SCRIPTING2: NodeJS & Server-side Development

Versie: 2.5 (2025)
Opleiding: Creative Software Developer (MBO 4)
Jaar: 2

Inhoudsopgave

Les 1: Introductie, Setup & Je Eerste Server

1.1 Wat is Node.js?

Node.js is een **runtime environment** voor JavaScript. Tot nu toe heb je JavaScript vooral *client-side* gebruikt (in de browser). Met Node.js kun je JavaScript *server-side* draaien. Dit betekent dat je toegang hebt tot het bestandssysteem van de server, databases kunt aanroepen en volledige back-end applicaties kunt bouwen.

Belangrijke begrippen:

- **V8 Engine:** Dezelfde engine die Google Chrome aandrijft, zorgt ervoor dat jouw JS razendsnel wordt vertaald naar machinecode.
- **Single-threaded & Non-blocking:** Node.js kan heel veel verzoeken tegelijk aan door niet te wachten (non-blocking) tot een taak klaar is, maar door asynchroon te werken.
- **NPM (Node Package Manager):** De grootste software registry ter wereld, waar je bibliotheken en tools van andere developers kunt downloaden.

1.2 Installatie & Setup

Op school staan de machines correct geïnstalleerd. Voor je eigen laptop (BYOD):

1. Ga naar nodejs.org.
2. Download de **LTS (Long Term Support)** versie (v20 of v22).
3. Installeer Node.js.

Project Setup (Standaard voor elk project):
Node.js ondersteunt tegenwoordig volledig de industriestandaard ES Modules (import / export). Wij leren je direct deze moderne standaard aan.

1. Maak een map aan en open deze in VS Code.
2. Open de terminal en typ: npm init -y
3. **BELANGRIJK:** Open package.json en voeg "type": "module", toe.

1.3 Opdracht: Hello World Server

We gaan een server bouwen *zonder* frameworks, om te begrijpen hoe het "onder de motorkap" werkt. Maak een bestand server.js.

```
// server.js
// 1. We importeren de ingebouwde 'http' module van Node.js
import http from 'node:http';

const hostname = '127.0.0.1'; // Localhost
const port = 3000;            // De poort waarop we luisteren

// 2. We maken de server aan. Deze functie wordt uitgevoerd bij ELK verzoek
(request).
const server = http.createServer((req, res) => {

    // 3. We geven aan dat het verzoek succesvol was (200 OK)
    res.statusCode = 200;

    // 4. We vertellen de browser dat we platte tekst terugsturen
    res.setHeader('Content-Type', 'text/plain');

    // 5. We sturen de daadwerkelijke tekst en sluiten de verbinding
    res.end('Hello World vanuit Node.js!');
});

// 6. Start de server en laat hem luisteren op poort 3000
server.listen(port, hostname, () => {
    console.log(`Server draait op http://${hostname}:${port}/`);
});
```

Wat gebeurt hier?

- http.createServer: Dit is de kern. Hier maak je een "luisteraar" die wacht op bezoekers. De callback functie (req, res) heeft twee parameters:
  - req (Request): Informatie over wat de bezoeker vraagt (welke URL? Welke browser?).
  - res (Response): Het antwoord dat wij terugsturen naar de bezoeker.

1.4 Opdracht: Routing & Bestanden Serveren

Een server die alleen tekst terugstuurt is saai. We willen HTML bestanden tonen! Omdat Node.js *server-side* is, kunnen we bestanden van de harde schijf lezen met de fs (File System) module.

Stappen:

1. Maak een bestand index.html met wat HTML content (bijv. <h1>Welkom!</h1>).
2. Maak een bestand about.html met informatie over jezelf.
3. Maak een bestand 404.html met een foutmelding.
4. Pas server.js aan:

```
import http from 'node:http';
import fs from 'node:fs'; // fs = File System module

const hostname = '127.0.0.1'; // Localhost
const port = 3000;            // De poort waarop we luisteren

const server = http.createServer((req, res) => {
    // We kijken naar de URL die opgevraagd wordt (req.url)

    if (req.url === '/') {
        // HOME PAGINA
        // We lezen het bestand 'index.html' van de harde schijf
        fs.readFile('index.html', (err, data) => {
            if (err) {
                // Ging er iets mis met lezen? (Bv. bestand bestaat niet)
                res.writeHead(500);
                res.end('Server Error');
            } else {
                // Gelukt! Stuur de HTML data terug naar de browser
                res.writeHead(200, {'Content-Type': 'text/html'});
                res.end(data);
            }
        });

    } else if (req.url === '/about') {
        // ABOUT PAGINA
        fs.readFile('about.html', (err, data) => {
            if (!err) {
                res.writeHead(200, {'Content-Type': 'text/html'});
                res.end(data);
            }
        });

    } else {
        // 404 - PAGINA NIET GEVONDEN
        fs.readFile('404.html', (err, data) => {
            res.writeHead(404, {'Content-Type': 'text/html'});
            res.end(data);
        });
    }
});

server.listen(port, hostname, () => {
```

```
    console.log(`Server draait op http://${hostname}:${port}/`);
});
```

Wat gebeurt hier?

- import fs from 'node:fs': We laden de module in die bestanden kan lezen en schrijven.
- if (req.url === '/'): We checken handmatig welke pagina de bezoeker wil zien.
- fs.readFile('bestand', (err, data) => ...): Dit leest het bestand *asynchroon* (op de achtergrond). Als het bestand gelezen is, wordt de functie uitgevoerd.
    - data: Dit is de ruwe inhoud van je HTML bestand.
    - res.end(data): We sturen die ruwe HTML direct naar de browser.

1.5 Opdracht: Introductie NPM & Lodash

Naast ingebouwde modules zoals http en fs, is de kracht van Node.js het enorme ecosysteem van **NPM** (Node Package Manager). Hierop staan meer dan 2 miljoen packages van andere developers die jij mag gebruiken.

Om te oefenen gebruiken we **Lodash**, een populaire library met handige hulpfuncties.

1. Open je terminal en installeer lodash: npm install lodash
   (Zie je dat er nu een node_modules map is bijgekomen?)
2. Maak een bestand test-lodash.js:

```
// We importeren de default export van lodash en noemen hem '_'
// Dit is een wereldwijde afspraak (conventie) onder developers.
import _ from 'lodash';

const studenten = ['Piet', 'Klaas', 'Janneke', 'Sanne', 'Mo'];

// 1. Shuffle: Husselt een array (lastig in standaard JS, simpel met
Lodash)
const gehusseld = _.shuffle(studenten);
console.log('Gehusseld:', gehusseld);

// 2. Kebab Case: Zet tekst om naar url-vriendelijk-formaat
const titel = "Hallo dit is een Title Voor Een Blogpost";
const urlSlug = _.kebabCase(titel);
console.log('URL Slug:', urlSlug);
// Output: hallo-dit-is-een-title-voor-een-blogpost
```

3. Draai het bestand: node test-lodash.js

Wat gebeurt hier?

- npm install: Downloadt de code van internet en zet het in node_modules.
- import _ from 'lodash': We laden de externe code in ons bestand.
- _.shuffle: Een functie uit de lodash bibliotheek die we nu gratis kunnen gebruiken!

📺 Handige Video's voor Les 1

- **Wat is Node.js? (1 minuut)** - Een supersnelle introductie.
  - YouTube: Node.js in 100 Seconds (Fireship)
- **Node.js Tutorial voor Beginners** - Een uitgebreide, moderne crash course.
  - YouTube: Node.js Crash Course (Net Ninja)
- **NPM Tutorial** - Hoe werkt NPM en package.json?
  - YouTube: NPM Crash Course (Traversy Media)

Les 2: ExpressJS Framework

Zoals je in Opdracht 1.4 merkte, is het handmatig uitlezen van bestanden en checken van elke URL (if...else if...else) veel werk en foutgevoelig. Daarom gebruiken we **Frameworks**. Het populairste framework voor Node.js is **Express**.

2.1 Installatie

Installeer Express in je projectmap: npm install express

2.2 Opdracht: De server herbouwen met Express

Herschrijf je code uit Les 1 naar Express.

```
import express from 'express';

const app = express(); // We maken een Express applicatie aan
const port = 3000;

// Middleware om statische bestanden (plaatjes, css) automatisch te
serveren
// Als je een map 'public' maakt, kan iedereen bij de bestanden daarin.
app.use(express.static('public'));

// Route voor Home ('/')
app.get('/', (req, res) => {
    // res.send stuurt direct tekst of HTML terug
    res.send('<h1>Welkom Home!</h1>');
});

// Route voor About ('/about')
app.get('/about', (req, res) => {
    res.send('<h1>Over Ons</h1><p>Met Express gaat dit veel sneller.</p>');
});

// Route voor JSON API ('/api/users')
app.get('/api/users', (req, res) => {
    const users = [
        { id: 1, name: 'Piet' },
        { id: 2, name: 'Jan' }
    ];
    // Express ziet dat dit een Array/Object is en maakt er automatisch
JSON van!
    res.json(users);
});

// 404 Afhandeling (Als geen enkele route hierboven matcht)
app.use((req, res) => {
    res.status(404).send('Oeps, 404! Pagina niet gevonden.');
});

app.listen(port, () => {
    console.log(`Express server draait op http://localhost:${port}`);
});
```

Wat gebeurt hier?

- app.get('/', ...): Express neemt het if (req.url === '/') werk van je over. Veel leesbaarder!
- res.send(): Een slimme functie die zelf kijkt of je tekst of HTML stuurt en de headers goed zet.
- res.json(): Zet JavaScript objecten direct om naar JSON string die API's gebruiken.

2.3 Opdracht: Environment Variables (Security)

In productie mag je nooit poortnummers of API-keys "hardcoden" (vast in de code zetten). Als je je code op GitHub zet, kan iedereen je wachtwoorden zien! We gebruiken hiervoor .env bestanden.

1. Installeer dotenv: npm install dotenv
2. Maak een bestand .env aan in de root:

```
PORT=8000
SECRET_KEY=MijnGeheimeSleutel123
```

3. Pas server.js aan:

```
import express from 'express';
// Importeer dotenv config, dit laadt de .env variabelen in
import 'dotenv/config';

const app = express();
// process.env bevat nu jouw variabelen uit het bestand
const port = process.env.PORT || 3000; // Gebruik 3000 als fallback

app.listen(port, () => {
    console.log(`Server draait op poort ${port}`);
});
```

Vergeet niet .env toe te voegen aan je .gitignore bestand!

📺 Handige Video's voor Les 2

- **ExpressJS Crash Course** - Alles wat je moet weten over Express in één video.
  - YouTube: Express JS Crash Course (Web Dev Simplified)
- **Environment Variables (.env)** - Waarom en hoe gebruik je dotenv?
  - YouTube: Hiding API Keys with Environment Variables (The Coding Train)

Les 3: NPM & Modules (Modern)

3.1 NPM Gebruiken

Je hebt al express en dotenv geïnstalleerd. Op npmjs.com staan meer dan 2 miljoen packages.

3.2 Opdracht: Je eigen module maken (ESM)

In de beroepspraktijk knip je code op in kleine stukjes (modules) zodat je bestanden niet te groot worden.

1. Maak een bestand utils/math.js:

```
// utils/math.js

// 'export' maakt deze functie beschikbaar voor andere bestanden
export function telOp(a, b) {
    return a + b;
}

export const PI = 3.14159;
```

2. Gebruik deze in je server.js:

```
// We importeren specifieke onderdelen tussen { }
import { telOp, PI } from './utils/math.js';

console.log(`5 + 5 = ${telOp(5, 5)}`);
```

3.3 Opdracht: Een Package maken (Advanced)

Je gaat nu een package maken die je in andere projecten kunt hergebruiken.

1. Maak een nieuwe map my-api-helper buiten je huidige project.
2. Draai npm init -y en voeg "type": "module" toe.
3. Maak index.js met functies (zie Les 4 voor de code).
4. **npm link:** In plaats van publiceren naar het internet (wat spam oplevert), linken we hem lokaal.
    - In de map my-api-helper: run het commando npm link.
    - In je project map: run het commando npm link my-api-helper.
5. Nu kun je in je project import { ... } from 'my-api-helper' doen!

📺 Handige Video's voor Les 3

- **ES Modules vs CommonJS** - Uitleg over de moderne import syntax vs de oude require.
    - YouTube: ES Modules in Node.js (Web Dev Simplified)
- **NPM Tutorial** - Hoe werkt NPM en package.json?
    - YouTube: NPM Crash Course (Traversy Media)

Les 4: Async Data & API Calls

Node.js heeft tegenwoordig **native fetch** ingebouwd. We gaan leren werken met **Async/Await**. Dit is de moderne manier om met data te werken die "even duurt" (zoals downloaden van internet) zonder je programma te blokkeren.

4.1 Opdracht: Async/Await & Try/Catch

Stel we willen data ophalen van een externe API (bijvoorbeeld JSONPlaceholder).

```
// dataService.js

// 'async' vertelt JS: "Deze functie duurt even, wacht niet op mij"
export async function getPosts() {
    try {
        // 'await' pauzeert DEZE functie tot de data binnen is
        const response = await
fetch('[https://jsonplaceholder.typicode.com/posts](https://jsonplaceholder
.typicode.com/posts)');

        // Check of het gelukt is (Status 200-299)
        if (!response.ok) {
            throw new Error(`HTTP fout! Status: ${response.status}`);
        }

        // De body komt binnen als tekststream, zet om naar JSON
        const data = await response.json();
        return data;

    } catch (error) {
        // Hier komen we als er iets mis ging (bv. geen internet)
        console.error('Er ging iets mis:', error.message);
        return [];
    }
}
```

Wat gebeurt hier?

- async: Maakt de functie asynchroon.
- await: Wacht tot de Promise (belofte van data) is ingelost. Zonder await zou je variabele response leeg zijn omdat de code doorgaat voordat de download klaar is.
- try / catch: Een veiligheidsnet. Als er *iets* fout gaat in het try blok, crasht je server niet, maar springt hij naar catch.

4.2 Opdracht: De API Helper Package (Eindopdracht voorbereiding)

Jouw custom package uit Opdracht 3.3 moet functionaliteit krijgen om data op te halen (GET), te versturen (POST) én te verwijderen (DELETE).

Gebruik onderstaande code in je index.js van je my-api-helper package:

```
// index.js van jouw 'my-api-helper' package

// Helper voor GET requests
export async function getData(url) {
    try {
        const response = await fetch(url);
        if(!response.ok) throw new Error('Fetch failed');
        return await response.json();
    } catch (err) {
        console.error('Fout bij ophalen:', err);
        return null;
    }
}

// Helper voor POST requests (data versturen)
export async function postData(url, data) {
    try {
        const response = await fetch(url, {
            method: 'POST',
            headers: {
                // Vertel de API dat we JSON sturen
                'Content-Type': 'application/json'
            },
            // Zet het JS object om naar tekst
            body: JSON.stringify(data)
        });
        if(!response.ok) throw new Error('Post failed');
        return await response.json();
    } catch (err) {
        console.error('Fout bij versturen:', err);
        return null;
    }
}

// Helper voor DELETE requests (data verwijderen)
export async function deleteData(url) {
```

```
    try {
        const response = await fetch(url, {
            method: 'DELETE'
        });
        if(!response.ok) throw new Error('Delete failed');
        return true; // Succesvol verwijderd
    } catch (err) {
        console.error('Fout bij verwijderen:', err);
        return false;
    }
}
```

📺 Handige Video's voor Les 4

- **Async Await in 10 minuten** - Heldere uitleg over het oplossen van Promises.
  - YouTube: JavaScript Async Await (Web Dev Simplified)
- **Native Fetch API** - Hoe gebruik je fetch (zonder externe libraries).
  - YouTube: Learn Fetch API In 6 Minutes (Web Dev Simplified)

Les 5: Eindopdracht REST API

Opdracht: Bouw een **REST API** met Express.js.

Casus

Je mag zelf een onderwerp kiezen (Takenlijst, Webshop, Game Highscores, etc.). Je bouwt *geen* frontend, puur de back-end. Je test met Postman of Thunder Client.

Eisen

1. **Framework:** Gebruik **Express.js**.
2. **Architectuur:** Gebruik **ES Modules** (import/export) en "type": "module".
3. **Configuratie:** Gebruik een **.env** bestand voor poortnummer. Commit deze NIET.
4. **Custom Module:** Gebruik je eigen NPM package (uit Opdracht 3.3/4.2) via npm link.
5. **Routes:**
    - Minimaal **2 GET routes**.
    - Minimaal **1 POST route**.
    - Minimaal **1 DELETE route**.
6. **Code Kwaliteit:** Gebruik **async/await** en **try/catch**.

Beoordeling (Rubric)

Onderdeel	Onvoldoende	Matig	Voldoende	Goed
Setup & Config	.env ontbreekt of node_modules in Git.	Rommelige structuur of hardcoded config.	Node/Express correct, .env gebruikt.	.gitignore correct, logische mappenstructuur.
Routes & Logica	< 3 werkende routes.	Routes werken deels of bevatten bugs.	Alle methodes (GET, POST, DELETE) werken.	Correcte HTTP status codes (200, 201, 404, 500) in alle situaties.
Code Kwaliteit	Veel callbacks/var/require.	Inconsistente code, weinig error handling.	const/let, ESM & code werkt.	Modern async/await & nette try/catch blokken.
Custom Package	Geen eigen module.	Module is triviaal of bevat fouten.	Module werkt via npm link.	Module is abstract, herbruikbaar & bevat commentaar.

📺 Handige Video's voor Les 5

- **REST API Bouwen** - Een complete walkthrough van het bouwen van een API.
  - YouTube: Build a REST API with Express (Programming with Mosh)

Inleveren

ZIP-bestand via Teams (ZONDER node_modules!).