



## INTRODUCCIÓN A ANGULAR 8 / 9

### DESDE 0

LAURA LUCENA BUENDÍA 2DAM

Angular es una plataforma/Framework que permite desarrollar aplicaciones web en la sección cliente, utilizando principalmente: HTML, JS (Typescript) y CSS

Son aplicaciones de una sola página (SPA: Single Page Application) realizando la carga de datos de forma asíncrona.

Angular está orientado a objetos, trabaja con clases y favorece el uso del patrón MVC.

***Introducción Typescript:** Es un lenguaje orientado a objetos que se traduce a JS, añadiéndole características que no posee. Permite desarrollar un código con menos errores.*

## 1 INSTALAR ANGULAR

### 1.1 PREPARACIÓN IDE (VSC - VISUAL STUDIO CODE)

## 2 ESTRUCTURA DEL PROYECTO

## 3 ANÁLISIS DE COMPONENTE

### 3.1 Creación de componentes y estructura de un componente

### 3.2 Mostrando un componente

# 1 INSTALAR ANGULAR. PASOS

## 1- INSTALAMOS/ACTUALIZAMOS NODE JS:

<https://nodejs.org/es/>

## 2- ABRIMOS CONSOLA/CMD.

Una vez ya instalado tenemos que actualizar el 'npm', es gestor de paquetes de Node.

Escribimos: `npm install -g npm@latest`

-Limpiamos el cache: `npm cache clean --force`

-Desactivamos las auditorias de npm para evitar fallos en la instalación: `npm set audit false`

## 3- DESINSTALAMOS E INSTALAMOS ANGULAR/ANGULAR CLI

-Si ya lo tuviéramos instalado habría que desinstalarlo con estos comandos:

`npm uninstall -g angular-cli`

`npm uninstall -g @angular/cli`

borramos caché: `npm cache clean --force`

E instalamos la última versión de Angular Cli:

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

**¡LISTO!**

## 1.2 Entramos en el directorio de nuestro proyecto

1- Accedemos a la carpeta en la consola:

`C:\Users\laura\OneDrive\Escritorio\Angular_Curso`

2- comando para crear un proyecto: `ng new`

3-Nos aparecerá:

Para dar nombre a nuestro proyecto:

? What name would you like to use for the new workspace and initial project? `Test1`

Le decimos que no, para hacerlo a mano porque puede generar fallos:

? Would you like to add Angular routing? `No`

Hoja de estilo:

? Which stylesheet format would you like to use? `CSS`

4- comando `ng serve`

Nos sirve para visualizar nuestra página



## 1.1 PREPARACIÓN IDE (VSC - VISUAL STUDIO CODE)

DESCARGA: <https://code.visualstudio.com/download>

NO SÉ UTILIZARLO / TUTORIAL (Sacarle el máximo partido):














<https://www.youtube.com/watch?v=ljz1mXQm7KU>







### EXTENSIONES NECESARIAS PARA TRABAJAR CON ÉL:

(Hay otras que no son necesarias, que son a gusto personal y porque las necesitaré para trabajar con él)

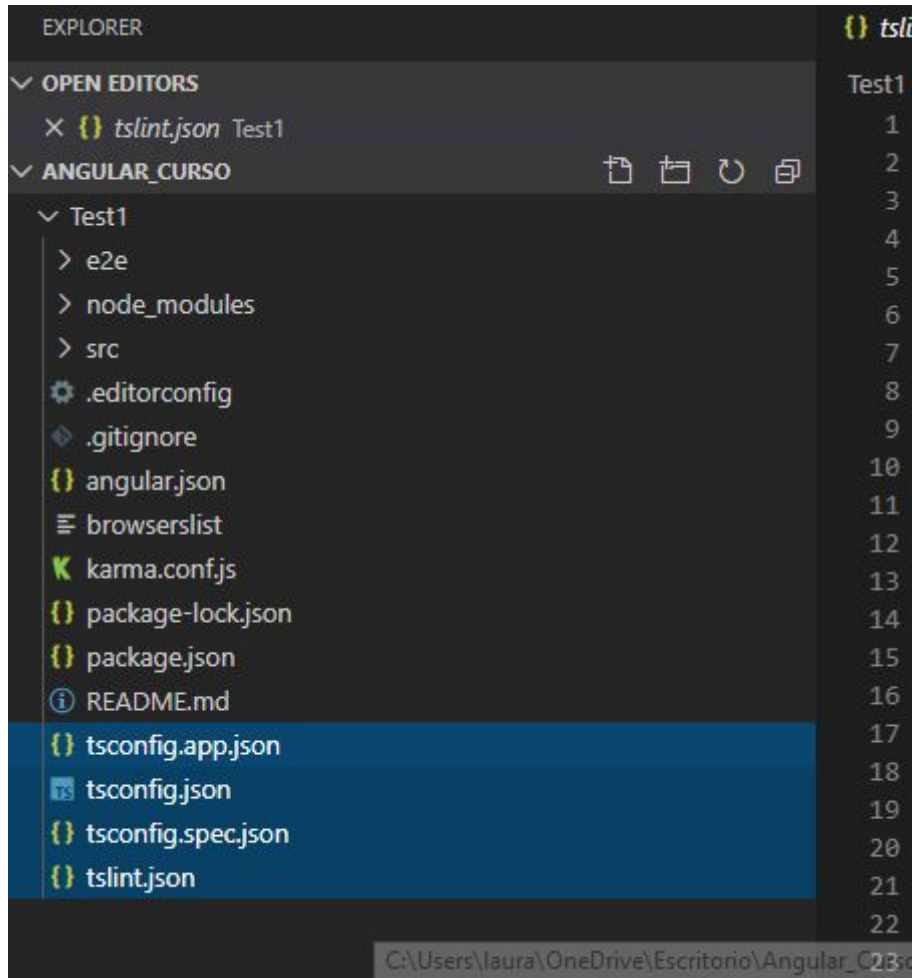
The screenshot shows the Visual Studio Code Extensions view with a list of installed and available extensions. The extensions are listed in a dark theme with their icons, names, versions, and descriptions.

- Angular 8 and TypeScript/HTML VS Code Snippets** 1.0.21  
VS Code snippets for Angular and TypeScript/HTML  
Dan Wahlin
- Angular 8 Snippets - TypeScript, Html, Angular Material, ngRx, RxJS & Flex Layout** 8.1.2  
242 Angular Snippets (TypeScript, Html, Angular Material, Flex Layout, ngRx, RxJS, PWA & Testing) Updated for v8 Beta  
Mikael Morlund
- Angular Essentials (Version 9)** 9.0.1  
Essential extensions for Angular developers  
John Papa
- Angular Files** 1.6.2  
Quickly scaffold angular file templates  
Alexander Ivanichev
- Angular Language Service** 0.900.18  
Editor services for Angular templates  
Angular
- Angular Schematics** 3.3.0  
Angular schematics (CLI commands) from files Explorer or Command Palette.  
Cyrille Tuzi
- Angular Snippets (Version 9)** 9.1.2  
Angular version 9 snippets by John Papa  
John Papa
- angular2-inline** 0.0.17  
Visual Studio Code language extension for javascript/typescript files that use Angular2.  
Nate Wallace
- Auto Close Tag** 0.5.6  
Automatically add HTML/XML close tag, same as Visual Studio IDE or Sublime Text  
Jun Han
- Auto Import** 1.5.3  
Automatically finds, parses and provides code actions and code completion for all available imports. Works with Typescript and TSX  
steoates
- Babel ES6/ES7** 0.0.4  
Adds JS Babel es6/es7 syntax coloring  
dzannotti
- Bracket Pair Colorizer** 1.0.61  
A customizable extension for colorizing matching brackets  
CoenraadS
- Close HTML/XML tag** 1.2.0  
Quickly close last opened HTML/XML tag  
Compulim

- 
**EditorConfig for VS Code** 0.14.4  
 EditorConfig Support for Visual Studio Code  
 EditorConfig
- 
**ESLint** 2.1.1  
 Integrates ESLint JavaScript into VS Code.  
 Dirk Baeumer
- 
**file-icons** 1.0.22  
 File-specific icons in VSCode for improved visual grepping.  
 file-icons
- 
**HTML CSS Support** 0.2.3  
 CSS support for HTML documents  
 emel
- 
**HTML Format** 0.0.4  
 Format HTML documents.  
 Mohamed Akram
- 
**HTML Preview** 0.2.5  
 Provides ability to preview HTML documents.  
 Thomas Haakon Townsend
- 
**HTML Snippets** 0.2.1  
 Full HTML tags including HTML5 Snippets  
 Mohamed Abusaid
- 
**IntelliSense for CSS class names in HTML** 1.19.0  
 CSS class name completion for the HTML class attribute based on the definitions found in your workspace.  
 Zignd
- 
**Jinja** 0.0.8  
 Jinja template language support for Visual Studio Code  
 wholroyd
- 
**JS-CSS-HTML Formatter** 0.2.3  
 Format ,prettify and beautify JS, CSS, HTML code by using shortcuts, context menu or CLI  
 lonefy
- 
**JSS Snippets** 0.2.2  
 CSS in JS snippets for VS Code  
 visioncan
- 
**Laravel Blade Snippets** 1.20.0  
 Laravel blade snippets and syntax highlight support  
 Winnie Lin
- 
**Laravel Extra Intellisense** 0.2.5  
 better intellisense for laravel projects.  
 amir

- 
**PHP IntelliSense** 2.3.14  
 Advanced Autocompletion and Refactoring support for PHP  
 Felix Becker
- 
**PHP Intelephense** 1.3.11  
 PHP code intelligence for Visual Studio Code  
 Ben Mewburn
- 
**PHP Debug** 1.13.0  
 Debug support for PHP with XDebug  
 Felix Becker
- 
**npm** 0.3.11  
 npm support for VS Code  
 egamma
- 
**npm** 3.3.0  
 npm commands for VSCode  
 Florian Knop
- 
**npm Intellisense** 1.3.0  
 Visual Studio Code plugin that autocompletes npm modules in import statements  
 Christian Kohler

## 2 Estructura del Proyecto



### De Arriba a Abajo:

**e2e:** Temas de Testing. Contiene los archivos de prueba, denominados end to end.

Se ejecutan con Jasmine.

**node\_modules:** Se guardan todas las dependencias, paquetes, librerías, etc.. de nuestro proyecto.

### PARTE IMPORTANTE

**src:** Aquí es donde vamos a estar trabajando.

Es donde estarán nuestros componentes.

**.editorconfig:** Es un fichero que contiene la configuración de nuestro editor.

**.gitignore:** Para el repositorio de Git. Ignora los archivos de cara al control de versiones.

**angular.json:** Nos sirve para configurar el proyecto en general, y cargar script, asset, librerías externas...

**karma.conf.js:** Fichero para configurar temas de prueba (Con Jasmine).

**package-lock.json:** Se genera cada vez que instalamos algún módulo de Node, etc

**package.json:** archivo donde configuramos la versión de framework, diferentes scripts y comandos, librerías...

Cuando instalemos un paquete lo veremos aquí

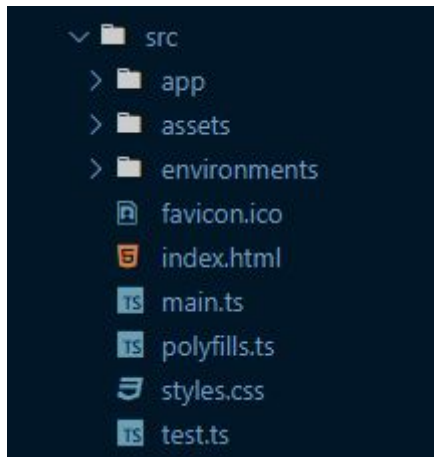
**README:** documentación

**Los archivos tsconfig/tslint:** Son archivos de configuración de TypeScript, que hace que el compilador se configure

## SRC: CONTIENE LOS ARCHIVOS FUENTES RAÍZ DE LA APP

### SRC

Aquí encontraremos:



**APP:** Carpeta raíz que contendrá los componentes, servicios, y resto de elementos que constituyen la aplicación.

**ASSETS:** Imágenes, vídeos y archivos en general. Que no son propiedad de ningún componente.

**ENVIROMENTS:** Características relativas el IDE.

**favicon.ico:** Img que permite identificar nuestra aplicación.

**index.html:** Archivo raíz donde se inicia la aplicación.

**main.ts:** El archivo principal de TS, donde se cargan todos los componentes y carga el módulo principal de Framework.

**polyfills.ts:** Asegura la compatibilidad de los navegadores.

**styles.css:** Estilos del proyecto.

**test.ts:** Puede contener test unitarios.

### 3 ANÁLISIS DE COMPONENTE

Un componente es una clase de JS a la que se añade un decorador (**@Component**)

*(englobada en un directorio junto con otros ficheros que necesita: html, css...)*

y que permite crear nuevas etiquetas HTML.

Los componentes básicamente, permiten crear tus propias etiquetas HTML personalizables

Básicamente los componentes se organizan en forma de árbol, teniendo un componente principal donde, por defecto, su propiedad 'selector' posee el valor **<app-root>**

Un componente es un 'trozo' de pantalla dentro de mi aplicación.

Es una pequeña parte de mi aplicación, la cual, estará formada por muchísimos componentes.

Todo lo que vamos a utilizar y todo lo que vamos a estar mostrando por pantalla es un componente.

**Los componentes se componen PRINCIPALMENTE de 3 partes:**

**-Selector:** Nombre a utilizar como Tag/Etiqueta en HTML.

**-Formato:** Puede ser **template** o **templateUrl**.

Es la plantilla HTML del componente.

Con **template** escribimos la plantilla HTML de nuestra página a 'pelo'

y con **templateUrl** añadimos la Url donde se encuentra nuestro archivo html.

**-Estilo:** Puede ser **style** o **styleUrl**.

Es la plantilla CSS del componente.

Con **style** escribimos nuestro diseño a 'pelo' (justo como template)

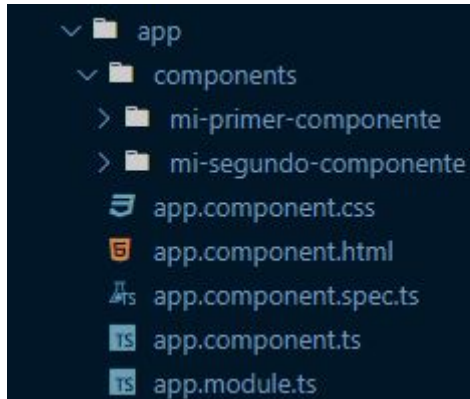
y con **styleUrl** añadimos la Url donde se encuentra nuestro archivo de hoja de estilos css.

Los componentes son reutilizables.

### 3.1 Creando componentes

Podemos crear los componentes de muy diversas maneras. Pero lo más importante es que estén bien organizados en nuestro proyecto.

#### EJEMPLO DE COMO TENDREMOS ESTRUCTURADOS NUESTROS COMPONENTES



```

└─ app
   └─ components
      ├── mi-primer-componente
      ├── mi-segundo-componente
      ├── app.component.css
      ├── app.component.html
      ├── app.component.spec.ts
      ├── app.component.ts
      └── app.module.ts

```

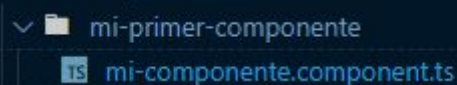
Dentro del directorio '**APP**' nos hemos creado otro directorio llamado '**components**', dónde irán los dos componentes que vamos a crear.

Puedes organizarlo/estructurarlo como desees pero es importante que los componentes tengan estén ordenados correctamente orden y lleven una estructura (ya que pueden haber componentes que estén formados por otros).

#### 1- Primera forma de crear un componente:

1. Nos creamos otro Directorio dentro de '*components*', el cual llamaremos como a nuestro componente, en este caso '*mi-primer-componente*'
2. Dentro del directorio, '*mi-primer-componente*', nos creamos el archivo TypeScript(TS).  
El cual, tendrá el mismo nombre que nuestro Directorio,  
y la extensión '*.component.ts*'.
3. Esta sería la estructura:      '*mi-primer-componente.component.ts*'.  
   '*nombreComponente.component.ts*'.

#### VISUALIZACIÓN DE CÓMO SERÍA EN EL PROYECTO:



```

└─ mi-primer-componente
   └─ mi-componente.component.ts

```



## 2- Segunda forma de crear un componente.

La segunda forma es mediante comandos, bastante útil, ya que nos generará todos los archivos de un componente

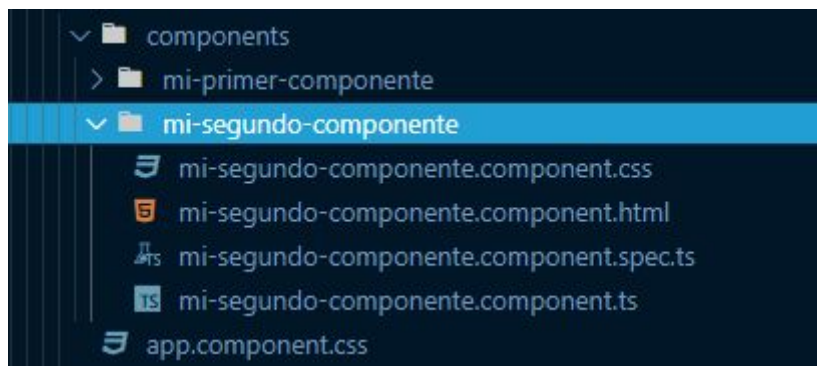
### 1. Nos dirigimos a la ruta donde queremos crear el componente

En este caso:

```
C:\Users\laura\OneDrive\Escritorio\0_Proyecto_Test\Test1\src\app\components>
```

### 2. Usamos el comando: `ng generate component mi-segundo-componente` `ng generate component nombreComponente`

**VISUALIZACIÓN DE CÓMO SERÍA EN EL PROYECTO:**



*La segunda forma es mucho más completa y útil que la primera*

*Además de que todos los elementos del componente están interrelacionados entre sí y no es necesario crearlos uno a uno como en el primer ejemplo.*

## 3.2 Mostrando un componente

Ejemplo de nuestro componente *'mi-primer-componente'*:

```
1  import { Component } from '@angular/core'; /*Importamos la clase Component*/
2
3
4  /* DECORADOR */
5  @Component({
6    selector: 'primer-componente',
7
8    template: `<h1>{{titulo}}</h1>
9                <hr/>>
10               <h2>{{comentario}}</h2>
11               <p>{{year}}</p>`
12  }) /*CLASE TS*/
14  export class MiPrimerComponente {
15    //VARIABLES DE LA CLASE
16    public titulo: string;
17    public comentario: string;
18    public year: number;
19
20    constructor() {
21      this.titulo = "Hola Mundo";
22      this.comentario = "Este es mi primer componente";
23      this.year = 2020;
24
25      console.log("Soy como el print");
26      console.log(this.titulo);
27    }
28  }
```

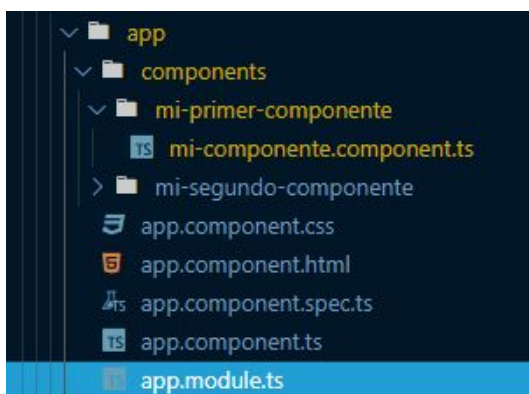
Para tener claros ciertos conceptos básicos y saber cómo funcionan.

En nuestro *mi-primer-componente.ts* y cualquier componente:

Siempre hay que importar la clase Component, añadir un decorador y exportar la clase. Nunca se cierra con ';', ya que es un JSON.

El decorador siempre debe tener mínimo 2 propiedades:

**SELECTOR** (nombre etiqueta) y **TEMPLATE**(HTML)



Para que este Componente funcione/se vea lo que tengo que hacer es cargarlo en el **app.module.ts** Alojado dentro de 'app'

Esto es para poder tenerlo disponible en cualquier parte de la aplicación

## EN 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 { MiSegundoComponenteComponent } from './components/mi-segundo-componente/mi-segundo-componente.component';
6 import { MiPrimerComponente } from './components/mi-primer-componente/mi-componente.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     MiPrimerComponente,
12     MiSegundoComponenteComponent
13   ],
14   imports: [
15     BrowserModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
```

1. Tenemos que **'importar'** el componente (*nombre de la clase*, acordemonos de que la clase tiene escrita la palabra **'export'**) y escribir la dirección de donde se encuentra
2. También en **'declarations'** debemos escribir el nombre de este (de la clase).

### ¡ATENCIÓN!

Si nos fijamos **'MiSegundoComponente'** está ahí también declarado.  
Esto es debido a que si nos creamos los componentes por el comando: `ng new component`  
Se nos cargará automáticamente dentro del `'app.module.ts'`.

Así que recomiendo quedarnos con la segunda opción para ahorrarnos trabajo.  
Y con este ejemplo sabemos un poco más de la estructura del funcionamiento del FW.

Así que ya dentro de `app.component.html` yo ya podría invocar ya a mi componente, como si fuera una etiqueta de html y visualizarlo