# Angular Best Friends

## Module 6 Exercise 3 – Routing parameters

## Goals

Most of the times simple routing is not enough and we need a way to also include different parameters and placeholders in our routes. In this exercise we’ll play around with route parameters, query parameters and few other very important Angular routing features.

## Steps

1. Go to the “**Module6Exercise3 > 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.

### Route parameters

1. Go to the “**src > app > products**” folder and open the **product.module.ts** file.
2. Note that we have only one route definition here that handles routing to the product list. Here we will need to add further route definitions to be able to open a single product, to edit a product or create a new product. So, under the existing route definition add the following route definitions objects to the array:

{ path: 'products/:id', component: ProductDetailComponent },

{ path: 'products/:id/edit', component: ProductEditComponent }

Note how we added the “id” as a route parameter or placeholder. Remember that the “:” sign is always used to put a placeholder in a route that will be populated at runtime with an appropriate value

1. Now we need to activate this route. As usual we can activate a route in the template (HTML) or in code. Let’s start with the template. What we want to achieve is that when we click on a product we will be routed to the product details page. So go to the **product-list.component.html** file
2. Notice the anchor tag on lines 50-52

<a>

{{ product.productName }}

</a>

1. This is how our products are populate in the list with ngFor. So let’s add a router link directive to it to activate the route:

<a [routerLink]="['/products', product.id]">

{{ product.productName }}

</a>

1. Let’s also activate the route for the edit product button. You’ll find the button in the same HTML template on lines 62-64. Add a routerLink directive to it by taking this code sample as example:

<button class="btn btn-outline-primary btn-sm"

[routerLink]="['/products', product.id, 'edit']">

Edit

</button>

1. Let’s also activate the route for the “Add product” menu option. This is defined in the app component template under **app.component.html** on lines 11-13. So add a routerLink to it. It should look the following way afterwards:

<li class="nav-item">

<a class="nav-link"

[routerLink]="['/products', 0, 'edit']">Add Product</a>

</li>

1. Now, as mentioned during the class, a very important step is to read the route parameters once the route gets activated. In this exercise we’ll use the snapshot approach. This means that the next actions need to be performed on the component that should actually be displayed when activating the route. Therefore, in VS Code go to the **product-detail.component.ts** file.
2. First import the ActivatedRoute component from Angular router. Place this import statement in the imports section on the top of the page

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

1. Add the ActivatedRoute as a dependency in the constructor of the component. After adding it, your constructor should look like this:

constructor(private productService: ProductService,

private route: ActivatedRoute) { }

1. Now we need to think about how to fetch the data we need from the data service. To to this, we’ll use the OnInit lifecycle hook in Angular. This life cycle hook gets called each time a component is being initialized. So by the time this gets called, we will have on the one site the data coming from the ActivatedRoute and the ability to make an asynchronous call to get the data for the product the user clicked on.
2. Add “**OnInit**” to the imports from Angular Core. You can just add it to the imports array on line 1. It should look like this when you’re done:

import { Component, OnInit } from '@angular/core';

1. Add the “implements” key word to the class definition. It should look like this:

export class ProductDetailComponent implements OnInit

1. Then we have to implement the **ngOnInit** function. So paste the following code under the constructor:

ngOnInit(): void {

const id = +this.route.snapshot.paramMap.get('id');

this.getProduct(id);

}

1. Run the application and play around with it. You should be able to navigate to single product detail page. Also from the products list you should be able to navigate to the edit product page.
2. However, on the single product details page you see two buttons “Back” and “Edit”. If you want to really get your hands dirty try to implement functionality so that the “Back” button will send the user back to the product list page and the “Edit” button will send the user to the product edit page.

Good luck!!