JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 6 (2018) 92-96
Wysłane: 2017-11-15
Przyjęte: 2017-11-21

# Porównanie możliwości implementacji usług REST w języku Java z wykorzystaniem popularnych frameworków aplikacji internetowych

Rafał Kwiatkowski*, Piotr Kopniak
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

**Streszczenie**. W artykule zostały zaprezentowane wyniki porównania efektywności i możliwości implementacji usług REST w języku Java przy użyciu frameworków takich jak Jersey, Apache CXF i Spring MVC. Analiza porównawcza została przeprowadzona na podstawie aplikacji zaimplementowanej za pomocą każdego z wymienionych frameworków. Aplikacja została zaimplementowana z wykorzystaniem takich narzędzi jak Spring Boot, Hibernate, Maven i MySQL.

**Słowa kluczowe:** REST; Mikrousługi; RESTful; java

* Autor do korespondencji.

Adres e-mail: raf.kwiatkowski93@gmail.com

# Comparison of capabilities to implement REST services in Java language using the popular web application frameworks.

Rafał Kwiatkowski*, Piotr Kopniak
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

**Abstract.** This article presents the results of a comparison of capabilities and efficiency of REST in Java using frameworks such as Jersey, Apache CXF and Spring MVC. Comparative analysis was conducted on the basis of application implemented by each of the above frameworks. Application have been implemented using technologies such as Spring Boot, Hibernate, Maven, and MySQL.

**Keywords:** REST; Microservices; RESTful; java

*Corresponding author.

E-mail address: raf.kwiatkowski93@gmail.com

## 1. Wstęp

Dzięki rozwojowi technologii coraz rzadziej korzysta się ze szkieletów sieciowych wywołujących usługi i tworzących strony internetowe takie jak RPC CORBA i usługi oparte na protokole SOAP wymagające wielu zależności i będące bardzo skomplikowane [1]. Obecnie częściej aplikacje są tworzone przy pomocy paradygmatu dostarczania i używania interfejsów API. Taka architektura daje większe możliwości jak m.in. stosowanie zwinnych technik programowania, łatwiejsze skalowanie, rozpowszechnianie czy integracja w każdej płaszczyźnie.

REST jest wzorcem narzucającym dobre praktyki tworzenia architektury aplikacji rozproszonych. RESTful Webservices (inaczej RESTful web API) jest usługą sieciową zaimplementowaną na bazie protokołu HTTP i warunków stylu REST. Moda na architekturę mikro serwisów czy konteneryzacja za pomocą takich narzędzi jak Docker, w które REST idealnie się wkomponowuje daje mu dodatkową popularność. Jest prosty, wydajny i daje duże możliwości.

Celem artykułu jest ocena przydatności frameworków Jersey, Apache CXF oraz Spring MVC dla usług RESTowych języka Java. Frameworki zostały porównane na podstawie analizy utworzonego kodu aplikacji oraz metryk statycznych.

Teza jaka została postawiona w artykule to „Spring MVC daje większe możliwości, niż pozostałe rozwiązania”.

W celu utworzenia aplikacji potrzebnej do przeprowadzenia badań wykorzystano następujące narzędzia:

- środowisko programistyczne InteliJ IDEA;
- język programowania Java w wersji 8;
- frameworki do przebadania tj. Jersey, Apache CXF i Spring MVC;
- frameworki do implementacji aplikacji tj. Hibernate, Spring, Spring Boot;
- do budowania projektu użyto narzędzia Maven;
- silnikiem bazy danych był MySQL.

## 2. Styl REST

Technologia Representational State Transfer (REST) wprost z rozwinięcia skrótu oznacza zmianę stanu poprzez reprezentację[13]. Termin Rest został zdefiniowany po raz pierwszy w roku 2000 przez Roya Fieldinga w jego pracy

doktoranckiej na temat „Style architektoniczne i projektowanie architektury oprogramowania opartych na sieci” [4]. Najczęściej można spotkać się z definicją, że jest on stylem architektonicznym reprezentowania i przesyłania danych. Z praktycznego punktu widzenia jest on odpowiedzialny za styl formatowania identyfikatorów URI reprezentujących zasoby, które są dostarczane i przechowywane w aplikacji.

Wyróżnia się 6 założeń REST [2]:

1) Klient – Serwer
2) Jednolity interfejs
3) Bezstanowość
4) Możliwość zapisywania danych w buforze
5) System warstwowy
6) Kod na żądanie.

Aplikacje określa się jako REST gdy spełnia pięć warunków z sześciu. Kod na żądanie jest warunkiem opcjonalnym [3].

## 3. Opis prezentowanych frameworków

Framework jest szkieletem do budowy aplikacji, usprawniającym powtarzające się mechanizmy dla określonej architektury. W przypadku frameworków dla usług RESTowych oczekuję się od niego, że będzie miał metody usprawniające tworzenie takich usług. Będą to np. przygotowane mechanizmy do odbierania żądań za pomocą odpowiednich metod HTTP, czy produkcja i konsumpcja typów MIME (min. XML, JSON, ATOM).

### 3.1. Jersey

Jersey RESTful framework jest otwartym (ang. open source) frameworkiem zbudowanym na podstawie specyfikacji JAX-RS (tj. JSR 311 oraz JSR 339) [5]. Jest to natywna implementacja tego protokołu rekomendowana przez firmę Oracle. Rozszerza implementacje JAX-RS o dodatkowe funkcjonalności (min. takie jak WSDL czy wsparcie dla formatów XML, JSON czy ATOM), w celu uproszczenia tworzenia usługi RESTful i łatwiejszej implementacji klienta. Wersja 2.X została zaprezentowana w 2012 roku [7].

### 3.2. Apache CXF

Apache CXF jest to framework o otwartym kodzie źródłowym przeznaczonym do tworzenia usług sieciowych. Powstał z połączenia kilku projektów open source: CELTIX, rozwijanego przez IONA technologies, Xfire, tworzony przez zespół działający w ramach Codehaus. Oraz biblioteki YOKO. Zostały połączone przez Apache Software Foundation i w 2008 roku wyszedł z fazy inkubacji [12]. Jest to dojrzały i elastyczny framework umożliwiający i dający narzędzia do kompletnej implementacji aplikacji sieciowych, zarówno za pomocą CORBA jak i SOAP czy REST [8].

### 3.3. Spring MVC

Spring REST jest częścią Spring MVC, który jest jednym z modułów najpopularniejszego frameworka w świecie Javy, jakim jest Spring firmy Pivotal. Cały framework posiada otwarty kod źródłowy. Część odpowiedzialna za REST została po raz pierwszy wydana wraz z wersją 3.0 pod koniec 2009 roku, a udoskonalony w wersji 4.0, Spring MVC stal się w pełni dojrzałą i stabilną technologią dla tworzenia serwisów RESTowych. Spring ma wiele modułów, które można dodawać do projektu w zależności od potrzeb, dlatego też Spring REST służy tylko do łatwego i efektywnego używania tego stylu architektonicznego. Spring jako jedyny z omawianych frameworków posiada pełne wsparcie dla najwyższego poziomu w modelu dojrzałości Richardsona jakim jest HATEOAS [6, 9].

### 3.4. Porównanie frameworków

Do zaprezentowania możliwości implementacji usług REST w języku Java zostały wybrane Jersey, Apache CXF oraz Spring MVC. Pierwszym argumentem uzasadniającym ten wybór jest ich dokumentacja, rekomendacje oraz popularność w renomowanym serwisie StackOverFlow. Każde z rozwiązań posiada obszerną dokumentacje, na którą składa się konfiguracja wstępna, opis funkcjonalności, przykłady implementacji oraz JavaDoc z opisem wszystkich metod. Poza tym Spring MVC oraz Jersey zawierają także blogi firmowe. Frameworki te posiadają też rekomendacje popularnych firm branżowych, co buduje większe zaufanie do rozwiązań, jakie prezentują Jersey, Apache CXF oraz Spring MVC. Popularność frameworków przekłada się na liczbę rozwiązanych problemów dostępnych w internecie, które występują w trakcie implementacji. Poniżej zostało przedstawione zestawienie liczby zapytań dla prezentowanych frameworków w serwisie StackOverFlow z dnia 01.10.2017 roku.

Tabela 1. Liczba zapytań na portalu Stackoverflow

| **Framework** | **Liczba zapytań** |
|---|---|
| Spring MVC | 42549 |
| Jersey | 31057 |
| Apache CXF | 4209 |

Jak wynika z powyższej tabeli w serwisie Stackoverflow najwięcej razy pytano o Spring MVC co ukazuje go jako najpopularniejsze rozwiązanie.

Drugim argumentem stojącym za tymi rozwiązaniami są ich funkcjonalności. Wszystkie 3 umożliwiają programowanie za pomocą adnotacji, zaawansowane wsparcie procesu logowania i testowania, wsparcie dla HATEOAS (w tym Spring posiada oddzielny moduł w pełni wpierający ten poziom w modelu dojrzałości Richardsona), oraz jako jedyne mają wsparcie dla Spring Boota co pozwala na łatwiejszą i optymalniejszą konfiguracje.

Ostatnim argumentem jaki stoi za wybraniem tych frameworków są ich różnice. Są to 3 świetne rozwiązania, jednocześnie całkowicie inne, implementujące inne biblioteki i standardy. Co więcej, Jersey jest małym frameworkiem służącym tylko wspomaganiu tworzenia usług REST, gdzie Apache CXF jest frameworkiem pozwalającym tworzyć kompletne aplikacje sieciowe, w którym obsługa RESTa

to mały fragment jego możliwości i Spring MVC, w którym wszystko jest w osobnych modułach – oddzielnych podprojektach, co stanowi ciekawe zestawienie do porównania.

## 4. Opis obiektu badań

W celu udowodnienia postawionej tezy zaprojektowano i zaimplementowano aplikacje, wykonane za pomocą 3 różnych RESTowych frameworków [5].

Projektem aplikacji do zaprezentowania możliwości frameworków jest aplikacja REST typu CRUD (*ang. Create, Read, Update, Delete*) opierającą się na operacjach tworzenia nowych pozycji, wczytywania, odświeżania i usuwania danych z bazą danych MySQL za pomocą metod HTTP takich jak GET, POST, PUT, DELETE. W aplikacji zostało zaimplementowane zamienianie formatu prezentowanych danych użytkownikowi w locie z formatu XML na JSON i odwrotnie (tzw. *marshalling*). W aplikacji dążono do spełnienia poziomu drugiego w modelu dojrzałości Richardsona.

## 5. Opis metody badawczej.

Badanie zostało przeprowadzone na podstawie porównania zaprojektowanych i zaimplementowanych aplikacji za pomocą dwóch metod:

- Analizy kodu – analiza ta polegała na zaprezentowaniu rozwiązań jakie oferują frameworki. W metodzie tej główny nacisk położono na różnice implementacyjne tych samych funkcjonalności tj. konfiguracji frameworków, przetwarzania danych, oraz przesyłania danych do klienta HTTP, oraz zmianie formatów wysyłanych danych.

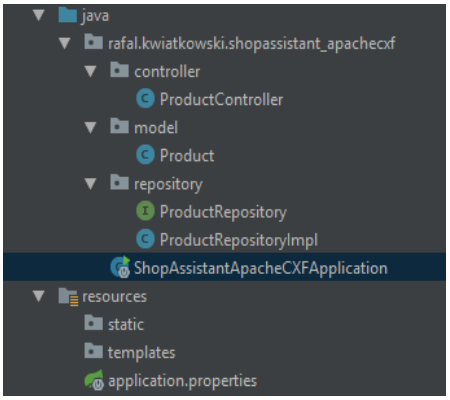
- Metryk statycznych – metryki te polegały na zliczeniu linii kodu z pominięciem pustych linii i komentarzy z podziałem na warstwy w każdej aplikacji i porównaniu ich. Zliczenie linii kodu ma na celu pokazanie efektywności frameworków. Im mniejszej liczby linii kodu potrzebowano do napisania tej samej aplikacji w danym rozwiązaniu tym wyższą ocenę framework uzyskał.

## 6. Badania

Aplikacje zbudowano zgodnie z założeniami REST oraz dobrymi praktykami programistycznymi. Aplikacja została także zaprojektowana z podziałem na warstwy:

- Model – zawiera modele danych mapowane za pomocą frameworka Hibernate,
- Repozytorium – zawiera interfejsy i klasy dające dostęp do danych,
- Kontroler – zawiera klasy odpowiedzialne za przechwytywanie żądań http.

Rysunek poniżej przedstawia strukturę aplikacji dla frameworka Apache CXF. Poszczególne implementacje mają różnice wewnątrz warstw, które są omówione w dalszej części artykułu lecz struktura katalogów jest taka sama.




Rys 1. Struktura projektu.

W celu ukazania funkcjonalności frameworków nie jest potrzebna skomplikowana logika biznesowa, dlatego została utworzona encja produktów w celu przeprowadzenia na niej operacji CRUD. Wykorzystano adnotacje JAXB w celu umożliwienia konwersji encji na typ XML [11].

```
@XmlRootElement
@Entity
public class Product {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private BigDecimal unitPrice;
    private BigDecimal quantity;
```

Rys 2. Encja produkt.

Aby zapewnić identyczne warunki do porównywania frameworków użyto tych samych implementacji do tworzenia ziaren, wstrzykiwania zależności oraz stereotypów z implementacji Springa.

### 6.1. Konfiguracja

Aby usprawnić proces implementacji konfiguracji wstępnej użyto nowoczesnego narzędzia jakim jest Spring Boot. Eliminuje on konieczność tworzenia rozbudowanych konfiguracji xml-owych czy rozbudowanych klas implementacyjnych, a do uruchomienia aplikacji wystarcza klasa uruchomieniowa opatrzona adnotacją @SpringBootAplication [10].

Do pełnego skonfigurowania frameworku Jersey wystarczyło utworzenie klasy konfiguracyjnej zawierającej lokalizacje klasy przechwytującej żądania HTTP i nadającą główną ścieżkę URL aplikacji co prezentuje rysunek 3.

```
@Component
@ApplicationPath("/")
 class JerseyConfig extends ResourceConfig {
    public JerseyConfig() { register(ProductController.class); }
}
```

Rys 3. Konfiguracja frameworku Jersey.

Framework Apache CXF ma bardzo szerokie zastosowanie i oprócz stylu REST, głównym jego zastosowaniem jest

tworzenie usług za pomocą protokołu SOAP. Pomimo informacji w dokumentacji tego frameworka o wsparciu dla JSON, format ten nie jest w pełni wspierany dla stylu REST. Aby zapewnić tą funkcjonalność została dodana biblioteka Jackson, co wymagało dodatkowej konfiguracji zaprezentowanej na rysunku 4.

```
@Bean
@ConditionalOnMissingBean
public JacksonJsonProvider jsonProvider(ObjectMapper objectMapper) {
    JacksonJaxbJsonProvider provider = new JacksonJaxbJsonProvider();
    provider.setMapper(objectMapper);
    return provider;
}
```

Rys 4. Konfiguracja biblioteki Jackson dla Apache CXF.

Oprócz tego Apache CXF wymaga ustawienia ścieżki URL aplikacji oraz przeszukiwania komponentów frameworka Spring w celu znalezienia implementacji protokołu JAX-RS przechwytujących żądania HTTP w pliku application.properties co prezentuje rysunek poniżej

```
cxf.path = /
cxf.jaxrs.component-scan = true
```

Rys 5. Konfiguracja frameworku Apache CXF.

Spring MVC nie wymaga żadnej dodatkowej konfiguracji. Po utworzeniu projektu w Spring bootcie jest gotowy do użycia. W przypadku niestandardowych ustawień można dowolnie modulować je w pliku application.properties.

Tabela poniżej prezentuje liczby linii kodu potrzebnych do kompletnego skonfigurowania frameworków.

Tabela 2. Liczba linii kodu potrzebnych do konfiguracji

| Framework | Liczba linii kodu |
|---|---|
| Jersey | 7 |
| Apache CXF | 11 |
| Spring MVC | 0 |

### 6.2. Dostęp do danych

Do implementacji aplikacji typu CRUD potrzebna jest warstwa dostępu do danych. W Jersey oraz Apache CXF stworzono interfejs ProductRepository, oraz utworzono jego implementacje. W implementacji zastosowano adnotacje @Transactional implementującą korzystanie z transakcji. Oprócz tego wstrzykiwany jest kontekst EntityManagera z frameworku Hibernate. Prezentuje to poniższy rysunek.

```
@Transactional
@Repository
public class ProductRepositoryImpl implements ProductRepository {

    @PersistenceContext
    private EntityManager entityManager;
```

Rys 6. Implementacja repozytorium dla Jersey oraz Apache CXF.

Aby uzyskać wszystkie dane należy wykonać zapytanie HQL oraz utworzyć zapytanie, którego wynik zostanie zwrócony jako lista elementów. Pozostałe metody są tworzone w sposób analogiczny.

```
@Override
public List<Product> findAll() {
    String hql = "FROM Product";
    return entityManager.createQuery(hql).getResultList();
}
```

Rys 7. Implementacja metody wyszukiwania wszystkich dla Jersey oraz Apache CXF.

W dostępie do danych w implementacji Spring MVC wspomaga go moduł Spring Data z dedykowanymi metodami dla aplikacji RESTowych (m.in. podmoduły Spring Data REST, oraz Spring Data HATEOAS). Dzięki niemu nie jest wymagane tworzenie implementacji metod. Wystarczy sam interfejs rozszerzony o interfejs JpaRepository. Zapewnia on implementacje wszystkich metod CRUD. Aby utworzyć niestandardowe zapytanie także nie jest konieczna implementacja. Wystarczy napisać ciało metody, a moduł ten stworzy sam odpowiednie zapytanie jak na rysunku 8.

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {
    List<Product> findByName(String name);
}
```

Rys 8. Implementacja całej warstwy dostępu do danych dla Spring MVC.

Tabela poniżej prezentuje liczby linii kodu potrzebnych do zaimplementowania wszystkich metod dla warstwy dostępu do danych potrzebnych do stworzenia aplikacji CRUD

Tabela 3. Liczba linii kodu potrzebnych do utworzenia warstwy danych

| Framework | Liczba linii kodu |
|---|---|
| Jersey | 57 |
| Apache CXF | 57 |
| Spring MVC | 3 |

### 6.3. Kontroler

Z uwagi na to iż Jersey i Apache CXF dla REST implementują protokół JAX-RS ich rozwiązania są analogiczne. Aby utworzyć kontroler w tych technologiach należało oznaczyć go komponentem Spring, dodać ścieżkę URL dla całej klasy, oraz wstrzyknąć repozytorium jak zostało to pokazane na rysunku 9.

```
@Component
@Path("product")
public class ProductController {
    @Autowired
    ProductRepository productRepository;
```

Rys 9. Implementacja kontrolera w Jersey oraz Apache CXF.

W Spring MVC należy dodać adnotacje @RestController, oraz także wstrzyknąć repozytorium, natomiast ścieżka dla całej klasy nie jest wymagana co prezentuje rysunek 10.

```
@RestController
public class ProductController {

    @Autowired
    ProductRepository productRepository;
```

Rys 10. Implementacja kontrolera w Spring MVC.

W podejściu Jersey oraz Apache CXF aby utworzyć metodę przechwytującą żądanie HTTP należy dodać adnotacje określająca rodzaj żądania, ścieżkę URL, oraz produkowany lub konsumowany format danych jak na rysunku poniżej.

```
@POST
@Path("/")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Product createProduct(Product product) {
    return productRepository.save(product);
}
```

Rys 11. Implementacja przykładowej metody kontrolera w Jersey oraz Apache CXF.

W podejściu Spring MVC aby utworzyć metodę przechwytującą żądanie HTTP należy dodać adnotacje określającą rodzaj żądania i w nim podać URL jako wartość. Domyślnie dane są produkowane i konsumowane za pomocą plików JSON co zostało przedstawione na rysunku 12.

```
@GetMapping(value = "product")
public List<Product> findAllProducts() {
    return productRepository.findAll();
}
```

Rys 12. Implementacja przykładowej metody kontrolera w Spring MVC.

W celu zmiany domyślnego formatu danych do adnotacji określającej metodę HTTP należy dodać potrzebny MediaType (tj. inny typ danych) jak prezentuje to rysunek 13.

```
@GetMapping(value = "/product/{id}", produces = MediaType.APPLICATION_XML_VALUE)
public Product findProduct(@PathVariable Integer id) {
    return productRepository.findOne(id);
}
```

Rys 13. Implementacja przykładowej zmiany typu danych w metodzie kontrolera dla Spring MVC.

Tabela 4 prezentuje liczby linii kodu potrzebnych do zaimplementowania wszystkich metod do kontrolera typu REST do stworzenia aplikacji CRUD

Tabela 4. Liczba linii kodu potrzebnych do utworzenia kontrolera.

| **Framework** | **Liczba linii kodu** |
|---|---|
| Jersey | 52 |
| Apache CXF | 52 |
| Spring MVC | 4 |

## 7. Wnioski

W obecnych czasach programista powinien skupiać się na realizacji procesów biznesowych i język wraz z dodatkowymi frameworkami i bibliotekami powinien to umożliwiać. Nowoczesne podejście takie jak zastosowanie programowania opartego na adnotacjach, konfiguracji za pomocą spring boota, czy zastosowanie serwisów RESTful wspomaga w tym procesie.

Dobra dokumentacja, oraz popularność frameworka, a co za tym idzie mnogość rozwiązanych już problemów dostępnych w internecie przyśpiesza proces implementowania rozwiązań biznesowych. Z rozdziału 3.4 wynika, iż Spring MVC posiada dobrą dokumentację, JavaDoc i bloga firmowego, oraz jednoznacznie można stwierdzić iż jest najpopularniejszym frameworkiem z prezentowanych w tym artykule w serwisie Stackoverflow.

Z prezentowanych badań można wywnioskować iż Spring MVC umożliwia utworzenie kompletnej aplikacji RESTowej typu CRUD z zachowaniem założeń REST oraz spełniając 2 poziom w modelu dojrzałości Richardsona. Co więcej w zestawieniu z frameworkiem Jersey oraz Apache CXF jest najlepszym do tego celu rozwiązaniem. Argumentem potwierdzającym jest fakt, iż nie wymaga żadnej konfiguracji, a pozostałe tego wymagały. W warstwie dostępu do danych poprzez moduł Spring Data bez dodatkowej implementacji dostępny jest gotowy zestaw metod a tworzenie nowych jest generowane na podstawie ciała metody, a pozostałe frameworki wymagały pełnej implementacji. W warstwie kontrolera zapewnia najkrótsze i najwygodniejsze przyjmowanie żądań HTTP z domyślnym formatem JSON do produkowania i konsumowania danych. Dlatego postawiona teza „Spring MVC daję większe możliwości, niż pozostałe rozwiązania" została udowodniona.

## Literatura


[1] K. Smita, S. Kumar Rath. Performance comparison of SOAP and REST based Web Services for EnterpriseApplication Integration. Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on. IEEE, 2015.

[2] Murat Yener, Alex Theedom, JavaEE Zaawansowane wzorce projektowe, Helion, 2015.

[3] Bhakti Mehta, RESTful Java Patterns and Best Practices, Packt Publishing, 2014.

[4] Praca doktorska Roya Fieldinga, 2000 https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[5] Bill Burke, RESTful Java with JAX-RS 2.0, O'Reilly, 2014.

[6] Martin Fowler, Model dojrzałości Richardsona, 2010 https://martinfowler.com/articles/richardsonMaturityModel.html

[7] Dokumentacja Jersey https://jersey.java.net/documentation/latest/user-guide.html [05.10.2017]

[8] Dokumentacja Apache CXF http://cxf.apache.org/docs/ [05.10.2017]

[9] Dokumentacja Spring MVC .https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html [05.10.2017]

[10] Dokumentacja Spring Boot https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/ [15.10.2017]

[11] Masoud Kalali, Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON, Packt Publishing, 2013

[12] Powstanie Apache CXF https://en.wikipedia.org/wiki/Apache_CXF [17.10.2017]

[13] Definicja REST https://pl.wikipedia.org/wiki/Representational_State_Transfer