

Desarrollo y programación de Aplicaciones Avanzadas Angular

4 Estructura de módulos,
componentes y ciclos de vida

CORE
networks



Estructura de archivos de un proyecto Angular

Creación de nuevo proyecto:

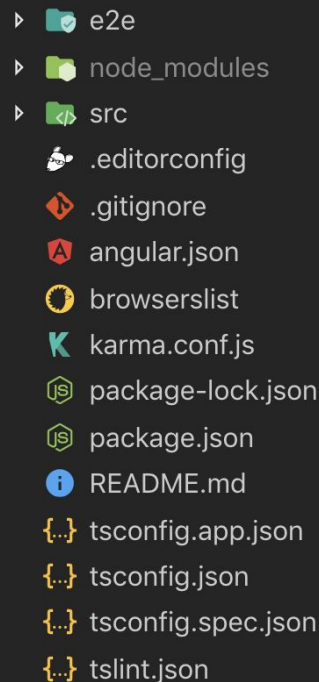
```
ng new nombreproyecto
```

El proyecto se crea en un directorio con el mismo nombre y puede ser levantado en modo de desarrollo (por defecto en <http://localhost:4200/>):

```
ng serve -o
```

Estructura de archivos de un proyecto Angular

Directorio e2e	Test end to end
Directorio node_modules	Librerías Angular y terceros
Directorio src	Archivos de la aplicación
Archivo angular.json	Configuración de Angular CLI
Archivos package.json	Scripts y dependencias
Archivo tsconfig.json	Configuración de tests
Archivo tslint.json	Configuración de TypeScript

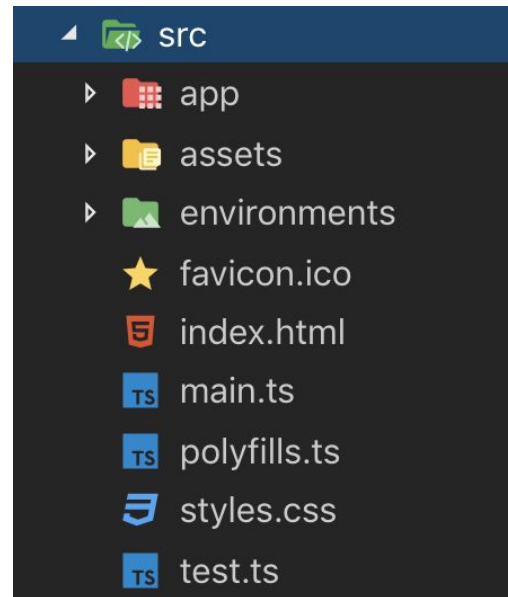


A screenshot of a file explorer showing the structure of an Angular project. The files and folders are listed with their respective icons: e2e (test icon), node_modules (package icon), src (code icon), .editorconfig (editor icon), .gitignore (git icon), angular.json (Angular icon), browserslist (browser icon), karma.conf.js (Karma icon), package-lock.json (package icon), package.json (package icon), README.md (info icon), tsconfig.app.json (JSON icon), tsconfig.json (JSON icon), tsconfig.spec.json (JSON icon), and tslint.json (JSON icon).

- ▶ e2e
- ▶ node_modules
- ▶ src
- .editorconfig
- .gitignore
- angular.json
- browserslist
- karma.conf.js
- package-lock.json
- package.json
- README.md
- tsconfig.app.json
- tsconfig.json
- tsconfig.spec.json
- tslint.json

Directorio src

Direct. app	Módulos y componentes aplicación
Direct. assets	Archivos estáticos
Direct. enviroments	Archivos para constantes dev/prod
Archivo index.html	Punto de entrada HTML
Archivo main.js	Punto de entrada JavaScript
Archivo styles.css	Estilos globales CSS



Módulos en Angular

Podemos definir un módulo en Angular (no confundir con los módulos ECMAScript 6) como un contenedor que agrupa componentes relacionados por su funcionalidad.

Una aplicación Angular tendrá siempre al menos un módulo, el raíz, generado por Angular CLI con el nombre app.

Los módulos se administran o configuran desde archivos TypeScript identificados como:

```
nombremodulo.module.ts
```

Módulo raíz app

app.module.ts

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



Importación librerías



Importación elementos aplicación



Función decoradora con elementos aplicación



Clase del módulo

Importación de dependencias


Todos los elementos que componen un proyecto Angular se relacionan entre sí mediante importaciones y exportaciones de clases, en las que están contenidas las propiedades y métodos TypeScript, usando la sintaxis:

```
import { NombredeLaClase } from 'ruta/archivosinextension';  
  
export class NombredeLaClase { }
```

Componentes en Angular

Podemos definir un componente como un elemento que encapsula los datos, el marcado HTML y la lógica de las vistas de una aplicación.

Una aplicación Angular tendrá siempre al menos un componente raíz, generado por Angular CLI con el nombre app y que está compuesto por 4 archivos:



app.component.css
app.component.html
app.component.spec.ts
app.component.ts

← Estilos
← Plantilla
← Test unitario
← Lógica

Archivo de la clase del componente TypeScript

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



Importaciones

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
  
|  
  
export class AppComponent {  
  title = 'practica01';  
}
```



Función decoradora a la que se le pasa un objeto de Metadatos



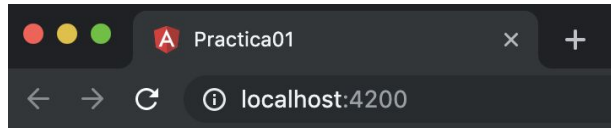
Clase del componente con el código TypeScript

Ejemplo '¡Hola Mundo!'

```
app.component.ts x
src ▶ app ▶ 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
9  export class AppComponent {
10 |   mensaje: any = '¡Hola Mundo!';
11 }
```

```
app.component.html x
src ▶ app ▶ app.component.html ▶ h1
1  <h1>{{mensaje}}</h1>
2
```

```
app.component.css x
src ▶ app ▶ app.component.css ▶ h1
1  h1 {
2    color: crimson;
3  }
```



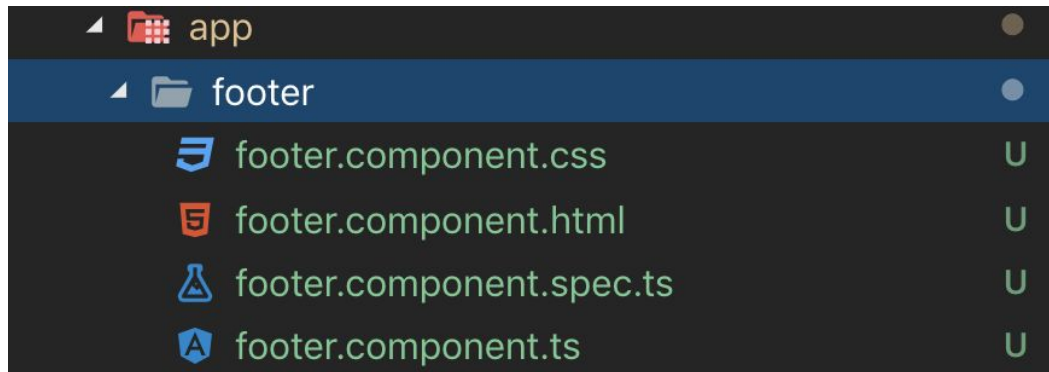
¡Hola Mundo!

Creación de nuevos componentes

Creación de nuevos componentes con Angular CLI:

```
ng g c nombrecomponente
```

Los archivos del componente se crean en un directorio con el mismo nombre.



Creación de nuevos componentes

Sus dependencias son actualizadas en el módulo (app.module.ts) de manera automática:

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



Inserción de un componente

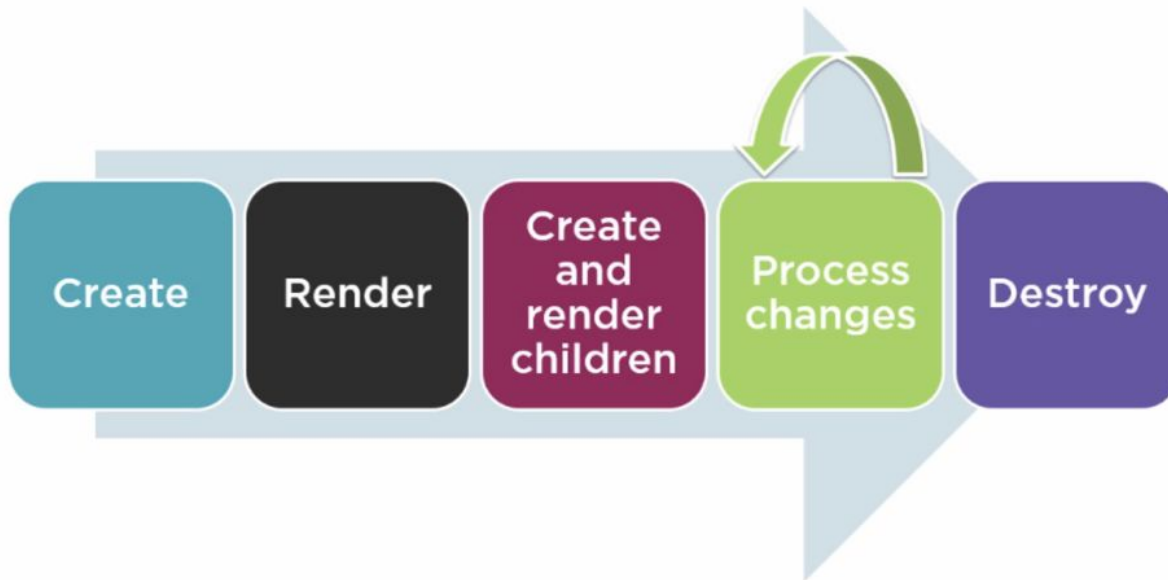
Los componentes son incorporados a cualquier archivo de plantilla (HTML) de otro componente mediante un elemento HTML identificado con el selector de éste:

```
app.component.html x

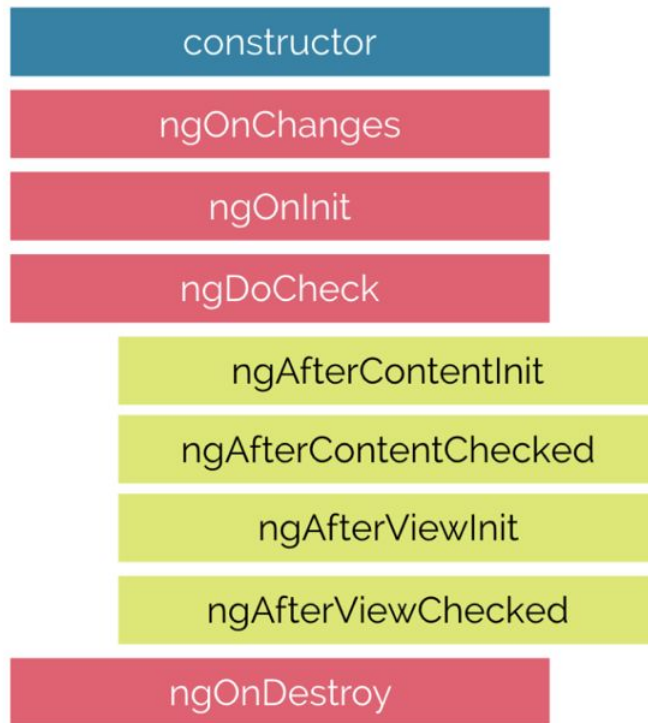
src ▸ app ▸ app.component.html ▸ ...
1   <h1>{{mensaje}}</h1>
2   <app-footer></app-footer>
3
```

Ciclo de vida de componentes

Angular dispone de una serie de métodos para ejecutar hooks en las diferentes fases del ciclo de vida de los componentes. Estos métodos son implementados a través de interfaces en la clase del componente.



Secuencia



Interfaces y métodos

Interfaz	Método	Descripción
OnChanges	ngOnChanges()	Responde cuando Angular restablece propiedades de entrada vinculadas a datos. El método recibe un objeto SimpleChanges de los valores de propiedad actuales y anteriores. Es llamado antes de ngOnInit() y cuando una o más propiedades de entrada enlazadas a datos cambien.
OnInit	ngOnInit()	Inicializa la directiva o el componente una vez que Angular haya mostrado las propiedades vinculadas a datos y establece las propiedades de entrada de la directiva o el componente. Es llamado solo una vez, justo después del primer ngOnChanges().

Interfaces y métodos

Interfaz	Método	Descripción
DoCheck	ngDoCheck()	Es activado durante cada detección de cambios, inmediatamente después de ngOnChanges() y ngOnInit().
AfterContentInit	ngAfterContentInit()	Es activado después de que Angular proyecte contenido externo en la vista del componente. Se llama una vez después del primer NgDoCheck().

Interfaces y métodos

Interfaz	Método	Descripción
AfterContentChecked	ngAfterContentChecked()	Responde después de que Angular comprueba el contenido proyectado en el componente. Es llamado después de ngAfterContentInit() y cada NgDoCheck() posterior.
AfterViewInit	ngAfterViewInit()	Responde después de que Angular inicializa las vistas del componente y las vistas secundarias. Se llama una vez después del primer ngAfterContentChecked().

Interfaces y métodos

Interfaz	Método	Descripción
AfterViewChecked	ngAfterViewChecked()	Responde después de que Angular comprueba las vistas del componente y las vistas secundarias. Se llama después de ngAfterViewInit() y cada ngAfterContentChecked() posterior.
OnDestroy	ngOnDestroy()	Es llamado justo antes de que Angular destruya la directiva o el componente. Anula la suscripción de observables y desconecta los manejadores de eventos para evitar fugas de memoria.