Clase 5

Rutas en Angular con RouterModule

RouterModule

- Sistema de navegación que convierte aplicaciones de una sola página en experiencias multi-página
- Permite navegación sin recargas (SPA)
- Gestiona el estado de la aplicación a través de URLs

Navegación Sin Recargas (SPA)

RouterModule permite que los usuarios naveguen por diferentes vistas sin recargar la página completa:

- **Experiencia fluida**: Similar a aplicaciones nativas
- **Rendimiento optimizado**: No se descarga toda la página
- **Transiciones suaves**: Animaciones entre vistas
- **Estado preservado**: Los datos en memoria se mantienen

RouterOutlet

```
<div class="app">
  <app-header></app-header>

  <!-- Aquí Angular renderiza el componente de la ruta activa -->
  <router-outlet></router-outlet>
</div>
```

`<router-outlet>` actúa como un contenedor dinámico donde se renderizan los componentes según la ruta actual.

URL Como Estado de la Aplicación

Las URLs representan el estado actual de la aplicación:

- **Route Parameters**: `/team/123` → ID del miembro
- **Query Parameters**: `/dashboard?filter=available` → Filtros activos
- **Fragments**: `/dashboard#statistics` → Sección específica

Configuración Básica de Rutas

```
// app.routes.ts
import { Routes } from '@angular/router'

export const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'team/:id', component: TeamDetailComponent },
  { path: 'settings', component: SettingsComponent },
  { path: '**', component: NotFoundComponent },
]
```

Navegación Declarativa

```
<!-- Navegación básica -->
<a routerLink="/dashboard">Dashboard</a>

<!-- Con parámetros -->
<a routerLink="/team/{{member.id}}">Ver Perfil</a>

<!-- Con estilos activos -->
<a routerLink="/dashboard" routerLinkActive="active">Dashboard</a>
```

routerLinkActive aplica clases CSS automáticamente cuando la ruta está activa.

Parámetros de Ruta

```
// En el componente
export class TeamDetailComponent implements OnInit {
  member: TeamMember | null = null

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.paramMap.subscribe((params) => {
      const id = Number(params.get('id'))
      // Usar el id para cargar datos
    })
  }
}
```

Navegación Programática

Navegar desde el código TypeScript permite lógica compleja:

```
export class TeamMemberCardComponent {
  constructor(private router: Router) {}

  viewMemberDetail() {
    this.router.navigate(['/team', this.member.id])
  }

}
```

¿Cuándo usar Navegación Programática?

- **Después de acciones**: Submit de formularios exitosos
- **Navegación condicional**: Basada en permisos o estado
- **Con lógica compleja**: Validaciones antes de navegar
- **Dinámicamente**: URLs construidas en runtime

Guards y Resolvers

Guards protegen el acceso a rutas:

```
@Injectable({ providedIn: 'root' })
export class AuthorizedGuard implements CanActivate {
  private isAuthorized = false

  constructor(private router: Router) {}

  canActivate(): boolean { }
}
```

Aplicando Guards a Rutas

```
export const routes: Routes = [
  { path: 'dashboard', component: DashboardComponent },
  {
    path: 'settings',
    component: SettingsComponent,
    canActivate: [AuthorizedGuard],
  },
]
```

El guard se ejecuta **antes** de activar la ruta.

Resolvers

Los **Resolvers** precargan datos antes de activar una ruta:

```
@Injectable({ providedIn: 'root' })
export class TeamMemberResolver implements Resolve<TeamMember | null> {
  constructor(private teamService: TeamMemberService, private router: Router) {}

  resolve(route: ActivatedRouteSnapshot): TeamMember | null {
    const id = Number(route.paramMap.get('id'))
    const member = this.teamService.getMemberById(id)
  }
}
```

Usando Resolvers en Rutas

```
export const routes: Routes = [
  {
    path: 'team/:id',
    component: TeamDetailComponent,
    resolve: { member: TeamMemberResolver },
  },
]
```

En el componente, acceder a los datos resueltos:

```
ngOnInit() {
  this.member = this.route.snapshot.data['member'];
}
```