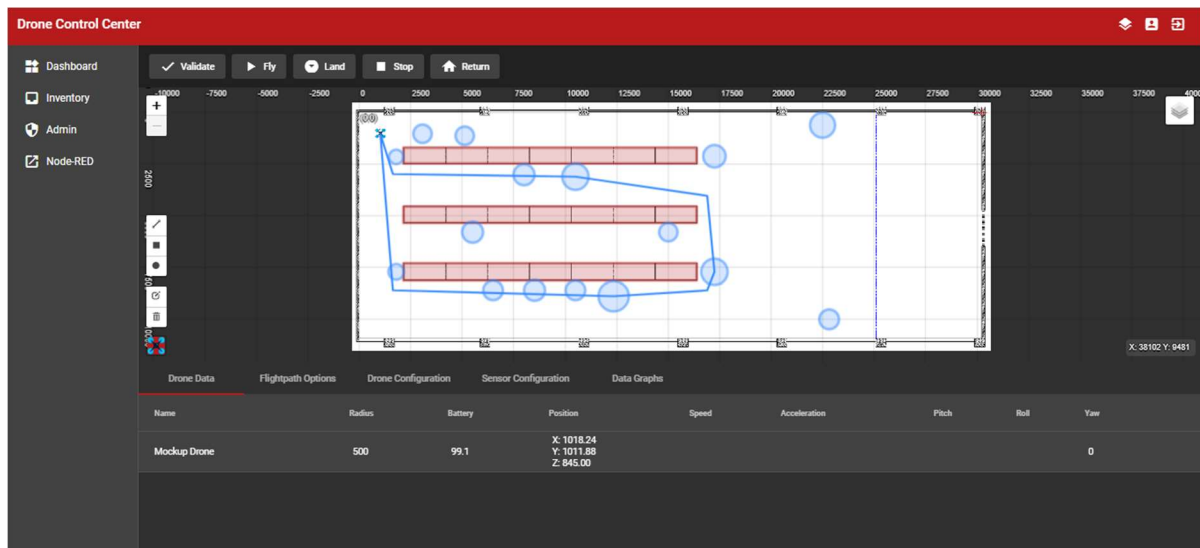


PATH PLANNER VOOR INVENTARIS MET AUTONOME DRONE



EINDVERSLAG

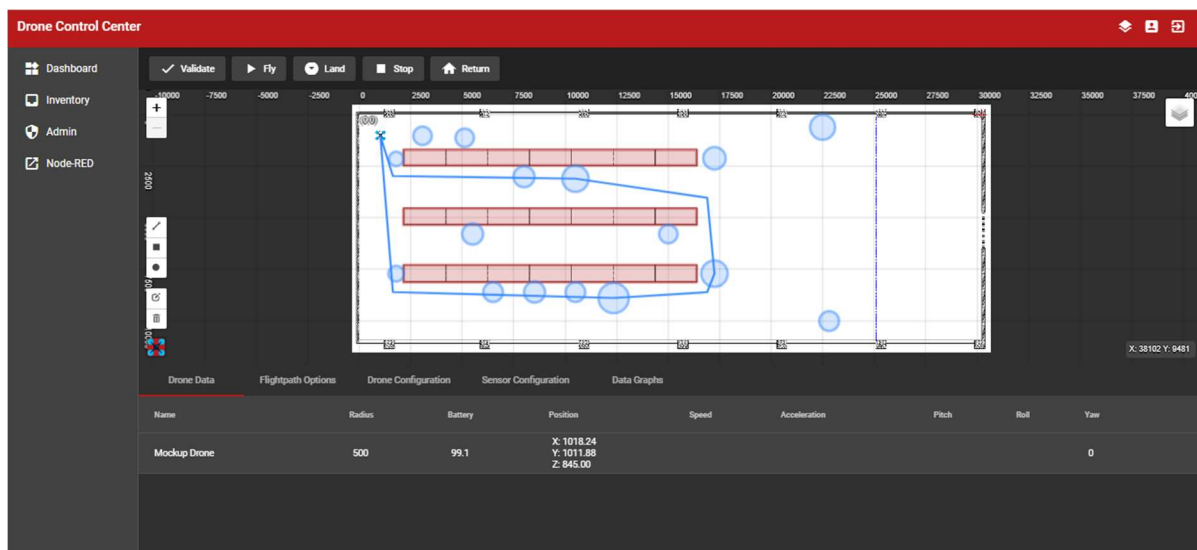
LEDEN: ROBBE DE SUTTER, KAREL EVERAERT, JOCHEN LAROY, ROEL MOEYERSOONS, WOUTER STEMGÉE

TEAM: DRONE1

Projectwebsite: <http://bpvop4.ugent.be:8081/>

20/05/2019

PATH PLANNER VOOR INVENTARIS MET AUTONOME DRONE



EINDVERSLAG

LEDEN: ROBBE DE SUTTER, KAREL EVERAERT, JOCHEN LAROY, ROEL MOEYERSOONS, WOUTER STEMGÉE

TEAM: DRONE1

WOORD VOORAF

Aan het begin van het tweede semester (2018-2019) kregen we een project voorgeschoteld dat we in het kader van onze bachelorproef moesten uitwerken. Het doel was om het werk van enkele onderzoekers van het IDLab van de UGent een beetje te vereenvoudigen. Om ons project tot een goed einde te brengen hebben we met verschillende technologieën geëxperimenteerd om uiteindelijk de meest geschikte technologieën te gebruiken in ons project. Tijdens het werken aan ons project hebben we heel veel bijgeleerd over een groot aantal onderwerpen.

Graag willen wij graag Nicola Macoir, Jono Vanhie-VanGerwen, Jen Rossey, Ann Van Overberge, Veerle Ongenae en Helga Naessens bedanken voor hun kritische, maar rechtvaardige kritieken.

Team Drone-1

Gent, 18 mei 2019

KORTE SAMENVATTING

Het IDLab is op zoek naar een manier om de inventarisering van grote warenhuizen en magazijnen te automatiseren zodat dit de hoge kostprijs door onder andere het nodige personeel, de kans op menselijke fouten en de gevaarlijke situaties voor de arbeiders kan verlagen.

Dit zou via drones verholpen kunnen worden, maar deze volgen momenteel enkel een manueel ingegeven pad aan de hand van coördinaten. Deze manier van werken is inefficiënt en laat toe dat er gemakkelijk fouten kunnen gemaakt worden bij het ingeven van de coördinaten.

Het doel van deze proef is om een webgebaseerde interface en een controlepaneel te ontwikkelen waar de drone kan gemonitord en bestuurd worden en de volledige inventaris van het magazijn kan weergegeven en aangepast worden.

Hiervoor wordt gebruik gemaakt van de MEAN-stack in combinatie met Node-RED om deze dynamische webapplicatie te maken. Door deze keuze is het mogelijk om het project modulair op te bouwen zodat er later extra functies kunnen toegevoegd worden.

Het eindresultaat is een modulaire webgebaseerde interface waarop de drone en het magazijn kunnen gemonitord worden.

INHOUDSOPGAVE

Woord vooraf	7
Korte Samenvatting	8
Inhoudsopgave	9
1 Inleiding	12
1.1 Context	12
1.2 Probleemstelling	13
1.3 Doelstelling	14
1.4 Structuur van het verslag	15
2 Gebruikersaspecten	16
2.1 High-level requirements	16
2.2 Use case diagrammen	17
2.3 Product backlog	19
3 Systemarchitectuur	23
3.1 High-level systeem model	23
3.2 Klassendiagrammen	30
3.3 Sequentiediagrammen	31
3.4 Databank	33
4 Testplan	34
4.1 Unit testen	34
4.2 Invoercontrole	34
4.3 Integration testen (manueel)	35
4.4 Usability testen	35
5 Evaluatie en discussies	37
5.1 Performantie	37
5.2 Beveiliging	38
5.3 Schaalbaarheid	38
5.4 Mogelijke uitbreidingen	39
6 Handleidingen	40
6.1 Installatiehandleiding lokaal en uitvoeren	40
6.2 Installatiehandleiding Docker	43
6.3 Node-RED configuratie voor eerste gebruik	44

6.4	Gebruikershandleiding	45
7	Besluit	47
	Referenties	48

Overzicht figuren

Figuur 1	Use case diagram van het dashboard	p. 17
Figuur 2	Use case diagram van het inventory scherm	p. 18
Figuur 3	Beschrijving van de containers met de gebruikte framework en protocollen.	p. 24
Figuur 4	Klassendiagram van de simulator/mockup drone	p. 30
Figuur 5	Sequentiediagram	p. 32
Figuur 6	Importeren van alle flows	p. 44
Figuur 7	MQTT-node configuratie	p. 44

1 INLEIDING

1.1 Context

Het concept van deze opdracht komt van het IDLab – Universiteit Gent. Deze instelling werkt momenteel aan een project om drones die momenteel al op de markt zijn, van extra functionaliteiten te voorzien om deze in te zetten voor het automatiseren van het inventariseren van magazijnen of grootwarenhuizen.

Het manueel inventariseren van een grootwarenhuis of een magazijn vergt namelijk heel veel werk, tijd en personeel. Deze zaken brengen niet alleen een hoge kostprijs met zich mee maar bovendien ook kansen op menselijke fouten of zelfs gevaarlijke situaties voor werknemers wanneer deze bijvoorbeeld producten moeten gaan scannen op grote hoogte.

Om dit te vermijden wil het IDLab kunnen gebruik maken van autonome drones die op elk willekeurig tijdstip van de dag, alle producten zou kunnen scannen zonder enige tussenkomst van een werknemer. Dit scannen zal geïmplementeerd worden door gebruik te maken van een QR-code of RFID-tag. Om dit te verwezenlijken zou de drone dus in staat moeten zijn om zelfstandig een getekend pad af te laten vliegen en hierbij de producten die hij onderweg tegenkomt te scannen, om zo het grootwarenhuis of het magazijn te inventariseren.

In het Technologiepark-Zwijnaarde, een technologiepark van de Universiteit Gent, gelegen in Zwijnaarde bij Gent, bezit het iGent¹ een ruimte die is omgebouwd tot een klein magazijn waar verschillende experimenten met de gemodificeerde drones kunnen uitgevoerd worden.

iGent¹: iGent is een nieuwbouw voor medewerkers van de dienst UGent TechTransfer en de vakgroepen Informatietechnologie (INTEC) en Elektronica en Informatiesystemen (ELIS) van de faculteit Ingenieurswetenschappen en Architectuur.

1.2 Probleemstelling

Op dit moment bestaat er geen geautomatiseerde manier om het pad van de drone te bepalen bij het scannen van een grootwarenhuis of magazijn. Elk pad moet manueel ingegeven worden aan de hand van coördinaten wat zeer inefficiënt en bovendien tijdrovend is. Dit project bestaat uit de taak om op basis van scanlocaties een automatisch pad te creëren, zodat de drone dit vervolgens kan afvliegen zonder enige menselijke tussenkomst. Er moet voorzien worden in een interface met controlepaneel waarop de drone zijn huidige locatie, de scanlocaties van de verschillende producten en het pad dat de drone zal afleggen om deze locaties te bereiken, getoond wordt. Via deze webgebaseerde applicatie moet de beheerder in staat zijn de drone te configureren, sensoren aan of uit te schakelen, producten toe te voegen of te verwijderen van een database, instructies door te geven aan de drone dat deze zijn route mag starten, pauzeren, moet stoppen, enz.

Het is de bedoeling dat de webapplicatie modulair wordt opgebouwd, zodat er nadien extra functionaliteiten kunnen worden toegevoegd of onderdelen kunnen verwijderd worden indien deze niet nodig zijn of vervangen moeten worden. Nadien kan er ook nog een mobiele applicatie ontwikkeld worden, zodat het project vanop een tablet of een ander mobiel apparaat kan gebruikt worden.

1.3 Doelstelling

Het doel van deze bachelorproef is om een interface en controlepaneel te voorzien voor een drone die aan de hand van een getekend pad een grootwarenhuis of een magazijn moet inventariseren. De gebruikers moeten gemakkelijk via een computer, tablet of een smartphone de drone zijn locatie kunnen monitoren, sensoren op de drone kunnen activeren of uitschakelen en de database kunnen raadplegen, om zo de beschikbare producten te bekijken en deze tevens toe te voegen, te verwijderen of aan te passen. Men moet de drone kunnen starten of stoppen met het afvliegen van zijn ‘door de gebruiker getekende’ vluchtroute.

De website zal natuurlijk ook over een vorm van authenticatie moeten beschikken zodat enkel gebruikers met de correcte toestemming de drone kunnen besturen of producten kunnen wijzigen in de database.

Om dit project te realiseren, zal gebruik gemaakt worden van de MEAN-stack, Node-RED en websocket.io. Angular zal samen met nginx (uitgesproken als “engine-x”) de dynamische front-end van de web applicatie verzorgen. Hierop kan de gebruiker de drone live monitoren, vluchtroutes tekenen, de productinventaris raadplegen, de verschillende mappen aanpassen, de drone configureren, enz.

Door middel van Express wordt een Restful API ter beschikking gesteld waarmee de front-end, door middel van http-requests, de verschillende objecten uit de MongoDB database kan ophalen, updaten, verwijderen of toevoegen. De logica van de backend bevindt zich in de Node.js applicatie. Deze logica omvat onder andere het sorteeralgoritme dat zal gebruikt worden voor de pathfinding, de connectie leggen met de databank, de express logica, ... Node-RED zorgt ervoor dat er kan worden gecommuniceerd met de drone. Zo is het dan mogelijk de drone een pad af te laten vliegen, hem op te laten stijgen, stoppen, landen, enzovoort. Data afkomstig van de drone wordt ook door Node-RED opgevangen en wordt realtime naar de front-end verstuurd via websocket.io. Producten die door de drone gescand worden, worden in de database opgeslagen door de express API aan te spreken.

1.4 Structuur van het verslag

Het verslag bestaat naast de inleiding (=hoofdstuk 1) nog uit 6 grote onderdelen: de gebruikersaspecten, de systeemarchitectuur, het testplan, de evaluatie en discussies, de handleidingen en het besluit.

In hoofdstuk 2 kan men de gebruikersaspecten vinden. Hier kunnen de gebruikers naast de High-level requirements ook een aantal ‘use case diagrammen’ terugvinden, samen met een volledige lijst met alle features die de webapplicatie voorziet en in welke sprint deze gerealiseerd zijn.

In hoofdstuk 3 staat beschreven uit welke onderdelen de webapplicatie bestaat en hoe dit project gerealiseerd is.

In hoofdstuk 4 staat het testplan. Dit is een overzicht van alle voorziene testplannen. Deze bevatten zowel unittesten, invoercontrole, code reviews, een strategie voor het automatisch compileren, user acceptance, enzovoort.

In hoofdstuk 5 is de evaluatie en discussie terug te vinden. Hierin wordt de performantie van de webapplicatie, de beveiliging, de schaalbaarheid, de eventuele problemen en geleerde lessen in vermeld.

Hoofdstuk 6 betreft de handleidingen waaruit de gebruiker kan leren hoe hij de webapplicatie lokaal kan installeren en hoe hij ze kan gebruiken.

Tot slot volgt in hoofdstuk 7 het besluit van het project.

2 GEBRUIKERSASPECTEN

2.1 High-level requirements

Er dient een webgebaseerde interface en controlepaneel gebouwd te worden om een autonome drone, die een grootwarenhuis of een magazijn moet inventariseren, moet monitoren en eventueel moet bijsturen. Deze webapplicatie moet ook verbinding maken met een database waarin de geïntariseerde producten kunnen opgeslagen worden samen met de drone en map configuraties. Enkel ingelogde gebruikers met de correcte toestemmingen mogen producten toevoegen, verwijderen of aanpassen in deze database. Hiervoor zal de webapplicatie over een authenticatie systeem moeten beschikken. De webapplicatie moet modulair worden opgemaakt zodat er makkelijk functies kunnen toegevoegd, verwijderd of vervangen kunnen worden. Eventueel kan er later een mobiele webapplicatie voorzien worden zodat de interface en het controle paneel via een tablet of een ander mobiel apparaat kan bekeken worden.

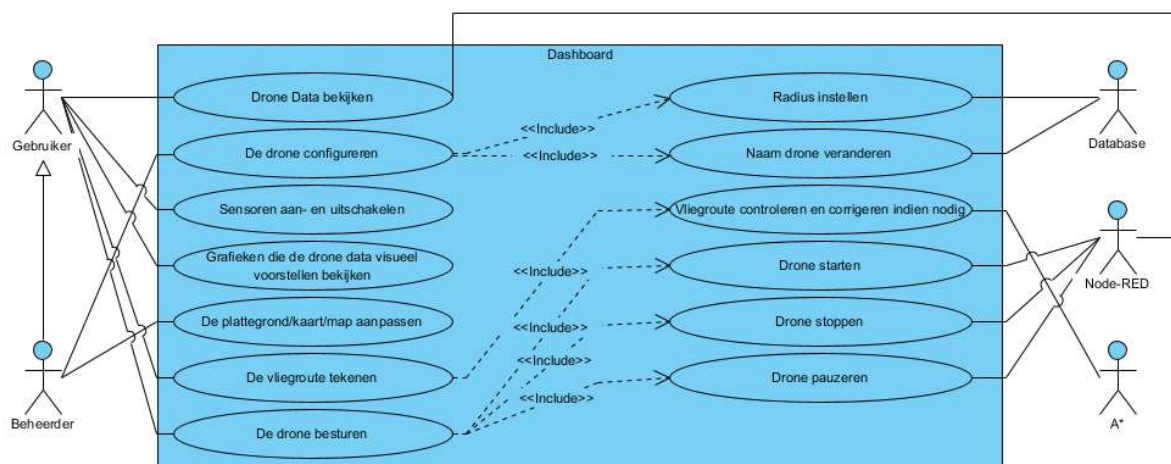
2.2 Use case diagrammen

2.2.1 Dashboard

Wanneer een gebruiker zich aanmeldt, komt deze terecht op het dashboard. Hier kan de gebruiker de drone data bekijken zoals de naam, de radius, het batterijniveau, de positie, de snelheid, de acceleratie en de pitch, roll en yaw van de drone. Enkel een beheerder, een gebruiker met admin rechten, kan de naam en radius van de drone aanpassen. Deze configuratie wordt nadien opgeslagen in de database.

Een gebruiker beschikt ook over de mogelijkheid om verschillende sensoren op de drone te activeren of te deactiveren. De waardes die deze sensoren meten kunnen op het dashboard aan de hand van grafieken gemonitord worden.

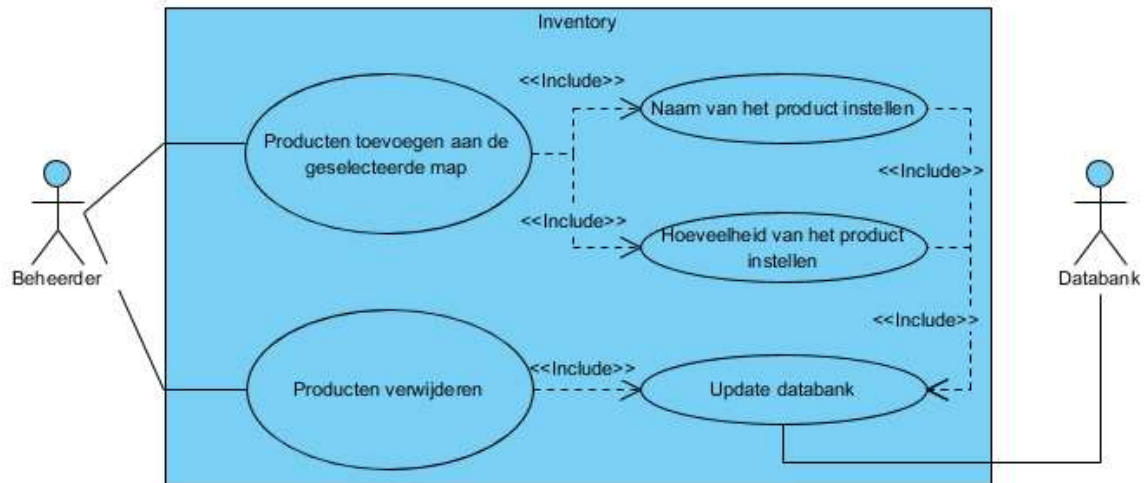
Een gebruiker kan via de kaart een vliegroute tekenen en nadat deze gecontroleerd is en eventueel gecorrigeerd via het A* algoritme kan de gebruiker de drone de gekozen vliegroute laten afvliegen. Deze commando's worden via Node-RED naar de drone verstuurd. Enkel een beheerder beschikt over de mogelijkheid om een kaart aan te passen door scanlocaties of een obstakel toe te voegen. Een overzicht hiervan wordt getoond in figuur 1.



Figuur 1: Use case diagram van het dashboard

2.2.2 Inventory

Als beheerder ben je in staat om via het inventory scherm de geïnventariseerde producten op te halen die bij de gekozen map horen. Je kan ook extra producten toevoegen door deze een naam en hoeveelheid mee te geven, zoals te zien is in figuur 2. Na het toevoegen van een product zal de databank ook geüpdatet worden door dit nieuwe product toe te voegen. De beheerder is ook in staat om een product uit deze lijst te verwijderen.



Figuur 2: Use case diagram van het inventory scherm

2.3 Product backlog

2.3.1 Volledige feature list

- Responsieve webapplicatie als front-end gebruikersinterface en controle paneel (eventueel later mobiele app ontwikkelen).
- Architectuur moet zeer modulair zijn zodat men gemakkelijk modules kan toevoegen of aanpassen (weinig afhankelijkheden gebruiken).
- Databank die de producten in het magazijn of grootwarenhuis bijhoudt, wanneer de drone items scant, wordt de database bijgewerkt met de recentste informatie.
- Mogelijkheid aanvinken welke specifieke data men wil weergeven of verbergen op de UI.
- Camera feed streamen naar canvas van webapplicatie.
- Dummy drone ontwikkelen die de sensor data stream van een vlucht simuleert.
- IP van drones moet dynamisch configureerbaar zijn.
- MQTT protocol gebruiken om data uit te wisselen tussen server en drone.
- Applicatie moet alerts geven indien de drone afwijkt van het pad.
- Dynamisch wisselen van datastream tussen verschillende sensoren indien bepaalde sensoren inaccuraat blijken te zijn.
- De drone besturen door een vliegroute mee te geven, deze vliegroute moet eerst gecontroleerd worden of de route veilig vliegen is voor de drone.
- De webapplicatie moet over een authenticatie en beveiligingssysteem beschikken.

2.3.2 Geselecteerde features per sprint

2.3.2.1 Sprint 1

- Communiceert via MQTT met de mockup voor het ontvangen van drone data en versturen van commando's. **15 story points.**
- Een API ter beschikking stellen met express voor de front-end UI zodat deze:
 - kaarten kan inladen, wijzigen en verwijderen
 - vliegroutes kan berekenen en uitwisselen met de drone
 - commando's vanaf de front-end kan doorsturen naar de mockup drone of naar een echte drone.
 - de productinventaris kan opvragen/aanpassen**20 story points.**
- zorgen voor persistentie met MongoDB voor het bijhouden van kaarten, droneconfiguraties en de productinventaris. **10 story points.**
- Berekenen van de optimale vliegroute tussen de gewenste waypoints of het controleren en corrigeren van een zelfgetekende vliegroute rekening houdend met obstakels. **15 story points.**
- Een simulator voorzien die de functionaliteit en alle dataflow kan testen, deze dient echter niet als eindproduct.
 - dynamisch inladen/aanpassen/opslaan van kaarten die de obstakels, waypoints en producten visualiseert
 - selecteren van waypoints en visualiseren van het optimale pad
 - visualiseert een vlucht van de mockup drone op basis van gesimuleerde data, deze data wordt in aparte componenten weergegeven**20 story points.**
- Weergeven van de productinventaris **5 story points.**
- Besturingselementen voorzien om de drone aan te sturen. **10 story points.**

2.3.2.2 Sprint 2

- De UI converteren naar een realistische 2D visualisatie door middel van leaflet.
 - Het voorzien van een continu coördinatenstelsel met de plattegrond van het gekozen magazijn of grootwarenhuis.
 - Voorzien van besturingselementen om de drone te besturen.
 - Toevoegen van scanlocaties waar de drone op basis van een oriëntatie de geselecteerde producten scant.
 - Voorzien van meldingen in verband met foutieve vliegroutes.

15 story points.

- Node-RED volledig implementeren en voor alle datastreams flows genereren. **15 story points.**
- Ondersteuning bieden voor mobiele apparaten door touch, scaling en gestures te voorzien. **15 story points.**
- De vliegroutes valideren met obstakels, met behulp van het snellere A* algoritme en collision detectie zodat niet correcte paden kunnen gecorrigeerd worden. **15 story points.**
- Het maken van een mockup drone voor het simuleren van vlucht data. **15 story points.**
- De ontvangen drone data van een echte drone of van de mockup drone verwerken en visueel weergeven op de webapplicatie door middel van grafieken, etc. **15 story points.**
- Toevoegen van authenticatie en beveiliging zodat niet geautoriseerde gebruikers geen toegang krijgen tot de webapplicatie. **5 story points.**
- Verschillende droneconfiguraties ondersteunen zodat er op een makkelijke manier tussen verschillende drones kan gewisseld worden. **5 story points.**

2.3.2.3 Sprint 3

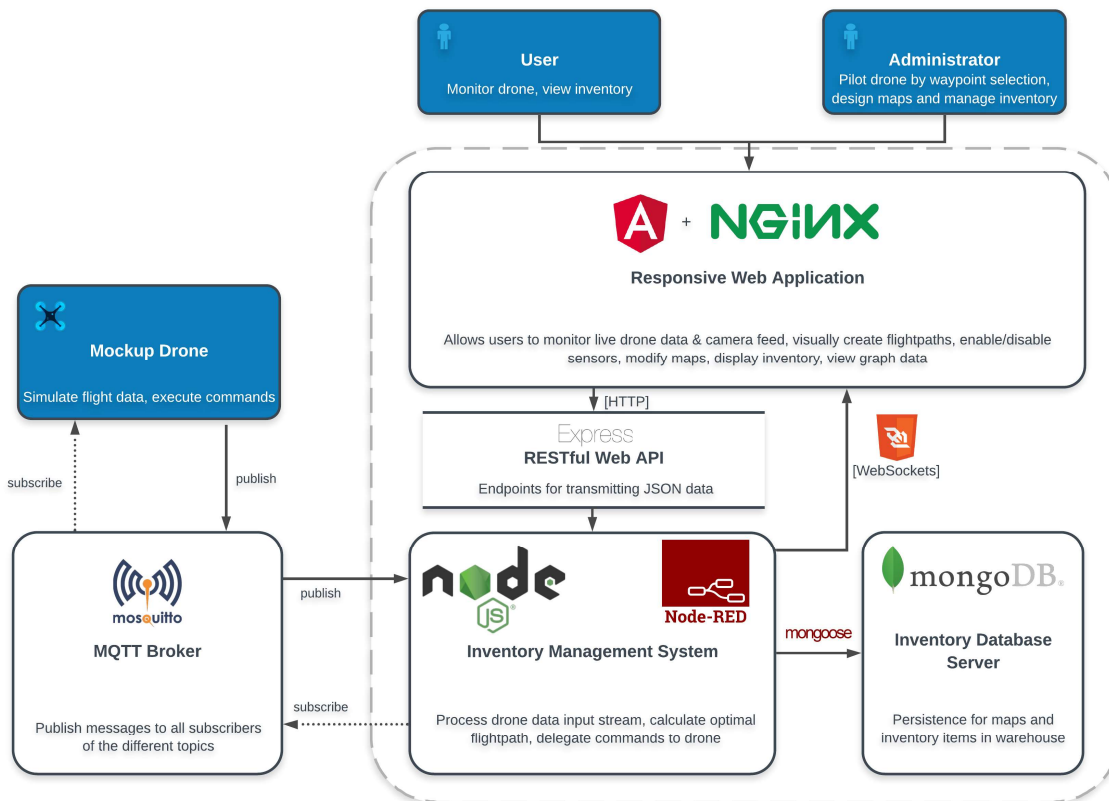
- Vliegroule opties selecteerbaar maken om te kiezen tussen het niet valideren, valideren, valideren en corrigeren of automatisch zoeken naar de beste vliegroule en het terugkeren naar de startpositie. **10 story points.**
- Vliegroules die niet valid zijn na bevestiging toch laten uitvoeren door gebruiker. **5 story points.**
- Oriëntatie van drone en scanzones beter visualiseren. **5 story points.**
- Drone-configuraties selecteerbaar maken. **10 story points.**
- Verbruik van drone toevoegen aan drone configuratie. **10 story points.**
- Alle drone data weergeven in grafieken. **15 story points.**
- Scanlocaties linken aan producten, route genereren door gewenste producten te selecteren om te gaan scannen. **10 story points.**
- Back-To-Home knop toevoegen, instelbare home locatie. **15 story points.**
- Applicatie moet alerts geven indien drone afwijkt van het gevraagde pad. **5 story points.**

3 SYSTEEMARCHITECTUUR

3.1 High-level systeem model

Voor de applicatie te realiseren wordt gebruik gemaakt van de MEAN-stack. MEAN is een acroniem dat staat voor de vier open source componenten: MongoDB, Express, Angular.js en Node.js die samen een end-to-end framework vormen om een dynamische webapplicatie te maken. MongoDB is een noSQL database waar dus geen gebruik gemaakt wordt van rijen en tabellen zoals in relationele databanken maar van documents en collections. Deze documenten zijn binary JSON-objecten, een variant op JSON, die meer data types voorziet. Express staat in voor het maken van een Restful API om ervoor te zorgen dat de front-end, geschreven in Angular, via http requests de verschillende objecten in de databank kan ophalen, toevoegen, aanpassen en verwijderen. Alle logica van de backend bevindt zich in de algemene Node.js applicatie. Deze bevat onder andere het sorteeralgoritme, de verbinding leggen met de databank, logica van Express, ... Het voordeel van Node is dat het beter schaalbaar is dan al zijn concurrentie, wat de performantie van de applicatie alleen maar ten goede komt. Algemeen kunnen we stellen dat de vier componenten die net besproken zijn zeer goed samenwerken door het feit dat ze allemaal javascript hanteren, of in het geval van Angular javascript related zijn. Ook werken ze alle vier samen met de node package manager om de verschillende delen te voorzien van de nodige packages.

Vervolgens zijn er hier een aantal lagen aan toegevoegd om de data van de drone te kunnen verwerken. De eerste laag die toegevoegd is, is Node-RED. Node-RED is een flowgebaseerde, visual programming tool, die zal instaan om de data van de drone op te vangen en deze daarmee toegankelijk te stellen in de stack maar ook om specifieke commando's naar de drone te sturen. Om de data realtime weer te geven in de front-end, wordt er gebruik gemaakt van websocket.io waarmee een bidirectionele, full-duplex, verbinding wordt gelegd tussen de front- en back-end om op die manier de data uit te wisselen met zeer weinig overhead.



Figuur 3: Beschrijving van de containers met de gebruikte framework en protocollen.

3.1.1 Front-end UI: Angular 7 op een nginx webserver

De front-end voorziet een continu coördinatenstelsel gebaseerd op een plattegrond van een grootwarenhuis of magazijn. Een beheerder kan op deze plattegrond verscheidende obstakels en scanlocaties aanduiden. Deze aanpassingen worden doorgestuurd naar de MongoDB databank waar de map wordt opgeslagen.

Deze map wordt gecreëerd door middel van leaflet, een open-source JavaScript library voor het creëren van interactieve kaarten. Leaflet is volledig geïntegreerd in de front-end als Angular Directive en biedt ondersteuning aan voor verschillende bestaande plugins zoals: live data, drawing, heatmaps, enz... Hierdoor is het mogelijk om zelfs op een mobiel apparaat met behulp van touch en gestures de map te vergroten, te verkleinen of aan te passen. De beheerder kan deze kaarten ook dynamisch aanpassen door scanzones en obstakels toe te voegen en de vliegroute voor de autonome drone te tekenen. Deze vliegroute wordt via Express naar de back-end gestuurd waar deze gecontroleerd en eventueel gecorrigeerd wordt. Vervolgens kan de beheerder met het gecontroleerde/berekende pad de drone het pad laten afvliegen. Er zijn ook extra besturingselementen aanwezig voor de drone stop te zetten, te laten landen, terug te laten keren naar zijn homelocatie, enz. Ook wordt data afkomstig van de drone gelogd onderaan, en zijn er verschillende tabs aanwezig voor configuratieparameters en preferenties in te stellen.

3.1.2 Back-end: Node.js en Node-RED

De back-end bestaat uit twee delen: Node.js en Node-RED.

De Node.js server stelt een API ter beschikking via Express waarmee de front-end via http-requests, de verschillende objecten uit de databank kan opvragen, toevoegen, aanpassen of verwijderen. Deze acties staan ook wel bekend als de CRUD-operaties.

Deze API kan getest worden door middel van de meegeleverde unit testen, zoals beschreven staat in hoofdstuk 4: Testplan.

Naast de API en de unit testen staat Node.js ook in voor het valideren van de vliegroutes. In een eerste versie werd dit gedaan door middel van Dijkstra.

Het doel van Dijkstra is om voor de routeplanner het optimale pad van de drone te berekenen doorheen alle waypoints. Het algoritme maakt gebruik van een graaf om het kortste pad te vinden. Een graaf bestaat uit verschillende knopen die met elkaar verbonden kunnen worden. Deze verbindingen bevatten op hun beurt een waarde/gewicht die bepaalt hoe lastig de verbinding is om af te leggen.

Het algoritme van Dijkstra werkt als volgt:

- Eerst wordt de afstand tot elke knoop op oneindig gezet buiten deze van de startknoop, die op 0 geïnitieerd wordt.
- Deze startknoop wordt nadien toegevoegd aan een prioritaire wachtrij. De positie van een knoop binnen deze wachtrij wordt bepaald aan de hand van de gekregen afstandswaarde. Hoe lager de afstandswaarde, hoe hoger de positie in de wachtrij.
- De startknoop wordt als bezocht gemarkeerd en als huidige knoop ingesteld.
- Vervolgens wordt er in de lijst met verbindingen gezocht naar elke verbinding met de huidige knoop. Indien het de eerste keer is dat het algoritme deze knoop bezoekt, wordt deze nieuwe knoop toegevoegd aan de prioritaire wachtrij en als bezocht gemarkeerd.
- De totale afstand/gewicht tot deze knoop vanaf de startknoop wordt berekend en indien deze lager is dan de vorige afstand/gewicht tot deze knoop wordt deze geüpdatet en verplaatst binnen de prioritaire wachtrij.
- Nadien wordt de knoop verwijderd en indien men de eindknoop bereikt heeft wordt het gevonden pad teruggegeven. Indien dit nog niet het geval is, wordt de meest prioritaire knoop opgehaald en als bezocht gemarkeerd en herhalen de laatste 2 stappen zich.

Dit werd verder geoptimaliseerd door gebruik te maken van de hierboven vermelde prioritaire wachtrij zodat de nabijgelegen knopen eerst worden onderzocht alvorens de verder gelegen knopen worden bekeken om zo sneller het gewenste pad te vinden. In dit project moet het kortste pad tussen de verschillende waypoints berekend worden. Hiervoor wordt het pad vanaf de begin positie tot elke waypoint berekend en vervolgens het dichtstbijzijnde waypoint gekozen als eerste tussenstop, nadien wordt dit herhaald met de tussenstop als startknoop totdat alle waypoints zijn opgenomen.

Deze oplossing was echter niet performant genoeg en bevatte nog geen manier om een getekend pad te valideren. Hierdoor werd besloten om over te stappen op het A* algoritme. A* verkiest namelijk om knopen te bekijken waarvan hij denkt dat deze in een lagere globale kost zal

eindigen, terwijl het algoritme van Dijkstra rekening blijft houden met alle omliggende knopen. Dit halveerde de zoektijd naar de kortste vluchtroute. Om de tijd om een vluchtroute te genereren nog meer te verlagen werd ervoor gekozen om de graaf, dat tot nu toe uit een volledige grid bestond bovenop de map, te vereenvoudigen. De autonome drone hoefde namelijk enkel van richting te veranderen indien er een obstakel in de weg stond. Zo werd ervoor gekozen om enkel in het midden van een scanzone een knoop te plaatsen en 4 knopen rondom een obstakel. Vanuit deze knopen wordt nadien gekeken welke met elkaar te verbinden zijn zonder te dicht bij een obstakel te komen. Zo wordt het aantal knopen in de graaf sterk verminderd wat de zoektijd minimaliseert. Hierdoor wordt het ook mogelijk om te controleren of een zelf getekend pad niet te dicht bij een obstakel kwam en indien dit wel het geval is, via het A* algoritme dit pad te corrigeren.

Node-RED verzorgt vervolgens nog de communicatie tussen de back-end en de MQTT-broker. Node-RED zal een cliënt vormen die zich subscribed op de gewenste data van de drone. Zo zal de data ontvangen van de drone of mockup drone in real time via Websockets verstuurd worden naar de front-end waar deze gevisualiseerd kan worden. De Node-RED flow bevat ook nog verschillende functies om bijvoorbeeld na te gaan of de drone zich op een scanlocatie bevindt en hier producten moet scannen, om na het toekomen op een locatie de nieuwe coördinaten door te sturen naar de drone zodat deze verder kan vliegen, enz...

Node-RED wordt gebruikt om flow-gebaseerd visueel te programmeren voor IoT of Internet of Things. Dit werkt volledig in de browser en is gemaakt op Node.js. Dit biedt de gebruiker de mogelijkheid om zelf delen van het programma te vervangen door een eigen alternatief. Er werd voor Node-RED gekozen aangezien deze MQTT, HTTP en WebSocket eindpunten bevat, zodat er makkelijk connectie kan gemaakt worden tussen de verschillende componenten van deze webgebaseerde applicatie. Dit in code schrijven vraagt wat meer (denk)werk terwijl in Node-RED deze eindpunten gewoon aan elkaar gelijmd kunnen worden, na wat configuratieparameters ingesteld te hebben.

3.1.3 MQTT en de Mockup drone

MQTT of message Queuing Telemetry transport is een M2M-publish/subscribe protocol. Dit protocol bestaat uit verschillende cliënten die specifieke data gaan publishen op verschillende topics. Deze berichten worden opgevangen door een broker (hier: mosquitto broker) die op zijn beurt deze berichten gaat doorsturen naar alle cliënten die gesubscribed zijn op dat bepaalde topic. In het geval van het project is er sprake van 2 cliënten, namelijk de drone zelf en de Node-RED die data met elkaar uitwisselen. De drone verstuurt zijn gegevens zoals positie, batterij, oriëntatie... en Node-RED verstuurt commando's. MQTT voorziet ook QoS die via een feedbackmechanisme nagaat of het bericht zeker bij de gesubscribeerde cliënt is toegekomen. Voor data van de drone te ontvangen vertraagt dit onnodig de doorvoersnelheid en wordt dit niet gebruikt, hier mag gerust een pakketje verloren gaan. Bij het opsturen van commando's echter is QoS zeker nodig, zo wil men niet dat een opgestuurd stopcommando verloren gaat en de drone blijft doorvliegen.

Om een drone te simuleren wordt er gebruik gemaakt van een mockup drone die geschreven is in een Node.js applicatie. Deze staat volledig los van de applicatie. Deze drone gaat een echte drone naspelen en op de verschillende MQTT-topics zijn gegenereerde data publishen, zodat deze kan getoond worden in de front-end. Ook is hij in staat de verschillende commando's voor de drone, zoals landen, stoppen, scannen, ... uit te voeren die hem worden toegestuurd. Met behulp van deze drone is de geschreven code grondig kunnen getest worden. Het is dan natuurlijk de bedoeling om achteraf zelf een echte drone te voorzien, die analoog data zal publishen en op de commando's kan reageren, dit staat los van het project.

3.1.4 Android applicatie

Naast het feit dat de front-end responsief is gemaakt, zodat deze ook op een mobiel apparaat in de browser mooi kan getoond worden, is er ook een native Android applicatie voorzien. Hiermee kan de gebruiker de applicatie ook als app installeren op een Android toestel. Deze app bevat dezelfde functionaliteiten als de website. Het leaflet canvas ondersteunt allerlei gestures, het is mogelijk om zich op de kaart te navigeren, in te zoomen, een pad in te geven door op de kaart te tikken...

3.2 Klassendiagrammen

3.2.1 Klassendiagram Mockup drone

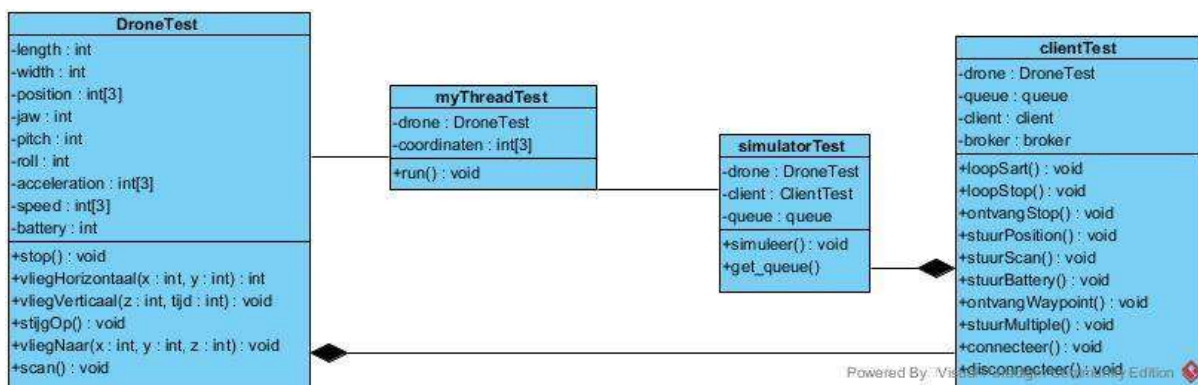
De (oude) Python mockup drone bestaat uit verschillende klassen.

De klasse drone bevat alle data van de drone, zoals de positie, breedte, oriëntatie, snelheid, etc., en methodes om het vliegen van de drone te kunnen simuleren. Voor alle data die de drone bevat worden getters en setters voorzien, deze worden echter niet op het klassendiagram weergegeven zodat het klassendiagram overzichtelijk blijft.

De klasse client zou het mogelijk maken om data te kunnen sturen en ontvangen met MQTT door te pushen en subscriben op een broker.

Tenslotte wordt er in de klasse simulator een eindeloze while-lus gestart waarin het gedrag van de drone gesimuleerd ging worden. Deze lus maakt gebruik van de klasse myThread, zodat de client de data van de drone kan blijven verzenden tijdens het vliegen van de drone.

De Node.js mockup drone die nu wordt gebruikt, werkt vrij analoog. Er wordt ook gebruik gemaakt van een drone klasse en een cliënt. De drone klasse genereert data en vult zijn eigen velden in. De cliënt voorziet communicatie over MQTT, het ontvangt de commando's van Node-RED en stuurt de drone aan, tegelijk verstuurt deze cliënt ook de data die de drone genereert. Het grote verschil tussen de python en de node drone is dat de node drone niet werkt met meerdere threads, hij draait en blijft slechts in één lus. Dit bespaart veel problemen maar kon enkel opgelost worden door vanaf de grond opnieuw een nieuwe drone op te bouwen.



Figuur 4: Klassendiagram van de simulator/mockup drone

3.3 Sequentiediagrammen

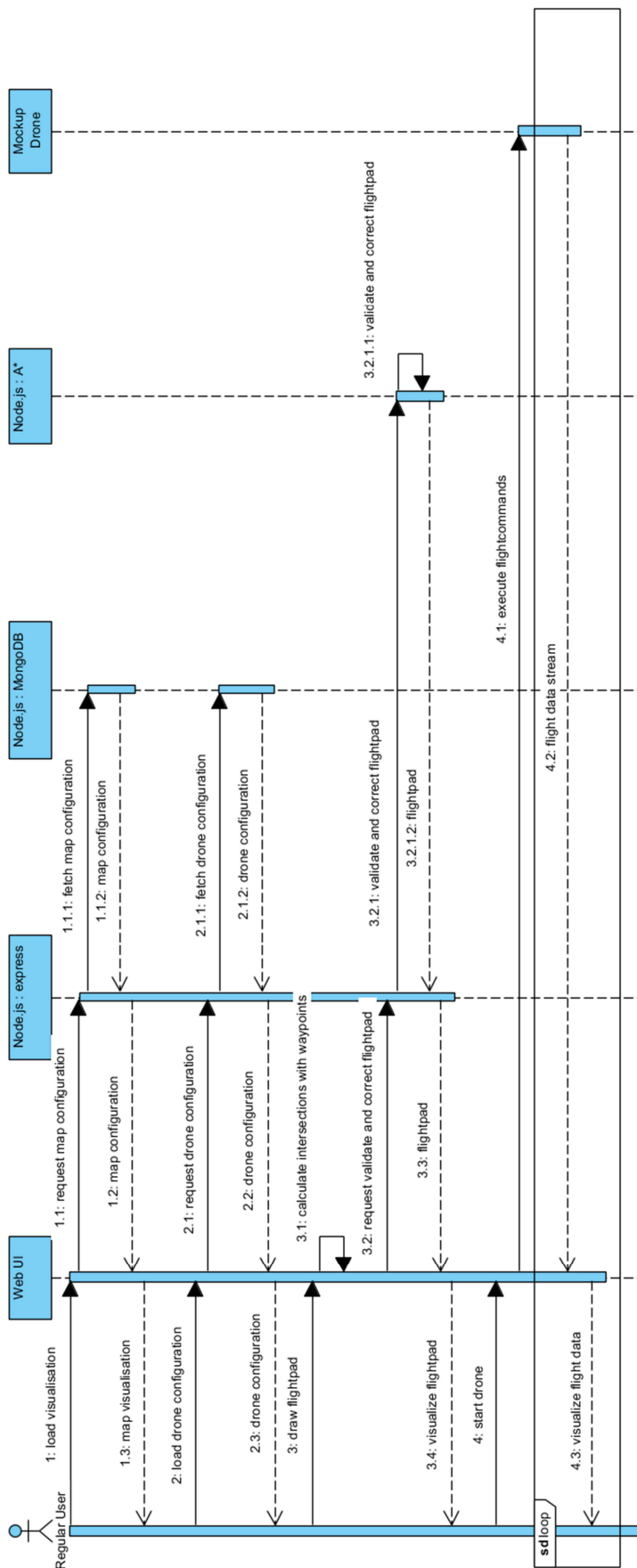
Bij het opstarten van de webapplicatie wordt zowel de map, de scanlocaties, de producten en de drone configuratie opgehaald uit de database. Indien de gebruiker van map verandert zal de map en de bijhorende producten van de nieuwe map opgehaald worden. In de backend wordt er een connectie gelegd met de drone via de broker. Van zodra deze connectie bestaat, wordt op het canvas op de website de drone weergegeven.

Vervolgens kan de gebruiker een vliegroute tekenen op de kaart, hierbij zal dan berekend worden welke scanlocaties op dit getekende pad liggen en wordt het pad gevalideerd en eventueel gecorrigeerd via het A* algoritme indien het pad door of te dicht bij een obstakel liep.

Hierna kan de gebruiker de drone starten en zal deze beginnen met het aflopen van het pad richting de scanlocaties. Hierbij zal de drone periodiek zijn nieuwe positie, snelheid, etc. doorsturen en zal dit visueel weergegeven worden op de kaart.

Wanneer de drone klaar is met zijn vliegroute zal deze terugkeren naar zijn oorspronkelijke positie en landen.

De gebruiker zal ook in staat zijn om de drone te pauzeren en eventueel te stoppen tijdens zijn vluchtroute.



Figuur 5: Sequentiedigram

3.4 Databank

Om de producten in de database te bewaren moet er eerst en vooral een schema opgesteld worden. Dit schema zegt hoe producten er moeten uitzien, en wordt door mongoose, de ORM van MongoDB, gebruikt om enkel correcte producten op te slaan. Er werd voor gekozen om een product een id, een naam, een hoeveelheid te geven. Het id van het product zal automatisch door mongoose gegenereerd worden. Verder zijn de naam en de hoeveelheid van het product verplicht op te geven. De producten kunnen vervolgens opgeslagen worden in de database. Deze zijn dan op te halen indien het id van het product gekend is. Ook Aanpassingen en het product compleet verwijderen zijn dan mogelijk. Bij het aanpassen geldt dezelfde regel: zowel een naam als een hoeveelheid moeten worden opgegeven. De toegang tot de database wordt geregeld via een REST API.

In de realiteit echter zijn de gebruikte schema's meer complex. Er is enkel een schema voorzien voor een volledige map, waarin genest dan schema's zitten voor obstakels, de grootte... en scanzones. Het zijn dan de scanzones die de producten bevatten. Zo zitten de producten genest in de corresponderende scanzone, die genest is in de map. Deze keuze werd gemaakt omdat zo het mogelijk is de drone te laten weten welke producten er vroeger in die scanzone zaten, die hij nu opnieuw scant. Dit kan gebruikt worden om te berekenen hoeveel er verkocht is, om fouten na te gaan, etc. en is ook handig om te weten voor de eindgebruiker. In de database zit er dan één groot mapobject (per map) die alle scanzones en producten etc. van de map bij houdt.

Dit groot mapobject is op dezelfde manier aanspreekbaar als in de eerste paragraaf. Ook zijn op alle geneste items dezelfde CRUD-operaties voorzien, zo kan dus een product in een scanzone in de map worden aangepast, wanneer zowel het scanzoneid als het productid gekend zijn. Nieuwe producten kunnen worden toegevoegd wanneer een scanzoneid gekend is binnen de map, etc... De front-end weet van al deze dingen af.

4 TESTPLAN

Alvorens een gebruiker de webapplicatie kan uittesten moet men deze eerst correct lokaal installeren. Een uitgebreide handleiding over hoe dit gedaan wordt staat in het hoofdstuk Handleidingen.

4.1 Unit testen

Om de verschillende API calls voor het toevoegen, aanpassen en verwijderen van de verschillende mappen, producten en droneconfiguraties en het valideren en corrigeren van een vluchtroute is een unit test meegegeven. Met behulp van Mocha en Chai kan iemand gemakkelijk de back-end testen. Deze test kan uitgevoerd worden door via een CMD-venster naar */project/backend/management-server* te navigeren en het commando *npm test* uit te voeren. Hiermee wordt het testbestand */project/backend/management-server/test/testAPI.js* uitgevoerd. Voor deze testen moet de database server zijn opgestart.

Bij het ontwikkelen werd vooral getest met het programma Postman waarmee de verschillende endpoints van express kunnen worden aangesproken. Eerst werd er een map opgeslagen, vervolgens kunnen er verschillende GET-operaties worden uitgevoerd op de geëmbbede objecten om te zien om deze wel correct werden opgeslagen, en het express endpoint werkt. Tot slot kan dan worden getest of deze objecten kunnen worden aangepast door PUT en DELETE operaties. Bij PUT moet natuurlijk de body van de http-request wel correct worden ingevuld met de vervangende data.

4.2 Invoercontrole

Elk invoerveld is voorzien van een controle zodat er in velden waar enkel numerieke waardes mogen ingevoerd worden enkel nummers kunnen ingevoerd worden. Ook wanneer er geen waarde ingevoerd wordt, wordt er een correcte foutboodschap op het scherm getoond.

4.3 Integration testen (manueel)

4.3.1 Testen van de vliegroutes

Om te testen of de vliegroutes correct berekend worden, kan de gebruiker een aantal waypoints aanduiden op de kaart. Onder het Flightpath Options tabblad bij het dashboard kan de gebruiker elke optie testen. Zo kan er nagegaan worden of paden door een obstakel worden afgekeurd indien de Validate path optie aanstaat. Dit pad zou wel moeten goedgekeurd worden indien de “Don’t validate path” optie is geselecteerd of goedgekeurd en aangepast worden indien de optie “Validate and correct path” was geselecteerd. Om de Calculate optimal path optie te testen selecteert de gebruiker het beste verschillende scanlocaties in een volgorde dat zeker niet de beste is, dit is bijvoorbeeld eerst een locatie ver van de drone, gevolgd door een locatie dicht bij de drone en opnieuw een locatie ver van de drone. Bij het klikken van de Validate knop moet de volgorde van de waypoints zijn aangepast. Met de Return to start optie zou het gevalideerde pad dezelfde begin- als eindlocatie moeten hebben.

4.3.2 Testen van de Sensor Configuratie

Door onder het Sensor Configuration tabblad bij het dashboard verschillende sensoren uit te schakelen en vervolgens in het Drone Data venster te kijken of de uitgeschakelde sensoren hun data verdwenen is, kan de gebruiker het subscriben op de verschillende MQTT-topics testen. Indien de gebruiker de sensoren terug aanschakelt, zou de data opnieuw moeten verschijnen.

4.3.3 Database testen

Op het inventory venster kan de gebruiker controleren welke producten er al reeds in de database zouden moeten zitten. De gebruiker kan hier ook testen of de connectie met de database correct is opgesteld door een product toe te voegen aan deze lijst en vervolgens manueel in de database te kijken of het nieuwe product is toegevoegd. Dit kan bijvoorbeeld gecontroleerd worden door een API call. Meer uitleg kan gevonden worden in de readme over het Inventory Management Server onder drone1/project/backend/README.md

4.4 Usability testen

Om de webapplicatie op gebruiksvriendelijkheid te testen, heeft een ander team gedurende 5 tot 10 minuten de webapplicatie getest. Hierbij werd duidelijk dat het inladen van de correcte flows in Node-RED niet voor de hand ligt.

Wanneer de gebruiker een waypoint plaatst dat te dicht bij een obstakel staat toont deze een bericht dat de vliegroute ongeldig is. Voor de gebruiker was dit niet helemaal duidelijk dat het probleem lag aan het feit dat de waypoint te dicht bij de muur lag. De leaflet kaart bevat vele knoppen, maar doordat deze veel informatie tonen indien de gebruiker over deze knoppen bij mouseover veel informatie tonen, is er weinig documentatie nodig over dit onderdeel van de webapplicatie.

Na het dashboard heeft de gebruiker weinig bijkomende informatie nodig omdat het inventory scherm geen ingewikkelde informatie of functies bevat.

5 EVALUATIE EN DISCUSSIES

5.1 Performantie

Tijdens het ontwikkelen van de applicatie werd er veel aandacht besteed aan de performantie ervan. De factor die de grootste invloed heeft op de performantie van onze applicatie is het A* algoritme waarmee de vliegrouete berekend of gevalideerd wordt. Het A* algoritme heeft als voordeel, ten opzichte van het Dijkstra algoritme, dat het enkel de knopen gaat bekijken waarvan het algoritme denkt dat deze tot een lagere globale kost zullen leiden. Dit halveerde de tijd nodig voor het genereren van een vliegrouete in vergelijking met de tijd die nodig was voor de generatie van een vliegrouete met het algoritme van Dijkstra. Een andere manier om de performantie te verhogen bestond eruit om de gebruikte graaf te vereenvoudigen, zodat de graaf niet eerst uit de volledige grid bestond. Er werd voor gekozen om enkel in het midden van een obstakel een knoop te plaatsen en 4 knopen rondom het obstakel. Zo werd het aantal aanwezige knopen in de graaf sterk verminderd waardoor de performantie steeg.

De front-end moet dan ook zeer performant zijn. Om deze reden werd er dan ook veel tijd gestoken in het oplossen van leaks. Het leaflet canvas is van nature niet zeer performant, maar voldoet zeker aan de verwachtingen, dit kan indien gewenst veel verbeterd worden door de grid layer ervan uit te schakelen. Indien er veel producten in de database zitten gaat de inventory tab zeer traag beginnen werken, wat de hele applicatie slecht beïnvloedt. De producten blijven ingeladen ook wanneer men terugkeert naar het dashboard. Dit kan opgelost worden door de productenlijst in een dynamische container te stoppen die enkel de “zichtbare” producten op de website laadt. Zo moet al de rest ook niet weergegeven worden. Vervolgens moet men ook de producten niet meer inladen wanneer de gebruiker op het dashboard zit. Een goede indicatie om na te gaan hoe vlot de applicatie nog loopt is te kijken hoe vlot de binnenkomende data van de drone de UI aan past.

Node-RED en de broker vertonen geen problemen, ook al plaatst de drone hier zeer veel operaties op.

5.2 Beveiliging

Om een eerste vorm van beveiliging te implementeren werden er verschillende soorten gebruikers gedefinieerd namelijk de beheerder en de gebruiker. De gebruiker kan de drone data bekijken, sensoren aan- en uitschakelen, grafieken bekijken, vliegroute tekenen en de drone besturen. De beheerder heeft alle rechten van de gebruiker, maar is ook in staat om de drone te configureren en de kaart aan te passen.

Om onderscheid te maken tussen de beheerder en gebruiker wordt er gebruik gemaakt van authenticatie. Bij het registreren van een nieuwe persoon moet er opgegeven worden of deze persoon een nieuwe gebruiker of een nieuwe beheerder voorstelt.

API-calls worden niet afgeschermd. Dit zorgt ervoor dat het testen van de applicatie met postman mogelijk is.

5.3 Schaalbaarheid

De front-end is zeer schaalbaar, mappen van alle groottes kunnen worden gebruikt in Leaflet (wat niet het geval is met een gewoon canvas). Door de optimalisaties op het algoritme blijven padberekeningen ook zeer performant. Het enige probleem is dat indien er veel producten in de map aanwezig zijn, de applicatie zeer traag zal werken zoals reeds vermeld.

De data wordt opgeslagen in een MongoDB-database. Dit is een NoSQL database, het voordeel van dit soort database is dat deze in staat is om te werken met een gigantische hoeveelheid data.

5.4 Mogelijke uitbreidingen

Voor deze paragraaf is een grondige kennis van de applicatie vereist. Hiervoor wordt verwezen naar de git repository.

Mogelijke uitbreidingen aan de back-end zijn:

- In Node-RED kan er meer statusinfo worden bijgehouden bijvoorbeeld of de drone aan het landen is vs of de drone gestopt is, zodat de front-end ook meer (gepaste) alerts kan geven.
- Het huidige scanned feedback mechanisme laat enkel toe dat er exact 1 product wordt aangepast/opgeslagen, omdat de drone dit productobject hier moet meegeven. Dit kan makkelijk opgelost worden door de flow te ontbinden in:
 - een flow waarop de drone de producten stuurt (eventueel 1 voor 1 om zo weinig mogelijk aanpassingen aan het huidige systeem te moeten doen) en zo de database aanpast
 - een flow waarbij de drone zegt dat hij klaar is met scannen en dat Node-RED het volgende knooppunt op stuurt
- Er wordt niet nagegaan of de drone het huidige pad wel daadwerkelijk volgt, enkel of hij op een knooppunt is toegekomen, hiervoor is een extra flow nodig (MQTT input node), er staat al een code snippet in management-server/api/drones.js die hiervoor zou kunnen worden gebruikt. Deze flow moet wel die data nog in de front-end kunnen krijgen.
- Meerdere drones worden door Node-RED niet ondersteund, tenzij hiervoor twee aparte backend processen (met brokers) runnen. De huidige flows aanpassen om rekening te houden met een drone id zou een oplossing zijn, die alle flows aanpast maar nergens anders impact op heeft.

Uiteraard zijn extra functionaliteiten en aanpassingen mogelijk door nieuwe Node-RED flows te schrijven.

6 HANDLEIDINGEN

6.1 Installatiehandleiding lokaal en uitvoeren

De gebruiker kan de webapplicatie lokaal testen of deze via de projectwebsite bereiken. De projectwebsite is in de voetregel of op het voorblad te vinden.

Om de webapplicatie lokaal te testen zijn er een aantal vereisten.

Namelijk:

- De broncode moet gedownload worden zodat deze lokaal beschikbaar is.
 - MongoDB moet geïnstalleerd worden en de database locatie moet worden aangemaakt. Minimum versie 4.0
 - Node.js en een Node Package Manager die automatisch met Node.js geïnstalleerd wordt. Minimum versie 10.15.1
 - Angular, Express en andere modules moeten correct geïnstalleerd worden. Minimum versie 4.16.0 voor Express en versie 7.3.2 voor Angular
 - Node-RED. Minimum versie 0.20.3
 - De Mosquitto broker. Minimum versie 1.5.8
-
- Om de broncode lokaal beschikbaar te maken, moet de GitHub repository gecloned worden. Hiervoor navigeert de gebruiker via een browser naar de hoofdpagina van de repository. In dit geval is dat dus <https://github.ugent.be/bp-vop-2019/drone1>. Op deze pagina klikt men op de 'Clone or download' knop waarna men twee opties heeft. Men kan op download ZIP klikken om vervolgens het ZIP-bestand uit te pakken in een locatie naar keuze of men maakt gebruik van de clone URL en Git Bash. Bij Git Bash verandert men de huidige working directory naar een locatie naar keuze en kan men de GitHub repository downloaden via het git clone commando. Hier is dit dus git clone <https://github.ugent.be/bp-vop-2019/drone1.git>. Nu is de broncode lokaal beschikbaar.
 - MongoDB is beschikbaar via <https://www.mongodb.com/download-center/community>. Op deze pagina selecteert men de versie en het besturingssysteem naar keuze. Voor linux is het aangeraden om de package manager te gebruiken om MongoDB te installeren. Men kiest best voor de recentste versie en download het MongoDB Server bestand. Tijdens deze installatie wordt aangeraden om de standaard instellingen te behouden. Naast MongoDB Server zal dit ook MongoDB Compass installeren, dit is een handige tool om connectie te maken met de database en deze ook te visualiseren. Collecties kunnen met MongoDB Compass ook aangesproken worden via CRUD operaties. Vervolgens moet de map '*Directory waar MongoDB is geïnstalleerd*'/data/db (of /data/db op linux) worden aangemaakt. Voor de meeste Windows gebruikers zal dit dus C:/data/db zijn. Hierin zal MongoDB zijn collecties bewaren. Indien de gebruiker een andere map locatie wenst te gebruiken, moet de gebruiker bij het runnen steeds de optie --bpath met als argument het gewenste pad mee geven.

- Node.js is een run-time environment, dit betekent dat het alle onderdelen bevat om de Javascript code van dit project te kunnen uitvoeren. Node.js wordt ook gebruikt voor het installeren van Angular en Express. Node.js kan geïnstalleerd worden door naar <https://nodejs.org/en/download/> te surfen en vervolgens de correcte installer te downloaden. Voor de meeste gebruikers zal dit de Windows Installer zijn onder het LTS tabblad. Tijdens deze installatie wordt aangeraden om de standaardinstellingen te behouden. Voor linux is het aangeraden om de package manager te gebruiken om Node.js te installeren.
- Vervolgens kunnen alle bijhorende modules waaronder ook Angular en Express geïnstalleerd worden. Hiervoor moet de gebruiker een CMD-venster openen en naar de folder locatie navigeren waar hij het project heeft opgeslagen. Eenmaal de gebruiker naar */project/backend/management-server* genavigeerd is, moet hij het commando *npm install* uitvoeren. Zo worden de nodige afhankelijkheden voor de back-end automatisch geïnstalleerd. Na afloop doet de gebruiker hetzelfde maar nu navigeert hij eerst naar de folder */project/web-ui/drone-control-center*. Nu worden de afhankelijkheden van de front-end automatisch geïnstalleerd. Indien men ervoor kiest om ook de mockup drone te gebruiken navigeert de gebruiker ook naar de folder */project/drone-mockup-node* en voert ook hier het *npm install* commando uit
- Om de mosquito broker, die voor de communicatie tussen de drone en de webapplicatie staat, te installeren surft de gebruiker naar <https://mosquitto.org/download/> en selecteert deze de correcte installer. Voor de meeste gebruikers zal dit de windows-x64.exe zijn onder Binary Installation. Tijdens deze installatie wordt aangeraden om de standaard instellingen te behouden.

6.1.1 Uitvoeren

Nu alle onderdelen correct geïnstalleerd zijn, kan de gebruiker de webapplicatie opstarten door de verschillende services op te starten. Hiervoor moet de gebruiker:

- De front-end te starten door naar de folder `/project/web-ui/drone-control-center` te navigeren in een CMD-venster. Vervolgens voert men het `ng serve` commando uit.
- De back-end te starten door naar de folder `/project/backend/management-server` te navigeren in een CMD-venster en vervolgens het `npm start` commando uit te voeren.
- De mockup drone kan op dezelfde manier gestart worden. Navigeer hiervoor naar `/project/drone-mockup-node` in een CMD-venster en voer vervolgens het `npm start` commando uit.
- De database kan gestart worden door in een CMD-venster te navigeren naar de installatie folder van MongoDB en vervolgens naar de folder `/Server/4.0/bin`. In deze map voert men het `mongod` commando uit.
- Als laatste start men de mosquitto broker door opnieuw in een CMD-venster te navigeren naar de mosquitto folder en hier het commando `mosquitto -v` uit te voeren.
- Bij het eerste gebruik moet ook eerst Node-RED worden geconfigureerd, hiervoor wordt verwezen naar 6.3.

Door via een browser naar keuze te navigeren naar <http://localhost:4200/dashboard> kan de gebruiker de webapplicatie uittesten. Men kan de Node-RED flows bekijken en eventueel aanpassen door naar <http://localhost:3000/editor> te surfen.

6.2 Installatiehandleiding Docker

De volledige webapplicatie kan ook op Docker gedeployed worden. Tijdens het project was het dan ook de bedoeling om aan het einde van elke sprint een werkende eindeversie te hebben op Docker. Onze versie is te vinden op <http://bpvop4.ugent.be:8081/>.


Om de webapplicatie op Docker te deployen werd er een server voorzien samen met een gebruikersnaam, wachtwoord en 2 poorten. Voor ons was dit:

- Server: bpvop4
- Gebruikersnaam: drone1
- Wachtwoord: Drone1_9523
- Poorten: 8081 en 8082

Om vervolgens de applicatie online te zetten open je lokaal een shell en navigeert u naar de root directory in de lokale git repository (/drone1). Vervolgens voert u het volgende commando uit om alle inhoud te kopiëren naar de remote server: `scp -r .\docker drone1@bpvop4.ugent.be:~/` Hierna opent u best een SSH-sessie in de shell om te controleren dat alle bestanden op de server geplaatst zijn via het `ssh bpvop4.ugent.be -l drone1` commando. Om verbinding te maken met de remote server is het meegegeven wachtwoord noodzakelijk. Vervolgens kan de Docker opgestart worden en is de webapplicatie gedeployed op Docker.

6.3 Node-RED configuratie voor eerste gebruik

Vooraleer een gebruiker de webapplicatie kan gebruiken, moet deze de Node-RED flows deployen. Nadat de gebruiker de webapplicatie heeft uitgevoerd, moet deze op de Node-RED

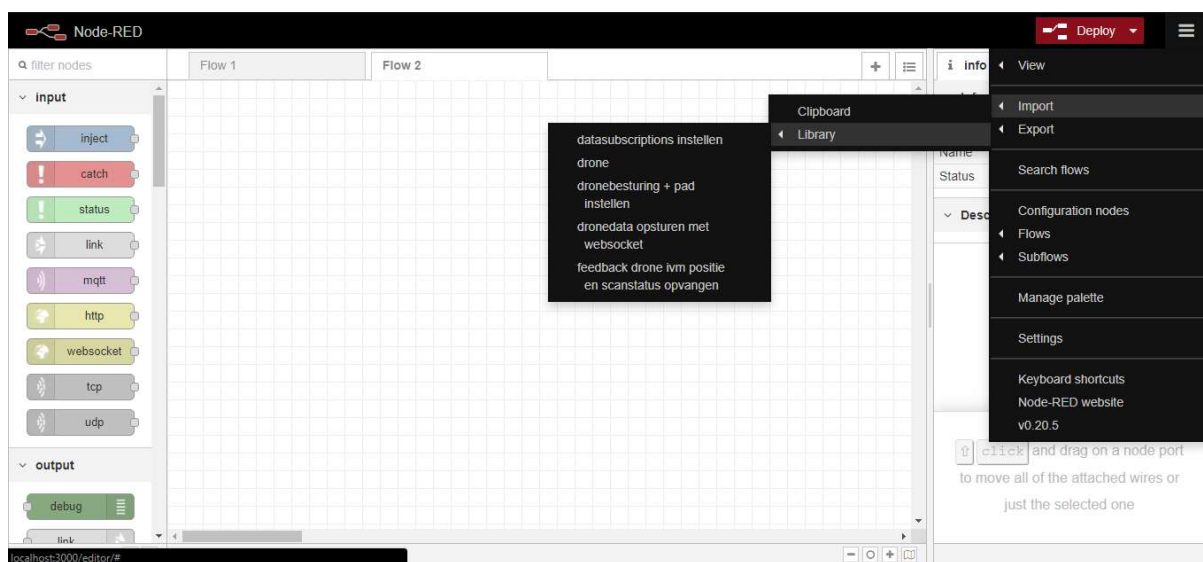
knop drukken. Vervolgens haalt de gebruiker alle flows op door via de extra opties  naar Import en vervolgens naar Library te gaan en hier elke flow importeert.

Nadat deze geïmporteerd zijn moet de gebruiker controleren of de Configuration nodes correct zijn ingesteld. Zorg ervoor dat er per flow slechts 1 mosquitto MQTT-broker aanwezig is en bij de 'dronedata opsturen met websocket' slechts 1 /red/ws/data websocket-listener is.

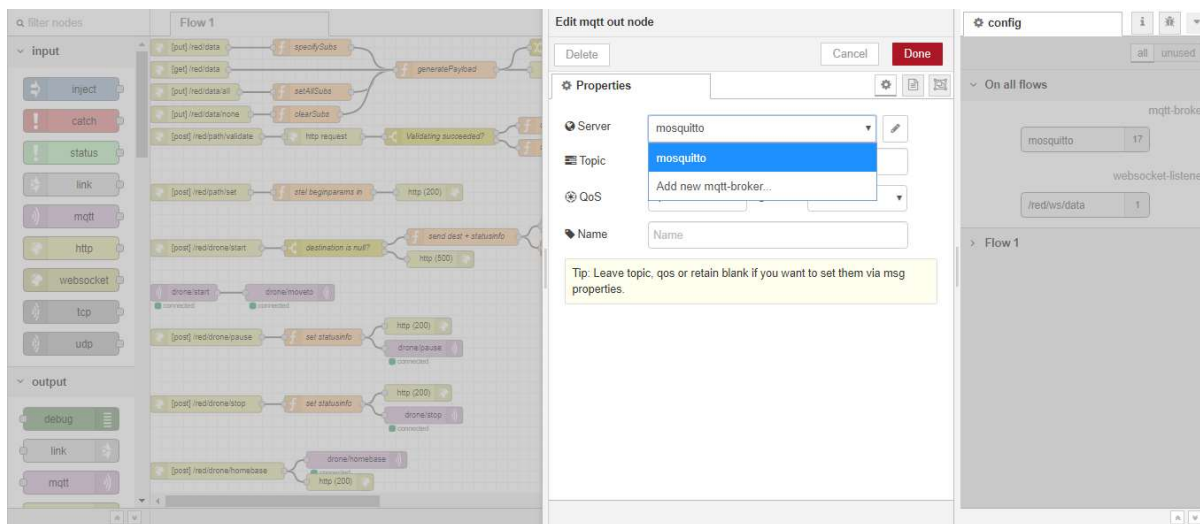
Vervolgens moet elke MQTT-node gecontroleerd worden op de correcte broker geselecteerd is zoals op Figuur 7. Analoog moet het correcte pad geselecteerd zijn bij de Websocket-node.

Indien de webapplicatie gedeployed is op Docker moet de gebruiker bij elke http request node de localhost in de URL vervangen door de correcte server waarop de back-end draait. Voor ons was dit dus *bpvop4.ugent.be:8082*.

Hierna moet de gebruiker enkel nog op de rode Deploy knop drukken zoals te zien is op Figuur 6. Dit zal de Node-RED server opstarten en is de webapplicatie klaar voor gebruik.



Figuur 6: Importeren van alle flows



Figuur 7: MQTT-node configuratie

6.4 Gebruikershandleiding

Bij het surfen naar de website moet de gebruiker eerst registreren of inloggen met een geregistreerd account. Indien de webapplicatie lokaal wordt gebruikt, is men vrij om zelf een account aan te maken. Indien men de webapplicatie via de projectwebsite raadpleegt, dient men in te loggen via het admin account om alle functies te gebruiken. Het admin account gebruikt als email: 'admin@ugent.be' en het wachtwoord is 'roeliewoelie'.

Vervolgens komt men op het dashboard scherm. Hierbij zal er een map ingeladen worden van de database of, indien deze leeg is, een nieuwe map worden aangemaakt. Via het dashboardscherm ziet men een plattegrond van het magazijn of grootwarenhuis met hierop de scanlocaties aangeduid via een blauwe cirkel en de obstakels via een rode rechthoek. Naast de scanlocaties en de obstakels wordt ook de huidige locatie van de drone aangeduid samen met zijn rotatie. Via de knoppen op de plattegrond kan men scanlocaties en obstakels toevoegen, scanlocaties en obstakels aanpassen of verwijderen, kan men een vliegroute tekenen en kan men het centrum van de map zich laten focussen op de locatie van de drone.

Onder de map zijn er 4 tabbladen te vinden. Eén voor de drone data: hier kan men de naam, de radius, het batterij niveau, de positie, de snelheid, de acceleratie en de pitch, roll en yaw van de drone bekijken.

In het tweede tabblad vindt de gebruiker de drone configuratie waar hij de naam en de radius van de drone kan configureren. Deze configuratie wordt nadien opgeslagen in de database.

Op het derde tabblad kan men de sensoren van de drone aanzetten of uitschakelen.

Het laatste tabblad bevat vervolgens nog enkele visuele grafieken waar de data van de drone visueel wordt voorgesteld.

Via de knoppen boven de plattegrond kan men de getekende vluchtroute valideren, de drone het pad laten afvliegen en de drone tijdens zijn vluchtroute laten pauzeren, terug verder laten vliegen of de drone volledig laten stoppen met het vliegen van zijn vluchtroute.

Links op het scherm kan men naast het dashboard de andere functionaliteiten van de webapplicatie bekijken. Het inventory scherm bevat een lijst met alle producten die gekoppeld zijn aan de huidige geselecteerde map. Hier kan een gebruiker met admin rechten ook producten aan toevoegen of deze weer verwijderen.

Via het Admin scherm kan een gebruiker met admin rechten kiezen om alle mappen en drones te verwijderen uit de database.

En als laatste is er ook een Node-RED tabblad voorzien zodat een gebruiker de flows kan bekijken en aanpassen.

In de rechterbovenhoek vindt de gebruiker 3 knoppen waarmee hij een plattegrond kan selecteren, zijn profiel kan bekijken en waarmee hij kan uitloggen uit de webapplicatie.

7 BESLUIT

Dit project was in opdracht van het IDLab - Universiteit Gent. Hun doel was om het inventariseren van een magazijn te automatiseren met behulp van drones, aangezien dit goedkoper en veiliger is. De drones zouden zelfstandig de aanwezige producten in het magazijn kunnen scannen. De onderzoekers van het IDLab moesten echter de coördinaten waar de drone naar toe moest vliegen telkens manueel instellen.

Tijdens dit project werd er een webinterface gecreëerd waarmee de producten die de drone moet scannen gemakkelijk aangeduid kunnen worden en waarmee de drone gemonitord kan worden tijdens zijn vlucht. Er kan eenvoudig een pad getekend worden. Dit pad wordt dan gevalideerd. Indien dit pad onmogelijk zou geacht worden dan krijgt de gebruiker een waarschuwing, maar hij heeft nog steeds de optie om het pad te laten uitvoeren.

De inhoud van het magazijn, die opgeslagen wordt in de database, moet ook eenvoudig weergegeven kunnen worden.

REFERENTIES

1. What is a RESTful API? | Creating a REST API with Node.js. (2017, November 28). Geraadpleegd op 19 februari 2019 via <https://www.youtube.com/watch?v=0oXYLzuucwE>
2. Javascript Dijkstra's algorithm and example | Code Road. (2017, August 12). Geraadpleegd op 18 februari 2019 via <https://computerscience.fyi/dijkstras-algorithm/>
3. Implementation of Priority Queue in Javascript - GeeksforGeeks. (2019, February 18). Geraadpleegd op 21 februari 2019 via <https://www.geeksforgeeks.org/implementation-priority-queue-javascript/>
4. Point, Line, Plane. (n.d.). Geraadpleegd op 28 maart 2019 via <http://paulbourke.net/geometry/pointlineplane/>
5. MEAN Stack: A Complete Guide. (2019, May 9). Geraadpleegd op 18 februari 2019 via <https://www.ibm.com/cloud/learn/mean-stack-explained>
6. MongoDB, de database van de toekomst? De evolutie van databases. (1BC, November 30). Geraadpleegd op 18 februari 2019 via <https://nljug.org/java-magazine/mongodb-de-database-van-de-toekomst-de-evolutie-van-databases/>
7. Node.js Express Framework. Geraadpleegd op 19 februari 2019 via https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm
8. Node-RED. Geraadpleegd op 19 februari 2019 via https://nodered.org/?fbclid=IwAR1SWcUUKfSwmtrOG-Ra_Njfr35aZh1P95Ytvq0UPc1uePpXAgPomAmAUhk
9. Leaflet — an open-source JavaScript library for interactive maps. Geraadpleegd op 16 maart 2019 via https://leafletjs.com/?fbclid=IwAR28EgRt9nxxVGX-FbCmRBKN5RLoEhoKO7i5ojHE0s_pcIRMj7mGeiCB7FI
10. Chai. Geraadpleegd op 7 maart 2019 via <https://www.chaijs.com/?fbclid=IwAR0F27eUAzX6DiCWV-uchnFLqS7PBUEXmQEdFmD0O-2BIF8eqaHiciZNPpc>
11. The most popular database for modern apps. Geraadpleegd op 18 februari 2019 via <https://www.mongodb.com/?fbclid=IwAR3OAjU2rgr8l5swNojFaFW54u7TZMmNn2tqsvS9TdTsEwCeo5QkauMzKGg>