

Zadanie egzaminacyjne - portfolio zdjęciowe

Ćwiczenie 1. Szablon portfolio zdjęciowego

Stwórz nowy projekt o nazwie `portfolio_TwojeNazwisko`, następnie wykonuj czynności zapisane poniżej. Zaczynamy od stworzenia szablonu statycznego dla naszego portfolio. Zrobimy to w pliku: **`app.component.html`**

1. Przebuduj istniejący plik tak, żeby zawierał:
 - a. Element nawigacji `<nav>`.
 - b. Zawartość strony w `<section class="container">`.
 - c. Nagłówek H1 z tytułem strony.
 - d. Krótki tekst opisowy.
2. Korzystając z siatki gridle utwórz 2 rzędy po 3 kolumny i wstaw w nie zdjęcie, tytuł zdjęcia i kiedy zostało wykonane, oraz (na razie ręcznie) wpisz ilość elementów, np.:
- 3.

Moje podróże

Tu będziemy wstawiać elementy galerii oparte o AngularJS.

Ilość elementów: 6



Rzym

Kiedy: grudzień 2015



Maroko

Kiedy: sierpień 2014



Tajlandia

Kiedy: listopad 2013



Kambodża

Kiedy: listopad 2013



Turcja

Kiedy: czerwiec 2012



Chiny

Kiedy: październik 2011

Uwaga: Zdjęcia umieść w katalogu `assets/img` w katalogu *gallery*, żeby później łatwo było dodać kolejne.

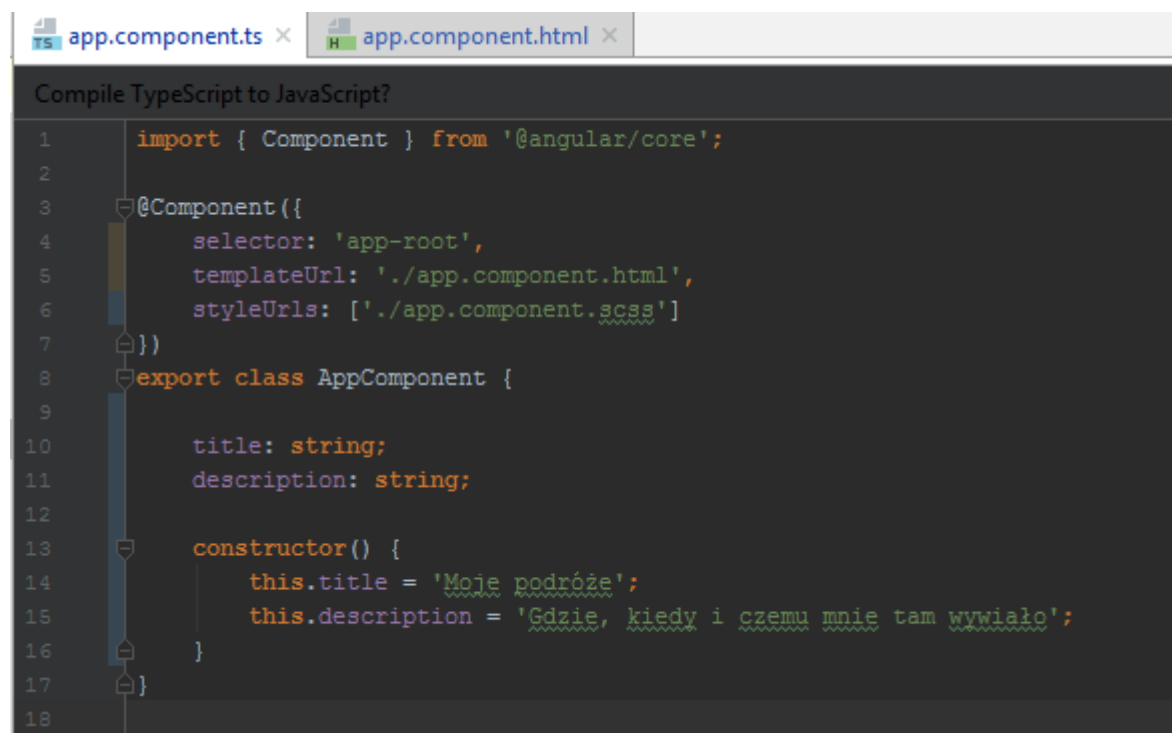
Wyobraźmy sobie, że zdjęć jest nie 6 ale 30. I chcemy dodać kolejne. A potem przenieść tytuł na górę zdjęcia. Za każdym razem musimy modyfikować liczbę zdjęć, dodawać kolejny kawałek kodu HTML, a każda zmiana w tym kodzie pociąga za sobą potrzebę zmiany we WSZYSTKICH 30+ jego wystąpieniach. Słabo...

Dzięki Angularowi możemy to znacznie usprawnić:

- kod HTML zdjęcia napisać tylko raz - dzięki temu zmiana położenia tytułu zostanie wprowadzona raz, a będzie dotyczyła wszystkich zdjęć.
- opis zdjęć umieścić w osobnym miejscu w postaci listy, w której łatwo je będzie dodawać,
- zrobić automatyczne zliczanie elementów.

Ćwiczenie 3. Zmienne w Angular

1. Na początek zdefiniujemy tytuł i opis. Zrobimy to w pliku **app.component.ts**.



```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.scss']
7  })
8  export class AppComponent {
9
10     title: string;
11     description: string;
12
13     constructor() {
14       this.title = 'Moje podróże';
15       this.description = 'Gdzie, kiedy i czemu mnie tam wywiało';
16     }
17   }
18
```

Przyjrzyjmy się, co tu mamy.

1. Na początek importujemy główny komponent Angulara.
2. Następnie definiujemy komponent **AppComponent**.

- Ustawiamy selektor na `app-root` – to znaczy, że w plikach html będziemy go wywoływać przez:

```
<app-root></app-root>
```

- Selektor zostanie wypełniony kodem html z pliku wstawionego do **templateUrl**.
 - Z kolei **styleUrls** pozwala nam ustawić plik, w którym będziemy pisać style tylko i wyłącznie dla tego komponentu.
3. Następnie tworzymy klasę **AppComponent** i ustawiamy w niej zmienne *title* i *description*. W konstruktorze klasy przypisujemy im wartości.
 2. Mamy zdefiniowane zmienne, пора je wyświetlić. W pliku **app.component.html** odzyskaj nagłówek `<h1>` i wpisz w nim **{{title}}**, zaś w miejscu opisu wstaw **{{description}}**.

To takie proste!

Ćwiczenie 4. Dyrektywy

Mimo wszystko, jest trochę biednie. Super by było, gdyby nie tylko tytuł i opis, ale i reszta informacji generowała się samodzielnie.

1. Na początek przeniesiemy więc galerię do tablicy w naszym komponencie. Na początek zdefiniujemy klasę dla obiektów galerii – tak zwany interface.

To ważne, w ten sposób określamy parametry obiektów galerii i ustalamy, jakie pola powinna posiadać, a jakich nie.

W katalogu **app** utwórz nowy katalog **interfaces**, a w nim plik o nazwie **IGallery.ts** – wielka litera i oznacza, że to interface.

Do pliku wstaw:

```
export class IGallery {
  galleryId: string;
  title: string;
  dateCreated: string;
  thumbUrl: string;
  description: string;
  tags: any;
  photos: any;
}
```

2. Teraz pora na plik z galeriami. W katalogu **app** utwórz kolejny katalog i nazwij go **constants** – w nim potrzebujemy nowego pliku **galleries.constant.ts** – umieścimy w nim tablicę z galeriami – podmień informacje na własne.

```
export const Galleries = [{
  'galleryId': '1',
  'title': 'Rzym',
  'dateCreated': '2015-12-15T00:00:00+00:00',
  'thumbUrl': './assets/img/galleries/rzym-2015/1-sm.jpg',
  'description': 'Kilka dni zwariowanego wypadu do Rzymu.',
  'tags': [],
  'photos': []
}, {
  'galleryId': '2',
  'title': 'Maroko',
  'dateCreated': '2015-08-07T00:00:00+00:00',
  'thumbUrl': './assets/img/galleries/maroko-2015/1-sm.jpg',
  'description': 'Tydzień zwiedzania południowego Maroka z ojcem.',
  'tags': [],
  'photos': []
}, {
  'galleryId': '3',
  'title': 'Tajlandia',
  'dateCreated': '2014-11-10T00:00:00+00:00',
  'thumbUrl': './assets/img/galleries/tajlandia-2014/1-sm.jpg',
  'description': 'Dwa tygodnie wycieczki po Tajlandii.',
  'tags': [],
  'photos': []
}]
```

Powyższa tablica zawiera tylko trzy obiekty. Zostawmy tak na razie.

Uzupełnijmy ją swoimi danymi.

Data została zapisana w formacie ISO8601, to preferowany format w języku Java Script:

```
JSON.stringify({'now': new Date()})
{"now":"2013-10-21T13:28:06.419Z"}
```

3. Pora to połączyć w pliku **app.component.ts**. Utwórzmy więc nową zmienną **galleries** typu **IGallery[]** – nawias kwadratowy oznacza, że będzie to tablica składająca się z elementów **IGallery**.

```
export class AppComponent{
  title: string;
  description: string;
  galleries: IGallery[];
```

Natomiast w konstruktorze przypiszemy do galleries nasze Galerie.

```
this.galleries = Galleries;
```

Zwróć uwagę, że Webstorm automatycznie dodaje import do wpisywanych rzeczy na górze pliku. Jeśli tego nie zrobił, to pewnie IGallery i Galleries są na czerwono – kliknij na nazwie i trzymając przycisk ALT wciśnij ENTER – powinieneś mieć opcję importu biblioteki lub biblioteka sama dodała się do importowanych elementów.

app.component.ts powinien teraz wyglądać tak:

```
import { Component } from '@angular/core';
import { IGallery } from '../interfaces/IGallery';
import { Galleries } from '../constants/galleries.constant';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title: string;
  description: string;
  galleries: IGallery[];

  constructor() {
    this.title = 'Moje podróże';
    this.description = 'Gdzie, kiedy i czemu mnie tam wywiało.';
    this.galleries = Galleries;
  }
}
```

4. Sprawdźmy czy działa. W pliku app.component.html w miejscu, gdzie mieliśmy ilość galerii wpiszmy:

```
{{galleries.length}}
```

Gotowe, możemy teraz wyświetlić nasze dane na stronie.

5. Angular zapewnia nam wbudowane mechanizmy do operowania na danych, tzw. dyrektywy.
Żeby wyświetlić dane z naszej tablicy wykorzystamy pętlę ***ngFor**, która pozwoli nam przejść przez wszystkie elementy tablicy i wygenerować dla nich kod na stronie.

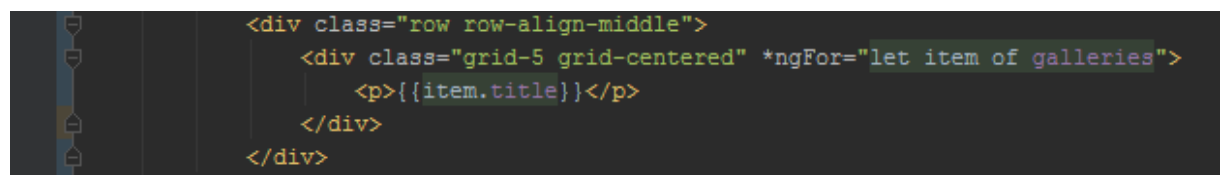
Prefiks (*) przed **ngFor** jest obowiązkowy – informuje aplikację, że element w którym się znajduje zostaje rozbudowany o dyrektywy Angulara.

Przejdźmy do pliku **app.component.html** i podmieńmy kod jednej z kolumn na ten poniżej.

```
<div class="grid-5 grid-centered" *ngFor="let item of galleries">
```

W ten sposób każdy obiekt z tablicy `galleries` zostaje przypisany do `item`.

Żeby wyświetlić tytuł w kolumnie, wystarczy wpisać `{{item.title}}`



6. Uzupełnij kolumnę o pozostałe dane: obrazek, datę, opis.
7. Na koniec dodamy jeszcze jeden element do naszej dyrektywy **track by**. Przy kolekcji obiektów, które chcemy modyfikować (dodawać, usuwać), Angular nie wie, które z nich się zmieniają i za każdym razem musi przebudowywać wszystkie.

Track by pozwala na określenie identyfikatora/kłucza do obiektów, dzięki czemu Angular wie, które obiekty ma zmieniać i przebudowuje tylko te, które się zmieniły.

Wstawmy więc **trackBy** do naszej dyrektywy.

```
<div class="grid-5 grid-centered"
*ngFor="let item of galleries; trackBy:item?.galleryId">
```

8. Inną dyrektywą strukturalną w Angularze jest dyrektywa ***ngIf** – sprawdźmy jak działa.

W **app.component.ts** zakomentuj linię zawierającą przypisanie galerii:

```
// this.galleries = GALLERY;
```

Otwórz stronę i CTRL+SHIFT+I – otwieramy narzędzia webmastera i przechodzimy do konsoli. Powinno być bardzo czerwono. To dlatego, że próbujemy działać na `this.galleries`, która nie istnieje.

9. Przejdźmy teraz do **app.component.html** i zastosujmy ***ngIf**.

```
<p *ngIf="galleries">Ilość galerii: {{galleries.length}}</p>
<div class="row row-align-middle" *ngIf="galleries">
```

Ponownie sprawdź w konsoli. Błędy zniknęły. Podobnie jak linia z ilością galerii.

10. Odkomentuj linię z `this.galleries` w **app.component.ts**. Aplikacja działa jak wcześniej.

***ngIf** to instrukcja warunkowa – jeśli wyrażenie jest prawdziwe (`this.galleries` istnieje) element zostanie dołączony do aplikacji. Jeśli nie – nie zostanie wywołany i nie zwróci błędów. **To bardzo przydatne!**

Więcej o dyrektywach strukturalnych przeczytasz tutaj:

<https://angular.io/guide/structural-directives>

Ćwiczenie 5. Filtry

Nasza data w tym momencie nie wygląda, co tu kryć. Pora ją trochę sformatować. Angular dostarcza szereg wbudowanych filtrów (pipes), z których możemy korzystać przy wyświetlaniu danych. Wstawiamy je po znaku |.

Pobawmy się więc datą: <https://angular.io/api/common/DatePipe>

1. Na początek sformatujemy ją do prostej formy: 2017-10-22

```
10 
11 <p><small>{{item.dateCreated | date: 'yyyy-MM-dd'}}</small></p>
12 <p>{{item.description}}</p>
```

2. Korzystając z innych formatów spróbuj sformatować datę do formatu:
22 październik 2017

Wszystko super, ale nasz miesiąc nie jest po polsku. Oprócz wbudowanych filtrów, mamy możliwość tworzenia też własnych. Zrobmy to.

W jaki sposób uzyskamy polską datę? To proste. Miesiący jest 12, w naszej dacie mamy numer miesiąca, wystarczy że utworzymy tablicę zawierającą polskie nazwy miesięcy i będziemy pobierać z niej element o numerze miesiąca -1. Czyli dla stycznia 0 element z tablicy, dla grudnia – 11 element.
Zaczynamy.

3. Otwieramy nowe okienko konsoli i wpisujemy:

```
ng generate pipe pipes/polishDate
```

Angular utworzy nam katalog z naszym pierwszym filtrem. Dodajmy od razu filtr do naszej daty w **app.component.html**

```
<p><small>{{item.dateCreated | polishDate}}</small></p>
```

4. Otwieramy plik **polish-date.pipe.ts**

5. Podobnie jak w przypadku komponentu u góry (nad @pipe) utworzymy sobie tablicę z nazwami miesięcy:

```
const MONTHS = ['styczeń', 'luty', 'marzec', 'kwiecień', 'maj',  
'czerwiec', 'lipiec', 'sierpień', 'wrzesień', 'październik',  
'listopad', 'grudzień'];
```

6. Poniżej wpisujemy:

```
8 export class PolishDatePipe implements PipeTransform {  
9  
10     date: Date;  
11  
12     transform(value: any, args?: any): any {  
13         console.log(value);  
14  
15         this.date = new Date(value);  
16         console.log(this.date);
```

Otwórz stronę i CTRL+SHIFT+I – otwieramy narzędzia webmastera i przechodzimy do konsoli. Powinno być widać naszą datę oraz datę po zmianie formatu.

7. Dodajmy kolejne dwie zmienne pod date: Date:

```
monthNumber: number;  
month: string;
```

8. A następnie pobierzmy numer miesiąca oraz jego nazwę.

```
transform(value: any, args?: any): any {  
  
    this.date = new Date(value);  
    console.log('fullDate', this.date);  
  
    this.monthNumber = this.date.getMonth();  
    console.log('monthNumber', this.monthNumber);  
  
    this.month = MONTHS[this.monthNumber];  
    console.log('month', this.month);  
  
    return this.date.getDate() + ' ' + this.month + this.date.getFullYear();  
}
```

9. Sprawdź działanie aplikacji.
10. Wykorzystaj kolejny filtr i zmień litery nazw miesiąca na duże korzystając z
| uppercase

Więcej o filtrach przeczytasz tutaj: <https://angular.io/guide/pipes#pipes>

Ćwiczenie 6. Filtrowanie danych

Oprócz wykorzystywania filtrów do formatowania tekstu i daty, możemy użyć ich też w bardziej kreatywny sposób. W naszej aplikacji dodamy wyszukiwarkę, która będzie przeszukiwała galerię i zwracała tylko te galerie, które zawierają konkretne słowa.

1. Na początek dodamy sobie pole input w pliku **app.component.html**.

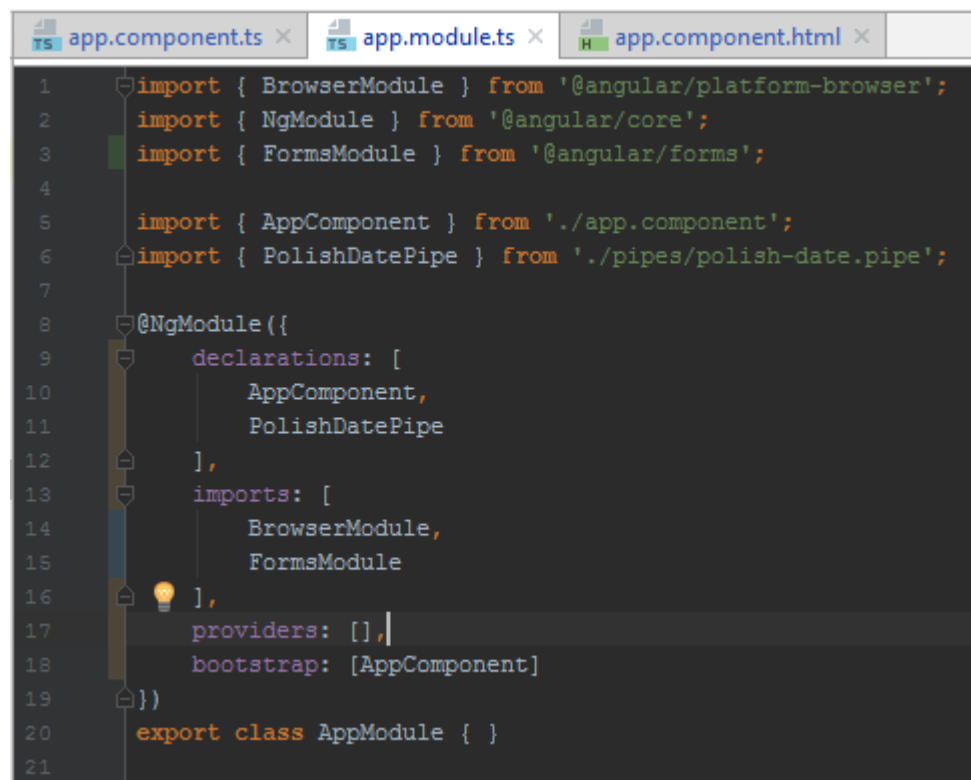
```
<input type="text" name="searchValue" [(ngModel)]="searchValue"
placeholder="Szukaj">
```

[(ngModel)] to zapis Angulara, który oznacza **two-way-databinding**, czyli jeśli wpiszemy coś w polu wyszukiwarki, zmiennej `this.searchValue` automatycznie zostanie przypisana ta wartość i vice versa, jeśli w **app.component.ts** przypiszemy jakąś wartość do `this.searchValue` – zmiana wyświetli nam się również w html.

2. Pod spodem dodaj linię:

```
<p>Szukam: {{searchValue}} </p>
```

3. Na razie jednak nic nie działa, a aplikacja sypie błędami. To dlatego, że aby korzystać z **ngModel** potrzebujemy modułu formularza, którego jeszcze w naszej aplikacji nie ma. Dopiszmy go w **app.module.ts**.



```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule } from '@angular/forms';
4
5 import { AppComponent } from './app.component';
6 import { PolishDatePipe } from './pipes/polish-date.pipe';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     PolishDatePipe
12   ],
13   imports: [
14     BrowserModule,
15     FormsModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

4. Przejdźmy teraz do pliku **app.component.ts**, gdzie zdefiniujemy *searchValue*.
Dodaj:

```
searchValue: string; oraz this.searchValue = '';
```

I spróbuj coś wpisać na stronie.

5. Wartość się zmienia, ale galeria nie. To dlatego, że musimy dopisać filtrowanie w **app.component.html**.

Do pola formularza **input** dopisz na końcu funkcję, którą będziemy wywoływać po każdym wciśnięciu przycisku:

```
(keyup) = „onSearchValue()”
```

6. Zdefiniujmy tę funkcję w **app.component.ts** – wstawiamy ją pod konstruktorem.

```
onSearchValue() {  
  this.galleries = Galleries;  
  
  if (this.searchValue) {  
    this.galleries = this.galleries.filter( callbackfn: (gallery: IGallery) => {  
      (gallery.title.indexOf(this.searchValue) !== -1) || (gallery.description.indexOf(this.searchValue) !== -1));  
    } else {  
      this.galleries = Galleries;  
    }  
  }  
}
```

Sprawdźmy jak to działa.

Działa! Ale... po co pisać funkcje filtrujące w komponencie, skoro można zrobić z nich pipe'a? Oczywiście, że go zrobimy!

7. W **app.component.html** wyrzucamy

```
(keyup) = „onSearchValue()”
```

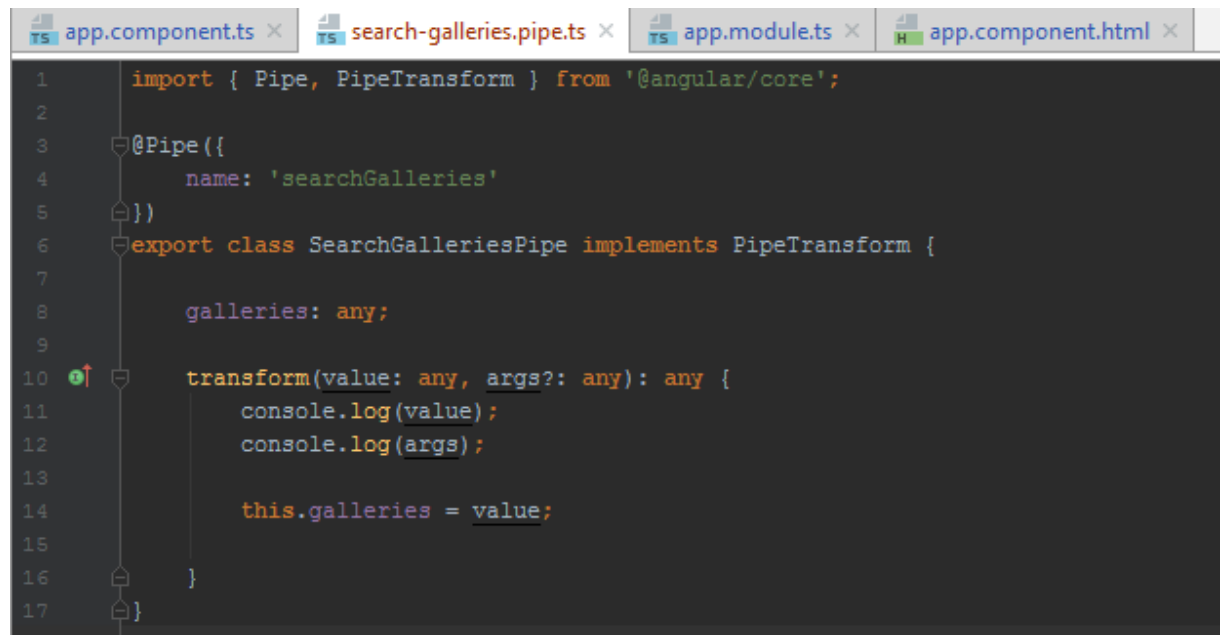
Za to ***ngFor** za **galleries** dopisujemy:

```
| searchGalleries : searchValue
```

8. Ponownie wpisujemy w konsoli komendę generującą pipe:

```
ng generate pipe pipes/searchGalleries
```

9. W pliku **search-galleries.pipe.ts** dodajmy zmienną **galleries** oraz wyświetlamy wartości przesyłane do filtra.



```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'searchGalleries'
5 })
6 export class SearchGalleriesPipe implements PipeTransform {
7
8   galleries: any;
9
10  transform(value: any, args?: any): any {
11    console.log(value);
12    console.log(args);
13
14    this.galleries = value;
15  }
16 }
17 }
```

10. Ok, mamy wszystko. Wystarczy przekleić zawartość funkcji z **app.component.ts** i podmienić `Galleries` i `this.searchValue` na `value` i `args`.

Na końcu należy dopisać `return this.galleries`.

11. Sprawdzić, czy działa. Usunąć funkcję z **app.component.ts**.

Zadanie samodzielne

1. Dodać pod polem `input` listę ul z latami i zmodyfikować filtr tak, żeby pozwalał filtrować galerię rocznikami.

Podpowiedź: Do li trzeba będzie dodać funkcję `onClick`.

2. Ostylować aplikację.