IOT Dashboard in NodeJS

Hamza Outataouine

Inhoud

[1 Installatie NodeJS 3](#_Toc91089858)

[2 Wat is NodeJS 3](#_Toc91089859)

[2.1 Voordelen 3](#_Toc91089860)

[2.2 Nadeel 3](#_Toc91089861)

[3 Npm 4](#_Toc91089862)

[3.1 Hoe een package installeren 4](#_Toc91089863)

[4 Starten aan IOT opdracht 4](#_Toc91089864)

[4.1 Map aanmaken en modules installeren 4](#_Toc91089865)

[4.2 Scripts aanmaken 4](#_Toc91089866)

[4.3 Simpele server 5](#_Toc91089867)

[5 Een site hosten 7](#_Toc91089868)

[6 Database 8](#_Toc91089869)

[6.1 mongoDB installeren 8](#_Toc91089870)

[6.2 Robo3T installeren 9](#_Toc91089871)

[6.3 Connectie met database via nodeJS 9](#_Toc91089872)

[6.4 schema maken voor database via nodeJS 10](#_Toc91089873)

[6.5 Database connectie implementeren in de server 11](#_Toc91089874)

[7 API 12](#_Toc91089875)

[7.1 Basisstructuur endpoint 12](#_Toc91089876)

[7.2 Werken met postman 12](#_Toc91089877)

[7.3 POST 14](#_Toc91089878)

[7.3.1 POST endpoint voor sensoren 14](#_Toc91089879)

[7.3.2 Example 15](#_Toc91089880)

[7.4 GET 16](#_Toc91089881)

[7.4.1 GET endpoint voor de site 16](#_Toc91089882)

[7.4.2 Testing met postman 17](#_Toc91089883)

[8 Client side javascript 19](#_Toc91089884)

[8.1 Html componenten aanmaken 19](#_Toc91089885)

[8.1.1 Form aanmaken 19](#_Toc91089886)

[8.1.2 Tabel aanmaken 19](#_Toc91089887)

[8.1.3 Grafiek aanmaken 19](#_Toc91089888)

[8.2 Javascript schrijven 21](#_Toc91089889)

[8.2.1 JS voor de form 21](#_Toc91089890)

[8.2.2 JS voor de tabel 22](#_Toc91089891)

[8.2.3 JS voor de grafiek 23](#_Toc91089892)

[8.3 alles samenvoegen 25](#_Toc91089893)

# Installatie NodeJS

1. Ga naar de NodeJS site en installeer NodeJS. (<https://nodejs.org/en/>) welke van de twee maakt niet uit.
2. Herstart je computer.
3. Open een command prompt en run “node -v”, normaal wordt dan de versie getoond. Vervolgens run “npm -v”, weeral moet je de versie zien. Als je deze niet ziet dan is er iets fout gelopen tijdens de installatie.
4. Maak een file aan genaamd “helloWrld.js”
5. Schrijf in deze file: “console.log(“hello world”)
6. open je command prompt en navigeer naar de locatie van de file.
7. Afbeelding met tekst

   Automatisch gegenereerde beschrijvingRun de commando “node helloWrld.js”. Normaal zie je nu hello world in de console.

# Wat is NodeJS

NodeJS is een runtime enviroment voor server side scripting alsook singlepage web-apps, en is uitermate geschikt voor API’s en real time apps zoals chatrooms Het is single threaded en werkt asynchroon met als voordeel dat het taken gelijktijdig kan uitvoeren.

## Voordelen

1. Schaalbaar
2. Snelheid
3. Grote community
4. Multi-platform
5. Handig: de front- en backend zijn in javascript

## Nadeel

1. Doordat het single threaded is, is NodeJS niet bedoeld voor zware wiskundige taken
2. Door de asynchrone natuur kan de code zeer complex worden. (callback hell)
3. De packages(libraries) voor Node worden niet gereguleerd, met als gevolg dat er geen zekerheid is als de package wel werkt of zelfs veilig is.

# Npm

Npm is zoals pip bij python. Het is een package manager waarmee modules geïnstalleerd worden. Modules kunnen gezien worden als libraries.

## Hoe een package installeren

1. De commando om iets te installeren is “npm i <package name>”
2. Afbeelding met tekst

   Automatisch gegenereerde beschrijvingBv. “Npm i chalk”

# Starten aan IOT opdracht

## Map aanmaken en modules installeren

1. Maak een map aan om in te werken
2. Navigeer naar deze map in je command line
3. Voeg deze drie modules toe: express, nodemon, env-cmd, mongoose, body-parser
4. Afbeelding met tekst

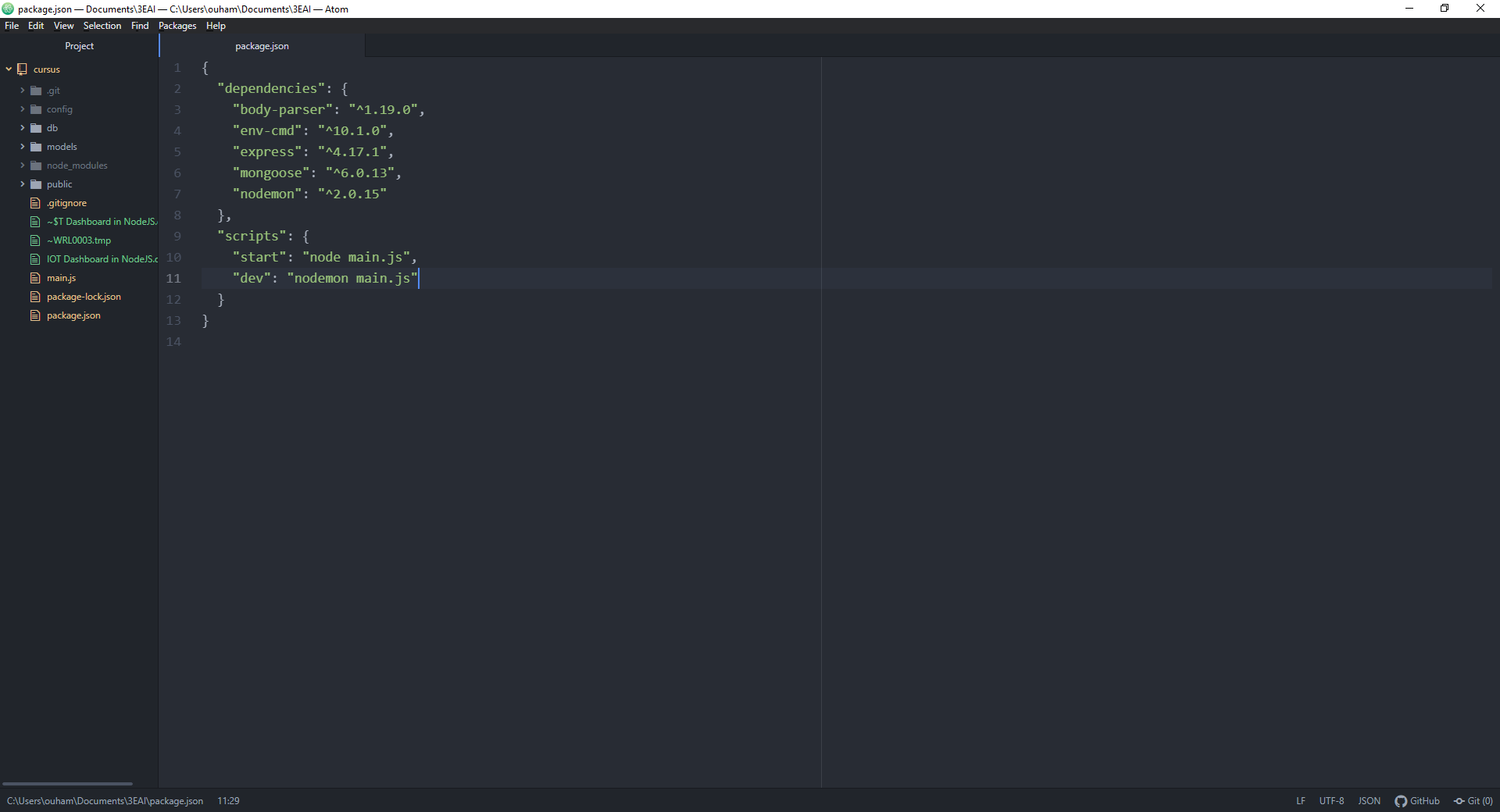
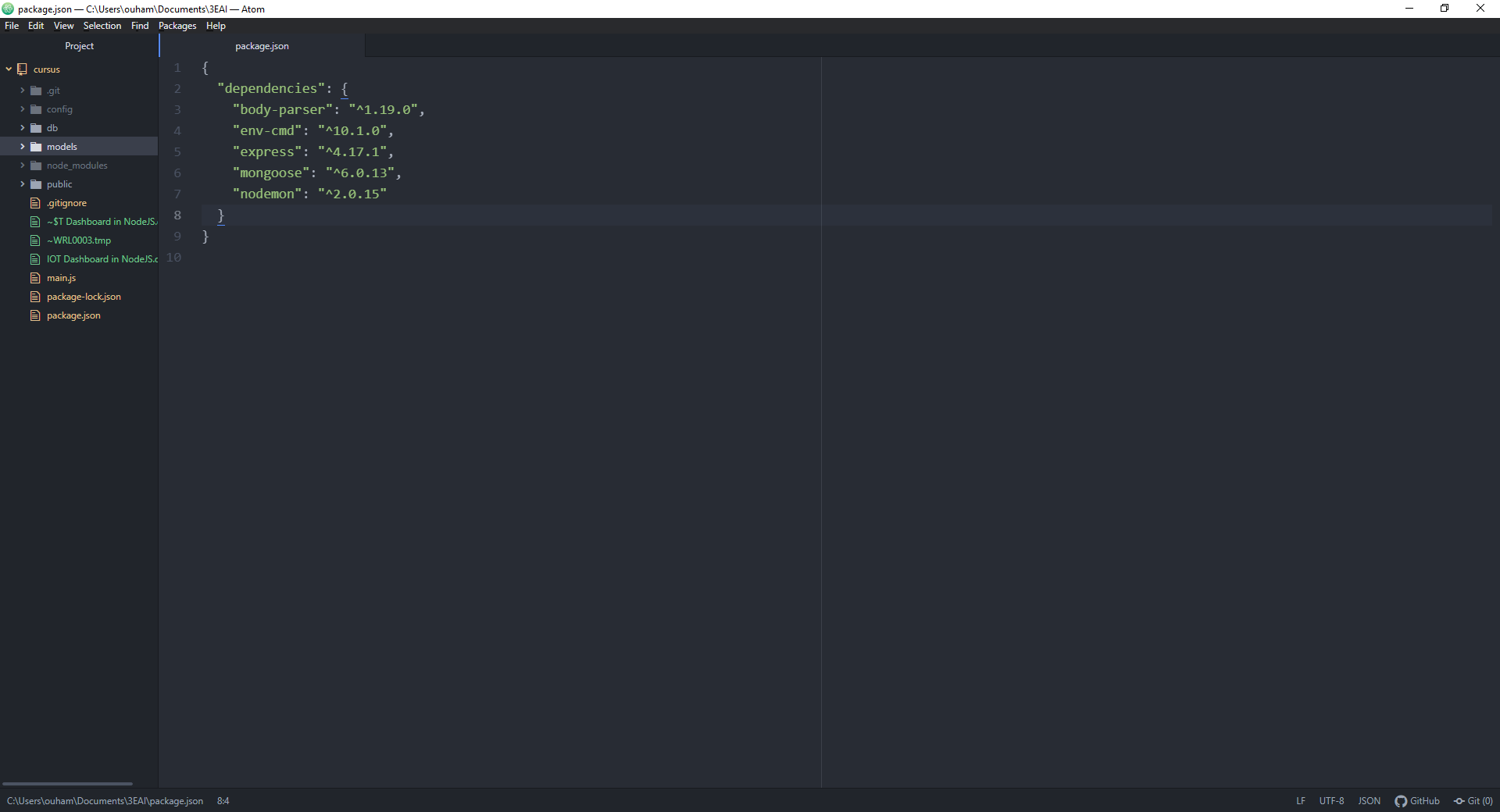
   Automatisch gegenereerde beschrijvingMaak een js file aan om in te werken bv. Main.js

I.p.v. “npm i” meerdere keren te runnen, kun je ook gewoon alle modules achter elkaar schrijven om ze zo in één keer te installeren.

## Scripts aanmaken

In Node kun je scripts maken waarmee je makkelijk je programma’s kunt runnen. Één van de modules die geinstaleerd is, is Nodemon. Nodemon zorgt ervoor dat de programma heropstart bij elke save. De commando om Nodemon te runnen is “nodemon <filename>”. Met een script kun je specifieke files runnen of files met andere enviroment variables runnen. Enviroment variables worden later besproken.

1. Bij het installen van de modules zijn er twee json files aangemaakt.
2. Open package.json
3. Voeg een object toe genaamd “script”
4. Geef je script een naam alsook wat de commando is die uitgevoerd moet worden



Afbeelding met tekst

Automatisch gegenereerde beschrijvingDe “dev” script zal gebruikt worden om alles locally te testen, en de “start” script zal gebruikt worden door Heroku. Heroku zal later gebruikt worden om de website online te zetten. Om de script te runnen type je “npm run <script name>”. Als je nu “npm run dev” uitvoert zal node dit omzetten naar de commando die je geschreven hebt, in deze situatie is dat nodemon main.js

\*om nodemon te stoppen doe je gwn ctrl+C en dan y.

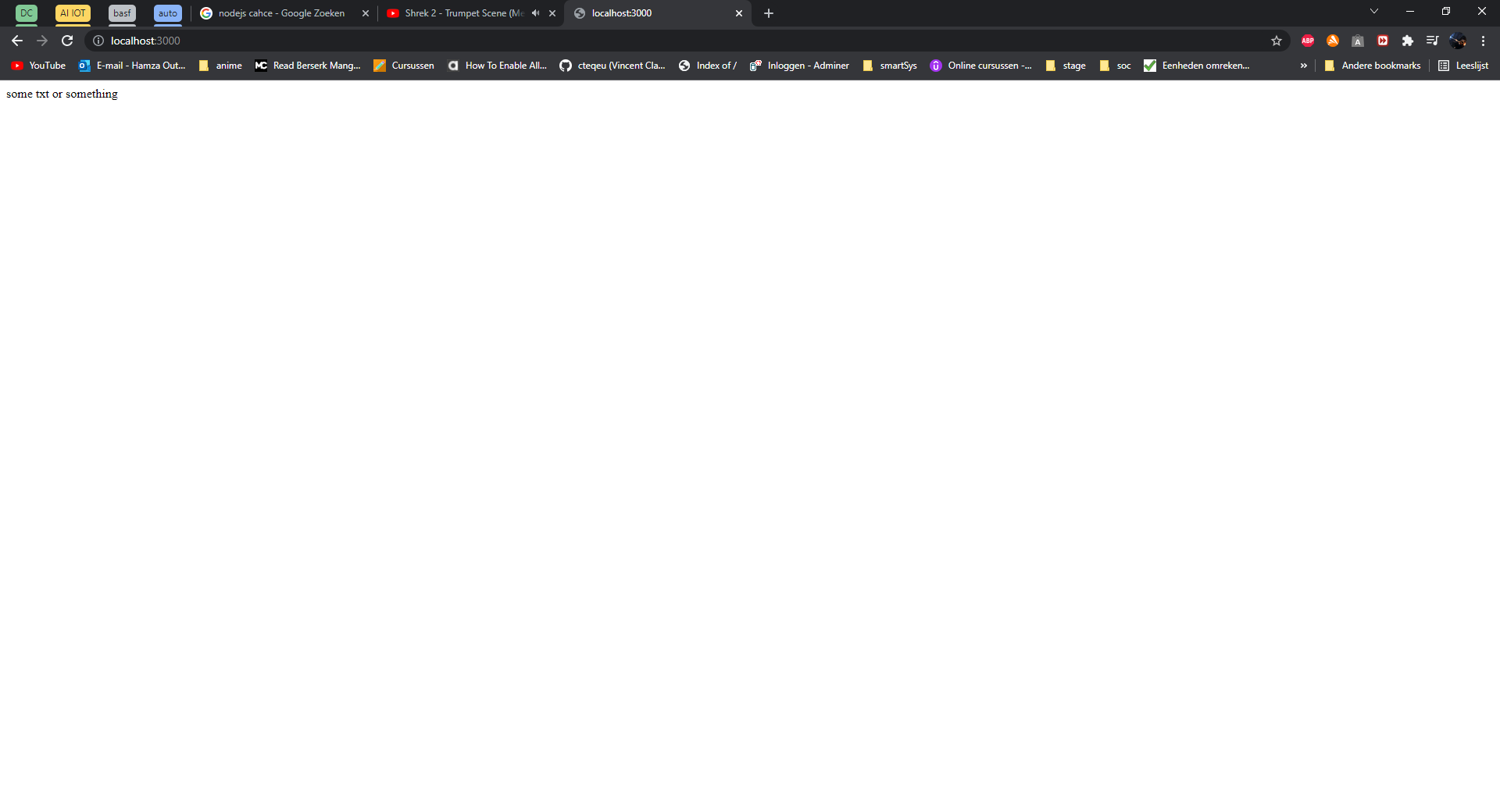
## Simpele server

Een van de modules die geinstaleerd is, is express. Express maakt het makkelijk om servers te maken alsook http endpoints. Een http endpoint is eigenlijk een communicatie punt waar er met de API gepraat kan worden. Bv. “http://sumSite.com/getData”, als er een GET request wordt gedaan naar deze site zal de server antwoorden met data.

Afbeelding met tekst, schermafbeelding, elektronica, scherm

Automatisch gegenereerde beschrijvingcode voor de simpele server:

als je nu naar <http://localhost:3000> surft zal je nu je site zien.



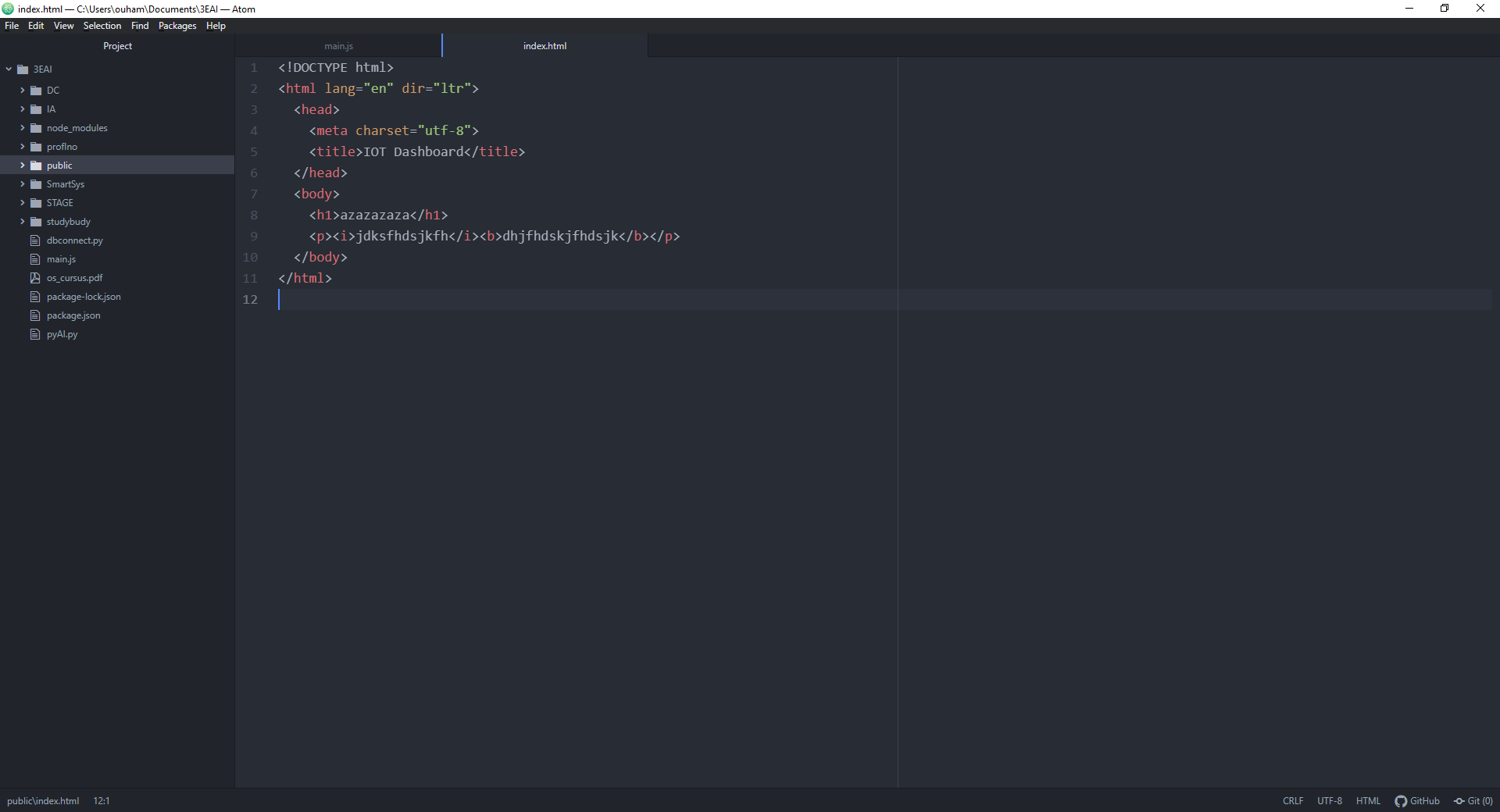
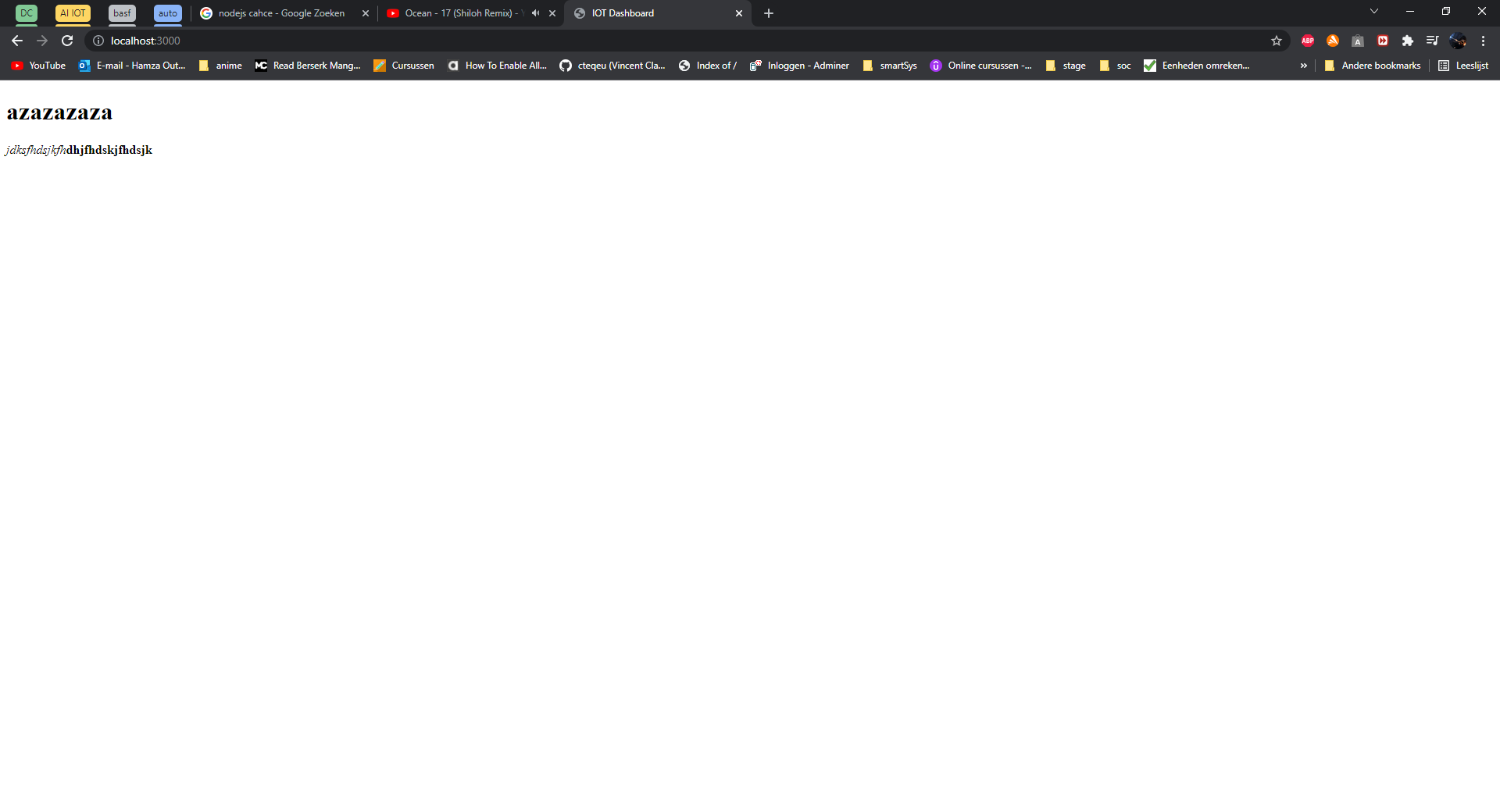
# Een site hosten

Met NodeJS kun je ook een html pagina tonen en de client-side javascript. DE MAP MOET “public” HETEN EN DE HTML BESTAND MOET DE NAAM “index.html” HEBBEN. Want express zoekt naar deze namen.

1. Maak een nieuwe map genaamd public
2. Voeg hier twee files aan toe: “index.html” en “app.js”.
3. In de index maak je een algemene html pagina, en bij app.js hoeft nu nog niks
4. Afbeelding met tekst, schermafbeelding, scherm

   Automatisch gegenereerde beschrijvingVoeg deze lijn code toe aan main.js: app.use(express.static(path.join(\_\_dirname,'/public'))). (hiervoor heb je de path module nodig)

App.use zegt tegen express dat die naar de public map moet gaan en hier de files moet gebruiken. Als je nu je site refreshed dan zal je html pagina getoond worden.



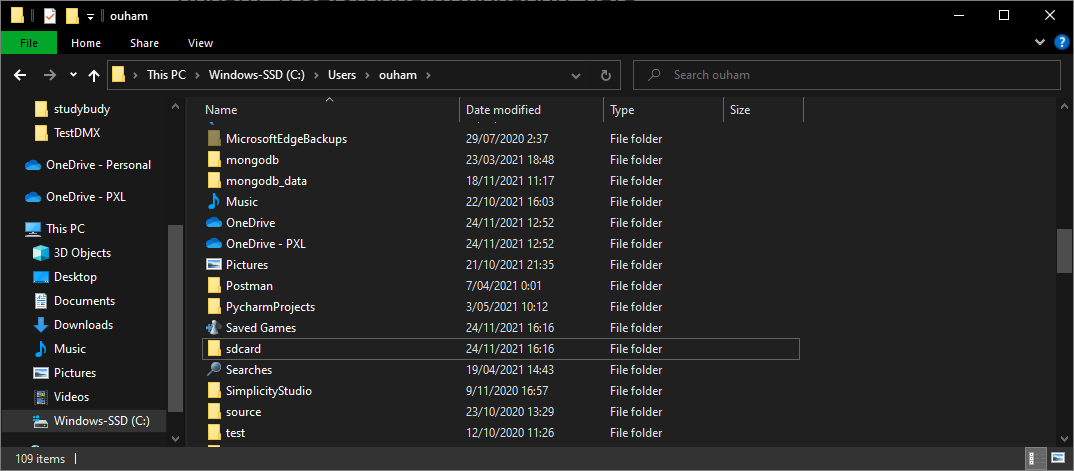
De site bestaat enkel uit html en is niet responsive.

# Database

Om de waardes op te slaan heb je een database nodig. Bij deze project zal mongoDB gebruikt worden. je kan ook een SQL database gebruiken maar ik kies voor mongoDB omdat het veel gemakkelijker werkt met JSON.

## mongoDB installeren

1. ga naar de mongoDB site en download mongoDB server. <https://www.mongodb.com/try/download/community>. Verander de package van msi naar zip
2. open de zip en extract het.
3. Verander de map naam naar iets korter. mongodb-win32-x86\_64-windows-5.0.4 🡪 mongodb
4. Verplaats de map naar de user map. Dus C:\users\<user naam>
5. Voeg nu een map toe mongodb\_data
6. Open nu een terminal
7. Vul nu de path in van de executable dan spatie --dbpath=<pad naar mongodb\_data>. Dus voor mij is dat: /Users/ouham/mongodb/bin/mongod.exe --dbpath=/Users/ouham/mongodb\_data
8. En normaal zal nu de local server starten



Afbeelding met tekst

Automatisch gegenereerde beschrijving

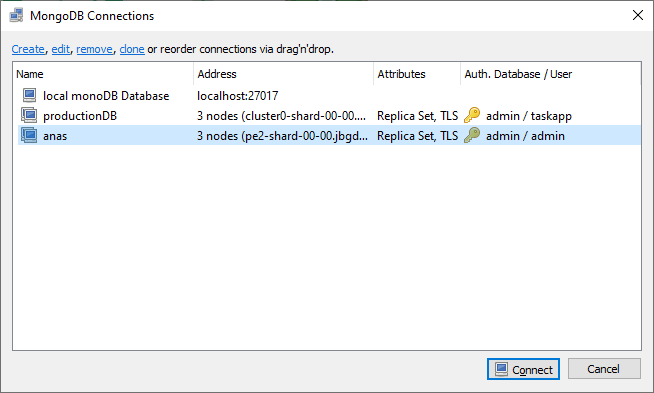
Na deze commando te runnen wordt er veel tekst geprint. Het belangrijkste is dat je ziet dat er staat listening for connections. Om de database af te sluiten doe je CTRL+C.

## Robo3T installeren

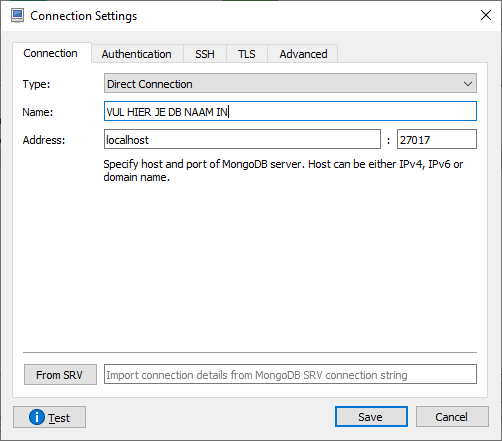
Robo3T is een GUI waarmee je de waardes in de database kan zien alsook de waardes veranderen of juist waardes toevoegen.

1. Download en installeer robo3t. <https://studio3t.com/download/>
2. Wanneer je robo3t opent zie je een connectie pop-up
3. Links boven in het blauw staat er Create, klik hierop en een nieuwe pop-up komt tevoorschijn
4. Onder connection bij name vul je een naam in voor je database. Bv. “Local mongoDB database”. De rest kan je laten want het staat al ingesteld op localhost
5. Klik op de test knop om je connectie te testen. (!! Je mongoDB local server moet wel aanstaan)
6. Als alles ok is kun je klikken op save en vervolgens op connect om met je database te verbinden

\*als de connect pop-up niet tevoorschijn zou komen, klik dan gewoon linksboven op het computer icoon.



1



3

2

4

Je zult zien dat er nog niks instaat maar dat zal snel veranderen.

## Connectie met database via nodeJS

1. Maak een nieuwe map aan db
2. Voeg hier een file toe genaamd mongoose.js

Deze script heeft maar twee lijnen code:

1. const mongoose = require("mongoose")
2. mongoose.connect(“mongodb://127.0.0.1:27017/IOT”)

mongoose.connect verwacht de adres naar de database alsook de naam ervan. De adres van de database is: mongodb://127.0.0.1:27017, en de naam is IOT. Deze worden gesplitst door een forward slash.

## schema maken voor database via nodeJS

Een van de geïnstalleerde packages is mongoose. Mongoose is een module die het makkelijk maakt om met mongoDB te werken. mongoDB werkt met schema’s, dit zijn files die de structuur van de database beschrijven. In dit project wordt de schema in een aparte map gestoken.

1. Maak een nieuwe map aan “models”
2. Afbeelding met tekst, schermafbeelding, monitor, scherm

   Automatisch gegenereerde beschrijvingVoeg hier een javascript file. Naam kies je zelf bv. sensorData.js

Op lijn 5 wordt de schema aangemaakt. De types moeten met hoofdletter geschreven worden.

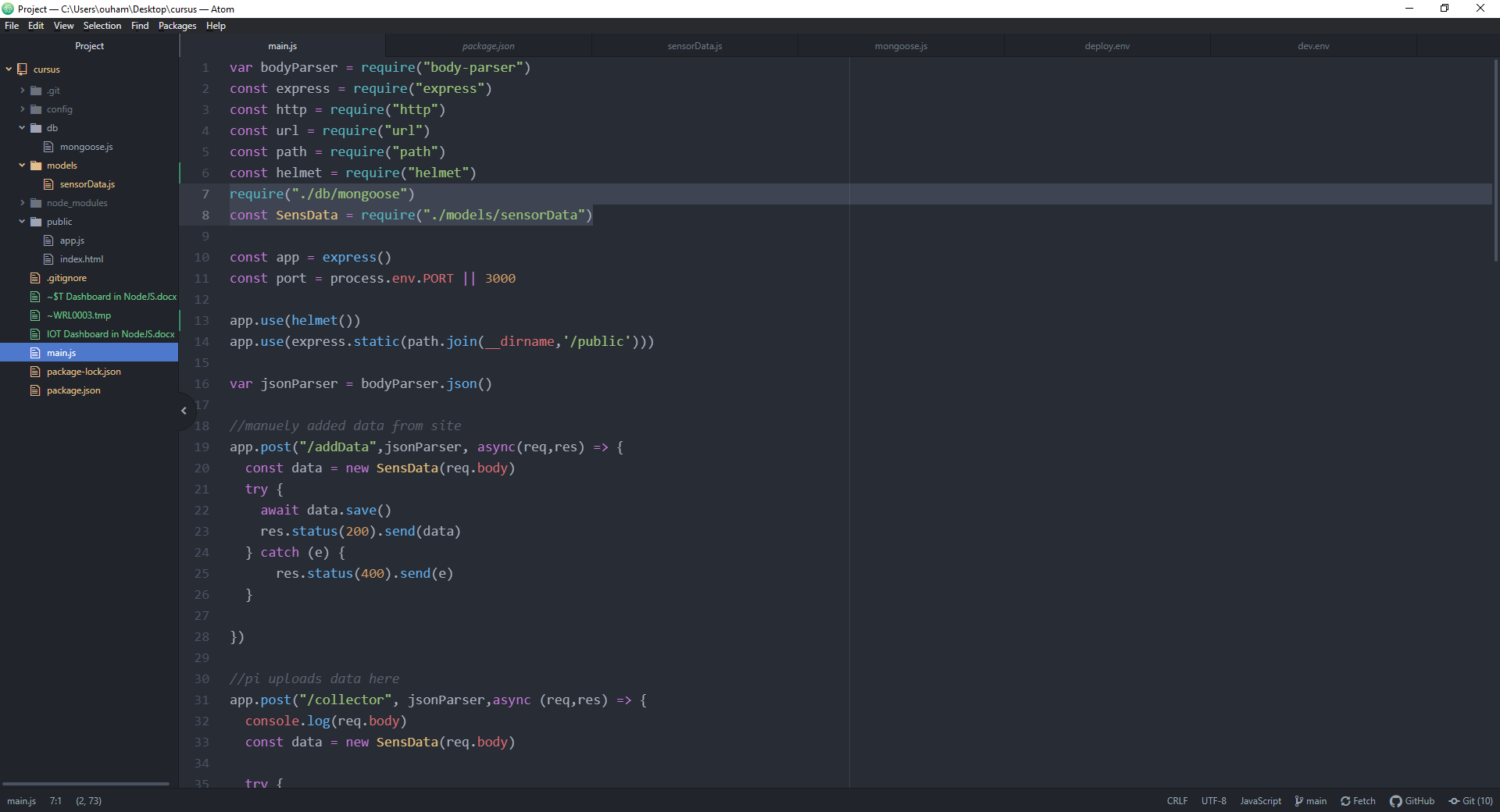
de functie in lijn 23 zorgt ervoor dat de ID en \_\_v field verwijdert worden wanneer er data wordt opgevraagd.

Op lijn 33 wordt er een model aangemaakt met naam SensorData en de schema die er beschreven is.

De laatste regel exporteert de object zodat deze in andere files zoals main.js gebruikt kan worden.

Normaal moet je file tree er nu zo uitzien:

De rode lijnen zijn files die er niet horen zoals deze word bestand.



## Database connectie implementeren in de server

In main.js moeten twee requires toegevoegd worden:

* require("./db/mongoose")
* const SensData = require("./models/sensorData")

Afbeelding met tekst, schermafbeelding, monitor, scherm

Automatisch gegenereerde beschrijvingde tweede require voegt de model object toe. Met dit object kun je data opvragen of in de database steken.

# API

De sensoren moeten hun data naar de database sturen. Dit kan gedaan worden d.m.v. een rest API. Deze hoofdstuk wordt de API geschreven waarmee de dashboard werkt. De twee die HTTP methodes die gebruikt zullen worden zijn GET en POST. Er zijn nog andere methodes maar die hebben hier geen toepassing.

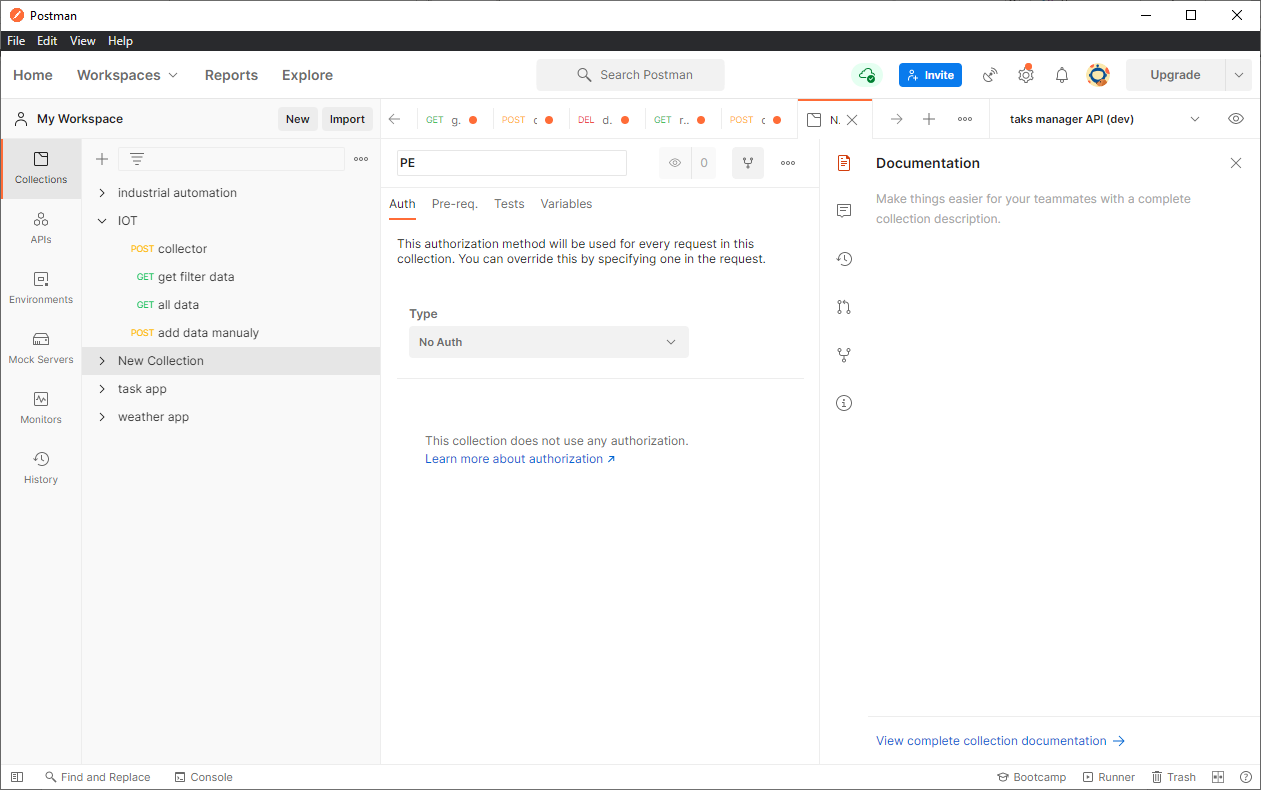
## Basisstructuur endpoint

1. Tussen de aanhalingsteken vul je de naam van de endpoint in
2. In de try-catch block schrijf je de code die er gerund moet worden
3. De res.status().send() kun je laten (moet er altijd zijn, want dit stuurt de waardes terug alsook de http status)

## Werken met postman

Postman maakt het makkelijk om requests te sturen en http endpoints te testen.

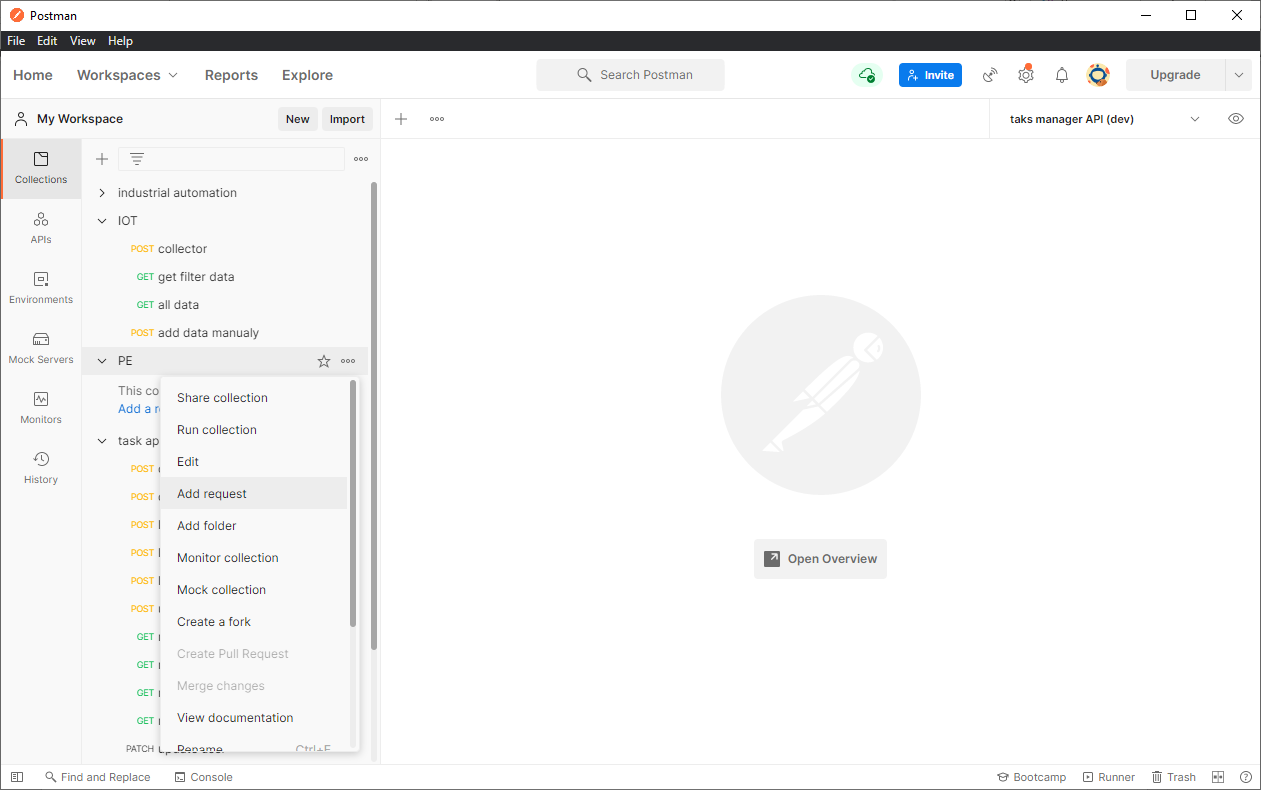
1. Ga naar de site en installeer postman. <https://www.postman.com/downloads/>
2. Open postman en maak een nieuwe collectie



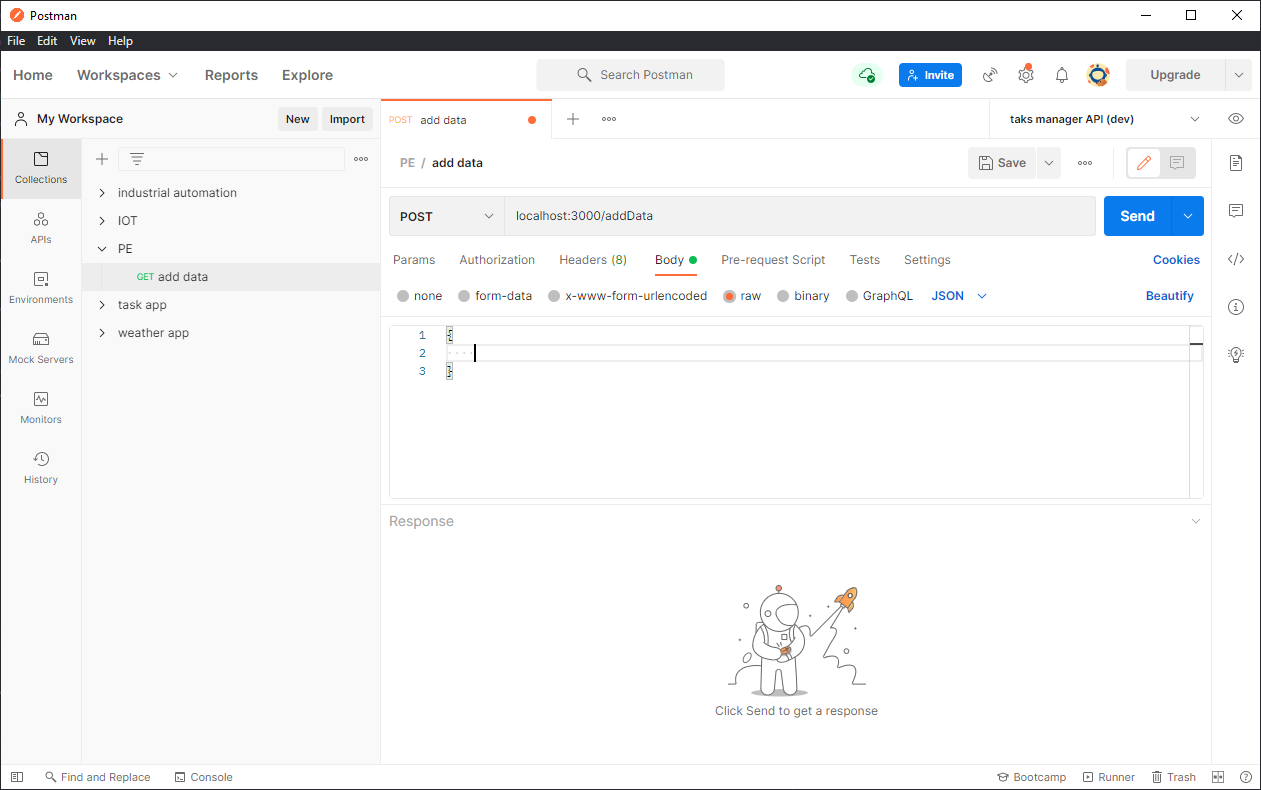
2)Naam voor collectie

1

1. Voeg hier een nieuwe request aan



1. Verander de naam 🡪 verander de methode in post 🡪 klik op body 🡪 vink raw aan 🡪 zet de formaat op JSON 🡪 type twee aanhalingsteken in de body



2

1

3

4

5

6

7

Als je op send klikt zal de request afgevuurd worden en komt er onderaan de response. Maar sinds de endpoint nog niet gemaakt is zal de request failen.

## POST

POST wordt gebruikt om data te sturen. Bij een POST kan je de data encrypteren, je kunt ook de type bepalen zoals XML, JSON of plain text.

### POST endpoint voor sensoren

Afbeelding met tekst, monitor, schermafbeelding, scherm

Automatisch gegenereerde beschrijvingDe sensoren moeten hun waardes uploaden naar de database. Deze POST endpoint zal ervoor zorgen dat de ontvangen waardes in de database gestoken worden. Je kunt de app.get die er al staat aanpassen. Deze endpoint kan ook gebruikt worden voor manueel data via de site in te dienen.

De variabele op lijn 15 parst de json in de body zodat deze makkelijk gebruikt kan worden, deze wordt gewoon toegevoegd aan de functie.

De code op lijn 19 zet de data in de juiste vorm zodat deze in de database gestoken kan worden. het is een constructor functie dat een object verwacht.

In de try blok wordt de data opgeslagen in de database. De await ervoor zorgt dat de programma wacht totdat deze actie voltooit is, anders zou de programma door runnen en weet de user alsook de developer niet als de data daadwerkelijk in de database opgeslagen is.

Als laatste wordt er een 200 code gestuurt alsook de data dat opgeslagen werd. Als er een error zou zijn wordt er een 400 gestuurd met de error. Je kunt ook een eigen message terug sturen.

### Afbeelding met tekst, schermafbeelding, computer, binnen Automatisch gegenereerde beschrijvingExample

De post request wordt gestuurd naar localhost:3000/addData. In de body zit de data in JSON formaat. Onderaan zie je response die de server terugstuurt, daarin zie je de ingevulde gegevens. !!!! je moet nooit gevoelige data terugsturen, in het algemeen moet je enkel iets terugsturen als het daadwerkelijk nodig is, want iedereen die weet hoe de chrome console werkt kan dan deze response zien en misbruiken. Voor developing is het handig maar verwijder de code responses voordat je de code in production zet.

## GET

Een GET request wordt gebruikt om data op te vragen. Je kunt query paramaters meegeven om specifieke data enkel op te vragen zoals: enkel de waardes tussen twee datums of enkel temperatuur terugsturen.

### GET endpoint voor de site

Afbeelding met tekst

Automatisch gegenereerde beschrijvingOm de grafiek en de tabel te maken en/of filteren heb je GET requests nodig, zodat je in de client side javascript de waardes kunt opvragen en gebruiken. Je kunt de bestaande GET request die aangemaakt was in het begin nu aanpassen sinds express nu weet waar het moet zoeken. Als je deze al verwijdert hebt is dat geen probleem en kun je gewoon een nieuwe maken. Je kan meerdere routes in dezelfde script hebben. Zorg er gewoon voor dat ze niet in elkaar zitten.

De structuur is hetzelfde als de post maar de interne code in de try/catch blok verandert.

Op lijn 29 wordt alle data opgevraagd. Door niks door te geven aan find() zal de functie alle waardes kiezen. Ze worden gesorteerd op datum ascending.

Op lijn 31 wordt de status alsook de data terug gestuurd

!!!! important note: bij errors stuur je beter niet de volledige error terug want de klant zal dit niet begrijpen maar belangrijker nog een hacker kan deze error zien en zo informatie krijgen. In development mag het maar zodra je dit in productie zet zorg dan dat er error messages geschreven zijn.

### Testing met postman

1. Maak een nieuwe request aan
2. Geef de request een titel en vul de url in
3. Afbeelding met tekst, schermafbeelding, computer, binnen

   Automatisch gegenereerde beschrijvingSend de request

Als alles goed is verlopen zie je nu alle waardes die in de database staan.

Sinds je enkel data moet opvragen en aanmaken zijn er maar twee http types GET en POST. Waardes worden niet geüpdatet of verwijdert dus heb je enkel deze twee. Het enige wat verandert is de naam van de end-point en de logica in de try/catch blok.

**Tussenstop**

Normaal moet je dit nu hebben in je main. (kon niet via atom naar word dus heb ik het maar via notepad++ gedaan)

const express = require("express")

const path = require("path")

const bodyParser = require("body-parser")

const url = require("url")

require("./db/mongoose")

const SensData = require("./models/sensorData")

const app = express()

const port = process.env.PORT || 3000

app.use(express.static(path.join(\_\_dirname,'/public')))

var jsonParser = bodyParser.json()//parst de body van de request

//manuely added data from site

app.post("/addData",jsonParser, async(req,res) => {

const data = new SensData(req.body)

try {

await data.save()

res.status(200).send(data)

} catch (e) {

res.status(400).send(e)

}

})

//alle waardes opvragen

app.get("/allSensorData", async (req,res) => {

try {

//geef mij alle waardes in stijgende volgorde oud --> nieuw

const data = await SensData.find({}).sort({datum:"asc"})

//stuur waardes door

res.status(200).send(data)

} catch (e) {

//als er een error is stuur deze terug

res.status(400).send(e)

}

})

app.listen(port, () => {

console.log("server is up on port " + port)

})

Je models js file moet hetzelfde zijn gebleven.

# Client side javascript

Op dit moment heb je een droge html site die niet responsive is en waar je niks mee kunt doen. D.m.v. wat javascript kan er een grafiek gemaakt worden, data verstuurt worden en een tabel ingeladen worden. in deze sectie wordt er gewerkt in de index.html en de app.js file waarvan beiden in de public folder staan.

## Html componenten aanmaken

### Form aanmaken

1. Voeg een form toe met twee inputs en een button. Geef elk een ID
2. Onderaan index.html voor de html closing tag voeg je een script tag toe met als src de pad naar “app.js”. (de pad wordt bekeken vanaf index.html)

### Tabel aanmaken

1. Voeg een tabel toe met de theads en de titels ervoor
2. Voeg een tbody toe
3. Geef de body een ID. deze ID zal gebruikt worden om vanuit de client side de tabel dynamisch aan te vullen.

### Grafiek aanmaken

1. Voeg een canvas toe met een ID
2. Voeg onderaan nog een script tag toe maar dan naar <https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js>

Dit is de code voor de index pagina:

1. <!DOCTYPE html>
2. <html lang="en" dir="ltr">
3. <head>
4. <meta charset="utf-8">
5. <title>IOT site</title>
6. </head>
7. <body style="background-color:#8E7575">
8. <canvas id="myChart" style="width:100%;max-width:1500px;background-color:white"></canvas>
9. <h1>send data</h1>
10. <form id="manualData">
11. <label>sensor type</label>
12. <input type="number" id="sensID" name="sensorID">
13. <label>waarde</label>
14. <input type="number" id="sensValue" name="sensorValue">
15. <button id="manual">send</button>
16. </form>
17. <table border="1px solid black">
18. <thead>
19. <tr>
20. <th>sensor ID</th>
21. <th>waarde</th>
22. <th>IP adress</th>
23. <th>datum</th>
24. </tr>
25. </thead>
26. <tbody id="tableBody"></tbody>
27. </table>
28. </body>
29. <script src="app.js"></script>
30. <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script>
31. </script>
32. </html>

De background heeft een simpele kleurtje gekregen en de scripts zijn toegevoegd alsook de forms.

## Javascript schrijven

D.m.v. de ID’s kunnen er variabelen eraan gelinkt worden en kunnen de listeners toegevoegd worden. Een event listener, zoals de naam het zegt, luistert naar een event en voert dat een functie uit. Een voorbeeld van een event is submit. Wanneer een form gesubmit wordt, zal de event listener getriggered worden en de functie runnen.

### JS voor de form

1. Gebruik document.queryselector om via de ID de elementen in een variabele te steken. (het is ook mogelijk om via de form rechtstreeks de waarden eruit te halen)
2. Maak een twee functies aan: één voor de data te sturen naar de server (via fetch() ) en een andere functie die deze oproept wanneer de event gebeurt
3. Voeg aan de variable uit stap 1 een event listener en de event is “submit” en de tweede argument is de naam van de 2de functie uit stap twee.

const sensorForm = document.querySelector("#manualData")

const sensorID = document.querySelector("#sensID")

const waarde = document.querySelector("#sensValue")

//async toevoegen, want fetch is asynchroon

async function sendData(data){

const rawResponse = await fetch("/addData",{

method: "POST",

headers:{

"Accept" : "application/json",

"Content-Type": "application/json"

},

body: JSON.stringify(data)

})

const content = await rawResponse.json()

//console.log(content)

}

function submitManualData(e){

e.preventDefault()

const sensData = {sensorID:sensorID.value, waarde:waarde.value, IP:"80.200.123.2",datum:new Date()}

//console.log(sensData.datum)

sendData(sensData)

}

sensorForm.addEventListener("submit",submitManualData)

de eerste drie lijnen worden de variabelen aangemaakt. De functie eronder is wat de data naar de database zal sturen via een POST. De functie daar onder zal runnen wanneer de form gesubmit wordt. De preventDefault zorgt ervoor dat de pagina niet herladen wordt. Via de variabelen die boven gedeclareerd zijn kunnen de waardes gewoon opgehaald worden door .value te doen. De waardes worden dan in een object gestoken en doorgegeven aan de sendData functie.

De async function sendData is geschreven met await en dus is er geen callback nodig. Hier is wel geen error handling dus de user weet niet als de request succesvol was of niet.

### JS voor de tabel

1. Gebruik querySelector om de element op te halen
2. Gebruik window.onload om en stel deze gelijk aan een functie bv initFunc()
3. In de initFunctie wordt de tabelFunctie opgeroepen.
4. In de tabel functie maak je een Get request voor alle data en steek je deze waardes in een HTML tabel

window.onload = initFunc()

function filterTable(e){

const table = document.getElementById("tableBody")

getAllData((data) => {

if (data.length == 0) {

alert("er is een error met de tabel in te laden")

}

table.innerHTML = ''

data.forEach( (item) => {

let row = table.insertRow()

let sensorID = row.insertCell(0)

sensorID.innerHTML = item.sensorID

let waarde = row.insertCell(1)

waarde.innerHTML = item.waarde

let IPAdress = row.insertCell(2)

IPAdress.innerHTML = item.IP

let datum = row.insertCell(3)

datum.innerHTML = item.datum

})

})

}

function getAllData(callback) {

fetch("/allSensorData").then((response) => {

response.json().then((data) => {

for (var i = 0; i < data.length; i++) {

data[i].datum = new Date(data[i].datum).toLocaleString("be-BE")

}

callback(data)

})

})

}

function initFunc(){

filterTable()

standardChart()

}

De window.onload object zegt dat wanneer de DOM geladen is run dan deze functie. Deze functie zal dan de grafiek en de tabel genereren voor alle data uit de database. De standardChart functie wordt in de volgende punt gemaakt. Deze kan nu even gecomment worden.

Weeral voor de request wordt een asynchrone fetch() gebruikt, maar hier wordt het via een callback gedaan. Wanneer de data opgeroepen is wordt het in JSON formaat gezet en wordt de datum naar de correcte tijdszone gezet.

In de callback wordt er via een for loop de waardes uitgelezen en in de tabel gestoken.

### JS voor de grafiek

1. Maak een functie aan genaamd standardChart.
2. In standardChart vraag je de data op via de functie die je ervoor gemaakt hebt. Je hebt enkel de datums en de waardes nodig.
3. In de laatste lijn van standard chart roep je een nieuwe functie op bv. drawChart
4. In drawChart wordt de grafiek daadwerkelijk getekend
5. Sinds de chart.js script ingeroepen is kun je een nieuwe chart object aanmaken en deze instellen en vervolgens linken aan de HTML-element
6. Tot slot voeg je in de initFunc deze lijn toe: setInterval(<funcNaam>,180000). Setinterval zorgt ervoor dat een functie om de zoveel tijd runt in deze voorbeeld om de 180000msec = 3 min.

Code loopt verder op volgende pagina

window.onload = initFunc()

function drawChart(dates,tempData,humData){

var ctx = document.getElementById("myChart").getContext('2d')

const myChart = new Chart(ctx, {

type: "line",

data: {

labels: dates,

datasets: [{

label: 'Temperatuur',

data: tempData,

backgroundColor: ['rgba(222, 7, 157, 0.2)'],

borderColor: ['rgba(148, 0, 103,1)'],

borderWidth: 1

},

{

label: 'Vochtigheid',

data: humData,

backgroundColor: ['rgba(7, 217, 24, 0.2)'],

borderColor: ['rgba(4, 148, 16,1)'],

borderWidth: 1

}]

},

options: {

scales: {

yAxes: [{

ticks: {

beginAtZero:true

}

}]

},

legend:{

position:'right',

labels:{

fontColor:"#000000",

fontSize:15

}

},

layout:{

padding:25

}

}

});

}

function standardChart(){

let tempData = []

let humData = []

let datum = []

let a = 0, b = 0

getAllData((data) => {

data.forEach((item) => {

if(item.sensorID == 1){

tempData[a] = item.waarde

datum[a] = item.datum

a++

}

else{

humData[b] = item.waarde

datum[b] = item.datum

b++

}

})

drawChart(datum,tempData,humData)

})

}

function initFunc(){

filterTable()

standardChart()

setInterval(standardChart,180000)

}

## alles samenvoegen

wanneer je bovenstaande codes samenvoegt krijg je normaal dit:

(loopt door)

const sensorForm = document.querySelector("#manualData")

const sensorID = document.querySelector("#sensID")

const waarde = document.querySelector("#sensValue")

window.onload = initFunc()

function filterTable(e){

const table = document.getElementById("tableBody")

getAllData((data) => {

if (data.length == 0) {

alert("er is een error met de tabel in te laden")

}

table.innerHTML = ''

data.forEach( (item) => {

let row = table.insertRow()

let sensorID = row.insertCell(0)

sensorID.innerHTML = item.sensorID

let waarde = row.insertCell(1)

waarde.innerHTML = item.waarde

let IPAdress = row.insertCell(2)

IPAdress.innerHTML = item.IP

let datum = row.insertCell(3)

datum.innerHTML = item.datum

})

})

}

async function sendData(data){

const rawResponse = await fetch("/addData",{

method: "POST",

headers:{

"Accept" : "application/json",

"Content-Type": "application/json"

},

body: JSON.stringify(data)

})

const content = await rawResponse.json()

//console.log(content)

}

function getAllData(callback) {

fetch("/allSensorData").then((response) => {

response.json().then((data) => {

for (var i = 0; i < data.length; i++) {

data[i].datum = new Date(data[i].datum).toLocaleString("be-BE")

}

callback(data)

})

})

}

function submitManualData(e){

e.preventDefault()

const sensData = {sensorID:sensorID.value, waarde:waarde.value, IP:"80.200.123.2",datum:new Date()}

//console.log(sensData.datum)

sendData(sensData)

}

function drawChart(dates,tempData,humData){

var ctx = document.getElementById("myChart").getContext('2d')

const myChart = new Chart(ctx, {

type: "line",

data: {

labels: dates,

datasets: [{

label: 'Temperatuur',

data: tempData,

backgroundColor: ['rgba(222, 7, 157, 0.2)'],

borderColor: ['rgba(148, 0, 103,1)'],

borderWidth: 1

},

{

label: 'Vochtigheid',

data: humData,

backgroundColor: ['rgba(7, 217, 24, 0.2)'],

borderColor: ['rgba(4, 148, 16,1)'],

borderWidth: 1

}]

},

options: {

scales: {

yAxes: [{

ticks: {

beginAtZero:true

}

}]

},

legend:{

position:'right',

labels:{

fontColor:"#000000",

fontSize:15

}

},

layout:{

padding:25

}

}

});

}

function standardChart(){

let tempData = []

let humData = []

let datum = []

let a = 0, b = 0

getAllData((data) => {

data.forEach((item) => {

if(item.sensorID == 1){

tempData[a] = item.waarde

datum[a] = item.datum

a++

}

else{

humData[b] = item.waarde

datum[b] = item.datum

b++

}

})

drawChart(datum,tempData,humData)

})

}

function initFunc(){

filterTable()

standardChart()

setInterval(standardChart,180000)

}

sensorForm.addEventListener("submit",submitManualData)

dit is de volledige javascript voor de site.

# Enviroment variables

Enviroment variables zijn variabelen die je een specifieke waarde kan meegeven voordat alles gestart wordt. Bv. De database URL is op dit moment local maar als je de site online wilt hosten zul je een online externe database hebben met een eigen link. En i.p.v. de hele tijd de link te veranderen kun je gewoon de enviroment variable declareren. En zeggen van gebruik deze enviroment variable

## Enviroment variables aanmaken

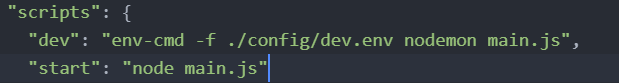
1. Maak een nieuwe map aan genaamd config
2. In deze folder voeg je een .env file aan bv. dev.env
3. Deze file werkt met key-value pairs waar de key’s in hoofdletters geschreven zijn
4. Voeg deze lijn toe MONGODB\_URL=<url naar je mongoDB database>
5. In mongoose.js in de db folder verander je de call naar connect. Dus bv. i.p.v. mongoose.connect("mongodb://127.0.0.1:27017/IOT") 🡪 mongoose.connect(process.env.MONGODB\_URL)

Vanaf nu zal npm run dev failen want hij kent geen environment variables deze zullen we moeten meegeven. In het begin heb je een module genaam env-cmd geinstalleerd deze zal nu gebruikt worden

1. Open package.json
2. Verander je dev script van “nodemon main.js” naar “env-cmd -f ./config/deploy.env nodemon main.js”
3. Sluit nodemon af en herstart deze
4. Normaal zal vanaf nu de enviroment variable gebruikt worden

Afbeelding met tekst

Automatisch gegenereerde beschrijving



# je project deployen

wanneer alles localy werkt kun je het pushen naar Heroku. Heroku is een service waarmee je je website kunt hosten. Zonder Heroku bestaat je website enkel local, en moet je constant de server laten runnen alsook je database. En als iemand op je website wilt geraken zul je een soort tunneling moeten toestaan om dit mogelijk te maken. Je hebt ook een online database nodig. Degene die nu gebruikt wordt werkt enke local en is bedoeld voor development en niet voor deployment

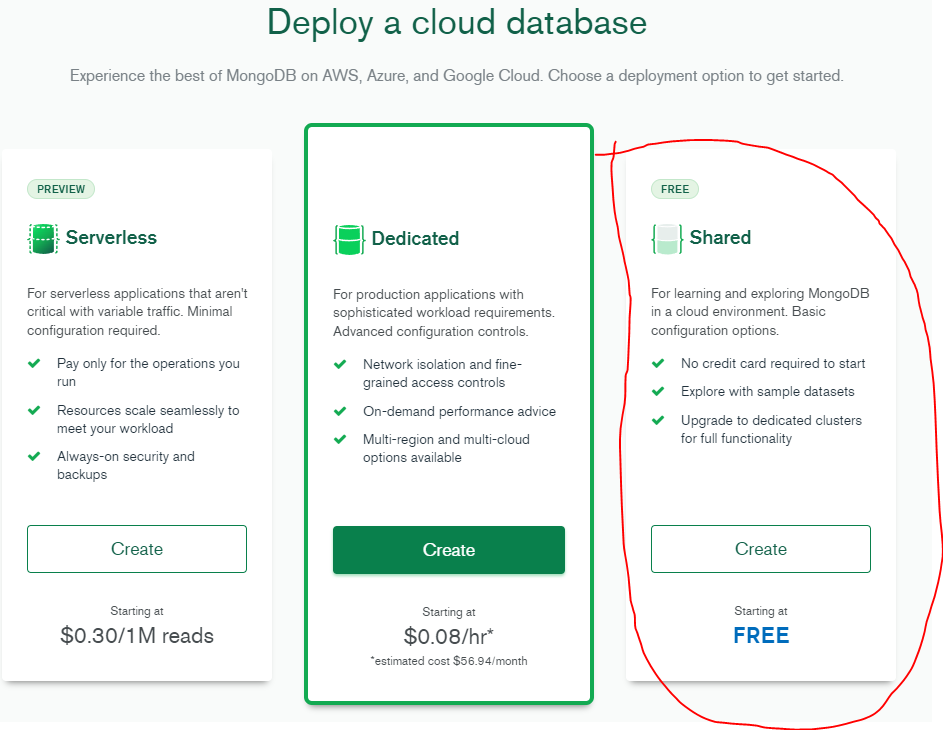
## online MongoDB database aanmaken

1. navigeer naar deze link en log je in <https://www.mongodb.com/free-cloud-database>
2. hierna krijg je wat gezever wat niet belangrijk is en je mag invullen wat je wilt

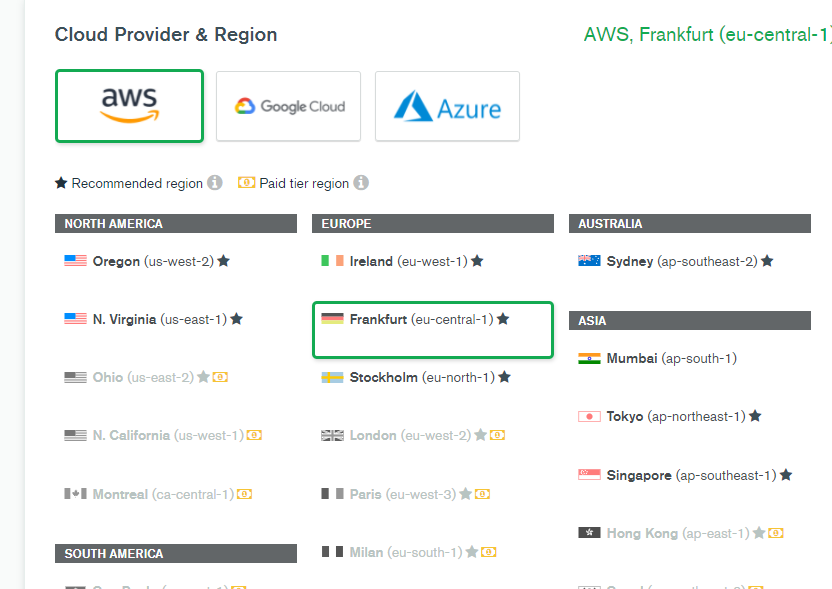
Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. hierna kies je shared want dat is gratis en normaal gezien heb je niet veel data nodig



1. je kiest je provider en je regio. Normaal kun je nu enkel kiezen tussen de gratis opties. Zo niet dan zie je normaal kleine blauwe tags staan met erbij FREE. Eronder staat cluster tier, deze laat je ook op FREE staan



1. hierna klik je onder op create cluster
2. nu klik je links op databases (op dit moment wordt je database aangemaakt het kan even duren)

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. nadat je cluster aangemaakt is kun je op connect klikken (de connect knop wat langs Cluster0 staat)
2. dan klik je op add a Different IP Address
3. Afbeelding met tekst

   Automatisch gegenereerde beschrijvingtype als ip 0.0.0.0/0 en dan klik je op Add IP
4. Crieeer nu een username en een password. En klik op create database user

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Klik vervolgens rechtsonder op Choose a connection method
2. Klik dan op connect using mongoDB compass
3. Bij step2 kopieer je de string. !!!!IN DEZE STRING ZIT JE WACHTWOORD DUS VERSPREID DEZE NIET ZOMAAR
4. Open nu robo3T en klik op create

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Onderaan zie je een knop from SRV hierlangs is een textbalk en daar paste je string in
2. Afbeelding met tekst

   Automatisch gegenereerde beschrijvingGEEN ENTER DUWEN, KLIK OP From SRV
3. Normaal zouden je iets over clusters moeten zien

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Nu kan je op save klikken en connecten met deze database

## Commiten naar github

Voordat je commit maak eerst een .gitignore file aan. En voeg deze twee regels toe: node\_modules en config. Dit moet gewoon onder elkaar staan. Dit zorgt ervoor dat github deze mappen ignored want node\_modules zijn te veel bestanden die nutteloos zijn, en config is een private map. Als iemand anders dit wil gebruiken zal die zijn eigen config file moeten aanmaken.

## Heroku installeren

Gebruik deze link om heroku te installeren: <https://devcenter.heroku.com/articles/heroku-cli>

## Een heroku app maken

1. Zorg ervoor dat je alles gecommit hebt naar je github repository
2. Navigeer in je command line naar de map die je gebruikt
3. Voer volgende commando’s uit
4. Heroku create <unieke naam voor je app>
5. Heroku config:set MONGODB\_URL=<SRV string van database>/IOT . op het einde van de srv staat /test vervang dit door /<NAAM VAN JE DB gebruikt bij local>.
6. Git push heroku master

Normaal zal heroku nu beginnen met de site op te bouwen en op het einde staat er een link naar je site