

4. Protokół HTTP

HTTP (ang. *Hypertext Transfer Protocol*) to protokół przesyłania dokumentów hipertekstowych. Protokół określa format żądania (zapytania) klienta wysyłanego na serwer i format odpowiedzi serwera. Protokół jest bezstanowy, to znaczy, że nie zapamiętuje stanu sesji z klientem. Specyfikacja protokołu jest opublikowana w dokumencie RFC 2616 [8,17].

Żądanie HTTP

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 – żądanie pobrania zasobu wskazanego przez URL, może mieć postać warunkową, jeśli w nagłówku występują pola warunkowe takie jak *If-Modified-Since*,
- HEAD – żądanie informacji o zasobie, stosowane do sprawdzania dostępności zasobu,
- PUT – żądanie przyjęcia danych przesyłanych od klienta do serwera, metoda najczęściej stosowana, aby zaktualizować istniejący zasób,
- POST – żądanie przyjęcia danych przesyłanych od klienta do serwera (np. wysyłanej przez klienta zawartości formularza), metoda najczęściej stosowana, aby utworzyć nowy zasób,
- 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 – żądanie aktualizacji części danych na serwerze, podobne do PUT.

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

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¶metr2=w2`

początek żądania będzie miał postać:

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

GET /test/get?parametr1=w1¶metr2=w2 HTTP/1.1
Host: lukan.sytes.net:1880

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

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 klucz wartość. W tabeli 4.1 przedstawiono ważne i często stosowane nagłówki wraz z ich najczęstszymi wartościami.

Nazwa nagłówka	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 – zawiera kolekcję ciasteczek	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 GMT

Tabela 4.1. Lista ważnych i często stosowanych nagłówków wraz z ich najczęstszymi wartościami

Po nagłówkach w żądaniu może pojawić się treść dokumentu (body, Content) zawierająca przesyłane na serwer dane.

Żądanie metodą GET można wysłać z dowolnej przeglądarki wpisując odpowiedni adres URL w linii adresu. Żądania innymi metodami przeglądarka może wysłać ze strony wyświetlonej w przeglądarce. Żądania HTTP wszystkimi metodami można wysyłać z programów tworzonych w różnych językach programowania, działających pod kontrolą różnych systemów operacyjnych.

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 rzeczą akcją może być włączenie lub wyłączenie urządzeń, przekazanie do urządzeń komend sterujących, odczytanie stanu urządzeń lub odczytanie wskazań przyrządów pomiarowych. W usługach internetowych akcją może być stworzenie na serwerze jakiegoś zasobu, modyfikacja zasobu czy usunięcie zasobu. Zasoby mogą być tworzone jako pliki w systemie plików serwera lub zapisy w bazach danych.

Odpowiedzią najczęściej jest dokument, który można wyświetlić w przeglądarce, ale może to być także plik binarny, graficzny czy dane zapisane w dowolnym formacie. Odpowiedź składa się podobnie jak żądanie z nagłówek i treści. Nagłówki są podobne do nagłówek 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. W tabeli 4.2 znajdują się 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)

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

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 żadanego 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

Tabela 4.2. Często stosowane kody statusu odpowiedzi wraz z opisem

Szczegóły odpowiedzi w przeglądarkach można zobaczyć wciskając F12 i otwierając narzędzia deweloperskie. Kody statusu odpowiedzi są dostępne w zakładce Network.

Lekkie serwery WWW realizowane w Node-RED

Prostą platformą do budowy lekkich serwerów WWW jest program Node-RED. Bloczki `http in` a `http response` potrzebne do stworzenia serwera przedstawiono w tabeli 4.3.

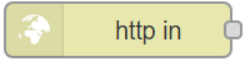
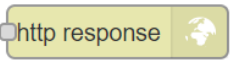
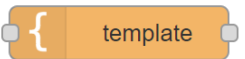
	Bloczek czeka na żądanie z sieci, po otrzymaniu żądania przesyła je dalej w obiekcie <code>msg</code> .
	Bloczek wysyła odpowiedź do klienta, dane klienta pobiera z obiektu <code>msg</code> utworzonego przez bloczek <code>http in</code> na podstawie odebranego żądania.
	Bloczek pozwala na ustawienie dowolnej właściwości obiektu <code>msg</code> , posiada edytor ułatwiający tworzenie kodu HTML, JSON, JavaScript, CSS, Python, SQL, ...

Tabela 4.3. Bloczki użyte do stworzenia lekkiego serwera WWW

Na rysunku 4.1 przedstawiono graf realizujący funkcję lekkiego serwera WWW.

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój



Rys. 4.1. Graf realizujący funkcję lekkiego serwera WWW

Listing 4.1. Kod grafu z rysunku 4.1

```
[{"id": "523bcf27.a504e", "type": "http
in", "z": "70559c28.b97c34", "name": "", "url": "/titi/plik/", "method": "get", "upload":
false, "swaggerDoc": "", "x": 110, "y": 60, "wires": [{"dbafc955.51cdc8"}]}, {"id": "dbafc
955.51cdc8", "type": "template", "z": "70559c28.b97c34", "name": "strona
WWW", "field": "payload", "fieldType": "msg", "format": "handlebars", "syntax": "mustach
e", "template": "<html>\n    <head></head>\n    <body>\n        <h1>Strona
testowa</h1>\n
</body>\n</html>", "x": 290, "y": 60, "wires": [{"f2b546f6.4f53a8", "96a794e8.c57128"}]
}, {"id": "f2b546f6.4f53a8", "type": "http
response", "z": "70559c28.b97c34", "name": "", "statusCode": "", "headers": {}, "x": 510, "
y": 60, "wires": []}, {"id": "96a794e8.c57128", "type": "debug", "z": "70559c28.b97c34", "
name": "", "active": true, "tosidebar": true, "console": false, "tostatus": false, "comple
te": "true", "targetType": "full", "statusVal": "", "statusType": "auto", "x": 450, "y": 10
0, "wires": []}]
```

Bloczek typu `http in` nasłuchuje żądań wysłanych metodą GET, których URL = `/titi/plik/`. Po otrzymaniu żądania bloczek wysyła obiekt `msg`, który posiada właściwości

- `payload` zawierającą otrzymaną treść,
- `req` reprezentującą parametry żądania,
- `res` reprezentującą parametry żądania.

Ścieżka grafu zaczynająca się od bloczka `http in` powinna kończyć się bloczkiem `http response`, który wysyła do klienta odpowiedź. Po drodze między `http in` a `http response` można odczytać nagłówki i treść żądania, wykonać zleconą akcję oraz przygotować treść odpowiedzi i dodać do odpowiedzi potrzebne nagłówki. W grafie z rysunku 4.1 między `http in` a `http response` jest tylko bloczek `strona WWW` zmieniający `msg.payload` na kod prostej strony HTML. Treść odpowiedzi wysyłanej przez bloczek `http response` to wartość `msg.payload` przychodzącą na wejście bloczka. Nagłówki można dodawać w oknie konfiguracji bloczka `http response` lub dodać do wiadomości wejściowej jako `msg.headers`. Poniżej kod strony HTML tworzonej przez bloczek `strona WWW` z rysunku 4.1.

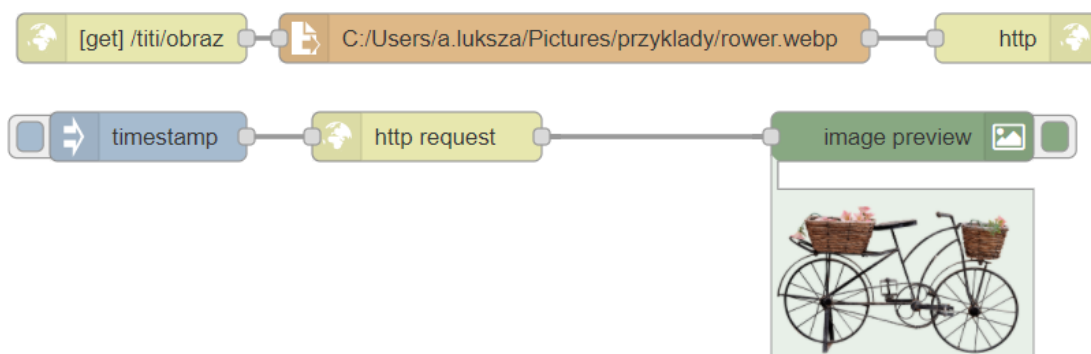
```
<html>
  <head></head>
  <body>
    <h1>Strona testowa</h1>
  </body>
</html>
```

Do serwera z rysunku 4.1 można wysłać żądanie z przeglądarki internetowej wpisując adres <http://host:1880/titi/plik/>, w odpowiedzi otrzyma się stronę stworzoną w bločku strona WWW.

Poniżej przedstawiona jest lista nagłówków żądania wysłanego z przeglądarki Chrome.

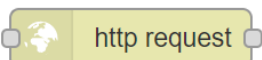
```
host: "127.0.0.1:1880"
connection: "keep-alive"
cache-control: "max-age=0"
upgrade-insecure-requests: "1"
user-agent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36"
accept: "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
sec-fetch-site: "none"
sec-fetch-mode: "navigate"
sec-fetch-user: "?1"
sec-fetch-dest: "document"
accept-encoding: "gzip, deflate, br"
accept-language: "pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7"
if-none-match: "W/"57-SA68Ni4B92hLN1bc0MTZWezLUa8""
```

W Node-RED żądania można wysyłać za pomocą bločku http request. Na rysunku 4.2 przedstawiono graf zawierający ścieżkę serwera i ścieżkę klienta wysyłającego żądanie do tego serwera. Odpowiedzą jest plik graficzny pobrany z dysku na serwerze. Aby plik był poprawnie wyświetlany w przeglądarkach do odpowiedzi należy dodać nagłówek Content-Type o wartości image/webp lub image/png, w zależności od formatu wysyłanego pliku graficznego



Rys. 4.2. Graf realizujący funkcję lekkiego serwera WWW i klienta

Bločky http request i file in użyte w grafie z rysunku 4.2 opisane są w tabeli 4.4.

	Bloček klienta HTTP. Po otrzymaniu wiadomości bloček wysyła żądanie HTTP a po otrzymaniu odpowiedzi przekazuje ją dalej. Adres URL wpisuje się w konfiguracji bločka, albo można go podać jako opcjonalną właściwość msg.url na wejściu. Bloček wysyła żądania
---	---

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

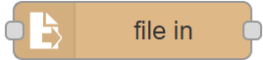
	metodami GET, PUT, POST, PATH lub DELETE.
	Bloczek odczytuje z dysku plik i wysyła go dalej jako . Adres pliku na dysku można wpisać na stałe jako parametr bloczka albo podać na wejście bloczka jako msg.filename.

Tabela 4.4. Opis bloczków http request i file in

Adresy przykładowych serwerów zrealizowanych na serwerze lukan.sytes.net:

- <http://lukan.sytes.net:1880/titi/plik> – serwer z rysunku 4.1,
- <http://lukan.sytes.net:1880/titi/obraz> – serwer z rysunku 4.2,
- <http://lukan.sytes.net:1880/sensors/bme280> – wskazania barometru i termometru w formacie JSON.

Serwer z rysunku 4.2 odsyła w odpowiedzi jeden wcześniej wybrany plik. Przy takim podejściu dla każdego pliku, który jest udostępniony na serwerze potrzebny jest jeden osobny punkt końcowy. Jest to rozwiązanie mało efektywne i mało przydatne, gdyby chcieć stworzyć serwer z większą ilością udostępnianych plików. W przypadku dużej liczby plików lepiej stworzyć jeden punkt końcowy, który będzie obsługiwał wiele adresów URL zawierających parametr, np. nazwę pliku. W tym celu można wykorzystać * w adresie URL bloczka. Jeśli parametr URL bloczka http in będzie zawierał na końcu * np. /titi/pliki/, to punkt końcowy będzie obsługiwał wszystkie żądania o adresie URL zaczynającym się od /titi/pliki/ resztę adresu przekazując jako parametr msg.req.params. Cały URL można odczytać jako msg.req.url. Użycie * zastępuje wiele poziomów adresu oddzielonych znakiem /. Jeden poziom adresu URL można zastąpić dwukropkiem „:” po którym następuje nazwa parametru. Parametr o podanej nazwie pojawi się w kolekcji msg.req.params. Punkt końcowy o adresie URL /path1/:param1/:param2/path4 będzie obsługiwał żądania o adresach czteropoziomowych zaczynających się od /path1 i kończących /path4 z dowolnymi łańcuchami tekstu wpisanymi na poziomach 2 i 3. Teksty wpisane na poziomach 2 i 3 będą dostępne w kolekcji msg.req.params jako parametry o nazwach param1 i param2.

Program Postman

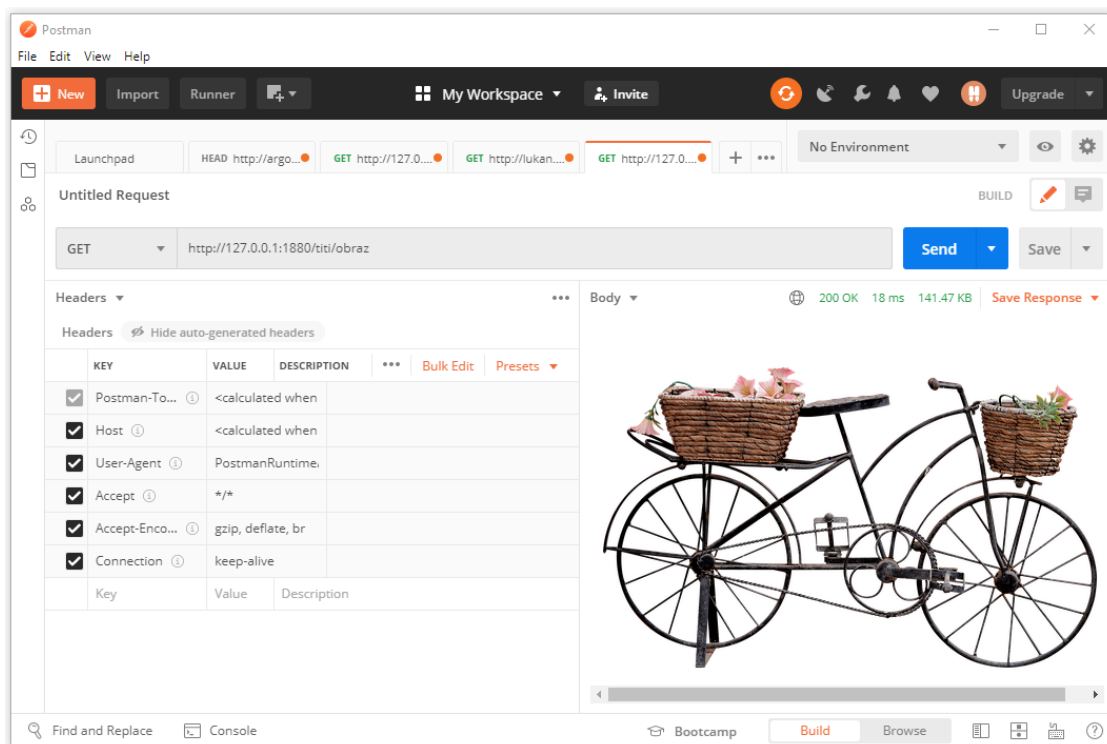
Wpisując URL do okna adresowego przeglądarki można wysłać żądanie metodą GET posiadające tylko nagłówki, ale nieposiadające treści. Bloczek http request wysyła żądania zawierające treść metodami GET, PUT, POST, PATH lub DELETE. Żądania można wysyłać z wielu programów napisanych w różnych językach programowania. Przed budową rozproszonego systemu dobrze jest przetestować komunikację internetową za pomocą przyjaznego programu ułatwiającego tworzenie żądań (zapytań). Jednym z takich programów jest program Postman.

Postman jest programem developerskim przeznaczonym głównie do testowania usług REST korzystających z protokołu HTTP. Program podobnie jak przeglądarka wysyła żądanie i wyświetla odpowiedź. W przeciwieństwie do przeglądarki, Postman w prosty sposób umożliwia kształtowanie żądania poprzez

- wybór dowolnej metody,
- dodawanie dowolnych nagłówków,
- umieszczenie dowolnej treści.

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

Odpowiedź można w wygodny sposób wyświetlić jako tekst, stronę HTML czy grafikę. Dodatkowo można wyświetlić otrzymane nagłówki. Na rysunku 4.4 przedstawiono okno programu Postman wysyłającego żądanie do punktu końcowego zrealizowanego na grafie Node-RED z rysunku 4.2.



Rys. 4.4. Okno programu Postman wysyłającego żądanie do grafu Node-RED z rysunku 4.2

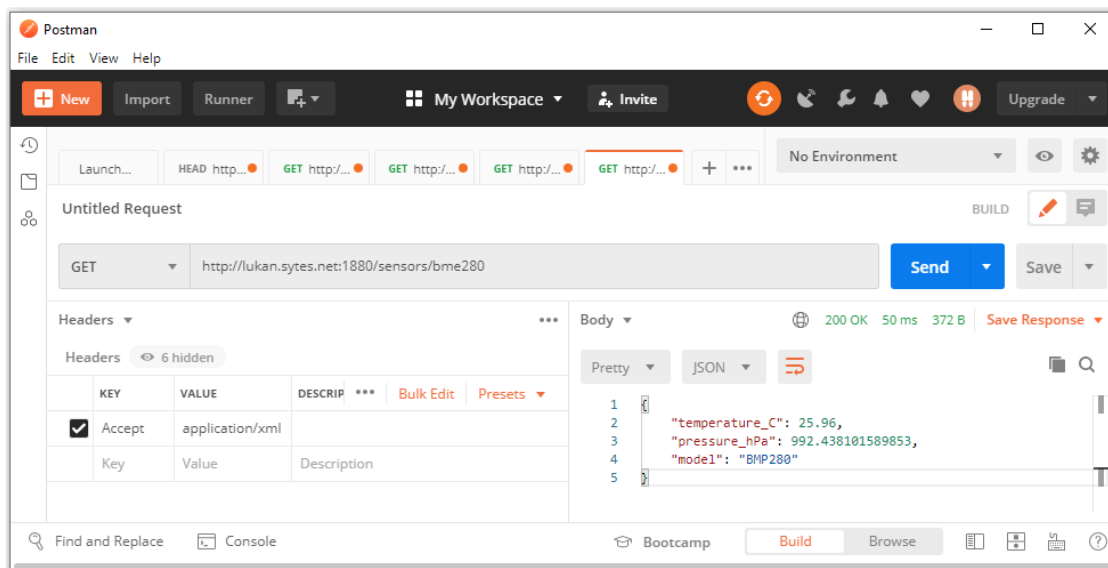
Po prawej stronie okna znajdują się automatycznie generowane nagłówki żądania, po lewej wyświetlona odpowiedź w formie graficznej. Na rysunku 4.5 przedstawiono nagłówki odpowiedzi.

Headers		200 OK 40 ms 141.47 KB Save Response
KEY	VALUE	
X-Powered-By	Express	
Access-Control-Allow-Origin	*	
content-type	image/png	
Content-Length	144636	
ETag	W/"234fc-hhpRCfCyTMvHt5HTOIz7SrD1ogk"	
Date	Wed, 19 Aug 2020 22:41:29 GMT	
Connection	keep-alive	

Rys. 4.5. Nagłówki odpowiedzi żądania z rysunku 4.4.

Na kolejnym rysunku 4.6 przedstawiono okno programu Postman po wysłaniu żądania do przykładowej usługi.

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój



Rys. 4.6. Okno programu Postman wysyłające żądanie do usługi odsyłającej dane w formacie JSON

Usługa odesłała dane w formacie JSON, które są wyświetlone po prawej stronie okna. Nagłówek żądania `Accept: application/xml` wskazuje, że klient chce uzyskać dane w formacie XML, ale ta konkretna usługa odsyła dane tylko w formacie JSON i nie czyta nagłówka `Accept`.

Po przetestowaniu usługi i żądania w programie Postman, można wysłać sprawdzone żądania z innych programów. Jedną z możliwości programu jest przygotowanie kodu wysyłającego przetestowane żądanie z różnych języków programowania, z różnych platform, z wykorzystaniem różnych bibliotek. Na listingu 4.2 przedstawiono kod w języku C# wysyłający żądanie przedstawione na rysunku 4.6 wygenerowany przez program Postman.

Listing 4.2. Kod w języku C# wysyłający żądanie wygenerowany przez program Postman

```
var client = new RestClient("http://lukan.sytes.net:1880/sensors/bme280");
client.Timeout = -1;
var request = new RestRequest(Method.GET);
request.AddHeader("Accept", "application/xml");
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Kod z listingu 4.3 przetestowano na platformie .NET Core i działa poprawnie. Na listingu 4.4 przedstawiono analogiczny kod w języku Python, który został przetestowany w środowisku Spyder.

Listing 4.4. Kod w języku Python wysyłający żądanie wygenerowany przez program Postman

```
import requests
url = "http://lukan.sytes.net:1880/sensors/bme280"
payload = {}
headers = {}
```

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

```
'Accept': 'application/xml'
}
response = requests.request("GET", url, headers=headers, data = payload)
print(response.text.encode('utf8'))
```

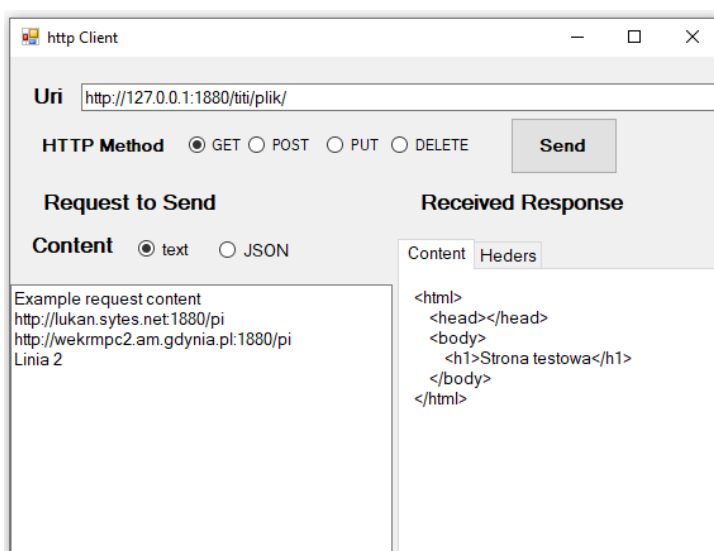
Podobne przykłady kodu są dostępne dla języków Java, JavaScript, PHP, Ruby, Objective-C i C

Proste programy klienta i serwera HTTP

Do testowania połączeń HTTP można wykorzystać programy stworzone na potrzeby laboratorium Technologii internetowych. Programy są napisane w języku C# i działają na platformie .NET. Program klienta HTTP Http Request Sender można zainstalować z adresu:

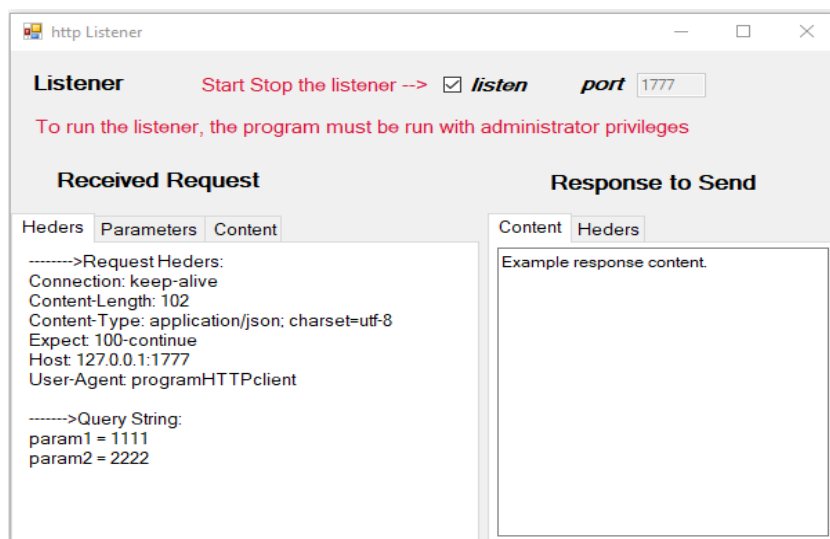
<http://argo.umg.edu.pl/www/Internet/SendHttpRequest/>.

Pełny kod programu jest dostępny jest na GitHubie pod adresem: <https://github.com/Andrzej1959/SendHttpRequest>. Na rysunku 4.7 przedstawiono okno programu z żądaniem wysłanym do serwera z rysunku 4.1.



Rys. 4.7. Okno programu Http Request Sender

Program lekkiego serwera Http Test Listener, który odbiera żądania, wyświetla ich treść, nagłówki i parametry oraz odsyła odpowiedź, można zainstalować z adresu: <http://argo.umg.edu.pl/www/Internet/Listener/publikacja.htm>. Pełny kod programu dostępny jest na GitHubie pod adresem: <https://github.com/Andrzej1959/TestHttpListener>. Na rysunku 4.8 przedstawiono okno programu.



Rys. 4.8. Okno programu Http Test Listener

W oknie programu widać nagłówki i parametry odebranego żądania wysłanego programem Http Request Sender na adres URL: <http://127.0.0.1:1777/?param1=1111¶m2=2222>.

Program powinien być uruchamiany z uprawnieniami administratora. Jeśli są problemy można wejść na adres <http://argo.umg.edu.pl/www/Internet/Listener/Application%20Files/>, pobrać odpowiedni plik wykonywalny exe i uruchomić go w trybie administratora. Można też pobrać kod programu z GitHuba i go skompilować w Visual Studio.

Ćwiczenie 4.1. Realizacja połączeń internetowych z wykorzystaniem protokołu HTTP

1. Zrealizować serwery przedstawione na rysunkach 4.1 i 4.2.
2. Przetestować działanie serwerów za pomocą programów Postman i Http Request Sender.
3. Uruchomić program Http Test Listener i wykonać połączenia do programu z programów Postman, Http Request Sender i zaprojektowanego samodzielnie grafu Node-RED.

Na listingu 4.5 przedstawiono żądanie wysłane przez program Postman na adres URL: <http://127.0.0.1:5742/A1/A2/A3?p1=w1&p2=w2>.

Listing 4.5. Treść żądania wysłanego przez program Postman

```
POST /A1/A2/A3?p1=w1&p2=w2 HTTP/1.1
Content-Type: text/plain
User-Agent: PostmanRuntime/7.26.3
Accept: */*
Postman-Token: 89456603-e58c-429e-80ea-dcb375d55cdb
```

Projekt „SezAM wiedzy, kompetencji i umiejętności” jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

Host: 127.0.0.1:5742
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 1

1

Odbiorcą żądania jest gniazdo TCP jak na rysunku 4.9.



Rys 4.9. Graf gniazda TCP

4. Zrealizować graf z rysunku 4.9.
5. Wysłać żądanie z programu Postman do punktu końcowego z grafu i porównać jego treść z listingiem 4.5.

Żądanie zawiera 8 nagłówek wygenerowanych automatycznie przez program Postman, treścią żądania jest cyfra 1, która w Internecie rzeczy może włączyć jakieś urządzenie. Nagłówki stanowią narzut na przesyłaną treść i jest ich wielokrotnie więcej niż samej treści.