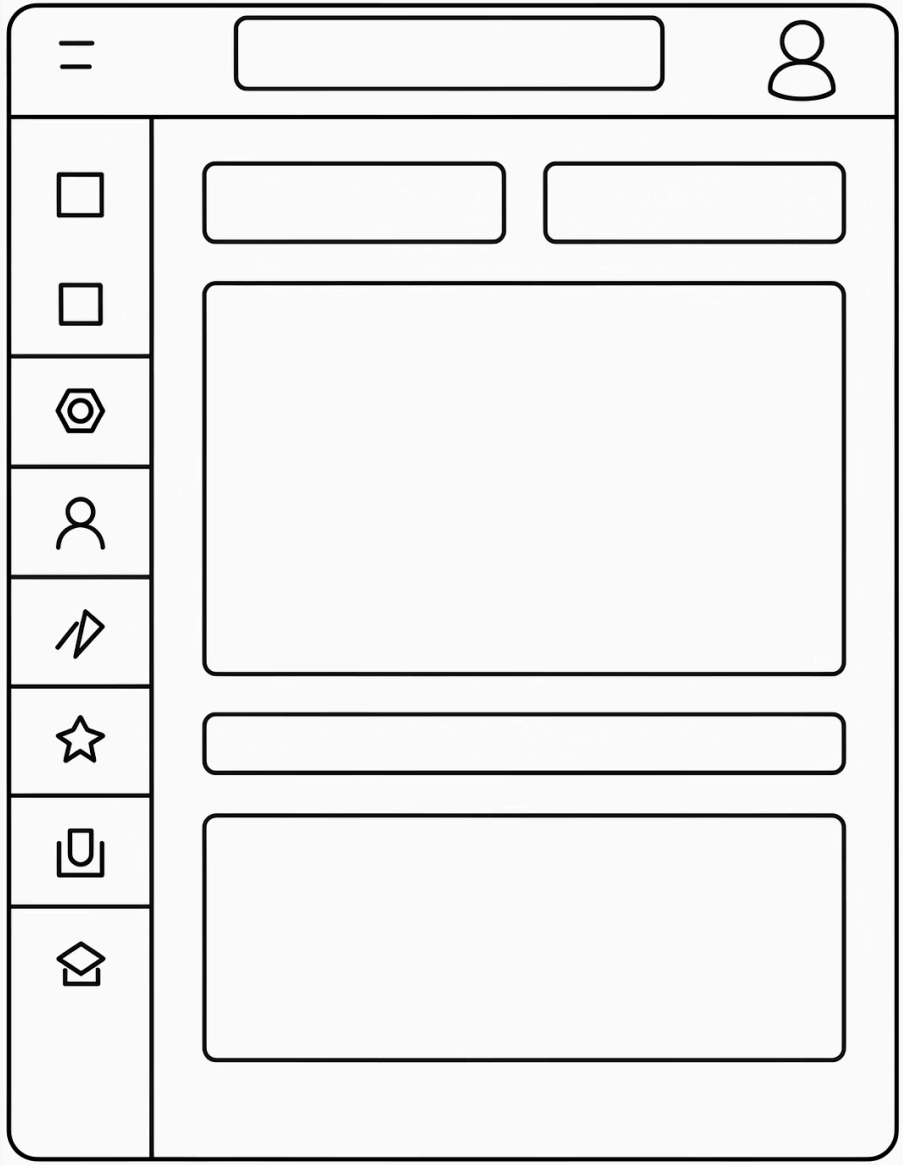
Rutas y navegación en Angular

Una guía práctica para entender y usar el sistema de routing en aplicaciones Angular

TEMA 8

2º DAM

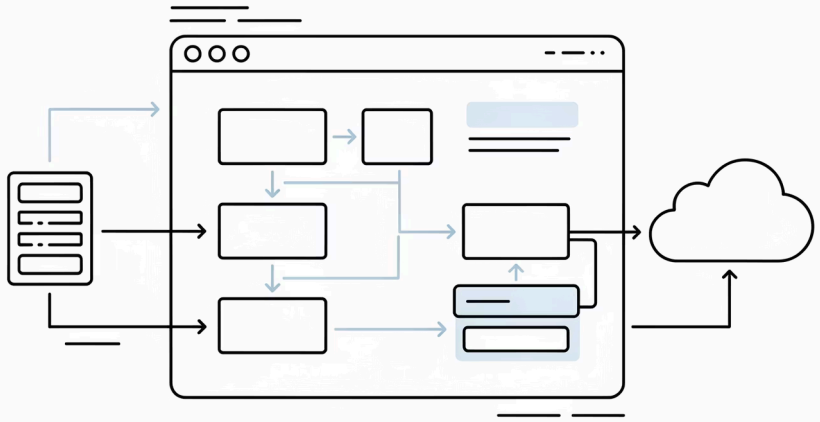


INTRODUCCIÓN

¿Qué es el routing y por qué lo necesitamos?

El **routing de Angular** es la capacidad de navegar entre diferentes vistas o componentes dentro de una misma aplicación web sin tener que recargar la página completa. Permite una experiencia de usuario fluida, similar a la de una aplicación de escritorio.

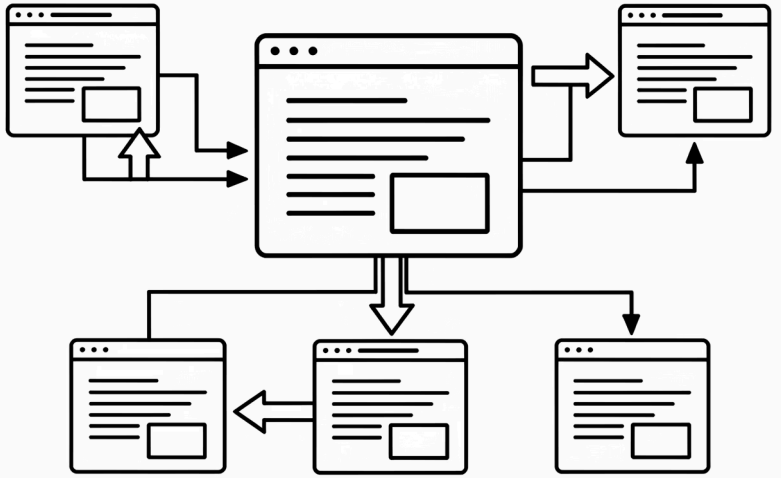
Esto es fundamental para construir **Single Page Applications (SPA)**, donde toda la aplicación se carga una sola vez y solo las partes necesarias se actualizan dinámicamente, mejorando significativamente el rendimiento y la interacción del usuario.



¿Qué pasa sin routing?

Sin un sistema de routing eficiente, las aplicaciones web tradicionales enfrentan varias limitaciones:

- Necesitas crear múltiples archivos HTML separados para cada página
- Cada navegación requiere recargar toda la página desde el servidor
- Pérdida de estado: los datos en memoria se pierden con cada recarga
- Experiencia de usuario más lenta y menos fluida
- Mayor consumo de ancho de banda al recargar recursos repetidos (CSS, JS, imágenes)
- Más difícil mantener una experiencia coherente entre páginas



Ventajas del routing en Angular

Navegación instantánea

Los usuarios cambian de vista sin esperar a que se recargue toda la página

URLs navegables

Cada vista tiene su propia URL que se puede compartir o guardar como favorito

Historial del navegador

Los botones de adelante y atrás funcionan correctamente dentro de la aplicación

Control de acceso

Podemos proteger ciertas rutas para que solo usuarios autorizados accedan

FUNCIONAMIENTO

¿Cómo funciona el routing en Angular?

El sistema de routing de Angular sigue un flujo bien definido para mostrar el contenido correcto al usuario:

1. URL

El usuario escribe una URL o hace clic en un enlace (ejemplo: /productos).

2. Router

Angular Router intercepta la navegación y busca la ruta coincidente en el array de `Routes`.

3. Componente

El Router identifica qué componente debe cargar según la configuración de rutas.

4. <router-outlet>

El componente se renderiza dentro del `<router-outlet>` en el template principal de la aplicación.

Ejemplo práctico de configuración y flujo:

```
const routes: Routes = [
  { path: 'productos', component: ProductosComponent }
];
```

Cuando el usuario navega a `/productos` → `ProductosComponent` se carga en `<router-outlet>`.

CONFIGURACIÓN INICIAL

Instalación del RouterModule

Angular CLI nos facilita el trabajo. Si creamos un proyecto nuevo y respondemos "Sí" cuando pregunta por el routing, Angular configura todo automáticamente. Pero veamos qué hace por nosotros:

01

Crear el archivo de rutas

Se genera `app-routing.module.ts` que contendrá todas nuestras rutas

02

Importar RouterModule

Angular añade el módulo de routing al módulo principal de la aplicación

03

Añadir router-outlet

Se incluye la etiqueta especial donde se cargarán los componentes

Consejo: Si tu proyecto ya existe y no tiene routing, puedes añadirlo ejecutando: ng generate module app-routing --flat --module=app

Estructura básica del archivo de rutas

El archivo `app-routing.module.ts` es el corazón del sistema de navegación. Aquí definimos qué componente se muestra para cada URL.

La estructura es sencilla: un array de objetos donde cada objeto representa una ruta con su **path** (la URL) y su **component** (el componente que se carga).

```
const routes: Routes = [
 { path: '', component: InicioComponent },
 { path: 'productos', component: ProductosComponent },
 { path: 'contacto', component: ContactoComponent }
];
```

¿Dónde vive el routing?

app-routing.module.ts

Aquí se definen todas las rutas (el array de `Routes` con `path` y `component`).

app.module.ts

Aquí se importa el `AppRoutingModule` para activar el routing en la aplicación.

app.component.html

Aquí se coloca el `<router-outlet>` donde se cargarán los componentes.

El router-outlet: donde ocurre la magia

¿Qué es?

Es una directiva especial de Angular que actúa como un marcador de posición. Le indica a Angular: "aquí es donde debes cargar los componentes según la ruta activa".

¿Dónde se coloca?

Normalmente en el template de `app.component.html`, justo donde queremos que aparezca el contenido dinámico de cada vista.

Ejemplo práctico

```
<nav>
  <a routerLink="/">Inicio</a>
  <a
routerLink="/productos">Productos<
/a>
</nav>
<router-outlet></router-outlet>
```

Cuando el usuario hace clic en un enlace, Angular carga el componente correspondiente dentro del `<router-outlet>`, manteniendo intacto el resto de la página (como el menú de navegación).

NAVEGACIÓN PRÁCTICA

Dos formas de navegar: routerLink y Router

routerLink (en el template)

La forma más común y sencilla. Se usa directamente en las etiquetas HTML del template:

```
<a routerLink="/productos">
  Ver productos
</a>

<button routerLink="/contacto">
  Contactar
</button>
```

Angular se encarga de todo: actualiza la URL, carga el componente y mantiene el historial del navegador.

Router (en el TypeScript)

Para navegaciones programáticas, cuando necesitamos redirigir después de una acción:

```
constructor(
  private router: Router
) {}

guardarDatos() {
  // lógica de guardado
  this.router.navigate(['/exito']);
}
```

Útil para redirigir tras validar un formulario, hacer login, etc.

routerLink vs Router.navigate

routerLink

Uso: En el template (HTML)

Tipo: Directiva declarativa

Cuándo usarlo: Para enlaces y botones que el usuario hace clic

```
<a routerLink="/productos">Productos</a>
<button routerLink="/contacto">Contactar</button>
```

Ventaja: Más simple y directo

Router.navigate

Uso: En el componente (TypeScript)

Tipo: Método programático

Cuándo usarlo: Para navegación después de lógica (validaciones, guardado, etc.)

```
constructor(private router: Router) {}

login() {
  if (this.validar()) {
    this.router.navigate(['/dashboard']);
  }
}
```

Ventaja: Más control y flexibilidad

Enlaces en menú	routerLink
Tras login	Router.navigate
Tras formulario	Router.navigate

Rutas con parámetros

A menudo necesitamos pasar información en la URL. Por ejemplo, para mostrar los detalles de un producto específico, necesitamos su ID. Angular permite crear rutas dinámicas con parámetros:

¿Para qué sirven los parámetros?

Los parámetros de ruta son esenciales para construir aplicaciones dinámicas y escalables, permitiendo:

- Mostrar detalles específicos de un elemento (producto, usuario, artículo) sin necesidad de una ruta fija para cada uno.
- Evitar crear una ruta diferente para cada elemento individual.
- Hacer las rutas dinámicas y reutilizables en toda la aplicación.

Por ejemplo, en lugar de crear `/producto1`, `/producto2`, `/producto3`... usamos una única ruta dinámica como `/producto/:id` para acceder a cualquier producto.

Cómo usar parámetros en rutas

1. Definir la ruta

```
{ path: 'producto/:id', component: DetalleComponent }
```

Los dos puntos indican que `id` es un parámetro variable

2. Navegar con parámetros

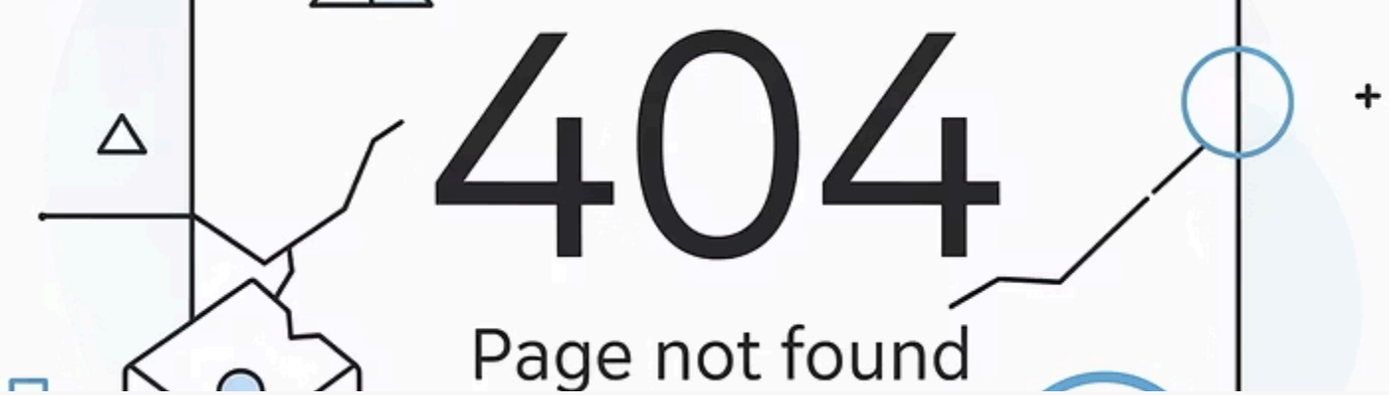
```
<a routerLink="/producto/42">Ver producto</a>
```

O desde TypeScript: `this.router.navigate(['/producto', 42])`

3. Leer el parámetro

```
constructor(private route: ActivatedRoute) {}

ngOnInit() {
  this.id = this.route.snapshot.paramMap.get('id');
}
```



Ruta comodín y redirecciones

Página 404 personalizada

Cuando un usuario escribe una URL que no existe, podemos mostrar un componente de error personalizado:

```
{
  path: '**',
  component: PaginaNoEncontradaComponent
}
```

Importante: Esta ruta debe ir siempre al final del array, ya que Angular evalúa las rutas en orden y `**` coincide con cualquier URL.

Redirecciones automáticas

Podemos redirigir automáticamente de una ruta a otra. Es útil para la ruta raíz o para URLs antiguas:

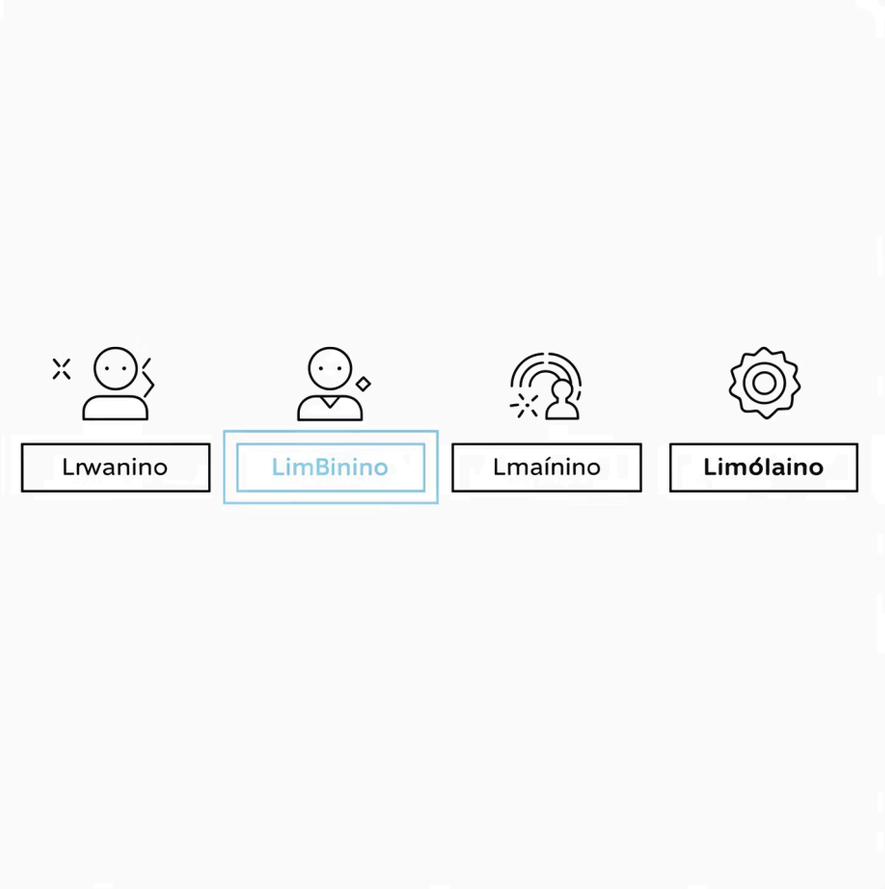
```
{
  path: '',
  redirectTo: '/inicio',
  pathMatch: 'full'
}
```

El `pathMatch: 'full'` asegura que solo redirija cuando la URL coincida exactamente (vacía en este caso).

routerLinkActive: resaltar la ruta activa

Es muy útil mostrar visualmente en qué página está el usuario. La directiva `routerLinkActive` añade automáticamente una clase CSS al enlace activo:

```
<nav>
  <a routerLink="/inicio"
     routerLinkActive="activo">
    Inicio
  </a>
  <a routerLink="/productos"
     routerLinkActive="activo">
    Productos
  </a>
  <a routerLink="/contacto"
     routerLinkActive="activo">
    Contacto
  </a>
</nav>
```



Después, en el CSS, defines el estilo para la clase `.activo` (por ejemplo, un color diferente o un borde inferior).

Para la ruta raíz `/`, añade `[routerLinkActiveOptions]="{exact: true}"` para que solo se active con coincidencia exacta.

Recapitulación: conceptos clave

1

Router Module

El módulo de Angular que gestiona toda la navegación de la aplicación

2

Routes array

Array de objetos que mapea URLs a componentes específicos

3

router-outlet

Directiva que marca dónde se cargarán los componentes según la ruta activa

4

routerLink

Directiva para crear enlaces de navegación en los templates HTML

5

ActivatedRoute

Servicio para acceder a información sobre la ruta actual, como parámetros

Con estos elementos ya puedes crear aplicaciones Angular con navegación completa y profesional. ¡Ahora toca practicar!

Recursos utilizados y para profundizar

Explore estos recursos clave para profundizar en el routing de Angular:

Guía oficial de Angular Router

La **documentación oficial** es el mejor punto de partida para entender a fondo la funcionalidad.

Angular University

En **Angular University (YouTube)** encontrarás tutoriales en vídeo y explicaciones prácticas sobre el ecosistema de Angular, incluyendo routing.

Ultimate Courses - Angular Router Tutorial

Ultimate Courses ofrece artículos detallados con ejemplos de código y buenas prácticas.

Stack Overflow

Explora **Stack Overflow**, una comunidad activa para encontrar soluciones a problemas comunes y aprender de otros desarrolladores.