



# Angular. Nivel I

Noviembre, 2018



## Objetivos del nivel

- Aprender a instalar Angular y crear aplicaciones
- Manipular vistas HTML
- Aprender a crear componentes
- Consumir servicios REST

## Prerrequisitos del nivel

- TypeScript Nivel I
- JavaScript Nivel III

## Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestra sitio Web [www.cadif1.com](http://www.cadif1.com), escribanos a la dirección de correo [cadi@cadif1.com](mailto:cadi@cadif1.com) o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños. Copyright 2018. Todos los derechos reservados.

ACADEMIA DE SOFTWARE



## Contenido del nivel

### Capítulo 1. Conociendo Angular

- 1.1.- Qué es Angular?
- 1.2.- Creación y Ejecución de un Proyecto.
- 1.3.- Estructura de un Proyecto Angular.

### Capítulo 2. Componentes

- 2.1.- Módulos de Typescript.
- 2.2.- Componentes.
- 2.3.- Crear Componentes.

### Capítulo 3. Vistas

- 3.1.- Data Bindings.
- 3.2.- Interpolation Binding.
- 3.3.- Property Binding.

### Capítulo 4. Patrón Mvc

- 4.1.- Patrones.
- 4.2.- Patón Mvc.
- 4.3.- Mvc en Angular.

### Capítulo 5. Directivas

- 5.1.- Concepto.
- 5.2.- Ngif.
- 5.3.- Ngfor.

### Capítulo 6. Manejo de Eventos

- 6.1.- Event Binding.
- 6.2.- Paso de Parámetros.

### Capítulo 7. Formularios

- 7.1.- Introducción.

- 7.2.- Acceso a Los Datos.
- 7.3.- Two-way Binding.

## Capítulo 8. Validaciones. Parte 1

- 8.1.- Estados de Control y Validación.
- 8.2.- Validators.
- 8.3.- Ngform.

## Capítulo 9. Validaciones. Parte 2

## Capítulo 10. Módulos Angular

- 10.1.- Root Module.
- 10.2.- Partes de un Módulo.
- 10.3.- Crear un Módulo.

## Capítulo 11. Router. Parte 1

- 11.1.- Concepto.
- 11.2.- Definición de Rutas.
- 11.3.- Router-outlet y Routerlink.

## Capítulo 12. Router. Parte 2

- 12.1.- Navegación Específica.
- 12.2.- Paso Parámetros.

## Capítulo 13. Servicios

- 13.1.- Concepto.
- 13.2.- Crear Servicio.
- 13.3.- Usar un Servicio.

## Capítulo 14. HttpClient. Parte 1

- 14.1.- Concepto.
- 14.2.- Uso.

## Capítulo 1. CONOCIENDO ANGULAR

### 1.1.- Qué es Angular ?

Angular es un framework (marco de trabajo o desarrollo) para crear aplicaciones Web, específicamente, el FrontEnd, usando JavaScript como lenguaje. En las últimas versiones se usa TypeScript, que es un súper conjunto de JavaScript. Es mantenido por Google. Una de las características resaltantes de Angular es que permite al programador crear la aplicación usando el patrón MVC (modelo-vista-controlador), lo que permite que el código de la aplicación sea más fácil de mantener.

Una característica distintiva de Angular es que permite hacer aplicaciones "Single-Page" o "Una Página", es decir, aplicaciones en las cuales no se hacen los tradicionales hipervínculos para cargar otras páginas.



Es importante señalar la diferencia entre AngularJS y Angular. AngularJS fue la primera versión del framework que estaba escrita en JavaScript, pero a partir de la versión 2 se creó prácticamente un nuevo producto que cambió radicalmente la estructura de las aplicaciones. Aún se mantiene AngularJS por fines de soporte para

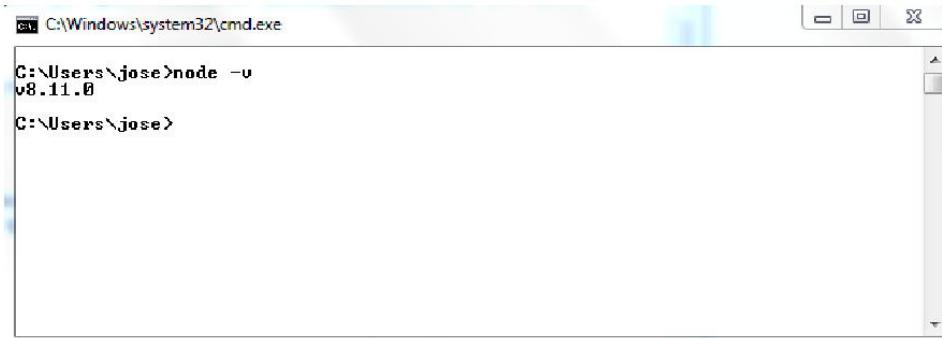
los programadores que decidieron usarlo, pero Google recomienda usar la versión más reciente. Se puede encontrar información adicional en el sitio oficial angular.io. En este curso se va a trabajar con la versión más reciente de Angular.



La instalación de Angular se realiza por medio del Administrador de Paquetes de Node (NPM), por lo tanto, es un prerequisito para instalar Angular tener instalado Node.js que instala automáticamente el NPM.

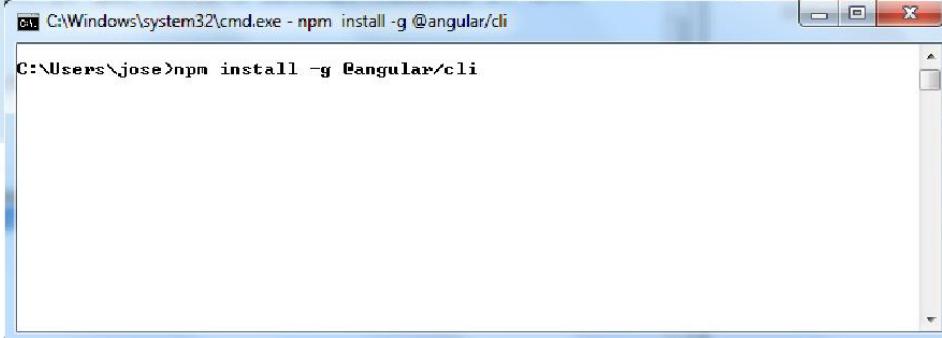


Para verificar si está instalado correctamente Node.js se ejecuta el comando "node -v".



```
C:\Windows\system32\cmd.exe
C:\Users\jose>node -v
v8.11.0
C:\Users\jose>
```

Luego de tener instalado Node, se ejecuta en la ventana de comandos la instrucción: npm install -g @angular/cli.



```
C:\Windows\system32\cmd.exe - npm install -g @angular/cli
C:\Users\jose>npm install -g @angular/cli
```

## 1.2.- Creación y Ejecución de un Proyecto

Una de las características singulares de Angular es que posee una herramienta llamada "Angular CLI", que es una interface de línea de comandos que permite, entre otras cosas: crear un proyecto, ejecutarlo, agregar archivos, probar y desplegar la aplicación para distribuirla. La primera actividad a realizar con Angular CLI es crear un proyecto. Para hacerlo, en la ventana de comandos hay que ubicarse en la carpeta donde se desea crear el proyecto y luego ejecutar el comando:

```
ng init
```

o  
ng new "aplicacion"

Donde "aplicación" es el nombre de la aplicación que se creará. La diferencia entre "ng init" y "ng new" es que el primero crea la aplicación en el directorio actual y el segundo crea un directorio con el nombre de la aplicación y luego crea la aplicación en ese nuevo directorio.

El siguiente ejemplo muestra como usar el comando para crear un proyecto con el nombre "ejemplo":

```
E:\Dropbox\cadif\curso angular>ng new ejemplo
CREATE ejemplo/angular.json (3557 bytes)
CREATE ejemplo/package.json (1311 bytes)
CREATE ejemplo/README.md (1024 bytes)
CREATE ejemplo/tsconfig.json (384 bytes)
CREATE exemplo/tslint.json (2805 bytes)
CREATE exemplo/.editorconfig (245 bytes)
CREATE exemplo/.gitignore (583 bytes)
CREATE ejemplo/src/environments/environment.prod.ts (51 bytes)
CREATE ejemplo/src/environments/environment.ts (631 bytes)
CREATE exemplo/src/favicon.ico (5430 bytes)
CREATE exemplo/src/index.html (294 bytes)
CREATE exemplo/src/main.ts (370 bytes)
CREATE exemplo/src/polyfills.ts (3194 bytes)
CREATE exemplo/src/test.ts (642 bytes)
CREATE exemplo/src/assets/.gitkeep (0 bytes)
CREATE exemplo/src/styles.css (80 bytes)
CREATE exemplo/src/browserslist (375 bytes)
CREATE exemplo/src/karma.conf.js (964 bytes)
CREATE exemplo/src/tsconfig.app.json (194 bytes)
CREATE exemplo/src/tsconfig.spec.json (282 bytes)
CREATE exemplo/src/tslint.json (314 bytes)
CREATE exemplo/src/app/app.module.ts (314 bytes)
CREATE exemplo/src/app/app.component.html (1141 bytes)
CREATE exemplo/src/app/app.component.spec.ts (998 bytes)
CREATE exemplo/src/app/app.component.ts (207 bytes)
CREATE exemplo/src/app/app.component.css (0 bytes)
CREATE exemplo/e2e/protractor.conf.js (752 bytes)
CREATE exemplo/e2e/src/app.e2e-spec.ts (303 bytes)
CREATE exemplo/e2e/src/app.po.ts (208 bytes)
CREATE exemplo/e2e/tsconfig.e2e.json (213 bytes)
```

Angular CLI realizará las siguientes acciones:

- creará una carpeta con el nombre del proyecto.
- creará los archivos mínimos para ejecutar un proyecto.
- instalará todos los paquetes de dependencias npm necesarios para el proyecto.
- configura TypeScript.
- configura "Karma" para pruebas unitarias.
- configura "Protractor" para pruebas 2e2 (end-to-end).
- configurará la carpeta como un repositorio git.

- se crean los archivos de entorno con valores por defecto.

Este comando puede tardar unos minutos mientras se descargan de Internet todos los módulos necesarios para el proyecto. Para conocer las opciones que posee Angular CLI se ejecuta el comando "ng new --help". Entre las opciones que se pueden usar al crear el proyecto esta saltar la instalación de las dependencias, que se logra usando el parámetro --skip-install. La forma de usarlo es la siguiente:

```
ng new ejemplo --skip-install
```

Esto se puede usar cuando las dependencias están instaladas globalmente o cuando se tiene una copia local de la carpeta "node\_modules".

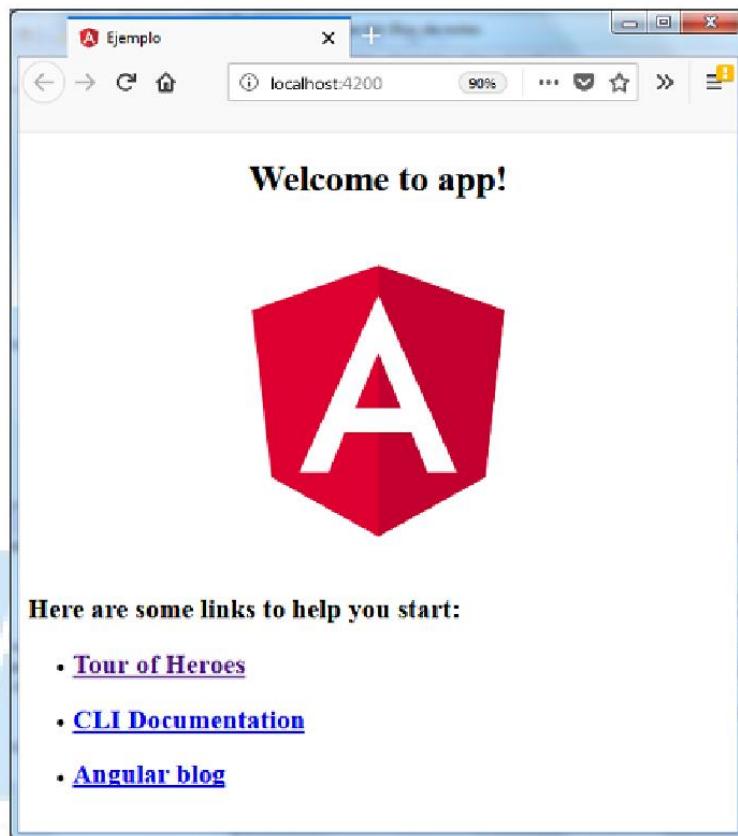
Luego de creado el proyecto, para ejecutarlo, estando en la carpeta del proyecto recién creado se ejecuta el comando:

```
ng serve
```

Este comando lanza un servidor local, mantiene los archivos del proyecto en watch para recompilar la aplicación al momento de hacer cambios. Para visualizar la aplicación se abre un navegador y se accede a la url <http://localhost:4200>. También se puede ejecutar el comando "ng serve --open" para abrir el navegador por defecto con esa url.

```
E:\Dropbox\cadis\curso angular\ejemplo>ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
10% building modules 0/1 modules 1 active .\cadis\curso angular\ejemplo\src\main
10% building modules 1/2 modules 1 active .\curso angular\ejemplo\src\polyfills
10% building modules 2/3 modules 1 active .\curso angular\ejemplo\src\styles
10% building modules 3/3 modules 0 active
```

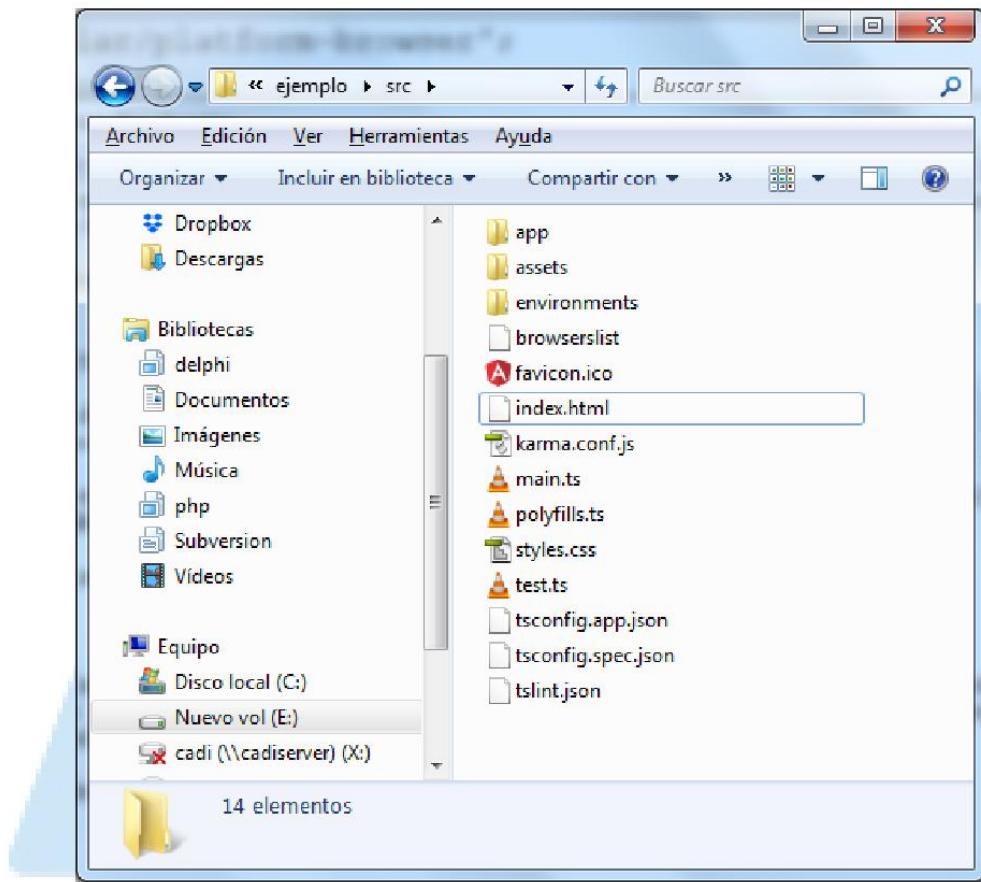
Al abrir el navegador y colocar la url <http://localhost:4200> se muestra una página como la siguiente:



### 1.3.- Estructura de un Proyecto Angular

Al crear el proyecto, Angular CLI crea en la carpeta del proyecto una carpeta con el nombre "SRC". Los archivos más relevantes de esta carpeta son los siguientes:

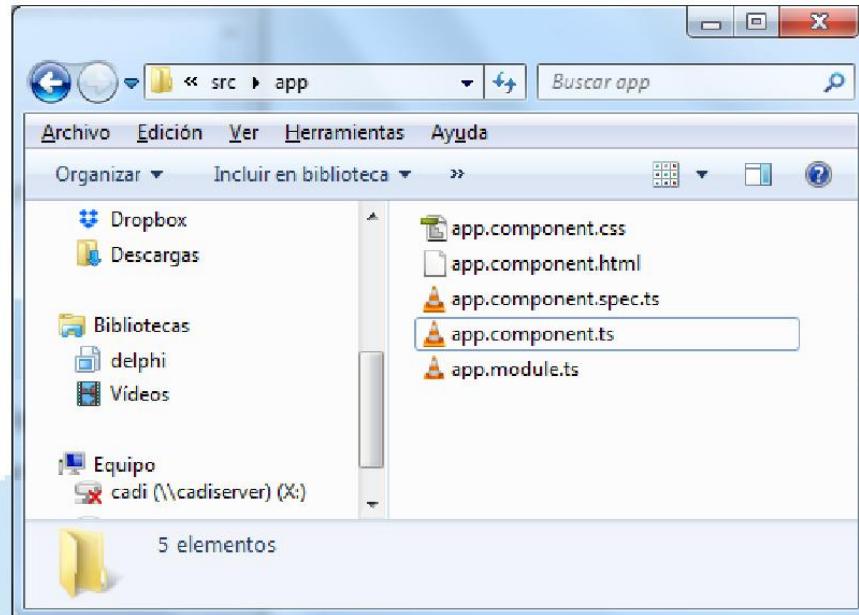
- style.css: la hoja de estilos principal del proyecto.
- main.ts: el punto de entrada del proyecto (equivalente al cuerpo principal de un programa).
- assets: carpeta para almacenar imágenes y cualquier otro archivo necesario para el proyecto.



En esta carpeta se encuentra el archivo "index.html", que es la página de inicio de la aplicación. En la página de inicio se hace referencia a una etiqueta que le indica a la aplicación donde se debe cargar.

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>Ejemplo</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
11 <body>
12     <app-root></app-root>
13
14 </html>
```

También se crea la carpeta "APP" donde se almacenan los archivos TypeScript del proyecto. Se crean varios archivos en esta carpeta, entre estos:



Los archivos de la carpeta App son:

- app.component.ts: es el componente principal de la aplicación.
- app.component.html: es la vista del componente principal.
- app.component.css: es la hoja de estilos de la vista principal.
- app.module.ts: donde se especifican los módulos de la aplicación.

El uso específico de cada uno de estos archivos se verá en capítulos posteriores.

## Capítulo 2. COMPONENTES

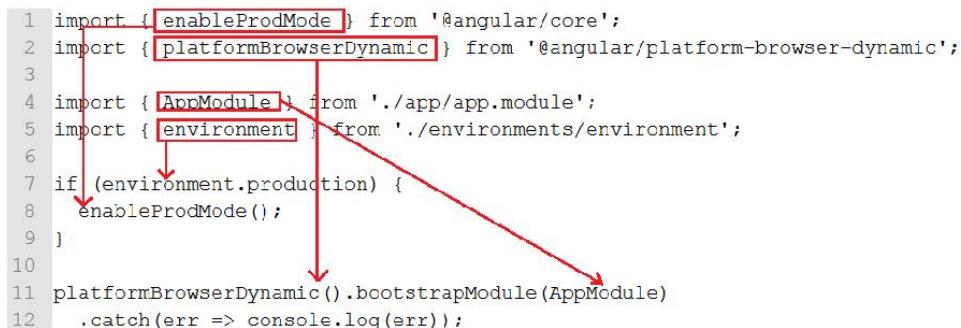
### 2.1.- Módulos de Typescript

Los módulos de TypeScript permiten crear una aplicación separando el código en archivos con contenido que puede ser utilizado en otros archivos, logrando así un diseño modular. Si hay definiciones en un archivo que serán utilizados en otros archivos, como por ejemplo, clases, funciones y/o variables, estos deben ser exportados. Si en un archivo se necesita usar algo de otro archivo, ese algo debe ser importado. En los archivos TypeScript, al inicio típicamente está la sección donde se importa lo que se necesita en el archivo, por ejemplo:

```
1 import { enableProdMode } from '@angular/core';
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5 import { environment } from './environments/environment';
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.log(err));
```

Los elementos que se importan es porque se necesitan, como se muestra en la siguiente imagen:

```
1 import {enableProdMode} from '@angular/core';
2 import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
3
4 import {AppModule} from './app/app.module';
5 import {environment} from './environments/environment';
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.log(err));
```



## 2.2.- Componentes

Los componentes son la base fundamental de la construcción en bloques de las aplicaciones en Angular. Permiten crear aplicaciones grandes de una forma más entendible y por lo tanto más mantenible, favoreciendo la reusabilidad. Un componente de Angular es una clase de TypeScript donde se definirán los datos que se mostrarán en las páginas, la respuesta a eventos, se obtienen los datos del backend, entre otros.

Un componente está compuesto típicamente por un archivo TS (TypeScript) y un archivo HTML con su respectivo CSS. Se estila colocar al nombre del archivo el sufijo ".component".

Al crear una nueva aplicación, se crea por defecto un componente con el nombre "app". En la carpeta App está el archivo "app.component.ts" y el archivo "app.component.html". El archivo TypeScript contiene la definición de una clase con el sufijo "Component", en este caso "AppComponent":

```
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 }
```

Adicionalmente en el archivo TS hay una sección previa a la definición de la clase. Es el uso de una función con el nombre `@Component`, donde se especifica información general del componente. La siguiente imagen resalta el contenido:

```
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 }
```

La información que se especifica es:

- selector: el nombre con el cual será identificado el componente en los archivos HTML.
- templateUrl: el nombre del archivo HTML que estará asociado al componente.
- styleUrls: hoja de estilos particular del componente.

### 2.3.- Crear Componentes

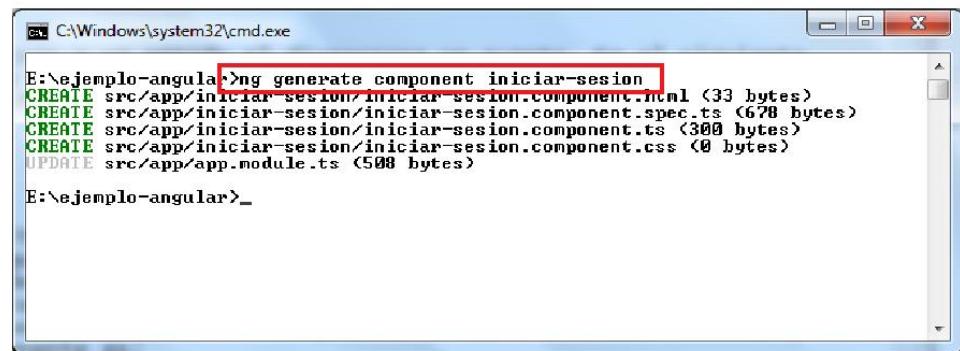
Al crear un proyecto nuevo, Angular crea por defecto un componente que lo define como el componente principal. Normalmente una aplicación tendrá más de un componente (cada funcionalidad o interfaz es un componente), y por lo tanto, será necesario agregar nuevos componentes. Se pueden crear los archivos manualmente o usar el "Angular CLI" para facilitar el trabajo. La forma de crear el componente es:

ng generate component mi-componente

o la forma abreviada:

ng g c mi-componente

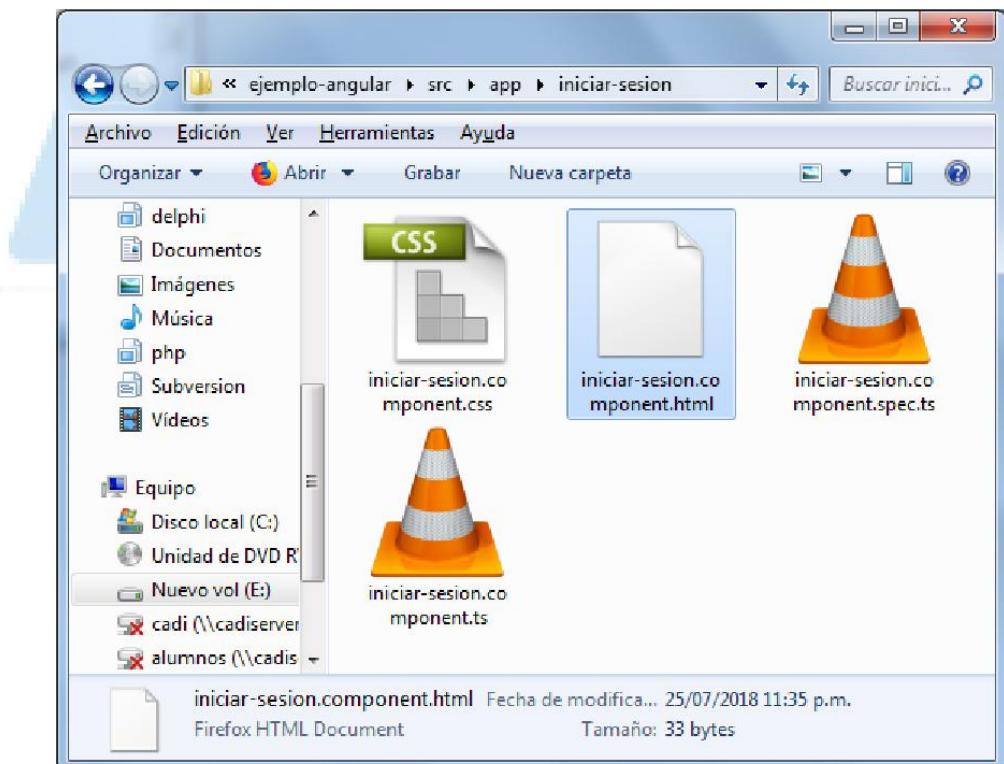
Por ejemplo:



```
C:\Windows\system32\cmd.exe
E:\ejemplo-angular>ng generate component iniciar-sesion
CREATE src/app/iniciar-sesion/iniciar-sesion.component.html (33 bytes)
CREATE src/app/iniciar-sesion/iniciar-sesion.component.spec.ts (678 bytes)
CREATE src/app/iniciar-sesion/iniciar-sesion.component.ts (300 bytes)
CREATE src/app/iniciar-sesion/iniciar-sesion.component.css (<0 bytes>
UPDATE src/app/app.module.ts (508 bytes)

E:\ejemplo-angular>_
```

Al ejecutar el comando, se crea en la carpeta "src/app" una carpeta con el nombre del componente y en su interior los archivos html, css y TypeScript. Además se actualiza el módulo principal de la aplicación:

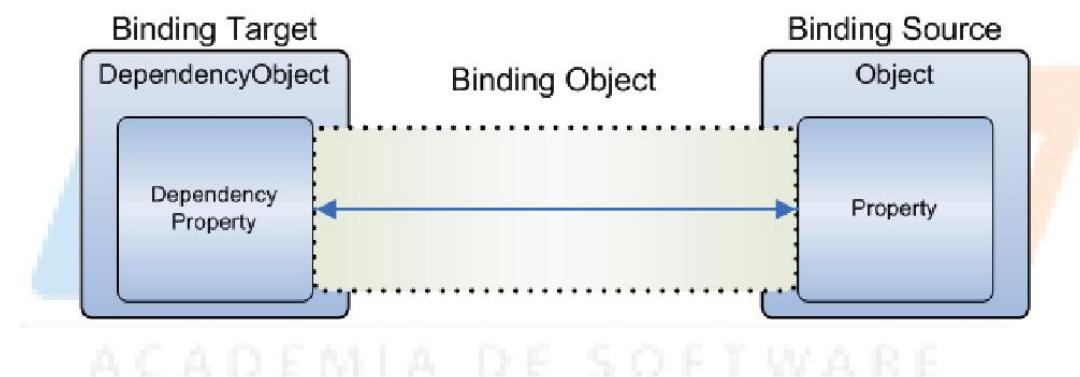


## Capítulo 3. VISTAS

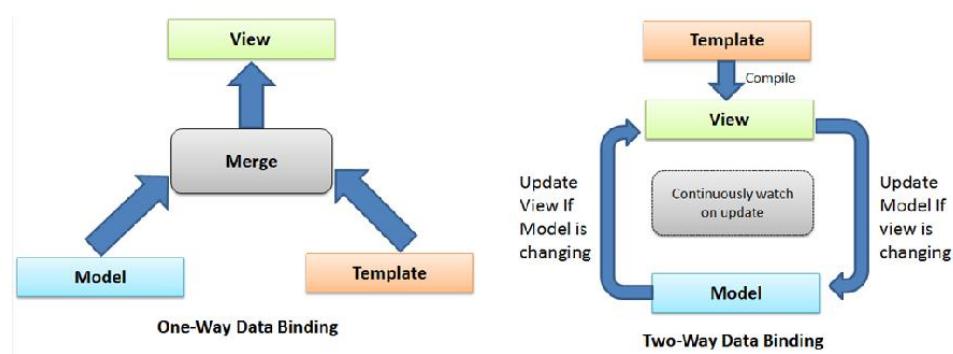
### 3.1.- Data Bindings

Como se explicó anteriormente, Angular utiliza el patrón MVC, por lo tanto, debe haber una separación entre el modelo (los datos) y las vistas (el HTML). El controlador (la clase TypeScript que identifica el componente), debe contener entre sus atributos los valores que van a ser usados en las vistas. En las vistas debe existir una forma de hacer referencia a los elementos del controlador.

Angular maneja un concepto llamado "DataBinding" (unión de datos), que es el mecanismo usado para pasar datos desde el controlador hacia las vistas y viceversa.



En general, existen 2 tipos de binding: binding de una vía (one way binding) y binding de doble vía (two way binding). La siguiente imagen muestra el esquema de funcionamiento de ambos tipos de bindings:

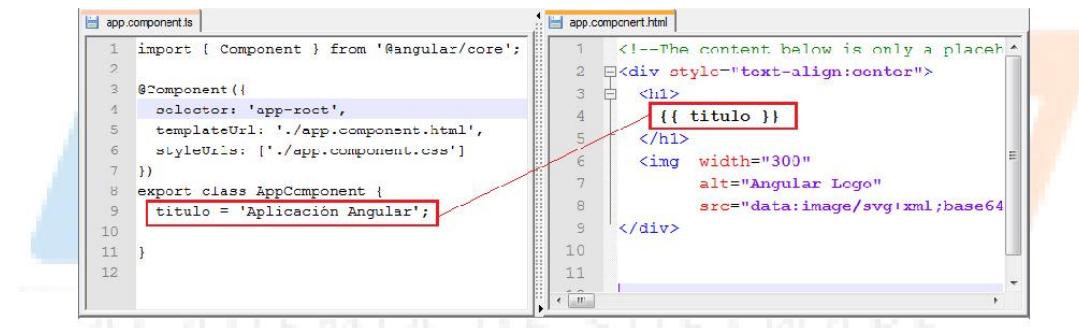


Existen 4 tipos de data binding que se pueden usar en Angular:

- Interpolation binding: para acceder a valores de variables.
- Property binding: para acceder a valores de variables.
- Event binding: para manejar eventos (se explicará en capítulos posteriores).
- Two-way binding: para manejar formularios (se explicará en capítulos posteriores).

### 3.2.- Interpolation Binding

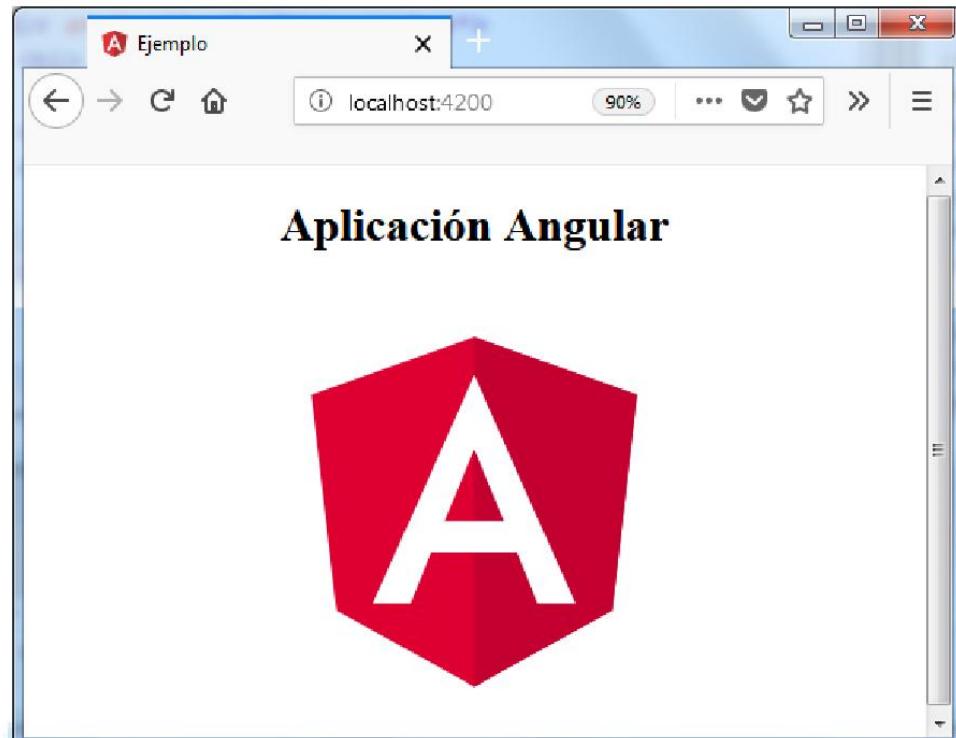
Este tipo de binding permite hacer referencia en el archivo HTML a un atributo definido en la clase TypeScript usando dos veces llaves alrededor del nombre de un atributo. En forma general: {{ atributo }}. Por ejemplo:



```
app.component.ts
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   titulo = 'Aplicación Angular';
10 }
11
12

app.component.html
1 <!--The content below is only a placeholder-->
2 <div style="text-align:center">
3   <h1>
4     {{ titulo }}
5   </h1>
6   
10
11
12
```

El resultado de la ejecución del ejemplo anterior se muestra en la siguiente imagen. Cabe destacar, que este binding es de una sola vía, es decir, el valor que toma un atributo en la clase es visible en el HTML, pero si el valor es modificado de alguna forma en la vista, en la clase no se conocerá el nuevo valor.



### 3.3.- Property Binding

Es un binding muy similar al anterior, con una diferencia de sintaxis. Este tipo de binding se utiliza para asignarle valor a una propiedad de un elemento de la vista, por ejemplo, la propiedad "src" de un "img". La sintaxis general es la siguiente:

```
[propiedad] = "atributo-clase"
```

Donde "propiedad" es la propiedad de un elemento HTML de la vista y "atributo-clase" es el atributo definido en la clase del componente, que contiene el valor que se desea asignar a la propiedad. Por ejemplo:

```
<img [src]="rutaimagen" />
```

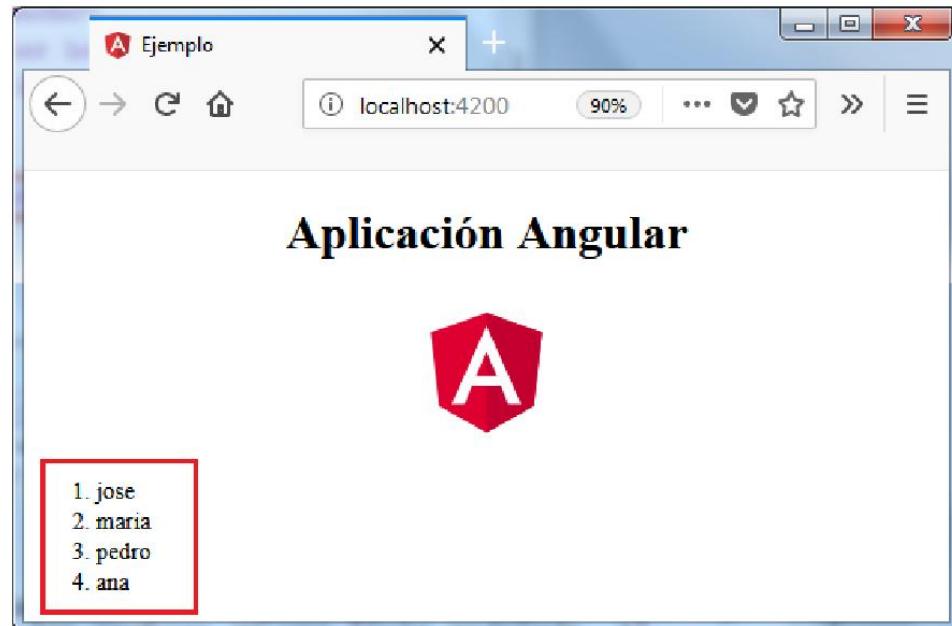
La siguiente imagen muestra el ejemplo en el código de una vista:

The image shows two code editor panes side-by-side. The left pane, titled 'app.component.ts', contains the following TypeScript code: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
10 titulo = 'Aplicación Angular';
11 ancho = 100;
12 }

The line 'ancho = 100;' is highlighted with a red box.

The right pane, titled 'app.component.html', contains the following HTML code:<! The content below is only a placeholder and will be replaced by the content of your route >
<div style="text-align:center">
 <h1>
 {{ titulo }}
 </h1>
 
2 <div style="text-align:center">
3   <h1>
4     {{ titulo }}
5   </h1>
6   
3   <h1>
4     {{ titulo }}
5   </h1>
6   ` element, an `<img>` element with a width of `ancho` and alt text of `"Angular Logo"`, and a `<ul>` element. The `<ul>` element uses `*ngFor` to iterate over the `nombrase` array, creating a `<li>` element for each name `n`.

En el ejemplo anterior, se definió en la clase del componente el arreglo de strings "nombres". En el HTML, se desea mostrar los nombres en una lista ordenada. Al elemento "<li>" (que será el que se repetirá) tiene la directiva ngFor, en la cual se define una variable "n" que tomará el valor de cada uno de los elementos del arreglo "nombres".



La directiva `ngFor` se puede aplicar a cualquier elemento HTML. En el siguiente ejemplo una etiqueta "div" se repetirá varias veces:

```
1  <!--The content below is only a placeholder and can be replaced-->
2  <div *ngFor="let n of nombres" style="text-align:center">
3      <h1>
4          {{ n }}
5      </h1>
6      
```

## **Capítulo 6. MANEJO DE EVENTOS**

## 6.1.- Event Binding

Como se explicó anteriormente, Angular implementa el patrón MVC y es el componente asociado un HTML (o plantilla) el encargado de manejar los eventos que se disparen en ésta. Al programar la respuesta a los eventos en el componente se está implementando la otra forma de hacer binding: "event binding". La forma general de programar la respuesta a un evento es la siguiente:

(evento)="instruccion"

Donde "evento" es el evento que se desea programar (por ejemplo: click, blur, keypress) e "instrucción" lo que se va a ejecutar al producirse el evento. Se debe evitar usar instrucciones complejas, por lo general, con sólo asignar el valor a una propiedad o hacer el llamado a un método debería ser suficiente.

El siguiente ejemplo ejecuta el método "agregar()" de la clase "AppComponent" cuando el usuario hace click sobre un elemento "a". El método solicita al usuario un valor y lo agrega a un arreglo:

The screenshot shows the Angular IDE interface with two tabs open:

- app.component.ts**: The code defines a class `AppComponent` with properties `titulo`, `ancho`, and `numeros`. It includes methods `agregar()` and `eliminar(i)`.
- app.component.html**: The template contains a centered title, a logo image, and a list of names with a delete link for each item.

A red box highlights the `agregar()` method in the TypeScript file, and another red box highlights the `(click)="agregar()"` event binding in the HTML template.

## 6.2.- Paso de Parámetros

Se pueden pasar parámetros a un método al dispararse un evento. En el siguiente ejemplo, al hacer click sobre alguno de los items se pasa por parámetro al método el nombre del elemento mostrado:

```
app.component.ts
15     "pedro",
16     "ana" ]
17
18     agregar(){
19         let valor = prompt("Nuevo valor:");
20         if (valor!=null)
21             if (this.nombres.indexOf(valor)==-1)
22                 alert("Ya esta registrado");
23             else
24                 this.nombres.push(valor);
25     }
26
27     eliminar(i){
28         if (confirm("Desea eliminar a "+i)){
29             let pos = this.nombres.indexOf(i);
30             this.nombres.splice(pos, 1);
31         }
32     }
33 }

app.component.html
2   <div style="text-align:center">
3     {{ titulo }}
4   </h1>
5   
13      {{ n }} <a href="#" (click)="eliminar(n)"> X </a>
14    </li>
15  </ol>
16  <a href="#" (click)="agregar()">Nuevo</a>
17
18
```

ACADEMIA DE SOFTWARE

## Capítulo 7. FORMULARIOS

### 7.1.- Introducción

Los formularios HTML permiten a la aplicación solicitar datos al usuario para interactuar con el. Angular posee varias herramientas que facilitan la implementación de actividades típicas en el uso de form, tales como: tomar los datos para procesarlos y validarlos antes.

Para trabajar con formularios, Angular necesita que estén instalados unas dependencias. Una de ellas es @angular/forms, por ello se debe instalar usando npm:

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

También debe agregarse al módulo principal.

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

### 7.2.- Acceso a Los Datos

Angular provee una forma sencilla de acceder a los valores escritos por el usuario en los inputs. Se llama: "template reference". Se logra colocando al input un atributo que comience con el carácter #.

### 7.3.- Two-way Binding

En capítulos anteriores se explicó el binding en Angular. Cuando se trabaja con formularios se necesita hacer un binding de doble via (two-way) de forma tal que se puedan conectar datos de la vista con el controlador y viceversa. Este binding se logra colocando en los inputs de los formularios la etiqueta [(ngmodel)], por ejemplo:

```
<input type="text" class="form-control" id="name"
      required
      [(ngModel)]="model.name" name="name">
      TODO: remove this: {{model.name}}
```

## Capítulo 8. VALIDACIONES. PARTE 1

### 8.1.- Estados de Control y Validación

El ngModel en los formularios ofrece más que el binding de doble vía. Este también permite determinar si el usuario ha tocado el control (touched), si el valor ha cambiado o si es válido. Cada input que tenga asignado un ngModel puede tener 6 posibles estados:

- ng-touched: ha sido visitado por el usuario
- ng-untouched: lo contrario al anterior
- ng-dirty: el valor del control ha cambiado.
- ng-pristine: lo contrario del anterior
- ng-valid: si el control tiene un valor valido
- ng-invalid: lo contrario al anterior.

La transición de los estados es:

1. ng-untouched,ng-valid y ng-pristine: al cargar la página
2. ng-touched,ng-valid y ng-pristine: al usuario hacer click en el elemento.
3. ng-touched,ng-valid y ng-dirty: si es modificado.
4. ng-touched,ng-invalid y ng-dirty: si al cambiar el valor no es válido.

Estos estados son simples clases CSS que Angular agrega o quitan al elemento según sea el caso. Para comprobar los cambios, se puede asignar un atributo al input con el carácter # por delante (por ejemplo #x), y usando este atributo como una variable y la interpolación, acceder a la propiedad "className": {{x.className}}

### 8.2.- Validators

Angular provee mecanismos de validación de formularios que facilitan este trabajo al programador. Existen 2 formas de validar formularios: con plantillas de formularios o formularios reactivos. En este capítulo se estudiarán las validaciones con plantillas. Esto se logra agregando a los inputs atributos correspondientes a la validación que se desea aplicar (como required de HTML 5). Los atributos pre definidos en Angular son:

- min: valor numérico mínimo
- max: valor numérico máximo
- required: verifica que no este vacío

- requiredTrue: se utiliza para evaluar que los checkboxes esten seleccionados
- email: sea un correo válido
- minLength: verifica que tenga una cantidad mínima de caracteres
- maxLength: verifica que tenga una cantidad máxima de caracteres
- pattern: verifica que cumpla con un patrón

Cada vez que el valor de un control cambia, Angular ejecuta las validaciones y genera una lista de errores. Para conocer el estado de un control en un momento dado, se puede asignar a una variable el valor del ngModel:

```
#name="ngModel"
```

Con la variable "name", se puede determinar:

- name.invalid: si el input es inválido
- name.dirty: si fue modificado
- name.touched: si fue tocado
- name.errors: los errores de validación que posee

Por ejemplo:

```
<label for="name">Name</label>
<input type="text" class="form-control" id="name"
       required
       [(ngModel)]="model.name" name="name"
       #name="ngModel">
<div [hidden]="name.valid || name.pristine"
      class="alert alert-danger">
    Name is required
</div>
```

### 8.3.- Ngform

Se debe colocar en la etiqueta de formulario la referencia una variable que la plantilla que tome el valor "ngForm". La directiva ngForm gobierna el comportamiento del formulario como un todo. Esta directiva permite monitorear todos los elementos del formulario que tengan un ngModel, incluyendo su validación. Por ejemplo:

```
#f="ngForm"
```

Con la variable "f" se puede determinar por ejemplo, si los datos del formulario son válidos o no, de la siguiente forma:

```
f.form.valid
```

Si es true, indica que los datos del formulario son válidos, de lo contrario, es que algún elemento del formulario no pasó la validación.

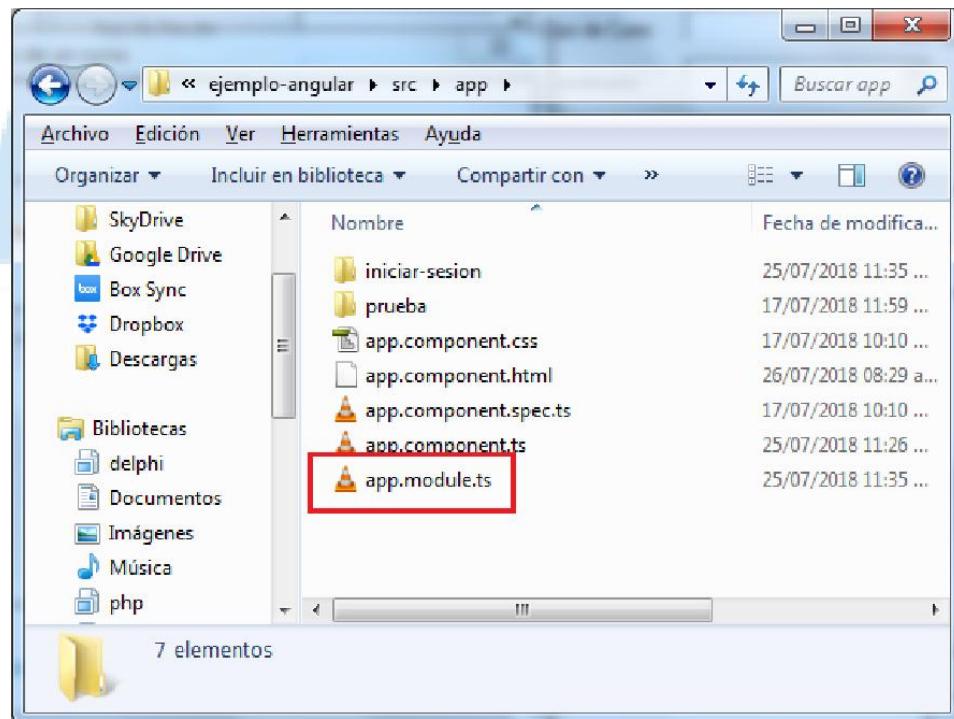


## Capítulo 9. MÓDULOS ANGULAR

### 9.1.- Root Module

Un módulo es una clase TypeScript marcada con el decorador `@NgModule`. En un módulo se identifican todos los elementos de la aplicación: librerías externas, componentes, directivas, servicios, entre otras cosas. Los módulos son una forma de organizar una aplicación y extender sus capacidades.

Toda aplicación Angular tiene al menos un módulo. Por tal razón, al crear una aplicación se crea por defecto un módulo, que se almacena en el archivo "app.module.ts" en la carpeta "app" del proyecto. A este módulo se le denomina "rootModule".



El main es un archivo TypeScript que carga el módulo principal, que a su vez hace referencia a un componente inicial (el que se ejecutará primero). Normalmente no hay que modificar este archivo. El contenido del main.ts es el siguiente:

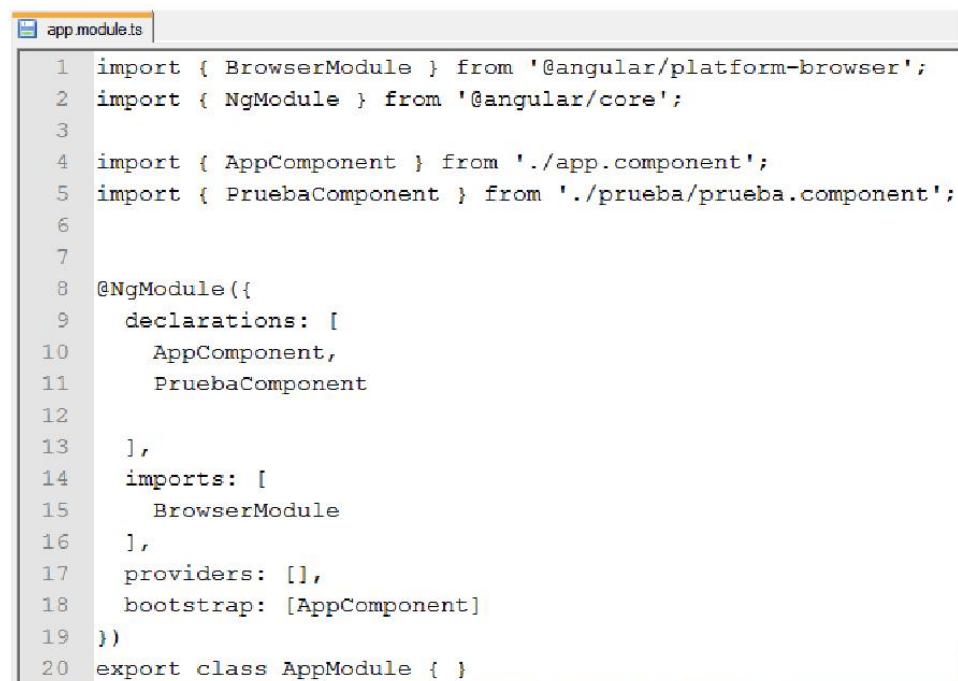
```
1 import { enableProdMode } from '@angular/core';
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5 import { environment } from './environments/environment';
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.log(err));
```

## 9.2.- Partes de un Módulo

Un módulo posee varias partes:

- declarations: esta es la lista de componentes, directivas y pipes que son parte de este módulo. Todo lo que esté en declaraciones es visible a la aplicación usando este módulo sin necesidad de requerir exportar explícitamente.
- imports: al usar la sentencia imports se hacen visibles otros módulos en el interior de este módulo.
- exports: es la lista de declaraciones que se desea sean visibles o estén disponibles para otros módulos que importen a este módulo.
- providers: aquí se listan los proveedores que se usaran para la inyección de datos (se estudiará al hablar de servicios).
- entryComponents: lista de componentes que el ComponentFactory debe generar.

La siguiente imagen muestra el contenido del módulo principal de una aplicación:



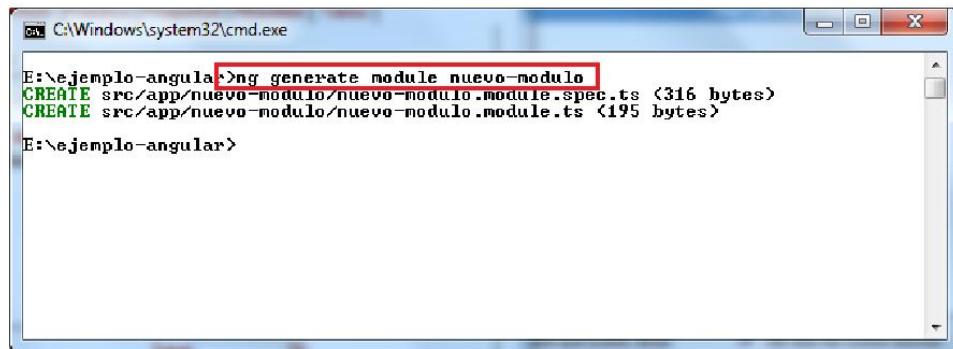
```
app.module.ts
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppComponent } from './app.component';
5 import { PruebaComponent } from './prueba/prueba.component';
6
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11 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

### 9.3.- Crear un Módulo

Aunque una aplicación por defecto ya posee un módulo es posible necesitar agregar un nuevo módulo. Para esto se usa el comando:

ng generate module nombre-modulo

Donde "nombre-module" es el nombre del módulo que se va a crear.



```
C:\Windows\system32\cmd.exe
E:\ejemplo-angular>ng generate module nuevo-modulo
CREATE src/app/nuevo-modulo/nuevo-modulo.module.spec.ts <316 bytes>
CREATE src/app/nuevo-modulo/nuevo-modulo.module.ts <195 bytes>
E:\ejemplo-angular>
```

## Capítulo 10. ROUTER. PARTE 1

### 10.1.- Concepto

Como se ha explicado desde el inicio del curso, Angular es un framework para crear aplicaciones web del tipo "Single Page Application" (aplicaciones web de una sola página). Esto tiene varias implicaciones, una de ellas es la forma de hacer vinculaciones entre páginas, es decir, la creación de hipervínculos entre las páginas del mismo sitio, porque al ser una sola página (el index), no hay necesidad de hacer vínculos a otras páginas (se puede usar la etiqueta `<a>` pero para programarle el evento click).

Pero, cómo se logra mostrar o cargar el contenido del HTML de otro componente desde el componente principal (`app.component.html`) ? La respuestas de Angular para este problema son los "routes", que combinados con unas etiquetas HTML especiales de Angular, pueden lograr el efecto que se logra en las aplicaciones multi page: cuando el usuario hace click en un hipervínculo, se carga otra página con otro contenido.

### 10.2.- Definición de Rutas

El concepto de "rutas" (route) en Angular no es más que la asociación de un texto con un componente de la aplicación, de modo que cuando en la URL aparezca ese texto, Angular sepa a cuál componente buscar para tomar el HTML asociado a éste y cargarlo.

Las rutas se definen generalmente en el módulo principal de la aplicación (aunque pudiera crearse un módulo especialmente para las rutas). Por lo tanto, lo primero que debe hacerse el módulo es importar los módulos correspondientes:

```
7  
8 import { RouterModule, Routes } from '@angular/router';  
9  
10 const appRoutes: Routes = [  
11 ];  
12  
13 @NgModule({  
14   declarations: [  
15     AppComponent,  
16     PruebaComponent,  
17     IniciarSesionComponent,  
18     PageNotFoundComponent  
19   ],  
20   imports: [  
21     BrowserModule,  
22     FormsModule,  
23     RouterModule.forRoot(  
24       appRoutes  
25     )  
26   ],  
27   providers: [],  
28   bootstrap: [IniciarSesionComponent]  
29 })
```

En el arreglo con el nombre "appRoutes" es donde se van a definir las rutas de la aplicación. Cada ruta debe tener una URL y la asociación con algún componente de la aplicación, por ejemplo:

```
9  
10 const appRoutes: Routes = [  
11   { path: 'iniciar', component: IniciarSesionComponent },  
12   { path: 'prueba', component: PruebaComponent },  
13 ];  
14
```

Ruta

Componente

### 10.3.- Router-outlet y Routerlink

Ya se tienen definidas las rutas pero ahora se debe definir cómo navegar hacia ellas. En Angular para vincular de un componente se usan las etiquetas "<a>" tradicionales, con una directiva llamada "routerLink". Por ejemplo:

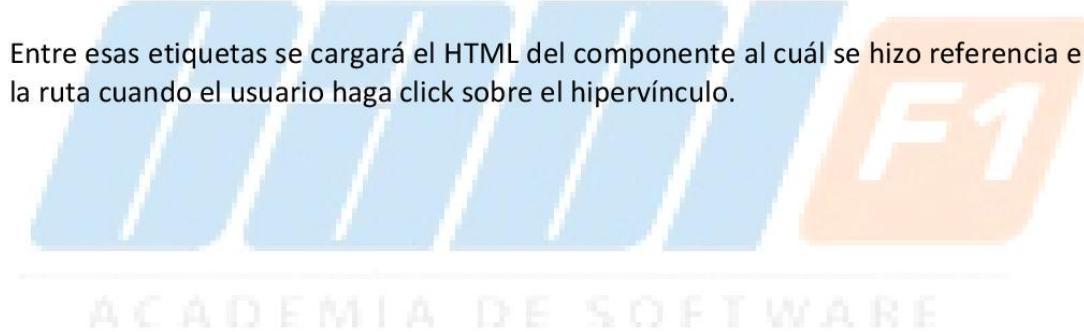
```
<a routerLink="/iniciar">Iniciar Sesión</a>
```

El texto que el usuario visualiza es "iniciar Sesión" y al hacer click sobre el hiper vínculo, se llama a la ruta identificada con el texto "sesion", que según la definición de las rutas, cargará el HTML del componente "IniciarSesionComponent"

En el componente principal, debe colocarse en algún lugar de la página las etiquetas:

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

Entre esas etiquetas se cargará el HTML del componente al cual se hizo referencia en la ruta cuando el usuario haga click sobre el hipervínculo.



## Capítulo 11. ROUTER. PARTE 2

### 11.1.- Rutas Especiales

Al momento de definir las rutas existen dos rutas especiales que pueden agregarse:

- una ruta por defecto: en el componente que se cargará por defecto cuando no se especifique ninguna ruta en la URL (ruta vacía).
- ruta desconocida: es el equivalente al 404 del servidor web (ruta wildcard \*\*).

La ruta desconocida debe ser la última ruta de la lista, de lo contrario, no funcionará como se espera. El siguiente ejemplo muestra la definición de rutas donde se agregan la ruta por defecto y la ruta 404:

```
9
10 const appRoutes: Routes = [
11   { path: 'iniciar', component: IniciarSesionComponent },
12   { path: 'prueba', component: PruebaComponent },
13   { path: '',
14     redirectTo: 'iniciar', pathMatch: 'full' },
15   { path: '**',
16     component: PageNotFoundComponent }
```

### 11.2.- Paso Parámetros

Es muy frecuente que en un sitio web exista el comportamiento "maestro-detalle", que permite mostrar un listado general de elementos donde el usuario puede hacer click sobre alguno de ellos para que se muestre otra página con los detalles del elemento seleccionado. Para lograr esto, una página le debe enviar parámetros a la otra.

Al definir las rutas, se puede definir el o los parámetros que el componente llamado recibirá. En el siguiente ejemplo, la ruta "detail" tiene el parámetro "id":

```
{ path: 'detail/:id', component: HeroDetailComponent }.
```

Al usar una ruta con parámetro, en el hipervínculo se debe agregar el valor del parámetro en el "routerlink". Por ejemplo:

#### src/app/dashboard/dashboard.component.html (hero links)

```
<a *ngFor="let hero of heroes"
    routerLink="/detail/{{hero.id}}">
    <div class="module hero">
        <h4>{{hero.name}}</h4>
    </div>
</a>
```

Y en la clase a la cual se esta haciendo el enrutamiento se debe acceder al valor del parámetro recibido inyectando al componente el servicio "ActivatedRoute", que permite hacer referencia a los datos que están en la ruta activa. Por ejemplo:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
    selector: 'app-hero-detail',
    templateUrl: './hero-detail.component.html',
    styleUrls: [ './hero-detail.component.css' ]
})
export class HeroDetailComponent implements OnInit {
    hero: Hero;

    constructor(
        private route: ActivatedRoute,
    ) {}
    ngOnInit(): void {
        this.getHero();
    }
    getHero(): void {
        const id = +this.route.snapshot.paramMap.get('id');
    }
}
```

## Capítulo 12. SERVICIOS

### 12.1.- Concepto

Como se explicó en el capítulo de MVC (Modelo-Vista-Controlador), en Angular, la capa del modelo la representan los "servicios". Un servicio de Angular es una clase de TypeScript con el decorador `@Injectable` en su definición, que contiene los datos que serán utilizados en los controladores y que serán compartidos entre estos. Los servicios también deberían ser los encargados de conectarse con el BackEnd, obtener los datos del localStorage o de cualquier otra fuente. Para crear un servicio nuevo, se utiliza el comando:

```
ng generate service "nombreServicio"
```

En la clase TypeScript que representa un servicio se deben declarar las variables que contendrán los datos que serán utilizados en los controladores, y los métodos a través de los cuales se accederá a estas variables.

Por ejemplo, el siguiente comando generará un servicio con el nombre "HeroService". En AngularCli le agrega automáticamente el sufijo "service" al nombre del archivo:

```
ng generate service hero
```

src/app/hero.service.ts (new service)

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

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor() { }

}
```

La instrucción `@Injectable` tiene el atributo "provideIn" con el valor "root". Esto significa que Angular creará y compartirá automáticamente una única instancia de esta clase, lo que permitirá usarla en cualquier clase que la requiera, es decir, tendrá un alcance global en la aplicación.

También es posible especificar que un servicio sólo estará disponible para los componentes de cierto módulo. Para lograrlo, a la etiqueta "provideIn" se le debe asignar el nombre del módulo donde será usado el servicio. También se puede especificar que un servicio puede ser usado sólo por un componente particular.

### 12.2.- Usar un Servicio

El patrón Dependency Injection (DI), en español, Inyección de Dependencias, es un patrón de diseño utilizado en Angular para aumentar la eficiencia y la modularidad de las aplicaciones. Este patrón se logra creando los servicios inyectables. Si un servicio tiene alcance de un módulo particular, debe agregarse en el módulo en la sección de "providers" los servicios que serán usados por los componentes pertenecientes en ese módulo. Por ejemplo:

```
src/app/user.service.ts
import { Injectable } from '@angular/core';
import { UserModule } from './user.module';

@Injectable({
  providedIn: UserModule,
})
export class UserService {
```

```
src/app/user.module.ts
import { NgModule } from '@angular/core';
import { UserService } from './user.service'

@NgModule({
  providers: [UserService],
})
export class UserModule {
```

En el componente donde se necesita usar el servicio, se importa la clase del servicio y pasa por parámetro al constructor. Por ejemplo:

```
import { Component } from '@angular/core';
import { Hero } from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'app-hero-list',
})
export class HeroListComponent {
  heroes: Hero[];

  constructor(heroService: HeroService) {
    this.heroes = heroService.getHeroes();
  }
}
```

## src/app/heroes/hero.service.ts

```
import { Injectable } from '@angular/core';
import { HEROES } from './mock-heroes';

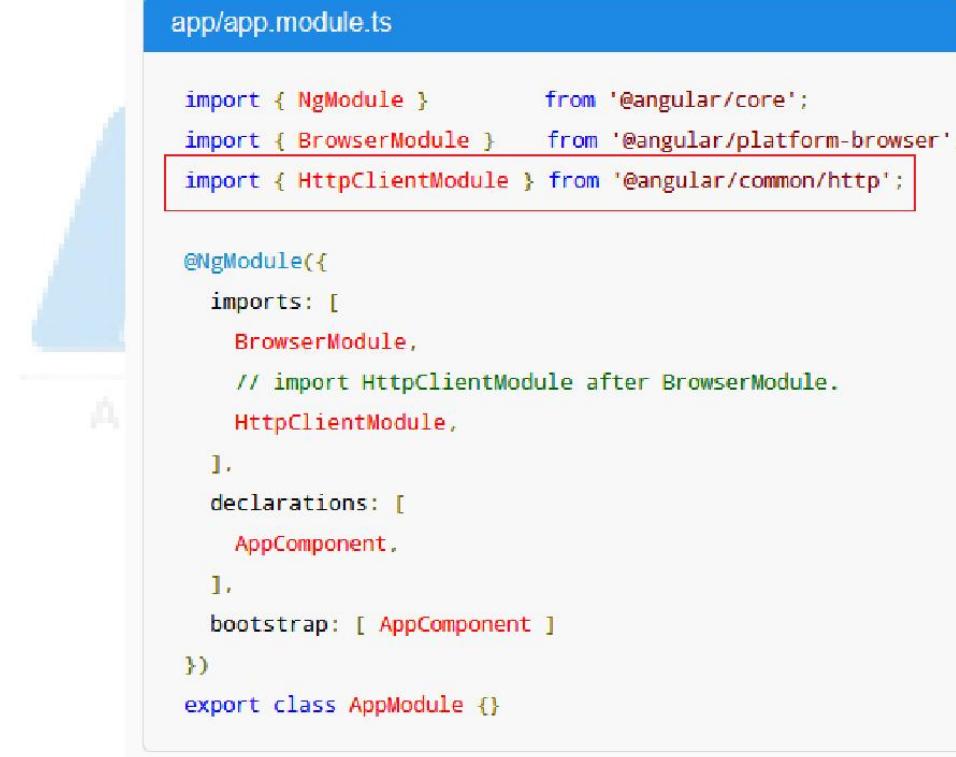
@Injectable({
  providedIn: 'root',
})
export class HeroService {
  getHeroes() { return HEROES; }
}
```



## Capítulo 13. HTTPCLIENT. PARTE 1

### 13.1.- Concepto

La mayoría de aplicaciones que necesitan comunicarse con un Backend lo hacen utilizando el protocolo HTTP. Angular posee una clase que permite utilizar el objeto XMLHttpRequest que provee el navegador. Su nombre es HttpClient, que es parte del módulo @angular/common/http. Por esa razón, lo primero que debe hacerse es importar este módulo en el módulo principal de la aplicación, para que esté disponible para cualquier componente o servicio:



```
app/app.module.ts

import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
  declarations: [
    AppComponent,
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Con la clase HttpClient se pueden ejecutar cualquiera de los métodos de envío de HTTP, entre los más usados:

- get
- post

- put
- delete

### 13.2.- Uso

Como se explicó en el capítulo anterior, lo típico es que en los servicios sea donde se programe la comunicación con el Backend. Al importar en el módulo el HttpClientModule ya se puede injectar en los servicios un objeto de tipo HttpClient



El método más usado de HttpClient es el método "get", que corresponde con el método "get" de HTTP. El método recibe por parámetro la URL con la cual se va a conectar y retorna un objeto "observable".