Technisch Ontwerp
Toilet Usage Monitor

Organisatie:	Hago & Christelijke Hogeschool Windesheim
Opdrachtgever:	HAGO Windesheim
Contactpersoon Opdrachtgever:	Tim ten Bokkel Huinink
Begeleider:	Gido Hakvoort
Versie:	2.1
Datum:	12 januari 2018
Studenten:	Willem Fikkert, Eldin Zenderink & Michiel van Dalfsen
Studentnummers:	S1079181, S1077709, S1068959
Instelling:	Christelijke Hogeschool Windesheim
Opleiding:	ICT Embedded Systems and Automation
Semester:	1
Jaar:	4

Documenthistorie

Datum	Versie	Beschrijving	Auteur
28 november 2017	1.0	Initiële versie	Michiel van Dalfsen, Eldin Zenderink en Willem Fikkert
1 januari 2018	2.0	Update diagrammen, zodat ze overeenkomen met realisatie	Willem Fikkert
13 januari 2018	2.1	Update To, zodat het overeenkomt met realisatie	Michiel van Dalfsen, Eldin Zenderink en Willem Fikkert

Inhoud

Inleiding

In dit document worden alle technische zaken van dit project beschreven. Denk hierbij aan welke data wordt overgestuurd tussen de verschillende hardware onderdelen. Allereerst worden de technische requirements beschreven. Hierbij hebben we enkele diagrammen gemaakt die het duidelijk maken hoe dit project gerealiseerd wordt:

- Sequence diagram
- Deployment diagram
- Klassendiagram
- State machine diagram
- Timing diagram
- Activity diagram
- Database ontwerp

Vervolgens wordt de code standaard die wij voor dit project gaan gebruiken beschreven. Daarna wordt de technische kant van security beschreven in dit document.

Concept

Toilet Usage Monitor(verder benoemd als T.U.M.), deze zal toilet gebruiken gaan registreren zodat een schoonmaker overzicht heeft over hoe vaak een toilet gebruikt is. Deze data moet beschikbaar zijn op een (web)interface om uitgelezen te kunnen worden. Hierdoor kunnen schoonmakers flexibeler ingezet kunnen worden. Zodat de wc's die vaker gebruikt worden ook vaker worden schoongemaakt.
Om dit te bereiken zal er gebruik gemaakt gaan worden van sensormodules die een toiletgebruik gaan registreren en mastermodules die de data van de sensormodules verzamelen om deze data vervolgens te verwerken en op te sturen naar een globale database.
Een sensormodule zal bestaan uit:

- Sensor
- Microcontroller
- Communicatie module (WiFi/ Bluetooth low energy)

Een mastermodule zal bestaan uit:

- Microcontroller
- 2 Communicatie modules (2 WiFi/ 1 WiFi en 1 Bluetooth low energy)

Technische requirements

Om het T.U.M. systeem goed te laten werken en aan de eisen van de opdrachtgever te laten voldoen moeten er een aantal requirements opgesteld worden.
Er moeten een aantal onderdelen komen:

- Een module die toilet gebruiken kan registreren .(sensormodule)
- Een module die de met het netwerk verbonden is. (mastermodule)
- Een database waar de gegevens in komen te staan. (database)
- Een interface waar de gegevens uit de database uitgelezen kunnen worden.(web interface)

Deze onderdelen moeten met elkaar kunnen communiceren. dit betekent:

- De sensormodule moet kunnen communiceren met de mastermodule.
- De mastermodule moet kunnen communiceren met de sensormodule.
- De mastermodule moet kunnen communiceren met de database.
- De web interface moet de database kunnen uitlezen.

Hier wordt steeds verwezen naar de user stories in het functioneel ontwerp die gerealiseerd worden.

1.1 Activeer sensor
Hiervoor is een sensor nodig die “objecten” kan detecteren binnen een vooraf bepaalde afstand. Ook is er een manier nodig om te communiceren naar een centraal punt (mastermodule).

1.2 Deactiveer sensor
Als de sensor gedeactiveerd (het object wat nu gedetecteerd is, bevindt zich op de vooraf bepaalde afstand) wordt moet er gekeken worden voor hoelang de sensor geactiveerd was, hierdoor kun je ongeveer bepalen of een persoon gebruik heeft gemaakt van het toilet.

2.3 Bevestig schoonmaken van toilet
Als een sanitaire ruimte is schoongemaakt moet er door de schoonmaker kunnen worden doorgegeven dat de ruimte weer schoon is, de schoonmaker kan dit door middel van een schakelaar die op de mastermodule bevestigd is. Aan de hand hiervan moet het aantal wc gebruikers worden gereset.

3.1 Inzage gebruik toiletten/sanitaire ruimte
Er moet minimaal een globaal overzicht waar alle data van de database zichtbaar is. Hier is dus ook te zien hoe vaak een toilet en sanitaire ruimte is gebruikt.

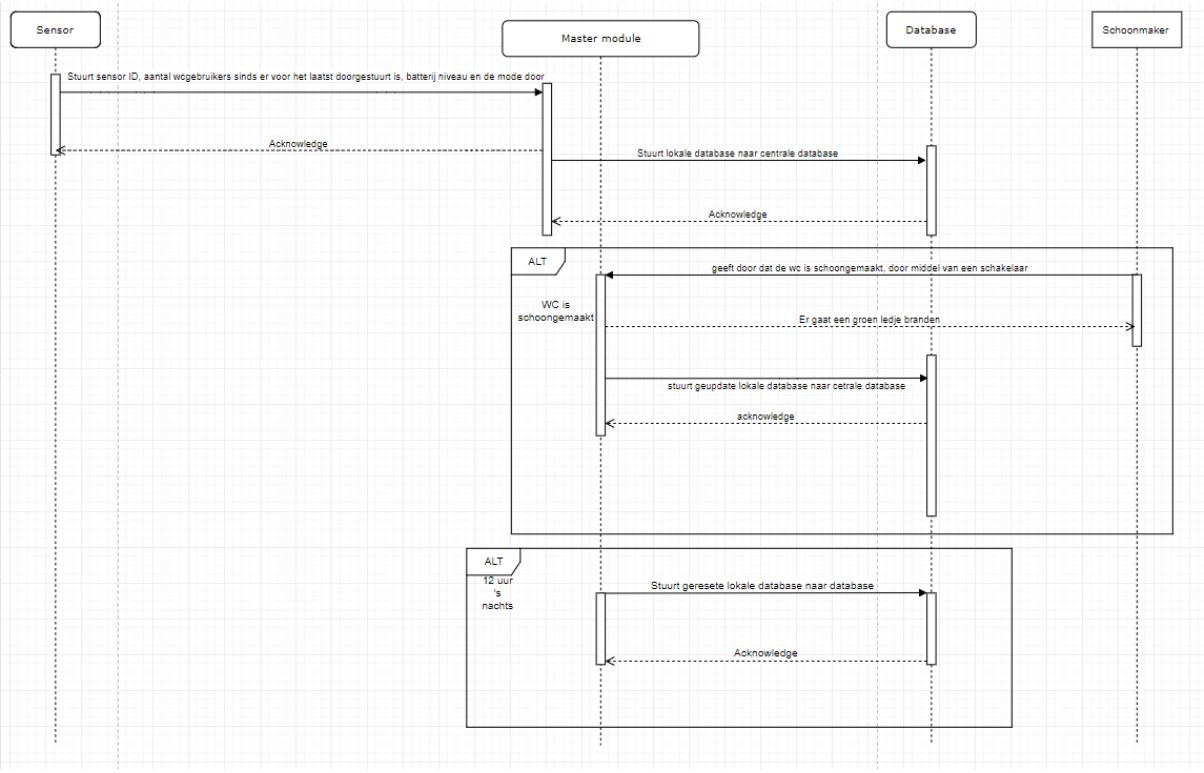
4.1 Inzage sensormodule statistieken (systeembeheerder)
Er moet minimaal een globaal overzicht waar alle data van de database zichtbaar is. Hier zijn dus ook de statistieken van de sensoren te zien.

5.1 Instellen mastermodule (toevoegen)
Er moet een nieuwe mastermodule aan het systeem toegevoegd kunnen worden, zodat het systeem ook uitbreidbaar is. Hier stel je bijvoorbeeld in bij welk gebouw, verdieping, geslacht de mastermodule hoort.

5.2 Instellen sensormodule
Er moet een nieuwe sensormodule aan het systeem toegevoegd kunnen worden zodat het systeem uitbreidbaar is. Hierbij kun je de standaard afstand instellen die het verschil tussen activeren en deactiveren bepaald. Ook stel je in bij welk gebouw, verdieping, geslacht.

5.3 Inzage sensormodule statistieken (installateur)
Een installateur moet de beschikking hebben om de sensormodule statistieken in te zien.

Sequence diagram

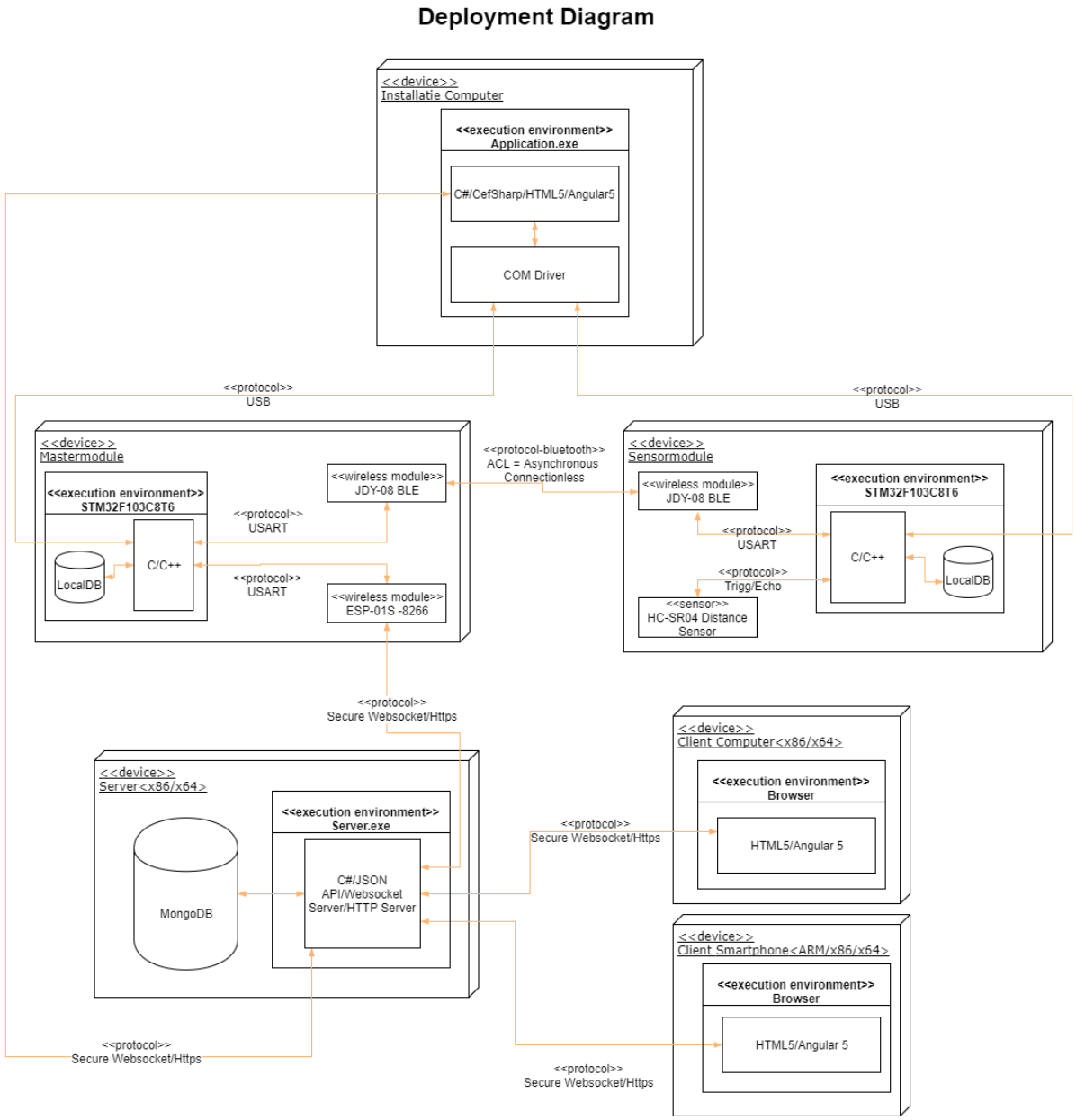


(figuur 1. Sequentie diagram)

Het sequence diagram in het technische ontwerp verschilt niet veel met het functioneel ontwerp, in dit diagram worden alleen de dingen getoond die gerealiseerd worden in dit project. In dit diagram wordt ook verteld wat er precies wordt doorgestuurd. De sensormodule stuurt zijn sensor ID, het aantal wc gebruikers sinds de laatste verbinding, de mode en hoe vol de batterij nog is naar de mastermodule. Deze stuurt een acknowledge om aan te geven dat de data ontvangen is. De mastermodule stuurt daarna de lokale database naar de centrale database die daar een acknowledge als antwoord op geeft. Als de sanitaire ruimte is schoongemaakt geeft de schoonmaker dat door door middel van een schakelaar. Als feedback daarop gaat er een groen ledje branden. Vervolgens wordt de geupdate lokale database weer naar de centrale database gestuurd, als antwoord hierop wordt een acknowledge teruggestuurd. Ook wordt er rond 12 uur ‘s nachts de lokale database gereset. Deze wordt vervolgens naar de database gestuurd, en de database stuurt vervolgens een acknowledge.

Deployment Diagram

Ontwerp voor realisatie



(figuur 2. Deployment diagram voor realisatie)

- Installatie Computer
    a. De installatie computer bevat een applicatie geschreven in c# met een interface geschreven in html5/angular5.
    b. De applicatie communiceert met de modules (sensor & master) via de COMpoort via USB. De applicatie communiceert via de JSON API met de database over Secure Websocket/HTTPS.
    c. De applicatie is bedoeld om de modules in te stellen.

- Mastermodule
    a. De mastermodule bevat een STM32F103C8T6 ARM microcontroller in combinatie met een bluetooth low energy module van type JDY-08 en een Wifi module van type ESP-01S gebaseerd op de ESP8266s. De microcontroller wordt met C/C++ geprogrammeerd.
    b. De microcontroller communiceert met de installatie computer via USB als de microcontroller ingesteld moet worden. De microcontroller communiceert via I2C met de bluetooth module. De bluetooth module wordt gebruikt voor de communicatie met de Sensormodule. De andere draadloze module communiceert met de server, dit wordt een Wifi module. De bedoeling is datdit via de HTTP(S) protocol gebeurt, met mogelijkheid voor Secure Websockets.
    c. De mastermodule is een gateway om de gegevens van de sensormodule door te geven aan de server. Verder wordt in de lokale database tijdelijk de gegevens opgeslagen.

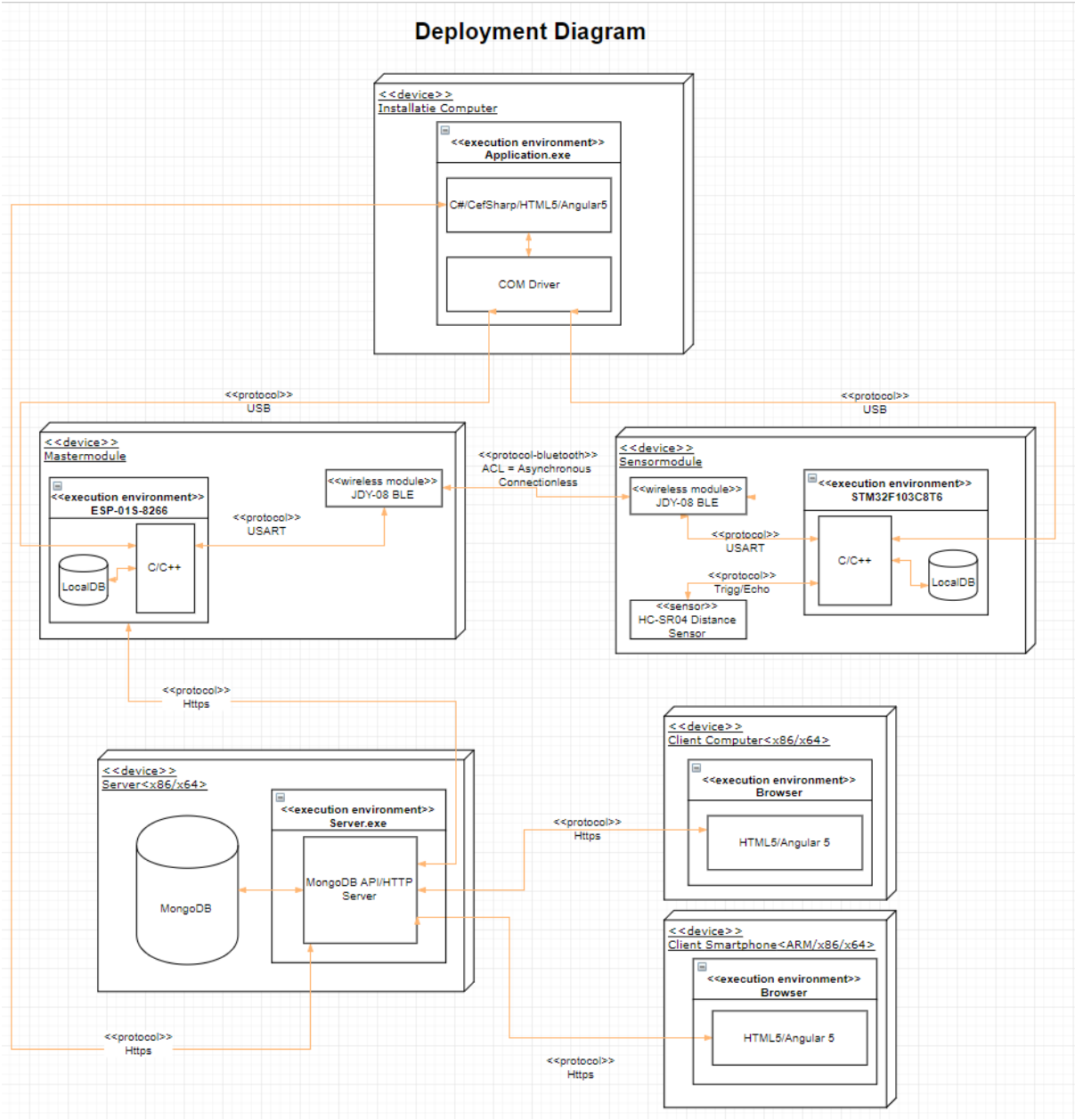
- Sensormodule
    a. De sensormodule bevat een STM32F103C8T6 ARM micro controller in combinatie met een afstand sensor van het type HC-SR04 en een bluetooth module van het type JDY-08. De microcontroller wordt met C/C++ geprogrammeerd.
    b. De microcontroller communiceert met de installatie computer via USB als deze ingesteld moet worden. Via de bluetooth module wordt er gecommuniceerd met de Mastermodule, er wordt een ACL(Asynchronous Connectionless) gebruikt. Via een specifiek voor de afstandsensor geschreven protocol wordt gecommuniceerd vanaf de microcontroller met de afstandsensor.
    c. De sensormodule geeft een signaal aan de mastermodule op vaste momenten en verstuurd dan de opgeslagen gegevens in zijn lokale database.

- Server
    a. Op server draait een applicatie dat geschreven wordt in C# en een MongoDB database.
    b. De applicatie communiceert met de MongoDB via een library, ook host deze een HTTP(S) en (Secure) Websocket server. De server communiceert met de MasterModule door middel van de JSON API via de https/Secure websocket protocol. De server communiceert met de client computer/smartphone via de https/Secure websocket protocol.
    c. De server krijgt gegevens van de mastermodule. Deze wordt in de MongoDB opgeslagen, deze gegevens kan worden opgevraagd via de JSON api. De client's kunnen de website opvragen via de server.

- Client Computer
    a. De client computer draait een browser die een website geschreven in HTML5/Angular5 opvraagt.
    b. De browser communiceert via HTTP(S) met de server om de website op te vragen. De browser waarin de website getoond wordt,communiceert ook via (Secure) Websocket om gegevens uit te wisselen door middel van de JSON API.
    c. De client computer kan de gegevens via de browser inzien en veranderen door middel van de website.

- Client Smartphone\
    a. De cliënt smartphone draait een browser of app die een website geschreven in HTML5/Angular5 opvraagt.
    b. De browser communiceert via HTTP(S) met de server om de website op te vragen. De browser waarin de website getoond wordt, communiceert ook via (Secure) Websocket om gegevens uit te wisselen door middel van de JSON API.
    c. De cliënt smartphone kan de gegevens via de browser inzien.

Ontwerp na realisatie



(figuur 3. Deployment diagram na realisatie

- Installatie Computer (concept)
    a. De installatie computer bevat een applicatie geschreven in c# met een interface geschreven in html5/angular5.
    b. De applicatie communiceert met de modules (sensor & master) via de COMpoort via USB. De applicatie communiceert via de HTTP API met mongodb
    c. De applicatie is bedoeld om de modules in te stellen.

- Mastermodule
    a. De mastermodule bevat een ESP8266s microcontroller in combinatie met een bluetooth low energy module van type JDY-08. De microcontroller beschikt tot een Wifi module van type ESP-01S. De microcontroller wordt met C/C++ geprogrammeerd.
    b. De microcontroller communiceert met de installatie computer via USB als de microcontroller ingesteld moet worden. De microcontroller communiceert via USART met de bluetooth module. De bluetooth module wordt gebruikt voor de communicatie met de Sensormodule. De andere draadloze module communiceert met de server, dit wordt een Wifi module. De bedoeling is dat dit via de HTTP(S) protocol gebeurt, met mogelijkheid voor Secure Websockets. (concept)
    c. De mastermodule is een gateway om de gegevens van de sensormodule door te geven aan de server. Verder wordt in de lokale database tijdelijk de gegevens opgeslagen.

- Sensormodule
    a. De sensormodule bevat een STM32F103C8T6 ARM micro controller in combinatie met een afstand sensor van het type HC-SR04 en een bluetooth module van het type JDY-08. De microcontroller wordt in C geprogrammeerd.
    b. De microcontroller communiceert met de installatie computer via USB als deze ingesteld moet worden. Via de bluetooth module wordt er gecommuniceerd met de Mastermodule. Via een specifiek voor de afstandsensor geschreven protocol wordt gecommuniceerd vanaf de microcontroller met de afstandsensor.
    c. De sensormodule geeft een signaal aan de mastermodule op vaste momenten en verstuurd dan de opgeslagen gegevens in zijn lokale database.

- Server
    a. Op server draait een MongoDB database.
    b. De applicatie communiceert met de MongoDB via een library, ook host deze een HTTP(S). De server communiceert met de MasterModule door middel van de JSON API via de https/Secure websocket protocol. De server communiceert met de client computer/smartphone via de https/Secure websocket protocol.
    c. De server krijgt gegevens van de mastermodule. Deze wordt in de MongoDB opgeslagen, deze gegevens kan worden opgevraagd via de MongoDB api. De client’s kunnen de website opvragen via de server.

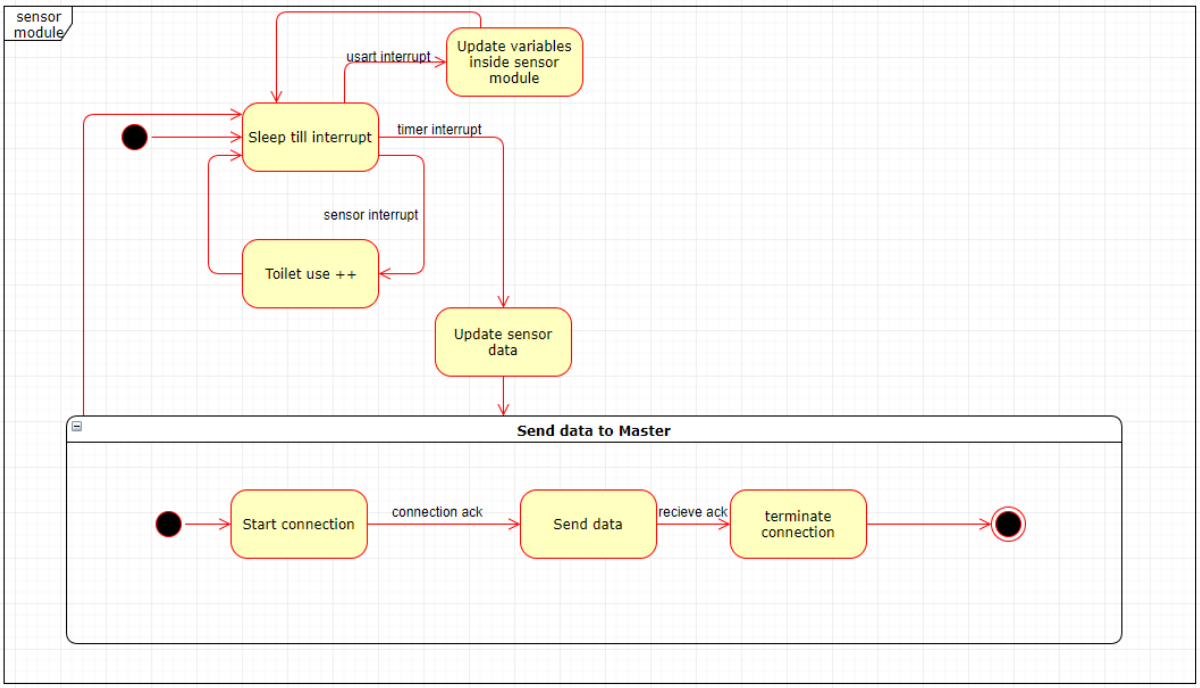
- Client Computer
    a. De client computer draait een browser die een website geschreven in HTML5/Angular5 opvraagt.
    b. De browser communiceert via HTTP(S) met de server om de website op te vragen. De browser waarin de website getoond wordt toont data die uit de MongoDB gehaald word.
    c. De client computer kan de gegevens via de browser inzien en veranderen door middel van de website.

- Client Smartphone (concept)
    a. De cliënt smartphone draait een browser of app die een website geschreven in HTML5/Angular5 opvraagt.
    b. De browser communiceert via HTTP(S) met de server om de website op te vragen. De cliënt smartphone kan de gegevens via de browser inzien.

State Machine Diagram

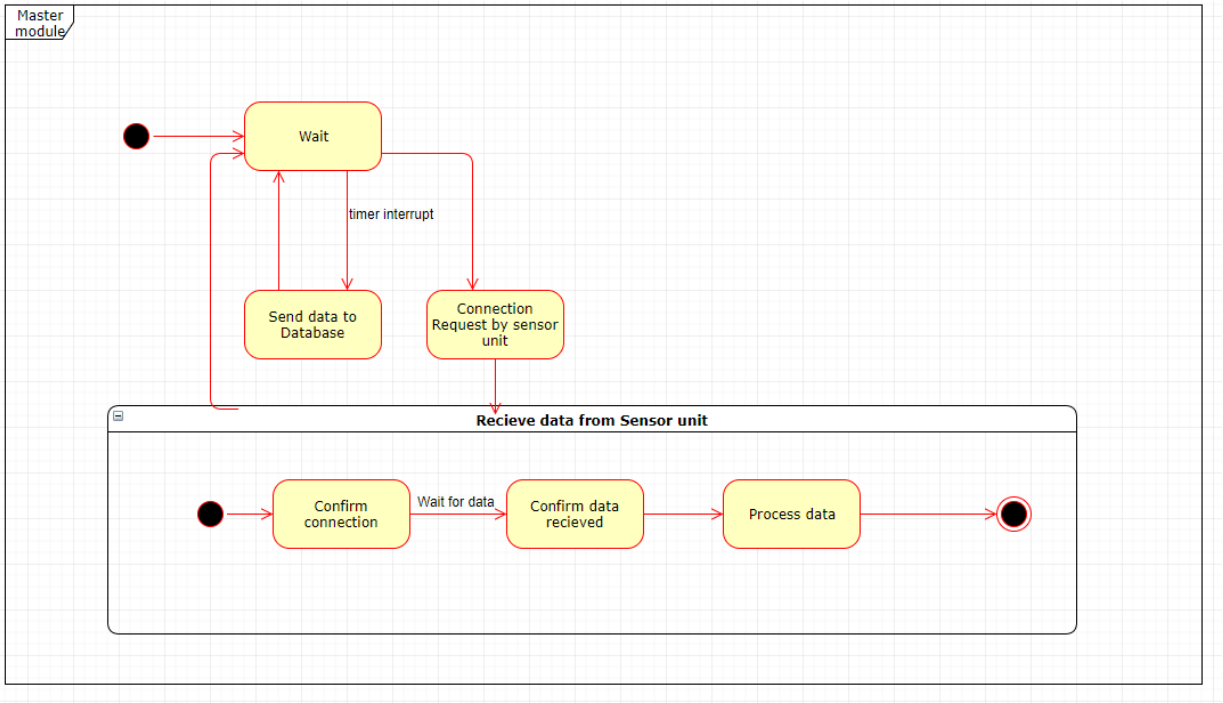
State Machine Diagram voor realisatie

Sensor module



(figuur 4. Statemachine diagram sensormodule)
Dit state machine diagram is van de sensormodule, de sensormodule wordt gebruikt voor de userstories over het registreren van een toilet gebruik en het registreren van sanitair gebruik. De sensormodule slaapt tot deze door een interrupt wakker gemaakt wordt. Er zijn drie verschillende interrupts die hiervoor gebruikt worden, een interrupt die wordt veroorzaakt door de sensor, een timer die na X aantal seconden/minuten activeert, of een USART interrupt die wordt veroorzaakt wanneer er met USART data wordt verstuurd naar de module.
Wanneer de sensor een interrupt veroorzaakt betekent dit dat er iemand de sensor heeft getriggerd wat betekent dat het toilet gebruikt is, deze variabele zal worden aangepast waarna de module weer terug gaat naar slaapmodus.
Wanneer de timer een interrupt veroorzaakt betekent dat het tijd is om data te versturen naar de mastermodule nadat de data verstuurd is zal de module weer in slaapmodus gaan.
Wanneer de USART interrupt wordt getriggerd betekent dat er data verstuurd wordt over usart, deze data wordt verwerkt en daarna zal de sensor weer in slaapmodus gaan.

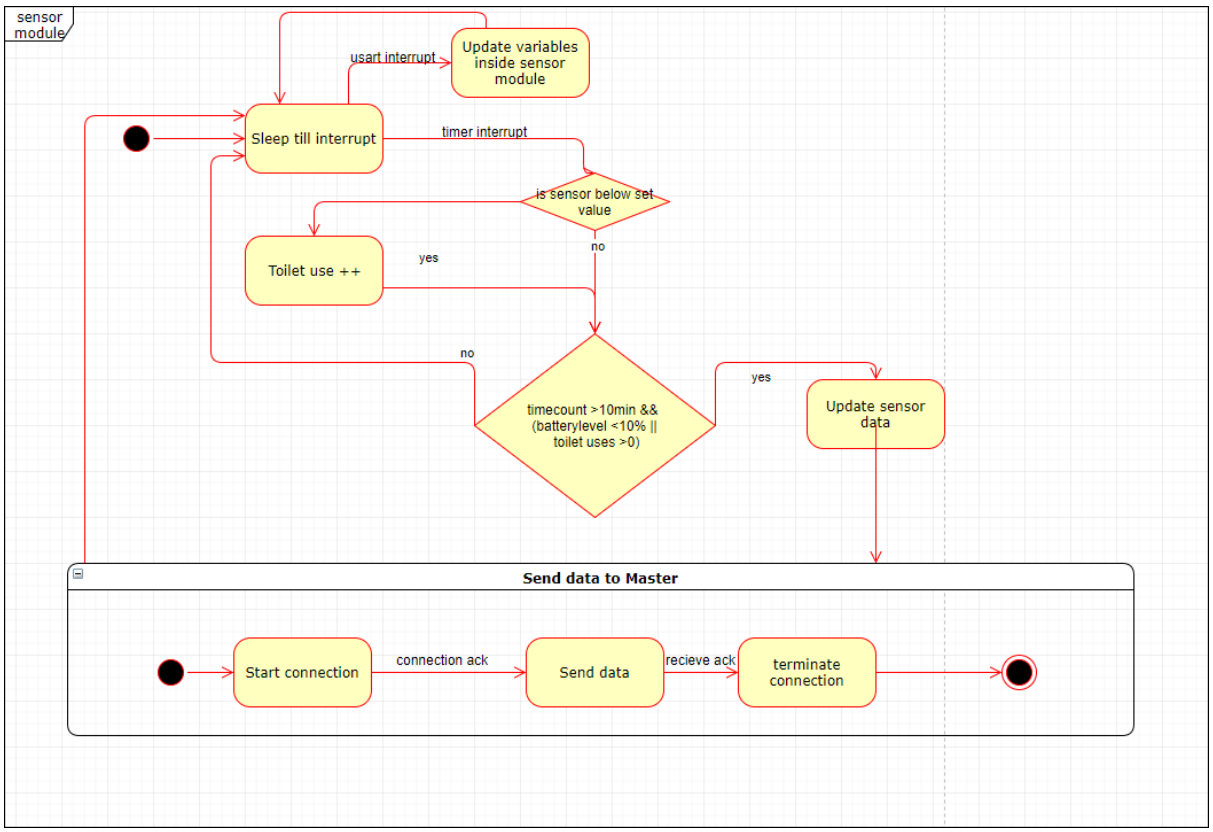
Master module



(figuur 5. State machine diagram Mastermodule)
De mastermodule is het centrale punt van een sanitaire ruimte, alle verschillende sensormodule zullen data versturen naar de mastermodule.
De mastermodule zit continu in een wachtende staat tot een sensormodule connectie wil maken waarna de ontvangen data verwerkt en in de interne database gezet wordt. Of tot de timer interrupt wordt getriggerd waarna de data uit de interne database naar de externe database wordt gestuurd.

State Machine Diagram na realisatie

Sensormodule



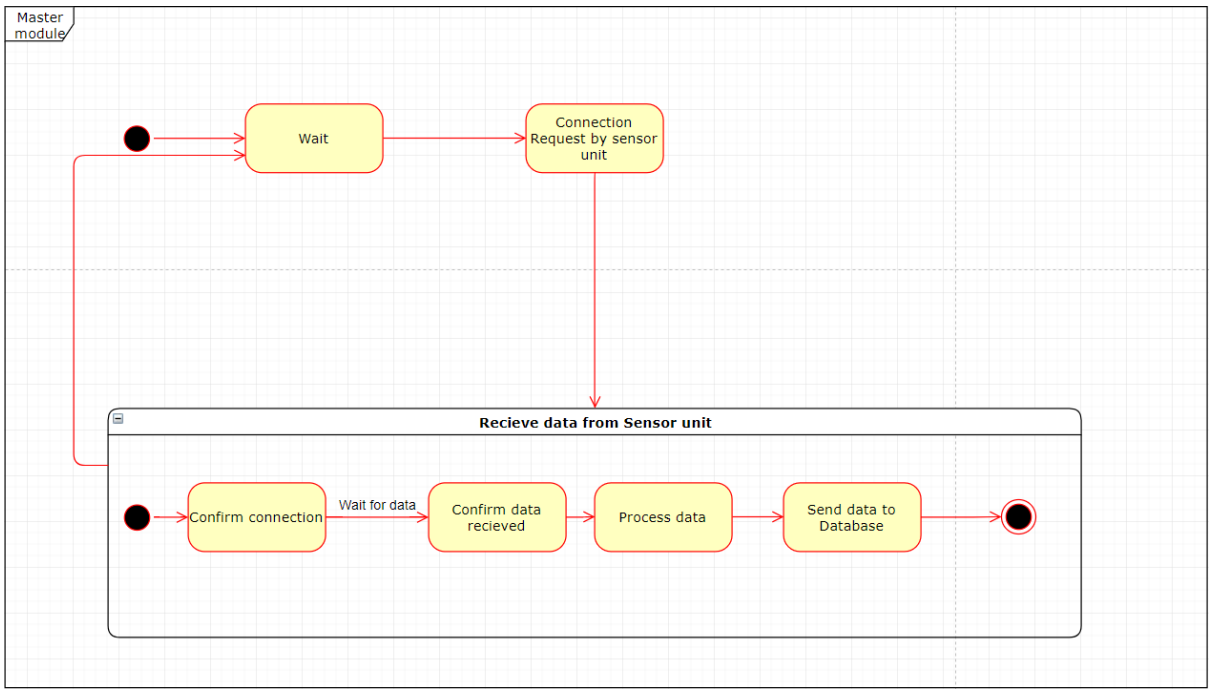
(figuur 6.)

Dit state machine diagram is van de sensormodule, de sensormodule wordt gebruikt voor de userstories over het registreren van een toilet gebruik en het registreren van sanitair gebruik. De sensormodule slaapt tot deze door een interrupt wakker gemaakt wordt. Er zijn twee verschillende interrupts die hiervoor gebruikt worden, een timer die na X aantal seconden/minuten activeert, of een USART interrupt die wordt veroorzaakt wanneer er met USART data wordt verstuurd naar de module.

Wanneer de timer een interrupt veroorzaakt betekent dit dat er een aantal seconden verstreken zijn. Nadat er een interrupt is geweest wordt er naar de sensor gekeken, als daar iemand dichter bij staat dan de ingestelde waarde dan wordt er een gebruik geregistreerd. Nadat er wel of geen gebruik geregistreerd is wordt de waarde van de batterij opgehaald, als deze onder de 10% is wordt er een error/warning gezet wat wordt doorgestuurd naar de master. Als dit niet het geval is kijkt men na de verstreken tijd, als dit boven de 10 minuten is en er is in ieder geval 1 gebruik geregistreerd wordt de data alsnog verstuurd naar de master, hierna keert de module terug naar de slaapstand.

Wanneer de USART interrupt wordt getriggerd betekent dat er data verstuurd wordt over usart, deze data wordt verwerkt en daarna zal de sensor weer in slaapmodus gaan.

Master module



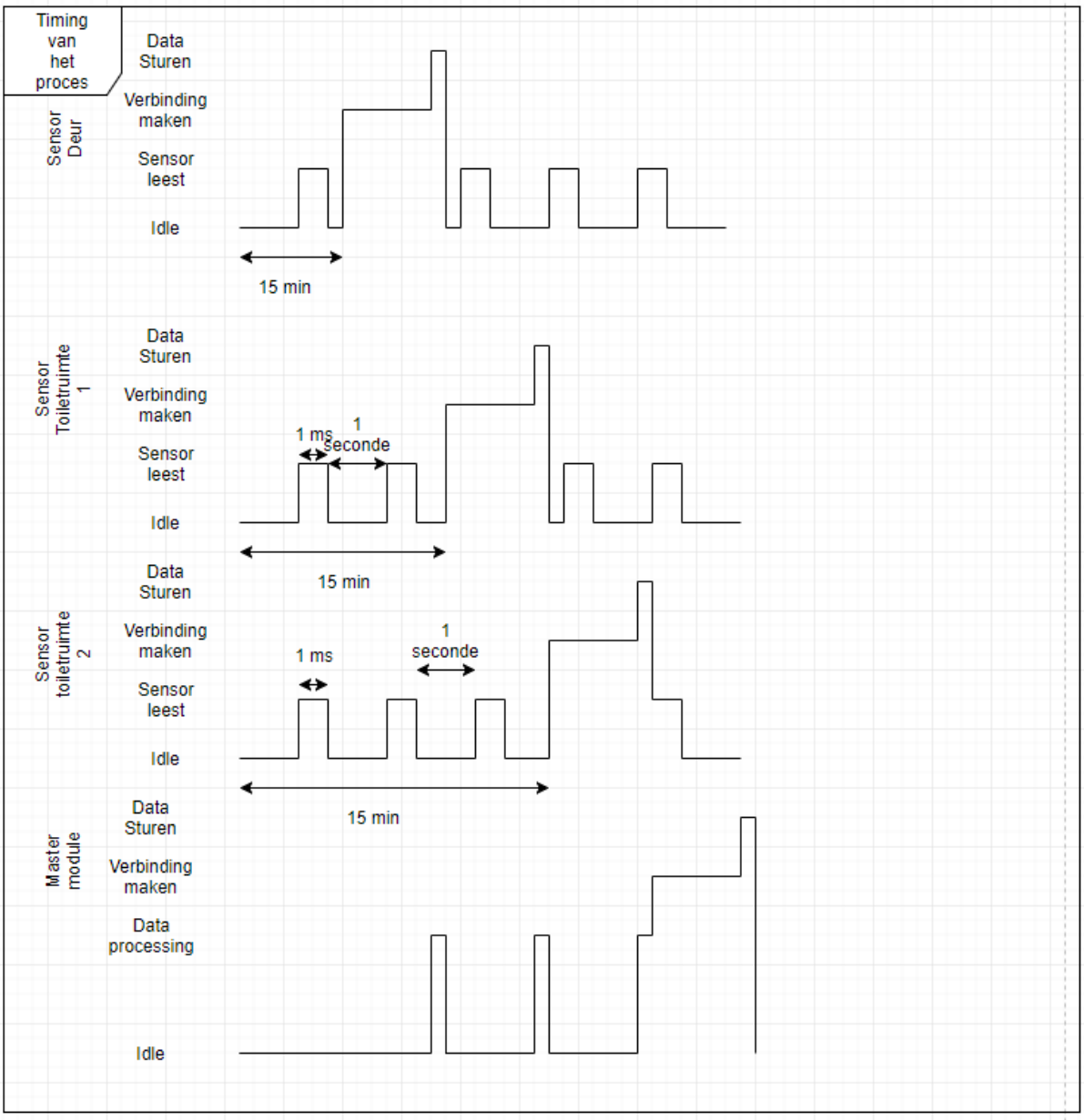
(figuur 7.)

De mastermodule is het centrale punt van een sanitaire ruimte, alle verschillende sensormodule zullen data versturen naar de mastermodule.

De mastermodule zit continu in een wachtende staat tot een sensormodule connectie wil maken waarna de ontvangen data verwerkt en in de interne database gezet wordt. Of tot de timer interrupt wordt getriggerd waarna de data uit de interne database naar de externe database wordt gestuurd.

Timing Diagram

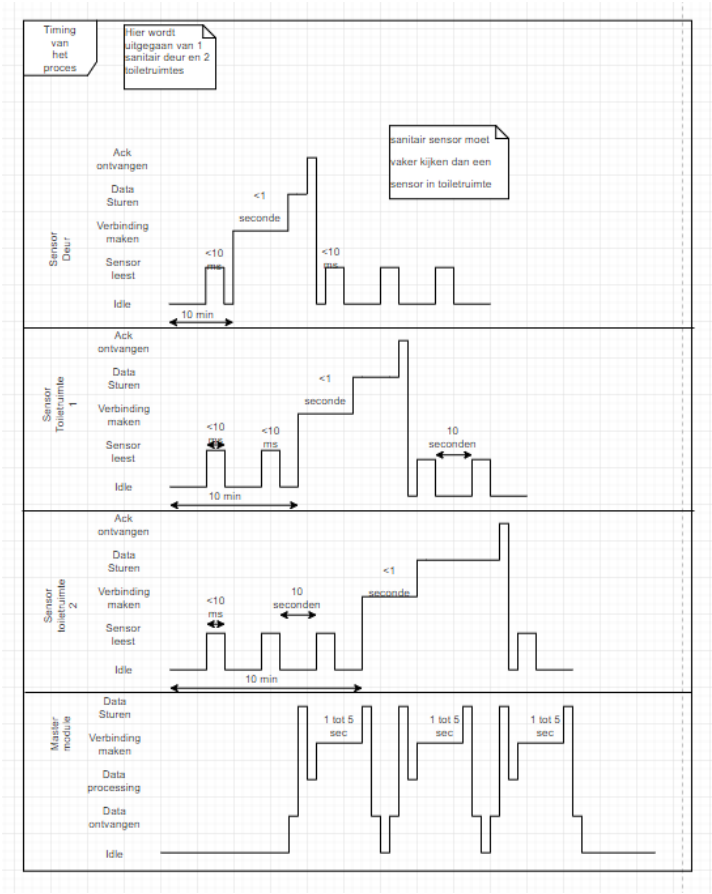
Timing diagram voor realisatie



(figuur 8. Timing diagram)

Als een sensor niets doet dan is deze idle. Een sensor gaat steeds even kijken of er een ‘object’ in zijn ‘zichtveld’ is. Na 15 minuten gaat een sensormodule data sturen naar de mastermodule. Op het moment dat de mastermodule de data ontvangt gaat deze de data gelijk verwerken. Op het moment dat de mastermodule de data van alle sensoren heeft ontvangen gaat deze verbinding maken met de centrale database. Als er verbinding is wordt de data verstuurd. Daarna gaat de mastermodule terug naar de idle state.

Timing diagram na realisatie

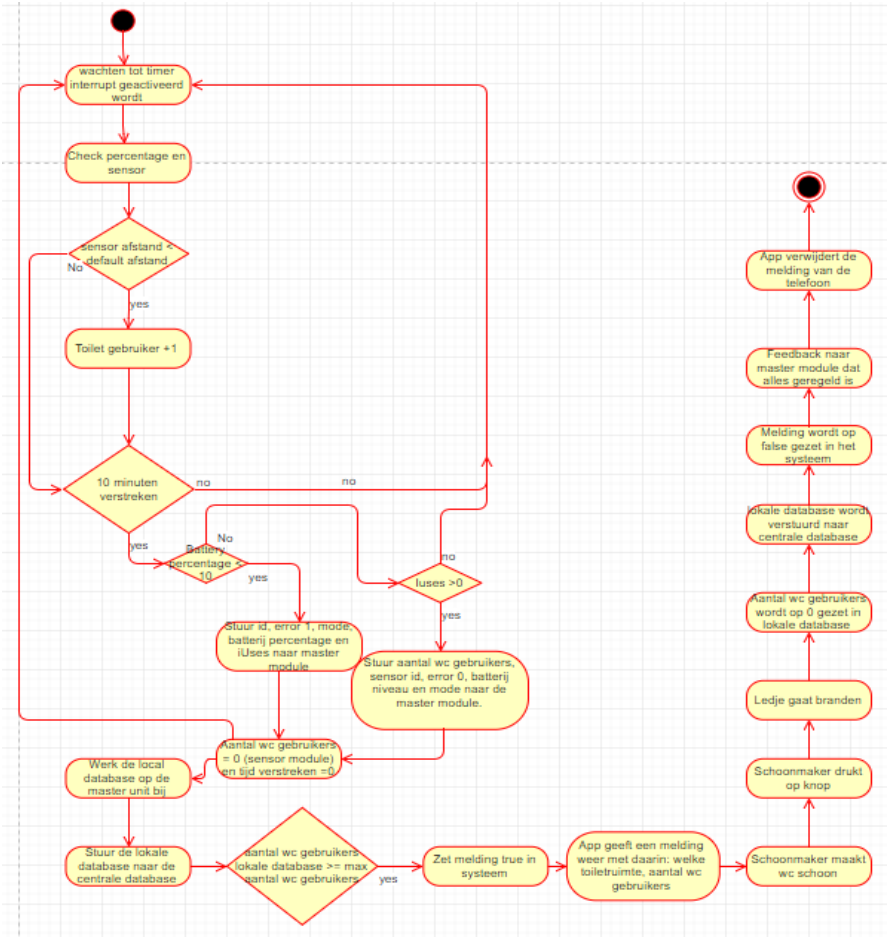


(figuur 9. Timing diagram na realisatie)

In de timing van het programma zijn ook wat dingen veranderd, zo wordt er bij de toilet sensoren om de 10 seconden gecheckt of iemand gebruik maakt van het toilet. De data wordt na 10 minuten naar de master module verzonden. Ook zijn er acknowledgements toegevoegd, als de master module de data van de sensoren heeft ontvangen verstuurt de master een ack naar de sensoren, waarna het versturen wordt gestopt. Zolang er geen ack binnenkomt blijft de sensor om ongeveer 15 seconden versturen.

De master module ontvangt data, verstuurt gelijk een ack en gaat daarna de data verwerken. Als de data is verwerkt wordt er verbinding gemaakt met de database en wordt dit verzonden, als de database de data heeft ontvangen verstuurt deze een ack, als deze ack is ontvangen gaat de master weer in zijn idle state totdat er weer data wordt ontvangen.

Activity Diagram

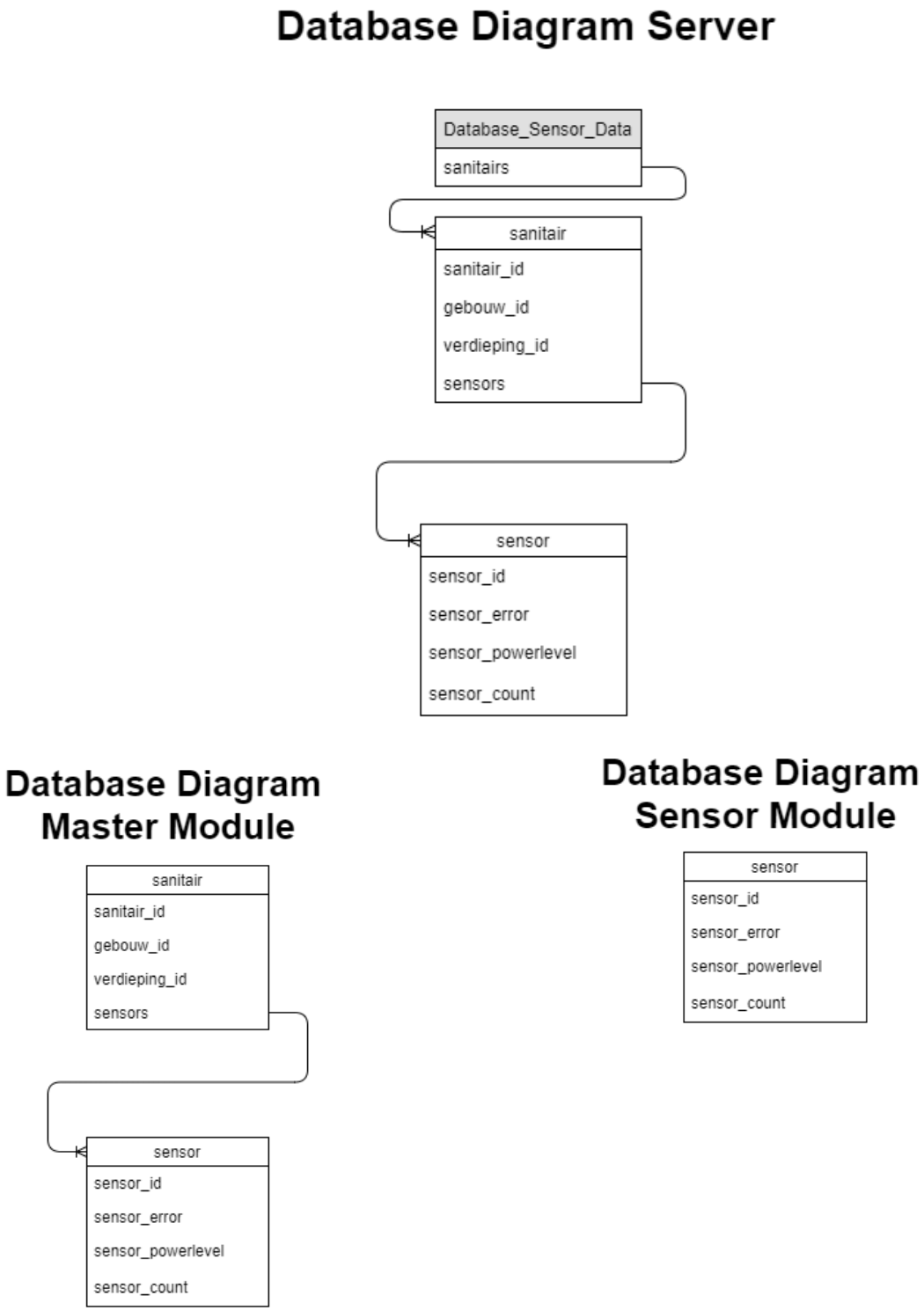


(figuur 10. Activity diagram)

Als het proces start wordt er eerst gewacht tot de timer interrupt getriggerd wordt. Als deze geactiveerd wordt, gaat de sensor kijken of de afstand die gemeten wordt kleiner is dan de default afstand, ook wordt het percentage van de batterij berekend. Als de sensor afstand kleiner is dan de default afstand dan wordt toilet gebruiker opgeteld. Daarna wordt er gekeken of de 10 minuten voorbij zijn. Zo ja, kijk of het percentage van de batterij lager is dan 10& of het toiletgebruik meer dan 0 is. Bij het geval van laag percentage wordt er een error status 1 doorgestuurd, deze krijgt altijd voorrang op toiletgebruik omdat toiletgebruik ook hierbij naar de database wordt verstuurd. Bij toiletgebruik van meer dan 0 wordt er error status 0 doorgestuurd, met de gebruikelijke data. Als dit verzonden is wordt toiletgebruik op 0 gezet en de timer counter wordt gereset. Daarna wordt er gewacht op een interrupt. De mastermodule werkt de lokale database bij. Als het aantal wc gebruikers van de lokale database groter is dan het maximum aantal dat is toegestaan moet er een melding worden verstuurd naar de telefoon van de schoonmaker. De schoonmaker maakt vervolgens de wc schoon. Als de sanitaire ruimte schoongemaakt is moet de schoonmaker dit doorgeven door middel van een knop in te drukken die bevestigd is op de mastermodule. Als feedback daarop gaat er een ledje branden. Het aantal wc gebruikers wordt op 0 gezet in de lokale database, die vervolgens weer naar de centrale database wordt verstuurd. Er wordt ook nog feedback naar de mastermodule gestuurd om door te geven dat alles geregeld is. Daarna verdwijnt de melding weer van de telefoon van de schoonmaker.

Database Ontwerp

Database Ontwerp voor realiseren



(figuur 11. Database ontwerpen voor realisatie)

Inleiding
Voor de database wordt niet het gebruikelijke SQL gebruikt maar JSON. Zo'n database wordt ook wel een NoSQL database genoemd. Dit betekent dat de manier waarop een database gebouwd wordt fundamenteel anders is dan bij SQL. Bijvoorbeeld, bij een JSON database is het namelijk de bedoeling dat de programmeur er voor zorgt dat er geen dubbele resultaten voorkomen. Dit heeft als voordeel dat dit erg schaalbaar en zeer efficiënt is.

De keuze om JSON te gebruiken is voornamelijk gedaan omdat dit zeer makkelijk in gebruik is binnen highlevel programmeer talen zoals JavaScript en TypeScript, omdat ze json standaard ondersteunen zonder gebruik te hoeven maken van third party libraries. Daardoor zal er weinig moeite nodig zijn om een basische front-end te ontwikkelen.

Om een niet zelf een half gebakken json serializer / api op te zetten zal er van een service gebruik gemaakt worden die "MongoDB as a service" genoemd wordt. Bijkomend voordeel bij het gebruik van zo'n dienst is dat zij al een volledig voorgeschreven HTTPS database API (zie kopje Database API) hebben, waardoor toegang via bijvoorbeeld ESP8266 makkelijker is. Tijdens de ontwikkeling zal gebruik gemaakt worden van de gratis ontwikkelaars abonnement, dit is een test database die NIET geschikt is voor gebruik bij oplevering! Mede omdat deze geen backup mogelijkheden heeft.

Om terug te komen op het gebruik van JSON met high level programmeertalen, MongoDB heeft een tal van open source libraries waarmee je zeer makkelijk kunt interfacen met de database. Zie bijvoorbeeld:
https://docs.mongodb.com/ecosystem/tools/http-interfaces/#http-interfaces

Omschrijving Database
De database hier beschreven bevat alleen de database model om de gegevens van alle sensor en master modules op te slaan, zoals aangegeven in de te realiseren use cases in het functioneel ontwerp.

Om te voorkomen dat er dubbele resultaten met als gevolg dat er geen overzicht is waarin te zien is welke wc/sanitair waar het vaakst gebruikt is, zijn er een paar regels voor het invullen van de database.

De database bevat gegevens over het sanitair. Om goed in te kunnen zien om welk sanitair het gaat is het van belang dat de combinatie tussen de velden: sanitair_id, gebouw_id en verdieping_id uniek is.

In feite is de database een text bestand waarin alle gegevens in wordt opgeslagen, welke wordt beheerd door MongoDB.

Als je zou kijken naar de structuur van de database, zou het er zo ongeveer uitzien:

Database_Sensor_Data :

```
{
    "sanitairs": [{
        "sanitair_id" : "mannelijk links van de trappen",
        "gebouw_id" : "Gebouw_X_adreshier",
        "verdieping_id" : "verdieping 1 begane ground",
        "sensors": [{
            "sensor_id": "urnior_1",
            "sensor_count": 0,
            "sensor_powerlevel": 100,
            "sensor_error": false
        }]
    }]
}
```

Database_Master_Module:

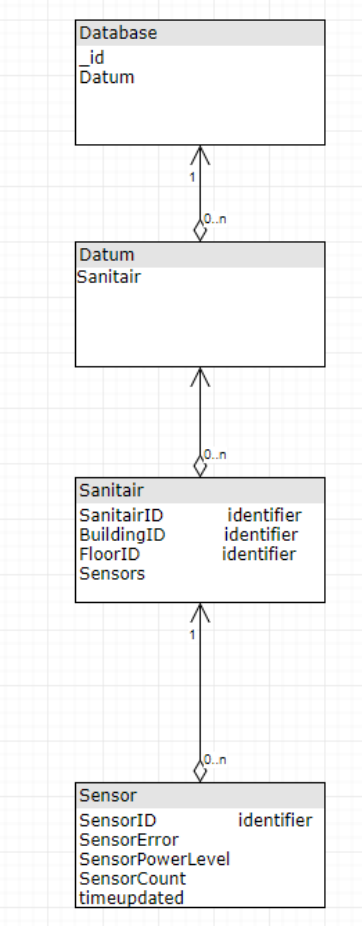
```
{
    "sanitair_id" : "mannelijk links van de trappen",
    "gebouw_id" : "Gebouw_X_adreshier",
    "verdieping_id" : "verdieping 1 begane ground",
    "sensors": [{
        "sensor_id": "urnior_1",
        "sensor_count": 0,
        "sensor_powerlevel": 100,
        "sensor_error": false
    }]
}
```

Database_Sensor_Module:

```
{
    "sensor_id": "urnior_1",
    "sensor_count": 0,
    "sensor_powerlevel": 100,
    "sensor_error": false
}
```

(figuur 12,13,14. Database voorbeelden voor realisatie)

Database ontwerp na realiseren



(figuur 15. Database ontwerp na realisatie)
Er wordt nog steeds gebruik gemaakt van JSON.
De Database bevat een id en 0 of meerdere van de tabel datum.
De tabel datum bevat 0 of meerdere sanitaire ruimtes.
De tabel Sanitair bevat een sanitairID, buildingID en floorID, deze bepalen de identifier van die tabel. Ook bevat een sanitaire ruimte 0 of meerdere sensors.
De tabel Sensor bevat een SensorID, deze bepaalt de identifier van de tabel. Ook bevat de tabel sensor: Error, powerlevel, count, en de tijd van de laatste update.
SensorError wordt gebruikt om bepaalde errors door te sturen:
0 = geen error
1 = batterij bijna leeg

Het grote verschil met de versie voor het realiseren is dat er een datum en een tijd van laatste update is toegevoegd, datum is toegevoegd om zo een soort log bij te houden van het toiletgebruik.
timeupdated in de sensor tabel is toegevoegd om het zo inzichtelijk te maken of een toilet op bepaalde dagen tussen een bepaalde tijd helemaal niet gebruikt wordt.
Op de volgende pagina is een voorbeeld van de database te zien.

Voorbeeld van database:

(figuur 16. Voorbeeld database na realisatie)

Database API

Inleiding

Hier zal besproken worden hoe gecommuniceerd wordt met de database binnen het project. De database heeft een https api die, hoewel niet uitgebreid, prima instaat is om te communiceren met de database en de acties uit te voeren die benodigd zijn voor het project.

MongoDB As A Service

Voor dit project is er besloten om gebruik te maken van MongoDB as A Service, wat inhoud dat er een dienst is die voor jou een server met daarop een mongodb deployed. Voordeel hiervan is dat meteen de database online toegankelijk is en dat er al een (https) API met endpoint klaar staat. Binnen het project is gekozen voor de gratis ontwikkelaars dienst van https://mlab.com . Hier krijgen we voor ontwikkelen een database met 500 MB gratis opslag. Het nadeel is dan weer dat de API specifiek voor deze dienst is ingericht dus is niet direct de HTTP REST API van MongoDB zelf beschikbaar. Je kunt ook in high level programmeertalen gebruik maken van de MongoDB Driver library, die via een specifiek voor MongoDB ontworpen protocol communiceert met de database. Verder brengt dit dus geheugen limitaties mee en daarmee ook een onderdeel wat MongoDB zo goed maakt, snelheid. MongoDB slaat de json normaal gesproken in het RAM op, maar omdat de gratis ontwikkelaars variant een hele kleine virtual machine eigenlijk huurt, heb je ook zeer weinig ram. Voor ontwikkeling geen probleem, maar wel als het systeem in de toekomst opgeleverd wordt. Dat houd dan in dat er naar een betaalde service overgestapt moet worden. Of de database moet zelf gehost gaan worden, waarbij mogelijk de code voor de Master Module herschreven moet worden om met die specifieke database te kunnen werken.

HTTP(s) REST API

Keuze verantwoording
Voor het project is dus voor het gebruik van HTTPS Rest API. Er bestaan C implementaties voor de MongoDB Driver maar helaas werken deze niet met de ESP8266. Er had gekozen kunnen worden om zelf een MongoDB driver te maken voor de ESP8266, maar de verwachting is dat daar niet genoeg tijd voor zal zijn. De ESP8266 heeft al een Wifi Stack met een community er om heen en bestaande libraries om TCP/HTTPS communicatie op te zetten. Daarom is gekozen om de HTTPS API te gebruiken. Uitgebreide informatie over de HTTPS API kan hier gevonden worden:
http://docs.mlab.com/data-api/

Mogelijkheden van de HTTPS API
Gezien het een REST API is bevat deze de meest belangrijke functionaliteiten:

- GET
- POST
- PUT
- DELETE

Daarbij bevat de API nog extra query options om bepaalde attributen in het JSON document aan te passen, te verwijderen of toe te voegen zonder het hele document opnieuw te uploaden.

Wel is de API gelimiteerd in het ophalen van informatie, je kan alleen een volledig document ophalen.

Gebruik met de ESP8266 op de Master Module
Gezien we een lokale versie van de json database voor de sanitair op de master module zelf houden, hoeven we alleen te “upserten”. Dit betekent dat we alleen hoeven te inserten als het nog niet bestaat, of updaten als het er al is. Dit kan met de api als je een bepaalde parameter meegeeft aan de url die je gebruikt om de PUT request uit te voeren. (Zie voor meer uitleg de documentatie van mlab: http://docs.mlab.com/data-api/ .) Vandaar dat de HTTPS API uitermate geschikt is voor onze doeleinden.

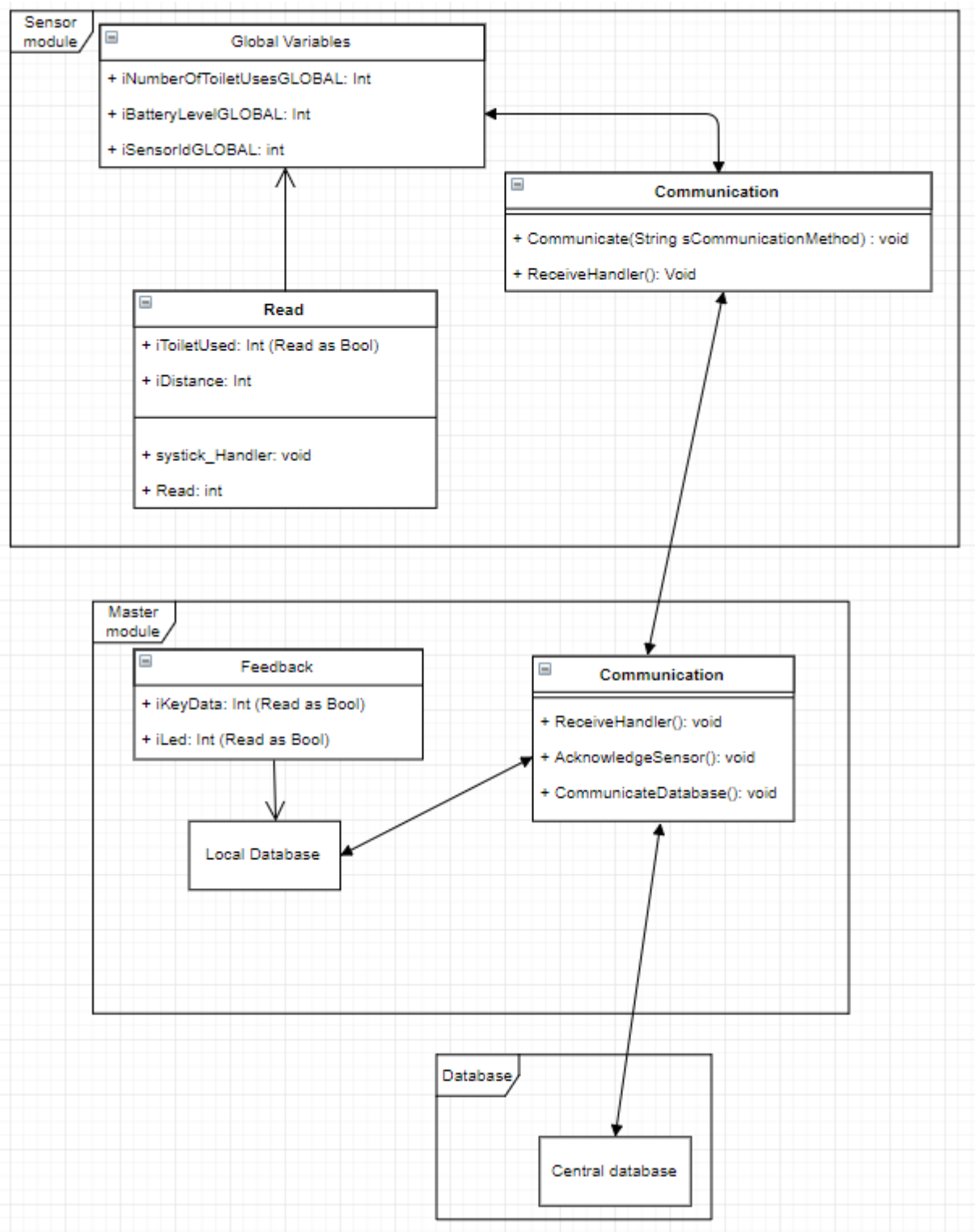
Wijzigingen tijdens de ontwikkeling.

Helaas bleek de ESP8266 de specifieke SSL/TLS protocol van de API niet te ondersteunen. Vanwege het feit dat we al zeer dichtbij de oplever datum zaten en de API verder overzichtelijk in gebruik is besloten we om zodoende gebruik te maken van een HTTP proxy: https://cors-anywhere.herokuapp.com/

Het is aan te raden om dit later te wijzigen naar een oplossing met SSL die wel in staat is om te communiceren met de ESP8266.

Class diagram

Class diagram voor realisatie



(figuur 17. Klassendiagram voor realisatie)

De sensormodule heeft een aantal globale variabelen:

iNumberOfToiletuses is een integer die bijhoudt hoe vaak een toilet is gebruikt nadat er voor het laatst verbinding is gemaakt met de mastermodule.
iBatteryLevel is een integer die aangeeft hoe vol de batterij nog is (0-100).
iSensorId is het id van de sensor. Deze integer moet vanaf de mastermodule gestuurd worden naar de sensormodule

Verder heeft de sensormodule een klasse Read. Deze klasse regelt alles van de sensor. Sensor data uitlezen en aan de hand daarvan bepalen of het toilet gebruikt is.

Daarna volgt de communicatie. Deze klasse heeft een communicate functie die de communicatie regelt van de sensormodule naar de mastermodule, hiermee geef je als parameter mee dat je met bluetooth of wifi stuurt. De ReceiveHandler is om de acknowledge van de mastermodule af te handelen.

De mastermodule heeft een feedback klasse. Deze regelt de handelingen die uitgevoerd moeten worden als de schoonmaker aangeeft of de wc schoongemaakt is. Deze handelingen zijn vooral waardes in de lokale database aanpassen. Op het moment dat de schoonmaker de sleutel omdraait moet er een ledje gaan branden.

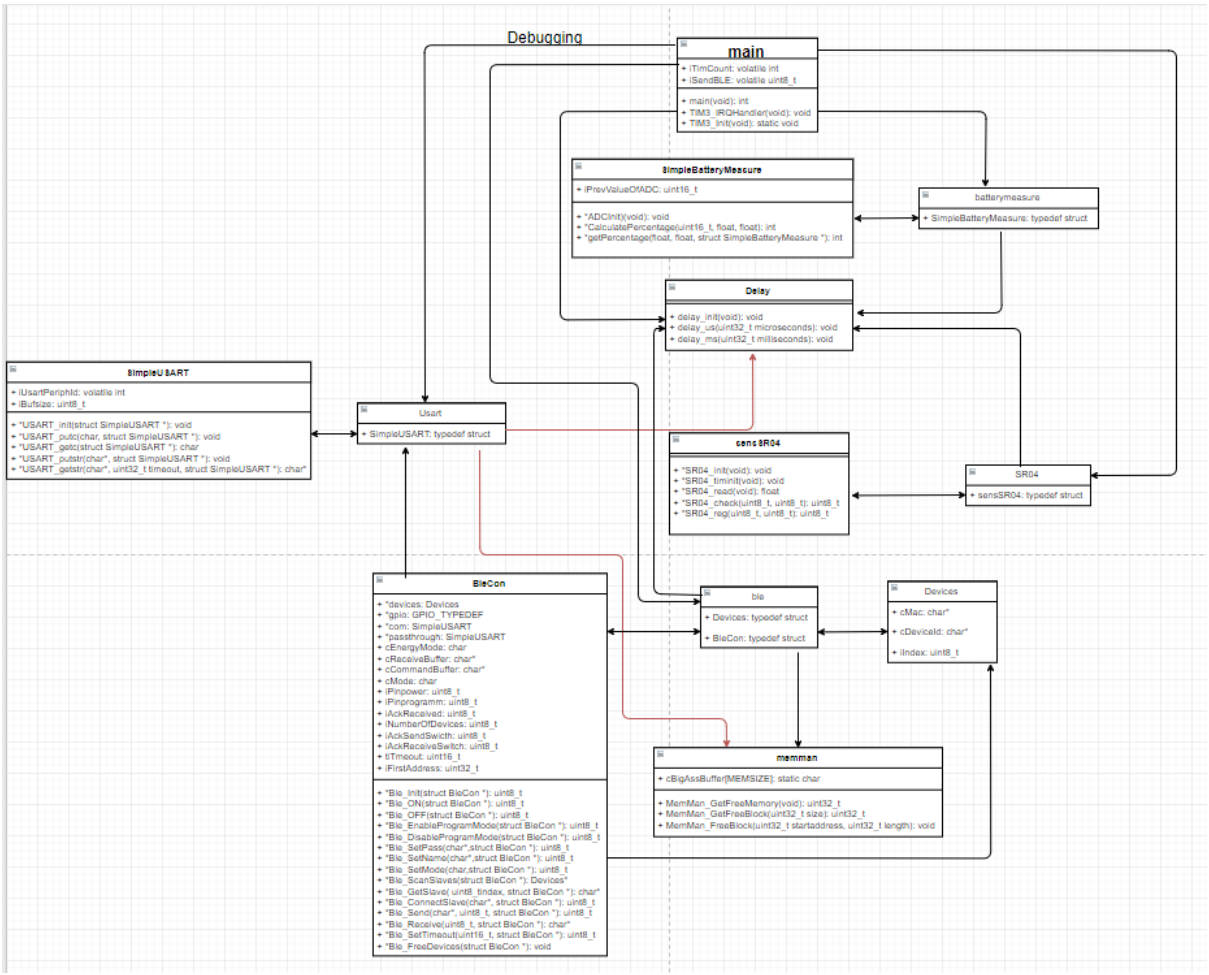
De mastermodule heeft een lokale database, wat hier in staat kun je zien in het database ontwerp van de lokale database.

De mastermodule moet ook communiceren, zowel naar de sensormodule als de centrale database. Hiervoor zijn enkele functies nodig. De ReceiveHandler handelt de gegevens die de sensormodule doorstuurt af en stuurt deze naar de lokale database. De AcknowledgeSensor regelt dat er na de ReceiveHandler een acknowledge naar de sensor wordt gestuurd. Daarnaast wordt CommunicateDatabase gebruikt om de data naar de centrale database te sturen en om de acknowledge van de database af te handelen.

Verder heb je nog een centrale database, wat hier in staat kun je zien in het database ontwerp van de lokale database.

Class diagram na realisatie

Sensormodule:



(figuur 18. Klassendiagram Sensor na realisatie)

Main:

De main heeft een connectie met batterymeasure, sr04, usart, delay en de ble library. Hier wordt het programma uitgevoerd, de juiste functies op de juiste manier en tijdstip aanroepen.

Functies:

- + TIM3_IRQHandler(void): void
  - De interrupt handler van Timer 3, telt elke keer als er getriggerd wordt een counter op, als deze 60 bereikt wordt een variabele waarde op 1 gezet.
  - Heeft geen parameters
  - Returned niets (void).
- + TIM3_Init(void): static void
  - De Timer init, hier wordt de timer ingesteld. De frequentie van de timer is 1/10Hz.
  - Heeft geen parameters
  - Returned niets (void).

BatteryMeasure:

De library batterymeasure meet een ADC waarde en berekent aan de and van deze ADC waardes het percentage van de batterij. De library bevat een typedef struct die een aantal functies bevat.

Functies:

- + *ADCInit)(void): void
    - Initialiseert de ADC, stelt alles goed in. Ook worden de GPIO pinnen geinitialiseerd
    - Heeft geen parameters.
    - Returned niets (void).
- + *getPercentage(float, float): int
    - Deze functie moet worden aangeroepen als je het percentage wilt berekenen, de ADC waarde wordt uitgelezen, en dan wordt de functie CalculatePercentage aangeroepen.
    - Heeft als parameter twee floats. De eerste float is het voltage van de batterij toen deze 100% vol was. De tweede parameter is de voltage dat de batterij 0% moet geven, dus wanneer de sensoren niet meer werken etc.
    - Returned een integer met het percentage.
- + *getPercentage(float, float, struct SimpleBatteryMeasure *): int
    - Deze functie berekent het percentage aan de hand van de ADC waarde.
    - Heeft als parameter een uint16_t en twee floats en de data van de batterymeasure. De uint16_t bevat de ADC waarde. De twee floats zijn hetzelfde als de functie getPercentage, deze worden automatisch doorgegeven van die functie.
    - Returned een int met het percentage.

SR04:

De library SR04 is voor de sensor, hier is de logica van het toiletgebruik geprogrammeerd. De library heeft een struct genaamd: sensSR04, deze struct heeft een aantal functies.

Functies:

- + *SR04_init(void): void
    - Initialiseert de GPIO pinnen die gebruikt worden door de sensor.
    - Heeft geen parameters.
    - Returned niets (void).
- + *SR04_timinit(void): void
    - Initialiseert Timer 2, deze wordt gebruikt voor de sensor.
    - Heeft geen parameters.
    - Returned niets (void).
- + *SR04_read(void): float
    - Meet de afstand tussen de sensor en een object.
    - Heeft geen parameters.
    - Returned de afstand in cm.
- + *SR04_check(uint8_t, uint8_t): uint8_t
    - Berekent een gemiddelde afstand over een aantal metingen, er wordt gekeken of dit lager is dan de afstand waarop het toiletgebruik moet worden opgeteld.
    - Heeft als parameter twee uint8_t variabelen. De eerste uint8_t is de afstand waarop gecheckt moet worden of het lager is dan dat. De tweede uint8_t is het aantal metingen waaruit een gemiddelde moet worden berekent.
- + *SR04_reg(uint8_t, uint8_t): uint8_t
    - Registreert toilet/sanitair gebruik.
    - Heeft als parameter twee uint8_t variabelen. De eerste uint8_t is de afstand waarop gecheckt moet worden. De tweede uint8_t is de mode van de sensor, 1 = toilet, 2 = sanitair.
    - Returned een 1 of een 0, afhankelijk van het resultaat.

Delay:

De library Delay regelt hoe lang een delay moet duren

Functies:

- + delay_init(void): void
  - Initialiseert de delay.
  - Heeft geen parameter.
  - Returned niets (void).
- + delay_us(uint32_t microseconds): void
  - Met deze functie kun je een aantal microseconden wachten.
  - Heeft als parameter een uint32_t, waaraan je het aantal microseconden kunt meegeven hoe lang je wilt wachten.
  - Returned niets (void).
- + delay_ms(uint32_t milliseconds): void
  - Met deze functie kun je een aantal milliseconden wachten.
  - Heeft als parameter een uint32_t, waaraan je het aantal milliseconden kunt meegeven hoe lang je wilt wachten.
  - Returned niets (void).

Usart

De library Usart kun je gebruiken voor seriele communicatie tussen twee apparaten. Deze library heeft een struct genaamd: SimpleUSART, deze struct heeft een aantal functies.

Functies:

- + *USART_init(struct SimpleUSART *): void
  - Initialiseert de USART
  - Heeft als parameter de data van de USART.
  - returned niets (void).
- + *USART_putc(char, struct SimpleUSART *): void
  - Stuur een character naar het seriële apparaat.
  - Heeft als parameter een char en de data van de USART. De char bevat de character die verstuurd moet worden verstuurd naar het seriële apparaat.
  - returned niets (void).
- + *USART_getc(struct SimpleUSART *): char
  - Ontvang een character van het seriële apparaat.
  - Heeft als parameter de data van de USART.
  - Returned de character die ontvangen wordt.
- + *USART_putstr(char*, struct SimpleUSART *): void
  - Stuur een string naar het seriële apparaat.
  - Heeft als parameter een char pointer en de data van de USART. De char pointer bevat de string die naar het seriële apparaat moet worden verstuurd.
  - returned niets (void).
- + *USART_getstr(char*, uint32_t timeout, struct SimpleUSART *): char*
  - Ontvang een string van het seriële apparaat.
  - Heeft als parameter een char pointer een uint32_t en de data van de USART. De char pointer bevat de buffer waar de string in gezet kan worden, de uint32_t bevat een timeout waarde waarna de buffer returned wordt.
  - Returned een buffer met de string.

memman

De library memman kan geheugen lezen, kijken of de ruimte vrij is en eventueel ook ruimte vrij maken.

Functies:

- + MemMan_GetFreeMemory(void): uint32_t
  - Vraag de hoeveelheid van het geheugen op dat nog leeg is.
  - Heeft geen parameters.
  - Returned het aantal bytes geheugen dat nog leeg is.
- + MemMan_GetFreeBlock(uint32_t size): uint32_t
  - Vraag een blok geheugen op, dat het aantal bytes aan leeg geheugen heeft.
  - Heeft als parameters een uint32_t, dit is het aantal bytes dat je nodig hebt voor jouw data.
  - Returend het startadres.
- + MemMan_FreeBlock(uint32_t startaddress, uint32_t length): void
  - Verwijdert data uit het geheugen
  - Heeft als parameter twee uint32_t variabelen. De eerste uint32_t is het startadres, op welk adres in het geheugen begonnen moet worden met leegmaken. De tweede uint32_t is het aantal bytes dat je wilt verwijderen.

ble

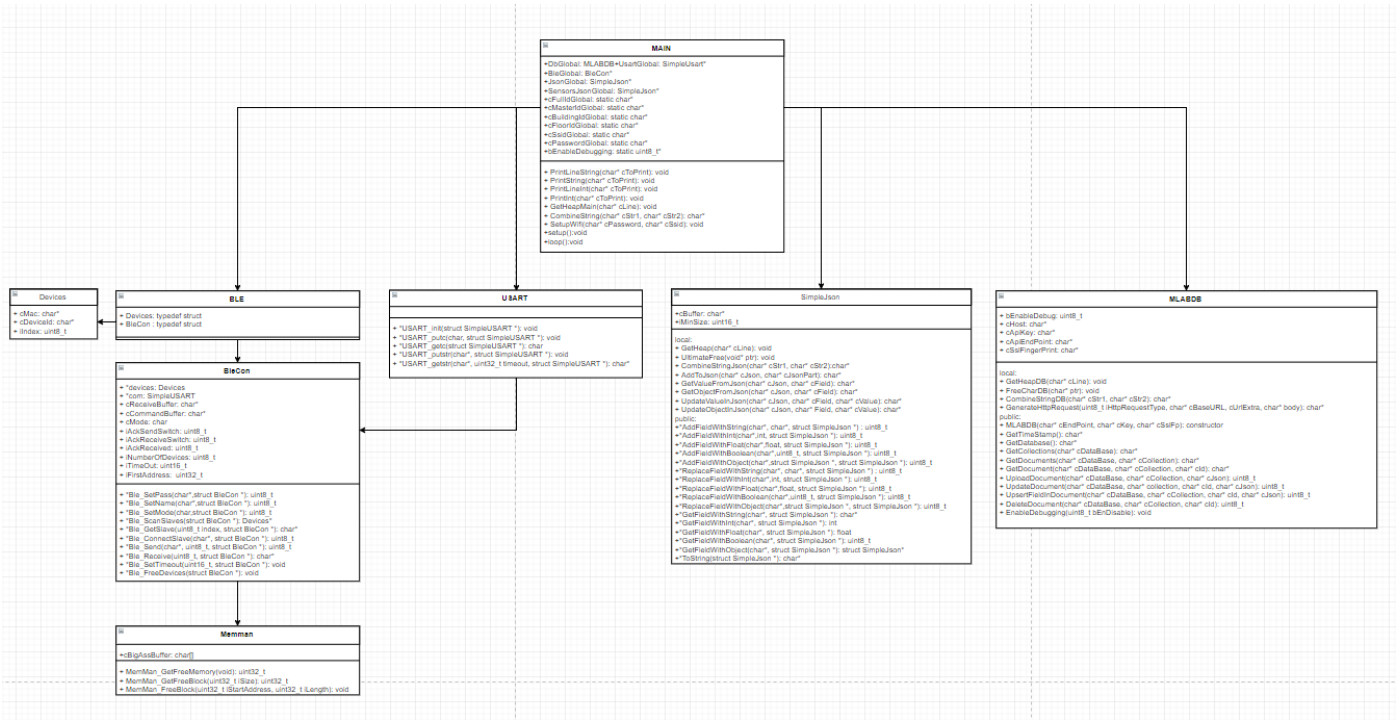
De ble library zorgt voor een bluetooth connectie en communicatie tussen apparaten.
De library ble heeft 2 structs: Devices en BleCon.
De struct Devices heeft drie variabelen: mac, deviceid en index.
De variabele cMac bevat het adres waar de gegevens van het apparaat staan opgeslagen.
cDeviceId is het ID die aan het device is gegeven.
In de iIndex variabele wordt een counter opgeslagen, na elk device wordt deze counter opgeteld. Dus bij device1 is index 0 enz.
De struct BleCon heeft een aantal functies.

Functies:

- + *Ble_Init(struct BleCon *): uint8_t
  - Initialiseert de ble module, de GPIO pinnen worden gedefinieerd voor de voeding en programmeren.
  - Heeft als parameter de data van de BLE.
  - Returned een 1.
- + *Ble_ON(struct BleCon *): uint8_t
  - Zet de voeding pin aan.
  - Heeft als parameter de data van de BLE.
  - Returned een 1.
- + *Ble_OFF(struct BleCon *): uint8_t
  - Zet de voeding pin uit.
  - Heeft als parameter de data van de BLE.
  - Returned een 1.
- + *Ble_EnableProgramMode(struct BleCon *): uint8_t
  - Zet de programmeer pin aan.
  - Heeft als parameter de data van de BLE.
  - Returned een 1,
- + *Ble_DisableProgramMode(struct BleCon *): uint8_t
  - Zet de programmeer pin uit.
  - Heeft als parameter de data van de BLE.
  - Returned een 1.
- + *Ble_SetPass(char*,struct BleCon *): uint8_t
  - Stuurt password command met nieuwe wachtwoord naar de BLE module
  - Heeft als parameter een char pointer en de BLE data. De char pointer bevat het nieuwe wachtwoord.
  - Returned 1 of 0, afhankelijk van het resultaat.

- + *Ble_SetName(char*,struct BleCon *): uint8_t
    - Stuurt naam command met nieuwe naam naar de BLE module
    - Heeft als parameter een char pointer en de data van de BLE. De char pointer bevat de nieuwe naam van de BLE module
    - Returned 1 of 0, afhankelijk van het resultaat.
- + *Ble_SetMode(char,struct BleCon *): uint8_t
    - Stuurt mode command met daarbij de nieuwe mode naar de BLE module.
    - Heeft als parameter een char en de data van de BLE. De char bevat de nieuwe mode.
    - Returned 1 of 0, afhankelijk van het resultaat.
- + *Ble_ScanSlaves(struct BleCon *): Devices*
    - Scant voor apparaten.
    - Heeft als parameter de data van de BLE.
    - Returned de devices die gevonden zijn.
- + *Ble_GetSlave(uint8_t index, struct BleCon *): char*
    - Vraagt het mac adres op van het apparaat waarmee verbonden moet worden.
    - Heeft als parameter een uint8_t en de data van de BLE. De uint8_t bevat de index waarmee een connectie geprobeerd wordt te maken
    - Returned een char pointer met het mac adres.
- + *Ble_ConnectSlave(char*, struct BleCon *): uint8_t
    - Maakt een connectie met het apparaat met het mac adres die is doorgegeven.
    - Heeft als parameter een char pointer en de dara van de BLE. de char pointer bevat het mac adres van het apparaat waarmee een connectie gemaakt moet worden.
    - Returned een 1.
- + *Ble_Send(char*, uint8_t, struct BleCon *): uint8_t
    - Stuur een string naar de BLE module
    - Heeft als parameter een char pointer een uint8_t en de data van de BLE. De char pointer bevat de string die naar de BLE module moet worden verstuurd. De uint8_t bevat een 1 of een 0, 1 betekent dat het een ack wil, een 0 betekent dat een ack niet belangrijk is.
    - Returned altijd 1
- + *Ble_Receive(uint8_t, struct BleCon *): char*
    - Ontvang een string van de BLE module
    - Heeft als parameter ene uint8_t en de data van de BLE. De uint8_t bevat een 1 of een 0, 1 wil zeggen dat er een ack moet worden verstuurd als er iets ontvangen wordt.
    - Returned de receive buffer.
- + *Ble_SetTimeout(uint16_t, struct BleCon *): void
    - Verander de timeout van de BLE module
    - Heeft als parameter een uint16_t en de data van de BLE. De uint16_t bevat de nieuwe waarde van de timeout.
    - Returned niets (void).
- + *Ble_FreeDevices(struct BleCon *): void
    - Functie om het geheugen leeg te maken met het gebruik van de memory management library
    - Heeft als parameter de data van de BLE.
    - Returned niets (void).

Master module:



(figuur 19. Klassendiagram Master na realisatie)

MAIN
Hier wordt alles bij elkaar gezet en hier bevind zich alle logica voor de master module.
Functies:

- PrintLineString, PrintString, PrintLineInt, PrintInt:
    - Deze functies zijn er om te debug gegevens te printen, in de Main zit een globale variabele dat heet bEnableDebugging, mocht deze op true (1) staan, dan printen deze functies via Serial.println of Serial.print de waardes die ze meekrijgen in hun parameters.
- GetHeapMain:
    - Print de overgebleven heap geheugen dat gebruikt kan worden, in de Main zit een globale variabele dat heet bEnableDebugging, mocht deze op true (1) staan, dan print deze functie die waarde via Serial.println en daarbij de tekst die je meegeeft in de parameter
- CombineString:
    - Deze functie combineert twee stringen waarna naar verwezen wordt door middel van twee pointers met elkaar en returned deze als een pointer naar de nieuwe gecombineerde string.

- SetupWifi:
    - Deze functie zorgt ervoor dat er verbonden wordt met Wifi Access Point waarvan de SSID via de parameters van de functie doorgegeven wordt, net zoals de password. Hij wacht oneindig totdat het hem lukt om verbinding te maken met de Wifi Access Point.
- setup:

1. Setup serial peripheral met baudrate 115200 (om te communiceren met de JDY-08)
2. Verbinden met Wifi Access Point.
3. Het ophalen van de datum en tijd.
4. De globale master json genereren waar alle informatie van de master module instaat zoals:
    a. MasterID
    b. SanitairID
    c. FloorID
    d. BuildingID
    e. Sensors (leeg)
5. Globale json genereren voor de toekomstige Sensors.
6. Het opzetten van de database communicatie.
7. De globale json te upserten in de database.
8. Dan de USART library opzetten (Alleen nodig voor BLE library)
9. De BLE library opzetten.
10. Instellen van de JDY-08 Bluetooth Module

- loop:
    1. Wacht voor een bericht van de JDY-08 Bluetooth Module.
    2. Als het bericht een JSON string bevat, zet dit om naar een SimpleJson Struct
    3. Haalt datum en tijd op.
    4. Kijkt of datum is gewijzigd in globale master json
        a. zo ja, reset local master json
    5. Haal ID van de Sensor uit de JSON string via SimpleJson Struct.
    6. Kijk of ID al in de globale JSON van de sensors zit.
        a. Zo niet
            i. Voeg een veld toe in de ontvangen sensor json de tijd en datum van het toevoegen toe.
            ii. Stop het ontvangen sensor json in de globale sensors json.
        b. Zo wel
            i. Haal de sensor gegevens json uit de globale sensors json.
            ii. Tel de teller waarde uit de ontvangen sensor json bij de teller waarde die al in de sensor gegevens json zit op.
            iii. Vervang de oude sensor gegevens met de nieuwe sensor gegevens in de globale sensors json.
    7. In geval dat de globale sensor json niet null (is geworden) is, vervang de Sensors json in de globale master json met de nieuwe sensors json.
    8. Upsert de huidige globale master json in de database samen met de datum van de upsert.

MLABDB

Dit is een in C++ geschreven library (alleen voor C++) die de communicatie met de database afhandel via de mlab api (Zie Database API).

Functies:

- Local:
  - + GetHeapDB(char* cLine): void
    - Print de vrije geheugen in de heap.
    - Heeft als parameter een char pointer nodig die verwijst naar een string
  - + FreeCharDB(char* ptr): void
    - Free'd een pointer en zet deze naar NULL
    - Heeft als parameter een char pointer nodig die verwijst naar een string
  - + CombineStringDB(char* cStr1, char* cStr2): char*
    - Combineert twee strings
    - Heeft als parameters twee char pointers die verwijzen naar een gealloceerde string nodig
    - Returned nieuwe gealloceerde char pointer die wijst naar de gecombineerde string
  - + GenerateHttpRequest(uint8_t iHttpRequestType, char* cBaseURL, cUrlExtra, char* body): char*
    - Genereert een GET,POST,PUT of DELETE request
    - Heeft als parameters een type nodig (0: GET, 1: POST, 2: PUT, 3: DELETE) als integer, een base url zonder http of www er in als char pointer die er naar verwijst, een query parameter (mag leeg zijn) en een body (mag mee zijn als char pointer.
    - Returned een nieuwe gealloceerde char pointer die wijst naar de response van de http request of een NULL als de request faalt.
- Public:
  - + MLABDB(char* cEndPoint, char* cKey, char* cSslFp): constructor
    - Constructor voor MLABDB
  - + GetTimeStamp(): char*
    - Vraagt de tijd en datum op via deze url: http://www.convert-unix-time.com/api?timestamp=now&timezone=amsterdam&format=english12
    - Returned een nieuwe gealloceerde char pointer die verwijst naar een string met daarin de datum en tijd of NULL als de http verzoek faalt.
  - + GetDatabase(): char*
    - Haalt alle databases die beschikbaar zijn op
    - Returned een nieuwe gealloceerde char pointer die verwijst naar een json array als string met daarin de beschikbare databases op, of een NULL als de http verzoek faalt.
  - + GetCollections(char* cDataBase): char*
    - Haalt alle collecties binnen een database op.
    - Heeft een char pointer as parameter die verwijst naar een string met daarin de naam van de database.
    - Returned een nieuwe gealloceerde char pointer die verwijst naar een json array als string met daarin alle beschikbare collecties binnen de database, of een NULL als de http verzoek faalt.
  - + GetDocuments(char* cDataBase, char* cCollection): char*
    - Haalt alle documenten binnen een collectie uit een database op.
    - Heeft een char pointer die verwijst naar een string met daarin de database naam en een char pointer die verwijst naar een string met de naam van de collectie nodig.
    - Returned een nieuwe gealloceerde char pointer die verwijst naar een json array als string met daarin alle beschikbare documenten binnen de database en collectie, of een NULL als de http verzoek faalt

- + GetDocument(char* cDataBase, char* cCollection, char* cId): char*
  - Haalt een document uit de collectie in een database op.
  - Heeft als parameters een char pointer nodig die verwijst naar een string met daarin de naam van de database, een char pointer die verwijst naar een string met de naam van de collectie en een char pointer die verwijst naar een string met daarin het id van het document nodig.
  - Returned een char pointer die verwijst naar een string met daarin de json van het document, of een NULL als de http verzoek faalt.
- + UploadDocument(char* cDataBase, char* cCollection, char* cJson): uint8_t
  - Upload een document naar een collectie in een database.
  - Heeft als parameters een char pointer nodig die verwijst naar een string met daarin de naam van de database, een char pointer die verwijst naar een string met de naam van de collectie en een char pointer die verwijst naar een string met daarin het volledige json document wat geupload wordt nodig.
  - Returned uint8_t met 1 als het gelukt is, een 0 als het niet gelukt is.
- + UpdateDocument(char* cDataBase, char* collection, char* cId, char* cJson): uint8_t
  - Update een document in een collectie, in een database.
  - Heeft als parameters een char pointer nodig die verwijst naar een string met daarin de naam van de database, een char pointer die verwijst naar een string met de naam van de collectie en een char pointer die verwijst naar een string met daarin het volledige json document wat geupload wordt nodig.
  - Returned uint8_t met 1 als het gelukt is, een 0 als het niet gelukt is.
- + UpsertFieldInDocument(char* cDataBase, char* cCollection, char* cId, char* cJson): uint8_t
  - Update een veld in een document, in een collectie, in een database.
  - Heeft als parameters een char pointer nodig die verwijst naar een string met daarin de naam van de database, een char pointer die verwijst naar een string met de naam van de collectie, een char pointer die verwijst naar een string met daarin het id van het document waarin wordt geupdate, en een char pointer die verwijst naar een string met daarin het volledige json document dat je wilt uploaden nodig.
  - Returned uint8_t met 1 als het gelukt is, een 0 als het niet gelukt is.
- + DeleteDocument(char* cDataBase, char* cCollection, char* cId): uint8_t
  - Verwijderd een document in een collectie
  - Heeft als parameters een char pointer nodig die verwijst naar een string met daarin de naam van de database, een char pointer die verwijst naar een string met de naam van de collectie en een char pointer die verwijst naar een string met daarin het id van het document nodig.
  - Returned uint8_t met 1 als het gelukt is, een 0 als het niet gelukt is.
- + EnableDebugging(uint8_t bEnDisable): void
  - Zet debugging aan of uit, die debug berichten worden via USART geprint.
  - Heeft als parameter een integer nodig die als boolean dient.

SimpleJson
Dit is een in C++ geschreven library met het doel dat deze ook in C zou moeten werken. Deze library handelt string operaties op een json string af, zoals toevoegen, bijwerken en ophalen. Verwijderen is met het oog op tijd nog niet toegevoegd, maar is ook niet nodig voor de werking van dit project.

Functies:

- Local:
  - + GetHeap(char* cLine): void
    - Print de vrije geheugen in de heap.
    - Heeft als parameter een char pointer nodig die verwijst naar een string
  - + UltimateFree(void* ptr): void
    - Free'd een pointer en zet deze naar NULL
    - Heeft als parameter een void pointer nodig.
  - + CombineStringJson(char* cStr1, char* cStr2):char*
    - Combineert twee strings
    - Heeft als parameters twee char pointers die verwijzen naar een gealloceerde string nodig
    - Returned nieuwe gealloceerde char pointer die wijst naar de gecombineerde string
  - + AddToJson(char* cJson, char* cJsonPart): char*
    - Voegt iets toe aan de meegegeven json string (iets omdat er niet wordt gecheckt of wat je toevoegt valid json is).
    - Heeft als parameters twee char pointers waarvan een naar een volledige Json string wijst, en de ander naar het veld en waarde die je wilt toevoegen.
    - Returned een nieuwe gealloceerde char pointer die wijst naar de nieuwe json.
  - + GetValueFromJson(char* cJson, char* cField): char*
    - Probeert een veld te vinden in de Json string, en returned die waarde. NB: Dit is alleen voor integers, floats, booleans and strings.
    - Heeft als parameters twee char pointers waarvan een naar de volledige Json String verwijst, de andere naar het veld in de Json string.
    - Returned een waarde in het veld als een chat pointer. Returned NULL als het veld niet bestaat of als er een error ontstaat.
  - + GetObjectFromJson(char* cJson, char* cField): char*
    - Probeert een veld te vinden in de Json string, en returned die waarde. NB: Dit is alleen voor objecten.
    - Heeft als parameters twee char pointers waarvan een naar de Json string verwijst, de andere naar het veld in de Json string.
    - Returned de waarde van het veld als een char pointer, Returned NULL als het veld niet bestaat of als er een error ontstaat.
  - + UpdateValueInJson(char* cJson, char* cField, char* cValue): char*
    - Zoekt voor een veld in de Json string en verandert de waarde naar de waarde die meegegeven is. NB: Dit is alleen voor integers, floats, booleans and strings.
    - Heeft als parameters drie char pointers, waarvan de eerste naar de Json string verwijst, de tweede naar het veld die geupdate moet worden, de derde bevat de waarde die naar het veld geschreven moet worden.
    - Returned de nieuwe Json string als een char pointer. Returned NULL als het veld niet gevonden kan worden of als er een error ontstaat.
  - + UpdateObjectInJson(char* cJson, char* Field, char* cValue): char*
    - Zoekt voor een veld in de Json string en verandert de waarde naar de waarde die meegegeven is. NB: Dit is alleen voor objecten.
    - Heeft als parameters drie char pointers, waarvan de eerste naar de Json string verwijst, de tweede naar het veld die geupdate moet worden, de derde bevat de waarde die naar het veld geschreven moet worden.
    - Returned de nieuwe Json string als een char pointer. Returned NULL als het veld niet gevonden kan worden of als er een error ontstaat.

- Public:
    - +*AddFieldWithString(char*, char*, struct SimpleJson *) : uint8_t
        - Stelt alle data in die nodig is om een string veld toe te voegen aan de Json string.
        - Heeft als parameter twee char pointers en een SimpleJson pointer. De eerste char pointer bevat het veld waar het aan toegevoegd moet worden. De tweede char pointer bevat de waarde die toegevoegd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft.
        - Returned 1 bij succes en 0 bij een error.
    - +*AddFieldWithInt(char*,int, struct SimpleJson *): uint8_t
        - Stelt alle data in die nodig is om een integer veld toe te voegen aan de Json string.
        - Heeft als parameter een char pointer, een int, en een SimpleJson pointer. De char pointer bevat het veld waar het aan toegevoegd moet worden. De int bevat de waarde die toegevoegd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.
    - +*AddFieldWithFloat(char*,float, struct SimpleJson *): uint8_t
        - Stelt alle data in die nodig is om een float veld toe te voegen aan de Json string.
        - Heeft als parameter een char pointer, een float, en een SimpleJson pointer. De char pointer bevat het veld waar het aan toegevoegd moet worden. De float bevat de waarde die toegevoegd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.
    - +*AddFieldWithBoolean(char*,uint8_t, struct SimpleJson *): uint8_t
        - Stelt alle data in die nodig is om een boolean veld toe te voegen aan de Json string.
        - Heeft als parameter een char pointer, een uint8_t, en een SimpleJson pointer. De char pointer bevat het veld waar het aan toegevoegd moet worden. De uint8_t bevat de waarde die toegevoegd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.
    - +*AddFieldWithObject(char*,struct SimpleJson *, struct SimpleJson *): uint8_t
        - Stelt alle data in die nodig is om een object veld toe te voegen aan de Json string.
        - Heeft als parameter een char pointer en twee SimpleJson pointers. De char pointer bevat het veld waar het aan toegevoegd moet worden. De eerste SimpleJson pointer bevat de waarde die toegevoegd moet worden. De tweede SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.
    - +*ReplaceFieldWithString(char*, char*, struct SimpleJson *) : uint8_t
        - Stelt alle data in die nodig is om een string veld te vervangen van de Json string.
        - Heeft als parameter twee char pointers en een SimpleJson pointer. De eerste char pointer bevat het veld waarvan de waarde vervangen moet worden. De tweede char pointer bevat de waarde waarnaar het vervangen moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.
    - +*ReplaceFieldWithInt(char*,int, struct SimpleJson *): uint8_t
        - Stelt alle data in die nodig is om een integer veld te vervangen van de Json string.
        - Heeft als parameter een char pointer, een int en een SimpleJson pointer. De char pointer bevat het veld waarvan de waarde vervangen moet worden. De int bevat de waarde waarnaar het vervangen moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.
    - +*ReplaceFieldWithFloat(char*,float, struct SimpleJson *): uint8_t
        - Stelt alle data in die nodig is om een float veld te vervangen van de Json string.
        - Heeft als parameter een char pointer, een float en een SimpleJson pointer. De char pointer bevat het veld waarvan de waarde vervangen moet worden. De float bevat de waarde waarnaar het vervangen moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
        - Returned 1 bij succes en 0 bij een error.

- +*ReplaceFieldWithBoolean(char*,uint8_t, struct SimpleJson *): uint8_t
  - Stelt alle data in die nodig is om een boolean veld te vervangen van de Json string.
  - Heeft als parameter een char pointer, een uint8_t en een SimpleJson pointer. De char pointer bevat het veld waarvan de waarde vervangen moet worden. De uint8_t bevat de waarde waarnaar het vervangen moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*ReplaceFieldWithObject(char*,struct SimpleJson *, struct SimpleJson *): uint8_t
  - Stelt alle data in die nodig is om een object veld te vervangen van de Json string.
  - Heeft als parameter een char pointer en twee SimpleJson pointers. De char pointer bevat het veld waarvan de waarde vervangen moet worden. De eerste SimpleJson pointer bevat de waarde waarnaar het vervangen moet worden. De tweede SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*GetFieldWithString(char*, struct SimpleJson *): char*
  - Stelt alle data in die nodig is om een string veld op te vragen van de Json string.
  - Heeft als parameter een char pointer en een SimpleJson pointer. De char pointer bevat het veld die opgevraagd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*GetFieldWithInt(char*, struct SimpleJson *): int
  - Stelt alle data in die nodig is om een integer veld op te vragen van de Json string.
  - Heeft als parameter een char pointer en een SimpleJson pointer. De char pointer bevat het veld die opgevraagd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*GetFieldWithFloat(char*, struct SimpleJson *): float
  - Stelt alle data in die nodig is om een float veld op te vragen van de Json string.
  - Heeft als parameter een char pointer en een SimpleJson pointer. De char pointer bevat het veld die opgevraagd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*GetFieldWithBoolean(char*, struct SimpleJson *): uint8_t
  - Stelt alle data in die nodig is om een boolean veld op te vragen van de Json string.
  - Heeft als parameter een char pointer en een SimpleJson pointer. De char pointer bevat het veld die opgevraagd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*GetFieldWithObject(char*, struct SimpleJson *): struct SimpleJson*
  - Stelt alle data in die nodig is om een object veld op te vragen van de Json string.
  - Heeft als parameter een char pointer en een SimpleJson pointer. De char pointer bevat het veld die opgevraagd moet worden. De SimpleJson pointer bevat de correcte Json struct waar de huidige Json verblijft
  - Returned 1 bij succes en 0 bij een error.
- +*ToString(struct SimpleJson *): char*
  - returned de hele Json string
  - Heeft als parameter een SimpleJson pointer, deze bevat de correcte Json struct waar de huidige Json verblijft
  - Returned de hele Json als een char pointer. Returned NULL als er een error ontstaat.

USART

Dit is een in C geschreven library die geport is naar C++ voor de ESP8266.

Deze library wordt alleen gebruikt om de BLE library werkend te krijgen, bevat functies zoals het verkrijgen en verzenden characters en strings.

Functies (allemaal public):

- + *USART_init(struct SimpleUSART *): void
  - Initialiseert de USART
  - Heeft als parameter de data van de USART.
  - returned niets (void).
- + *USART_putc(char, struct SimpleUSART *): void
  - Stuur een character naar het seriële apparaat.
  - Heeft als parameter een char en de data van de USART. De char bevat de character die verstuurd moet worden verstuurd naar het seriële apparaat.
  - returned niets (void).
- + *USART_getc(struct SimpleUSART *): char
  - Ontvang een character van het seriële apparaat.
  - Heeft als parameter de data van de USART.
  - Returned de character die ontvangen wordt.
- + *USART_putstr(char*, struct SimpleUSART *): void
  - Stuur een string naar het seriële apparaat.
  - Heeft als parameter een char pointer en de data van de USART. De char pointer bevat de string die naar het seriële apparaat moet worden verstuurd.
  - returned niets (void).
- + *USART_getstr(char*, uint32_t timeout, struct SimpleUSART *): char*
  - Ontvang een string van het seriële apparaat.
  - Heeft als parameter een char pointer een uint32_t en de data van de USART. De char pointer bevat de buffer waar de string in gezet kan worden, de uint32_t bevat een timeout waarde waarna de buffer returned wordt.
  - Returned een buffer met de string.

BLE

Dit is een in C geschreven library die geport is naar C++ voor de ESP8266.

Deze library wordt gebruikt om de JDY-08 Bluetooth Module in te stellen en om er mee te communiceren.

Functies (allemaal public):

- + *Ble_SetPass(char*,struct BleCon *): uint8_t
  - Stuurt password command met nieuwe wachtwoord naar de BLE module
  - Heeft als parameter een char pointer en de BLE data. De char pointer bevat het nieuwe wachtwoord.
  - Returned 1 of 0, afhankelijk van het resultaat.
- + *Ble_SetName(char*,struct BleCon *): uint8_t
  - Stuurt naam command met nieuwe naam naar de BLE module
  - Heeft als parameter een char pointer en de data van de BLE. De char pointer bevat de nieuwe naam van de BLE module
  - Returned 1 of 0, afhankelijk van het resultaat.
- + *Ble_SetMode(char,struct BleCon *): uint8_t
  - Stuurt mode command met daarbij de nieuwe mode naar de BLE module.
  - Heeft als parameter een char en de data van de BLE. De char bevat de nieuwe mode.
  - Returned 1 of 0, afhankelijk van het resultaat.
- + *Ble_ScanSlaves(struct BleCon *): Devices*
  - Scant voor apparaten.
  - Heeft als parameter de data van de BLE.
  - Returned de devices die gevonden zijn.

- + *Ble_GetSlave(uint8_t index, struct BleCon *): char*
    - Vraagt het mac adres op van het apparaat waarmee verbonden moet worden.
    - Heeft als parameter een uint8_t en de data van de BLE. De uint8_t bevat de index waarmee een connectie geprobeerd wordt te maken
    - Returned een char pointer met het mac adres.
- + *Ble_ConnectSlave(char*, struct BleCon *): uint8_t
    - Maakt een connectie met het apparaat met het mac adres die is doorgegeven.
    - Heeft als parameter een char pointer en de dara van de BLE. de char pointer bevat het mac adres van het apparaat waarmee een connectie gemaakt moet worden.
    - Returned een 1.
- + *Ble_Send(char*, uint8_t, struct BleCon *): uint8_t
    - Stuur een string naar de BLE module
    - Heeft als parameter een char pointer een uint8_t en de data van de BLE. De char pointer bevat de string die naar de BLE module moet worden verstuurd. De uint8_t bevat een 1 of een 0, 1 betekent dat het een ack wil, een 0 betekent dat een ack niet belangrijk is.
    - Returned altijd 1
- + *Ble_Receive(uint8_t, struct BleCon *): char*
    - Ontvang een string van de BLE module
    - Heeft als parameter ene uint8_t en de data van de BLE. De uint8_t bevat een 1 of een 0, 1 wil zeggen dat er een ack moet worden verstuurd als er iets ontvangen wordt.
    - Returned de receive buffer.
- + *Ble_SetTimeout(uint16_t, struct BleCon *): void
    - Verander de timeout van de BLE module
    - Heeft als parameter een uint16_t en de data van de BLE. De uint16_t bevat de nieuwe waarde van de timeout.
    - Returned niets (void).
- + *Ble_FreeDevices(struct BleCon *): void
    - Functie om het geheugen leeg te maken met het gebruik van de memory management library
    - Heeft als parameter de data van de BLE.
    - Returned niets (void).

Security

Inleiding

Gezien dit netwerk een IoT oplossing is met veel kleine apparaten die via het netwerk verbonden is, is het van belang om na te gaan denken hoe je de gegevens die ze uitwisselen gaat beschermen.

Scope

Vanuit de opdrachtgever is het niet van uiterst belang dat de gegevens beveiligd worden. Wel is het handig dat deze gegevens niet aangepast kan worden door derden. Ook, om toekomstige lolbroeken die de IoT apparaten wil gebruiken voor bijvoorbeeld een DDOS Botnet, is het verstandig om de communicatie tussen de server en de IoT apparaten te beveiligen.

Gezien dit project uitgevoerd wordt binnen een relatief kleine periode zal eerst alles op een simpele onbeveiligde manier in werking gesteld worden. Er wordt wel gewerkt aan de optie om dit te beveiligen, maar of er tijd overblijft voor de implementatie daarvan hangt af van hoe het project verloopt.

De verbindingen

Alle verbindingen naar buiten vanaf de mastermodule zal om overname van bijvoorbeeld de IoT apparaten te voorkomen zo goed mogelijk beveiligd moeten worden. Daar zijn meerdere mogelijkheden voor. De eerste en meest effectieve is om de verbinding tussen de mastermodule en de server te versleutelen.

Een SSL certificate moet opgevraagd worden via bijvoorbeeld https://letsencrypt.org/. Vervolgens moet deze geïmplementeerd worden aan de server kant door middel van OpenSSL of WolfSSL (PicoTCP):
https://github.com/openssl/openssl
https://www.wolfssl.com/c-wrapper-for-wolfssl/

Implementatie van SSL op de microcontroller(s) zou via PicoTCP (WolfSSL) gebeuren:
http://www.picotcp.com/picotcp-and-cyassl/

De gegevens

Ook zou je de gegevens zelf kunnen versleutelen via een unieke sleutel die alleen de programmeurs van het systeem weten, deze zou je dan op een bepaalde manier hardcoded in de code kunnen zetten (mogelijk meerdere sleutels).

Om te voorkomen dat de code te herleiden valt, zal bij het publiekelijke uitbrengen van de code gezorgd worden dat de sleutel niet terug te vinden is in de code, en is het de bedoeling dat de programmeur die de software wil hergebruiken voor zijn eigen doeleinden zijn eigen sleutel implementeren.

Verder is het nodig om te gaan onderzoeken hoe we de gecompileerde code zo kunnen maken dat deze niet gedecompileerd kan worden. Dit geldt voor de server applicatie en microcontroller.