Serwer Web

Wprowadzenie

- Implementacja serwera w oparciu o:
  - Moduł http
  - Moduł expres
- Moduły i klasy pomocnicze

Moduł http

- Jest modułem niskopoziomowym
- Nie zapewnia wsparcia dla ciasteczek i routingu
- Szybki – do zastosowań dla usług zaplecza

Obsługa URL

- Moduł url składa się:
- Protokół://uzytkownik:haslo@host.com:port/ścieżka/zasób?zapytanie=lancuch#położenie_fragmentu
- Do parsowania stosuje się moduł url:
  - url.parse(łańURL,[analizowany łańcuch(bool)],[host(bool)]);
  - Alternatywnie można użyć funkcji format

Właściwości zwracane

- Href, Protocol, Host, Auth, Hostname, Port
- Pathname - ścieżka
- Search – zapytanie z ?
- Path – pełna ścieżka z zapytaniem
- Query – zwraca parametry zapytania i wartość (jeżeli true przy parsowaniu)
- Hash – ostatnia część z uwzględnieniem #

Modyfikacja URL

- Adres URL w nowym położeniu:
  - url.resolve(z, do)
- Przykład:

  z=http:/us:haslo@host.com/sciezka/zasob?zap=1

  do=/nowy/zasob?zar=2

  wynik:

http:/us:haslo@host.com/nowy/zasob?zar=2

Przetwarzanie parametrów

- Parametry zapytania są parami element = wartość
- Można parsowanie przeprowadzić na stringach lub użyć modułu querystring:
- Querystring.parse(łańcuch, [separator], [znak przypisania], [opcje=maxkeys=1000])
- Przykład:

Var qs = require(‘querystring’);

Var params = qs.parse(„name=Marcin&oczy=szare&oczy=niebieskie”);

- Params -> {name:’Marcin’,oczy:{‘szare’,’niebieskie’}”}

Obiekty żądań i odpowiedzi serwera

- Serwery HTTP budowane są na bazie obiektów:
  - ClientRequest
  - ServerResponse
  - IncomingMessage
  - oraz Server

http.ClientRequest

- Obiekt tworzony po wywołaniu metody http.request()
- Umożliwia inicjowanie, monitorowanie i obsługę odpowiedzi z serwera
- Implementuje strumień Writable: write()
- Implementuje się poprzez:

  http.request(opcje,wywołanie_zwrotne);

  opcje – obiekt definiujący sposób wysyłania żądań do serwera,

  w_z – wywoływane po pojawieniu się żądania do serwera i jako parametr otrzymuje obiekt IncomingMessage

Typy opcji

- Host – domena lub IP serwera do którego zostanie wysłane żądanie dom: localhost
- Hostname, port,
- localaddress- adres lokalny do wiązania połączeń sieciowych
- Method –post,get,connect,options
- Path – np.:/dane.html?par=1
- Headers: {content-length:’700’,’content-type’:’text/plain’}
- Auth : utoryzacja: user:haslo
- Agent: np. keep-alive , lub własny

Przykład

```
Var http = require('http');
Var options = {
    hostname: 'www.mojserwer.com',
    path: '/',
    port: '8080',
    method: 'POST' };
Var req = http.request(options,function(response){
Var str='';
Response.on('data',function (chunk) {
    str+=chunk;
});
Response.on('end',function(){
    console.log(str);
});
});
req.end();
```

Zdarzenia clientResponce

- Response – emitowane po odebraniu z serwera odpowiedzi na żądanie. W tym przypadku jedyny parametr IncomingMessage
- Socket – emitowane po przypisaniu gniazda do żądania
- Connect – emitowane każdorazowo gdy serwer odpowiada na żądanie
- Upgrade – emitowane gdy w nagłówku żądanie aktualizacji
- Continue – emitowane gdy serwer wyśle odpowiedź 100 continue

Odpowiedzi

- Ze względu na implementacje strumieni możliwa jest ingerencja w odpowiedź:
- Write(porcja,[kodowanie]) –zapisuje w żądaniu porcję danych
- End([dane],[kodowanie]) – zapisuje dodatkowe informacje a następnie opróżnia strumień i kończy żądanie
- Abort() – przerywa żądanie
- setTimeout(limit,[w_z]) – ustawia limit czasu gniazda

ServerResponse

- Serwer tworzy obiekt po otrzymaniu zdarzenia request
- Przekazywany jest do procedury obsługi jako drugi
- Implementuje strumień writable
- Jest wykorzystywany do budowania odpowiedzi
- Zdarzenia: close, headersSent, sendDate, statusCode

metody

- writeContinue() – żądanie danych treści
- writeHead(kod,[przyczyna],[nagłówki])
- setTimeout(liczba_milisekund,w_z)
- setHeader(nazwa, wartość)
- getHeader(nazwa)
- Write(porcja,[kodowanie]) – obiekty typy buffer lub string
- addTrailers(nagłówki) – dodaje nagłówki końcowe
- End ([dane],[kod]) – opróżnia strumień

IncomingMessage

- Tworzony przez serwer lub klienta HTTP
- Po stronie serwera żądanie jest przechowywane jako IM,
- po stronie klienta odpowiedź reprezentowana przez ten sam obiekt
- Implementuje strumień readable – mogą byś używane po obu stronach

właściwości

- Close – zamknięcie gniazda
- httpVersion – wersja http używana do budowy
- Headers – nagłówki
- Trailers – nagłówki końcowe
- Method, url, statusCode
- Socket – uchtyt gniazda do komunikacji z serwerem
- setTimeout() – limit czasu dla gniazda przy wowołaniu zwrotnym

Obiekt Serwer

- Obiekt oferuje podstawowy sposób implementacji serwera HTTP
- Oferuje gniazdo do nasłuchiwania żądań
- Wysyła odpowiedzi
- Implementuje obiekt EventEmiter obsługującego zdarzenia:
  - Request, connection(event: connect), close, checkContinue, upgrade, clientError

Wywołania funcji zwrotnych

- Request -> w_z(żądanie,odpowiedz)
- Connection-> w_z(gniazdo);
- checkContinue -> w_z(żądanie,odpowiedz)
- Connect -> w_z(żądanie,gniazdo,nagłówek)
- Upgrade - > w_z(żądanie,gniazdo,nagłówek)
- ClientError - > w_z(błąd, gniazdo)

Implementacja

- Tworzenie serwera:
  - http.createServer([proces nasłuchiwania])
- Rozpoczęcie nasłuchiwania:
  - Listen(port,[nazwa_hosta],[zaległe],[w_z]);
    - Nazwa_hosta- określa kiedy serwer będzie akceptował połączenia, domyślnie wszystkie
    - Zaległe – maksymalna liczba kolejkowanych połączeń
    - W_z – wywołanie zwrotne – reakcja zwiera 2 parametry IM oraz SR

Przykład

Var http = require(‘http’);

http.createServer(function(req,res){

//kod odpowiedzi i obsługi

}).listen(8080);

listen(ścieżka/uchwyt,[w_z]) – nasłuchiwanie dla systemu plików

Close([w_z]) – zakończenie nasłuchiwania

Implementacja statyczna: server

```
Var fs = require('fs');
Var http = require('http');
Var url = require('url');
Var root_dir = „html/";
http.createServer(function(req,res){
    var urlOBJ = url.parse(req.url,true,false);
    fs.readFile(root_dir+urlOBJ.pathname,function(err,data){
            if(err){
                    res.writeHead(404);
                    res.end(JSON.stringify(err));
                    return;
            }
            res.writeHead(200);
            res.end(data);
    });
}).listen(8080);
```

Implementacja statyczna: klient

```
Var http = require('http');
Var options = {
    hostname: 'localhost',
    path: '/hello.html',
    port: '8080'
    };
Function handleResponse(response){
Var str='';
response.on('data',function (chunk) {
    str+=chunk;
});
response.on('end',function(){
    console.log(str);
});
};

Var req = http.request(options,function(response){
handleResponse(response);
}).end();
```

Implementacja dynamicznych

- Treść jest budowana w odpowiedzi:

```
Res.setHeader(„Content-Type”,”text-html”);
Res.writeHead(200);
Res.write(‘<html><head><title>Prosty
   serwer</title></head>’);
Res.write(‘body’);
//logika dokumentu
Res.end(‘</body></html>);
```

Klient

- W przypadku klienta procedura żądania nie zmienia się,
- W tym przypadku nie jest konieczne określanie ścieżki
- Możliwe jest odczytanie dodatkowych danych:
  - Response.statusCode
  - Response.headers

Żądania POST

- Mogą mięć format HTTP, JSON lub innych danych
- Po stronie serwera wczytujemy zawartość/ transformacja/ zwrócona wartość
- Po stronie klienta generowanie zapytania wysłanie i oczekianie na odpowiedź

Przykład serwer

```
Var http = require('http');
http.createServer(function(req,res){
    var jsData = '';
    req.on('data',function(chunk){
            jsData+=chunk;
    });
    req.on('end',function(){
    var ob=JSON.parse(jsData);
    var ob={
    message:  „Witaj:"+ob.name;
    }
    ees.writeHead(200);
    res.end(JSON.stringify(ob));
    });
}).listen(8080);
```

Po stronie klienta

```
Var http = require('http');
Var options = {
      hostname: 'localhost',
      path: '/',
      port: '8080'
      method: 'POST';
      };
Function handleResponse(response){
Var str='';
response.on('data',function (chunk) {
      str+=chunk;
});
response.on('end',function(){
      jsData= JSON.parse(str);
      console.log(jsData.message);
});
};

var req = http.request(options, handleResponse);
req.write('{„name":"jestem alk"}');
req.end();
```

Interakcja z zewnętrznymi źródłami

- HTTP wykorzystywany jest do pobierania informacji z zewnętrznych źródeł
- W odpowiedzi znajduje się funkcja obsługi żądania z serwera zewnętrznego:

```
Req.on('end',function(){
Var postParams = qstring.parse(reqData);
getWeather(postParams.city,res);
});
```

Funkcja obsługi

```
Function parseWeather(wetherResponse,res){
Var weatherData=‘’;
weatherResponse.on(‘data’,function(chunk){
weatherData+=chunk;
});
weatherResponse.on(‘data’,function(){
sendResponse(weatherData,res);
}
}

function getWeather(city,res){
Var options = {
host: ‘api.openweathermap.org’,
path: ‘data/2.5/weather?q=‘ + city’
};
http.request(options,function(weatherResponse){
parseWeather(weatherResponse, res);
}).end();
}
```

HTTPS

- Do wygenerowania certyfikatu wykorzystywany jest moduł **openssl**

Openssl genrsa –out server.pem 2048

Openssl req –new –key server.pem –out server.csr

Generowanie samopodpisanego certyfikatu:

Openssl x509 –req –days 365 –in server.csr –signkey server.pem –out server.crt

Wszystkie opcje

- Pfx – certyfikat w tym formacie
- Key – klucz prywatny na potrzeby SSL
- Passphrase – hasło lub klucz prywatnego
- Cert – certyfikat publiczny
- Ca – tablica zaufanych certyfikatów
- Ciphers – lista szyfrów dozwolonych lub zabronionych
- Crl – certyfikaty dla serwera
- secureProtocol – wymuszenie protokołu SSL np. SSLv3_method

Przykład klient

```
Var options = {
Hostname: ‘secure.strona.com’,
Port: 443,
Path: ‘/’,
Method: ‘GET’,
Key: fs.readFileSync(‘test/keys/client.pem’),
Cert: fs.readFileSync(‘test/keys/client.crt’),
Agent: false
};
Var req = https.request(options, function(res){
…
}
```

Przykład serwer

```
https = require('https');
var options = {
Key: fs.readFileSync('test/keys/server.pem'),
Cert: fs.readFileSync('test/keys/server.crt'),
};
https.createServer(options,function(req,res){
//kod odpowiedzi i obsługi
}).listen(443);
```

Gniazda

- Wykorzystuje moduł net
- Obiekty socket i server umożliwiający nawiązywanie i oczekiwanie na połączenie
- Implementacja na zdarzeniach i strumieniach jak w przypadku http

Net.connect(opcje,[proces_nasłuchwania]);

Opcje: port, host, localAddress, allowHalfOpen

Operacje

- Zdarzenia: Connect, data, end, timeout, drain (empty buffer), error, close
- Metody: write, end, destroy, pause, resume, setTimeout, setNoDelay(Nalge'a buforowanie), setKeepAlive, address, ref, unref
- Właściwości obiektów: bufferSize, remoteAdress, remotePort, localAdress, loclPort, bytesRead, bytesWritten

Przykład

```
Var net = require('net');
Var client = net.connect({port:8107,host:'localhost'}, function(){
Console.log("nawiązanie poł. z klientem");
Client.write("dane");
});
Client.on('data',function(data){
    Console.log(data.toString());
    Client.end();
});
Client.on('end',function(data){
Console.log("klient został rozłączony");
});
```

Socket serwer

- Tworzony:

Net.createServer([opcje],[poces nasł]);

Opcje: allowHalfOpen

Zdarzenia: listening, connection, close, error

Metody: listening(port, [host],[w_z]), listen(uchwyt,[w_z]), getConnections, close, address, ref, unref

Przykład

```
Var net = require('net');
Var server = net.createServer(function(client){
    console.log('Nawiązano połączenie');
    client.on('data', function('data'){
            Console.log('Klient wysłał:'+data.toString());
    });
    Client.on('end',function(){
            Console.log('Klient rozłączony');
    });
Client.write('Witaj');
});
Server.listen(8107, function(){
    console.log('Serwer nasłuchuje na połączenie');
});
```

Express

- Jest modułem, który upraszcza wykorzystanie modułu http oraz rozszerza jego możliwości
  - Upraszcza obsługę tras, odpowiedzi, informacji cookie oraz statusu żądań
- Instalacja: npm install **express@4.0.0**
- Możliwe jest dodanie do package.json
- Przykład:

```
var express = require('express');
var app = express();
```

Konfiguracja komponentu

- Env – definiuje środowisko: development, testing i production
- Trust proxy – obsługa proxy
- Json callback name – nazwa wywołania zwrotnego: ?callback=
- Json replacer – definicje funkcję wywołania zwrotnego (null)
- Json spaces – liczba spacji w podczas formatowania odpowiedzi
- Case sensitive routing – rozróżnianianie /home a /Home (disabled)
- Strict routing – rozróżnia /home oraz /home/
- View cache – włącz/wyłącza buforowanie widoków szablonu
- View engine – określa domyślne rozszeżenie
- Views – określa ścieżkę dla mechanizmu szablonów

- Ustawiane poprzez funkcję set(ustawienia,wartość) lub enable(ustawienie), disable(ustawienie)

Uruchamianie serwera-przykład

```
Var express = require('express');
Var http = require('http');
Var https = require(https');
Var fs = require('fs');
Var app = express();
Var options = {
    host:'127.0.0.1',
    key: fs.readFileSync('ssl/server.key'),
    cert: fs.readFileSync('ssl/server.crt')
};
http.createServer(app).listen(80);
https.createServer(options,app).listen(443);
app.get('/', function(req,res){
    res.send('Witaj w komponencie Express');
});
```

Konfigurowanie tras

- Realizowany dwustopniowo:
  - Definicja typu żądania: GET/ POST
  - Ścieżka: / , /login, /cart, itd.
- apt.get(ścieżka, [funkcja pośrednia], w_z);
- apt.post(ścieżka, [funkcja pośrednia], w_z);
  - Funkcja pośrednia realizowana przez w_z
- Funkcja ogólna all i *, np.:

```
app.all('/user/*' function(req,res){
  //globalna procedura
});
```

Parametry w trasach

- Możliwe są cztery sposoby implementacji parametrów:
  - Łańcuchy zapytania - ?klucz=wartość
  - Parametry POST - w treści żądania
  - Wyrażenie regularne – może być zdefiniowane jako część trasy
  - Zdefiniowane parametry – łańcuch :<nazwa param> w części ścieżki trasy.

Przykłady

```
var url_parts = url.parse(req.url, true);
var query = url_parts.query;
res.send('info:'+query.info);

Apt.get(/^/info\/(\w+):(\w+)?$/, function(....
Dla /info/10:30 -> req.params[0]=10, [1]=30

App.get('/user/:userid', function(....)
Dla /user/1234 -> req.param(„userid")=1234
```

Funkcja wywołania zwrotnego

- Rejestrowanie funkcji:
  - App.param(param, function(req,res,next, value){});
  - Next wywoływane są dla kolejnych wywołań app.param()
- Przykład:

```
App.param('userid', function(req,res,next,value){
  console.log(„parametr:"+value);
});
```

Request

- Obiekt zwracany w jako pierwszy parametr
- Zawiera podstawowe informacje o żądaniu:
  - orginalUrl, protocol, ip, path, host, method, query
  - Nowe: fresh (last-modified), stale (last-modified), secure, acceptsCharset, get (zwraca nagłówek), headers (nagłówek jako obiekt)

Przykład:

req.get('connection')

JSON.stringify(req.headers,null,2);

Ustawianie odpowiedzi

- Content-type

Res.set('Content-Type','text/plain');

- Status:

Res.status(200); //ok.

Res.status(300); //redirect

Res.ststus(40x); //brak dostępu

Res.status(500); // błąd serwera

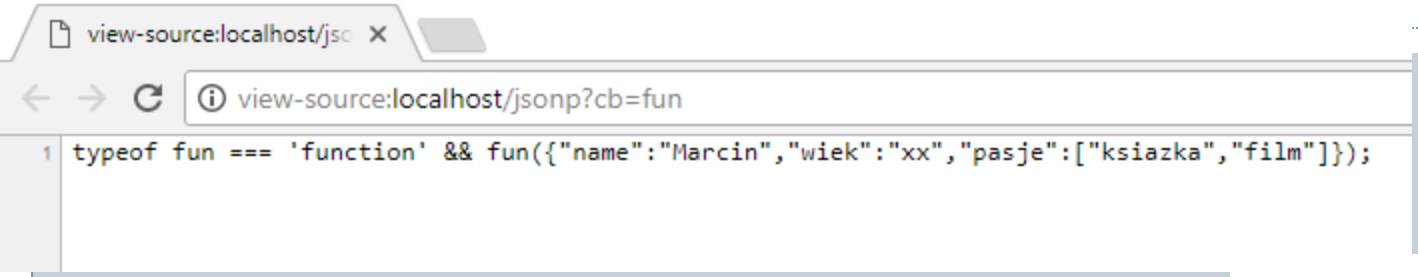
- Wysłanie odpowIedzi

Res.send(new Buffer('<html><body>info</body></html>'));

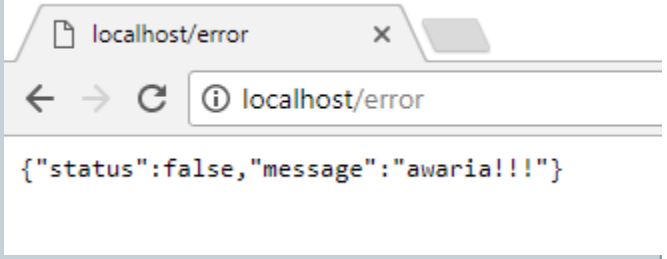
Zwracanie odpowiedzi JSON

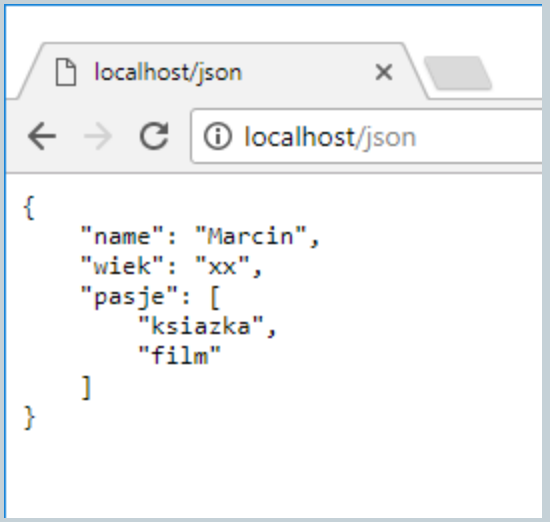
- Dostępne dwie funkcje: json oraz jsonp z parametrami:
  - (status,[obiekt]) lub (obiekt)
- W przypadku jsonp adres URL zawiera parametr ?wywołanie zwrotne=<metoda>. Łańcuch jest opakowany w funkcję z nazwą metody, która może zostać wywołana z poziomu klienta

Przvkład




```
var express = require('express');
var url = require('url');
var app = express();
app.listen(80);
app.get('/json',function(req,res){
		app.set('json spaces',4);
		res.json({name:"Marcin",wiek:"xx",pasje:['ksiazka','film']});
});
app.get('/error',function(req,res){
		res.json(500,{status:false, message:"awaria!!!"});
});
app.get('/jsonp',function(req,res){
		app.set('jsonp callback name','cb');
		res.jsonp({name:"Marcin",wiek:"xx",pasje:['ksiazka','film']});
});
```

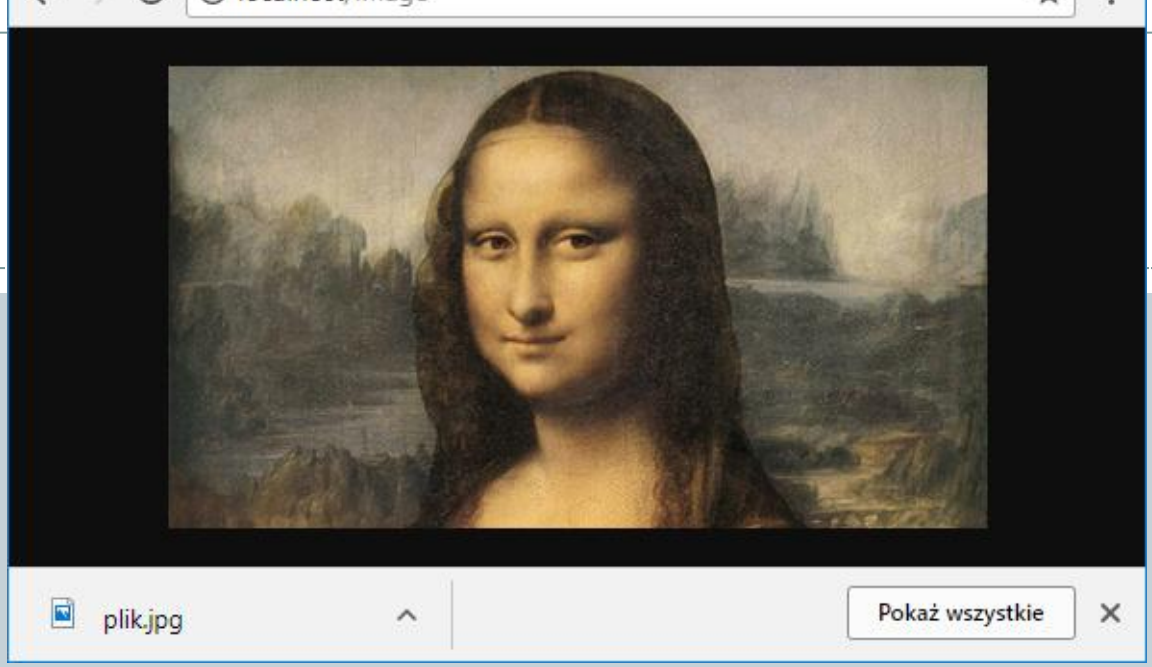




Wysyłanie plików

- Do wysyłania plików wykorzystywana jest metoda:
  - Sendfile(sciezka do pliku);
- Metoda ta dodatkowo:
  - Ustawia nagłówek Content-Type na podstawie rozszeżenia pliku
  - Ustawia pozostałe nagłówki n.p. content-length
  - Ustawia status odpowiedzi
  - Wysyła zawartość pliku do klienta w obrębie obiektu response
- Może zawierać opcje jak np. długość życia pliku (maxAge)
- Podobnie działa: res.download(ścieżka, [nazwa_pliku],[w_z]);

Przykład

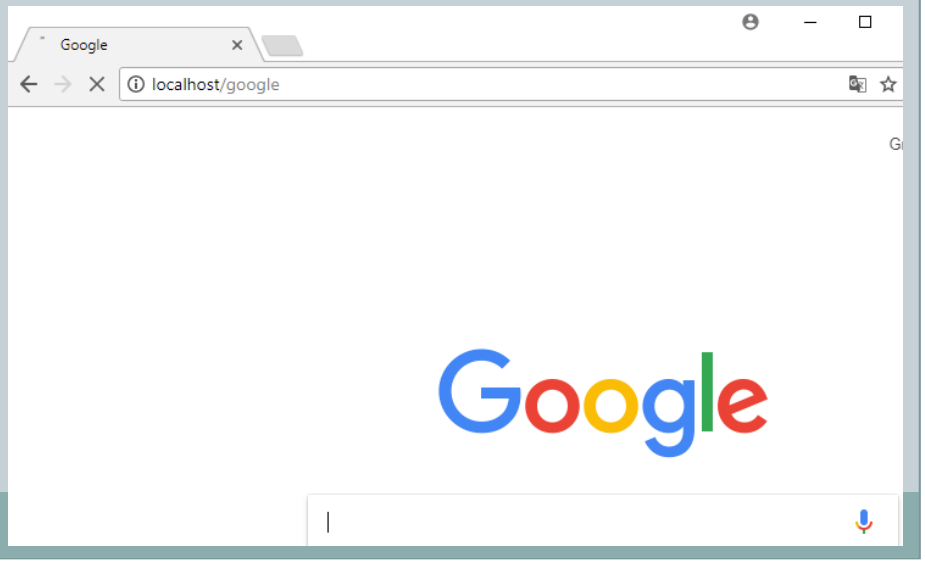



```
var express = require('express');
var url = require('url');
var app = express();
app.listen(80);
app.get('/image',function(req,res){
        res.sendfile('plik.jpg',{maxAge:
1},function(err){
        if(err) console.log('nie udalo sie');
        else console.log('ok');
        });
});
```

Przekierowanie odpowiedzi

- Do przekierowywania służy: res.redirect(ścieżka);

```
var express = require('express');
var url = require('url');
var app = express();
app.listen(80);
app.get('/google',function(req,res){
        res.redirect('http://google.com');
});
```



Implementowanie szablonów

- Zamiast budować strony uzupełnia się szablon o informacje
  - Prostota
  - Szybkość
- Przedstawiono dwa mechanizmy: Jade i EJS (Embeded JS)
- Do obsługi modułów szablonów zastosowano:
  - Npm install jade@1.3.1
  - Npm install ejs@1.0.0

Definiowanie mechanizmów

- Zdefiniowanie domyślnego mechanizmu obsługi oraz katalogu z szablonami
  - App.set('views','./views');
  - App.set('view engine','jade');
- Powiązanie rozszeżeń plików z silnikiem:
  - App.engine('jade',require('jade').__express)
  - App.engine('ejs',require('ejs').__express)

UWAGA. Funkcje domyślne działają tylko dla domyślnych rozszerzeń:

App.engine('html',require('ejs').renderFile)

Obiekt locals

- Dane do szablonu mogą być generowane poprzez obiekt locals .
- App.locals.title =‘moja app’;
- Lub przez funkcję: app.locals={title:’moja app’};

Tworzenie szablonów

- Zalety:
  - Możliwość ponownego wykorzystania
  - Wielkość – można dzielić szablony wg elementów
  - Hierarchia – szablony mogą być przyporządkowane do sekcji
- Wada
  - Wymagają poznania składni
  - Nie wszystkie, poznamy później

Szablony

user_ejs.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Szablon mechanizmu EJS</title>
</head>
<body>
    <h1> Uzytkownik szablonu EJS</h1>
    <ul>
        <li>Nazwa: <%= unazwa %></li>
        <li>Kolor: <%= ukolor %></li>
        <li>marka: <%= umarka %></li>
    </ul>
</body>
</html>
```

user_ejs.html | main_jade.jade

```
doctype html
html(lang="pl")
    head
        title="Szablon JADE"
    body
        block content
```

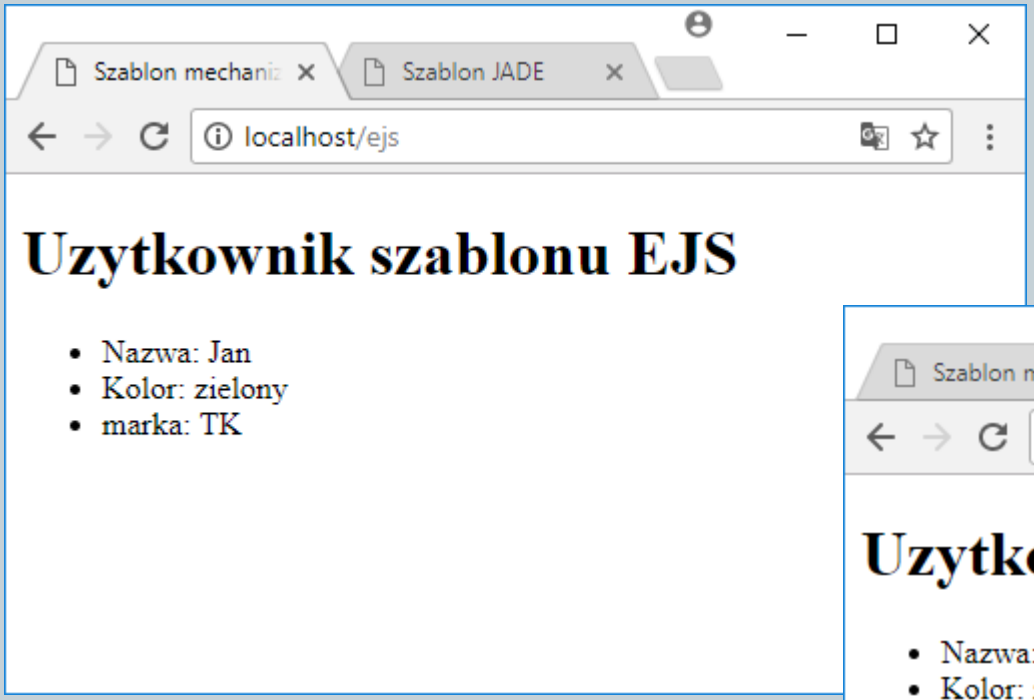
user_ejs.html | main_jade.jade | user_jade.jade

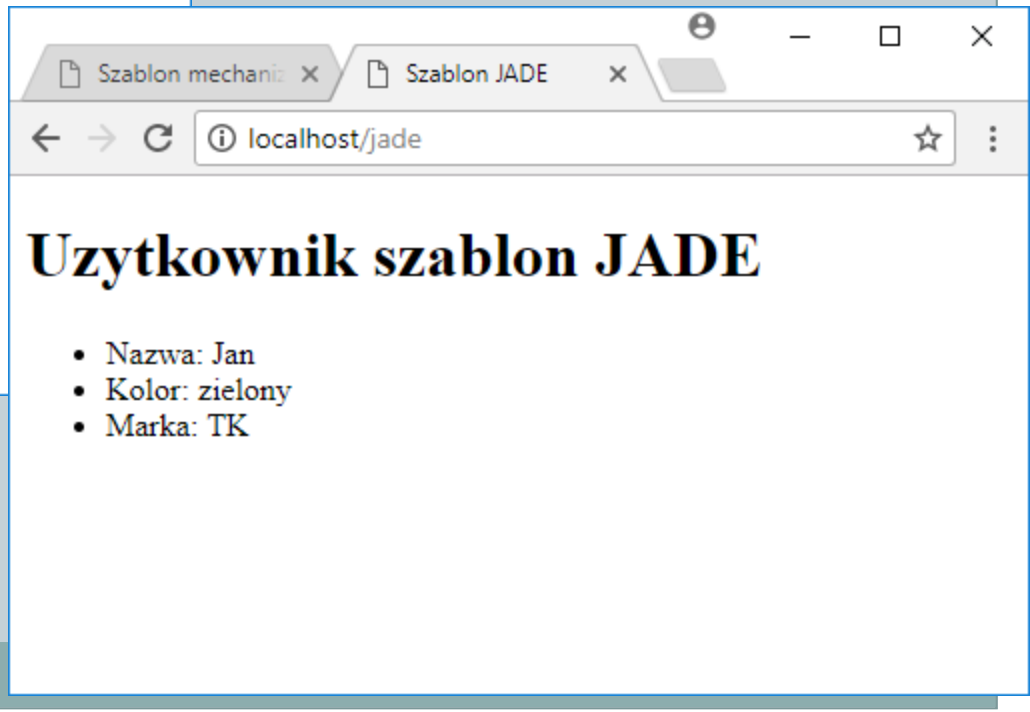
```
extends main_jade
block content
    h1 Uzytkownik szablon JADE
    ul
        li Nazwa: #{unazwa}
        li Kolor: #{ukolor}
        li Marka: #{umarka}
```

Renderowanie szablonów

```
var express = require('express');
        jade = require('jade');
        ejs = require('ejs');
var app = express();
app.set('views','./views');
app.set('view engine','jade');
app.engine('jade',jade.__express);
app.engine('html',ejs.renderFile);
app.listen(80);
app.locals={unazwa: 'Jan',ukolor: 'zielony',umarka: 'TK'};
app.get('/jade',function(req,res){
        res.render('user_jade');      ///!!!!jako res
});
app.get('/ejs', function(req, res){
        app.render('user_ejs.html',function(err,renderedData){ //!!!jako app
                res.send(renderedData);
        });
});
```

Szablony przykład





Funkcje pośrednie

- Dodawane poprzez:
  - app.use([ścieżka],fun_poś)
- Funkcja pośrednia jest wywoływana jako element łańcucha i powinna wywołać kolejna funkcję next();
- Miejsce umieszczania
  - App.get(/parsedRoute, bodyParser(),function(.....
- Mechanizm stosowany dla: plików statycznych, ciasteczek, sesji,...

Przykład

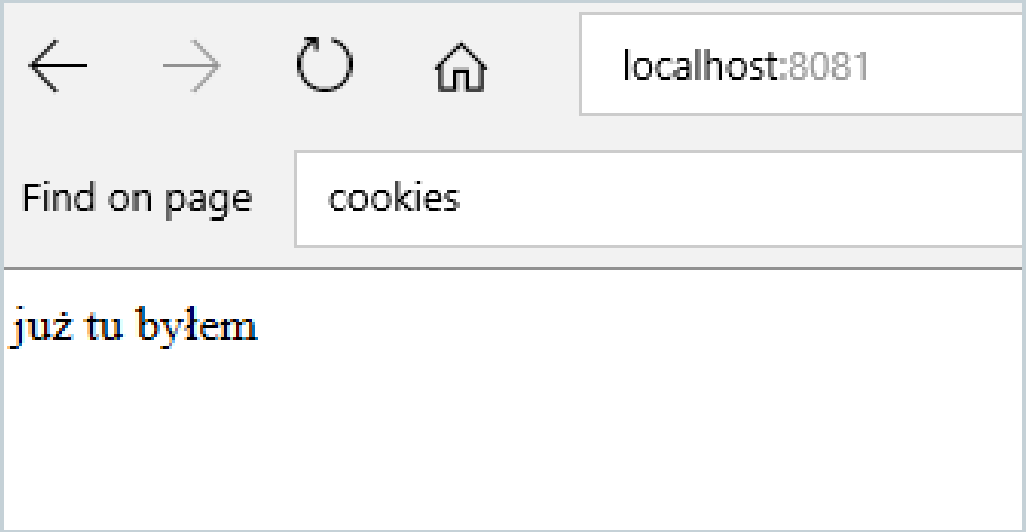
- $ npm install cookie-parser

Welcome | cookies.js

```
var express      = require('express')
var cookieParser = require('cookie-parser')

var app = express()
app.use(cookieParser())

app.get('/', function(req, res) {
  console.log("Cookies: ", req.cookies);
  info ="już tu byłem";
  if (!req.cookies.hasVisited){
    res.cookie('hasVisited','1',{maxAge: 60*60*1000,httpOnly: true, path:'/'});
    info = "mój pierwszy raz";
  }
      res.send(info);
})

app.listen(8081);
```

- Wynik

```
C:\proj_js>node cookies.js
Cookies:  { hasVisited: '1' }
Cookies:  { hasVisited: '1' }
```



Użycie midleware (POST)

```
var bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser

app.post('/process_post', urlencodedParser, function (req, res) {

   // Prepare output in JSON format
   response = {
       first_name:req.body.first_name,
       last_name:req.body.last_name
   };
   console.log(response);
   res.end(JSON.stringify(response));
})

var urlencodedParser = bodyParser.urlencoded({ extended: false })
```