Programmeren 4 – Practicum 1

# Lees dit eerst

In dit practicum ga je werken met Node.js, Git, Heroku en een aantal ondersteunende software tools. Het practicum sluit aan op de inhoud van Training 1.

Voer de opdrachten uit in **groepjes van 2 studenten**.

Maak na elk practicum samen met je medestudent van jullie uitwerkingen van alle opdrachten een kort verslag in PDF formaat (1 of 2 A4-tjes). Lever het verslag individueel in via de inleverlink op Blackboard (let op de inleverdeadline op BB).

De docent die het practicum begeleidt komt tijdens het volgende practicum bij elk groepje langs om de uitwerking te bekijken en eventuele vragen te beantwoorden.

We gaan in dit practicum een eenvoudige web server opzetten met Node.js en Express. Die server breidt je daarna stap voor stap uit tot hij de API van http://randomuser.me/api exact nabootst, zodat je je Android app van de vorige periode eraan kunt koppelen.

# Voorbereidingen

Installeer Postman (een Chrome app) als je dat nog niet gedaan hebt.

<https://www.getpostman.com/>

Vraag een student license aan bij JetBrains en download en installeer de WebStorm IDE. Als je liever een andere IDE wilt gebruiken (b.v. Eclipse), installeer die dan in plaats van WebStorm. In de voorbeelden gaan we echter WebStorm gebruiken.

<https://www.jetbrains.com/student/>

Installeer Node.js (als je dat nog niet gedaan hebt).

<https://nodejs.org/en/download/>

Installeer Git (als je dat nog niet gedaan hebt).

<https://git-scm.com/downloads>

Je kunt als je wilt ook SourceTree installeren, een grafische schil rondom Git. Die is handig maar niet verplicht, want je kunt Git ook met command line commando’s bedienen.

<https://www.atlassian.com/software/sourcetree>

Voor je Git gaat gebruiken moet je eerst twee dingen instellen als je dat nog niet gedaan hebt:

git config --global user.name "<je naam>"

git config --global user.email <je emailadres>

Maak een gratis account aan op GitHub (als je dat nog niet gedaan hebt).

<https://github.com/>

Maak een gratis Heroku account aan (als je dat nog niet gedaan hebt).

<https://www.heroku.com/>

Maak een werkdirectory aan op je laptop voor alle opdrachten die je bij dit vak Programmeren 4 gaat maken. Noem die bijvoorbeeld 'programmeren-4'. In die werkdirectory maak je straks mappen voor alle losse projecten die je in het practicum (en straks voor het individueel assessment) gaat bouwen. Van deze werkdirectory gaan we straks een Git repository maken die we op GitHub kunnen zetten.

# Opdracht 1

Oefen in het gebruik van Postman door samen met je medestudent **twee** van de volgende RESTful web services te benaderen en daar enkele GET requests te doen. Lees de online documentatie hoe je de API moet benaderen.

* http://swapi.co/ (Star Wars API)
* https://pokeapi.co/ (Pokémon API)
* http://www.omdbapi.com/
* https://randomuser.me/
* https://www.rijksoverheid.nl/opendata/nieuwsberichten (Informatie van de rijksoverheid)
* https://www.rijksoverheid.nl/opendata/schoolvakanties

Beantwoord samen met je medestudent **twee** van onderstaande vragen (welke je kiest hangt af van welke API's je bekeken hebt):

* Wat is de naam van planeet nummer 7 en nummer 18 in de Star Wars database? Laat het JSON fragment zien.
* Wat is de naam van Pokémon nummer 7 en nummer 18? Laat het JSON fragment zien met het antwoord (meerdere mogelijkheden).
* Hoeveel films over "Harry Potter" zijn er in de OMDb API te vinden? Welke film is als eerste verschenen (year of release), welke als laatste? Toon het JSON fragment.
* Wie is de eerste persoon uit de randomuser.me REST API als je het seed 'Programmeren4' gebruikt en je beperkt tot het zoeken naar Nederlandse personen?
* Wat is het meest recente nieuwsbericht over drones dat je via de RESTful web service van de rijksoverheid kunt vinden? In welk format staat de informatie van die service?
* Wanneer begint en eindigt de Meivakantie in 2017 volgens de RESTful web service van de rijksoverheid? Toon de URL die je gebruikt om dat te vinden.

# Opdracht 2

Maak een nieuw project aan in WebStorm (of in een andere IDE naar keuze, b.v. Eclipse), in een eigen map binnen je werkdirectory (die je tijdens de voorbereidingen hebt gemaakt, zie boven). Noem die map bijvoorbeeld 'MyRandomUser'.

Open een terminal (command prompt) en voer in

npm init

Dit maakt een package.json file aan op basis van een aantal gegevens die je moet invullen:

name: <de naam van je project, zonder hoofdletters>

version: 1.0.0

description: Random User Generator REST API in Node.js

entry point: <naam van je Javascript file, b.v. server.js of myrandomuser.js>

test command: <laten we voorlopig leeg>

git repository: <deze stap komt later pas, nu nog even leeg laten>

keywords: <kun je leeg laten>

author: <je eigen naam>

license: <accepteer de default>

Bekijk de nieuw gegenereerde file package.json in WebStorm.

We gaan werken met Express, een uitbreiding op Node.js, dus die moeten we eerst installeren met npm:

npm install --save express

De --save optie voegt Express toe aan de dependencies in package.json (kijk maar in die file).

Je zult zien dat npm ook een map 'node\_modules' heeft toegevoegd met daarin de gevraagde module (Express) en alles wat nodig is om die module te laten werken (kijk maar eens in die map).

Zoek de documentatie over package.json op (<https://docs.npmjs.com/getting-started/using-a-package.json>) en beantwoord de volgende vragen:

1. Aan welke eisen moet de "name" van je project voldoen?
2. Wat betekent de '^' (*caret* in het Engels) voor het typenummer achter "express":? Als er "^4.15.2" staat, welke nieuwe versies van express zijn dan voor dit project (deze package.json) toegestaan en welke niet:

4.0.0 4.15.0 4.15.3 4.16.0 4.16.2 5.0.0 5.15.2 6.3.4

*Opmerking: In de beschrijving zie je dat versies die met 0.x of 0.0.x beginnen apart worden behandeld, maar voor express geldt dat gelukkig niet want deze module is inmiddels al bij versie 4.x.*

Maak een nieuwe Javascript file in WebStorm met als naam dat wat bij 'entry point' staat in package.json . Noem die file bijvoorbeeld myrandomuser.js.

Zet in die nieuwe file de volgende Javascript code:

var express = require('express');

var app = express();

app.get('/', function(request, response) {

response.send('Hello Avans!');

})

app.get('/about', function(request, response) {

response.send('Written by <jouw naam hier invullen>');

})

app.post('/', function(request, response) {

response.send('Hello Avans, POST request received!');

})

app.put('/', function(request, response) {

response.send('Hello Avans, PUT request received!');

})

app.listen(3000, function() {

console.log('Server app is listening on port 3000');

})

Start de server met 'node <jouw Javascript file>' in het Terminal venster in WebStorm en test met Postman een GET, POST en PUT request naar <http://localhost:3000/>. Beantwoord deze vragen:

1. Welke response zie je in Postman? Klopt dat met de Javascript source code?
2. Probeer een aantal paden in combinatie met localhost:3000 en beschrijf wat je ziet:

http://localhost:3000/about

http://localhost:3000/about/

http://localhost:3000/About

http://localhost:3000/ABOUT

http://localhost:3000/AboutThisApp

http://localhost/about

http://localhost:3000/api

http://localhost:3000/api/

*Tip: Om de server te stoppen voer je Ctrl-C in in het Terminal venster. Telkens als je de Javascript code hebt gewijzigd moet je de server opnieuw starten.*

Inmiddels heb je als het goed is een indruk wat app.get(...) doet.

1. Lees de documentatie op https://expressjs.com/en/guide/routing.html en vat je conclusies kort samen (<= 10 regels tekst).

Je hebt gezien dat lang niet alle URLs een response opleveren en dat zou eigenlijk wel moeten. Wijzig de code zoals hieronder om dat op te lossen:

var express = require('express');

var app = express();

*... (bestaande app.get(...) regels)*

**app.get('\*', function(request, response) {**

**response.status(404);**

**response.send('404 - Not found');**

**})**

app.listen(3000, function() {

console.log('Server app is listening on port 3000');

})

1. Test met een GET request naar localhost:3000/api in Postman. Werkt het ook voor een **POST** naar localhost:3000/api?
2. Als we app.get('\*', ...) vervangen door app.all('\*', ...), lost dit dan het probleem op met de POST? Test in Postman en toon je resultaat.
3. Wat gebeurt er als we de app.all('\*', ...) bovenaan zetten in de file, boven de eerste app.get(...)? Test met Postman, beschrijf je resultaat.

Zet app.all(...) terug onderaan de file, vlak boven app.listen(...).

Daarmee is de eerste versie van onze code klaar.

# Opdracht 3

Het wordt tijd om onze werkende code onder versiebeheer te plaatsen.

Maak van je werkdirectory voor Programmeren 4 (dus de directory waarin straks alle projectmappen staan) een Git repository. Ga daartoe in je Terminal in WebStorm (of in een ander command line venster) naar je **werkdirectory** (dus **niet** naar je projectdirectory, want dat is een subdirectory van je werkdirectory) en voer in:

git init

Git heeft nu een (verborgen) .git repository (een hidden directory) aangemaakt in je werkdirectory (check maar in een Explorer venster).

*Tip: Als je in WebStorm VCS > Refresh File Status doet zie je dat ook WebStorm nu ziet dat het project in een Git repository staat.*

We kunnen nu onze files toevoegen aan de Git repository door in de Terminal terug te gaan naar onze projectdirectory en in te typen:

git add myrandomuser.js

git add package.json

Vraag de Git status op:

git status

We zien dat .idea/ en node\_modules/ 'untracked' zijn (dus niet in versiebeheer staan) en dat willen we graag zo houden. In de '.idea' directory zet WebStorm zijn projectinstellingen. In de 'node\_modules' directory staan alleen maar modules die npm voor ons heeft geïnstalleerd en die kunnen we op ieder moment opnieuw aanmaken, dus ze hoeven **niet** onder versiebeheer (algemene stelregel: wat je automatisch kunt genereren hoort niet onder versiebeheer thuis, uitzonderingen daargelaten).

Om te voorkomen dat Git ze per ongeluk toch in de repository opslaat gaan we ze uitsluiten van versiebeheer.

Maak in je werkdirectory een nieuwe file aan die '.gitignore' heet. Daarin slaan we voor al onze Node.js projecten op welke files niet onder versiebeheer thuishoren.

In deze '.gitignore' file zet je de volgende regels:

# Files and directories to be ignored by git

# Node build artifacts

node\_modules

# WebStorm IDE

.idea

# MacOS

.DS\_Store

*Opmerking: De regels die met # beginnen zijn commentaren. De overige regels bevatten informatie voor Git.*

*Tip: Een (heel complete) .gitignore file voor Node.js projecten is bijvoorbeeld: https://github.com/github/gitignore/blob/master/Node.gitignore*

Voeg ook de .gitignore file toe aan de repository:

git add .gitignore

Commit alle drie de files die nu in de staging area staan in de repository met een commentaar bij die commit:

git commit -m "Initial commit"

Verifieer dat alle wijzigingen nu opgeslagen zijn in de repository:

git status

*Opmerking: Git spreekt van een 'master branch', daar komen we volgende week op terug (speelt deze week nog geen rol).*

# Opdracht 4

Op dit moment is onze Git repository nog slechts lokaal (= staat op je eigen laptop en verder nergens). Om de repository ook remote (= op een centrale server) te zetten moeten we die remote repository aanmaken en de repositories koppelen. Wij gaan dat doen op GitHub.

Maak op GitHub een nieuwe repository aan. Noem die bijvoorbeeld 'programmeren-4'.

Koppel nu je remote repository aan je lokale repository door in je terminal in je werkdirectory in te voeren:

git remote add origin <URL van je remote repository op GitHub, zie de GitHub pagina>

Je remote repository is bij Git nu bekend als 'origin'. Kijk welke remote repositories Git kent:

git remote -v

Tot slot schrijven we de huidige inhoud van onze lokale repository naar de remote repository:

git push origin master

*Opmerking: Met 'origin' geef je aan dat je naar de remote repository wilt pushen die je zojuist gekoppeld hebt. Met 'master' geef je aan dat je de 'master' branch wilt pushen. Daarover gaan we het volgende week hebben. Voor nu speelt dat nog geen rol.*

Git vraagt om je accountnaam en password van je GitHub account om toegang te krijgen tot je remote repository bij het pushen. Als die juist zijn krijg je te zien dat alles in orde is. Gefeliciteerd, je hebt je eerste code in een Git repository op GitHub gezet!

*Belangrijke opmerking: Telkens als je iets wijzigt in je lokale Git repository (git add, git commit) wordt die wijziging alleen lokaal opgeslagen. De wijziging komt pas op je remote GitHub repository te staan als je de wijziging opnieuw pusht naar de remote repository (git push).*

# Opdracht 5

Tot nu toe heb je je server code lokaal op je eigen laptop gedraaid met Node.js. Om de server beschikbaar te maken voor iedereen gaan we hem naar Heroku uploaden en daar (in de 'cloud') draaien.

Er zijn drie manieren om code naar Heroku te uploaden (deployment). Wij zullen daarvoor gebruik maken van GitHub (één van de drie manieren).

Ga naar je Heroku Dashboard pagina op de Heroku website en klik op de New button rechtsboven.

In het volgende venster kun je aangeven dat je app in Europa gehost moet worden, en kun je er een naam aan geven. Als je de naam leeg laat maakt Heroku er eentje voor je aan.

In het volgende venster (Deploy) klik je bij Deployment method op GitHub. GitHub vraagt je of je Heroku toegang tot je GitHub account wilt geven. Bevestig dat verzoek.

Geef daarna op de Heroku Deploy pagina aan met welke GitHub repository je verbinding wilt maken. Voer de naam van de repository in die je eerder gemaakt hebt. Klik op Search, daarna op Connect als hij gevonden is.

We willen voorlopig onze code handmatig deployen op Heroku. Klik op Deploy Branch (de default branch is 'master' en dat is op dit moment OK).

Je zult zien dat de build op Heroku faalt omdat Heroku niet kon vinden welke taal (platform, in ons geval Node.js) gebruikt moet worden. Dat komt omdat ons project niet in de root directory maar in een subdirectory van de root van onze repository staat, en dat verwacht Heroku niet.

Om dat op te lossen moeten we een package.json file in de root directory van de repository zetten. Daarin geven we aan dat de code die gestart moet worden door node in een subdirectory staat. Neem als basis voor je package.json de file die eerder al is aangemaakt (met npm init) en pas hem zo aan:

{

... (bestaande regels)

"main": "./<jouw subdirectory>/<jouw Javascript file>",

"scripts": {

"start": "node ./<jouw subdirectory>/<jouw Javascript file>"

},

... (bestaande regels)

}

In de root directory van de repository zetten we ook een tekstfile met de naam 'Procfile' (niet 'Procfile.txt' en ook niet 'procfile'), waarin dit staat:

web: node ./<jouw subdirectory>/<jouw Javascript file>

Voeg deze files toe aan de Git repository (git add, git commit) en push ze naar GitHub (git push).

Doe opnieuw op Heroku in je app een Deploy Branch en als het goed is moet het nu wel lukken.

1. Test of je app werkt met Postman, maar geef nu de URL op die Heroku aan je app heeft toegewezen. Werkt je app op Heroku? Beschrijf het resultaat.

# Opdracht 6

De app die je zojuist hebt gemaakt mist nog een aantal dingen:

* De response is gewoon tekst en nog geen JSON.
* De server luistert op port 3000 (hardcoded) maar zou eigenlijk naar environment variabele PORT op Heroku moeten luisteren.

Om de output in JSON formaat te versturen kun je gebruik maken van code zoals deze:

app.get(**'/json'**, **function**(request, response) {  
 response.json({  
 **'some\_name'**: **'Value'**,  
 **'an\_array\_of\_objects'**: [  
 {  
 **'another\_name'**: **'Another value'**,  
 **'a\_further\_name'**: **"A further value"** },  
 {**'yet\_another\_name'**: **'Yet another value'**}  
 ],  
 **'some\_boolean'**: **true**,  
 **'some\_integer'**: 42,  
 **'array\_of\_ints'**: [  
 2, 3, 5, 7, 11, 13  
 ],  
 **'array\_of\_strings'**: [  
 **"twee"**, **"drie"**, **"vijf"**, **"zeven"** ]  
 })  
})

*Tip: Je kunt ook gebruik maken van de Javascript JSON.stringify() methode in combinatie met response.send().*

*Tip: De specificatie van wat een JSON bericht kan bevatten kun je vinden op:*

<http://www.json.org/>

Om je server te laten luisteren naar het juiste poortnummer op Heroku (en niet altijd naar 3000) pas je je code als volgt aan:

**var** port = process.**env**.PORT || 3000;

… (overige code)

app.listen(**port**, **function**() {  
 **console**.log(**'Server app is listening on port ' + port**);  
})

Variabele port krijgt zo de waarde van environment variabele PORT, of als die niet bestaat de waarde 3000 als voorheen.

1. Ga na dat je app nu JSON output verstuurt op port 80 (de default voor HTTP).

# Opdracht 7

In deze laatste opdracht maak je je RESTful web service helemaal af om hem te laten samenwerken met je Android Contact Card App die je in de vorige periode hebt gemaakt.

Haal de JSON data van pakweg 3 tot 5 random gebruikers uit <http://randomuser.me/api> en zet die in je REST server app (hardcoded, dat is voor dit moment voldoende).

1. Pas de code zo aan dat telkens de data van één van die gebruikers in JSON formaat wordt teruggestuurd op een GET request naar het pad ‘/api’ van je app, net als bij <http://randomuser.me/api>. Als je Android app kan selecteren op gebruikers (b.v. alleen Nederlandse gebruikers of zoeken op naam), pas dan je REST server code zo aan dat ook die selectie in de API werkt.  
   Neem je gewijzigde code op in je verslag (niet de hele file, alleen het belangrijke stuk).
2. Deploy je wijzigingen op Heroku en test met Postman. Zet een screenshot van Postman in je verslag om te laten zien dat het werkt.
3. Pas je Android app zo aan dat hij niet met <http://randomuser.me/api> maar met je eigen app op Heroku verbinding maakt. Verifieer met je Android app dat het werkt en dat je gebruikersgegevens uit je eigen app op Heroku kunt laten zien. Voeg bijvoorbeeld een fictieve nieuwe gebruiker in die niet in randomuser.me zit maar alleen in je eigen app op Heroku.

Demonstreer in het volgende practicum je Android app aan de begeleidende docent.

Succes!