Backendprogrammering 1

EXPRESS

Utbildare: Mikael Olsson

mikael.olsson@emmio.se

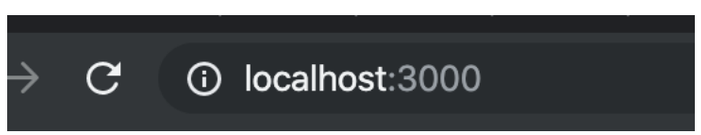
076-174 90 43

NACKADEMIN

Localhost

- DNS:en översätter domännamn till IP-adresser.

- Vissa namn är reserverade, t ex `localhost` som betyder samma som `127.0.0.1`.



use strict

- The "use strict" directive was new in ECMAScript version 5.
- It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
- The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
- With strict mode, you can not, for example, use undeclared variables.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

```
1 // Whole-script strict mode syntax
2 'use strict';
3 var v = "Hi! I'm a strict mode script!";
```

URL parameters

- Vi kan hämta delar av URL:en som parametrar med kolon.
- Skriv en funktion som lyssnar på adressen `/age/` och skriver ut olika saker beroende på vilken ålder användaren angett i URL:en.
    - Obs! Parametrar är alltid strängar!

```
// Request comes in to /instructor/ANYTHING
app.get('/instructor/:name', function (request, response) {
    response.send('Läraren heter ' + request.params.name);
});
```

Formulär

- *action*: The resource/URL where data is to be sent for processing when the form is submitted. If this is not set (or set to an empty string), then the form will be submitted back to the current page URL.

- *method*: The HTTP method used to send the data: POST or GET.

- The POST method should always be used if the data is going to result in a change to the server's database, because this can be made more resistant to cross-site forgery request attacks.

- The GET method should only be used for forms that don't change user data (e.g. a search form). It is recommended for when you want to be able to bookmark or share the URL.

```html
<form method="get" action="/handle-data">
    Name:<br>
    <input type="text" name="name"><br>
    Age:<br>
    <input type="number" name="age"><br>
    Favorite color:<br>
    <select name="color">
        <option value="green">Green</option>
        <option value="red">Red</option>
        <option value="blue">Blue</option>
    </select><br>
    <input type="submit" value="Send">
</form>
```

Ta emot formulärdata

- GET: `request.query.`*`komponent-namn`*
- POST: lite knepigare:
  ```
  npm install body-parser // ev. --save
  app.use(bodyParser.urlencoded({extended: true}));
  ```
- Nu kan vi använda: `request.body.`*`komponent-namn`*

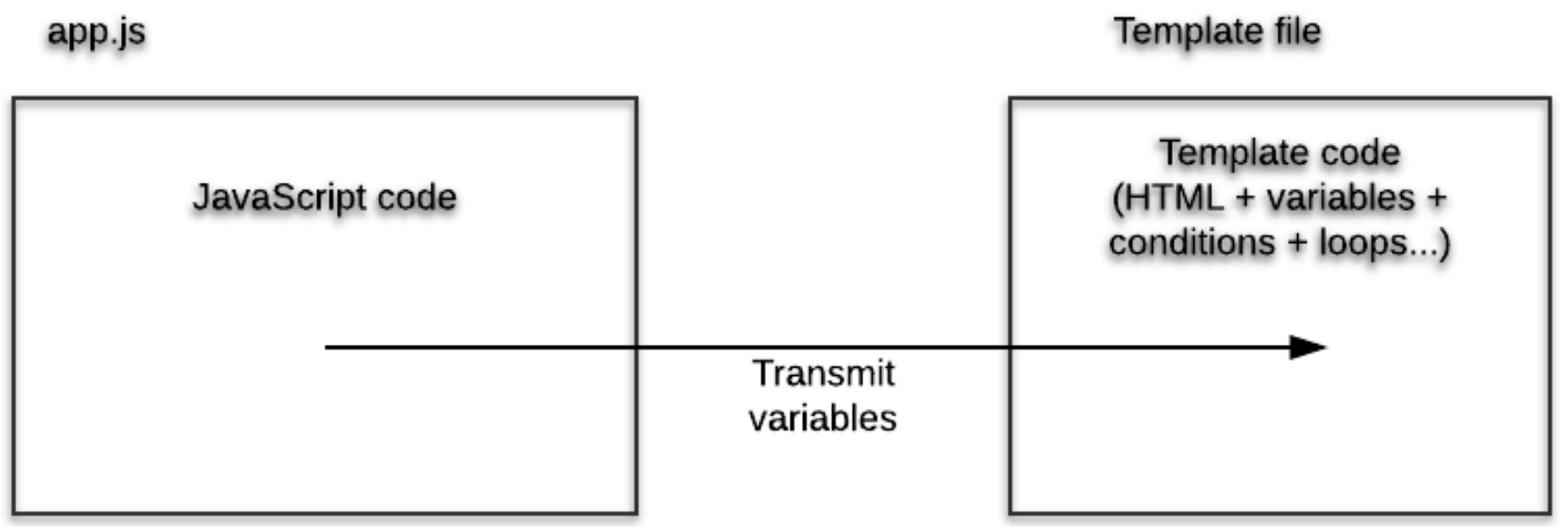
Templates

- Vi har skickat tillbaka enstaka strängar eller hela html-filer hittills.

```
res.write('<!DOCTYPE html>'+
'<html>'+
'    <head>'+
'        <meta charset="utf-8" />'+
'        <title>My Node.js page!</title>'+
'    </head>'+
'    <body>'+
'       <p>Here is a paragraph of <strong>HTML</strong>!</p>'+
'    </body>'+
'</html>');
```

- Express låter oss använda olika *template engines* (eller *template languages*) för att skapa sidor enklare.

- Vi ska titta lite på ett av dem: EJS. (Embedded JavaScript)



EJS

- `npm install ejs --save`

- Nu kan vi skicka med variabler till en template:

- `res.render('bedroom.ejs', {floor: req.params.floornum});`

- This code calls for a `bedroom.ejs` file that must be found in a sub-folder called ”views".

Multiple parameters and loops

- JS:

```
app.get('/', function(req, res) {
    let names = ['Robert', 'Jack', 'David'];
    res.render('page.ejs', {names: names, subject: 'Lorem ipsum'});
});
```

- Template:

```
<h1><%= subject %></h1>


<ul>
  <% for(let i = 1 ; i <= names.length ; i++) { %>
    <li><%= names[i] %></li>
<% } %>
</ul>
```

Uppgift

- Gör en sida som renderar ett formulär som innehåller en siffra. Formuläret ska postas (POST) till en annan sida. En router ska ta hand om det anropet och räkna från 1 till det nummer användaren postade.

66 Send

I’m going to count to 66

1... 2... 3... 4... 5... 6... 7... 8... 9... 10... 11... 12... 13... 14... 15... 16... 17... 18... 19... 20... 21... 22... 23... 24... 25... 26... 27... 28... 29... 30... 3
38... 39... 40... 41... 42... 43... 44... 45... 46... 47... 48... 49... 50... 51... 52... 53... 54... 55... 56... 57... 58... 59... 60... 61... 62... 63... 64... 65...

Middlewares

- Express is a framework based on middleware, which are application modules each providing a specific feature. You can decide which middleware you want to load.

- Express comes with over 15 basic pieces of middleware, and of course developers can add others via NPM. Each piece of middleware provides a micro-feature. Here are a few examples:

- *compression*: enables for gzip compression of pages for faster sending to the server.
- *cookie-parser*: allows you to work with cookies.
- *cookie-session*: allows you to generate session information (during a visitor’s stay on the website).
- *serve-static*: allows the return of static files contained in a folder (images, files to download, etc.).
- *serve-favicon*: manages your website's favicon.
- etc.

Middlewares

```
1  var express = require('express');
2  var morgan = require('morgan'); // loads the piece of middleware for logging
3  var favicon = require('serve-favicon'); // loads the piece of middleware for the favicon
4
5  var app = express();
6
7  app.use(morgan('combined')) // loads the piece of middleware for logging
8  .use(express.static(__dirname + '/public')) // Specifies that the /public folder includes
   static files (basic piece of middleware loaded)
9  .use(favicon(__dirname + '/public/favicon.ico')) // Activates the favicon specified
10 .use(function(req, res){ // finally answers
11     res.send('Hello');
12 });
13
14 app.listen(8080);
```

You'll need to install the middleware you need with `npm install` before running this code.

The calling order for the middleware is extremely important. For example, we start by activating the logger. If we did it last, we wouldn't be able to log anything! When you call the middleware, think of the order, because it can have a big impact on the functioning of your app.

Express application generator

- Hjälper oss att generera ett ”skelett” för ett projekt.
- `npm install -g express-generator`

```
→  03 git:(master) ✗ express --help

  Usage: express [options] [dir]

  Options:

        --version        output the version number
    -e, --ejs            add ejs engine support
        --pug            add pug engine support
        --hbs            add handlebars engine support
    -H, --hogan          add hogan.js engine support
    -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
        --no-view        use static html instead of view engine
    -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
        --git            add .gitignore
    -f, --force          force on non-empty directory
    -h, --help           output usage information
```

Express application generator

- Skapa projekt med de options du vill, t ex:

- `express --view=ejs --css=sass --git my_proj`
- `cd my_proj`
- `npm install`
- `DEBUG=myapp:* npm start`

```
→  03 git:(master) ✗ express --help

  Usage: express [options] [dir]

  Options:

        --version        output the version number
    -e, --ejs            add ejs engine support
        --pug            add pug engine support
        --hbs            add handlebars engine support
    -H, --hogan          add hogan.js engine support
    -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
        --no-view        use static html instead of view engine
    -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
        --git            add .gitignore
    -f, --force          force on non-empty directory
    -h, --help           output usage information
```

Express

Welcome to Express

Session

- En session är lite som en cookie. Kan användas för att spara temporära data på servern, t ex om användaren är inloggad eller inte.

- https://www.npmjs.com/package/cookie-session

Todo-list

- *Uppgift*: Skapa en todo-lista. Skapa applikationen mha express-generate.

- We can add elements to the to do list via the form.

- We can delete items by clicking on the crosses in the list.

- The list is stored in the visitor's session. If someone else connects to the site, they will have their own list.

- In theory, we can associate a route to each of these features: (Vilken metod ska vi lyssna efter på varje?)

- `/todo`: list of tasks.
- `/todo/add`: add a task.
- `/todo/delete/:id`: delete task n°id.