Komunikacja urządzeń mobilnych

Protokół HTTP, usługi sieciowe REST

PUM

Protokół HTTP

- HTTP (ang. *Hypertext Transfer Protocol*) protokół przesyłania dokumentów hipertekstowych.
- Protokół określa format żądania klienta wysyłanego na serwer i format odpowiedzi serwera.
- Żądanie poza adresem zwrotnym może zawierać parametry przekazywane do aplikacji serwerowej np. dane formularza przesłane metodą GET lub POST.
- Żądanie metodą GET można wysłać z dowolnej przeglądarki.
- Protokół jest bezstanowy – nie zapamiętuje stanu sesji z klientem.
- Specyfikacja protokołu w dokumencie RFC 2616.

Protokół HTTP Metody

Żądanie HTTP rozpoczyna się od podania nazwy metody. Metoda jest umowną nazwą działania, które powinno być wykonane po stronie serwera. Podstawowe metody HTTP to:

- GET – pobranie zasobu wskazanego przez URI, może mieć postać warunkową jeśli w nagłówku występują pola warunkowe takie jak "If-Modified-Since"
- HEAD – pobiera informacje o zasobie, stosowane do sprawdzania dostępności zasobu
- PUT – przyjęcie danych przesyłanych od klienta do serwera, najczęściej aby zaktualizować wartość encji
- POST – przyjęcie danych przesyłanych od klienta do serwera (np. wysyłanie zawartości formularzy)
- DELETE – żądanie usunięcia zasobu, włączone dla uprawnionych użytkowników
- OPTIONS – informacje o opcjach i wymaganiach istniejących w kanale komunikacyjnym
- TRACE – diagnostyka, analiza kanału komunikacyjnego
- CONNECT – żądanie przeznaczone dla serwerów pośredniczących pełniących funkcje tunelowania
- PATCH – aktualizacja części danych

Metody powinny być implementowane zgodnie z powyższym opisem, chociaż protokół tego nie wymusza. Stosowanie się do powyższych zaleceń jest dobrą praktyką programistyczną.

Żądanie HTTP

Po nazwie metody podawany jest identyfikator zasobu i wersja protokołu. Identyfikator zasobu pobierany jest z adresu URL. URL (*Uniform Resource Locator* – jednolity lokalizator zasobu) ma postać:

<schemat>:/<podmiot><ścieżka>/[?zapytanie][#fragment]

gdzie:
- ♦ <schemat> to http albo https,
- ♦ <podmiot> to nazwa lub IP hosta z opcjonalnym numerem portu i danymi uwierzytelniającymi,
- ♦ <ścieżka> to hierarchiczna ścieżka dostępu do zasobu, musi zaczynać się od znaku /,
- ♦ [zapytanie] jest opcjonalne i zawiera opcjonalne parametry.

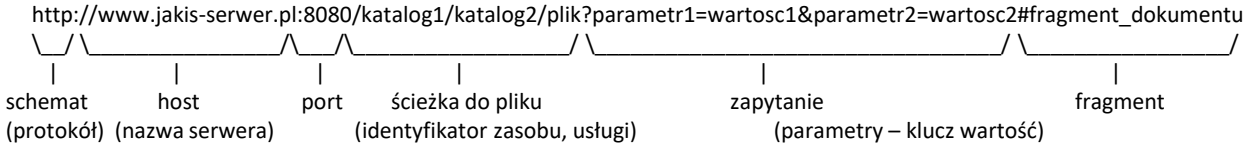
Dla następującego adresu URL wpisanego do przeglądarki:
http://lukan.sytes.net:1880/test/get?parametr1=w1&parametr2=w2
początek żądania będzie miał postać:

```
GET /test/get?parametr1=w1&parametr2=w2 HTTP/1.1
Host: lukan.sytes.net:1880
```

Druga linia żądania to obowiązkowy nagłówek Host. Po nagłówku obowiązkowym mogą pojawić się nagłówki opcjonalne.

Uniform Resource Identifier - URI

- *Ujednolicony Identyfikator Zasobów – identyfikuje zasoby w sieci*



Protokół HTTP Nagłówki

- Nagłówki są częścią składową żądania i odpowiedzi, zawierają informacje dla serwera i przeglądarki o parametrach żądania i odpowiedzi.
- Nagłówki między innymi informują drugą stronę w jakim formacie przesyłane są dane (treść).
- Standardowych nagłówków jest kilkadziesiąt, można dodawać własne niestandardowe nagłówki.
- Nagłówki są przesyłane jako kolekcja par nazwa-wartość.

Ważne nagłówki

Nazwa	Często stosowane wartości
Accept – informuje serwer o akceptowanych przez klienta typach dokumentu MIME, Content-Type – informuje o formacie MIME przesyłanej treści	text/plain text/html application/text; charset=utf-8 application/json application/xml application/soap+xml application/x-www-form-urlencoded
Accept-Encoding, Content-Encoding	gzip
Accept-Language, Content-Language	en, pl
Accept-Charset,	utf-8, iso-8859-1, iso-8859-2
Cookie	name=value; name2=value2; name3=value3
Allow – informuje o metodach HTTP obsługiwanych przez serwer	GET, POST, HEAD
If-Modified-Since – nakazuje przesłać dokument, jeśli został zmodyfikowany po podanej dacie	Data np.: 30 Oct 2019 20:00:00 GM

Odpowiedź HTTP

- Po otrzymaniu żądania serwer wykonuje zleconą akcję i wysyła odpowiedź. Akcją może być tylko przygotowanie odpowiedzi, co jest bardzo częste, gdy klient przegląda zasoby Internetu.
- W aplikacjach Internetu rzeczy akcją może być włączenie lub wyłączenie urządzeń, przekazanie do urządzeń komend sterujących, odczytanie stanu urządzeń lub wskazań przyrządów pomiarowych.
- W usługach internetowych akcją może być stworzenie na serwerze jakiegoś zasobu, jego modyfikacja lub usunięcie.
- Odpowiedzią najczęściej jest dokument, który można wyświetlić w przeglądarce, ale może to być także plik binarny, graficzny czy dane zapisane w dowolnym formacie. Odpowiedź, podobnie jak żądanie, składa się z nagłówków i treści. Nagłówki są podobne do nagłówków żądania.
- Odpowiedź zaczyna się od wersji protokołu HTTP, po którym następuje liczbowy kod statusu odpowiedzi i słowny opis statusu odpowiedzi np.:

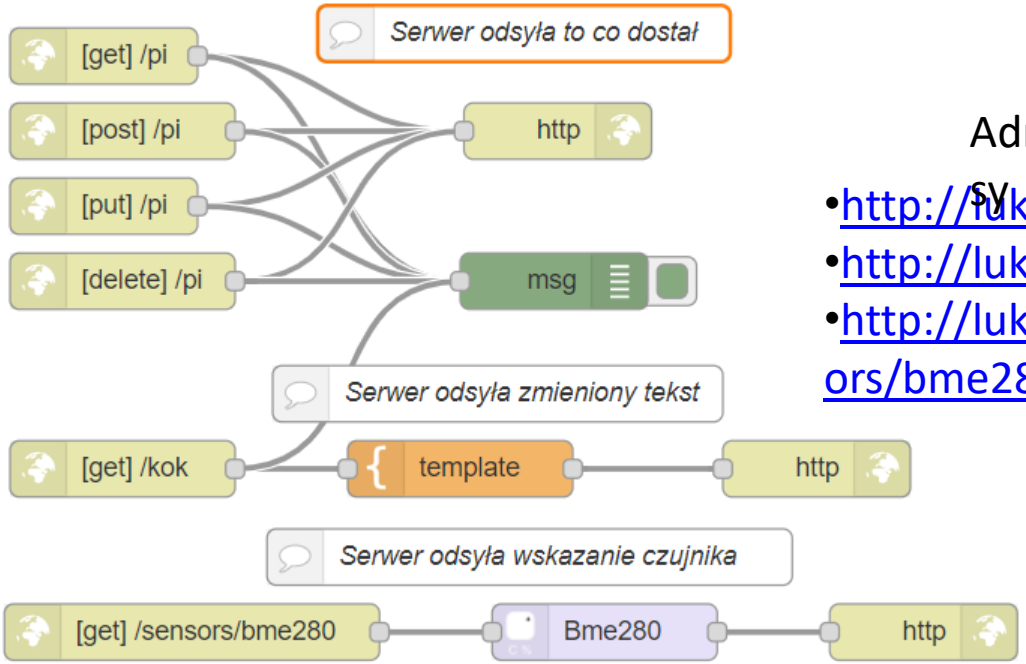
HTTP/1.1 200 OK
HTTP/1.1 400 Bad Request

- Kod status odpowiedzi informuje aplikację klienta o statusie realizacji jego żądania.

Często stosowane kody statusu odpowiedzi

Kod	Opis słowny	Znaczenie, zwrócony zasób
Kody informacyjne		
110	Connection Timed Out	Przekroczono czas połączenia, serwer zbyt długo nie odpowiada
111	Connection refused	Serwer odrzucił połączenie
Kody powodzenia		
200	OK	Zwrócono żądany dokument – najczęściej zwracany status
201	Created	Wysłany dokument został zapisany na serwerze
202	Accepted	Żądanie zostało przyjęte, ale jego realizacja jeszcze się nie skończyła
204	No content	Serwer zrealizował żądanie klienta i nie potrzebuje zwracać żadnej treści
Kody przekierowania		
304	Not Modified	Nie zmieniono zasobu, zawartość zasobu nie podległa zmianie według warunku przekazanego przez klienta (np. daty ostatniej wersji zasobu pobranej przez klienta i zapamiętanej w pamięć podręcznej przeglądarki)
Kody błędów po stronie klienta		
400	Bad Request	Żądanie nie może być obsłużone przez serwer z powodu nieprawidłowości postrzeganej jako błąd użytkownika
401	Unauthorized	Nieautoryzowany dostęp – realizacja żądania wymaga uwierzytelnienia
403	Forbidden	Serwer zrozumiał zapytanie, lecz konfiguracja bezpieczeństwa zabrania mu zwrócić żądany zasób
404	Not Found	Klient połączył się z serwerem, ale serwer nie może znaleźć żądanego zasobu – częsty błąd
Kody błędów po stronie serwera		
500	Internal Server Error	Wewnętrzny błąd serwera – serwer napotkał błąd, który uniemożliwił zrealizowanie żądania
501	Not Implemented	Nie zaimplementowano – serwer nie dysponuje funkcjonalnością określoną w żądaniu

Przykład lekkiego serwera zrealizowanego w Node-RED



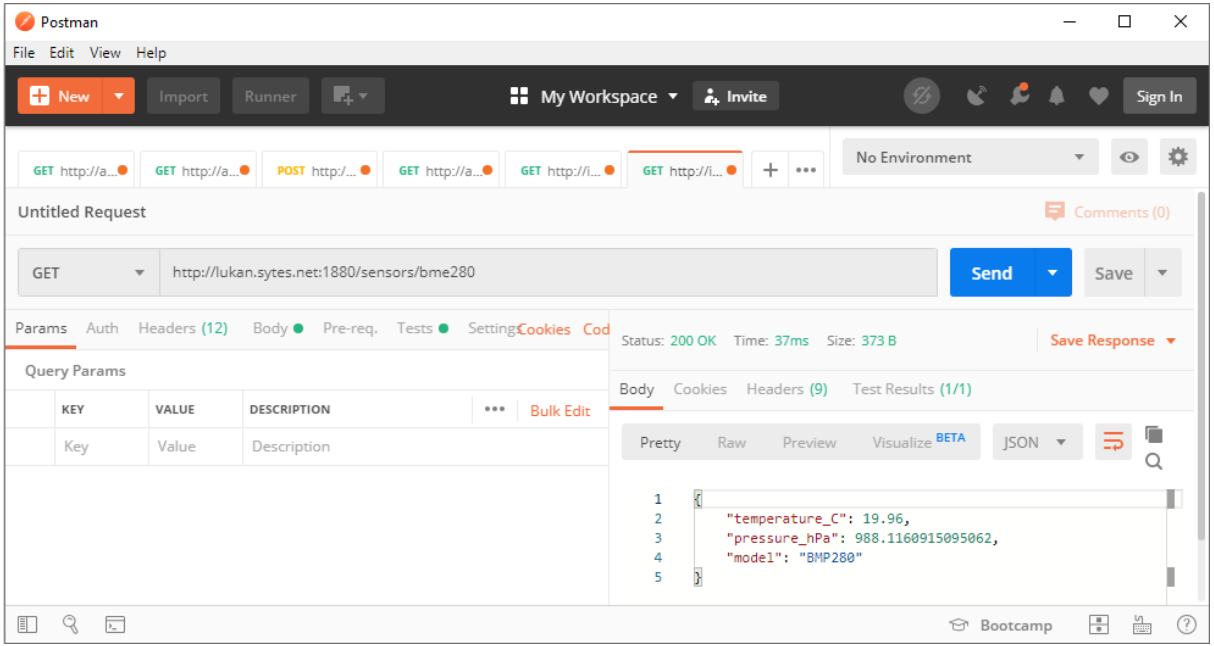
Adresy

- http://lukan.sytes.net:1880/pi
- http://lukan.sytes.net:1880/kok
- http://lukan.sytes.net:1880/sensors/bme280

- Graf http://lukan.sytes.net:1880/#flow/95c9e167.8f0de

Postman

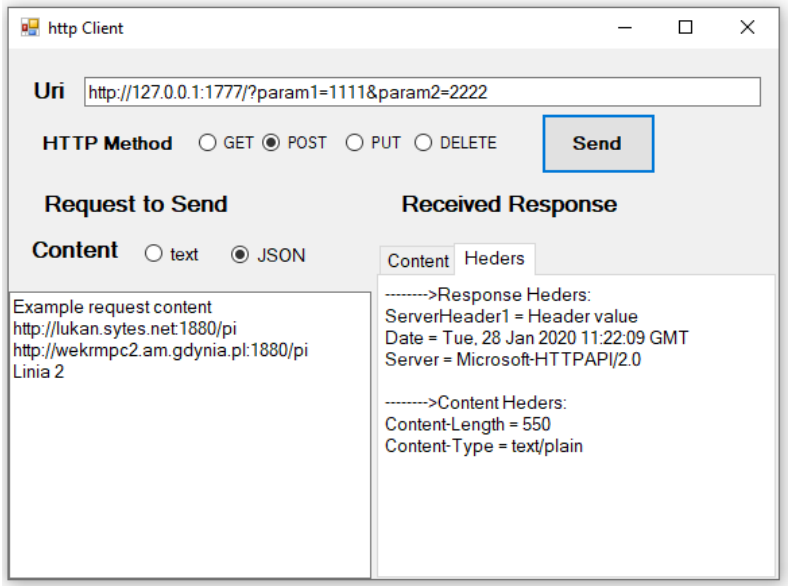
- Program do testowania usług korzystających z protokołu HTTP.
- Przeglądarka może wysyłać żądania GET z linii adresu, pozostałymi metodami z formularza na stronie.
- Program Postman wysyła żądania wieloma metodami z dowolną treścią i nagłówkami.

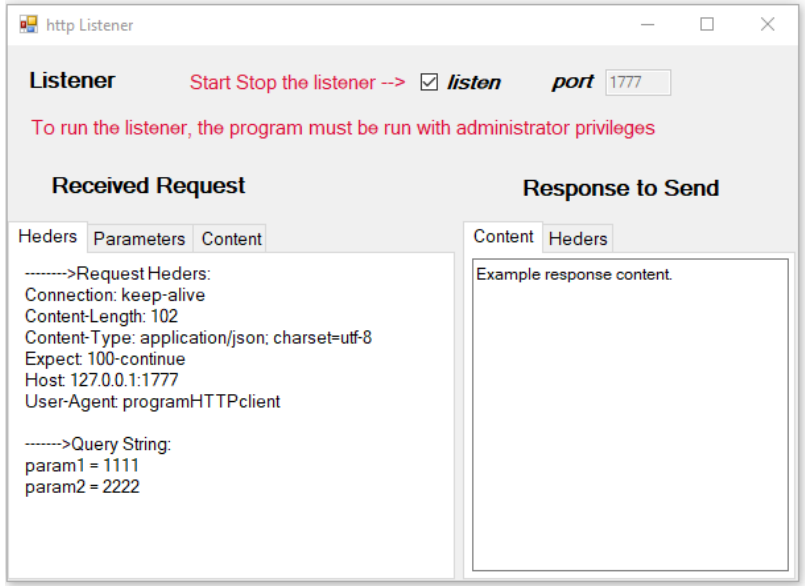


Programy klienta i serwera HTTP

Program: http://argo.umg.edu.pl/www/Internet/SendHttpReguest/
Kod: https://github.com/Andrzej1959/SendHttpRequest

Program: http://argo.umg.edu.pl/www/Internet/Listener/publikacja.htm
Kod: https://github.com/Andrzej1959/TestHttpListener





Message Queuing Telemetry Transport (MQTT)

- Protokół MQTT jest prostym i lekkim protokołem warstwy aplikacji.
- Protokół MQTT oparty o wzorzec publikacja/subskrypcja.
- Przeznaczony jest do transmisji dla urządzeń niewymagających dużej przepustowości.
- Poprzez ograniczenie prędkości transmisji, protokół zapewnia większą niezawodność.
- Protokół ten idealnie sprawdza się przy połączeniach maszyna-maszyna, w Internecie rzeczy, w urządzeniach mobilnych, oraz tam, gdzie wymagana jest oszczędność przepustowości, oraz energii.
- **Broker MQTT** pełni rolę serwera, z którym łączą się klienci (publikatorzy i subskrybenci), aby za jego pośrednictwem publikować i subskrybować wiadomości.
- Broker wymaga publicznego adresu IP, publikatorzy i subskrybenci nie wymagają.

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.

Formaty kodowania danych stosowane w usługach sieciowych

JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
     "streetAddress": "21 2nd Street",
     "city": "New York",
     "state": "NY",
     "postalCode": "10021"
   },
  "phoneNumber":  [
              {
                "type": "home",
                "number": "212 555-1234"
              },
              { "type": "fax",
                "number": "646 555-4567"
              }
   ],
  "gender": {
                  "type": "male"
   }
}
```

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
<firstName>John</firstName><lastName>Smith</lastName>
<age>25</age>
<address>
<streetAddress>21 2nd Street</streetAddress>
<city>New York</city>
<state>NY</state>
<postalCode>10021</postalCode></address>
<phoneNumber>
<type>home</type><number>212 555-
1234</number></phoneNumber>
<phoneNumber>
<type>fax</type><number>646 555-4567</number></phoneNumber>
<gender><type>male</type></gender>
</root>
```

Formaty kodowania danych stosowane w usługach sieciowych

JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
     "streetAddress": "21 2nd Street",
     "city": "New York",
     "state": "NY",
     "postalCode": "10021"
   },
  "phoneNumber":  [
              {
                "type": "home",
                "number": "212 555-1234"
              },
              { "type": "fax",
                "number": "646 555-4567"
              }
   ],
  "gender": {
                  "type": "male"
   }
}
```

YAML

```
firstName: John
lastName: Smith
age: 25
address:
  streetAddress: 21 2nd Street
  city: New York
  state: NY
  postalCode: '10021'
phoneNumber:
  - type: home
    number: 212 555-1234
  - type: fax
    number: 646 555-4567
gender:
  type: male
```

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 niezależna 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 zmieni.

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

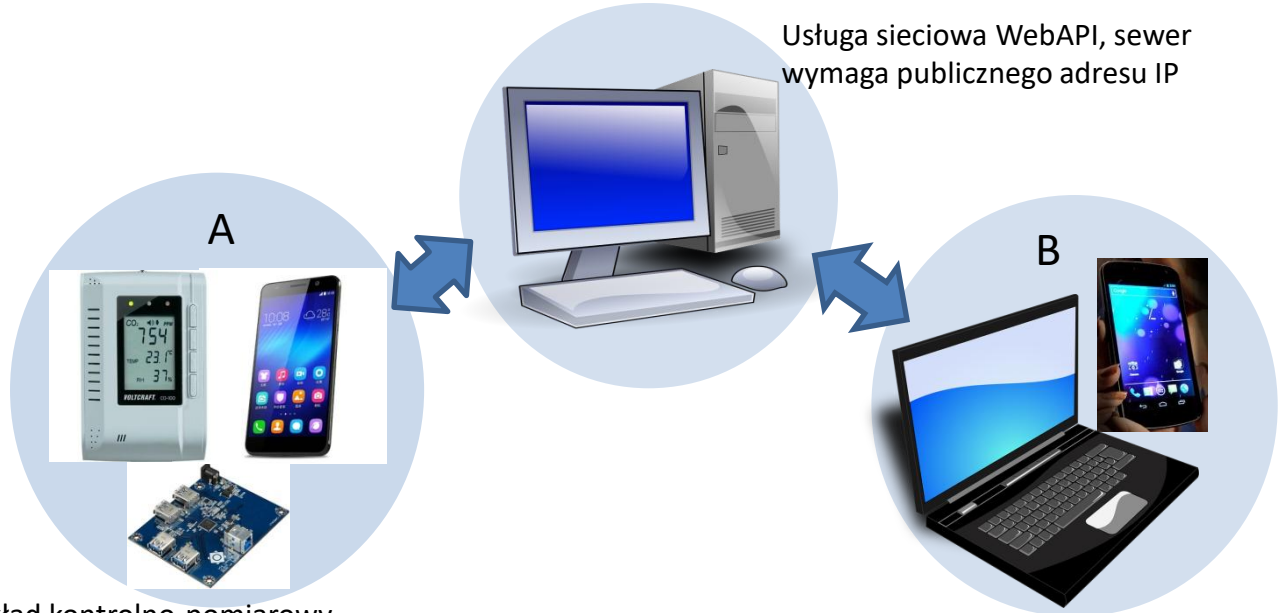
- API – *Application Programming Interface*) – interfejs programowania aplikacji, interfejs programistyczny aplikacji – sposób w jaki programy komunikują się ze sobą.
- WebAPI – sposób w jaki programy komunikują się ze sobą przez sieć.
- Usługi sieciowe SOAP i REST tworzą interfejs WebAPI.
- Facebook, 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



Usługa sieciowa WebAPI, sewer
wymaga publicznego adresu IP

Układ kontrolno-pomiarowy
klient HTTP nie ma publicznego adresu IP

Zdalne sterowanie i odczyt pomiarów
klient HTTP nie ma publicznego adresu IP

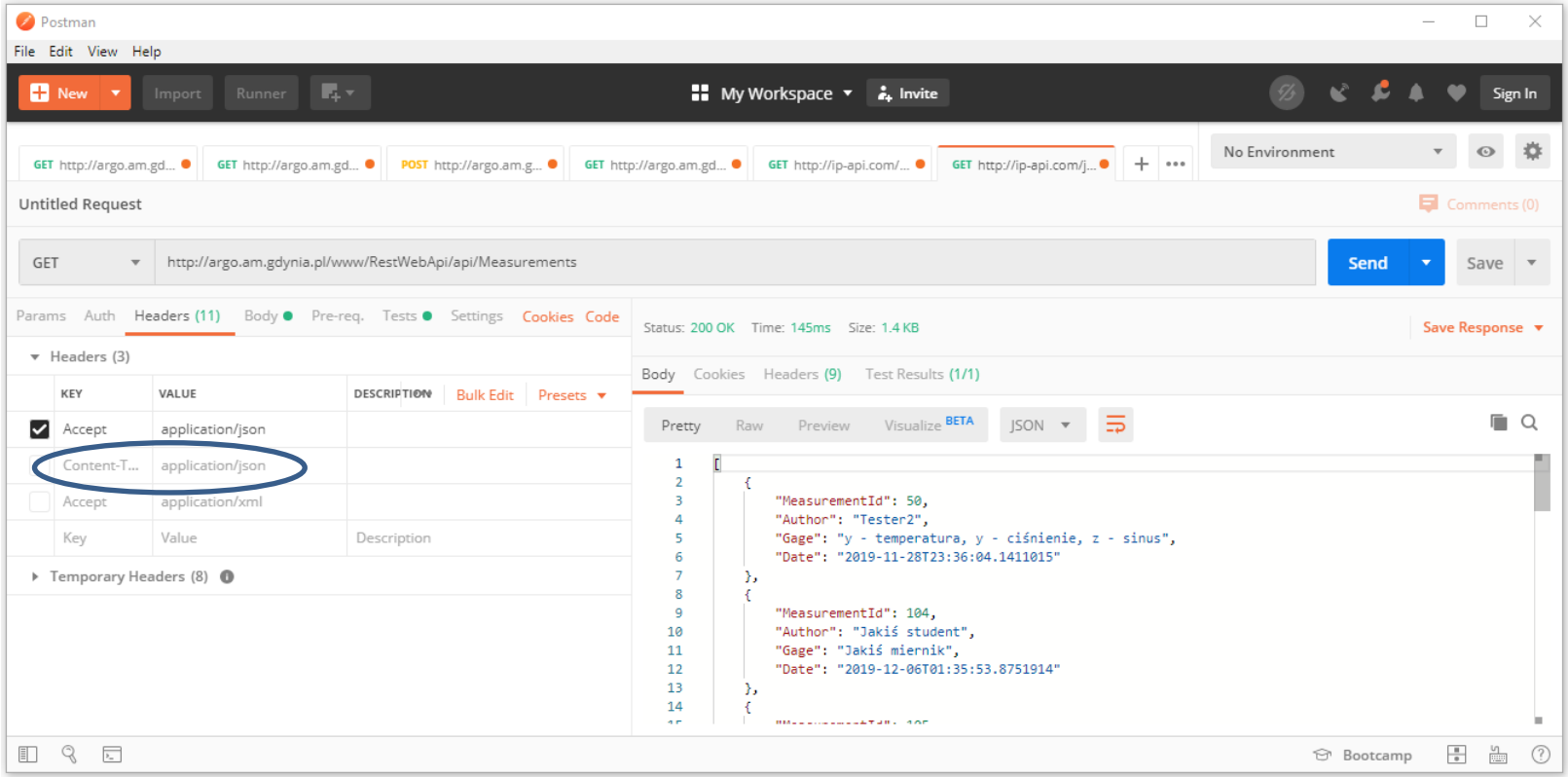
Przykładowa Usługa WebAPI realizuje operacje CRUD na tabeli Measurement o następującej strukturze

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.

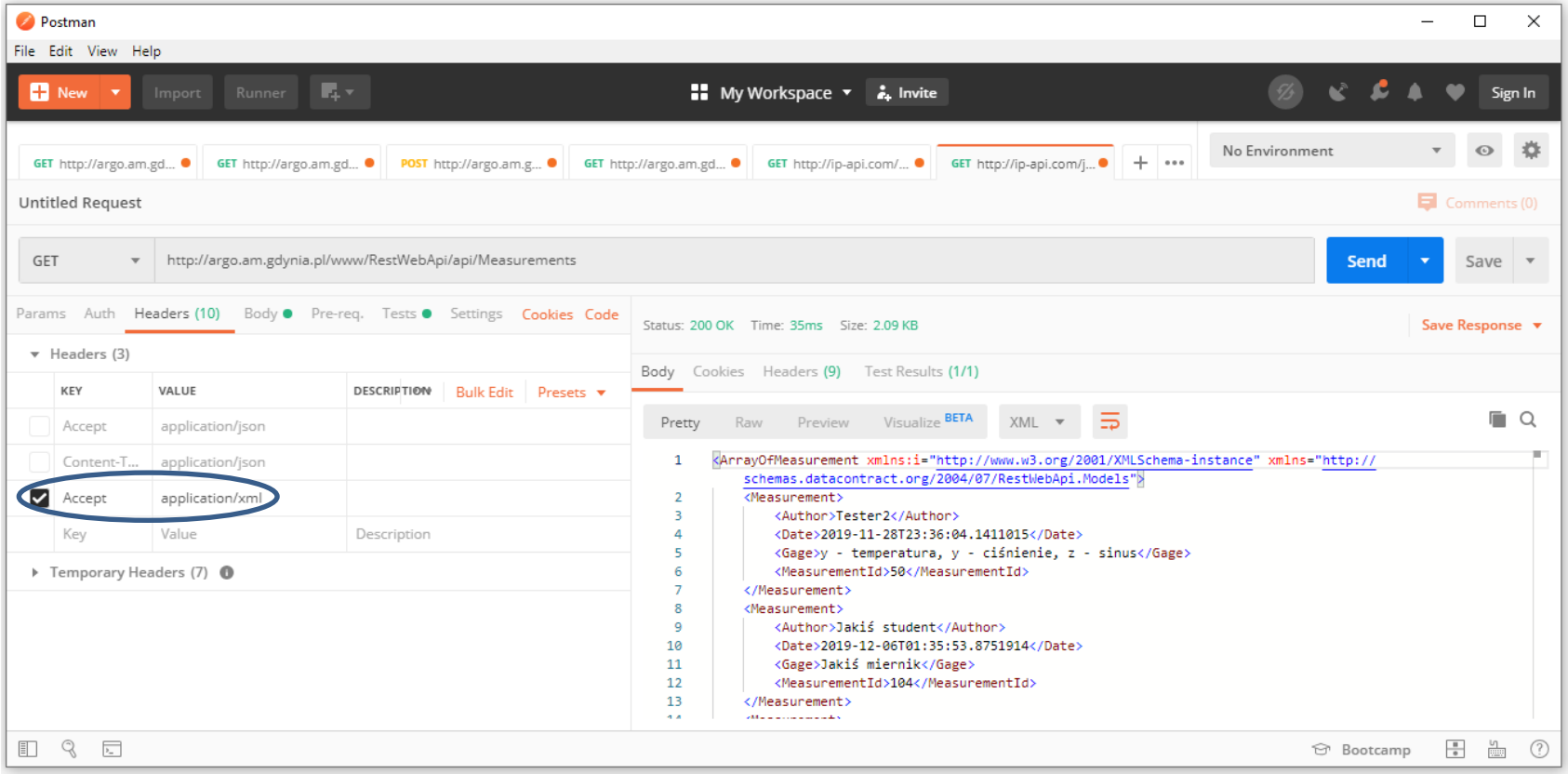
Metody usługi WebAPI realizującej operacje CRUD na tabeli Measurement

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



Żądanie metodą GET i odpowiedź w formacie XML



Metoda POST - dodawanie wierszy do tabeli

- Format żądania: application/json, text/json

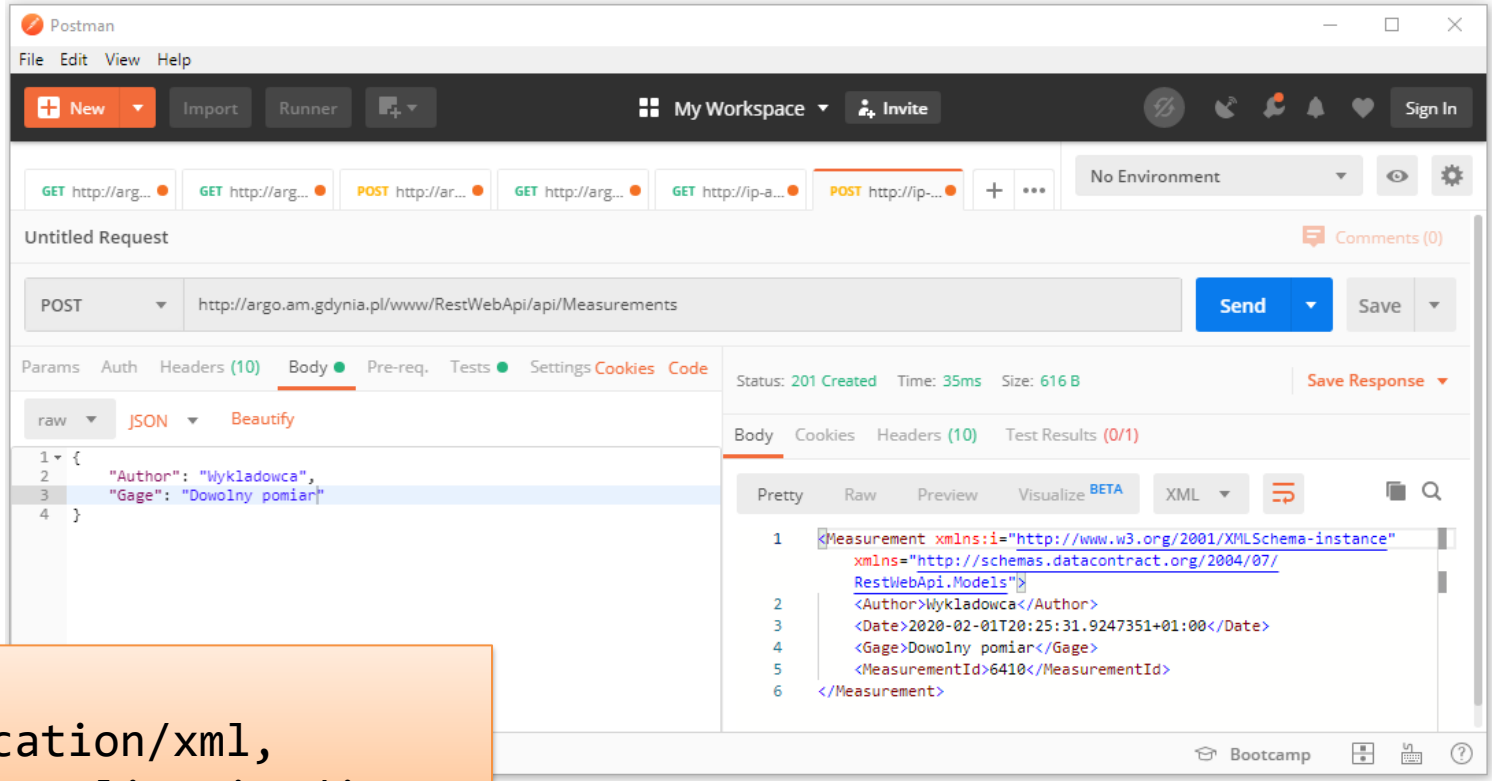
```
{
  "MeasurementId": 1,
  "Author": "sample string 2",
  "Gage": "sample string 3",
  "Date": "2020-02-01T19:19:38.934637+01:00"
}
```

- 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
  - odpowiedzą 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