Przetwarzanie dokumentów XML i zaawansowane techniki WWW

"Wykorzystanie API do zasilania aplikacji danymi"

(Zajęcia 12 - 06.06.2016 r.)

1. *Wykorzystanie techniki JSON-P*

Wykorzystanie techniki JSON-P, jest bardzo od strony składniowej jest bardzo podobne do skorzystania z serwera proxy CORS. Różnica polega na określeniu innego typu danych odbieranych z serwera w dyrektywie:

```
dataType: 'jsonp'
```

Tym samym informujemy metodę ajax o wykonaniu zapytania JSONP.

Aby przetestować tą metodę skorzystamy z serwisu testowego API dostępnego pod adresem:

http://jsonplaceholder.typicode.com/

Zadanie

Proszę przygotować szablon wraz z metodą testową z w/w strony z przykładem JSONP:

```
var root = 'http://jsonplaceholder.typicode.com';

$.ajax({
  url: root + '/posts/1',
  method: 'GET',
  dataType: 'jsonp'
}).then(function(data) {
  console.log(data);
});
```

Następnie proszę przygotowac miejsce do którego będa trafiać pobrane dane, może to być np. Element blokowy .

2. *Wykorzystanie REST API do zasilania aplikacji danymi*

*Aby zrozumieć ideę architektury REST niezbędne jest poznanie podstaw systemu WWW (ang. World Wide Web). Istnieje kilka fundamentalnych zasad, których ściśle trzyma się powyższy system. W kolejnych akapitach omówiono najistotniejsze z nich. Jednym z elementarnych aspektów funkcjonowania WWW jest jednoznaczne identyfikowanie zasobów współdzielonych w ramach usługi. Do tego celu stosuje się tzw. standard URI (ang. Uniform Resource Identifier).*Jest to adres zasobu reprezentowany w postaci ciągu znaków o ściśle określonej strukturze. Warunkuje on istnienie zasobu, innymi słowy nie można mówić o zasobie w rozumieniu systemu WWW, jeżeli nie jest on precyzyjnie zaadresowany przy użyciu URI. Kolejnym istotnym elementem uważanym za podstawę działania usługi WWW jest ustandaryzowany format zasobów, definiowany za pomocą łańcuchów znaków określających typ główny i podtyp. Oficjalnie istnieje kilkaset różnego typu formatów, najpopularniejsze z nich to np. application/json, image/jpeg , text/html oraz wiele innych, które są stosowane powszechnie. Dzięki powyższej standaryzacji, autorzy aplikacji klienckich służących do

komunikacji z serwerami WWW i interpretujących zawartość zwracanych przez nie zasobów (np. przeglądarek internetowych), mogą poczynić założenia, że zasób konkretnego typu zawsze będzie reprezentowany w ten sam sposób. Do pełnego funkcjonowania systemu usług WWW niezbędny jest jeszcze jeden istotny element. Jest nim protokół, który pełni rolę interfejsu, za pośrednictwem którego aplikacje klienckie odwołują się do zasobów pod danym adresem. Popularnym i istotnym w kontekście omawianego projektu protokołem jest HTTP (ang. Hyper Text Transfer Protocol). Komunikacja przy jego użyciu opiera się o tzw. metody HTTP, które określają jaka operacja powinna zostać wykonana na zlokalizowanym za pomocą URI zasobie. Najczęściej stosowanymi metodami HTTP są:

- GET – służy do pobrania informacji o wskazanym zasobie,
- POST – zlecenie stworzenia nowego zasobu o podanych parametrach,
- PUT – określająca zmiany, jakie należy nanieść na danej encji,
- DELETE – wykorzystywana do usuwania zasobów z serwera.

Powyższa lista nie wyczerpuje wszystkich dostępnych metod, jednakże jedynie wspomniane cztery są istotne z perspektywy prostych aplikacji internetowych. Opisane powyżej elementy systemu WWW stały się bazą dla architektury REST, która przeznaczona jest do projektowania tzw. usług internetowych (ang. web services). Opisane powyżej kluczowe cele stosowania REST mają pokrycie w zestawie formalnych wymagań dotyczących tej architektury. Podstawowym wymogiem jest separacja logiki pomiędzy serwer (usługodawcę), a aplikację kliencką (konsumenta usługi), którzy komunikują się przy użyciu tzw. ujednoliconego kontraktu (ang. Uniform Contract). Kolejnym istotnym warunkiem, które musi zostać spełniony przez architekturę REST,jest bezstanowość systemu. Należy przez to rozumieć, że system nie powinien przechowywać żadnych informacji pomiędzy kolejnymi żądaniami – zatem każde z nich musi dostarczyć wszelkie informacje niezbędne do jego poprawnego przetworzenia. Takie podejście upraszcza budowę usługi, ponieważ odpowiedzialność za utrzymanie stanu interakcji leży po stronie aplikacji klienckiej, dzięki temu usługa nie musi dokonywać analizy wstecznej przychodzących żądań – korzysta jedynie z informacji dostarczonych w bieżącym żądaniu. Z drugiej strony, wadą konieczności przesyłania wszystkich niezbędnych danych jest wzrost obciążenia sieci – a co za tym idzie – zwiększenie ryzyka błędów transmisji oraz wydłużenie czasu potrzebnego na otrzymanie odpowiedzi.

W przykładzie z punktu (1) pobrano przykładowe dane wykonując asynchroniczne zapytanie do pod adres url:

http://jsonplaceholder.typicode.com/posts/1

Końcówka adresu URL która występuje po nazwie domeny może być draktowana jako “ścieżka” która wskazuje na wykonanie odpowiedniej operacji REST. W powyższym przykładzie mamy zdefiniowane następujące zasoby:

```
/posts      - lista postów
/comments   - lista komentarzy
/albums     - lista albumów
/photos     - lista zdjęć
/todos      - lista “ToDo” (do zrobienia).
/users      - lista użytkowników
```

Odpytanie dowolnej z tych sieżek zwróci dane które można odpowiednio opracować a

następnie wyświetlić użytkownikowi na zdefiniowanym szablonie.
Dodatkowo poza pobieraniem zasobów możemy dodawać nowe zasoby również stosując architekturę REST:

GET /posts
GET /posts/1
GET /posts/1/comments
GET /comments?postId=1
GET /posts?userId=1

POST	/posts	-	metoda dodająca nowe zasób na serwer
PUT	/posts/1	-	metoda uaktualniąca zasób
PATCH	/posts/1	-	metoda uaktualniająca danyc zasób
DELETE	/posts/1	-	metoda usuwająca dany zasób

Przykładowo kiedy chcielibyśmy przesłać nowe dane do serwera API poprzez metodę POST postać funkcji jest następująca:

```
$.ajax('http://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  data: {                // przesyłane dane
    title: 'foo',
    body: 'bar',
    userId: 1
  }
}).then(function(data) {
  console.log(data);
});

/* typ zwracanej odpowiedzi że rekord został dodany
{
  id: 101,
  title: 'foo',
  body: 'bar',
  userId: 1
}
*/
```

Aby usunąć zasób z serwera wystarczy użyć metody DELETE:

```
 $.ajax('http://jsonplaceholder.typicode.com/posts/1', {
  method: 'DELETE'
});
```

Zadanie:

Proszę rozbudować poprzedni przykład tak aby były pobierane i wyświetlane wszystkie dane dostępne w serwisie. Proszę również zaimplementować użycie metody POST, PUT i DELETE.