



## ► 2. Navegación y datos


### Objetivos

- Comprender los eventos de navegación
- Usos previos y posteriores a la navegación
- Parámetros y consultas

### Referencia

#### Resolving route data in Angular

Not long ago, we wrote about Navigation Guards and how they let us control the navigation flow of our application's users. Guards like CanActivate, CanDeactivate and CanLoad are great when it comes to taking the decision if a user is allowed to activate a certain route, leaving a certain route, or even

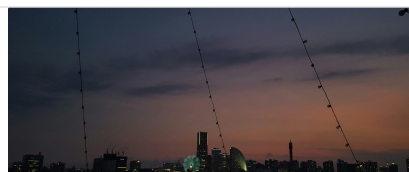
 <https://blog.thoughttram.io/angular/2016/10/10/resolving-route-data-in-angular-2.html>



#### Angular Router Series: Secondary Outlets Primer - Angular inDepth


In this short article, we're going to explore secondary outlets (sometimes called named router outlets), and see the role they play in routing.

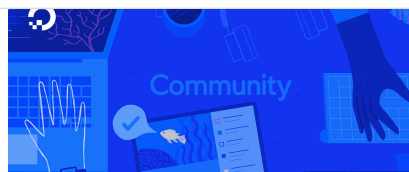
 <https://indepth.dev/posts/1025/angular-router-series-secondary-outlets-primer>



#### Cómo usar los parámetros de consulta en Angular | DigitalOcean

Los parámetros de consulta en Angular permiten pasar los parámetros opcionales a través de cualquier ruta en la aplicación. Los parámetros de consulta son diferentes a los parámetros de ruta regulares, que solo están disponibles en una ruta y no son opcionales (por ejemplo, ).

 <https://www.digitalocean.com/community/tutorials/angular-query-parameters-es>



#### Angular Pass Data to Route: Dynamic/Static - TekTutorialsHub


Angular allows us to pass data to the route. The route data can be either static or dynamic. The static data use the Angular route data property, where you can store arbitrary data associated with this specific route. For to pass dynamic data (or an object), we can make use of the history state object.

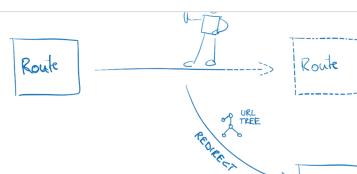
 <https://www.tektutorialshub.com/angular/angular-pass-data-to-route/>



#### Better Redirects in Angular Route Guards

Angular version 7.1 not only comes with bugfixes but includes also a bunch of new features. In this post I'd like to particularly take a look at the optimizations added to route guards. Contents are based on Angular version >= 7.1.0 Here is the release changelog of the features added in Angular version 7.1.0.

 <https://juristr.com/blog/2018/11/better-route-guard-redirects/>



## Angular 11 ActivatedRoute Route Tutorial with Example - positronX.io

In this tutorial you will learn about Angular 11 ActivatedRoute interface class with example, Angular offers ActivatedRoute interface class, it carries the information about a route linked to a component loaded into the Angular app template. An ActivatedRoute contains the router state tree within the angular app's

<https://www.positronx.io/understand-angular-7-activatedRoute-route-and-its-usage/>

positronX.io

## ▶▶ 0 - Enrutado para resource/:id

### Domain Resource Lib

Generar módulo lazy con su componente page para la ruta raíz y un servicio...

```
# 📁 Domain 📦 Resource Lib
ng g library resource --directory=domain --buildable --enableIvy --importPath=@ab/resource --prefix=ab-resource
--routing --parentModule='apps\www\src\app\core\core-routing.module.ts' --lazy
--simpleModuleName --strict --tags='domain, route'

# 📄 Resource Page
ng g c resource --project=domain-resource --flat --inlineStyle --skipSelector --type=Page --skipTests=true

# 🧑 Category data Service
ng g s data/resource --project=domain-resource
```

```
export class ResourceService {
  private readonly resourcesUrl =
    'https://api-angularbuilders.herokuapp.com/v1/resources';
  constructor(private http: HttpClient) {}
  getResourceById(resourceId: string) {
    return this.http
      .get<{ data: Resource }>(`${this.resourcesUrl}/${resourceId}`)
      .pipe(map((result) => result.data));
  }
}
```

## ▶▶ 1 - Ejecuciones previas a la navegación

### Resolvers

Servicios que obtiene los datos para que los componentes trabajen de forma síncrona

```
ng g resolver resource --project domain-resource --flat --skip-tests
```

```
export class ResourceResolver implements Resolve<Resource> {
  constructor(private service: ResourceService) {}
  resolve(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<Resource> {
    const resourceId = route.params.id;
    return this.service.getResourceById(resourceId);
  }
}
```

Asignar el resolver a la ruta

```
@NgModule({
  imports: [
    CommonModule,
    RouterModule.forChild([
      {
        path: ':id',
        pathMatch: 'full',
        component: ResourcePage,
        resolve: { resource: ResourceResolver },
      },
    ]),
  ],
  declarations: [ResourcePage],
})
export class ResourceModule {}
```

Consumir el dato recién obtenido en el componente

```
export class ResourcePage {
  resource = this.route.snapshot.data.resource;
  constructor(private route: ActivatedRoute) {}
}
```

Después se pueden mostrar directamente

```
<ab-ui-box>
  <div class="block is-size-4"><label class="mr-2">Web Page: </label> <a
    href="{{resource.url}}"
    target="_blank">{{resource.url}}</a>
  </div>
  <div class="block has-text-primary is-size-5"> 🚧 <i> Coming soon links to github, tutorials and
    your comments...</i>
  </div>
  <div class="block">
    <a class="button"
      [routerLink]='["/category", resource.categoryId]'>Back to category page</a>
  </div>
</ab-ui-box>
```

## Guards

Se generan con su propio comando



Idealmente una librería dedicada a autorizaciones: <https://github.com/angularbuilders/ab-generators/blob/main/tools/scripts/6-ab-ng-auth.sh>

```
nx g guard auth --implements='CanActivate,CanLoad' --no-interactive --project=shared-auth --flat
```



Can-Can... ¿Qué **puedes** hacer?

El guardia es similar a un resolver. Por ahora deja pasar a cualquiera, en el futuro lo determinará un usuario autorizado.

```
# Esta es implementación inicial
export class AuthGuard implements CanActivate, CanLoad {
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
    | UrlTree {
    return true;
  }
  canLoad(
    route: Route,
    segments: UrlSegment[]
  ):
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
    | UrlTree {
    return true;
  }
}
```

Y lo asignamos en el enrutador para evitar la descarga de un módulo al que no tenemos permiso

```
echo " generate resource-new page";
nx g library resource-new --importPath=@ab/resource-new --prefix=ab --routing --lazy --parentModule='apps\www\src\app\core\core-routing.mod
nx g c resource-new --project=domain-resource-new --flat --skipTests=false --skipSelector --type=Page
nx g c resource-new --project=domain-resource-new
nx g s resource-new --project=domain-resource-new --flat
git add *
git commit -m 'feat: generate resource-new page'
echo " generate not-found page";
```

```
import { AuthGuard } from '@ab/auth';
{
  path: 'resource-new',
  canLoad: [AuthGuard],
  loadChildren: () =>
    import('@ab/resource-new').then((module) => module.ResourceNewModule),
},

// Alternativa
@NgModule({
  imports: [
    CommonModule,
    RouterModule.forChild([
      {
        path: '',
        pathMatch: 'full',
        canActivate: [AuthGuard],
        component: ResourceNewPage,
      },
    ]),
  ],
  declarations: [ResourceNewPage, ResourceNewComponent],
})
export class ResourceNewModule {}
```

Para probarlo, agregamos un enlace al menú

```
<nav class="navbar" role="navigation" aria-label="main navigation">
  <div class="navbar-brand">
    <a class="navbar-item" href="/"> <strong class="ml-4 is-size-2">{{ title }}</strong> </a>
  </div>
  <div id="navbarItems" class="navbar-menu is-active">
    <div class="navbar-end">
      <a [routerLink]='resource-new'>Add a resource</a>
    </div>
  </div>
```

```
</div>
</nav>
```



Estos guardias se emplean para mejorar la **usabilidad**. Realmente no protegen contra intrusos. Se necesita igualmente la **seguridad implementada a nivel de API**. Se verá en un tema específico.

## ▶ 2 - Parámetros y consultas

### 🔍 El buscador: Una funcionalidad para todas las rutas

```
# 📁 Domain 📦 SearchBox Lib
nx g library search-box --importPath=@ab/search-box --prefix=ab --tags='domain, widget'

# 📁 SearchBox Component Widget
nx g c search-box --project=domain-search-box --flat --skipTests=false --type=Widget --export=true

# 📁 SearchBox Component Presenter
nx g c search-box --project=domain-search-box
# 🤖 Search data Service
nx g s search-box --project=domain-search-box --flat
```



Agregar sufijo **Widget** a `.eslintrc.json`

```
"@angular-eslint/component-class-suffix": [ "error", { "suffixes": ["Component", "Page", "Widget"] } ],
```

Empieza siendo casi un átomo

```
<input class="input is-medium mt-4 ml-4" type="text" placeholder="🔍 Type to find resources"
      #searchInput autocomplete="off">
```

Pero es un organismo con lógica de presentación específica del negocio.

```
export class SearchBoxComponent implements AfterViewInit{
  @Input() set initial(value: string) {
    if (this.searchInput && !value) {
      this.searchInput.nativeElement.value = value;
    }
  }
  @Output() search = new EventEmitter<string>();
  @ViewChild('searchInput') searchInput!: ElementRef;

  ngAfterViewInit(): void {
    fromEvent<{target:{value: unknown}}>(
      this.searchInput.nativeElement,
      'keyup'
    )
    .pipe(
      map((event) => event.target.value),
      map((value) => value as string),
      debounceTime(400),
      distinctUntilChanged()
    )
    .subscribe((searchTerm) => this.search.next(searchTerm));
  }
}
```

Su contenedor es muy simple (en apariencia)

```
<ab-search-box [initial]="initial" (search)="onSearch($event)"></ab-search-box>
```

Pero con toda la lógica... para responder a cambios y controlar inicializaciones. Es como una página, pero sin ruta asociadas: **un widget**.

!¿ Curiosamente en este caso su almacén de datos no es un servicio sino el propio router. De él obtiene los datos iniciales y en él escribe los valores recibidos del usuario.

```
export class SearchBoxWidget implements OnInit {
  initial = '';
  private current = '';
  constructor(
    private router: Router,
    private route: ActivatedRoute,
    private cdr: ChangeDetectorRef
  ) {}

  ngOnInit(): void {
    this.route.queryParamMap
      .pipe(
        map((params) => params.get('q') || ''),
        filter((q) => this.initial !== q && this.current !== q)
      )
      .subscribe((q) => {
        this.initial = q;
        this.cdr.markForCheck();
      });
  }

  onSearch(searchTerm: string): void {
    if (this.current === searchTerm) return;
    this.current = searchTerm;
    if (searchTerm.length >= 2) {
      this.router.navigate(['search'], {
        queryParams: { q: searchTerm },
        queryParamsHandling: 'merge',
      });
    } else {
      this.router.navigate(['']);
    }
  }
}
```

Este **widget** se consume directamente en el **layout** principal



Hay que importar el módulo del widget en el `CoreModule`

```
<nav class="navbar" role="navigation" aria-label="main navigation">
  <div class="navbar-brand">
    <a class="navbar-item" href="/"> <strong class="ml-4 is-size-2">{{ title }}</strong> </a>
  </div>
  <div id="navbarItems" class="navbar-menu is-active">
    <ab-search-box-widget class="navbar-start"></ab-search-box-widget>
    <div class="navbar-end">
      <a [routerLink]="['resource-new']">Add a resource</a>
    </div>
  </div>
</nav>
```



## Los resultados

Creamos toda la infraestructura (librería, módulo, enrutado, página, organismo, servicio...)

```
# 📁 Domain 📖 Search Lib
nx g library search --importPath=@ab/search --prefix=ab
  --routing --lazy --parentModule='apps\www\src\app\core\core-routing.module.ts' --tags='domain, page'

# 📄 Search Page
nx g c search --project=domain-search --flat --skipTests=false --skipSelector --type=Page

# 🧑 Search data Service
nx g s search --project=domain-search --flat
# 🌱 Presentacional component organism
nx g c result --project=domain-search --type=List
```

Y vamos de dentro hacia afuera. Empezado por el servicio que realiza el filtro en local (el API en este caso no lo provee...)

```
export class SearchService {
  private readonly resourcesUrl = 'https://api-angularbuilders.herokuapp.com/v1/resources';
  constructor(private http: HttpClient) {}

  getResourceByQuery$(query: string) {
    return this.http.get<{ data: Resource[] }>(`${this.resourcesUrl}`).pipe(
      map((result) => result.data),
      map((resources) => resources.filter((r) => this.matchesQuery(r, query)))
    );
  }
  private matchesQuery(resource: Resource, query: string) {
    const cleanQuery = query.toLowerCase().trim();
    if (cleanQuery.length === 0) return false;
    if (resource.name.toLowerCase().includes(cleanQuery)) return true;
    if (resource.description.toLowerCase().includes(cleanQuery)) return true;
    return false;
  }
}
```

La página obtiene parámetros del router y consulta al servicio. Usamos `switchMap` para encadenar ambos observables

```
export class SearchPage {
  resources$: Observable<Resource[]>;
  queryTerm = '';
  constructor(private route: ActivatedRoute, private service: SearchService) {}

  ngOnInit(): void {
    this.resources$ = this.route.queryParamMap.pipe(
      map((params) => params.get('q') || ''),
      tap((query) => (this.queryTerm = query)),
      switchMap((query) => this.service.getResourceByQuery$(query))
    );
  }
}
```

El resto es pintar os resultados de forma digna...

```
<ab-result-list *ngIf="resources$ | async as resources" [resources]="resources"
  [queryTerm]="queryTerm">
</ab-result-list>
```

```
<main>
  <ab-ui-header [header]="getHeader()"></ab-ui-header>
  <section class="columns is-multiline">
    <ab-ui-card
      *ngFor="let resource of resources"
      [card]="getCardFrom(resource)"
      class="column is-one-quarter-desktop is-half-tablet">
    </ab-ui-card>
  </section>
</main>
```

```

export class ItemsComponent {
  @Input() queryTerm = '';
  @Input() resources: Resource[] = [];

  private header = {
    heroClass: 'is-warning',
    title: `...`,
    subtitle: ' No resources yet ',
  };

  getHeader() {
    const header = { ...this.header };
    if (this.queryTerm) {
      header.title = `List of resources for ${this.queryTerm}`;
    }
    if (this.resources.length) {
      header.subtitle = `Found ${this.resources.length} resources`;
    }
    return header;
  }

  getCardFrom(resource: Resource) {
    return {
      title: resource.name,
      description: resource.description,
      link: '/resource/' + resource.id,
    };
  }
}

```

## ▶ 3 - Ejecución posterior a la navegación

**A** Útil para **SEO, analytics...**

### ⚡ Control de eventos

```

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot(
      [
        {
          path: '',
          data: { title: 'A.B Catalog' },
          loadChildren: () =>
            import('@ab/home').then((module) => module.HomeModule),
        },
        {
          path: 'category',
          data: { title: 'A.B Category view' },
          loadChildren: () =>
            import('@ab/category').then((module) => module.CategoryModule),
        },
        {
          path: 'search',
          data: { title: 'A.B Searching' },
          loadChildren: () =>
            import('@ab/search').then((module) => module.SearchModule),
        },
      ],
      { initialNavigation: 'enabled' }
    ),
  ],
  providers: [],
  bootstrap: [AppComponent],
})

```



```
export class AppModule {
  constructor(router: Router, activatedRoute: ActivatedRoute, title: Title) {
    router.events
      .pipe(
        filter((routerEvent) => routerEvent instanceof NavigationEnd),
        tap((event) => gtag('config', environment.ga, { page_path: event.urlAfterRedirects})),
        map(() => activatedRoute.firstChild?.snapshot.data.title),
        filter((title) => !!title)
      )
      .subscribe({
        next: (titleText) => title.setTitle(titleText),
      });
  }
}
```



**Alberto Basalo**