Routing

Siddharth Ajmera **@SiddAjmera**

Content

- Routing
- Child Routes
- RouteParams
- Guards

Routing - The Basics

- In Single Page Applications(SPAs), when there's a need for some new content, the whole page never changes. Only the content on that particular page changes. This gives the App a more Desktop Application like feeling.

- SPAs are faster as compared to normal Web Apps for the same reason.
- Routing is an Important Part of this behavior that SPAs exhibit.

Routing - How To?

1. Create a separate module for routing.
2. Import RouterModule, Routes in your AppRoutingModule.
   import { Routes, RouterModule } from '@angular/router';
3. Create a Routes Config.

```
const appRoutes: Routes = [
    { path: 'home', component: HomeComponent },
    { path: 'user-list', component: UserListComponent },
    { path: 'parent', component: ParentComponent },
    { path: '', redirectTo: '/home', pathMatch: 'full' }
];
```

4. Call RouterModule.forRoot() and give it the Routes config that you just created.
5. Export this Module into your RootModule.

```
@NgModule({
    imports: [ RouterModule.forRoot(appRoutes) ],
    exports: [ RouterModule ]
})
```

6. Place a <router-outlet></router-outlet> tag in your template where you want to perform it.
7. Place links that will take your user to those routes and use routerLink attribute to give them links.

Child Routes & Params

1. Add a children property to a route, the value of it would be an array of routes.
2. Each child route in the children will again contain a path property and a component property.

```
{ path: 'users', component: UsersComponent, children: [
    { path: '', component: NotFoundComponent },
    { path: ':id', component: UserDetailComponent }
] },
```

3. If you want to show the component content in some content that's already present in <router-outlet>, you'll have to add another router outlet in its parent's template.
4. You can configure a route to take params as well. Do that by supplying a colon(:) in front of the param name.
5. You can get the value of the current route params or route query params using ActivatedRoute as a dependency.
6. ActivatedRoute exposes a params Observable you can subscribe to, to get the params on the current route.
7. ActivatedRoute also exposes a queryParams Observable you can subscribe to, to get the query params on the current route.

```
this.activatedRoute.queryParams.subscribe((queryParams) => {
    console.log('got the params query params as : ', queryParams);
});
```

Types of Route Paths

1. Absolute Path: Has '/' in the front. Takes you to hostname:port/name-of-the-supplied-path.
2. Relative Path: Has './' or nothing in front. Takes you to the current route followed by the route name provided. Eg: hostname:port/path-on/path-provided.
3. Parent Path: Has '../' in front. Takes you one level up in the route structure. Eg, if you're on hostname:port/level1/level2, it will take you to hostname:port/level1.

“If Westeros has the Wall, Angular routes have Guards”

ANOTHER RANDOM GoT FAN

Guards

- Guards are a way of performing checks before we start navigating to or from different routes in our application
- They allow us to restrict access to certain routes in our application to certain users
- They allow us to validate/confirm before navigating out of routes
- Guards themselves are simple classes, which can have dependencies injected into them
- Guard functions return booleans, or Observables and Promises which resolve booleans
- Navigation is carried out if boolean returned is true, else it is prevented
- A single route can have multiple guards, and they are checked in the order of injection

Types of Guards

- CanActivate: Checks to see if a user can visit a route
- CanActivateChild: Checks to see if a user can visit a routes children
- CanDeactivate: Checks to see if a user can exit a route
- CanLoad: Checks to see if lazy-loaded modules should be loaded
- Resolve: Performs route data retrieval before route activation

CanActivate/CanActivateChild

- CanActivate checks to see if a user can visit a route
- CanActivateChild checks to see if a user can visit a routes children
- Class which implements CanActivate/CanActivateChild interface from @angular/router
- Accepts the arguments:
  - route: ActivatedRouteSnapshot - Future route. Contains params
  - state: RouterStateSnapshot - Future RouterState. Contains URL
- Needs to be registered on the providers array of module
- Added to the canActivate/canActivateChild Array of route
- Most commonly used to check if user is logged in or has sufficient previledges

CanDeactivate

- CanDeactivate checks to see if a user can exit a route
- Class which implements CanDeactivate interface from @angular/router
- Accepts the arguments:
  - component: Component - The current component
  - route: ActivatedRouteSnapshot - Future route. Contains params
  - state: RouterStateSnapshot - Future RouterState. Contains URL
- Needs to be registered on the providers array of module
- Added to the canDeactivate Array of route
- Most commonly used to check if user is navigating out of a route without saving some changes

Resolve

- Resolve performs route data retrieval before route activation
- Class which implements Resolve interface from @angular/router
- Accepts the arguments:
  - route: ActivatedRouteSnapshot - Future route. Contains params
  - state: RouterStateSnapshot - Future RouterState. Contains URL
- Needs to be registered on the providers array of module
- Added to the resolve Object of route with a data key
- Accessed in component as route.snapshot.data['key']
- Used to load necessary data before loading a route, often to set flags or prevent undefined/nulls