Zadania na ćwiczenia

Cele ćwiczenia:

- Zapoznanie się z budową projektu angularowego
- Zapoznanie się z budową komponentu webowego

Wykorzystywane narzędzia:

- Angular-cli: https://cli.angular.io/
- Node.js

1. Wygeneruj nowy projekt angularowy za pomocą komendy:

```
ng new blog-demo --skip-tests
```

Komenda utworzy podstawową hierarchie folderów oraz niezbędne pliki potrzebne do uruchomienia projekty pomijając testy jednostkowe.

Pliki które będą nas interesować to

- package.json - konfiguracja node'a

  Sprawdź jakie paczki są zainstalowane oraz jakie komendy są tam zadeklarowane.
- .angular-cli.json – konfigaracja dla angulara – tutaj będziemy zaglądać innym razem
- src/main.ts – kod do inicjalizacji projektu wykorzystującego moduł node'a `'@angular/platform-browser-dynamic'` - posiada on linijkę

```
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

Która jest niezbędna do uruchomienia projektu, a dokładniej do uruchomienia modułu angularowego o nazwie „AppModule"

- src/app/app.module.ts – skrypt posiadajacy definicję modułu AppModule – jest to pusta klasa -tutaj ważny dla nas bedzie tzw. dekorator klasy który posiada konfigurację modułu:

```
@NgModule({
  declarations: //=> deklaracje komponentów z których zbudowany jest moduł
  [
    AppComponent
  ],
  imports: //=> deklaracja modułów z których chcemy korzystać
          //tworząc własne rozwiązanie
  [
    BrowserModule
  ],
  providers: [], //=> klasy (serwisy) które chcemy udostępniać innym modułom
  bootstrap: [AppComponent]//=> komponent który ma się uruchomić
                           // zazwyczaj ten komponent jest rodzicem
wszystkich komponentów w module
})
```

```
export class AppModule { }
```

- src/app/app.component.ts – skrypt posiada klasę komponentu. Zwróć uwagę na dekorator klasy – jest w nim zawarta definicja web komponentu

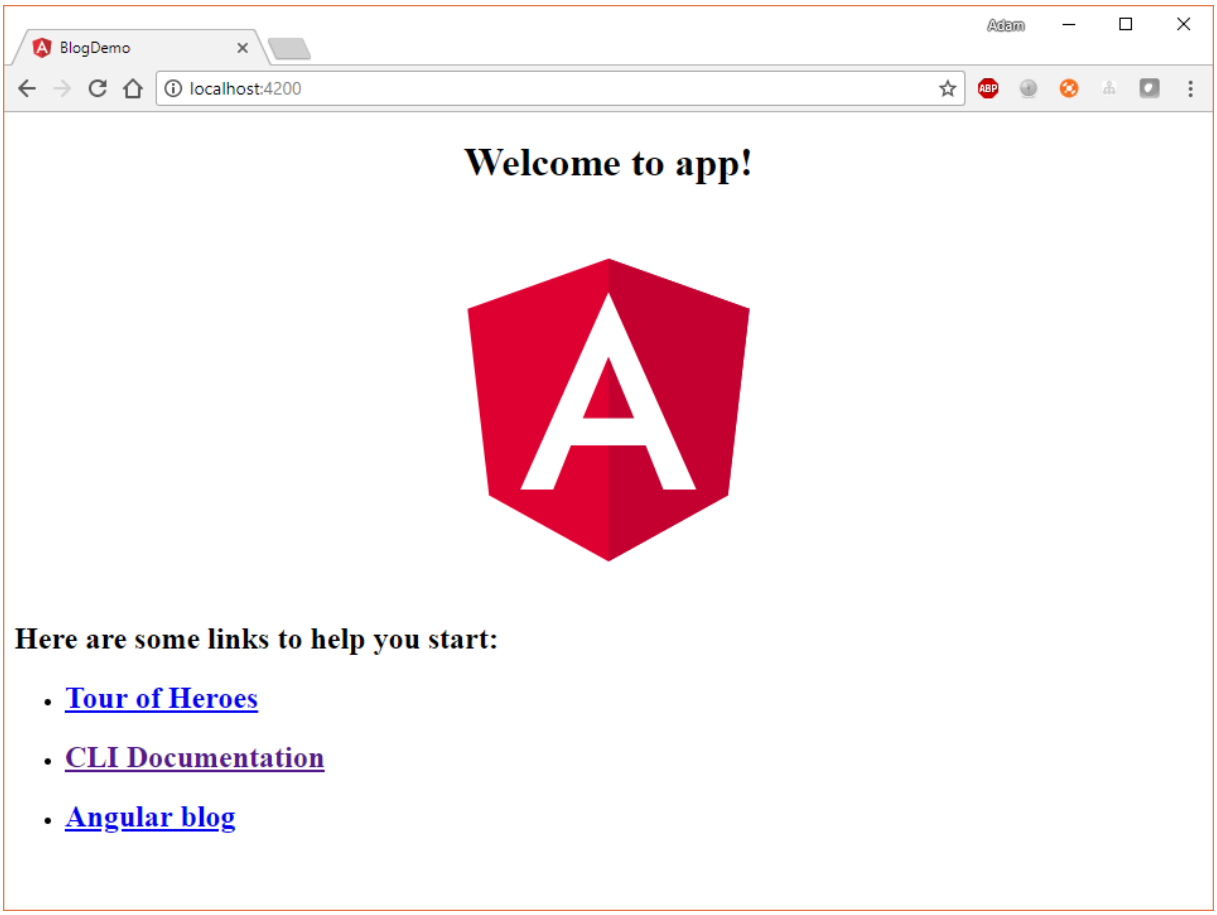
```
@Component({
  selector: 'app-root',//=> nazwa "taga" którą możemy się posłużyć
                       // gdy chcemy wyświetlić dany komponent na stronie
  templateUrl: './app.component.html',//=> lokalizacja pliku html który
definije widok komponentu
  styleUrls: ['./app.component.css']//=> lokalizacja plików z klasami css,
                  //które mają być użyte tylko i wyłącznie w kodzie html
komponentu
})
export class AppComponent {
  title = 'app';
}
```

- src/index.html – strona iternetowa która zostanie wyświetlona użytkownikowi – kod jest bardzo prostu, ponieważ to co wyświetla to komponent „app-root”

```
<body>
  <app-root></app-root> <!-- tutaj wyświetlony jest komponent -->
</body>
```

Możesz uruchomić aplikację komendą

```
ng serve
```



Zwróć uwagę, że przeglądarka wygenerowała widok opisany w kodzie html komponentu app.

Zwróć uwagę na linijkę (src/app/app.component.html)

```
<h1>
  Welcome to {{title}}!
</h1>
```

Podwójne nawiasy klamerkowe pozwalają na sczytywanie wartości pola w klasie komponentu – zauważ, że w klasie komponentu znajduje się jedno pole title które ma wartość „app” – sprawdź co się stanie, jeśli zmienisz wartość tego pola.

2. Dodanie nowego komponentu.

Komponent angularowy składa się z kilku elementów:

- Klasy komponentu z dekoratorem @Component, w którym musi być zadeklarowana unikalna nazwa komponentu
- Templatu html -może być w osobnym pliku, ale również w dekoratorze klasy (o ile jest bardzo krótki)
- Pliku z klasami css (opcjonalnie)
- Testy jedostkowe (opcjonalnie)
- Modelu dla komponentu (opcjonalnie)

Ja widać definicja całego web komponentu może zawierać się w wielu plikach, stąd dobrą praktyką jest tworzenie odrębnego folderu dla każdego komponentu.

Do katalogu src/app dodajmy folder ‘blog-post’ – w nim będziemy przetrzymywać pliki komponentu. Tym komponentem będziemy mogli zarządzać treścią konkretnego posta.

Pliki które będą nam potrzebne:

- blog-post.component.ts – skrypt z definicją komponentu
- blog-post.component.html – widok komponentu
- blog-post.ts – skrypt z klasą modelu

zacznijmy od prostego templatu html

```
<h1>Tytuł</h1>
<h2>data dodania</h2>
<span>treść</span>
```

Oraz deklaracji komponentu

```
import { Component } from "@angular/core";

@Component({
    selector:'blog-post',
    templateUrl:'./blog-post.component.html'
})
export class BlogPostComponent{}
```

każdy nowy komponent należy zarejestrować w decoratorze modułu (plik src/app.module.ts)

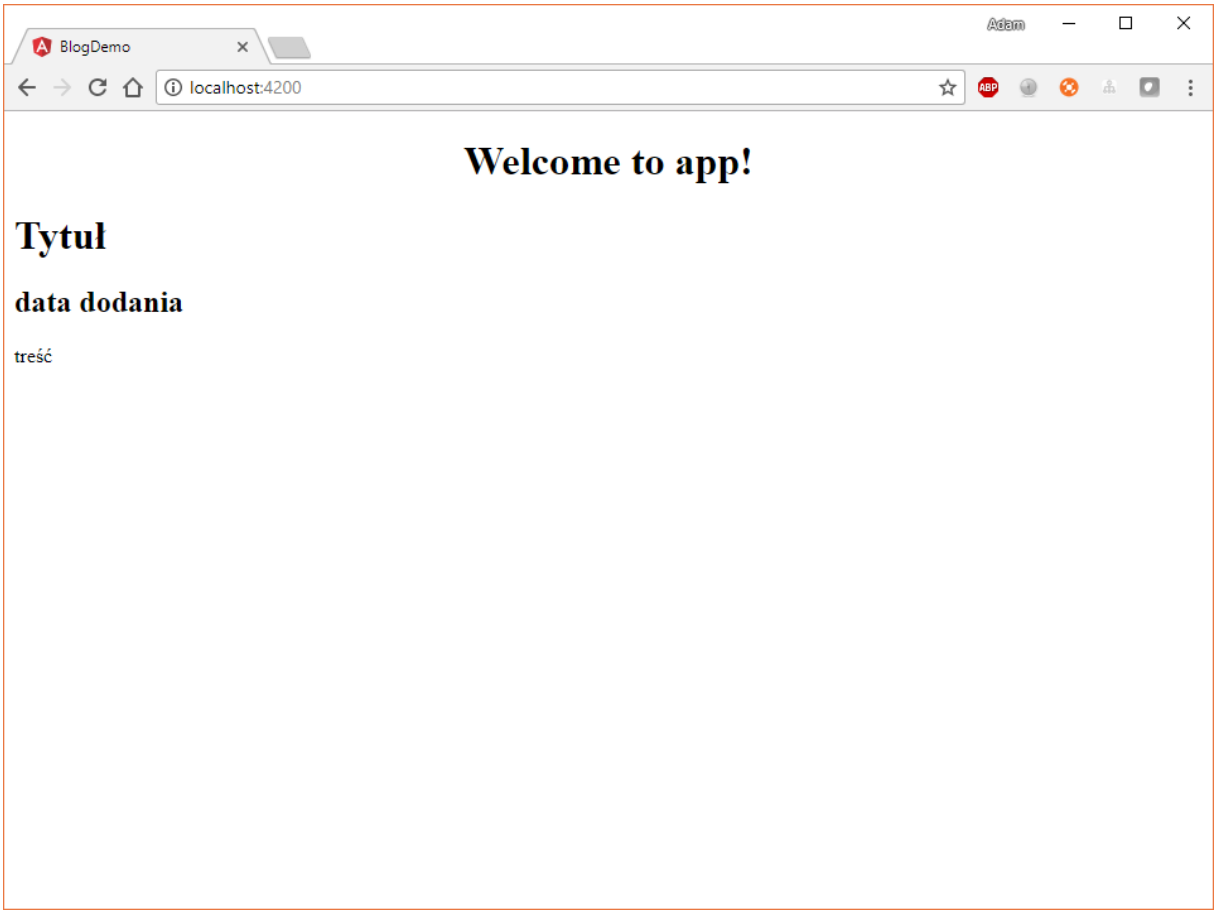
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { BlogPostComponent } from './blog-post/blog-post.component';

@NgModule({
  declarations:
  [
    AppComponent,
    BlogPostComponent //=> o tutaj !!
  ],
  imports:
  [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Zadanie do samodzielnego wkonania. Utwórz komponent o nazwie post-commment (tylko zestaw plików bez zawartości) oraz zarejestruj go w module.

teraz możemy przetestować, czy nasz komponent się wyświetli na stronie – aby to zrobić w pliku app.component.html zamienimy logo oraz linki do podstron dokumentacji angulara na taga z nazwą naszego koomponentu

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
<blog-post></blog-post>
```

W przeglądarce powinna się teraz pokazać następująca strona



Teraz możemy zająć się logiką komponentu

Zacznijmy od stworzenia modelu dla komponentu (blog-post.ts)

```
export class BlogPost{
    constructor(
        public title:string,
        public date: Date,
        public content:string,
        public comments:Array<string>=[]){}
}
```

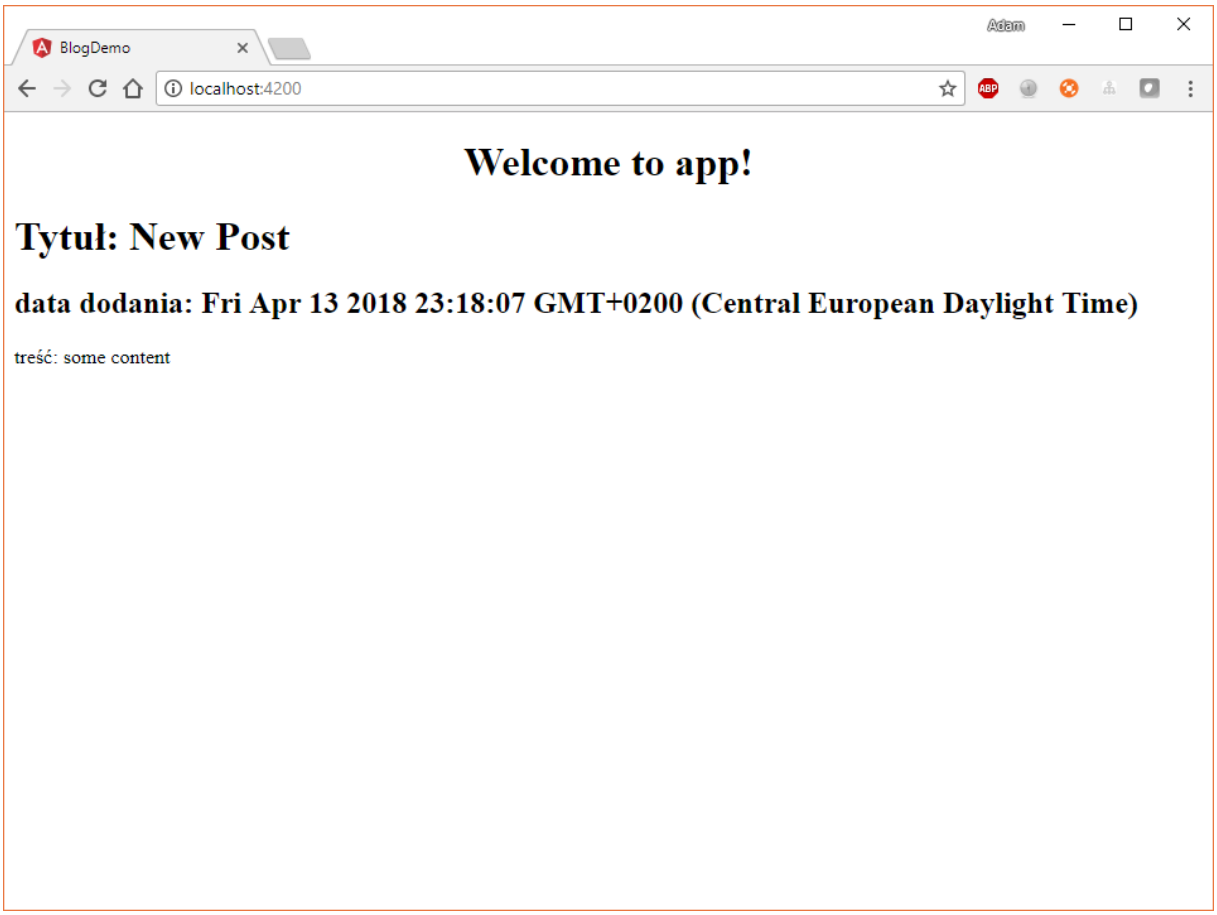
W klasie kompoentu (blog-post.component.ts) zadeklarujmy nowe pole typu BlogPost – w interfejsie użytkownika będziemy chcieli wyświetlić przykładowe pola z klasy modelu

```
import { Component } from "@angular/core";
import { BlogPost } from "./blog-post";

@Component({
    selector:'blog-post',
    templateUrl:'./blog-post.component.html'
})
export class BlogPostComponent{

    post = new BlogPost(
        "New Post",
        new Date(),
        "some content",
        ["comment 1", "comment 2"]
    );

}
```

W kodzie html wykorzystamy nawiasy klamrowe, aby wyświetlić dane posta

```
<h1>Tytuł: {{post.title}}</h1>
<h2>data dodania: {{post.date}}</h2>
<span>treść: {{post.content}}</span>
```

Powinniśmy otrzymać następujący efekt



Jest w miare dobrze, ale lepiej by wyglądało gdybyśmy mogli trochę inaczej sformatować date w tym celu mozna się posłużyć operatorem | (pipe) i formatem dla dat
https://angular.io/api/common/DatePipe

```
<h1>Tytuł: {{post.title}}</h1>
<h2>data dodania: {{post.date | date:'shortDate'}}</h2>
<span>treść: {{post.content}}</span>
```

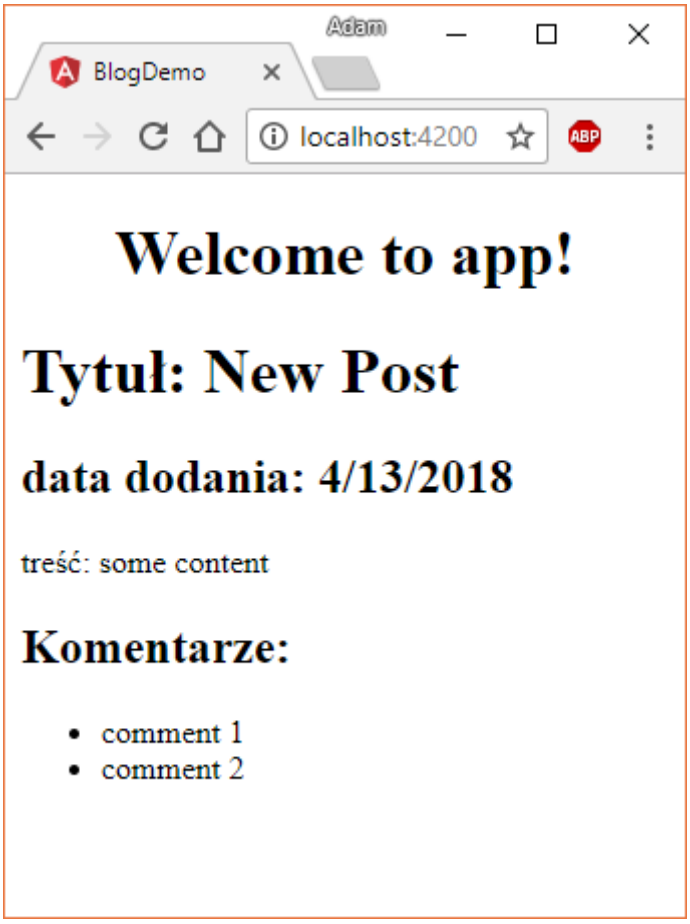
Więcej informacji na temat formatowania danych przy pomocy pipe'a :
https://angular.io/guide/pipes

3. Przekazywanie modelu między komponentami oraz dyrektywy *ngFor.

Na stronie naszego posta nie są wyświetlane komentarze – aby to zrobić wykorzystamy dyrektywę *ngFor, która na podstawie zadanej kolekcji wygenerować kod html

```
<h1>Tytuł: {{post.title}}</h1>
<h2>data dodania: {{post.date | date:'shortDate'}}</h2>
<span>treść: {{post.content}}</span>
<h2>Komentarze:</h2>
<ul>
    <li *ngFor="let comment of post.comments">{{comment}}</li>
</ul>
```

To wystarczy aby uzyskać następujący efekt:



Zajmijmy się teraz przekazywaniem modelem miedzy komponentem posta a komponentem komentarza.

Plik html będzie raczej prosty (post-comment.component.html):

```
<p>
  {{comment}}
</p>
```

Tak samo jak klasa komponentu z tym, że dla pola comment użyjemy dekoratora @Input – pozwala on na przekazanie wartości do pola klasy z innegoo komponentu (post-comment.component.ts)

```
@Component({
  selector: 'post-comment',
  templateUrl: './post-comment.component.html',
  styleUrls: ['./post-comment.component.css']
})
export class PostCommentComponent {

  @Input() comment:string;
}
```

Teraz możemy zmienić kod html komponentu posta tak aby użył komponentu do wyświetlania komentarza

```
<h1>Tytuł: {{post.title}}</h1>
<h2>data dodania: {{post.date | date:'shortDate'}}</h2>
<span>treść: {{post.content}}</span>
<h2>Komentarze:</h2>
<ul>
    <li *ngFor="let comment of post.comments">
        <post-comment [comment]="comment"></post-comment>
    </li>
</ul>
```

4. Obsługa interakcji z użytkownikiem.

Kolejnym etapem będzie utworzenie prostej interakcji z użytkownikiem, polegającej na włączeniu/wyłączeniu trybu edycji posta– w związku z tym stworzymy pole tekstowe które będzie związane z polem tytułu posta, jednakże aby móc używać mechanizmu wiązania należy w module AppModule (app.module.ts), zadeklarować użycie modułu „FormsModule” z @angular/forms

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

import { AppComponent } from './app.component';
import { BlogPostComponent } from './blog-post/blog-post.component';
import { PostCommentComponent } from './post-comment/post-comment.component';
import {FormsModule} from '@angular/forms';
@NgModule({
  declarations:
  [
    AppComponent,
    BlogPostComponent,
    PostCommentComponent //=> o tutaj !!
  ],
  imports:
  [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

przejdźmy do kodu html komponentu posta i dodajmy tam pole tekstowe z atrybutem [(ngModel)]=”post.title”

```
<h1>Tytuł: {{post.title}}</h1>
<input type="text" [(ngModel)]="post.title"/>
<h2>data dodania: {{post.date | date:'shortDate'}}</h2>
<span>treść: {{post.content}}</span>
<h2>Komentarze:</h2>
<ul>
    <li *ngFor="let comment of post.comments">
        <post-comment [comment]="comment"></post-comment>
    </li>
</ul>
```

Przetestuj działanie w przeglądarce – teraz tytuł posta będzie się dynamicznie zmieniał wraz ze zmianami dokonanymi w polu tekstowym.

Zobaczmy teraz jak teraz oprogramować widoczność tego pola – chcielibyśmy aby na stronie był przycisk który by pokazywał/ukrywał te pole teksowe. To co należy zrobić to:

- Dodać pole boolowskie w klasie komponentu posta

```
@Component({
    selector:'blog-post',
    templateUrl:'./blog-post.component.html'
})
export class BlogPostComponent{

    isInEditMode = false;

    post = new BlogPost(
        "New Post",
        new Date(),
        "some content",
        ["comment 1", "comment 2"]
    );

}
```

- Dodać metodę która zmieni wartość tego pola na przeciwną

```
@Component({
    selector:'blog-post',
    templateUrl:'./blog-post.component.html'
})
export class BlogPostComponent{

    isInEditMode = false;

    post = new BlogPost(
        "New Post",
        new Date(),
        "some content",
        ["comment 1", "comment 2"]
    );
```

```
    toggleEditMode():void{
        this.isInEditMode=!this.isInEditMode;
    }
}
```

- W kodzie html komponentu zastosować dyrektywę *ngIf która przyjmuje owolne wyrażenie boolowskie od którego zależy czy dany element się wygeneruje na stronie bądź nie

```
<h1 *ngIf="!isInEditMode">Tytuł: {{post.title}}</h1>
<input *ngIf="isInEditMode" type="text" [(ngModel)]="post.title"/>
<h2>data dodania: {{post.date | date:'shortDate'}}</h2>
<span>treść: {{post.content}}</span>
<h2>Komentarze:</h2>
<ul>
    <li *ngFor="let comment of post.comments">
        <post-comment [comment]="comment"></post-comment>
    </li>
</ul>
```

- Do strony dodać przycisk który będzie połączony do metody toggleEditMode – tutaj trzeba będzie się podpiąć pod zdarzenie click, aby to zrobić należy wykorzystać atrybut (click)=”toggleEditMode()”

```
<button (click)="toggleEditMode()">edytuj</button>
<h1 *ngIf="!isInEditMode">Tytuł: {{post.title}}</h1>
<input *ngIf="isInEditMode" type="text" [(ngModel)]="post.title"/>
<h2>data dodania: {{post.date | date:'shortDate'}}</h2>
<span>treść: {{post.content}}</span>
<h2>Komentarze:</h2>
<ul>
    <li *ngFor="let comment of post.comments">
        <post-comment [comment]="comment"></post-comment>
    </li>
</ul>
```

Sprawdź działanie strony.

Zadania do samodzielnego wykonania:

- Dodaj model komentarza który powinien zawierać nick osoby, date dodania oraz tekst (użyj go w modelu posta)
- Dodaj formularz dodawania komentarza dla powyższych danych
- Dodaj możliwość usuwania/edytowania pojedynczego komentarza
- Dodaj możliwość odpowiedzi na komentarz (za 1 dodatkowy punkt)