

1-Base

Base para una aplicación Angular

- 1. Módulos**
- 2. Componentes**
- 3. Visibilidad entre componentes**
- 4. Transitividad y Organización**

1. Módulos

Anatomía de un módulo

Generación de módulos

1.1 Anatomía de un módulo

Un módulo es una clase decorada en **TypeScript**

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

Árbol de módulos mediante el array de `imports: []`

1.2 Generación de módulos

Usando el programa `ng` con el comando `generate` con la opción `module` y un nombre

```
ng g m core
```

Resulta en el fichero `core/core.module.ts`

```
@NgModule({  
  imports: [],  
  declarations: []  
})  
export class CoreModule {}
```

Árbol de módulos mediante el array de `imports: []`

Se agrega al array de importaciones en `AppModule`

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

Recap:

1. Módulos

Anatomía de un módulo

Generación de módulos

2. Componentes

Anatomía de un componente

Generación de componentes

2.1 Anatomía de un componente

- Un componente es una clase decorada en **TypeScript**
- Asociada a una plantilla **html**
- Con un selector **html**

```
import { Core } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styles: []  
})  
export class AppComponent {}
```

Para ser consumido

- Requiere un módulo donde declararse

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

- Y está listo para ser instanciado

```
<body>  
  <app-root></app-root>  
</body>
```

1.2 Generación de componentes

Usando el programa `ng` con el comando `generate` con la opción `component`

```
ng g c core/shell
ng g c core/shell/header
ng g c core/shell/main
ng g c core/shell/footer
```

Resulta en ficheros como `core/shell.component.ts`

```
@Component({
  selector: 'app-shell',
  templateUrl: './shell.component.html',
  styles: []
})
export class ShellComponent implements OnInit {
  constructor() {}
  ngOnInit() {}
}
```

Composición de componentes

```
<app-header></app-header>  
<app-main></app-main>  
<app-footer></app-footer>
```

| Recap:

2. Componentes

Anatomía de un componente

Generación de componentes

3. Visibilidad entre componentes

Componentes públicos y privados

Importación y exportación entre módulos

3.1 Componentes públicos y privados

Los componentes inicialmente **sólo pueden usarse en su propio módulo**

■ Para poder usar un componente fuera de su módulo necesito

Exportar el componente

```
@NgModule({  
  declarations: [ShellComponent, HeaderComponent, MainComponent, FooterComponent],  
  imports: [CommonModule, RouterModule],  
  exports: [ShellComponent]  
})  
export class CoreModule {}
```

■ y algo más...

3.2 Importación y exportación entre módulos

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

y entonces `app.component.html` queda ridículamente simple:

```
<app-shell></app-shell>
```


La componentización implica mover contenido

- El contenido de `app.component.html` irá a *Header, Main y Footer*
- La propiedad `title` se moverá a `header.component.ts`
- ¿y qué pasa con `<router-outlet></router-outlet>`?
- Falla porque no es conocido en `CoreModule`; hay que importarlo

```
@NgModule({  
  declarations: [ShellComponent, HeaderComponent, MainComponent, FooterComponent],  
  imports: [CommonModule, RouterModule],  
  exports: [ShellComponent]  
})  
export class CoreModule {}
```

3.2.1 Dos mundos paralelos: imports de Angular e import de TypeScript

■ En TypeScript cada fichero es un módulo

Para que algo sea visible desde fuera

Primero debe exportarlo

```
export class AppComponent {}
```

Y luego importarlo

```
import { AppComponent } from './app.component';
```

| Recap:

3. Visibilidad entre componentes

Componentes públicos y privados

Importación y exportación entre módulos

4. Transitividad y Organización

Transitividad en una cadena de módulos

Organización de la aplicación en módulos

4.1 Transitividad en una cadena de módulos

Un módulo puede exportar sus componentes
Pero también los de otros módulos relacionados
Incluso un módulo completo

- Al mover contenido de `app.component.html` a los componentes de `CoreModule`.
- Para que funcionase hubo que importar el `RouterModule`, necesario para usar `<router-outlet>`.
- ¿Cómo es que **antes** funcionaba?
- Por la **transitividad** usada en `AppRoutingModule`

Imports - Exports

`AppRoutingModule` importa y exporta a `RouterModule`

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Luego el contenido `RouterModule` se podía usar directamente en `AppModule`

En `app.component.html`

```
<h2>Here are some links to help you start...</h2>
<router-outlet></router-outlet>
```

4.2 Organización de la aplicación en módulos

- Los programas se organizan a partir de piezas menores.
- Los principios de **código limpio** nos permiten identificarlas y reutilizarlas.
- Los módulos y los componentes son piezas reutilizables
- Habrá piezas *funcionales* y otras de *infraestructura*.
- Alguna será de uso único como el `CoreModule`
- Y otras serán compartidas como el `SharedModule`

ng g m shared

El bosque de módulos a vista de pájaro

```
AppModule
|
|--AppRoutingModule
|   |--RouterModule
|
|--BrowserModule
|
|--CoreModule
|   |--CommonModule
|   |--RouterModule
SharedModule
```


El bosque de componentes a vista de pájaro

```
AppComponent
|
+--ShellComponent
|
|   +--HeaderComponent
|   |
|   +--MainComponent
|   |   |
|   |   +--RouterOutletComponent
|   |
|   +--FooterComponent
```

Recap:

4. Transitividad y Organización

Transitividad en una cadena de módulos

Organización de la aplicación en módulos

| Next:

Páginas y rutas Angular SPA

Rutas

Lazy Loading

Parámetros

Rutas anidadas

| **Blog de apoyo:** [Base para una aplicación Angular](#)

| **By** [Alberto Basalo](#)