UNIT 2

Angular

Exercise week 9

Client-side Web Development
2nd course – DAW
IES San Vicente 2020/2021
Author: Arturo Bernal Mayordomo

Index

Exercise

Update the exercise from last week (week 9) with the following changes:

- We'll use web services from Unit 1 week 3 exercise (arturober.com:5007). No login (for now).
- We are going to implement a "product details" page, so create the corresponding component: **product-details**.

Routes

Create these routes in your app. First, create a separate module for the routes (We'll talk about modules in the 4th week) and put your routes in the **routes array** on that file. Don't forget to put the **router-outlet** component in the app component template.

```
ng g module --flat -m app app-routing
```

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule, Route } from '@angular/router';

const routes: Route[] = [
  // Your routes go here
];

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

Order is important!. products/add must go before products/:id, because :id is a variable and if it goes first, It will always load the detail page take 'add' as the id value.

- **products** → ProductsPageComponent
- **products/add** → ProductFormComponent
- **products/:id** → ProductDetailComponent (Create this component)
- **Default and other** → Redirect to '/products'

Update the navbar at the top of your application (you can reuse the one in the unit 1 project and modify it), with the following links:

- **Home** → '/products'

- **Create product** → '/products/add'

Interfaces

Create the necessary interfaces for the objects and server responses: Product, Category, ProductsResponse, ProductResponse, CategoriesResponse,…

Important: When creating a new product, the category is a number (the id), but when getting products from the server, it will return an object instead. There's a problem when using more than one type for the category (you cannot use casting in the HTML template), so it's better to create a Product interface for the form (**category: number**), and other for the rest of the application (**category: Category**). The best way to do this is creating a base interface and then the 2 derived interfaces:

```
interface ProductBase {
  id?: number;
  title: string;
  description: string;
  price: number;
  mainPhoto: string;
}

export interface Product extends ProductBase {
 category: Category;
}

export interface ProductAdd extends ProductBase {
 category: number;
}
```

Services

Create the following services:

- **CategoriesService** → This service will manage the product's categories with the following methods
    - **getCategories(): Observable<Category[]>**
- **ProductsService** → This service will manage everything related to products and have at least the following methods:
    - **getProducts(): Observable<Product[]>**
    - **getProduct(id: number): Observable<Product>**
    - **addProduct(product: ProductAdd): Observable<Product>**
    - **deleteProduct(id: number): Observable<void>**

Pages

'**products**' → Get products from Products.getProducts(). Product's image and title will be links to the details page. Don't forget to implement the delete button functionality!.

'**products/add**' → Contains the product form. When you add a new form successfully, redirect to 'products' instead of resetting the form. Don't forget to change the boolean that indicates you have created a product, so the CanDeactivate guard lets you leave the page without asking.

'**products/:id**' → ProductDetailComponent will only show the card (**product-card**) of the product, and a button to go back to the products page:

```
… (product card here) ...
<div class="mt-4 mb-4">
  <button class="btn btn-success" (click)="goBack()">Go back</button>
</div>
```

Guards

Create 2 guards for the routes:

- **NumericIdGuard** (CanActivate) → Use it for the product detail's page. Check if the id in the url is numeric, or redirect to the products page.
- **PageLeaveGuard** (CanDeactivate) → Ask the user if he/she wants to leave the page (use a confirm dialog). Use it for the add product page. Also, create a boolean in the component (false) and set it to true when you add a product, so it only asks the user **only** when no product has been added.

Resolvers

Create a resolver called **ProductResolver** (Resolve<Product>) and apply it to the '**product/:id**' route as a resolver. It will get the product data before loading the page. If there's an error, redirect to '**products**' page.

Interceptors

Create and interceptor called **BaseUrlInterceptor** that adds the server prefix (example → http://localhost:3000/) to the url of any http call. Updating the url in a request is done the same way as changing the headers. Clone the request and update what you want when cloning the object.

```
const reqClone = req.clone({
   url: `${environment.baseUrl}/${req.url}`
});
```

Tips

Methods that return Observable<void>

We use this type of return value when we want an observable that indicates that an operation was successful or not (deleting an element for example). But instead of returning a boolean, it will throw an error if the operation couldn't be completed.

We can use the map method to process the value and don't return anything **Observable<ResponseType>.pipe(map(v => { don't return anything })); → Observable<void>**. Or, if we are making a call to a web service that doesn't return any data, just use <void> in the response type: **http.delete<void>(…) → Observable<void>**.

Global values

If we are using constant values that can change if we are in development or production mode (like the url to the web services), we can use the **environment** object. Put the global attribute in **src/environments/environment.ts** and also in the production file **environment.prod.ts** (When compiling or running in production, this is the object that will be used).

```
export const environment = {
  production: false,
  baseUrl: 'http://arturober.com:5007'
};
```

Just import the environment constant anywhere you want to use it (**never** import from the production file, webpack will do that for you when compiling in production mode).