Laboratorium 5: RESTful API

Zagadnienia na wejściówkę na końcu

To, co daje siłę aplikacjom opartym o frameworki JavaScript to niewątpliwie ich zdolność do przetwarzania żądań do/z serwera w sposób asynchroniczny, bez przeładowywania strony.

Angular doskonale nadaje się do pracy w aplikacjach wykorzystujących REST (ang. Representational State Transfer) - wzorzec narzucający dobre praktyki tworzenia architektury aplikacji rozproszonych w oparciu o protokół http.

Wzorzec ten udostępnia nam cztery podstawowe metody:

GET — pobieranie (zarówno kolekcji, jak i pojedynczego elementu)

POST — tworzenie (tylko kolekcji)

PUT — aktualizacja (tylko pojedynczego elementu)

DELETE — usuwanie (tylko pojedynczego elementu)

Korzystaliśmy już z tego wcześniej, na zajęciach z XML, korzystając z obiektu XMLHttpRequest.

Więcej: http://yarpo.pl/2012/07/29/rest-ciekawszy-sposob-na-komunikacje-client-server/

Ściąga z API:

http://project.usagi.pl/user/add/{your_email}

POST 'http://project.usagi.pl/gallery' – tworzy nową galerię

GET 'http://project.usagi.pl/gallery' – pobiera wszystkie galerie

GET 'http://project.usagi.pl/gallery/{galleryId}' – pobiera galerię

POST 'http://project.usagi.pl/gallery/delete/{galleryId}' - usuwa galerię

Celem laboratoriów będzie połączenie galerii z zewnętrznym API w celu lepszego zarządzania jej danymi. Cała galeria wraz z komentarzami zostanie przeniesiona do bazy danych znajdującej się na zewnętrznym serwerze, a następnie pobrana stamtąd do naszej aplikacji.

Dodamy też opcje usuwania galerii.

Zacznijmy od przeniesienia galerii na serwer.

Ćwiczenie 1. Przygotowanie do pracy z HTTP

1. Ponieważ wszystkie galerie będą znajdowały się w jednej bazie danych, na początek musimy wygenerować indywidualne ID, które zapewni nam, że każdy dostanie dostęp tylko do tych galerii, które sam stworzy.

   Wpisz w przeglądarce:

   http://project.usagi.pl/user/add/

   i na końcu dopisz swój email. Strona powinna wyświetlić twój unikalny identyfikator. Zapamiętaj go sobie.

   Jeśli go zapomnisz, wystarczy ponownie wpisać ten sam email.

2. Mamy identyfikator, pora do naszej aplikacji dodać możliwość korzystania z protokołu http. W tym celu otwórz ***app.module.ts*** i dopisz po linią z importem *BrowserModule*:

```
import { HttpClientModule } from '@angular/common/http';
```

   Następnie w sekcji **imports** również pod *BrowserModule* dodaj HttpClientModule.

   Uwaga, jeśli używasz starszej wersji angular-cli, możliwe, że będziesz musiał zmienić HttpClientModule na Http : https://angular.io/api/http/Http

Ćwiczenie 2. Przenoszenie wszystkich albumów na serwer

Ok, protokół http dostępny, skorzystajmy z niego. Na początek coś prostego. Jak do tej pory wszystkie galerie trzymamy w folderze constants i wczytujemy jako Galleries. Szkoda byłoby stracić te dane, dlatego wyeksportujemy je na serwer.

1. Przejdźmy do naszego pliku ze wszystkimi galeriami **galleries.component.ts**. W konstruktorze dodamy http jako serwis, z którego będziemy korzystać.

   ```
   constructor(private http: HttpClient) {
   ```

   Jeśli HttpClient nie został dodany automatycznie jako import na górze pliku, wciśnij ALT, kliknij na niego myszką, a następnie wciśnij ALT+ENTER.

2. Nad konstruktorem zdefiniujemy sobie nagłówek http, który będzie odpowiadał za autoryzację naszej współpracy z serwerem:
3.

```
httpOptions = {
    headers: new HttpHeaders({
        'Content-Type': 'application/json',
        'Authorization': 'twój_identyfikator'
    })
};
```

Jeśli HttpHeaders nie zostały zaimportowane, ponownie wykonaj akcję z ALT+ENTER.

Jako twój_indentyfikator podstaw swój NUMER!

4. Teraz pora na część główną. Nad lub pod galeriami w **galleries.component.html** dodaj przycisk i przypisz mu funkcję eksportu galerii exportGalleries().

```
<button class="button button-small" (click)="exportGalleries()">Export
galleries</button>
```

5. Zdefiniujmy tę funkcję w **galleries.component.ts**:

```
exportGalleries() {
    Galleries.forEach((gallery: IGallery) => {
        delete(gallery.galleryId);

        this.http.post('http://project.usagi.pl/gallery', gallery,
this.httpOptions).toPromise().then((response: IGallery) => {
            console.log('success', response);
            this.galleries.push(response);
        }, (errResponse) => {
            console.log('error', errResponse);
        });
    });
}
```

Co tu się wydarzyło?

- W pierwszym kroku wzięliśmy nasze galerie i zastosowaliśmy na nich pętlę forEach.
- Z każdej galerii usunęliśmy galleryId, ponieważ w bazie danych otrzyma nowe. Stare Id nadane przez nas mogłoby pokryć się z czymś, co już w bazie jest.
- Następnie skorzystaliśmy z **http.post** i wysłaliśmy galerię na serwer pod adres `http://project.usagi.pl/gallery` dodając wcześniej zdefiniowane nagłówki.
- Jak pamiętamy http korzysta w tym wypadku z obiektu XMLHttpRequest – zapytania tego typu są asynchroniczne, a więc potrzebują czasu na odpowiedź serwera. Po to nam `.subscribe` – metoda, która oczekuje na odpowiedź serwera i w zależności od sukcesu lub porażki, zwraca rezultat.
- Jeśli operacja się powiedzie, wstawiamy galerię do naszej tablicy `this.galleries` przez polecenie `this.galleries.push`

6. Otwórz okno konsoli i przejdź do zakładki Network.
   - Skorzystaj z przycisku Export galleries. Zobacz, co się pojawi w zakładce.
   - Kliknij na link – to Twoje zapytanie. Jeśli knikniesz na **Preview** zobaczysz odpowiedź serwera.

   To te same dane, które masz też wypisane przez console.log w Konsoli.

Ćwiczenie 3. Wyświetlanie albumów z serwera

Ok, wyeksportowaliśmy galerię na serwer, ale co nam to dało? U nas nadal działa z pliku. Bez sensu. Pobierzmy, co wyeksportowaliśmy.

Tym razem skorzystamy z metody GET.

1. W konstruktorze dodamy metodę, która do naszych *this.galleries* wczyta nie Galleries, ale dane z serwera. Zastąpmy więc zapis

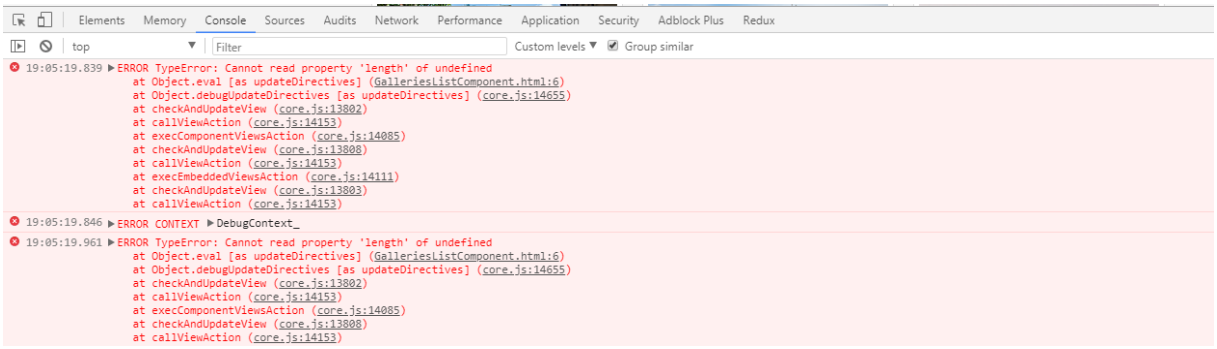
   ```
   this.galleries = Galleries;
   ```

   kodem:

   ```
   this.http.get('http://project.usagi.pl/gallery',this.httpOptions).toPromise().then((response: IGallery[]) => {
      console.log(response);
      this.galleries = response;
   });
   ```

   Sprawdźmy efekt na stronie. Pięknie?

   To otwórzmy konsolę i odświeżmy stronę. Już nie tak pięknie ☺



Czemu? To proste – zapytanie na serwer jest asynchroniczne. Angular próbuje odczytać długość tablicy, której jeszcze nie ma. Na stronie wygląda, jakby wszystko było ok, bo chwilkę później, tablica się doczytała, ale część operacji zdążyła już rzucić błędem.

Co z tym zrobić? Trzeba upewnić się, że wszystko odbędzie się we właściwej kolejności. Musimy najpierw zdefiniować pustą tablicę, a następnie dopisać do niej dane z serwera.

```
this.galleries = [];

this.http.get('http://project.usagi.pl/gallery',
this.httpOptions).toPromise().then((response: IGallery[]) => {
    console.log(response);
    this.galleries = response;
});
```

A w pliku html dodać warunek do elementów opartych o dane galleries:

```
*ngIf="galleries.length"
```

Ćwiczenie 4. Usuwanie albumów z serwera

Umiemy galerie wyeksportować na serwer, umiemy je pobrać, to teraz je usuniemy.

Skorzystamy z adresu do API http://project.usagi.pl/gallery/{gallery_id}

Ponieważ naszym celem na razie jest usunąć WSZYSTKIE galerie od razu, więc powtórzymy kroki z eksportu, tylko w drugą stronę.

1. Dodamy drugi przycisk „Remove galleries" z funkcją removeGalleries().

2. W funkcji removeGalleries() zrobimy pętlę po wszystkich this.galleries i wykorzystamy w niej http.post oraz galleryId:

```
removeGalleries() {
    this.galleries.forEach((gallery: IGallery) => {
        this.http.post('http://project.usagi.pl/gallery/delete/' +
gallery.galleryId, {}, this.httpOptions).toPromise().then((response) =>  {
            this.galleries.splice(0, 1);
            console.log('success', response);
        }, (errResponse) => {
            console.log('error', errResponse);
        });
    });
}
```

Sprawdźmy jak to działa.

Tym razem bierzemy galerie, które wyświetlamy na stronie *this.galleries* a nie *Galleries*, ale reszta jest podobna. Znów mamy pętlę, tyle, że tym razem usuwamy w niej galerię o danym galleryId.

Ponieważ `this.http.post` usuwa galerię z serwera, a nie tablicy `this.galleries` więc dodatkowo, jeśli usuwanie na serwerze się powiedzie, sami usuwamy galerię ze strony przez `this.galleries.splice(0,1)` – odwrotne do `this.galleries.push`.

Uwaga. Normalnie używalibyśmy w tym przypadku http.delete, ale ponieważ serwer Home.pl nie obsługuje DELETE i PUT, musimy to obejść przez POST.

Tak wyglądałoby to z http.delete:

```
this.http.delete('http://project.usagi.pl/gallery/' +
gallery.galleryId, this.httpOptions).toPromise().then((response)
=> {
```

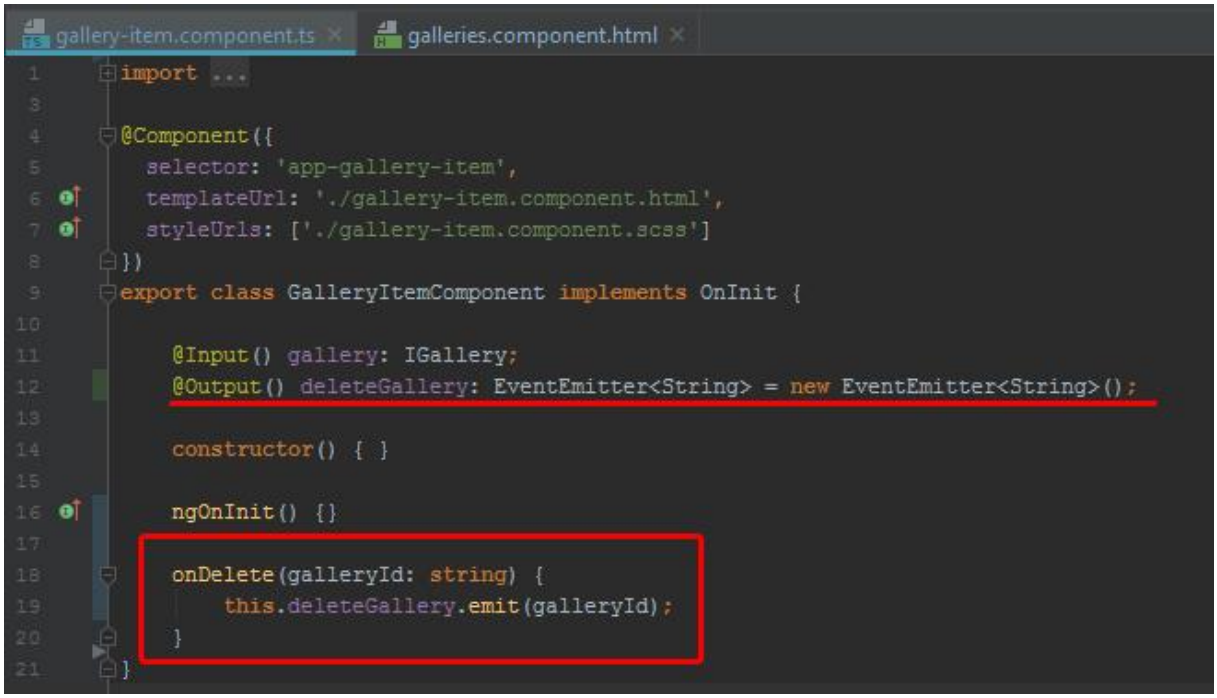
Ćwiczenie 5. Usuwanie pojedynczej galerii

Umiemy już stworzyć listę galerii oraz ją usunąć. Ale co, jeśli chcielibyśmy usunąć tylko jedną galerię? Wykorzystamy już to, co umiemy.

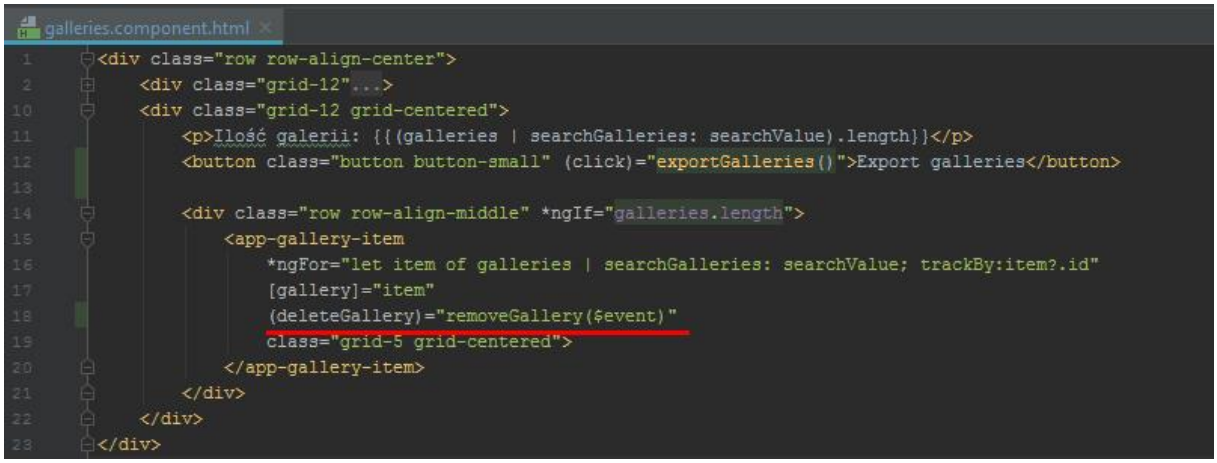
1. W komponencie pojedynczej galerii – **gallery-item.component.html** dodamy przycisk Delete

   ```
   <a (click)=”onDelete(gallery.galleryId)”>Delete</a>
   ```

2. który w **gallery-item.component.ts** wywoła funkcję `onDelete(galleryId)`. Pamiętaj, że emitujemy zdarzenie (kliknięcie) do komponentu wyżej, musimy użyć EventEmittera!



3. Funkcja ta przekaże wyżej, do komponentu **galleries.component.html**, polecenie skasowania galerii o tym konkretnym galleryId.



4. W **galleries.component.ts** ponownie wykorzystamy POST *http://project.usagi.pl/gallery/{gallery_id}* i usuniemy galerię z listy galerii. Tym razem jednak nie będziemy potrzebowali pętli.

   Żeby usunąć galerię również z tablicy `this.galleries` ponownie wykorzystamy **splice()**, jednak tym razem będziemy musieli poszukać indeksu usuwanej galerii. Zrobimy to za pomocą `this.galleries.indexFind()`

```
removeGallery(galleryId) {
    const index = this.galleries.findIndex((gallery: IGallery) =>
gallery.galleryId === galleryId);

    this.http.post('http://project.usagi.pl/gallery/delete/' + galleryId,
{}, this.httpOptions).toPromise().then((response) =>  {
        this.galleries.splice(index, 1);
        console.log('success', response);
    }, (errResponse) => {
        console.log('error', errResponse);
    });
}
```

Zadanie samodzielne

1. Napraw podstrony galerii – pobierz galerię po ID nie z pliku, ale z serwera.

   `GET 'http://project.usagi.pl/gallery/{galleryId}'`

Wejściówka

1. Opisz jak działa .push() i .splice()
2. Co to jest promise w wywołaniach http w Angularze
3. O co chodzi w Observables i czym się różnią od Promises : http://www.angular.love/2018/07/04/rxjs-w-angular-co-wypada-wiedziec/
4. Wymień i opisz metody POST