

REST API

v 1.4

Plan

- JSON
- SERIALIZACJA OBIEKTÓW
- AJAX
- REST API



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.
- Prosta składnia = ograniczone zastosowania.

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

Obiekt

{ "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');  
JSON.parse(' "foo" ');  
JSON.parse('[1, 5, "false"]');  
JSON.parse('null');
```

```
// {}  
// true  
// "foo"  
// [1, 5, "false"]  
// null
```

```
JSON.stringify({});  
JSON.stringify(true);  
JSON.stringify('foo');  
JSON.stringify([1, 'false', false]);  
JSON.stringify({ x: 5 });  
JSON.stringify(new Date(2006, 0, 2, 15, 4, 5));
```

```
// '{}'  
// 'true'  
// ' "foo" '  
// '[1,"false",false]'  
// '{"x":5}'  
// ' "2006-01-02T15:04:05.000Z" '
```

JSON w PHP

Dekodowanie danych w formacie JSON

```
mixed json_decode ( string $json [, bool $assoc = false [, int $depth = 512 [, int $options = 0 ]]] )
```

Kodowanie danych do formatu JSON

```
string json_encode ( mixed $value [, int $options = 0 [, int $depth = 512 ] ] )
```

Błędy w czasie dekodowania/kodowania

```
int json_last_error ( void )
```


Przykłady

```
$json = '{"ksiazki": [  
{"tytul": "T1","liczba_stron": 10,"dostepna": false},  
{"tytul": "T2","liczba_stron": 50,"dostepna": true}  
]}';
```

```
json_decode($json);  
  
object(stdClass)#1 (1) {  
    ["ksiazki"]=> array(2) {  
        [0]=> object(stdClass)#2 (3) {  
            ["tytul"]=> string(6) "Title1"  
            ["liczba_stron"]=> int(10)  
            ["dostepna"]=> bool(false)  
        }  
        [1]=> object(stdClass)#3 (3) {  
            ["tytul"]=> string(6) "Title2"  
            ["liczba_stron"]=> int(50)  
            ["dostepna"]=> bool(true)  
        }  
    }  
}
```

Przykłady

```
$json = '{"książki": [  
{"tytuł": "T1","liczba_stron": 10,"dostępna": false},  
{"tytuł": "T2","liczba_stron": 50,"dostępna": true}  
]}';
```

```
json_decode($json, true);  
array(1) {  
    ["książki"]=> array(2) {  
        [0]=> array(3) {  
            ["tytuł"]=> string(6)  
            "Title1"  
            ["liczba_stron"]=>  
            int(10)  
            ["dostępna"]=>  
            bool(false)  
        }  
        [1]=> array(3) {  
            ["tytuł"]=> string(6)  
            "Title2"  
            ["liczba_stron"]=>  
            int(50)  
            ["dostępna"]=>  
            bool(true)  
        }  
    }  
}
```

Przykłady

```
$dane = array('książki' => array(  
    array('title' => 'T1', 'liczba_stron' => 12),  
    array('title' => 'T2', 'liczba_stron' => 12),  
));
```

```
json_encode($dane);  
  
{  
    "książki": [  
        {"title": "T1", "liczba_stron": 12},  
        {"title": "T2", "liczba_stron": 12}  
    ]  
}
```

SERIALIZACJA OBIEKTÓW

Serializacja Obiektów w PHP

W PHP możemy łatwo zaimplementować za pomocą interfejsu obsługę serializacji naszego obiektu. Standardowe przekazanie obiektu do funkcji `json_encode()` nie zadziała.

Pozwoli nam to bezpośrednio w obiekcie za pomocą metody interfejsu obsłużyć dane jakie mają być zwrócone do funkcji `json_encode()` a PHP automatycznie wywoła tą metodę.

Implementacja interfejsu oznacza, że w naszej klasie **MUSIMY** użyć metod jakie ten interfejs definiuje.

Potraktujcie interfejs jako swego rodzaju szablon, który dodajemy do klasy.

Interfejsy dzielą się na wbudowane w PHP (które pewne akcje wykonują automatycznie) oraz takie, które piszemy samodzielnie, wówczas sami wywołujemy poszczególne metody.

Czym są interfejsy dowiecie się dokładnie na zajęciach z Zaawansowanego PHP.

Interfejs JsonSerializable

Interfejs JsonSerializable

Pozwala na użycie obiektu w funkcji `json_encode()`.

W metodzie `jsonSerialize()` implementujemy co ma być zwrócone i przekazane do funkcji `json_encode()` np. String lub Array

```
$obj = new ExampleObject();  
json_encode($obj); //wywoła metode  
jsonSerialize() i przekaże jej rezultat do funkcji  
json_encode()
```

Metoda `JsonSerializable` jest wywoływana automatycznie przy przekazaniu obiektu do funkcji `json_encode()`.

JsonSerializable


```
JsonSerializable{  
    /* Metody */  
    abstract public mixed jsonSerialize(void)  
}
```

Przykład JsonSerializable

class Person implements JsonSerializable


```
{  
    public function __construct($name, $age)  
    {  
        $this->name = $name;  
        $this->age = $age;  
    }  
  
    public function jsonSerialize()  
    {  
        return [  
            'name' => $this->name,  
            'age' => $this->age  
        ];  
    }  
}
```

Implementujemy metodę definiowaną przez interfejs



```
$person1 = new Person('Marek', 27);
```

```
$serializedData = json_encode($person1);
```



W tym momencie PHP automatycznie wywoła metodę jsonSerialize() w naszym obiekcie ponieważ zaimplementowaliśmy interfejs. Takie wywołanie spowoduje zwrócenie do funkcji serialize() tablicy i jej zserializowanie.

```
echo $serializedData;  
//zwróci  
{"name":"Marek","age":27}
```



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, w jaki 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 ) {}  
});
```

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 przechowuje żadnych informacji między zapytaniami.

Warstwowość

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

Cachable

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

Jednostajny interfejs

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

Metody REST

Typowe adresy 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.

Metoda HTTP	Zasób	Opis
GET	/movies	Pobranie 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 (dostępne w superglobalnej `$_POST`) i GET (dostępne w `$_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.

albo DELETE, POST, GET



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

Odczytywanie PUT i DELETE

PHP nie ma wbudowanego sposobu odczytywania danych z metod PUT i DELETE.

Można te dane przeczytać z inputu otrzymywanego przez PHP z serwera. Input ten jest zapisany w ścieżce:

➤ **php://input**

Input od serwera odczytujemy w następujący sposób:

```
parse_str(file_get_contents("php://input"), $put_vars);
```

Dzięki temu w zmiennej **\$put_vars** będziemy mieli wartości przechowane w tablicy asocjacyjnej. Czyli tak jakbyśmy sami używali **\$_GET** lub **\$_POST**.

Odczytywanie PUT i DELETE

```
if($_SERVER['REQUEST_METHOD'] == 'GET') {  
    echo("GET<br>");  
    var_dump($_GET);  
} elseif($_SERVER['REQUEST_METHOD'] == 'PUT')  
    echo("PUT<br>");  
    parse_str(file_get_contents("php://input"), $put_vars);  
    var_dump($put_vars);  
} elseif($_SERVER['REQUEST_METHOD'] == 'DELETE')  
    echo("DELETE<br>");  
    parse_str(file_get_contents("php://input"), $del_vars);  
    var_dump($del_vars);  
}
```