Laboratorium 6: RESTful API

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/

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:

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

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

3. Przy okazji dopiszmy jeszcze, że nasze `galleries: any = [];`

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

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

```
exportGalleries() {
    GALLERY.forEach(gallery => {
        delete(gallery.galleryId);

        this.http.post('http://project.usagi.pl/gallery', gallery,
this.httpOptions).toPromise().then((response) => {
        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.
- Dla 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 /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.

6. Otwórz okno konsoli i skorzystaj z przycisku. Zobacz, jaką odpowiedź da Ci serwer.
7. Popraw adres na prawidłowy i sprawdź jeszcze raz.

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

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.

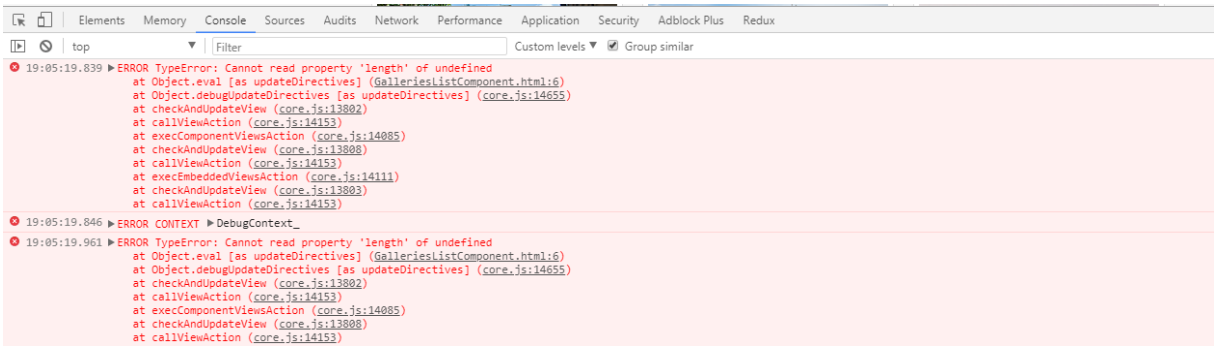
Tym razem skorzystamy z metody GET.

1. W konstruktorze dodamy metodę, która do naszych *this.galleries* wczyta nie GALLERY, ale dane z serwera.

```
this.http.get('http://project.usagi.pl/gallery',this.httpOptions).toPromise
().then((response) => {
   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. Najprościej przenieść wczytanie galerii do ngOnInit().

```
ngOnInit() {

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

A w pliku html dodać warunek do całego komponentu:

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

Ćwiczenie 4. Usuwanie albumów z serwera

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

Skorzystamy z *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 zrobimy pętlę po wszystkich this.galleries i wykorzystamy w niej http.post oraz galleryId:

```
removeGalleries() {
    this.galleries.forEach(gallery => {
        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 działa.

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) => {
```

Zadanie samodzielne

1. Wykorzystując wiedzę z powyższych przykładów dodaj usuwanie POJEDYNCZYCH galerii.
2. Napraw podstrony galerii – pobierz galerię po ID nie z pliku, ale z serwera.

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