netmind
a bts company

Angular 14 - 11

Routes and Navigation

Angular routing

- The **Angular router** is a core part of the Angular platform. It enables developers to build Single Page Applications with multiple views and allow navigation between these views.
  - https://angular.io/guide/routing-overview

- Angular router is componsed by:
  - An router module: **`app-routing.module.ts`**
  - A directive: **`<router-outlet></router-outlet>`**
  - Navigation directive: **`[routerLink] = ["route to navigate"]`**
  - We can also navigate programatically using the **`Router`** object.

The routing module

- We will need a module that takes care of the routes.
  - We define a decorated class with annotation **@NgModule**
    - https://angular.io/guide/ngmodules
  - Uses two classes from the @angular/router: **Routes** and **RouterModule**.
  - **Imports** the basic RouterModule to be configured.
  - **Exports** the configured RouterModule.
  - Must import all the components that are going to the routes destination.
  - Must be imported by main module to apply the routes.
- The array of type Routes holds the **route definitions**:
  - It defines pairs `{path,component}`, the routes we want for our application.
  - Route specification: https://angular.io/api/router/Route
  - `{ path: 'products', component: ProductsListComponent }`
- **Redirects** may be defined too:
  - `{path: '', redirectTo: 'home', pathMatch: 'full'}`
  - For the special case of an empty URL, we also need to add the **pathMatch: 'full'** property,
  - so Angular knows how to exactly match the empty string and not partially match that empty string.
  - Also a route to fit all not definied routes (404): `{path: '**', component: NotFoundComponent}`
- Then we use the router outlet selector:
  - `<router-outlet></router-outlet>`
  - Here the router will show the related component to the route.

The routing module

```
// routing module
import { RouterModule, Routes } from '@angular/router’;
// import components

const routes: Routes = [
  { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
  { path: 'products', component: ProductsListComponent },
  { path: 'dashboard', component: DashboardComponent },
  { path: '**', component: NotFoundComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

```
// main module
import { AppRoutingModule } from './app-
routing.module';

@NgModule({
  declarations: [...],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
// main component template
...
<router-outlet></router-outlet>
...
```

🔗 **In the browser you can try these routes:**

- http://localhost:4200/
- http://localhost:4200/products
- http://localhost:4200/users

The routing module

- The **forRoot** is the method that configures the root routing module for the app.
- We have **forChild** method: is the method that we will call to register routes throughout the app and will be used it inside of the child, routing modules that you create.
  - We will see this later in modules section.
  - https://angular.io/api/router/RouterModule

Links to routes in the view

- We use the **[routerLink]** attributes, which associate the indicated value with the defined routes.

```
<section>
  <h1>{{title}}</h1>
  <header>
    <nav>
      <ul>
        <li><a [routerLink]="['']" routerLinkActive="router-link-active">Home</a></li>
        <li><a [routerLink]="['products']">Products</a></li>
      </ul>
    </nav>
  </header>
  <router-outlet></router-outlet>
</section>
```

- Note that [routerLink] has as its value another array as its attribute value. Namely.:
  - `[routerLink] = "['subroute1','subroute2','subroute3', ...]"`
  - These subpaths are joined to form the route being pointed to.
- Besides this, we can mark the active route element with **`RouterLinkActive`** directive.

Let's put it into practice: Tasks/Projects App

- Split views of task and project lists.
    - Create a different route for each view.
- Add components for the header menu, footer and for 404 route.

Routes with path params

- We can generate patterns for routes that involve parameters, so that all routes that match the pattern are served by the same component.
- The format it will adopt will be the following:
  - `{ path : 'path/:param1/subpath/:param2', component : Component }`
  - For example:
  - `{ path : 'product/:code', component : ProductDetailComponent }`
- Usually we will need to read the parameter in the destination component, to load the proper data.
  - To read the value of the parameters we will use the **ActivatedRoute** service
  - Then we will inject the service into the constructor.
  - Finally we can subscribe to the params property of the route in the destination component to read the parameter information

Routes with path params

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { IProduct } from 'src/app/models/iproduct';

@Component({
  selector: 'app-product-detail',
  templateUrl: './product-detail.component.html',
  styleUrls: ['./product-detail.component.scss'],
})
export class ProductDetailComponent implements OnInit {
  constructor(private _route: ActivatedRoute) {}

  code: string = '';
  product: IProduct|null = null;

  ngOnInit(): void {
    this._route.params.subscribe((params) => {
      console.log('params:', params);
      this.code = params['code'];
      // load product by code
    });
  }
}
```

Programmatic Navigation

- We can navigate to routes using the **Router service.**
- In the constructor we will inject the router and later we will use the navigate method to navigate

```
import { ActivatedRoute, Router } from '@angular/router';

@Component()
export class ProductDetailComponent implements OnInit {
  constructor(private _route: ActivatedRoute, private _router: Router) {}
  ...
  goHome() {
    this._router.navigate(['']);
  }

  goProducts() {
    this._router.navigate(['products']);
  }
}
```

Child routes

- In Angular we can create child routes within a specific route, using **children** attribute within the routes configuration

```
...
  {
    path: 'products/:code',
    component: ProductDetailComponent,
    children: [
      { path: '', redirectTo: 'features', pathMatch: 'full'  },
      { path: 'features', component: NotFoundComponent },
      { path: 'images', component: NotFoundComponent },
    ],
  },
...
```

- In the parent component template (ProductDetailComponent in the example), we will use **router-outlet** to indicate where the content of the child route should appear.

```
<p>
<a [routerLink]="['./features']">Tracks</a> |
<a [routerLink]="['./images']">Albums</a>
</p>

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

- As can be seen, the routerLink will use relative references for the routes.

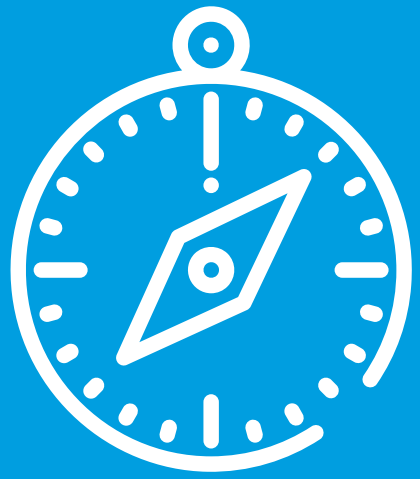
Access to parent's path params

- When working with child routes, it is very likely that we will need to access the path params defined in the parent.
- The way to do it is also using ActivatedRoute, but this time subscribing to the parameters of the **parent route**

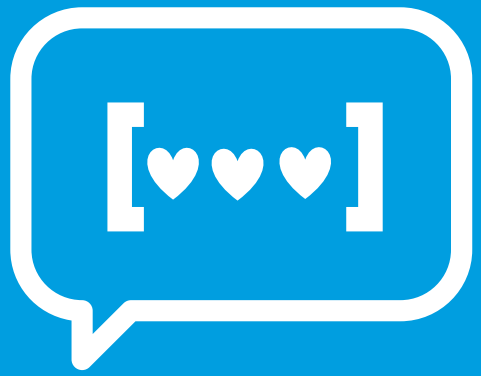
```
ngOnInit(): void {
    this._route.parent.params.subscribe(params => {
      //do something
    });
}
```

Let's put it into practice: Tasks/Projects App

1. Create a component that allows you to see the details of a task.
    - The specific task will be identified by its id that must come in the route.
    - Add buttons in the detail view, for going to the next or previous task (programatically).
2. Do the same for projects.
3. Add a two tab view to a project detail using child routes:
    - Default detail
    - Team members



Next steps



We would like to know your opinion!

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

netmind
a bts company

Thanks!

Follow us: