

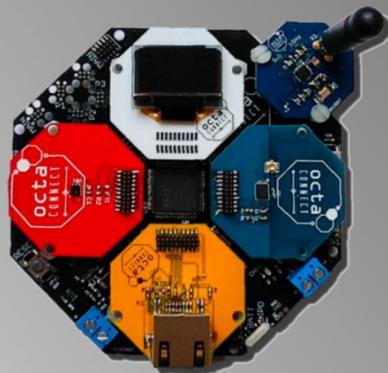
Scriptie ingediend tot het behalen van de graad van  
PROFESSIONELE BACHELOR IN DE ELEKTRONICA-ICT

# Controle-applicatie voor het OCTA-Connect open-source IoT-Platform

Kevin Van de Mieroop

academiejaar 2015-2016

AP Hogeschool Antwerpen  
Wetenschap & Techniek  
Elektronica-ICT



# Table of Contents

Abstract	1
Dankwoord	2
Introductie	3
Universiteit Antwerpen	3.1
Situering: OCTA-Connect	3.2
OCTA-Connect Gateway	3.2.1
Controle-Applicatie	3.2.2
Technisch	4
Analyse	4.1
Oorspronkelijke architectuur	4.1.1
Design	4.2
Client	4.2.1
MQTT Broker	4.2.2
Server	4.2.3
Ontwikkeling	4.3
Communicatie	4.3.1
Software	4.3.2
Hardware	4.3.3
Gebruikte tools	4.4
Conclusie en samenvatting	5
Conclusie & resultaat	5.1
Desktopapplicatie	5.1.1
Web-based applicatie	5.1.2
Verbeteringen	5.1.3
Samenvatting	5.2
Bibliografie	6
Glossary	7

# Abstract

De opdracht die tijdens de stage bij de Universiteit Antwerpen gerealiseerd moest worden was een full-stack applicatie (Node.js, ExpressJS, AngularJS en MongoDB) die de metingen van een development board en bijhorende sensoren verwerkt en visualiseert in de vorm van grafieken. OCTA-Connect is een platform ontwikkeld door een onderzoeksgrond van de Universiteit Antwerpen. Het draait rond een development board dat gebouwd is voor allerlei Internet of Things projecten waar veel data bij gemoeid is. De data wordt doorgestuurd vanaf sensoren naar een OCTA-Connect Gateway. Deze applicatie verwerkt de data en slaat ze op in een database om later gevisualiseerd te worden.

De OCTA-Connect Gateway wordt aangesloten op een gateway (PC) via UART. Deze PC stuurt de data live door naar de backend via een MQTT broker, waar het verwerkt wordt en *raw* opgeslagen wordt in een database. De data kan dan worden opgehaald nadat de gebruiker is ingelogd. Per account kan een OCTA ID worden toegevoegd zodat enkel de eigen data binnenkomt. De *raw* data wordt dan geparsed in de frontend om zo de laatste entry live te tonen. De historic data kan echter ook worden opgehaald. Deze historic data wordt getoond aan de hand van verschillende grafieken en figuren, om de evolutie duidelijk zichtbaar te maken.

De bedoeling is om een zo breed mogelijk publiek te hebben die gebruik kan maken van het platform. Zo kan bijvoorbeeld een serre gemonitord worden (luchtkwaliteit en zo verder).

Dit zijn de *core features* die zeker geïmplementeerd moesten worden. Eventuele bonusen of uitbreidingen zijn bijvoorbeeld een dynamieke parser in de front-end die ingesteld kan worden per user. Zo kan een pakketje individueel geparsed worden om nog meer persoonlijke data binnen te krijgen.

# Dankwoord

Aanvankelijk wist ik niet wat te verwachten: 12 weken stage op een plek waar ik één keer ben geweest, en een project uitvoeren met mensen die ik één keer heb ontmoet op een kort kennismakingsgesprek. De opdracht was toen nog vaag, ik wist enkel dat het rond een eigen development board zou draaien. Of ik met embedded systemen moest werken of high-level moest programmeren was nog niet duidelijk.

Dit veranderde toen ik de eerste dag op m'n stageplek zat. Glenn Ergeerts en Dragan Subotic hadden me goede leerstof gegeven om me op gang te helpen, ik wist al meteen wat te doen. De dag erop werd ik naar m'n definitieve plek gewezen. Deze plaats kende ik al wel (op de oude AP-campus, in het lokaal waar ik in het prille begin nog Netwerken volgde van Mr. Masset), en het voelde alsof ik m'n collega's al even lang kende.

De 3 studenten waar ik de komende 12 weken (vrijwel) dagelijks mee aan de bureau zou zitten waren Jens Vanhooydonck, Didier Verachtert en Thomas Van Loon, die alledrie een master Industrieel ingenieur elektronica-ict hadden en hun postgraduaat zouden behalen. Van een gespannen werksfeer waar ik voor vreesde was al meteen geen sprake, aangezien ze zelf nog niet al te lang waren afgestudeerd. De eerste meeting verliep vlot en ik had meteen een duidelijke opdracht; zelfs een waar ik zonder probleem thuis ook aan verder kon werken.

De amicale werksfeer is iets waarvan ik denk dat ik het nergens anders had kunnen vinden, maar ik had vooral niet m'n project kunnen maken zonder de talloze hulp die ze me verleend hebben doorheen de weken. Vooral Jens heeft ontzettend veel gedaan om me op het juiste spoor te zetten en me steeds te blijven opvolgen.

Ook Glenn Ergeerts en Dragan Subotic wil ik hiervoor bedanken. Wanneer Jens het niet wist (zelden) kon ik meteen bij hen terecht voor de nodige uitleg.

Het proeflezen heb ik aan mijn vader, Karel Van de Mieroop, overgelaten. Zijn ervaring in IT heeft ertoe geholpen om niet alleen deze paper een duidelijke structuur te geven, maar hij heeft ook de nood voor een spellchecker geëlimineerd.

Ten slotte wil ik ook graag de docenten van AP (zeker Tim Dams en Yves Masset voor de begeleiding de afgelopen jaren) en Maarten Luyts bedanken. Hij heeft me als stagebegeleider geholpen de nodige formaliteiten correct in te leveren, met nuttige feedback die ik nergens anders had kunnen krijgen.

*Kevin Van de Mieroop*

*10 juni 2016*

*Antwerpen*

# 3 Introductie

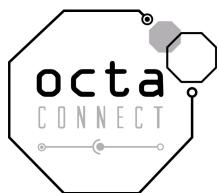
Deze bachelorproef draait vooral rond 1 onderwerp: het Internet of Things. IoT is een concept dat onzettend snel is gegroeid: het is onmisbaar omdat het voor ontelbare problemen een oplossing biedt. Het is vooral ook een breed onderwerp: het verbinden en met elkaar laten communiceren van alle toestellen. De hardware is hiervoor belangrijk, maar even belangrijk is ook de (embedded) software. De onderzoeksgroep elektronica-ict van de Universiteit Antwerpen spitst zich hier dan ook op toe. Ze behandelen en onderzoeken verschillende protocols om modules data aan elkaar te laten doorgeven. Een platform dat door hen zelf werd ontwikkeld is OCTA-Connect.

## 3.1 Universiteit Antwerpen

Breed gesproken: de UAntwerpen is een van de 2 grootste universiteiten van België en heeft een zeer breed aanbod van diploma's. Ze heeft echter ook een hele hoop onderzoeksgroepen die experimenteren in verschillende wetenschappen. Wat meer over de onderzoeksgroep elektronica-ict van de UAntwerpen. Ze spitsen zich dus toe op embedded systemen ontwikkelen. Het team, geleid door professor Maarten Weyn van de UA, bestaat uit programmeurs die elk een eigen specialiteit hebben. Een belangrijk project is het City of Things: sensoren in de stad plaatsen om zo verschillende waarden constant op te meten en algemene data te verzamelen op verschillende plekken in de stad. Zo plaatsen ze sensoren in de Velo-pilaren doorheen Antwerpen om bijvoorbeeld luchtkwaliteit op te nemen en deze via het LoRa-netwerk door te sturen. Het biedt verschillende mogelijkheden:

- Nieuwe netwerktechnologieën testen op een groot gebied.
- Ontzettend veel data verkrijgen die zorgt voor nieuwe informatie.
- Antwerpenaren actief laten meedoen met field tests, en zo automatisch een groot genoeg publiek creëren die betrouwbaardere data geeft.

## 3.2 OCTA-Connect



Bringing IoT to the industry

OCTA-Connect is een Internet of Things platform dat bestaat uit een OCTA-Connect gateway, een OCTA mini en diverse sensoren om data te verzamelen. Verschillende nodes (OCTA-mini) verbinden met de OCTA-gateway en geven elk data door. Aan de nodes kunnen willekeurige sensoren worden gekoppeld en communiceren hiervan de data (b.v. temperatuur) via onder andere DASH7 door aan

de gateway. Deze gateway is met een UART verbonden met de computer van de gebruiker. Hier past dit project in: de data moet meteen worden doorgegeven aan een backend die de data behandelt en ze in een database steekt. De data kan daarna via een dashboard worden opgehaald in de vorm van grafieken en historic data. Deze data wordt ook geparsed om verstaanbaar gevisualiseerd te worden.

Een voorbeeld van een project dat uitgevoerd kan worden met het OCTA-Connect platform in combinatie met het DASH7 Alliance protocol is het bird tracking project (*DASH7 Alliance Protocol in Monitoring Applications*). Hier werd een endnode ontwikkeld, zo klein dat het kon gevestigd worden op een klein vogeltje. Het vogeltje werd getracked vanaf geboorte om te zien waar het een nest zou bouwen. De endnode stuurde de locatie van de vogel met een tijdsinterval door naar een sub-controller die dan op zich de data naar een gateway (300-400 meter verder) verzond.

### 3.2.1 OCTA gateway

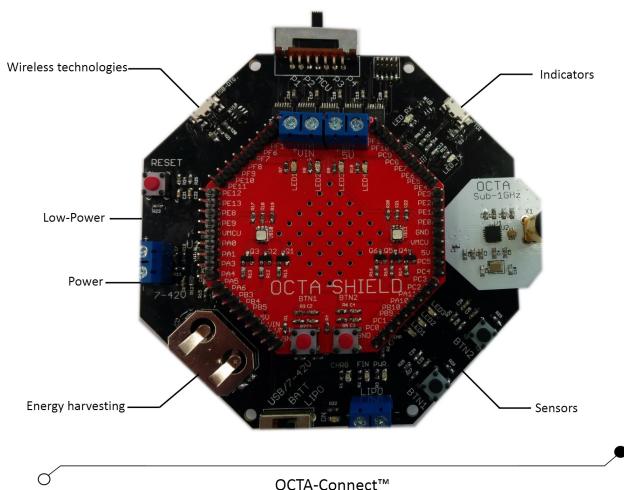


Fig. 1: OCTA-Connect Gateway

De "core" van het idee van OCTA-Connect is *modulariteit*. Dit wil zeggen dat alle modules "stackable" zijn; op de gateway kunnen voorgemaakte modules gekoppeld worden door ze gewoonweg op elkaar te stapelen. Zo is er een LoRa-module, een DASH7-module, een Sigfox-module, en zo verder. Deze modules behandelen dus elk een eigen protocol om zo data van verschillende bronnen binnen te kunnen krijgen.

De gateway zelf kan geprogrammeerd worden aan de hand van een eigen OCTA IDE, die te vinden is op de *Downloads*-pagina van OCTA-Connect (<http://octa-connect.com/download.php>).

### 3.2.2 Controle-applicatie

Dit project draait rond een monitoring applicatie die gebruikt kan worden bij een OCTA-Connect Gateway. De data wordt gegenereerd door sensoren die op een OCTA-mini zijn aangesloten. Deze OCTA-mini stuurt de data door naar de OCTA-Connect Gateway die op zijn beurt aangesloten is op een PC. De gateway verstuurde data over MQTT naar de backend, die de data in een database zet. De gebruiker kan zijn data, in de vorm van grafieken en tabellen, oproepen door in te loggen op het Angular-based platform.

De werking van de applicatie en het ontwikkelproces ervan zal in detail worden besproken in dit document. Tijdens de ontwikkeling werd een OCTA-Connect gateway gesimuleerd aan de hand van een Giant Gecko development board gepaard met een DASH7 module. De module is exact dezelfde die op een OCTA-Connect gateway zal zitten dus de werking en de verwerking van de data zal op dezelfde manier gebeuren. De data die tijdens dit project binnenkomt is afkomstig van een OCTA-mini die in een waterproof casing zit samen met een GPS-module, een lichtsensor, een CO2-sensor, een temperatuursensor, een VOC-sensor en een RH-sensor. Deze doos is onderdeel van een afzonderlijk project in samenwerking met bpost: verschillende dozen worden gevestigd op verschillende postwagens om zo data op te nemen en door te sturen terwijl de wagen rondrijdt. Het biedt een goed voorbeeld van een toepassing voor het platform en daarbij ook data die in deze applicatie gebruikt kan worden; alle data in dit document zijn dus hieruit afkomstig.

# 4. Technisch

## 4.1 Analyse

Op de eerste dag van de stage werd samengezeten om een gepaste opdracht uit te stippen aangezien deze nog niet was bepaald. Het enige dat zeker was, was dat de applicatie rond het OCTA Platform zou draaien. Hiervoor bestonden uiteindelijk nog niet veel applicaties, wat wel nodig is voor een product dat commercieel gelanceerd zou worden. Er werd uiteindelijk besloten om een algemene full-stack applicatie te ontwikkelen die de binnenkomende data van elke OCTA-Gateway kon verwerken en visualiseren; een applicatie die door iedereen gebruikt kon worden die het toestel zou aanschaffen.

### 4.1.1 Oorspronkelijke architectuur

Er werd een architectuur geschetst, die open stond voor verandering:

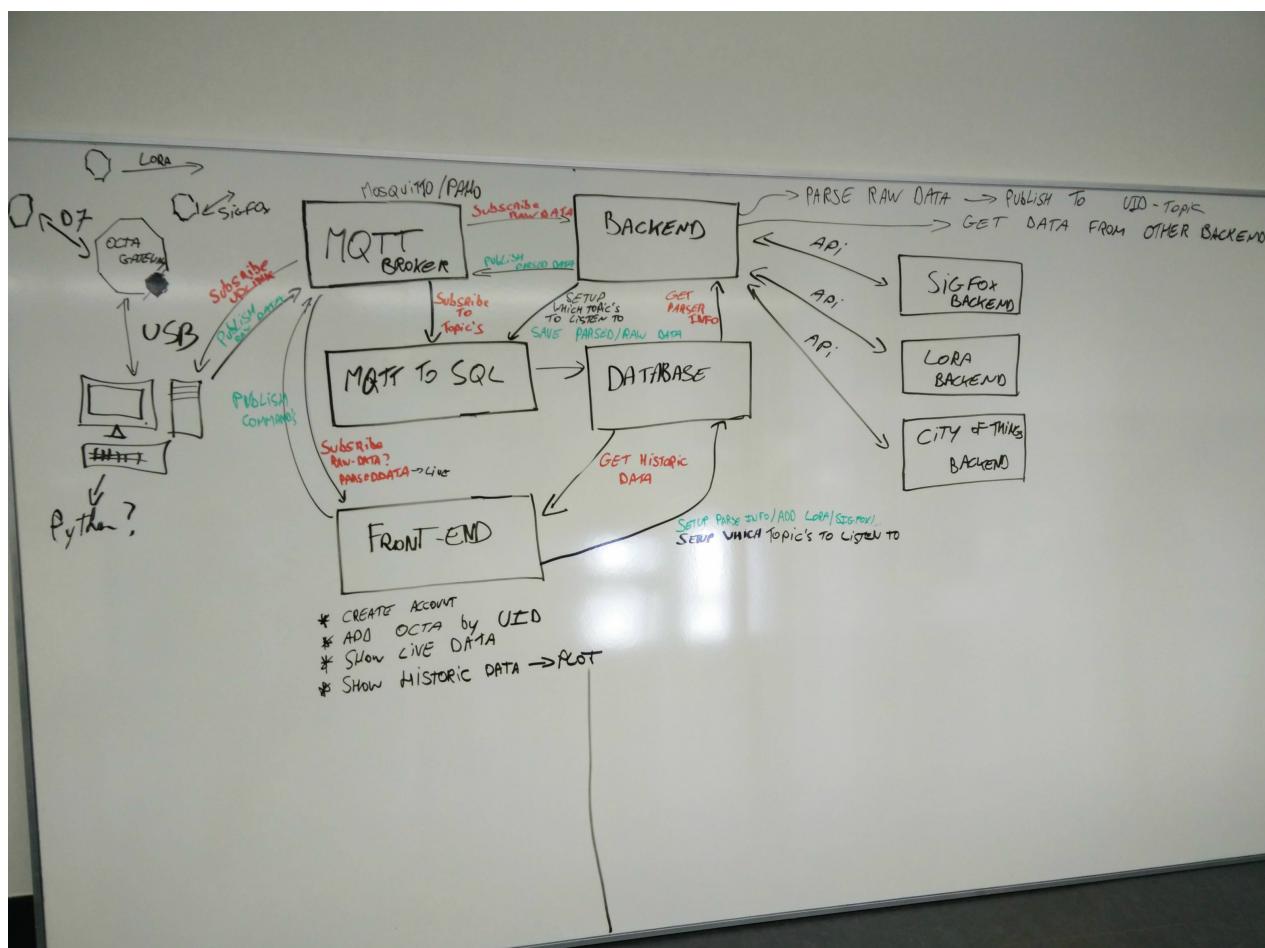


Fig. 2: Oorspronkelijke architectuur

Hierin werd voorgesteld om de gateway-software in Python te schrijven. Deze gateway-software stuurde de data door naar de MQTT-broker. De MQTT-broker kon een third party service zijn zoals

HiveMQ, of een broker die werd gehost op eigen servers, zoals de open-source Mosquitto. De MQTT werd omgezet naar SQL queries om in een database te zetten. Deze SQL werd dan weer opgehaald door de backend, en de front-end zou op zijn beurt alle data weergeven. In de front-end zouden de belangrijkste functies zijn:

- Een account aanmaken
- Een OCTA Connect-gateway toevoegen per UID (OCTA ID)
- Live data laten zien die automatisch dus werd geüpdateerd
- Historic data tonen aan de hand van charts en figuren
- Toekomstig werk zou dan kunnen zijn: support toevoegen voor het SigFox-netwerk en het LoRa-netwerk (ook het gehele City of Things-netwerk zou bereikt kunnen worden)

Na wat opzoek- en analyseerwerk werd het duidelijk dat voor sommige onderdelen er betere alternatieven waren. Een non-relationele database zou ideaal zijn aangezien elke datapoint dan als JSON-object opgeslagen kon worden. Een aparte collectie in de MongoDB database zou authenticatie behandelen en een userbase bevatten. De backend is uiteindelijk in Node.js geschreven, en om homogeniteit in het platform te behouden werd ook de gateway-software in JavaScript geschreven, gebruik makend van technologieën die desktop-applicaties kunnen creëren met het Node.js framework (Electron e.d.).

Op de uiteindelijke architectuur van het platform zal zo dadelijk dieper worden ingegaan, in het hoofdstuk *Design*.

Het onderdeel dat niet door mij behandeld werd is de hardware. Om een OCTA Connect-gateway te simuleren werd een Gecko development board gebruikt. Deze was echter nog niet klaar dus om dummy data te genereren heb ik een opstelling met een Arduino Uno en een lichtsensor gebruikt, die de data ook serieel doorgeeft na een bepaald tijdsinterval.

## 4.2 Design

Hier een schema van de uiteindelijke architectuur:

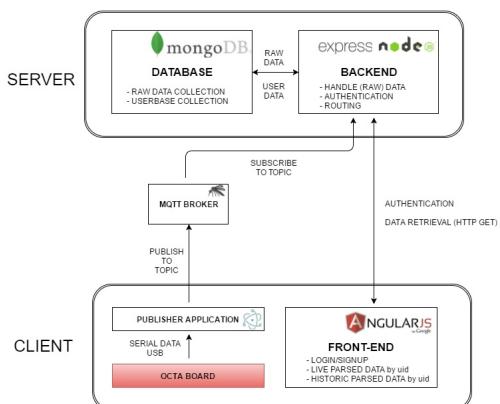
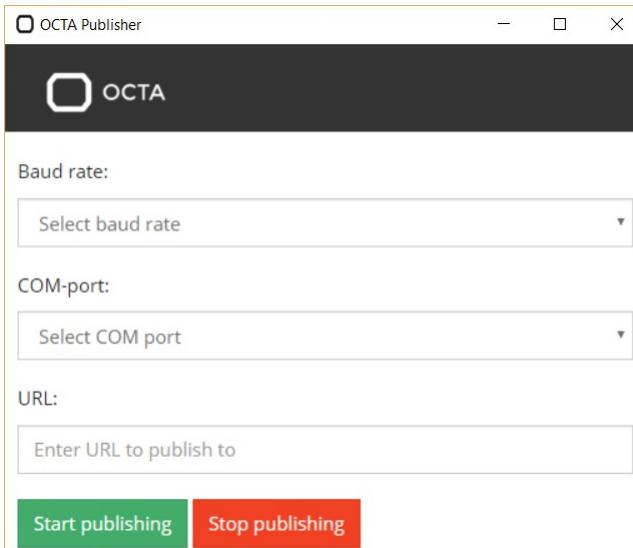


Fig. 3: Architectuur. Rood aangeduid: wat in dit document onbesproken blijft.

### 4.2.1 Client

#### 4.2.1.1 Desktop applicatie



\*Fig. 4: Beta-versie van de publishing desktop applicatie

De client side bestaat er een desktop applicatie. Deze cross-platform applicatie behandelt de data die binnenkomt via de OCTA-Connect gateway. Echte OCTA-Gateways zijn nog niet beschikbaar dus werden ze voor dit project gesimuleerd met een Gecko development board. Deze gateway krijgt de data binnen (voornamelijk via het DASH7 Alliance Protocol, door middel van een DASH7 module ontwikkeld door de Universiteit Antwerpen zelf) en stuurt ze onmiddellijk via de UART door in seriële vorm naar de PC van de gebruiker. Op de desktop-applicatie kan gekozen worden door welke COM-poort de gegevens binnenkomen. Ze worden dan gepubliceerd naar een topic van de MQTT broker die gehost wordt op eigen servers.

De applicatie zelf is geschreven in JavaScript, gebruik makend van het Electron framework. Dit wil zeggen dat het in essentie een Node.js applicatie is die lokaal draait. Het grote voordeel hiervan is dat Node.js modules gebruikt kunnen worden voor allerlei taken en er dus overbodige code weggelegd kan worden. Om de publisher te maken moest gebruik kunnen worden gemaakt van de USB-poort en de data die hier binnenviel; hiervoor bestond de *serialport* module. Ook moest er een module gevonden worden voor MQTT. Hiervoor bestond de vanzelfsprekende *mqtt* module.

Op de modules, en waarom eventuele alternatieven niet gebruikt werden, zal dieper worden ingegaan tijdens het ontwikkelproces.

#### 4.2.1.2 Web-based applicatie

Nadat de gegevens verstuurd zijn naar de backend kan door de gebruiker ook worden ingelogd op het platform dat web-based is. Voor het dashboard werd bootstrap gebruikt; specifieker een bootstrap template genaamd *Admin-LTE*. In deze template werd AngularJS gegoten om het dashboard dynamisch te maken. Standaard had de template allerlei extra's, zoals notificaties, die niet noodzakelijk zijn en dus verwijderd werden. De vensters en knoppen zitten nog wel in de stylesheet, dus kunnen eventueel later nog geïmplementeerd worden.

Voor de authenticatie was enkel een form nodig die de nodige gegevens naar de backend stuurde. Wanneer ingelogd komt men op het dashboard terecht, waar een overzichtje wacht met alle recentste datapunten. Hieruit wordt ook eerst de OCTA ID gehaald om zeker te zijn dat geen irrelevantie data/data van iemand anders wordt getoond. De datapunten worden opgehaald door een http get in de controller van de applicatie. Links is een sidebar met links naar de charts, lists, en account info. In de charts wordt historische data gevisualiseerd aan de hand van grafieken. Hiervoor werd ChartJS gebruikt. In de lists wordt een tabel getoond met alle historische datapunten en timestamps. Bij account info kan je uitloggen of je account verwijderen. Future work hier is extra OCTA-gateways per UID toevoegen, en de parser instellen om persoonlijker data te verkrijgen.

## 4.2.2 MQTT Broker

De MQTT broker wordt momenteel gehost op een server buiten de UAntwerpen; op naam van Jens Vanhooydonck en op basis van Mosquitto. Deze kan gemakkelijk vervangen worden door een alternatieve server of broker zonder dat de nieuwe server hiervoor geconfigureerd hoeft te worden. Enkel de verwijzingen in de applicatie zelf moeten aangepast worden.

## 4.2.3 Server

De backend, die op Node.js draait, wordt eveneens gehost op eigen servers. Hier draaien verschillende modules op die elk een eigen taak heeft. Op de server wordt gesubscribed op hetzelfde topic van de broker, waardoor de data onmiddellijk binnenkomt in de backend. Ze worden volgens een model rechtstreeks rauw opgeslagen in de database, met samen in het object de timestamp waarop het binnenkomt en de uid van de gateway. Op de backend wordt de authenticatie behandeld door PassportJS, een npm module, waarover later meer. Wanneer op het OCTA-Connect platform wordt ingelogd, geeft de backend door middel van Express routing de datapunten door aan de front-end.

## 4.3 Ontwikkelproces & gebruikte technologieen

In dit hoofdstuk zal in detail en in min of meer chronologische volgorde worden uitgelegd hoe de applicatie ontwikkeld werd. Bij elke stap zal meer (eventueel achtergrond-)informatie worden gegeven over de module en eventuele alternatieven. We gaan uit van de basis van het platform; zowel client-side als server-side zijn gebouwd met hetzelfde JavaScript framework.

### 4.3.1 Communicatie

Een van de belangrijkste onderdelen van elke Internet of Things applicatie is zonder twijfel **communicatie**. De manier waarop de verschillende onderdelen met elkaar praten is zeer belangrijk; aldus het protocol. Alle protocollen voor datacommunicatie passen binnen een referentiemodel genaamd het OSI (Open Systems Intercommunications) model. Deze bevat verschillende lagen, ofwel (fysieke en logische) stadia:

- Toepassingslaag (b.v.: Network File System (NFS))
- Presentatielaag (b.v.: HyperText Markup Language (HTML))
- Sessielaaag (b.v.: Telnet)
- Transportlaag (b.v.: MQTT)
- Netwerklaag (b.v. IPv4)
- Datalinklaag (b.v. IEEE 802.2, LoRa)
- Fysieke laag (b.v. USB, LoRa)

Van deze lagen zal dieper worden ingegaan op de vierde laag, waar de volgorde en segmentatie van de berichten worden bepaald. Hier zit het MQTT-protocol, wat voor de applicatie gebruikt zal worden om te communiceren tussen de publisher-applicatie en de backend.

#### 4.3.1.1 MQTT

MQTT (voormalig MQ Telemetry Transport) is het protocol dat het vaakst gebruikt wordt bij projecten omtrent Internet of Things. De werking ervan draait vooral rond de MQTT broker: deze krijgt berichtjes binnen (gewoonlijk zeer korte berichtjes van enkele bytes lang), en verdeelt ze dan weer onder diegenen die willen. Hier zijn benamingen voor, gelijkaardig met een forum: diegenen die berichtjes versturen zijn de publishers; zij publiceren berichten op een bepaald topic. Diegenen die de berichtjes willen lezen zijn de subscribers; zij abonneren op een bepaald topic en krijgen real time de berichtjes binnen als er nieuwe worden gepubliceerd op het gewenste topic. Zelfs veelgebruikte instant messaging of chat-applicaties zijn opgebouwd rond MQTT; zoals Messenger van Facebook.

MQTT biedt drie verschillende niveaus van Quality of Service aan. Dit wil zeggen wat de module kan aanbieden in mate van features. Hoe hoger de Quality of Service, hoe complexer het wordt en hoe meer het vergt van de server (hier de broker):

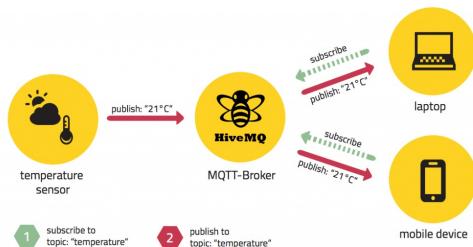
- QoS 0: "At most once"; het bericht wordt eenmalig verstuurd vanaf de publisher, de subscriber kan het eenmalig ontvangen

- QoS 1: "At least once"; het bericht wordt bijgehouden door de afzender totdat een acknowledgment is teruggekregen (PUBACK)
- QoS 2: "Exactly once"; het bericht wordt gepubliceerd: PUBREC(eived) flag, dan een PUBREL(eased) flag, en ten slotte een PUBCOMP(leted) flag

Om kennis te maken met het principe en de werking ervan zal in het onderdeel *Software* een korte demo worden uitgelegd die de basis vormt van de applicatie. Vooraleer aan de software begonnen kan worden, zal er eerst een broker vastgesteld moeten worden die gebruikelijk is voor de demo.

Hiervoor bestaan verschillende alternatieven:

### HiveMQ



*Fig. 5: Schema van een HiveMQ opstelling (Bron: [www.hivemq.com](http://www.hivemq.com))*

Een third-party service die gratis hun servers aanbiedt om gebruik te maken van brokers. Persoonlijk gebruik ervan is gratis; een licentie is echter vereist voor een commerciële applicatie.

### Mosquitto



Een open-source tool die in een console venster loopt op een eigen server. Zeer gemakkelijk en licht, dus de snelheid waarop data passeert hangt volledig van de server af.

In dit project zal steeds Mosquitto gebruikt worden. Aanvankelijk zal het lokaal gedraaid worden, maar daarna wordt een nieuwe broker geinitialiseerd op een externe server.

## 4.3.2 Software

### 4.3.2.1 Node.JS



De fundering van zowel de desktop-applicatie als de web-based applicatie en dus van dit gehele project is Node.JS. Voornamelijk bedoeld om server-side applicaties te ontwikkelen die web-based zijn, is Node.JS een enorm populair en veelzijdige runtime waarop ook desktop applicaties gebouwd kunnen worden. Applicaties gebouwd op Node.JS werken asynchroon, wat ervoor zorgt dat enorm veel threads of uitvoeringen tegelijk kunnen draaien (met 1 hoofd-thread die wordt geïnformeerd als er

een thread in de *threadpool* zijn taak heeft uitgevoerd). Node.JS is gebouwd op de V8 engine om JavaScript uit te voeren. Deze werd op zijn beurt gebouwd voor Google Chrome, en werd in 2008 open-source gemaakt door Google zelf. In plaats van het rechtstreeks uit te voeren zet de V8 engine de JavaScript code eerst om naar machine code.

In Node.JS zit een ingebouwde package manager, simpelweg genaamd Node Package Manager (**npm**). Hiermee kunnen alle, inclusief non-officiële, modules worden gedownload, geïnstalleerd, verwijderd en gemodificeerd. De modules dienen niet als code, maar worden gezien als *dependencies*, waarnaar verwezen wordt in de applicatie zelf zodat functies van die dependencies gebruikt kunnen worden (te vergelijken met libraries). Deze dependencies worden opgeslagen in de folder **node\_modules**, en opgenomen in het **package.json** bestand van de applicatie. Ook algemene taken die de applicatie moet uitvoeren zijn hierin terug te vinden. Bijvoorbeeld: normaliter moet elke Node.JS applicatie opgestart worden met

```
node app.js
```

waarin *node* het commando dat die uitgevoerd wordt (node moet het dus openen) en *app.js* de effectieve applicatie is die gestart moet worden. In het *package.json* bestand kan dan een variabele worden toegevoegd aan het *scripts* object: start betekent "node app.js" daarna moet enkel

```
npm start
```

worden ingegeven om de applicatie te doen lopen. Hier bestaat echter een beter alternatief voor, maar het geeft een beeld van het nut van het *package.json* bestand.

## Nodemon

Nodemon ([nodemon.io](http://nodemon.io)) is een module die veranderingen in een Node.JS applicatie automatisch opspoort. Wanneer veranderingen worden gevonden, wordt de applicatie opnieuw opgestart. Zo moet de Node.JS service niet telkens worden stopgezet a.d.h.v. *Control-C*. De module wordt globaal geïnstalleerd via npm, en de applicatie wordt nu op deze manier opgestart:

```
nodemon app.js
```

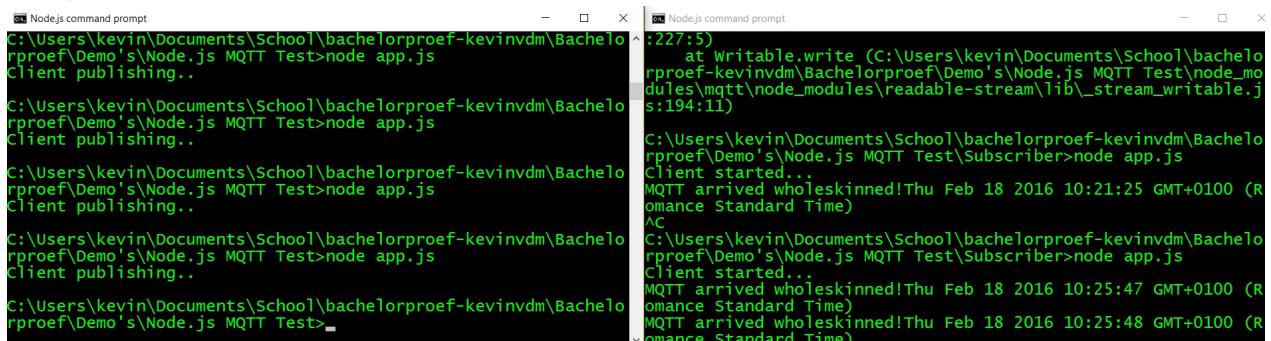
Er wordt gekeken naar alle javascript (.js) bestanden binnen de projectfolder en wanneer er een verandering opduikt wordt de gehele Node.JS service gestopt en opnieuw opgestart.

## Mosquitto demo

Deze demo is oorspronkelijk te vinden op <http://thejackalofjavascript.com/getting-started-mqtt/>. Node.js biedt enorm veel modules aan, eveneens voor het MQTT protocol, waarbij de host van de broker wordt opgegeven als parameter. In de modules zitten functies verwerkt die het publishen en subscriben van een topic behandelen.

Voorlopig wordt Mosquitto lokaal geïnstalleerd, zodat gerefereerd kan worden naar de localhost. Standaard draait Mosquitto op poort 1883. Indien publieke toegang nodig is, moet deze poort dus wel open gezet worden op het netwerk (hoe dit gebeurt hangt af van netwerk tot netwerk).

De demo maakte oorspronkelijk gebruik van 1 Node.js client. De publisher en subscriber zaten dus beiden in 1 app. De publisher subscribet op een topic "presence" en stuurt een berichtje erin naar de MQTT broker op localhost:1883. De subscriber subscribet op het topic "presence" en schrijft het ontvangen berichtje automatisch weg naar de console, om zo een succesvolle overdracht aan te tonen. Om de demo verder uit te werken, worden de publisher en subscriber beiden in een aparte app gezet.



The screenshot shows two separate Node.js command prompt windows. Both windows have the title 'Node.js command prompt' and are running on the same machine. The left window shows a publisher client publishing messages to a topic. The right window shows a subscriber client receiving these messages and logging them to the console. The publisher's log shows it publishing 'Client publishing..' messages at various intervals. The subscriber's log shows it receiving 'MQTT arrived wholeskinned!' messages at the same times, indicating successful delivery from the publisher.

```
C:\Users\kevin\Documents\School\bachelorproef-kevinvdm\Bachelorproef\Demo's\Node.js MQTT Test>node app.js
Client publishing..
```

```
:227:5)          at Writable.write (C:\Users\kevin\Documents\School\bachelorproef-kevinvdm\Bachelorproef\Demo's\Node.js MQTT Test\node_modules\mqtt\node_modules\readable-stream\lib\_stream_writable.js:194:11)
C:\Users\kevin\Documents\School\bachelorproef-kevinvdm\Bachelorproef\Demo's\Node.js MQTT Test>Subscriber>node app.js
Client started...
MQTT arrived wholeskinned!Thu Feb 18 2016 10:21:25 GMT+0100 (Romance Standard Time)
^C
C:\Users\kevin\Documents\School\bachelorproef-kevinvdm\Bachelorproef\Demo's\Node.js MQTT Test>Subscriber>node app.js
Client started...
MQTT arrived wholeskinned!Thu Feb 18 2016 10:25:47 GMT+0100 (Romance Standard Time)
MQTT arrived wholeskinned!Thu Feb 18 2016 10:25:48 GMT+0100 (Romance Standard Time)
```

\*Fig. 6: MQTT bericht over 2 Node.JS clients.

Dit is de allerlaagste basis van de uiteindelijke applicatie en geeft de werkelijke communicatie (data-overdracht) tussen client en server weer.

#### 4.3.2.2 Arduino programmatie

Oorspronkelijk werd een Arduino Uno gebruikt om data door te geven. Dit gebeurde via seriële communicatie in decimale vorm. De opstelling wordt uitgelegd in het gedeelte *Hardware*. Na de Arduino met USB aan te sluiten en een serial monitor op te starten, werd het volgende vastgesteld: Weinig licht is minder dan 600; middelmatig licht is tussen 600 en 800 en veel licht resulteert in een waarde hoger dan 800. Hierbij werden de LEDjes kleuren gegeven: bij weinig licht gaat het rode lichtje branden, bij middelmatig licht het gele en bij veel licht het groene lichtje. Een Arduino kan worden geprogrammeerd door gebruik te maken van de bijgeleverde IDE, waar men programmeert in een aparte taal. Deze Arduino taal is een versimpelde versie van C; wanneer de code gecompileerd wordt wordt hij eerst doorgegeven aan een C compiler. Hieronder een snippet van de code:

```

Serial.print(light);
if (light >= 900)
{
    digitalWrite( greenPin, HIGH );
    digitalWrite( yellowPin, LOW );
    digitalWrite( redPin, LOW );
}
else if ((800 <= light) && (light < 900))
{
    digitalWrite( greenPin, LOW );
    digitalWrite( yellowPin, HIGH );
    digitalWrite( redPin, LOW );
}
else
{
    digitalWrite( greenPin, LOW );
    digitalWrite( yellowPin, LOW );
    digitalWrite( redPin, HIGH );
}
delay(10000);

```

Arduino code: dit is de loop zelf. In een block erboven worden de variabelen gedefinieerd in de *setup*. In de loop wordt gekeken naar de data van de lichtsensor (die ook serieel wordt doorgegeven), en de juiste LED gaat branden. De loop is de block data die wordt herhaald (hier telkens met een delay van 10 seconden).

Om de data weer te geven werd voorlopig de ingebouwde serial monitor van Arduino zelf gebruikt. Hiervoor bestaat een Node.JS module genaamd *serialport*, die gegevens binnentrekt via een van de poorten. De module wordt geïnstalleerd via npm en geïnitialiseerd, waarin meteen de COM-poort wordt gedefinieerd. Hier wordt ook de gewenste baud-rate ingegeven.

De seriële data die binnen komt wordt rechtstreeks gelogd in het consolevenster, zodat een simpelere en uitbreidbare serial monitor wordt verkregen. De 2 apps die de basis vormen, de publisher en subscriber, worden gecombineerd met deze serial monitor. De *serialport.on* functie steekt de verkregen seriële data weg in een variabele *data*. In dezezelfde functie, waar eerst de console logger werd gestoken, laten we de app de functie *client.publish* van de *mqtt* module uitvoeren. Als parameter wordt de seriële data meegegeven. Zo wordt de data meteen gepubliceert als ze via de poort binnenkomt.

```

sp.on('open', function(){
    console.log('Serial Port Opened');
    sp.on('data', function(data){
        client.publish('lightmeasuring', data);
        console.log(data);
    });
});

```

The screenshot shows a Windows desktop with four open windows:

- Command Prompt - mosquitto**: Shows the command `mosquitto` being run in the Program Files (x86) directory.
- Node.js command prompt - node app.js**: Shows the command `node app.js` being run, followed by the message "Client started..." and a series of numbers from 545 to 578.
- npm**: Shows the command `> node app.js` and the message "Serial Port Opened".
- app.js (Node.js MQTT Test) - Brackets**: A code editor window showing the `app.js` file. It highlights the line `client.publish("lightmeasuring", data);` and contains a tooltip "node.js app code (publisher)". The code also includes a JSLint Problems panel with several errors related to the `var` keyword.

Fig. 7: Console vensters en de publisher

De aangekregen data wordt succesvol weergegeven in een apart consolevenster. Deze subscriber moest dan uiteindelijk de verkregen data automatisch wegschrijven in een database. Zo volgt het volgende punt.

### 4.3.2.3 Database

De database is een belangrijk element voor elk platform, maar vooral voor dit project. Wat er uiteindelijk in opgeslagen moet worden, is een collectie van alle datapunten die binnenkomen via alle OCTA-Gateways die in gebruik zijn.

Eerst was er het idee om dit met SQL te doen, en dus een relationele database te gebruiken. Hierover wat meer uitleg.

#### SQL, ofwel relationele databases

Relationele databases zijn de oudste manier van data-opslag. De taal die het vaakst gebruikt wordt om queries uit te voeren is hiervoor SQL, ofwel Structured Query Language. Het relationele model zet de data in verschillende kolommen en rijen in tabellen (relaties). Elke tabel stelt een *entiteit* voor, ofwel een concept dat meerdere instanties kan bevatten. Elke rij stelt dan een instantie of *record* voor, en elke kolom stelt velden voor die elke instantie kan bevatten. Bijvoorbeeld, "Kris" is een instantie van een persoon, dus Kris krijgt een rij. "Leeftijd" is een veld dat een persoon kan hebben dus krijgt een kolom. Elke instantie kan een *key* bevatten, dit is de "hoofdeigenschap" die een instantie kan hebben.

Voordelen:

- Keys laten toe om tabellen te verbinden aan elkaar
- Uitgebreide queries; b.v. *join* zijn enorm nuttig

Nadelen:

- Moeilijk scalable (verticaal)

## NoSQL, ofwel non-relationele databases (MongoDB)

Non-relationele of NoSQL databases is de verzamelnaam voor alle databanken die niet het SQL model volgen, en waarin de data niet opgeslagen wordt in de vorm van tabellen. De data wordt meestal opgeslagen in verschillende documenten, die gemakkelijk horizontaal uitgebreid kunnen worden. Dit wil zeggen dat, om uit te breiden, meer machines worden toegevoegd in plaats van de eerste hostmachine alleen te versterken. Het voornaamste voorbeeld hiervan is MongoDB die data in een eigen formaat opslaat dat enorm op JSON lijkt, namelijk BSON.

Voordelen:

- Simplistisch (of dit een voordeel is hangt af van applicatie tot applicatie)
- Heel gemakkelijk scalable (horizontaal)

Nadelen:

- Geen standaard

We kiezen voor MongoDB, aangezien dit zorgt voor een *scalable* database, en het gebruiksgemak die het levert in samenspraak met Node.JS. De database wordt voorlopig lokaal gedraaid, aangezien er nog geen sprake is van een userbase en de database zelf dan nog gemakkelijk te configureren is. Hiervoor wordt de community server gebruikt, gratis voor non-commercieel gebruik. De installatie gebeurt via een msi-bestand dat terug te vinden is op <https://www.mongodb.com/download-center#community>.

MongoDB werkt niet out-of-the-box met Node.JS. Hiervoor wordt de *mongodb* module geïnstalleerd: zo kunnen basis queries worden uitgevoerd op de database, zoals nieuwe collecties aanmaken en uiteraard nieuwe objecten toevoegen aan een collectie.

In een command line venster moet eerst gewisseld worden naar de installatiemap van MongoDB. Hoogstwaarschijnlijk is dit in de *Program Files (x86)* map. Hierin kan enerzijds de MongoDB service worden opgestart (de database zelf):

```
mongod
```

En als de database geconfigureerd moet worden:

```
mongo
```

Zo wordt de MongoDB client opgestart, die meteen met de standaard database verbindt. Om een lijst van alle databases te bekijken:

```
show dbs
```

De collectie waarnaar gepushed zal worden heet "lights". Wanneer mongod actief is moet eerst gewisseld worden naar de juiste database:

```
use lightdb
```

Dit werkt ook met databases die nog niet bestaan. Indien er naar zulk een database wordt gewisseld, wordt deze automatisch aangemaakt. Hierna kan gecheckt worden welke collecties aanwezig zijn in de actieve database:

```
show collections
```

Een nieuwe collectie kan aangemaakt worden

```
db.createCollection(lights, options)
```

met eventuele opties, bv. max aantal objecten.

Nadat de database is opgesteld en alle modules geinitialiseerd zijn, worden de nodige uitvoeringen geschreven in de Node.JS applicatie. Ze wordt meteen gefuseerd met de korte MQTT applicatie die eerst werd opgesteld.

```
function setupCollection(err, db) {
  if(err) throw err;
  collection=db.collection("lights");
  client.subscribe('lightmeasuring')
  client.on('message', function(topic, message) {
    var datastream = message.toString();
    console.log(datastream);
    collection.update(
      { _id:"ArduinoLightSensor" },
      { $push: { events: { value:datastream, when:new Date()} } } ,
      { upsert:true },
      function(err,docs) {
        if(err) { console.log("Insert fail"); }
      }
    );
  });
}
```

De collectie, op dit moment genaamd *lights*, wordt gedefinieerd in deze snippet. De *push* functie voegt het datapunt in, met als veld *value* de data die binnenkomt uit de mqtt *subscribe* functie. De *update* functie van mongodb wordt elke keer opnieuw uitgevoerd wanneer er nieuwe data binnenkomt. Daaronder is een veldje *when*, waar een nieuwe instantie van *Date()* wordt ingevoerd. Hier komt dus de timestamp waarop de data binnenkomt. Het nadeel hiervan is dat het niet de timestamp is waarop de data wordt waargenomen, maar dit scheelt een fractie van een seconde omdat de data zo snel wordt overgedragen door de MQTT broker. Het nieuwe object wordt telkens onder het vorige geplaatst.

Nu de data binnenkomt in de database moet er een manier zijn om aan de database te geraken zonder de MongoDB client te gebruiken. Hiervoor worden er routes geinitialiseerd, aan de hand van Express.

#### 4.3.2.4 Maken van een RESTful API

Er wordt een RESTful api gemaakt, om de applicatie te voorzien van eventuele toekomstige queries, zoals een object verwijderen (DELETE), invoegen (POST), vervangen (PUT) of op te halen (GET). Hiervoor werd een tutorial gevolgd op Scotch.io (<https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4>).

##### Express

Express of ExpressJS is een web framework gebouwd voor Node.JS. Het zorgt voor de routing binnen een web applicatie, zodat de juiste data wordt doorgegeven per request van de gebruiker. Na express via npm te hebben geïnstalleerd is gemakkelijk te integreren in een applicatie. Een statische html pagina kan worden doorgegeven door simpelweg

```
express.createServer().get('/') ...
```

te gebruiken, en de html pagina als parameter door te geven. Wanneer dan een poort wordt opgezet met *listen*, kan deze pagina bereikt worden door in een browser de localhost als URL in te geven. Zo wordt een route gecreëerd naar het juiste html bestand. Momenteel wordt poort 8080 gebruikt.

Om de applicatie volledig te maken en MongoDB te combineren met Express, wordt Mongoose geïnstalleerd.

##### Mongoose

Mongoose is een npm module die zorgt voor Object Related Mapping (ORM) binnen MongoDB en Node.JS. Het maakt schema's die gebruikt worden voor elk object dat in een Mongo database wordt opgeslagen. Hier zal het nooit van afwijken; passief toegevoegde velden worden niet aanvaard en zullen niet worden toegevoegd aan het object. Een schema wordt door de gebruiker zelf opgesteld in JSON vorm. In een apart bestand (meestal) wordt dit schema geinitialiseerd, zodat er extern naar gelinkt kan worden binnen de app, wanneer een query uitgevoerd moet worden. Hieronder het model dat gebruikt wordt voor de applicatie:

```
//MONGOOSE MODEL//
```

De belangrijkste query die uitgevoerd wordt is GET. Hier wordt de volledige database doorgegeven. Indien een MongoDB wordt weergegeven moet deze in een zichtbare of bruikbare vorm worden omgezet; daarom wordt de data uit het schema van Mongoose, indien er geen error is, eerst geparsed naar JSON formaat alvorens als response doorgegeven te worden aan de route \*/data/.

```

router.route('/data')
  .get(function(req, res) {
    Light.find(function(err, data) {
      if (err)
        res.send(err);
      res.json(data);
    });
  });

```

Code die de gehele database ophaalt. Light is de variabele voor het Mongoose-schema.

Om de database weer te geven kan een gewone browser gebruikt worden. Dit is echter minder overzichtelijk omdat de data zonder enige structuur wordt weergegeven, dus wordt er aangeraden om een specifieke applicatie te gebruiken zoals Postman. Hierover wordt gesproken in het hoofdstuk *Gebruikte tools*.

The screenshot shows the Postman application interface. At the top, there's a navigation bar with tabs for 'Builder', 'Runner', and 'Import'. Below the navigation bar, the URL is set to 'localhost:8080/api/data'. The main area shows a 'GET' request to 'localhost:8080/api/data'. Under the 'Authorization' tab, it says 'No Auth'. The 'Body' tab is selected, showing the JSON response from the server. The JSON output is:

```

1. [
2.   {
3.     "_id": "56dd930c46d913e81e126fcf",
4.     "lightamount": "800",
5.     "events": [
6.       {
7.         "events": [
8.           {
9.             "event": {
10.               "when": "2016-03-08T14:02:02.788Z",
11.               "value": "939"
12.             }
13.           },
14.           {
15.             "event": {
16.               "when": "2016-03-08T14:02:03.748Z",
17.               "value": "940"
18.             }
19.           },
20.           {
21.             "event": {
22.               "when": "2016-03-08T14:02:04.732Z",
23.               "value": "939"
24.             }
25.           },
26.           {
27.             "event": {
28.               "when": "2016-03-08T14:02:05.737Z",
29.               "value": "937"
30.             }
31.           },
32.           {
33.             "event": {
34.               "when": "2016-03-08T14:02:06.737Z",
35.               "value": "933"
36.             }
37.           },
38.           {
39.             "event": {
40.               "when": "2016-03-08T14:02:07.741Z",
41.               "value": "930"
42.             }
43.           }
44.         ],
45.         "events": []
46.       }
47.     ]
48.   ]
49. ]

```

Fig. 8: Inhoud van de database via Postman

### 4.3.2.5 Frontend

Het volgende punt was de frontend maken, wat de gebruiker uiteindelijk ziet, aangezien een simpele JSON niet echt mooi of overzichtelijk is. Hiervoor werd AngularJS gebruikt, omdat het gebruiksvriendelijk is terwijl het de extensiviteit behoudt die we nodig hebben voor de applicatie. Er werd ook gebruik gemaakt van Bootstrap voor de esthetica, en uiteindelijk een bootstrap template voor het dashboard zelf.

#### AngularJS

AngularJS is een open source framework bedoeld om webapplicaties te schrijven. Het is vooral gericht op one-pagers, dus applicaties of webpagina's die volledig in één pagina staan, en wordt hoofdzakelijk onderhouden door Google. Er bestaat een nieuwe Angular 2 versie maar in dit project wordt AngularJS 1 gebruikt, simpelweg omdat de functies van Angular 2 niet noodzakelijk zijn voor de werking van dit project.

AngularJS volgt het model-view-controller patroon (MVC). Dit wil zeggen dat het belangrijkste element het *model* is. Deze geeft de *view* weer; dat wat door de gebruiker gezien en gebruikt wordt. Hier zit ook de GUI in, met alle input en output. De user gebruikt alles in de view, wat de nodige commands doorgaat aan de *controller*. Hier zit de werking van de applicatie zelf. De commands worden omgezet en de juiste variabelen worden uiteindelijk aan het model gegeven, die het op zijn beurt opnieuw in de view steekt voor de gebruiker. Dit is een cyclus die de applicatie doet updaten. Hieronder een schets:

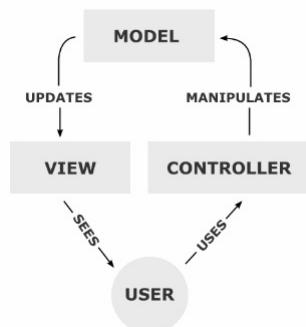


Fig. 9: Het MVC-model.

De basis van elke AngularJS applicatie bestaat uit een html bestand (model/view) en een controller (JavaScript bestand).

## Bootstrap

Bootstrap is een framework bedoeld voor front-end ontwikkeling en design. Bootstrap zelf is een collectie van configurerbare CSS, die een heleboel layout standaardiseren. Ook is het een verzameling van alle templates die er gebruik van maken (inclusief HTML en CSS). Het is ontwikkeld door enkele ingenieurs op Twitter om intern de layout consistent te houden over alle tools die ze ontwikkelden. Dit draaide uit op een geheel framework dat universeel gebruikt wordt. Sinds versie 2 ondersteunt het *responsive web design*, zodat de layout zich automatisch aanpast naargelang de grootte van het gebruikte scherm (denk desktops, mobieljes, tablets). Sinds 3.0 staat dit centraal onder het *mobile-first* principe. De basis hiervan ligt bij een grid dat verdeeld wordt in 12 kolommen om zo de juiste verdeling van elementen te voorzien op elk toestel.

Bootstrap kan gemakkelijk geïntegreerd worden als een stylesheet, als volgt:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" integrity="sha384-1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7" crossorigin="anonymous">
```

Officiële link die te verkrijgen is op [getbootstrap.com](http://getbootstrap.com).

De controller wordt aangemaakt.

```
angular.module("lightmeasurer",[]).controller("lightController", function ($scope, $http)
```

Er wordt een overkoepelende module aangemaakt zodat meerdere controllers gebruikt kunnen worden (voor charts: hierover later meer)

```
angular.module('app', ['lightmeasurer', 'chart']);
```

Hierin is het belangrijkste element een http-request die alle data ophaalt.

```
function LoadData(){
  $http({
    method: 'GET',
    url: 'http://localhost:8080/api/data'
  }).then (function successCallback(response) {
    console.log("JSON loaded!");
    var json = response;
    console.log(json);
  }, function errorCallback(response) {
    //alert("An error occurred while fetching the sensor data!")
  });
}
```

Deze \$http functie kan alle RESTful services uitvoeren. Hier wordt GET gebruikt, deze haalt het gehele document op die verkregen wordt op de gegeven URL. De route die zonet werd aangemaakt aan de hand van Express wordt nu ingegeven. De response van de \$http get wordt in een variabele *json* gestoken.

Het gemak van AngularJS zit hem in de connectie tussen de controller en de view. Er is een functie \$scope in Angular, die een variabele zichtbaar en bruikbaar maakt voor zowel de controller als de view. Wanneer we dan de json variabele in de scope zetten, kan deze worden gebruikt in de view:

```
$scope.lightdata = 0;
$scope.lightdata = json;
```

Er wordt een \$scope variabele *lightdata* aangemaakt waarin de json wordt gestoken. De \$scope wordt telkens weer leeggemaakt wanneer hij wordt opgehaald aangezien de json zich dan niet onderaan de bestaande scope vestigt. Nu moet eerst de view verbonden worden aan de controller.

Achter de body tag in het html-bestand plaatsen we de directive *ng-app*. Hierin wordt verwezen naar de juiste (overkoepelende, zodat ze allemaal worden geladen) Angular module, namelijk *app*. Een andere directive die we gaan gebruiken is *ng-repeat*. Deze haalt alle data uit een json object, en kijkt welke velden getoond moeten worden in een tabel. Zo wordt een lijst aangemaakt van deze velden van alle opeenvolgende objecten. De ng-repeat wordt in een table gestoken, die op zijn beurt in een div wordt gestoken. In deze div wordt eerst de juiste controller gelinked, namelijk *lightController*.

Vervolgens dient de json voldoende geanalyseerd te worden. De console logt automatisch het gehele object, dus kan het bekijken worden in een browser. Door op F12 te klikken wordt deze geopend, en het object kan geopend worden.

```
//FOTO JSON IN CONSOLE CHROME//
```

Hier wordt duidelijk dat er in de json verschillende verzamelingen zitten. *Data* is diegene die nodig is, hierin zit de verzameling *events* waarin alle datapunten verscholen zitten. De verzameling data is een array, dus moeten we er juist naar linken:

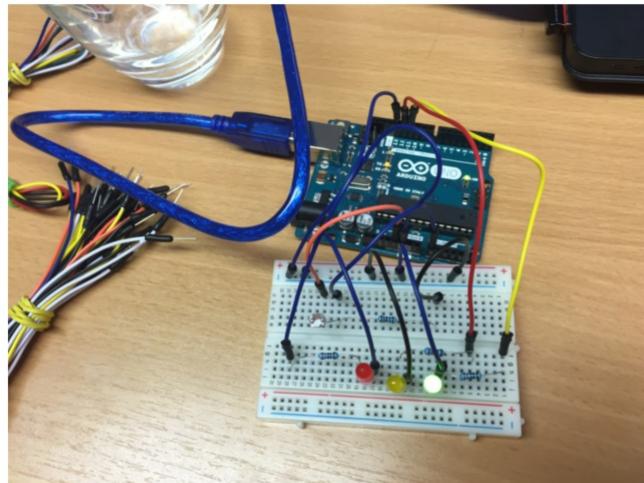
```
ng-repeat="events in lightdata.data[0].events"
```

De database bleek niet helemaal de correcte vorm te hebben. Er werd een delimiter in de Arduino code gestoken terwijl hij al aan het versturen was, dus moet de collectie gereset worden. Dit kan gedaan worden door de Mongo client te openen in een console venster en te veranderen naar de juiste database, om vervolgens de collectie leeg te maken.

```
db.lights.drop()
```

De Arduino blijft data versturen, dus de collectie wordt snel opnieuw opgevuld. Wanneer de pagina gerefreshed wordt, verschijnt een tabel met alle data van de lichtsensor.

## Measured light in D22



947  
2016-03-10T13:11:27.039Z

932  
2016-03-10T13:11:28.019Z

930  
2016-03-10T13:11:29.031Z

931

*Fig. 10: Data wordt getoond in een lijst*

```

[{"events": [{"when": "2016-03-10T14:27:36.793Z", "value": "950"}, {"when": "2016-03-10T14:27:37.792Z", "value": "950"}, {"when": "2016-03-10T14:27:38.782Z", "value": "949"}, {"when": "2016-03-10T14:27:39.811Z", "value": "950"}, {"when": "2016-03-10T14:27:40.824Z", "value": "949"}, {"when": "2016-03-10T14:27:41.833Z", "value": "949"}, {"when": "2016-03-10T14:27:42.789Z", "value": "948"}]}

```

*Fig. 11: Database via Postman*

Het volgende plan was de view automatisch laten updaten en eventueel een chart of meerdere in de view verwerken.

De controller zat op dit moment nog integraal in het html document. Dit moest extern gelinked worden maar er verscheen steeds een server side error in de console (500).

Na Stack Overflow te raadplegen bleek dat dit een probleem was dat met express te maken had. Niet alle correcte bestanden werden door express doorgegeven dus moesten enkele lijnen toegevoegd worden aan de server app:

```
app.use(express.static('public'));
app.use('/bower_components', express.static(__dirname + '/bower_components'));
```

Hierdoor wordt de public folder doorgegeven (waarin de view en de javascript folders zitten verwerkt). De tweede lijn geeft alle bower components door. Deze map zit niet in de public map, en zal later worden uitgelegd, aangezien het nodig is voor de charts.

Er wordt wat meer ingegaan op de mogelijkheden die Angular ons biedt. Enkele features die nog nodig zijn: de applicatie *dynamisch* maken en de view automatisch laten updaten en een \$scope variabele laten overgaan tussen controllers; dit is nodig voor de chart die nu opgesteld zal worden.

## Charts

De data wordt momenteel getoond in de vorm van een simpele tabel. Dit moet veranderen in een overzicht (dat uiteindelijk automatisch updateert) dat de verandering gemakkelijk aantoonbaar voor de gebruiker. Er waren 2 grote mogelijkheden: ofwel een Bootstrap template gebruiken die reeds de nodige charts had en deze gebruiken, ofwel een JavaScript library te zoeken die gepersonaliseerd kon worden. Daarom werd er voor het laatste gegaan en onderzoek gedaan naar een passende AngularJS plugin om charts (grafieken) te tonen. Hier alvast twee opties:

### *AngularJS Charts van FusionCharts*

FusionCharts is een library die verschillende charts aanbiedt voor web applicaties gebouwd met JavaScript. Er is enorm veel keuze, van line charts tot 3D pie charts, en vooral ook compatibel met alle platformen, inclusief mobile. De charts zelf zijn simpel en duidelijk genoeg om overzichtelijk te blijven, met bonussen zoals mouse-overs om meer info te laten zien over de gevisualiseerde data. Voordelen:

- Uitgebreide keuze
- Enorm responsief
- Charts exporteren als afbeeldingen of PDF

### Nadelen

- Prijs ligt hoog (gratis trial maar 200\$ voor een developer's license)

Te vinden op: [www.fusioncharts.com/angularjs-charts/](http://www.fusioncharts.com/angularjs-charts/)

### *Angular Chart.js*

Angular Chart is een open-source library geschreven door Jerome Touffe-Blin (@jtblin op GitHub). De charts zijn minder uitgebreid dan de library van FusionCharts en ze laten op vlak van gebruiksgemak meer over aan de gebruiker zelf. Er zijn 7 verschillende grafieken, waarvan de belangrijkste de line chart. Mouse-overs zijn hier ook beschikbaar.

### Voordelen:

- Simpele library, open-source
- Responsief (met animatie)
- Gratis

Nadelen:

- iets minder gebruiksvriendelijk; setup duurt iets langer

Te vinden op: <http://jtblin.github.io/angular-chart.js/>

Vooral kijkend naar de prijs, valt de keuze op de tweede optie; Angular Chart.js. De installatie gebeurt via bower.

## Bower

Bower is een package manager, niet heel verschillend met npm, maar is niet gelimiteerd tot Node.JS modules. Het wordt (globaal) geïnstalleerd via npm:

```
npm install -g bower
```

Bower kan modules beheren, installeren en verwijderen, die dus niet enkel Node.JS applicaties beïnvloeden maar vooral gericht zijn op front-end ontwikkeling. Ze kunnen dus ook HTML, CSS, algemene JavaScript en afbeeldingen bevatten. Bower bevat een json bestand gelijkaardig met het package.json bestand waar het alle versies van alle geïnstalleerde modules bijhoudt genaamd bower.json. De installatie van packages gebeurt zoals dat met een npm package:

```
bower install <package>
```

Nadat bower geïnstalleerd is kunnen we beginnen met de installatie van Chart.js. Dit gebeurt, na in het command line venster te hebben gewisseld naar de projectmap, zo:

```
bower install angular-chart.js
```

Dit installeert de nodige bestanden die nodig zijn. Er wordt een map bower\_components aangemaakt. Hierin zit een bestand angular-chart.js/dist/angular-chart.js dat de nodige JavaScript bevat voor alle grafieken. Hier moet naar gelinked worden, maar ook naar de nodige css (die mee wordt geïnstalleerd met bower) wordt een link aangemaakt in de pagina:

```
<script src="bower_components/angular-chart.js/dist/angular-chart.js"></script>
<link rel="stylesheet" href="bower_components/angular-chart.js/dist/angular-chart.css">
```

In de controller.js moet een nieuwe dependency worden gedefinieerd voor de chart.js module. We gebruiken een line chart:

```

angular.module("app", ["chart.js"]).controller("LineCtrl", function ($scope) {

    $scope.labels = [];
    $scope.data = [
        []
    ];
    $scope.onClick = function (points, evt) {
        console.log(points, evt);
    };
});

```

In de \$scope.labels array moeten alle nodige labels komen die onderaan de chart komen te staan. Hier zullen de timestamps komen. In de \$scope.data array komen alle datavalues terecht. De arrays moeten even lang zijn zodat de juiste value bij de juiste label staat. Deze setup kan gemakkelijk getest worden door er willekeurige waardes in te steken. In de html pagina wordt er een canvas aangemaakt waarin de chart terecht komt:

```

<canvas id="line" class="chart chart-line" chart-data="data" chart-labels="labels" chart-legend="true" chart-series="series" chart-click="onClick"></canvas>

```

Als de html pagina nu wordt geladen verschijnt er een lege grafiek. Om de data in de grafiek te laten moeten we de \$scope eerst zien over te brengen naar de chart module. Hiervoor wordt er eerst wat onderzoek gedaan naar eventuele mogelijkheden van AngularJS.

We lijsten mogelijkheden op met de \$scope functie:

- \$scope.\$apply: een functie die opgeroepen kan worden om bindings te updaten
- \$scope.\$watch: een functie die steeds kijkt naar bindings (scope) om te zien of de gegevens veranderd zijn; zoja wordt er een functie uitgevoerd.
- \$scope.\$timeout: een functie wordt uitgevoerd na een bepaalde tijdsduur
- \$scope.\$interval: een functie wordt uitgevoerd met intervallen ertussen

Ook bestaan er opties om een \$scope variabele te laten overbruggen tussen 2 controllers:

- \$scope.\$broadcast: zendt de scope van de ene controller naar alle andere controllers, die er op hun beurt op kunnen "subscriben" indien ze de data nodig hebben.
- Services: overkoepelende "controllers" waarin de data wordt opgeroepen en alle children de data kunnen overnemen.

De \$broadcast functie wordt gebruikt, zodat de data die wordt doorgegeven opnieuw gebruikt kan worden als er een nieuwe controller wordt aangemaakt die niet afstamt van een overkoepelende controller, of een die afstamt van een andere irrelevante controller.

In de controller waar de \$scope gebroadcast moet worden, moet gebruik worden gemaakt van de \$rootScope. Elke applicatie heeft 1 \$rootScope, en elke andere \$scope stamt hier van af. Hier de code van de broadcaster:

```
$scope.lightdata = json;
$rootScope.$broadcast('lightcast', $scope.lightdata);
```

De naam van de broadcast is *lightcast*. Als data die gebroadcast moet worden wordt de `$scope.lightdata` meegegeven; de json die werd opgehaald. In de andere controller wordt de functie `$on` gebruikt, die alle data van een bepaalde topic binnenhaalt.

```
$scope.$on('lightcast', function lightCast(events, args){
    $scope.lightdata = args;
    console.log($scope.lightdata);
```

De data van het topic "lightcast" komt binnen en wordt weggeschreven in het argument `args`. Hiervoor wordt een nieuwe `$scope` gedefinieerd, eveneens genaamd *lightdata* waarin de data meteen wordt gestoken. Wanneer de pagina wordt geladen, schrijft de console opnieuw het gehele object uit (nu twee keer; een keer door de oorspronkelijke *broadcastcontroller*, en nu een tweede keer). Dit bewijst dat de broadcast actief is. Op te merken: het is vanzelfsprekend dat tijdens het debuggen de server draait, anders kan geen data worden doorgegeven en wordt er niets opgehaald om getoond te worden in de console.

De data komt nu binnen in json formaat; hier kan niets mee gedaan worden in de chart, die enkel arrays aanvaardt van pure data. De juiste velden moeten dus uit de json gehaald worden. De verwijzing hiernaar gebeurt op ongeveer dezelfde manier als waarop de data in de eerste instantie in de front-end werd geladen:

```
$scope.lightdata = args;
for(var i = 0; i < 25; i++)
{
    $scope.datapoint = parseInt($scope.lightdata.data[0].events[i].value);
    $scope.dataarray.push($scope.datapoint);
    $scope.datepoint = $scope.lightdata.data[0].events[i].when;
    $scope.datearray.push($scope.datepoint);
    if ($scope.dataarray.length > 25)
    {
        $scope.dataarray.shift($scope.dataarray[0]);
        $scope.datearray.shift($scope.datearray[0]);
    }
}
```

Hierboven de for-loop die wordt gebruikt om de data in twee arrays te steken. Er wordt gekeken naar de laatste 25 events in de database, enerzijds naar de *value* (die in de dataarray wordt gepushed) en anderzijds de *when* ( deze wordt in de datearray gepushed). Er werd voor 25 gekozen door de grootte van de chart. Als het aantal kleiner was, zou er gewoonweg onvoldoende data worden getoond. Was het groter, zou de grafiek te chaotisch zijn. Aangezien de arrays beiden dezelfde grootte hebben, zal het altijd correct zijn; er zullen altijd evenveel values als timestamps zijn.

Het volgende is het automatisch updaten van de view.

## \$timeout of \$interval

Om een view automatisch te updaten is er een timer nodig. Gelukkig zitten er in AngularJS verschillende functies die daarbij helpen: onder andere `$timeout` en `$interval`. Dit zijn allebei wrappers voor timers die integraal in JavaScript verwerkt zitten, namelijk `window.setTimeout` en `window.setInterval`. Het verschil ertussen is simpel: `$timeout` neemt een periode waarna de functie eenmaal wordt uitgevoerd. `$interval` neemt als argument eveneens een tijdspanne, maar voert de functie telkens opnieuw uit na elk `tijdsinterval`.

De `LoadData()` functie van eerder wordt dus in een `$interval` gestoken. Zo wordt de http request om de zoveel seconden (10) uitgevoerd, en wordt de view automatisch geüpdated. Met de nieuwe automatisch updatende grafiek vormde zich een nieuw probleem: De Angular chart heeft een animatie die de grafiek telkens opnieuw tekende omdat de label onderaan veranderde. De timestamps zullen altijd veranderen dus moet deze animatie stop worden gezet. Dit kan gedaan worden door in de opties van de chart de boolean `animation` naar `false` te zetten.

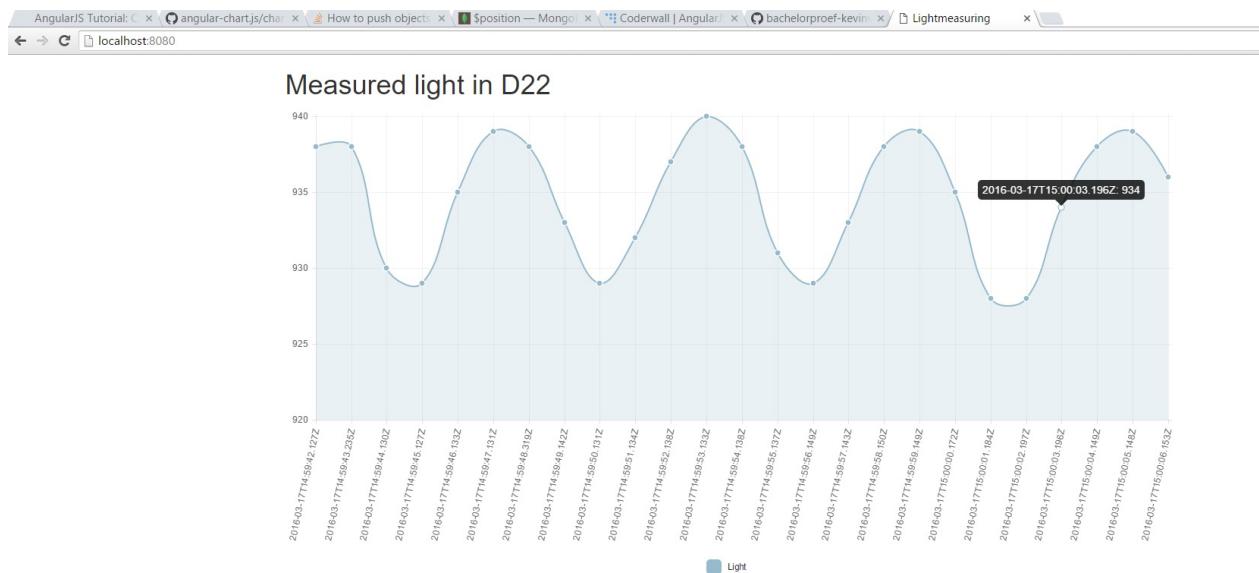


Fig. 12: Chart van lichtdata

Dit was de allereerste versie van het dashboard. Het moest veel aantrekkelijker zijn en dus werd er gezocht naar een passend dashboard-template in Bootstrap. Er werd gekozen voor *SB Admin*, te vinden op StartBootstrap (<http://startbootstrap.com/template-overviews/sb-admin/>). Het gebruik hiervan is volledig gratis. Enkele features van het dashboard, die werden geschrapt:

- Notificatie-panels
- Messaging tussen gebruikers
- Pop-ups

Wat overbleef was een simpel maar elegant dashboard dat gemakkelijk te personaliseren was. Het integreren van de charts verliep vlot; het moest enkel in een div worden geplaatst en de verwijzingen moesten overgeplaatst worden naar de header van de template. Het voordeel aan het dashboard is dat het mobiel zonder problemen loopt; indien het scherm te klein is wordt het menu gewoonweg ingevouwd, en kan worden uitgeklapt met een button.

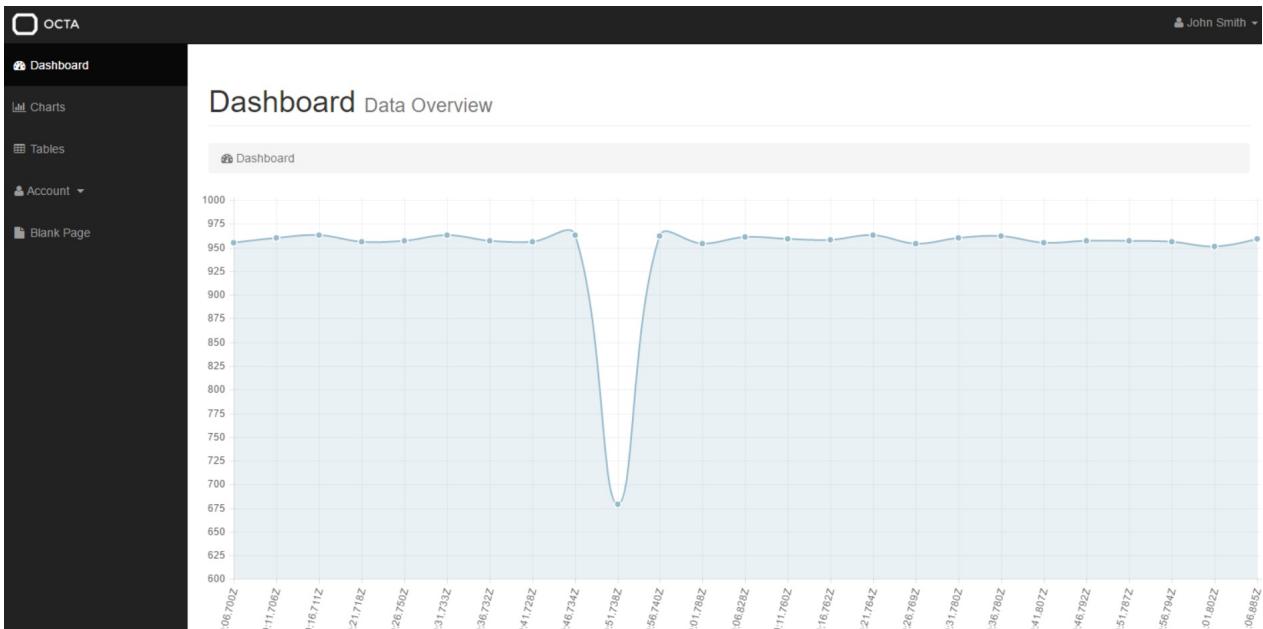


Fig. 13: Het dashboard vanaf het moment dat de chart en de template werden samengevoegd

## Google Maps

Op de pagina *places* zullen alle locaties worden geladen in een Google Map. Deze locaties komen van de coordinaten die doorgegeven werden door de gps-module. Om een Google Map te laden moet de API gebruikt worden. Hiervoor moet een key aangevraagd worden op <https://developers.google.com/maps/documentation/javascript/get-api-key>. De API dient geladen te worden:

```
<script src="https://maps.googleapis.com/maps/api/js?key=KEYHERE&callback=initMap" async defer></script>
```

Als dit gebeurd is kan een map geladen worden door gewoonweg een map container op te stellen:

```
<div id="map"></div>
```

In een JavaScript bestand kan een scriptje worden opgesteld om dan de map in te stellen. Zo kunnen markers geinitialiseerd worden en een route getraceerd. Markers kunnen allerlei labels met zich meedragen die ook hier opgesteld kunnen worden. Een goede gids is te vinden op de Google Maps website zelf: <https://developers.google.com/maps/documentation/javascript/tutorial>.

Ze worden opgeslagen als Little Endian Floats in hexadecimale vorm. De endianness (zie *glossary*) moet dus eerst worden omgezet.

### 4.3.2.7 Authenticatie

Het laatste element dat geïntegreerd is in de applicatie is de authenticatie ervan. Hiervoor bestaat een module genaamd PassportJS die geïnstalleerd wordt via npm.

#### PassportJS



PassportJS identificeert zichzelf als authenticatie *middleware* voor NodeJS. Middleware is de verzamelnaam voor de hoop acties die uitgevoerd moeten worden vooraleer de data zijn eindbestemming bereikt. Hier hoort vooral Express bij, aangezien de data eerst de routes moet volgen die door Express worden opgesteld. PassportJS is een ander voorbeeld hiervan aangezien er eerst moet gekeken worden naar de ingelogde gebruiker, en of die wel de juiste rechten bezit om aan die data te geraken. Het ondersteunt authenticatie voor een username en paswoord, maar ook authenticatie via Facebook, Twitter, Google Plus en meer. Authenticatie via een provider zoals Facebook gebeurt via OAuth 2.0; een vorm van token-based authenticatie. Dit wil zeggen dat wanneer een gebruikersnaam en paswoord correct worden ingegeven, de gebruiker een *token* of "toegangspas" krijgt. Deze is voor een bepaalde tijd geldig en verleent toegang tot enkele diensten zonder de gebruikersnaam en paswoord opnieuw te hoeven ingeven. Voor dit project wordt echter gewone authenticatie gebruikt (via de passport-local module), en is enkel een gebruikersnaam en paswoord nodig (en een OCTA UID).

Express is wel nodig vooraleer PassportJS gebruikt kan worden. Als Express echter reeds is geïnstalleerd is het gebruik ervan enorm vlot. De installatie van de modules gebeurt via npm (passport en passport-local tegelijk):

```
npm install passport-local passport
```

Hier zullen beknopt de belangrijkste elementen worden uitgelegd om authenticatie te laten werken aan de hand van PassportJS. De volledige uitleg is te vinden op de officiële documentatie: <http://passportjs.org/docs/username-password>. Allereerst moet de module gedefinieerd worden. Dan wordt een route aangemaakt om in te loggen:

```
app.post('/login',
  passport.authenticate('local', { successRedirect: '/',
    failureRedirect: '/login',
    failureFlash: true })
);
```

In de form in de view wordt via de submit-knop een link gemaakt naar deze `/login` route. De local strategy wordt dan geladen en er wordt gekeken of de gebruikersnaam en paswoord effectief correct zijn. Zoja, vindt er een redirect plaats naar de route `/`, en dus de index pagina van de applicatie. Wanneer de gebruiker moet worden opgeroepen kan gewoonweg `req.user` worden gebruikt. Dit laadt alle gegevens van de momenteel ingelogde gebruiker. Voor het uitloggen en registreren worden er eveneens routes aangemaakt.

Op het dashboard kan nu gemakkelijk rechtsboven weergegeven worden welke gebruiker momenteel is ingelogd, aan de hand van de syntax die gebruikt wordt voor JSON bestanden en de naam van de velden: `req.user.firstName` in dit geval.

```
//SCREENSHOT DASHBOARD RECHTSBOVEN//
```

Aan de hand van een filter in de Angular code bij ng-repeat kan enkel de data van de momenteel ingelogde gebruiker worden getoond. Dit werkt altijd: is er niemand ingelogd en het dashboard wordt op een of andere manier toch geladen, verschijnt er niets aangezien er gefilterd wordt op *undefined*.

```
ng-repeat="events in lightdata.data[0].events | filter: { deviceid: $scope.currentuserid }"
```

#### 4.3.2.6 Maken van een executable

De manier waarop de applicatie momenteel opgestart wordt, is via de Node.JS command prompt die bij Node.JS geleverd wordt en *nodemon app.js* te runnen. Dit is niet de manier waarop software normaal geleverd wordt; er moet dus een executable van gemaakt worden, eventueel zelfs een GUI. Eerst wordt gekeken hoe een executable gemaakt kan worden van een Node.JS applicatie. Hiervoor bestaan verschillende manieren.

##### nexe

Gemaakt door Jared Allard en te vinden op zijn GitHub: <https://github.com/jaredallard/nexe>. Nexe zorgt ervoor dat de applicatie heel snel werkt, maar het nadeel hiervan is dat dynamische module requirements niet werken. Dit wil zeggen dat een module niet opgeroepen wordt als de *require* statement in een andere statement zit. Momenteel is dit niet het geval, maar dit kan in de toekomst problemen leveren. Er bestaan betere alternatieven.

##### EncloseJS

Geschreven door Igor Klopov en te vinden op de homepagina: enclosejs.com. EncloseJS doet hetzelfde als nexe, maar simpeler en het ondersteunt wel dynamische requirements. Andere voordelen, ook terug te vinden op de website:

- Node.JS en npm hoeven niet geinstalleerd te worden om de applicatie te gebruiken
- Alle bestanden uit de node\_modules hoeven niet geinstalleerd te worden, ze worden mee gepackaged in een bestand
- Assets, zoals HTML en CSS bestanden worden eveneens mee gepackaged in de distributable
- EncloseJS werkt vrijwel meteen; wanneer de command ingevoerd wordt staat de executable er binnen enkele seconden

Het installeren ervan gebeurt globaal via npm:

```
npm install -g enclose
```

De applicatie zelf draait door de command op te starten binnen de projectfolder:

```
enclose
```

zonder argumenten.

Nu draait de applicatie sneller en gemakkelijker door enkel de executable te hoeven openen. Het volgende is een GUI toevoegen aan de applicatie.

#### 4.3.2.6.1 GUI toevoegen

De GUI van de applicatie is wat de gebruiker ziet, met allerlei invoerknoppen en invoervelden. Dit is noodzakelijk voor een package die gedistribueerd wordt. Ook hiervoor bestaan programma's die helpen bij het maken van een desktopapplicatie van Node.JS met een GUI. Hiervoor bestaat een enorm populair framework:

#### 4.3.2.6.2 Electron



Electron is ontwikkeld door ingenieurs van GitHub zelf; dezelfde mensen die de text editor Atom hebben gemaakt. Versie 1.0 is op dit moment net gereleased. Het debuggen binnen Electron werkt op een speciale manier. De engine moet eerst geïnstalleerd worden vooraleer het gebruikt kan worden. Het is gebouwd op Chromium; het open-source project van Google dat de volledige broncode van Google Chrome heeft geleverd. De engine hiervan is *WebKit*. Wanneer een applicatie gebouwd op Electron wordt gestart, draait deze dus op een "eigen" browser genaamd Chromium. In deze basis browser zijn verschillende elementen rechtstreeks geïntegreerd:

- Flash Player
- PDF-lezer
- Programma dat de browser automatisch updatet naar de recentste versie (Google Update)

Een basis applicatie gebouwd op Electron bevat de volgende elementen:

- Package.json. Deze houdt alle dependencies bij van alle modules en de locatie van de hoofdapplicatie.
- Main.js. Dit is de hoofdapplicatie, hier wordt het venster in geinitialiseerd en alle code achter de applicatie zit hierin.
- Index.html. Dit is de view. Hier wordt de GUI geprogrammeerd in HTML.

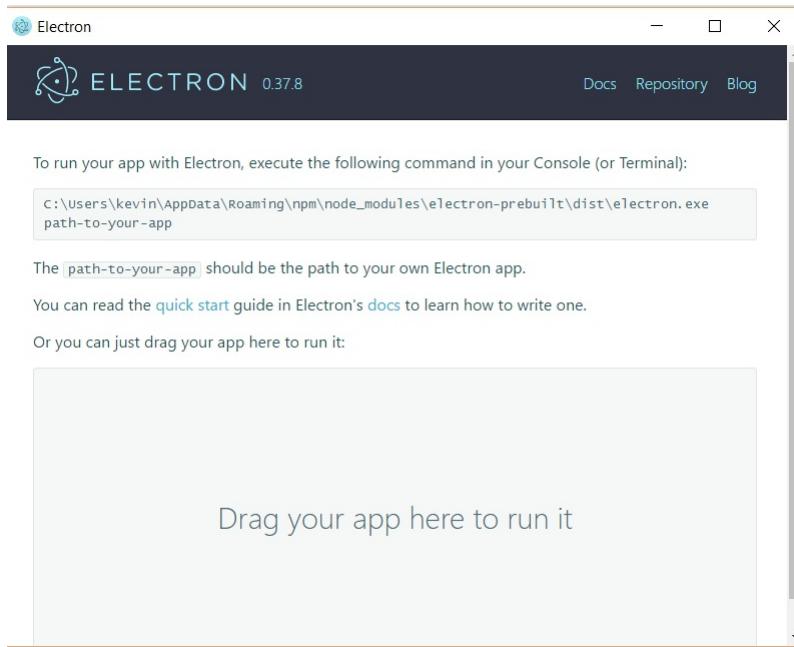
Een goede manier om met Electron aan de slag te gaan is de Quick Start applicatie die beschikbaar wordt gesteld op GitHub: <https://github.com/electron/electron-quick-start>. Wanneer Electron dan geïnstalleerd is, zijn ook de juiste commands geïnstalleerd om het framework te gebruiken. Om de applicatie te personaliseren en esthetischer te maken wordt dus de HTML geconfigureerd. Het grote voordeel hieraan is dat ook stylesheets hiervoor ondersteund worden. Dit bewijst ook dat bootstrap gebruikt kan worden, en zelfs de eventuele templates. Dit laatste is echter niet nodig voor zulk een kleine applicatie.

```

const electron = require('electron')
// Module to control application life.
const app = electron.app
// Module to create native browser window.
const BrowserWindow = electron.BrowserWindow

```

In de header van het main.js bestand worden de referenties gemaakt naar noodzakelijke modules. Merk op dat dit heel gelijkaardig is aan een node.js bestand met de require functie om externe modules te integreren. Met de functie createWindow() wordt het venster geinitialiseerd, inclusief de grootte ervan. Deze 2 onderdelen zijn het belangrijkst om een electron app te maken. Andere belangrijke functies zijn de app.on() en app.quit() functies. Het debuggen (of gewoonweg runnen) van de applicatie voordat ze gepackaged wordt werkt als volgt: in de root van de applicatie zelf wordt in een command line venster het commando "electron" gerund (nadat electron globaal of lokaal geïnstalleerd is via npm install electron-prebuilt). Dit start de splash screen op:



*Fig. 14: De splash screen die opgestart wordt als Electron wordt uitgevoerd*

Hierin kan de hoofdapplicatie gesleept worden. Er wordt dan naar de package.json gekeken en de index.html pagina wordt geladen. In het Chromium venster wordt dan de gehele applicatie geladen.

Voor de publishing applicatie is enkel een form nodig, die de nodige input aanvaardt en deze door geeft aan het main.js bestand. Hier kan de baud rate ingesteld worden en de COM-poort geselecteerd worden. Ook kan er eventueel een URL worden ingegeven als de data naar een andere locatie verzonden moet worden maar dit is enkel noodzakelijk voor third party dashboards (waarover later meer). Als alles is ingevuld kan er op de groene button geklikt worden (submit) om te beginnen uitlezen via de seriële poort en onmiddelijk te publishen. Zo start in de main.js applicatie de loop die hetzelfde doet als de applicatie die ervoor werd gemaakt om de data te verzenden over MQTT.

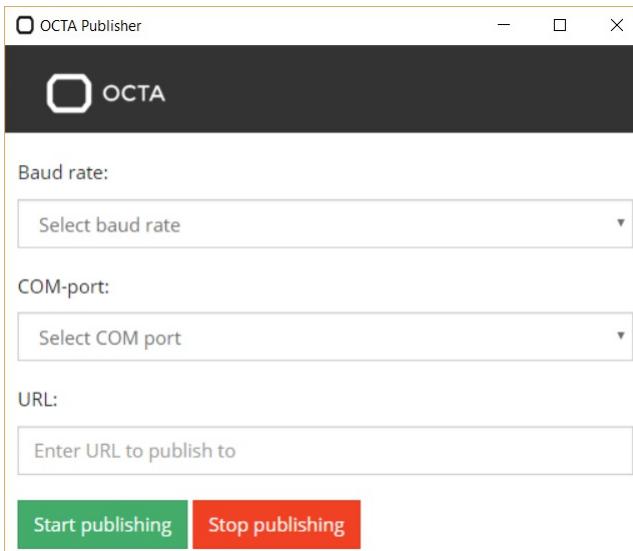


Fig. 15: Eerste versie van de Electron app

De applicatie liep vlekkeloos, totdat de serialport module werd geïntegreerd. Deze leverde een DLL error op:

```
C:\Users\kevin\Documents\School\Electron\electron-quick-start\node_modules\serialport\node_modules\bindings\bindings.js:8
Uncaught Error: A dynamic link library (DLL) initialization
routine failed.
\\?\C:\Users\kevin\Documents\School\Electron\electron-quick-
start\node_modules\serialport\build\Release\serialport.node
```

Fig. 16: DLL Error die zich voordeed wanneer de serialport module werd gedefinieerd

Hiervoor werd enorm veel tijd gestoken in het debuggen. Op fora werd gezegd dat het opgelost kon worden door de applicatie opnieuw te bouwen, maar tevergeefs. Het bleek te gaan om een populair probleem, zo frequent zelfs dat enkelen een aangepaste module hiervoor hadden gemaakt, specifiek voor electron (<https://github.com/voodootikigod/node-serialport> is de originele actieve module en <https://github.com/usefulthink/node-serialport> is een forked versie hiervan die tegen de Electron runtime werd gemodificeerd). Er verscheen een nieuwe error, dat de serialport.node module gewoonweg niet werd gevonden. Hoogstwaarschijnlijk was de repo *deprecated* en dus niet meer werkende. Wanneer dan hierop de originele serialport module werd geïnstalleerd kwam opnieuw de gevreesde error. Er werd dieper ingegaan op het probleem.

De meeste Node.JS modules zijn gebouwd met de node-gyp tool, gemaakt om addons te ontwikkelen. Het probleem was dat node-serialport hier niet mee werd gebouwd maar met node-pre-gyp. Voodootikigod (de ontwikkelaar van serialport) heeft hier zelf een artikel over geschreven: <http://www.voodootikigod.com/on-maintaining-a-native-node-module/>. Hij beschrijft hierin het probleem dat hij zag in de gewone node-gyp tool. Alle dependencies moesten apart geïnstalleerd worden indien er een package via node-gyp gebruikt zou worden. Hij kreeg ontzettend veel issues binnen van mensen die problemen hadden met het installeren. Node-pre-gyp had dit probleem niet want deze tool gebruikte op voorhand gecompileerde libraries om een addon te maken. Het probleem was dat Electron geen ondersteuning biedt voor node-pre-gyp modules. Het voordeel is dat er enorm veel mensen werken om zulke modules te laten werken met Electron; het nadeel is dat dit te laat zal zijn voor dit project. Er moet dus een drastische beslissing genomen worden.

#### 4.3.2.6.3 Migratie naar NW.JS

Er werd overgeschakeld naar het alternatief, NW.js. Het is minder populair en biedt dus minder support aan maar het werkt op exact dezelfde manier als Electron.

## NW.JS

NW.JS, te vinden op nwjs.io, heette voormalig node-webkit. Het is een engine die de Node.js applicaties rechtstreeks vanuit het DOM runt (Document Object Model: een object-georienteerde structuurstandaard van bv HTML-bestanden die werd ontwikkeld omdat vroeger elke browser een eigen manier van toenadering had tot bestanden). De applicatie zelf wordt dus met webtechnologieen geschreven. De view wordt in HTML geschreven en de code die erachter zit, die de applicatie werkende maakt, is in JavaScript geschreven. Het voordeel hiervan is dat modules met npm geïnstalleerd kunnen worden en gebruikt kunnen worden zoals in normale Node.JS applicaties. Die worden dan uiteindelijk mee gepackaged in de distributable of deliverable. NW.js heeft een hoogsteigen engine waarop de applicatie loopt, die in essentie dus eigenlijk niets meer is dan een stripped down browser, inclusief developer tools. Als je de basis nwjs executable opent, zie je dan ook een blank venster met de toolbar die wijst op een browservenster (deze toolbar kan verwijderd worden in het configuratie bestand).

Wanneer het gedownload is (<http://dl.nwjs.io/v0.15.1/nwjs-v0.15.1-win-x64.zip>) zit alles er reeds in. Een executable die, wanneer geopend, een leeg venster toont met bovenaan de toolbar. In de rootmap van de applicatie kunnen alle views gestoken worden, waarvan *index.html* als prioritair wordt bezien. Deze zal de GUI van de applicatie worden. Assets kunnen ook in de rootmap geplakt worden, zoals CSS bestanden en JavaScript bestanden. Het distribueren zelf kan op verschillende manieren gebeuren. Het kan eventueel gewoonweg in een executable worden gepackaged, maar normaal gezien wordt een NW.js applicatie gepackaged in een *.nw* bestand. Hiervoor moet de gebruiker dan nog wel de NW.js engine downloaden vooraleer de applicatie zelf gebruikt kan worden. Dit is minder wenselijk.

De migratie van de Electron app naar NW.js verliep enorm vlot; de code kon gewoonweg gekopieerd worden, inclusief de code van de view. Een screenshot is dus niet noodzakelijk aangezien de applicatie er hetzelfde uitziet. Deze kan nu wel gemakkelijker opgestart worden door gewoonweg de executable te openen. De form werkte en de buttons startte de publicatie van de data (de functie uitvoeren), of stopte ze (de functie stoppen). Nadelig is wel dat de grootte van de applicatie nu aanzienlijk steeg; NW.js heeft meer assets nodig om te werken dan Electron.

Nu de programmatie achter de applicatie uitgelegd is, kan worden overgegaan naar het volgende gedeelte.

### 4.3.3 Hardware

Aanvankelijk werd een Arduino Uno gebruikt als bron van data. Dit werd gedaan omdat de Giant Gecko nog niet helemaal gereed was, en de manier van communicatie (serieel) was hetzelfde bij de twee. Zo kon er gebruik gemaakt worden van dummy data om aan de ontwikkeling van de applicatie te beginnen.

#### 4.3.3.1 Arduino Uno



De Arduino Uno is een development board, met een zeer breed doelpubliek. De bedoeling ervan is mensen heel simpele prototypes doen ontwikkelen, en beginners wegwijs te maken in de wereld van embedded systems en algemeen/objectgeoriënteerd programmeren. De Arduino Uno die gebruikt werd heeft een ATmega328P met kloksnelheid 16 MHz en kan 5V leveren aan randmodules. Deze kunnen verbonden worden aan 14 digitale I/O pinnen (waarvan 6 PWM output pinnen om b.v. een LED te dimmen) en 6 analoge I/O pinnen.

De bron was een lichtsensor. Op een breadboard werd de lichtsensor geplaatst samen met 3 LED'jes, als volgt:

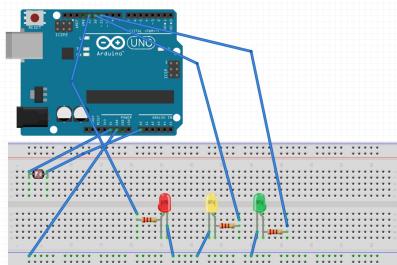
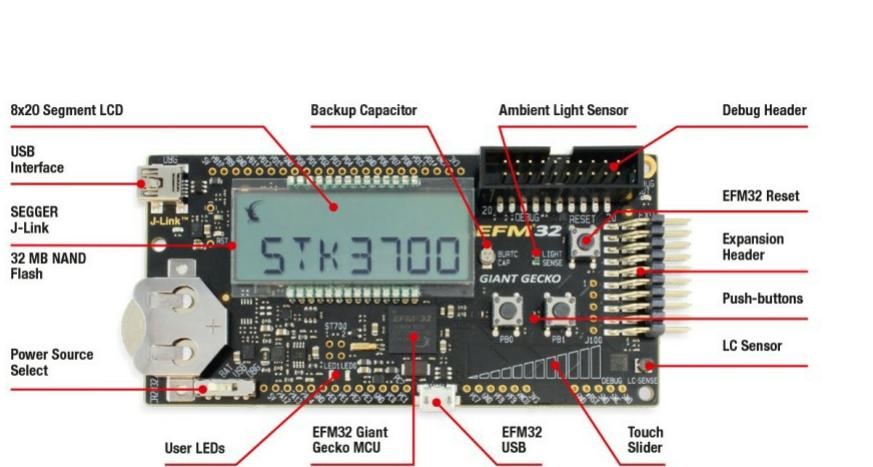


Fig. 18: Schema van Arduino-opstelling

#### 4.3.3.2 Migratie naar Gecko development board

Een Arduino gebruiken als databron was allesbehalve het einddoel en kon vervangen worden vanaf het moment dat de simulatie van de OCTA-gateway compleet was. Hiervoor werd een Gecko Giant development board gebruikt, met een DASH7 module ontworpen door de UAntwerpen zelf.

##### Gecko Giant



\*Fig. 18: De Giant Gecko development board, ontwikkeld door Silicon Labs.

De EFM32 Giant Gecko is een 32-bit microcontroller bedoeld om prototypes van toestellen te ontwikkelen. Het bezit tot 1024kB flash geheugen en 128 kB RAM geheugen. Met CPU snelheden tot 48 MHz wordt de Giant Gecko vooral gebruikt voor low-energy applicaties maar met hoge connectiviteit. Hij is uitgerust met een ARM Cortex-M3 CPU. Deze features zijn te vinden op <http://www.silabs.com/products/mcu/32-bit/efm32-giant-gecko/pages/efm32-giant-gecko.aspx>.

Op de Giant Gecko zit dus een DASH7 Alliance Protocol module, die zorgt voor de communicatie tussen de OCTA-mini in het bpost doosje en de gateway zelf. Hiervoor zullen shields bestaan als de gateway eenmaal in productie is. Ook voor LoRa en SIGFOX worden onder andere shields gemaakt.



Fig. 19: Giant Gecko met DASH7 module

De data wordt via de UART doorstuurd naar een USB-poort op de computer. Hier komt de data om de 10 seconden serieel binnen. De data wordt echter niet, zoals met de Arduino, voorgesteld in decimale vorm, maar in hexadecimale vorm.

De baud rate dient ingesteld te worden op 115200. De serial monitor wordt opgestart en de data vloeit binnen (LET OP: er moet voldoende connectie zijn of de pakketjes komen niet volledig binnen. Indien dit het geval is moet de connectie als eerste nagekeken worden):

```
gn npm
<Buffer 44 04 87 00 00 20 24 c4 c3 04 54 ef 97 b7 30 40 00 17 12 1b 97 43 00 00 22 44 1f 00 e3 00 e2 01 32 b4 e5 4c 42 f4 37 8d 40>
<Buffer 44 04 87 00 00 20 24 c4 c3 04 54 ef 97 b7 30 40 00 17 f5 16 97 43 00 00 22 44 1f 00 e2 00 c2 01 ff b5 e5 4c 42 33 38 8d 40>
<Buffer 44 04 5d 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 d6 12 97 43 00 00 22 44 1f 00 e2 00 c2 01 ec a8 e5 4c 42 05 38 8d 40>
<Buffer 44 04 2c 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 d6 12 97 43 00 00 22 44 1f 00 fb 00 c2 01 05 9d e5 4c 42 d1 37 8d 40>
<Buffer 44 04 7d 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 3e 14 97 43 5c 8f 20 44 1f 00 fa 00 c2 01 4f 93 e5 4c 42 79 37 8d 40>
<Buffer 44 04 9c 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 d6 12 97 43 29 5c 19 44 1f 00 e6 00 c2 01 6c 90 e5 4c 42 80 37 8d 40>
<Buffer 44 04 89 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 3e 14 97 43 29 5c 19 44 1f 00 fa 00 c2 01 e2 94 e5 4c 42 95 37 8d 40>
<Buffer 44 04 d8 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 93 0a 97 43 29 5c 19 44 1f 00 e7 00 c2 01 29 99 e5 4c 42 c6 37 8d 40>
<Buffer 44 04 1f 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 93 0a 97 43 03 0a d7 1f 44 1f 00 e7 00 c2 01 29 99 e5 4c 42 c6 37 8d 40>
<Buffer 00 c2 01 8b 9b e5 4c 42 c3 37 8d 40>
<Buffer 44 04 42 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 9a 0a 97 43 0a d7 1f 44 1f 00 d2 00 c2 01 76 92 e5 4c 42 8e 37 8d 40>
<Buffer 44 04 35 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 3a 09 97 43 0a d7 1f 44 1f 00 f5 00 c2 01 38 96 e5 4c 42 61 37 8d 40>
<Buffer 44 04 33 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 db 07 97 43 b3 1e 1f 44 1f 00 e6 00 c2 01 bd 9e e5 4c 42 87 37 8d 40>
<Buffer 44 04 9a 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 3a 09 97 43 14 ae 1d 44 1f 00 e6 00 c2 01 08 9d e5 4c 42 79 37 8d 40>
<Buffer 44 04 d5 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 3a 09 97 43 71 3d 1c 44 1f 00 e7 00 c2 01 f4 9e e5 4c 42 61 37 8d 40>
<Buffer 44 04 fe 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 9a 09 97 43 71 3d 1c 44 1f 00 d1 00 c2 01 1e a0 e5 4c 42 4f 37 8d 40>
<Buffer 44 04 29 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 9b 09 97 43 1f 85 1b 44 1f 00 d2 00 c2 01 95 ac e5 4c 42 b5 37 8d 40>
<Buffer 44 04 f9 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 59 0d 97 43 e1 7a 16 44 1f 00 cf 00 c2 01 a6 b3 e5 4c 42 09 38 8d 40>
<Buffer 44 04 15 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 59>
<Buffer 44 04 b0 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 7a 16 44 1f 00 e6 00 c2 01 7e b3 e5 4c>
<Buffer 0d 97 43 e1 7a 16 44 1f 00 e7 00 c2 01 be b6 e5 4c 42 63 38 8d 40>
<Buffer 44 04 b0 00 00 20 24 c4 c3 04 54 ef 97 b7 20 40 00 17 17 10 97 43 e1 7a 16 44 1f 00 e6 00 c2 01 7e b3 e5 4c>
<Buffer 44 04 63 38 8d 40>
<Buffer 44 04 33 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 17 10 97 43 e1 7a 16 44 1f 00 e4 00 c2 01 7c b0 e5 4c 42 21 38 8d 40>
<Buffer 44 04 90 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 f9 0b 97 43 e1 7a 16 44 1f 00 fa 00 c2 01 6f ab e5 4c 42 bc 37 8d 40>
<Buffer 44 04 06 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 3a 09 97 43 8f c2 15 44 1f 00 fa 00 c2 01 a3 ab e5 4c 42 ad 37 8d 40>
<Buffer 44 04 40 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 7d 06 97 43 8f c2 15 44 1f 00 fa 00 c2 01 cd ab e5 4c 42 8d 37 8d 40>
<Buffer 44 04 f4 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 9a 09 97 43 8f c2 15 44 1f 00 fa 00 c2 01 2c ad e5 4c 42 84 37 8d 40>
<Buffer 44 04 05 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 9a 09 97 43 8f c2 15 44 1f 00 fa 00 c2 01 c6 ac e5 4c 42 b5 37 8d 40>
<Buffer 44 04 87 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 db 07 97 43 e1 7a 16 44 1f 00 fa 00 c2 01 4d b4 e5 4c 42 f7 37 8d 40>
<Buffer 44 04 33 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 bc 03 97 43 e1>
<Buffer 7a 16 44 1f 00 fb 00 c2 01 2b e5 4c 42 33 38 8d 40>
<Buffer 44 04 e2 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 1c 09 97 43 3d 0a 15 44 1f 00 e4 00 c2 01 61 c1 e5 4c 42 40 38 8d 40>
<Buffer 44 04 1c 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 db 07 97 43 3d 0a 15 44 1f 00 d1 00 c2 01 0f bf e5 4c 42 33 38 8d 40>
<Buffer 44 04 a4 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 f9 0b 97 43 8f c2 15 44 1f 00 d5 00 c2 01 3f be e5 4c 42 4e 38 8d 40>
<Buffer 44 04 62 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 9a 0a 97 43 3d 0a 15 44 1f 00 d2 00 c2 01 d2 bf e5 4c 42 67 38 8d 40>
<Buffer 44 04 2b 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 7b 06 97 43 3d 0a 15 44 1f 00 e4 00 c2 01>
<Buffer c1 b5 e5 4c 42 47 38 8d 40>
```

Fig. 20: Screenshot van de data die binnenvloeit op de serial monitor

Aangezien de data in hexadecimale vorm (en in buffers) binnenkomt, moet hij dus eerst geparsed worden, ofwel omgezet naar decimaal. Hiervoor werd een parser gegeven in python. Hier alvast een snippet:

```

sensor_data_bytes = [
0xe6, 0x16, 0x95, 0x43,      # temperature
0x9a, 0x99, 0x6d, 0x44,      # light
0x25, 0x00,                  # RH
0x9c, 0x16,                  # CO2
0x00, 0x00,                  # VOC
0x8f,                         # sum
198, 229, 76, 66,            # lat
128, 55, 141, 64             # lon
]

```

Vooraleer we parsen ontleden we een volledig pakketje. Op de screenshot kan gezien worden dat de langste buffer 82 karakters lang is. In hexadecimaal wordt 1 byte voorgesteld met 2 karakters, dus kunnen we met zekerheid zeggen dat de grootte van een volledige string 41 bytes is. Uit de snippet hierboven kan afgeleid worden hoe groot elk element in de string is, en hoe hij algemeen wordt opgedeeld. Het eerste deel stelt de temperatuur voor, dan de hoeveelheid licht, dan de RH-waarde, dan de hoeveelheid CO<sub>2</sub>, dan de VOC, dan een checksum en dan de coordinaten (eerst latitude dan longitude). Respectievelijk zijn deze data 4, 4, 2, 2, 2, 1, en 8 (2 keer 4) bytes groot. Hieronder wordt een voorbeeld ontleed om een duidelijker beeld te geven.

```

<Buffer 44 04 87 00 00 20 24 e4 c3 04 54 ef 97 b7 20 40 00 17 db 07 97 43 e1 7a 16 44 1f 00 fa 00 c2
01 4d b4 e5 4c 42 f7 37 8d 40>

```

Achteraan beginnend: 4 bytes stelt de longitude voor (f7 37 8d 40), dan 4 bytes latitude (b4 e5 4c 42), dan de checksum (4d), dan de VOC (c2 01), dan de CO<sub>2</sub> (fa 00), RH (1f 00), lichthoeveelheid (e1 7a 16 44), en temperatuur (db 07 97 43). Het device ID nummer staat ervoor (24 e4 c3 04 54 ef 97 b7). Wanneer de data zo wordt omgezet naar decimaal, zijn de resultaten niet correct.

Bijvoorbeeld: wanneer f7378d40 (zogezegd longitude) wordt omgezet naar decimaal wordt de waarde 4147612992 verkregen. Aardrijkskundig is dit uiteraard geen mogelijke waarde voor de longitude van een locatie. De reden hiervoor is dat wanneer een waarde wordt omgezet naar decimaal, standaard een UINT32 vorm wordt gebruikt. De coordinaten zijn echter weggeschreven in een Little Endian Float vorm. Dit kan gemakkelijk geanalyseerd worden aan de hand van een online tool van Scadacore: <http://www.scadacore.com/field-applications/programming-calculators/online-hex-converter/>. Wanneer een Little Endian Float wordt gebruikt, wordt de verkregen waarde 4.41308165. De latitude (b4e54c42) wordt dan 51.22432. Volgens google maps komen deze coordinaten uit op de campus Paardenmarkt. Hieronder ter verduidelijking een volledig schema van een binnenkomend DASH7 pakket:



Fig. 21: Schematische voorstelling van een binnenkomend DASH7 pakket

De data zal *raw* in de database terecht komen. De data wordt pas geparsed als het wordt getoond in de frontend. Dit zodat de parser zelf nog kan worden aangepast moet er iets fout zijn. Indien er een foutje in de parser zit en de data wordt zo in de database weggeschreven, is de data inconsistent doorheen het bestand. Er wordt wel gefilterd, omdat met onvolledige pakketjes niet veel gedaan kan worden. Dit is niet echt een probleem aangezien er toch voldoende data geproduceerd wordt. De buffers die binnenkomen moeten wel geparsed worden naar 1 samenhangende string. Dit gebeurt door de `toString()` functie uit te voeren van JavaScript, met als parameter *hex*. Zo weet de applicatie om welke vorm het gaat. Op de volgende manier wordt er gefilterd:

```
var filteredstring;
if (datastream.length == 82)
{
    filteredstring = datastream;
}
console.log(filteredstring);
if (filteredstring != null)
{
    collection.update(
    ...
}
```

De onvolledige pakketjes worden dan in de console weergegeven als *undefined*, en aangezien ze geen waarde hebben (*null*) worden ze gewoonweg niet weggeschreven in de database. Hieronder een screenshot van ongeparsinge data via Postman:

```

11  {
12      "id": "24e4c30454ef97b7",
13      "when": "2016-05-12T12:58:51.543Z",
14      "value": "44040500002024e4c30454ef97b720400017b80e9743b81e1f441f002401c201e0b9e54c42fb378d40"
15  },
16  {
17      "id": "24e4c30454ef97b7",
18      "when": "2016-05-12T12:58:11.409Z",
19      "value": "4404dc00002024e4c30454ef97b720400017b80e9743b81e1f441f00e700c201a2bfe54c42c6378d40"
20  },
21  {
22      "id": "24e4c30454ef97b7",
23      "when": "2016-05-12T12:58:21.345Z",
24      "value": "4404c000002024e4c30454ef97b720400017b80e9743b81e1f441f00fb00c201b6c3e54c42c6378d40"
25  },
26  {
27      "id": "24e4c30454ef97b7",
28      "when": "2016-05-12T12:58:31.278Z",
29      "value": "44044b00002024e4c30454ef97b720400017d6129743b81e1f441f000d01c201ebcf54c42df378d40"
30  },
31  {
32      "id": "24e4c30454ef97b7",
33      "when": "2016-05-12T12:58:51.148Z",
34      "value": "44040400002024e4c30454ef97b72040001795159743b81e1f441f00fb00c2019ac9e54c42f0378d40"
35  }

```

Fig. 22: Screenshot van Postman die de volledige database ophaalt

Zoals te zien in de afbeelding hierboven wordt ook de UID uit de data gehaald en in een apart veld genaamd deviceid weggeschreven. Dit is nodig zodat elk datapunt een bron heeft en de juiste gebruiker uiteindelijk enkel zijn eigen data te zien krijgt.

De data wordt in de controller van de front-end opgehaald en meteen individueel geparsed. Er wordt bijvoorbeeld voor de coördinaten gekeken naar de 8 laatste bytes; vervolgens worden ze omgezet (de endianness wordt eerst omgekeerd door een array te maken van alle bytes en de volgorde te veranderen: 0xABCD wordt 0xDCBA) naar een \$scope variabele die dan alle decimale waarden bevat van de coördinaten. In de frontend worden ze weergegeven aan de hand van dezelfde grafieken die op voorhand besproken werden.

```
for(var i = 0; i >= 0; i++)
{
    var temp, light, rh, co2, voc, lat, long;
    var temp = parseInt("0x" + $scope.lightdata.data[0].events[i].value.substring(36, 44));
    $scope.temparray.push(temp);
    var light = parseInt("0x" + $scope.lightdata.data[0].events[i].value.substring(44, 52));
    $scope.lightarray.push(light);
    var rh = parseInt("0x" + $scope.lightdata.data[0].events[i].value.substring(52, 56));
    $scope.rharray.push(rh);
    var co2 = parseInt("0x" + $scope.lightdata.data[0].events[i].value.substring(56, 60));
    $scope.co2array.push(co2);
    var voc = parseInt("0x" + $scope.lightdata.data[0].events[i].value.substring(60, 64));
    $scope.vocarray.push(voc);
}
```

De data wordt individueel geparsed aan de hand van de parseInt functie. Er wordt telkens *0x* voor gezet om aan te duiden dat het om een hexadecimale waarde gaat. Een andere manier om dit aan te duiden is een extra parameter in de parseInt functie om de radix aan te duiden (bij hexadecimaal 16). Ze worden individueel in arrays gezet om getoond te worden in de grafieken. De *substring*-functie duidt de locatie van de waardes aan binnen de complete string. De coördinaten worden apart geparsed omdat deze een andere endianness bezitten, en ze op een andere manier getoond moeten worden binnen een Google Map.

## 4.4 Gebruikte tools

Hieronder zullen de verschillende tools opgenoemd en uitgelegd worden die gebruikt werden om deze applicatie te verwezenlijken. Hier moet bij gezegd worden dat elke tool zijn voor- en nadelen heeft. Deze worden telkens toegelicht zodat een persoonlijke voorkeur gemaakt kan worden.

### 4.4.1 Brackets

Brackets is een open-source IDE vooral bedoeld voor web development. Het programma zelf is ook geschreven in HTML, CSS en JavaScript. Het is ontwikkeld door Adobe als een code editor gedreven door de community, en is cross-platform beschikbaar voor Mac-, Windows- en Linux-systemen.

Het voordeel aan Brackets is de customization. Er kunnen thema's toegevoegd worden en de juiste syntax-highlighting kan gebruikt worden die best bij jou past. Het is echter vooral ook volledig gratis voor gebruik.

Nadeel is dat er niet al te veel mogelijke plugins voor zijn. Het programma is wel open-source, dus moesten er noodzakelijke dingen aan toegevoegd worden kan dit altijd door de gebruiker zelf worden gedaan. Er zijn echter betere mogelijkheden.

### 4.4.2 Atom

Atom is een beter alternatief. Het is eveneens een open-source text editor, maar met veel meer mogelijkheden dan Brackets. Relatief recent is Atom ontwikkeld door het team van GitHub zelf, dus ze weten wat nodig is in een IDE. Hun slogan voor Atom is: *Hackable to the core*. Customizations kunnen dus worden toegevoegd naar keuze, maar het is van het begin af aan toegankelijk zonder zelfs een configuratiebestand aan te raken. Zo zijn er ontzettend veel plugins te gebruiken voor alle mogelijke programmeertalen en doeleinden. Atom kan dienen voor webtechnologieën (het is zelf geschreven met het Electron framework in JavaScript), maar ook als Markdown editor en zo verder.

Een van de meest populaire (snelste) text editors is Sublime. Hiervoor kan een package manager geïnstalleerd worden om plugins te installeren maar dit is minder gebruiksvriendelijk om op te stellen. Atom heeft een geïntegreerde package manager met enorm veel keuzes, die elk op zich nog aparte configuraties bieden. Net iets trager dan Sublime, maar ontzettend gebruiksvriendelijk is Atom zeer sterk aan te raden als algemene text editor.

### 4.4.3 Node.JS (Command Prompt)

Node.js is een software omgeving waarop applicaties geschreven in JavaScript ontwikkeld kunnen worden. De omgeving biedt een eigen runtime waarop de applicaties draaien, en biedt dan server-side services aan voor iedereen die er gebruik van maakt. Het grote voordeel van Node.js is dat ze cross-platform zijn, en dus zowel op Mac, Windows als Linux kunnen draaien.

In de Node.js runtime zit een reeks command-line tools, die gebruikt worden om de ontwikkelde applicaties te starten, te stoppen, te configureren en te debuggen.

## 2.4.4 MongoDB

De MongoDB runtime is eveneens een reeks command-line tools die zorgen voor het hosten van een eigen database. Als de database op een third party service wordt gehost moet er niets op de computer zelf geïnstalleerd worden. Indien het wel nodig is, dienen de tools om databases aan te maken, te configureren of de toegankelijkheid (rechten) van databases of collecties aan te passen.

## 4.4.5 Google Chrome

Om het project te debuggen werd vooral Google Chrome gebruikt. Dit omdat het de nuttigste en overzichtelijkste debug tools bevat met een duidelijke console.

# 4.5 Third-party dashboard services

Het laatste dat wordt aangekaart is een set van externe dashboards die gemakkelijk en gratis gebruikt kunnen worden. Hier zit authenticatie reeds in verwerkt maar de personalisatie laat soms wat te wensen over.

## 4.5.1 Dashing

Dashing is een framework om kleurvolle dashboards aan te maken. Het is gebouwd op Sinatra, een web-applicatie library in Ruby dat een alternatief biedt voor bijvoorbeeld Ruby on Rails (maar het gebruikt niet het MVC-model van RoR). Features van Dashing:

- Maak gebruik van widgets die vooraf gemaakt zijn of schrijf ze zelf in scss (sassy CSS), html en coffeescript (taal die in JavaScript compileert, maar het overzichtelijker en korter maakt)
- Dashing heeft een eigen API die gebruikt kan worden om data naar het dashboard te pushen
- Charts kunnen versleept worden

Het wordt vooral gebruikt voor Ruby projecten. Het is te vinden op <http://dashing.io/>.

## 4.5.2 Freeboard.io

Freeboard is een open source real-time dashboard platform voor allerlei Internet of Things projecten. Het gebruik ervan is volledig gratis, en in het platform zit alles, inclusief authenticatie, reeds ingebouwd. Wanneer er wordt ingelogd kan elke gebruiker zijn eigen dashboard personaliseren en widgets toevoegen. Aan elke widget wordt een databron toegevoegd, die deze data automatisch ophaalt. Een voorbeeld van zulk een databron is JSON. Op te merken indien een externe JSON wordt geladen in freeboard: de connectie moet veilig zijn en dus moet het gehost worden op een *https* server. Hiervoor moet poort 443 open staan op de server.

- Authenticatie ingebouwd

- Widgets gebruiksvriendelijk, enkel databron moet toegevoegd worden

Het werkt goed samen met dweet.io. Dit is een service die zorgt voor de opslag en routing van binnengkomende datapoints. Hier hangt echter een maandelijkse kost aan vast als ze opgeslagen moeten worden. Het is te vinden op <http://freeboard.io/>.

### 4.5.3 Blynk

Blynk is een mobile-only IoT dashboard dat gebruikt kan worden met vrijwel alle IoT projecten. Er moet een library worden geïnstalleerd in de code van het development board die de nodige pins verbindt met het dashboard. De app wordt dan gedownload op de smartphone en met token-based authentication wordt er verbonden met het project. De data verschijnt dan automatisch in het dashboard. Het is te vinden op de App Store en Google Play. De website van Blynk is <http://www.blynk.cc/>.

- Mobile-only
- Enorm gebruiksvriendelijk

# 5. Conclusie en samenvatting

## 5.1 Conclusie & resultaat

In dit hoofdstuk zal de front-end van de applicatie in zijn geheel, en elke functie ervan getoond worden. Elke pagina zal uitgelegd worden en wat erop staat.

De applicatie is uiteindelijk verwezenlijkt kunnen worden met alle features die oorspronkelijk aanwezig moesten zijn. Ze zal gehost worden op Heroku zodat ze toegankelijk is voor iedereen. Een account kan gemakkelijk aangemaakt worden. Om het platform zelf te gebruiken is wel een geldige OCTA-ID nodig, aangezien er anders geen data binnen zal komen. Eerst zal de applicatie voorgesteld worden hoe ze er nu uit ziet. Elke pagina wordt besproken om zo het nut ervan duidelijk te maken. Daarna zal gekeken worden hoe de applicatie eventueel verbeterd zou kunnen worden.

### 5.1.1 Desktop applicatie

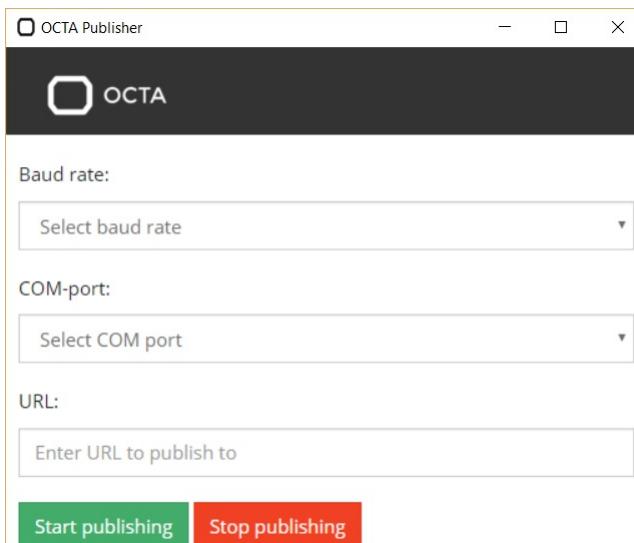


Fig. 4: Desktop applicatie

De desktop-applicatie wordt geïnstalleerd aan de hand van een setup. De executable wordt dan geopend om bovenstaand venster te krijgen (Fig. 23). Hier wordt de COM-poort geselecteerd waar de gateway op is aangesloten, en de baud rate ervan (standaard 115200). Eventueel kan een URL ingegeven worden indien er een third-party dashboard gebruikt wordt in plaats van de ontwikkelde Angular applicatie. Op de groene button kan geklikt worden om de publishing-functie uit te voeren, en wanneer het publishen gestopt moet worden kan op de rode button geklikt worden. Indien de applicatie begint met publishen, zal de data doorgestuurd worden naar de MQTT broker. Deze houdt de data bij op een topic. De backend subscribet op dit topic en zet de ontvangen data meteen in een database. De database wordt doorgegeven aan de front-end, die nu uitgelegd zal worden:

### 5.1.2 Web-based applicatie

De index ziet er zo uit:

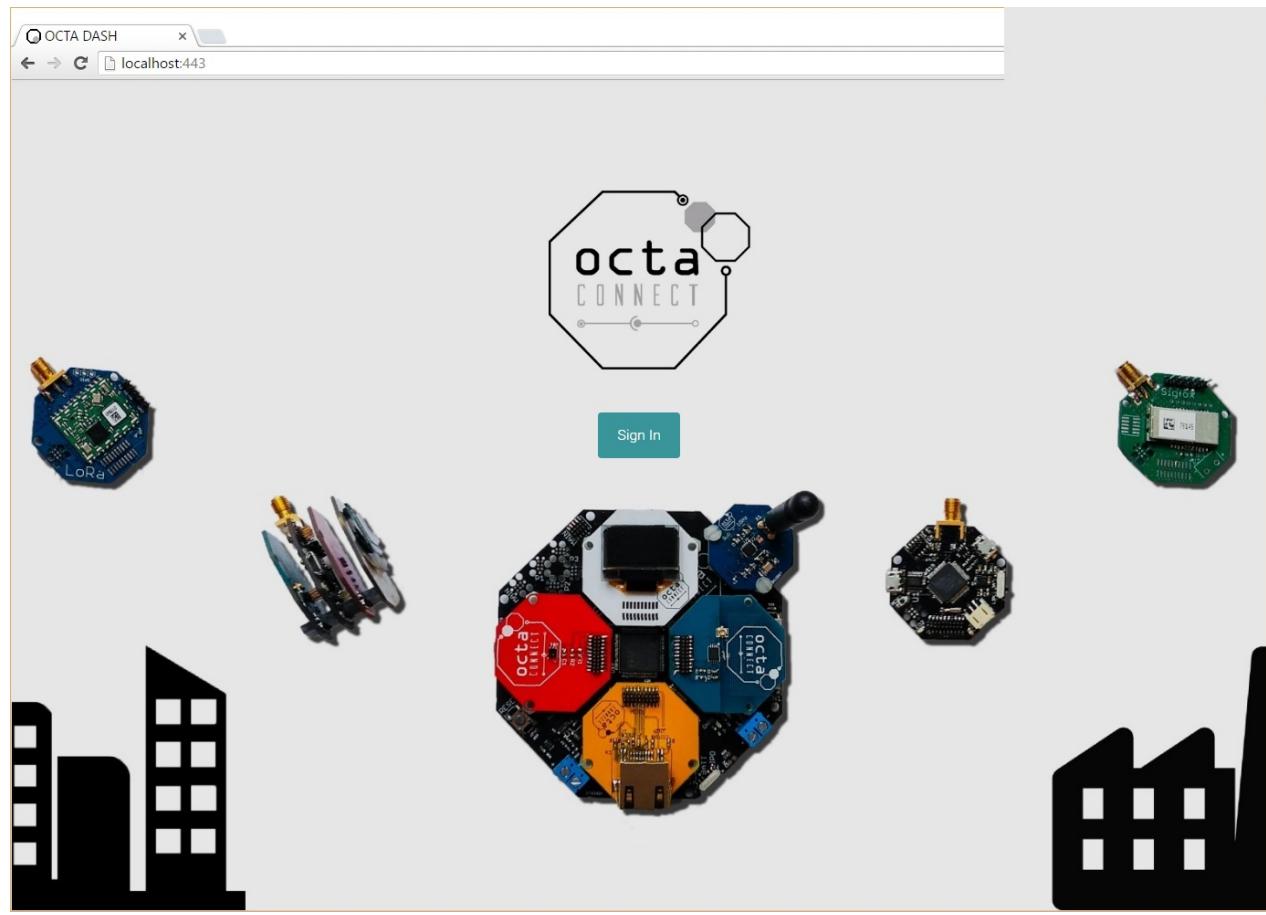


Fig. 24: Splash screen

Hier kan ingelogd worden door de modal te openen. Wanneer de modal geopend wordt ziet de login er zo uit:

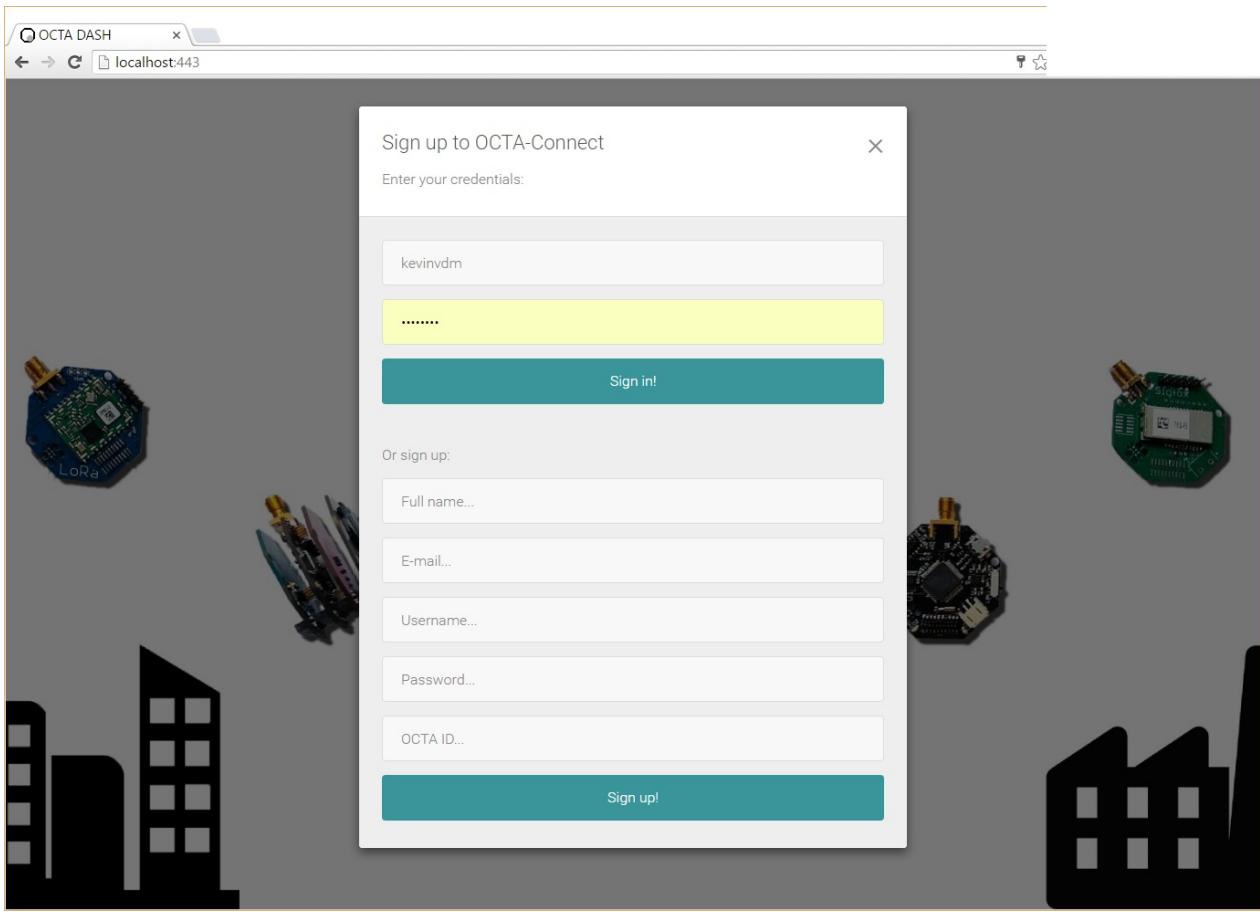


Fig. 25: Login modal

Hier kan een account aangemaakt worden en ingelogd worden. Als er een fout is met de login (verkeerde gebruikersnaam of paswoord) of de registratie (gebruikersnaam bestaat reeds), wordt de index opnieuw geladen. Als de login succesvol is wordt er naar het dashboard verwezen:

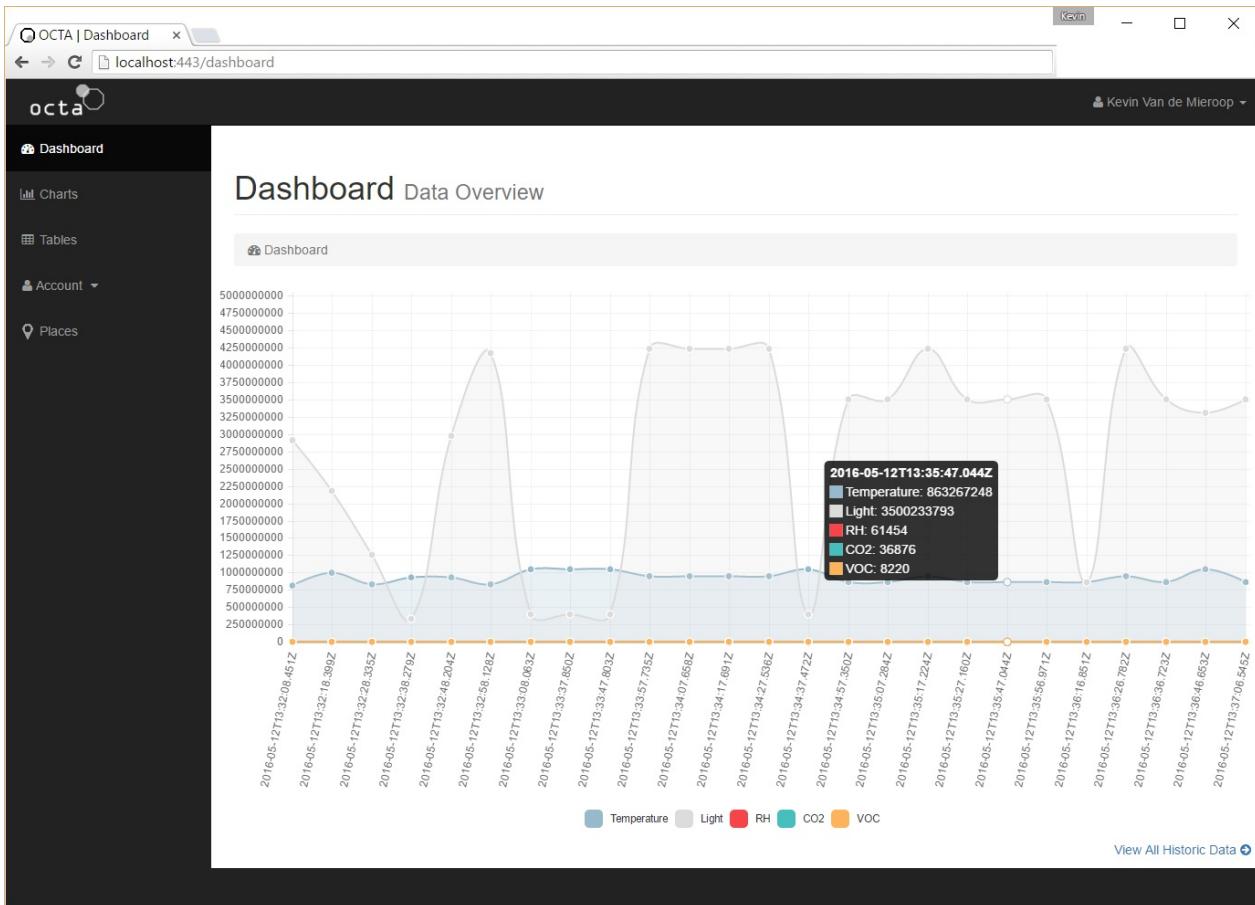


Fig. 26: Dashboard

Hierin staan gewone grafieken. Links staat een sidebar/menu, die zichzelf inklapt als er een mobiel scherm gedetecteerd wordt:

The screenshot compares the OCTA menu on a desktop (left) and mobile (right) screen. Both versions have a similar structure: a sidebar with "octa" logo, "Dashboard", "Charts", "Tables", "Account" (with dropdown), and "Places". The mobile version shows a collapsed sidebar with three horizontal dots. The main content area on both sides shows the same "Tables" view, which includes a table of data and a scrollable list below it.

Temperature	Light	RH	CO2	VOC	Timestamp
997517696	2179068993	61440	61468	8220	2016-05-12T12:57:31.694Z
997517696	2179068993	61442	16412	8222	2016-05-12T12:57:51.543Z
997517696	2179068993	61454	28684	8218	2016-05-12T12:58:11.409Z
997517696	2179068993	61455	45068	8219	2016-05-12T12:58:21.345Z
997468513	2179068993	61440	53276	8222	2016-05-12T12:58:31.278Z
997480785	2179068993	61455	45068	8217	2016-05-12T12:58:51.148Z
997505329	2179068993	61455	40972	8219	2016-05-

Fig. 27: Menu. Links de desktopversie, rechts de mobiele versie

Door op charts te klikken verschijnt een grafiek met alle laatste 25 datapunten. Deze worden allemaal geladen door angular-chart.js, en zien er zo uit:

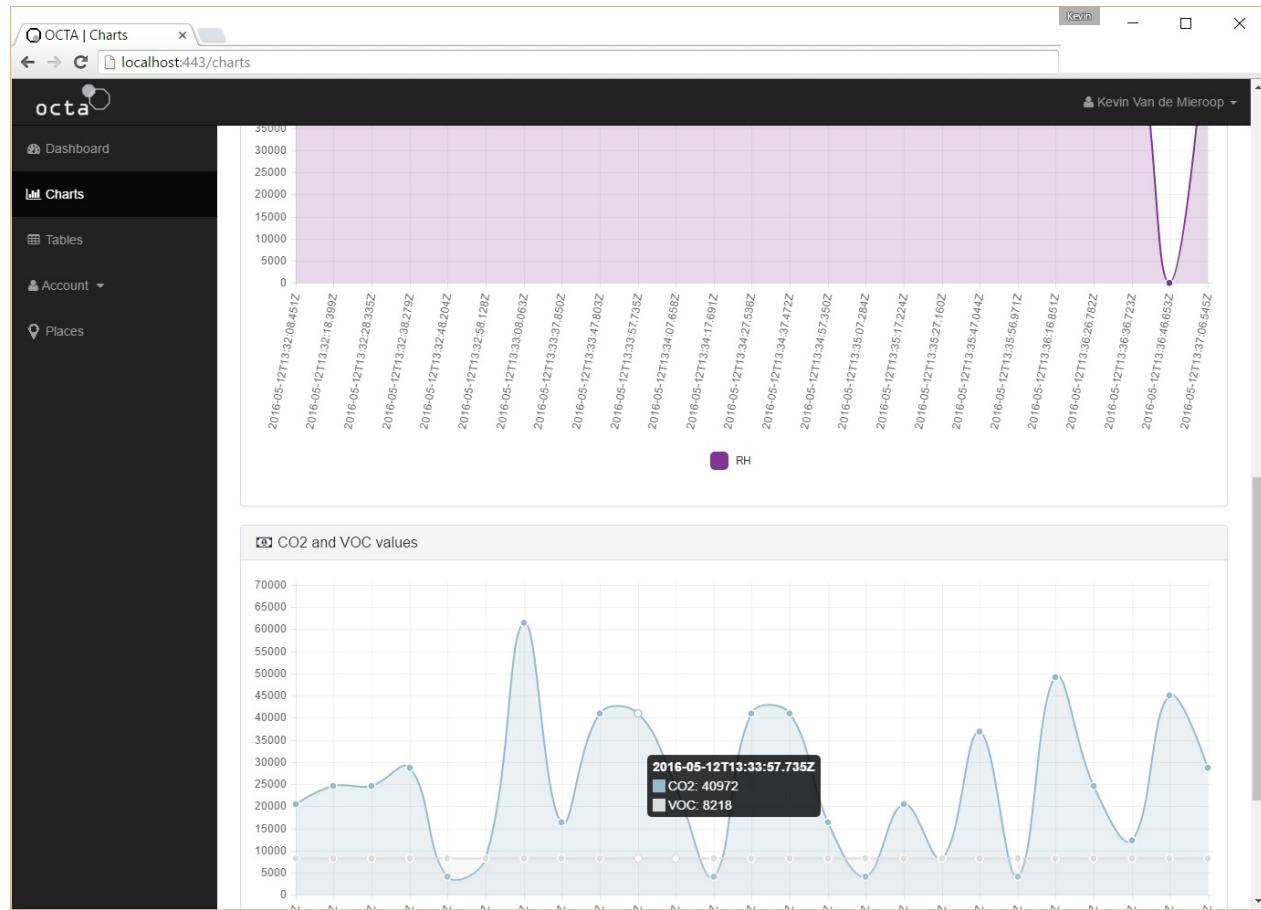


Fig. 28: Charts

In de link *tables* zit volgende pagina:

Temperature	Light	RH	CO2	VOC	Timestamp
997517696	2179068993	61440	61468	8220	2016-05-12T12:57:31.694Z
997517696	2179068993	61442	16412	8222	2016-05-12T12:57:51.543Z
997517696	2179068993	61454	28684	8218	2016-05-12T12:58:11.409Z
997517696	2179068993	61455	45068	8219	2016-05-12T12:58:21.345Z
997468513	2179068993	61440	53276	8222	2016-05-12T12:58:31.278Z
997480785	2179068993	61455	45068	8217	2016-05-12T12:58:51.148Z
997505329	2179068993	61455	40972	8209	2016-05-12T12:59:11.036Z
997513505	2179068993	61440	61468	8223	2016-05-12T12:59:20.955Z
997509409	2179068993	61441	12316	8217	2016-05-12T12:59:30.878Z
913627425	1717691457	61455	57356	8221	2016-05-12T12:59:40.807Z
913627425	1717691457	61442	12316	8223	2016-05-12T12:59:50.741Z
829733169	1256313921	61455	40972	8208	2016-05-12T13:00:00.682Z
947173681	1588687937	61455	4108	8218	2016-05-12T13:00:10.609Z
1047841057	396452929	61441	28	8209	2016-05-12T13:00:20.532Z
947149137	1588687937	57344	61468	8212	2016-05-12T13:00:30.461Z
1047812433	396452929	61455	57356	8209	2016-05-12T13:00:40.397Z

Fig. 29: Tabellen

Hierin wordt een lijst opgeroepen met alle individuele datapunten in een tabel en de bijhorende timestamp. Er is nog een pagina genaamd *places*, waarin de coordinaten in een Google Map worden geladen. Hiervoor werd de Google Maps API gebruikt, en worden de laatste 5 coordinaten in geladen als markers. Tussen de markers wordt een *traceroute* getrokken om de route aan te duiden die de gps-module heeft gevolgd:

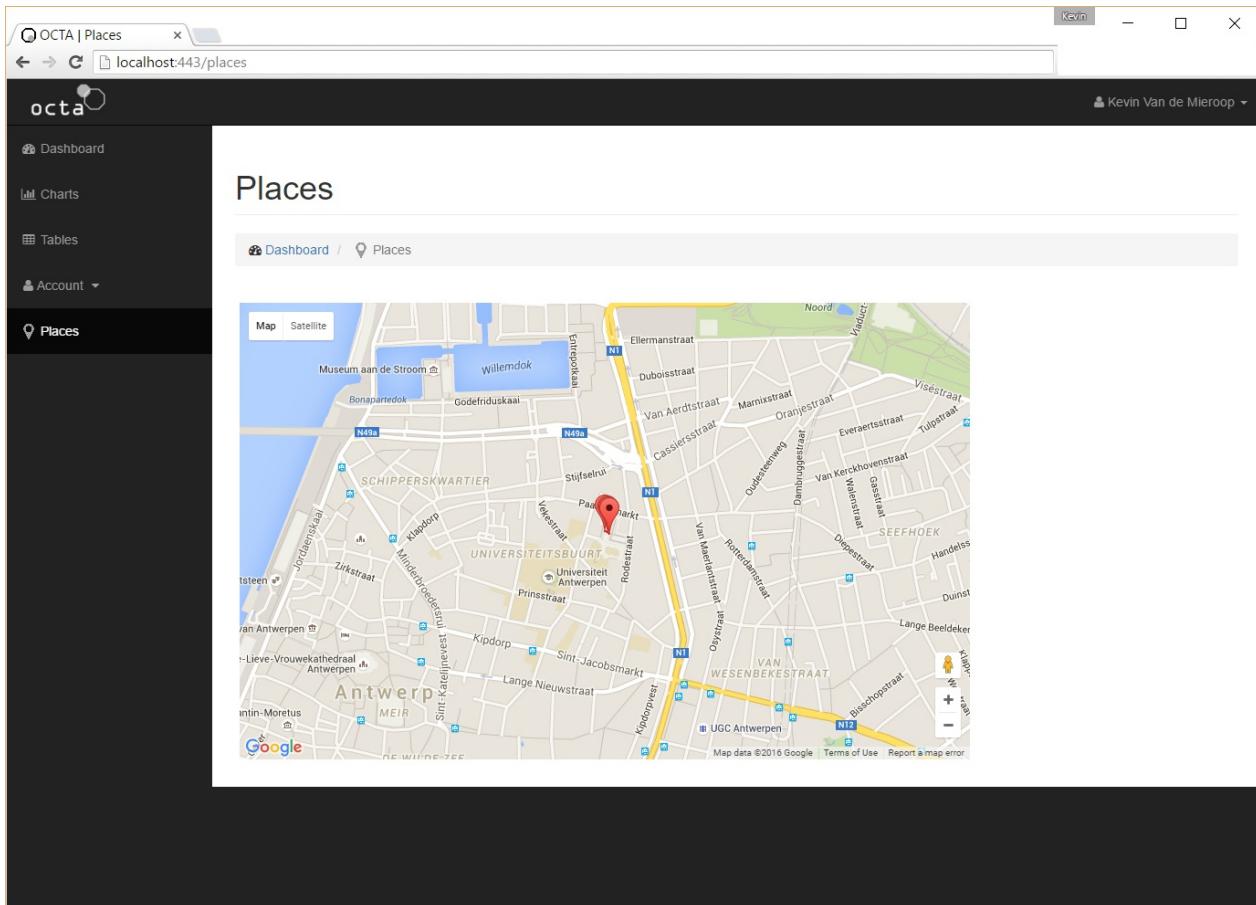


Fig. 30: Places

Op de *Account* pagina kan de gebruiker uitloggen. Hier komen in de toekomst nog meer opties, zo om bijvoorbeeld de parser in te stellen zodat de pakketjes die binnenkomen persoonlijk kunnen worden. Momenteel is het een statische parser. Ook de mogelijkheid om meer OCTA-Gateways toe te voegen zal hier komen.

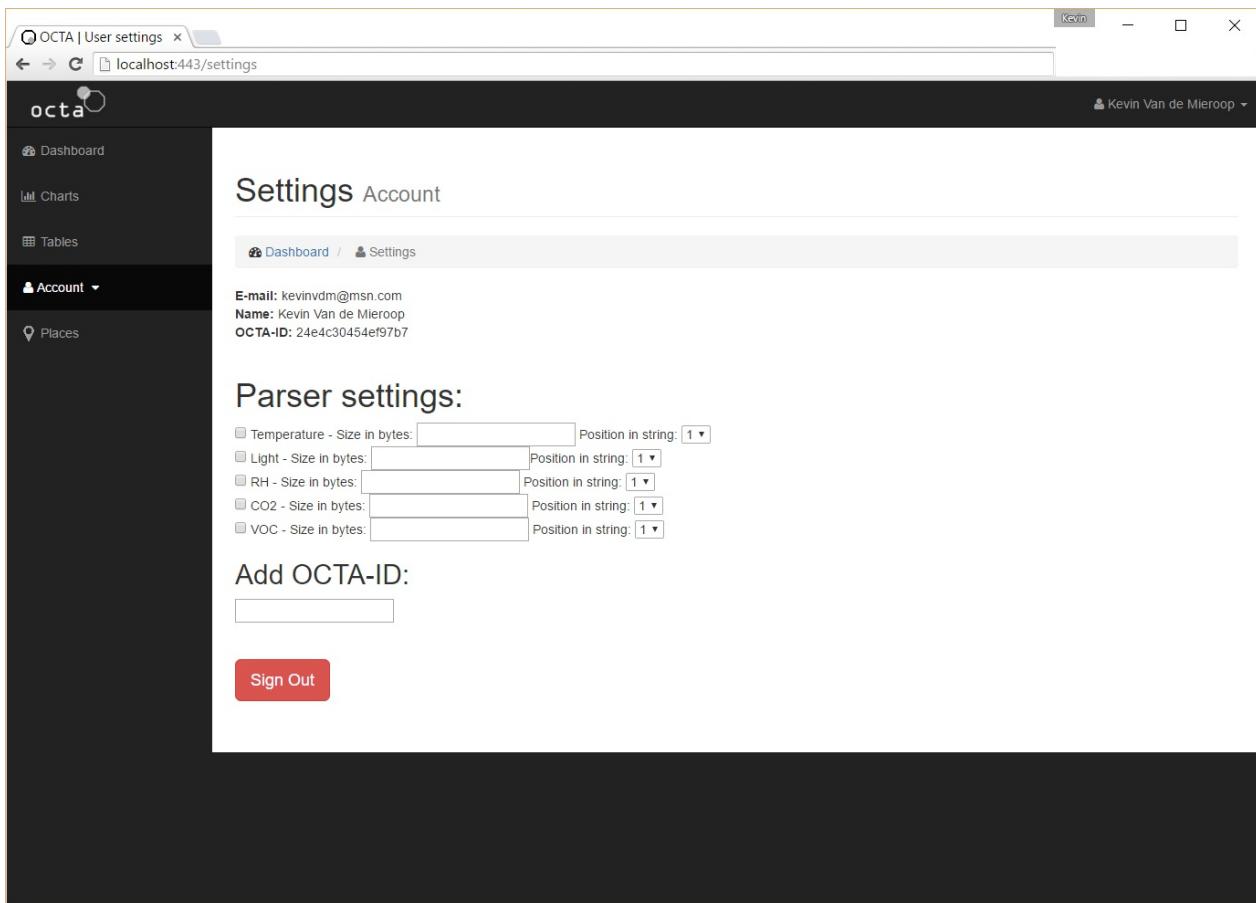


Fig. 31: Settings

Rechtsboven staat op alle pagina's een welkomstbericht en de mogelijkheid om gemakkelijk uit te loggen op elke pagina.

### 5.1.3 Hoe zou het verbeterd kunnen worden?

- Een dynamische parser zou geïmplementeerd moeten worden. Dit wil zeggen dat de pakketjes in de toekomst er anders zouden kunnen uitzien. De parser is momenteel statisch; er wordt gekeken naar de positie van alle karakters in de binnenkomende string en zet ze dan individueel in een variabele die getoond zal worden. De parser zou dan ingesteld kunnen worden op de account pagina van de ingelogde gebruiker.
- Er zou ondersteuning moeten komen voor de API's van het LoRa en het SigFox netwerk. Hiervoor wordt momenteel een individuele backend gebruikt waarop alle data binnenkomt maar zou even goed geïmplementeerd kunnen worden in deze applicatie.
- De parser staat nog niet volledig op punt. De endianness van de data wordt correct omgekeerd van groot naar klein, de data wordt eerst naar integer omgezet om dan naar floating point te worden geparsed, maar de data wordt nog steeds niet correct getoond. De onderlinge verhoudingen zijn echter correct dus de grafieken en tabellen doen wat ze moeten doen, en verschillen kunnen gemakkelijk opgespoord worden.

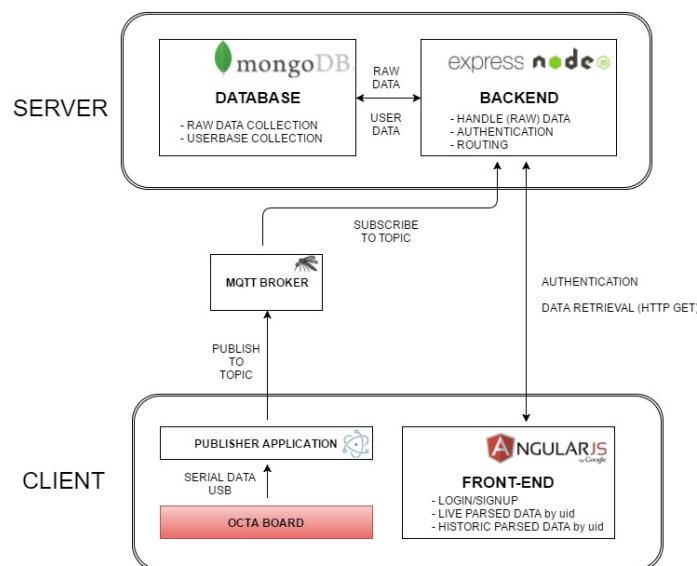
## 5.2 Samenvatting

Deze scriptie draait rond het concept Internet of Things. Rond dit idee werd OCTA-Connect geboren, een development board ontwikkeld door enkele onderzoekers aan de Universiteit Antwerpen, waarbij verschillende sensoren data verzamelen en doorsturen naar een gateway. Voor dit platform werd een controle-applicatie of monitoring applicatie geschreven.

De OCTA-Connect Gateway wordt aangesloten op een PC, die de data doorstuurt naar de publisher application over USB. De publisher application is een desktop-applicatie ontwikkeld met Node.JS op de Electron runtime, en stuurt de data rechtstreeks door over MQTT naar de broker. Deze broker behandelt de data en zal ze verdelen onder alle subscribers. De subscriber hier is de volledige backend, die in JavaScript geschreven is op het Node.JS framework. De backend slaat de data op in een Mongo database.

Een gebruiker maakt een account aan op de AngularJS-based front-end. De authenticatie hiervan wordt eveneens behandeld door de backend, die de gebruikersinformatie opslaat in dezelfde Mongo database (weliswaar in een andere collectie). Als de authenticatie succesvol is, wordt alle data van deze gebruiker doorgegeven en geladen in de front-end. De data wordt dan overzichtelijk getoond in de vorm van grafieken.

Hieronder nog eens de uiteindelijke architectuur van de gehele applicatie:



# 6. Bibliografie

## 6.1 Gevolgde tutorials

- Codecademy - Python (<http://www.codecademy.com>)
- Scotch.io - Build RESTful API (<https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4>)
- Jackal of JavaScript - Getting started with MQTT (<http://thejackalofjavascript.com/getting-started-mqtt/>)
- Tuts Plus - Authenticating Node.JS Applications with Passport  
(<http://code.tutsplus.com/tutorials/authenticating-nodejs-applications-with-passport--cms-21619>)

## 6.2 Geraadpleegde bronnen

### 6.2.1 Online bronnen

- Google (<http://www.google.com>)
- Wikipedia (<http://www.wikipedia.org>)
- StackOverflow (<http://www.stackoverflow.com>)
- iMinds - City of Things (<https://www.iminds.be/en/succeed-with-digital-research/city-of-things>)
- MongoDB Manual (<https://docs.mongodb.org/manual>)
- Google Maps API (<https://developers.google.com/maps/documentation/javascript/tutorial>)
- DASH7 Backend readme, Christophe VG

### 6.2.2 Gepubliceerde papers

- ERGEERTS Glenn, *Dash7 Alliance Protocol in Monitoring Applications*, November 2015
- WEYN Maarten, *DASH7 Alliance Protocol 1.0: Low-Power, Mid-Range Sensor and Actuator Communication*, Oktober 2015

## 7. Glossary

**6LowPAN:** (IPv6 over Low Power Personal Area Network) Elke node krijgt een IPv6 adres, er wordt via open standaarden gecommuniceerd over het netwerk.

**ANT+:** Gebruikt ook de 2.4GHz band. ANT+ heeft een reikwijdte van om en bij de 50m, een overdrachtsnelheid van ongeveer 1Mbit/s, en is bedoeld om weinig data op te sturen (enkele bytes, zoals text messages). Het nadeel ten opzichte van BLE is dat ANT+ minder support heeft onder moderne devices en dus vaak een dongle nodig is.

**API:** Application Programming Interface. Een API is een groep functies en objecten die worden gebruikt om een applicatie op te bouwen die communiceert met een ander systeem. Ze komen vaak in de vorm van *libraries*. Een voorbeeld hiervan is de Google Maps API, die ook in dit project gebruikt werd.

**Baud rate:** de snelheid waarop data binnenkomt in een systeem. Het wijst op ofwel het aantal signaalwisselingen of symbolen per seconde op een kanaal. Het is niet gelijk aan het aantal bits per seconde, al wordt dit vaak gedacht.

**Bluetooth Low Energy:** Gebruikt evenals de band van 2.4GHz, en bezit een gelijkaardige reikwijdte/overdrachtsnelheid van 50m/1Mbit/s. BLE transfereert maximum pakketjes van 20 bytes. Het voordeel is dat er veel support voor is. De grote bedrijven staan er allemaal achter. BLE heeft ook een laag energieverbruik.

**DASH7 Alliance Protocol:** Er kan gecommuniceerd worden van meters tot kilometers.

- B.L.A.S.T. Principe:
  - *Blasty:* Data-overdracht is abrupt en bevat geen inhoud zoals video of audio.
  - *Light:* Voor de meeste applicaties zijn de pakketgroottes beperkt tot 256 bytes.
  - *Asynchronous:* De command-reactie vereist door het ontwerp geen "handshake" of synchronisatie tussen apparaten.
  - *Stealthy:* DASH7 maakt niet gebruik van discovery beacons.
  - *Transitional:* DASH7 is upload-centric, niet download-centric, dus de apparaten moeten niet uitgebreid beheerd worden door vaste infrastructuren.

**Endianness:** De volgorde van bytes. Bijvoorbeeld: in een *Big Endian* hexadecimale string zal de *big end* van de string eerst zetten en dus de meest significante byte eerst geplaatst (ABCD). Big Endian is het vaakst voorkomend. Bij Little Endian zal de minst significante byte eerst geplaatst worden (DCBA).

**LoRa:** (Long Range Radio) is een netwerk dat bestaat uit zendmasten waarnaar zeer korte pakketjes verstuurd kunnen worden. Het is bedoeld voor toestellen die bv enkel statusberichtjes hoeft te verzenden en niet een constante verbinding nodig hebben. De snelheid is beperkt tot maximum 50kb/s en de reikwijdte van 2.5 tot 15km.

**MQTT:** Messaging protocol voor bij TCP/IP. Werkt op basis van publish-subscribe (afzender "publicht"; dus stuurt het bericht de wereld in en "subscribers" krijgen berichten aan die ze willen zien).

- Methoden:
  - *Connect*: Wachten op connectie met de server.
  - *Disconnect*: Wacht totdat MQTT klaar is met taak, en tot de TCP/IP sessie gedaan is.
  - *Subscribe*: Wacht totdat de Subscribe of Unsubscribe methode klaar is.
  - *Unsubscribe*: Vraagt aan de server om te unsubscribe van een of meer onderwerpen.
  - *Publish*: Stuur een bericht naar de server en keert terug als het afgeleverd is.
- Broker: de broker is verantwoordelijk voor het verdelen van de berichten naargelang het topic.

**SIGFOX:** SIGFOX is een bedrijf dat zich specialiseert in draadloze netwerken, niet anders dan LoRa. Hun bedoeling is low-energy apparaten te laten communiceren. De apparaten sturen dan constant korte berichtjes. De reikwijdte van een SIGFOX base station is ongeveer 40-50km en de pakketjes zijn ongeveer 12 bytes groot. Al concurren SIGFOX en LoRa op het Internet of Things gebied, bestaan er dual modules die beide netwerken ondersteunen.

**UART:** Universal Asynchronous Receiver/Transmitter. Een UART is een onderdeel van een systeem dat data omzet van parallel naar serieel of omgekeerd. Het wordt vaak in microcontrollers gebruikt om data (asynchroon) naar een seriële poort op een computer te versturen.

**VOC:** Volatile Organic Compound (Nederlands: VOS) is een verzamelnaam voor koolwaterstoffen die gemakkelijk verdampen. Ze worden gezien als milieuonvriendelijk, en kunnen schadelijke effecten hebben op mensen. Een VOC-Sensor is dus een sensor die deze stoffen opspoort in de lucht.

**Z-WAVE:** Wordt gebruikt in domotica. Kan tot tussen de 50 en 150m gebruikt worden over two-way communicatie. Betrouwbaar: als het pakketje niet aankomt probeert de sender het opnieuw. Lukt het weer niet wordt een andere route gezocht. Kostprijs ligt hoog.