Material Routing Angular

Información Importante
Este material fue extraído de la documentación oficial de Angular el día 15/06/2020 y al momento de la lectura es posible que existan actualizaciones, por eso recomendamos encarecidamente consultar regularmente el sitio oficial.

Documentación oficial: https://angular.io/guide/router#router-reference

Router Reference

The folllowing sections highlight some core router concepts.

Router imports

The Angular Router is an optional service that presents a particular component view for a given URL. It is not part of the Angular core and thus is in its own library package, `@angular/router`.

Import what you need from it as you would from any other Angular package.
src/app/app.module.ts (import)

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

For more on browser URL styles, see `LocationStrategy` and browser URL styles.

Configuration

A routed Angular application has one singleton instance of the `Router` service. When the browser's URL changes, that router looks for a corresponding `Route` from which it can determine the component to display.

A router has no routes until you configure it. The following example creates five route definitions, configures the router via the `RouterModule.forRoot()` method, and adds the result to the `AppModule`'s `imports` array.
src/app/app.module.ts (excerpt)

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(
      appRoutes,
      { enableTracing: true } // <-- debugging purposes only
    )
    // other imports here
  ],
  ...
})

   export class AppModule { }
```

The appRoutes array of routes describes how to navigate. Pass it to the RouterModule.forRoot() method in the module imports to configure the router.

Each Route maps a URL path to a component. There are no leading slashes in the path. The router parses and builds the final URL for you, which allows you to use both relative and absolute paths when navigating between application views.

The :id in the second route is a token for a route parameter. In a URL such as /hero/42, "42" is the value of the id parameter. The corresponding HeroDetailComponent uses that value to find and present the hero whose id is 42.

The data property in the third route is a place to store arbitrary data associated with this specific route. The data property is accessible within each activated route. Use it to store items such as page titles, breadcrumb text, and other read-only, static data. You can use the resolve guard to retrieve dynamic data.

The empty path in the fourth route represents the default path for the application—the place to go when the path in the URL is empty, as it typically is at the start. This default route redirects to the route for the /heroes URL and, therefore, displays the HeroesListComponent.

If you need to see what events are happening during the navigation lifecycle, there is the enableTracing option as part of the router's default configuration. This outputs each router event that took place during each navigation lifecycle to the browser console. Use enableTracing only for debugging purposes. You set the enableTracing: true option in the object passed as the second argument to the RouterModule.forRoot() method.

Router outlet

The RouterOutlet is a directive from the router library that is used like a component. It acts as a placeholder that marks the spot in the template where the router should display the components for that outlet.

```
<router-outlet></router-outlet>

   <!-- Routed components go here -->
```

Given the configuration above, when the browser URL for this application becomes /heroes, the router matches that URL to the route path /heroes and displays the HeroListComponent as a sibling element to the RouterOutlet that you've placed in the host component's template.

Router links

To navigate as a result of some user action such as the click of an anchor tag, use RouterLink.

Consider the following template:
src/app/app.component.html

```
<h1>Angular Router</h1>
<nav>
  <a routerLink="/crisis-center" routerLinkActive="active">Crisis Center</a>
  <a routerLink="/heroes" routerLinkActive="active">Heroes</a>
</nav>

   <router-outlet></router-outlet>
```

The RouterLink directives on the anchor tags give the router control over those elements. The navigation paths are fixed, so you can assign a string to the routerLink (a "one-time" binding).

Had the navigation path been more dynamic, you could have bound to a template expression that returned an array of route link parameters; that is, the link parameters array. The router resolves that array into a complete URL.

Active router links

The RouterLinkActive directive toggles CSS classes for active RouterLink bindings based on the current RouterState.

On each anchor tag, you see a property binding to the RouterLinkActive directive that looks like routerLinkActive="...".

The template expression to the right of the equal sign, `=`, contains a space-delimited string of CSS classes that the Router adds when this link is active (and removes when the link is inactive). You set the `RouterLinkActive` directive to a string of classes such as `[routerLinkActive]="'active fluffy'"` or bind it to a component property that returns such a string.

Active route links cascade down through each level of the route tree, so parent and child router links can be active at the same time. To override this behavior, you can bind to the `[routerLinkActiveOptions]` input binding with the `{ exact: true }` expression. By using `{ exact: true }`, a given `RouterLink` will only be active if its URL is an exact match to the current URL.

Router state

After the end of each successful navigation lifecycle, the router builds a tree of `ActivatedRoute` objects that make up the current state of the router. You can access the current `RouterState` from anywhere in the application using the `Router` service and the `routerState` property.

Each `ActivatedRoute` in the `RouterState` provides methods to traverse up and down the route tree to get information from parent, child and sibling routes.

Activated route

The route path and parameters are available through an injected router service called the ActivatedRoute. It has a great deal of useful information including:

Property	**Description**
`url`	An `Observable` of the route path(s), represented as an array of strings for each part of the route path.
`data`	An `Observable` that contains the `data` object provided for the route. Also contains any resolved values from the resolve guard.

`paramMap`	An `Observable` that contains a map of the required and optional parameters specific to the route. The map supports retrieving single and multiple values from the same parameter.
`queryParamMap`	An `Observable` that contains a map of the query parameters available to all routes. The map supports retrieving single and multiple values from the query parameter.
`fragment`	An `Observable` of the URL fragment available to all routes.
`outlet`	The name of the `RouterOutlet` used to render the route. For an unnamed outlet, the outlet name is primary.
`routeConfig`	The route configuration used for the route that contains the origin path.
`parent`	The route's parent `ActivatedRoute` when this route is a child route.
`firstChild`	Contains the first `ActivatedRoute` in the list of this route's child routes.

`children`	Contains all the child routes activated under the current route.

Two older properties are still available, however, their replacements are preferable as they may be deprecated in a future Angular version.

- `params`: An `Observable` that contains the required and optional parameters specific to the route. Use `paramMap` instead.
- `queryParams`: An `Observable` that contains the query parameters available to all routes. Use `queryParamMap` instead.

Router events

During each navigation, the `Router` emits navigation events through the `Router.events` property. These events range from when the navigation starts and ends to many points in between. The full list of navigation events is displayed in the table below.

Router Event	**Description**
`NavigationStart`	An event triggered when navigation starts.
`RouteConfigLoadStart`	An event triggered before the `Router` lazy loads a route configuration.

`RouteConfigLoadEnd`	An event triggered after a route has been lazy loaded.
`RoutesRecognized`	An event triggered when the Router parses the URL and the routes are recognized.
`GuardsCheckStart`	An event triggered when the Router begins the Guards phase of routing.
`ChildActivationStart`	An event triggered when the Router begins activating a route's children.
`ActivationStart`	An event triggered when the Router begins activating a route.
`GuardsCheckEnd`	An event triggered when the Router finishes the Guards phase of routing successfully.
`ResolveStart`	An event triggered when the Router begins the Resolve phase of routing.
`ResolveEnd`	An event triggered when the Router finishes the Resolve phase of routing successfuly.

`ChildActivationEnd`	An event triggered when the Router finishes activating a route's children.
`ActivationEnd`	An event triggered when the Router finishes activating a route.
`NavigationEnd`	An event triggered when navigation ends successfully.
`NavigationCancel`	An event triggered when navigation is canceled. This can happen when a Route Guard returns false during navigation, or redirects by returning a `UrlTree`.
`NavigationError`	An event triggered when navigation fails due to an unexpected error.
`Scroll`	An event that represents a scrolling event.

When you enable the `enableTracing` option, Angular logs these events to the console. For an example of filtering router navigation events, see the router section of the Observables in Angular guide.

Router terminology

Here are the key `Router` terms and their meanings:

Router Part	**Meaning**
`Router`	Displays the application component for the active URL. Manages navigation from one component to the next.
`RouterModule`	A separate NgModule that provides the necessary service providers and directives for navigating through application views.
`Routes`	Defines an array of Routes, each mapping a URL path to a component.
`Route`	Defines how the router should navigate to a component based on a URL pattern. Most routes consist of a path and a component type.
`RouterOutlet`	The directive (`<router-outlet>`) that marks where the router displays a view.

`RouterLink`	The directive for binding a clickable HTML element to a route. Clicking an element with a `routerLink` directive that is bound to a *string* or a *link parameters array* triggers a navigation.
`RouterLinkActive`	The directive for adding/removing classes from an HTML element when an associated `routerLink` contained on or inside the element becomes active/inactive.
`ActivatedRoute`	A service that is provided to each route component that contains route specific information such as route parameters, static data, resolve data, global query params, and the global fragment.
`RouterState`	The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree.
Link parameters array	An array that the router interprets as a routing instruction. You can bind that array to a `RouterLink` or pass the array as an argument to the `Router.navigate` method.
Routing component	An Angular component with a `RouterOutlet` that displays views based on router navigations.