Input()`.



8. Trabajando con componentes

Data Binding – Property binding

everFuture: Home Sección 1 Sección 2 Sección 3



Angular con everFuture

Bootstrap está incorporado a Angular.

Yá estamos preparados para nuevos pasos...

hijo-pr works!

Ahora vamos a pasar la variable al hijo desde el selector:

```
src > app > principal > 5 principal.component.html > ...
15           <app-hijo-pr [propiedadHijo]="personaData"></app-hijo-pr>
```

Tras ello en la lógica del hijo la recogeremos:

```
src > app > hijo-pr > A hijo-pr.component.ts > HijoPrComponent > constructor
1   import { Component, OnInit, Input } from '@angular/core';
2
3   @Component({
4     selector: 'app-hijo-pr',
5     templateUrl: './hijo-pr.component.html',
6     styleUrls: ['./hijo-pr.component.css']
7   })
8   export class HijoPrComponent implements OnInit {
9
10   /* Me defino un decorador: */
11   @Input("propiedadHijo") datosPasadoAlHijo: any;
```

Y ya podemos usarla en la vista del hijo:

```
src > app > hijo-pr > 5 hijo-pr.component.html > p
1   <p> {{datosPasadoAlHijo.nombre}}:| {{datosPasadoAlHijo | json}} </p>
```

Property binding: (Hacia el DOM)

Al hacer `[todo]="selectedTodo"`, Angular está pasando el **objeto** `selectedTodo` del Componente padre a la propiedad `todo` del Componente hijo, en este caso de `TodoDetailComponent`. Recordemos de la **sección Componentes** que para esto el componente `TodoDetailComponent` habrá definido una propiedad `todo` con el decorador `@Input()`.



8. Trabajando con componentes

Data Binding – Property binding

everFuture: Home Sección 1 Sección 2 Sección 3



everFuture: Home Sección 1 Sección 2 Sección 3



Angular con everFuture

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...

hijo-pr works!



Angular con everFuture

Bootstrap está incorporado a Angular.

Ya estamos preparados para nuevos pasos...

María: { "nombre": "María", "apellidos": "De la O", "edad": 27 }



- **Two-way binding:** (Desde/Hacia el DOM)

Un caso importante que no hemos visto con los ejemplos anteriores es el *binding bi-direccional*, que combina *event binding* y *property binding*, como podemos ver en el siguiente ejemplo:

```
<input [(ngModel)]="todo.subject">
```



8. Trabajando con componentes

Data Binding – Two-way binding

Ahora vamos a crear un formulario y asociar los campos de nuestro objeto al mismo:

```
personaData: any = {nombre: "María", apellidos: "De la O", edad: 27};
```

Lo primero que faremos será definir en el **app.component.ts** los **imports** que vamos a utilizar:

```
src > app > A app.module.ts > ...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule } from '@angular/forms';
4
5 import { AppRoutingModule } from './app-routing.module';
6 import { PrincipalComponent } from './principal/principal.component';
7 import { HeaderComponent } from './header/header.component';
8 import { FooterComponent } from './footer/footer.component';
9 import { HijoPrComponent } from './hijo-pr/hijo-pr.component';
10
11 @NgModule({
12   declarations: [
13     PrincipalComponent,
14     HeaderComponent,
15     FooterComponent,
16     HijoPrComponent
17   ],
18   imports: [
19     BrowserModule,
20     AppRoutingModule,
21     FormsModule
22   ],
23   providers: [],
24   bootstrap: [PrincipalComponent]
25 })
26 export class AppModule { }
```



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

import { AppRoutingModule } from './app-routing.module';
import { PrincipalComponent } from './principal/principal.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { HijoPrComponent } from './hijo-pr/hijo-pr.component';

@NgModule({
  declarations: [
    PrincipalComponent,
    HeaderComponent,
    FooterComponent,
    HijoPrComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [PrincipalComponent]
})
export class AppModule { }
```

- **Two-way binding:** (Desde/Hacia el DOM)

Un caso importante que no hemos visto con los ejemplos anteriores es el *binding bi-direccional*, que combina *event binding* y *property binding*, como podemos ver en el siguiente ejemplo:

```
<input [(ngModel)]="todo.subject">
```

8. Trabajando con componentes

Data Binding – Two-way binding

A continuación modificaremos la vista del componente **Principal**:

src > app > principal > principal.component.html > div.container > div.row

```
8   <div class="row">
9     <section id="home">
10       <div class="jumbotron">
11         <h1 class="display-4"> {{titulo}} </h1>
12         <p class="lead">Bootstrap está incorporado a Angular.</p>
13         <hr class="my-4">
14         <p>Ya estamos preparados para nuevos pasos...</p>
15       </div>
16     </section>
17     <section id="uno">
18       <h3>Sección 1:</h3>
19       <div class="row">
20         <div class="col-6">
21           <div class="form-group">
22             <label for="usr">Nombre:</label>
23             <input type="text" [(ngModel)]="personaData.nombre" class="form-control">
24           </div>
25           <div class="form-group">
26             <label for="usr">Apellidos:</label>
27             <input type="text" [(ngModel)]="personaData.apellidos" class="form-control">
28           </div>
29
30           <button type="button" class="btn btn-primary btn-blok">Evento Angular</button>
31         </div>
32         <div class="col-6">
33           <div class="form-group">
34             <label for="usr">Edad:</label>
35             <input type="text" [(ngModel)]="personaData.edad" class="form-control">
36           </div>
37           <br/><br/>
38           <app-hijo-pr [propiedadHijo]="personaData"></app-hijo-pr>
39         </div>
40       </div>
41     </section>
42     <section id="dos">Contenido de la sección 2</section>
43     <section id="tres">Contenido de la sección 3</section>
44   </div>
```



```
<section id="uno">
  <h3>Sección 1:</h3>
  <div class="row">
    <div class="col-6">
      <div class="form-group">
        <label for="usr">Nombre:</label>
        <input type="text" [(ngModel)]="personaData.nombre" class="form-control" formControlName="nombre">
      </div>
      <div class="form-group">
        <label for="usr">Apellidos:</label>
        <input type="text" [(ngModel)]="personaData.apellidos" class="form-control" formControlName="apellidos">
      </div>
      <button type="button" class="btn btn-primary btn-blok">Evento Angular</button>
    </div>
    <div class="col-6">
      <div class="form-group">
        <label for="usr">Edad:</label>
        <input type="text" [(ngModel)]="personaData.edad" class="form-control" formControlName="edad">
      </div>
      <br/><br/>
      <app-hijo-pr [propiedadHijo]="personaData"></app-hijo-pr>
    </div>
  </div>
</section>
<section id="dos">Contenido de la sección 2</section>
<section id="tres">Contenido de la sección 3</section>
```



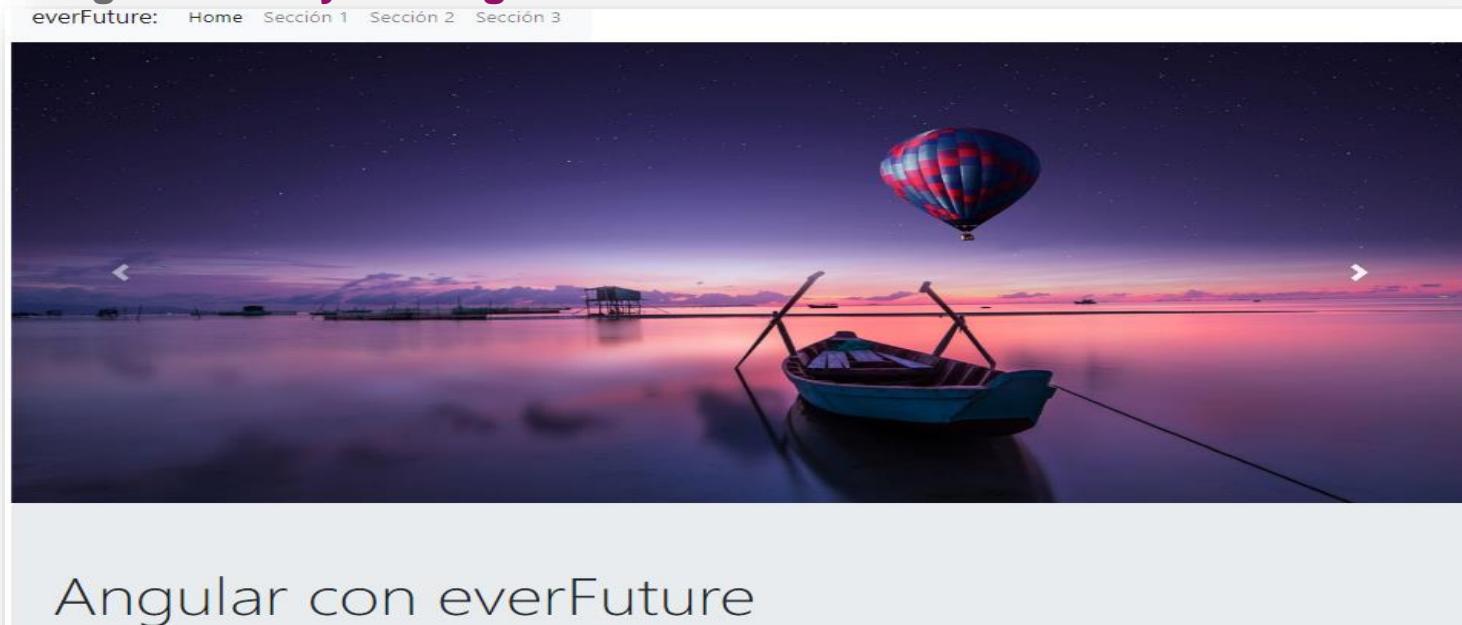
- **Two-way binding:** (Desde/Hacia el DOM)

Un caso importante que no hemos visto con los ejemplos anteriores es el *binding bi-direccional*, que combina *event binding* y *property binding*, como podemos ver en el siguiente ejemplo:

```
<input [(ngModel)]="todo.subject">
```

8. Trabajando con componentes

Data Binding – Two-way binding



ever
FUTURE
LOQUE



Event binding: (Desde el DOM)

Al hacer `(click)="selectTodo(todo)"`, le indicamos a Angular que cuando se produzca un evento `click` sobre esa etiqueta `<div>`, llame al método `selectTodo` del Componente, pasando como atributo el objeto `todo` presente en ese contexto. (Aunque hemos simplificado el ejemplo, esto venía de un bucle que itera el array `todos` obteniendo la variable `todo`).



8. Trabajando con componentes

Data Binding – Event binding

Ahora vamos a asociar un evento al botón creado. Así como primer paso indicaremos que función se va a encargar de atender dicho evento:



```
<button type="button" (click)="eventoBotonAngular()" class="btn btn-primary btn-blok">Evento Angular</button>
```

Tras ello implementamos la lógica del mismo:

```
src > app > principal > principal.component.ts > ...
19   eventoBotonAngular() {
20     console.log ('Ejecutamos el evento del botón para '+this.personaData.nombre+' '+this.personaData.apellidos);
21 }
```

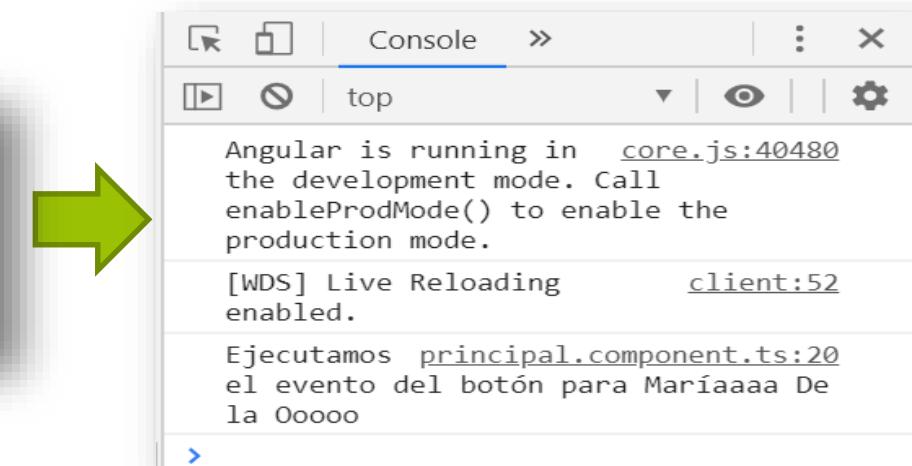
Sección 1:

Nombre: Edad:

Apellidos:

Mariaaaa: { "nombre": "Mariaaaa", "apellidos": "De la Ooooo", "edad": 27 }

Evento Angular



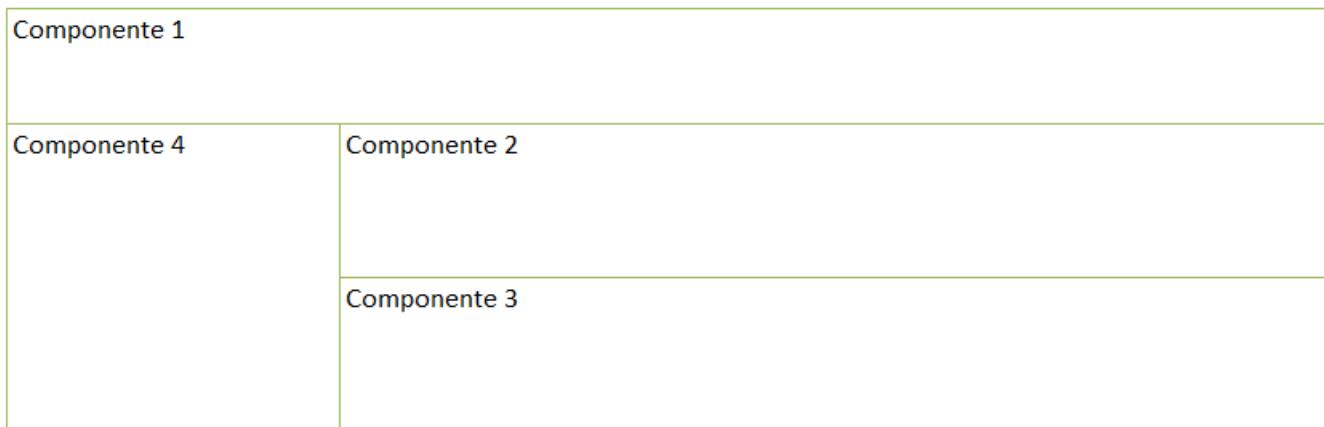


*Time for
Action*

Shift

Actividad final día 5 1/2

Crear una SPA en la que haya un componente principal que dentro contenga a cuatro componentes, enmaquetados con Bootstrap con la siguiente estructura:



- El componente 1 tendrá dos variables curso con el siguiente contenido:

```
curso1 = { 'nombre':'Beca Angular', 'total_días':9, 'descripcion':'Curso sobre Angular y Typescript'};  
curso2 = { 'nombre':'Beca Java', 'total_días':10, 'descripcion':'Formación sobre Java'};  
contador:number=0;
```

- El constructor del componente 4 escribirá el mensaje en consola (con console.log) 'Creación del componente 4.'
- El componente 4 implementará el evento ngOnInit que escribirá en consola 'Iniciando el componente 4'.

Actividad final día 5 2/2

- ➔ El componente 2 va a recibir 2 variables del componente1 ('curso1' y 'contador') y escribirá en el HTML tanto el valor de 'curso1.nombre' como de 'curso1.descripcion' y de 'contador'.
- ➔ El componente 3 va a recibir 2 variables del componente1 ('curso2' y 'contador') y escribirá en el HTML tanto el valor de 'curso2.nombre' como de 'curso2.descripcion' y de 'contador'.
- ➔ El componente 1 contendrá un botón que indicará el texto 'intercambiar' y que al pulsarlo provocará que se incremente en 1 la variable 'contador' y que se intercambien el nombre y la descripción de 'curso1' y de 'curso2'.
- ➔ Además el componente 1 contendrá un input text que estará enlazado (por doble binding) con 'contador' y se podrá modificar manualmente.

```
<input type="number">
```

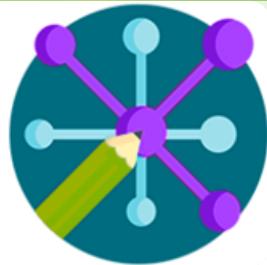




everis

an NTT DATA Company

ever
FUTURE
LOIQUE



everis (an NTT DATA Company)
Consulting, IT & Outsourcing Professional Services