Laboratorium 7: Comments API

Tym razem celem laboratoriów będzie wykorzystanie API do zapisu komentarzy do galerii. Wykorzystamy do tego rozwiązania, które opracowaliśmy wcześniej w laboratorium o RESTful APi.

Do dyspozycji mamy tym razem:

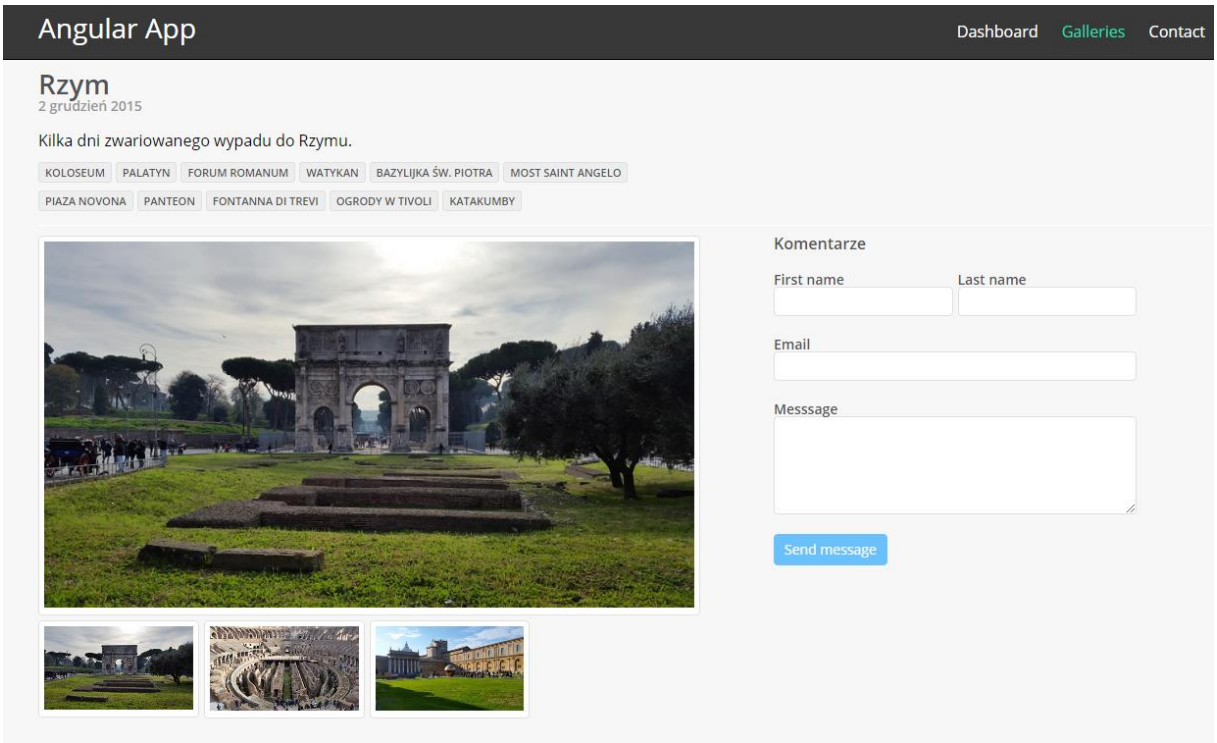
POST 'http://project.usagi.pl/comment' – tworzy komentarz

GET 'http://project.usagi.pl/comment/byGallery/{galleryId}' – pobiera wszystkie komentarze w galerii

POST 'http://project.usagi.pl/comment/delete/{commentId}' - usuwa komentarz

Na początek stworzymy formularz do dodawania komentarzy oraz mechanizmy odpowiadające za ich usuwanie z listy. W zadaniu wykorzystamy możliwości Angulara w zakresie walidacji formularzy.

Strona powinna później wyglądać mniej więcej tak:



Ćwiczenie 1. Komponent formularza

1. Na początek utworzymy nowy komponent *comment-form*.

   ```
   ng generate component components/comments/comment-form
   ```

2. Wstawimy go w **gallery.component.html** i przekażemy do niego galleryId, żeby podczas zapisu było wiadomo, która galeria została skomentowana.

```
<api-comment-form [galleryId]="gallery.galleryId"></api-comment-form>
```

3. W kolejnym kroku zdefiniujemy interfejs komentarza IComment.ts:

```
export interface IComment {
    galleryId: string;
    commentId?: string; //zostanie dodany podczas zapisu do bazy
    firstName: string;
    lastName: string;
    email: string;
    message: string;
    dateCreated: Date;
}
```

4. Tym razem nie będziemy mieć gotowej listy komentarzy. Będziemy za to potrzebować pustego obiektu comment, do którego odnosi się formularz. Wstawimy go w komponencie **comment-form.component.ts** w ngOnInit():

```
ngOnInit() {
    this.comment = this.setEmptyComment();
}

private setEmptyComment() {
    const newDate = new Date();

    return {
        galleryId: this.galleryId,
        firstName: '',
        lastName: '',
        email: '',
        message: '',
        dateCreated: newDate
    };
}
```

   Dopiszmy w ngOnInit

   ```
   console.log(this.comment)
   ```

Ćwiczenie 2. Formularz

1. Teraz czas na formularz. Budujemy go prawie tak jak zawsze w HTML.
   Zbuduj formularz! Będzie miał pola: Imię, nazwisko, email i wiadomość – jak na zrzucie ekranu powyżej.

   Ważnych jest kilka zmian. W znaczniku form musimy powiedzieć Angularowi, że to jego formularz i nadać mu nazwę:

```
<form #commentForm="ngForm" (ngSubmit)="onSubmit(commentForm)" novalidate>
```

#commentForm to nasza nazwa formularza, zaś ngSubmit –funkcja wywoływana przy wysyłaniu formularza, po naciśnięciu przycisku.

2. Następnie musimy połączyć pola formularza z naszym obiektem komentarzy. Tu przykład dla firstName.

```
<input type="text" class="input-control" name="firstName"
#firstName="ngModel" [(ngModel)]="comment.firstName" required>
```

Już coś w tym stylu robiliśmy wcześniej przy przeszukiwaniu galerii. Teraz musieliśmy jeszcze powiedzieć Angularowi, jak chcemy żeby identyfikował pola #firstName.

Popraw pozostałe pola!!

3. Teraz musimy dodać przycisk do wysyłania danych.

```
<button class="button button-primary"
[disabled]="!commentForm.form.valid">Send message</button>
```

Zwróć uwagę na [disabled] –ten zapis oznacza, że formularz wyśle się tylko, gdy będzie poprawny. Pamiętaj, że wszystkie pola formularza, muszą być required!

4. Na koniec zdefiniujemy naszą funkcje onSubmit() w komponencie comment-form.component.ts:

```
onSubmit(commentForm) {
    console.log('submited!', this.comment);
}
```

Na razie niewiele robi, ale po wysłaniu formularza, wyświetli nam dane w konsoli.

Cały komponent powinien wyglądać tak:

comment-form.component.ts | comment-form.component.html | _buttons.scss

```ts
import { Component, Input, OnInit } from '@angular/core';
import { IComment } from '../../../interfaces/IComment';

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

    @Input() galleryId: string;

    comment: IComment;

    constructor() {}

    ngOnInit() {
        this.comment = this.setEmptyComment();
    }

    onSubmit(commentForm) {
        console.log('comment', this.comment);
    }


    private setEmptyComment() {
        const newDate = new Date();

        return {
            galleryId: this.galleryId,
            firstName: '',
            lastName: '',
            email: '',
            message: '',
            dateCreated: newDate
        };
    }
}
```

Ćwiczenie 3. Sprawdzanie poprawności

1. Wspomniałam wcześniej, że Angular potrafi kontrolować wartości w formularzu. Wykorzystuje do tego kilka stanów: https://angular.io/guide/forms#track-control-state-and-validity-with-ngmodel

   Otwórz konsolę w przeglądarce (Prawym myszy + Zbadaj) i zobacz kod html dla formularza. Wpisz różne wartości. Zauważ, że Angular dopisuje do formularza różne stany. Dzięki temu możemy kontrolować wprowadzane dane.

2. Wykorzystamy to u siebie.

   Pod każdym polem w formularzu dodajmy linię z błędem. Linia ta będzie ukryta, jeśli pole jest poprawne.

```
<p [hidden]="firstName.valid || firstName.pristine">First name
required!</p>
```

3. Pole email może mieć nie tylko błąd, jeśli jest puste, ale także, gdy ktoś wpisał niepoprawny email. Zapewnijmy sprawdzanie obu przypadków przez dodanie do pola argumentu pattern. Jeśli email nie będzie mu odpowiadał, wyświetli błąd.

```
<label>Email</label>
<input type="email" class="input-control" name="email" #email="ngModel"
[(ngModel)]="comment.email" required pattern="[a-z0-9!#$%&'*+\/=?^_`{|}~.-
]+@[a-z0-9]([a-z0-9-]*[a-z0-9])?(\.[a-z0-9]([a-z0-9-]*[a-z0-9])?)*">

<p class="form-error" *ngIf="email.invalid && !email.pristine &&
!email.dirty">Email required!</p>

<p class="form-error" *ngIf="email.invalid && !email.pristine &&
email.dirty">Wrong email!</p>
```

Uzupełnij pozostałe pola.

Sprawdź teraz działanie formularza.

Ćwiczenie 4. Zapis komentarza

1. Zapiszmy teraz komentarz do bazy. Oczywiście w komponencie comments-form potrzebujesz httpOptions oraz klienta http, tak jak robiliśmy to wcześniej podczas dodawania Galerii.

   W funkcji onSubmit() dodaj komentarz do bazy danych wykorzystując API POST: 'http://project.usagi.pl/comment' – tworzy komentarz

```
this.http.post(`http://project.usagi.pl/comment`, this.comment,
this.httpOptions).toPromise().then((response: IComment) => {
    console.log(response);
});
```

   to jednak nie wszystko –nasz formularz nadal przechowuje dane, dobrze by było je usunąć, dlatego pod `console.log(response)` dodamy jeszcze resetowanie formularza:

```
this.comment = this.setEmptyComment();
```

1. **Zadanie samodzielne**

   Pod szczegółami galerii wyświetl wszystkie komentarze.

   Wykorzystaj GET 'http://project.usagi.pl/comment/byGallery/{galleryId}' – pobiera wszystkie komentarze w galerii.

   - W gallery.component.ts w ngOnInit() pod pobraniem danych o galerii dodaj api do pobierania wszystkich komentarzy z galerii.

   - Wykorzystując ngFor wyświetl komentarze pod szczegółami galerii – możesz zrobić komponent comment-item (tak jak mieliśmy zrobione pojedyncze galerie na liście)

2. **Zadanie samodzielne**

   Usuwanie poszczególnych komentarzy – obok każdego z komentarzy dodaj przycisk/link do usunięcia – działający na onClick – robiliśmy coś takiego przy usuwaniu pojedynczych galerii.

   Wykorzystamy do tego POST 'http://project.usagi.pl/comment/delete/{commentId}' - usuwa komentarz

   Uaktualnij listę po usunięciu komentarza (Output())

3. Zadanie samodzielne

Uaktualnij listę komentarzy po dodaniu komentarza – do comment-form dodaj funkcję emitującą zdarzenie

```
@Output() addComment: EventEmitter<IComment> = new
EventEmitter<IComment>();
```

I wywołaj ją podczas dodawania komentarza do bazy.

```
this.addComment.emit(response);
```

W **gallery.component.ts** dodaj komentarz do tablicy komentarzy.

```
this.comments.push($event);
```