Modern Web tools with Node.js

Jukka Paasonen & Co

2014

Table of Contents

Modern Web tools with Node.js - Course book

> Nykyaikaiset Node.js työkalut tehokkaaseen Web työskentelyyn TT00CC06-3001

A course of using Node.js and related tools in Web development, in Autumn 2014. Held on every Tuesday at 16:40 - 20:15, class room U423 in Bulevard Campus of Metropolia University of Applied Sciences, Helsinki, Finland.

Lecturer Jukka Paasonen, `<first name . last name @ school name . country ISO code>`.

Please see index.md for further contents.

Gitbook

book passing dependencies up-to-date analytics GA

This project is a Gitbook, which is published on each `git push` to Modern Web tools with Node.js.

Build locally, the `LC` variable seems to be needed in Mac, also `sudo` might be needed in Linux:

```
npm install --global gitbook gitbook-plugin-anchors gitbook-plugin-exercises
LC_CTYPE=en_US.UTF-8 gitbook build
```

Cover image shape created by Toni Suni.

Useful links:

- Awesome list about Node.js things, which most likely will help to find the correct tool any of the tasks in this book.
- GitHub Student Developer Pack offer

Contributing

Please refer to a GitHub blog post on how to create somewhat perfect pull request.

Course Definition

Book for the Node.js course help in the Metropolia University of Applied Sciences, Helsinki, Finland, during Autumn 2014. Developed during the course, by all course participants.

- Questions, issues
- Tieto- ja viestintätekniikan tutkinto-ohjelman avoimen opintotarjonta Helsingin toimipisteessä

It should be noted that throughout this course, the Node.js version in question is `0.10.x`.

Lectures

Lectures are held in Bulevard Campus, class room U423, every Tuesday at 16:40 - 20:45.

Any questions that might come up during or between the lectures, please open an issue if you feel that the subject would be useful for everyone.

1. Introduction to Node.js and JavaScript
2. GitHub forks and pull requests
3. Modules and npm
4. Code validation, conventions, linting
5. HTTP, Connect, Express
6. Common task runners
7. Unit testing, Jasmine, PhantomJS
8. Automation and continuous integration
9. Package dependencies
10. Front end third party dependencies and Code coverage
11. Web performance
12. More about unit testing and code coverage
13. Security
14. Database
15. Heroku and other free hosting services
16. Examples for previous tasks
17. Feedback on the projects

Assignments

The course contains four assignments, which are given in three weeks cycles. Each assignment should be returned in two weeks when it has been given, preferably via GitHub.

1. First assignment
2. Second assignment
3. Third assignment
4. Fourth assignment

The projects which are used for completing the assignments, are listed here.

Presentations

Each lecture contains two presentations made by student. Each presentation can last about ~15 minutes.

Presentation subjects are divided according to the themes of each lesson and assigned to the given students according to their own wishes. Please open a pull request for marking your presentation.

Subjects, presenters and dates.

Once a presentation has been held, the material should be linked in the above list.

Pre-recorded presentations

Each lecture contains one or two relevant presentations that are presented by inspiring people around the Node.js community.

Based on the contents of the presentations, each subject has quizzes and exercises.

Targets defined as course delivery

- The student is able to recognise the possibilities available via Node.js in a user interface development.
- The student can understand the meaning and reasoning for tools such as Grunt, Gulp, Bower and Jasmine in order to reduce the time used for repetitive tasks and to increase code quality.
- The student can create small scripts and plugins that can be used via command line.
- The student can publish their own tools and keep them up to date via version control systems.
- The student learns how to use GitHub service for group work.

Evaluation criteria

Each student is evaluated according to the following criteria, based on their returned assignments, presentations and other studying activities during the lessons.

Student participation matrix should contain links to the relevant activities.

Modest

- The student can install Node.js and its related tools and applications
- The student can run existing Node.js tools from the commandline
- The student can create simple applications and to use them
- The student can understand how modules properties and methods are shown to third party scripts
- The student can understand the basics of unit testing

Good

- The student can publish their own tool to the npm registry
- The student can create tests for their own tools/plugins
- The student can use the Node.js API efficiently and find methods that work for the given task

Brilliant

- The tools that the student has published have continuous automated testing
- The student is able to update their own tools via version control system
- The student can grasp what third party plugins can achieve and can use them efficiently in their own project
- The student is able to participate the development of other tools via pull requests

Lecture 1. - Introduction to Node.js and JavaScript

> Date 26/08/2014

This course (TT00CC06-3001) is about getting to know Node.js and tools that are made with it.

The primary focus is towards functionality and tooling that can improve the workflow of a Web Developer.

JavaScript does have partially the same name as Java, but that is basically all what they have in common. For example variable type requirements are very different as this cartoon explains.

Testing your initial JavaScript skills

You can't JavaScript under pressure

Below is all the test game extracted from the above game, with possible solutions.

Exercise

`i` will be an integer. Double it and return it.

```
function doubleInteger(i) {
    return i;
}
```

```
function doubleInteger(i) {
    return i * 2;
}
```

Exercise

`i` will be an integer. Return true if it's even, and false if it isn't.

```
function isNumberEven(i) {

}
```

```
function isNumberEven(i) {
    return i % 2 === 0;
}
```

Exercise

`i` will be a string, but it may not have a file extension. return the file extension (with no period) if it has one, otherwise `false`.

```
function getFileExtension(i) {

}
```

```
function getFileExtension(i) {
    return i.indexOf('.') !== -1 ? i.split('.').pop() : false;
}
```

Exercise

You'll get an array `i`. Return the longest string inside it.

```
function longestString(i) {

}
```

```
function longestString(i) {
    return i.reduce(function (curr, prev) {
        if (typeof curr !== 'string' || prev && prev.length > curr.length) {
            return prev;
        }
        return curr;
    });
}
```

Exercise

`i` will be an array, containing integers, strings and/or arrays like itself. Sum all the integers you find, anywhere in the nest of arrays.

```
function arraySum(i) {

}
```

```
function arraySum(i) {
    var total = 0;
    var iterate = function (list) {
        list.forEach(function (value) {
            if (typeof value === 'number') {
                total += value;
            }
            else if (value instanceof Array) {
                iterate(value);
            }
        });
    };
    iterate(i);
    return total +1;
}
```

Testing Node.js in the cloud

While the local development environment is not necessarily available, there are few service which provide virtual machines that can host environments such as Node.js.

- Codeanywhere
- Nitrous.IO

Links related to the lecture subject

- Eloquent JavaScript
- Javascript BEST PRACTICES PART 1
- Short, in-depth explanations of JavaScript code and concepts
- Programming JavaScript Applications
- Understanding Scope and Context in JavaScript
- The Nodefirm - Training Courses
- nodemeatspace.com
- nodeschool.io
- An Absolute Beginner's Guide to Node.js
- What's in a Function Name?
- Every Possible Way to Define a Javascript Function
- Node.js tools for Visual Studio
- Free Visual Studio for Developers

Lecture 2. - GitHub forks and pull requests

> Date 02/09/2014

Git client should be configured with your GitHub account information, so that the commits are registered to your user account at GitHub, instead of the current computer. Example commands below, please substitute with your proper values:

```
git config --global user.name "Juga Paazmaya"
git config --global user.email paazmaya@yahoo.com
git config --global github.user paazmaya
```

In case you wish to make the above configuration changes to a given repository, omit the `--global` parameter.

Tasks that are done during this lecture:

1. Create local repository, `git init`
   - Add files to it, `git add`
   - Commit those files with a meaningful message, `git commit -m`
2. Create an empty repository at GitHub
   - Add that as a remote for the repository made in task 1.
   - Push your local repository to GitHub, making it public, `git push`
3. Create new repository at GitHub, with predefined files
   - Clone the repository, `git clone`
   - Make changes and push them
4. Tag the latest commit of the repository made in 3. with Semantic Versioning number, `git tag`
   - Add release notes to it in GitHub, thus making it a release
5. Create a fork from a repository of the student sitting next to you, that was created in 3.
   - Make few changes to the `README.md` and push
   - Create a Pull Request
6. Make changes to your own repository, which the other has forked
7. Update the fork created in 5., `git fetch` , `git merge`

Now that you are familiar with Git, GitHub and forking, feel free to fork this repository and create a pull request that would add your presentation subject and date #11.

Links related to the lecture subject

- GitHub Guides: Hello World
- GitHub Glossary
- GIT FAQ
- Visualizing Git Concepts with D3
- Git for Everyone
- github-cheat-sheet
- A Visual Git Reference
- Using pull requests
- Configuring a remote for a fork
- How to Write a Git Commit Message
- Git and GitHub Workflows at the Utah JUG
- Javascript BEST PRACTICES PART 2
- Repository is marked as the wrong language

Examples for the tasks

1. Local repository with a file and a commit

```
mkdir first-git-project # create directory
cd first-git-project # go to the directory
git init # initialise a local git repository
touch README.md # create empty file
git add README.md # add a file under version control
git commit -m "First file created" README.md # commit the file
```

2. Create empty repository at GitHub and push to it

After creating the empty repository via GitHub web page, add it as a remote and push to it.

```
git remote add origin git@github.com:paazmaya/modern-web-tools-with-node-js-book.git
git push -u origin master
```

Please note that the `-u` parameter is a short version of `--set-upstream` which is seen in the documentation. It needs to be used only at the first time pushing, after setting the remote repository URL.

3. Repository with predefined files

After creating the repository via GitHub web page, and selecting a predefined `Node` specific `.gitignore` file, license and default `README`, clone it to your computer.

```
git clone git@github.com:paazmaya/modern-web-tools-with-node-js-book.git
cd modern-web-tools-with-node-js-book # go to the directory
nano README.md # open the file in an editor
git commit -m "Updated description" README.md # commit the changed file
git push # make the changes public
```

4. Tagging

```
git tag v0.1.0 -m "Time to make the first release with basic functionality"
git push --tags
```

Now in the release page at GitHub repository, the release needs to be created.

5. Forking

```
git clone git@github.com:paazmaya/modern-web-tools-with-node-js-book.git
cd modern-web-tools-with-node-js-book # go to the directory
nano README.md # open the file in an editor
git commit -m "Updated description" README.md # commit the changed file
git push # make the changes public
```

The rest of the task needs to be done in the GitHub page of the fork repository, where exists buttons for `compare` and `pull request`.

7. Updating fork

Assuming that the current working directory is the fork and it has `master` checked out:

```
git remote add upstream git@github.com:paazmaya/modern-web-tools-with-node-js-book.git
git fetch upstream master
git merge upstream/master
git push
```

Lecture 3. - Modules and npm

> Date 09/09/2014

Node.js applications are JavaScript files that are run via `node` binary.

Scope in Node.js is file based, thus each file can be used as a self contained module. In case there are several files which are to be used together, one should be the main (as `main()` would be in languages like C) while others are modules exporting their functionality. In the tasks to be done below, the `task-1.js` is the main file for the first task.

The tasks done during this lecture should be saved in a GitHub repository called `hello-node-js`. Each file should be named in lowercase, starting with `task-`, followed by the task number and end with `.js`. Thus running a command `node task-1.js` would run the code created in the first task (and print out "Hello World").

1. Create a "Hello World" application, an application that prints out the text "Hello World"
2. Create an application that takes one numerical argument, doubles it and prints it
3. Create an application that uses a module.
   - The module has one method
   - That method takes one parameter, a filename
   - That method returns the timestamp of the last modification time for the file if it exists. Else it should return false
   - The main part of the application should format the given timestamp to match Finnish date and time format
   - The result is finally printed out.
4. Fork the repository of your classmate and create a pull request on their task 3, by doing the following changes:
   - Find an existing module from npmjs.org that can be used for formatting the timestamp
   - Replace the formatting functionality with one from the module found.
5. Optional: Come up with a repetitive task that you might be doing every now and then, for example renaming files based on certain criteria, resizing images, trimming whitespace, etc.
   - Create an application that simplifies that work for you
   - Consider using `.forEach` at least once
   - Possibly use other programs installed via `spawn`.
6. Optional: Extend the task 5 to be used as a module
   - Create `package.json` with required information and license of your choosing
   - Tag `v0.1.0` release
   - Optional: publish it to npm.

Please note that in task 4, GitHub will take care of any naming collisions that might occur if the repositories have the same name.

Feel free to add any links below that were helpful for you when completing the above tasks, via pull requests.

Links related to the lecture subject

- Node.js API documentation
- Setting up Node.js Package Manager (npm)
- npm cheat sheet
- How to npm
- Browserify handbook
- Video: Understanding Scope in JavaScript
- Example code repository for the tasks 1-4
- MIT License explained
- npm 1.0: Global vs Local installation
- SemVer 0.y.z

Code examples

Task 1

Simply just using `console.log()` method prints out to the command prompt, as described in the relevant API documentation.

```
// task-1.js
var words = "Hello World";
console.log(words);
```

The string which is saved in the variable `words`, could also be written directly inside the `console.log()` call, but for the sake of maintainability and possibly further development it is better to use variables even for such a simple task.

Another option would be to use the `util` module. For example:

```
// task-1-util.js
var util = require('util');
var words = "Hello World";
util.puts(words);
```

Task 2

In order to get access on the command line arguments, use global `process` variable. It is also documented in the API documentation.

```
// task-2.js
var arg = process.argv[2];
console.log(arg * 2);
```

Run the given application `node task-2.js 3`. It should output `6`.

The `process.argv` is an array of the command line arguments, starting with the `node` at index 0. Therefore the index 2 contains the first argument given to the script, while the script filename is available at the index 1.

Task 3

Use the native module called `fs` for file system access.

```
// last-mod.js
var fs = require('fs');

module.exports = function(filename) {
    if (fs.existsSync(filename)) {
        var stat = fs.statSync(filename);
        return stat.mtime;
    }
    return false;
};
```

The above use synchronous method `fs.statSync()` returns a statistics object which provides the information needed in its `mstat` property.

It should be noted that the returned value was mistakenly assumed to be a timestamp (number of seconds from 1.1.1970), but it is in fact an instance of the JavaScript Date object.

Now the `last-mod.js` file can be used as a module, by requiring it in the same fashion as other modules. In order to make it clearer that it is in the same directory, the `./` is used.

The main file could look something similar as shown below. Note that the date formatting takes up most of the lines.

```
// task-3.js
var m = require('./last-mod');

var filename = process.argv[2];
var timestamp = m(filename);
if (timestamp !== false) {
    var date = timestamp;

    var fin = date.getDate() + '.' +
        (date.getMonth() + 1) + '.' +
        date.getFullYear();

    fin += '  ' + date.getHours() + '.' +
        date.getMinutes();

    console.log(fin);
}
else {
    console.log("Error message");
}
```

Task 4

Once the fork has been done at GitHub and cloned locally, `package.json` configuration file needs to be created, in order to define the required depencency.

```
npm init
```

The above command will ask few questions, such as name and version. It will create something similar to:

```
{
  "name": "Lecture-3",
  "version": "0.1.0",
  "description": "",
  "main": "task-3.js",
  "dependencies": {},
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "MIT"
}
```

It is fine to ignore the error text in the test section. The testing subject will be discussed further in a later lecture.

Now the module need to be installed and saved to the dependency list. The following command does both of those:

```
npm install --save moment
```

Thus the `package.json` will be updated with the following information:

```
"dependencies": {
  "moment": "^2.8.3"
}
```

In order to take the `moment` module in to use, it needs to be imported with `var moment = require('moment');`. Now the module is available with `moment` variable name and the date formatting can be simplified to a single line, such as `moment(date).format('D.M.YYYY H.m');`. Formatting options are described in detail at the moment module documentation.

Resulting updated main file would look something like shown below:

```
var m = require('./last-mod');
var moment = require('moment');

var filename = process.argv[2];
var timestamp = m(filename);
if (timestamp !== false) {
    var date = new Date(timestamp);

    var fin = moment(date).format('D.M.YYYY  H.m');

    console.log(fin);
}
else {
    console.log("Error message");
}
```

The modules downloaded from npmjs.org are placed in a directory called `node_modules`. In most cases that folder should not be placed under version control, thus it should be ignored by Git. This can be achieved with the `.gitignore` file, which contains a list of files which will be ignored.

In this case the contents of `.gitignore` would look something similar to:

```
# Dependency directory
node_modules
```

The file `.gitignore` itself should be under version control. It can be created and added to the reporitory with these commands:

```
touch .gitignore
git add .gitignore
```

Lecture 4. - Code validation, conventions, linting

> Date 16/09/2014

First assignment is given.

Why code style, validation and linting matters?

Whether there is just one or more developers working on a given code base, over time it comes difficult to read one's own code if the code style differs. Any shared code should follow certain style guides for making it consistent.

Code validation checks against syntax failures, possible naming collitions and other relevant things that might be needed in order for the code to work in a given set of environments. What works in one browser, might not work in another.

For Node.js, the environment is somewhat stable and while updating patch level (`0.0.x`) versions, should not break existing code. Also since is is always made from the same basic components, namely V8, it has somewhat similar runtime expectations between minor (`0.x.0`) versions.

Code linting validates and evaluates code against a set of given rules, usually based on code style and validation best practises, as well as personal taste.

Tasks for the day

Use the repository `hello-node-js` that was created during the previous lecture.

1. Automatic Semicolon Insertion (ASI) is a handy helper found in many browsers, but it is usually best to be avoided
   - Found out about ASI and write about it few lines in your `README.md`
   - Validate your code, `task-*.js` files against ASI and add a validation configuration file in your repository
2. Choose an existing Node.js module from npm which does not have code validation included
   - Create a fork from its source
   - Add code validation configuration, which would approximately match the existing code style used in that module
   - Optional: Create pull request to the module
   - Link your pull request or fork to the list below
3. Take a look at the source codes of the following linters and observe how they are achieving similar things differently. Then write down in your `hello-node-js` repository `linters-study.md` file about the similarities and differences, as well as external module usage. Write a short configuration example how to prevent ASI failures, for each three linters
   - jslint
   - JSHint
   - ESLint
4. Optional: Write a rule for one of the three linters
   - Create a new repository or clone the original linter project
   - Instead of using the `master` branch, create a new branch with name of your new rule, prepended by `rule-`
   - Feeling adventurous: create a pull request to the linter project for adding your rule in it.

Links to the pull requests and forks made for task 2

- ESLint validation added to node-easyimage / Heikki Alanen
- JSHint validation added to ncht / Joonas Meriläinen
- ESLint validation added to gulp-minify-css / Jukka Rautanen
- JSHint validation added to async / Markus Iivonen
- JSHint validation added to Lo-Dash / Marko Hamppula
- Toomaksen pull request

Links related to the lecture subject

- A Bit of Advice for the JavaScript Semicolon Haters
- winning style
- Airbnb JavaScript Style Guide
- JavaScript Code Style checker
- Create a new branch with git and manage branches
- ESLint Rules List, with comparison of jslint, jscs and JSHint equivalents

Examples for the tasks

Lecture 5. - HTTP, Connect, Express

> Date 23/09/2014

Setting up a minimal HTTP server with `connect` is just a matter of few lines:

```
var connect = require('connect');
var http = require('http');

var app = connect();

app.use('/', function(req, res){
  res.end('Welcome to Connect!\n');
});

http.createServer(app).listen(6500);
```

Similar approach can be take with `express`:

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

var app = express();

app.get('/', function (req, res) {
  res.send('Welcome to Express!\n');
});

app.listen(6500);
```

Adding a middleware in both `connect` and `express` is pretty trivial:

```
var responseTime = require('response-time');

app.use(responseTime());
```

The `response-time` middleware adds back end execution time header on the request, which can be used for testing.

Another perhaps more useful middleware is called `body-parser` which can be used to retrive the form values once such a form has been submitted.

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

app.use(bodyParser.urlencoded());
```

Please note that in order to have the above middlewares available to your application, as well as `connect` or `express`, they need to be installed locally first, for example:

```
npm install --save body-parser
```

The task for the day

Create a group of three members, of which one is the owner of the given new repository, and the two others are marked as its contributors.

Half of the class will use Connect for the given task, while the other half shall use Express.

Application requirements:

- Run a web server that shows a feedback form in its index page, started with a command `npm start`
- The feedback form contains these items:
  - Name
  - Email
  - Feedback content
  - Submit button
- The form should be sent to the back end with a post request
- The input fields needs to be validated both front end and back end for not being empty and email being valid
- Valid feedback should be send to an email address configured via `package.json` OR send as a Gist
- On successful form sending the page should show a suitable gif animation
- On any error cases, the form should remain usable and point out the possible errors
- Use of existing npm modules is encouraged
- All self made JavaScript code should be under linting rules, which can be run with `npm test`
- Application logic is separated to different files and used as modules

Divide the work for all the three members and work together in the same repository.

Once the work has started, add a link to the main repository in the list below.

List of projects

Links to the project repositories, along with the group member names.

- Jukka Rautanen, Sopheak Kong, Joonas Meriläinen
- Oona, Joose, Maxim
- HeikkiAlanen, olemstrom, tuunanen
- Marko H, Markus I

Links related to the lecture subject

- Better logging in Node.js
- Connect
- Express
- `http` API documentation
- GitHub Help - About gists
- Express Routing - The Beginners Guide
- Video: Fluent 2014 Introduction to ExpressJS

Examples related to the tasks

```
// connect-example.js
var connect = require('connect');
var http = require('http');
var responseTime = require('response-time');
`
var app = connect();

app.use(responseTime());

app.use('/', function(req, res){
  res.end('Welcome to Connect!\n');
});

http.createServer(app).listen(6500);
```

```
// body-parsing.js
var express = require('express');

// https://github.com/expressjs/body-parser
var bodyParser = require('body-parser');

var app = express();

app.use(bodyParser.urlencoded());

app.get('/', function (req, res) {
    res.type('html');
    res.write('<form method="post">');
    res.write('<input name="email" value="hello@there">');
    res.write('<button>Send</button>');
    res.write('</form>');
    res.end();
});

app.post('/', function (req, res) {
  res.setHeader('Content-Type', 'text/plain');
  res.write('you posted:' + req.param('email'));
  res.write('\n');
  res.write('you posted:\n');
  res.end(JSON.stringify(req.body, null, 2));
});

app.listen(6500);
```

```
npm start
// https://www.npmjs.org/doc/cli/npm-start.html
// http://browsenpm.org/package.json#scripts.start
```

HTML5 validation via `required` and `pattern` properties. http://html5pattern.com/

Reading values from `package.json`

```
// read-package-author.js
var fs = require('fs');
var json = fs.readFileSync('package.json');
var pkg = JSON.parse(json);
// http://browsenpm.org/package.json#author

console.log('Package author: ' + pkg.author);
```

Lecture 6. - Common task runners

> Date 30/09/2014

Developers often need to do repetitive manual work which could be automated.

Grunt will be used as the primary example since it is the oldest, most mature and with largest community.

Tasks can be created for command line tools as well as programmable APIs. Both of these approaches will be looked at, via existing implementations, namely grunt-eslint and grunt-webp.

Tasks for the day

1. Create a pull request at GitHub to someone else's repository, for example a classmate
   - Add a task runner
   - Configure the existing linting configuration to be run via task runner
   - In case there is no linting, add it in a separate configuration file
   - Link the pull request below.
2. Compare how achieve the same with another task runner or a build tool (one of Grunt, gulp and Broccoli)
   - Create a single file in your `hello-node-js` repository and name it `[task runner name]-comparison.js`
   - The file should be able to complete the same task as the one in the pull request
   - If any additional configuration is needed, add an example of that in the file header comment section.
3. Optional: find a useful tool, which does not have one of the presented task runner versions available and create such
   - It can also be a command line tool, not necessarily made with Node.js
   - It can even be a web service which has a RESTful API for processing files

Pull requests related to the first task

- Task 1-2, Oona
- TomTerin pull request lalallala
- Task 1, Maxim D
- @olemstrom
- HeikkiAlanen
- Sopheak Kong
- Jukka Rautanen
- Onnia
- merilainen-metropolia
- Markus Iivonen

Links related to the lecture subject

- Broccoli
- Grunt
- gulp.js

Lecture 7. - Unit testing, Jasmine, PhantomJS

> Date 07/10/2014

Second assignment is given.

Unit testing in general

Unit testing, as the name suggests, is a way of testing units or portions of code.

Popular unit testing frameworks

- Jasmine
- Mocha
- Qunit
- nodeunit

PhantomJS, a headless WebKit browser

In an environment where there is no availability of screens, such as a server, continuous integration server specifically, is a need for tools which do not require the screen. One such tool specifically made for browser based testing, is PhantomJS.

However the latest release of PhantomJS, `v1.9.8` from October 2014, is based on a WebKit from 2011. The next major release (`v2.0.0`) should be out soon and it will be based on a much newer WebKit, via Qt `v5.3` which is used underneath.

Tasks for the day

1. Add unit tests to your `hello-node-js` repository by using `nodeunit`
   - Unit test spec files are placed in a directory called `tests`
   - Each unit test file named as its targeting source file and `_spec.js` suffix
2. Add a task runner task for running the unit tests
   - Make sure that tests can be run after a fresh clone of the repository, followed by `npm i` and `[task runner] test` , for example `grunt test`

Links related to the lecture subject

- Testing with Jasmine
- PhantomJS quick how to
- Video: PhantomJS and Jasmine how to
- Testing Private Functions in JavaScript Modules

Examples for the tasks

1. Unit tests with nodeunit

Install dependencies first:

```
npm i nodeunit --save-dev
```

Create directory for tests and initial empty test file:

```
mkdir tests
touch tests/last-mod_spec.js
```

The test can be written as the following JavaScript, assuming that the `LICENSE` file exists in the repository and its modification date is in October 2014.

```
// last-mod_spec.js

var lastMod = require('../last-mod');

exports.lastMod = function(test){
  test.expect(4);

    // Non existing file
    var non = lastMod('not-to-be-found.js');
    test.strictEqual(non, false, 'False returned when file not existing');

    // Existing file
    var yes = lastMod('LICENSE'); // Date object
    test.equal(yes instanceof Date, true, 'Date object returned when file exists');

    // Correct year and month
    test.equal(yes.getFullYear(), 2014, 'Year is matching');
    test.equal(yes.getMonth(), 9, 'Month is October');

  test.done();
};
```

Run the tests with:

```
node_modules/.bin/nodeunit last-mod_spec.js --reporter verbose
```

2. Task runner for the unit tests

```
npm i grunt-contrib-nodeunit --save-dev
```

Lecture 8. - Automation and continuous integration

> Date 14/10/2014

Automated testing and building services which support Node.js, such as Codeship, Drone.io, Travis CI, and Wercker, can be linked to a GitHub repository with a hook, from where they fetch the latest changes on every `git push` and publish the test/build results.

Tasks for the day

1. Make sure your `hello-node-js` has linting and unit tests, that can be executed with `npm test`
   - Add one of the services mentioned above so that `npm test` is executed every time you push code to the repository
   - Add a badge on the top part of the `README.md` file which shows the current state
2. Find a npm module which has unit tests but is not using automated testing service
   - Create an issue for taking such service in use
   - The issue should include instructions of how to take such a service in use
   - The issue should describe how to add a badge that shows the current state
   - Link the issue in the list below

Issues for adding automated testing

- jukra @ html-compress
- tariel, simple-oauth2
- tuunanen @ object-merge
- HeikkiAlanen @ nomnom
- Oona @ node-trello-slack
- merilainen-metropolia @ json-segment
- markoham @ node-verifier

Links related to the lecture subject

Writing Beautiful JavaScript Tests

Lecture 9. - Package dependencies

> Date 21/10/2014

Due to the rapid cycle of development and changes in the third party modules that are used in a given project, someone should look after those dependencies being up to date. Free services for monitoring those third party modules, such as David, Gemnasium and VersionEye offer email notifications and project badges to be used in `README.md` files, so that project developers and other could see the dependency state, them being outdated or not.

All of the above mentioned services can be used for free and registration can be done with the GitHub account. There are several other similar services, but they might not support `npm` specific dependencies.

Tasks for the day

1. Add dependency version monitoring to your `hello-node-js` repository
   - You should have at least `moment` marked as a dependency in the `package.json` from earlier tasks
   - Add the badge displaying the current status in the top area of `README.md` file
2. Find an open source project which is using dependencies from `npm`
   - Create an issue to that project which requests adding one of those dependency monitoring services
   - The issue should be helpful enough so that the project owner can easily take such service in use
   - Assume that the project owner has no knowledge of these services and be kind
   - Link the issue to the list below
3. Write a note about the differences in version requirement syntax used in `package.json`
   - The text should be in the `README.md` of your `hello-node-js`, under a section called "Version selector madness"
4. Optional: Create a graphical presentation of the `hello-node-js` dependency map
   - Save it as an image, save it in the repository and display in `README.md`

Dependency monitoring issues

- HeikkiAlanen @ node-geocoder
- Oona @ node-trello-slack
- Jukra @ staged-github-files
- merilainen-metropolia @ template

Links related to the lecture subject

- Package.json dependencies done right
- npm-ls - List installed packages
- The semver parser for node (the one npm uses)
- Continuous Updating with VersionEye at code.talks 2014
- More badges, Shields.IO

Lecture 10. - Front end third party dependencies and Code coverage

> Date 28/10/2014

Third assignment is given.

Front end dependencies

While Node.js is a back end JavaScript environment, the dependency management touched in the earlier lecture is equally important for front end dependencies. There dependency managers such as Bower and Component, which are very similar to npm. Many packages that are available in one of them, is also available in the two other.

Tasks for front end dependencies

1. In the project made in the web server lecture
   - Add a library for doing DOM selection and event handler binding by using a dependency manager
   - Use that library for putting the cursor in the first field when the user enter the feedback page
2. Write in `hello-node-js` repository `README.md` file when would you use a front end package manager and when you would add the 3rd party library directly under the project version control

Code coverage of unit tests

Unit testing is usually providing stability to a project, but without knowing how much code the tests are covering, it is difficult to trust the given stability. Code coverage is a method used to describe how much of the code executed is truly tested.

One popular code coverage tool is istanbul, which can be installed with a command:

```
npm i -g istanbul
```

Coverage report can be generated with a command:

```
istanbul cover node_modules/.bin/nodeunit -- .
```

The above command runs `nodeunit .` throught `istanbul` which instruments the source and evaluates the unit tests, resulting a folder called `coverage` that contains the report in several different formats.

The output should be something similar to:

```
=============================================================================
Writing coverage object [hello-node-js/coverage/coverage.json]
Writing coverage reports at [hello-node-js/coverage]
=============================================================================

=========================== Coverage summary ========================
Statements   : 83.08% ( 54/65 )
Branches     : 66.67% ( 8/12 )
Functions    : 85.71% ( 6/7 )
Lines        : 83.08% ( 54/65 )
=============================================================================
```

A full working example with `mocha` unit tests and `istanbul` code coverage can be seen at the `analyze-css` module.

Tasks for code coverage

1. Make sure that the unit tests and the task runner made in the unit testing lecture are working
    - Add code coverage check via command line for the project
    - Configure the command so it can be excuted with `npm run-script coverage`
    - Use coverage report to find out areas where tests are not touching
    - Add more unit tests to achieve at least 95% coverage
2. Push the code coverage report made in the 1st task to GitHub Pages ('gh-pages') branch, so it becomes available at `username.github.io/project-name`
3. Optional: Find a npm module that has unit tests and is running them via task runner
    - Add code coverage command to `package.json`, which can be executed with `npm run-script coverage`
    - Create pull request
    - Link the pull request to the list below

Code coverage task runner pull requests

- HeikkiAlanen @ filesize.js

Links related to the lecture subject

- JavaScript Code Coverage with Istanbul
- GitHub Pages

Lecture 11. - Web performance

> Date 04/11/2014

Web performance is about making things in web preform better. It usually boils down making web pages load faster and use less resources such as CPU and network bandwidth.

Easiest ways to start working towards better performing web sites, is to reduce the amount and the sizes of any files. Concatenation, minification and compression can reduce the size of any assets considerably.

In most cases images are the biggest single file type served, however you will find just one task related them from below. Another recently grown media type is video, but due to the complexity, it is not in the scope of this lecture.

Please see a great presentation from Fluent 2014 conference by Ilya Grigorik, video and slides.

Further more JavaScripts and CSS files are usually minified from their development version, that contain additional information such as usage examples and other comment, are removed along with any unnecessary white space. Some minifiers also change the variable and function names so that they become one or two character long.

Node.js based minifiers such as UglifyJS are clean-css among the most popular and also have task runners available.

Other ways for making better Web performance can be methods such as removing duplicate code, with jsinspect for example.

The tools originally designed for Web performance can be used to benefit a Node.js developer.

Tasks for the day

1. Given that a popular npm module is downloaded several thousand times in a day, evaluate how much bandwidth can be saved on a one day interval if the given JavaScript is minified
   - Find a popular npm module
   - Calculate average download count for a single day based on one week
   - How big is the module and thus how much is being downloaded every day from npm registry
   - How much difference would be with a minified version
   - Write all this briefly in `hello-node-js` repository `README.md`, under a title "Minified npm packages"
2. Take a look at the project called Sitespeed.io
   - Create a similar tool that downloads a web page and measures its weight
   - Evaluate which low level tool would be the most helpul, perhaps `phantomjs` or `request`
   - Write a script that downloads HTML and all JavaScript files of a given URL
   - The script should be added to the `hello-node-js` repository
   - Run with a command `node measure-site.js URL`, or `phantomjs measure-site.js URL`
   - It should then output the file sizes of the given downloaded files
   - Optional: compare original file sizes with ones minified locally
3. Optional: Find an open source web site which is nice, and measure its Web performance
   - The site should have a GitHub repository
   - Create an issue with at least one suggestion for making the web site perform better
   - Link the issue in the list below

Issues for better Web performance

Links related to the lecture subject

- Performance tooling today

- Perf Rocks Tools
- Integrating performance tools into continuous delivery
- Book: High Performance Browser Networking
- npm and front-end packaging
- 600 million downloads from @npmjs in the last 30 days
- Presentation: The web is too slow
- How to Make a Performance Budget

Examples for tasks

2. Download and measure

Once HTML is downloaded, it should be parsed for finding the JavaScript element, namely `script`.

A HTML parsing tool called cheerio is used in the below example, as well as request for downloading the assets:

```
// measure-sizes.js
var request = require('request');
var cheerio = require('cheerio');

var url = 'http://npmjs.org';

request(url, function (error, response, body) {
  if (!error && response.statusCode == 200) {

    console.log(body);
    var $ = cheerio.load(body);
    var scripts = $.find('script');

    console.log(scripts);
  }
});
```

Before running the script, dependencies should be installed:

```
npm i request cheerio
```

Running the script:

```
node measure-sizes.js
```

Should provide something similar to:

```

```

Lecture 12. - More about unit testing and code coverage

> Date 11/11/2014

Continuing with unit tests and code coverage as they are crucial parts for making any given project trustworthy and prone against regressions.

Tasks for the day

1. Find a project that some what small and might even have initial unit test done with `nodeunit`, but other unit testing frameworks are fine too
   - It could be the one you used in task 2 of an earlier leture
   - Extend those unit test or add them, to cover more methods of that module
2. Add code coverage check to the project used in first task
   - `script` property in `package.json` should contain the command which can be run with `npm run-script coverage`
3. Push coverage results of the previous tasks to coveralls.io
   - Either by yourself from your own clone
   - Or write short instructions in an issue of how to do it
4. Optional: Do the optional (third) task from an earlier lecture

Lecture 13. - Security

> Date 18/11/2014

Fourth and last assignment is given.

Security is probably the most important aspect of any software development.

Discussion items

During the lecture, we had a discussion about the security aspects of 3rd party packages. In summarise the following points were brought up when considering what should be checked before taking any 3rd party code in use:

- Validate te code, lint
- Check the origin, who made it, is it public and how active, where is it downloaded from, suspicious web site
- `.exe` suffix
- Does it do what it is supposed to
- Does it do something that it is not supposed to do
- Suspicious instructions, such as using `sudo` or giving access to passwords
- Collects information, surveillance traffic
- Amount of downloads

Tools that could help in checking the above criteria:

- plato
- nsp

Links related to the lecture subject

- Node Security Project resources
- Checking for vulnerable Node.js modules
- Debugging Node.js - How we found memory leaks and slow/infinite loops
- Node.js Security Tips
- Google Chrome - Gradually sunsetting SHA-1
- OWASP Node js Goat Project

Example for creating brute force passwords

... in order to prove how easy it is and why should sensitive data be handled with special care.

```
/**
 * Brute force a password that uses SHA1 hash for encryption
 */
'use strict';

var crypto = require('crypto'),
    util = require('util');

var start = (new Date()).getTime();
var time = function () {
    return (new Date()).getTime() - start;
};

// from https://www.npmjs.org/package/char-range
var range = function(start, stop) {
    var result = [];
    for (var idx = start.charCodeAt(0), end = stop.charCodeAt(0); idx <= end; ++idx) {
        result.push(String.fromCharCode(idx));
    }
    return result;
};

var chars = range('a', 'z').concat(range('A', 'Z'),
    ['å', 'ä', 'ö'], ['Å', 'Ä', 'Ö'],
    range('0', '9'));
//util.puts(util.inspect(chars));

var createHash = function (word) {
    var shasum = crypto.createHash('sha1');
    shasum.update(word);
    return shasum.digest('hex');
};

var maxLength = 3;
var password = 'Hel'; // Password that is looked for
var passHash = createHash(password);

var iterateChars = function(prefix) {
    var len = chars.length;
    for (var i = 0; i < len; ++i) {
        var word = prefix + chars[i];
        var sha1 = createHash(word);
        //util.puts(time() + ' ' + word + ' = ' + sha1);
        if (sha1 === passHash) {
            util.puts('Password found. Time used: ' + time() + ' ms');
            return;
        }
        if (word.length < maxLength) {
            iterateChars(word);
        }
    }
};

iterateChars('');
```

In order to improve the example and its efficiency, the iteration should be done first against all characters in the first index, than second and third, as opposed now iterating whole maximum length with first index being at the first character.

That is why the running time increases when the maximum length is increased, even while the matching password is somewhat short.

Lecture 14. - Database

> Date 25/11/2014

Some projects require the use of database due to the need for persistent data, such as user account information, settings and other similar things that should be kept same between different sessions.

Node.js has several modules that provide bindings to native database libraries, and some written completely in JavaScript.

npm registry itself uses CouchDB, which will be queried in the example below.

```
/**
 * Search latest version of the given keyword
 * matching package
 * registry-search.js
 */
'use strict';

var name = 'node-sass';
var url = 'http://registry.npmjs.org';

var util = require('util');
var nano = require('nano');

if (process.argv.length === 3) {
    name = process.argv.pop();
}

var server = nano(url);
var registry = server.use(name);

//util.puts(util.inspect(registry));

registry.get('latest', function(err, body) {
    if (err) {
        util.error(err);
    }
    //util.puts(util.inspect(body));

    util.puts('Latest version of ' + name + ' is ' + body.version);
});
```

Install dependencies:

```
npm install nano
```

Run via command line, by searching the default package name defined in line 8:

```
node registry-search.js
```

Should provide an output similar to:

```
Latest version of node-sass is 1.2.3
```

Running with a package name defined via command line:

```
node registry-search.js nano
```

Outputs:

```
Latest version of nano is 6.0.2
```

Links related to the lecture subject

- npm - The JavaScript Package Registry
- nano - minimalistic couchdb driver for node.js
- CouchDB is a database that completely embraces the web

Lecture 15. - Heroku and other free hosting services

> Date 02/12/2014

Task for today

Create a blog that is based on Express and publish it to Heroku/OpenShift or similar.

- Create a free account in the given hosting provider and setup SSH keys
- Create a repository at GitHub that is used for the development of the blog
- Publish the blog by pushing it to the provider, such as `git push heroku master`
- Profit...?

While doing the task, find out whether to use existing blogging tool or create your own with Express.

Links related to the lecture subject

- Node.js High Availability at Box
- Video: What does an Open Source Microsoft Web Platform look like?

Examples for the task

Plain Express initiated with its generator

http://expressjs.com/starter/generator.html

Source code https://github.com/expressjs/generator

```
npm i -g express-generator
express lorenzos-sweets
cd lorenzos-sweets
npm i
DEBUG=lorenzos-sweets ./bin/www
```

Now open the browser at `http://localhost:3000` and you should see simple start page.

Ghost

https://ghost.org/ http://support.ghost.org/installation/

Download the latest release from https://github.com/TryGhost/Ghost/releases, unzip it and rename the folder as `lorenzos-sweets`.

```
npm i
npm start
```

Now open the browser at `http://localhost:2368` and you should see something, but there is most likely an error due to missing database configuration.

Lecture 16. - Examples for previous tasks

> Date 09/12/2014

Links related to the lecture subject

Lecture 17. - Feedback on the projects

> Date 16/12/2014

What to do next?

Links related to the lecture subject

- Node Forward
- Nodebugme will randomly give you an open Node issue to triage
- Node.js Advisory Board
- Fragmentation
- Top 10 Mistakes Node.js Developers Make

First assignment

> Given in 16th Sep 2014, to be returned by 7th Oct 2014

Come up with an idea for a project that you can use in your daily/weekly life, in order to shorten the time used for repetitive tasks.

It could be renaming large quantities of files, resizing or applying filters on images, or something which you might do from time to time.

Once you have the idea, create a GitHub repository for it:

- with a distinctive name
- license for open source
- introduction and feature/release plan for Autumn 2014 in the `README.md`
- use Semantic Versioning

The project can start as a command line tool, but later on should be able to be used as a Node.js module inside someone else's application.

Usually a command line tool accepts certain parameters and options, thus there should be at least the following two:

- `-h` and `--help` - Help and usage output
- `-V` and `--version` - Version, license and copyright information

There should be a `package.json` file describing the properties of this project. It should contain all the required fields as per npm requirements. Also consider the optional fields which will make it easier to use your project.

The amount of downloaded packages from npm are increasing every day and while the amount of packages increases, the package owners should take care of including only the `files` which are necessary in their packages. Make sure this is done in your project.

This assignment will be marked as complete when the above requirements have been filled, before the dead line, which it when the next assignment is given (7th October 2014) and all the relevant code is available at GitHub.

Please create a pull request for adding a link to your project in the assignment projects list before the dead line.

Second assignment

> Given in 7th Oct 2014, to be returned by 28th Oct 2014

Extend your existing project which was created in the first assignment.

Given that your application is triggered via command line as required in the first assignment, now separate the functionality so that it can be used programmatically.

The entry point of the application, as described in `package.json` with the `main` property, this script will be able to use the separate custom modules by requiring them. In a same way these custom modules are used for unit tests.

This assignment has the following tasks:

- Before starting, tag the project with a release `v0.1.0` which should fulfill the requirements of the first assignment
- Add a task runner, such as `Grunt` or `gulp` to your project for linting and for running unit tests
- Add unit testing, by using a suitable library of your choosing
- Both linting and unit tests should be able to be run with a command `npm test`
- These tests are executed automatically on every push by a continuous integration service, such as Travis CI or Wercker
- The `README.md` should display a badge showing the current state of those tests
- Start planning how to create a task runner plugin of your project, so other could use it via task runner, and write about in `README.md` and also include it in a release plan
- Once the assignment is ready to be returned, tag is as a new release `v0.2.0`

Example projects that have both command line and programmatic usage available. Please note that much more exists and you are free to take example from anywhere else. The following two have pretty much the similar structure:

- CSScomb
- ESLint
- Sakugawa

Third assignment

> Given in 28th Oct 2014, to be returned by 18th Nov 2014

Extend your existing project which was started in the first assignment and continued further in the second assignment.

Use the automated tests done in the second assignment and add code coverage testing on them. Push the results to Coveralls.io and display the results in your projects `README.md`.

Once the tests and coverage results appear, increase the test coverage percentage by adding unit tests to your project. The code coverage should be over 90% when returning this assignment.

This assignment is evaluated by a successful installation of Coveralls.io service to the given project. The history shown at Coveralls.io for the project should show an increase in the test coverage percentage.

Once the assignment is considered to be complete, tag it with `v0.3.0`.

Fourth assignment

> Given in 18th Nov 2014, to be returned by 9th Dec 2014

Extend your existing main project which was originally started in the first assignment, continued in the second and the third assignments.

Now that the project can be executed via command line and as a part of another application, it should be able to be executed via task runner.

A successful completion of this assignment is done when the following points can be evaluated as completed:

- Choose a task runner, for example gulp or Grunt
- Create a new repository, named after the task runner and your initial main project, for example `gulp-image-thumbnailer`, in case the main project was called `image-thumbnailer`
- Follow the best practises for the given task runner and its plugin guidelines when initialising the plugin, such as how to configure files
- Add your original project as a git url dependency, so it can be used
- The task runner project should contain a working example of how the plugin is used, for example in a `gulpfile.js` and via `devDependencies`
- By running `[task runner name]` in the project repository, after `npm i` should result success
- The initial project should accomplish the behaviour it was initially planned.

Please note that due to the requirements, the project done in previous assignments, should be in a working state and thus usable for the task runner.

Elevate the release status of the main project to `1.0.0`, by making a tag and use that version tree for the task runner project. Once the task runner is to be considered ready, it should also be tagged as `1.0.0`.

Assignment projects

This page should contain links to the projects that are made in order to complete the four assignments.

Assignments

- First assignment description
- Second assignment description
- Third assignment description
- Fourth assignment description

Projects

Write in the format `* [First name Last name initial](link-to-repository "Project title")`, for example `* [Jukka P](https://github.com/paazmaya/matsumura-rohai "Translate PO files with the help of Microsoft Translate API")`

- Heikki A
  - grunt-hal-image-optimizer
- Jarkko T
  - grunt-camelton
- Joonas M
  - grunt-trelloler
- Jukka R
  - gulp-web-tweak-n-optimize
- Marko H
  - gulp-carpo-logger
- Markus I
  - gulp-twitter-data
- Maxim D
  -
- Onni A
  -
- Oona L
  -
- Oscar L
  -

Presentations

Each presentation should be about 15 minutes, going through briefly one tool or functionality related to Node.js.

Please mark your name next to the subject you wish to present. In case the "thing" you wish to present is not in the list, feel free to add it. Each lesson should have two to three presentations.

List of possible subjects, with approximate chronological order and grouped by subject:

- GitHub forks and pull request - 2/9/2014 jhaap
- npm - 9/9/2014 Oona npm basics
  - Name expansions of `npm`
- JSLint & JSHint (16.9.2014/Maxim Dolgobrod)
- ESLint (16.9.2014/Heikki Alanen)
- Connect
- Express (Sopheak Kong - 23.9.2014)
- Grunt) (30.9.2014 / Oscar Lemström)
- Gulp (30.9.2014 / Joonas Meriläinen)
- Broccoli
- Jasmine (7.10.2014 - Tuukka Laitinen)
- Mocha (7.10.2014 - Toomas Tero)
- Qunit (7.10.2014 - Toomas Tero)
- nodeunit
- PhantomJS
- Travis CI
- Wercker
- Gemnasium
- VersionEye
- Bower Markus, Marko 28.10.2014
- Component
- UglifyJS (Onni Aaltonen - 11.11.2014)
- CSSmin (Jukka Rautanen - 4.11.2014)
- Sitespeed.io (Jarkko Tuunanen - 4.11.2014)
- Coveralls.io (Jukka Rautanen - 4.11.2014)
- CouchDB
- MongoDB / Mongoose Markus, Marko 18.11.2014.
- Crypto (Oscar Lemström 18.11.2014)
- npm registry (Oona - 9.2.2014)

Coveralls.io quiz

A quiz about Coveralls.io

Quiz

Coveralls.io is a service that..

Question 1 of 1

	`true`	`false`
`validates your js files`	☐	☐
`gives you test coverage history`	☐	☐
`is free for open source`	☐	☐

	`true`	`false`
`validates your js files`	☐	☑
`gives you test coverage history`	☑	☐
`is free for open source`	☑	☐

Quiz

Coveralls.io provides..

Question 1 of 1

	`true`	`false`
`only general repository coverage statistics`	☐	☐
`individual file coverage reports`	☐	☐
`line by line coverage reports`	☐	☐

	`true`	`false`
`only general repository coverage statistics`	☐	☑
`individual file coverage reports`	☑	☐
`line by line coverage reports`	☑	☐

Quiz

You can use Coveralls.io for example with..

Question 1 of 1

	`true`	`false`

`Mocha + JSCoverage`	☐	☐
`Istanbul + Nodeunit`	☐	☐
`lab`	☐	☐

	`true`	`false`
`Mocha + JSCoverage`	☑	☐
`Istanbul + Nodeunit`	☑	☐
`lab`	☑	☐

CSSmin quiz

A quiz about CSSmin.

Quiz

CSSmin can be used to

Question 1 of 1

	`true`	`false`
`make your CSS files smaller`	☐	☐
`unminify your CSS files`	☐	☐
`minimize specific files or all css files`	☐	☐

	`true`	`false`
`make your CSS files smaller`	✓	☐
`unminify your CSS files`	☐	✓
`minimize specific files or all css files`	✓	☐

Quiz

By minimizing your CSS files

Question 1 of 1

	`true`	`false`
`your site loads slower`	☐	☐
`your site is more secure`	☐	☐
`your site looks cooler`	☐	☐

	`true`	`false`
`your site loads slower`	☐	✓
`your site is more secure`	☐	✓
`your site looks cooler`	☐	✓

Quiz

You can use CSSmin with

Question 1 of 1

	`true`	`false`

`Gulp`	☐	☐
`Grunt`	☐	☐
`Node.js`	☐	☐

	`true`	`false`
`Gulp`	☑	☐
`Grunt`	☑	☐
`Node.js`	☑	☐

A quiz about ESLint

Quiz

ESLint is used to

Question 1 of 1

	yes	no
`compile JavaScript`	☐	☐
`check style issues`	☐	☐
`to detect unused variables`	☐	☐
`measure code complexity`	☐	☐
`to detect division by zero`	☐	☐
`enforce coding conventions`	☐	☐
`measure test coverage`	☐	☐

	yes	no
`compile JavaScript`	☐	☑
`check style issues`	☑	☐
`to detect unused variables`	☑	☐
`measure code complexity`	☐	☑
`to detect division by zero`	☑	☐
`enforce coding conventions`	☑	☐
`measure test coverage`	☐	☑

Quiz

How do you install ESLint globally?

Question 1 of 1

	yes	no
`install eslint`	☐	☐
`npm install eslint`	☐	☐
`npm install eslint -g`	☐	☐

	yes	no
`install eslint`	☐	☑

`npm install eslint`	☐	☑
`npm install eslint -g`	☑	☐

Which configuration options should be selected, if you want to set

- as errors: missing semicolon, empty block statements and use of constant expressions in conditions
- as warnings: whitespace at the end of line, mixed spaces and tabs for indentation (Smart Tabs off) and disallow labels that share a name with a variable
- other possible options off

	yes	no
`"semi": 1,`	☐	☑
`"semicolon": 2,`	☐	☑
`"semi": 2,`	☑	☐
`"no-empty": 2,`	☑	☐
`"no-empty": 1`	☐	☑
`"no-empty-class": 1`	☐	☑
`"no-constant-condition": 1,`	☐	☑
`"no-constant-condition": 2,`	☑	☐
`"no-trailing-spaces": 1,`	☑	☐
`"no-trailing-spaces": 2,`	☐	☑
`"no-plusplus": 2,`	☐	☑
`"no-plusplus": 1,`	☐	☑
`"no-mixed-spaces-and-tabs": [1, false],`	☑	☐
`"no-mixed-spaces-and-tabs": [1, true],`	☐	☑
`"no-mixed-spaces-and-tabs": [2, false],`	☐	☑
`"no-label-var": 1,`	☑	☐
`"no-label-var": 2,`	☐	☑

> Hey, let's be careful out there!

Gulp

A quiz about JavaScript task runner Gulp.js

Quiz

Gulp.js basics

Question 1 of 1

	`true`	`false`
`Is command-line tool`	☐	☐
`Makes website faster`	☐	☐
`Runs tasks`	☐	☐
`Uses Node.js stream API`	☐	☐
`Is website builder`	☐	☐

	`true`	`false`
`Is command-line tool`	☑	☐
`Makes website faster`	☐	☑
`Runs tasks`	☑	☐
`Uses Node.js stream API`	☑	☐
`Is website builder`	☐	☑

Quiz

Gulpfile

Question 1 of 1

	`true`	`false`
`Usually tasks are in file named gulptasks.js`	☐	☐
`Tasks are written inside 'task' functions`	☐	☐
`Gulpfile is like JSON styled file`	☐	☐
`Tasks can be only Gulp plugins`	☐	☐

	`true`	`false`
`Usually tasks are in file named gulptasks.js`	☐	☑
`Tasks are written inside 'task' functions`	☑	☐
`Gulpfile is like JSON styled file`	☐	☑

Tasks can be only Gulp plugins	☐	✓

Quiz

Using Gulp.js

Question 1 of 1

	true	false
Plugins can't be downloaded via npm	☐	☐
Gulp.js must be installed globally and locally	☐	☐
Running task 'gulp' will run first task on the gulpfile	☐	☐
It is possible to run multiple tasks at the same time	☐	☐

	true	false
Plugins can't be downloaded via npm	☐	✓
Gulp.js must be installed globally and locally	✓	☐
Running task 'gulp' will run first task on the gulpfile	☐	✓
It is possible to run multiple tasks at the same time	✓	☐

A quiz about Javascript scope

Which statemens apply to javascript scope

```
var result = 4;

function calculate(x, y) {
  var x = 8;
  var result = x + y;
}

calculate(5, 10);

console.log(result);
```

	true	false
`x in result is 5`	☐	☑
`result in calculate function is 18`	☑	☐
`output of console log is 18`	☐	☑

> Sorry... Try again

Which statemens apply to javascript scope

```
function setUsername(name) {
  username = name;
  console.log("1:" + username);
}

setUsernama("Kate");

console.log("2:" + username);
```

	yes	no
`output of console log 1 is undefined`	☐	☑
`output of console log 2 is Kate`	☑	☐

> Sorry... Try again

Which statemens apply to javascript scope

```
var todos = [];

function addTodo(todo) {
  todos.push(todo);
}
addTodo("Make coffee");

console.log("1:" + todos.length);

addTodo("Buy more coffee");
addTodo("Outsource node.js tasks to Chine");

console.log("2:" + todos[2]);
```

	yes	no
`output of console log 1 is 1`	✓	
`output of console log 2 is buy more coffee`		✓

> Sorry... Try again

Which statemens apply to javascript scope

	yes	no
`The lifetime of a JavaScript variable starts when it is declared.`	✓	
`Local variables stay in scope after the function is completed.`		✓
`Global variables are deleted when you close the page.`	✓	

> Sorry... Try again

JavaScript type quiz

A quiz about JavaScript types and their conversion.

Quiz

Select the correct evaluation result

Question 1 of 1

	`true`	`false`
`0 == false`	☐	☐
`0 === false`	☐	☐
`1 == true`	☐	☐
`'1' == false`	☐	☐

	`true`	`false`
`0 == false`	☑	☐
`0 === false`	☐	☑
`1 == true`	☑	☐
`'1' == false`	☐	☑

Quiz

Choose the returned value for each `typeof` operator use cases. For example the first one would be executed as `typeof "Hello"` and would return a `"string"`.

Question 1 of 1

`typeof ...`	`"object"`	`"number"`	`"array"`	`"string"`	`"function"`	`"undefined"`	`"boolean`
`"Hello"`	☐	☐	☐	☐	☐	☐	☐
`true`	☐	☐	☐	☐	☐	☐	☐
`108`	☐	☐	☐	☐	☐	☐	☐
`[0, 1, 2]`	☐	☐	☐	☐	☐	☐	☐
`{}`	☐	☐	☐	☐	☐	☐	☐
`new Date()`	☐	☐	☐	☐	☐	☐	☐
`undefined`	☐	☐	☐	☐	☐	☐	☐
`"true"`	☐	☐	☐	☐	☐	☐	☐
`12.8`	☐	☐	☐	☐	☐	☐	☐
`alert`	☐	☐	☐	☐	☐	☐	☐

`typeof` ...	`"object"`	`"number"`	`"array"`	`"string"`	`"function"`	`"undefined"`	`"boolean`
`"Hello"`	☐	☐	☐	☑	☐	☐	☐
`true`	☐	☐	☐	☐	☐	☐	☑
`108`	☐	☑	☐	☐	☐	☐	☐
`[0, 1, 2]`	☑	☐	☐	☐	☐	☐	☐
`{}`	☑	☐	☐	☐	☐	☐	☐
`new Date()`	☑	☐	☐	☐	☐	☐	☐
`undefined`	☐	☐	☐	☐	☐	☑	☐
`"true"`	☐	☐	☐	☑	☐	☐	☐
`12.8`	☐	☑	☐	☐	☐	☐	☐
`alert`	☐	☐	☐	☐	☑	☐	☐

What are the output values of the following expressions?

`1 + 2`

- ☐ `12`
- ☑ `3`

`"1" + 2`

- ☐ `12`
- ☑ `"12"`
- ☐ `3`
- ☐ `"3"`

`+"1" + 2`

- ☐ `12`
- ☐ `"12"`
- ☑ `3`
- ☐ `"3"`

`1 + "2"`

- ☐ `12`
- ☑ `"12"`
- ☐ `3`
- ☐ `"3"`

`1+ +"2"`

- ☐ `12`
- ☐ `"12"`
- ☑ `3`
- ☐ `"3"`

> String always wins! ...unless it is turned to a number first.

JSLint and JSHint quiz

A quiz about JSLint and JSHint.

Quiz

JavaScript code linters are used to

Question 1 of 1

	`true`	`false`
`execute the code to analyzes it for pontential errors`	☐	☐
`find JavaScript incompatibility issues in older browsers`	☐	☐
`improve code quality`	☐	☐
`check code intendation`	☐	☐
`check for outdated third party dependencies`	☐	☐
`enforce common code style`	☐	☐

	`true`	`false`
`execute the code to analyzes it for pontential errors`	☐	☑
`find JavaScript incompatibility issues in older browsers`	☑	☐
`improve code quality`	☑	☐
`check code intendation`	☑	☐
`check for outdated third party dependencies`	☐	☑
`enforce common code style`	☑	☐

Quiz

Which of these statements are true or false about JSLint?

Question 1 of 1

	`true`	`false`
`JSLint can detect memory leaks.`	☐	☐
`JSLint can operate on JavaScript as well as on JSON texts.`	☐	☐
`JSLint expects that all variables and functions are declared before they are used or invoked.`	☐	☐
`JSLint does not expect to see "new Object" and accepts only "{}".`	☐	☐
`JSLint doesn't expect that every statement to be followed by ";".`	☐	☐
`JSLint performs a flow analysis to determine that variables are assigned values before used.`	☐	☐

	true	false
JSLint can detect memory leaks.	☐	☑
JSLint can operate on JavaScript as well as on JSON texts.	☑	☐
JSLint expects that all variables and functions are declared before they are used or invoked.	☑	☐
JSLint does not expect to see "new Object" and accepts only "{}".	☑	☐
JSLint doesn't expect that every statement to be followed by ";".	☐	☑
JSLint performs a flow analysis to determine that variables are assigned values before used.	☐	☑

Quiz

Which of these statements are true or false about JSHint?

Question 1 of 1

	true	false
JSHint does not come with a default set of warnings.	☐	☐
JSHint has options for checking the complexity of a function.	☐	☐
JSHint options can be save in "package.json".	☐	☐
JSHint has only enforcing type of options.	☐	☐
JSHint "evil" flag is used to warn about the use of eval.	☐	☐
JSHint warns when you omit break or return statements within switch statements.	☐	☐

	true	false
JSHint does not come with a default set of warnings.	☐	☑
JSHint has options for checking the complexity of a function.	☑	☐
JSHint options can be save in "package.json".	☑	☐
JSHint has only enforcing type of options.	☐	☑
JSHint "evil" flag is used to warn about the use of eval.	☑	☐
JSHint warns when you omit break or return statements within switch statements.	☑	☐

A quiz about MongoDB

Quiz

Which characteristic apply to mongodb

Question 1 of 7

	true	false
`MongoDB is relational database`	☐	☐
`Key–Value Store`	☐	☐
`Document Database`	☐	☐
`Scalable`	☐	☐
`XML-style Document`	☐	☐
`JSON-style Document`	☐	☐

	true	false
`MongoDB is relational database`	☐	✓
`Key–Value Store`	☐	✓
`Document Database`	✓	☐
`Scalable`	✓	☐
`XML-style Document`	☐	✓
`JSON-style Document`	✓	☐

Question 2 of 7

How do you install MongoDB?

How do you install MongoDB?

Question 3 of 7

	yes	no
`npm install mongodb`	☐	☐
`npm install mongodb -g`	☐	☐
`apt-get install mongodb`	☐	☐

	yes	no
`npm install mongodb`	☐	✓
`npm install mongodb -g`	☐	✓
`apt-get install mongodb`	✓	☐

Question 4 of 7

How do you insert data into a MongoDB?

How do you insert data into a MongoDB?

Question 5 of 7

	yes	no
`db.collection.insert({foo:'bar'})`	☐	☐
`INSERT INTO collection (foo) VALUES ('bar')`	☐	☐
`put 'collection', 'row1', 'vi:foo', 'bar'`	☐	☐

	yes	no
`db.collection.insert({foo:'bar'})`	✓	☐
`INSERT INTO collection (foo) VALUES ('bar')`	☐	✓
`put 'collection', 'row1', 'vi:foo', 'bar'`	☐	✓

Question 6 of 7

How do you get all data from a MongoDB?

How do you get all data from a MongoDB?

Question 7 of 7

	yes	no
`SELECT * FROM collection`	☐	☐
`db.collection.find()`	☐	☐
`scan 'collection'`	☐	☐

	yes	no
`SELECT * FROM collection`	☐	✓
`db.collection.find()`	✓	☐
`scan 'collection'`	☐	✓

A quiz about npm commands

Quiz

Which command creates the package.json file interactively?

Question 1 of 1

	yes	no
`npm install`	☐	☐
`npm init`	☐	☐
`npm package.json`	☐	☐

	yes	no
`npm install`	☐	✓
`npm init`	✓	☐
`npm package.json`	☐	✓

Quiz

How do you install a package and save it as a dependency in the package.json file?

Question 1 of 1

	yes	no
`npm install <package name> --save`	☐	☐
`npm install <package name> --s`	☐	☐
`npm install <package name> -S`	☐	☐

	yes	no
`npm install <package name> --save`	✓	☐
`npm install <package name> --s`	☐	✓
`npm install <package name> -S`	✓	☐

Quiz

How do you list packages?

Question 1 of 1

	yes	no
`npm list`	☐	☐

`npm ls`	☐	☐
`npm -l`	☐	☐
`npm list -g`	☐	☐

	yes	no
`npm list`	☑	☐
`npm ls`	☑	☐
`npm -l`	☐	☑
`npm list -g`	☑	☐

A quiz about npm registry

How many official public NPM registries are there?

	yes	no
one	✓	☐
two	☐	✓
many	☐	✓

An NPM mirror...

	yes	no
is a read-only copy of the main NPM registry	✓	☐
lets you install any module that exists in NPM	✓	☐
lets you publish updates and new packages	☐	✓

There are at least three ways to use a mirror. How do you specify the registry when installing an NPM package?

	yes	no
`npm install <package name> --registry http://registry.npmjs.org`	✓	☐
`npm <package name> --registry http://registry.npmjs.org`	☐	✓

Sitespeed.io

Quiz

Sitespeed.io...

Question 1 of 1

	`true`	`false`
`makes my website faster if I install it`	☐	☐
`analyses and checks websites against performance best practices`	☐	☐
`sends me weekly reports on my website's performance`	☐	☐
`can be integrated with some continuous-integration tools`	☐	☐
`can only be used from the command-line`	☐	☐

	`true`	`false`
`makes my website faster if I install it`	☐	☑
`analyses and checks websites against performance best practices`	☑	☐
`sends me weekly reports on my website's performance`	☐	☑
`can be integrated with some continuous-integration tools`	☑	☐
`can only be used from the command-line`	☑	☐

Quiz

Sitespeed.io rules...

Question 1 of 1

	`true`	`false`
`are based on YSlow`	☐	☐
`are strict and hurt my feelings`	☐	☐
`cannot be created by third-party developers`	☐	☐

	`true`	`false`
`are based on YSlow`	☑	☐
`are strict and hurt my feelings`	☑	☐
`cannot be created by third-party developers`	☐	☑

Quiz

Sitespeed.io v3.x

Question 1 of 1

	`true`	`false`
`API is fully backwards-compatible with v2.x`	☐	☐
`does not depend on Java`	☐	☐
`is the first NodeJS based version`	☐	☐
`works in Windows environments`	☐	☐

	`true`	`false`
`API is fully backwards-compatible with v2.x`	☐	☑
`does not depend on Java`	☐	☑
`is the first NodeJS based version`	☑	☐
`works in Windows environments`	☐	☑

Travis CI

A quiz about continuous integration tool Travis CI

Quiz

What is Travis CI?

Question 1 of 1

	`true`	`false`
`Integration tool only for Node.js projects`	☐	☐
`It fetch repository changes and runs test/build scripts`	☐	☐
`It only works with GitHub`	☐	☐

	`true`	`false`
`Integration tool only for Node.js projects`	☐	☑
`It fetch repository changes and runs test/build scripts`	☑	☐
`It only works with GitHub`	☑	☐

Quiz

How to use it?

Question 1 of 1

	`ture`	`false`
`Simply activate your repos on Travis CI, and integration is done`	☐	☐
`Insert configurations into travis.json file`	☐	☐
`You must insert badge to readme.md to get Travis CI working`	☐	☐

	`ture`	`false`
`Simply activate your repos on Travis CI, and integration is done`	☑	☐
`Insert configurations into travis.json file`	☐	☑
`You must insert badge to readme.md to get Travis CI working`	☐	☑

Quiz

What is the benefit of using Travis CI?

Question 1 of 1

	`ture`	`false`

It will automatically run tests and notify if they fail	☐	☐
It will fix bugs	☐	☐
It can save time when building apps	☐	☐

	ture	false
It will automatically run tests and notify if they fail	☑	☐
It will fix bugs	☐	☑
It can save time when building apps	☑	☐

Student participation matrix

The tables below contain links to the relevant repositories, projects and pieces of code, which can be used to determine whether a certain task has been completed.

In order to make it easy for both the student and the lecturer, these tables are used to show the current state for each student.

The tables are prefilled with the GitHub usernames of those who have so far participated in this repository by commenting or creating issues and pull requests.

Presentations

This table is for marking if a given presentation related task has been done. Each presentation includes presenting it, linking the presentation material in the presentations list, and finally creating a quiz of the presentation subject.

GitHub username	Presented	Material linked	Quiz done
HeikkiAlanen	☑	☑	☑
Markoham	☑	☑	☐
Oona	☑	☑	☑
SkyFire-	☑	☑	☐
jhaap	☑	☐	☐
jukra	☑	☑	☑
merilainen-metropolia	☑	☑	☑
mpiivonen	☑	☑	☐
olemstrom	☑	☑	☐
rafuke	☐	☐	☐
tariel	☑	☑	☑
tomter	☑	☑	☐
tuukkalai	☐	☐	☐
tuunanen	☑	☑	☑
onnia	☐	☐	☐

Lecture 2 - GitHub forks and pull requests

GitHub username	Task 1-2	Task 3-4	Task 5, 7
HeikkiAlanen	x	x	x
Markoham	x	x	x
Oona	x	x	
SkyFire-			

jhaap			
jukra	x	x	x
merilainen-metropolia	x	x	x
mpiivonen	x	x	x
olemstrom	x	x	x
rafuke			
tariel			
tuukkalai			
tuunanen	x	x	
onnia			

Lecture 3 - Modules and npm

GitHub username	Task 1	Task 2	Task 3	Task 4	Task 5-6
HeikkiAlanen	x	x	x	x	x
Markoham	x	x	x	x	
Oona		x	x	x	x
SkyFire-					
jhaap					
jukra	x	x	x	x	x
merilainen-metropolia	x	x	x		
mpiivonen	x	x	x	x	
olemstrom	x	x	x	x	
rafuke					
tariel			x	x	
tuukkalai					
tuunanen	x	x	x		
onnia	x	x	x		

Lecture 4 - Code validation, conventions, linting

GitHub username	Task 1	Task 2
HeikkiAlanen	x	x
Markoham	x	x
Oona	x	x
SkyFire-		
jhaap		
jukra	x	x
merilainen-metropolia	x	x
mpiivonen	x	x

olemstrom	x	x
rafuke		
tariel		x
tuukkalai		
tuunanen	x	x
onnia		

Lecture 5 - HTTP, Connect, Express

GitHub username	Task
HeikkiAlanen	x
Markoham	x
Oona	x
SkyFire-	
jhaap	
jukra	x
merilainen-metropolia	x
mpiivonen	x
olemstrom	x
rafuke	
tariel	x
tuukkalai	
tuunanen	x
onnia	x

Lecture 6 - Common task runners

GitHub username	Task 1	Task 2
HeikkiAlanen	x	x
Markoham	x	x
Oona	x	x
SkyFire-		
jhaap		
jukra	x	x
merilainen-metropolia	x	x
mpiivonen	x	x
olemstrom	x	x
rafuke		
tariel	x	x
tuukkalai		

tuunanen		
onnia	x	

Lecture 7 - Unit testing, Jasmine, PhantomJS

GitHub username	Task 1-2
HeikkiAlanen	x
Markoham	x
Oona	x
SkyFire-	
jhaap	
jukra	x
merilainen-metropolia	x
mpiivonen	☑ (https://github.com/mpiivonen/hello-node-js/tree/Lecture-7-Task-1)
olemstrom	x
rafuke	
tariel	x
tuukkalai	
tuunanen	x
onnia	

Lecture 8 - Automation and continuous integration

GitHub username	Task 1	Task 2
HeikkiAlanen	x	x
Markoham	x	x
Oona	x	x
SkyFire-		
jhaap		
jukra	x	x
merilainen-metropolia	x	x
mpiivonen	x	x
olemstrom		
rafuke		
tariel	x	x
tuukkalai		
tuunanen	x	x
onnia		

Lecture 9 - Package dependencies

GitHub username	Task 1	Task 2	Task 3
HeikkiAlanen	x	x	x
Markoham	x		
Oona	x	x	x
SkyFire-			
jhaap			
jukra	x	x	x
merilainen-metropolia	x	x	x
mpiivonen	x	x	
olemstrom			
rafuke			
tariel	x	x	x
tuukkalai			
tuunanen			
onnia			

Lecture 10 - Front end third party dependencies and Code coverage

GitHub username	Front end 1	Front end 2	Code cover 1	Code cover 2
HeikkiAlanen	X	X	X	X
Markoham				
Oona				
SkyFire-				
jhaap				
jukra	X	X	X	X
merilainen-metropolia	x	x	x	x
mpiivonen			x	x
olemstrom	x	x	x	x
rafuke				
tariel	x		x	
tuukkalai				
tuunanen	x	x		
onnia				

Lecture 11 - Web performance

GitHub username	Task 1	Task 2
HeikkiAlanen	X	X
Markoham		

Oona	x	
SkyFire-		
jhaap		
jukra	X	X
merilainen-metropolia	x	x
mpiivonen	x	x
olemstrom	x	x
rafuke		
tariel		
tuukkalai		
tuunanen	x	x
onnia		

Lecture 12 - More about unit testing and code coverage

GitHub username	Task 1	Task 2	Task 3
HeikkiAlanen	X	X	X
Markoham			
Oona			
SkyFire-			
jhaap			
jukra	X	X	X
merilainen-metropolia	x	x	x
mpiivonen			
olemstrom			
rafuke			
tariel			
tuukkalai			
tuunanen			
onnia			