

Usługi sieciowe

Technologie Internetowe

Usługi sieciowe

1. Usługa sieciowa, w odróżnieniu od aplikacji sieciowej, nie dostarcza interfejsu użytkownika, ale metody sieciowe, z których może korzystać klient.
2. Klient wysyła do usługi żądanie, które może zawierać dane, serwer wykonuje zlecone zadanie i odsyła odpowiedź z danymi w ustalonym formacie.
3. Usługi sieciowe nie są ograniczone do jednego systemu operacyjnego: są oparte na otwartych standardach.

Usługi sieciowe

Obecnie wykorzystuje się głównie dwa typy usług internetowych: usługi SOAP i usługi REST.

1. SOAP (*Simple Object Access Protokol*) jest protokołem stworzonym przez Microsoft w 1998 roku.
 - Usługa sieciowa SOAP dostarcza metody sieciowe, które może wywoływać aplikacja klienta.
 - Komunikacja między klientem a usługą odbywa się za pomocą metody POST protokołu HTTP, możliwe jest wykorzystanie protokołów SMTP i UDP.
 - Wymiana danych odbywa się w formacie XML.
2. REST (*Representational State Transfer*) jest to styl architektoniczny oprogramowania zaproponowany w 2000 roku w pracy doktorskiej Roya Fieldinga

Usługi sieciowe REST

1. REST jest zbiorem zasad, które można zastosować w dowolnym systemie rozproszonym.
2. Jeśli system spełnia ograniczenia REST, jest nazywany systemem typu RESTful.
3. REST, w odróżnieniu od SOAP, jest nie standardem, lecz zbiorem zasad, których stosowanie jest dobrą praktyką programistyczną.
4. Podstawowym formatem danych wykorzystywanym w usługach REST jest JSON, ale można też wykorzystywać formaty HTML, XML, YAML i zwykły tekst.
5. Komunikacja klienta z usługą odbywa się z wykorzystaniem protokołu HTTP (metody GET, POST, PUT i DELETE).

Wymogi REST

Model klient-serwer

- Pierwszym ograniczeniem jest przyjęcie modelu klient-serwer opartego na komunikacji typu żądanie-odpowiedź.
- Model klient-serwer zapewnia separację komponentów, gdyż klient nie musi znać aspektów działania serwera: musi tylko wiedzieć, jak wysłać żądanie w celu realizacji zleconego zadania.
- Separacja umożliwia niezależną ewolucję komponentów, co poprawia ich skalowalność i przenaszalność.

Bezstanowość.

- Kontekst i stan klienta powinny być przechowywane przez klienta, a nie przez serwer.
- Każde żądanie przesłane na serwer powinno zawierać informację o stanie klienta.
- Serwery i aplikacje mogą być stanowe, ograniczenie wymaga jedynie, aby interakcje pomiędzy klientami i serwerami zawierały informacje o ich stanie.

Wymogi REST

Wykorzystanie pamięci podręcznej.

- Klienci i elementy pośredniczące mogą przechowywać część danych lokalnie, co przyspiesza ich wczytywanie, gdyż nie trzeba ich pobierać z serwera przy każdym żądaniu.

Jednolity interfejs.

- Główną cechą wyróżniającą styl architektoniczny REST jest nacisk na jednolity interfejs, który ma być używany przez wszystkie komponenty systemu.
- Jednoznaczne proste interfejsy można łatwo rozszerzać o różnego typu treści. Ma to istotne znaczenie w Internecie rzeczy, gdzie nowe urządzenia mogą być dodawane i usuwane w każdej chwili, a interakcja z nimi powinna być prosta i standardowa.

System warstwowy.

- Stworzenie jednolitych interfejsów ułatwia zaprojektowanie i budowę systemu warstwowego.
- Styl systemu warstwowego pozwala na złożenie architektury z warstw hierarchicznych poprzez ograniczenie widoczności komponentów w taki sposób, że komponent widzi tylko komponenty warstwy, z którą wchodzi w bezpośrednie interakcje.

Jednolity interfejs

REST jest architekturą zorientowana na zasoby – ROA (*resource-oriented architecture*), w której każdy komponent aplikacji jest nazywany zasobem.

- W Internecie rzeczy zasobami są czujniki, ich parametry oraz elementy wykonawcze.
- Gdy usługa REST jest interfejsem WebAPI bazy danych, zasobami są tabele i wiersze w tabelach.

Jasne i zrozumiałe komunikaty – klienci wykonują na zasobach operacje: Create, Read, Update i Delete; w tym celu mogą używać udostępnionych przez protokół HTTP metod: GET, POST, PUT i DELETE.

Adresowanie zasobów. W usługach REST powinna istnieć możliwość adresowania zasobów za pomocą adresów URL.

Adresowanie zasobów

W usługach REST powinna istnieć możliwość adresowania zasobów za pomocą adresów URL.

URL zasobu powinien mieć składnię: `<schemat>://<podmiot><ścieżka>[?zapytanie][#fragment]`

Podczas tworzenia nazw zasobów powinno się kierować następującymi wytycznymi:

1. nazwy powinny być opisowe,
2. w adresach URL nie powinno się używać czasowników, zarezerwowanych dla metod protokołu HTTP, zamiast `/garagedoor/open` należy stosować `/garagedoor/status`,
3. grupy (kolekcje) zasobów powinny być nazywane rzeczownikami w liczbie mnogiej, np. `/sensors` – czujniki, czy `/actuators` – elementy wykonawcze.

Poniżej przykłady adresów URL zasobów zgodnych z powyższymi wytycznymi typowe dla Internetu rzeczy:

- ♦ `http://lukan.sytes.net/dom/salon/actuators/leds/led7`
- ♦ `http://lukan.sytes.net/dom/salon/actuators/lights/main/status`
- ♦ `http://lukan.sytes.net/dom/kuchnia/sensors/Bme280/pressure_hPa`
- ♦ `http://lukan.sytes.net/dom/kuchnia/sensors/Bme280/temperature_C`

<http://lukan.sytes.net:1880/sensors/bme280/>

http://lukan.sytes.net:1880/sensors/bme280/pressure_hPa

http://lukan.sytes.net:1880/sensors/bme280/temperature_C

<https://lukan.sytes.net:1881/sensors/bme280/>

https://lukan.sytes.net:1881/sensors/bme280/pressure_hPa

https://lukan.sytes.net:1881/sensors/bme280/temperature_C

Metody REST

Metody klasyfikuje się pod kątem ich bezpieczeństwa i idempotentności.

Idempotentność oznacza, że niezależnie od tego, ile razy operacja zostanie wykonana, stan zasobu się nie zmienia.

GET Metoda GET jest przeznaczona do odczytu stanu zasobu.

- Jest to metoda bezpieczna – odczyt nie zmienia stanu zasobu, i idempotentna – nie zmienia go również wielokrotny odczyt tego stanu.
- Za pomocą tej metody należy czytać (pobierać) dokumenty reprezentujące stan zasobu – może to być dokument HTML lub stan zasobu systemu Internetu rzeczy zapisany w formacie JSON lub innym.
- Jeśli klient chce sprawdzić tylko, czy zasób jest dostępny, może użyć metody HEAD. Na żądanie metodą HEAD serwer odsyła w odpowiedzi tylko kolekcję nagłówków.

POST Metody POST należy używać do tworzenia nowych zasobów, które jeszcze nie mają własnego identyfikatora – adresu URL.

- Operacja tworzenia nowych zasobów nie jest bezpieczna, ponieważ zmienia stan serwera. Nie jest też idempotentna, ponieważ każde jej wykonanie tworzy nowy zasób.

PUT Metody PUT należy używać do modyfikowania zasobów.

- Operacja modyfikacji zasobów nie jest bezpieczna, ponieważ zmienia stan serwera. Jest idempotentna, ponieważ wielokrotne wysłanie tego samego żądania PUT daje ten sam efekt.
- Żądań PUT należy używać do zmiany stanu czegoś, co już istnieje, a nie do tworzenia nowych zasobów, do czego jest przeznaczona metoda POST.
- W Internecie rzeczy żądanie PUT powinno być wykorzystywane do zmiany stanu zasobu, np. do włączenia/wyłączenia grzejnika, zapalenia/zgaszenia światła.

DELETE Metody DELETE należy używać do usuwania zasobu.

- Metoda jest idempotentna i niebezpieczna.

Publiczne usługi REST

W Internecie dostępnych jest wiele ogólnodostępnych usług w większym lub mniejszym stopniu spełniających wymagania REST, w części bezpłatnych. Korzystanie z usług bezpłatnych często wymaga rejestracji użytkownika. Adresy tych usług są publikowane w Internecie.

Linki do stron z listami ogólnie dostępnych usług:

- ♦ <https://any-api.com/>,
- ♦ <https://rapidapi.com/collection/list-of-free-apis>,
- ♦ <https://github.com/public-apis/public-apis>.

Ogólnodostępne API zwykle publicznie udostępniają możliwość czytania zasobów metodą GET, bez możliwości ich modyfikacji.

Ważną częścią API jest jego opis, bez którego klienci nie mogą z niego korzystać. Do najczęściej wykorzystywanych do testowania należą API udostępniające dane o pogodzie. Oto przykłady usług:

♦ prognoza pogody:

- <http://www.7timer.info/bin/api.pl?lon=18.31&lat=54.3&product=astro&output=json> ,
- <http://www.7timer.info/bin/api.pl?lon=18.31&lat=54.3&product=astro&output=xml> ,
- <https://www.metaweather.com/api/> ,
- <https://openweathermap.org/api>,

♦ kursy walut:

- <https://api.ratesapi.io/api/2019-11-30>, na końcu należy podać datę,
- <https://api.exchangeratesapi.io/latest> (kursy bieżące).

WebAPI

- API – *Application Programming Interface*) – interfejs programowania aplikacji, interfejs programistyczny aplikacji – sposób w jaki programy komunikują się ze sobą.
- WebAPI – sposób w jaki programy komunikują się ze sobą przez sieć.
- Usługi sieciowe SOAP i REST tworzą interfejs WebAPI.
- Facebook, Twitter i mapy Google udostępniają API, z którego można korzystać np. z Node-RED.

Usługa RESTful WebAPI na argo

Linki:

<http://argo.umg.edu.pl/www/RestWebApi/Help>

<http://argo.umg.edu.pl/www/RestWyniki>

Odczyt BME280

<https://lukan.sytes.net:1881/sensors/bme280>

Przykład systemu wykorzystującego HTTP i usługę WebAPI do komunikacji dwóch maszyn nie posiadających publicznego IP, łączność między maszynami A i B jedynie poprzez usługę i wspólny zbiór danych na serwerze



Przykładowa Usługa WebAPI realizuje operacje CRUD na tabeli Measurement o następującej strukturze

Name	Description	Type	Additional information
MeasurementId	Identyfikator pomiaru nadawany automatycznie, można go pominąć.	integer	None.
Author	Autor pomiaru.	string	String length: inclusive between 0 and 50
Gage	Opis przyrządu lub co jest mierzone.	string	String length: inclusive between 0 and 50
Date	Data pomiaru, można pominąć, wtedy będzie wpisana bieżąca data i czas.	date	None.

Metody usługi WebAPI realizującej operacje CRUD na tabeli Measurement

API	Description
GET api/Measurements	Pobiera dane wszystkich pomiarów .
GET api/Measurements/{id}	Pobiera dane pomiaru o identyfikatorze id.
PUT api/Measurements/{id}	Zmienia dane pomiaru o identyfikatorze id.
POST api/Measurements	Wprowadza dane nowego pomiaru.
DELETE api/Measurements/{id}	Usuwa dane pomiaru o identyfikatorze id.

Żądanie metodą GET i odpowiedź w formacie JSON

The screenshot shows the Postman application interface. At the top, there's a menu bar with 'File', 'Edit', 'View', and 'Help'. Below it is a toolbar with buttons for 'New', 'Import', 'Runner', and 'Invite'. The main workspace shows a list of requests, with the selected one being a GET request to 'http://ip-api.com/j...'. The 'Untitled Request' section shows the method 'GET' and the URL 'http://argo.am.gdynia.pl/www/RestWebApi/api/Measurements'. The 'Headers' tab is active, showing a table with headers. The 'Body' tab is also active, showing the response in JSON format. The response is a 200 OK status with a time of 145ms and a size of 1.4 KB. The JSON response contains two measurement objects.

Postman

File Edit View Help

New Import Runner Invite

My Workspace Invite

No Environment

GET http://argo.am.gdynia.pl/www/RestWebApi/api/Measurements

Send Save

Params Auth Headers (11) Body Pre-req. Tests Settings Cookies Code

Headers (3)

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
☒	Accept	application/json		
☒	Content-Type	application/json		
☐	Accept	application/xml		
	Key	Value	Description	

Temporary Headers (8)

Status: 200 OK Time: 145ms Size: 1.4 KB

Body Cookies Headers (9) Test Results (1/1)

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   {
3     "MeasurementId": 50,
4     "Author": "Tester2",
5     "Gage": "y - temperatura, y - ciśnienie, z - sinus",
6     "Date": "2019-11-28T23:36:04.1411015"
7   },
8   {
9     "MeasurementId": 104,
10    "Author": "Jakiś student",
11    "Gage": "Jakiś miernik",
12    "Date": "2019-12-06T01:35:53.8751914"
13  },
14  {
15    "MeasurementId": 105
```

Żądanie metodą GET i odpowiedź w formacie XML

The screenshot shows the Postman application interface. At the top, there's a menu bar with 'File', 'Edit', 'View', and 'Help'. Below it is a toolbar with 'New', 'Import', 'Runner', and a plus icon. The main workspace shows 'My Workspace' and an 'Invite' button. A list of requests is visible at the top, with the selected one being 'GET http://ip-api.com/j...'. The 'Untitled Request' section shows a 'GET' method and the URL 'http://argo.am.gdynia.pl/www/RestWebApi/api/Measurements'. The 'Headers' tab is active, showing a table with headers. The 'Accept' header is highlighted with a red circle. The 'Body' tab is also active, showing the response in XML format. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 35ms', and 'Size: 2.09 KB'.

Postman

File Edit View Help

New Import Runner +

My Workspace Invite

GET http://argo.am.gd... GET http://argo.am.gd... POST http://argo.am.g... GET http://argo.am.gd... GET http://ip-api.com/... GET http://ip-api.com/j... + ... No Environment

Comments (0)

GET http://argo.am.gdynia.pl/www/RestWebApi/api/Measurements Send Save

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies Code

Status: 200 OK Time: 35ms Size: 2.09 KB Save Response

Body Cookies Headers (9) Test Results (1/1)

Pretty Raw Preview Visualize BETA XML

```
1 <ArrayOfMeasurement xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
2   <Measurement>
3     <Author>Tester2</Author>
4     <Date>2019-11-28T23:36:04.1411015</Date>
5     <Gage>y - temperatura, y - ciśnienie, z - sinus</Gage>
6     <MeasurementId>50</MeasurementId>
7   </Measurement>
8   <Measurement>
9     <Author>Jakiś student</Author>
10    <Date>2019-12-06T01:35:53.8751914</Date>
11    <Gage>Jakiś miernik</Gage>
12    <MeasurementId>104</MeasurementId>
13  </Measurement>
```

Bootcamp

Metoda POST - dodawanie wierszy do tabeli

- **Format żądania: application/json, text/json**

```
{  
  "MeasurementId": 1,  
  "Author": "sample string 2",  
  "Gage": "sample string 3",  
  "Date": "2020-02-01T19:19:38.934637+01:00"  
}
```

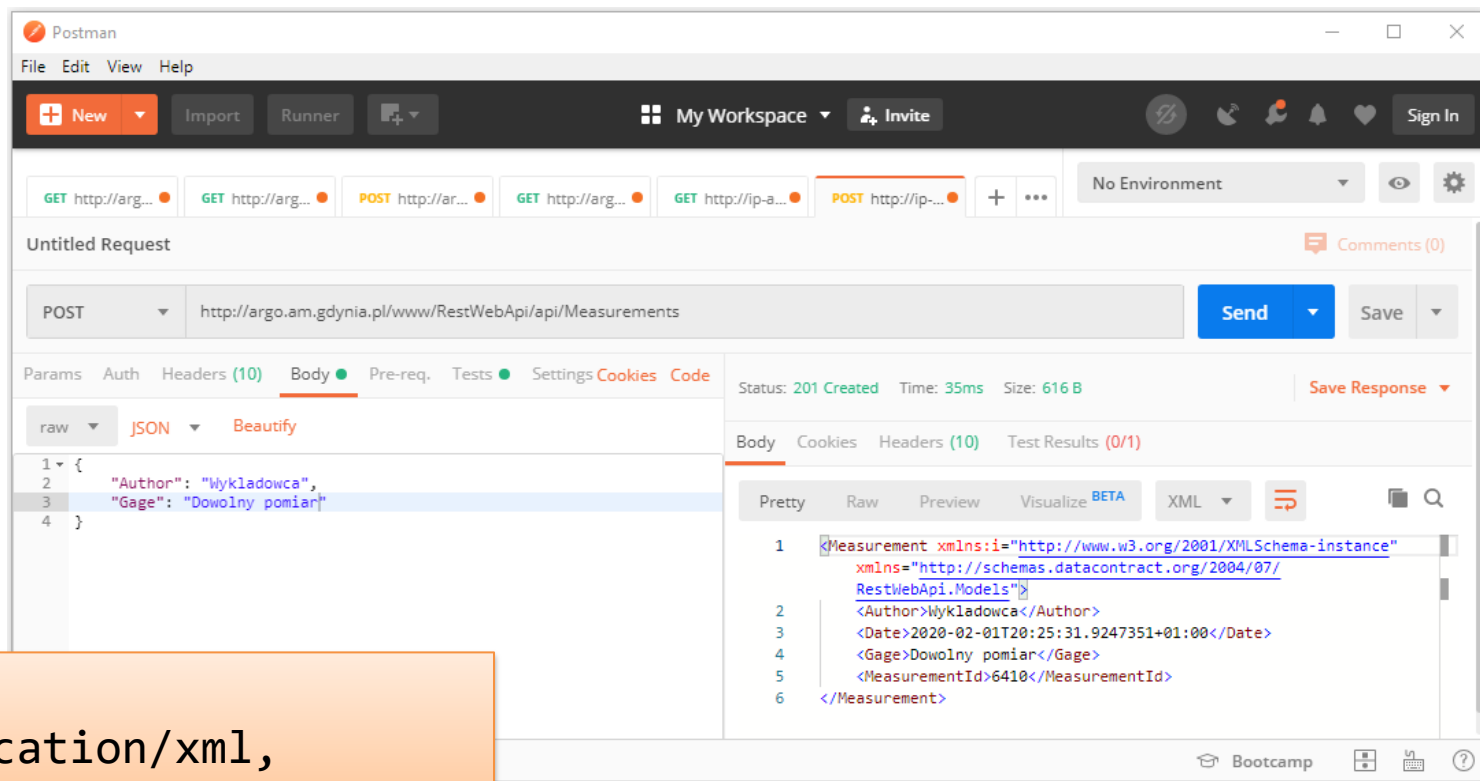
- **Format żądania: application/xml, text/xml**

```
<Measurement xmlns:i="http://www.w3.org/2001/XMLSchema-  
instance" xmlns="http://schemas.datacontract.org/2004/07/RestWebApi.Models">  
  <Author>sample string 2</Author>  
  <Date>2020-02-01T19:19:38.934637+01:00</Date>  
  <Gage>sample string 3</Gage>  
  <MeasurementId>1</MeasurementId>  
</Measurement>
```

- **Format odpowiedzi**

- odpowiedź jest dodany do tabeli wiersz w odpowiednim z powyższych formatów JSON lub XML, pola "MeasurementId" i "Date" są opcjonalne i w żądaniu można je pominąć, w odpowiedzi dostaje się pełne dane z wartościami domyślnymi dodanymi przez serwer

Żądanie metodą POST i odpowiedź XML – dodanie wiersza (zasobu)



Nagłówki:

Accept: application/xml,

Content-Type: application/json