

# Kennismaking NodeJS

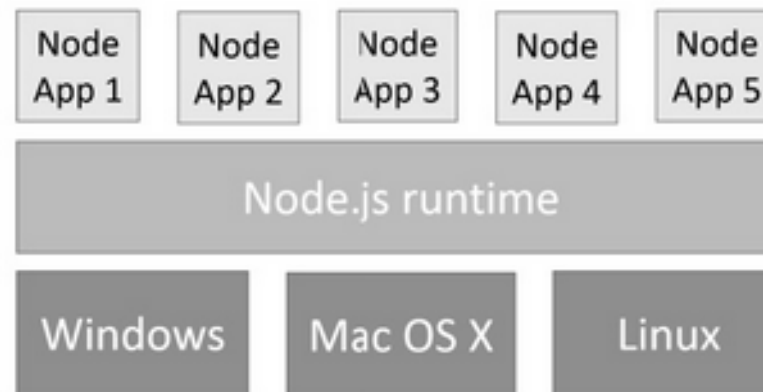


# Kenmerken van NodeJS

- Javascript kan server-sided werken (= losgetrokken uit de browser)
- Maakt gebruik van event-driven architectuur (= code die reageert op events d.m.v. callback functies => gevolg: goede performance, uitvoering van een programma wordt nooit geblokkeerd omdat het programma wacht op een berekening. Callback functie wordt uitgevoerd als het resultaat van de berekening er is.
- Gemaakt voor het bouwen van snelle, schaalbare werkapplicaties
- Er is geen andere webserver nodig (apache, IIS, ...)

# Kenmerken van NodeJS

- NodeJS is geschreven voor JavaScript
- NodeJS zelf is geschreven in C++
- Toegankelijk voor Windows, Linux en Mac
- Het moet apart worden geïnstalleerd
- Alle Node-toepassingen zelf zijn geschreven in JavaScript en hoeven dus maar één keer te worden geschreven. Nadien draaien ze op alle systemen



# Kenmerken van NodeJS

- Geen DOM beschikbaar, wel requests & responses
- In plaats van DOM zal je vaak `console.log()`-meldingen schrijven
- Een applicatie die data retourneert, zal (bijna) altijd JSON-formaat gebruiken
- MEAN-stack:
  - M( MongoDB): database
  - A (Angular): framework voor de client-side toepassing
  - E (Express): Module die wordt ingezet om webserver, routing en API te maken (server-side)
  - N (NodeJS): De motor voor M & E (server-side)

# Waarom NodeJS gebruiken

- Eén programmeertaal (JavaScript in de client en op de server)
- Code opnieuw gebruiken (server-side)
- Snellere ontwikkeltijd (NodeJS-apps moeten niet gecompileerd worden)
- Ondersteuning van de community

<https://nodejs.org/en>

# Development omgeving

- Besturingssystemen
  - Windows 8.1 of Windows 10
  - Mac OS X 10.10+
  - Linux
- Editor
  - WebStorm
  - Visual studio (Code)
  - Atom
  - ...

# Development omgeving



## Node.js heeft geen interface

Onthoud: Node.js heeft geen user interface. Het draait puur om code. Het opsporen van fouten (debuggen) kan daarom lastig zijn. Want als een programma een fout bevat, wordt het afgebroken zodra de fout optreedt en kan Node.js alleen een foutmelding in de console schrijven. Hopelijk wordt u daar wat wijzer van, maar vaak is niet meer te achterhalen dan het regelnummer waar de fout is opgetreden. Een visuele debugger waarbij u breekpunten kunt zetten, variabelen en objecten kunt inspecteren en stapsgewijs door de code kunt lopen is dan handig. Daar gaan we in het volgende hoofdstuk op in.

# NodeJS testen

- De installatie van node controleren via REPL:
  - Open een opdrachtvenster / terminal en typ node
  - Je komt nu in de REPL interface (Read, Evaluate, Print & Loop) waarin rechtstereks JavaScript gebruikt wordt
  - Voorbeeld:
  - Afsluiten kan door 2x op ctrl+c te drukken



The screenshot shows a macOS terminal window titled "PeterKassenaar — B". The active tab is "Black & White — node", with the word "node" circled in black. The terminal prompt is "MacBook-Pro:~ PeterKassenaar node". Three large black arrows point to the input lines: "> 2 + 2", "> var x = 'Hello World'", and "> x". The output shows "4" for the first command, "undefined" for the second, and "'Hello World'" for the third. The prompt ">" is followed by a cursor.

```
MacBook-Pro:~ PeterKassenaar node
> 2 + 2
4
> var x = 'Hello World'
undefined
> x
'Hello World'
>
```



# Hello World in Node.js

- Download de voorbeelden uit Resources/CH10\_Voorbeelden
- Open /01\_helloworld.js

```
// eerste Node.js-bestand. Toon 'Hello World' in de console  
var msg = 'Hello World';  
console.log(msg);
```

- Je kan dit uitvoeren in het opdrachtvenster



# Hello World in Node.js

- Bekijk /02\_helloworld2.js

```
console.log(2 + 2);  
// werken met objecten  
var persoon = {voornaam : 'Peter', achternaam : 'Kassenaar'};  
console.log('Persoon: ', persoon.voornaam + ' ' + persoon.achternaam);
```

- Uitvoeren via opdrachtvenster:

```
Hello World  
4  
Persoon: Peter Kassenaar
```

# Een eenvoudige webserver maken

- Bekijk /03\_server.js
- **Stap 1 De module http laden**
  - Module http wordt beschikbaar gemaakt via de variabele server
  - `createServer()` heeft een functie als parameter, dewelke wordt aangeroepen als de server een verzoek van de browser ontvangt
  - De parameters request & response
    - **Request bevat alle gegevens van het binnenkomend verzoek**
    - **Response bevat de inhoud die wordt teruggegeven aan de browser**

```
var http = require('http');  
var server = http.createServer(function (request, response) {  
  
});
```

# Een eenvoudige webserver maken

- **Stap 2 De webserver schrijven**

- Een header geeft aan dat plain text wordt teruggestuurd met als inhoud “hello world”

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

- De server wordt gestart door te luisteren naar poort 3000 in onderstaande code (standaardpoort voor nodeApps)

```
server.listen(3000); // start webserver  
console.log('Server gestart op http://localhost:3000...');
```

# Een eenvoudige webserver maken

- Open een browser en ga naar <http://localhost:3000>



The image shows a code editor window with a file named `03_server.js` containing the following JavaScript code:

```
1 // Hello World als webserver.  
2 var http = require('http');  
3 var server = http.createServer(function (request, response) {  
4     response.writeHead(200, {'Content-Type': 'text/plain'});  
5     response.write('Hello World');  
6     response.end();  
7 });  
8 server.listen(3000); // server starten  
9 console.log('Server gestart op http://localhost:3000 ...');
```

Below the code editor, a web browser window is shown with the address bar set to `localhost:3000`. The browser displays the text `Hello World`, which is the response from the web server.

# Een eenvoudige webserver maken



## Asynchroon

Het programma dat u nu hebt geschreven is een perfect voorbeeld van de event driven en asynchrone werkwijze van Node.js. De functie `.createServer()` heeft een functie als parameter die pas wordt uitgevoerd als een event binnenkomt (namelijk: een nieuwe request). Als de functie is uitgevoerd keert hij weer terug in de wachtstand. Onderbreek het luisteren op poort 3000 met Ctrl+C.

Wat er nu ook gebeurt, bij elke request zal de server Hello World terugsturen. U kunt probleemloos verschillende URL's testen: `http://localhost:3000/index.html`, `http://localhost:3000/test.html`, `http://localhost:3000/blablabla`; het resultaat? Hello World!

# Een eenvoudige webserver maken

- Bekijk /04\_server2.js
- **Stap 3 HTML retourneren**
  - Aanpassingen van content type geeft vervolgens HTML als inhoud weer

```
var server = http.createServer(function (request, response) {  
  var url = request.url;  
  response.writeHead(200, {'Content-Type': 'text/html'});  
  response.write('<h1>De gevraagde URL: ' + url + '</h1>');  
  response.end();  
});
```

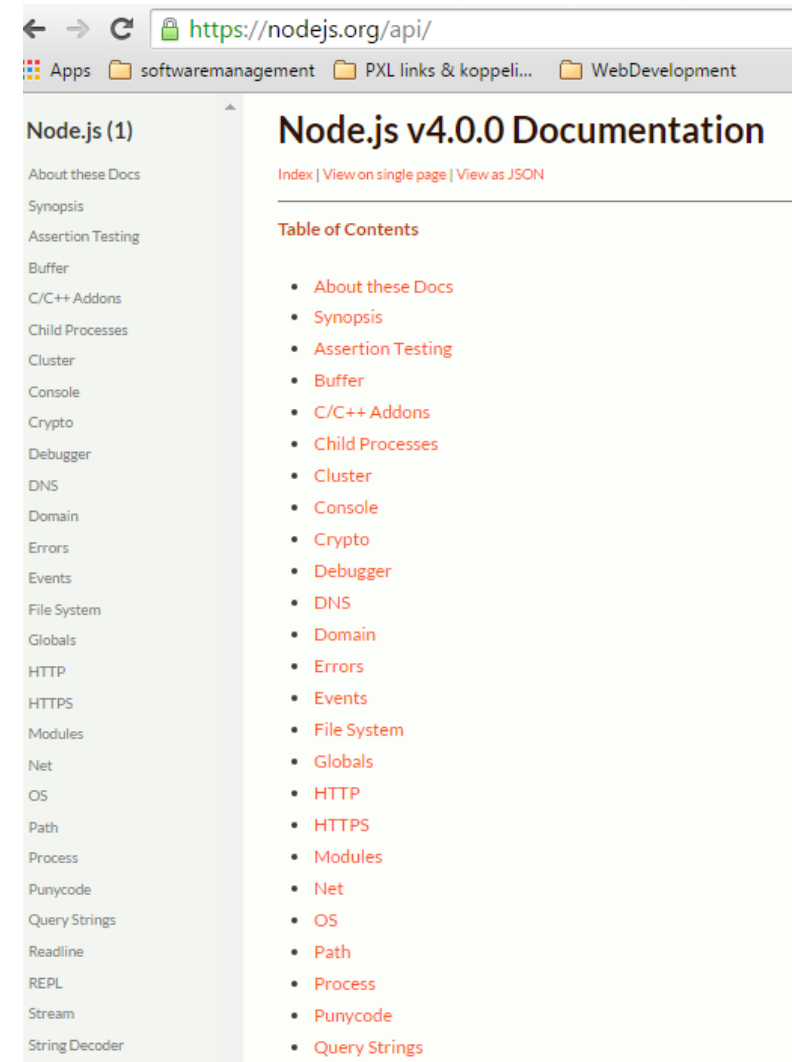
- Uitvoering in de browser



# NodeJS-documentatie leren lezen

- Helpbestanden zijn online te vinden op

<https://nodejs.org/api/>





# NodeJS-documentatie leren lezen

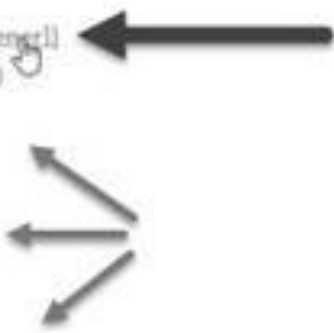
- De module “HTTP”

**Table of Contents**

- HTTP
  - `http.METHODS`
  - `http.STATUS_CODES`
  - `http.createServer([requestListener])`
  - `http.createClient([port][, host])`
  - Class: `http.Server`
    - Event: 'request'
    - Event: 'connection'
    - Event: 'close'
    - Event: 'checkContinue'
    - Event: 'connect'
    - Event: 'upgrade'
    - Event: 'clientError'
    - `server.listen(port[, hostname][, backlog][, callback])`
    - `server.listen(path[, callback])`
    - `server.listen(handle[, callback])`

Klik door naar `http.createServer([requestListener])` en vervolgens naar `http.Server`. Daarna leest u dat elke request een instantie is van `http.IncomingMessage` en elke response een instantie is van `http.ServerResponse`.

- Klik ook door op deze klassen om te zien welke eigenschappen beschikbaar zijn en hoe ze worden gebruikt.



Afbeelding 2.14 Het detailoverzicht laat zien welke methodes, classes en events beschikbaar zijn in de module HTTP. Hier staat ook de functie `http.createServer()` beschreven die we hebben gebruikt.

# NodeJS-documentatie leren lezen

- De module “HTTP” (vervolg)



**Afbeelding 2.15** Een request in Node.js is van het type `http.IncomingMessage`, zo blijkt uit de documentatie. Hier wordt beschreven welke eigenschappen dit object heeft.

# NodeJS-documentatie leren lezen

- Een praktijkvoorbeeld “http.IncomingMessage”
  - Bekijk het bestand /05\_Server3.js

```
var server = http.createServer(function (request, response) {  
    console.log(request.headers);  
});
```

Behalve `request.headers` zijn ook de eigenschappen `request.rawHeaders`, `request.httpVersion` en vele andere beschikbaar. Lees dit zelf in de documentatie!



Afbeelding 2.17 De headers van een willekeurige aanvraag zijn gelogd in de console

# NodeJS – Modules & packages



# Node.js-modules en -packages

- Node gebruikt JS files voor applicaties
- Volledige applicaties zullen nooit in één file geschreven worden  
⇒ Opsplitsing in “modules”
- Binnen een module geven we met `module.exports` aan welke modules voor de buitenwereld beschikbaar zijn.
- Waar de module gebruikt moet worden, schrijven we `require( './pad/naar/module' )`
- De toevoeging `.js` is niet verplicht.

# Node.js-modules en -packages



http is ook een module

Denk nog even terug aan de eenvoudige web-server uit het vorige hoofdstuk. Hierin hebt u al de regel `require('http')` gebruikt. Dit betekent dat in de applicatie de module `http` wordt ingeladen. Omdat dit een ingebouwde module is, hoeft u geen pad aan te geven. Later zult u meer in detail zien wanneer u wel- en geen padnaam opgeeft.

# Node.js-modules en -packages

- Buiten modules wordt er ook veel gebruik gemaakt van package
- NPM: node **package** manager
- Packages zijn containers met een bundeling van allerlei losse modules die op deze manier samen een applicatie vormen.
- Elke module heeft zijn eigen referenties naar andere modules.
- Door het bestand package.json te installeren via npm worden alle modules (en daarvan afhankelijke modules) opgehaald

# Praktijk: een logging module schrijven

- Herbruikbare logger: meldingen, fouten en algemene informatie naar de console schrijven
- Bekijk bestand /1006\_logger/logger.js
  - Stap 1 De logger schrijven

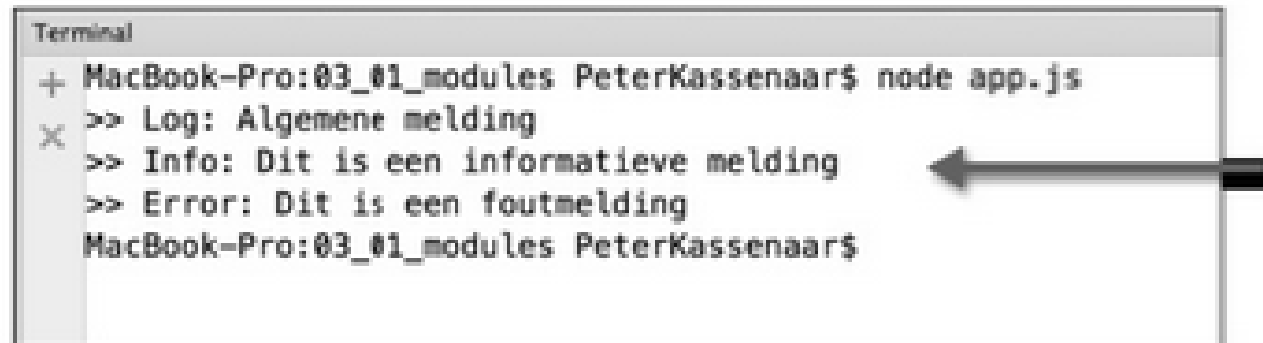
```
// logger.js - exporteer verschillende algemene logging-methoden
module.exports.log = function(msg){
    console.log('>> Log: ' + msg);
};
module.exports.info = function(msg){
    console.info('>> Info: ' + msg);
};
module.exports.error = function(msg){
    console.error('>> Error: ' + msg);
};
```



# Praktijk: een logging module schrijven

- Bekijk bestand /1006\_logger/app.js
  - Stap 2 De app schrijven

```
// app.js - laad de logger en test de drie functies.  
var logger = require('./logger');  
logger.log('Algemene melding');  
logger.info('Dit is een informatieve melding');  
logger.error('Dit is een foutmelding');
```



A terminal window titled 'Terminal' showing the execution of a Node.js script. The prompt is 'MacBook-Pro:03\_01\_modules PeterKassenaar\$'. The command 'node app.js' is entered. The output shows three lines: '>> Log: Algemene melding', '>> Info: Dit is een informatieve melding', and '>> Error: Dit is een foutmelding'. The prompt returns to 'MacBook-Pro:03\_01\_modules PeterKassenaar\$'. A black arrow points from the right towards the 'Info' and 'Error' output lines.

```
Terminal  
+ MacBook-Pro:03_01_modules PeterKassenaar$ node app.js  
X >> Log: Algemene melding  
  >> Info: Dit is een informatieve melding  
  >> Error: Dit is een foutmelding  
MacBook-Pro:03_01_modules PeterKassenaar$
```

Afbeelding 3.3 Het resultaat van de logger is niet verrassend, maar bedenkt nu een goede, modulaire programmeerwijze wordt gebruikt.

# Praktijk: een logging module schrijven

- Bekijk bestand /1007\_logger2/logger2.js
  - Een andere schrijfwijze voor de logger

```
// logger2.js - exporteer logging-methoden als functie
module.exports = function () {
  this.log = function (msg) {
    console.log('>> Log: ' + msg);
  };

  this.info = function (msg) {
    console.info('>> Info: ' + msg);
  };

  this.error = function (msg) {
    console.error('>> Error: ' + msg);
  };

  return this;
};
```

# Praktijk: een logging module schrijven

- Bekijk bestand /1007\_logger2/app2.js
  - De app herschreven

```
// app2.js - logger.js exporteert nu een functie met verschillende methods.  
var logger = require('./logger2');  
var myLog = logger(); // functie invoking  
myLog.info('Dit is informatie');  
myLog.error('Dit is een foutmelding');
```

# Modules laden in andere modules

- Bekijk bestanden in de map /1108\_logger3/

- getTime.js

```
// getTime.js - module een geformatteerde tijd (hh:mm:ss) teruggeeft.  
module.exports = function () {  
    var now = new Date();  
    var tijd = now.getHours() + ':' + now.getMinutes() + ':' + now.getSeconds();  
    return tijd;  
};
```

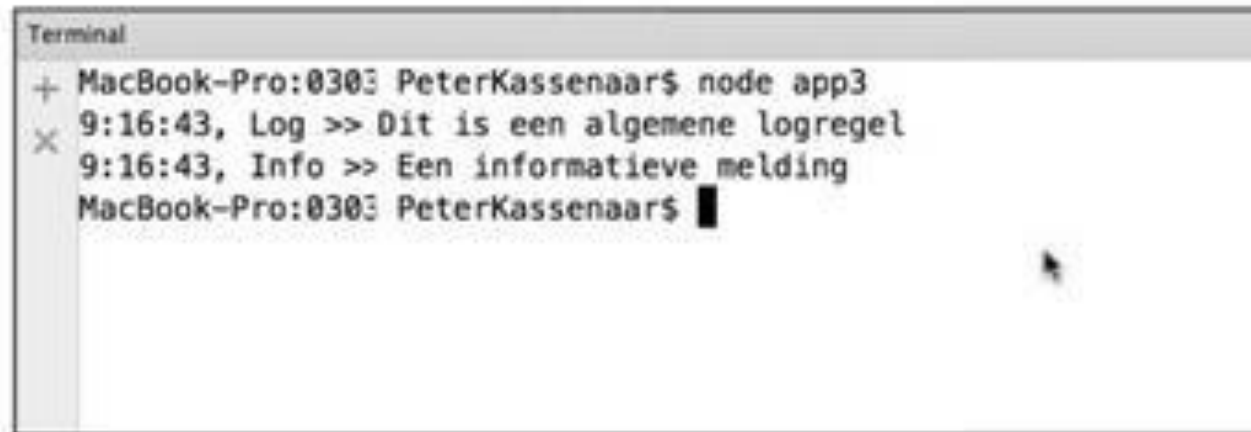
- Logger3.js

```
// logger3.js - require een andere module  
var time = require('./getTime');  
module.exports = function () {  
    this.log = function (msg) {  
        console.log(time() + ', Log >>' + msg);  
    };  
    ...  
    return this;  
};
```

# Modules laden in andere modules

- Bekijk bestanden in de map /1008\_logger3/

Tot slot hoeft in app3.js niks gewijzigd te worden. De publieke interface van logger is immers hetzelfde gebleven. De uitvoer ziet er bijvoorbeeld uit zoals in de afbeelding.

A screenshot of a macOS Terminal window. The title bar says "Terminal". The prompt is "MacBook-Pro:0303 PeterKassenaar\$". The command "node app3" has been executed. The output shows two log messages: "9:16:43, Log >> Dit is een algemene logregel" and "9:16:43, Info >> Een informatieve melding". The prompt "MacBook-Pro:0303 PeterKassenaar\$" is visible again at the bottom.

```
Terminal
+ MacBook-Pro:0303 PeterKassenaar$ node app3
9:16:43, Log >> Dit is een algemene logregel
9:16:43, Info >> Een informatieve melding
MacBook-Pro:0303 PeterKassenaar$
```

Afbeelding 3.4 De logger voegt nu ook de tijd toe aan de melding. Deze staat in een afzonderlijke module.

# NPM gebruiken

- In de vorige slides werd beschreven hoe je zelf modules kan schrijven
- Via NPM kunnen duizenden modules & packages gedownload & gebruikt worden
- NPM is samen met NodeJS geïnstalleerd

```
npm install <package-naam> // installeer package lokaal, in huidige project.  
npm install <package-naam> -g // installeer package globaal, overal beschikbaar.
```

# NPM gebruiken

- Belangrijke NPM-opdrachten om te onthouden zijn:
  - **npm install** installeer een package
  - **npm uninstall** verwijder een package (inclusief dependencies)
  - **Npm cache clean** Maak de cache leeg
  - **npm update** Update een package
  - **npm init** Maak een package.json-bestand met package beschrijving
  - **npm publish** Publiceer een package naar de registry zodat anderen hem ook kunnen gebruiken
- Meer leren over NPM: <http://docs.npmjs.org>

# De module 'moment' gebruiken

- Bekijk bestanden in de map /1009\_moment
- Moment
  - is een bibliotheek voor het werken met datums en tijden
  - Handiger dan default methodes van JavaScript
  - <http://momentjs.com/>

```
npm install moment
```



# De module 'moment' gebruiken

- Bekijk bestand /1009/logger4.js

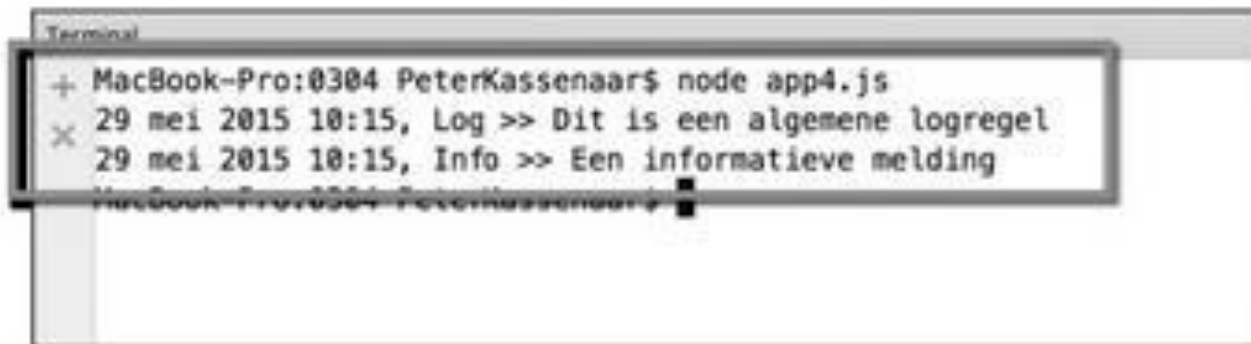
```
var moment = require('moment');
module.exports = function () {
  moment.locale('nl'); // Stel Nederlandse notatie in
  this.log = function (msg) {
    console.log(moment().format('lll') + ', Log >> ' + msg);
  };
  --
}
```

- Het statement `moment.locale('nl')` zorgt er voor dat de Nederlandse datum- en tijdsnotatie wordt gebruikt.
- De formatstring `.format('lll')` zorgt voor een standaardweergave in lang datumformaat.
- Zie de documentatie (en de afbeelding) voor meer mogelijkheden voor formatteren van datum en tijd.

# De module 'moment' gebruiken

- Bekijk bestand /1009\_moment/app4.js

In app.js hoeft wederom niets gewijzigd te worden, alle aanpassingen waren op het niveau van de module logger. Draai daarom de applicatie met de volgende opdracht:



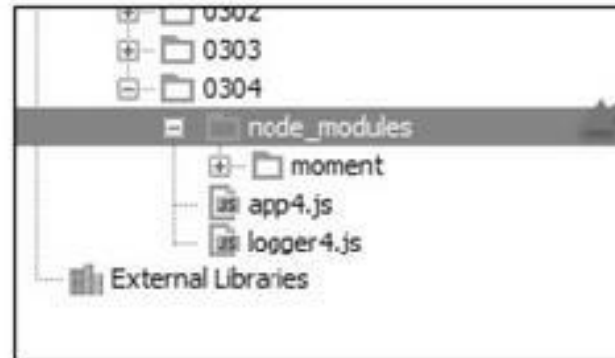
```
Terminal
+ MacBook-Pro:0304 PeterKassenaar$ node app4.js
x 29 mei 2015 10:15, Log >> Dit is een algemene logregel
  29 mei 2015 10:15, Info >> Een informatieve melding
MacBook-Pro:0304 PeterKassenaar$
```

Afbeelding 3.7 *Moment zorgt voor de weergave in het Nederlandse datum- en tijdformaat.*

# De map node\_modules

- Bekijk de map /1009/node\_modules

Packages die via NPM aan het project worden toegevoegd, worden opgeslagen in een map node\_modules. NPM maakt deze map als hij nog niet bestaat. Als u nu in het project gaat kijken, zult u deze map dus – na het toevoegen van moment – ongetwijfeld tegenkomen.



Afbeelding 3.8 NPM heeft de map node\_modules toegevoegd. Hierin staan lokale modules voor het huidige project.

De map moment is aanwezig als submap van node\_modules. Zo is altijd snel te inspecteren welke modules in een project geladen zijn.

# Enkele populaire NPM packages

- Underscore & lodash
  - Helper libraries voor functies voor het werken met collecties, arrays, objecten en variabelen
  - <http://underscorejs.org/> en <https://lodash.com/>

```
npm install underscore  
npm install lodash
```

Deze twee bibliotheken zijn overigens direct een uitzondering op de regel dat de variabelenaam meestal gelijk is aan de package-naam. In een applicatie worden ze vaak als volgt gebruikt:

```
var _ = require ('underscore'); // OF  
var _ = require ('lodash');
```

# Enkele populaire NPM packages

- Toepassing op Underscore en lodash

```
_ .first([1,2,3,4,5]); // 1  
_ .first([1,2,3,4,5], 3); // 1, 2, 3. Eerste drie elementen uit de array  
_ .last([1,2,3,4,5]); // 5
```

Meer informatie over underscore en lodash is te vinden op [underscorejs.org](http://underscorejs.org) en [lodash.com](http://lodash.com). Google op **underscore vs lodash** voor discussies over welke bibliotheek “het beste” is, of de overeenkomsten en verschillen tussen beide.

# Enkele populaire NPM packages

- Request

Request is een handige bibliotheek om te werken met http-calls vanuit uw applicatie. Dit is dus iets anders dan de webserver die u in hoofdstuk 2 hebt gemaakt; deze *ontvangt* http-calls en reageert daarop. De module request wordt bijvoorbeeld als volgt ingezet:

```
npm install request
```

En vervolgens

```
var request = require('request');
request('http://www.webdevelopmentlibrary.nl', function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log(body) // Toon de HTML van webdevelopmentlibrary.nl in de console
  }
})
```

Meer informatie over request is te vinden op  
[www.npmjs.com/package/request](http://www.npmjs.com/package/request)

# Enkele populaire NPM packages

- Colors (/1010\_colors)
  - Geeft meldingen in de console weer in verschillende kleuren

```
npm install colors
```

Daarna kunnen we de logger als volgt aanpassen:

```
var moment = require('moment'),
    colors = require('colors');
module.exports = function () {
  moment.locale('nl'); // Stel Nederlandse notatie in
  this.log = function (msg) {
    var prefixString = moment().format('lll') + ', Log >> ';
    console.log(prefixString.green + msg);
  };
  ... console.log(colors.green(prefixString + msg));
}
```

# Zelf packages maken met npm init

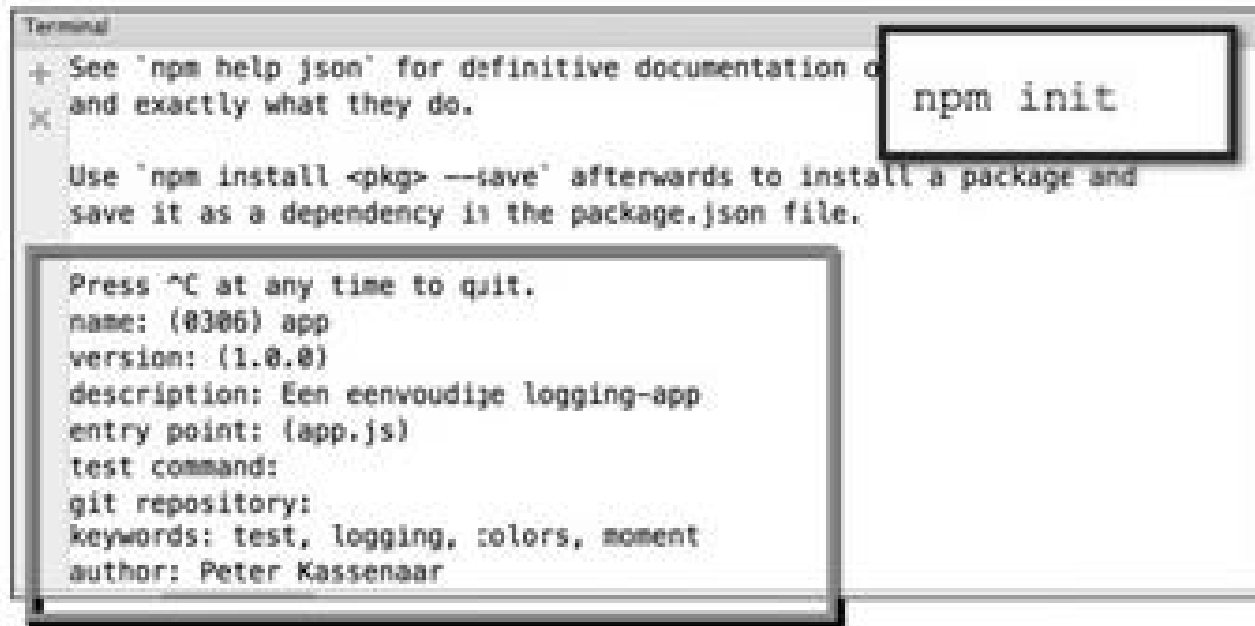
- **Package.json**

- Handig om in dit bestand te beschrijven van welke andere modules deze package afhankelijk is
- Behalve de afhankelijkheden staat in dit .json-object nog veel meer informatie; auteur, versie, gebruikerslicentie, testopdrachten beschikbaar, minimale versie van node.js, enz



# Zelf packages maken met npm init

- Maak een nieuwe directory en plaats hierin alle bestanden die je zelf geschreven hebt (app.js en logger.js)
- Open de CLI in de nieuwe folder en typ **npm init**
- Beantwoord de vragen die verschijnen (niet alle vragen zijn verplicht)



```
Terminal
+ See 'npm help json' for definitive documentation of
X and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (0306) app
version: (1.0.0)
description: Een eenvoudige logging-app
entry point: (app.js)
test command:
git repository:
keywords: test, logging, colors, moment
author: Peter Kassenaar
```

The image shows a terminal window with the output of the 'npm init' command. The command 'npm init' is highlighted in a box. The terminal output shows the command's purpose, usage instructions, and a series of prompts for package metadata. The prompts are answered with the following values: name: (0306) app, version: (1.0.0), description: Een eenvoudige logging-app, entry point: (app.js), test command: (empty), git repository: (empty), keywords: test, logging, colors, moment, and author: Peter Kassenaar.

# Zelf packages maken met npm init

- Package.json

```
3   "version": "1.0.0",
4   "description": "Een eenvoudige logging-app",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  }
12
```

# Zelf packages maken met npm init

- Modules toevoegen aan package.json vanuit de project dir:  
    `npm install moment`  
    `npm install --save colors`
- Deze modules komen in de map `.\node_modules` te staan. De link wordt gelegd in de dependencies binnen package.json

```
"main": "app5.js",  
"dependencies": {  
  "moment": "^2.10.3",  
  "colors": "^1.1.0"  
},
```

# Zelf packages maken met npm init

- Package (her)installeren
  - Met `npm install` (dus zonder packagenaam) gaat:
    - NPM zoeken naar `package.json` in de huidige map
    - Alle dependencies uit `package.json` laden in de folder `node_modules`
  - Je kan dit testen door de map `node_modules` te verwijderen en `npm install` uit te voeren

# Regels voor require()

- Als een padnaam is aangegeven, zoekt Node de module in het aangegeven pad

Voorbeeld: `require( './logger' );`

- Als er geen padnaam is aangegeven, zoekt Node in `node_modules`. De module moet met `npm install <package>` geïnstalleerd zijn

Voorbeeld: `require( 'moment' );`

- Als hij niet wordt aangetroffen in `node_modules`, is de volgende stap dat Node zoekt in beschikbare globale modules (bv: `http`, `path`, `file system`, ...)

Voorbeeld: `require( 'http' );`

- Indien hij nog niet gevonden is: `error: cannot find module`

# De webserver uitbreiden



# Enkele belangrijke variabelen en modules

- Globals `__filename` en `__dirname`

Node.JS kent deze variabelen om de huidige directory en bestandsnaam makkelijk beschikbaar te maken.

```
console.log('De bestandsnaam is: ', __filename);  
console.log('De huidige directory is: ', __dirname);
```

# Enkele belangrijke variabelen en modules

- De module Path
- Globale module die altijd beschikbaar is (zoals bv ook http)
- Eerst require gebruiken:  

```
var path = require('path');
```
- Vervolgens kunnen volgende methodes gebruikt worden:
  - `path.normalize(pad)` normaliseren van paden
  - `path.join([pad1],[pad2])` voegt parameters samen tot 1 pad
  - `path.resolve(pad)` geeft het absolute pad van de meegegeven dir
  - `path.dirname(pad)` returned de dir naam uit een pad
  - `path.basename(pad)` returned laatste deel uit een pad (bestandsnaam)
  - `path.extname(pad)` returned de extentie van het pad



# Enkele belangrijke variabelen en modules

- De module path
  - Bekijk de bestanden in /1012\_Path

```
var path = require('path');  
var voorbeeldPath = ('Users/Peter Kassenaar/Documents/test.html');  
console.log('normalize: ', path.normalize(voorbeeldPath));  
console.log('resolve: ', path.resolve(voorbeeldPath));  
console.log('dirname: ', path.dirname(voorbeeldPath));  
console.log('basename: ', path.basename(voorbeeldPath));  
console.log('extname: ', path.extname(voorbeeldPath));
```

# Enkele belangrijke variabelen en modules

- De module File System
  - Afgekort fs -> `var fs = require( 'fs' );`
- Globale module die altijd beschikbaar is (zoals bv ook http)
- Volgende methodes kunnen gebruikt worden:
  - `fs.readFile(filename[, options], callback)`  
De asynchrone manier om bestanden in te lezen. Callbackfunctie om aan te geven wat er moet gebeuren met het bestand na het inlezen.
  - `fs.readFileSync(filename [, options])`  
De synchrone manier om bestanden in te lezen. Dit wordt afgeraden omdat de thread geblokkeerd is zolang Node.js bezig is met het inlezen van bestanden.
  - Daarnaast ook `fs.writeFile()` en `fs.writeFileSync`

# Enkele belangrijke variabelen en modules

- De module fs
  - Bekijk de bestanden in /1013\_FileSystem

Bij het inlezen moet u het encoding-type ogeven. Anders retourneert Node een <Buffer>-object.

```
var fs = require('fs');
var msg = 'Hello World';

// #1. Bestand opslaan
fs.writeFile('hello.txt', msg, function () {
  console.log('bestand opgeslagen!')
});
```

```
// #2. Bestand inlezen en in de console tonen
fs.readFile('hello.txt', 'utf8', function (err, data) {
  if (err) {
    console.log('Error: ', err);
  } else {
    console.log('bestand ingelezen: ', data);
  }
});
```

# Enkele belangrijke variabelen en modules

- De module fs
  - Bekijk de bestanden in /1012\_Path

```
// #4. Eerst testen of bestand bestaat, voordat unlink wordt aangeroepen:
fs.exists('test.txt', function (exists) {
  if (exists) {
    deleteFile('test.txt');
  } else {
    console.log('text.txt niet gevonden! Kan niet verwijderen.')
  }
});|
```

```
// #3. Bestand verwijderen via unlink
fs.unlink('text.txt', function (err) {
  if (err) {
    console.log('Error: ', err);
  } else {
    console.log('Text.txt is verwijderd');
  }
});
```

# De webserver uitbreiden

- Bekijk het bestand /1014\_server/server\_01.js

```
// Een eenvoudige webserver.  
// 0. initialisatie en variabelen  
var http = require('http'),  
    fs = require('fs'),  
    path = require('path'),  
    root = __dirname + '/public/'; // magic variable  
  
// 1. Maak de webserver  
var server = http.createServer(function (req, res) {  
  
});  
  
// 2. Start de server  
server.listen(3000);  
console.log('Server gestart op http://localhost:3000');
```

# De webserver uitbreiden

- De homepage serveren (en andere pages)
  - Dit is het bestand \public\index.html

```
// 1. Maak de webserver
var server = http.createServer(function (req, res) {
  // 1a. Check of de root wordt opgevraagd.
  var fileName = '';
  var url = req.url;
  if (url === '/') {
    url = 'index.html'; // redirect als geen bestandsnaam is opgegeven
  }
  fileName = root + url; // root = '/public/'
  console.log('Gevraagd bestand: ', path.basename(fileName));
});
```

# De webserver uitbreiden

- Checken of het gevraagde bestand bestaat

```
// 1b. Check of bestand bestaat.  
fs.exists(fileName, function (exists) {  
  if (exists) {  
    serveFile(fileName); // ja.  
  } else {  
    fileName = root + '404.html'; // nee  
    serveFile(fileName);  
  }  
});
```

# De webserver uitbreiden

- Streams – het bestand serveren via een helperfunctie en events
  - Events 'on' met 2 parameters: wat er binnenkomt en een callback function

```
// 1c. Serveer gevraagde bestand.  
function serveFile(requestFile) {  
  // 2. Maak een stream en server op basis van Events  
  var stream = fs.createReadStream(requestFile);  
  stream.on('data', function (chunk) {  
    res.write(chunk);  
  });  
  stream.on('end', function () {  
    res.end();  
  });  
  stream.on('error', function (err) {  
    console.log('error: ' + err);  
  });  
}
```

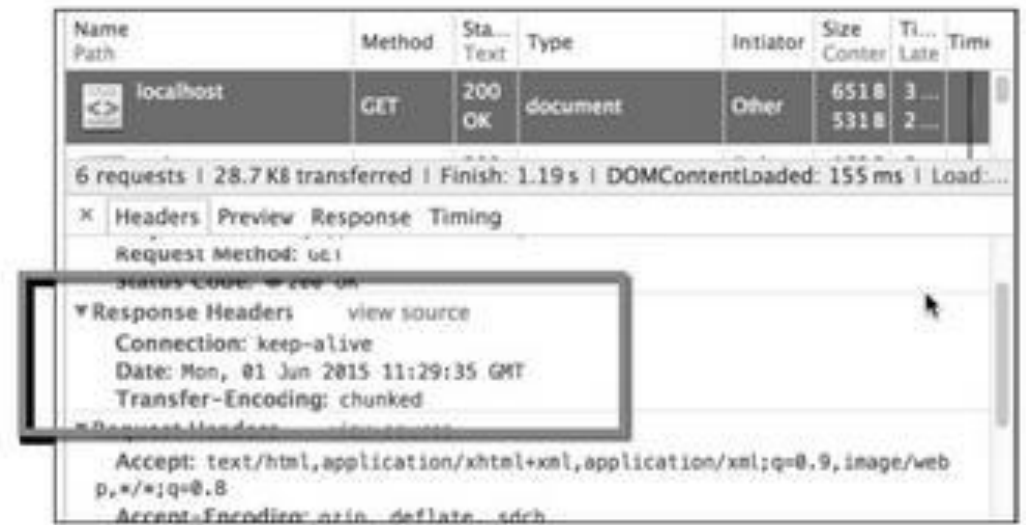


# De webserver uitbreiden

- MIME-types
  - Gaat goed omdat de browser een gok doet en minimale headers voorziet
  - Afhandeling zou serversided moeten gebeuren



Afbeelding 4.6 De pagina bevat nu HTML, stylesheets en een afbeelding.



Afbeelding 4.7 In de netwerktafel van de developer tools is te zien dat elk bestand nu nog met een minimale set http-headers wordt geretourneerd.

# De webserver uitbreiden

- MIME-types
  - Eenvoudigste oplossing:
    - Afbeeldingen / css zal ook als “tekst/html” verzonden worden.

```
function serveFile(requestFile) {  
  // 2. Maak een stream en serveer op basis van events  
  var stream = fs.createReadStream(requestFile);  
  res.writeHead(200, {'Content-Type' : 'text/html'});  
  stream.on('data', function (chunk) {  
    res.write(chunk);  
  });  
  ...  
}
```

# De webserver uitbreiden

- MIME-types
  - Dynamische bepaling van MIME-types kan met de module mime  
`npm install --save mime`
  - Vervolgens kan na de require een lookup gedaan worden van de MIME types  
`var mime = require('mime');`

```
mime.getType('index.html');  
mime.getType('styles.css');  
mime.getType('foto.png');
```

# De webserver uitbreiden

- Bekijk het bestand /1014\_server/server\_02.js

```
var http = require('http'),  
    fs = require('fs'),  
    path = require('path'),  
    mime = require('mime'),  
    root = __dirname + '/public/'; // magic variable
```

```
function serveFile(requestFile) {  
    // 2. Maak een stream en server op basis van Events  
    res.writeHead(200, {'Content-Type': mime.getType(requestFile)});  
    var stream = fs.createReadStream(requestFile);  
    stream.on('data', function (chunk) {  
        res.write(chunk);  
    });  
    stream.on('end', function () {  
        res.end();  
    });  
    stream.on('error', function (err) {  
        console.log('error: ' + err);  
    });  
}
```

# De webserver uitbreiden

- Betere methode voor 404
  - Momenteel wordt de 404 error ook met een http-header 200 OK
  - Aparte methode voorzien voor het renderen van de 404 pagina:
    - Hier met fs.readFile() in plaats van streams

```
// 1d. Serveer 404, inclusief juiste http-header
function serve404(requestFile) {
  res.writeHead(404, {'Content-Type': mime.lookup(requestFile)});
  fs.readFile(requestFile, 'utf8', function (err, data) {
    if (err) {
      console.log('Error: ', err);
    } else {
      res.end(data);
    }
  })
}
```

# De webserver uitbreiden

- Betere methode voor 404
  - Aanpassen fs.exists() methode:

```
// 1b. Check of bestand bestaat.  
fs.exists(fileName, function (exists) {  
  if (exists) {  
    serveFile(fileName); // ja.  
  } else {  
    fileName = root + '404.html'; // nee  
    serve404(fileName);  
  }  
});
```