

# Warsztaty Javascript i jQuery: REST

v. 1.0.0

# Plan

- JSON
- AJAX
- REST API
- CEL WARSZTATÓW

# JSON

# JSON

**JavaScript Object Notation** to lekki format wymiany danych.

JSON jest formatem tekstowym przyjaznym użytkownikom oraz całkowicie zrozumiałym dla większości języków programowania.

## Główne cechy JSON-a

- Prosta, naturalna składnia.
- Mały narzut.

JSON jest formatem wymiany danych często wypierającym XML, gdy rozbudowane możliwości XML-a nie są wymagane.

# Typy danych

- Liczba
- Ciąg znaków
- Wartość logiczna
- Obiekt
- Tablica
- Null

## Składnia

### Dane

```
"nazwa": wartość
```

### Obiekty

```
{ "nazwa" : wartość,  
  "nazwa2" : "wartość2" }
```

### Tablica

```
[ { "nazwa": 12 }, { "nazwa": 13 } ]
```

# JSON w JavaScriptcie

Do pracy z formatem JSON w JavaScriptcie używa się obiektu o nazwie `JSON`. Jest to obiekt globalny mający dwie metody:

- **`JSON.parse(text)`** – metoda pobiera tekst (w formacie JSON) i zwraca dane jako wartość JavaScript.
- **`JSON.stringify(value)`** – metoda pobiera dane JavaScript i zwraca tekst, który reprezentuje te dane w formacie JSON.

# Przykłady

```
JSON.parse('{}');           // {}
JSON.parse('true');          // true
JSON.parse(' "foo" ');      // "foo"
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');          // null
JSON.stringify({});          // '{}'
JSON.stringify(true);        // 'true'
JSON.stringify('foo');       // ' "foo" '
JSON.stringify([1, 'false', false]); // '[1,"false",false]'
JSON.stringify({ x: 5 });     // '{"x":5}'
JSON.stringify(new Date(2006, 0, 2, 15, 4, 5)); // ' "2006-01-02T15:04:05.000Z" '
```

# JSON w Javie

Pracując z danymi w formacie JSON wykorzystujemy dodatkowe biblioteki.

Najpopularniejsze z nich to

- **Jackson**
- **Gson**
- **Org.JSON**



# AJAX

# Czym jest Ajax?

## Technika komunikacji klienta i serwera

**Asynchronous JavaScript and XML to metoda pozwalająca na odpytanie serwera i podmianę części treści bez przeładowania całej strony.**

Podstawowo AJAX jest zaimplementowany przez **XMLHttpRequest**. Ale na szczęście teraz jest to łatwiejsze dzięki użyciu funkcji z biblioteki jQuery.

## Asynchroniczność

- Asynchroniczność oznacza, że wywołanie jakiejś funkcji nie powoduje zablokowania działania naszej strony (lub aplikacji).
- Funkcja taka (zazwyczaj) sama poinformuje nas o ukończeniu swojego działania.

# Metody w HTML

W HTML mamy do czynienia z czterema głównymi metodami, jakimi możemy wysyłać żądanie do strony.

Są to:

- **GET** – używany do uzyskiwania danych,
- **PUT** – używany do zmiany informacji (update),
- **POST** – używany do wysyłania informacji (np. z formularzy),
- **DELETE** – używany do usuwania danych.

# AJAX w jQuery

W jQuery mamy kilka funkcji służących do wywoływania zapytań AJAX. Najważniejszą z nich jest **\$.ajax**. Funkcja ta jest bardzo rozbudowana i przyjmuje wiele różnych parametrów oraz konfiguracji.

**\$.ajax( url [, settings] )**

## Funkcja \$.ajax

```
$.ajax({  
    url: "example.html/my/example",  
    data: {},  
    type: "GET",  
    dataType : "json",  
    success: function( json ) {},  
    error: function( xhr, status,  
                    errorThrown ) {},  
    complete: function( xhr, status ){}  
});
```

# AJAX w jQuery

W jQuery mamy kilka funkcji służących do wywoływania zapytań AJAX. Najważniejszą z nich jest **\$.ajax**. Funkcja ta jest bardzo rozbudowana i przyjmuje wiele różnych parametrów oraz konfiguracji.

`$.ajax( url [, settings] )`

## Funkcja \$.ajax

```
$.ajax({  
  url: "example.html/my/example",  
  data: {},  
  type: "GET",  
  dataType : "json",  
  success: function( json ) {},  
  error: function( xhr, status,  
                  errorThrown ) {},  
  complete: function( xhr, status ){}  
});
```

Adres URL, z którego chcemy pobrać dane.

# \$.ajax

- **url:** adres strony, do której wysyłane jest zapytanie,
- **data:** dane wysłane do tej funkcji, (np. dane z formularza – są konwertowane do query string),
- **type:** typ zapytania HTML, który użyjemy,
- **dataType:** typ danych, jaki otrzymamy (text, html, script, json, xml).
- **done:** funkcja wywołana, jeżeli zapytanie się uda,
- **fail:** funkcja, która zostanie wywołana, jeżeli napotkamy błąd,
- **always:** funkcja, która zostanie wywołana po ukończeniu zapytania (nieważne czy się uda czy nie).

<http://learn.jquery.com/ajax/jquery-ajax-methods>

# \$.ajax – done(), fail(), always()

Metody **done()**, **fail()** oraz **always()** są proponowanym rozwiązaniem od wersji jQuery 1.8. Wciąż jednak można się spotkać z powoli wycofywanymi metodami:

- success(),
- error()
- complete().

Chcesz wiedzieć więcej o **done()**, **fail()** i **always()**? Poczytaj o obiekcie deferred:

- <http://api.jquery.com/jquery.deferred>

```
$.ajax({  
  url: 'www.example.com/seans.html'})  
  .done(function(){ //... })  
  .fail(function(){ //... })  
  .always(function(){ //... });
```

```
$.ajax({  
  url: 'www.example.com/seans.html',  
  success: function(){ //... },  
  error: function(){ //... },  
  complete: function(){ //... }  
});
```



# Inne funkcje AJAX

## ➤ \$.get

- zapytanie AJAX, w którym **type** jest ustawiony na GET.

## ➤ \$.post

- zapytanie AJAX, w którym **type** jest ustawiony na POST.

## ➤ \$.getJSON

- zapytanie AJAX, w którym **dataType** jest ustawiony na JSON.



# REST API

# REST API

## API

- API jest skrótem od **Application Programming Interface** (Interfejs Programistyczny Aplikacji).
- Jest to ściśle określony zestaw reguł (i ich opisów), w jaki programy komputerowe komunikują się między sobą.

## REST

- **Representational State Transfer (REST)** jest specjalnym typem API (bardzo popularnym), stosowanym do tworzenia aplikacji z silnie oddzieloną bezstanową relacją klient – serwer.
- Przez bezstanowość rozumiemy fakt, że serwer nie trzyma żadnych informacji o kliencie (nie używam superglobalnej SESSION).

# Zasady REST

## Klient - serwer

Zasada mówiąca, że strona kliencka jest całkowicie odseparowana od strony serwerowej.

## Bezstanowość

Architektura REST nie przetrzymuje żadnych informacji między zapytaniami.

## Warstwowość

Klient nie powinien być w stanie powiedzieć, czy jest podłączony bezpośrednio do serwera, czy do pośrednika.

## Cashable

Strona kliencka powinna być w stanie przetrzymywać zasoby po swojej stronie.

## Jednostajny interfejs

Systemy **REST** powinny utrzymywać jednostajny interfejs dla wszystkich swoich zasobów.

# Metody REST

Rest opiera się na implementacji następujących metod dla każdego zasobu:

- GET – używane do odczytania zasobu z serwera.
- POST – używane do stworzenia nowego zasobu.
- PUT – używane do modyfikacji zasobu do serwera.
- DELETE – używane do usunięcia danego zasobu.

## Typowe adresy REST

Metoda HTTP	Zasób	Opis
GET	/movies	Pobieranie wszystkich filmów
PUT	/movies	Edycja filmu
POST	/movies	Dodaje film do kolekcji
DELETE	/movies	Usuwa całą kolekcję
GET	/movies/1	Pobiera film o id=1
PUT	/movies/1	Edycja filmu o id=1
DELETE	/movies/1	Usuwa film o id=1

# Praca z PUT i DELETE

- Do tej pory odczytywaliśmy dane POST i GET
- Dane przekazane za pomocą metod PUT i DELETE tworzone i odczytywane są inaczej, niż POST i GET.

# Generowanie PUT i DELETE

- Nie jest możliwe wygenerowanie zapytań PUT i DELETE z kodu HTML.
- Najprostszym sposobem jest użycie AJAX.

```
$.ajax({  
  url: 'http://example.com/books/1',  
  type: 'PUT',  
  data: 'Name=Paragraf22&Autor=Heller',  
  success: function() { alert('PUT completed'); }  
});
```

# Generowanie PUT i DELETE

- Nie jest możliwe wygenerowanie zapytań PUT i DELETE z kodu HTML.
- Najprostszym sposobem jest użycie AJAX.

```
$.ajax({  
  url: 'http://example.com/books/1',  
  type: 'PUT',  
  data: 'Name=Paragraf22&Autor=Heller',  
  success: function() { alert('PUT completed'); }  
});
```

W tym miejscu możemy wpisać inne metody HTTP. Np. DELETE, POST, GET.

# REST w Javie

**JAX-RS** to interfejs wprowadzony w Javie EE 6, opisujący sposób budowania usług sieciowych opartych o REST.

Referencyjna biblioteka to **Jersey** ( <https://jersey.java.net/> ), oparty na niej został dostarczony do warsztatu serwer.

Inne implementacje to:

- Apache CXF
- REST Easy



# Cel warsztatów

Celem warsztatów jest napisanie pełnej i funkcjonalnej aplikacji frontendowej do katalogowania książek metodą REST.

Projekt składa się z dwóch części:

- **Serwer** – napisany w Javie z wykorzystaniem Jersey (gotowy, do pobrania z Githuba),
- **Klient** – napisany w HTML-u i JavaScriptcie, komunikujący się z serwerem za pomocą AJAX.

Serwer implementuje klasę Book mającą swój identyfikator, isbn, tytuł, autora, wydawcę i gatunek.

# Cel warsztatów

Metoda HTTP	ADRES	CO ROBI?
GET	/books/	Zwraca listę wszystkich książek.
POST	/books/	Tworzy nową książkę na podstawie danych przekazanych z formularza i zapisuje ją do bazy danych.
GET	/books/{id}	Wyświetla informacje o książce o podanym id.
PUT	/books/{id}	Zmienia informacje o książce o podanym id na nową.
DELETE	/books/{id}	Usuwa książkę o podanym id z bazy danych.

# Cel warsztatów

- Klient ma implementować tylko stronę główną.
  - Strona ta ma pokazać wszystkie książki stworzone w systemie. Dane mają być wczytane AJAX-em z adresu **/books/**.
  - Na górze tej strony ma być też formularz do tworzenia nowych książek wysyłający dane AJAX-em (metoda POST).
- Gdy użytkownik kliknie na nazwę książki, pod nią ma się rozwijać **div** z informacjami na temat tej strony wczytane za pomocą AJAX (GET) z endpointu **/books/{id-książki}**. Div ten ma też zawierać formularz służący do edycji tej książki (AJAX, metoda PUT na endpointzie **/books/{id-książki}**).
  - Obok nazwy ma się znajdować guzik służący do usuwania książki (AJAX, metoda DELETE na endpoint **/books/{id-książki}**).

Pod koniec zajęć wyślij głównemu wykładowcy informację z adresem repozytorium, na którym znajduje się twój kod z listą zaimplementowanych funkcjonalności.

# Zadanie rozgrzewkowe

- Używając AJAX-a pobierz aktualną datę z endpointu <http://date.jsontest.com>
- Używając AJAX-a pobierz opis postaci filmowej z endpointu <http://swapi.co/api/people/4/>

# Zadanie 1

## Przygotowanie

Sklonuj serwer API z udostępnionego repozytorium wraz zadaniami dla modułu.

Po pobraniu serwera, będąc w folderze pobranego projektu wykonaj polecenie:

```
mvn install
```

W celu uruchomienia projektu z wbudowanym serwerem:

```
mvn clean compile exec:java
```

Więcej o wykonywanych komendach dowiemy przy okazji zajęć omawiających narzędzie Maven.

Oprócz źródeł w repozytorium znajduje się również archiwum zip zawierające biblioteki oraz plik z rozszerzeniem **jar**.

Wykładowca w celu uruchomienia serwera może korzystać z polecenia:

```
java -jar nazwa_pliku.jar
```

# Opis api

Opis wszystkich metod znajdziesz pod adresem <http://localhost:8282/application.wadl>

**WADL** - stanowi opis wszystkich metod dla REST, porównać go można do **WSDL** - opis dla SOAP.

[https://en.wikipedia.org/wiki/Web\\_Application\\_Description\\_Language](https://en.wikipedia.org/wiki/Web_Application_Description_Language)



# Zadanie 1

## Testowanie api

Do sprawdzenia poprawności api możemy użyć narzędzia cURL

<https://pl.wikipedia.org/wiki/CURL>

Jeżeli użyłeś naszego skryptu instalacyjnego do przygotowanie swojego komputera to powinieneś mieć już zainstalowany program cURL.

W testowaniu dowolnego api, bardzo przydatne są narzędzia umożliwiające przesłania żądań do serwera z poziomu interfejsu graficznego:

- Postman
- Advanced Rest Client
- soapui

W celu czytelnego wyświetlania danych w formacie JSON za pomocą przeglądarki chrome możesz zainstalować wtyczkę **JSONView**.

# Zadanie 1

## Testowanie api GET

Metodę get możesz wywołać bezpośrednio z poziomu przeglądarki wpisując adres:

<http://localhost:8282/books/>

W ten sposób łatwo sprawdzić czy prawidłowo wykonąły się metody wykonujące modyfikacje na danych.

- POST
- PUT
- DELETE



# Zadanie 1

## Testowanie api cURL

Wykonaj poniższe polecenia przy uruchomionym serwerze:

Metoda **POST** - dodanie danych

```
curl -X POST -i -H "Content-Type: application/json" -d  
  '{"isbn":"34321","title":"Thinking in Java",  
  "publisher":"Helion","type":"programming",  
  "author":"Bruce Eckel"}' http://localhost:8282/books/add
```

Jeżeli kopiujesz polecenie - musi ono być w jednej linii.

# Zadanie 1

## Testowanie api cURL

Metoda **PUT** - aktualizacja danych

```
curl -X PUT -i -H "Content-Type: application/json" -d  
  '{"id":1,"isbn":"32222","title":"Thinking in Java",  
  "publisher":"Helion","type":"programming",  
  "author":"Bruce Eckel"}' http://localhost:8282/books/1/update
```

Metoda **DELETE** - usuwanie danych

```
curl -X DELETE -i http://localhost:8282/books/remove/1
```

Po wykonaniu każdego polecenia sprawdzaj wyniki przy użyciu metody **GET**

Jeżeli kopiujesz polecenie - musi ono być w jednej linii.

# Zadanie 2

## Przygotowanie

- Przygotuj folder pod aplikację (inny niż serwera książek),
- Pamiętaj o tworzeniu commitów (najlepiej co ćwiczenie).
- Stwórz plik .gitignore i dodaj do niego elementy ignorowane np. podstawowe pliki projektu (.project, .settings).
- Możesz skorzystać z serwisu <https://www.gitignore.io/>

# Zadanie 3

## Stworzenie strony głównej

- W katalogu / stwórz plik index.html, a w nim podstawowy layout swojej strony.
- Użyj JQuery do stworzenia zapytania AJAX do endpointu **/books/** metodą GET.
- Po otrzymaniu wszystkich danych w formacie JSON wyświetl wszystkie tytuły na stronie.
- Pod każdym tytułem zostaw pusty **div**, w którym później będziemy wyświetlać informacje na temat książki.

# Zadanie 4

## Strona główna

W katalogu / w pliku index.html:

- Do każdego tytułu książki podepnij funkcję **click**.
- Funkcja ta ma rozwinąć pusty div znajdujący się pod książką, wysłać zapytanie AJAX do serwera na **endpoint /books/{id-książki}** i wyświetlić odebrane informacje.

# Zadanie 5

## Strona główna

W katalogu / w pliku index.html:

- Stwórz na górze strony formularz służący do tworzenia nowej książki.
- Ma on mieć podpiętego AJAX-a, który metodą POST wyśle dane na **endpoint /books/**.

Po otrzymaniu wiadomości o udanym dodaniu ma on przeładować od nowa wszystkie książki (żeby pokazać tę nowo dodaną).

# Zadanie 6

## Strona główna

W katalogu / w plik index.html:

- Obok każdej książki wyświetl link służący do usunięcia książki.
- Podepnij do niego funkcję **click**.
- Funkcja ta ma przesyłać AJAX-em zapytanie DELETE na **endpoint /books/{id-książki}**
- Potem ma załadować od nowa wszystkie książki (żeby nie wyświetlać usuniętej).

# Zadanie 7

## Strona główna

Przerób teraz całego AJAX-a znajdującego się w aplikacji tak, aby:

- W całej aplikacji frontendowej była tylko jedna funkcja AJAX-a.
- Funkcja powinna pobierać z datasetów (utwórz je) co ma zrobić (GET, POST, itp.) oraz ewentualne ID książki i wszystkie potrzebne dane z odpowiednich elementów strony,
- Na podstawie pobranych danych, komunikuje się z REST API i wysyła / odbiera potrzebne dane.