Laboratorium 4: Routing

Wprowadzenie

Strona główna naszej galerii gotowa, pora dodać coś bardziej złożonego. Po tych zajęciach powinniśmy mieć możliwość przechodzenia między zbiorami zdjęć po kliknięciu na okładkę albumu zdjęciowego. Nowa podstrona powinna zawierać rozszerzone informacje na temat konkretnego albumu oraz zbiór zdjęć wchodzących w jego skład.

Zdjęcia powinny reagować na kliknięcia i prezentować duży podgląd. A wszystko w oparciu o pliki w formacie JSON.

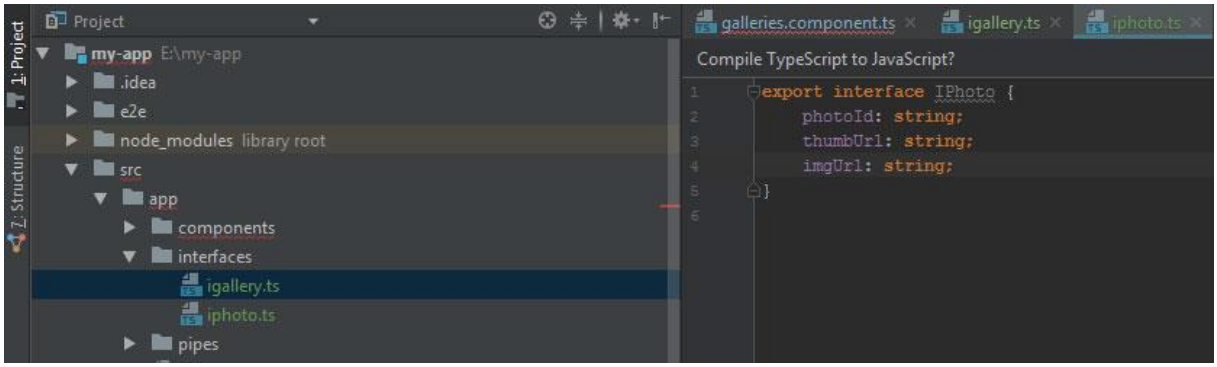
Ćwiczenie 1. Refaktor kodu

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.

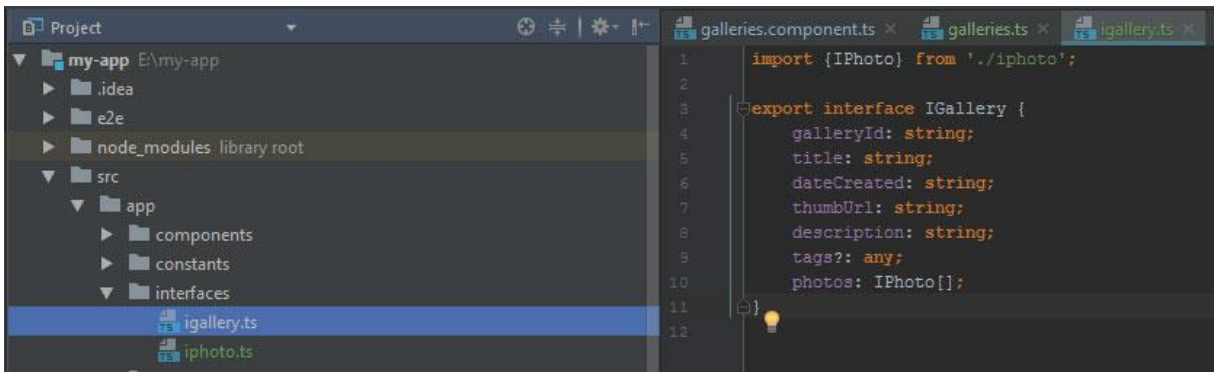
Jak to zrobić? Na początek dodamy zdjęcia i uporządkujemy kod. Wydzielimy *interfejsy* (czyli typy danych, z których będziemy korzystać) i oddzielimy dane od kodu. Zaczynamy!

1. W głównym katalogu naszej aplikacji tworzymy katalog: *interfaces*. W nim tworzymy dwa pliki: **igallery.ts** oraz **iphoto.ts**.

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



3. Do pliku **igallery.ts** przenosimy całą klasę z typami z **galleries.component.ts** i dodajemy zdjęcia oraz tagi:



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. Przeniesiemy też wszystkie dane z galerii. Tworzymy kolejny katalog: **constants** a w nim plik: **galleries.ts**. Przenosimy do niego całą naszą tablicę z danymi galerii.

   Zwróć uwagę, że program raportuje nam błąd – typ Gallery jest nieznany. Zmieńmy sposób deklarowania galerii, na `export default [{….`

5. Nadal mamy błędy, ale to dlatego, że w interfejsie zadeklarowaliśmy, że galeria posiada **galleryId** – u nas jest po prostu **id** oraz zdjęcia i tagi, których w naszych danych nie ma.

   Na początek zmieniamy wszystkie id na galleryId.

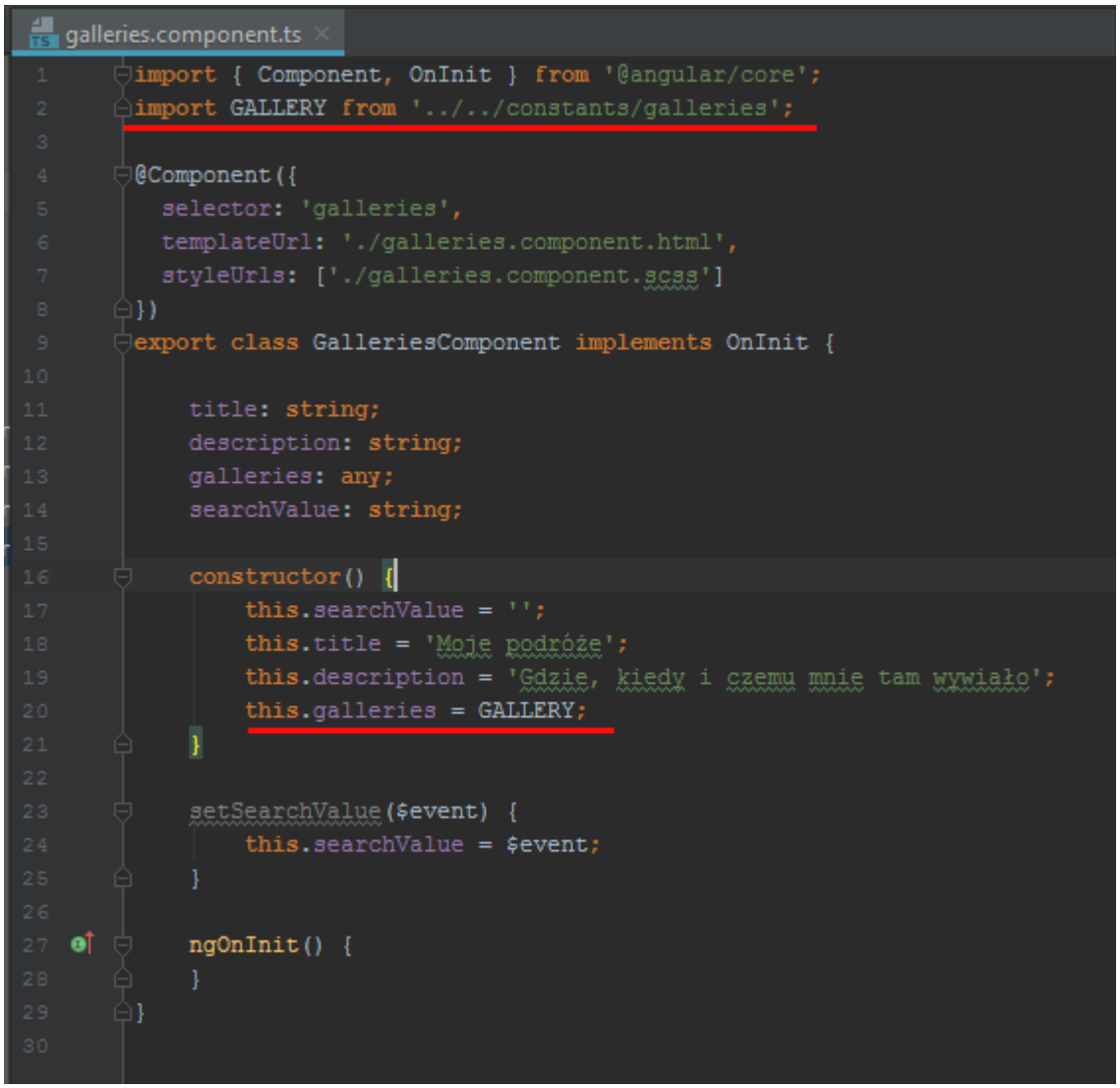
   Następnie dodajemy tagi i zdjęcia.

   Zrób to dla dwóch pierwszych galerii, do reszty dodaj tylko ‘photos’: []

galleries.ts

```ts
export default [{
        'galleryId': '1',
        'title': 'Rzym',
        'dateCreated': '2015-12-15T00:00:00+00:00',
        'thumbUrl': './assets/img/rzym-2015/1-sm.jpg',
        'description': 'Kilka dni zwariowanego wypadu do Rzymu.',
        'tags': [
            'Koloseum',
            'Palatyn',
            'Forum Romanum',
            'Watykan',
            'Bazylijka św. Piotra',
            'Most Saint Angelo',
            'Piaza Novona',
            'Panteon',
            'Fontanna di Trevi',
            'Ogrody w Tivoli',
            'Katakumby'
        ],
        'photos': [{
            'photoId': '1',
            'thumbUrl': './assets/img/rzym-2015/1-sm.jpg',
            'imgUrl': './assets/img/rzym-2015/1.jpg',
        }, {
            'photoId': '2',
            'thumbUrl': './assets/img/rzym-2015/2-sm.jpg',
            'imgUrl': './assets/img/rzym-2015/2.jpg',
        }, {
            'photoId': '3',
            'thumbUrl': './assets/img/rzym-2015/3-sm.jpg',
            'imgUrl': './assets/img/rzym-2015/3.jpg',
        }]
    }, {
        'galleryId': '2',
        'title': 'Maroko',
        'dateCreated': '2015-08-07T00:00:00+00:00',
        'thumbUrl': './assets/img/maroko-2015/1-sm.jpg',
        'description': 'Tydzień zwiedzania południowego Maroka z ojcem.',
        'tags': [],
        'photos': [{
            'photoId': '4',
            'thumbUrl': './assets/img/maroko-2015/1-sm.jpg',
            'imgUrl': './assets/img/maroko-2015/1-sm.jpg',
        }, {
            'photoId': '5',
            'thumbUrl': './assets/img/maroko-2015/2-sm.jpg',
            'imgUrl': './assets/img/maroko-2015/2-sm.jpg',
        }, {
```

6. Na koniec trzeba tylko zaimportować dane do **galleries.component.ts**



Ćwiczenie 2. Przygotowanie routingu

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 Single Page 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:

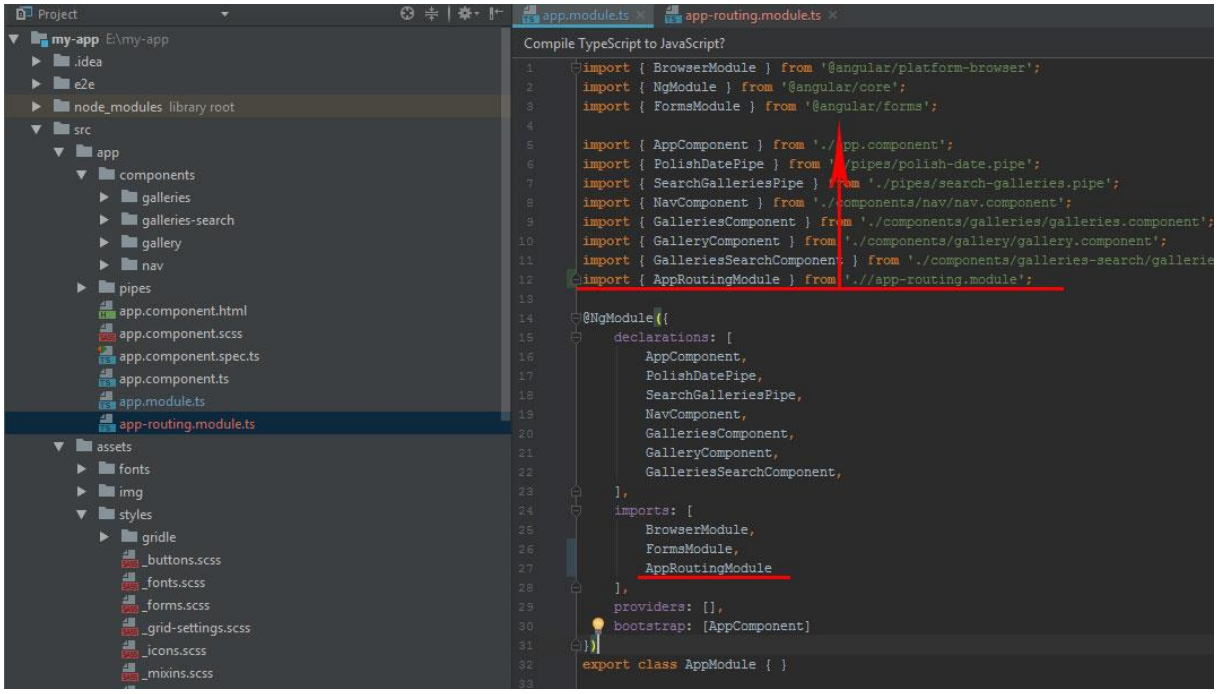
   ```
   ng generate module app-routing --flat --module=app
   ```

Przyjrzyjmy się tej komendzie.

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

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

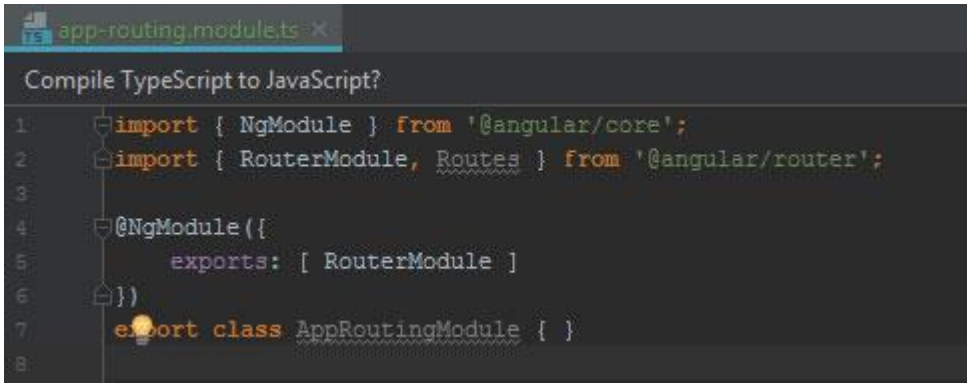
Jak widać poniżej, AppRoutingModule został dodany pod naszymi własnymi komponentami. Przenieśmy go wyżej, do sekcji z modułami, jako że to jedna z najważniejszych części naszej aplikacji.



2. Zajrzyjmy do modułu routingu. Nie wygląda imponująco:

Trzeba w nim kilka rzeczy zmienić. Nie będziemy w nim używać deklaracji komponentów, więc możemy usunąć linię oraz linie od 5 do 8 (włącznie).

3. Będziemy za to korzystać z Routes i RouterModule, więc musimy je dodać:



4. 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**. Zaimportujmy go, żeby router wiedział, gdzie go ma.

   ```
   import { GalleriesComponent }
       from './components/galleries/galleries.component';
   ```

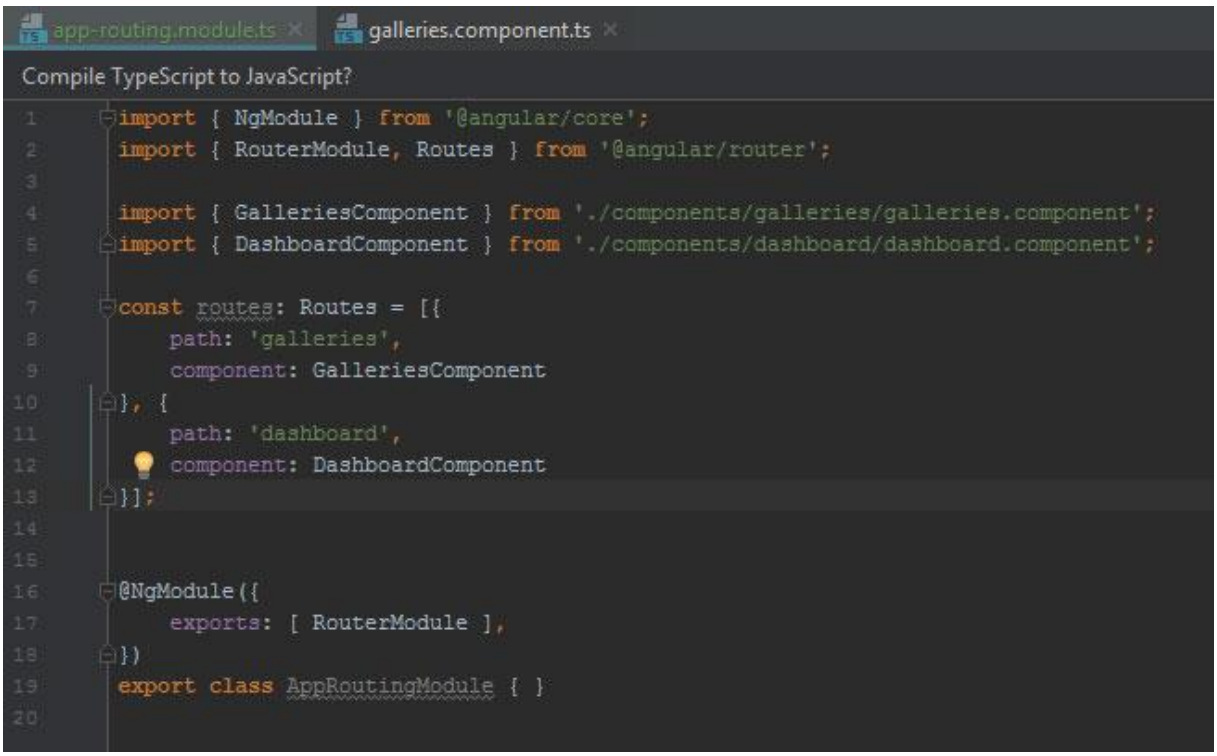
5. Typowa ścieżka posiada dwa parametry:
   path: odpowiada adresowi URL w przeglądarce
   component: component, który zostanie wywołany przez router pod podanym adresem.
   Dodajmy ją pod zaimportowanym komponentem:

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

6. No dobrze, a co jak nic nie kliniemy? Powinno pojawiać się coś domyślnie, prawda? Użyjemy do tego nowego komponentu **dashboard.component.ts**.

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

7. Podobnie jak poprzednio zaimportujmy go na górze pliku i dodajmy do niego ścieżkę.



8. Tylko jak ustawić, że **dashboard** jest uruchamiane domyślnie, jeśli nic nie klikniemy?

   Wystarczy na końcu dodać ścieżkę domyślną, wskazującą na **dashboard**:

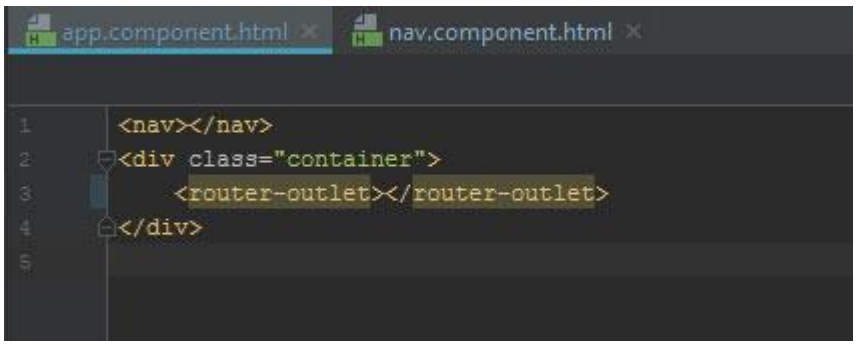
```
{
    path: '',
    redirectTo: '/dashboard',
    pathMatch: 'full'
}
```

   Jeśli path jest pusta, to przekieruj na dashboard. Proste!

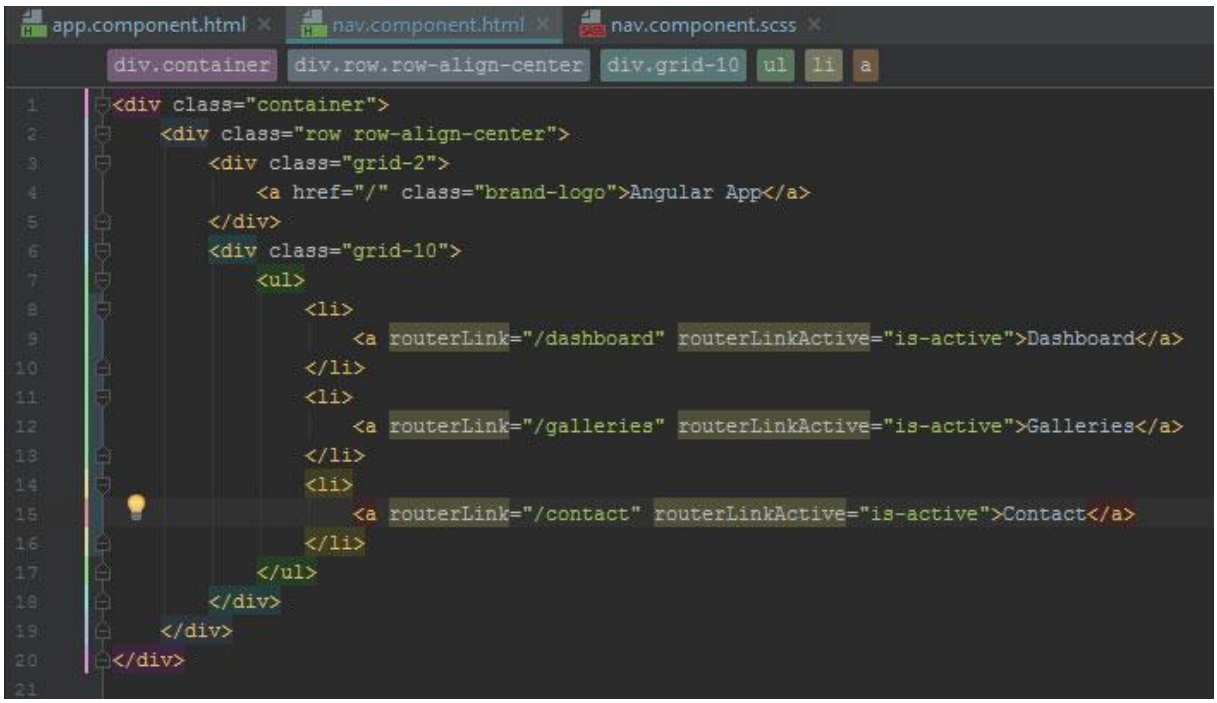
9. Mamy ścieżki, teraz musimy uruchomić router, żeby nasłuchiwał, czy w nie klikamy. W tym celu dodajemy router w @NgModule:

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

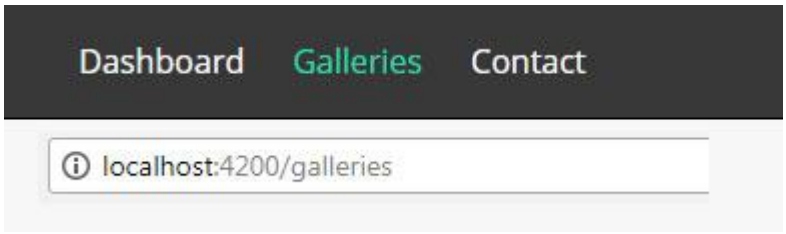
10. No i pozostało nam już tylko zmodyfikować kod html, tak, żeby w **app.component.html** zamiast komponentu `<galleries></galleries>` wczytywał komponenty ze ścieżek.
    Podmieńmy `<galleries></galleries>` na komponent routera `<router-outlet></router-outlet>`.

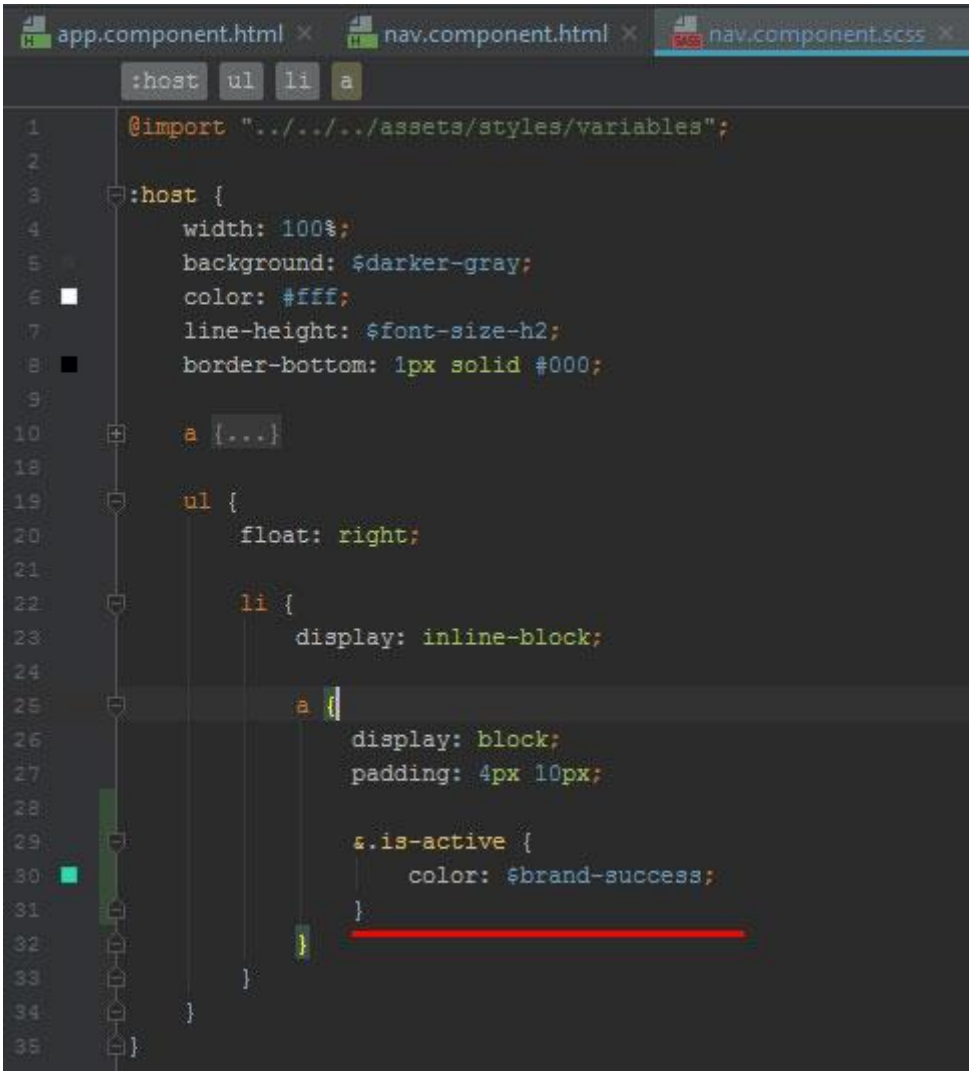


11. W nawigacji dodamy ścieżki do naszych podstron.



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





Ćwiczenie 3. 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. Tworzymy nowy komponent:

   ```
   ng generate component components/gallery-details
   ```

2. Przechodzimy do pliku **app-routing.module.ts** i dodajemy tam ten komponent, jak poprzednio.

3. W tablicy ze ścieżkami definiujemy nową ścieżkę, zawierająca parametr – id galerii, którą chcemy otworzyć.

   ```
   {
       path: 'galleries/:galleryId',
       component: GalleryDetailsComponent,
   }
   ```

4. Przechodzimy do pliku **gallery.component.html** i zamykamy zdjęcie w znaczniku <a> z nowododaną ścieżką:

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

5. Spróbujmy kliknąć – w adresie widać parameter i **gallery-details component works!**

Przechodzimy do komponentu **gallery-details**. Co musimy teraz zrobić, żeby wyświetlić dane galerii?

- Pobrać z adresu **galleryId**
- Wyciagnąć dane galerii
- Wyświetlić je na stronie

To po kolei.

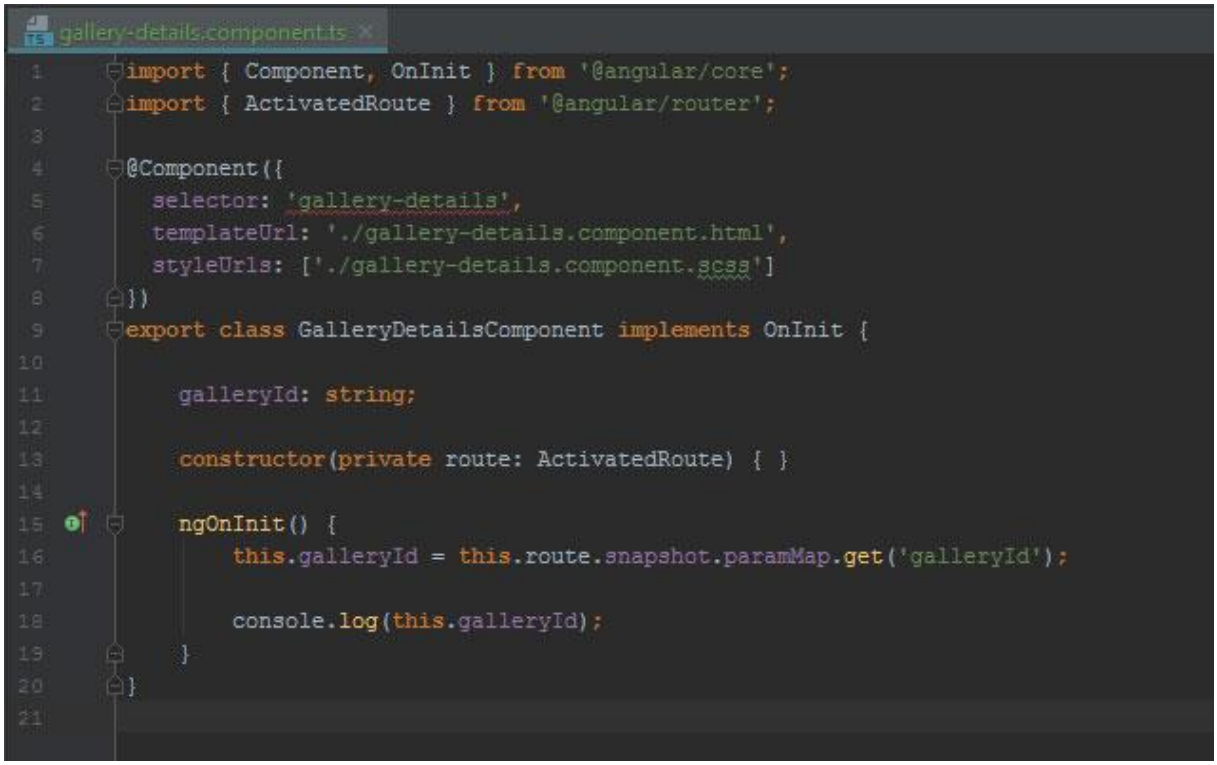
6. Zacznijmy od wyciagnięcia parametru z adresu. Potrzebujemy do tego zaimportować **ActivatedRoute** do komponentu.
   Na samej górze pliku **gallery-details.component.ts** dodajemy import:

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

7. W konstruktorze dopisujemy:

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

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



9. Skoro mamy galleryId możemy wyciągnąć dane całej galerii!

   Na początek definiujemy **gallery** i przypisujemy jej interface **IGallery**.

   ```
   gallery: IGallery;
   ```

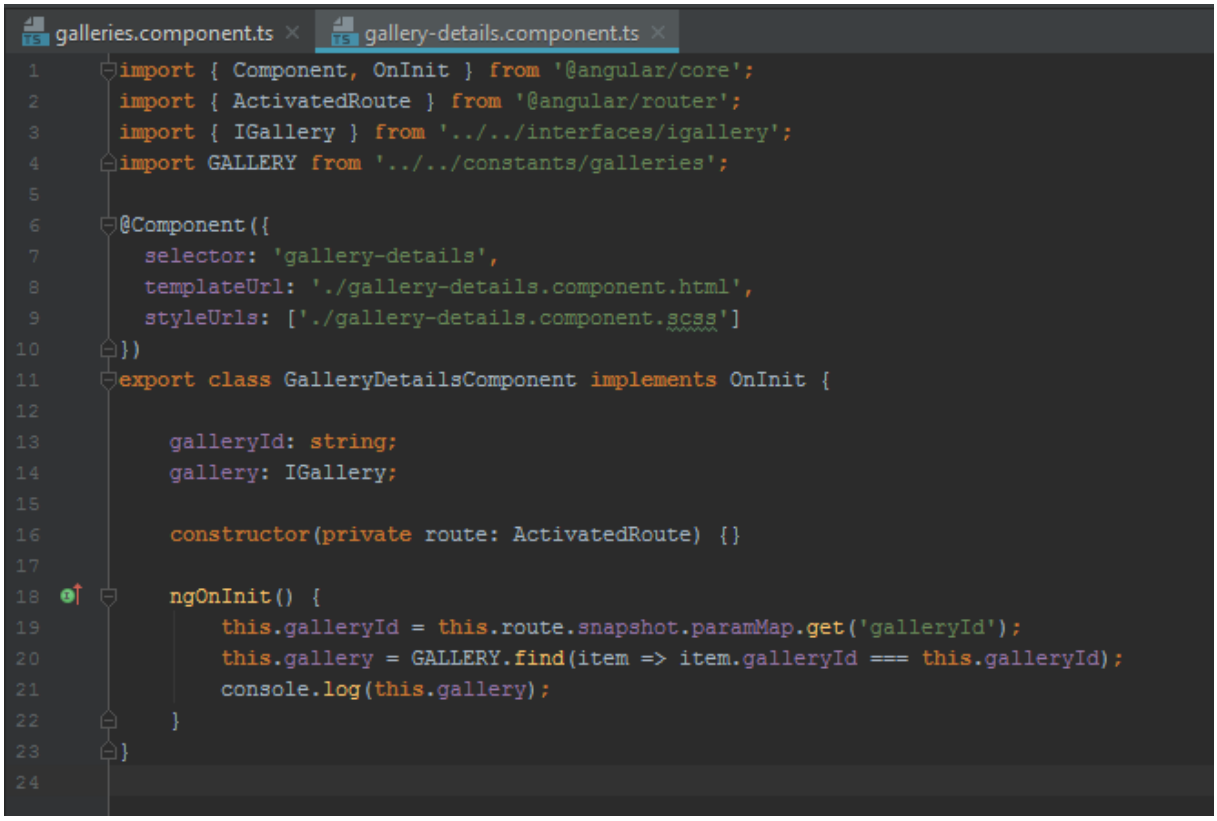
   Następnie w **ngOnInit()** dodajemy linię:

   ```
   this.gallery = GALLERY.find(item => item.galleryId ===
   this.galleryId);
   ```

   `console.log(this.galleryId)` zmieniamy na `console.log(this.gallery);`

   Co zrobiliśmy ostatnią linią? Pobraliśmy tablice galerii GALLERY i poszukaliśmy w niej obiektu, który miał takie samo **galleryId** jak to pozyskane z parametru z odnośnika.

Pamiętaj o zaimportowaniu interfejsu i pliku z galerami! Teraz jak klikniesz w którąś z galerii, w konsoli wyświetlą Ci się wszystkie jej dane.



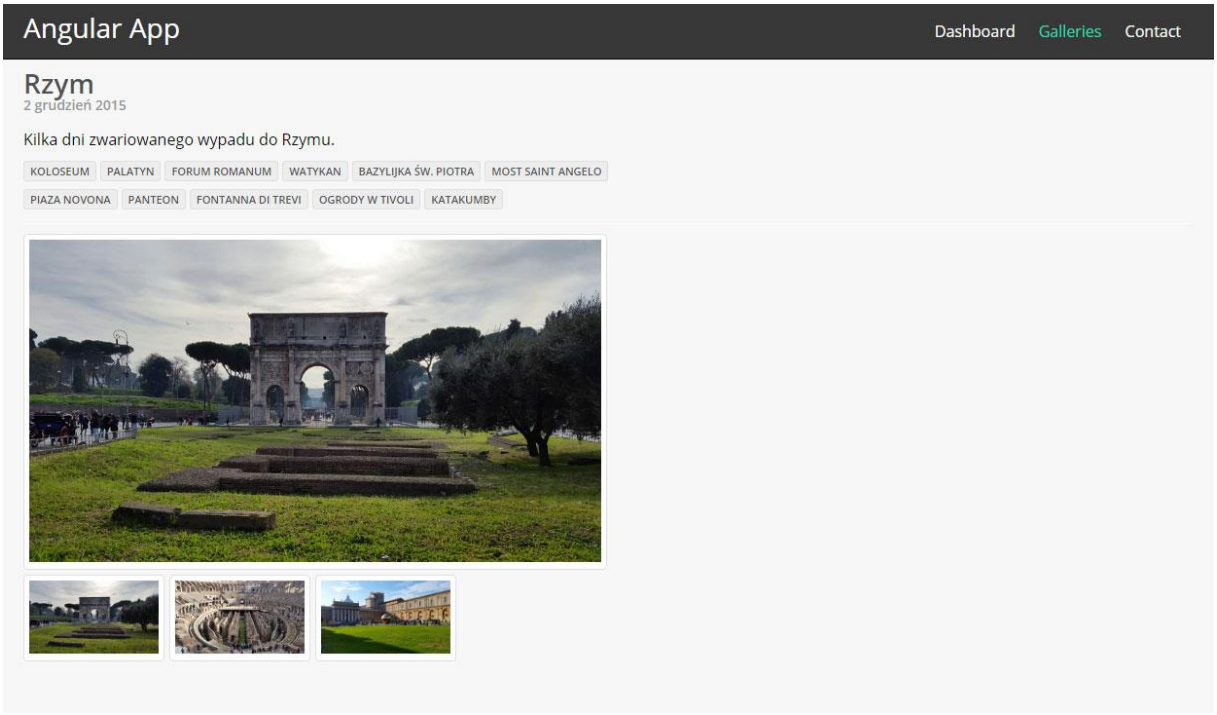
10. Ostatnia rzecz to wyświetlenie informacji o galerii na stronie. Nic trudnego – przejdź do pliku **gallery-details.component.html** i wpisz tam:

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

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:



3. Uzupełnij **dashboard.component.ts** o dowolny kontent, np. trzy najnowsze galerie, informacje o albumie, aktualności/wpisy (na razie podobnie jak galerie z pliku news.ts).