# Unidad 6: Organización y navegación

# Fundamentos del enrutamiento

## Introducción al router

El router en Angular es una herramienta esencial que permite la navegación entre vistas en aplicaciones de una sola página (SPA, por sus siglas en inglés). Facilita el manejo de rutas para crear una experiencia de usuario fluida y dinámica.

**Conceptos clave:**

1. **¿Qué es el Router en Angular?**
   * Es un sistema de navegación que permite mostrar diferentes componentes según la URL de la aplicación.
   * Las rutas se configuran en el archivo app-routing.module.ts utilizando un arreglo de objetos.
2. **¿Por qué es importante?**
   * Evita recargar la página al cambiar de vista, proporcionando una experiencia similar a las aplicaciones nativas.
   * Facilita la creación de aplicaciones con múltiples vistas, como paneles de administración o tiendas en línea.
3. **Estructura básica de rutas:**
   * Una ruta se compone de:
     + **path:** El segmento de la URL que activa la ruta.
     + **component:** El componente que se mostrará.
   * Ejemplo:  
     const routes: Routes = [

{ path: 'home', component: HomeComponent },

{ path: 'about', component: AboutComponent },

{ path: '', redirectTo: '/home', pathMatch: 'full' },

];

4. **Errores comunes:**

* No definir un redirectTo para rutas vacías.
* Configurar rutas duplicadas.

### Uso del Router Outlet para la navegación de vistas

<router-outlet> es una directiva que actúa como un contenedor de vistas. Cuando se navega a una ruta específica, Angular "inyecta" el componente correspondiente dentro del router-outlet. Esto permite cambiar de vista sin recargar la página.

**¿Cómo se usa <router-outlet>?**

El router-outlet se coloca en el archivo app.component.html o en la vista principal de un módulo.

**Ejemplo:**

<nav>

<a routerLink="/home">Inicio</a>

<a routerLink="/about">Acerca de</a>

</nav>

<router-outlet></router-outlet>

**Puntos clave**:

* Sirve como contenedor de las vistas según la URL.
* Es obligatorio para navegar entre componentes de forma dinámica.

## Métodos forRoot y forChild en RouterModule

En Angular, el sistema de enrutamiento se basa en el módulo RouterModule, el cual permite definir y gestionar rutas. Existen dos métodos principales para configurar rutas:

* **forRoot()**: Se utiliza una sola vez para la configuración inicial de la aplicación.
* **forChild()**: Se usa en módulos secundarios para definir rutas específicas dentro de ese módulo.

Conocer la diferencia entre estos dos métodos es clave para estructurar una aplicación de forma modular y reutilizable.

### ¿Qué es **forRoot()**?

El método forRoot() se usa para configurar el enrutamiento global de la aplicación. Este método se aplica una vez en la raíz del proyecto, generalmente en app-routing.module.ts, y se encarga de crear la instancia principal del enrutador.

**Ejemplo de uso**

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

const routes: Routes = [

{ path: 'home', component: HomeComponent },

{ path: 'about', component: AboutComponent },

{ path: '', redirectTo: '/home', pathMatch: 'full' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

**Puntos clave**:

* Se utiliza **una sola vez** en la aplicación.
* Define las rutas principales de la aplicación.
* Es parte de la configuración inicial del AppRoutingModule.

### ¿Qué es **forChild()**?

El método forChild() se utiliza en módulos secundarios para definir rutas específicas para esos módulos. Es útil para dividir la lógica de la aplicación en **módulos de características (Feature Modules)**.

**Ejemplo de uso**

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { ProfileComponent } from './profile/profile.component';

import { SettingsComponent } from './settings/settings.component';

const routes: Routes = [

{ path: 'profile', component: ProfileComponent },

{ path: 'settings', component: SettingsComponent }

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

export class UserRoutingModule { }

**Puntos clave**:

* Se usa para módulos secundarios o **feature modules**.
* Permite organizar mejor las rutas, evitando que todas estén en un solo archivo.
* Usa RouterModule.forChild(routes) para cargar rutas adicionales.

### forRoot() vs. forChild()

| Aspecto | forRoot() | forChild() |
| --- | --- | --- |
| Uso | Se usa solo una vez | Se usa en módulos secundarios |
| Dónde se usa | AppRoutingModule | Módulos secundarios |
| Instancia de Router | Crea la instancia global | Usa la instancia ya creada |

# 

# Navegación avanzada

## Parámetros en rutas

El Router de Angular permite pasar datos específicos mediante parámetros en las rutas, lo que es útil para personalizar el contenido mostrado en los componentes.

### Conceptos clave:

1. **Parámetros dinámicos en rutas:**
   * Se definen con : en el archivo de rutas.
   * Ejemplo:  
     const routes: Routes = [

{ path: 'product/:id', component: ProductComponent },

];

1. **Acceso a parámetros desde un componente:**

* Usar el servicio ActivatedRoute para obtener los parámetros:  
  constructor(private route: ActivatedRoute) {}

ngOnInit() {

const id = this.route.snapshot.paramMap.get('id');

}

1. **Query Parameters (parámetros opcionales):**

* También se pueden enviar parámetros opcionales utilizando queryParams.
* Ejemplo:  
  this.router.navigate(['/products'], { queryParams: { category: 'books' } });

1. **Errores comunes:**

* Intentar acceder a parámetros sin configurar correctamente las rutas.
* No diferenciar entre parámetros de ruta y parámetros opcionales.

### Rutas con parámetros de navegación avanzada

**Parámetros de ruta vs. Parámetros de consulta**

| **Parámetro** | **Descripción** | **Ejemplo de URL** |
| --- | --- | --- |
| Parámetro de ruta | Se definen en la URL. | /product/42 |
| Parámetro de consulta | Se agregan como query string | /product?category=books |

**Acceder a los parámetros**

1. Parámetros de ruta: Usar this.route.snapshot.paramMap.get('id').
2. Parámetros de consulta: Usar this.route.queryParams.subscribe(params => console.log(params)).

## Rutas hijas y navegación programática

Las rutas hijas (nested routes) y la navegación programática son características avanzadas que permiten estructurar jerárquicamente las vistas y manejar transiciones dinámicas entre ellas.

**Conceptos clave:**

1. **Rutas hijas:**
   * Configuración jerárquica de rutas para vistas relacionadas.
   * Ejemplo:  
     const routes: Routes = [

{

path: 'user',

component: UserComponent,

children: [

{ path: 'profile', component: ProfileComponent },

{ path: 'settings', component: SettingsComponent },

],

},

];

1. Navegación programática:
   * Uso del servicio Router para redirigir dinámicamente a otras vistas:  
     constructor(private router: Router) {}

navigateToProfile() {

this.router.navigate(['/user/profile']);

}

* + Se usa this.router.navigate() para navegar a otra ruta.
  + Los parámetros de la ruta se envían como un array ['/product', id].

1. Errores comunes:
   * Configurar rutas hijas sin definir correctamente el componente principal.
   * Usar navegación programática sin importar el módulo RouterModule.

# Modularización en Angular

## Organización modular

La modularización permite dividir el código de un proyecto en unidades manejables y escalables.

**Conceptos clave:**

1. **¿Qué es un módulo?**

Una clase decorada con @NgModule que agrupa componentes, servicios y directivas relacionadas.

1. **Tipos de módulos**

| **Tipo de módulo** | **Descripción** |
| --- | --- |
| **Root Module** | Configura la aplicación principal |
| **Core Module** | Contiene servicios singleton |
| **Shared Module** | Reutilización de componentes |
| **Feature Module** | Agrupa funcionalidades específicas |

1. **Beneficios de la modularización:**

Mejora la organización y facilita la colaboración en proyectos grandes.

Permite dividir la lógica de la aplicación según sus funcionalidades.

1. **Estructura típica de un módulo:**

Ejemplo de un módulo básico:  
@NgModule({

declarations: [LoginComponent, RegisterComponent],

imports: [CommonModule],

exports: [LoginComponent],

})

export class AuthModule {}

1. **Errores comunes:**
   * No importar CommonModule en módulos secundarios.
   * Duplicar declaraciones de componentes en varios módulos.
2. **Recomendaciones:**
   * Crear un archivo de rutas (RoutingModule) por módulo.
   * Usar forRoot() solo una vez.
   * Usar forChild() para módulos secundarios.
   * No duplicar la declaración de componentes.
   * Importar CommonModule en módulos secundarios, no en el AppModule

## Tipos de módulos: Core, Shared y Feature

### ¿Qué son los Core, Shared y Feature Modules?

En Angular, los módulos (NgModules) permiten organizar una aplicación dividiendo su funcionalidad en bloques reutilizables y gestionables. Un módulo agrupa **componentes, directivas, pipes y servicios** relacionados entre sí. Esta modularización facilita la escalabilidad y el mantenimiento de la aplicación.

Angular proporciona diferentes tipos de módulos según su objetivo y alcance. Los principales son:

* **Core Module**
* **Shared Module**
* **Feature Module**

### ¿Qué es el Core Module?

El **Core Module** contiene los servicios y configuraciones globales que se usan en toda la aplicación, pero que solo se deben crear **una vez**.

**¿Para qué se usa el Core Module?**

* Se encarga de los **servicios singleton** que deben ser accesibles desde cualquier parte de la aplicación.
* **No debe importar componentes**, solo servicios y configuraciones globales.
* Se importa **una sola vez** en AppModule.

**Ejemplo de Core Module**

**core.module.ts**

import { NgModule, Optional, SkipSelf } from '@angular/core';

import { AuthService } from './services/auth.service';

@NgModule({

providers: [AuthService] // Servicios únicos para toda la aplicación

})

export class CoreModule {

constructor(@Optional() @SkipSelf() parentModule: CoreModule) {

if (parentModule) {

throw new Error('CoreModule ya ha sido cargado. Importa este módulo solo en AppModule.');

}

}

}

**app.module.ts**

import { NgModule } from '@angular/core';

import { CoreModule } from './core/core.module';

@NgModule({

imports: [

CoreModule, // Solo se importa una vez

]

})

export class AppModule { }

**Puntos clave**:

* Usa la técnica @Optional() @SkipSelf() para evitar múltiples importaciones del CoreModule.
* Contiene **servicios globales** y no se deben incluir componentes.

### ¿Qué es el Shared Module?

El **Shared Module** contiene **componentes, directivas y pipes reutilizables** que pueden usarse en toda la aplicación. Su objetivo es evitar la duplicación de código.

**¿Para qué se usa el Shared Module?**

* Agrupa **componentes reutilizables** (botones, tarjetas, íconos, etc.).
* Exporta componentes, directivas y pipes para que otros módulos puedan reutilizarlos.
* Se importa en **todos los módulos** que necesiten sus componentes.

**Ejemplo de Shared Module**

**shared.module.ts**

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { ButtonComponent } from './components/button/button.component';

@NgModule({

declarations: [ButtonComponent], // Componentes reutilizables

imports: [CommonModule],

exports: [ButtonComponent] // Exporta para que esté disponible en otros módulos

})

export class SharedModule { }

**user.module.ts**

import { NgModule } from '@angular/core';

import { SharedModule } from '../shared/shared.module';

@NgModule({

imports: [

SharedModule // Se importa para reutilizar ButtonComponent

]

})

export class UserModule { }

**Puntos clave**:

* Contiene **componentes, directivas y pipes reutilizables**.
* Se importa **varias veces** en los módulos que necesiten sus componentes.
* No contiene servicios ni lógica global de la aplicación.

### ¿Qué es el Feature Module?

El **Feature Module** se encarga de encapsular una funcionalidad específica de la aplicación, como un módulo de "Usuarios", "Productos" o "Dashboard".

**¿Para qué se usa el Feature Module?**

* Divide la aplicación en **funcionalidades independientes**.
* Cada módulo tiene su propio archivo de rutas con RouterModule.forChild().
* Se pueden **cargar de forma perezosa (lazy loading)** para optimizar la aplicación.

**Ejemplo de Feature Module**

**user.module.ts**

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { UserListComponent } from './user-list/user-list.component';

import { UserDetailComponent } from './user-detail/user-detail.component';

import { UserRoutingModule } from './user-routing.module';

@NgModule({

declarations: [

UserListComponent,

UserDetailComponent

],

imports: [

CommonModule,

UserRoutingModule // Se usa forChild() en este módulo

]

})

export class UserModule { }

**user-routing.module.ts**

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { UserListComponent } from './user-list/user-list.component';

import { UserDetailComponent } from './user-detail/user-detail.component';

const routes: Routes = [

{ path: 'users', component: UserListComponent },

{ path: 'users/:id', component: UserDetailComponent }

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

export class UserRoutingModule { }

**Puntos clave**:

* Cada **Feature Module** encapsula una funcionalidad específica.
* Se usa **RouterModule.forChild()** para definir rutas propias.
* Se puede cargar de forma **perezosa (lazy loading)**.

### Core vs. Shared vs Feature Modules

| **Aspecto** | **Core Module** | **Shared Module** | **Feature Module** |
| --- | --- | --- | --- |
| **Contenido** | Servicios únicos | Componentes reutilizables | Vistas y funcionalidad |
| **Uso** | Importado una vez | Importado en varios módulos | Se carga por funcionalida |
| **Componentes** | ❌ No | ✅ Sí | ✅ Sí |
| **Servicios** | ✅ Sí (Singleton) | ❌ No | ✅ Sí (locales) |
| **Rutas** | ❌ No | ❌ No | ✅ Sí (forChild) |
| **Ejemplo** | AuthService | ButtonComponent | UserModule, AdminModule |

Los módulos **Core, Shared y Feature** permiten organizar de forma eficiente y escalable una aplicación Angular. Cada uno tiene una responsabilidad específica. Entender sus diferencias ayuda a crear aplicaciones más **modulares, mantenibles y escalables**.

## Módulos compartidos

Los módulos compartidos son útiles para reutilizar componentes, directivas y servicios en diferentes partes de la aplicación.

**Conceptos clave:**

1. **¿Qué es un módulo compartido?**
   * Un módulo que exporta elementos comunes para ser utilizados en otros módulos.
   * Ejemplo:

@NgModule({

declarations: [SharedButtonComponent],

exports: [SharedButtonComponent],

})

export class SharedModule {}

1. **Buenas prácticas:**
   * Exportar módulos o componentes innecesarios.
   * No actualizar el módulo compartido al añadir nuevos elementos.

# Actividades para la clase práctica en vivo- Unidad 6

## Actividad 1. Creación de componentes y rutas

**Duración**: 20 minutos

**Objetivo**: Crear una aplicación con componentes y rutas.

**Instrucciones**:

1. Crear una aplicación con Angular CLI.
2. Generar dos componentes: **ComponenteA** y **ComponenteB**.
3. Añadir un archivo de rutas para redirigir a estos componentes.
4. Usar una barra de navegación con dos botones para redirigir:
   * **Botón 1**: Redirige utilizando el routerLink en HTML.
   * **Botón 2**: Redirige mediante una función en **app.component.ts**.

**Resolución**:  
  
// app-routing.module.ts

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { ComponenteAComponent } from './componente-a/componente-a.component';

import { ComponenteBComponent } from './componente-b/componente-b.component';

const routes: Routes = [

{ path: 'componenteA', component: ComponenteAComponent },

{ path: 'componenteB', component: ComponenteBComponent }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule {}

// app.component.ts

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

import { Router } from '@angular/router';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

constructor(private router: Router) {}

navegarAComponenteB() {

this.router.navigate(['/componenteB']);

}

}

<!-- app.component.html -->

<button routerLink="/componenteA">Ir a Componente A</button>

<button (click)="navegarAComponenteB()">Ir a Componente B</button>

## Actividad 2. Ruteo con parámetros

**Duración**: 15 minutos

**Objetivo**: Crear un ruteo con parámetros para mostrar detalles en un componente.

**Instrucciones**:

* 1. Crear un componente **UsuarioComponent**.
  2. Recibir un parámetro **id** de la ruta.
  3. Mostrar el **nombre** y **apellidos** recibidos a través del cuerpo de la ruta.

**Resolución**:  
  
// usuario.component.ts

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

import { ActivatedRoute } from '@angular/router';

@Component({

selector: 'app-usuario',

templateUrl: './usuario.component.html',

styleUrls: ['./usuario.component.css']

})

export class UsuarioComponent implements OnInit {

id!: string;

nombre!: string;

apellidos!: string;

constructor(private route: ActivatedRoute) {}

ngOnInit(): void {

this.id = this.route.snapshot.paramMap.get('id')!;

this.nombre = this.route.snapshot.queryParamMap.get('nombre')!;

this.apellidos = this.route.snapshot.queryParamMap.get('apellidos')!;

}

}

<!-- usuario.component.html -->

<h1>Detalle del Usuario</h1>

<p>ID: {{ id }}</p>

<p>Nombre: {{ nombre }}</p>

<p>Apellidos: {{ apellidos }}</p>

## Actividad 3. Creación y uso de un módulo

**Duración**: 15 minutos

**Objetivo**: Crear y utilizar un módulo.

**Instrucciones**:

* 1. Crear un módulo llamado **UsuarioModule**.
  2. Importar el módulo y utilizarlo en la aplicación.

**Resolución**:  
  
// usuario.module.ts

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { UsuarioComponent } from './usuario/usuario.component';

@NgModule({

declarations: [UsuarioComponent],

imports: [CommonModule],

exports: [UsuarioComponent]

})

export class UsuarioModule {}

// app.module.ts

import { UsuarioModule } from './usuario/usuario.module';

@NgModule({

declarations: [...],

imports: [BrowserModule, AppRoutingModule, UsuarioModule],

bootstrap: [AppComponent]

})

export class AppModule {}

## Actividad 4. Creación de una aplicación con Core & Shared Modules

**Duración**: 20 minutos

**Objetivo**: Crear una aplicación con **CoreModule** y **SharedModule**.

**Instrucciones**:

* 1. Crear un módulo de administración con un componente de **login**.
  2. Añadir un **SharedModule** asociado al módulo.
  3. Utilizar **CoreModule** para elementos compartidos globalmente.

**Resolución**:  
  
// core.module.ts

import { NgModule } from '@angular/core';

import { HttpClientModule } from '@angular/common/http';

@NgModule({

imports: [HttpClientModule],

exports: [HttpClientModule]

})

export class CoreModule {}

// shared.module.ts

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

@NgModule({

imports: [CommonModule],

exports: [CommonModule]

})

export class SharedModule {}

## Actividad 5. Creación de un Feature Module

**Duración**: 20 minutos

**Objetivo**: Agregar un **Feature Module** a la aplicación.

**Instrucciones**:

* 1. Crear un **FeatureModule** de **Registro**.
  2. Utilizar **CoreModule** y **SharedModule** en este módulo.

**Resolución**:  
  
// registro.module.ts

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { RegistroComponent } from './registro/registro.component';

import { SharedModule } from '../shared/shared.module';

import { CoreModule } from '../core/core.module';

@NgModule({

declarations: [RegistroComponent],

imports: [CommonModule, SharedModule, CoreModule],

exports: [RegistroComponent]

})

export class RegistroModule {}

// app.module.ts

import { RegistroModule } from './registro/registro.module';

@NgModule({

declarations: [...],

imports: [BrowserModule, AppRoutingModule, RegistroModule],

bootstrap: [AppComponent]

})

export class AppModule {}