Angular: Conceptos Fundamentales

¿Qué es Angular?

Angular organiza los proyectos de forma modular, clara y escalable. La arquitectura está dividida en carpetas y ficheros con responsabilidades bien definidas, lo que facilita el mantenimiento y la reutilización.

Estructura General del Proyecto

Esta es la estructura típica generada por el CLI (`ng new nombre-proyecto`):

estructura-proyecto.txt

```
my-angular-app/
├── node_modules/            # Dependencias instaladas por npm
├── src/                     # Código fuente principal
│   ├── app/                 # Módulos, componentes, servicios principales
│   │   ├── core/            # Servicios singleton, guardas, interceptores
│   │   ├── shared/          # Componentes y pipes reutilizables
│   │   ├── features/        # Módulos funcionales (ej: usuarios, productos)
│   │   ├── services/        # Servicios específicos
│   │   ├── app.component.ts  # Componente raíz
│   │   └── app.module.ts    # Módulo raíz
│   ├── assets/              # Imágenes, iconos, fuentes
│   ├── environments/        # Configuraciones por entorno (dev, prod)
│   └── index.html           # Página principal de la aplicación
├── angular.json             # Configuración del proyecto Angular y builds
├── package.json             # Dependencias y scripts npm
├── tsconfig.json            # Configuración TypeScript global
├── tsconfig.app.json        # Configuración específica para la app
├── tsconfig.spec.json       # Configuración para pruebas unitarias
├── .editorconfig            # Reglas de formato de código
└── README.md                # Documentación del proyecto
```

Carpetas Principales

- **src/app/core:** contiene servicios globales, guardas de rutas e interceptores de HTTP.
- **src/app/shared:** incluye componentes, pipes y directivas reutilizables en distintos módulos.
- **src/app/features:** alberga los módulos de características, cada uno con su propio routing y componentes.
- **src/app/services:** guarda servicios específicos que no necesitan ser globales.

> 💡 **Regla general:** Los componentes compartidos van en `shared`, mientras que las secciones o módulos funcionales completos (como "usuarios" o "productos") van en `features`.

Ficheros de Configuración

1. package.json

Define las dependencias, scripts y metadatos del proyecto. Angular usa npm para gestionar todas las librerías y versiones.

package.json

```json
{
  "name": "my-angular-app",
  "version": "1.0.0",
  "scripts": {
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test"
  },
  "dependencies": {
    "@angular/core": "^18.0.0",
    "rxjs": "^7.8.1"
  },
  "devDependencies": {
    "typescript": "^5.3.0"
  }
}
```

2. angular.json

Controla la configuración del CLI: cómo se construye, sirve o prueba la aplicación. Define los proyectos y sus rutas.

angular.json

```json
{
  "projects": {
    "my-angular-app": {
      "root": "",
      "sourceRoot": "src",
      "projectType": "application",
      "architect": {
        "build": { "builder": "@angular-devkit/build-angular:browser" },
        "serve": { "builder": "@angular-devkit/build-angular:dev-server" }
      }
    }
  }
}
```

3. tsconfig.json

Configura el compilador de TypeScript. Define reglas como el target de compilación y los paths.

tsconfig.json

```json
{
  "compilerOptions": {
    "target": "es2022",
    "module": "es2022",
    "strict": true,
    "skipLibCheck": true,
    "baseUrl": "./",
    "paths": { "@app/*": ["src/app/*"] }
  }
}
```

Componentes, Servicios y Directivas

Los **componentes** son la base de la UI. Los **servicios** manejan la lógica y las peticiones HTTP. Las **directivas** modifican el comportamiento del DOM (por ejemplo, `*ngIf` o `*ngFor`).

user.component.ts

```typescript
@Component({
  selector: 'app-user',
  template: `<h2>Usuario: {{ nombre }}</h2>`
})
export class UserComponent {
  nombre = 'Carlos';
}
```

Formularios en Angular

Angular ofrece dos enfoques: **Template-driven forms** y **Reactive forms** (más potente y tipado). Se usa `FormGroup`, `FormControl` y `FormBuilder` desde `@angular/forms`.

login-form.component.ts

```typescript
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-login',
  template: `
    <form [formGroup]="form">
      <input formControlName="email" placeholder="Correo" />
      <input formControlName="password" type="password" />
      <button type="submit">Entrar</button>
    </form>
  `
})
export class LoginFormComponent {
  form: FormGroup;

  constructor(private fb: FormBuilder) {
    this.form = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
      password: ['', Validators.required]
    });
  }
}
```

Tipado en TypeScript y Buenas Prácticas

TypeScript permite definir tipos estrictos que ayudan a prevenir errores y mejorar la mantenibilidad del código. Angular aprovecha esto para tipar componentes, servicios y formularios.

tipos.ts

```typescript
let nombre: string = 'Carlos';
let edad: number = 30;
let activo: boolean = true;

interface Usuario {
  id: number;
  nombre: string;
  correo?: string; // opcional
}

const user: Usuario = { id: 1, nombre: 'Carlos' };
```

Routing y Navegación

El sistema de rutas en Angular permite navegar entre vistas sin recargar la página. Se configura en el módulo de routing con `RouterModule.forRoot()`.

app-routing.module.ts

```typescript
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './features/home/home.component';
import { UserComponent } from './features/user/user.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'user/:id', component: UserComponent },
  { path: '**', redirectTo: '' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

La navegación se realiza con la directiva `routerLink` o desde código con `Router.navigate()`.

Reutilización de Componentes y Módulos

Para mantener un proyecto escalable, Angular recomienda separar la aplicación en módulos funcionales y compartidos.

- **CoreModule:** Servicios globales, guardas, interceptores. Importado solo en `AppModule`.
- **SharedModule:** Componentes, directivas y pipes reutilizables. Importado en todos los módulos que los necesiten.
- **FeatureModules:** Módulos de secciones funcionales como "Usuarios", "Productos", etc.

shared.module.ts

```typescript
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ButtonModule } from 'primeng/button';

@NgModule({
  exports: [CommonModule, ButtonModule]
})
export class SharedModule {}
```

Observables y RxJS

RxJS permite manejar flujos de datos asíncronos con observables. Angular lo usa para eventos, formularios y peticiones HTTP.

user.service.ts

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({ providedIn: 'root' })
export class UserService {
  constructor(private http: HttpClient) {}

  getUsuarios(): Observable<any[]> {
    return this.http.get<any[]>('/api/users');
  }
}
```

⚙️ Dependencias Útiles Recomendadas

Estas son algunas de las dependencias más utilizadas en proyectos Angular, con una breve descripción y su comando de instalación:

Paquete	Descripción	Instalación
`@angular/forms`	Formularios reactivos y template-driven	*Incluido por defecto*
`@angular/router`	Sistema de enrutamiento SPA (Single Page Application)	*Incluido por defecto*
`@angular/material`	Componentes de interfaz basados en Material Design	`npm install @angular/material`
`primeng`	Librería de componentes UI muy personalizable	`npm install primeng primeicons`
`ngx-translate`	Soporte para internacionalización (i18n)	`npm install @ngx-translate/core @ngx-translate/http-loader`
`rxjs`	Programación reactiva con observables	*Incluido por defecto*
`bootstrap`	Framework de estilos CSS popular	`npm install bootstrap`

Tips para un Código Limpio y Eficiente

- ✅ Usa `OnPush` en la detección de cambios para optimizar rendimiento.
- ✅ Tipar siempre los datos provenientes de servicios (`Observable<Usuario[]>`).
- ✅ Evita lógica compleja en los componentes; trasládala a servicios.
- ✅ Usa `async pipe` en plantillas para manejar observables sin suscribirte manualmente.
- ✅ Aprovecha `standalone components` (Angular 17+) para proyectos más ligeros.