# Angular Best Friends

## Module 6 Exercise 2 – Routing to features

## Goal

By now we know that we need to organize the application features into modules. During this exercise we’ll learn how to implement routing in our feature modules.

## Steps

1. Go to the “**Module6Exercise2 > initial**” folder
2. Right click it and open he folder in VS Code
3. In the terminal window run **npm install** to install the needed node modules. This step is needed only when you run the application for the first time.

### Routing to features

1. Go to the “**src > app > products**” folder. This folder contains our product module. So open the “**product.module.ts**” file
2. Import the RouterModule on the top of the page

import { RouterModule } from '@angular/router';

1. In the imports array of NgModule, make sure to add the RouterModule and configure routes

@NgModule({

imports: [

SharedModule,

RouterModule.forChild([

{ path: 'products', component: ProductListComponent }

])

],

Notice that here we need to use the forChild() function since this router module resides inside a feature module.

We define a path called products and define that each time this route is matched, the product list should be displayed.

1. We need now to activate this route in the template. Meaning on the nav bar we have. So, to do this, go to the “**app.component.html**” file
2. In the anchor tag for “Product list” add a routerlink directive, just like we did for “Home”

<li class="nav-item">

<a class="nav-link" [routerLink]="['/products']">Product List</a>

</li>

### Implement login and logout

1. Now we want to create routes for login and logout and learn how to activate routes from code. Login and logout functionality are inside the user module. So navigate to the “**src > app > user**” folder and open the “**user.module.ts**” file.
2. Import the RouterModule where all imports are made

import { RouterModule } from '@angular/router';

1. In the imports array of NgModule import the RouterModule.forChild() and provide the necessary route configurations for a path called “login”

@NgModule({

imports: [

SharedModule,

RouterModule.forChild([

{ path: 'login', component: LoginComponent }

])

],

declarations: [

LoginComponent

]

})

Note that this is the entire NgModule. You need to only add the routing part in the existing code

1. Activate the new route in “**app.component.html**” on the navbar. Lookup the “Login” menu item and add a routerlink directive to it

<li class="nav-item"

\*ngIf="!isLoggedIn">

<a class="nav-link"

[routerLink]="['/login']">Log In</a>

</li>

1. Run the app with ng serve –open. Notice that if you click on “Product list” the product list component is displayed. If you click on login, then the login component is displayed.
2. You can try the login out with any username and password. You will notice that after the login you will remain on the same login component. So this is the place where we would like to redirect users back to the previous page after login. Let’s implement this feature!
3. Locate and go to the “**login.components.ts**” file
4. Import the RouterModule on the top of the page

import { RouterModule } from '@angular/router';

1. Inject the Router service in the constructor of the component:

constructor(private authService: AuthService,

private router: Router) { }

1. The login() function is called when we log in. So here we can add some logic. So after the successful login add a route activation in the code

// Navigate to the Product List page after log in.

this.router.navigate(['/products']);

The entire login function should look like this:

login(loginForm: NgForm) {

if (loginForm && loginForm.valid) {

const userName = loginForm.form.value.userName;

const password = loginForm.form.value.password;

this.authService.login(userName, password);

// Navigate to the Product List page after log in.

this.router.navigate(['/products']);

} else {

this.errorMessage = 'Please enter a user name and password.';

}

}

1. Test again the login functionality. You should be redirected to the products list after successful login
2. Similarly we want to redirect users to the welcome page after logout. So we need to activate the welcome route in the logout() function.
3. Go to the **app.component.ts** file where we have the logout() method. Import the router and inject it into the constructor (see steps 16 and 17)
4. In the logout function add the route activation

logOut(): void {

this.authService.logout();

//This needs to be added

this.router.navigateByUrl('/welcome');

}

1. Play around with the new features

### Defining a routing module

1. In the **app** folder create a new file called “**app-routing.module.ts**”
2. Paste the following code in the new file

import { NgModule } from '@angular/core';

import { RouterModule } from '@angular/router';

import { WelcomeComponent } from './home/welcome.component';

import { PageNotFoundComponent } from './page-not-found.component';

@NgModule({

imports: [

RouterModule.forRoot([

{ path: 'welcome', component: WelcomeComponent },

{ path: '', redirectTo: 'welcome', pathMatch: 'full' },

{ path: '\*\*', component: PageNotFoundComponent }

])

],

exports: [RouterModule]

})

export class AppRoutingModule { }

We have basically moved the route configuration in this module and imported the necessary components.

1. Import the new routing module in “**app.module.ts**”

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

1. In the imports array of the module delete the routing configuration and add the AppRoutingModule instead. However, add it at the end of the array. At the end, your new imports array should look like this:

imports: [

BrowserModule,

HttpClientModule,

InMemoryWebApiModule.forRoot(ProductData, { delay: 1000 }),

ProductModule,

UserModule,

MessageModule,

AppRoutingModule

],

1. Play around and check that all routes are still working