Online Multimedia

Winter Semester 2019/20

Tutorial 07 – NodeJS

Today's Agenda

- Quicktest ☺
- NodeJS:
  - Setup
  - Express Generator
  - Routes & Middleware
- Break Out
- Quiz


OPEN
LAB
DAY
Current Research
Student Projects
Medieninformatik
Mensch-Maschine-Interaktion
Human-Centered Ubiquitous Media
9.DEC
2019
18:00-21:30
Frauenlobstr. 7a
LMU
Medien Informatik München
Human Computer Interaction munich
medien.ifi.lmu.de/openlab


JAVASCRIPT
LIGHTYEAR
JAVASCRIPT EVERYWHERE
imgflip.com

NodeJS

NodeJS - About

- What is it?
  "Node.js® is a platform built on **Chrome's JavaScript runtime** for easily building fast, scalable network applications" (official website, nodejs.org)
- Node apps are JavaScript files → no additional language
- Why do we want it?
  - non-blocking I/O
  - scalability
  - web-apps can act as standalone web-server
  - largest ecosystem of open source libraries

NodeJS – Who is using it?

https://github.com/nodejs/node-v0.x-archive/wiki/Projects,-Applications,-and-Companies-Using-Node

Installing NodeJS

- Windows and Linux:
  - Download installer and run it
    https://nodejs.org/en/download/
  - make sure to install npm during the installation (default)
- OS X
  - Option 1: download and install package from nodejs.org
    https://nodejs.org/en/download/
  - Option 2: Homebrew
    brew install node
    http://shapeshed.com/setting-up-nodejs-and-npm-on-mac-osx/
- Find out if the installation was successful. Type in a terminal:
  - node -v
  - npm -v

Starting a node app from the Command Line

- On your own machine:
  ```
  $ node <path_to_file>
  ```

- On a CIP pool computer:
  ```
  $ nodejs <path_to_file>
  ```

Hello World!

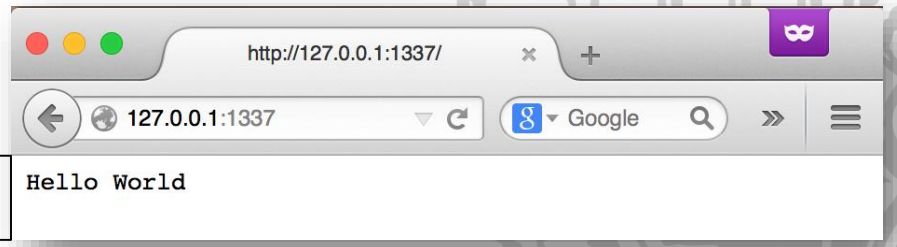
```
var http = require('http');
var port = 8976;
var host = '127.0.0.1';

var server = http.createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.end('Hello World\n');
});

server.listen(port, host);
console.log(`Server running at http://${host}:${port}/`);
```

Then in a the terminal type:

```
$ node helloworld.js
```



examples/helloworld.js

Node Package Manager

- Node is highly modular and extensible with “packages”
- npm allows you to easily handle packages from the command line and declare them as dependencies
- Most important operations:
  - Global package installation
    ```
    npm install -g PACKAGE [PACKAGE2 PACKAGE3 ..]
    ```
  - Local package installation (only for the app)
    ```
    npm install PACKAGE [PACKAGE2 PACKAGE3 ...]
    ```
  - Local package installation & saving to dependencies list:
    ```
    npm install --save PACKAGE [PACKAGE2 ...]
    ```

Package script – package.json

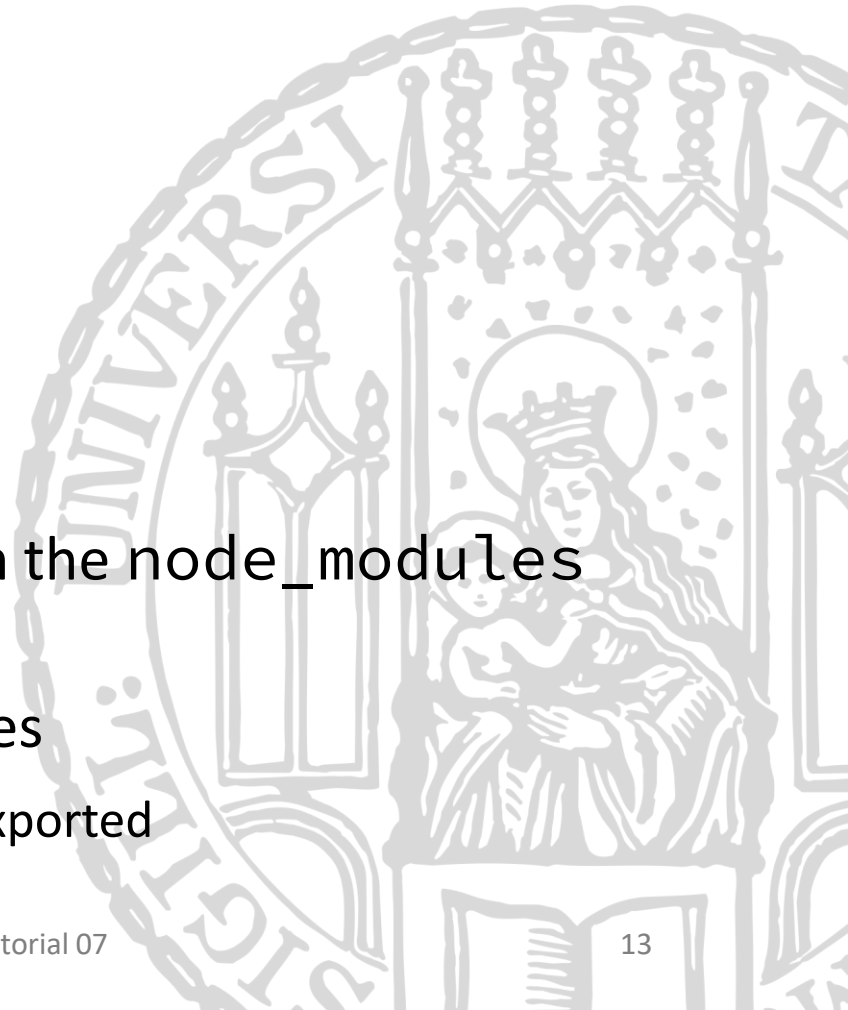
```
{
  "name": "examples",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "nodejs app.js"
  },
  "author": "",
  "license": "MIT",
  "dependencies": {
    "body-parser": "^1.14.1"
  }
}
```

$ **npm install**
installs all dependencies that are listed in the package script into the node_modules directory

Require

- To import a module, you use require():
```
var http = require('http');
```

- Mind the difference:
```
require('module');
require('./module');
```
  - Without path specification, node tries to import from the `node_modules` directory (dependencies managed by npm)
  - With path specification you refer to your own modules
    - `module.exports = something;` defines *what* is exported



ES6 import/export

examples/doublerequire.js

```
// lib.js
function double (x) {
      return x * 2;
}

module.exports = {
      double: double,
};
```

```
// main.js
var double = require('./lib').double;
console.log(double(2));
```

examples/doublees6.js

```
// lib.js
export function double (x) {
   return x * 2
}
```

```
// main.js
import { double } from './lib';
console.log(double(2));
```

Hands On: Create your first node app

- Open a bash and type
  mkdir tutorial07 && cd tutorial07
  npm init

  follow the wizard (always hitting the return key is okay).

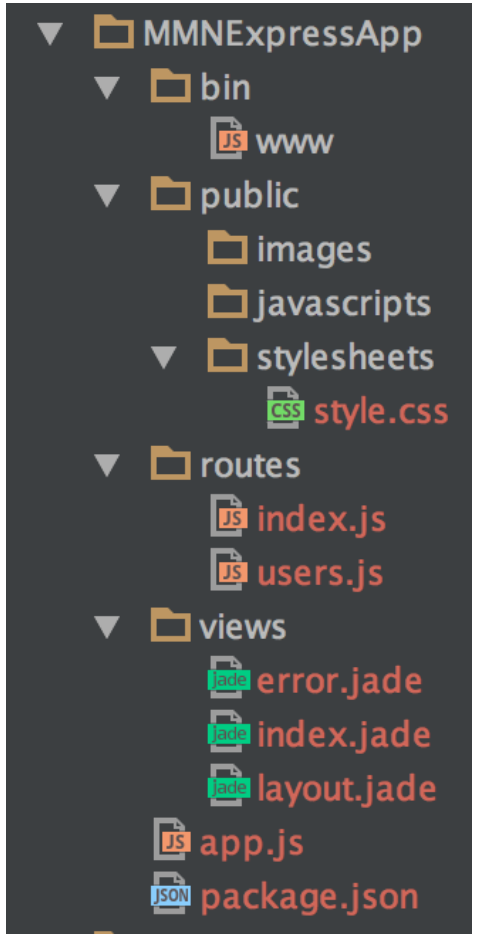
- Now you'll have your own package script package.json to which you can add your dependencies

Express – a web application framework

- One of the most popular NodeJS frameworks.
- Characteristics:
  - minimalistic
  - easy to use API
  - many utility methods and middleware functionalities
  - thin layer on top of NodeJS
- Side note:
  - responsible for the letter **E** in the MEAN stack

- Find the documentation here: http://expressjs.com/

Express generator



- Goal: automatically generate the basic structure
  of an express app that includes
  views, routes, common dependencies
- Requirements: Install the generator globally:

```
$ npm install -g express-generator
$ express OMMExpressApp
```

- Documentation:
  http://expressjs.com/starter/generator.html
- You still have to install the dependencies manually:

```
$ cd OMMExpressApp && npm install
```

Good to know…

- Never add the “node_modules” directory to version control (Git). It is enough to declare the dependencies in `package.json`

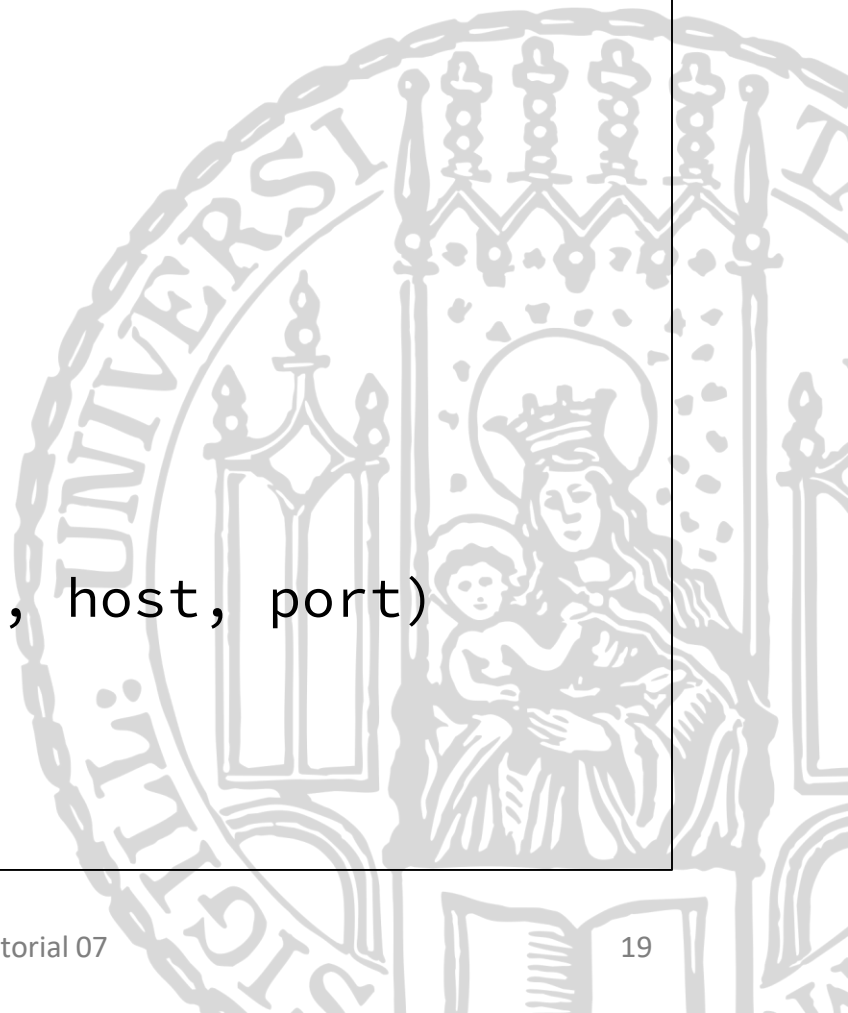
- Some .gitignore files ignore the bin/ folder. Add it with
  `$ git add -f bin`

Basic Express App

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

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

var server = app.listen(3000, () => {
  var host = server.address().address;
  var port = server.address().port;
  console.log('app listening at http://%s:%s', host, port)
});
```



Breakout: Use Express Generator

- You can also generate the app on your own
  - use the express generator
  - install the dependencies
  - run the app

- For CIP pool users: we provide an express-generated app in 07-nodeks/examples/OMMExpressApp
  - install the dependencies (npm install)
  - run the app (npm start)

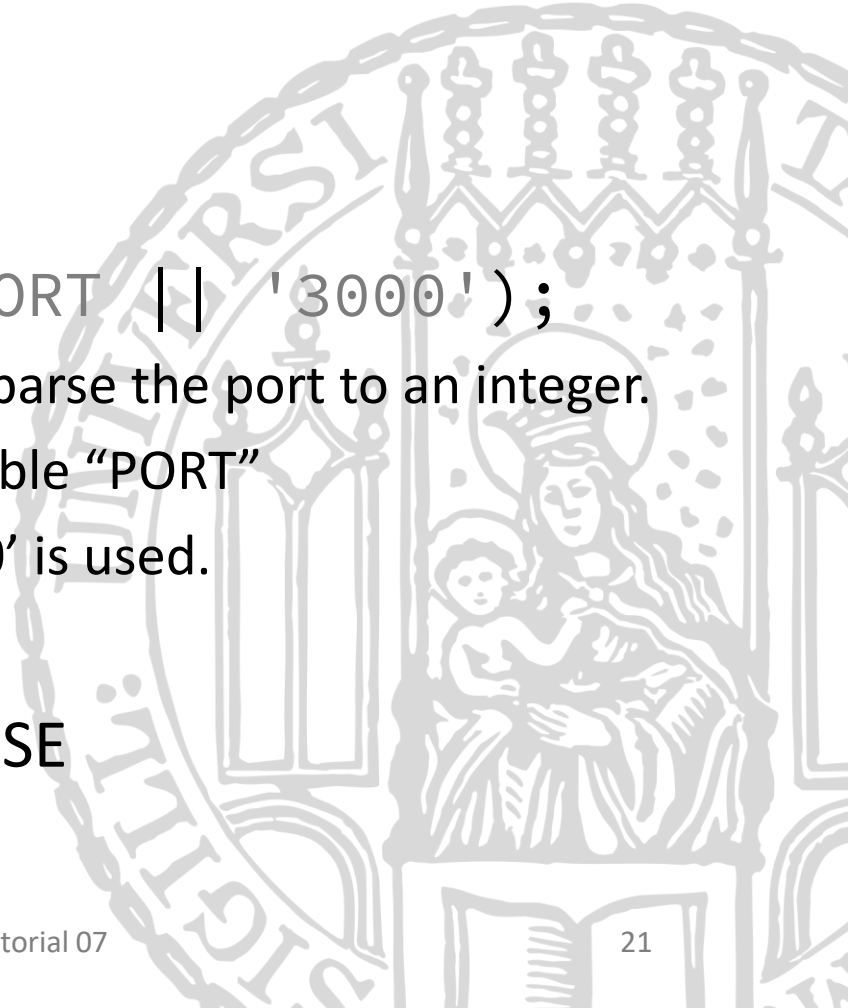
- Check: http://localhost:3000

Express Generator: bin/www

- The server process is started here (executable file)
  $ node bin/www

- Port:
  ```
  var port = normalizePort(process.env.PORT || '3000');
  ```
  - normalizePort is also defined in this file. It simply tries to parse the port to an integer.
  - process.env.PORT: accesses the environment variable “PORT”
  - If there is no such environment variable, the default ‘3000’ is used.

- Common error handlers for EACCESS and EADDRINUSE



Errors

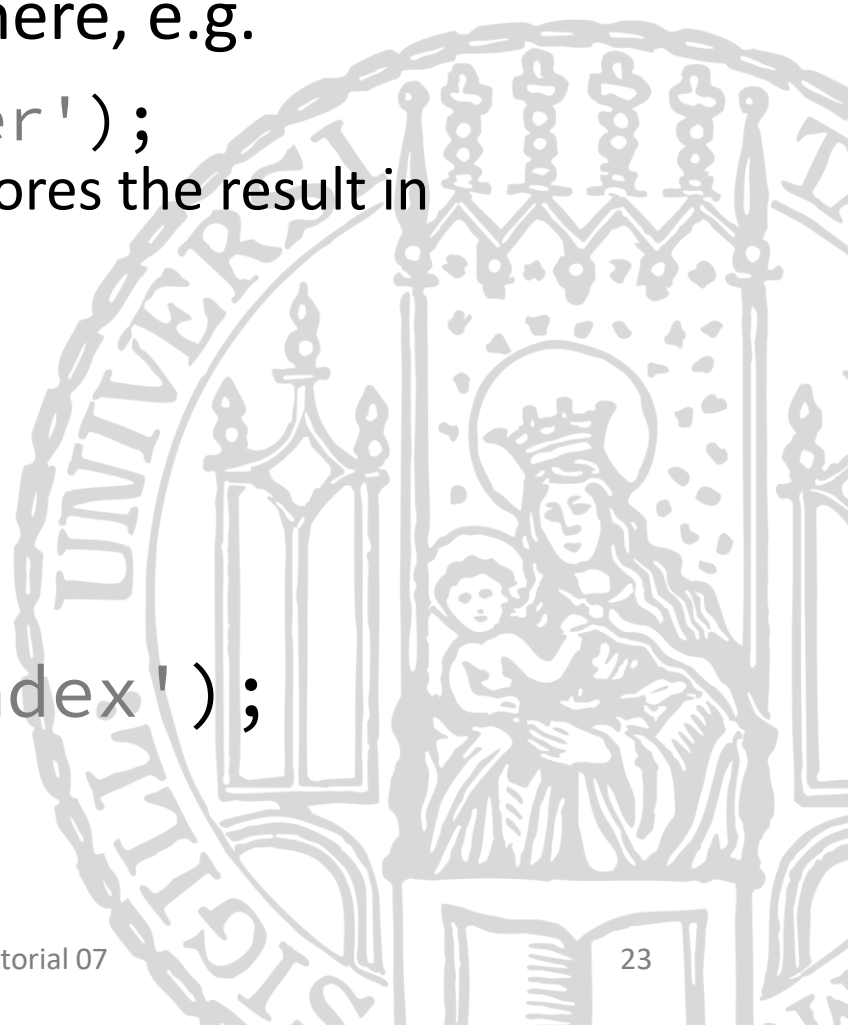
- The EACCESS error:
  - No access rights for the directory/binaries
- The EADRRINUSE error:
  - The port is already used by another application
  - Try a different port (high number) or
  - Quit the application that’s using the port, e.g.
    $ killall node

Express Generator: app.js

- All non-core modules for express are imported here, e.g.
  - `var bodyParser = require('body-parser');`
    Middleware that parses HTTP message bodies and stores the result in req.body (POST) or req.query (GET)
  - `var logger = require('morgan');`
    HTTP Request logger (generates console output)

- Middleware and basic routing setup

```
var routes = require('./routes/index');
app.use('/', routes);
```



Express Generator: routes/*.js (1)

- **Routing** is the definition of end points (URIs)
  - Express: more details how to handle paths and routing

- Route ≈ Path ≈ Mountpoint ≈ Endpoint
- Example:

```
app.get('/a-route/sub-route', (req, res) => {
  res.send('You are on a sub-route.')
});
```

http://localhost/a-route/sub-route/

- The route definitions can have placeholders for data
  - see a later tutorial for details

http://expressjs.com/starter/basic-routing.html

Express Generator: routes/*.js (2)

- Express includes a lightweight router module
```
var router = express.Router();
```
- Defining a route that handles GET requests:
```
router.get('/getRoute', function (req, res) {
  res.send('hello world');
});
```
- Defining a route to handle POST requests:
```
router.post('/postRoute', function (req, res){
  res.send('hello world, from post!');
});
```
- To use the route in the app, the router object needs to be exported:
```
module.exports = router;
```

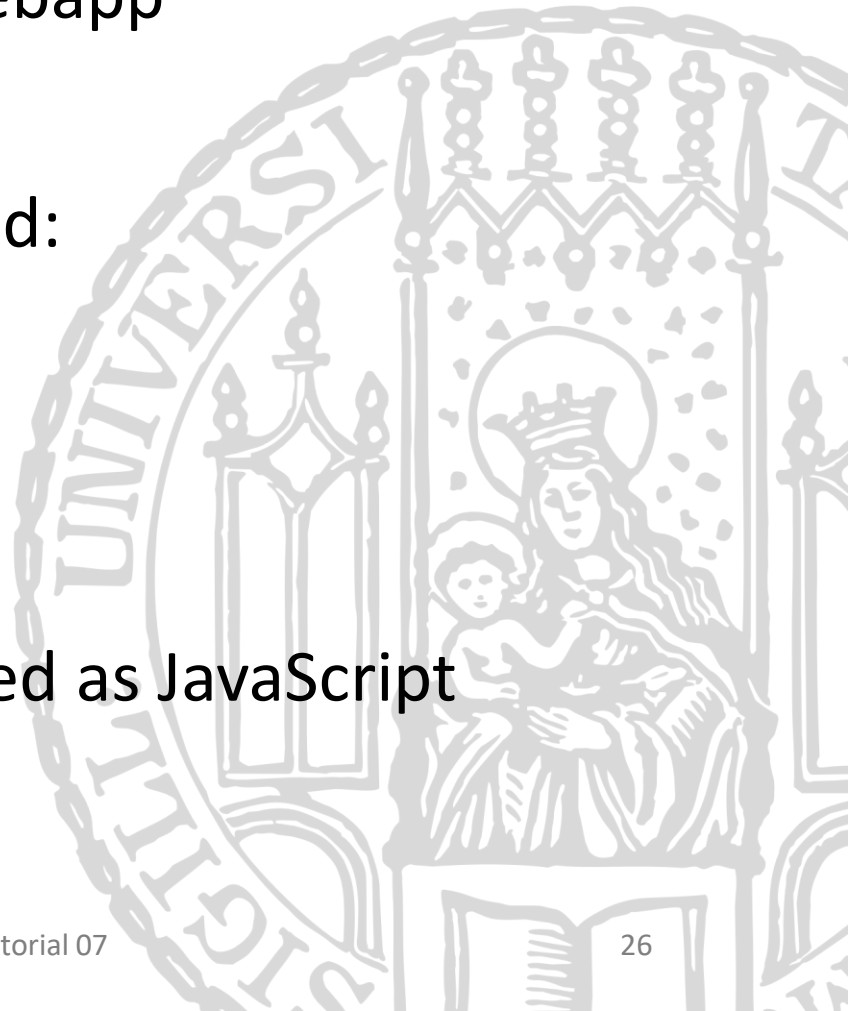
http://expressjs.com/guide/routing.html

Express Generator: views/*.pug or *.jade

- Views are templates that are compiled by the webapp
- The default rendering engines are Jade and Pug
- routes/index.js shows how a template is rendered:

```
router.get('/', (req, res, next) => {
  res.render('index', { title: 'Express' });
});
```

- The parameters to render the template are passed as JavaScript object



Break Out: A new route

- Copy the file routes/index.js to routes/products.js
- Include the new route '/products' in app.js
- Store the product list in a variable
- The route should offer:
- **get**: the app responds with a JSON object like this:

```
{
  "code": 200,
  "products": [
    { "name": "Product A", "price": 30, "id": "id_A" },
    { "name": "Product B", "price": 50, "id": "id_B" },
  ]
}
```

- **post**: the app stores a new product in product list. Response:

```
{
  "code": 201,
  "message": "created product",
  "products": [ {
    "id": "d5e89b80-b56e-11e6-adf3-75d0cc9cd069",
    "name": "Product C",
    "price": 100
  } ]
}
```

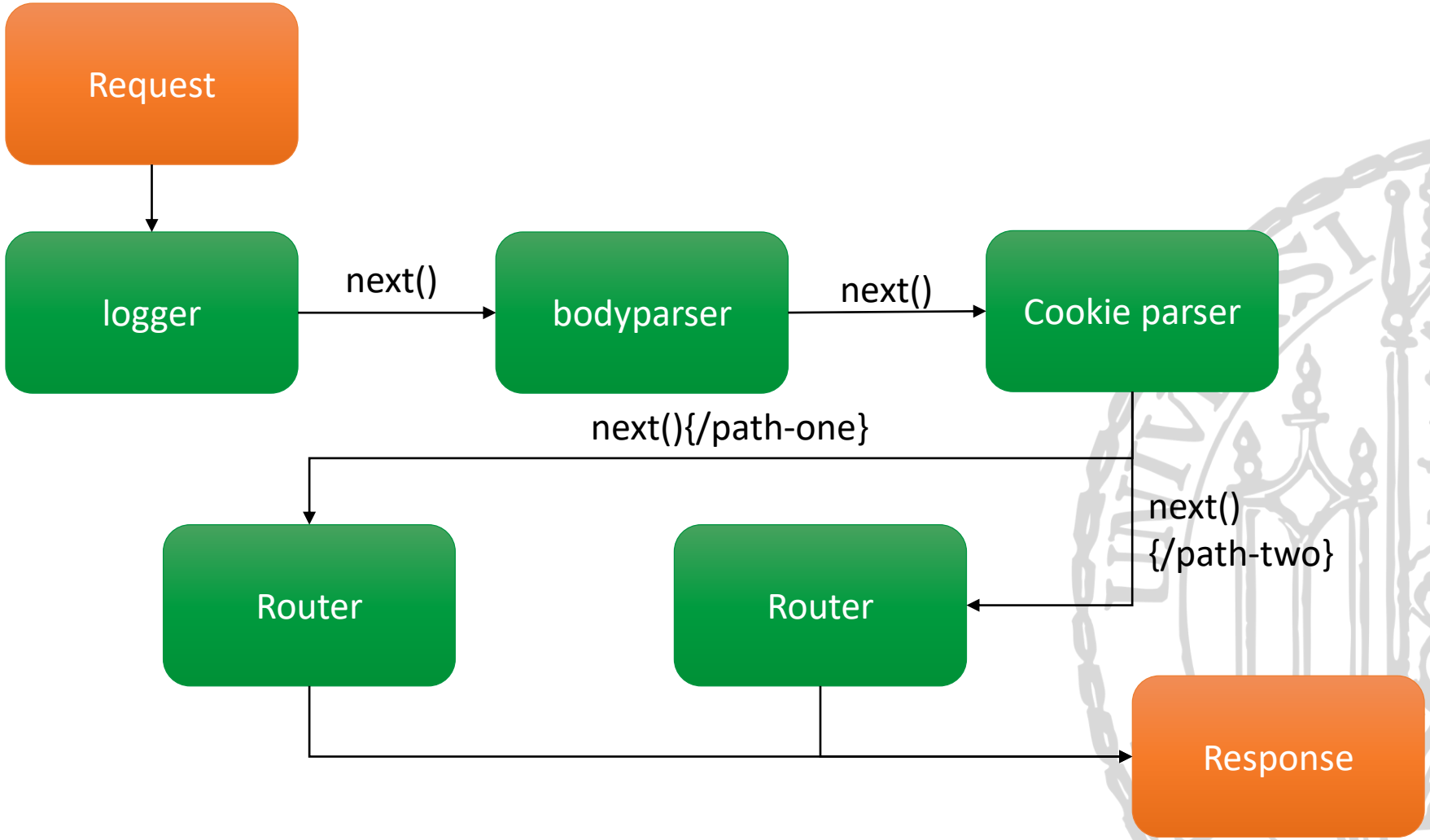
Middleware

- A middleware is a function that sits between the request and the response (“in the middle”)
- Does something with the request. Typical tasks:
  - parse the HTTP message body to a JSON object (`body-parser`)
  - parse cookies (`cookie-parser`)
  - authenticate the user and (dis)allow a request
- Usually more than one middleware per route (middleware chain)

```
app.use('/', (req, res, next) => {
  // do something with req.object
  // then either send the response or call:
  next();
});
```

- Am express router is a middleware!

Middleware visualization



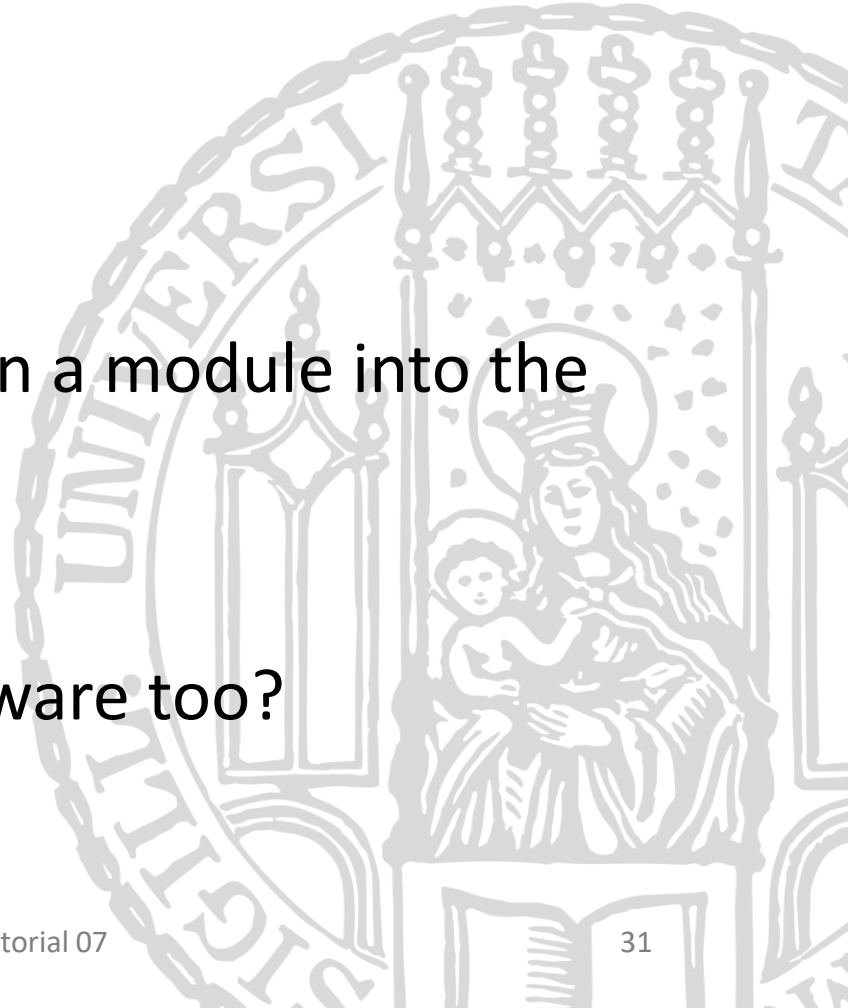
Serving static files and directories

- Express has a built-in middleware to serve directories, e.g. for HTML, CSS, JavaScript, or image files: express.static
- Example usage in app.js to serve a directory named “public”:
  ```
  app.use(express.static('public')));
  ```
- Content of the directory is accessible from the server root:
  http://localhost:3000/images/kitten.jpg
  http://localhost:3000/hello.html
- You can also specify a mount-path:
  ```
  app.use('/kittens', express.static('/kittens'));
  ```
- The path is relative to the node process, so do this:
  ```
  app.use(express.static(__dirname + '/public'));
  ```

http://expressjs.com/starter/static-files.html

Round-up Quiz

1. Where does NodeJS run, client or server?
2. What does `require(...)` do?
3. How do you generate a package script?
4. How do you conveniently save a dependency on a module into the package script?
5. What does the body-parser middleware do?
6. What is a middleware? Is every route a middleware too?



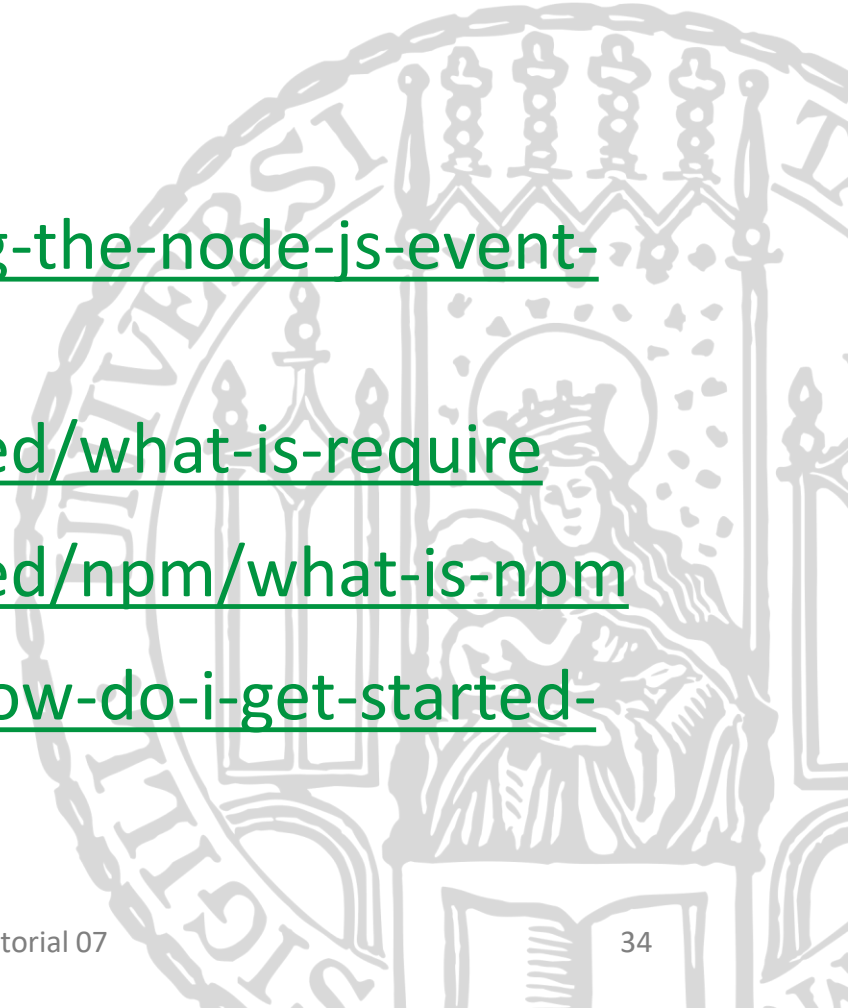
Thanks!

What are your questions?

Appendix

Some more links

- http://nodeschool.io/
- http://nodeguide.com/beginner.html
- http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/
- http://docs.nodejitsu.com/articles/getting-started/what-is-require
- http://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm
- http://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js



Sources

- http://www.talentbuddy.co/blog/building-with-node-js-at-ebay/
- http://www.talentbuddy.co/blog/building-with-node-js-at-netflix/
- http://venturebeat.com/2011/08/16/linkedin-node/