design your future

Hoe onderbouwd irrigeren

door gebruik van Internet of Things

studiegebied industriële wetenschappen en technologie

bachelor in de elektronica-ICT

campus Kortrijk

Decoster Sam

Vandewalle Wouter

academiejaar 2016-2017

vives katholieke hogeschool
associatie KU Leuven

design your future

Hoe onderbouwd irrigeren

door gebruik van Internet of Things

studiegebied industriële wetenschappen en technologie

bachelor in de elektronica-ICT

campus Kortrijk

Decoster Sam

Vandewalle Wouter

academiejaar 2016-2017

vives katholieke hogeschool
associatie KU Leuven

design your future

How to motivate irrigation decisions with Internet of Things

studiegebied industriële wetenschappen en technologie

bachelor in de elektronica-ICT

campus Kortrijk

Decoster Sam

Vandewalle Wouter

academiejaar 2016-2017

vives katholieke hogeschool
associatie KU Leuven

Woord vooraf

Wij, Decoster Sam en Vandewalle Wouter, zijn twee laatstejaarsstudenten in de bachelor richting Elektronica-ICT met als afstudeerrichting ICT die op zoek waren naar een stageplaats die ons een uitdagend project kon geven omtrent de “Internet of Things” en het bouwen van een webapplicatie hierrond. Na wat bezoeken zijn we bij Korazon te Rumbeke terechtgekomen die een voorstel hadden waar “Internet of Things” een rol speelde en waar er een webapplicatie werd verwacht. Dit sprak ons natuurlijk meteen aan en zonder veel te twijfelen beslisten we om stage te lopen bij Korazon.

De opdracht was echter veranderd tijdens de grote vakantie en er werd beslist om voor het bedrijf Inagro een applicatie uit te werken waar dezelfde vereisten gevraagd werden. We dienden dus al onze kennis en vaardigheden te gebruiken om zo goed mogelijk het einddoel te bereiken die Korazon en Inagro samen vooropstelden.

Deze stage ging echter niet mogelijk gemaakt kunnen zijn, ware het niet voor deze personen die we hier even willen bedanken. Eerst en vooral willen we dhr. Carlier Jurgen, zaakvoerder van Korazon en onze mentor dhr. Lefebure Kevin bedanken voor de kans die ze ons gaven om deze stage bij Korazon te kunnen uitvoeren. Wij bedanken ook dhr. Billiet Yves, dhr. Thoré Steven en dhr. Catteeuw Wim, bij Inagro, die ons telkens antwoord gaven op al onze vragen omtrent het project. Wij bedanken ook alle medewerkers van Korazon, die zorgden voor een goede werksfeer en een leuke samenwerking, maar ook telkens ons konden helpen indien we vragen hadden. Als laatste willen wij ook nog onze mentor van school, dhr. Lefebvre Sam, bedanken voor het opvolgen van onze stage en voor de tips die we gekregen hebben om deze stage tot een goed einde te kunnen brengen.

Decoster Sam, Vandewalle Wouter

Rumbeke, juni 2017

Abstract

This internship/thesis was aimed at developing a web application which uses the Internet of Things to help farmers with deciding how much irrigation their crop needs. This web application is needed for farmers in Flanders because of the water scarcity that is more and more becoming an issue. The farmer thus needs to be more careful on the way he handles the water that he uses to irrigate his crops. This means that the farmer no longer can depend on his “feel of drought” and needs to have data to support his choice of irrigation on a certain crop. This data can be achieved by using different databases and open weather data, this data can have different weather readings and even forecasts for rain and such, which then can be used to furthermore support the decision the farmer made. This internship/thesis is a proof of concept, which means that it is in no way a finished product but shows what the web application could look like when it is used by farmers in Flanders.

The aim of this paper is to show the process we went through for creating this web application and will show all aspects that the web application consists of. This paper gives an impression on how we did research to try and build such a web application. There is also a lot of code in this paper that will show, in more detail, what is happening in the background of the web application when the farmer makes a certain request on the application. The research and coding of the web application happened at the same time, but this paper will sketch a logical buildup of the web application, with research and coding separated from each other.

The finished web application at the end of our internship has all the necessary attributes to give the farmer all the information he needs to make a good decision on the irrigation of his crop. This means that the farmer has access to open weather data, but he can also place his own sensors on his parcel to do local measurements and view these on the web application as well. There is also a possibility for the farmer to enter his irrigations in the web application, this furthermore helps the farmer decide if he needs to irrigate his crop. An example can be that the farmer doesn’t need to irrigate his crop when he irrigated it two days ago. For the future of this web application there needs to be some kind of model, which the farmer can use to make the decision for him. This model then takes in account all the variables that are in play when making this decision. But this is beyond this thesis and requires a whole internship devoted to this single model to successfully implement this. Nevertheless, we did a little research as to how this could be achieved and discussed this in this paper, however this research was done to sketch an image on the possibilities of such a model and is in no way a definite solution to the problem.

Auteurs: Decoster Sam & Vandewalle Wouter, june 2017

Inhoudsopgave

Woord vooraf

Abstract

Inhoudsopgave 3

Inleiding 7

1 Voorsteling van de bedrijven 9

1.1 Korazon 9

1.1.1 Achtergrond 9

1.1.2 Webwinkels 9

1.1.3 Web toepassingen 9

1.2 Inagro 10

1.2.1 Achtergrond 10

1.2.2 Missie en doelstellingen 10

1.2.3 Projecten 10

2 Business case 11

2.1 Beschrijving business case 11

2.1.1 Fase één 12

2.1.2 Fase twee 12

2.1.3 Fase drie 13

2.1.4 Fase vier 13

2.1.5 Fase vijf 13

2.2 Wireframe 14

2.2.1 Layout 14

2.2.2 Opmerkingen 15

3 Internet of Things 16

3.1 Hoofdblokken Internet of Things netwerk 16

3.1.1 Data capteren 17

3.1.2 Communicatie 18

3.1.3 Platform 18

3.1.4 Data-analyse 18

3.1.5 Visualisatie 18

4 Huidig sensornet 19

4.1 Blokschema 19

4.2 Algemene info 20

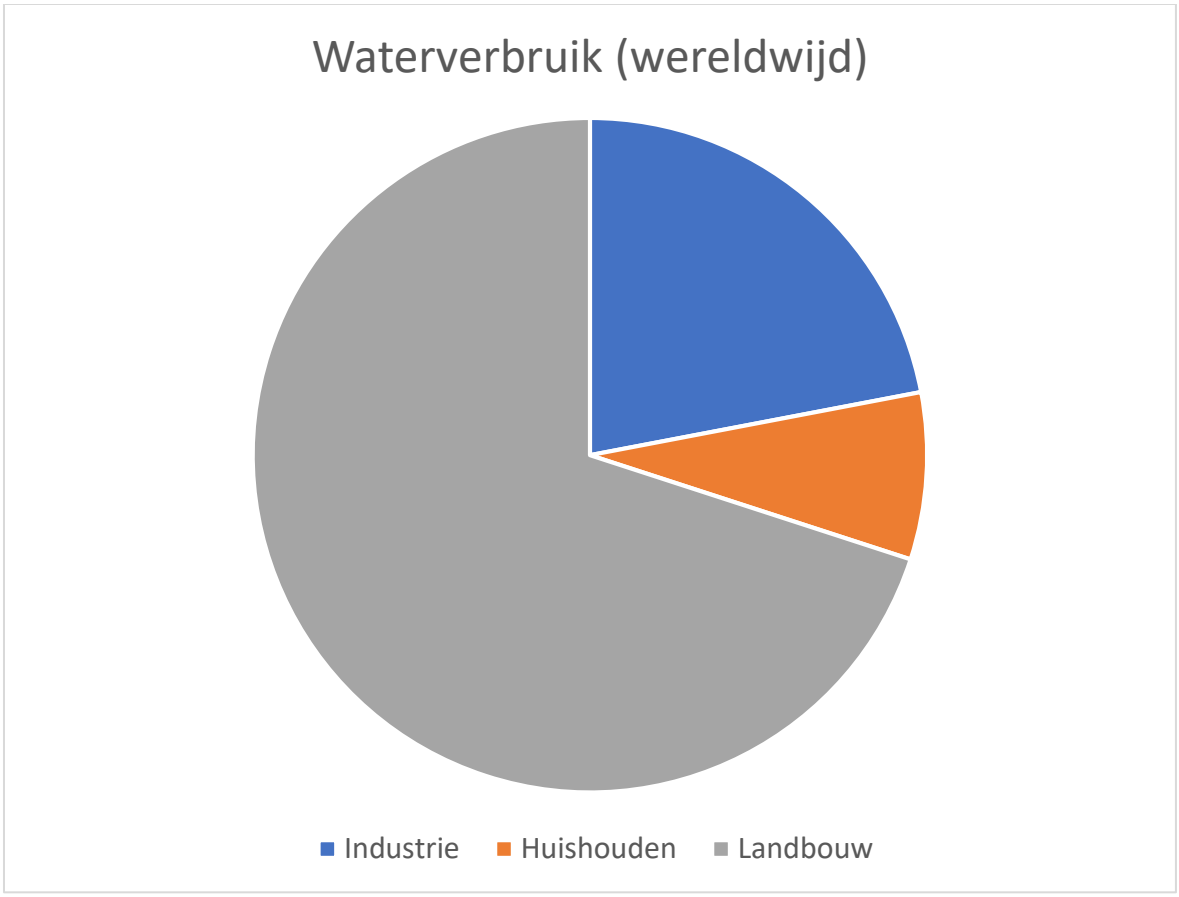
4.3 Onderdelen in het sensornet 21
4.3.1 Sensors en sensorgroepen 21
4.3.2 Server/platform 22
4.3.3 Gebruikersinterface 23
4.4 Dataformaat 24
4.4.1 Getal representatie 24
4.4.2 Tijd representatie 24
4.4.3 Opbouw van een bericht 25
5 Azure Cloud platform 26
5.1 Cloud services 27
5.1.1 Web role 27
5.1.2 Worker role 27
5.1.3 Communicatie tussen verschillende rollen 27
5.2 Opslagaccount 28
5.3 SQL-database en server 28
5.4 Blokschema 29
6 Webapplicatie 30
6.1 ASP.NET & MVC 30
6.1.1 ASP.NET 30
6.1.2 MVC 31
6.2 Entity Framework 32
6.2.1 Installatie 32
6.2.2 Aanmaken datamodellen 33
6.2.3 Aanmaken Database Context 35
6.2.4 Gebruiken van Database Context 36
6.3 Inloggen via “Mijn Bedrijf” 38
6.3.1 WCF 38
6.3.2 Klasses voor inloggen 38
6.4 Locaties 41
6.5 Sensoren verkrijgen in sensornet 43
6.5.1 Klasses 43
6.5.2 Gebruik in de applicatie 46
6.6 PDF-bestanden genereren 47

7 Worker role 49
7.1 TCP Listener 49
7.1.1 Socket port 49
7.1.2 WorkerRole.cs 50
7.2 Weerdata afhalen 51
7.2.1 Yahoo Weather API 51
7.2.2 OpenWeatherMap 53
7.3 Scheduler Quartz 54
7.3.1 Quartz 54
7.3.2 Installeren van de library 54
7.3.3 Het aanmaken van een Job 55
7.3.4 Het instellen van de scheduler 56
7.3.5 Resultaat 58
7.4 Mail alarmering 59
8 Beslissingsmodel 61
8.1 Regressieanalyse 62
8.2 Regressie in C# 64
9 Testopstelling 68
9.1 DHT-sensor 68
9.2 Raspberry Pi 68
9.3 Raspbian 69
9.3.1 Linux 69
9.3.2 Python 70
9.4 Windows 10 IoT core 73
9.4.1 Installatie van Windows 10 IoT core 73
9.4.2 UWP-applicaties 75
9.4.3 Demo applicatie 75
10 Algemene besluiten 82
10.1 Keuze platform 82
10.2 Standaarden in IoT 82
10.3 Beslissingsmodel 82
10.4 Internet of Things 83

Verklarende woordenlijst 84
Bibliografie
Bijlagen
Lijst van illustraties

Inleiding

Doordat de aarde steeds een groter watertekort lijdt volgens cijfers van het milieurapport in 2013 (Peeters, B., april 2013), moet er steeds meer gekeken worden naar hoe we omgaan met het bruikbare water die we nu nog ter beschikking hebben. De grootste verbruiker van water over de hele wereld is de landbouw. De landbouw verbruikt ongeveer 70% van het bruikbare water over de hele wereld zoals te zien in onderstaande afbeelding (zie Figuur 1) (Vilt, 2012). In Vlaanderen kunnen de telers zelf bepalen wanneer ze irrigeren en wanneer niet, volgens hun “gevoel van droogte”. Hierdoor kan er niet gecontroleerd worden of de beslissing om te irrigeren correct en rechtvaardig is. Aangezien het watertekort uiteindelijk kan leiden tot waterschaarste dient er hier dus een oplossing voor gezocht te worden.



Figuur 1: Waterverbruik (wereldwijd)

Het doel van deze thesis is om deze oplossing zo goed mogelijk uit te voeren. Er wordt dan bijvoorbeeld bepaald aan de hand van de weersomstandigheden, de vooruitzichten van het weer en de huidige staat van het gewas of er al dan niet moet geïrrigeerd worden en in welke hoeveelheid.

Dit wordt verwerkt in een webapplicatie die toegankelijk is voor elke teler binnen een bepaald netwerk. Deze webapplicatie maakt gebruik van het Internet of Things. Het nut van Internet of Things zal ook in detail worden besproken in deze scriptie. De webapplicatie dient snel en gemakkelijk te zijn om te gebruiken en zorgt voor zowel een voldoende aan informatie aan de teler over de omgeving waarin zijn perceel zich in bevindt alsook een geformuleerde beschrijving van wat de best mogelijke irrigatie is voor dat perceel. De webapplicatie zal gemaakt worden in het Azure Cloud platform, hoe dit platform precies in elkaar zit en waarom er voor dit platform werd gekozen wordt ook verduidelijkt in deze scriptie.

Een opmerking die we hier dienen te maken is dat deze thesis een “Proof of concept” is, wat wil zeggen dat het een proefproject is om de mogelijkheden van deze webapplicatie te onderzoeken. Dit project is niet iets wat meteen beschikbaar kan gezet worden voor alle telers in Vlaanderen en dient dus enkel om het bedrijf waarmee we samenwerken een beeld te geven van hoe het er zou kunnen uitzien.

In het eerste hoofdstuk zullen we de bedrijven voorstellen waar we mee hebben samengewerkt in de loop van deze stage.

In hoofdstuk twee zullen we de eigenlijke probleemstelling/business case uiteenzetten en deze in meer detail bespreken.

Het derde hoofdstuk zal volledig toegewijd zijn aan het Internet of Things en alles wat je dient te weten hierover zal hierin uitgelegd worden.

In hoofdstuk vier zullen we spreken over het huidig sensornet die beschikbaar is voor de telers, dit sensornet zal gebruikt worden binnenin de webapplicatie.

In hoofdstuk vijf zullen we uitleggen wat het Azure Cloud platform is, we zullen hierin spreken over “web roles” en “worker roles”. Dit platform wordt gebruikt om onze webapplicatie te kunnen laten op draaien.

In hoofdstuk zes wordt de volledige webapplicatie uitgelegd wat zal draaien op deze zogenaamde “web roles” binnenin Azure.

Hoofdstuk zeven zal dan logischerwijs alle “worker roles” bespreken die we in het Azure Cloud platform gebruikt hebben om de webapplicatie uit te voeren.

Zoals vermeld dient er dus een beslissing gemaakt te worden omtrent het irrigeren, hoe dit precies zou werken wordt verduidelijkt in hoofdstuk acht.

Om ervoor te zorgen dat we een goede demo kunnen tonen op de presentatie van deze thesis dienen we een testopstelling te maken, dit wordt besproken in hoofdstuk negen.

In het laatste hoofdstuk, hoofdstuk tien zullen we een algemeen besluit vormen van de gehele opdracht en bespreken we of er aan de verwachtingen voldaan werd en wat er verder kan gebeuren met dit project aangezien het nog steeds een “Proof of concept” is.

1 Voorsteling van de bedrijven

1.1 Korazon

Figuur 2: Logo Korazon

1.1.1 Achtergrond

Korazon is een web development bedrijf met momenteel dertien bekwame en ervaren web professionals. Korazon is gesitueerd te Rumbeke, bestaat reeds 25 jaar en realiseerde in die periode meer dan 2000 projecten. Daar zitten grote, langlopende projecten bij, maar ook korte (campagnes) of eenmalige projecten (zoals het verzenden van nieuwsbrieven).

Korazon heeft een aparte positionering ten aanzien van internetklanten, waarbij vooral de nadruk gelegd wordt op integratie tussen websites, webwinkels en web toepassingen. Om hun klanten extra vrijheid te geven heeft Korazon ook een eigen ontwikkeld CMS genaamd Liquifi, waarmee de klanten de mogelijkheid krijgen om hun website volledig zelf te beheren. Het logo van Korazon is in Figuur 2 hierboven zichtbaar.

1.1.2 Webwinkels

Omdat klanten vandaag online aankopen willen doen, waar en wanneer ze willen, maakt Korazon webwinkels responsive zodat de webwinkel perfect werkt op desktop, tablet en smartphone. Het is belangrijk dat de klanten snel vinden wat ze zoeken en dat ze eenvoudig hun aankopen kunnen afronden. Daarom besteedt Korazon veel aandacht aan interface design.

Dit stopt niet bij het technisch realiseren van een mooi resultaat, maar er wordt hier uiteraard verder gekeken naar SEO en internetmarketing.

Een goede webshop is volledig geïntegreerd in het verkoopproces van de organisatie. Dit doet Korazon door de webshop te koppelen aan het bedrijfsbeheersysteem en door de e-commerce value chain er naadloos in te verwerken.

1.1.3 Web toepassingen

Met behulp van een projectmatige aanpak van A tot Z bouwt Korazon webapplicaties die volledig zijn afgestemd op de werking van het bedrijf van hun klanten, al dan niet gekoppeld aan de reeds aanwezige bedrijfsbeheersystemen. Hierin werken specialisten in interface design die ervoor zorgen dat de gebruikers vlot met het systeem kunnen werken.

1.2 Inagro

Figuur 3: Logo Inagro

1.2.1 Achtergrond

Inagro begon als een onderzoeks –en voorlichtingscentrum voor nijverheidsteelten in 1956. Het bedrijf werd opgericht met als doel een provinciale onderzoeksinstelling op te richten waarbij de technische problemen met nijverheidsgewassen bestudeerd werden. Op 26 mei 1956 kwam de oprichting hiervan door middel van een Koninklijk Besluit. De eerste directeur van het onderzoekscentrum was Lucien Bockstaele en de eerste onderzoeker werd aangeworven. Met zijn tweeën traden ze op 1 november 1956 in dienst. Bij de start van dit centrum werd het ingericht als een wetenschappelijke instelling. In 1970 werd dan een champignonproefbedrijf opgericht, waarna alles plots sneller evolueerde. Er werden meerdere serres en labo's gebouwd om een zo groot mogelijke variëteit aan te bieden. Dit hele verhaal leidt tot het Inagro dat vandaag de dag gevestigd is in Rumbeke-Beitem Ieperseweg 87. Het logo van Inagro vandaag is in bovenstaande afbeelding (zie Figuur 3) zichtbaar.

1.2.2 Missie en doelstellingen

De missie van Inagro is het leveren van het juiste advies aan de land –en tuinbouwsector met aandacht voor economie, ecologie en de samenleving. Inagro onderneemt duurzaam, netwerkt en ambieert een uitstekende reputatie.

Inagro staat voor praktijkgericht wetenschappelijk onderzoek en het geven van advies en professionele oplossingen. Er worden analyses gemaakt op water, plant, bodem en mest. Inagro staat ook nog voor innovatie, de zorg voor milieu en landschap en heeft ook aandacht voor de buur en de burger.

1.2.3 Projecten

Inagro heeft veel lopende projecten, deze projecten zijn dan ook de motor binnenin het bedrijf in verband met innovatie. Deze projecten stimuleren de kennisuitwisseling, de creativiteit en de samenwerking binnen in het bedrijf en ver daarbuiten. Daarom werkt Inagro ook graag mee aan deze bachelorproef rond Internet of Things, zij zien ook de mogelijkheden om de productiviteit in de landbouwsector te analyseren en vervolgens te optimaliseren.

2 Business case

2.1 Beschrijving business case

De totale business case die ons werd voorgelegd is eigenlijk een project dat niet te realiseren is in de tijd die voor ons beschikbaar is om dit project tot een goed einde af te ronden. Daarom werd de business case in fases opgedeeld. Deze fases dienen om telkens afgewerkte delen te verkrijgen zodat, wanneer de stageperiode ten einde is, er verder gewerkt kan worden aan dit project zonder dat er veel veranderd hoeft te worden. Belangrijk om op te merken is dat deze fases niet per sé in chronologische volgorde dienen te gebeuren, een fase kan individueel afgewerkt worden volgens de mogelijkheden die zich bieden op het moment zelf. Dit wordt duidelijk als we de business case zullen bekijken.

De business case die we verkregen hebben van Inagro zal eruit bestaan dat er verder gebouwd wordt op een bestaande applicatie. Deze applicatie werd gemaakt door Inagro en heeft als naam “Mijn Bedrijf”. De applicatie is een website die telers in België de mogelijkheid biedt om allerhande informatie van hun percelen te verkrijgen. Deze informatie wordt ter beschikking gesteld op de site door gebruik te maken van WCF’s. Een uitleg over hoe deze WCF’s werken staat verder in hoofdstuk 6.3.1. Deze WCF’s tonen dan de informatie die de teler wil te zien krijgen van een specifiek perceel die hij in zijn bezit heeft. Wat hij precies wil te zien krijgen van een bepaald perceel kan ingesteld worden in de applicatie Mijn Bedrijf.

Het is het doel van deze business case om een nieuw onderdeel aan te maken die meteen kan geïmplementeerd worden binnenin “mijn bedrijf”. Dit onderdeel, dat hier in een andere site nog wordt verwerkt, geeft als informatie mee hoeveel de teler mag irrigeren op zijn perceel volgens een bepaald advies. Deze webapplicatie kan eventueel later herzien worden om ook als een WCF op de applicatie Mijn Bedrijf geïmplementeerd te worden.

GlobalGap (een instantie die instaat voor controle en certificatie) heeft een opmerking gegeven aan Inagro omtrent het irrigeren van percelen. Tot op vandaag wordt het irrigeren bepaald door de telers en hun “gevoel van droogte”. Dit is echter niet meer voldoende aangezien er geen duidelijke controles kunnen uitgevoerd worden waarom er op een bepaald moment geïrrigeerd werd. Aangezien er steeds meer waterschaarste is, is dit dus een duidelijk probleem dat opgelost dient te worden. Deze toepassing, die beschikbaar zal zijn vanuit de webapplicatie Mijn Bedrijf, zou dit probleem moeten aankaarten en uiteindelijk moeten wegwerken. De applicatie produceert dan een advies aan de teler wanneer hij moet irrigeren en in welke mate, bepaald door het gewas op het perceel, de bodemvochtigheid van het perceel, de grondtemperatuur en de weervoorspellingen. Uit deze data wordt het advies opgebouwd volgens een beslissingsmodel. De data die gebruikt wordt in het beslissingsmodel kan van veel verschillende bronnen gehaald worden. Zo kunnen er zich sensoren plaatsvinden op het perceel, die dan data doorsturen naar een database, maar ook kan er data opgehaald worden van weerstations die zich in de buurt van het perceel bevinden. Natuurlijk kan er ook altijd gebruik gemaakt worden van open data geleverd door andere weersmetingen, waar er al voorspellingen in kunnen gebeuren, etc.

De moeilijkheid van deze business case ligt hem dus in het beslissingsmodel dat opgemaakt moet worden en de vele verschillende databanken die dit beslissingsmodel dient te raadplegen. Om deze business case nu even te schetsen zijn hier de verschillende fases:

1. De teler krijgt een app ter beschikking (die draait op Mijn Bedrijf of alleenstaand in dit geval) die informatie zal weergeven op basis van zijn locatie omtrent de neerslag die gevallen is en een prognose over de neerslag voor de komende dagen.

2. In fase twee is het de bedoeling dat er ook data verkregen wordt van lokale sensoren. Dit zijn sensoren die zich bevinden op het perceel zelf. Zoals bodemvochtmeters, tensiometers, etc. Eventueel kan de teler ook handmatige metingen uitvoeren.

3. Fase drie bestaat eruit om de verkregen data te correleren met de gegevens over bodemgesteldheid die beschikbaar zal zijn binnenin Inagro zelf en eventuele openbare platformen. Deze data kan bestaan uit doordringbaarheid, absorptie, helling, etc.

4. De vierde fase bestaat eruit om al deze data nu in een teeltmodel te plaatsen (dit vormt het beslissingsmodel). Het teeltmodel bepaalt de waterbehoefte van een perceel en zal dus bepalen hoeveel er precies geïrrigeerd moet worden.

5. In de vijfde fase is het dan de bedoeling om dit zoveel mogelijk te automatiseren op basis van luchtfotografie. Hieronder kunnen satellietbeelden bevinden of eventueel zelfs beelden gemaakt door een drone.

Nu we even de verschillende fases kort hebben aangekaart is het al duidelijk dat dit nooit gerealiseerd kan worden binnenin de tijdspanne die wij hebben in deze stage. Maar om toch een duidelijk beeld te scheppen over waar dit project naartoe kan gaan worden alle fases hier verder nog even in detail uitgelegd. Deze fases kunnen in de loop van het project eventueel nog gewijzigd worden of compleet herzien worden, maar dat hangt volledig af van waar de business case op dat moment staat.

2.1.1 Fase één

Fase één bestaat eruit om informatie weer te geven over de neerslag die gevallen is en een prognose te geven over de neerslag voor de komende dagen. Deze informatie wordt bepaald door de locatie waarop de teler zich momenteel bevindt. De locatie wordt bepaald via gps-bepaling op de teler zijn smartphone. Daarna wordt het dichtstbijzijnde perceel geselecteerd dat overeenstemt met deze locatie en tot slot zal de teler de neerslagegevens te zien krijgen die van tel is voor dit perceel. Wanneer de teler nu echter niet in de buurt is van zijn perceel, en er dus geen perceel kan gelinkt worden met de locatie van de teler zelf, kan de teler een perceel selecteren en zo toch weerdata verkrijgen.

2.1.2 Fase twee

In fase twee dient er data opgehaald te worden uit lokale sensoren. We kunnen natuurlijk meteen sensoren gebruiken die bij Inagro beschikbaar zijn, maar omdat er een vraag was van Inagro om nieuwe sensoren uit te testen en na te gaan of er een goedkopere, snellere en betere oplossing was voor de bestaande sensoren, hebben we er uiteindelijk voor gekozen om gebruik te maken van een Raspberry Pi die sensoren zal inlezen en deze data zal verwerken en doorsturen naar een database. De Raspberry Pi en hoe we deze gebruiken wordt in hoofdstuk 9.2 besproken.

2.1.3 Fase drie

In de derde fase zal er data, die verkregen werd door allerhande sensoren en weerstations, worden gecorreleerd met gegevens die beschikbaar zijn binnenin Inagro over de bodemgesteldheid. Maar er kunnen eventueel ook gegevens hierover opgehaald worden bij andere instanties. De bedoeling hiervan is dat er door het gebruik van deze gegevens wordt nagekeken of de gemeten data wel enige waarheid bevat. Dit is bijvoorbeeld nuttig als we willen nagaan of de sensoren nog altijd goed werken. Dit lijkt overbodig voor de sensoren die bovengronds werken, maar sensoren die de bodemvochtigheid en dergelijke meten zijn moeilijker bereikbaar en dus wordt het belangrijk om de werking van deze sensoren te controleren. Verder kan de data van de bodemgesteldheid ook gebruikt worden voor de advisering.

2.1.4 Fase vier

Uit fase vier zal het beslissingsmodel worden gevormd voor de advisering die er dient te gebeuren. Dit beslissingsmodel zal de basis vormen voor het volledig uitgewerkte beslissingsmodel. Het is logisch dat dit beslissingsmodel een simpele vorm zal hebben en zeker niet perfect en tot in de details zal uitgewerkt worden in de eerste fases van dit beslissingsmodel. Het is de bedoeling dat we in dit beslissingsmodel een beeld kunnen scheppen van hoe een bepaald advies kan gegeven worden en hoe dit beïnvloed wordt door de metingen en de databanken die gebruikt worden binnenin dit beslissingsmodel.

2.1.5 Fase vijf

In de vijfde en laatste fase is het de bedoeling om het volledig geheel te automatiseren. Dit kan gebeuren op basis van fotografie uit de lucht. Hieronder vallen onder andere satellietbeelden en zelfs beelden van drones die over een perceel kunnen vliegen. Door het gebruik van deze beelden kan er meteen bepaald worden of er al dan niet moet geïrrigeerd worden. Natuurlijk dient er nog steeds een adviesrapport gegenereerd te worden, maar dit zal allemaal automatisch gebeuren. Deze fase is zeer toekomstgericht maar lijkt mogelijk om te bereiken. Voordat er gewerkt wordt aan deze fase dient alles al goed samen te werken zodat er geen fouten meer kunnen optreden bij het volledig automatiseren van dit geheel project.

2.2 Wireframe

Om de business case te verduidelijken werd er door het stagebedrijf voorgesteld om een wireframe te maken van de webapplicatie. Aan de hand van een wireframe kan er ook getoetst worden of aan alle verwachtingen voldaan wordt binnen de business case.

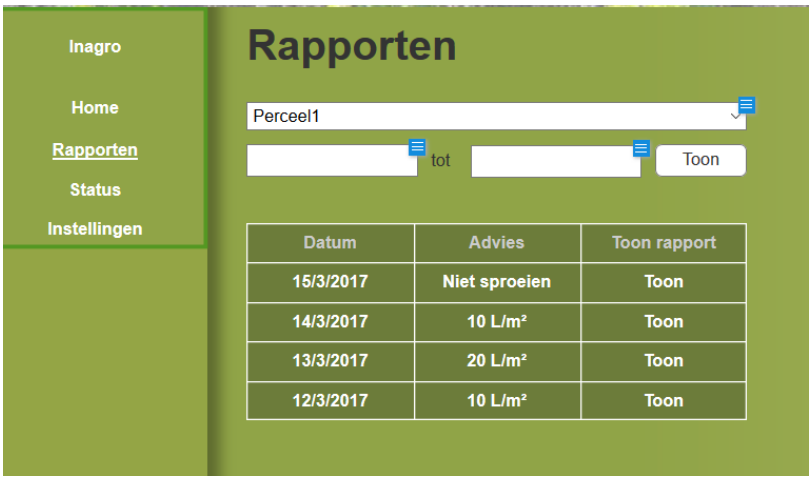
Een wireframe is een eenvoudige schematische voorstelling van de structuur of ruwbouw van een website of webapplicatie. Een wireframe illustreert de opbouw van de pagina's van een webapplicatie en hoe alle pagina's met elkaar verbonden zijn. Wireframes zijn een efficiënt hulpmiddel om in de ontwerpfase, de wensen van de klant (Inagro) in het ontwerp van de webapplicatie te verwerken, er kan dus aan de hand hiervan ook nagegaan worden of alle wensen van de klant verwerkt zitten in de ruwbouw van de website.

Het wireframe die we hebben gemaakt werd ontworpen in Axure RP8. Axure RP8 is een software tool waarmee het maken van een wireframe op een eenvoudige manier kan gerealiseerd worden.

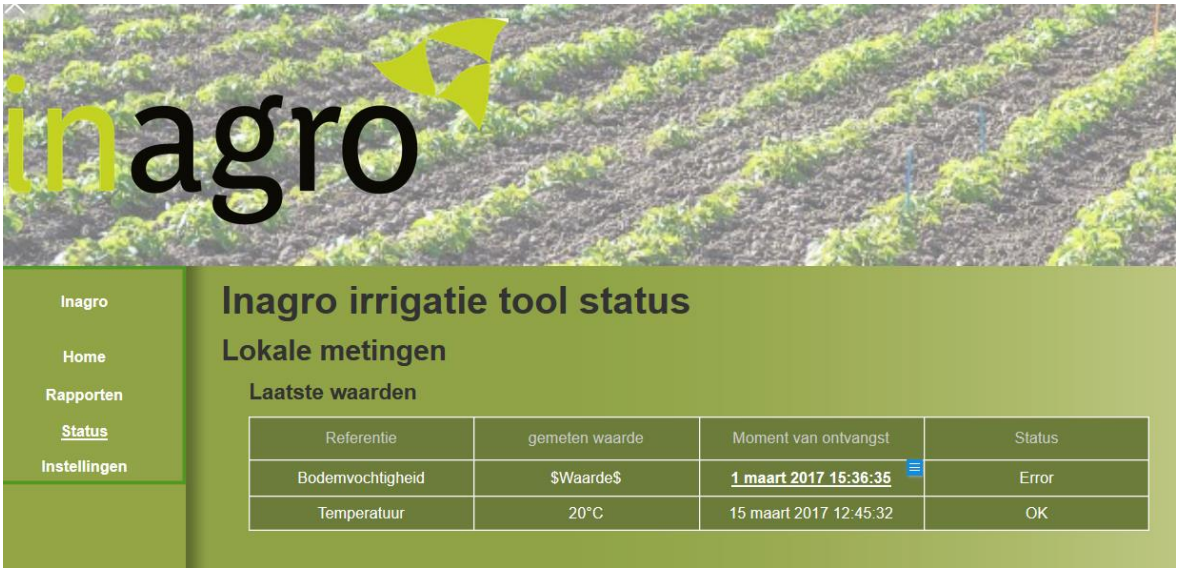
2.2.1 Layout

Het wireframe die we hebben aangemaakt had een zeer eenvoudige layout, zoals te zien op de afbeeldingen (zie Figuur 4, Figuur 5 en Figuur 6) hieronder. Dit heeft als reden dat dit wireframe enkel dient om de functies te tonen aan de klant.

Figuur 4: Inloggen Wireframe



Figuur 5: Rapporten Wireframe



Figuur 6: Status dashboard Wireframe

2.2.2 Opmerkingen

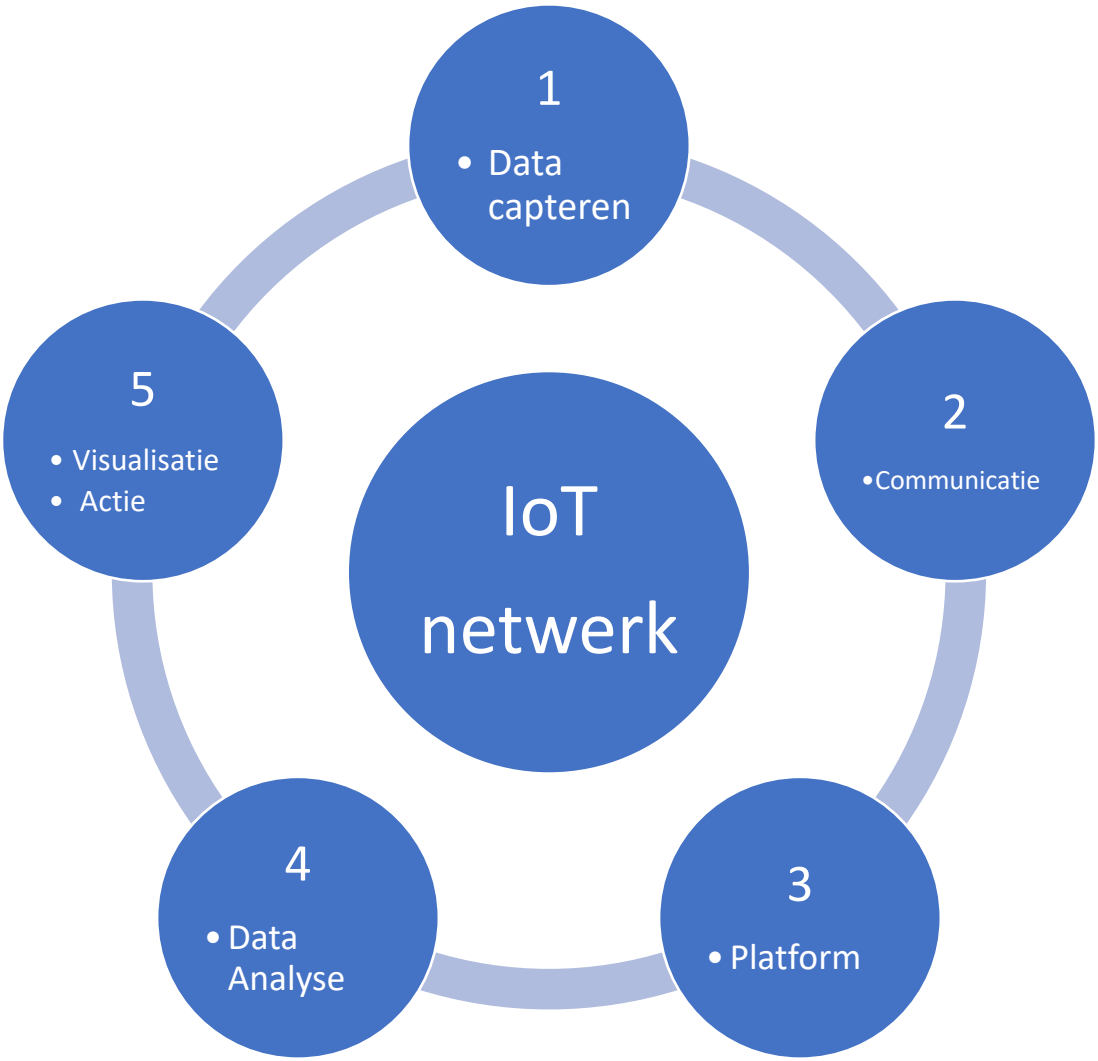
Deze wireframe werd gemaakt volgens de visie die we op dat moment hadden van het gehele project. Natuurlijk is het onmogelijk om meteen een correcte representatie te hebben van de wensen van de klant, daardoor is een wireframe ook steeds noodzakelijk. Maar eens de klant het wireframe te zien heeft gekregen, kon de klant (Inagro) meer richting geven aan de webapplicatie. Toen Inagro het wireframe te zien heeft gekregen hebben ze enkele opmerkingen gegeven, deze opmerkingen hebben we steeds in het achterhoofd gehouden bij het uitwerken van de webapplicatie.

Het wireframe werd gemaakt in het voortraject van deze stage en zorgde voor een basis. De webapplicatie die we gemaakt hebben tijdens de eigenlijke stage lijkt niet veel mee te hebben van het wireframe, maar dat komt omdat de business case steeds concreter werd en we dus steeds beter de webapplicatie konden opbouwen.

3 Internet of Things

3.1 Hoofdblokken Internet of Things netwerk

Als we kijken naar een standaard Internet of Things netwerk, ongeacht de toepassing, kunnen we de volgende vijf belangrijke blokken waarnemen in Figuur 7.



Figuur 7: Hoofdblokken IoT netwerk

Deze vijf hoofdblokken worden in de volgende pagina's in meer detail besproken om een compleet beeld te kunnen scheppen van hoe een Internet of Things netwerk er precies uitziet. Deze uitwerking is de standaard, er kunnen hier ook lichte variaties zijn die we hier niet bespreken.

3.1.1 Data capteren

Sensors

Met sensoren verzamelen apparaten (embedded systems) data over hun omgeving. Toestellen hebben dus met behulp van sensoren de mogelijkheid om fysische grootheden in hun omgeving op te nemen. Het toestel heeft dan de mogelijkheid om hierop meteen te reageren of om de data door te sturen via een communicatiemethode.

Voorbeelden hiervan zijn:

- Temperatuursensor
- Lichtsensor
- Luchtdruksensor
- Microfoon
- Camera

Toestellen

In plaats van data van de omgeving van een toestel te gebruiken, kan het nuttig tot zelfs noodzakelijk zijn om ook de toestand van het toestel zelf uit te lezen, dit al dan niet met behulp van sensoren. Met deze data is het mogelijk om vroegtijdig in te grijpen bij storingen en om falen te vermijden of zelfs te elimineren.

Voorbeelden hiervan kunnen zijn:

- Temperatuur van een processor;
- Uren dat een toestel in werking is;
- Reden van laatste uitval/herstart;
- Batterij status.

Extern

Data uitlezen van andere sites met bijvoorbeeld webfeeds of open data.

Voorbeelden van webfeeds en open data zijn:

- Weersverwachtingen afhalen van een weersite;
- Twitter berichten;
- Verkeersinfo van externe partner.

3.1.2 Communicatie

In deze stap wordt de gecapteerde data overgebracht naar het platform of naar een ander toestel. De communicatiemethodes voor het transporteren van data tussen toestellen (embedded systems) zijn zeer uiteenlopend. Elk communicatiesysteem heeft zo zijn eigen voor- en nadelen. Zoals de hoeveelheid data, het energieverbruik en de performantie (bandbreedte en hoeveelheid data).

Voorbeelden van communicatiemethoden zijn:

- Wifi
- Ethernet
- Lora
- Sigfox
- Zigbee
- BLE

3.1.3 Platform

Het platform is in een IoT netwerk het centrale aanspreekpunt voor de apparaten.
Het kan de gegevens die het krijgt van toestellen opslaan of verwerken en doorsturen naar andere toestellen die hierop moeten reageren.
Het platform heeft ook de mogelijkheid om in bepaalde omstandigheden beslissingen te nemen, zoals een alarmsignaal versturen wanneer er problemen optreden, of zelf acties ondernemen om mogelijke problemen tegen te gaan.

3.1.4 Data-analyse

Bij data-analyse wordt alle beschikbare data doorlopen en geanalyseerd. Door met de opgeslagen data en verschillende algoritmes wiskundige modellen te realiseren, waarmee er dan eventueel voorspellingen in de toekomst kunnen gemaakt worden. Maar data-analyse zorgt er ook voor dat we kunnen nagaan of de data die ontvangen werden van een bepaalde sensor wel correct is. Hierdoor kunnen foutief werkende sensoren snel opgespoord en dus ook vervangen worden.

3.1.5 Visualisatie

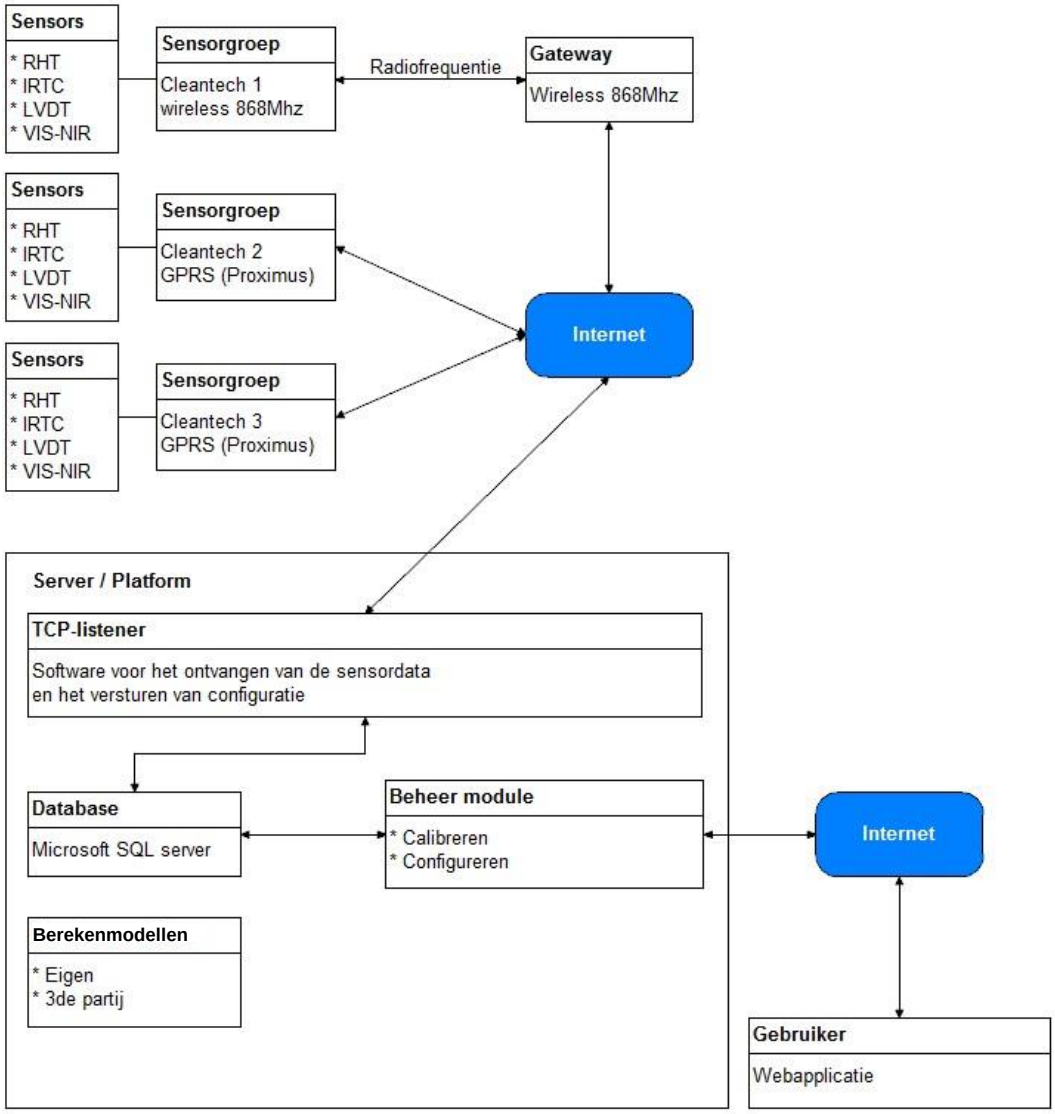
Bij Internet of Things toepassingen moeten slimme objecten kunnen communiceren met de gebruiker. Hiervoor moet dus een gebruikersinterface voorzien worden waar de gebruiker bepaalde configuraties kan op instellen, en waarop de gebruiker grafieken of schema’s kan zien van de gecapteerde en verwerkte data, ook kan er een mogelijkheid zijn om bepaalde actuatoren aan te sturen via de gebruikersinterface.

Voorbeelden van visualisatie zijn:

- Dashboard in desktopapplicatie
- Webapplicatie
- Mobile applicatie
- Alarmering
- Genereren van rapporten

4 Huidig sensornet

4.1 Blokschema



Figuur 8: Blokschema huidig Inagro sensornetwerk

Dit blokschema (zie Figuur 8) is getekend op basis van gegeven documentatie van Inagro (zie Bijlage 1).

Verduidelijking van de afkortingen die gebruikt worden bij de sensoren:

- RHT: Relative Humidity and Temperature;
- IRTC: Infrared Thermocouples;
- LVDT: Linear Variable Differential Transformer;
- VIS-NIR: Visible Near-infrared.

4.2 Algemene info

Inagro maakt momenteel gebruik van een eigen sensornet dat al enkele jaren dienstdoet. Dit sensornet opereert op een eigen server en is een op maat gemaakt systeem.

Inagro gebruikt nu voornamelijk sensoren en sensorgroepen (embedded systemen) die op maat gemaakt zijn en enkel kunnen communiceren met hun eigen systeem, dit omdat er hiervoor gebruik gemaakt wordt van een eigen dataformaat dat besproken wordt in hoofdstuk 4.4.

Door de voortdurende groei en standaardisatie in de wereld van Internet of Things, zijn bepaalde onderdelen in het huidige systeem achterhaald. Daarom kwam de vraag van Inagro naar Korazon om te zoeken naar een meer universelere oplossing en een systeem dat op het Azure Cloud platform kan werken.

4.3 Onderdelen in het sensornet

4.3.1 Sensors en sensorgroepen

Zoals eerder beschreven maakt Inagro gebruik van eigen sensoren en sensorgroepen die op maat gemaakt zijn voor Inagro.

Hieronder is een afbeelding (zie Figuur 9) te zien van een sensorgroep (links) die de temperatuur en luchtvochtigheid waarneemt met behulp van sensors (rechts vochtigheid sensor) en doorstuurt naar een gateway via XBee. Deze gateway kan dan de data van de sensorgroep verder versturen via het internet naar de server.

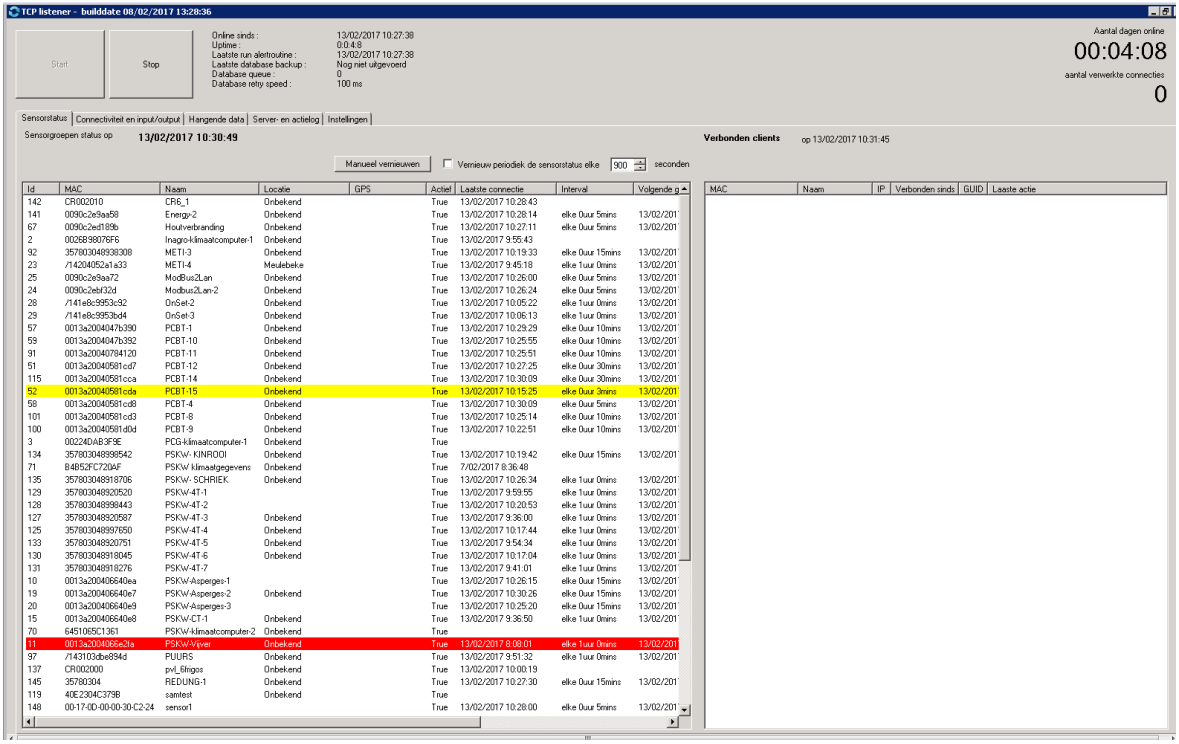
Figuur 9: (links) sensorgroep, (rechts) vochtigheidssensor

Eén van de wensen van Inagro is om minder afhankelijk te zijn van de huidige leverancier van sensoren. Momenteel kunnen de toestellen enkel communiceren met het platform wanneer deze gebruik maken van het huidige dataformaat en protocol. Dit is geen standaard manier van communiceren, en daarom ook moeilijker (vergt extra werk) om toe te passen in nieuwe toestellen, wanneer deze niet van hun gebruikelijke leverancier komen.

4.3.2 Server/platform

Op de server draait het programma “TCP listener” dat de connectie verzorgt met de toestellen die willen connecteren.

De TCP listener leest de data die verstuurd werden door de toestellen in en verwerkt deze zodat ze correct in de database kunnen geplaatst worden. Een afbeelding (zie Figuur 10) van de TCP listener in werking is hieronder te zien.

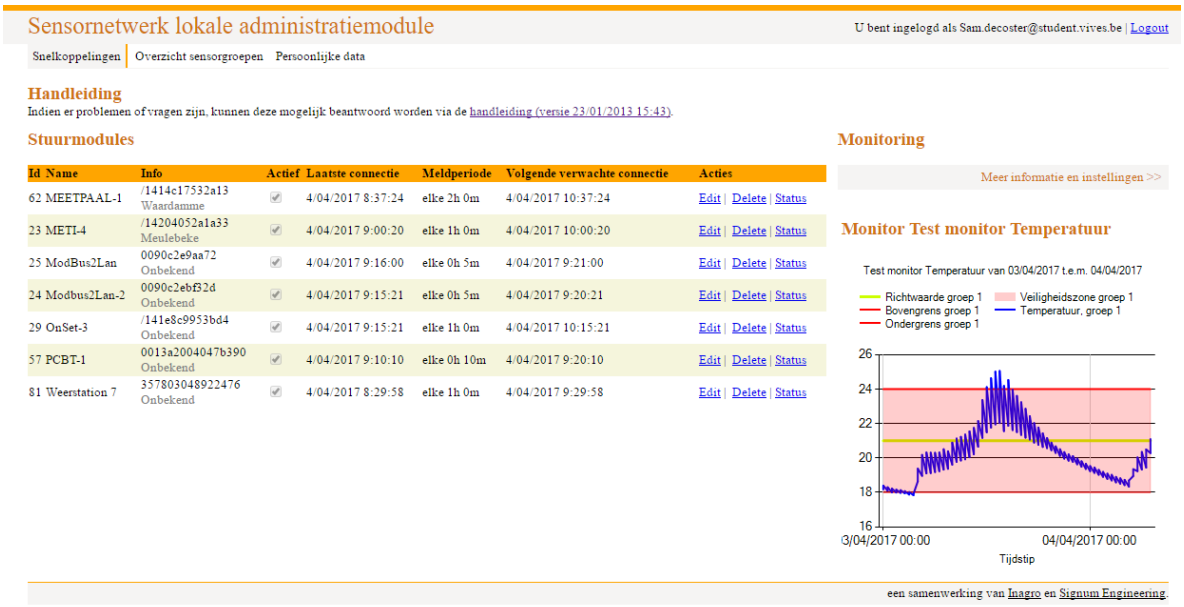


Figuur 10: TCP listener huidig sensornetwerk

De TCP listener verzorgt ook de communicatie van server naar toestel. Dit gebeurt doormiddel van het lezen van de database waarin de commando’s staan die nog naar een bepaald toestel moeten gestuurd worden. Wanneer het desbetreffende toestel connectie maakt zal de TCP listener het commando vanuit de database doorgeven naar het toestel, een database als buffer is hierin cruciaal aangezien een commando niet direct verstuurd kan worden, maar er moet gewacht worden op het initiatief van het toestel om connectie te maken.

4.3.3 Gebruikersinterface

De interface bestaat momenteel uit een webapplicatie die instaat voor het monitoren en configureren van het sensornetwerk. Er is hierin ook een mogelijkheid voor alarmering bij “errors” via mail. Dit is weliswaar zonder instellingen van prioriteit van de error, en dus een minder overzichtelijke methode van alarmering. Een schermafdruk (zie Figuur 11) van de hoofdpagina van de gebruikersinterface is hieronder weergegeven. Een handleiding over de gebruikersinterface is te vinden in Bijlage 2.



Figuur 11: Webapplicatie voorbeeld huidig sensornetwerk

4.4 Dataformaat

Zoals eerder vermeld is het dataformaat dat gebruikt wordt bij het communiceren tussen de toestellen en het platform een eigen dataformaat. Dit “Protocol” (Versie 1.0) werd opgezet in november 2011 (zie Bijlage 3 voor uitgebreide info). Voor ons stageproject is het belangrijk om met dit dataformaat rekening te houden omdat er al een beduidend aantal sensoren in omloop zijn bij Inagro en de kost om deze allemaal te vervangen zou te hoog zijn.

4.4.1 Getal representatie

Getallen kunnen worden weergegeven in decimale en hexadecimale waarden. Hexadecimale waarden worden voorafgegaan door ‘/’. Zo komt /34 overeen met het decimale getal 52.

Reële waarden worden weergegeven als *<geheel deel>.<fractie>,* waarbij ‘.’ als decimaal teken wordt genomen.

4.4.2 Tijd representatie

Voor de tijdsaanduiding in het dataformaat wordt gebruik gemaakt van de Unix-timestamp. Unix-time is een getal (32 bit) dat de tijd uitdrukt in het aantal seconden lopende sinds 1 januari 1970.

Volgens de documentatie van het dataformaat wordt dit getal weergegeven als een hexadecimale notatie. Zo staat: /4C10C9C0 voor: 1276168640, wat op zijn beurt staat voor: Maandag, 13-Feb-17 14:04:06 UTC.

4.4.3 Opbouw van een bericht

Een bericht kan opgedeeld worden in drie grote delen. Deze delen hebben onderstaande opbouw, de opbouw gebeurt steeds in dezelfde volgorde:

1. Message Header: *MAC
2. Bericht data
3. End of data message: *EOD

Header

De header bevat minimaal de bron van de boodschap, in de meeste gevallen is dit het MAC-adres van het toestel dat de gegevens gegenereerd heeft. De structuur van de header wordt hieronder weergegeven:

**MAC, <mac>[,<friendlyName>][,<Latitude decimaal>][,<Longitude decimaal>];*

Bericht data

In de bericht data zijn alle waarden terug te vinden die gemeten werden door de sensoren, met de bijhorende tijd van meting. De structuur van de bericht data worden hieronder weergegeven:

**<kanaalType>,<channelNr>,<unixTime>,<friendlyName>,<rawData>[,<adjusted>[,…]];*

Het is mogelijk om meerdere commando's mee te geven in de bericht data, deze commando's worden dan telkens gescheiden door een ";".

EOD

Het *EOD-commando is het commando dat op het einde van het bericht wordt meegestuurd. Dit commando heeft een volgende opbouw:

**EOD,<cs>;*

De <cs> staat in voor de checksum en is een facultatieve parameter. Deze checksum is een 16-bit waarde berekend als som van alle bytes uit de boodschappen.

5 Azure Cloud platform

Tijdens het voortraject werd er beslist om gebruik te maken van het Azure Cloud platform om de toepassing te realiseren. Deze beslissing werd gemaakt door de bedrijven zelf maar werd stevig onderbouwd in het voortraject.

Het Microsoft Azure Cloud platform is een Cloud computing platform van Microsoft waarin een aantal internetdiensten/software aangeboden kunnen worden als een service. Deze software hoeft niet geïnstalleerd te worden op de servers van de gebruiker, maar draait in het datacenter van Microsoft. Alles gebeurt over het internet. Ook de toegang tot bestanden en mappen gebeurt over het internet. Het voordeel van het Azure Cloud platform is dat verschillende services met elkaar op een eenvoudige manier kunnen gekoppeld worden. Zo is het mogelijk om verschillende “modules” bij te plaatsen, dit volgens de noden van het project.

Microsoft heeft ervoor gezorgd dat de functies in het Azure Cloud platform eenvoudig te schalen zijn, dit wil zeggen dat je enkel maar hoeft te betalen volgens de capaciteit die je gebruikt. Verder kan je de functies in Azure verschalen wanneer de applicatie groeit of wanneer de applicatie meer belast wordt.

Voor het realiseren van de applicatie voor Inagro zullen we volgende functies gebruiken in Azure:

- Cloud services
- Opslagaccount
- SQL-database
- SQL-server

Deze functies worden verder in het rapport nader besproken.

5.1 Cloud services

Azure Cloud services is de computing service die we zullen gebruiken in het Azure Cloud platform voor het realiseren van de applicatie voor Inagro.

Azure Cloud services is dus een computing service. Hiermee wordt er bedoeld dat we in deze service, bewerkingen kunnen uitvoeren. We zullen ervoor zorgen dat de volledige logica en gebruikersinterface van de applicatie gebouwd kan worden op deze service. Om dit te kunnen realiseren hebben we in deze Cloud services twee rollen, namelijk: web role en worker role.

5.1.1 Web role

De web role is het gedeelte in de Cloud services die zal instaan voor de webapplicatie. Deze webapplicatie is een ASP.NET MVC webapplicatie die geprogrammeerd zal worden in C#. De webapplicatie die zal draaien in de web role zal automatisch geïnitialiseerd worden op een IIS (Internet Information Service) van Microsoft.

Aangezien we hier spreken over een webapplicatie, zal deze web role dus voornamelijk instaan voor de gebruikersinterface van de volledige applicatie.

5.1.2 Worker role

Het programma dat we zullen schrijven voor de worker role is een programma dat niet geïnitialiseerd wordt op een IIS, in tegenstelling tot de web role. Het programma dat we hierin zullen laten draaien, zal zich voornamelijk bezighouden met achtergrondtaken zoals het verzorgen van connectie met de sensoren, het controleren en verwerken van de ontvangen data, het genereren van dagelijkse rapporten, etc.

De functie van deze role is te vergelijken met de TCP listener die momenteel gebruikt wordt in het huidige sensornet.

5.1.3 Communicatie tussen verschillende rollen

De verschillende rollen zullen met elkaar kunnen communiceren door middel van een SQL-database die later uitvoeriger besproken wordt. Wanneer de gebruiker bijvoorbeeld configuraties van het programma in de worker role wil aanpassen, zullen deze aanpassingen geplaatst worden in een database door de web role, zodat de worker role deze aanpassingen kan zien in de database en deze zal toepassen.
Omgekeerd zal de worker role de data die hij binnenkrijgt van sensoren op de database plaatsen, zodat deze via de webapplicatie kunnen weergegeven worden in de web role.

5.2 Opslagaccount

Om een Azure Cloud service te kunnen aanmaken hebben we een opslagaccount nodig, omdat de programma's die in de Cloud services zullen draaien ergens moeten opgeslagen worden. Deze is dus met andere woorden onze "harde schijf" waarop de applicatie zal staan.

Het gebruikte Azure opslagaccount wordt in eerste plaats enkel gebruikt voor het opslaan van het programma in het Azure Cloud platform, maar kan in een later stadium ook gebruikt worden om bestanden in op te slaan die vervolgens geanalyseerd kunnen worden door services in Azure die instaan voor data-analyse.

Azure Storage is zeer schaalbaar en kan honderden terabytes aan gegevens opslaan en verwerken. Dit maakt het dus een interessante ondersteuning voor big data scenario's.

5.3 SQL-database en server

Net zoals bij de andere Azure services, heb je bij de Azure SQL-database en server ook de mogelijkheid om deze service te schalen naar de noden van de applicatie.

De Azure SQL-database is de database die gebruikt wordt om al de gegevens in op te slaan. Deze database is ook de database die gebruikt zal worden als communicatiemiddel tussen de verschillende rollen in de Azure Cloud services. Wat wel belangrijk is om te weten is dat de SQL-database in Azure altijd de UTC-tijd[1] zal hebben, ongeacht de locatie waar de SQL-database zich in bevindt. Dit is belangrijk als we werken met tijd, dan is het belangrijk om alles die de SQL-database zal leveren aan informatie overeenkomt met de tijdzone waarin de gebruiker zich op dat moment bevindt.

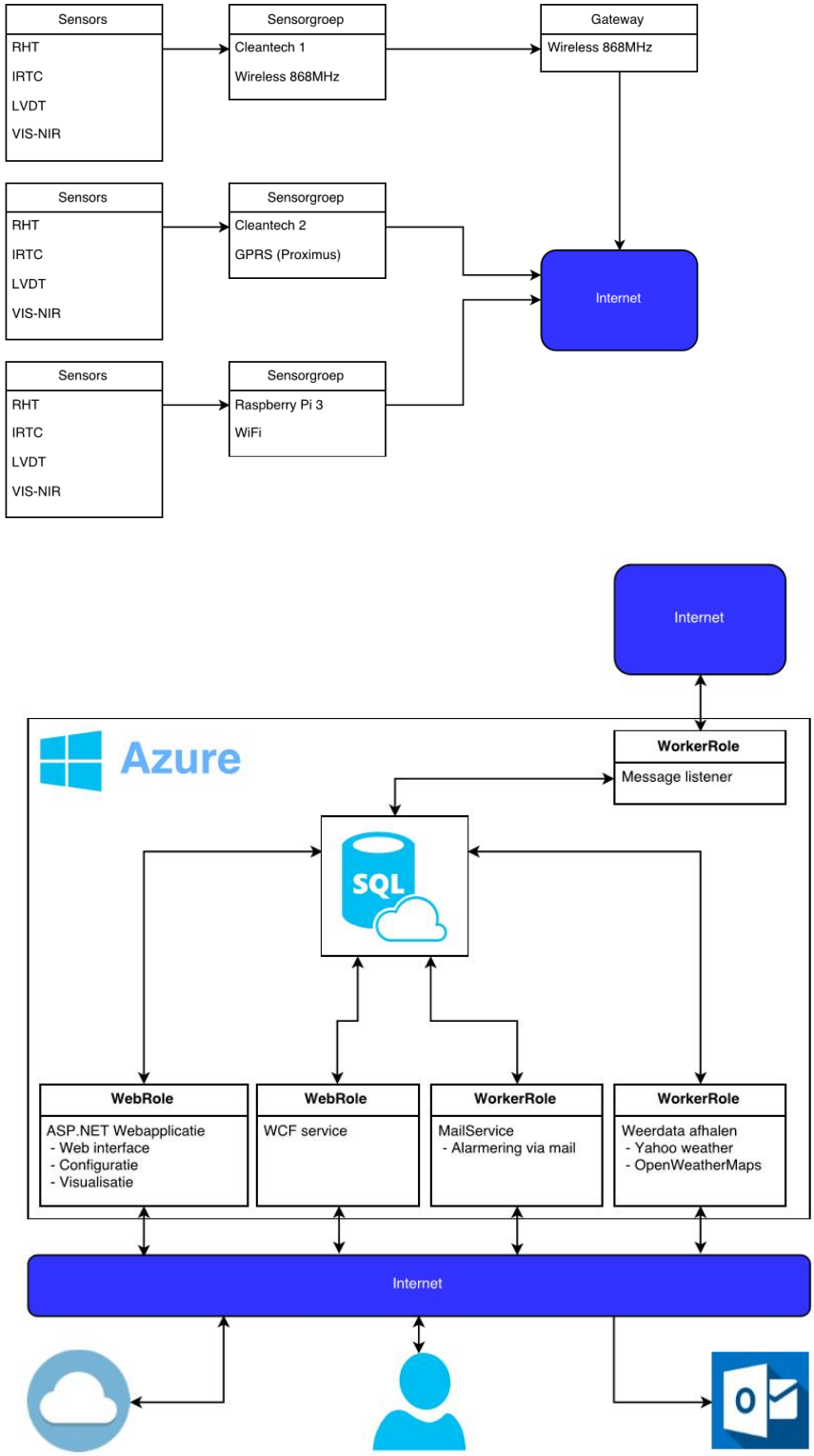
Aangezien deze database meegegeven wordt door Microsoft Azure als een service, kan deze eenvoudig gekoppeld worden aan andere services binnenin het Azure platform. Dit kan handig zijn voor eventuele uitbreidingen waarin andere Azure services moeten gebruikt worden.

De gegevens in deze database zullen in de applicatie beheerd worden door middel van het Entity Framework. In hoofdstuk 6.2 wordt meer info gegeven wat dit framework precies inhoudt en wat er mee gerealiseerd kan worden.

[1] UTC is een standaardtijd, deze UTC kan gelijk gesteld worden (met enkele milliseconden verschil) aan de Greenwich Mean Time (GMT). In België geldt de UTC+1 (wintertijd) en de UTC+2 (zomertijd).

5.4 Blokschema

In Azure is het de bedoeling om een IoT systeem te realiseren. Dit IoT systeem zal ervoor zorgen dat telers hun eigen sensoren kunnen koppelen aan dit netwerk. Dit systeem zal gebaseerd zijn op onderstaand blokschema (zie Figuur 12).



Figuur 12: Eigen blokschema met Azure Cloud platform

6 Webapplicatie

6.1 ASP.NET & MVC

6.1.1 ASP.NET

Tijdens onze stage hebben we onze webapplicatie gerealiseerd in ASP.NET. ASP.NET is een server-side webapplicatie framework van Microsoft die het gebruik van hun .NET technologie mogelijk maakt op websites. ASP staat voor Active Server Pages.

ASP.NET is een manier om op een webserver, websites of webapplicaties aan te maken met behulp van programmacode (standaard C# of VB.NET). Hiermee kunnen vaste HTML-codes gecombineerd worden met variabele inhoud die door een programma wordt geproduceerd, dit al dan niet volgens de input van de gebruiker. De server-side applicatie kan dus data verwerken en gegevens uit een database halen om te verwerken in de HTML-code om zo een pagina te genereren.

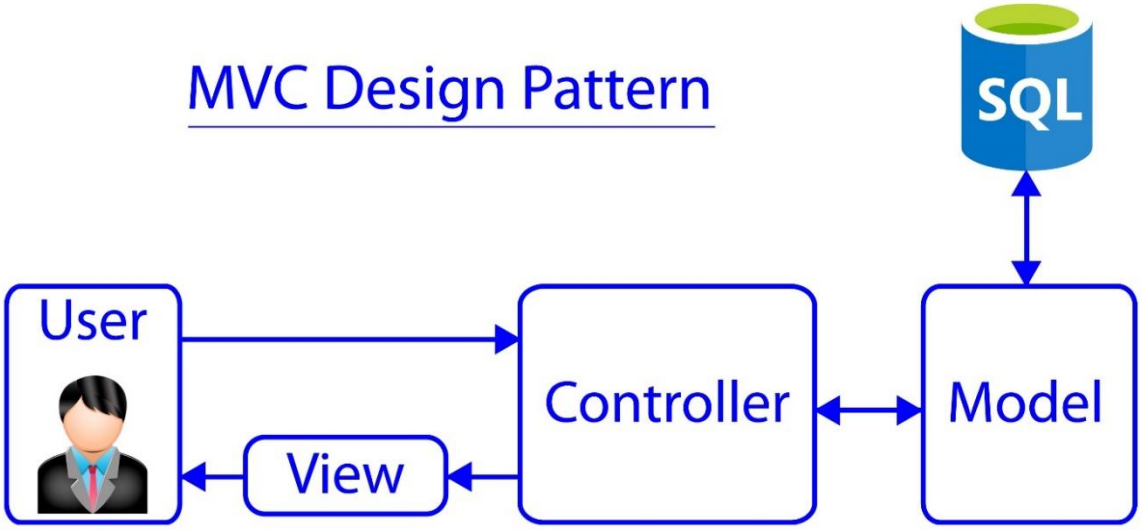
We kunnen dus concluderen dat we met behulp van dit framework onze webapplicatie een dynamisch karakter kunnen geven. Met dynamisch karakter wordt bedoeld dat de website/webapplicatie duizenden mogelijke resultaten kan produceren, afhankelijk van wat de gebruiker/bezoeker vraagt.

Het interessante aan ASP.NET in ons stageproject is dat ASP.NET applicaties gerealiseerd kunnen worden in een web role op Cloud services in Azure. Dit wil dus zeggen dat we via ASP.NET in C# onze webapplicatie voor Inagro kunnen bouwen en deze op onze Cloud services in Azure kunnen plaatsen. Deze kunnen we dan laten samenwerken met andere Azure toepassingen zoals de Azure SQL-database waarop al onze data zal worden opgeslagen van het project.

Naast dit server-side applicatie framework zullen we in de webapplicatie ook gebruik maken van andere frameworks die open-source zijn, zoals JQuery en Bootstrap. Deze frameworks zijn dan wel op client side, wat wil zeggen dat deze instaan voor verwerking en/of design op de computer van de gebruiker en niet worden uitgevoerd op de server waarop de webapplicatie staat.
Dit is louter een methode om op een eenvoudige manier ervoor te zorgen dat de webapplicatie -wat tegenwoordig een must is in de wereld waarop veel websites en webapplicaties gebruikt worden- mobielvriendelijk is, en ervoor zorgt dat eventuele JavaScript code snel en gemakkelijk kan geschreven worden voor de applicatie. Bij JQuery is het bijvoorbeeld eenvoudiger om AJAX te gebruiken in de applicatie en zo extra data van de server te vragen, zonder dat de pagina volledig herladen moet worden, wat voor mobiele toepassingen dan terug een voordeel is op gebied van datatransport over het internet die hiermee gereduceerd wordt.

6.1.2 MVC

De webapplicatie wordt dus gemaakt in het ASP.NET framework. In ASP.NET zullen we ook het MVC design pattern gebruiken. MVC staat voor Model-View-Controller, en is een design pattern die het mogelijk maakt om de geschreven code beter te lezen en te hergebruiken. Met MVC splitsen we de werking van de webapplicatie op in drie grote delen: Models, Views en Controllers. Een algemeen blokschema van het MVC design pattern is hieronder weergegeven (zie Figuur 13).



Figuur 13: Blokschema MVC Design Pattern

Model

Een model in het MVC design staat voor de data die gebruikt wordt. Een model geeft een betekenis aan ruwe data door een relatie te leggen tussen data en logica. Het model vereenvoudigt het uiteindelijke opslaan van de data. Het ontvangen en schrijven van data, van en naar de database, gebeurt via een data access layer (DAL), deze laag is optioneel en niet vereist voor het MVC design pattern.

View

Informatie wordt weergegeven via de View. Hierin worden elementen gedefinieerd die noodzakelijk zijn voor de gebruikersinterface. De View staat niet in voor het verwerken of berekenen (zoals beslissingen nemen, data parsen, …) van de data die weergegeven worden.

Controller

De controller staat in voor bewerkingen en het verwerken van events, wat in de meeste gevallen operaties zijn van de gebruiker. Zoals bijvoorbeeld het aanklikken van een knop die dan ervoor zorgt dat er bepaalde bewerkingen uitgevoerd dienen te worden.

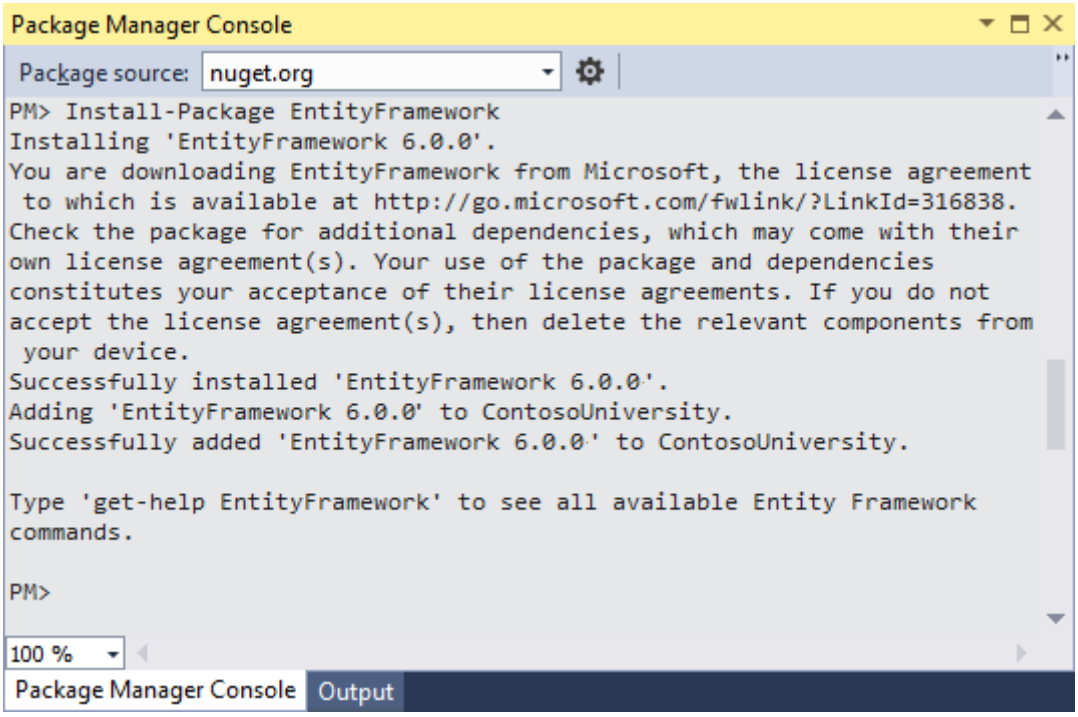
6.2 Entity Framework

Het Entity Framework (EF) is een framework/tool in .NET die we zullen gebruiken als bovenliggende laag voor het beheren van de data.

6.2.1 Installatie

Deze tool kan eenvoudig gebruikt worden in Visual Studio met behulp van de NuGet Package Manager.
In de Package Manager Console dient volgend commando ingegeven te worden voor het installeren van het Entity Framework: “Install-Package EntityFramework”.

Nadat het commando ingegeven is, zal het Entity Framework geïnstalleerd worden in de applicatie zoals hieronder weergegeven (zie Figuur 14).



Figuur 14: Installatie EntityFramework via Package Manager Console

Nadat deze tool is geïnstalleerd kan er overgegaan worden naar de volgende stap, het aanmaken van de benodigde datamodellen.

6.2.2 Aanmaken datamodellen

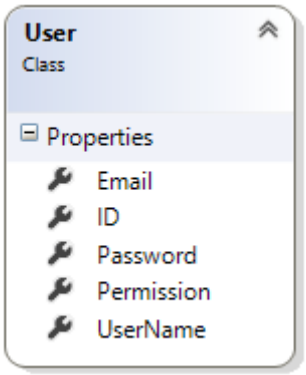
Om de data te beheren via het Entity Framework moeten er datamodellen voorzien worden die beschrijven welke data er gebruikt worden en hoe deze eruit zien. Deze datamodellen zijn klasses in C# en ze worden volgens het MVC design pattern geplaatst in de map “Models”. Als voorbeeld zullen we in dit hoofdstuk het datamodel van de gebruikers bespreken.

Hieronder zie je de klasse die geschreven is voor de gebruikers.

Models/Users.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Site.Models
{
    public class User
    {
        public int ID { get; set; }
        public string Email { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }

        /* User Permission
         * 1 = Admin
         * 2 = Tester/Developer
         * 3 = Standaard User
         */
        public int Permission { get; set; }
    }
}
```

Het klasse diagram (zie Figuur 15) van deze aangemaakte klasse ziet er als volgt uit:



Figuur 15: Klasse voor gebruikers

Hierin zie je dat we voor elke gebruiker vijf parameters hebben, namelijk:

- Email
- ID
- Password
- Permission
- UserName

Deze klasse wordt in het volgend hoofdstuk 6.2.3 gebruikt in de databasecontext en beschrijft de parameters die beheerd moeten worden in de database door middel van het Entity Framework. Deze klasse is ook niet volledig bindend en kan in een later stadium aangepast worden indien er uitbreidingen nodig zijn.

6.2.3 Aanmaken Database Context

Zoals in het vorige punt besproken werd, dient het model voor de database context. Deze database context is opnieuw een klasse, maar hierin staan alle modellen die tot een bepaalde database context behoren. In de applicatie wordt deze klasse gebruikt voor het beheren en bewerken van de data die vervolgens wordt opgeslagen in de database.

Aangezien we als voorbeeld een model hebben gemaakt van de gebruikers, zullen we nu een database context beschrijven die gebruikt wordt voor de gebruiker. Om onze bestand-locaties eenvoudig en duidelijk te houden, maken we een nieuwe map aan in het project met de naam “DAL” hierin staat DAL voor Data Access Layer. Een Data Acces Layer zorgt ervoor dat er snel en gemakkelijk data kan geraadpleegd worden vanuit de database.

In deze map DAL maken we een nieuw bestand genaamd “UsersContext.cs” met volgende inhoud:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Site.Models;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace Site.DAL
{
    public class UsersContext : DbContext
    {
        public UsersContext() : base("UsersContext")
        {
        }
        public DbSet<User> Users { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
}
```

Vooraleer we deze “UsersContext” kunnen gebruiken in de webapplicatie dienen we enkel nog de “connectionString” te plaatsen in het “Web.config” bestand. Deze “connectionString” laat ons toe om een verbinding op te zetten met de database, en ziet er als volgt uit in het project:

```
<connectionStrings>
      <add name="UsersContext" connectionString="Data
      Source=inagrosql.database.windows.net;Initial Catalog=inagrodb;User
      ID=inagro;Password=Tgg8877j" providerName="System.Data.SqlClient" />
</connectionStrings>
```

6.2.4 Gebruiken van Database Context

In dit punt wordt besproken hoe de aangemaakte “UsersContext” in de applicatie gebruikt kan worden. Als voorbeeld hebben we een “AdminModuleController” gemaakt in de map Controllers. In deze “AdminModule” zullen we onze gebruikers weergeven, bewerken en aanmaken.

We beginnen hierin met een pagina waarin we al onze gebruikers kunnen zien. Dit doen we door een View aan te maken met de naam “Users” in de map “Views”.

Om ervoor te zorgen dat we het model User kunnen gebruiken in deze View moeten we dit bovenaan in het document duidelijk maken. Dit ziet er als volgt uit:

```
@model IEnumerable<Site.Models.User>
```

In een tabel kunnen alle gebruikers dan geplaatst worden door een foreach-lus te gebruiken zoals hieronder weergegeven.

```
@foreach (var item in Model)
{
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.ID)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.UserName)
        </td>
        <td>
            @if (item.Permission == 1)
            {
                <p>Admin</p>
            }
            else
            {
                <p>User</p>
            }
        </td>
        <td>
            @Html.ActionLink("Edit", "EditUser", new { id = item.ID }) |
            @Html.ActionLink("Details", "DetailsUser", new { id = item.ID }) |
            @Html.ActionLink("Delete", "DeleteUser", new { id = item.ID })
        </td>
    </tr>
}
```

De data kan via de controller worden doorgegeven naar de View op volgende manier.

```
public ActionResult Users()
        {
            if (checkPermission())
            {
                return View(db.Users.ToList());
            }
            else
            {
                return RedirectToAction("Index", "Home");
            }

        }
```

Het object “db” dat hier meegegeven wordt in de View is het object dat we bovenaan in onze klasse “AdminModuleController” van de klasse “UsersContext” aanmaken.

Dit object stelt ons ook in staat om data in de database te bewerken, of nieuwe data in de database te plaatsen. Als voorbeeld hebben we hieronder een stuk code geplaatst die toont hoe een bepaalde gebruiker bewerkt wordt.

```
[HttpPost]
public ActionResult EditUser(int id)
{
       User user = db.Users.Find(id);
       user.Email = Request["Email"];
       user.UserName = Request["UserName"];
       user.Permission = Convert.ToInt32(Request["Permission"]);

       db.SaveChanges();
       return RedirectToAction("Users");
}
```

Hierbij moeten we wel opmerken dat het nodig is om iedere keer wanneer er data aangepast is, de functie “SaveChanges” in het object “db” moet opgeroepen worden, wanneer we willen dat deze data ook effectief naar de database wordt weggeschreven.

6.3 Inloggen via “Mijn Bedrijf”

6.3.1 WCF

WCF is een afkorting voor Windows Communication Foundation. WCF is van Microsoft en is ontworpen als communicatieplatform voor gedistribueerde applicaties. Het programmeermodel is ingebouwd in het .NET-framework van Microsoft. Bij gedistribueerde applicaties verstaan we hieronder “distributed computing”. Dit is een techniek waarbij rekentaken niet door één enkele computer worden uitgevoerd, maar door een cluster van machines die via een netwerk met elkaar verbonden zijn. Dit houdt in dat rekentaken of andere taken zoals opslag, gedeeld kunnen worden met andere computers.

WCF’s worden in Mijn Bedrijf van Inagro gebruikt om de verschillende webapplicaties van Inagro de mogelijkheid te geven om via één account (het Mijn Bedrijf account) in te loggen. Zo kan men met dit ene account de verschillende webapplicaties ter beschikking te krijgen.

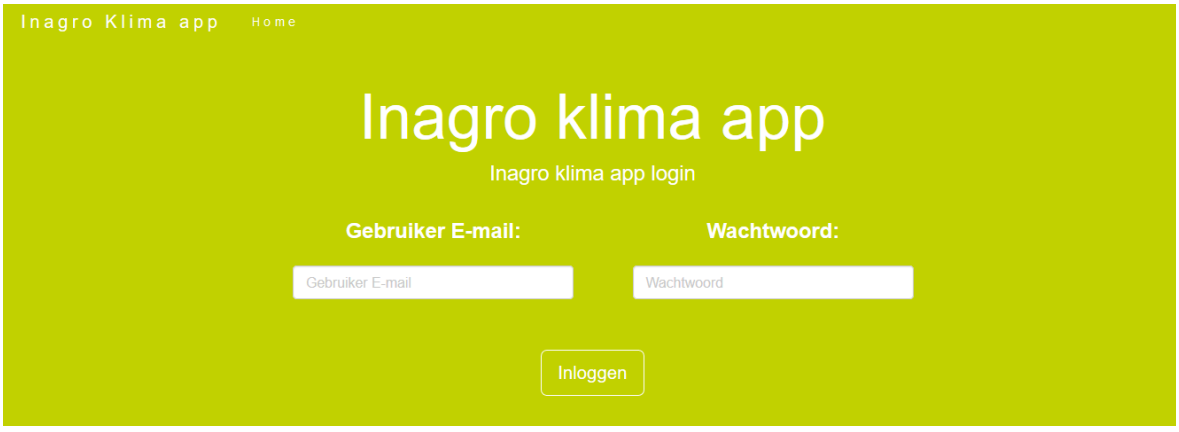
6.3.2 Klasses voor inloggen

In de WCF die we kregen van Inagro zitten twee belangrijke klasses die de gebruikers de mogelijkheid geven om in te loggen op de webapplicatie.

- SessieClient
- clsApplicatiesessie

SessieClient

Bij het openen van de webapplicatie, krijg je op de homepagina een inlogformulier te zien zoals hieronder weergegeven (zie Figuur 16).



Figuur 16: Inlog formulier op Home pagina

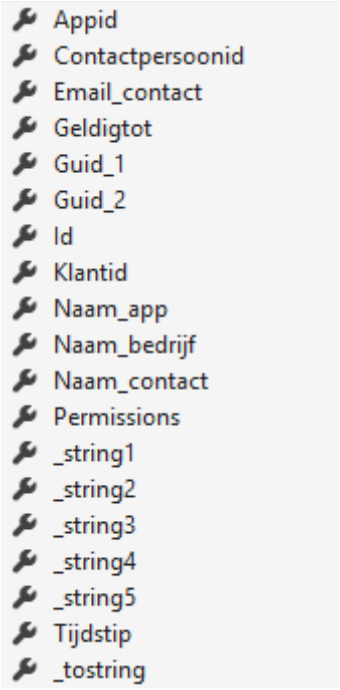
Deze form stuurt je door naar het pad: “/Home/Login” en stuurt de inloggegevens mee via POST-data.

In de HomeController.cs hebben we een method die de meegegeven inloggegevens verwerken via de SessieClient klasse van in de WCF.

```
[HttpPost]
public ActionResult Login()
{
        string Email = Request["UserMail"];
        string PassArg = Request["Password"];

        SessieClient MijnBedrijfSessieClient = new SessieClient();
        clsApplicatiesessie MijnBedrijfGebruiker;
        MijnBedrijfGebruiker =
                MijnBedrijfSessieClient.VerifieerGebruiker("F6C1EA3A-1C1E-4C34-
                A096-61DDF7D2ED87", Email, PassArg);

        if (MijnBedrijfGebruiker.Contactpersoonid > 0)
        {
                if(db.Users.Where(r => r.InagroId ==
                        MijnBedrijfGebruiker.Contactpersoonid).Count() == 0)
                {
                        MijnBedrijfGebruikers currentUser = new
                                MijnBedrijfGebruikers();
                        currentUser.InagroId =
                        Convert.ToInt32(MijnBedrijfGebruiker.Contactpersoonid);
                        currentUser.Klantid =
                        Convert.ToInt32(MijnBedrijfGebruiker.Klantid);
                        currentUser.NaamBedrijf = MijnBedrijfGebruiker.Naam_bedrijf;
                        currentUser.Naam_contact =
                                MijnBedrijfGebruiker.Naam_contact;
                        currentUser.Permission =
                                Convert.ToInt32(MijnBedrijfGebruiker.Permissions);
                        currentUser.Email_Contact =
                                MijnBedrijfGebruiker.Email_contact;

                        db.Users.Add(currentUser);
                        db.SaveChanges();
                    }
                    Session["UserId"] = db.Users.Where(r => r.InagroId ==
                        MijnBedrijfGebruiker.Contactpersoonid).First().id;
                    return RedirectToAction("Dashboard", "User");
        }
        else
        {
                ViewBag.Error = true;
                return View("Index");
        }
}
```

clsApplicatiesessie

Deze klasse wordt gebruikt om een object te creëren waarin de data van de ingelogde gebruiker zich in zullen bevinden. De parameters die via deze WCF worden verkregen zijn in onderstaande afbeelding (zie Figuur 17) te zien.



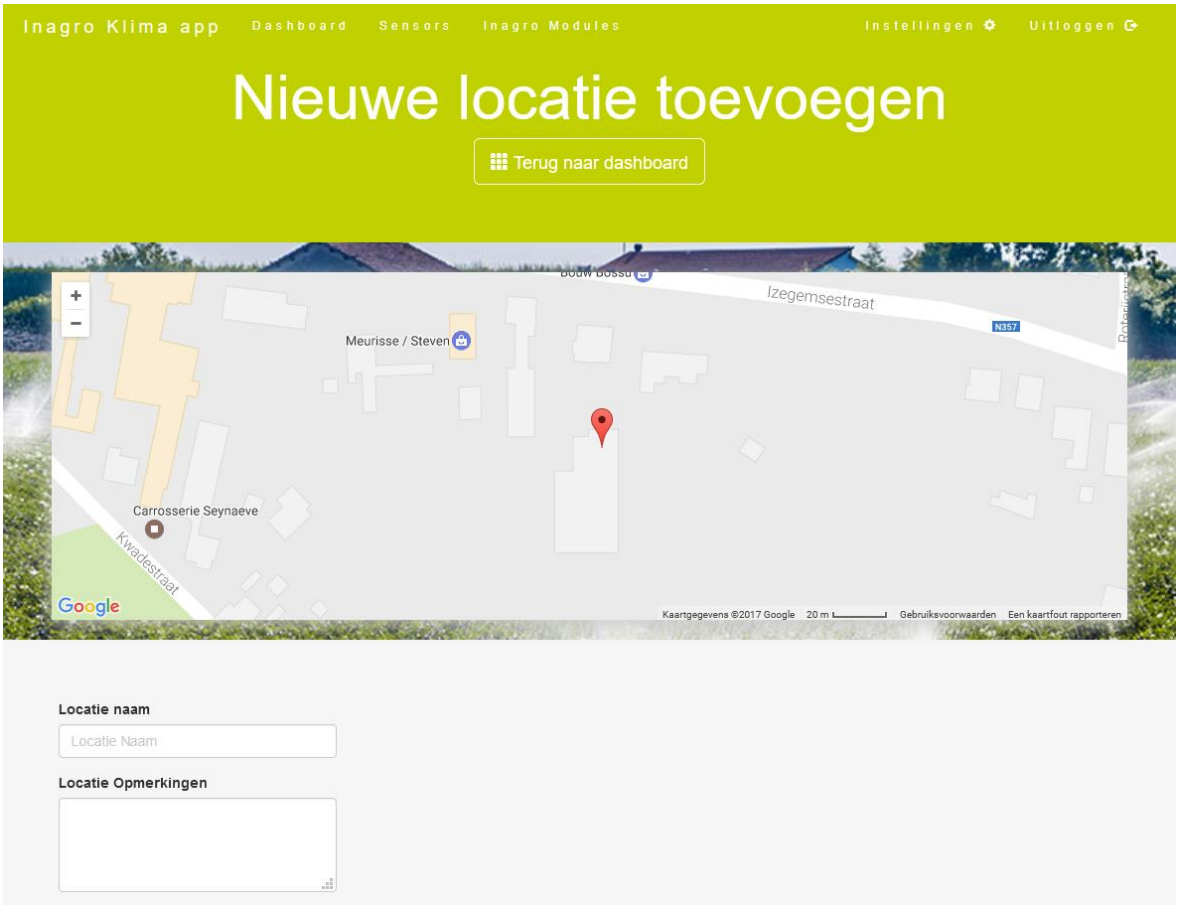
Figuur 17: Parameters van de gebruikers via Mijn Bedrijf

6.4 Locaties

In de webapplicatie is het belangrijk dat de gebruiker locaties (=percelen) kan toevoegen op zijn account. Om dit op een visuele manier te realiseren, hebben we gebruik gemaakt van de Google Maps API. Deze API geeft ons de mogelijkheid om Google Maps te implementeren in onze webapplicatie met behulp van enkele JavaScript-commando's.

Het is echter niet mogelijk om deze API te gebruiken indien er gewerkt wordt met versies van Chrome die hoger zijn dan 50. Chrome laat enkel toe dat de locatie eventueel gedeeld mag worden met een site indien deze een https-site is. Indien dit niet het geval is zal Chrome automatisch het delen van de locatie blokkeren, zonder dat de gebruiker hierin een keuze heeft. Maar omdat dit eindwerk zich niet focust op een volledige werking over alle beschikbare browsers en meer focust op het scheppen van een beeld van de webapplicatie zelf, is dit hier dus van minder belang. Het spreekt voor zich dat er in een later stadium hiervoor een oplossing dient gezocht te worden.

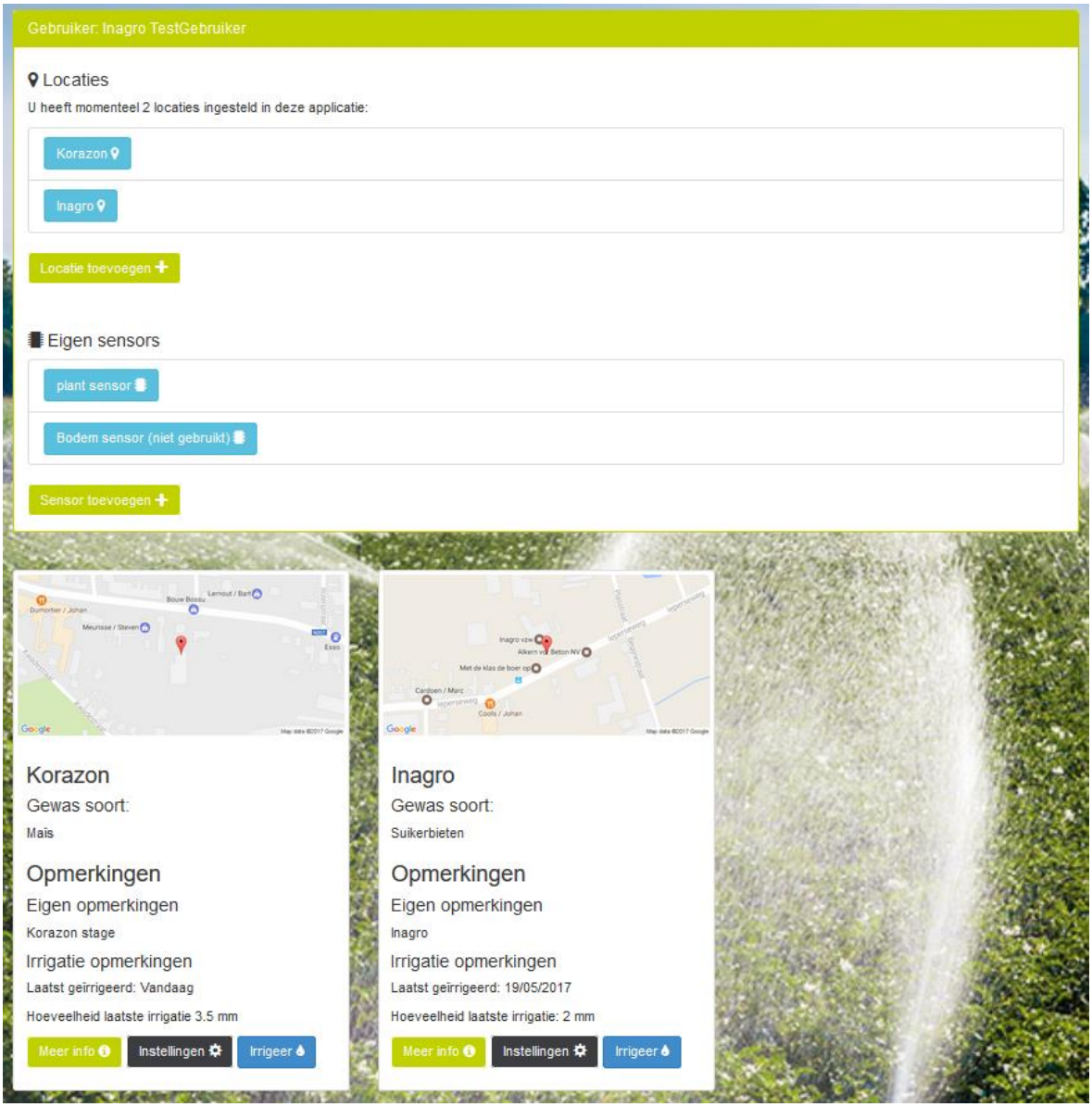
Een voorbeeld van hoe dit er praktisch uitziet in de webapplicatie, is hieronder te zien op de afbeelding (zie Figuur 18).



Figuur 18: Google Maps integratie voor het toevoegen van een locatie

Op deze kaart kan de gebruiker een locatie aanduiden, die vervolgens op zijn naam wordt opgeslagen in de database. Hierdoor kunnen andere functies van de applicatie, data capteren op basis van deze locatie die in de database terug te vinden is. De teler kan dan vervolgens een naam meegeven aan de locatie en eventuele opmerkingen erbij plaatsen, dit zal ervoor zorgen dat elke locatie die aangemaakt wordt gemakkelijk terug te vinden is voor de teler als hij niet meteen ter plaatse is.

Google Maps moet niet noodzakelijk dynamisch gebruikt worden in de webapplicatie. Als visuele weergave hebben wij bijvoorbeeld geopteerd om een afbeelding van de locaties te gebruiken met behulp van Google Maps. Hiermee krijgt de gebruiker een extra visuele ondersteuning om de opgeslagen locaties terug te vinden en dus hoeft hij niet enkel gebruik te maken van de naam die aan een locatie wordt gegeven, zoals hieronder geïllustreerd (zie Figuur 19).



Figuur 19: Visualisatie van locaties via Google Maps

6.5 Sensoren verkrijgen in sensornet

Het verkrijgen van de sensoren die zich in het sensornet van Inagro bevinden kunnen we ook snel en gemakkelijk bereiken met het behulp van een WCF, die ons beschikbaar werd gesteld door Inagro. Hiervoor werd er reeds uitgelegd wat een WCF precies inhoudt, dus introductie is hier niet meer nodig, we zullen hier meteen de klasses uitleggen en hoe er precies te werk gegaan moet worden om een correct resultaat te verkrijgen van deze WCF.

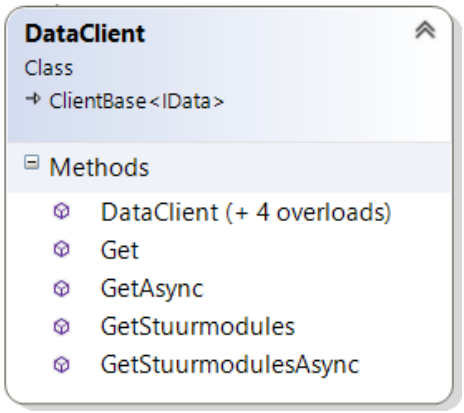
6.5.1 Klasses

In deze WCF bevinden zich drie klasses, elke klasse is nodig om tot het gewenste resultaat te komen. De klasses hebben volgende namen:

- DataClient
- DataRecord
- Stuurmodule

DataClient

De klasse DataClient zal instaan voor het afhalen van alle data, dit zijn zowel de sensoren zelf (die de naam stuurmodules hebben) als de data waarover deze stuurmodules beschikken. Op de afbeelding (zie Figuur 20) hieronder is het klassediagram te zien.



Figuur 20: Klasse van DataClient

De belangrijkste methodes hier zijn dus “Get” en “GetStuurmodules”. Deze twee methodes zullen nu verder uitgelegd worden.

Get

Get zal data afhalen van een specifieke stuurmodule (sensor) aan de hand van het ID van deze module. Er moet echter wel een parameter meegegeven worden die zal aantonen welke data we precies willen verkrijgen van deze stuurmodule. Deze parameter is iets in de lijn van: rht, ns, etc. Tot slot dient er dan nog een start- en eindtijd meegegeven te worden, zodat we data krijgen tussen die tijdspanne. De Get methode en de parameters die erbij horen is hieronder te zien:

Get(**string** *appKey*, **long** *stuurmoduleId*, **string** *parameter*, **System.DateTime** *start*, **System.DateTime** *eind*)

GetStuurmodules

Deze method GetStuurmodules zal, de naam geeft het zelf al voor een stuk weg, alle stuurmodules teruggeven die we wensen te krijgen. De stuurmodules die we terugkrijgen worden bepaald door de argumenten die we meegeven in de method. We dienen een parameter in te geven, eenzelfde parameter zoals bij de Get method, deze zorgt ervoor dat we alle stuurmodules zullen terugkrijgen die over deze parameter zal beschikken. We dienen dan ook nog een boolean mee te geven “locatieGekend”, als deze boolean “false” is dan zullen we alle modules terugkrijgen, indien “true”, dan krijgen we enkel de modules terug die over een locatie beschikken (en dus een latitude en longitude hebben). De GetStuurmodules methode en de parameters die erbij horen is hieronder te zien:

GetStuurmodulesAsync(**string** *appKey*, **string** *parameter*, **bool?** *locatieGekend*)

DataRecord

DataRecord zal alle data die teruggestuurd worden met de method Get in de klasse DataClient kunnen verwerken en in properties kunnen plaatsen. Deze method returnd een array van het type DataRecord, aangezien de method Get meerdere lijnen data kan teruggeven. Iedere index in de array zal dan een nieuwe lijn data voorstellen. DataRecord beschikt over volgende properties:

- Extra1
- Extra2
- Extra3
- Extra4
- Extra5
- InsertTime
- Tijdstip
- Value

Het gebruik van DataRecord in de code is hieronder te zien:

```
DataRecord[] records = client.Get("#appKey", 25, "rht", DateTime.Now.AddHours(-5), DateTime.Now);
```

Stuurmodule

De laatste klasse Stuurmodule zal alle modules die worden teruggegeven door de method GetStuurmodules van in de klasse DataClient bewaren. Opnieuw dient er eenzelfde manier van werken toegepast te worden zoals bij DataRecord, we dienen een array aan te maken voor alle mogelijke stuurmodules die terug kunnen gestuurd worden. De properties in de klasse Stuurmodule zijn als volgt:

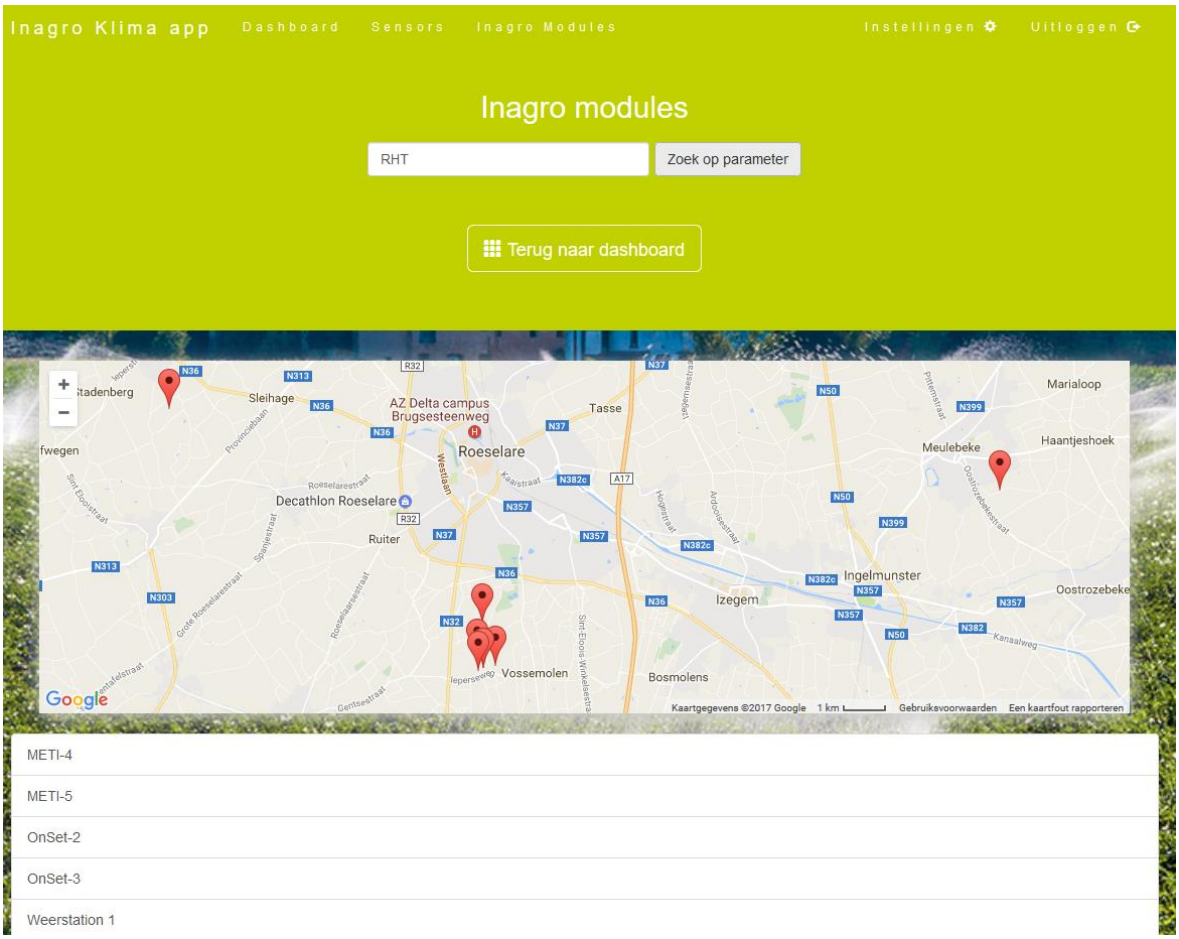
- Id
- Latitude
- Longitude
- Name

Het gebruik van de klasse Stuurmodule in de code is hieronder te zien:

```
Stuurmodule[] stuurmodules = client.GetStuurmodules("#appKey", "rht", true);
```

6.5.2 Gebruik in de applicatie

Deze WCF kan dan worden geïmplementeerd in de applicatie als zijnde dat de teler alle beschikbare modules kan te zien krijgen op een kaart, en de gewenste module kan selecteren en data kan verkrijgen van deze module. De modules die zichtbaar worden gemaakt voor de teler worden bepaald door de parameter die de teler zal meegeven. Een voorbeeld van hoe dit eruit ziet op de applicatie is in onderstaande afbeelding (zie Figuur 21) te zien.



Figuur 21: Weergave van Inagro sensoren in de webapplicatie

6.6 PDF-bestanden genereren

Op de pagina waar sensordata te vinden is, is er ook een mogelijkheid om hiervan een rapport te genereren in .pdf-formaat. Om dit te kunnen realiseren, maken we gebruik van “Rotativa” (Rotativa 1.6.4 NuGet, 24 maart 2015).

Rotativa is een library die het mogelijk maakt om in ASP.NET met het MVC design pattern pdf-bestanden te generen met behulp van een layout in HTML5.

Als voorbeeld is hieronder een stuk code in de SensorController weergegeven die een pdf-bestand teruggeeft als respons.

```
public ActionResult SensorToPDF(int? id)
{
        var list = new List<InagroSensors>();
        List<InagroSensorGroup> data = db.InagroGoups.Where(r => r.ID ==
                id).ToList();

        IEnumerable<InagroData> Meetdata = db.InagroSensorData.Where(r =>
                r.GroupId == id);

        List<string> commands = Meetdata
                .Where(e => e.GroupId == id)
                .Select(e => e.cmd)
                .Distinct()
                .ToList();

        ViewBag.commands = commands;

        ViewBag.SensorName = data[0].Name;
        ViewBag.SensorID = id;
        return new Rotativa.ViewAsPdf(){ FileName =
                "Sensor"+id+"_Created_"+DateTime.Now.ToString("yyyy-MM-d")+".pdf"
        };
}
```

Op de volgende pagina zie je een voorbeeld (zie Figuur 22) van een pdf-bestand dat aangemaakt geweest is door deze methode.

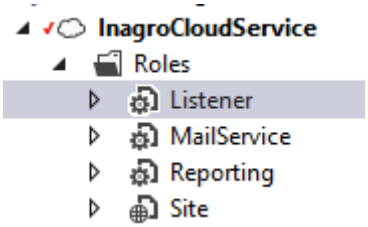
Figuur 22: Voorbeeld van pdf-rapport die gegenereerd is

7 Worker role

7.1 TCP Listener

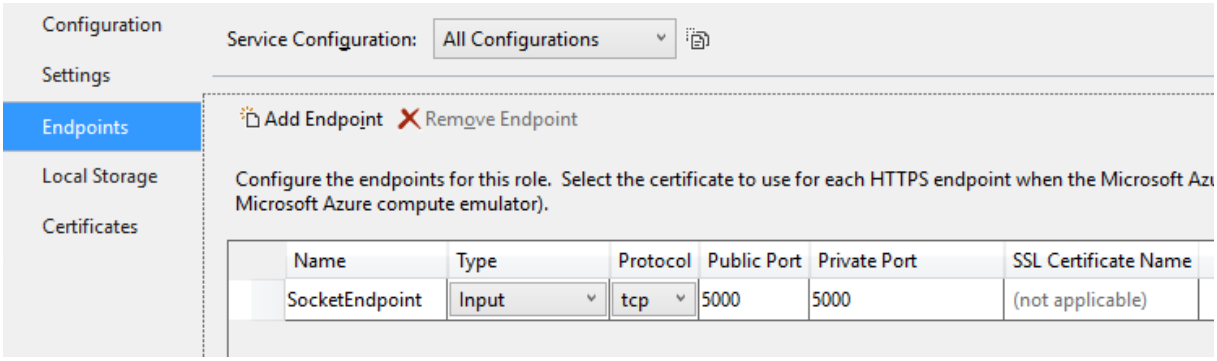
7.1.1 Socket port

In onze Azure Cloud Services wordt er een worker role voorzien die een TCP socket listener zal bevatten. Om ervoor te zorgen dat een toestel verbinding kan maken met deze socket, moet er een koppeling gebeuren van het public IP-adres naar het private IP-adres (waarop de socket listener draait). Dit kan door in de roles van onze Cloud services de properties van de listener role aan te passen, te zien op onderstaande Figuur 23.



Figuur 23: Listener in projectstructuur

Dit onder het tabblad “Endpoints” zoals weergegeven op onderstaande Figuur 24:



Figuur 24: Endpoint instellingen van de listener

Door een endpoint toe te voegen met “Public Port” gelijk aan 5000 en Private Port met hetzelfde poortnummer, zal het netwerkverkeer dat binnenkomt via poort 5000 op het publiek IP-adres, doorverwezen worden naar poort 5000 van de listener.

7.1.2 WorkerRole.cs

In de “WorkerRole.cs” staat de code voor onze TCP listener. Hierin wordt bij het opstarten de method Run() opgeroepen. In de method Run() starten we eerst de server socket op. Vervolgens zullen door een while(true) lopen waarin het volgende gebeurt:

```
while (true)
{
        clientSocket = serverSocket.AcceptTcpClient();
        HandleClient client = new HandleClient();
        client.startClient(clientSocket);

}
```

Er wordt hier voortdurend gewacht op een nieuwe connectie. Wanneer er een nieuwe connectie gemaakt wordt, wordt er een object gemaakt van de klasse “HandleClient”.

Hierin wordt de method startClient() opgeroepen die er als volgt uitziet:

```
public void startClient(TcpClient inClientSocket)
{
        this.clientSocket = inClientSocket;
        Thread ctThread = new Thread(HandleConnection);
        ctThread.Start();
        Trace.WriteLine(" >> New thread started");

}
```

We starten hierin dus een method in een nieuwe thread die de connectie verder zal verzorgen. Deze method bevat een while(true) die enkel de while-lus verlaat wanneer de connectie wordt verbroken.

Deze controle gebeurt als volgt:

```
if (!clientSocket.Connected)
{
        Trace.WriteLine(" >> Closing Thread: " +
                Thread.CurrentThread.ManagedThreadId);
        clientSocket.Close();
        break;

}
```

Wanneer de connectie niet verbroken is, lezen we de binnengekomen data in een try en catch blok. Het lezen van de data gebeurt met behulp van een object van het type NetworkStream waarin we de method Read() oproepen.

Wanneer we de data goed hebben ontvangen, kunnen we de binnengehaalde string parsen naar een object en dit vervolgens opslaan in de database met behulp van het Entity Framework.

7.2 Weerdata afhalen

In de applicatie wordt een systeem voorzien die de weersverwachtingen kan afhalen en opslaan, zodat op een later tijdstip de weersverwachtingen kunnen gebruikt worden bij het onderbouwde advies voor het al dan niet irrigeren.

7.2.1 Yahoo Weather API

Een weerservice die we gebruiken hiervoor is de “Yahoo Weather API”. Met behulp van deze API kunnen we op basis van de locatie allerhande data zoals de huidige weerssituatie en weersvoorspellingen terugkrijgen in JSON-formaat.

Temperatuur converteren

Voor het omzetten van Fahrenheit naar Celsius en omgekeerd, hebben we twee methods voorzien in een klasse WeatherConvertion. Hierin worden de basisformules gebruikt voor het omzetten van de grootheden.

```
public static decimal ConvertFahrenheitToCelsius(double FahrenheitArg)
{
	decimal MultiplyFactor = Convert.ToDecimal(5.0 / 9.0);
	return Convert.ToDecimal(Convert.ToDecimal(FahrenheitArg - 32) *
		MultiplyFactor);
}
```

```
public static decimal ConvertCelsiusToFahrenheit(double FahrenheitArg)
{
	decimal MultiplyFactor = Convert.ToDecimal(9.0 / 5.0);
	return Convert.ToDecimal(Convert.ToDecimal(
		Convert.ToDecimal(FahrenheitArg) * MultiplyFactor + 32) );
}
```

Afhalen van JSON data

In de method Read() maken we een string met URL die we gebruiken voor het afhalen van de JSON data. Deze URL ziet er als volgt uit:

```
string URL =
"https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecast%20wh
ere%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text=%27" +
LocationArg + "%27)&format=json";
```

Met “LocationArg” die wordt meegegeven als een variabele in de method. Deze “LocationArg” is de locatie van waar we de weerdata willen verkrijgen.

Vervolgens maken we een variabele “JSON_Response” aan waarin we de JSON-data plaatsen op volgende manier:

```
string JSON_Response;
using (WebClient client = new WebClient())
{
        JSON_Response = client.DownloadString(URL);
}
```

Met deze string kunnen we een object maken met de weersverwachtingen. Dit kunnen we doen door het deserializeren van de JSON-string:

```
this.WeatherData = JsonConvert.DeserializeObject<WeatherAPI>(JSON_Response);
```

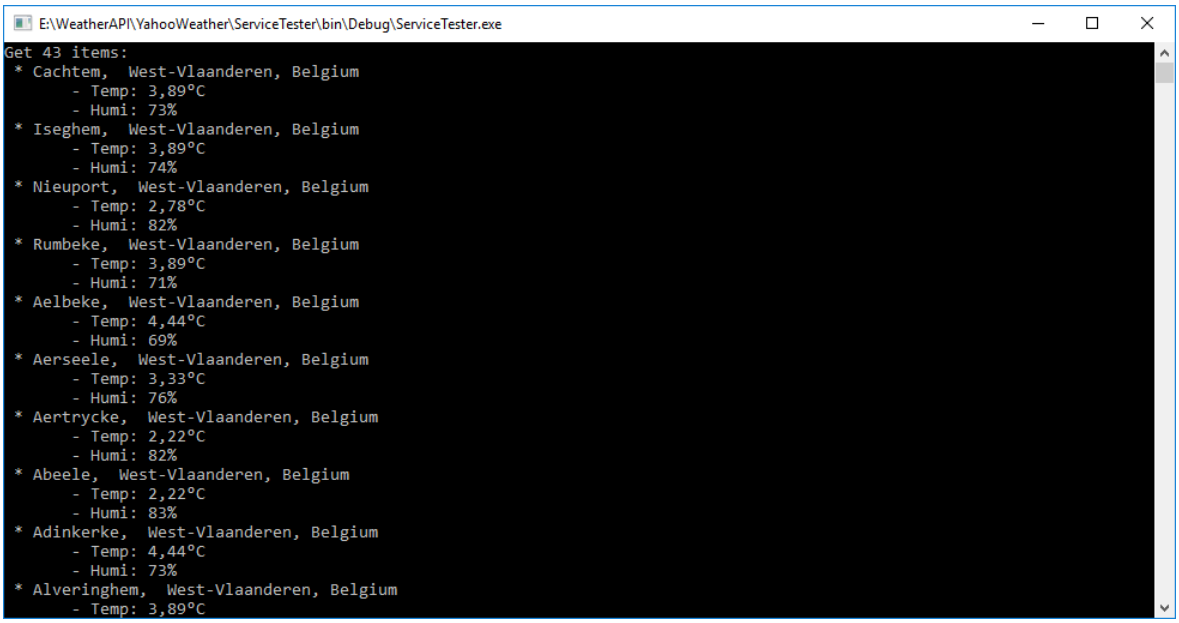
Gebruik van de .dll

Om de weerdata af te halen, dien je eerst een object te maken van het type “WeatherReader”.

Vervolgens kun je de method read(Dorp) (met Dorp de naam van de stad of het dorp waarvan je de weerdata wilt verkrijgen) oproepen van het object.

Test van de .dll

Om de .dll te testen hebben we van een lijst van dorpjes en steden enkele data opgevraagd. Deze data is zichtbaar op onderstaande afbeelding (zie Figuur 25).



Figuur 25: Print screen van testapplicatie voor weerdata

7.2.2 OpenWeatherMap

Aangezien er via de Yahoo Weather API geen gedetailleerde data beschikbaar zijn van de regenval en regenverwachtingen, hebben we gekeken om een extra service te gebruiken die deze data wel ter beschikking stelt. De API die dit wel mogelijk maakt is OpenWeatherMap.

Dataformaat

Idem zoals bij de Yahoo weather API is het formaat dat teruggestuurd wordt na een request in JSON-formaat.

Dit betekent dat het opnieuw mogelijk is om de JSON-data te parsen naar een object in C# via de Json.NET library van newtonsoft.

Reader klasse

Om op een eenvoudige manier de data van deze API te kunnen raadplegen, hebben we een klasse gemaakt die een request kan sturen naar de service en de JSON-response van deze request kan verwerken.

```
public class OpenWeatherReader
{
	public OpenweathermapData Data { get; set; }
	private string AppKey { get; set; }
	public string JsonData { get; set; }
	public OpenWeatherReader(string AppKeyArg)
	{
		this.AppKey = AppKeyArg;
	}

	public void Read(double Lat, double Lon)
	{
		NumberFormatInfo nfi = new NumberFormatInfo();
		nfi.NumberDecimalSeparator = ".";

		string URL = http://api.openweathermap.org/data/2.5/forecast?lat= +
		Lat.ToString(nfi) + "&lon=" + Lon.ToString(nfi) + "&appid=" +
		this.AppKey + "&units=metric";


		using (WebClient client = new WebClient())
		{
			this.JsonData = client.DownloadString(URL);
		}
		this.Data = JsonConvert.DeserializeObject
			<OpenweathermapData>(this.JsonData);
	}
}
```

7.3 Scheduler Quartz

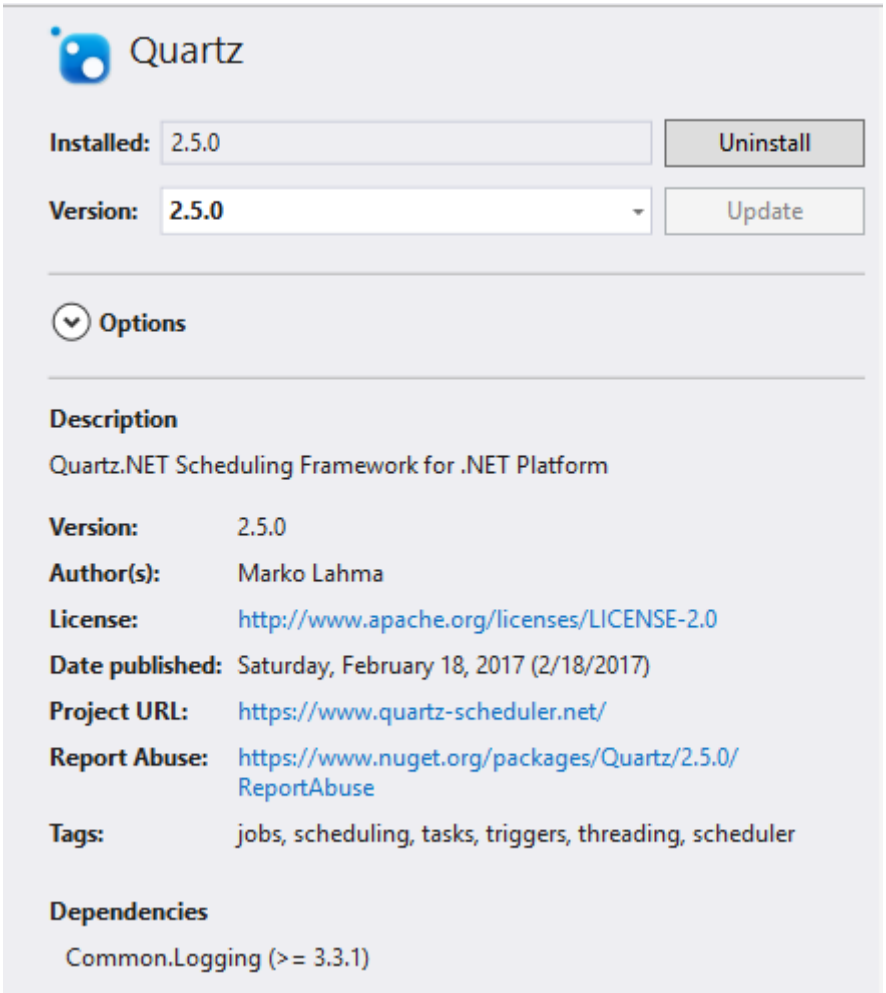
7.3.1 Quartz

Quartz (Quartz Enterprise Scheduler .NET, 18 februari 2017) is een open source library in .NET die geschreven is in C#. Met behulp van deze library is het mogelijk om in .NET taken automatisch uit te voeren op een bepaalde tijd.

Deze library gebruiken we in enkele worker rollen, om dagelijkse taken uit te voeren, of om op geregelde tijd data af te halen van externe bronnen die in de database moeten opgeslagen worden. Als voorbeeld van hoe deze library gebruikt wordt leggen we uit hoe de OpenWeatherData worker role om de vier uur data afhaalt van de ingestelde locaties in het platform.

7.3.2 Installeren van de library

Quartz.NET is beschikbaar in de NuGet Gallery en kan afgehaald worden via de NuGet Package Manager. In ons project maken we gebruik van de versie 2.5.0 (zie Figuur 26).



Figuur 26: Quartz library

7.3.3 Het aanmaken van een Job

Voor het aanmaken van een Job hoeven we enkel een nieuwe klasse aan te maken. In deze klasse implementeren we de interface IJob op deze manier:

```
internal class WeatherJob : IJob
```

Deze interface IJob heeft een method:

```
public void Execute(IJobExecutionContext context)
```

Deze method wordt automatisch opgeroepen volgens de ingestelde momenten in de scheduler.

We kunnen dus in de klasse WeatherJob de volgende method beschrijven:

```
public void Execute(IJobExecutionContext context)
{
        //DoJob
        inagrodbEntities db = new inagrodbEntities();
        OpenWeatherReader WeatherReader = new
                OpenWeatherReader("b0139671a5a8342a78738da20988c6eb");
        List<Location> Locs = db.Locations.ToList();
        foreach (var Loc in Locs)
        {
                WeatherReader.Read(Loc.Latitude, Loc.Longitude);
                OpenWeatherData data = new OpenWeatherData();
                data.LocationId = Loc.id;
                data.dt = DateTime.Now;
                data.Data = WeatherReader.JsonData;
                db.OpenWeatherDatas.Add(data);
                db.SaveChanges();
        }
}
```

7.3.4 Het instellen van de scheduler

Om ervoor te zorgen dat de aangemaakte job kan uitgevoerd worden op de gewenste momenten, moeten we hiervoor de scheduler in Quartz instellen zodat deze de method in de klasse WeatherJob kan uitvoeren.

Hiervoor voorzien we in de worker role een variabele van het type: IScheduler en noemen deze “sched”.

Om deze scheduler te configureren, maken we een method in de worker role, genaamd ConfigureScheduler(). Deze ziet er als volgt uit:

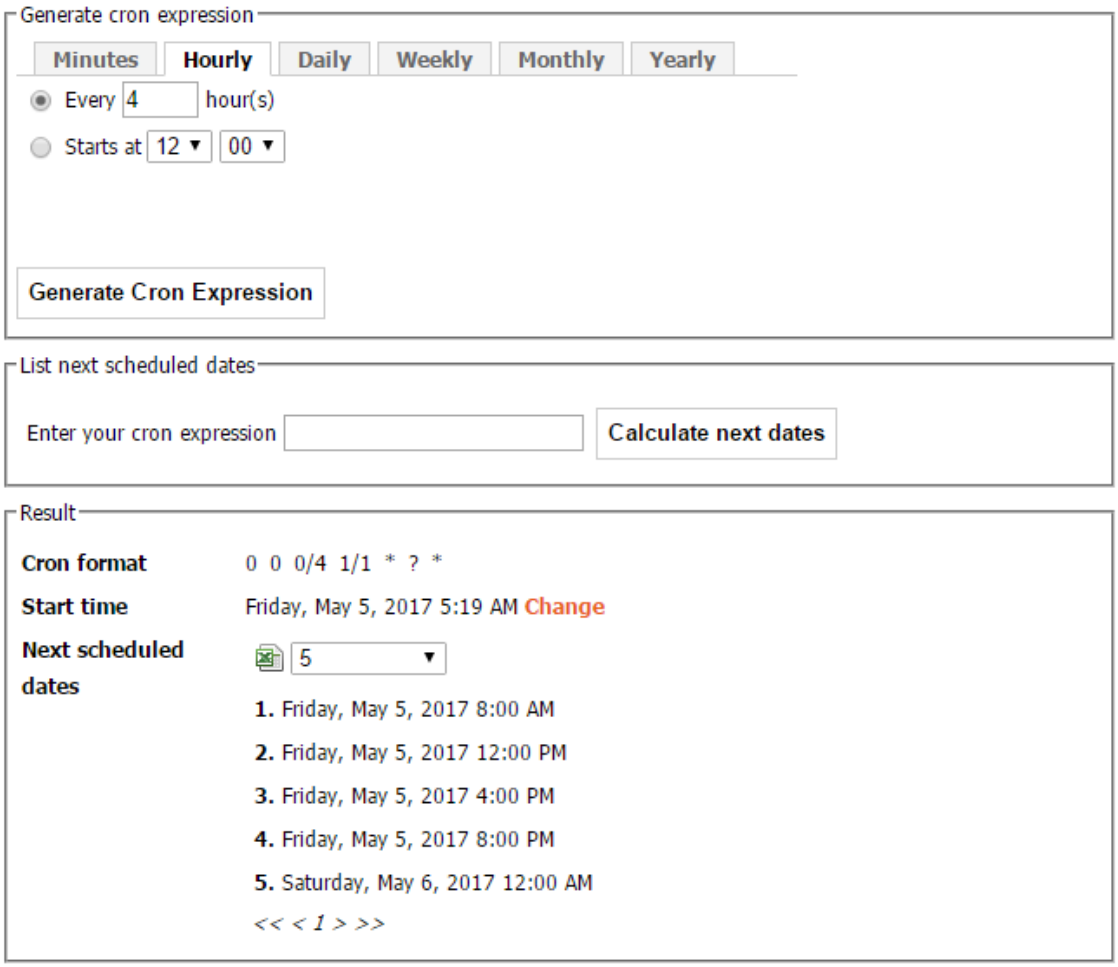
```
private void ConfigureScheduler()
{
        var schedFact = new StdSchedulerFactory();

        this.sched = schedFact.GetScheduler();
        var job = new JobDetailImpl("WeatherJob", null, typeof(WeatherJob));
        var timeZoneInfo = TimeZoneInfo.FindSystemTimeZoneById("Romance Standard
                Time");
        var cronScheduleBuilder = CronScheduleBuilder.CronSchedule(
                "0 0 0/4 1/1 * ? *").InTimeZone(timeZoneInfo);
        var trigger = TriggerBuilder.Create()
                .StartNow()
                .WithSchedule(cronScheduleBuilder)
                .Build();

        this.sched.ScheduleJob(job, trigger);
        this.sched.Start();

}
```

Het instellen van de momenten waarop de job moet uitgevoerd worden gebeurt via de CronScheduleBuilder. Hierin wordt een string meegegeven, waardoor de scheduler kan bepalen wanneer de job moet uitgevoerd worden.

Op de website[2] van cronmaker is een tool beschikbaar waarmee een string in Cron-formaat kan gegenereerd worden.

[2] http://www.cronmaker.com/

We wensen in de worker role om de vier uur weerdata af te halen. Hiervoor stellen we de volgende parameters in op de tool (zie Figuur 27).



Figuur 27: Cron web tool

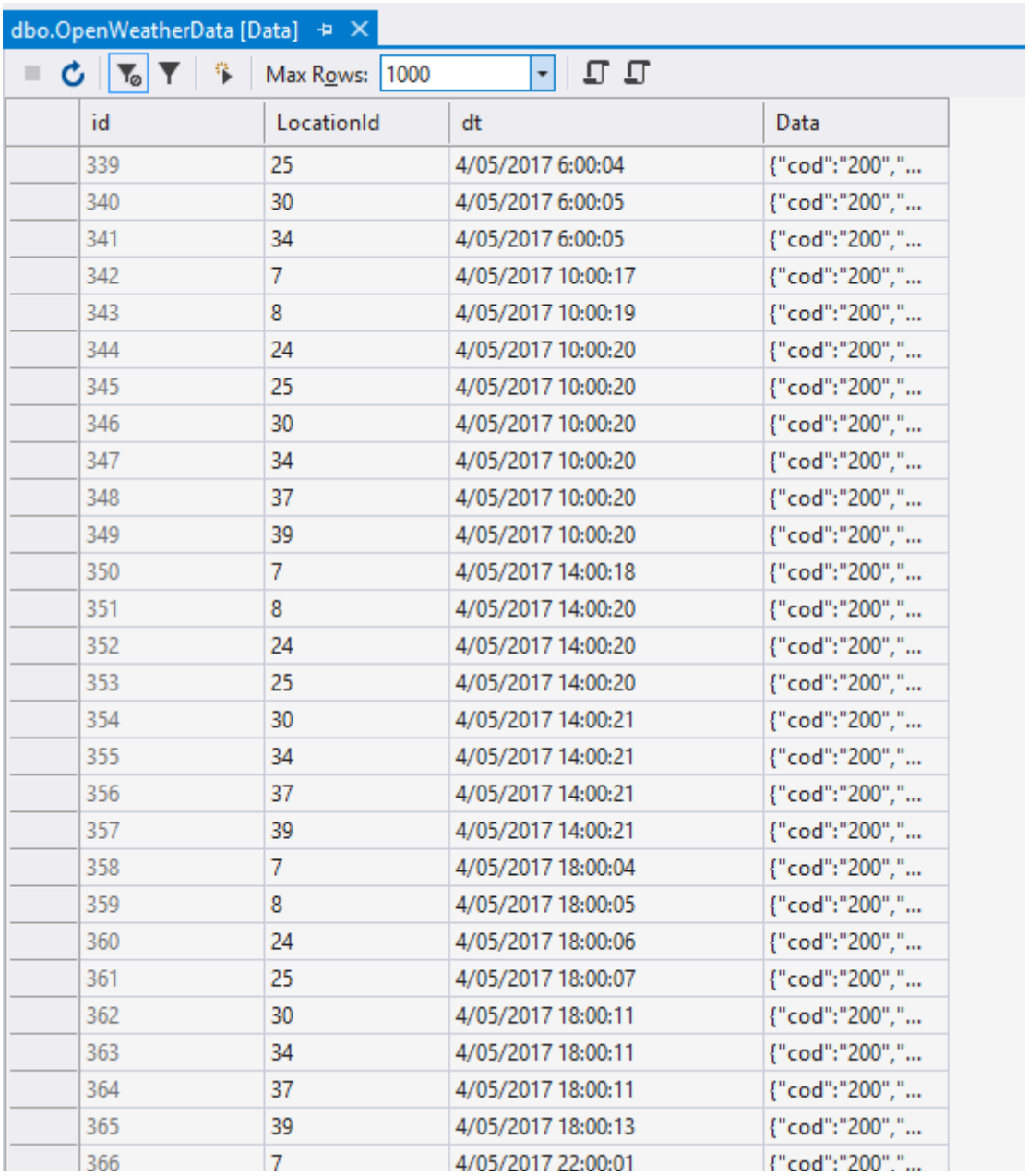
De string in Cron-formaat om dit te realiseren gaat als volgt: `"0 0 0/4 1/1 * ? *"`

Om ervoor te zorgen dat deze configuratie wordt ingesteld bij het opstarten van de worker role roepen we in de method OnStart() de method ConfigureScheduler() op.

7.3.5 Resultaat

Wanneer we de worker role laten draaien op Azure kunnen we vaststellen dat om de vier uur, de weerdata wordt afgehaald en opgeslagen in de database volgens zijn locatie Id.

Dit is terug te zien op onderstaande afbeelding (zie Figuur 28):



id	LocationId	dt	Data
339	25	4/05/2017 6:00:04	{"cod":"200","...
340	30	4/05/2017 6:00:05	{"cod":"200","...
341	34	4/05/2017 6:00:05	{"cod":"200","...
342	7	4/05/2017 10:00:17	{"cod":"200","...
343	8	4/05/2017 10:00:19	{"cod":"200","...
344	24	4/05/2017 10:00:20	{"cod":"200","...
345	25	4/05/2017 10:00:20	{"cod":"200","...
346	30	4/05/2017 10:00:20	{"cod":"200","...
347	34	4/05/2017 10:00:20	{"cod":"200","...
348	37	4/05/2017 10:00:20	{"cod":"200","...
349	39	4/05/2017 10:00:20	{"cod":"200","...
350	7	4/05/2017 14:00:18	{"cod":"200","...
351	8	4/05/2017 14:00:20	{"cod":"200","...
352	24	4/05/2017 14:00:20	{"cod":"200","...
353	25	4/05/2017 14:00:20	{"cod":"200","...
354	30	4/05/2017 14:00:21	{"cod":"200","...
355	34	4/05/2017 14:00:21	{"cod":"200","...
356	37	4/05/2017 14:00:21	{"cod":"200","...
357	39	4/05/2017 14:00:21	{"cod":"200","...
358	7	4/05/2017 18:00:04	{"cod":"200","...
359	8	4/05/2017 18:00:05	{"cod":"200","...
360	24	4/05/2017 18:00:06	{"cod":"200","...
361	25	4/05/2017 18:00:07	{"cod":"200","...
362	30	4/05/2017 18:00:11	{"cod":"200","...
363	34	4/05/2017 18:00:11	{"cod":"200","...
364	37	4/05/2017 18:00:11	{"cod":"200","...
365	39	4/05/2017 18:00:13	{"cod":"200","...
366	7	4/05/2017 22:00:01	{"cod":"200","...

Figuur 28: Resultaat in database van Quartz job

7.4 Mail alarmering

In de webapplicatie is het mogelijk dat de gebruiker simpele beslissingsregels instelt, deze regels maken het mogelijk om een automatische email te sturen wanneer een sensor een bepaalde ingestelde waarde heeft bereikt.

Het versturen van de mail gebeurt in een worker role met de naam “MailService”.
Hierin maken we gebruik van de .NET library “system.Net.Mail”. Deze library stelt ons in staat om in te loggen op een SMTP-server en vervolgens via het betreffende mailaccount een e-mail te sturen. Tijdens het stageproject, hebben we voor dit gedeelte gebruik gemaakt van Gmail als e-mailaccount.

Om ervoor te zorgen dat er niet herhaaldelijk in een lus e-mails worden verstuurd, hebben we gekozen om de tijd waarop een mail verstuurd wordt volgens een beslissing op te slaan. Zo kan er gecontroleerd worden of er deze dag al een mail verstuurd is, wanneer de alarmwaarde bereikt werd. Indien ja; zal er geen nieuwe mail meer verstuurd worden die dag, is dit echter niet het geval, dan wordt de mail meteen verstuurd naar het desbetreffende e-mailadres.

Dit komt er dus op neer dat iedere keer wanneer een mail verstuurd wordt, er voor deze alarminstelling de alarmeringstijd opgeslagen moet worden. Dit gebeurt met de zelf aangemaakte method die hieronder te zien is.

```
public void AlarmDone(int id)
{
	inagrodbEntities db = new inagrodbEntities();

	var result = db.SensorAlerts.SingleOrDefault(b => b.id == id);
	if (result != null)
	{
		result.TempDate = DateTime.Now;
		db.SaveChanges();
	}
}
```

De beslissing voor het versturen van een mail gebeurt in de method Run() zoals hieronder weergegeven:

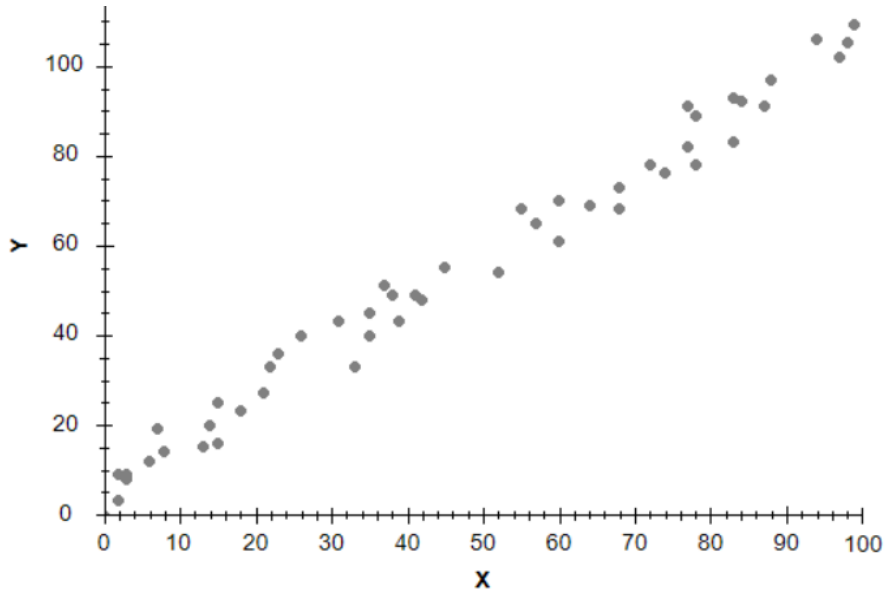
```
public override void Run()
{
        Trace.TraceInformation("MailService is running");
        inagrodbEntities db = new inagrodbEntities();

        while (true)
        {
                Thread.Sleep(5000);

                List<SensorAlert> Alarmeringen = new List<SensorAlert>();
                DateTime Vandaag = DateTime.Now;
                Alarmeringen = db.SensorAlerts.Where(A =>
                        DbFunctions.TruncateTime(A.TempDate) !=
                        DbFunctions.TruncateTime(Vandaag)).ToList();

                foreach (var Alarm in Alarmeringen)
                {
                        double? CurrentValue = GetFieldValue(Alarm.sensorId,
                                Alarm.Field);
                        switch (Alarm.check)
                        {
                            case ">":
                                if (CurrentValue > Alarm.value)
                                {
                                    SendMail(Alarm.MailTo, Alarm.MailText);
                                    AlarmDone(Alarm.id);
                                }
                                break;
                            case "<":
                                if (CurrentValue < Alarm.value)
                                {
                                    SendMail(Alarm.MailTo, Alarm.MailText);
                                    AlarmDone(Alarm.id);
                                }
                                break;
                            case "=":
                                if (CurrentValue == Alarm.value)
                                {
                                    SendMail(Alarm.MailTo, Alarm.MailText);
                                    AlarmDone(Alarm.id);
                                }
                                break;
                            default:
                                break;
                        }
                }
        }
}
```

De “`Thread.Sleep(5000);`” die in bovenstaande code gebruikt wordt, dient om de CPU in de Cloud service niet te overbelasten.

8 Beslissingsmodel

Om nu een beslissingsmodel te realiseren moeten we kunnen voorspellen wat de teler precies moet irrigeren bij bepaalde omstandigheden. Er bestaat een methode om de waternood van een bepaald gewas te berekenen volgens een formule die rekening houdt met de temperatuur, windsnelheid en verschillende andere factoren. Deze methode heeft als naam de FAO56 methode en doet een dagelijkse berekening van de vochtvraag tegenover een standaardgewas (= gras). Deze berekeningen gebeuren op basis van de data van verschillende weerstations in Vlaanderen. De waarde die uit deze FAO56 methode zal komen is dus de vochtvraag, wat dus betekent dat hierop een bepaalde irrigatie moet gebeuren. Hier zal het beslissingsmodel het overnemen.

Nadat de teler zelf enkele irrigaties heeft uitgevoerd volgens de waarde die de FAO56 methode leverde, kunnen er enkele punten uitgezet worden op een grafiek. Onderstaande afbeelding (zie Figuur 29) toont enkele van deze punten uitgezet op een grafiek.



Figuur 29: Datapunten, spreiding

Omdat we nog geen verdere informatie hebben gekregen omtrent de waarde die deze FAO56 methode zal meegeven, werken we hier voorlopig met x en y als waarden. Y kan de irrigatie voorstellen en x stelt hier dan de waarde van de FAO56 methode voor. Er zit dus een duidelijke spreiding op deze waarden. De bedoeling van het beslissingsmodel is om te proberen van de best passende lijn door deze verschillende waarden te trekken. Hierdoor kan de teler telkens een betere beslissing maken of hij dient te irrigeren of niet en indien ja, in welke hoeveelheid.

Een methode die we kunnen gebruiken om deze best passende lijn te trekken is om gebruik te maken van een regressieanalyse. Deze regressieanalyse wordt nu verder uitgelegd.

8.1 Regressieanalyse

Regressieanalyse (Regressie-analyse, in wikipedia, 9 mei 2017) is een techniek die gebruikt wordt voor het analyseren van gegevens op statistische wijze waarin (mogelijk) sprake is van een bepaalde samenhang van deze gegevens. Deze samenhang van gegevens is de regressie. Om niet te veel in detail te treden omtrent deze regressieanalyse zullen we kort schetsen wat dit precies inhoudt zonder het gebruik van bewijzen en formules.

De samenhang van gegevens wil eigenlijk zeggen dat de afhankelijke variabele die gebruikt wordt, afhangt van een of meer instelbare variabelen (deze kunnen eender wat zijn). De afhankelijke variabele wordt meestal met Y aangeduid en de onafhankelijke variabele met X. Het verband tussen deze twee variabelen is dan:

$$Y = f(X) + U$$

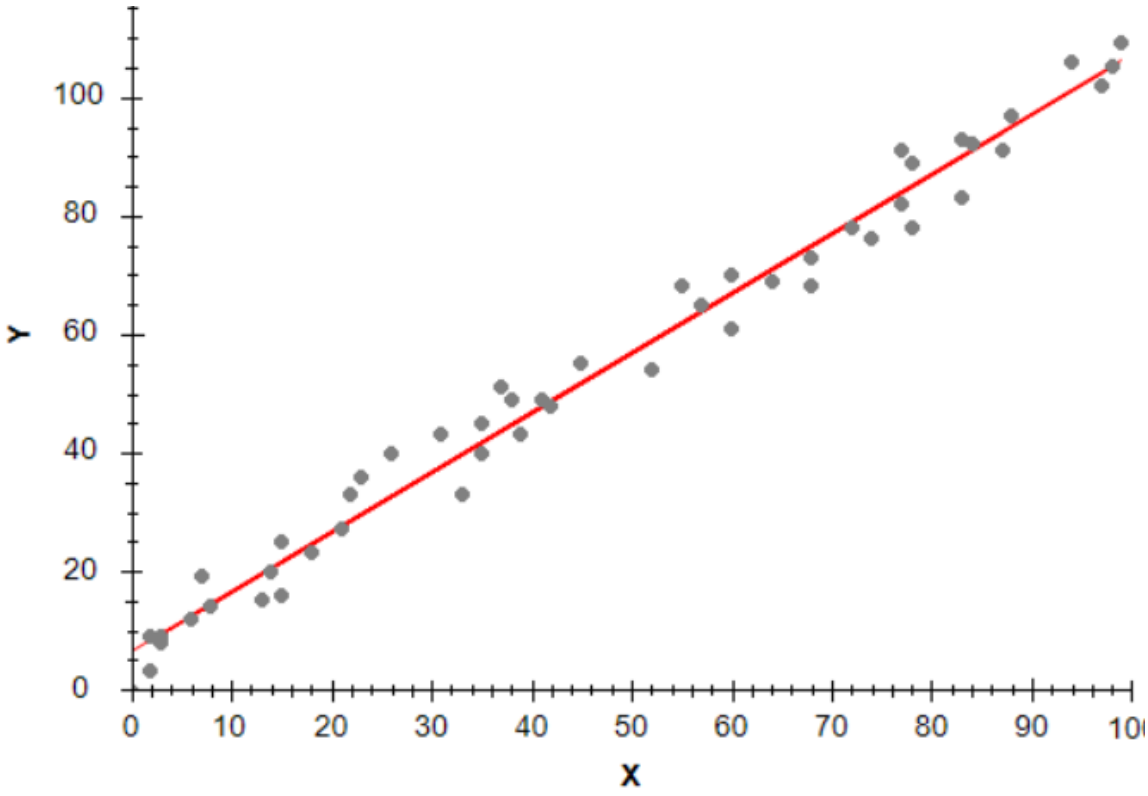
Hierin stelt U de storingsterm voor, deze is onafhankelijk van X. Hiermee nemen we dus aan dat de volledige variatie te wijten is aan een mogelijke fout in Y.

De functie f is in deze vergelijking onbekend, maar als we enkel en alleen kijken naar de toepassing voor regressieanalyse behoort deze functie wel tot een bepaalde klasse. Deze klasse kan namelijk beschreven worden met een beperkt aantal parameters. We spreken van lineaire regressie als de functie f een lineaire functie is van de verklarende variabelen.

We kunnen dit eenvoudig voorstellen:

- De gemiddelde waarde van alle X-waarden zal een waarde voor Y opleveren die dicht bij de gemiddelde Y-waarde ligt. Het punt ($X_{gemiddeld}$, $Y_{gemiddeld}$) is het uitgangspunt voor de lijn.
- De waarde van de helling van de lijn ligt hoogstwaarschijnlijk dicht bij de gemiddelde waarde van alle hellingen die ontstaan als elk meetpunt verbonden wordt met het punt ($X_{gemiddeld}$, $Y_{gemiddeld}$).

Een voorbeeld van hoe een lineaire regressie er kan uitzien is te zien op onderstaande afbeelding (zie Figuur 30).



Figuur 30: Gevisualiseerde regressie

8.2 Regressie in C#

We dienen deze regressie natuurlijk te verwerken in de webapplicatie aan de hand van C#-code. Maar aangezien de data niet beschikbaar zijn van de FAO56 methode zullen we een Windows Form applicatie maken die dient om de mogelijkheden van deze regressie en de werking ervan tentoon te stellen. Deze applicatie heeft onder meer bovenstaande afbeeldingen omtrent de regressie aangemaakt.

We maken gebruik van het Accord.Net NuGet pakket om deze regressie toe te kunnen passen. De gebruikte code is te vinden op de site van Accord.net. Uit de uitleg van wat regressie precies inhoudt hebben we geleerd dat we dus enkel een X- en Y-waarde nodig hebben om regressie toe te kunnen passen. De code ziet er dan als volgt uit:

```
//Ordinary Least Squares aanmaken om regressie aan te leren
OrdinaryLeastSquares ols = new OrdinaryLeastSquares();

//Lijn inschatten die door de punten (x, y) gaat
SimpleLinearRegression regression = ols.Learn(x, y);

//Bereken de waarden die voorspeld zijn door de regressie
//voor de oorspronkelijke input punten
double[] commonOutput = regression.Transform(x);
```

Hierin is ‘OrdinaryLeastSquares’ een klasse die gebruikt maakt van Machine Learning algoritmes, wat deze klasse precies doet zullen we hier niet bespreken. De method die we hier gebruiken om een regressie te bekomen is Learn(x,y). Je ziet dus dat deze method de eerder besproken X- en Y-waarden zal meenemen en hiervan dus een regressie zal maken, die we in een variabele met klasse ‘SimpleLinearRegression’ kunnen plaatsen, aangezien we al weten dat we hier een lineaire regressie willen bekomen. Ten laatste dienen we dan nog alle punten te berekenen die op de lijn liggen van deze regressie met behulp van de method Transform(x).

Daarna dienen we enkel nog deze punten uit te tekenen op de grafiek. Door deze in een PointPairList te plaatsen. Dit kan op volgende manier gecodeerd worden:

```
PointPairList  listOfRegression = new PointPairList(x, slr);
```

De punten kunnen we dan uittekenen in een grafiek en zo bekomen we de afbeelding die te zien is in Figuur 30 op de vorige pagina.

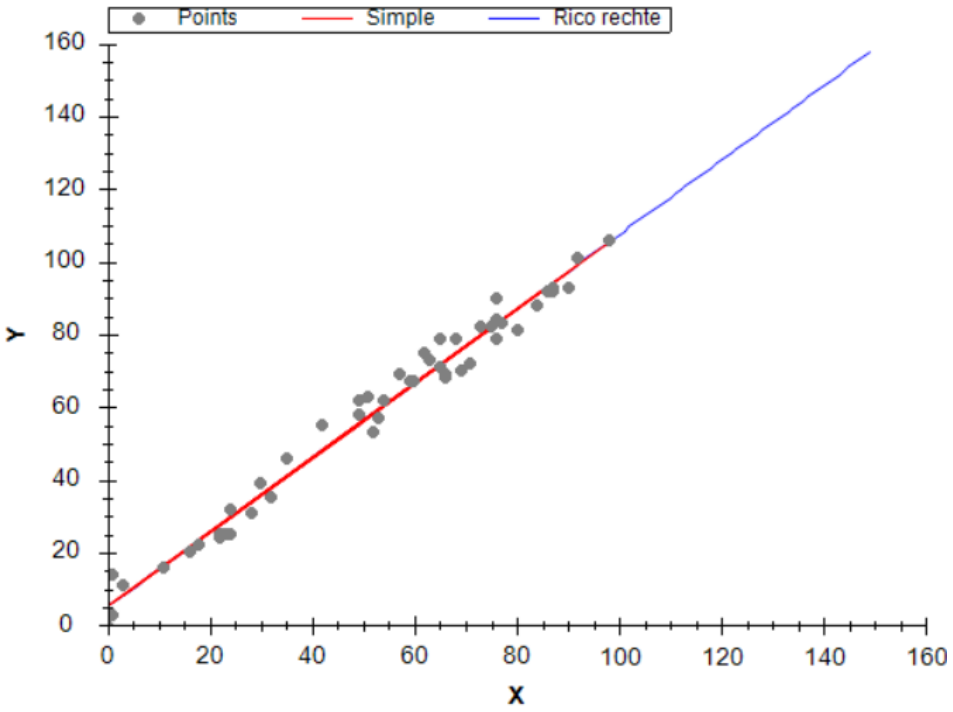
Belangrijk om op te merken is dat deze regressie enkel de best passende rechte zal tekenen tot het laatste gekende punt. Om verder te gaan dan dit laatste gekende punt, en dus een beslissing te kunnen maken op punten die nog niet bereikt zijn, moeten we gebruik maken van de richtingscoëfficiënt van de regressie rechte. Dit kunnen we bereiken door twee punten te nemen die op deze best passende rechte liggen en hiervan de richtingscoëfficiënt bepalen aan de hand van volgende berekening:

$$Richtingscoëfficiënt = \frac{Yb - Ya}{Xb - Xa}$$

We kunnen dit verwerken in de code op volgende manier:

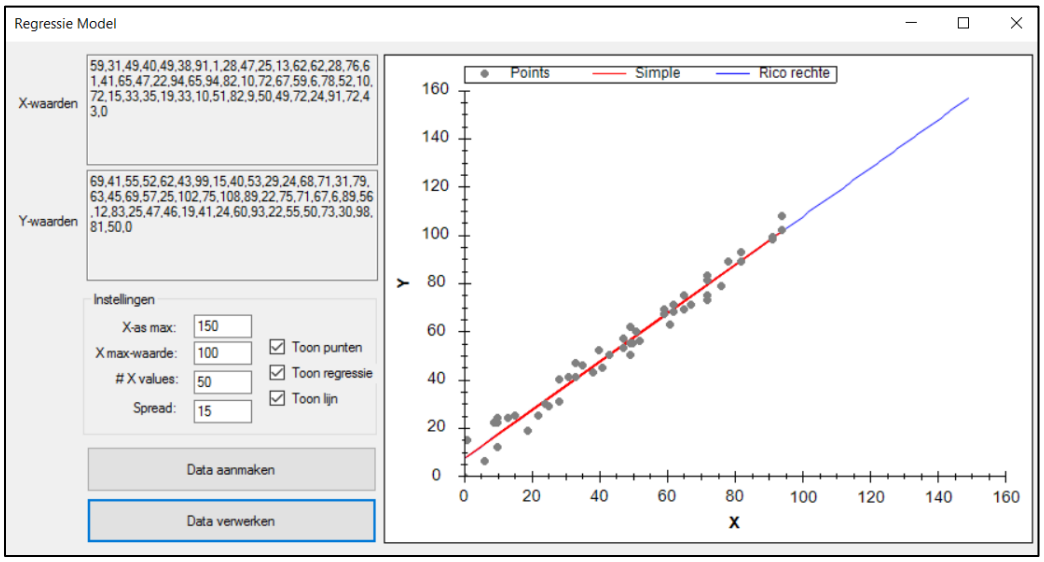
```
//Aanmaken rechte met rico
double Xa = x[0];
double Ya = commonOutput[0];
double Xb = x[x.Length - 2];
double Yb = commonOutput[commonOutput.Length - 2];

//Bereken rico
double rico = (Yb - Ya) / (Xb - Xa);

double[] XAs = new double[Max_XAs];
double[] yRico = new double[XAs.Length];

//Vul x-as
for (int i = 0; i < XAs.Length;i++){
	XAs[i] = i;
}

//Bereken rico waarden
for(int i=0; i<yRico.Length;i++){
	yRico[i] = (XAs[i] * rico) + commonOutput[NrofValues];
}

PointPairList ricoRechte = new PointPairList(XAs,yRico);
```

Zoals te zien in de code werken we met dezelfde berekening en dienen we de bekomen richtingscoëfficiënt enkel nog te verwerken om een rechte te bekomen die op de best passende rechte komt te liggen. Dit doen we door de richtingscoëfficiënt te vermenigvuldigen met alle waarden van de x-as en met een offset te geven waar de rechte begint bij x gelijk aan nul. De offset bekomen we door de laatste waarde te nemen van de regressie, aangezien deze waarde altijd berekend wordt vanuit het nulpunt (door de opbouw van de geschreven code). Een voorbeeld van hoe dit eruit ziet op de grafiek is op de volgende pagina afgebeeld (zie Figuur 31). Deze grafiek maakt duidelijk zichtbaar dat de zelfberekende rechte exact overeenkomt met de best passende rechte berekend door de regressieanalyse en ook correct verder doorloopt.



Figuur 31: Functie voorschrift visualiseren uit regressie lijn

Hoe deze Windows Form applicatie er precies uitziet is op onderstaande afbeelding (zie Figuur 32) te zien.

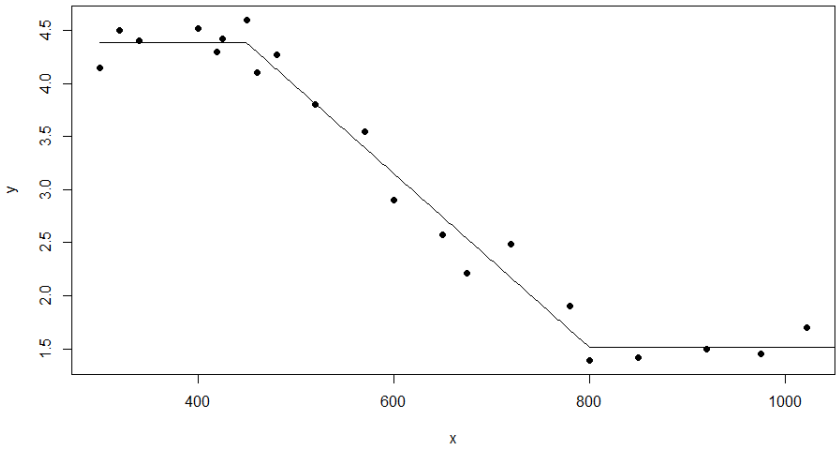


Figuur 32: Print screen Windows Form van regressie applicatie

Er kunnen dus enkele dingen ingesteld worden die interessant zijn om dit beslissingsmodel te kunnen tonen:

- De maximumwaarde van de x-as kan ingesteld worden zodat er kan gecontroleerd worden als de zelfberekende lijn een correct verloop toont;
- De maximumwaarde voor x;
- De aantal x-waarden die geplot dienen te worden;
- De standaarddeviatie die er mag optreden tussen de waarden.

Een opmerking die we hier dienen te maken is dat dit niet meteen voorstelt hoe het beslissingsmodel precies zal werken, en deze applicatie enkel dient om de mogelijkheid van lineaire regressie aan te tonen. Indien de FAO56 methode en de irrigatie die er dient te gebeuren geen lineair verloop kennen, is lineaire regressie hier niet meer van toepassing. Er kan dan echter wel gebruik gemaakt worden van ‘stuksgewijze regressieanalyse’ of ‘gesegmenteerde regressieanalyse’. Dit is feitelijk regressieanalyse die opgedeeld wordt om telkens de best passende rechte te trekken in bepaalde groepen (segmenten) van waarden. Een voorbeeld hiervan is te zien op onderstaande afbeelding (zie Figuur 33) (Roland, april 2016).



Figuur 33: Gesegmenteerde regressie

9 Testopstelling

Voor het testen van de applicatie werd er gedurende de stageperiode een testopstelling gerealiseerd. Deze opstelling wordt ook gebruikt als demo tijdens de presentatie over het stage/eindwerk. De testopstelling heeft als doel om een beeld te scheppen aan de bedrijven hoe de applicatie precies werkt. We zullen met deze testopstelling de gehele Internet of Things applicatie tonen door meteen ook eigen sensoren te gebruiken. De testopstelling zal dus zorgen voor een visualisatie van de gehele werking van het platform en zal dus zorgen voor een aantrekkelijkere demo om te tonen aan bedrijven/klanten.

9.1 DHT-sensor

Voor de testopstelling gebruiken we twee verschillende DHT-sensoren. Deze zijn de DHT11 en de DHT22. Met deze sensors kunnen we de temperatuur en de relatieve luchtvochtigheid meten.

9.2 Raspberry Pi

Als testtoestel voor het capteren van de data en het doorsturen naar de server, hebben we gekozen om gebruik te maken van een Raspberry Pi.

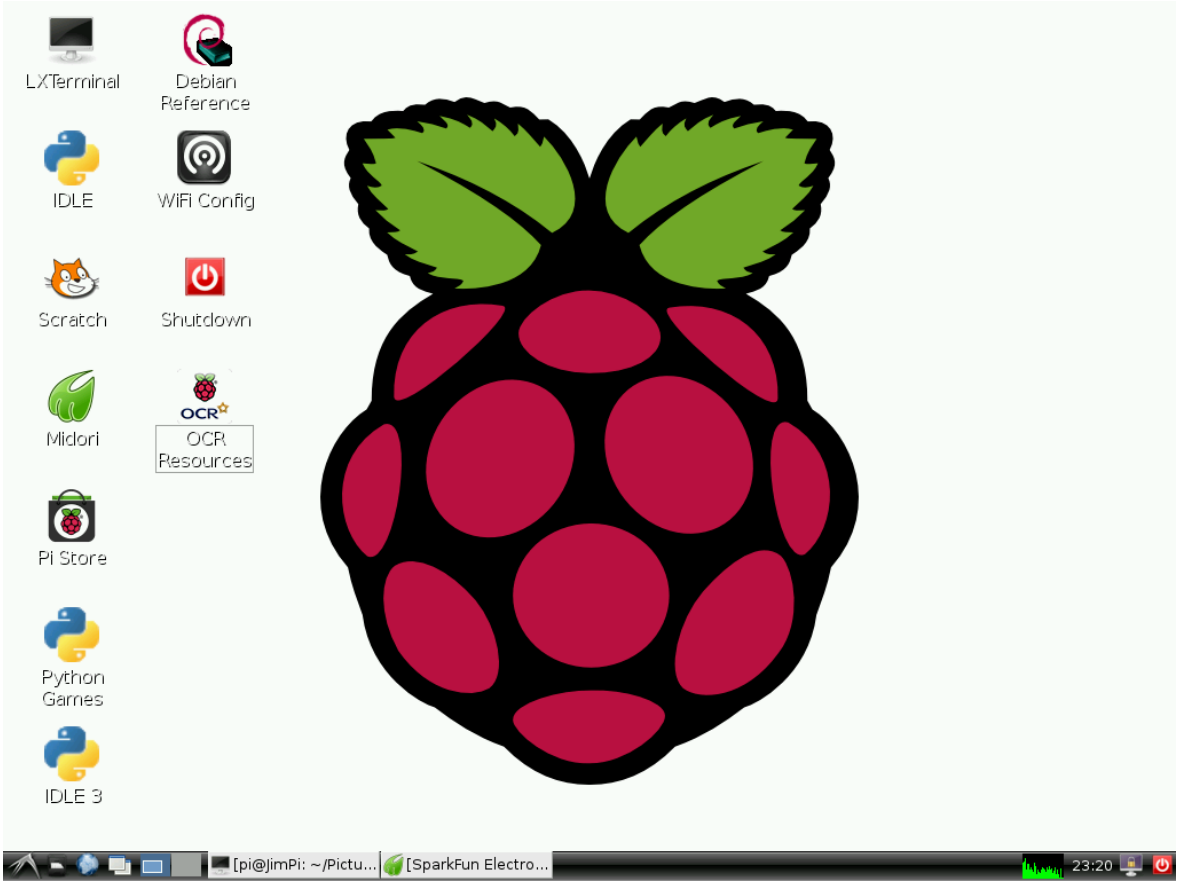
Een Raspberry Pi is een singleboard computer, voorzien van een ARM-processor. Dit is oorspronkelijk ontworpen voor educatieve doelstellingen, maar kan in onze stage perfect gebruikt worden als testopstelling.

Gedurende de stage hebben we twee besturingssystemen gebruikt voor het realiseren van de testopstelling. Enerzijds hebben we de Raspberry Pi gebruikt met het besturingssysteem Raspbian, wat standaard het meest gebruikt wordt. Anderzijds hebben we ook de Windows 10 IoT core geïnstalleerd. Deze twee besturingssystemen, en hoe we deze gebruiken in de testopstelling worden in de volgende puntjes besproken.

9.3 Raspbian

9.3.1 Linux

Raspbian is een Linux distributie die gebaseerd is op Debian, maar dan specifiek voor de Raspberry Pi met een ARM-processor. Raspbian maakt het mogelijk om een Raspberry Pi te gebruiken als een “volwaardige” computer. Een afbeelding (zie Figuur 34) van het bureaublad bij Raspbian is hieronder te zien.



Figuur 34: Print screen Raspberry Pi desktop

9.3.2 Python

Om de Raspberry Pi met het Linux systeem te kunnen gebruiken in de testopstelling, moet er een programma geschreven worden dat bepaalde data (temperatuur en vochtigheid) capteert en doorstuurt naar de TCP listener op Azure via een socket.

De programmeertaal die we zullen gebruiken om dit te realiseren is Python.
In dit programma zitten er drie belangrijke onderdelen:

- Sensor uitlezen;
- Data in dataformaat plaatsen;
- Data versturen via socket.

Sensor data uitlezen

Om de DHT-sensor te kunnen uitlezen met behulp van Python, maken we gebruik van een library van Adafruit. Deze kunnen we installeren in de command line in Linux door volgende commando's in te geven in de terminal:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git

cd Adafruit_Python_DHT

sudo apt-get update

sudo apt-get install build-essential python-dev python-openssl

sudo python setup.py install
```

Wanneer we gebruik willen maken van deze library hoeven we enkel bovenaan in ons script volgende lijn code te plaatsen:

import Adafruit_DHT

Sensortypes, kunnen we aanmaken in python door een array te maken. Deze array ziet er als volgt uit:

```
sensor_args = { '11': Adafruit_DHT.DHT11,
                '22': Adafruit_DHT.DHT22,
                '2302': Adafruit_DHT.AM2302
}
```

In de code kunnen we dan onze DHT-sensor aanmaken met de volgende lijn code:

DHT_Sensor = sensor_args[DHT_Type]

Hierin is "DHT_Type" een variabele met de waarde (string) "22". Dit geeft aan dat we gebruik zullen maken van een DHT22-sensor.

Het uitlezen van de temperatuur en vochtigheid met deze sensor kan gebeuren met behulp van deze lijn:

humidity, temperature = Adafruit_DHT.read_retry(DHT_Sensor, DHT_Pin)

Met hierin DHT_Sensor de sensor die we hadden aangemaakt op de vorige pagina, en DHT_Pin de pin waarop de sensor is aangesloten (hier is dit pin 4).

Volgens de standaard sensoren van Inagro wordt er in het *RHT-commando ook het dauwpunt verwacht. Dit kan berekend worden aan de hand van de twee waarden uitgelezen worden door de DHT-sensor (temperatuur en luchtvochtigheid).

Dit kan met behulp van volgende formule:

$$Td = \frac{243{,}12 \cdot A}{17{,}62 - A} \ (°C)$$

Waarbij:

$$A = \frac{\log_{10}(\frac{H}{100})}{\log_{10} 2{,}718282} + \frac{17{,}62 \cdot T}{243{,}12 \cdot T}$$

Waarbij:

T = Temperatuur in °C

H = Luchtvochtigheid in %

In python kunnen we deze wiskundige formule berekenen door gebruik te maken van de Math library. Deze moet eerst geïmporteerd worden door volgend lijntje code:

import math

Vervolgens kan in het script de formule berekend worden door deze code te plaatsen in het script als volgt:

```
Temp = math.log10(humidity/100) / math.log10(2.718282) +
                    (17.62 * temperature / (243.12 + temperature))

Td = 243.12 * Temp / (17.62 - Temp)
```

Data in het Inagro-dataformaat

Nadat de sensordata gelezen is, kan al deze data in één string verwerkt worden, zoals beschreven in het dataformaat van Inagro.

Dit kan met onderstaande lijn code:

*data = "*MAC,50500000001,ARDUINO_DHT11;*RHT,2," + str(timestamp) + ",DHT11,"*
*+ str(temperature) + "," + str(humidity) + "," + str(Td) + ";*EOD;"*

Hierin is timestamp, de Unix-timestamp die we eerder bekomen hadden met volgend commando:

timestamp = int(time.time())

Let op dat hiervoor het packet “time” moet geïmporteerd worden.

Data versturen via socket

Na het capteren van de data, en nadat alle data in de string in het correcte dataformaat zijn geplaatst, kunnen we overgaan naar het versturen van de data. Hiervoor maken we gebruik van een socket. Deze socket, zal verbinding maken met de TCP socket listener die zich bevindt op de Cloud services bij Azure.

Het connecteren met de TCP socket gebeurt met volgend commando:

sock.connect((HOST, PORT))

Waarbij:

- HOST: het IP-adres van onze Cloud services in Azure, hier is dit "23.97.164.58".

- PORT: de poort waarop de TCP socket in de services luistert, namelijk poort 5000.

Het versturen van de string (data) gebeurt door volgende functie uit te voeren:

sock.sendall(bytes(data + "\n"))

Wanneer we alle stappen van capteren tot en met versturen van data in een while-lus plaatsen, zullen deze voortdurend uitgevoerd worden.
Hierbij moet enkel nog een delay geplaatst worden die ervoor zorgt dat er maar om de x-aantal seconden data gecapteerd en verstuurd worden.

In de opstelling werd gekozen voor een interval van twintig seconden. Dit kan bekomen worden door in het begin van de while-lus volgende code te plaatsen:

time.sleep(20)

Hierdoor zal python twintig seconden wachten vooraleer de rest van het script wordt uitgevoerd.

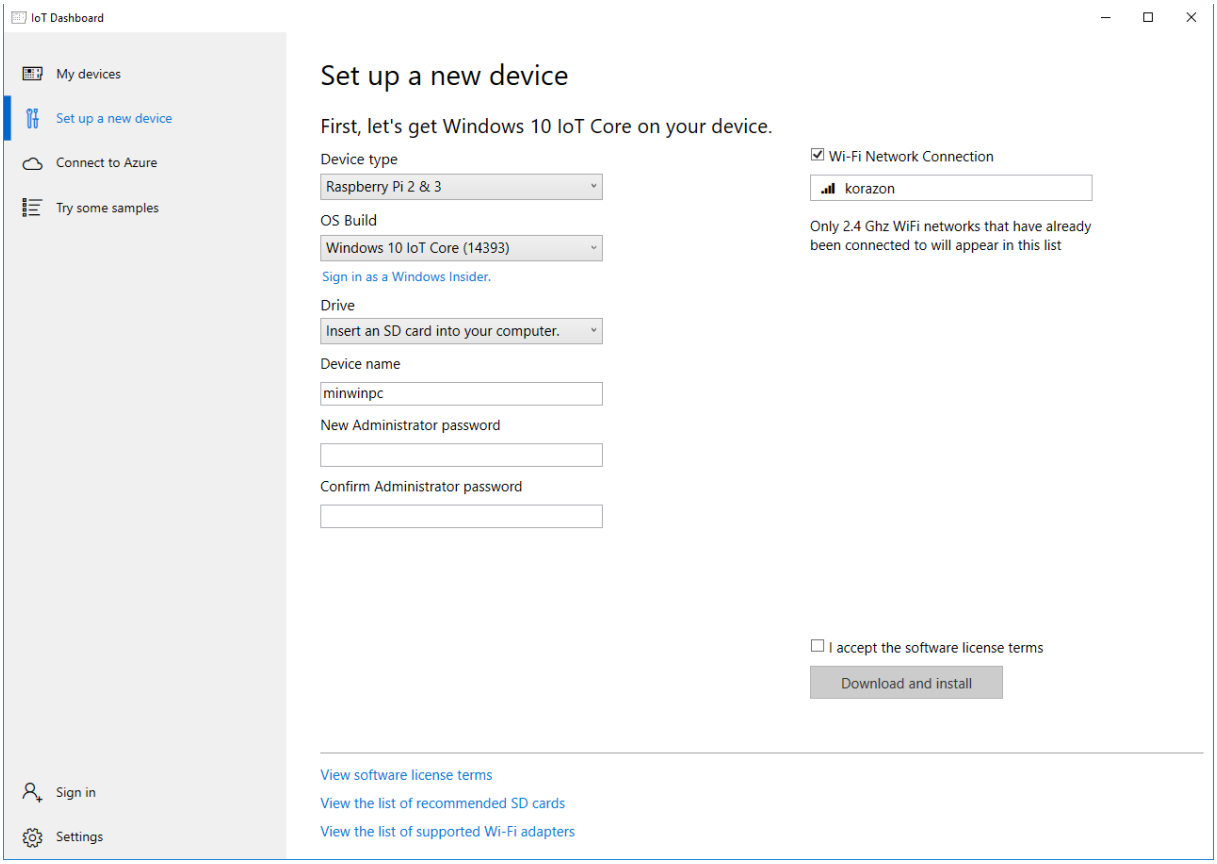
9.4 Windows 10 IoT core

Windows heeft een versie van Windows 10 die op een Raspberry Pi 2 en 3 kan werken. Deze versie van Windows 10 heet de Windows 10 IoT core. Dit is geen standaard Windows 10 versie zoals gewoonlijk geïnstalleerd is op de meeste pc's.

9.4.1 Installatie van Windows 10 IoT core

Voor het installeren van de Windows 10 IoT core op de SD-kaart, dient op voorhand het Windows 10 IoT Core Dashboard geïnstalleerd te worden. Deze tool kan gevonden worden op de site[3] van Microsoft zelf.

Na het installeren van het dashboard, kan je op het tabblad "Set up a new device" alles voorbereiden voor het klaarmaken van de SD-kaart. Dit is te zien op onderstaande afbeelding (zie Figuur 35).

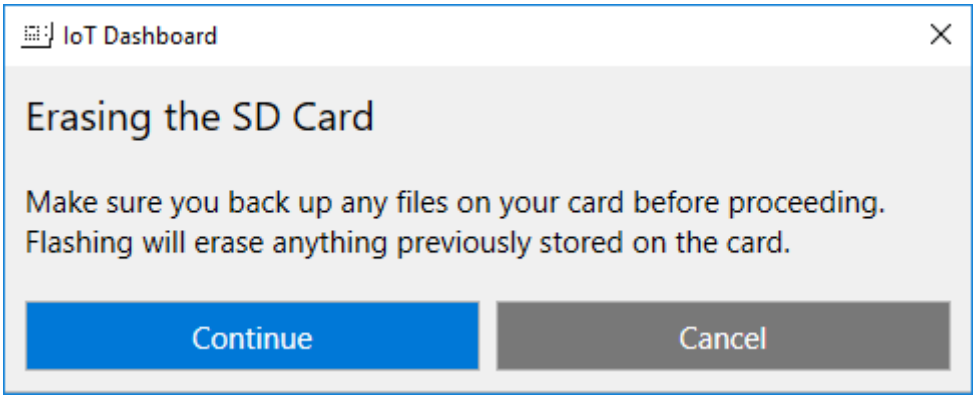


Figuur 35: Print screen IoT dashboard

Nadat alle parameters ingesteld zijn, hoef je enkel de licentie te accepteren en op "Download and install" te klikken.

[3]https://developer.microsoft.com/enus/windows/iot/Docs/GetStarted/rpi2/sdcard/stable/GetStartedStep1.htm

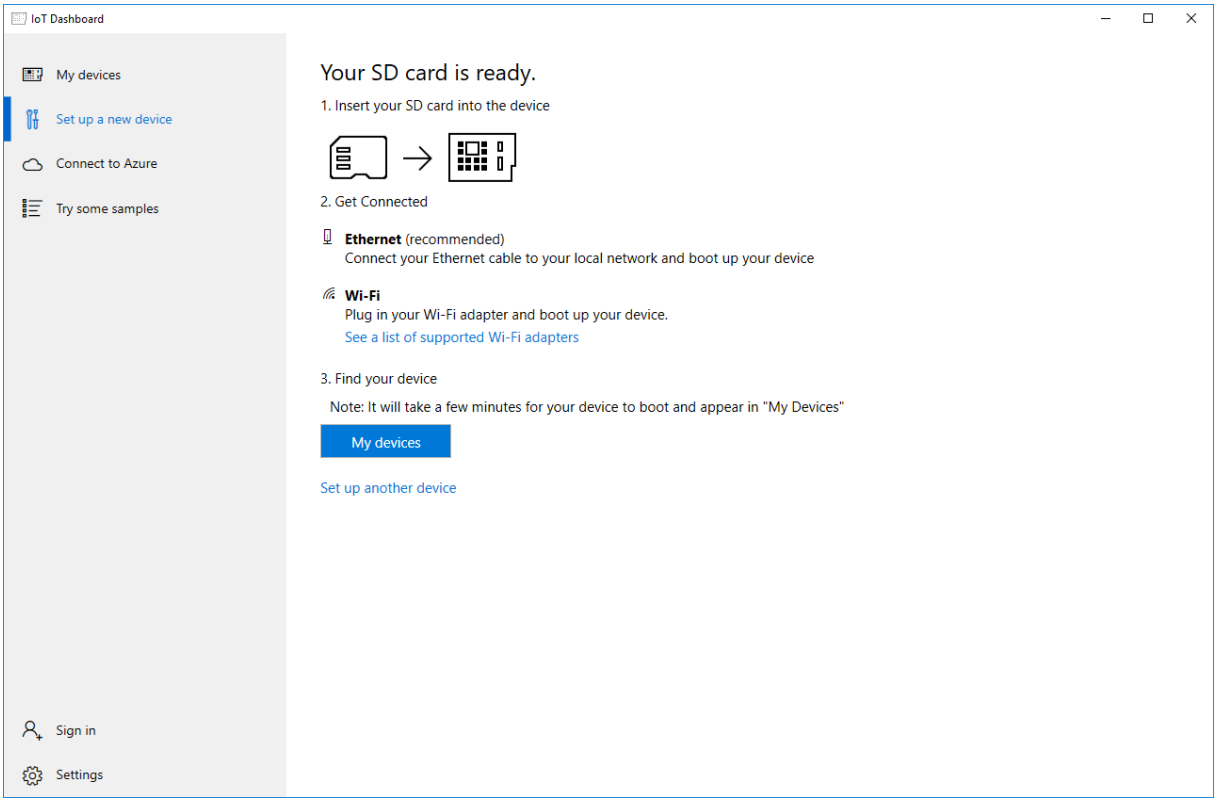
Wanneer er zich bestanden bevinden op de SD-kaart, krijg je het volgende scherm te zien (zie Figuur 36):



Figuur 36: Wissen van de SD kaart vooraleer schrijven

Bij het drukken op “Continue” worden alle bestanden op de SD-kaart gewist, en wordt vervolgens de image van de Windows 10 IoT core geplaatst op de SD-kaart.

Nadat de image geïnstalleerd is, dient de SD-kaart in de Raspberry Pi geplaatst te worden zoals weergegeven op onderstaande afbeelding (zie Figuur 37).



Figuur 37: Melding dat SD kaart geschreven is in IoT Dashboard

9.4.2 UWP-applicaties

De Universal Windows Platform (UWP) is het app platform voor Windows 10. Via dit app platform is het mogelijk om met slechts één API-set en één app-pakket alle Windows 10 apparaten te bereiken zoals: pc, tablet, Windows 10 telefoon, Xbox, Hololens, surface hub, Windows IoT core, etc.

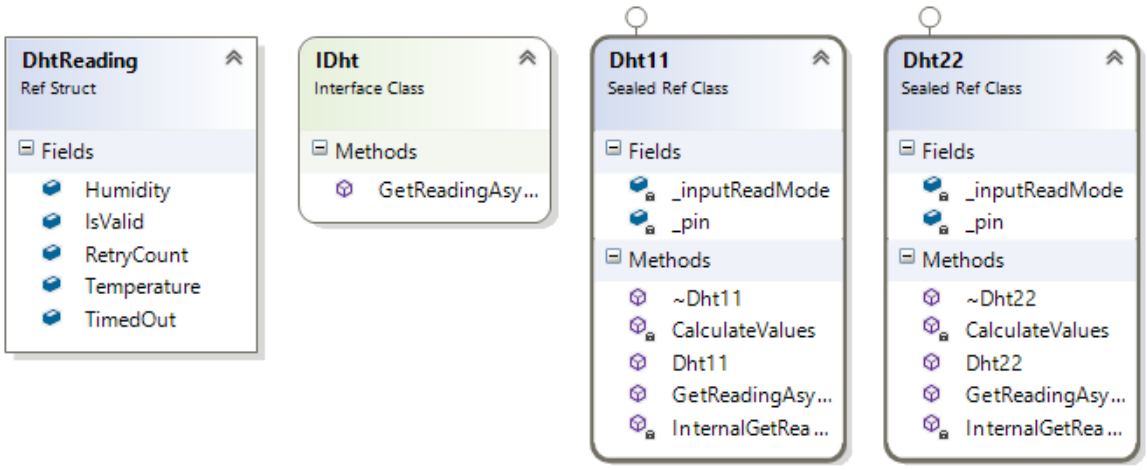
Voor het realiseren van een UWP-applicatie is het niet noodzakelijk om gebruik te maken van C# en XAML. Maar aangezien we de ASP.NET webapplicatie, en de worker roles in de Azure Cloud Services in C# gerealiseerd hebben, zullen we voor de eenvoud bij het programmeren van een applicatie op de Windows 10 IoT core ook gebruik maken van C# aanvullend met XAML voor de GUI layout van onze applicatie.

9.4.3 Demo applicatie

Als onderdeel van de demo opstelling hebben we voor de Windows IoT core een demo applicatie gemaakt. De functie van deze applicatie is het uitlezen van een DHT11-sensor (temperatuur en relatieve luchtvochtigheid). Deze data worden vervolgens grafisch voorgesteld en doorgestuurd naar het platform doormiddel van een socket en geplaatst in het huidige dataformaat van Inagro.

Klasses

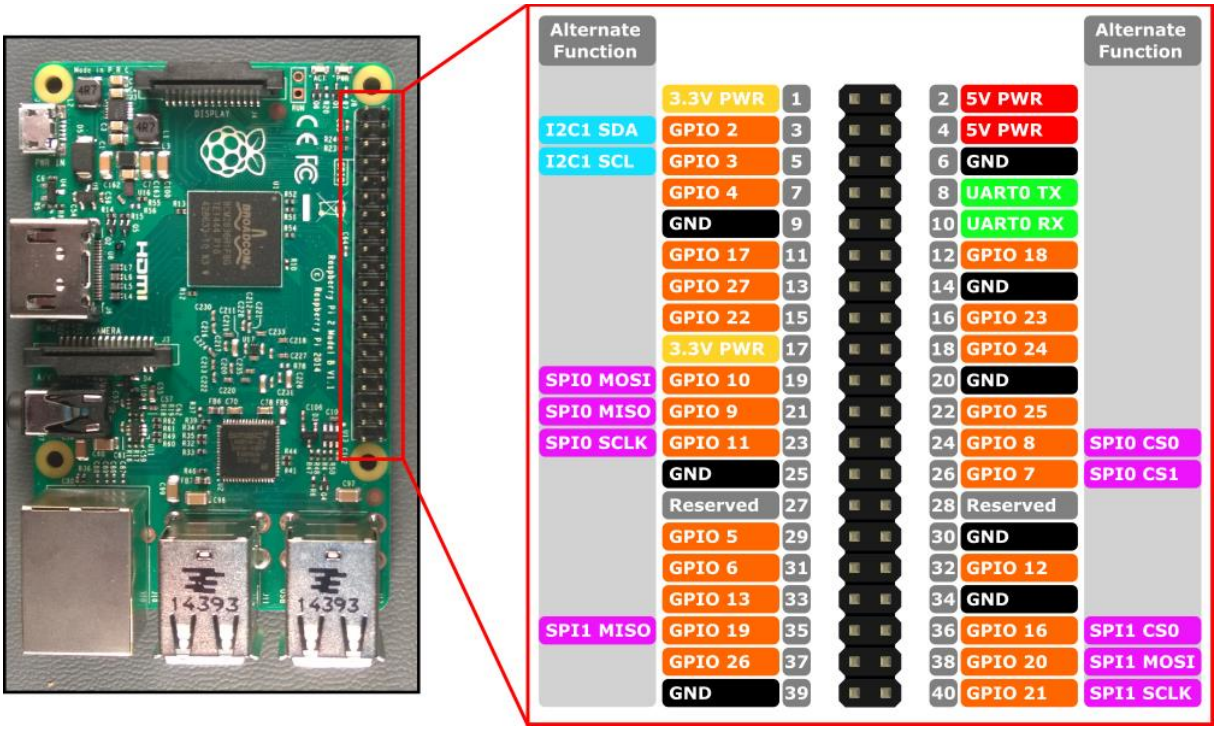
Om de UWP-applicatie eenvoudig te houden, hebben we gebruik gemaakt van een klasse waarin de benodigde functies staan voor het bekomen van de sensordata. Een klassediagram is hieronder weergegeven (zie Figuur 38).



Figuur 38: Klasses voor de benodigde sensordata te verwerken

GPIO pin

Om na te gaan welk GPIO-pinnummer in de code (C#) overeenkomt met welke fysieke pin op de Raspberry Pi, kan er gebruik gemaakt worden van onderstaande afbeelding (zie Figuur 39).



Figuur 39: Raspberry Pi GPIO-nummering

Voor deze opstelling die gebruik maakt van Windows 10 IoT core wordt de GPIO 4 pin gebruikt om de DHT11-sensor uit te lezen. Dit is pin 7 zoals te zien op het schema hierboven afgebeeld.

C# code

De constructor van onze “MainPage” ziet er als volgt uit:

```
public MainPage()
{
	this.InitializeComponent();

	InitHardware();

	_temperaturePin = GpioController.GetDefault().OpenPin(4,
		GpioSharingMode.Exclusive);
	_dhtInterface = new Dht11(_temperaturePin, GpioPinDriveMode.Input);

	_startedAt = DateTimeOffset.Now;

	ClockTimer = new DispatcherTimer();
	ClockTimer.Tick += ClockTimer_Tick;
	ClockTimer.Interval = new TimeSpan(0, 0, 1);
	ClockTimer.Start();

	dispatcherTimer = new DispatcherTimer();
	dispatcherTimer.Tick += dispatcherTimer_Tick;
	dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
	dispatcherTimer.Start();
	lvLogs.IsEnabled = false;
	addlog("App started");
}
```

Hierin is de functie InitHardware() als volgt:

```
private void InitHardware()
{
	_temperaturePin = null;
	_dhtInterface = null;
	_retryCount = new List<int>();
	_startedAt = DateTimeOffset.Parse("1/1/1");

}
```

Het event “ClockTimer_Tick” dient om het label waarin de huidige tijd staat te vernieuwen. Dit gebeurt als volgt in de code:

```
private void ClockTimer_Tick(object sender, object e)
{
	lblClock.Text = DateTime.Now.ToLocalTime().ToString();

}
```

In het event “dispatcherTimer_Tick” wordt de sensordata uitgelezen en de nodige data in de labels en/of in de logging list geplaatst.

```
private async void dispatcherTimer_Tick(object sender, object e)
{
	try
	{
		DhtReading reading = new DhtReading();


		reading = await _dhtInterface.GetReadingAsync().AsTask();

		_retryCount.Add(reading.RetryCount);

		if (reading.IsValid)
		{
			lblTemp.Text = "T: " + reading.Temperature.ToString() + "°C";
			lblHum.Text = "H: " + reading.Humidity.ToString() + "%";

			lblStatus.Text = "Running";
			lblStatus.Foreground = new  SolidColorBrush(Colors.White);
			//this.LastUpdated = DateTimeOffset.Now;
		}
		 else // log if the reading is not in valid state
		{
			addlog("No correct data from DHT");
			lblStatus.Text = "Error";
			lblStatus.Foreground = new SolidColorBrush(Colors.Red);
		}
	}
	catch (Exception ex) // log any exception that occurs
	{
		addlog("Error by reading data from DHT");
	}
}
```

In de code wordt verschillende keren de functie addlog (string LogArg) gebruikt. Deze functie zorgt ervoor dat de nodige data in de logging list geplaatst wordt. De functie wordt hieronder weergegeven.

```
private void addlog(string LogArg)
        {
            if (lvLogs.Items.Count > 3)
            {
                lvLogs.Items.RemoveAt(1);
            }

            lvLogs.Items.Add(DateTime.Now.ToString() + " >> " + LogArg);

        }
```

Als laatste zijn er nog enkele event handlers voorzien voor het reageren op events van drukknoppen. Twee hiervan zijn hieronder te zien:

```
private void btnGraphBack_Click(object sender, RoutedEventArgs e)
{
       GridGraph.Visibility = Visibility.Collapsed;
       GridMenu.Visibility = Visibility.Visible;
}

private void btnShowGraph_Click(object sender, RoutedEventArgs e)
{
       GridMenu.Visibility = Visibility.Collapsed;
       GridGraph.Visibility = Visibility.Visible;
       ImgaGraphView.Source = new Windows.UI.Xaml.Media.Imaging.BitmapImage(new
              Uri("http://inagrocloud.cloudapp.net/Sensor/ChartFromData/3",
              UriKind.Absolute));

}
```

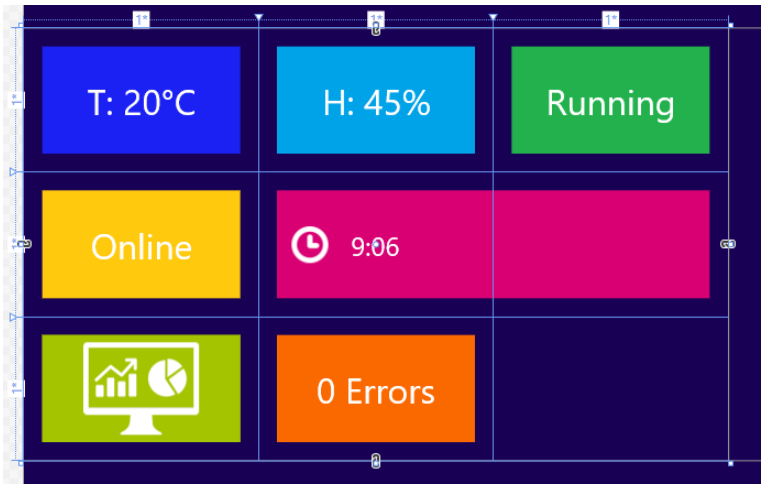
Applicatie layout

Hieronder is een voorbeeld (zie Figuur 40) te zien van de algemene stijl die we gebruiken voor onze applicatie op de Raspberry Pi met Windows 10 IoT core op geïnstalleerd.

Figuur 40: Layout van de UWP applicatie

Zoals eerder vermeld is deze applicatie gemaakt in C#, en de lay-out in XAML. In XAML hebben we gebruik gemaakt van verschillende grids om verschillende gebieden af te bakenen en verhoudingen in te stellen voor elk gebied.

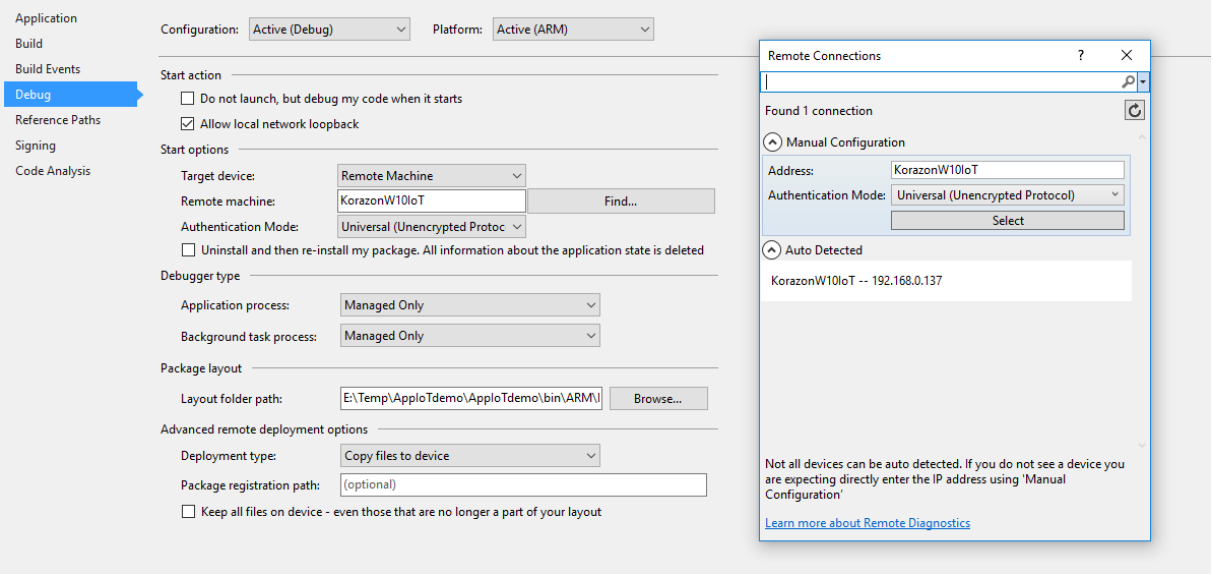
Hieronder is een afbeelding (zie Figuur 41) weergegeven die de grid layout toont in het “tegel gedeelte” (= gedeelte in de applicatie waarin de verschillende blokken met knoppen en info staat).



Figuur 41: Grid lay-out in de UWP applicatie

Initialiseren van applicatie op de Raspberry Pi

Voor het initialiseren en debuggen van de applicatie op de Raspberry Pi, moet bij de projectinstellingen (zie Figuur 42) ingesteld worden dat het target device een “remote machine” is. Wanneer de Raspberry Pi met Windows 10 IoT core in hetzelfde netwerk zit als de pc waarop er geprogrammeerd wordt, wordt deze standaard meegegeven als remote machine.



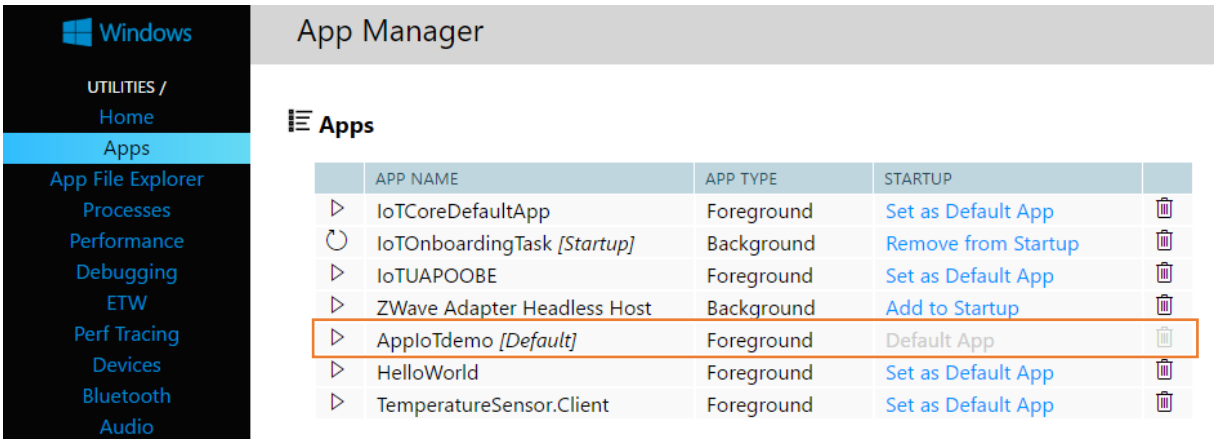
Figuur 42: Print screen instellingen VisualStudio voor programmeren Raspberry Pi

Instellen als startup applicatie

Via de web interface kan de startup applicatie ingesteld worden. De startup applicatie is de applicatie die standaard wordt opgestart bij het opstarten van de Raspberry Pi.

De web interface kan bekomen worden door in de browser te surfen naar de hostname van de Pi, op poort 8080. Hier is dit: “http://korazonw10iot:8080/”.

Vervolgens kan er op de pagina “Apps” (zie Figuur 43) ingesteld worden dat een bepaalde applicatie als “startup”-applicatie moet dienen door deze app als “Default App” in te stellen. Zoals hieronder weergegeven voor de applicatie “AppIoTdemo”.



Figuur 43: Web interface van Windows IoT core

10 Algemene besluiten

10.1 Keuze platform

Tijdens het voortraject hadden we drie spelers op de markt onderzocht die IoT en Cloud platformen aanbieden. Bij deze keuze van het platform was het dus zeer belangrijk om te kijken naar de toepassing die gerealiseerd moest worden en naar de noden van het platform. Zoals vastgesteld tijdens de analyse, zijn er ook platformen die zich voornamelijk focussen op het automatiseringsgedeelte binnen Internet of Things, en hierbij dus minder voorzieningen hebben op vlak van dataopslag en methoden om data uit een database te analyseren. Na uitgebreid onderzoek en in overleg met Inagro besloten we om gebruik te maken van het Azure Cloud platform om deze toepassing op te realiseren.

Na het realiseren van deze applicatie kunnen we vaststellen dat in deze toepassing de keuze van Azure als platform een goeie keuze was waarop alles wat benodigd was in de applicatie kon gerealiseerd worden.

10.2 Standaarden in IoT

Met behulp van de gegevens die we opgezocht hebben tijdens de analyse van verschillende IoT platformen in het voortraject, kunnen we vaststellen dat ondanks de sterkere standaardisatie binnenin de wereld van Internet of Things er toch nog duidelijke verschillen zijn op gebied van dataformaat en/of protocollen die gebruikt worden voor de communicatie tussen het toestel en het platform. Deze verschillen treden op tussen de verschillende platformen. Daarom geven de meeste leveranciers van een platformservice hun eigen SDK mee die gebruikt kan worden voor het programmeren van het toestel dat je wilt laten communiceren met het platform.

Om een teler de mogelijkheid te geven om zelf eigen sensoren aan het systeem toe te voegen, hebben we in de webapplicatie een systeem opgebouwd, die dit mogelijk maakt. Deze mogelijkheid geven aan een teler zorgt ervoor dat er al heel wat gestandaardiseerd wordt in verband met deze eigen sensoren, aangezien elke mogelijke sensor ingegeven kan worden.

10.3 Beslissingsmodel

In de webapplicatie is een systeem voorzien dat telers de mogelijkheid biedt om de irrigatiehoeveelheid in te geven op een bepaalde locatie. Op deze locatie is ook de gewassoort bekend.

In de toekomst kunnen door middel van het toepassen van lineaire regressie op de opgeslagen data, adviezen gegenereerd worden. Door deze vorm van Machine Learning wordt zo de irrigatie hoeveelheid geoptimaliseerd.

10.4 Internet of Things

In hoofdstuk 3 zagen we op Figuur 7 de vijf hoofdblokken die plaatsvinden in een IoT netwerk. Wanneer we alle besproken punten in deze scriptie eens bekijken, kunnen we besluiten dat deze punten perfect aansluiten binnen het IoT netwerk volgens de vijf hoofdblokken.

1. **Data capteren**

   In hoofdstuk 9, waar we de testopstelling besproken, hebben we aangetoond dat het mogelijk is om een Raspberry Pi te gebruiken om data te capteren voor het betreffende IoT netwerk. Het is ook mogelijk om op een ruimere manier data te verkrijgen. Zoals in hoofdstuk 7.2 werd aangehaald, kan er ook gebruik gemaakt worden van externe services om bijvoorbeeld weerdata binnen te halen.

2. **Transport**

   De datatransport methode die gebruikt werd tijdens de testopstelling is WiFi, om ervoor te zorgen dat het platform en de sensor met elkaar kunnen communiceren, werd een TCP listener gemaakt. Hierdoor is het mogelijk om de connectie tussen sensor en platform te verzorgen in een TCP-socket.

3. **Platform**

   Zoals in hoofdstuk 5 werd besproken, maken we gebruik van het Azure Cloud platform waarin we Cloud services gebruiken.

4. **Data-analyse**

   De data-analyse in het IoT netwerk kan gaan van kleine zaken, zoals de beslissing voor het versturen van een mail zoals in hoofdstuk 7.4, tot iets complexere zaken zoals het toepassen van regressie zoals in hoofdstuk 8.

5. **Visualisatie**

   De visualisatie die we hebben toegepast, gaat van de grafieken die in de webapplicatie worden getoond, tot de e-mails die automatisch verstuurd worden bij het bereiken van een alarmwaarde.

Verklarende woordenlijst

Begrip	Definitie/verklaring
ARM-processor	De ARM-architectuur is een processorarchitectuur van de Britse computerfabrikant Acorn Computers Ltd. De naam ARM staat voor Acorn RISC Machine, waarbij RISC een afkorting is van Reduced Instruction Set Computer.
ASP.NET	Framework die gebruikt wordt in .NET om websites te creëren.
Big data	Big data is een term die gebruikt wordt wanneer er gesproken wordt over gigantische hoeveheelheden data.
Checksum	Een som die dient om na te gaan of alle data die doorgestuurd werden goed zijn aangekomen, zonder eventuele fouten.
Cloud computing platform	Een platform dat toelaat om via het internet verschillende services aan te bieden, onder deze services kan zowel hardware als software zitten.
CMS	Content management system: een softwaretoepassing, meestal een webapplicatie, die het mogelijk maakt dat mensen eenvoudig, zonder veel technische kennis, documenten en gegevens op internet kunnen publiceren.
Dashboard	De hoofdpagina van een site, hier worden alle belangrijke delen van de site op getoond.
Design pattern	Een design pattern/ontwerppatroon wordt gebruikt bij het ontwerpen van software en is een methode die de programmeur gebruikt om zo sneller en gemakkelijker tot een totaal geheel te komen.
E-commerce value chain	E-commerce: kopen of verkopen over het internet (aan de hand van een site). Value chain/waardeketen: telkens wanneer het te verkopen product door een bepaalde activiteit gaat zal er waarde toegevoegd worden aan het product.
Embedded system	Een apparaat dat beschikt over een geïntegreerd systeem die de bedoeling heeft om het apparaat van een bepaalde vorm van intelligentie te voorzien.
Gateway	Een soort tussenstop om data die doorgestuurd werd met een ander protocol, te ontvangen en deze dan door te sturen met het correcte protocol naar een server.
Interface design	De opmaak van de gebruikersinterface.
IoT	Afkorting voor Internet of Things.

Begrip	Definitie/verklaring
Irrigatie	Het toevoegen van water, door allerlei middelen, aan landbouwgewassen bij een tekort aan water.
Library (programmeren)	Een library (bibliotheek) binnen de informatica is een verzameling van code (functies/routines) die door programma's kunnen worden gebruikt. Het voordeel van een library is dat programmeurs geen (nieuwe) code dienen te schrijven voor bepaalde algemene bewerkingen. De library kan dan gebruikt worden om functies aan te roepen die de bewerking uitvoeren zonder zelf veel code te schrijven.
MVC	Model-View-Controller is een design pattern dat gebruikt wordt bij het maken van websites.
NuGet Package Manager	Een tool in Visual Studio die de programmeur toelaat om verschillende hulpprogramma's te installeren die het werken binnenin Visual Studio en het eventuele project dat dient gemaakt te worden kan vergemakkelijken.
Open data	Data die beschikbaar is voor iedereen om te gebruiken (soms onder bepaalde voorwaarden).
Responsive	"Responsive" is een term die gebruikt wordt wanneer een website bij verschillende schermformaten nog steeds een gebruiksvriendelijke opmaak heeft.
SEO	Search engine optimization/ zoekmachineoptimalisatie: het optimaliseren van de zoekresultaten bij een bepaalde zoekterm.
Singleboard computer	Een singleboardcomputer (SBC) is een complete computer gebouwd op een enkele printplaat, met een microprocessor, geheugen, input/output, en andere onderdelen nodig voor een functionele computer.
SMTP	Simple Mail Transfer Protocol: is het standaard protocol bij het versturen van een e-mail over het internet.
TCP listener	Transmission Control Protocol Listener: programma dat over het TCP-protocol luistert of er inkomende connecties zijn en nagaat of deze mogen verwerkt worden door de server.
Tensiometer	Een tensiometer is een meetinstrument dat gebruikt wordt voor het meten van het vochtgehalte van de bodem aan de hand van de zuigspanning.
Webfeeds	Een webfeed laat de gebruiker toe om snel en gemakkelijk data te zien te krijgen van sites die deze data openstellen. Zonder te weten wat er zich allemaal afspeelt binnenin de webfeed zelf.

Begrip	Definitie/verklaring
Websocket	Computercommunicatie protocol die communicatie in 2 richtingen mogelijk maakt via een TCP-verbinding.
XAML	Extensible Application Markup Language: is een taal die gebruikt wordt om gestructureerde waarden en objecten te initialiseren. Het dient als een soort opmaaktaal voor de gebruikersinterface, om allerhande elementen, gebeurtenissen en andere onderdelen daarvan te definiëren.
XBee	XBee van Digi XBee is een draadloze communicatie module die gebruikt maakt van de ZigBee standaard, ZigBee is een draadloze communicatie die een grotere afstand overbrugt dan Bluetooth en een lager stroomverbruik heeft dan Wifi.

Bibliografie

ASP.NET (z.j.). Geraadpleegd op 20 maart 2017, van https://www.asp.net/.

Axure (z.j.). Geraadpleegd op 22 februari 2017, van https://www.axure.com.

Billiet, Y. en Thoré, S. (2012). Gebruikshandleiding voor het sensornetwerk. Rumbeke, België: Inagro

Boer zkt. alternatief voor grondwater (2012, 26 maart). Geraadpleegd op 17 april 2017, van http://www.vilt.be/Waterverbruik_van_de_Vlaamse_landbouw_Boer_zkt_alternatief_voor_grondwater

Decoster, S. en Vandewalle, W. (2017). Voortraject stage/eindwerk. Rumbeke, België: Vives

Definities voor verklarende woordenlijst. (z.j.) In Wikipedia. Geraadpleegd op april 2017, van https://nl.wikipedia.org/wiki/(Begrip in woordenlijst).

Google Maps JavaScript API (z.j.). Geraadpleegd op 25 april 2017, van https://developers.google.com/maps/documentation/javascript/.

Inagro (z.j.). Geraadpleegd op 20 februari 2017, van http://leden.inagro.be/.

Inagro klima app (2017). Geraadpleegd van http://inagrocloud.cloudapp.net/.

Korazon (z.j.). Geraadpleegd op 20 februari 2017, van https://www.korazon.be/.

Machine Learning made in a minute (z.j.). Geraadpleegd op 9 mei 2017, van http://accord-framework.net/.

Microsoft Azure (z.j.). Geraadpleegd op 6 maart 2017, van https://azure.microsoft.com/nl-nl/.

OpenWeatherMap API (z.j.). Geraadpleegd op 25 april 2017, van https://openweathermap.org/api.

Peeters, B. (2013, april). MIRE Themabeschrijving Waterkwantiteit. Geraadpleegd op 17 april 2017, van http://www.milieurapport.be/upload/main/themabeschrijvingen/Themabeschrijving_Waterkwantiteit_april_2013_TW.pdf

Protocol Sensornetwerk Inagro (2011). Rumbeke, België: Inagro.

Quartz Enterprise Scheduler .NET (2017, 18 februari). Geraadpleegd op 22 maart 2017, van https://www.quartz-scheduler.net/.

Regressie-analyse. (z.j.) In Wikipedia. Geraadpleegd op 9 mei 2017, van https://nl.wikipedia.org/wiki/Regressie-analyse.

Roland. (2016, april). Piecewise regression with constraints. Geraadpleegd op 9 mei 2017, van https://stats.stackexchange.com/questions/149627/piecewise-regression-with-constraints.

Rotativa 1.6.4 NuGet (2015, 24 maart). Geraadpleegd op 22 maart 2017, van https://www.nuget.org/packages/Rotativa.

Sensornetwerk lokale administratiemodule. (z.j.). Geraadpleegd op 14 februari 2017, van http://sensornetwerk.inagro.be/overzicht/Login.aspx.

Teach, Learn and make with Raspberry Pi (z.j.). Geraadpleegd op 27 maart 2017, van https://www.raspberrypi.org/.

Thoré, S. (2011). Sensornetwerk Cleantech in de praktijk. [Presentatie] Rumbeke, België: Inagro

Windows 10 IoT Core (z.j.). Geraadpleegd op 30 maart 2017, van https://developer.microsoft.com/en-us/windows/iot.

Yahoo Weather API (z.j.). Geraadpleegd op 3 april 2017, van https://developer.yahoo.com/weather/.

Bijlagen

De bijlagen zijn te vinden op de Cd-rom die bij deze scriptie zit. Ook zijn de bijlagen online beschikbaar in de bibliotheek van Vives[4].

Bijlage nummer	Titel
Bijlage 1	PowerPoint Inagro met documentatie sensornet
Bijlage 2	Handleiding Sensornet Inagro
Bijlage 3	Documentatie Inagro protocol (2011)
Bijlage 4	Realisaties en kosten

[4] http://limo.libis.be/primo_library/libweb/action/search.do?vid=VIVES_KATHO

Lijst van illustraties

Figuur 1: Waterverbruik (wereldwijd) 7
Figuur 2: Logo Korazon 9
Figuur 3: Logo Inagro 10
Figuur 4: Inloggen Wireframe 14
Figuur 5: Rapporten Wireframe 15
Figuur 6: Status dashboard Wireframe 15
Figuur 7: Hoofdblokken IoT netwerk 16
Figuur 8: Blokschema huidig Inagro sensornetwerk 19
Figuur 9: (links) sensorgroep, (rechts) vochtigheidssensor 21
Figuur 10: TCP listener huidig sensornetwerk 22
Figuur 11: Webapplicatie voorbeeld huidig sensornetwerk 23
Figuur 12: Eigen blokschema met Azure Cloud platform 29
Figuur 13: Blokschema MVC Design Pattern 31
Figuur 14: Installatie EntityFramework via Package Manager Console 32
Figuur 15: Klasse voor gebruikers 34
Figuur 16: Inlog formulier op Home pagina 38
Figuur 17: Parameters van de gebruikers via Mijn Bedrijf 40
Figuur 18: Google Maps integratie voor het toevoegen van een locatie 41
Figuur 19: Visualisatie van locaties via Google Maps 42
Figuur 20: Klasse van DataClient 43
Figuur 21: Weergave van Inagro sensoren in de webapplicatie 46
Figuur 22: Voorbeeld van pdf-rapport die gegenereerd is 48
Figuur 23: Listener in projectstructuur 49
Figuur 24: Endpoint instellingen van de listener 49
Figuur 25: Print screen van testapplicatie voor weerdata 52
Figuur 26: Quartz library 54
Figuur 27: Cron web tool 57
Figuur 28: Resultaat in database van Quartz job 58
Figuur 29: Datapunten, spreiding 61
Figuur 30: Gevisualiseerde regressie 63
Figuur 31: Functie voorschrift visualiseren uit regressie lijn 66

Figuur 32: Print screen Windows Form van regressie applicatie....66
Figuur 33: Gesegmenteerde regressie....67
Figuur 34: Print screen Raspberry Pi desktop....69
Figuur 35: Print screen IoT dashboard....73
Figuur 36: Wissen van de SD kaart vooraleer schrijven....74
Figuur 37: Melding dat SD kaart geschreven is in IoT Dashboard....74
Figuur 38: Klasses voor de benodigde sensordata te verwerken....75
Figuur 39: Raspberry Pi GPIO-nummering....76
Figuur 40: Layout van de UWP applicatie....80
Figuur 41: Grid lay-out in de UWP applicatie....80
Figuur 42: Print screen instellingen VisualStudio voor programmeren Raspberry Pi....81
Figuur 43: Web interface van Windows IoT core....81

design your future

vives katholieke hogeschool
associatie KU Leuven