

# Node.js: modul Express

17.03.2022

Marek Nagy

# modul Express

- mini webový framework
- poskytovanie statických súborov z disku
- pravidlá mapovania URL cesty

```
$ npm install express
```

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

Vytvorí vždy novú  
callback funkciu.

- **app** je vlastne callback funkcia, ktorá bude obsluhovať http požiadavky

# Prepojenie s HTTP serverom

```
const http = require ('http');  
const app = require ('express') ();  
  
// const server = http.createServer( (req, res) => {  
//   // Obsluha http servera  
// });  
  
const server = http.createServer (app);  
  
server.listen (9000, function () {  
  console.log ('Listen on port 9000');  
});
```

Úsporný  
zápis 2 v 1.

Spomínate si. V **req**  
sú informácie o HTTP  
požiadavke a do **res**  
sa konštruuje  
odpoveď, ktorá sa  
odošle klientovi.

```
// Alebo je aj zjednodušené:  
const app = require ('express') ();  
  
app.listen (9000)
```



Spracovanie požiadavky

# Obsluha podľa typu a URL cesty

```
app.METHOD ( PATH, HANDLER [,HANDLER ...] );
```

- **METHOD**

- get, post, all, ...
- all zahŕňa všetky možnosti

- **PATH**

- cesta z URL adresy

- **HANDLER**

- callback funkcia, ktorá sa zavolá
- podobne ako pri klasickom HTTP serveri bude mať argumenty: požiadavka **req** a odpoveď **res**

Nech Vás to nezmätie. **app** je funkcia, ale v podstate je to funkciový objekt, ktorý môže mať tiež svoje metódy. Pomocou týchto metód je možné nastaviť správanie sa funkcie **app**.

Princíp **app** je jednoduchý, podľa príslej URL sa rozhodne, ktorú svoju callback funkciu zavolať.

# Základné mapovanie URL cesty

```
app.get('/', (req, res) => {  
  res.send('Ahoj!');  
})  
  
app.post('/user', (req, res) => {  
  res.send('Ahoj user!');  
})  
  
app.all('/test', (req, res) => {  
  res.send('Ahoj user!');  
})
```

URL cesta musí  
presne zodpovedať  
danému reťazcu.

- odpoved' **res** sa musí vždy vygenerovať!
  - princíp HTTP protokolu: požiadavka → odpoved'

# Použitie regulárnych výrazov na URL cestu

- zjednodušený regex:
  - opakovania \* + ? a zoskupenie ()

```
app.get ('/ab(cd)+x', (req, res) => {  
  res.send ('Ahoj user!');  
});
```

Zjednodušený regulárny výraz. Zadáva sa priamo do reťazca.

```
app.get (/.*tucniak$/, (req, res) => {  
  res.send ('Ahoj tucniak!');  
});
```

Pravý regulárny výraz. Zadáva sa v lomkách. T.j. `RegExp` objekt.





# Parametre z URL cesty

- názvy parametrov začínajú dvojbodkou
  - dvojbodka nemôže byť v URL ceste...
- objekt **req.params**
  - namapované hodnoty týchto "premenných"

```
app.get ('/users/:userId/books/:bookId-:bookNum', (req, res) => {  
    res.send (req.params);  
});
```

get URL: **http://localhost:9000/users/345/books/a22s-223**

result: {"userId": "345", "bookId": "a22s", "bookNum": "223"}

# Parametre z URL query reťazca

- pri metóde **GET** sa query reťazec vyparsuje do objektu **req.query**

```
app.get ('/cmd_get', (req, res) => {  
  res.send (req.query);  
});
```

get URL: **http://localhost:9000/cmd\_get?id=3&name=Jony**

result: {"id": "3", "name": "Jony"}



Zoznam get parametrov v URL linke.



Vytvorenie odpovede

# Odpoved' send

```
res.send (data)
```

- odosiela dáta klientovi
- reťazec, bufer, objekt, ... rozhodne sa podľa typu
- nastaví správne hlavičku odozvy

```
app.get ('/', (req, res) => {  
    res.send ('Ahoj!');  
});
```

# Odpoved' json

```
res.json (data)
```

- objekt prevedie na JSON reťazec a odošle
- nastaví aj správnu hlavičku odozvy

```
app.get ('/', (req, res) => {  
    res.json ({a:1,b:2,c:[1, 2, 3]});  
});
```

# Odpoved' **redirect**

```
res.redirect (url)
```

- presmeruje klienta na novú URL adresu
- môže byť aj relatívne vzhľadom na aktuálny server

```
app.get ('/', (req, res) => {  
    res.redirect ('/novaVerzia');  
});  
  
app.get ('/www', (req, res) => {  
    res.redirect ('https://www.uniba.sk');  
});
```

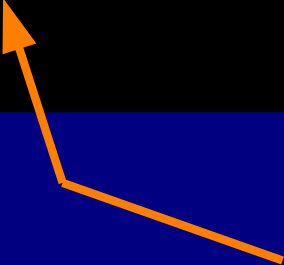


# Odpoved' **setStatus**

```
res.setStatus (code)
```

- odošle status kód odpovede aj s popisom chyby
- podľa HTTP špecifikácie

```
app.get ('/', (req, res) => {  
    res.setStatus (500);  
    // ekvivalent:  
    // res.status (500).send('Internal Server Error');  
});
```



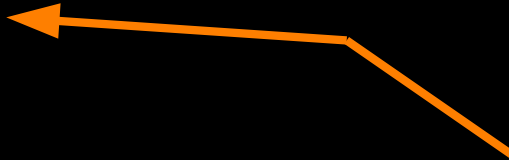
Nastaví iba kód odpovede.  
Odpoved' (telo) treba ešte  
dokončiť.

# Odpoved' **sendFile**

```
res.sendFile (filename [,options] [,callback])
```

- odošle obsah súboru
- nastaví aj typ odpovede podľa typu súboru
- ak sa nepodarí načítať súbor, odpoved' treba ukončiť

```
app.get ('/file/:name', (req, res) => {  
  let fn = req.params.name;  
  
  res.sendFile (fn, {root:'cesta/ku/suborom'}, err => {  
    if (err) res.sendStatus (404);  
    else console.log ('OK')  
  });  
});
```



Ak súbor na disku  
nenájde, treba  
poslať inú odpoveď.

# Odpoved' download

```
res.download (filename [,name] [,options] [,callback])
```

- odošle súbor **filename** s vynútením uloženia
- u klienta sa uloží pod názvom **name**

```
app.get ('/file2/:name', (req, res) => {  
  let fn = req.params.name;  
  
  res.download ('www/'+fn, 'tucniak.jpg', (err) => {  
    if (err) res.sendStatus (404);  
    else console.log ('OK')  
  });  
});
```

# Ukončenie odpovede

```
res.end ()
```

- ukončí HTTP odpoveď
  - už by sa nemalo nič posielat' do res
- táto metóda prislúcha modulu **http**
- všetky **spomínané metódy** automaticky ukončia odpoveď a tak ju netreba volať

```
app.get ('/', (req, res) => {  
    res.end ('Ahoj!');  
});
```



Použitie šablón

# Odpoved' render

```
res.render (view [,options] [,callback])
```

- narábanie so šablónami (template)
  - načíta šablónu **view** a doplní ju z objektu **options**
  - podľa prípony **view** sa vyberie správny templatovač
  - vyparsované šablóny sa defaultne cache-ujú

```
// priečinok so šablónami
app.set ('views', './templates')
// či sa bude cache-ovať
app.set ('view cache', false)
// defaultná prípona (templatovač) šablóny. (Ak nie je uvedená.)
app.set ('view engine', 'html')

app.get ('/', (req, res) => {
  res.render ('index', {title: 'Úvod', message: 'Vitaj stránka'});
});
```

# Templatovače

- napr.: pug, mustache, ...

express aktivuje pug automaticky. T.j. netreba pridávať engine.

```
$ npm install pug
```

```
// priečinok s template súbormi
app.set ('views', './views')
// default prípona template súborov je nastavená na .pug
app.set ('view engine', 'pug')
```

```
$ npm install mustache-express
```

```
const mustacheEx = require ('mustache-express');
// registrácia prípony (templatovača)
app.engine ('html', mustacheEx());

// priečinok s template súbormi
app.set ('views', './views')
app.set ('view engine', 'html')
```

Niektoré templatovače sa potrebujú zaradiť do expressu explicitne.



# Mustache - příklad

```
<html>
  <head>
    <title>{{title}}</title>
    {{#tmp}}
    <script src="socket.io/socket.io.js"></script>
    {{/tmp}}
  </head>
  <body>
    <div>{{{name.first}}} {{{name.surname}}}</div>
    <ul>
      {{#musketeers}}
      <li>{{.}}</li>
      {{/musketeers}}
    </ul>
  </body>
</html>
```

```
{
  title: 'Moja stránka',
  tmp: true,
  name: {first: 'John', surname: 'White'},
  musketeers: ["Athos", "Aramis", "Porthos", "D'Artagnan"]
}
```

# Vlastný templatovač

- vytvorenie

```
app.engine('tpl', (filePath, options, callback) => {  
  fs.readFile (filePath, (err, content) => {  
    if (err) return callback (err);  
  
    let rendered = content.toString()  
      .replace('#title#', options.title)  
      .replace('#message#', options.message);  
  
    return callback (null, rendered);  
  })  
})
```

Objekt, ktorý príde  
s rôznymi nastaveniami.

Túto funkciu treba  
na záver zavolať  
s výsledkom.

Callback má dva  
argumenty: chybový objekt  
a výsledný reťazec  
s dokumentom.

```
//app.set('views', './templates')  
//app.set('view engine', 'tpl')
```

# php - templatovač

- pomocou externého CLI príkazu **php**
  - pozor na globálne php premenné `$_SERVER`, ...
  - `_GET`, `_POST` sa nastaví

```
$ npm install php-express
```

```
const phpEx = require('php-express') ({binPath: ' /bin/php'});  
app.engine ('php', phpEx.engine);
```

```
app.set ('views', './mojePhpSubory')  
app.set ('view engine', 'php')
```

```
app.all (/.+\.php$/, phpEx.router)
```

Aktívne pre všetky  
súbory čo majú v URL  
príponu php.



# Middleware funkccie

# Reťazenie pravidiel

- pravidlo pre URL cestu sa hľadá v poradí
  - prvé vyhovujúce zareaguje
- callback má okrem **req**, **res** aj funkciu **next**
  - čo je pokyn na hľadanie ďalšieho pravidla
  - nazýva sa **middleware** funkcia

```
app.get ('/pokus', (req, res, next) => {  
  console.log ('Vypis');  
  next ();  
});
```

```
// ....
```

```
app.get ('/pokus', (req, res) => {  
  res.send ('Ahoj');  
});
```

Väčšinou posledný callback v poradí ukončí odpoveď a už nevolá next ().

# Reťazenie pravidiel

- callbacky-middleware funkcie môžu byť aj priamo ako viaceré argumenty jedného pravidla

```
app.get ('/pokus',  
  (req, res, next) => {  
    console.log ('Vypis');  
    next ();  
  },  
  (req, res) => {  
    res.send ('Ahoj');  
  }  
);
```





# Mapovanie prefixu URL cesty

```
app.use (PATHprefix, callback)
```

- PATHprefix zodpovedá **prefixu** URL cesty
  - reťazec, reg. výraz, default je '/'

```
app.use ('/', (req, res, next) => {  
  console.log ('Požiadavka'); // Vždy vypíše tento text  
  next ();  
});  
  
//....  
  
app.get ('/', (req, res) => {  
  res.send ('Ahoj');  
});
```

# Mapovanie súborov z disku

```
express.static (rootDir, options)
```

- vyrobí **middleware** funkciu
  - posiela súbory z priečinka **rootDir** (aj rekurzívne)
  - ak sa súbor nenájde, pomocou next() pošle hľadanie do ďalších pravidiel
  - v options napr. zoznam prípon, ktoré sa budú skúšať
  - z URL cesty sa najprv odstráni prefix pravidla

```
app.use ('/obrazky', express.static ('/data/server/images'));  
  
app.use ('/obrazky', (req, res) => {  
  res.send ('Nenasiel som');  
});
```



# Získanie POST parametrov

- je potrebný middleware - parser tela HTTP požiadavky
- middleware funkcia **urlencodedParser**

```
const bodyParser = require('body-parser');  
// Create application/x-www-form-urlencoded parser  
const urlencodedParser = bodyParser.urlencoded({extended:false});  
  
app.post ('/cmd_post', urlencodedParser, (req, res) => {  
    res.send (req.body);  
    // req.body = {"first_name":"Marek","last_name":"Nagy"}  
});
```

Zaradená middleware funkcia, ktorá vyparsuje telo požiadavky. Nájdené POST premenné sa uložia do objektu req.body.



Ďakujem za pozornosť