











Angular 17

Tema 9. Rutas

Objetivos

-  Aprender el manejo de rutas
-  Uso de rutas desde TypeScript

Contenidos

-  Configuración. Interfaz Route
-  Rutas hijas
-  Carga diferida mediante rutas
-  «router-outlet» auxiliares
-  Envío de datos y gestión de rutas
-  Introducción a «guards»

Configuración de Rutas

Se configura como un servicio en «appConfig», con el método **provideRouter()**:

```
export const appConfig: ApplicationConfig = {  
  providers: [provideRouter(rutas)]  
};
```

Las «rutas» son un array de objetos que implementan la interfaz **Route**:

```
export const rutas: Routes = [  
  {path: 'rojo', component: RojoComponent},  
  ...  
];
```

Se usan junto a las directivas **router-outlet** y **routerLink**:

```
<ul>  
  <li><a routerLink="rojo">Rojo</a></li>  
  ...  
</ul>  
  
<article>  
  <router-outlet></router-outlet>  
</article>
```

Interfaz Route

La interfaz **Route** permite configurar varios campos:

path	La URL de la ruta. En la definición la ruta no puede comenzar por '/'. Cuidado, el orden importa.
component	El componente que se dibujará en «router-outlet».
redirectTo / pathMatch	Redirección a una ruta distinta.
title	Título de la lengüeta del navegador.

```
export const rutas: Routes = [  
  {path:'rojo', component:RojoComponent, title:'color rojo'},  
  {path:'color/verde', component:VerdeComponent, title:'color verde'},  
  {path:'', redirectTo:'/rojo', pathMatch:'full'},  
  {path:'**', component:NoEncontradoComponent, title:'404'}  
];
```

children	Definición de rutas hijas.
loadComponent	Carga diferida del componente
outlet	«router-outlet» auxiliares.
data, 'url/:datos'	Paso de datos adicionales.
canActivate...	Definiciones de «guards»

Rutas hijas (children)

El objetivo es la creación de componentes que se dibujen dentro de otros, pero seleccionándolos a través de rutas:

```
{path:'naranja', component:NaranjaComponent, children:[
  {path:'rojo', component:RojoComponent, title:'naranja - rojo'},
  {path:'verde', component:VerdeComponent, title:'naranja - verde'},
  {path:'azul', component:AzulComponent, title:'naranja - azul'}
]}
```

Para que funcione, el componente padre debe contener un «router-outlet»:

```
<h1>Soy el componente naranja</h1>
<router-outlet></router-outlet>
```

Se seleccionan uniendo los «path»:

```
<li><a routerLink="/naranja/rojo">Rojo</a></li>
```

Permiten definir una **jerarquía lógica** entre los componentes, por ejemplo:

- Secciones (componentes) principales y secundarios
- Modularidad de la aplicación, agrupando funcionalidades
- Navegabilidad, dividiendo rutas complejas en otras más simples

Carga diferida de componentes. LoadComponent

Las rutas permiten que un componente pueda cargarse de forma diferida. Hasta que no se utilice su ruta, Angular no lo añade al código de JavaScript del navegador cliente:

```
{path: 'negro', loadComponent: () => import('./varios/negro/negro.component').then(c => c.NegroComponent)}.
```

Afecta al **rendimiento**, no al comportamiento de la aplicación.

Angular 17 dispone de **@defer**, mucho más versátil (tema 4, «Plantillas y Data Binding»)

Si vas a usar aplicaciones tradicionales, hay un concepto equivalente aplicable a módulos, «loadChildren». No vamos a verlo en este curso.

Rutas auxiliares

En un mismo componente se pueden definir varios «router-outlet»:

```
<article>
  <router-outlet></router-outlet>
</article>
<aside>
  <router-outlet name="secundario"></router-outlet>
</aside>
```

Si se quieren definir rutas para los «outlet auxiliares» (los que no se llaman «primary») hay que indicarlo en la ruta con **outlet**:

```
{path:'rojo-sec', component:RojoComponent, outlet:'secundario'}.
```

Y los enlaces se definen de forma distinta:

```
<li><a [routerLink]="[{outlets:{secundario:'rojo-sec'}}]">Rojo en auxiliar</a></li>
<li><a [routerLink]="[{outlets:{secundario:'azul-sec'}}]">Azul en auxiliar</a></li>
<li><a [routerLink]="[{outlets:{secundario:'verde-sec', 'primary':'negro'}}]">Negro en principal y verde en auxiliar</a></li>
```

Permite una sintaxis especial para enlaces «normales» o para la barra de direcciones del navegador:

enlace_principal(nombre_outlet:enlace_auxiliar)

```
localhost:4200/verde(secundario:azul-sec)
```


Servicios de rutas

Disponemos de varios servicios que podemos inyectar en nuestros componentes para gestionar las rutas desde TypeScript

Router Permite inspeccionar o modificar la ruta actual.

navigate(), navigateByUrl(), url...

```
private router=inject(Router);  
...  
this.router.navigate([{outlets:{secundario:'rojo-sec', primary:'black'}}]);  
this.router.navigateByUrl('/black(secundario:rojo-sec)');
```

ActivatedRoute Información sobre la ruta actual. Cuidado, casi toda la información se muestra mediante observables.

snapshot(), title, url, paramMap, queryParamsMap, data, outlet...

```
private activatedRoute=inject(ActivatedRoute);  
...  
//Mmm, cuidado  
console.log('La ruta es ' + this.activatedRoute.snapshot.url);  
  
this.activatedRoute.url.subscribe(ruta=>console.log('La ruta es ' + ruta));
```

Envío de datos

Podemos asociar **datos estáticos** a la ruta con **data**:

```
{path:'verde-uno', component:VerdeComponent, data:{identificador:'primero'}},
{path:'azul-dos', component:AzulComponent, data:{identificador:'segundo', tipo:'ejemplo'}},
{path:'azul-tres', component:AzulComponent, data:{identificador:'tercero'}}
```

Se leen desde el servidor con **ActivatedRoute.data**:

```
this.ar.data.subscribe(datos=>{
  for (let campo in datos) {
    if (this.texto!='') this.texto+=' / ';
    this.texto+=campo + '=' + datos[campo];
  }
});
```

Otra posibilidad es usar **parte de la ruta** para definir los datos, por ejemplo «/producto/42»:

```
{path:'negro/:valor', component:NegroComponent, title:'valores en la ruta'}
```

Podemos definir las rutas de la manera estándar o usando la sintaxis de directivas de «RouterLink»:

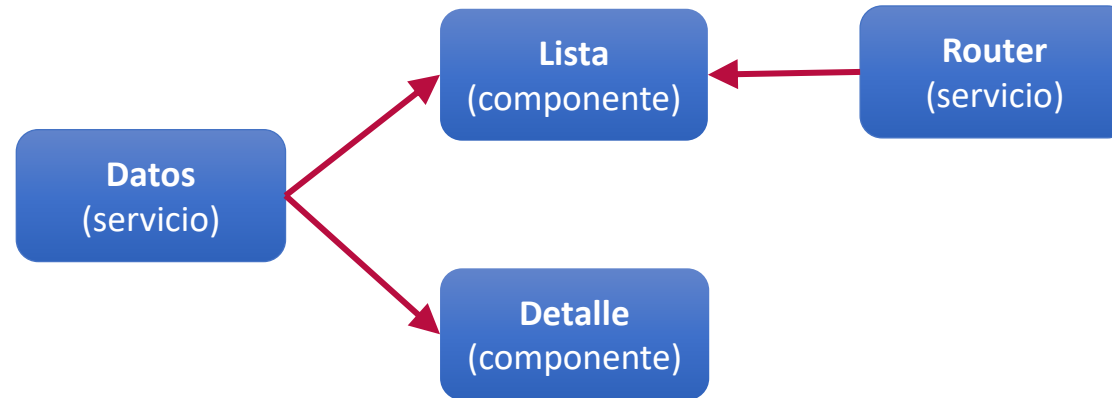
```
<li><a routerLink="/negro/saludo">Negro saludo</a></li>
<li><a routerLink="/negro/100">Negro 100</a></li>
<li><a [routerLink]="['negro',{valor:'prueba'}]">negro prueba</a></li>
```

Y se lee desde el componente con **ActivatedRoute.paramMap**:

```
mensaje:string | null=null;
private ar=inject(ActivatedRoute);
...
this.ar.paramMap.subscribe(p=>this.mensaje=p.get('valor'));
```

Ejercicio 09 A (planteamiento)

Quiero que escribas la típica aplicación que a partir de una lista muestra el detalle seleccionado, pero usando **rutas auxiliares**, en vez de comunicación entre componentes con «@Input» y «@Output».



La instrucción que deberás ejecutar en el componente «Lista» será similar a ésta:

```
this.router.navigate([outlets:{NOMBRE-OUTLET-DETALLE:'NOMBRE-RUTA-DETALLE/' + ID-DEL-DETALLE}]]);
```

```
this.router.navigate([outlets:{derecha:'detalle-producto/' + id}]]);
```

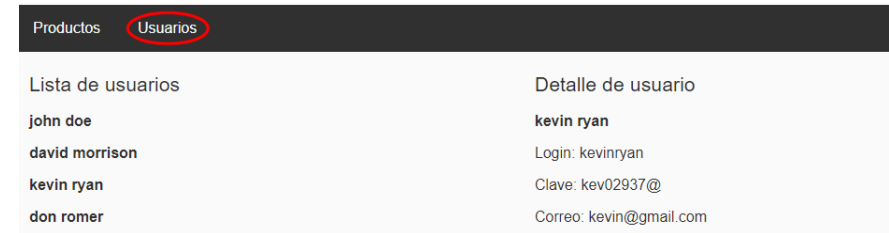
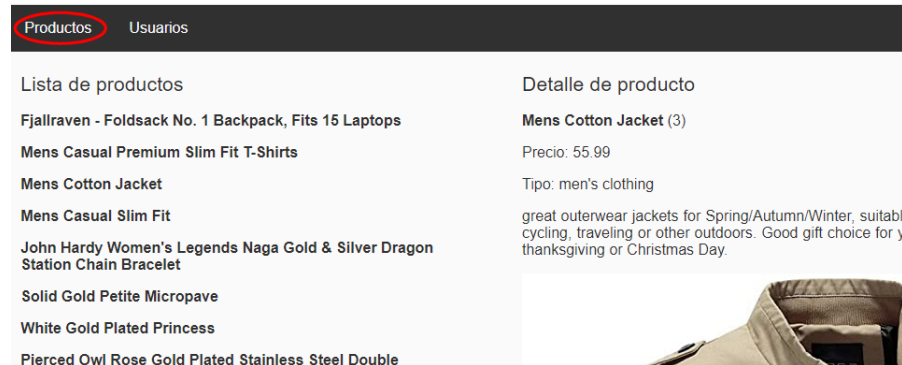
Se supone que tendrás un par de «router-outlets» disponibles:

```

<article>
  <section id="lista">
    <router-outlet></router-outlet>
  </section>
  <section id="detalle">
    <router-outlet name="derecha"></router-outlet>
  </section>
</article>
  
```

Ejercicio 09 A

El programa mostrará **dos parejas** de lista/detalle, «<https://fakestoreapi.com/users>» y «<https://fakestoreapi.com/products>»:



Definirás «lista», «detalle» y un servicio de datos tanto para «products» como para «users». El modelo de datos quiero que sea éste:

```
export interface Nombre {
  firstname: string,
  lastname: string
}
export interface Usuario {
  id: number,
  email: string,
  username: string,
  password: string,
  name: Nombre
}
export interface ResumenUsuario {
  id: number,
  name: Nombre
}
```

```
export interface Producto {
  id: number,
  title: string,
  price: number,
  description: string,
  category: string,
  image: string
}
export interface ResumenProducto {
  id: number,
  title: string
}
```

Introducción a guards

Funciones que permiten o no el acceso a una ruta

No son un sistema de seguridad. El código fuente de JavaScript se ejecutan en el cliente.

Tradicionalmente se definían mediante servicios que implementaban ciertas interfaces, pero ahora son «simples» funciones cuyos parámetros y tipos de retorno dependen del tipo de «guard» definido en la ruta.

La función más simple devuelve «true» o «false», sin parámetros.

Existen cuatro tipos de guards, con su correspondiente campo en «Route»:

canActivate	La ruta se puede utilizar.
canDeactivate	La ruta se puede abandonar (usando «Router» o «RouterLink»).
canActivateChild	Se pueden usar las rutas hijas.
canMatch	Si se puede hacer coincidir la solicitud actual con la ruta. Usado para prevenir la carga diferida de un componente cuando no se tiene acceso al mismo.

```
const f1={()=>Math.random()>0.5;
const f2={()=>inject(PruebasService).siempreTrue();
const f3={()=>inject(PruebasService).siempreFalse();

export const routes: Routes = [
  {path:'uno', component:UnoComponent},
  {path:'dos', component:DosComponent, canActivate:[f1,f2]},
  {path:'tres', component:TresComponent, canActivate:[f1,f2], canDeactivate:[f3]},

  {path:'', redirectTo:'/uno', pathMatch:'full'},
  {path:'**', component:NoEncontradoComponent}
];
```

Introducción a guards. Referencia

Las firmas (simplificadas) de las funciones son éstas:

CanActivate (actual: ActivatedRouteSnapShot, estado: RouterStateSnapshot)=>boolean | UrlTree

CanDeactivate<T> (componente: T, actual: ActivatedRouteSnapShot , estadoActual: RouterStateSnapshot , estadoSiguiente: RouterStateSnapshot)=>boolean | UrlTree

NombreDeClase**Snapshot** son versiones **síncronas** de las clases normales.

RouterState Es similar a «ActivatedRoute», pero nos da información de **todas** las rutas activas (nodos de rutas hijas, rutas auxiliares)

UrlTree es simplemente la URL, pero procesada en trozos (raíz, segmento final, parámetros...)

Introducción a guards. Técnicas

Se suelen definir con métodos estáticos (recuerda las validaciones manuales en formularios reactivos):

```
export const routes: Routes = [
  {path:'uno', component:UnoComponent, canActivate:[
    Acceso.aleatorio(0.5, true),
    Acceso.comprobacionCredenciales()
  ]},
  ...
];
```

```
export class Acceso {
  static aleatorio(cantidad:number, respuesta:boolean) {
    return ()=>{
      if (Math.random()>=cantidad) return respuesta;
      else return !respuesta;
    }
  }
  static comprobacionCredenciales() {
    const datos=inject(PruebasService);
    return datos.simulacionDeComporbacion();
  }
}
```

En muchos ejemplos verás el empleo de «function»:

```
export const routes: Routes = [
  {path:'uno', component:UnoComponent, canActivate:[
    aleatorioV2(0.5, true),
    Acceso.comprobacionCredenciales()
  ]},
  ...
];
```

```
function aleatorioV2(cantidad:number, respuesta:boolean) {
  return ()=>{
    if (Math.random()>=cantidad) return respuesta;
    else return !respuesta;
  }
}
```

¿Qué hemos aprendido?

- Campos de la interfaz Route y configuración de rutas
- Gestión de rutas en TypeScript: ActivatedRoute, Router
- Router-outlet auxiliares y anidados
- Introducción a guards

Resumen de comandos

npm

npm install [-g] paquete
npm install
npm uninstall paquete
npm -version

ng (1)

ng version
ng help
ng serve [-o]
ng build
ng new nombre_proyecto
 --routing false
 --skip-tests
 --skip-git
 --no-standalone

ng (2)

ng generate component nombre_componente / ng g c
ng generate service nombre_servicio / ng g s
ng generate module nombre_módulo / ng g m

Ejercicio 09 B

Cada uno en su sitio

Quiero que realices algunas pruebas con las rutas. Define tres «router-outlet» distintos y crea un menú que dibuje «perro», «gato» y «jirafa» en el de la izquierda, el centro y la derecha respectivamente.

Para hacer las pruebas te vendrá bien «limpiar todo». Como no he definido rutas por defecto, «» borra todo.

Experimentos

El primer enlace dibuja un cuarto componente a la izquierda, con un botón y un formulario. El botón hace que aparezcan de golpe perro, gato y jirafa. El formulario dibuja un componente a la derecha que muestra el «código de producto» que se supone buscaría.

La segunda opción del menú dibuja de nuevo el gato, pero «de algún modo» aparece con un mensaje adicional.

