

# 5

## OBJETIVOS

- I. Conocer la diferencia entre librería y framework.
- II. Identificar las características generales de los frameworks a nivel general y de Angular a nivel particular.
- III. Realizar las instalaciones necesarias para implementar proyectos con Angular.
- IV. Crear un proyecto con Angular.
- V. Crear componentes con Angular.



## Índice

<b>NOTAS DEL AUTOR .....</b>	<b>3</b>
<b>INTRODUCCIÓN .....</b>	<b>4</b>
<b>CARACTERÍSTICAS GENERALES .....</b>	<b>4</b>
<b>INSTALACIONES PREVIAS .....</b>	<b>5</b>
NODEJS Y NPM.....	5
TYPESCRIPT .....	6
ANGULARCLI .....	7
<b>ANGULAR .....</b>	<b>8</b>
CREACIÓN DE UN PROYECTO CON ANGULAR .....	8
ESTRUCTURA DE UN PROYECTO EN ANGULAR .....	10
<i>Carpeta “e2e” .....</i>	<i>10</i>
<i>Carpeta “node_modules” .....</i>	<i>11</i>
<i>Carpeta “src” .....</i>	<i>11</i>
<i>Carpeta “app” .....</i>	<i>12</i>
<i>Carpeta “assets” .....</i>	<i>12</i>
<i>Carpeta “environment” .....</i>	<i>12</i>
<i>Carpeta “dist” .....</i>	<i>12</i>
DESPLEGAR UN PROYECTO EN ANGULAR.....	13
FLUJO DE EJECUCIÓN DE UNA APLICACIÓN BÁSICA CON ANGULAR.....	13
COMPONENTES.....	17
<i>Crear un componente.....</i>	<i>17</i>
<i>Estructura del componente .....</i>	<i>18</i>
<i>Añadir el componente a nuestra aplicación .....</i>	<i>19</i>
<i>Partes de un componente .....</i>	<i>19</i>
<b>EJEMPLO PRÁCTICO 1 .....</b>	<b>21</b>
CREAMOS LA APLICACIÓN .....	21
CREACIÓN DEL COMPONENTE COCHES .....	23
MODIFICACIÓN DEL COMPONENTE COCHES .....	23
CREACIÓN DE UNA LISTA DE COCHES .....	26
SELECCIONAR UNA ELEMENTO DE LA LISTA DE COCHES (MAESTRO - DETALLE) .....	28
<b>EJEMPLO PRÁCTICO 2 .....</b>	<b>33</b>

## UT5. Angular

---

CREAMOS LA APLICACIÓN .....	33
CREACIÓN DE LA CLASE CLIENTE .....	34
CREACIÓN Y MODIFICACIÓN DEL COMPONENTE PARA EL FORMULARIO .....	35
CREACIÓN DE LA VISTA DEL FORMULARIO.....	37
RELACIONAR LA VISTA Y EL MODELO .....	38
DETECCIÓN DE ERRORES EN LA ENTRADA DE DATOS.....	39
PROCESAMIENTO DEL FORMULARIO.....	41

## Notas del autor

El propósito de esta documentación es proporcionar al alumno una guía que incluya el acceso a los contenidos que considero necesarios para el desarrollo de esta unidad de trabajo.

Estos contenidos han sido recopilados de diferentes libros, webs y revistas citados todos en el apartado bibliográfico, así como de experiencias personales.

Si llega a tu mano esta documentación, no dudes en usarla, ya que ese es el objetivo que persigo al realizarla, sólo debes tener en cuenta una cosa, **aglutinar toda esta información lleva mucho tiempo por lo que si vas a usar este documento ten el detalle de citar al autor del mismo.**

Víctor Manuel Garrido Cases

Profesor Técnico de Formación Profesional  
Sistemas y Aplicaciones Informáticas

## Introducción

Javascript es un lenguaje interpretado que se ejecuta en el lado del cliente a través de un navegador web. **El lenguaje Javascript es un lenguaje que requiere mucho esfuerzo (líneas de código) para desarrollar cualquier aplicación** por lo que hasta que no **surgieron librerías que facilitarán las labores de desarrollo** no adquirió popularidad. Fue con la aparición de la librería jQuery con la que el uso del lenguaje comenzó a despegar.

Un framework es mucho más una librería, como ya sabemos, **una librería es un conjunto de código que nos facilita la construcción de nuestro programa (jQuery), un framework**, además de esto nos proporciona una homogeneidad a la hora de desarrollar nuestros programas reduciendo la flexibilidad, pero aumentando la productividad.

En este enlace se explica muy bien la diferencia entre framework y librería con un ejemplo muy clarificador <https://www.arquitecturajava.com/framework-vs-libreria/>

En 2010 surge AngularJS comenzando a popularizarse los frameworks de desarrollo en Javascript. Hay muchos más pero Angular es uno de los más populares

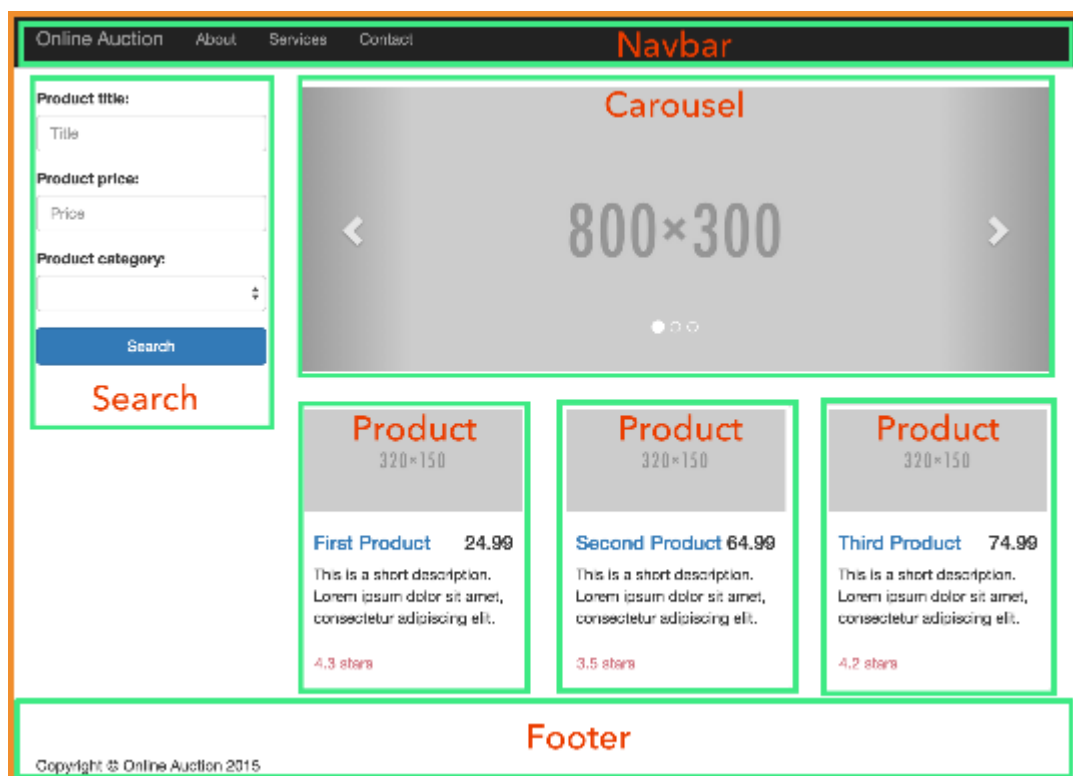
## Características generales

Como hemos visto anteriormente, un framework nos proporciona un entorno de trabajo homogéneo en el que desarrollar nuestras aplicaciones. Entre sus principales ventajas encontramos:

- Eficiencia y velocidad de desarrollo
- Mantenimiento
- Trabajo en equipo
- Seguridad

Angular es uno de los framework más utilizados. Existen dos versiones distintas, AngularJS y Angular (anteriormente Angular 2). Angular tiene las siguientes características.

- **SPA (Single page application) para dar fluidez a la navegación.** Todo lo que se muestra y se procesa está dentro de la misma página, así que al pasar de una opción a otra no hace falta recargar el navegador y lo normal es que sea un único archivo desde el que se reproduce absolutamente todo.
- **Desarrollado en Typescript, que es un derivado de Javascript.** Entre sus principales características se encuentra la declaración tipada de variables y la existencia de objetos basados en clases (ECM6)
- **Arquitectura MVC basada en componentes.**



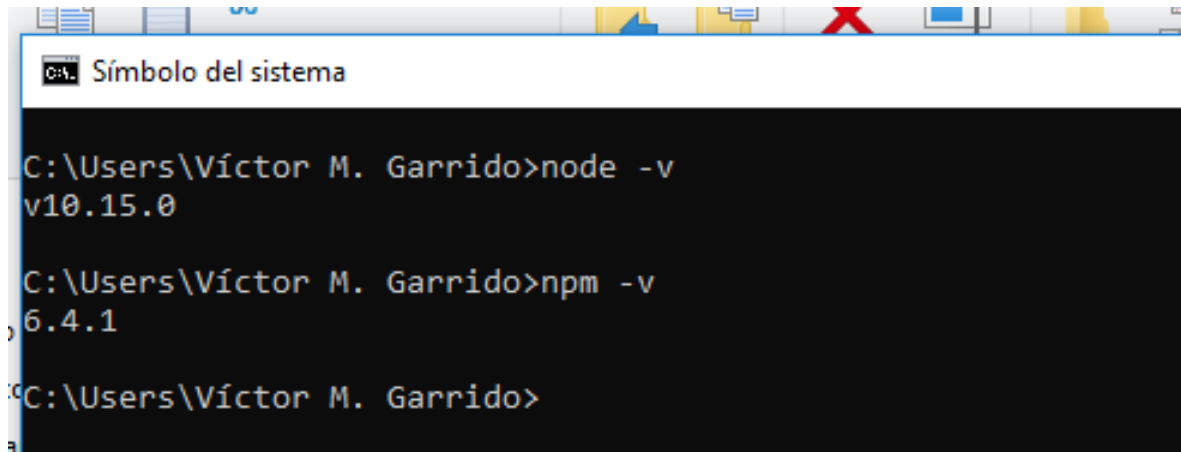
## Instalaciones previas

### NodeJS y npm

Desde la web de NodeJS (<https://nodejs.org>) **nos descargamos la última versión de este framework** que nos va a permitir ejecutar código javascript en el lado del servidor, junto con NodeJS se instalará npm que es un gestor de dependencias que nos permitirá tener organizadas las librerías de terceros que vayamos a usar.

## UT5. Angular

Para asegurarnos que se ha instalado correctamente tecleamos desde una consola `node -v` y `npm -v` para ver la versión que se nos ha instalado



```
C:\Users\Víctor M. Garrido>node -v
v10.15.0

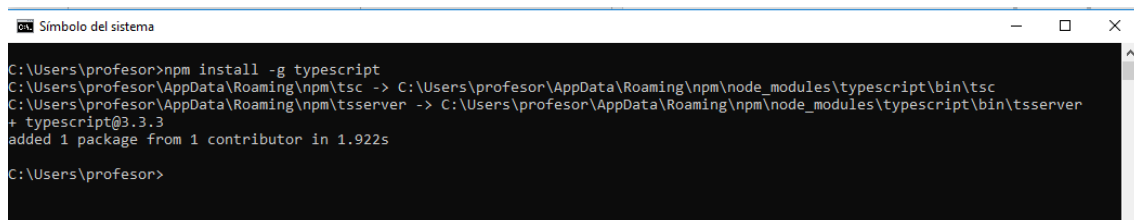
C:\Users\Víctor M. Garrido>npm -v
6.4.1

C:\Users\Víctor M. Garrido>
```

### Typescript

Como los navegadores no entienden el código typescript, vamos a necesitar un **compilador que traduzca nuestro código al lenguaje javascript**. Lo obtendremos ejecutando el siguiente comando:

```
npm install -g typescript
```



```
C:\Users\profesor>npm install -g typescript
C:\Users\profesor\AppData\Roaming\npm\tsc -> C:\Users\profesor\AppData\Roaming\npm\node_modules\typescript\bin\tsc
C:\Users\profesor\AppData\Roaming\npm\tsserver -> C:\Users\profesor\AppData\Roaming\npm\node_modules\typescript\bin\tsserver
+ typescript@3.3.3
added 1 package from 1 contributor in 1.922s

C:\Users\profesor>
```

De forma general, cada vez que nos queramos descargar algún paquete javascript, lo haremos con el comando “npm install”, y, si indicamos el parámetro -g, el paquete se instalará de forma global. De este modo, al habernos bajado el compilador de typescript con este parámetro, podremos utilizar en cualquier sitio el **comando “tsc”** que se ocupará de compilar el código typescript.

Para verificar que todo ha ido bien tecleamos el comando `tsc -v`. y nos aparecerá la versión instalada.

## UT5. Angular

AngularCLI

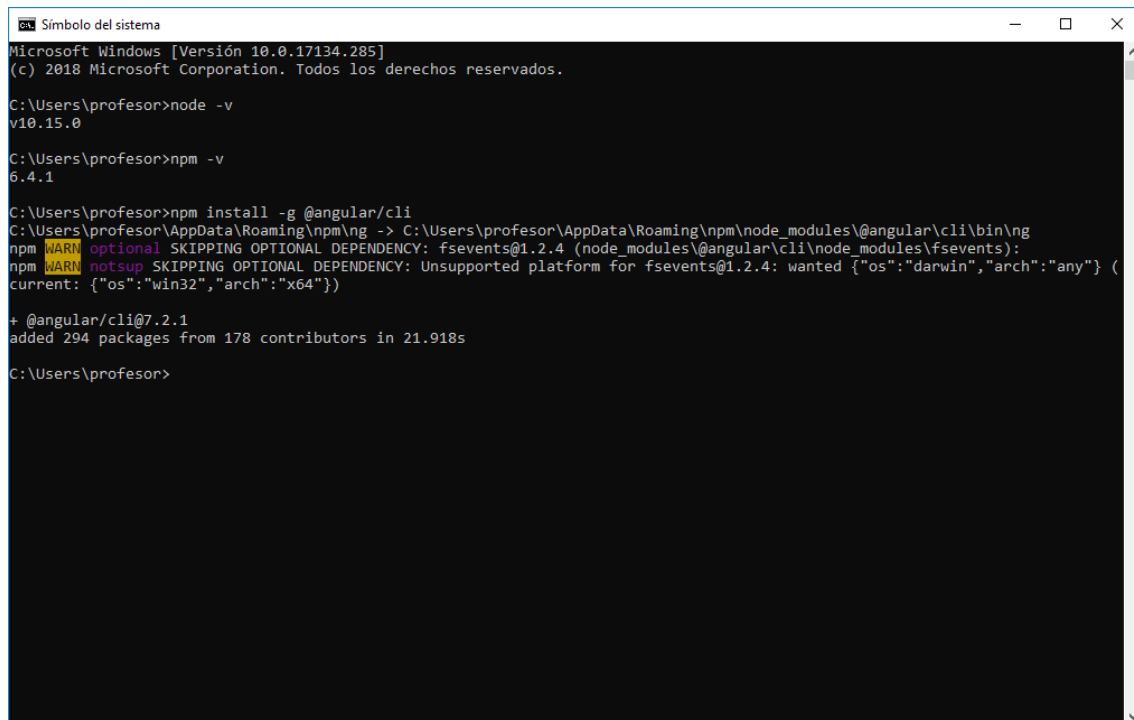
Ahora vamos a descargar **AngularCLI** que es una de las herramientas esenciales para desarrollar con el nuevo framework Angular 2.

Para instalarlo teclearemos desde una consola el siguiente comando

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

Si os encontráis por alguna documentación esta forma de instalarlo `npm install -g angular-cli` **indicaros que está desactualizada** y debéis usar la anterior.

Comenzará el proceso de instalación que durará varios minutos



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.17134.285]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\profesor>node -v
v10.15.0

C:\Users\profesor>npm -v
6.4.1

C:\Users\profesor>npm install -g @angular/cli
C:\Users\profesor\AppData\Roaming\npm\ng -> C:\Users\profesor\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\@angular\cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (
current: {"os":"win32","arch":"x64"})

+ @angular/cli@7.2.1
added 294 packages from 178 contributors in 21.918s

C:\Users\profesor>
```

Una vez instalado dispondrás del comando "ng" a partir del cual lanzarás cualquiera de las acciones que se pueden hacer mediante la interfaz de comandos de Angular. Puedes comenzar lanzando el comando de ayuda:

```
ng help
```



## Angular

### Creación de un proyecto con Angular

Para la creación de un nuevo proyecto basta con ejecutar desde la línea de comandos:

```
ng new <nombreProyecto>
```

Nos aparecerán las siguientes preguntas:

- Do you want to enforce stricter type checking and stricter bundle budgets in the workspace?
  - Yes
- Would you like to add Angular routing?
  - Yes
- Which stylesheet format would you like to use?
  - CSS

Se creará una carpeta igual que el nombre del proyecto indicado y dentro de ella se generarán una serie de subcarpetas y archivos.

Además, 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. Entre otras cosas tendremos:

- Un servidor web 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.

Este proceso tardará varios minutos. Una vez creado el proyecto inicial podemos entrar en la carpeta con el comando cd.

```
cd miPrimerProyectoA
```

## Qué es un sistema de routing?

En **cualquier sitio web** generalmente tienes varias direcciones que son entregadas por un servidor, para mostrar diferentes contenidos del sitio. Podemos tener una portada, una página de productos, una de contacto, etc. Cada una de esas páginas se presenta en una ruta diferente del sitio web, que podrían ser como `example.com`, `example.com/productos/index.html`, `example.com/contacto.html`, etc. Cada una de esas rutas podría tener un archivo HTML, que se sirve con el contenido de esa sección, es así, en líneas generales como funcionan prácticamente todos los sitios web.

Sin embargo, **en las aplicaciones Angular sólo tenemos una página**, el `index.html` y toda la acción se desarrolla dentro de esa página. En Angular lo común es que el index sólo tenga un componente en su BODY y realmente toda la acción se desarrollará en ese componente. **Todas las "páginas" (pantallas o vistas) del sitio web se mostrarán sobre ese índice, intercambiando el componente que se esté visualizando en cada momento.**

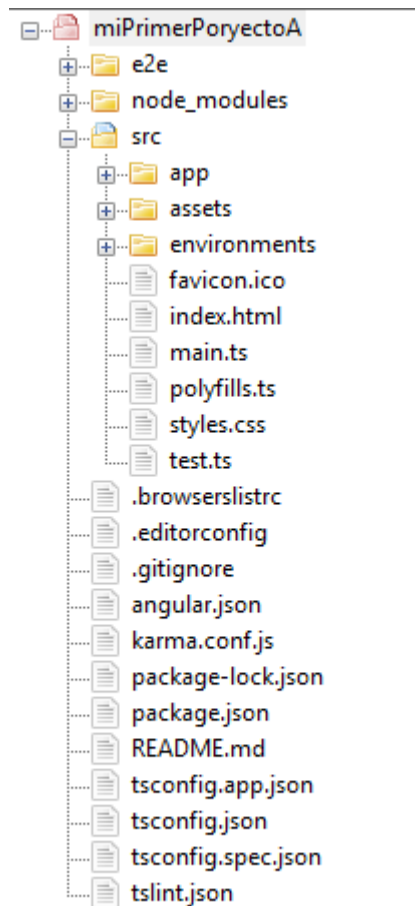
Para facilitar la navegación por un sitio donde realmente sólo hay un index, existe lo que llamamos el sistema de routing, que tiene el objetivo de permitir que en el sitio web haya rutas internas, respondiendo a rutas "virtuales" como las que existen en los sitios tradicionales.

Llamamos "virtuales" a esas rutas, porque realmente sólo existe un "index.html", no habrá un archivo "contacto.html" o "productos.html" para cada ruta, sino que será realmente siempre el "index.html" el que se entregue al navegador

**El sistema de routing es el encargado de reconocer cuál es la ruta que el usuario quiere mostrar, presentando la pantalla correcta en cada momento.**

## Estructura de un proyecto en Angular

Una vez creado nuestro primer proyecto, podemos observar la existencia de numerosas carpetas y archivos. Vamos a ver qué contiene cada una de estas carpetas.



### Carpeta "e2e"

Es para el desarrollo de las pruebas de integración. Viene de "end to end" testing. Este tipo de pruebas permiten asegurar el funcionamiento integral de nuestra aplicación. Las pruebas de integración de Angular funcionan con la librería Protractor ( [www.protractorest.org](http://www.protractorest.org) ). Esta librería hace pruebas de la aplicación dentro de un navegador real, interactuando con ella como lo haría un usuario real.

UT5. Angular

---

*Carpeta “node\_modules”*

Son los archivos de las dependencias que mantenemos vía npm. Por tanto, **todas las librerías que se declaren como dependencias en el archivo `package.json` deben estar descargados en esta carpeta `node_modules`.** Esta carpeta podría haber estado dentro de `src`, pero está colgando de la raíz porque vale tanto para las pruebas, como para la aplicación cuando la estás desarrollando.

*Carpeta “src”*

En la carpeta `src` es **donde vas a realizar todo tu trabajo como desarrollador** ya que es el lugar donde se coloca el código fuente de tu proyecto, más concretamente en la carpeta “**app**” que está dentro de él. Viene con diversos contenidos:

- `index.html`. Fichero que debe servir como página de inicio.
- `favicon.ico`. El icono que se mostrará en la ventana del navegador.
- `main.ts`. El archivo raíz a partir del cual empezará a compilarse nuestra aplicación.
- `polyfills.ts`. Un archivo en el que podremos ocuparnos de temas de compatibilidad con diversos navegadores.
- `styles.css`. El archivo donde tendremos los estilos globales del proyecto, aunque normalmente los estilos irán definidos a nivel de componente.
- `test.ts`. El archivo raíz a partir del cual se compilarán los tests de unidad que creemos.

Existirá además uno o **varios archivos `tsconfig*`**, que son código de configuración de fuente TypeScript. También encontrarás **ficheros `*.ts`**, estos archivos solo existen en la etapa de desarrollo, es decir, en el proyecto que el navegador debe consumir no encontrarás archivos `.ts`, básicamente porque el navegador no entiende TypeScript. Esos archivos son los que se compilarán para producir el código `.js` que sí entienda el navegador.

## UT5. Angular

### Carpeta “app”

Dentro de esta carpeta tenemos una serie de **ficheros llamados component.\***. Estos archivos son el componente principal de nuestra aplicación.

Junto a estos ficheros tenemos uno que se llama **app.module.ts** que es el **módulo raíz** que indicará cómo montar la aplicación a partir de los componentes.

```
C:\Users\Víctor M. Garrido\miPrimerProyecto\src\app>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 864F-43C9

Directorio de C:\Users\Víctor M. Garrido\miPrimerProyecto\src\app

10/01/2019  08:36    <DIR>          .
10/01/2019  08:36    <DIR>          ..
10/01/2019  08:36                0 app.component.css
10/01/2019  08:36               23 app.component.html
10/01/2019  08:36            1.046 app.component.spec.ts
10/01/2019  08:36             214 app.component.ts
10/01/2019  08:36            437 app.module.ts
                5 archivos             1.720 bytes
                2 dirs    4.345.290.752 bytes libres
```

### Carpeta “assets”

En esta carpeta irán todas las imágenes y ficheros estáticos que vayamos a necesitar para el despliegue de nuestro proyecto.

### Carpeta “enviroment”

Contiene variables de entorno de nuestra aplicación.

### Carpeta “dist”

Es la versión de tu aplicación que subirás al servidor web para hacer público el proyecto. En dist aparecerán todos los archivos que el navegador va a necesitar y nunca código fuente en lenguajes no interpretables por él. Muy probablemente tengas que iniciar el servidor web integrado en Angular CLI para que aparezca la carpeta "dist" en el directorio de tu proyecto.

## UT5. Angular

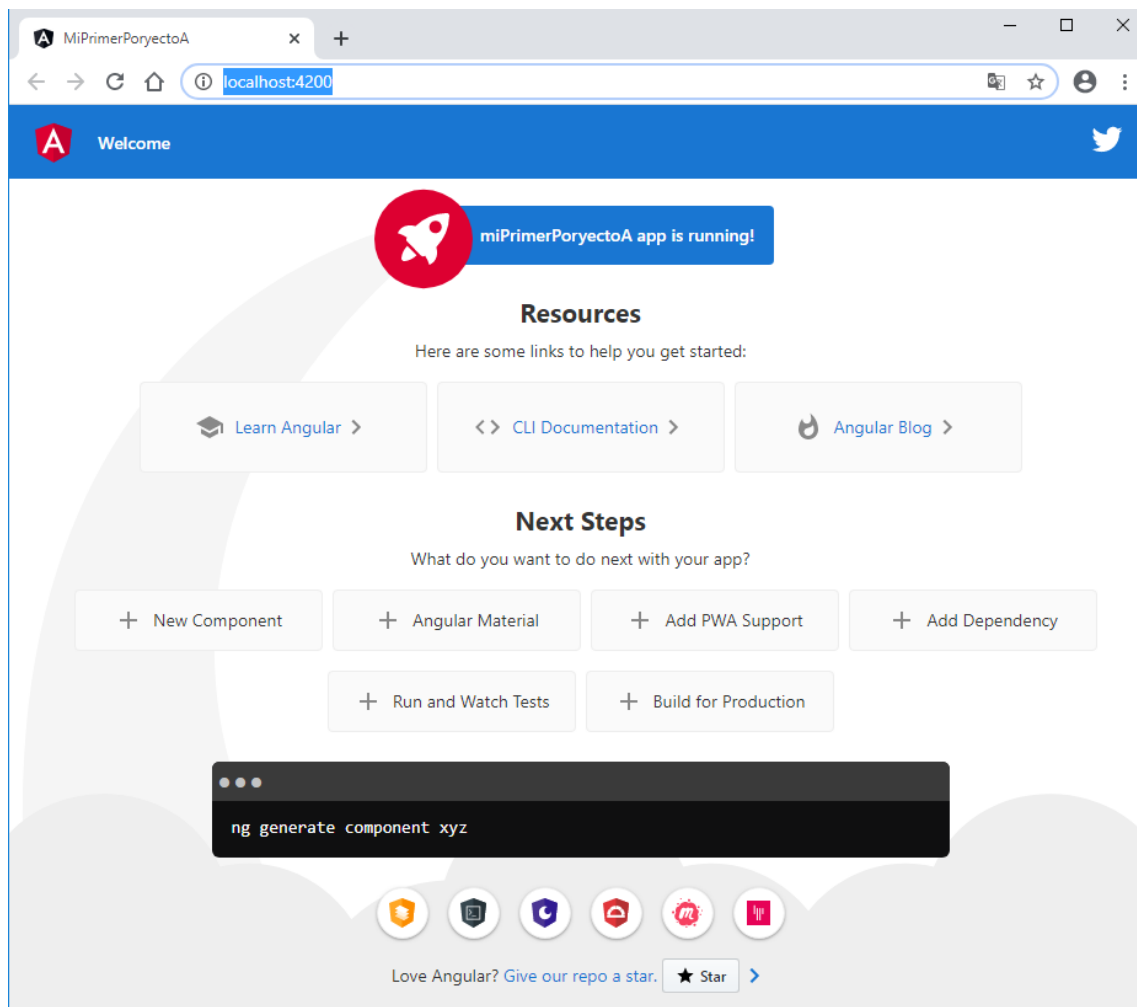
## Desplegar un proyecto en Angular

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" desde línea de comando.

```
ng serve --open
```

Se nos abrirá una pestaña en nuestro navegador por defecto con la página inicial de nuestro proyecto. Si no se nos abre, podemos acceder entrando a <http://localhost:4200>.

Teniendo el servidor abierto de esta forma, la pestaña se nos recargará cada vez que hagamos cualquier cambio en nuestro código.



## Flujo de ejecución de una aplicación básica con Angular

Vamos a ver el flujo de ejecución de una aplicación básica realizada con Angular.

## UT5. Angular

Como acabamos de ver se encuentra en el directorio src. Es el archivo principal de nuestra aplicación web y **el primero que se ejecuta**. Tiene el siguiente aspecto:



```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>MiPrimerProyecto</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
```

Es un fichero bastante sencillo, sin código javascript y con el body prácticamente vacío. Tenemos un componente raíz de la aplicación llamado "app-root" y que posteriormente veremos cómo se genera.

En Angular y en la mayoría de librerías y frameworks actuales se ha optado por crear las aplicaciones en base a componentes. Existe un componente global, que es la aplicación entera y a su vez éste se basa en otros componentes para implementar cada una de sus partes. Cada componente tiene una representación y una funcionalidad. En resumen, **las aplicaciones se construyen mediante un árbol de componentes**, que se apoyan unos en otros para resolver las necesidades. En el `index.html` encontramos lo que sería el componente raíz de este árbol.

Si vemos el código que se genera en tiempo de ejecución veremos muchas diferencias

## UT5. Angular

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>MiPrimerProyecto</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <style type="text/css">...</style>
    <style>...</style>
  </head>
  <body> == $0
    <app-root _ngcontent-c0 ng-version="7.2.0">
      <div _ngcontent-c0 style="text-align:center">
        <h1 _ngcontent-c0> Welcome to miPrimerProyecto! </h1>
        
      </div>
      <h2 _ngcontent-c0>Here are some links to help you start: </h2>
      <ul _ngcontent-c0>
        <li _ngcontent-c0>
          <h2 _ngcontent-c0>
            <a _ngcontent-c0 href="https://angular.io/tutorial" rel="noopener" target="_blank">
              Tour of Heroes</a>
          </h2>
        </li>
        <li _ngcontent-c0>
          <h2 _ngcontent-c0>
            <a _ngcontent-c0 href="https://angular.io/cli" rel="noopener" target="_blank">CLI
              Documentation</a>
          </h2>
        </li>
        <li _ngcontent-c0>
          <h2 _ngcontent-c0>
            <a _ngcontent-c0 href="https://blog.angular.io/" rel="noopener" target="_blank">
              Angular blog</a>
          </h2>
        </li>
      </ul>
      <router-outlet _ngcontent-c0></router-outlet>
    </app-root>
    <script type="text/javascript" src="runtime.js"></script>
    <script type="text/javascript" src="polyfills.js"></script>
    <script type="text/javascript" src="styles.js"></script>
    <script type="text/javascript" src="vendor.js"></script>
    <script type="text/javascript" src="main.js"></script>
  </body>
</html>

```

En tiempo de ejecución es inyectado el código al componente principal de nuestra aplicación.

Al servir la aplicación (`ng serve`), o al llevarla a producción (`ng build`) la herramienta Webpack genera los paquetes (bundles) de código del proyecto y coloca los correspondientes scripts en el index.html, para que todo funcione.

El archivo por el que Webpack comienza a producir los bundles es el `main.ts`, **es el segundo en ejecutarse** y tiene el siguiente aspecto:

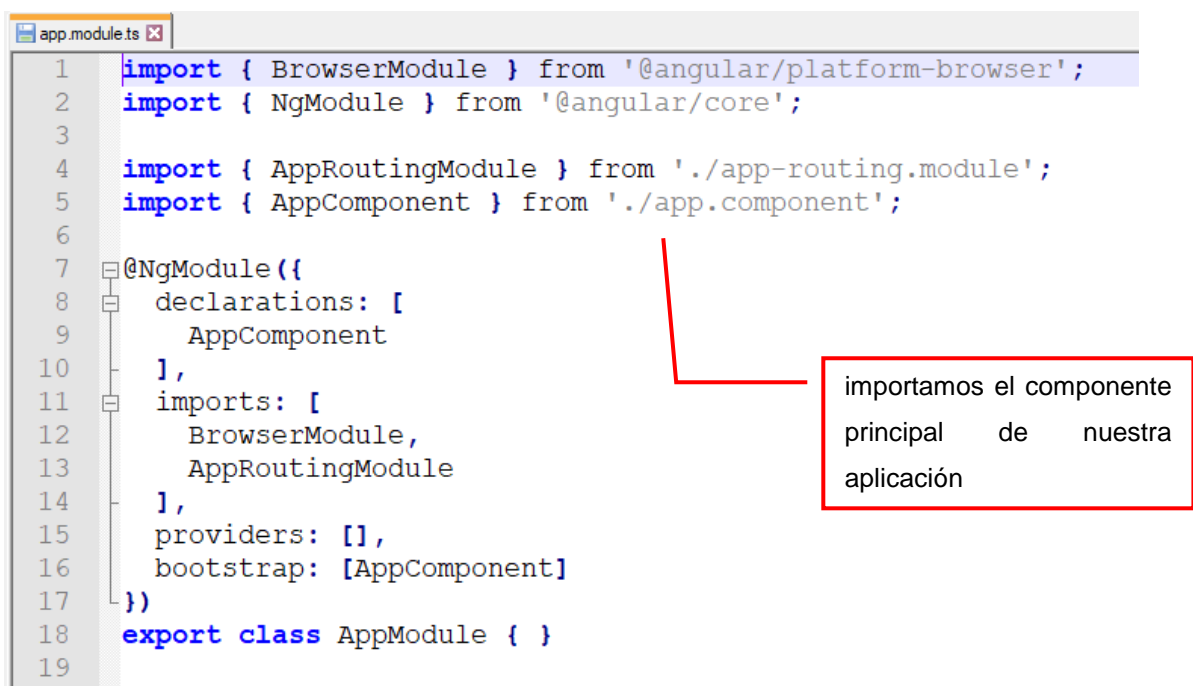


## UT5. Angular



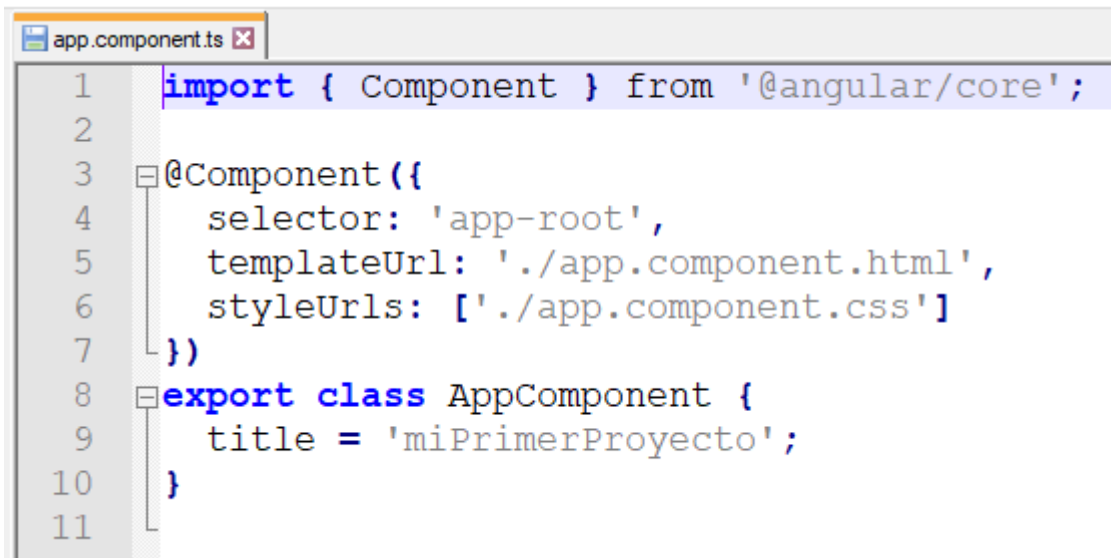
Dentro del `main.ts` encontramos diferentes imports, unos a librerías externas y otros imports a código interno de nuestra aplicación. Inicialmente encontramos un import del módulo principal (`AppModule`) y la llamada al sistema de arranque (`bootstrapModule`) en la que pasamos por parámetro el módulo principal de la aplicación.

**El tercer fichero** es `app.module.ts` y tiene el siguiente aspecto



En el módulo principal se importa el componente raíz (`AppComponent`) y en el decorador `@NgModule` se indica que este componente forma parte del bootstrap.

**Por último llegamos al componente en sí**, que se encuentra en el fichero `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 = 'miPrimerProyecto';
10 }
11
```

El código del componente raíz tiene el **selector "app-root"**, que es la etiqueta que aparecía en el html dónde queramos incluir el componente.

El template del componente raíz, contiene el HTML que se visualiza al poner en marcha la aplicación en el navegador.

## Componentes

En nuestra primera aplicación hemos introducido el concepto de componente. **Un componente nos es más que un elemento que nosotros creamos y que puede realizar las funciones que nosotros le digamos.** Para definir un componente es necesario html, css y por supuesto javascript para definir su funcionalidad.

Los componentes son las piezas fundamentales de las aplicaciones en Angular ya que **una aplicación desarrollada con este framework es un árbol de componentes**. Si pensamos en una página web, tenemos un árbol de etiquetas, siendo BODY la raíz de la parte del contenido. La diferencia es que las etiquetas generalmente son para mostrar un contenido, mientras que los componentes no solo encapsulan un contenido, sino también una funcionalidad.

### Crear un componente

Para crear un nuevo componente nos serviremos de AngularCLI. Teclearemos desde nuestra ventana de comandos y dentro de nuestro proyecto el nuevo componente de la siguiente forma

## UT5. Angular

ng generate component nombreDelComponente

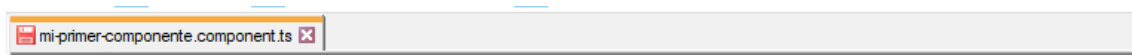
```
C:\Users\profesor\miPrimerProyecto>ng generate component miPrimerComponente
CREATE src/app/mi-primer-componente/mi-primer-componente.component.html (39 bytes)
CREATE src/app/mi-primer-componente/mi-primer-componente.component.spec.ts (714 bytes)
CREATE src/app/mi-primer-componente/mi-primer-componente.component.ts (323 bytes)
CREATE src/app/mi-primer-componente/mi-primer-componente.component.css (0 bytes)
UPDATE src/app/app.module.ts (527 bytes)
```

Nos crea los ficheros que componen la estructura de nuestro componente

*Estructura del componente*

Los ficheros generados son:

1. `NombreDelComponente.component.html`. Aquí ira el código html asociado al componente.
2. `NombreDelComponente.component.spec.ts`. Aquí
3. `NombreDelComponente.component.ts`. Este fichero contiene el código javascript y el decorador del componente y tiene la siguiente estructura.



```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-mi-primer-componente',
5   templateUrl: './mi-primer-componente.component.html',
6   styleUrls: ['./mi-primer-componente.component.css']
7 })
8 export class MiPrimerComponenteComponent implements OnInit {
9
10
11   constructor() { }
12
13
14   ngOnInit() {
15   }
16
17
18 }
19
```

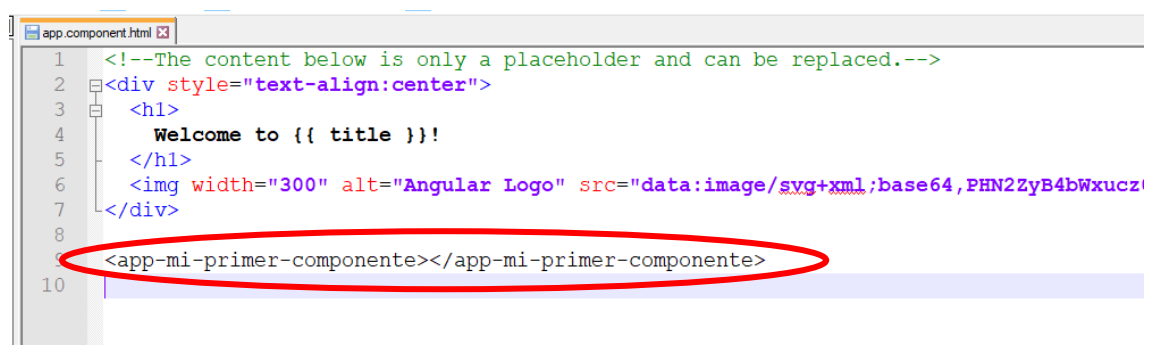
- a. La propiedad **“selector”** es el nombre de la etiqueta nueva que crearemos cuando se procese el componente. Es la etiqueta que usarás cuando quieras colocar el componente en cualquier lugar del HTML.

## UT5. Angular

- b. La propiedad **“templateUrl”** es el nombre del archivo .html con el contenido del componente, en otras palabras, el que tiene el código de la vista.
  - c. La propiedad **“styleUrls”** es un array con todas las hojas de estilos CSS que deben procesarse como estilo local para este componente. Como ves, podríamos tener una única declaración de estilos, o varias si lo consideramos necesario.
4. **NombreDelComponente.component.css**. Aquí irá todo el código css asociado al componente.

### Añadir el componente a nuestra aplicación

Para añadir el componente creado a nuestra aplicación tendremos que modificar el fichero app.component.html añadiendo nuestro componente tal y como se indica en la siguiente imagen



```
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   ng new listadoCoches-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE listadoCoches-app/angular.json (3867 bytes)
CREATE listadoCoches-app/package.json (1317 bytes)
CREATE listadoCoches-app/README.md (1033 bytes)
CREATE listadoCoches-app/tsconfig.json (435 bytes)
CREATE listadoCoches-app/tslint.json (2824 bytes)
CREATE listadoCoches-app/.editorconfig (246 bytes)
CREATE listadoCoches-app/.gitignore (587 bytes)
CREATE listadoCoches-app/src/favicon.ico (5430 bytes)
CREATE listadoCoches-app/src/index.html (303 bytes)
CREATE listadoCoches-app/src/main.ts (372 bytes)
CREATE listadoCoches-app/src/polyfills.ts (3571 bytes)
CREATE listadoCoches-app/src/test.ts (642 bytes)
CREATE listadoCoches-app/src/styles.css (80 bytes)
CREATE listadoCoches-app/src/browserslist (388 bytes)
CREATE listadoCoches-app/src/karma.conf.js (980 bytes)
CREATE listadoCoches-app/src/tsconfig.app.json (166 bytes)
CREATE listadoCoches-app/src/tsconfig.spec.json (256 bytes)
CREATE listadoCoches-app/src/tslint.json (314 bytes)
CREATE listadoCoches-app/src/assets/.gitkeep (0 bytes)
CREATE listadoCoches-app/src/environments/environment.prod.ts (51 bytes)
CREATE listadoCoches-app/src/environments/environment.ts (662 bytes)
CREATE listadoCoches-app/src/app/app-routing.module.ts (245 bytes)
CREATE listadoCoches-app/src/app/app.module.ts (393 bytes)
CREATE listadoCoches-app/src/app/app.component.html (1152 bytes)
CREATE listadoCoches-app/src/app/app.component.spec.ts (1128 bytes)
CREATE listadoCoches-app/src/app/app.component.ts (221 bytes)
CREATE listadoCoches-app/src/app/app.component.css (0 bytes)
CREATE listadoCoches-app/e2e/protractor.conf.js (752 bytes)
CREATE listadoCoches-app/e2e/tsconfig.e2e.json (213 bytes)
CREATE listadoCoches-app/e2e/src/app.e2e-spec.ts (309 bytes)
CREATE listadoCoches-app/e2e/src/app.po.ts (204 bytes)
npm WARN deprecated circular-json@0.5.9: CircularJSON is in maintenance only, flattened
> node-sass@4.10.0 install C:\Users\profesor\listadoCoches-app\node_modules\node-sass
> node scripts/install.js

Cached binary found at C:\Users\profesor\AppData\Roaming\npm-cache\node-sass\4.10.0\w
> node-sass@4.10.0 postinstall C:\Users\profesor\listadoCoches-app\node_modules\node-
> node scripts/build.js

Binary found at C:\Users\profesor\listadoCoches-app\node_modules\node-sass\vendor\win
Testing binary
Binary is fine
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.6 (node_modules\fsevents
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.6

added 1141 packages from 1182 contributors and audited 40183 packages in 47.107s
found 0 vulnerabilities

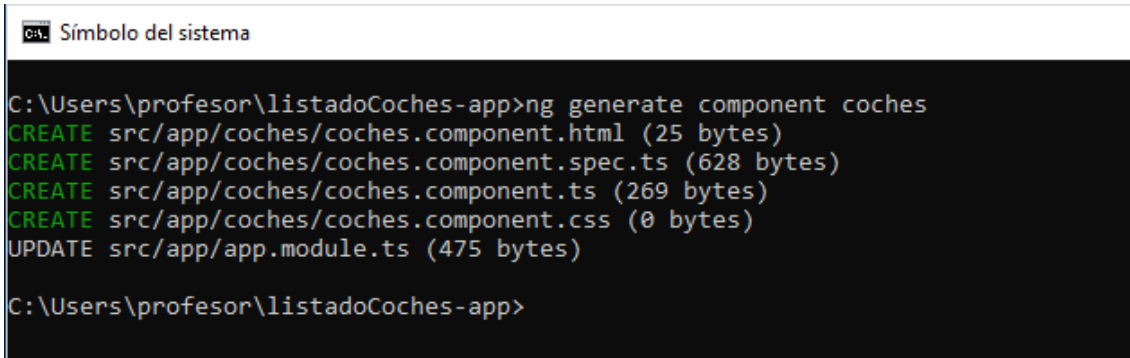
"git" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\profesor>cd listadoCoches-app

C:\Users\profesor\listadoCoches-app>
```

## Creación del componente coches

Creamos el componente coches que nos servirá para mostrar las propiedades de un coche. Desde el propio proyecto tecleamos `ng generate component coches`



```
C:\Users\profesor\listadoCoches-app>ng generate component coches
CREATE src/app/coches/coches.component.html (25 bytes)
CREATE src/app/coches/coches.component.spec.ts (628 bytes)
CREATE src/app/coches/coches.component.ts (269 bytes)
CREATE src/app/coches/coches.component.css (0 bytes)
UPDATE src/app/app.module.ts (475 bytes)

C:\Users\profesor\listadoCoches-app>
```

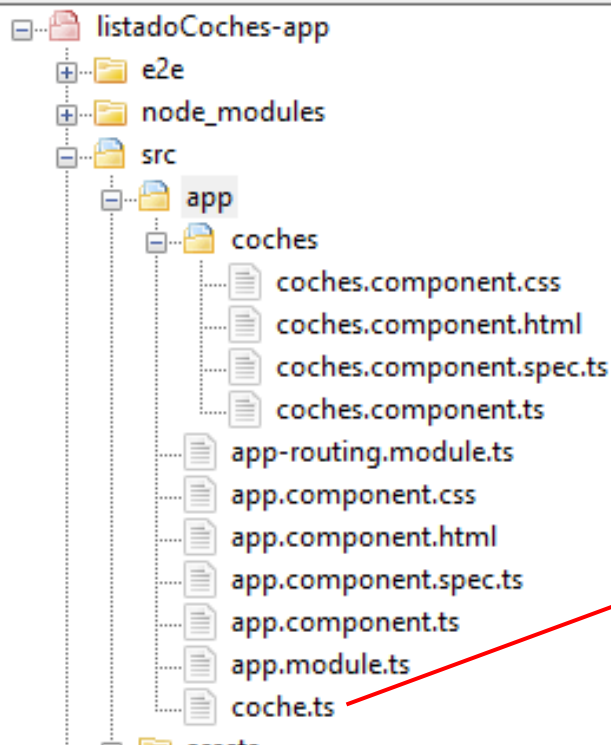
## Modificación del componente coches

Puesto que queremos que de cada coche se muestren una serie de datos como pueden ser: color, matricula, número de puertas, etc, vamos a ver lo que tenemos que hacer para que nuestra web nos pueda proporcionar toda esta información.

1. Vamos a crear una clase que nos muestre la información que queremos almacenar. Para hacer esto creamos el fichero `coche.ts` dentro de la carpeta `app`



## UT5. Angular



Creamos el fichero `coche.ts` dentro de la carpeta `app`

2. Añadimos al fichero creado en el paso anterior la información que queremos almacenar de cada coche

```
1 export class Coche {  
2  
3  
4     modelo! : string;  
5     matricula! : string;  
6     color! : string;  
7     nPuertas! : number;  
8  
9 }  
10
```

Desde typescript 2.7 en adelante, si indicamos que queremos comprobación estricta de tipos en la creación del proyecto, debemos inicializar los atributos de las clases o indicar el símbolo "!" en su declaración.

3. Modificamos el componente y le añadimos una propiedad que será de la clase que acabamos de crear.

## UT5. Angular

```

coches.component.ts
1  import { Component, OnInit } from '@angular/core';
2  import { Coche } from '../coche';
3
4  @Component({
5    selector: 'app-coches',
6    templateUrl: './coches.component.html',
7    styleUrls: ['./coches.component.css']
8  })
9  export class CochesComponent implements OnInit {
10
11    coche : Coche = {
12
13      modelo    : 'Alfa Romeo Giulia',
14      matricula  : '8020GCV',
15      color     : 'negro',
16      nPuertas  : 4
17    };
18
19    constructor() { }
20
21    ngOnInit() {
22    }
23  }
24

```

4. Modificamos el html del componente para que nos aparezcan los datos de nuestro coche.

```

coches.component.html
1  <!--coches.component.html-->
2  <h2>Datos del vehículo: {{coche.modelo}}</h2>
3  <div><span>Matricula:</span> {{coche.matricula}}</div>
4  <div><span>Color:</span> {{coche.color}}</div>
5  <div><span>Nº de puertas:</span> {{coche.nPuertas}}</div>

```

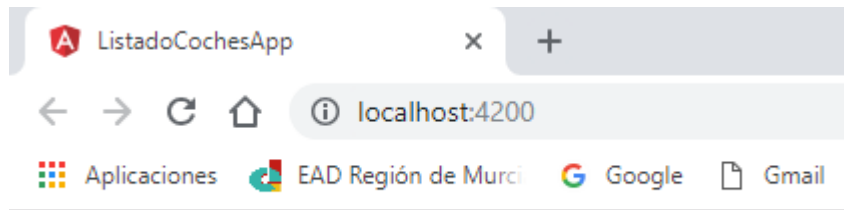
5. Añadimos nuestro componente en el fichero app.component.html quedando con el siguiente aspecto.

```

app.component.html
1  <!--app.component.html-->
2  <app-coches></app-coches>
3

```

Si no hemos cometido ningún error, nos debe aparecer una web con el siguiente aspecto:



## Datos del vehículo: Alfa Romeo Giulia

Matricula: 8020GCV

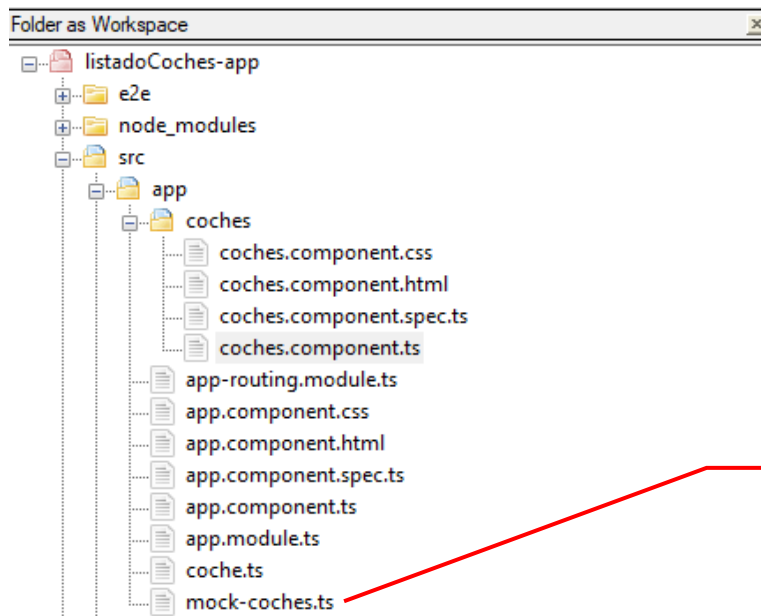
Color: negro

Nº de puertas: 4

### Creación de una lista de coches

Puesto que el objetivo de nuestra aplicación es crear un listado de coches, vamos a ver los pasos que tenemos que dar para convertir lo que hemos hecho en el apartado anterior en un array de coches, lo mostraremos como una lista.

1. En primer lugar, necesitaremos coger una lista de coches de algún sitio. Normalmente estos datos los cogeremos de un servicio web pero como no lo tenemos nos vamos a crear una lista con estos datos. Creamos un fichero llamado `mock-coches.ts` en `app`



Creamos el fichero `mock-coche.ts` dentro de la carpeta `app`

2. Modificamos el fichero creado en el paso anterior para que contenga la lista de coches.

## UT5. Angular

```

mock-coches.ts
1 // src/app/mock-coches.ts
2
3 import { Coche } from './coche';
4
5 export const COCHES: Coche[] = [
6   { modelo: 'Alfa Romeo Giulia', matricula: '8020GCV', color: 'negro', nPuertas: 4 },
7   { modelo: 'Mercedes c200 cdi', matricula: '3536HFS', color: 'plateado', nPuertas: 4 },
8   { modelo: 'Renault megane', matricula: '5075GWX', color: 'rojo', nPuertas: 5 },
9   { modelo: 'Wolswagen', matricula: '2000BCK', color: 'azul', nPuertas: 3 },
10 ];

```

3. Importaremos esa colección en nuestro componente CochesComponent y lo añadiremos como una propiedad

```

coches.component.ts
1 import { Component, OnInit } from '@angular/core';
2 import { Coche } from '../coche';
3 import { COCHES } from '../mock-coches';
4
5 @Component({
6   selector: 'app-coches',
7   templateUrl: './coches.component.html',
8   styleUrls: ['./coches.component.css']
9 })
10 export class CochesComponent implements OnInit {
11
12   coches = COCHES;
13
14   constructor() { }
15
16   ngOnInit() {
17   }
18 }

```

4. Ahora tenemos que mostrar la información por pantalla, para ello modificamos el fichero coches.component.html y lo dejamos tal y como se muestra en la siguiente imagen.

```

coches.component.html
1 <!--coches.component.html-->
2
3 <h2>Mis coches</h2>
4
5 <ul class="coches">
6   <table>
7     <tr>
8       <td *ngFor="let coche of coches">
9         <tr><span class="destacado">{{coche.modelo}}</span></tr>
10        <tr><span class="caracteristica">Matricula: </span>{{coche.matricula}} </tr>
11        <tr><span class="caracteristica">Color: </span> {{coche.color}}</tr>
12        <tr><span class="caracteristica">N. puertas: </span> {{coche.nPuertas}}</tr>
13      </td>
14    </tr>
15  </table>
16 </ul>

```

Hemos usado la directiva `*ngFor` que nos permite iterar sobre una colección de elementos de forma que se imprime el listado a la vez que

## UT5. Angular

se realiza la iteración. En la imagen anterior estamos diciendo que el elemento `<td>` se repita tantas veces como elementos haya en la colección.

##### 5. Modificamos el css del componente para mejorar el aspecto

The image shows a code editor with the following CSS code in `coches.component.css`:

```

1  /*coches.component.css*/
2
3  td {background-color:#D3D3D3;
4      padding: 15px;}
5
6  .destacado {color: red;
7              font-weight: bold;
8              text-transform: uppercase;
9  }
10 .caracteristica {font-weight: bold;}

```

Below the code editor is a browser preview of the application 'ListadoCochesApp' running on `localhost:4200`. The page title is 'Mis coches' and it displays a table with four car models:

<b>ALFA ROMEO GIULIA</b> Matricula: 8020GCV Color: negro N. puertas: 4	<b>MERCEDES C200 CDI</b> Matricula: 3536HFS Color: plateado N. puertas: 4	<b>RENAULT MEGANE</b> Matricula: 5075GWX Color: rojo N. puertas: 5	<b>WOLSVAGEN POLO</b> Matricula: 2000BCK Color: azul N. puertas: 3
---	--	---	---

### Seleccionar una elemento de la lista de coches (maestro - detalle)

Vamos a hacer ahora que cuando yo seleccione uno de los coches que tengo, muestra toda la información asociada a ese modelo.

Para ello lo primero que vamos a hacer es modificar los ficheros que hace de “base de datos” en nuestro ejemplo, estos son `coche.ts` y `mock-coches.ts` quedando con el siguiente aspecto:

## UT5. Angular

```

1 // src/app/coche.ts
2
3 export class Coche {
4
5     modelo      : string;
6     matricula   : string;
7     color       : string;
8     nPuertas    : number;
9     cCrucero    : string;
10    manoLibres   : string;
11    aseguradora   : string;
12    techoSolar   : string;
13
14 }

```

```

1 // src/app/mock-coches.ts
2
3 import { Coche } from './coche';
4
5 export const COCHES: Coche[] = [
6     { modelo: 'Alfa Romeo Giulia', matricula: '8020GCV', color: 'negro', nPuertas: 4, cCrucero: 'Si', manoLibres: 'Si', aseguradora: 'Mapfre', techoSolar: 'No' },
7     { modelo: 'Mercedes c200 cdi', matricula: '3536HFS', color: 'plateado', nPuertas: 4, cCrucero: 'No', manoLibres: 'Si', aseguradora: 'AXA', techoSolar: 'No' },
8     { modelo: 'Renault megane', matricula: '5075GXX', color: 'rojo', nPuertas: 5, cCrucero: 'Si', manoLibres: 'No', aseguradora: 'Allians', techoSolar: 'Si' },
9     { modelo: 'Wolswagen Polo', matricula: '2000BCK', color: 'azul', nPuertas: 3, cCrucero: 'No', manoLibres: 'No', aseguradora: 'Caser', techoSolar: 'No' },
10 ];

```

Una vez añadida más información a nuestra “base de datos” comenzamos con el proceso.

1. Añadimos el evento “click” al elemento td en el fichero `coches.component.ts` quedando el fichero con el siguiente aspecto.

```

1
2
3 <h2>Mis coches</h2>
4
5 <ul class="coches">
6     <table>
7         <tr>
8             <td *ngFor="let coche of coches" (click)="seleccionarVehiculo(coche)">
9
10                <h2 class="destacado">{{coche.modelo}}</h2>
11                <div><span class="caracteristica">Matricula: </span>{{coche.matricula}}</div>
12                <div><span class="caracteristica">Color: </span>{{coche.color}}</div>
13                <div><span class="caracteristica">Nº de puertas: </span>{{coche.nPuertas}}</div>
14            </td>
15        </tr>
16    </table>
17    <tr>
18        <td colspan="4">
19            <app-coche-detalle [coche]="cocheSeleccionado"></app-coche-detalle>
20        </td>
21    </tr>
22 </ul>
23
24

```

## UT5. Angular

Le decimos al componente que al pulsar sobre el `<td>` se llame a la función `onSelect` del componente.

2. Creamos la función `onSelect` en el fichero `coches.component.ts`.

```

coches.component.ts
1  import { Component, OnInit } from '@angular/core';
2  import { Coche } from '../coche';
3  import { COCHES } from '../mock-coches';
4
5  @Component({
6    selector: 'app-coches',
7    templateUrl: './coches.component.html',
8    styleUrls: ['./coches.component.css']
9  })
10
11  export class CochesComponent implements OnInit {
12
13
14    coches = COCHES;
15
16    cocheSeleccionado! : Coche;
17
18    seleccionarVehiculo(coche: Coche): void {
19
20      this.cocheSeleccionado = coche;
21
22    }
23
24    constructor() { }
25
26    ngOnInit(): void {
27
28    }
29  }
30

```

3. Creamos un nuevo componente que llamaremos `cocheDetalle` (`ng generate component cocheDetalle`) y que será el que nos muestre la información del coche seleccionado.
4. Modificamos el html de este nuevo componente. Usaremos la directiva `*ngIf` para controlar si la ficha detalle debe mostrarse.

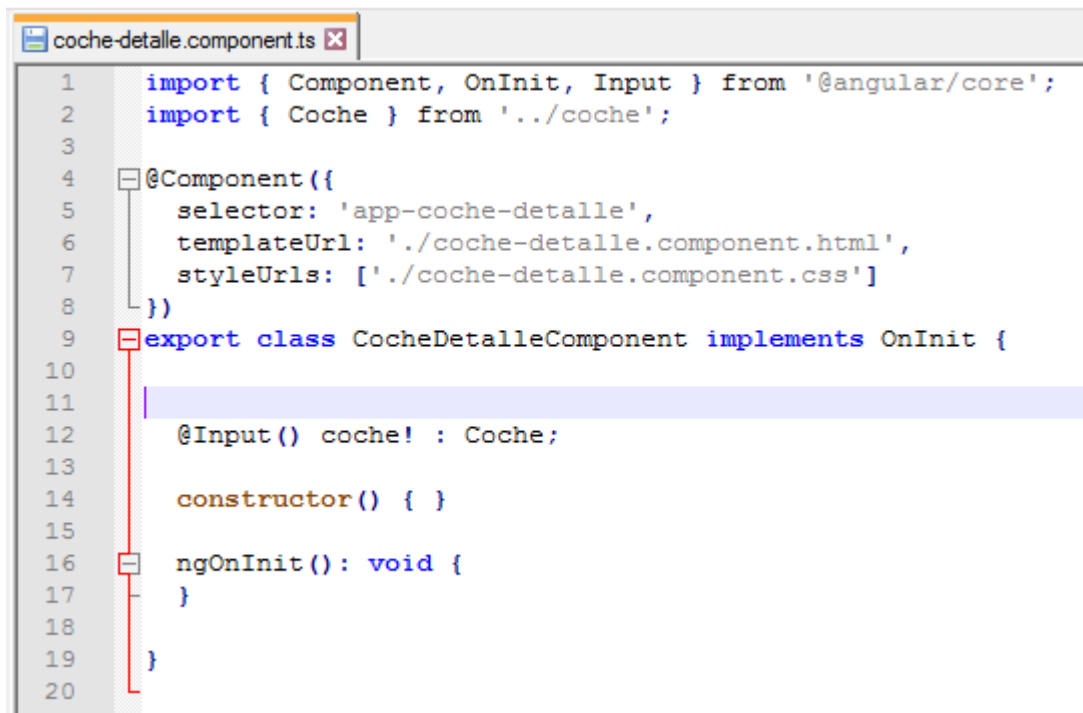
```

coche-detalle.component.html
1  <div *ngIf="coche">
2    <h2 class="destacado">Detalles de {{ coche.modelo | uppercase }}</h2>
3    <div><span class="caracteristica">Matricula: </span>{{coche.matricula}}</div>
4    <div><span class="caracteristica">Color: </span>{{coche.color}}</div>
5    <div><span class="caracteristica">Puertas: </span>{{coche.nPuertas}}</div>
6    <div><span class="caracteristica">Control de crucero: </span>{{coche.cCrucero}}</div>
7    <div><span class="caracteristica">Manos libres: </span>{{coche.manoLibres}}</div>
8    <div><span class="caracteristica">Aseguradora: </span>{{coche.aseguradora}}</div>
9    <div><span class="caracteristica">Techo solar: </span>{{coche.techoSolar}}</div>
10  </div>

```

## UT5. Angular

5. Ahora vamos a declarar la propiedad “coche” en el componente, ya que la estamos usando en nuestra vista. Vamos a anotar la propiedad con el decorador `@Input`, ya que la propiedad vendrá introducida directamente por otro componente:



```

1  import { Component, OnInit, Input } from '@angular/core';
2  import { Coche } from '../coche';
3
4  @Component({
5    selector: 'app-coche-detalle',
6    templateUrl: './coche-detalle.component.html',
7    styleUrls: ['./coche-detalle.component.css']
8  })
9  export class CocheDetalleComponent implements OnInit {
10
11
12    @Input() coche! : Coche;
13
14    constructor() { }
15
16    ngOnInit(): void {
17    }
18
19  }
20

```

6. Ahora actualizamos el componente `coche` para que nos muestre el componente `detalleCoche` que acabamos de crear y que es donde queremos que aparezca la información detallada del coche. Añadiremos también una clase al elemento seleccionado para que se resalte del resto



Detalle información

Clase seleccionado, la creamos en el css de nuestro componente con el aspecto que queremos.

```

1  <h2>Mis coches</h2>
2
3  <ul class="coches">
4    <table>
5      <tr>
6        <td *ngFor="let coche of coches" [class.seleccionado]="coche===cocheSeleccionado" (click)="seleccionarVehiculo(coche)">
7          <h2 class="destacado">{{coche.modelo}}</h2>
8          <div><span class="caracteristica">Matricula: </span>{{coche.matricula}}</div>
9          <div><span class="caracteristica">Color: </span>{{coche.color}}</div>
10         <div><span class="caracteristica">Nº de puertas: </span>{{coche.nPuertas}}</div>
11       </td>
12     </tr>
13     <tr>
14       <td colspan="4">
15         <app-coche-detalle [coche]="cocheSeleccionado"></app-coche-detalle>
16       </td>
17     </tr>
18   </table>
19 </ul>
20
21
22
23
24

```

7. Modificamos los css a nuestro gusto



## UT5. Angular

```
coches.component.css
1  td { background-color: #D3D3D3;
2    padding: 15px; }
3
4  .destacado { color: red;
5              font-weight: bold;
6              text-transform: uppercase; }
7
8  .caracteristica { font-weight: bold; }
9
10 .seleccionado { background-color: #086A87; }
11
12
```

```
coche-detalle.component.css
1  /*coches-detalle.component.css*/
2
3  .destacado { color: red;
4              font-weight: bold;
5              text-transform: uppercase; }
6
7  .caracteristica { font-weight: bold; }
8
```

```
coches.component.css
1  td { background-color: #D3D3D3;
2    padding: 15px; }
3
4  .destacado { color: red;
5              font-weight: bold;
6              text-transform: uppercase; }
7
8  .caracteristica { font-weight: bold; }
9
10 .seleccionado { background-color: #086A87; }
11
12 .detalle { background-color: #F5F6CE; }
```

El resultado final debe tener el siguiente aspecto

## UT5. Angular



## Ejemplo práctico 2

Para practicar con lo que hemos visto hasta ahora con Angular e introducir la realización de formularios, vamos a construir un **sistema de alta de clientes y un listado de clientes** que irá incrementando ítems, a medida que los demos de alta.

En Angular disponemos de dos formas de realizar formularios

- **Formularios guiados por plantilla.** Los formularios guiados por la plantilla son aquellos formularios en las que la lógica, la validación, los controles, están dentro de la propia vista, en forma de html.
- **Formularios reactivos.** A diferencia de los anteriores, van a tener, por un lado, los controles (referenciados como objetos) con los datos que hayamos puesto, y, por el otro, los datos “persistentes” y no va a haber un vínculo directo entre ellos.

## Creemos la aplicación

Como ya hemos visto, usaremos AngularCLI para crear nuestra aplicación.

Tecleamos `ng new clientes-app`

## UT5. Angular

```

C:\Users\profesor>ng new clientes-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE clientes-app/angular.json (3822 bytes)
CREATE clientes-app/package.json (1311 bytes)
CREATE clientes-app/README.md (1028 bytes)
CREATE clientes-app/tsconfig.json (435 bytes)
CREATE clientes-app/tslint.json (2824 bytes)
CREATE clientes-app/.editorconfig (246 bytes)
CREATE clientes-app/.gitignore (587 bytes)
CREATE clientes-app/src/favicon.ico (5430 bytes)
CREATE clientes-app/src/index.html (298 bytes)
CREATE clientes-app/src/main.ts (372 bytes)
CREATE clientes-app/src/polyfills.ts (3571 bytes)
CREATE clientes-app/src/test.ts (642 bytes)
CREATE clientes-app/src/styles.css (80 bytes)
CREATE clientes-app/src/browserslist (388 bytes)
CREATE clientes-app/src/karma.conf.js (980 bytes)
CREATE clientes-app/src/tsconfig.app.json (166 bytes)
CREATE clientes-app/src/tsconfig.spec.json (256 bytes)
CREATE clientes-app/src/tslint.json (314 bytes)
CREATE clientes-app/src/assets/.gitkeep (0 bytes)
CREATE clientes-app/src/environments/environment.prod.ts (51 bytes)
CREATE clientes-app/src/environments/environment.ts (662 bytes)
CREATE clientes-app/src/app/app-routing.module.ts (245 bytes)
CREATE clientes-app/src/app/app.module.ts (393 bytes)
CREATE clientes-app/src/app/app.component.html (1152 bytes)
CREATE clientes-app/src/app/app.component.spec.ts (1113 bytes)
CREATE clientes-app/src/app/app.component.ts (216 bytes)
CREATE clientes-app/src/app/app.component.css (0 bytes)
CREATE clientes-app/e2e/protractor.conf.js (752 bytes)
CREATE clientes-app/e2e/tsconfig.e2e.json (213 bytes)
CREATE clientes-app/e2e/src/app.e2e-spec.ts (304 bytes)
CREATE clientes-app/e2e/src/app.po.ts (204 bytes)
npm WARN deprecated circular-json@0.5.9: CircularJSON is in maintenance only, flattened is its successor.

> node-sass@4.10.0 install C:\Users\profesor\clientes-app\node_modules\node-sass
> node scripts/install.js

Cached binary found at C:\Users\profesor\AppData\Roaming\npm-cache\node-sass\4.10.0\win32-x64-64_binding.node

> node-sass@4.10.0 postinstall C:\Users\profesor\clientes-app\node_modules\node-sass
> node scripts/build.js

Binary found at C:\Users\profesor\clientes-app\node_modules\node-sass\vendor\win32-x64-64_binding.node
Testing binary
Binary is fine
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.6 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.6: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

added 1141 packages from 1182 contributors and audited 40183 packages in 59.062s
found 0 vulnerabilities

"git" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.


C:\Users\profesor>cd clientes-app
C:\Users\profesor\clientes-app>

```

## Creación de la clase cliente

Vamos a crear una clase llamada cliente, para ellos tecleamos

## UT5. Angular

 Símbolo del sistema

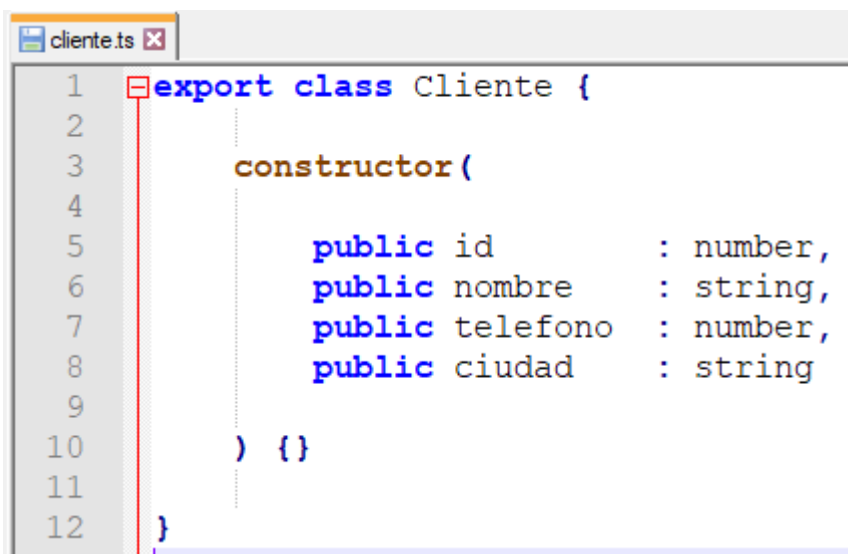
```
Microsoft Windows [Versión 10.0.17134.285]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\profesor>cd clientes-app

C:\Users\profesor\clientes-app>ng generate class cliente
CREATE src/app/cliente.spec.ts (158 bytes)
CREATE src/app/cliente.ts (25 bytes)

C:\Users\profesor\clientes-app>
```


En el fichero generado añadimos la siguiente información



```
1 export class Cliente {
2
3     constructor(
4
5         public id      : number,
6         public nombre  : string,
7         public telefono: number,
8         public ciudad  : string
9
10    ) {}
11
12 }
```

### Creación y modificación del componente para el formulario

Creamos en primer lugar el componente ClienteForm

 Símbolo del sistema

```
C:\Users\profesor\clientes-app>ng generate component ClienteForm
CREATE src/app/cliente-form/cliente-form.component.html (31 bytes)
CREATE src/app/cliente-form/cliente-form.component.spec.ts (664 bytes)
CREATE src/app/cliente-form/cliente-form.component.ts (292 bytes)
CREATE src/app/cliente-form/cliente-form.component.css (0 bytes)
UPDATE src/app/app.module.ts (497 bytes)

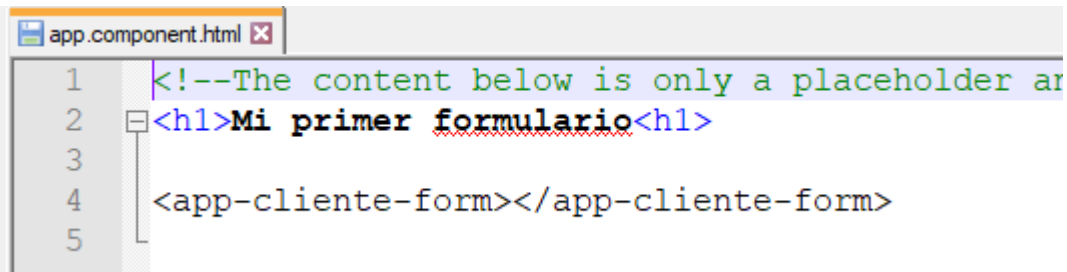
C:\Users\profesor\clientes-app>
```

Cómo los formularios guiados por plantilla tienen su propio módulo, vamos a incluirlo en nuestro módulo raíz, para ellos modificamos el fichero `app.module.ts` añadiendo la siguiente línea de código

## UT5. Angular

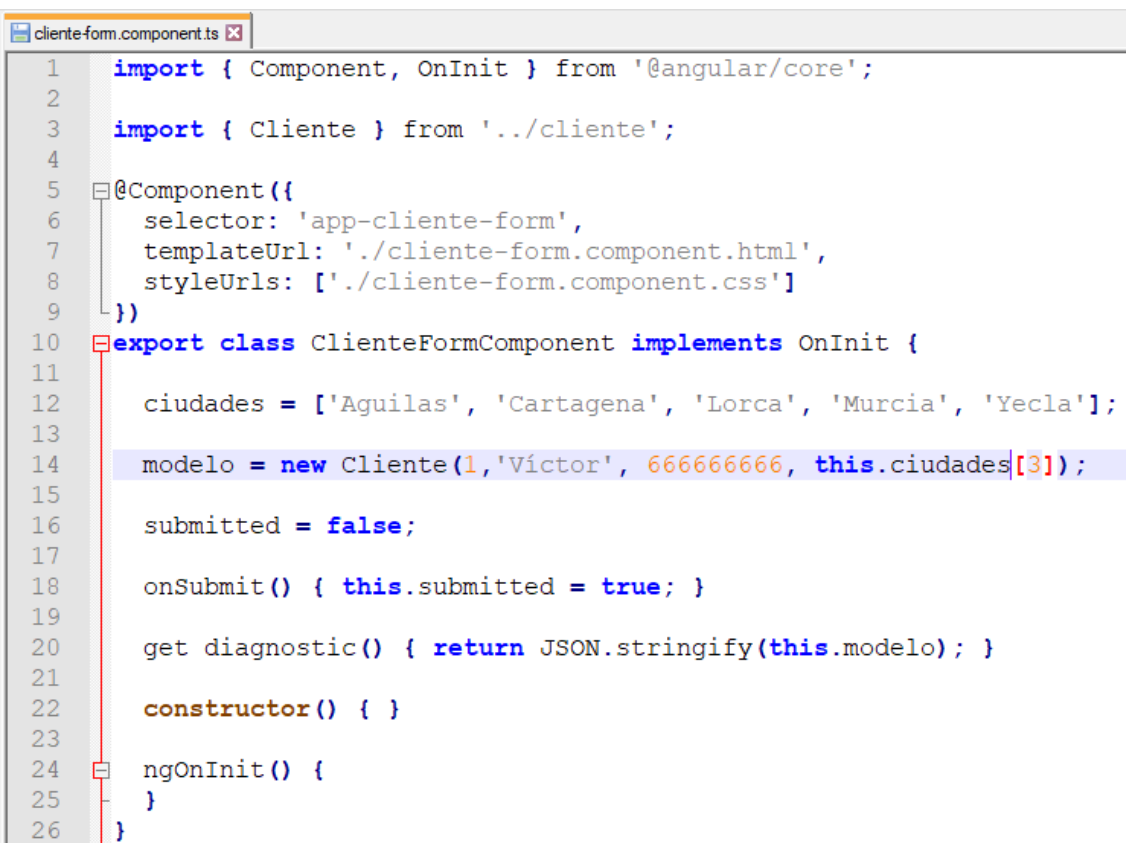
```
import { FormsModule } from '@angular/forms';
```

También modificaremos nuestro componente raíz para incluir la referencia a nuestro componente formulario



```
1 <!--The content below is only a placeholder ar
2 <h1>Mi primer formulario</h1>
3
4 <app-cliente-form></app-cliente-form>
5
```

Ahora implementamos la clase de nuestro componente



```
1 import { Component, OnInit } from '@angular/core';
2
3 import { Cliente } from '../cliente';
4
5 @Component({
6   selector: 'app-cliente-form',
7   templateUrl: './cliente-form.component.html',
8   styleUrls: ['./cliente-form.component.css']
9 })
10 export class ClienteFormComponent implements OnInit {
11
12   ciudades = ['Aguilas', 'Cartagena', 'Lorca', 'Murcia', 'Yecla'];
13
14   modelo = new Cliente(1, 'Víctor', 666666666, this.ciudades[3]);
15
16   submitted = false;
17
18   onSubmit() { this.submitted = true; }
19
20   get diagnostic() { return JSON.stringify(this.modelo); }
21
22   constructor() { }
23
24   ngOnInit() {
25   }
26 }
```

- **Propiedad “ciudades”**. Listaremos las posibles ciudades del cliente.
- **Propiedad “modelo”**. La instancia del cliente que editaremos.
- **Propiedad “submitted”**. Lo usará la plantilla para saber si el formulario ha sido enviado ya.
- **Funcion submit()**. La llamaremos cuando queramos procesar el formulario.

## UT5. Angular

- **Función `diagnostic()`.** La usaremos a modo de debug para ver los cambios en la instancia del cliente.

### Creación de la vista del formulario

Añadimos a nuestro fichero `cliente-form.component.html` el siguiente código

```
<div class="container">

  <form>

    <fieldset>

      <legend>Alta cliente</legend>

      <div class="form-group">

        <label for="name">Nombre</label>

        <input type="text" class="form-control" id="name" required>

      </div>

      <div class="form-group">

        <label for="telefono">Telefono</label>

        <input type="number" class="form-control" id="telefono">

      </div>

      <div class="form-group">

        <label for="ciudad">Ciudad</label>

        <select class="form-control" id="ciudad" required>

          <option *ngFor="let ciudad of ciudades"
[value]="ciudad">{{ciudad}}</option>

        </select>

      </div>

      <button type="submit" class="btn btn-success">Submit</button>

    </fieldset>

  </form>

</div>
```

Aunque aparentemente no hay nada de angular en este código, al haber incluido el `FormsModule` en el `AppModule`, al ver la etiqueta `<form>` angular ya sabe que vamos a utilizar el `FormsModule`.

## UT5. Angular

Las clases que hemos añadido a los elementos son del framework Bootstrap, así que vamos a añadirlo en nuestro archivo global de estilos para estilizar nuestro formulario, para ellos añadimos al fichero `styles.css` el siguiente código:

```
@import
url('https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bo
otstrap.min.css');
```

### Relacionar la vista y el modelo

Una vez hecha la vista, vamos a relacionarla con el modelo. Para hacer esto vamos a vincularle la propiedad “modelo” de nuestro componente para que se vea afectado por los cambios que realicemos. Esto lo haremos con la directiva `[(ngModel)]`

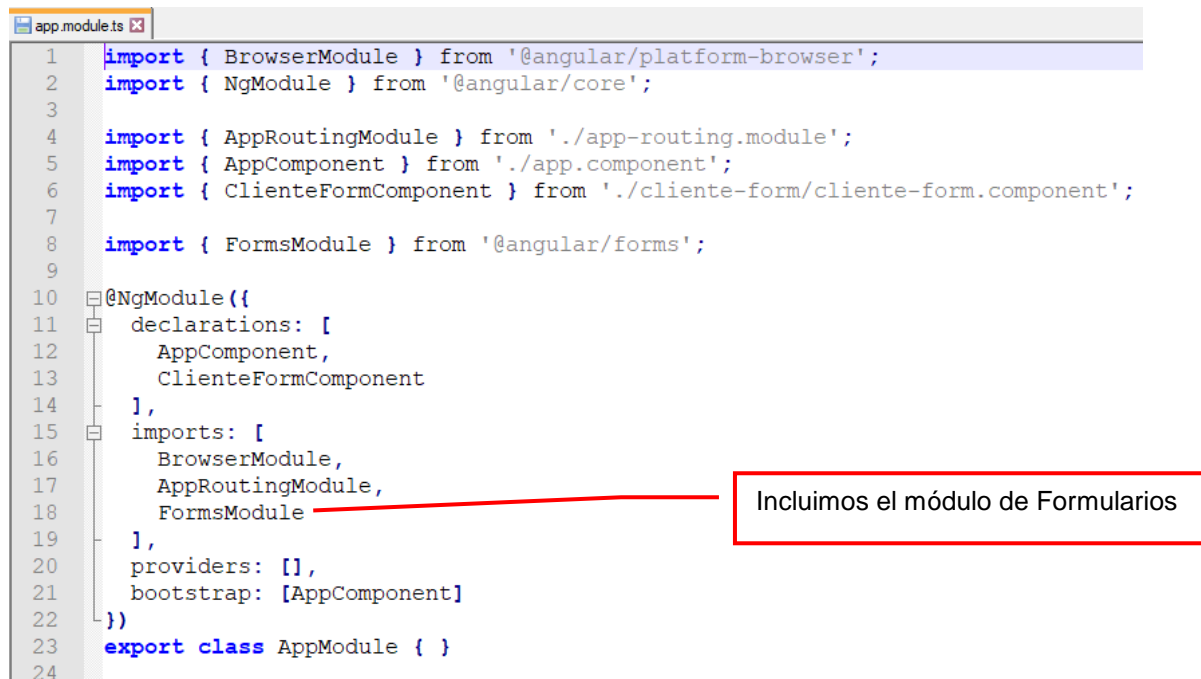
The screenshot shows the `cliente-form.component.html` file with the following HTML structure and annotations:

- Identificamos el formulario #clienteForm**: Points to the `<form #clienteForm="ngForm">` tag on line 2.
- Propiedad diagnostic del modelo**: Points to the `{{diagnostic}}` interpolation on line 3.
- Relación modelo - vista**: Points to the `[(ngModel)]="modelo.nombre" name="nombre"` binding on line 9.
- Relación modelo - vista**: Points to the `[(ngModel)]="modelo.telefono" name="telefono"` binding on line 14.
- Relación modelo - vista**: Points to the `[(ngModel)]="modelo.ciudad" name="ciudad"` binding on line 20.
- Relación modelo - vista**: Points to the `[(ngModel)]="modelo.ciudad" name="ciudad"` binding on line 20.

The HTML code is as follows:

```
1 <div class="container">
2   <form #clienteForm="ngForm">
3     {{diagnostic}}
4     <fieldset>
5       <legend>Alta cliente</legend>
6       <div class="form-group">
7         <label for="name">Nombre</label>
8         <input type="text" class="form-control" id="name" required
9           [(ngModel)]="modelo.nombre" name="nombre">
10      </div>
11      <div class="form-group">
12        <label for="telefono">Telefono</label>
13        <input type="number" class="form-control" id="telefono"
14          [(ngModel)]="modelo.telefono" name="telefono">
15      </div>
16      <div class="form-group">
17        <label for="ciudad">Ciudad</label>
18        <select class="form-control" id="ciudad" required
19          [(ngModel)]="modelo.ciudad" name="ciudad">
20          <option *ngFor="let ciudad of ciudades" [value]="ciudad">{{ciudad}}</option>
21        </select>
22      </div>
23      <button type="submit" class="btn btn-success">Submit</button>
24    </fieldset>
25  </form>
26 </div>
```

## UT5. Angular



```

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 { ClienteFormComponent } from './cliente-form/cliente-form.component';
7
8 import { FormsModule } from '@angular/forms';
9
10 @NgModule({
11   declarations: [
12     AppComponent,
13     ClienteFormComponent
14   ],
15   imports: [
16     BrowserModule,
17     AppRoutingModule,
18     FormsModule
19   ],
20   providers: [],
21   bootstrap: [AppComponent]
22 })
23 export class AppModule { }
24

```

Incluimos el módulo de Formularios

Con la propiedad “diagnostic” declarada en el componente vamos a ver como cambián los datos del modelo conforme vamos modificando la vista.

### Detección de errores en la entrada de datos

Con la **directiva ngModel**, además de conseguir relacionar el modelo con la vista (data binding bidireccional), estamos haciendo que se realice de manera automática una comprobación de cada uno de los campos relacionados.

Para detectar el estado se hace uso de diferentes clases que se van asignando a los campos en función de su estado. Cada campo relacionado puede tener los siguientes estados.

Estado	Clase si “true”	Clase si “false”
El input ha sido “visitado”	ng-touched	ng-untouched
El valor del input ha cambiado	ng-dirty	ng-pristine
El valor del input es valido	ng-valid	ng-invalid

Vamos a añadir a nuestra vista que muestre la clase asignada a uno de los campos conforme se va modificando su estado.



## UT5. Angular

```

cliente-form.component.html
1 <div class="container">
2   <form #ciudadForm="ngForm">
3     {{diagnostic}}
4     <fieldset>
5       <legend>Alta cliente</legend>
6       <div class="form-group">
7         <label for="name">Nombre</label>
8         <input type="text" class="form-control" id="name" required
9           [(ngModel)]="modelo.nombre" name="nombre" #campoNombre>
10      </div>
11      <div class="form-group">
12        <label for="telefono">Telefono</label>
13        <input type="number" class="form-control" id="telefono"
14          [(ngModel)]="modelo.telefono" name="telefono">
15      </div>
16      <div class="form-group">
17        <label for="ciudad">Ciudad</label>
18        <select class="form-control" id="ciudad" required
19          [(ngModel)]="modelo.ciudad" name="ciudad">
20          <option *ngFor="let ciudad of ciudades" [value]="ciudad">{{ciudad}}</option>
21        </select>
22      </div>
23      Clases del campo "nombre": {{campoNombre.className}}
24      <button type="submit" class="btn btn-success">Submit</button>
25    </fieldset>
26  </form>
27 </div>

```

Etiquetamos el campo

Imprimimos las clases del campo

Si añadimos a nuestro fichero de estilos generales (styles.css) el siguiente código conseguiremos mejorar el aspecto visual de nuestro formulario cuando se produce algún error

```

.form-control.ng-valid[required], .form-control.ng-valid.required {
border-left: 5px solid #42A948; /* green */}

.form-control.ng-invalid:not(form) {
border-left: 5px solid #a94442; /* red*/}

```

Si además de los tics visuales queremos mostrar mensajes de error haremos lo siguiente

```

cliente-form.component.html
1 <div class="container">
2   <form #ciudadForm="ngForm">
3     {{diagnostic}}
4     <fieldset>
5       <legend>Alta cliente</legend>
6       <div class="form-group">
7         <label for="name">Nombre</label>
8         <input type="text" class="form-control" id="name" required
9           [(ngModel)]="modelo.nombre" name="nombre" #nombre="ngModel">
10        <div [hidden]="nombre.valid || nombre.pristine" class="alert alert-danger">
11          El nombre es obligatorio
12        </div>
13      </div>
14      <div class="form-group">
15        <label for="telefono">Telefono</label>
16        <input type="number" class="form-control" id="telefono"
17          [(ngModel)]="modelo.telefono" name="telefono">
18      </div>
19      <div class="form-group">
20        <label for="ciudad">Ciudad</label>
21        <select class="form-control" id="ciudad" required
22          [(ngModel)]="modelo.ciudad" name="ciudad" #ciudad="ngModel">
23          <option *ngFor="let ciudad of ciudades" [value]="ciudad">{{ciudad}}</option>
24        </select>
25        <div [hidden]="ciudad.valid || ciudad.pristine" class="alert alert-danger">
26          La ciudad es obligatoria
27        </div>
28      </div>
29      <button type="submit" class="btn btn-success">Submit</button>
30    </fieldset>
31  </form>
32 </div>

```

Referenciamos al ngModel en los inputs implicados

Añadimos un div oculto que se mostrará en caso de error

## UT5. Angular

Con la **directiva [hidden]** estamos indicando que el div va a estar oculto si se cumplen las condiciones que le pasemos. En este caso el error saldrá siempre a menos que el campo sea válido o no haya sido tocado nunca.

La condición de que no haya sido tocado nunca es para que al usar el formulario por primera vez en blanco, no nos salga ya el error de que es requerido.

### Procesamiento del formulario

Una vez que tenemos implementada la validación de los campos del formulario, vamos a procesarlo, es decir, enviar los datos a la base de datos para que se almacenen, para ellos capturaremos el **evento ngSubmit del formulario** que llamará a la función onSubmit creada anteriormente. Además vamos a modificar el botón de submit para que solo esté activo cuando el formulario esté validado.

```

1 <div class="container">
2   <form #clienteForm="ngForm" (ngSubmit)="onSubmit()">
3     {{diagnostic}}
4     <fieldset>
5       <legend>Alta cliente</legend>
6       <div class="form-group">
7         <label for="name">Nombre</label>
8         <input type="text" class="form-control" id="name" required
9           [(ngModel)]="modelo.nombre" name="nombre" #nombre="ngModel">
10        <div [hidden]="nombre.valid || nombre.pristine" class="alert alert-danger">
11          El nombre es obligatorio
12        </div>
13      </div>
14      <div class="form-group">
15        <label for="telefono">Telefono</label>
16        <input type="number" class="form-control" id="telefono"
17          [(ngModel)]="modelo.telefono" name="telefono">
18      </div>
19      <div class="form-group">
20        <label for="ciudad">Ciudad</label>
21        <select class="form-control" id="ciudad" required
22          [(ngModel)]="modelo.ciudad" name="ciudad" #ciudad="ngModel">
23          <option *ngFor="let ciudad of ciudades" [value]="ciudad">{{ciudad}}</option>
24        </select>
25        <div [hidden]="ciudad.valid || ciudad.pristine" class="alert alert-danger">
26          La ciudad es obligatoria
27        </div>
28      </div>
29    </form>
30    <button type="submit" class="btn btn-success" [disabled]="!clienteForm.form.valid">Submit</button>
31  </fieldset>
32 </div>
33

```

Capturamos el evento ngSubmit

Habilitamos el botón Submit cuando el formulario esté validado.

Vamos a hacer ahora que cuando pulsemos el "Submit" el formulario se esconda y aparezca un mensaje de confirmación del envío. Para hacer esto envolvemos nuestro formulario con un div que mostraremos u ocultaremos en

## UT5. Angular

función de si el formulario ha sido enviado (submitted) o no enviado (!submitted). Crearemos otro div que funcionará a la inversa

```

1 <div class="container">
2
3 <div [hidden]="submitted">
4
5 <form #clienteForm="ngForm" (ngSubmit)="onSubmit()">
6   {{diagnostic}}
7   <fieldset>
8     <legend>Alta cliente</legend>
9     <div class="form-group">
10      <label for="name">Nombre</label>
11      <input type="text" class="form-control" id="name" required
12      [(ngModel)]="modelo.nombre" name="nombre" #nombre="ngModel">
13      <div [hidden]="nombre.valid || nombre.pristine" class="alert alert-danger">
14        El nombre es obligatorio
15      </div>
16    </div>
17    <div class="form-group">
18      <label for="telefono">Telefono</label>
19      <input type="number" class="form-control" id="telefono"
20      [(ngModel)]="modelo.telefono" name="telefono">
21    </div>
22
23    <div class="form-group">
24      <label for="ciudad">Ciudad</label>
25      <select class="form-control" id="ciudad" required
26      [(ngModel)]="modelo.ciudad" name="ciudad" #ciudad="ngModel">
27        <option *ngFor="let ciudad of ciudades" [value]="ciudad">{{ciudad}}</option>
28      </select>
29      <div [hidden]="ciudad.valid || ciudad.pristine" class="alert alert-danger">
30        La ciudad es obligatoria
31      </div>
32    </div>
33    <button type="submit" class="btn btn-success" [disabled]="!clienteForm.form.valid">Submit</button>
34  </fieldset>
35 </form>
36 </div>
37
38 <div [hidden]="!submitted">
39   <p> Cliente {{modelo.nombre}} dado de alta</p>
40 </div>
41 </div>

```

Ocultamos el formulario cuando la propiedad submitted del componente sea true. Esta propiedad la actualizamos en la función onSubmit del componente.

Se mostrará al enviar el formulario, es decir, propiedad submitted del componente a true.