# Laboratorium 4: Routing

## Ćwiczenie 1. Przygotowanie routingu

Czas na coś bardziej skomplikowanego. Strona główna naszej galerii gotowa, pora dodać coś bardziej złożonego. Każdy nasz album zdjęciowy powinien dawać możliwość przejrzenia w nim zdjęć, prawda? Czyli klikamy na okładkę galerii i przechodzimy głębiej. Zajmiemy się tym teraz.

Czym jest routing? To nic innego jak ścieżki, po których można poruszać się w aplikacji. Na stronie www będzie to mapa ścieżek do poszczególnych podstron.

W aplikacjach typu SinglePage routing nieco się różni, ponieważ nie przenosi nas między podstronami, ale doładowuje nowe widoki do głównej strony w zależności od parametrów, które mu podamy. Tak więc cały czas operujemy na index.html, jedynie jego zawartość zmienia się dynamicznie, bez przeładowywania strony.

1. Utwórzmy więc oddzielny moduł do obsługi ścieżek. Angular zrobi to za nas:

   ```
   ng generate module app-routing --flat --module=app
   ```

   Przyjrzyjmy się tej komendzie.

   **--flat** - moduł app-routing zostanie utworzony w głównym katalogu aplikacji

   **--module=app** – doda moduł do tabeli **imports** w **app-modul**

   Jak widać poniżej, AppRoutingModule został dodany pod naszymi własnymi komponentami. Przenieśmy go wyżej, do sekcji z modułami, jako że to jedna z najważniejszych części naszej aplikacji.

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule} from '@angular/forms';

import { AppComponent } from './app.component';
import { NavComponent } from './components/nav/nav.component';
import { PolishDatePipe } from './pipes/polish-date.pipe';
import { SearchGalleriesPipe } from './pipes/search-galleries.pipe';
import { GalleriesComponent } from './components/galleries/galleries/galleries.component';
import { GalleryItemComponent } from './components/galleries/gallery-item/gallery-item.component';
import { GallerySearchComponent } from './components/galleries/gallery-search/gallery-search.component';
import { AppRoutingModule } from './app-routing.module';

@NgModule({
    declarations: [
        AppComponent,
        NavComponent,
        PolishDatePipe,
        SearchGalleriesPipe,
        GalleriesComponent,
        GalleryItemComponent,
        GallerySearchComponent
    ],
    imports: [
        BrowserModule,
        FormsModule,
        AppRoutingModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

2. Zajrzyjmy do modułu routingu. Ponieważ będziemy korzystać z dodatkowego modułu RouterModule, więc musimy go dodać – zrobimy to analogicznie do **imports** ale jako **exports**:

app-routing.module.ts

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

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

3. Trzeba pomyśleć o ścieżkach.

   My będziemy mieć dwie ścieżki. Domyślną, pustą, opartą o adres localhost:4200, która powinna uruchamiać **dashboard.component.ts** i ścieżkę do naszych galerii, do których będziemy przechodzić po kliknięciu odnośnika na górze strony:

   `localhost:4200/galleries`

Wynika z tego, że będziemy korzystać z komponentu **galleries.component.ts** ***oraz*** **dashboard.component.ts**, którego jeszcze nie mamy. Utwórzmy go:

```
ng generate component components/dashboard/dashboard
```

4. Typowa ścieżka posiada dwa parametry:
   **path:** odpowiada adresowi URL w przeglądarce
   **component:** component, który zostanie wywołany przez router pod podanym adresem.
   My będziemy mieć tablicę ścieżek. Dodajmy ją **nad** *@NgModule*

```
const routes: Routes = [{
    path: 'galleries',
    component: GalleriesComponent
}, {
    path: 'dashboard',
    component: DashboardComponent
}];
```

5. No dobrze, a co jak nic nie kliniemy? Domyślnie powinien pojawić się **dashboard.** Wystarczy na końcu dodać ścieżkę domyślną - jeśli path jest pusta, to przekieruj na dashboard.

```
const routes: Routes = [{
    path: 'galleries',
    component: GalleriesComponent
}, {
    path: 'dashboard',
    component: DashboardComponent
}, {
    path: '',
    redirectTo: '/dashboard',
    pathMatch: 'full'
}];
```

6. Mamy ścieżki, teraz musimy uruchomić router, żeby nasłuchiwał, czy w nie klikamy. W tym celu dodajemy router w @NgModule:

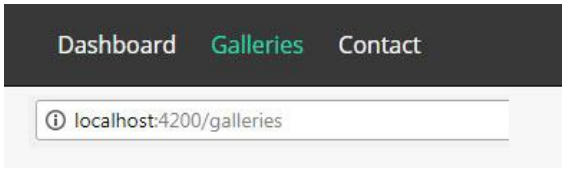
```
imports: [
    CommonModule,
    RouterModule.forRoot(routes)
],
```

7. Router przygotowany, ale nie mamy jeszcze nawigacji, żeby go przetestować. Przejdźmy do **nav.component.html** i dodajmy ścieżki do naszego menu u góry.

nav.component.html

```
<div class="container">
    <div class="row">
        <div class="grid-2">
            <a href="/" class="brand-logo">Angular App</a>
        </div>
        <div class="grid-14">
            <ul>
                <li><a routerLink="/dashboard" routerLinkActive="is-active">Main site</a></li>
                <li><a routerLink="/galleries" routerLinkActive="is-active">Galleries</a></li>
                <li><a routerLink="/contact" routerLinkActive="is-active">Contact</a></li>
            </ul>
        </div>
    </div>
</div>
```

**routerLinkActive** doda dam klasę ‘is-active’ do odnośnika, jeśli ścieżka będzie się zgadzała. Dzięki temu możemy podświetlić link od aktywnej podstrony, np. tak.




app.component.html | nav.component.html | nav.component.scss

:host ul li a

```
@import "../../../assets/styles/variables";

:host {
    width: 100%;
    background: $darker-gray;
    color: #fff;
    line-height: $font-size-h2;
    border-bottom: 1px solid #000;

    a {...}

    ul {
        float: right;

        li {
            display: inline-block;

            a {
                display: block;
                padding: 4px 10px;

                &.is-active {
                    color: $brand-success;
                }
            }
        }
    }
}
```

8. Jeśli teraz klikniesz na linki u góry, nic się nie wydarzy. To dlatego, że nasz router nie wie, gdzie ma wstawiać komponenty, które mu podaliśmy w ścieżce. Trzeba mu to miejsce wyznaczyć.

   Przejdźmy do **app.component.html** i nad komponentem z galeriami `<app-galleries></app-galleries>` dodajmy `<router-outlet></router-outlet>`

```
<app-nav></app-nav>

<div class="container">
    <router-outlet></router-outlet>
    <app-galleries></app-galleries>
</div>
```

   Kliknij odnośniki i sprawdź, co się dzieje. Możesz już stąd usunąć `<app-galleries></app-galleries>`

## Ćwiczenie 2. Podstrona galerii

Czas na coś bardziej skomplikowanego. Każdy nasz album zdjęciowy powinien dawać możliwość przejrzenia w nim zdjęć, prawda? Czyli klikamy na okładkę galerii i przechodzimy głębiej. Jak to zrobić? Pomocny znów będzie router.

Router daje nam możliwość nie tylko dodania czystej ścieżki, ale też zdefiniowania parametrów, z którymi będzie wywoływana. Co to oznacza? Że nie tylko możemy wywołać ścieżkę **localhost:4200/gallery**, ale też możemy określić, o którą galerię w niej chodzi.

1. Utwórzmy zupełnie nowy komponent gallery, który będzie odpowiadał za wyświetlanie pojedynczej galerii:

   `ng generate component components/galleries/gallery`

2. W **app-routing.module.ts** w tablicy ze ścieżkami dodajmy nową ścieżkę, zawierająca parametr – id galerii, którą chcemy otworzyć.

```
const routes: Routes = [{
    path: 'galleries',
    component: GalleriesComponent
}, {
    path: 'galleries/:galleryId',
    component: GalleryComponent
}, {
    path: 'dashboard',
    component: DashboardComponent
}, {
    path: '',
    redirectTo: '/dashboard',
    pathMatch: 'full'
}];
```

3. W routingu to wszystko. Teraz trzeba sprawić, żeby po kliknięciu w zdjęcie galerii, został wywołany właściwy adres. Przechodzimy do pliku **gallery-item.component.html** i zamykamy zdjęcie w znaczniku <a> z nową ścieżką:

```
<a routerLink="/galleries/{{gallery.galleryId}}" class="thumb-
container">
    <img src="{{gallery.thumbUrl}}">
</a>
```

4. Spróbujmy kliknąć – w adresie widać parameter i **gallery.component works!** Nie widać jedynie danych z galerii. Co musimy teraz zrobić, żeby wyświetlić te dane?
   - Pobrać z adresu **galleryId**
   - Wyciągnąć dane galerii z tablicy z galeriami
   - Wyświetlić je na stronie

5. Zacznijmy od wyciagnięcia parametru z adresu. Potrzebujemy do tego **ActivatedRoute**. W pliku **gallery.component.ts** w konstruktorze wpisujemy :

```
constructor(private route: ActivatedRoute) { }
```

6. Następnie definiujemy **galleryId** i w **ngOnInit()** nadajemy jej wartość wyciągniętą z routera:

gallery.component.ts

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

@Component({
  selector: 'app-gallery',
  templateUrl: './gallery.component.html',
  styleUrls: ['./gallery.component.scss']
})
export class GalleryComponent implements OnInit {

  private galleryId: string;

  constructor(private route: ActivatedRoute) { }

    ngOnInit() {
        this.galleryId = this.route.snapshot.paramMap.get('galleryId');
    }
}
```

7. Skoro mamy *galleryId* możemy wyciągnąć z tablicy z galeriami wszystkie dane galerii!
   Pod zmienną *galleryId* dodajemy kolejną zmienną *gallery: IGallery*. Następnie w ngOnInit() dopisujemy linię:

```
this.gallery = Galleries.find((item: IGallery) => item.galleryId === this.galleryId);
```

   Co zrobiliśmy ostatnią linią? Przeszukaliśmy *Galleries* w poszukaliśmy obiektu, który miał takie samo *galleryId* jak to pozyskane z parametru z odnośnika. Metoda ***.find*** to skrócona metoda działająca jak pętla for – przeszukuje wszystkie elementy tablicy (u nas 'item' typu IGallery) i sprawdza, który z tych elementów spełnia zadany warunek – u nas porównanie dwóch wartości galleryId.

8. Dopisz jeszcze wyświetlanie galerii w konsoli przegladarki  isprawdź jak to działa.

```
console.log(this.gallery);
```

   **Pamiętaj o zaimportowaniu interfejsu i pliku z galerami! Jeśli podczas wpisywania nie dodały się same na górze pliku, kliknij na nazwie a następnie wciśnij ALT + ENTER.**

9. Ostatnia rzecz to wyświetlenie informacji o galerii na stronie. Nic trudnego – przejdź do pliku **gallery.component.html** i wpisz tam:

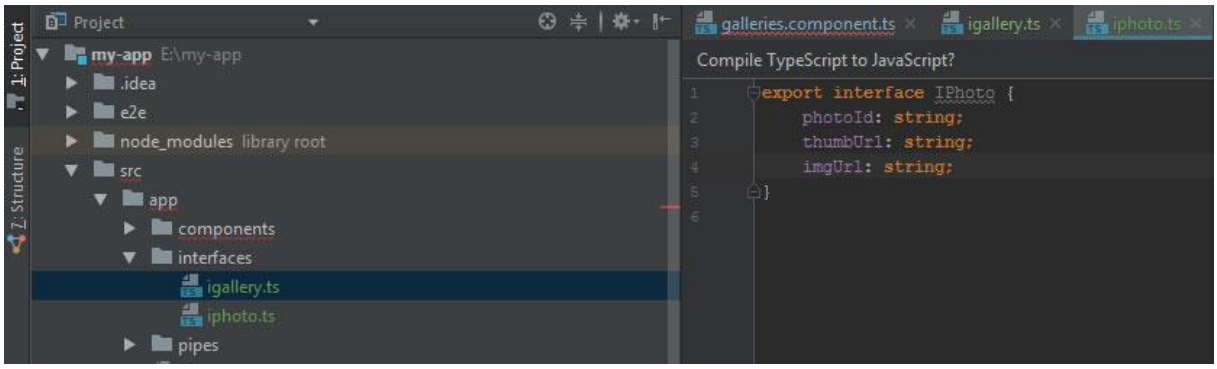
```
<h2>Galeria: {{gallery.title}}
    <br><small>galleryId: {{gallery.galleryId}}</small>
</h2>
```

## Ćwiczenie 3. Rozbudowa galerii

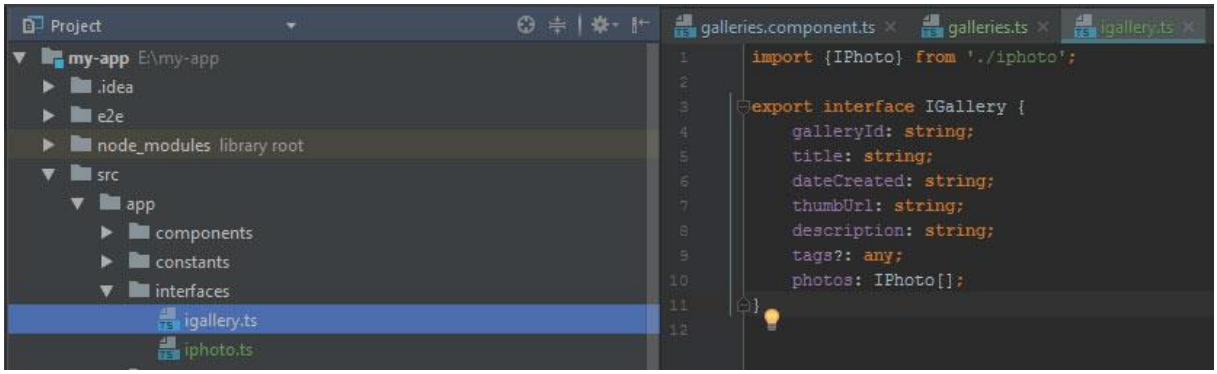
Nasza galeria nie ma w sobie wielu informacji. Rozbudujemy ją nieco.

1. W katalogu  *interfaces* utwórz plik **iphoto.ts**.

2. W pliku **iphoto.ts** tworzymy nowy interface **IPhoto** – nasze zdjęcia będą miały trzy pola:



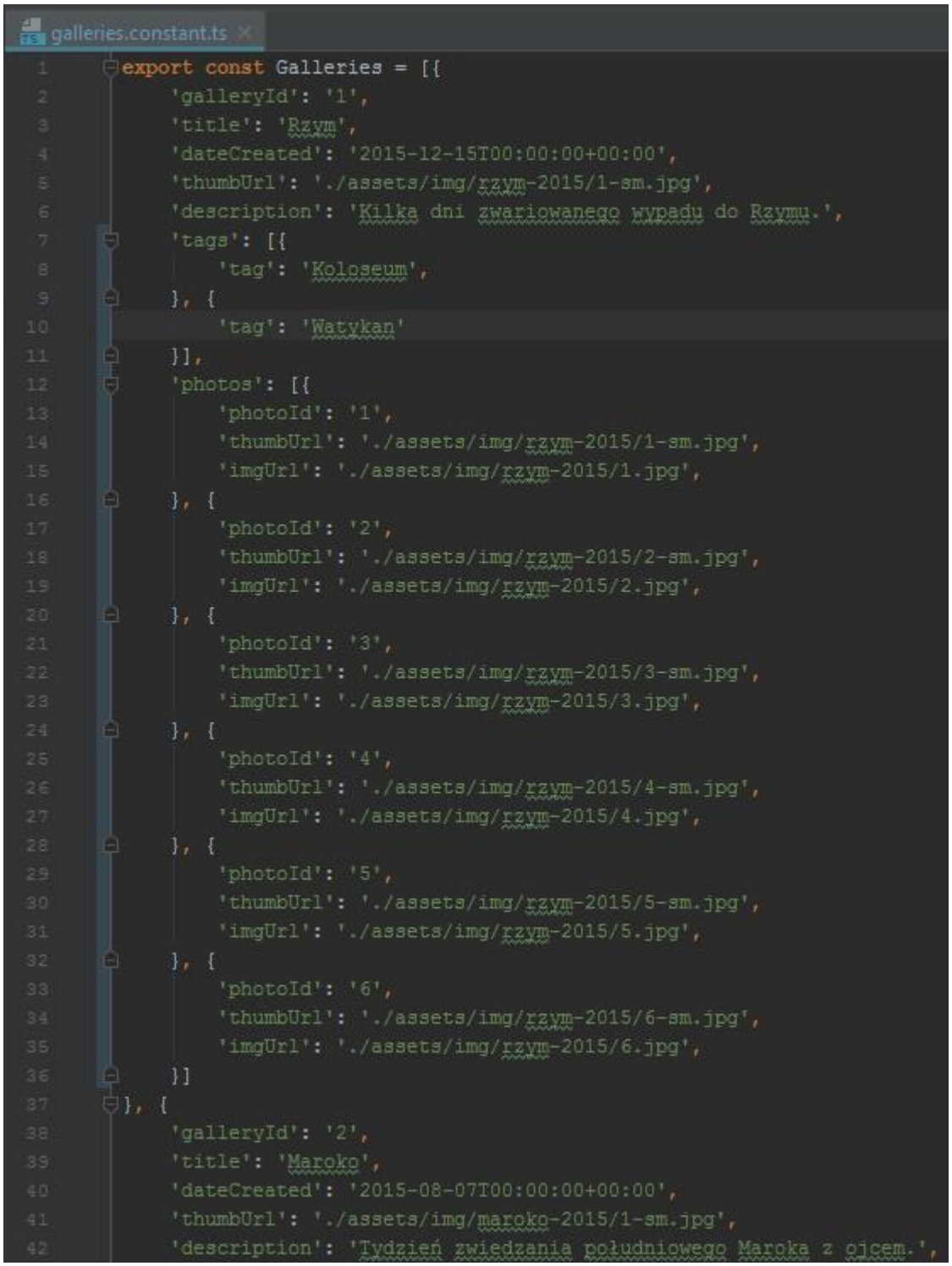

3. W pliku **igallery.ts** podmieniamy photos na tablicę z obiektami typu IPhoto:




Znak zapytania w **tags** oznacza, że jest to pole opcjonalne i nie musi występować. Ponieważ tablica ze zdjęciami korzysta z typu danych **IPhoto**, zadeklarujemy to tutaj. **[]** oznaczają, że będzie to **tablica obiektów** tego typu.

4. W pliku **galleries.constant.ts** dodajemy tagi i zdjęcia.

   Zrób to dla dwóch pierwszych galerii, do reszty dodaj tylko tags: [], photos: [] – uzupełnisz to kiedy indziej.

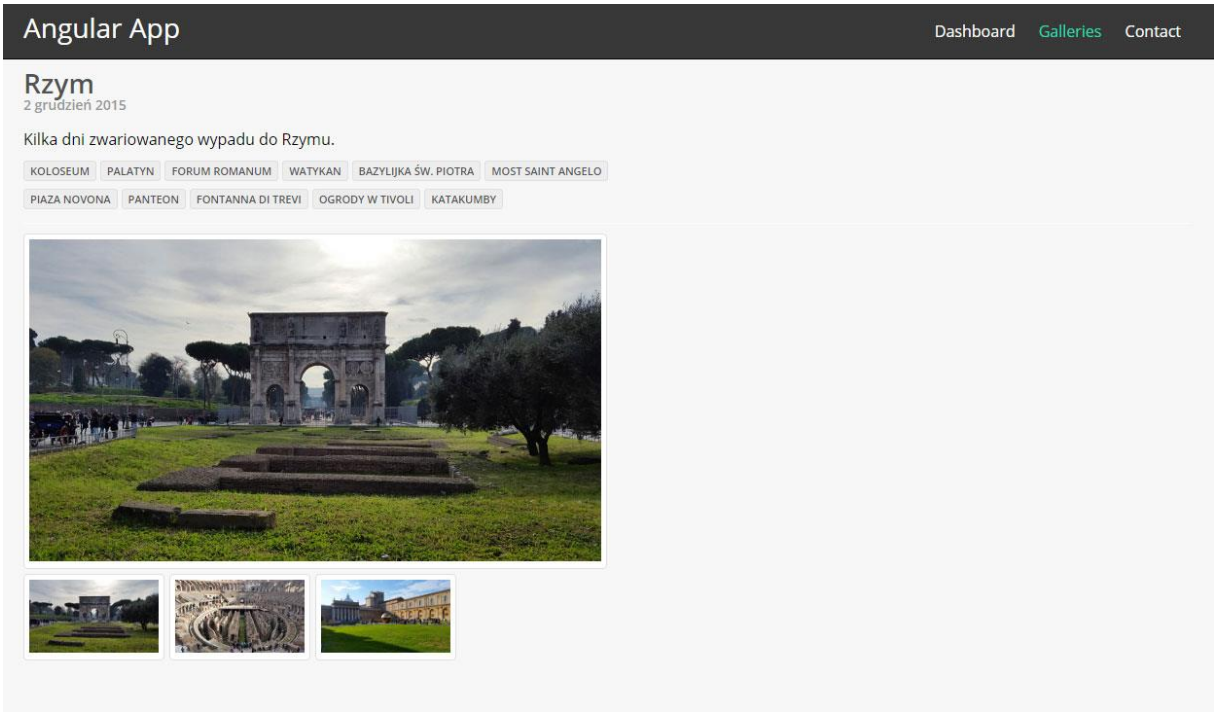



## Ćwiczenie 4. Samodzielne

1. Uzupełnij dane reszty galerii i dodaj zdjęcia (minimum 4 na galerię).
2. Uzupełnij stronę poszczególnych galerii. Powinna zawierać:

- Tytuł galerii
- Datę galerii w takim samym formacie jak na stronie z galeriami
- Tagi
- Jedno duże zdjęcie (pierwsze)
- Miniaturki pozostałych zdjęć

Strona powinna wyglądać mniej więcej tak:




## Zagadnienia na wejściówkę:

1. Czym jest routing?
2. Czym się różni metoda .find od .findIndex oraz .indexOf?
3. Na czym polega zasięg zmiennych i hoisting w JS?
4. Czym się różni display: block od display: inline-block w CSS?