REST i JSON

Michał Gałka

JSON

JSON

- JSON - JavaScript Object Notation
- Lekki, tekstowy format wymiany danych.
- Nie jest związany z konkretnym językiem programowania.

Format JSON

- Wszystkie dane są zmiennymi.
- Nazwy składników obiektu otoczone są cudzysłowem.
- Wartości mogą być następujących typów:
  - `string` - napisy
  - `number` - liczby
  - jedna ze stałych: `null`, `true`, `false`
  - lista złożona z powyższych typów.
  - obiekt

Format JSON


```
{
    "table": "A",
    "currency": "funt szterling",
    "code": "GBP",
    "rates": [{
        "no": "243/A/NBP/2018",
        "effectiveDate": "2018-12-14",
        "mid": 4.7940
    }]
}
```

REST

REST

- REST - REpresentational State Transfer.
- Sposób komunikacji pomiędzy systemami komputerowymi w Internecie.
- Usługi sieciowe (ang. web service) oparte o REST pozwalają na dostęp i manipulację tekstowymi danymi reprezentującymi zasoby sieciowe.
- Dostęp do danych odbywa się w oparciu o **jednolity**, **predefiniowany** zestaw **bezstanowych** operacji.
- Usługi sieciowe oparte o REST bywają nazywane także RESTful Web Service.

REST

- REST stał się wzorcem architektonicznym dla projektowania API (ang. Application Programming Interface) usług sieciowych wykorzystujących protokół HTTP.
- Zdefiniowano 6 reguł projektowania RESTful API.

Reguły REST

- Model Klient-Serwer (ang. Client-Server)
- Bezstanowość (ang. Stateless)
- Możliwość cache'owania danych (ang. Cacheable)
- Warstwowość (ang. Layered System)
- Jednolity interfejs (ang. Uniform Interface)
- Kod na żądanie (ang. Code on demand)

Reguły REST - Client-Server

- W usłudze sieciowej powinny istnieć dobrze wyodrębnione role serwera, który oferuje usługę i klienta który pobiera dane.
- Klient i serwer stanowią dwa niezależne byty.
- Nie muszą znać wzajemnie swoich szczegółów implementacyjnych.
- Mogą być wykonane w zupełnie różnych technologiach.

Reguły REST - Stateless

- Każde zapytanie do serwera musi zawierać wszystkie dane pozwalające na jego realizację.
- Serwer nie może przechowywać danych na temat jednego zapytania i wykorzystywać ich do obsługi innego.

Reguły REST - Cacheable

- Serwer musi powiadomić klienta, czy odpowiedź może zostać przechowana w pamięci podręcznej.
- Przechowowywanie danych z reguły ma większy sens dla operacji odczytu niż zapisu i modyfikacji danych.

Reguły REST - Layered System

- System powinien być projektowany tak, aby odpowiedź do klienta mogła być dostarczona przez serwer pośredniczący.
- Dla klienta takie przekazanie powinno być przezroczyste - odpowiedź powinna wyglądać tak jakby była wysłana przez właściwy serwer.

Reguły REST - Uniform Interface

- Iterfejs komunikacji pomiędzy serwerem a klientem powinien być jednolity i dobrze określony.
  - Interfejs oparty jest na zasobach - każdy zasób jest jednoznacznie identyfikowany przez URI (ang. Unified Resource Identifier).
  - Sposób reprezentacji zasobu powinien być niezależny od danych zasobu, np. dane mogą być dostarczane w postaci XML lub JSON.

- Manipulacja danymi odbywa się także poprzez ich reprezentację.
- Odpowiedzi serwera powinny powinny być samoopisujące np. mieć dobrze zdefiniowany MIME type, aby można było jednoznacznie zinterpretować jak je obsłużyć.
- HATEOAS (ang. Hypermedia as the engine of application state)
  - Klient wraz z zapytaniem wysyła stan, parametry żądania, nagłówki oraz URI zasobu.
  - Serwer dostarcza treść, kod i nagłówki.
  - Wszystkie akcje jakie klient może wykonać są dostarczane w formie zasobów zwracanych z serwera.

Reguły REST - Code on demand

- Jako jedynej, implementacja tej reguły jest opcjonalna.
- Serwer może dostarczyć kod do wykonania w kontekście klienta.

Protokół HTTP

- Usługi sieciowe REST od zostały zaprojektowane tak, aby wpisywały się w struktrę protokołu HTTP.
- Centralnym pojęciem architektury REST zasób (ang. resoruce).
- Każdy zasób jest jednoznacznie identyfikowany przez URI.
- Aplikacje klienckie wykorzystują URI do zaadresowania zapytań do serwera.
- Klient do komunikacji używa metod HTTP.

Metody HTTP

- GET
  - Pobiera dane n.t. zasobu
    - `http://example.com/api/cameras`
    - `http://example.com/api/cameras/10`
- POST
  - Tworzy nowy zasób na podstawie danych przekazanych w zapytaniu.
    - `http://example.com/api/cameras`
- PUT
  - Modyfikuje dane n.t. zasobu na podstawie danych przekazanych w zapytaniu
    - `http://example.com/api/cameras/10`
- DELETE
  - Usuwa zasób o podanym URI
    - `http://example.com/api/cameras/10`

Dane w metodach HTTP

- REST nie wymaga danych przekazywanych w określonym formacie.
- Najczęściej dane przekazywane są w ciele zapytania w formacie `JSON`.
- Czasami wykorzystuje się także argumenty przekazywne jako ciąg argumentów (ang. query string)zapytania.

URI

```
http://[hostname]/cameras/api/v1/
```

- Dobrą praktyką jest zawarcie w URI następujących elementów:
  - nazwa usługi (cameras)
    - Tworzy przestrzeń nazw pozwalającą utrzymywać więcej niż jedną usługę na danym serwerze.
  - wersja API
    - Pozwala utrzymywać kilka wersji API.
    - Przydatne w przypadku kiedy wprowadzamy zmiany w API nie zachowujące kompatybilności wstecz.

Przykład API

- Załóżmy, że projektujemy REST API do zarządzania danymi n.t. aparatów fotograficznych.
- Przyjmijmy, że dla każdego aparatu będziemy przechowywać następujące dane:
  - **name** - nazwa i model aparatu
  - **release_year** - rok premiery
  - **price** - cena aparatu

Przykład API

- Do manipulacji zasobami użyjemy następujących metod:
  - `http://[hostname]/myservice/api/v1/cameras`
    - Metoda HTTP: `GET`
    - Zwraca listę wszystkich aparatów
  - `http://[hostname]/myservice/api/v1/cameras/[id]`
    - Metoda HTTP: `GET`
    - Zwraca dane n.t. aparatu o podanym id.

- `http://[hostname]/myservice/api/v1/cameras`
  - Metoda HTTP: `POST`
  - Dodaje informacje n.t. nowego aparatu do serwisu.
- `http://[hostname]/myservice/api/v1/cameras/[id]`
  - Metoda HTTP: `PUT`
  - Modyfikuje dane n.t. aparatu o podanym id.
- `http://[hostname]/myservice/api/v1/cameras/[id]`
  - Metoda HTTP: `DELETE`
  - Usuwa informacje o danym aparacie z serwisu.

Moduł `requests`

- Wygodnym narzędziem do obsługi zapytań do REST API jest moduł `requests`
- Jest to zewnętrzny pakiet, wymaga więc osobnej instalacji

```
pip install requests
```

Moduł requests - przykład

```
post_id = 1
url =
    'http://jsonplaceholder.typicode.com/posts/{0}'.format(
r = requests.get(url)
if r.status_code == 200:
    if 'content-type' in r.headers and
        'application/json' in
        r.headers['content-type']:
        data = r.json()
print(data)
```

Moduł requests - przykład

```
post_id = 10
url =
    'http://jsonplaceholder.typicode.com/posts/{0}/comments
r = requests.get(url)
if r.status_code == 200:
    if 'content-type' in r.headers and
        'application/json' in
        r.headers['content-type']:
        comments = r.json()
 print(comments)
```

Moduł requests - przykład

```
query = {'user_id': 1}
post_titles = []
url = 'http://jsonplaceholder.typicode.com/posts'
r = requests.get(url, params=query)
if r.status_code == 200:
    if 'content-type' in r.headers and
        'application/json' in
        r.headers['content-type']:
        posts = r.json()
print(posts)
```