

Lista 5

Zadanie 1.

W zadaniu trzeba przetestować oraz zastanowić się jak działa poniższy program serwera HTTP.

```
1  use HTTP::Daemon;
2  use HTTP::Status;
3  #use IO::File;
4
5  my $d = HTTP::Daemon->new(
6      LocalAddr => 'localhost',
7      LocalPort => 4322,
8  ) || die;
9
10 print "Please contact me at: <URL:", $d->url, ">\n";
11
12
13 while (my $c = $d->accept) {
14     while (my $r = $c->get_request) {
15         if ($r->method eq 'GET') {
16
17             $file_s = "./index.html";    # index.html - jakiś istniejący plik
18             $c->send_file_response($file_s);
19
20         }
21         else {
22             $c->send_error(RC_FORBIDDEN)
23         }
24     }
25     $c->close;
26     undef($c);
27 }
28
29
```

Omawiając powyższy program warto zacząć od parametru `LocalAddr` gdzie trzeba określić lokalny adres IP komputera. W przykładzie wyżej mam tam wartość `localhost` który oznacza adres IP komputera lokalnego (równie dobrze można tam podać IP: **127.0.0.1** bo właśnie taka wartość komputer będzie rozumiał, `localhost` to tylko zarezerwowana domena). Komunikacja z `127.0.0.1` oznacza wymianę informacji wewnątrz jednego hosta. Następnie mamy parametr `LocalPort` który określa na jakim porcie server będzie nasłuchiwał. Kolejna linijka drukuje na konsolę komunikat pod jakim adresem url server będzie dostępny. Poniżej znajduje się już tylko pętla która akceptuje zadanie GET i jeśli klient wysłał prawidłowe zadanie to server mu odpowie tym co znajduje się w pliku `index.html` oczywiście jeśli takowy istnieje, jeśli nie istnieje to zostanie wyświetlony błąd 404 który świadczy o odwołaniu się do nieistniejącej lokalizacji bądź w naszym przypadku pliku.

W tym zadaniu plik index.html nie zawiera nic szczególnego, jest to tylko text: "Hello world"

```
1 <title>Hello world!</title>
2 <h1>Hello world!</h1>
3
```

Aby przetestować powyższy program posłużyłem się programem curl który służy do przekazywania danych do lub z serwera za pomocą różnych protokołów.

```
$ curl -i -X GET http://127.0.0.1:4322/
HTTP/1.1 200 OK
Date: Sun, 18 Jun 2017 23:18:32 GMT
Server: libwww-perl-daemon/6.01
Content-Type: text/html
Content-Length: 50
Last-Modified: Sun, 18 Jun 2017 22:30:21 GMT

<title>Hello world!</title>
<h1>Hello world!</h1>
```

Program curl posiada wiele opcji i możliwości jedna z nich jest:

- i – Dołącza HTTP header czyli nagłówek do outputu (nagłówek to pierwsze 6 linijek outputu)

Pierwsza linijka to jest kod statusu odpowiedzi. W naszym przypadku otrzymaliśmy kod 200 OK co jest standardową odpowiedzią dla pomyślnego zadania HTTP. Druga linijka to czas w którym serwer odebrał zadanie (in "HTTP-date" format as defined by [RFC 7231 Date/Time Formats](#)). Trzecia linijka zawiera informacje na temat oprogramowania używanego przez serwer do obsługi zapytań. Kolejna linijka mówi nam o formacie pliku zadania. Content-Length mówi nam o długości zadania w bajtach. Ostatnia linijka mówi nam kiedy była ostatnia modyfikacja obiektu o który pytamy.

- X określa metodę zadania klienta w naszym przypadku ta metoda to była GET.

Po dodatkowych opcjach określamy url.

Oczywiście w outputcie dostaliśmy także to co tak na prawdę jest najważniejsze czyli zawartość naszego pliku index.html – czyli to co będzie interpretowała przeglądarka.

Zadanie 3.

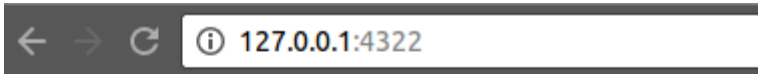
Nawiązanie połączenia pomiędzy przeglądarką internetową a naszym serverem jest raczej proste wystarczy uruchomić server:

```
$ perl server3.pl
```

Server odpowie nam na pod jaki url musimy użyć aby się z nim połączyć:

```
Please contact me at: <URL:http://127.0.0.1:4322/>
```

I gotowe



Hello world!

Połączyliśmy się z serverem i odpowiedział nam tym co znajdowało się w pliku index.html

Zadanie 4.

Aby wykonać to zadanie postanowiłem napisać własny server i wykorzystać do tego język programowania JavaScript pomagając sobie platformą node.js oraz frameworkiem express. Server prezentuje się następująco:

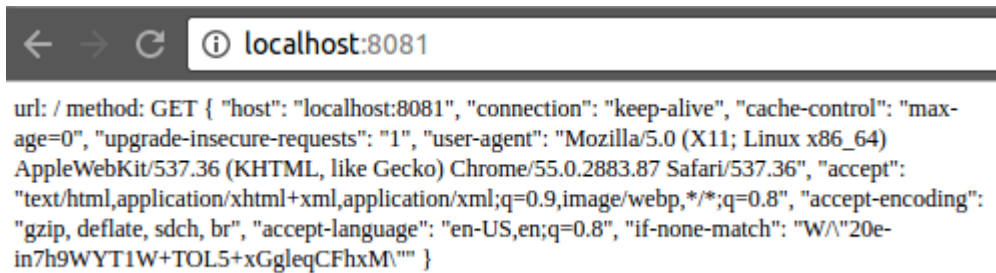
```
1 var express = require('express');
2 var app = express();
3
4 app.get('/*', function (req, res) {
5   res.send('url: ' + req.url + '\n' + 'method: ' + req.method + '\n' + JSON.stringify(req.headers, null, 4));
6 })
7
8 var server = app.listen(8081, function () {
9   var host = server.address().address
10  var port = server.address().port
11
12  console.log("Example app listening at http://localhost:%s", port)
13 })
14
```

Zaczynając od góry pierwsza linijka mówi nam że będziemy korzystać z frameworka express, w drugiej linijce deklarujemy go do zmiennej, co będzie nam pomagało w odbieraniu zadań od klientów. W linijce nr 4 widać że przechwytyjemy wszystkie adresy url oraz wysyłamy do klienta odpowiedź w której będzie się zawierało jego własne zadanie. Następnie tylko mówimy na jakim porcie ma działać server oraz określamy zmienne port oraz host potrzebne nam do wyświetlenia komunikatu pod jakim adresem url będzie dostępny nasz server.

Server odpalamy I widzimy ze odpowiada nam pod jakim url bedzie dostepny

```
$ node server.js  
Example app listening at http://localhost:8081
```

Po wpisaniu naszego url do przegladarki otrzymujemy nasze wlasne zadanie jakie przegadarka wyslala do naszego servera.



The screenshot shows a web browser's developer console with the address bar displaying 'localhost:8081'. Below the address bar, the raw HTTP request is visible, starting with 'url: / method: GET' followed by a JSON object containing various headers like 'host', 'connection', 'cache-control', 'max-age', 'upgrade-insecure-requests', 'user-agent', 'accept', and 'accept-encoding'.

```
url: / method: GET { "host": "localhost:8081", "connection": "keep-alive", "cache-control": "max-age=0", "upgrade-insecure-requests": "1", "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36", "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-encoding": "gzip, deflate, sdch, br", "accept-language": "en-US,en;q=0.8", "if-none-match": "W/\"20e-in7h9WYT1W+TOL5+xGgleqCFhxM\""} }
```

Zadanie 5.

Ponizej znajduje sie kod mojego servera, sklada sie on z 3 podstron czyli obsluguje 3 zadania get. Server rowniez potrafi obsluzyc zadanie post.

```
1  var express = require('express');
2  var app = express();
3  var bodyParser = require('body-parser');
4
5  var all_users = [];
6  app.use(bodyParser.urlencoded({
7    extended: true
8  }));
9  app.use( bodyParser.json() );
10
11 app.use(express.static('public'));
12
13 app.get('/', function (req, res) {
14   res.sendFile( __dirname + "/" + "index.html" );
15 })
16
17 app.get('/add_user', function (req, res) {
18   res.sendFile( __dirname + "/" + "add_user.html" );
19   response = {
20     first_name:req.query.first_name,
21     last_name:req.query.last_name
22   };
23   all_users.push(response);
24   console.log(JSON.stringify(all_users));
25 })
26
27 app.get('/user_list', function (req, res) {
28   res.end(JSON.stringify(all_users));
29 })
30
31 app.post('/add_user', function (req, res) {
32   // Prepare output in JSON format
33   response = {
34     first_name:req.body.first_name,
35     last_name:req.body.last_name
36   };
37   console.log("POST");
38   console.log(response);
39   all_users.push(response);
40   res.end(JSON.stringify(response));
41 })
42
43 var server = app.listen(8081, function () {
44   var host = server.address().address
45   var port = server.address().port
46   console.log("Example app listening at http://127.0.0.1:%s", port)
47 })
48 })
```

Testy zadania post przeprowadzałem za pomocą polecenia:

```
$ curl -X POST -d '{"first_name":"xyz","last_name":"zxy"}' -H 'content-type:application/json' "http://127.0.0.1:8081/add_user"
```

Wysyłając przykładowe dane w formacie JSON. Po wysłaniu oczywiście zmiany da się zauważyć na serwerze przez przeglądarkę lub w inny sposób.

Server obsługuje również dwa pliki HTML które znajdują się poniżej :

```
1  <html>
2    <body>
3      <form>
4        First Name: <input type = "text" name = "first_name"> <br>
5        Last Name: <input type = "text" name = "last_name">
6        <input type = "submit" value = "Add User">
7      </form>
8      <form action = "http://127.0.0.1:8081/user_list" method = "GET">
9        <input type = "submit" value = "Show User List">
10     </form>
11
12   </body>
13 </html>
14
```

```
1  <html>
2    <body>
3      <form>
4        First Name: <input type = "text" name = "first_name"> <br>
5        Last Name: <input type = "text" name = "last_name">
6        <input type = "submit" value = "Add User">
7      </form>
8      <form action = "http://127.0.0.1:8081/user_list" method = "GET">
9        <input type = "submit" value = "Show User List">
10     </form>
11
12   </body>
13 </html>
14
```


Zadanie 6.

Komunikaty z I do servera przechwytuje za pomoca programu wireshark oraz developer tools przegladarki chrome oraz zadania wysylam programem curl. Widac ze dane sa poprawnie odbierane I wysylane na server.

▼ General

Request URL: http://127.0.0.1:8081/add_user
127.0.0.1:8081/add_user Method: GET
Status Code: 200 OK
Remote Address: 127.0.0.1:8081

▼ Response Headers [view source](#)

Accept-Ranges: bytes
Cache-Control: public, max-age=0
Connection: keep-alive
Content-Length: 389
Content-Type: text/html; charset=UTF-8
Date: Mon, 19 Jun 2017 03:30:36 GMT
ETag: W/"185-15cbe60f6f7"
Last-Modified: Mon, 19 Jun 2017 03:24:04 GMT
X-Powered-By: Express

▼ Request Headers [view source](#)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Host: 127.0.0.1:8081
If-Modified-Since: Mon, 19 Jun 2017 02:11:57 GMT
If-None-Match: W/"1d8-15cbe1ef0dd"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36

1300	38.056252372	127.0.0.1	127.0.0.1	HTTP	244 POST /add_user HTTP/1.1 (application/json)
▶ Frame 1300: 244 bytes on wire (1952 bits), 244 bytes captured (1952 bits) on interface 0					
▶ Linux cooked capture					
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1					
▶ Transmission Control Protocol, Src Port: 47072 (47072), Dst Port: 8081 (8081), Seq: 1, Ack: 1, Len: 176					
▶ Hypertext Transfer Protocol					
▶ JavaScript Object Notation: application/json					
0000	00 00 03 04 00 06 00 00	00 00 00 00 15 b7 08 00	
0010	45 00 00 e4 53 4e 40 00	40 06 e8 c3 7f 00 00 01	E...SN@. @.....		
0020	7f 00 00 01 b7 e0 1f 91	f9 4a 39 47 58 7c 87 05J9GX ..		
0030	80 18 01 56 fe d8 00 00	01 01 08 0a 0a 6c 6a 0a	...V....l.j.		
0040	0a 6c 6a 0a 50 4f 53 54	20 2f 61 64 64 5f 75 73	.l.j.POST /add_us		
0050	65 72 20 48 54 54 50 2f	31 2e 31 0d 0a 48 6f 73	er HTTP/ 1.1..Hos		
0060	74 3a 20 31 32 37 2e 30	2e 30 2e 31 3a 38 30 38	t: 127.0 .0.1:808		
0070	31 0d 0a 55 73 65 72 2d	41 67 65 6e 74 3a 20 63	1..User- Agent: c		
0080	75 72 6c 2f 37 2e 34 37	2e 30 0d 0a 41 63 63 65	url/7.47 .0..Acce		
0090	70 74 3a 20 2a 2f 2a 0d	0a 63 6f 6e 74 65 6e 74	pt: /*. .content		
00a0	2d 74 79 70 65 3a 61 70	70 6c 69 63 61 74 69 6f	-type:ap plicatio		
00b0	6e 2f 6a 73 6f 6e 0d 0a	43 6f 6e 74 65 6e 74 2d	n/json.. Content-		
00c0	4c 65 6e 67 74 68 3a 20	33 38 0d 0a 0d 0a 7b 22	Length: 38....{"		
00d0	66 69 72 73 74 5f 6e 61	6d 65 22 3a 22 78 79 7a	first_na me";"xyz		
00e0	22 2c 22 6c 61 73 74 5f	6e 61 6d 65 22 3a 22 7a	", "last_ name": "z		
00f0	78 79 22 7d		xy"} }		