

Bachelorproef: eindverslag

Droneplanning-tool



<https://github.ugent.be/bp-2020/drone1>

Bryan Van Huyneghem

Philip Kukoba

Nathan Beyne

Niels Hauttekeete

Promotoren: Prof. Helga Naessens, Prof. dr. Veerle Ongenae

Klant: Jan De Nul

Bachelorproef voorgelegd voor het behalen van de graad bachelor in de Bachelor of Science in de industriële wetenschappen: informatica

Academiejaar: 2019-2020

# Abstract



Inhoudsopgave

[Abstract 3](#_Toc39955129)

[Lijst van figuren 5](#_Toc39955130)

[Inleiding 7](#_Toc39955131)

[1. Gebruikersaspecten 10](#_Toc39955132)

[1.1 Databank 10](#_Toc39955133)

[1.2 Webapplicatie 10](#_Toc39955134)

[1.3 Use case-diagrammen 11](#_Toc39955135)

[1.4 Featurelijst 14](#_Toc39955136)

[1.4.1 Sprint 1 14](#_Toc39955137)

[1.4.2 Sprint 2 15](#_Toc39955138)

[1.4.3 Sprint 3 15](#_Toc39955139)

[2. Systeemarchitectuur 16](#_Toc39955140)

[2.1 High-levelsysteemmodel (*deployment diagram*) 16](#_Toc39955141)

[2.2 Databankdiagram 17](#_Toc39955142)

[2.2.1 DroneFlighttabel 17](#_Toc39955143)

[2.2.2 QualityReporttabel 20](#_Toc39955144)

[2.2.3 DroneLogEntrytabel 21](#_Toc39955145)

[2.3 MVC 22](#_Toc39955146)

[2.3.1 Model 22](#_Toc39955147)

[2.3.1.1 Simple Factory pattern voor parser-klassen 22](#_Toc39955148)

[2.3.2 Controllers 25](#_Toc39955149)

[2.3.2.1 ‘View’-controllers 25](#_Toc39955150)

[2.3.2.2 ‘Web API’-controllers 26](#_Toc39955151)

[2.3.3 Views 27](#_Toc39955152)

[2.4 Javascriptklassen 28](#_Toc39955153)

[2.4.1 Uploaden van files 28](#_Toc39955154)

[2.4.2 Visualisatie met ArcGIS JS API 36](#_Toc39955155)

[2.4.3 Data Tables 40](#_Toc39955156)

[2.5 Sequence diagrams 41](#_Toc39955157)

[3. Testplan 42](#_Toc39955158)

[3.1 Web API Controllers 42](#_Toc39955159)

[3.2 “View” Controllers 42](#_Toc39955160)

[4. Evaluaties en discussies 43](#_Toc39955161)

[5. Handleidingen 44](#_Toc39955162)

[5.1 Inhoud van de distributie 44](#_Toc39955163)

[5.2 Installatiehandleiding voor ontwikkelaar 44](#_Toc39955164)

[5.2.1 Vereiste software 44](#_Toc39955165)

[5.2.2 Aanmaken van de databank 45](#_Toc39955166)

[5.2.3 Opstarten van de webapplicatie (beter: deployen) 46](#_Toc39955167)

[5.3 Gebruikershandleiding 46](#_Toc39955168)

[5.4 Problemen die zich kunnen voordoen 46](#_Toc39955169)

[Besluit 49](#_Toc39955170)

[Future work 50](#_Toc39955171)

[Referenties 52](#_Toc39955172)

[Appendix A: use case-diagrammen stories 53](#_Toc39955173)

# Lijst van figuren

[Figuur 1: toevoegen en verwijderen van entiteiten in het systeem 12](#_Toc39955174)

[Figuur 2: wijzigen en details bekijken van entiteiten in het systeem 13](#_Toc39955175)

[Figuur 3: deployment diagram 16](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955176)

[Figuur 4: DroneFlighttabel met al haar relaties 18](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955177)

[Figuur 5: QualityReporttabel met al haar relaties 20](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955178)

[Figuur 6: DroneLogEntrytabel met al haar relaties 21](#_Toc39955179)

[Figuur 7: simple factory pattern met parser-klassen 22](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955180)

[Figuur 8: pop-up van een afbeelding in de view map 24](#_Toc39955181)

[Figuur 9: de Images-pagina van een dronevlucht 24](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955182)

[Figuur 10: detailpagina van een Drone Flight 27](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955183)

[Figuur 11: code snippet: abort-functionaliteit 28](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955184)

[Figuur 12: code snippet: controle op bestandsextensie 29](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955185)

[Figuur 13: code snippet: verbergen van div-elementen 29](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955186)

[Figuur 14: code snippet: beforeSend callback bij uploaden van één of meerdere bestanden 30](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955187)

[Figuur 15: code snippet: uploadProgress callback bij het uploaden van één of meerdere bestanden 30](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955188)

[Figuur 16: code snippet van de functie startParsing 31](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955189)

[Figuur 17: code snippet: helper-functie updatet de progress bar via aan HTTP get call 31](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955190)

[Figuur 18: code snippet: instellen en tonen van alle resultaatvelden na afronden parsen 32](#_Toc39955191)

[Figuur 19: code snippet: de verschillende error codes bij het uploaden van bestanden 33](#_Toc39955192)

[Figuur 20: berekenen van totale bestandsgroottes 35](#_Toc39955193)

[Figuur 21: controle bestandsgroottes 35](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955194)

[Figuur 22: pop-up voor een vlucht in een map overview 36](file:///s:\Documents\Universiteit%20Gent\Jaar%203\Semester%20VI\Bachelorproef\2020\drone1\verslag\verslag-drone1.docx#_Toc39955195)

[Figuur 23: de map overview met verschillende datapunten van een dronevlucht 37](#_Toc39955196)

[Figuur 24: LayerList widget 38](#_Toc39955197)

[Figuur 25: search widget 38](#_Toc39955198)

[Figuur 26: legend klasse 38](#_Toc39955199)

[Figuur 27: PopupTemplate 39](#_Toc39955200)

[Figuur 28: Feature klasse 40](#_Toc39955201)

# Inleiding

Jan De Nul (de klant) is een grote multinational die zeer veel data genereert, maar waarvan slechts een deel benut wordt om visualisaties te maken. Er worden drones gebruikt om de werf veiliger te maken voor het &personeel en om op een eenvoudige manier, vanop een welbepaalde hoogte, een afgebakend oppervlak in kaart te brengen. Dit betekent dat het vaak niet langer nodig is om de werf fysiek te betreden om foto’s te nemen. Bovendien kunnen ook moeilijk bereikbare plaatsen toch makkelijk gefotografeerd worden. Andere data, zoals de data die gelogd wordt in de logbestanden van een drone, is beschikbaar, maar wordt op dit moment niet onmiddellijk verwerkt en gebruikt.

Jan De Nul heeft nood aan een droneplanning-tool met een uitgebreide, centrale databank die al deze volumes aan data verwerkt en opslaat. Voorbeelden van data zijn: dronevluchtdata, coördinaten, locatiedata, foto’s en logboekdata. Nadien kan Jan De Nul via de visualisatie van deze data een overzicht krijgen van de werf en inzicht verwerven om vervolgens optimalisaties toe te laten. Dit betekent dat de ontwikkelde tool in staat moet zijn om verschillende kenmerken, zoals de diepte van een rivier, te accentueren met een gepaste visualisatietechniek. Hiervoor werkt de klant op dit moment met de *Geographical Information System* (GIS) software *ArcGIS*. Dit softwarepakket bevat uitgebreide opties aan programmeertaalkeuzes en een ruim aanbod aan visualisaties.

Op basis van de informatie die Jan De Nul verstrekte, werden enkele doelstellingen opgesteld:

1. De eerste en onmiddellijk meest cruciale doelstelling bestaat erin alle aanwezige data in kaart te brengen en logisch te groeperen. Vervolgens wordt hieruit in *SQL Server Management Studio* met *SQL Server 2019* een databankmodel aangemaakt dat de relaties tussen de verscheidene datagroepen beschrijft en vastlegt. Dit model kan eenvoudig uitgebreid worden, indien hier een noodzaak voor zou bestaan.
2. De databank, beschreven door voorgaand model, moet nu data gaan bevatten. De tweede doelstelling beoogt daarom om echte data los te laten op dit model en het te onderwerpen aan enkele testen. De data van Jan De Nul, die relevant zijn voor dit project, zijn beschikbaar onder verschillende bestandsvarianten, zoals csv, pdf, tfw en xyz. Jan De Nul genereert kwaliteitsrapporten van hun vluchten en wil deze informatie makkelijk kunnen opslaan in de databank. Verder vult elke piloot op dit moment een papieren logboek in voor hun dronevluchten. Dit moet vervangen worden door een eenvoudige interface die hoort bij de databank en deze informatie moet bevatten.

Dit betekent dat er een applicatie ontworpen moet worden die bestanden van deze types automatisch kan verwerken na uploaden. Deze gegevensverwerking of *parsing* van data gebeurt in dit geval met *parser-*klassen die via een *simple factory pattern* beschreven worden.

Er wordt gebruikgemaakt van het *Entity Framework* (EF) in de computertaal C# dat via *Object Relational Mapping* (ORM) objecten aanmaakt van de databanktabellen. Alle ingelezen data wordt weggeschreven in deze objecten en nadien opgeslagen in de databank.

1. De derde doelstelling beschrijft hoe Jan De Nul deze data kan raadplegen en op welke manier ermee gewerkt kan worden. Er wordt een webapplicatie ontworpen die toelaat deze data te bekijken en, indien noodzakelijk, aan te passen met een manuele ingreep. Deze applicatie beschrijft in eerste instantie alle dronevluchten, alle drones en alle piloten die in de databank aanwezig zijn. In tweede instantie kan meer gedetailleerde data geraadpleegd worden door de details van deze hoofdcategorieën te bekijken. Voorbeelden hiervan zijn de eerder vermelde kwaliteitsrapporten en dronelogboeken. Zeer specifieke data, die gebruikt worden in de vierde doelstelling, visualisatie, worden niet getoond in de webapplicatie en zijn enkel rechtstreeks aanspreekbaar via de databank. Het gaat hier immers om zeer grote hoeveelheden, moeilijk leesbare data.
2. De vierde doelstelling bestaat erin om, zoals eerder werd vermeld, de gigantische volumes aan data te visualiseren met de ArcGIS API. Deze visualisaties kunnen aangesproken worden in de webviewer sectie van de webapplicatie en hebben een belangrijke subdoelstelling: de webviewer moet eenvoudig navigeerbaar zijn voor leken en hen toelaten om op intuïtieve manier een beeld van een visualisatie te delen. Enkele voorbeelden van visualisaties zijn: het tonen van dronepaden; het tonen van *ground control points* (GCP); het visualiseren van het batterijgebruik van de drone; het visualiseren van hoogteverschillen in de gescande oppervlakte; etc. De bibliotheek van ArcGIS heeft een ruim aanbod aan functionaliteiten, wat toelaat om op vraag nadien nog visualisaties toe te voegen aan de webapplicatie.
3. Als laatste doelstelling moet het mogelijk zijn voor Jan De Nul om de ingelezen data opnieuw te kunnen exporteren naar bestanden van een ander bestandsformaat.

Het uiteindelijke doel van deze bachelorproef is om deze doelstellingen te verwezenlijken. In het eerste hoofdstuk, **Gebruikersaspecten**, wordt een gedetailleerde beschrijving van de opdracht vanuit het standpunt van de gebruiker (hier: Jan De Nul) gegeven. In dit hoofdstuk worden verder de gewenste softwarevereisten, use case-diagrammen en volledige featurelijst beschreven.

Het tweede hoofdstuk, **Systeemarchitectuur**, bespreekt hoe het programma deze doelstellingen en vereisten realiseert en hoe deze gestructureerd zijn. Aan de hand van een databankdiagram en klassendiagrammen zal de opbouw van de applicatie beschreven worden en getoond worden op welke manier de verschillende onderdelen samenwerken.

Het derde hoofdstuk, **Testplan**, geeft een overzicht van de voorziene testplannen. Er wordt per type test een voorbeeld gegeven.

Het vierde hoofdstuk, **Evaluaties en discussies**, behandelt de performantie van de applicatie, de beveiliging ervan en zijn schaalbaarheid. Als laatste wordt ook even ingegaan op problemen en geleerde lessen.

Het vijfde en laatste hoofdstuk, **Handleidingen**, is bedoeld voor de klant, Jan De Nul, en haar eindgebruikers. Dit hoofdstuk is van cruciaal belang voor hen, omdat het de installatiehandleiding en gebruikershandleiding bevat.

Doorheen dit verslag zal verwezen worden naar de initiële doelstellingen, zodat het voor de lezer op elk moment duidelijk is wanneer een doelstelling behandeld en geïmplementeerd wordt.

# Gebruikersaspecten

De drones van Jan De Nul verzamelen veel gegevens op hun vluchten, zoals foto’s van de site, coördinaten van de drone en data die door de drone zelf gelogd wordt in zijn intern logboek. De gebruiker verwacht een databank waaraan al deze vluchtgegevens eenvoudig toegevoegd en later opnieuw opgehaald kunnen worden.

Naast de gegevens die door de drones verzameld worden, zijn er nog talrijke andere gegevens beschikbaar. Allereerst houdt elke pilot momenteel een papieren logboek bij voor de drone en zichzelf. Het is de bedoeling dat de piloot deze gegevens in de databank kan ingeven en deze logboeken in de databank bijgehouden worden. Verder wordt na elke dronevlucht een kwaliteitsrapport opgesteld dat een analyse van de door de drone verzamelde gegevens bevat. Dit kwaliteitsrapport moet eveneens in de databank komen.

De klant heeft dus nood aan twee hoofdcomponenten: een databank om makkelijk data te kunnen verwerken en opslaan, en een interface om gegevens toe te voegen aan de databank en ook te kunnen opvragen.

## Databank

De databank wordt ontworpen in *SQL Server Management Studio* (SSMS) van Microsoft met *SQL Server* 2019, omdat Jan De Nul *in-house* eveneens met SQL Server werkt. Via een databankmodel worden de verscheidene datagroepen beschreven en hun onderlinge relaties vastgelegd. Dit model moet eenvoudig uitbreidbaar zijn indien nieuwe documentatie en data toegevoegd zou moeten worden aan databank.

## Webapplicatie

Er moet makkelijk naar een vlucht genavigeerd kunnen worden, opdat zijn gegevens opgevraagd kunnen worden. De interface komt er onder de vorm van een webapplicatie waarin het bovendien mogelijk is om data te visualiseren op basis van gespecifieerde attributen.

De applicatie wordt gebouwd met ASP.NET MVC 5 in Microsoft’s Visual Studio 2019. ASP.NET is een *open-source* *server-side* *web-application framework* dat toelaat om in C# moderne webapplicaties en -services te bouwen in de .NET omgeving. Het eenvoudige *Model-View-Controller* *pattern* laat toe om dynamisch krachtige webpagina’s aan te maken.

Verder wordt gewerkt met het *Entity Framework,* om met *Object Relational Mapping* (ORM) het databasemodel te *mappen* op automatisch gegenereerde klassen. In dit eindwerk zal met de benaming entiteit verwezen worden naar objecten van deze klassen.

Visualisaties van geografische data gebeuren binnen Jan De Nul met het ArcGIS platform (ArcGIS for Developers, z.j.), waardoor bijgevolg verwacht wordt dat ook de webapplicatie hiervan gebruik zal maken. Hiervoor wordt de ArcGIS API voor JavaScript gebruikt en wordt binnen de webapplicatie een aparte sectie voorzien voor visualisaties.

## Use case-diagrammen

Het gebruik van de webapplicatie wordt verduidelijkt in **Figuren 1 en 2** op pagina’s 10 en 11 via twee use case-diagrammen. Hierop is te zien hoe de actor (de gebruiker) kan interageren met de verschillende functionaliteiten van het systeem. Een gebruiker beschikt over CRUD (*Create, Read, Update, Delete)* functionaliteiten.

Een ingelogde gebruiker kan een dronevlucht, drone of piloot toevoegen (**Figuur 1**) aan het systeem (de databank) en onmiddellijk reeds enkele beschrijvende gegevens toevoegen aan deze entiteiten. Deze gegevens worden opgeslagen in de databank.

Een ingelogde gebruiker kan ook een dronevlucht, drone of piloot verwijderen (**Figuur 1**) uit het systeem. Verder kan een gebruiker de bijhorende en beschrijvende informatie van dronevluchten, drones of piloten wijzigen (**Figuur 2**). Deze aanpassingen hebben onmiddellijk effect op de databank aan server-sidekant.

Een gebruiker die *niet* ingelogd is, kan geen gegevens toevoegen, aanpassen of verwijderen. Hij is bovendien ook beperkt in het bekijken van informatie: hij kan enkel dronevluchten bekijken. Drone-informatie en pilootinformatie zijn aldus niet toegankelijk voor deze gebruiker. Een ingelogde gebruiker, daarentegen, kan *wel* alle informatie bekijken (**Figuur 2**).

Een complete beschrijving van de verscheidene interacties die plaatsvinden binnen deze use case-diagrammen is te vinden in Appendix A.



Figuur 1: toevoegen en verwijderen van entiteiten in het systeem



Figuur 2: wijzigen en details bekijken van entiteiten in het systeem

## Featurelijst

Hieronder volgt een opsomming van alle features in dit project, inclusief hun complexiteit[[1]](#footnote-1) en prioriteit[[2]](#footnote-2), per sprint.

## Sprint 1

* Ontwerp van een databankmodel dat op een logische manier de relaties tussen de verschillende entiteiten beschrijft (gemiddeld, zeer belangrijk);
* Keuze van de databank (SQL Server 2019) en ORM (Entity Framework 6) (makkelijk, redelijk belangrijk);
* Ontwerp en implementatie van het model: de parser-klassen via een *simple factory pattern* (vrij moeilijk, zeer belangrijk);
* Keuze voor MVC *pattern* (makkelijk, redelijk belangrijk);
* Implementeren van de Controllers (vrij moeilijk, zeer belangrijk);
* CRUD-functionaliteit voor dronevluchten, drones en piloten (vrij gemakkelijk, zeer belangrijk);
* Schrijven van de Views (de website) (vrij moeilijk, belangrijk);
* Implementeren van *searchable* en *sortable* tabellen met *paging* (gemiddeld, onbelangrijk);
* *Dependency Injection* met Unity (makkelijk, redelijk belangrijk);
* Uploaden van bestanden die geparset kunnen worden (moeilijk, zeer belangrijk)
* Voorzien van additionele razorpagina’s (View) die details geven van de data die ingelezen wordt (vrij makkelijk, redelijk belangrijk);
* Tonen van gepaste foutpagina’s op de website aan de gebruiker (makkelijk, belangrijk).

## Sprint 2

* Projecttabelin databank met benodigde velden (vrij gemakkelijk, belangrijk)
* Totale vliegtijd van een drone automatisch berekenen en toevoegen aan de databank (vrij gemakkelijk, belangrijk);
* Velden uit logboeken(zoals *type of activity*) toegevoegd aan databank en mogelijkheid voorzien om deze velden in te vullen via de *user interface* (vrij gemakkelijk, belangrijk);
* *Reverse geocoding*, omvormen van latitude- en longitudecoördinaten naar de naam van de locatie (vrij moeilijk, belangrijk);
* Duidelijk weergeven van verplichte velden bij invullen van gegevens in de *user interface* (vrij makkelijk, redelijk belangrijk);
* Dronevluchten per piloot en drone in apart overzicht; te bekijken in *user interface* (gemiddeld, redelijk belangrijk);
* Extra functies GUI, zoals van piloot naar bijhorende vluchten kunnen gaan (gemiddeld, redelijk belangrijk);
* Bij het uploaden van files staat een *progress bar* die weergeeft hoeveel procent van de file reeds geüpload en verwerkt (geparset) is (moeilijk, redelijk belangrijk);
* *Controletool ctrl points* in *point cloud* (vrij moeilijk, belangrijk);
* De *track* van de drone in de map view kunnen visualiseren op basis van een attribuut (hoogte, batterijstand...) met een gepaste *color ramp*  (vrij moeilijk, zeer belangrijk);
* Legende met informatie over de vlucht in de *map view* (moeilijk, belangrijk);
* Visualisatie van *point clouds*, gcp’s en ctrl’s (vrij moeilijk, zeer belangrijk);
* Een *toggle list* bij de *map view* zodat de gebruiker zelf kan kiezen welke *layer* (soort data) getoond wordt van een dronevlucht (gemiddeld, belangrijk);
* *Pop-up templates* als de gebruiker op eender welk punt (gcp, ctrl...) klikt om de bijhorende coördinaten en andere attributen te zien (vrij moeilijk, belangrijk);
* Zoekbalk en *measurement widget* in de *map view* (gemakkelijk, redelijk belangrijk);
* Er kan van attribuut (en bijhorende *color ramp)* gewisseld worden door op een bepaalde toets te drukken om zo de *track* te visualiseren. De legende van een vlucht kan verborgen of getoond worden, eveneens met een toets. (gemakkelijk, vrij belangrijk).

## Sprint 3

* *Images* die door de drone genomen werden kunnen upgeload worden en bekeken worden op hun eigen pagina en in de *map view* (gemiddeld, belangrijk);
* De *parsers* voor .xyz en .dat bestanden werden herschreven wat leed tot grote performantiewinsten (moeilijk, belangrijk);
* Een *map overview* van alle dronevluchten waarbij de gebruiker naar een specifieke vlucht kan navigeren (gemiddeld, belangrijk);
* Verbeteringen in de *map view* GUI, waaronder een aanklikbare menu voor *color coding* (gemiddelijk, vrij belangrijk);

# Systeemarchitectuur

In dit hoofdstuk wordt besproken hoe het programma doelstellingen 1, 2, 3 en 4 realiseert en hoe deze opgebouwd zijn. Allereerst wordt de webapplicatie beschreven vanuit het oogpunt van het *deployment diagram*. Vervolgens wordt dieper ingegaan op de databank a.d.h.v. een databankdiagram. Verder wordt het MVC pattern dat wordt gebruikt in ASP.NET toegelicht m.b.t. dit project. Als laatste worden de front-end en de javascriptbestanden beschreven, en wordt de flow van de webapplicatie aan de hand van twee *sequence diagrams* geïllustreerd.

## High-levelsysteemmodel (*deployment diagram*)

Het *deployment diagram* in **Figuur 3** bestaat uit twee onderdelen: de SQL Server component en de ASP.NET MVC 5 component. De SQL Server bevat de databank waarin alle data over dronevluchten bijgehouden wordt en waaraan de gebruiker data kan toevoegen door middel van de CRUD-functionaliteiten. De databank wordt in de model component van het MVC pattern voorgesteld als een verzameling van entiteiten aan de hand van het Entity Framework 6.



Figuur 3: deployment diagram

De views zijn niet rechtstreeks gekoppeld aan het model maar communiceren via tussenliggende *controllers*. De gepaste controller wordt gekozen via routing en geeft steeds de veranderingen door aan het model. Op basis van dit model wordt de databank ten slotte aangepast.

## Databankdiagram

Het databankdiagram visualiseert de structuur van de informatie die opgeslagen dient te worden in de databank en de relaties tussen deze data. Dit model bestaat uit drie grote onderdelen: DroneFlight, QualityReport en DroneLogEntry. Deze zijn respectievelijk te zien in **Figuren 3**, **4** en **5** op pagina’s 17, 19 en 20.

Alle andere tabellen zijn aan deze hoofdtabellen gelinkt met een één-op-één relatie of een één-op-veel relatie. Het linken gebeurt respectievelijk aan de hand van een *mapping* van een *Primary Key* op een andere *Primary Key*, of aan de hand van een *mapping* van een *Primary Key* op een *Foreign Key*. Een verbinding met een sleutel aan beide kanten wijst op een één-op-één relatie en een verbinding met een sleutel en een oneindigheidsteken wijst op een één-op-veel relatie.

Zo is de *Primary Key* van de tabel DepartureInfo, DepartureInfoId, één-op-één gemapt op de *Primary Key* van de tabel DroneFlight, FlightId. Dit betekent dat een DroneFlight slechts één DepartureInfo kan hebben. Een dronevlucht kan immers maar één starttijdstip hebben.

Een voorbeeld van een één-op-veel relatie gebeurt met een mapping van een *Primary Key* op een *Foreign Key*. Zo heeft een dronevlucht (tabel DroneFlight) meerdere Ground Control Points (tabel GroundControlPoints). Deze laatstgenoemde tabel heeft een *Foreign Key*, DroneId, die gemapt wordt op de *Primary Key*, DroneId, van de tabel DroneFlight. De *Foreign Key* kan dus gezien worden als een sleutel die de tabel met de *Primary Key* toegang geeft tot de tabel die de *Foreign Key* bevat, in dit geval de tabel GroundControlPoints.

## DroneFlighttabel

De eerste en tevens ook belangrijkste tabel in het databankdiagram is de DroneFlight tabel, te zien op **Figuur 4** op pagina 17.



Figuur 4: DroneFlighttabel met al haar relaties

Deze tabel vormt de basis van het model en houdt alle informatie bij over de dronevluchten, zoals de datum en locatie van de vlucht. Verder bevat deze tabel ook een reeks booleans (hasTFW, hasGCPs, hasCTRLs, hasDepInfo, hasDestInfo, hasQR, hasXYZ en hasDroneLog) die bijhouden of een bepaald bestand of een bepaald type informatie ingelezen is en op dit moment bijgehouden wordt.

Deze tabel heeft bovendien drie *Foreign Keys*: een DroneId, PilotId en ProjectId. Op deze manier kunnen drones en piloten informatie opvragen over hun vluchten en kan elke vlucht gelinkt worden met het project waartoe de vlucht behoort. Merk op dat een drone, een pilot en een project met een dronevlucht een één-op-veel relatie beschrijven. Een drone en een pilot kunnen immers meerdere dronevluchten uitvoeren en binnen een project kunnen er meerdere dronevluchten plaatsvinden.

De DepartureInfo en DestinationInfo tabellen bevatten informatie over een drone zijn tijdstip van vertrek en van aankomst bij een dronevlucht. In deze tabellen wordt ook informatie bijgehouden over de coördinaten van vertrek en aankomst.

De tabel PointCloudXYZ bevat de coördinaten en RGB-waarden die nodig zijn om een *point cloud* aan te maken. Een vlucht heeft vaak miljoenen PointCloudXYZ *entries*.

De tabel GroundControlPoints beschrijft de coördinaten van een *ground control point* en bevat ook een *Foreign Key*, FlightId, dat een vlucht toelaat om deze informatie op te vragen. Deze tabel bevat x-, y- en z-waarden om alle foto’s uit de RawImages tabel geografisch juist te positioneren (Understanding world files, z.j.). Zodoende kunnen alle foto’s aan elkaar gehangen worden om zo een groot beeld te verkregen van de werf.

Het voorgaande geldt ook voor de tabel CTRLPoints, dewelke punten beschrijft die gebruikt worden ter verificatie van de juistheid van ingelezen datapunten.

De tabel TFW bevat rotatie- en translatie informatie, alsook wereldcoördinaten die gebruikt worden bij een *tiff image* bestand in een GIS applicatie. Deze informatie laat toe een andere, externe databank aan te spreken en de juiste resultaten te verkrijgen in de vorm van een kaart.

De laatste tabel DroneAttributeValues houdt informatie over de drone bij die tijdens de dronevlucht wijzigt en informatief kan zijn voor analyse naderhand.

De DroneFlighttabel is met een één-op-één relatie verbonden met de QualityReport tabel.

## QualityReporttabel

In de QualityReporttabel in **Figuur 5** wordt alle data bijgehouden die ingelezen wordt uit het kwaliteitsrapport. Dit is een PDF-bestand dat de output beschrijft van een analyse in het programma Pix4D. Tabellen die een relationeel verbonden zijn met deze tabel zijn: Uncertainty, AbsoluteGeolocationVariance en GCPError. Op deze manier is de ingelezen informatie uit het kwaliteitsrapport op een logische manier gegroepeerd.



Figuur 5: QualityReporttabel met al haar relaties

## DroneLogEntrytabel

In **Figuur 6** staat de DroneLogEntrytabel centraal. Deze tabel bevat voor elke *tick* data ingelezen uit het dronelogbestand. Alle tabellen (DroneGPS, DroneOA, DroneIMU\_ATTI, DroneRTKData, DroneMotor en DroneRC) hebben een één-op-één relatie met de hoofdtabel, DroneLogEntry, en bevatten dus waarden voor elke *dronelog entry*. De tabel DroneLogEntry zelf heeft een één-op-veel relatie met de DroneFlighttabel, aangezien elke dronevlucht meerdere *dronelog entries* heeft.



Figuur 6: DroneLogEntrytabel met al haar relaties

## MVC

De ontwikkeling van de webapplicatie gebeurt met ASP.NET in de .NET omgeving van Microsoft met een combinatie tussen de *high-level* computertaal C#, HTML, CSS en JavaScript. Er werd gekozen voor een MVC (Model-View-Controller) *pattern*.

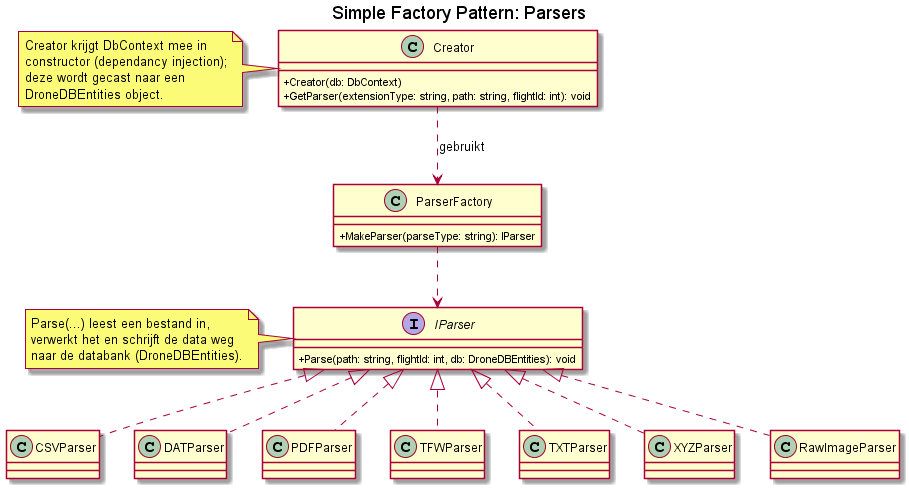
## Model

Op basis van het databankmodel worden met het Entity Framework 6 via ORM entiteitsklassen aangemaakt. Hier worden databanktabellen voorgesteld als klassen en databankkolommen als velden van de overeenkomstige velden. Dit laat toe om op eenvoudige wijze te communiceren met de databank zonder echt expliciet SQL te moeten spreken. Dit geheel stelt de *Model* component in MVC voor.

## Simple Factory pattern voor parser-klassen

Dit onderdeel beschrijft het *simple factory pattern* voor het parsen van de bestanden. Dit parsen van data uit bestanden gebeurt via parser-klassen. Deze klassen worden gegroepeerd in een *simple factory pattern* (**Figuur 7**), waarbij een klasse Creator aangemaakt wordt in de webapplicatie. Deze klasse spreekt vervolgens met de methode GetParser een factory aan, ParserFactory, die vervolgens met de methode MakeParser de juiste parser-klasse aanmaakt en teruggeeft aan de Creator. Daarna vindt het parsen plaats met de methode Parse van de aangemaakte parser-klasse.

De constructor van Creator krijgt via *Dependency Injection* (met *Unity*) een instantie van de databank mee, zodat de parsers de ingelezen en verwerkte data kunnen wegschrijven naar de databank. Dankzij deze injectie wordt het aanmaken van dure databankconnecties beperkt.



* CSVParser

Figuur 7: simple factory pattern met parser-klassen

Deze *parser* leest de CSV-bestanden in die de *ground control points* (GCPs) en *controle points* (CTRLs) bevatten voor een dronevlucht.

* DATParser

Deze *parser* converteert een *drone log* DAT-bestand automatisch naar een CSV-bestand. Dit bestand wordt vervolgens geparset en alle nuttige informatie wordt weggeschreven naar de juiste dronevlucht in de databank.

Deze *parser* maakt gebruik van ADO.NET. ADO.NET leidt tot grote performantiewinsten in vergelijking met Entity Framework. Dit komt door het *object-oriented* principe van Entity Framework, wat leidt tot het aanmaken van objecten bij elke te *parsen* lijn. Het aanmaken van objecten is een dure operatie tijdens het *parsen*. Het wegschrijven van deze data naar de databank kan bovendien ook niet snel genoeg gebeuren, waardoor de keuze voor ADO.NET snel gemaakt kon worden. Hier worden *commands* gebruikt die bij elke gelezen lijn uitgevoerd worden om de data efficiënt weg te schrijven naar de databank. Elke *command* wordt ingesteld op basis van Parameter objecten om *sql injections* te voorkomen.

* PDFParser

Deze *parser* leest het kwaliteitsrapport van een dronevlucht in en schrijft de nuttige informatie weg naar de databank. Hier wordt er gebruik gemaakt van *Ivy Pdf Parser*. Deze library is betalend.

* TFWParser

Deze *parser* leest het TFW-bestand van een dronevlucht in en schrijft deze informatie weg naar de databank.

* TXTParser

Ongebruikt.

* XYZParser

Deze *parser* leest een XYZ-bestand lijn per lijn in en schrijft de XYZ-coördinaten en RGB-waarden weg naar de juiste dronevlucht in de databank. Indien er ook een *intensity* aanwezig is op één of meerdere lijnen in dit bestand, dan wordt dit ook mee ingelezen.

Het gebruik van ADO.NET is analoog aan dat van de DATParser.

* RawImageParser

De gebruiker kan voor een vlucht afbeeldingen uploaden. Eerst worden de *images* ingelezen door de gepaste *parser*-klasse RawImageParser*.* Deze *parser* leest een afbeelding en zijn *meta data* in. Dit wordt gerealiseerd met de *library* ‘MetadataExtractor’. Deze *meta data* bevat belangrijke info, zoals de coördinaten waar deze foto genomen werd en bijkomende informatie over de drone GPS en camera. De afbeelding wordt volledig ingelezen met een FileStream en daarna weggeschreven naar de databank onder de vorm van een byte *array*.

Een bijkomende *thumbnail* wordt met een hulpmethode aangemaakt en weggeschreven naar de databank. Deze *thumbnails* worden gebruikt in *pop-ups* in de view map van een individuele vlucht (zie **Figuur 8**). Elke entry in de RawImagestabel van de databank bestaat uit de afbeelding en de *thumbnail* (beide onder de vorm van bytes), en de *meta data*.



Figuur 8: pop-up van een afbeelding in de view map

De klant kan voor een vlucht deze afbeeldingen bekijken in de webapplicatie indien deze geüpload werden. Dit kan door voor een dronevlucht op 'Images' te klikken in het overzicht van de dronevluchten of door naar de detailpagina van een dronevlucht te surfen. De gebruiker kan nu alle geüploade images en hun bijhorende *meta data* zien (**Figuur 9**)*.*



Figuur 9: de Images-pagina van een dronevlucht

## Controllers

## ‘View’-controllers

Het project bevat verscheidene ‘View’-*controllers* (de C in het MVC *pattern*) die instaan voor navigatie (routing):

* de ProjectController;
* de DroneFlightsController;
* de DronesController;
* de PilotsController;
* de FilesController;
* de MapController;
* de AccountController;
* de ManageController;
* de HomeController.

De vier eerste controllers beschikken allen over CRUD-functionaliteit (Create, Read, Update, Delete), maar de DroneFlightsController beschikt ook nog over de mogelijkheid om te routen naar razorpagina’s die additionele informatie tonen, zoals de inhoud van het kwaliteitsrapport of een overzicht van afbeeldingen.

De FilesController zorgt ervoor dat de bestanden die de gebruiker uploadt voor een vlucht juist worden afgehandeld en maakt hiervoor gebruik van de *parser*-klassen. Deze controller beschikt bovendien voor een methode Export dat de informatie van een piloot of een drone (inclusief de toegekende vluchten) exporteert naar een CSV-bestand of PDF-bestand. Als laatste heeft de controller ook een functie GetStatus dat wordt gebruikt om met de *client-side* de progressie van het *parsen* van een bestand te communiceren.

Indien er tijdens het uploaden iets fout gaat, dan kan deze *controller* de *client* hiervan op de hoogte brengen door een *error code* te communiceren (cf. 2.2.1 Uploaden van files).

De MapController zorgt ervoor dat een id wordt meegegeven in de ViewBag van een de ViewMap.cshtml, zodat in de View de juiste javascript kan worden aangesproken.

* Er wordt een overzicht gebouwd van alle aanwezige dronevluchten met een *drone log* indien er geen id wordt meegegeven (id = NULL).
* Er wordt een track gebouwd voor een individuele dronevlucht indien er een geldige id wordt meegegeven (d.i. een id van een dronevlucht dat effectief bestaat). De controle hiervoor wordt in de *controller* uitgevoerd.

TODO: Niels (Account & Manage Controllers)

## ‘Web API’-controllers

Naast 'View'-controllers heeft het project verschillende 'Web API'-controllers. Deze klassen staan in voor de communicatie tussen de client (in dit project de Javascript code) en de server. Deze zijn:

* de CTRLPointsController;
* de DroneFlightsAPIController;
* de DroneGPsController;
* de GCPController;
* de PointCloudXYZsController;
* de RawImagesController;
* de ThumbnailsController;

not sure of deze uitleg voor figuur 10 er bij moet en/of deze uitleg volledig is

Figuur Werking Web API controllers

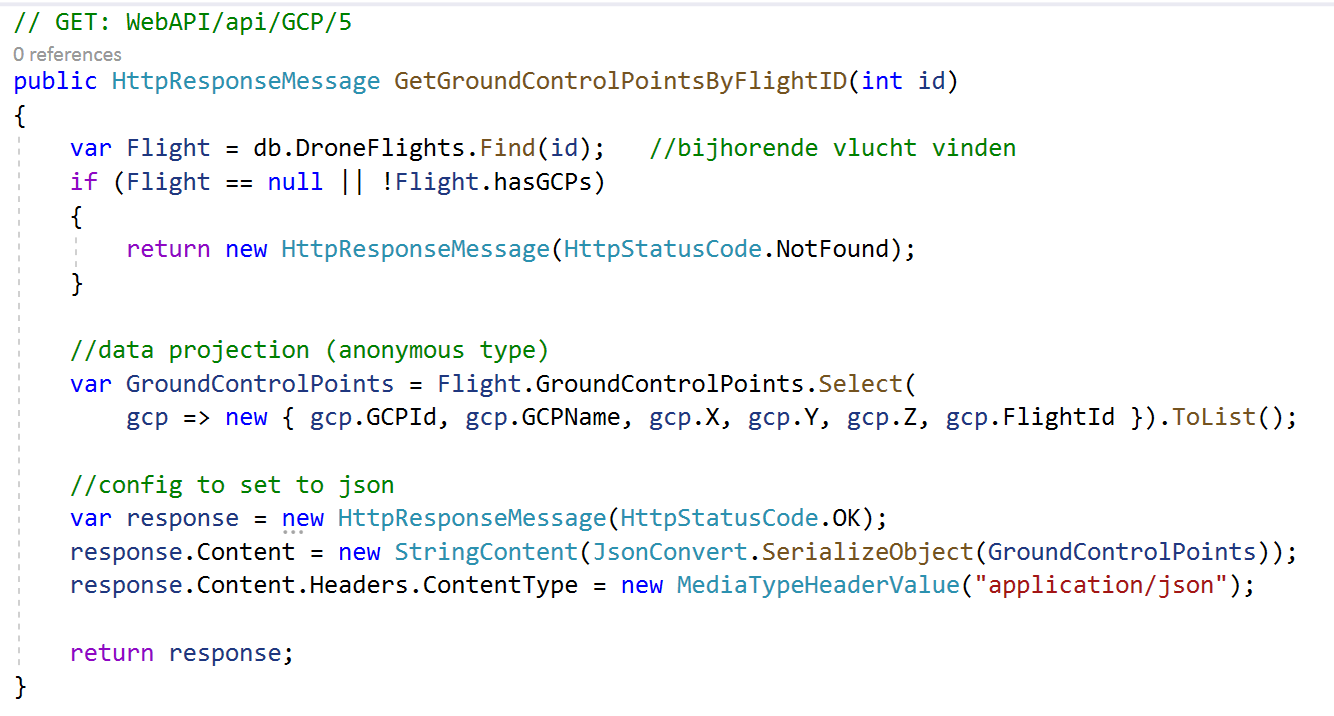
De Web API controllers vragen data op uit de databank. Op basis van de bestaande Model klassen wordt een HTTP response gemaakt met de *projected data.*

Als eerste moeten de controllers geconfigureerd worden op hun *routing.* *De ASP.NET* Routing-module is verantwoordelijk voor het toewijzen van inkomende browseraanvragen

aan bepaalde MVC-controlleracties. In het project wordt er gewerkt met de HTTP-route "WebAPI/api/{controller}/{id}/{imageid}". Elke data wordt opgevraagd op basis van een *flight id* en eventueel een *image id* voor *images.*

De eerste vijf controllers zijn verantwoordelijk voor datapunten van de drone. Elk soort data heeft een eigen controller. De controllers handelen elk een GET-request af op basis van een *flight id.*

Deze klassen werken op een gelijkaardige manier. Als eerste vragen ze de nodige data op uit de databank met behulp van *Entity Framework.*  Ten tweede wordt een *anonymous type* object aangemaakt op basis van deze data met behulp van een *lambda expression.* Als laatste wordt de *anonymous type* geconfigureerd tot een *HTTP Response Message* die een *JSON* object bevat. Deze JSON wordt teruggegeven door de methode.



Figuur GET methode van de GCPController

De laatste twee controllers staan in voor *images* and hun *thumbnails.* Als eerste halen ze de juiste image op uit de databank op basis van een *flight id* en een *image id.* Dit gebeurt aan de hand van een *sql query*. Deze *query* gebruikt Parameter objecten om zich af te schermen tegen *sql injections*.

Sinds het *bad practice* is om afbeeldingen als bytes door te sturen naar de front-end, handelen deze controllers de conversie af van een *byte array* naar een werkelijke afbeelding en sturen ze deze terug.

De RawImagesController heeft daarnaast ook een methode om alle *image* data op te vragen op basis van een *flight id*, gelijkaardig aan de andere controllers.

## Views

Razorpagina’s zijn bestanden van het type cshtml en beschrijven de *View* in het MVC pattern. Voor elke methode in een *controller*, die een ActionResult teruggeeft als return-type, bestaat een razorpagina met overeenkomstige naam. In onderstaande afbeelding, **Figuur 10**, is de cshtml-pagina te zien waar de gedetailleerde informatie van een dronevlucht op weergegeven wordt. Een groene of rode knop wijst respectievelijk op het aanwezig of afwezig zijn van het overeenkomstige document.



Figuur 12: detailpagina van een Drone Flight

## Javascriptklassen

In dit onderdeel wordt de functionaliteit van de webapplicatie beschreven met betrekking tot de javascriptklassen in dit project. Er wordt JavaScript gebruikt om de gebruiker welbepaalde bestanden te kunnen laten uploaden (2.4.1), om de vluchten te visualiseren (2.4.2) en functionaliteit te voorzien om zogenaamde *Data Tables* te kunnen aanmaken (2.4.3).

## Uploaden van files

De gebruiker kan aan een dronevlucht documenten toevoegen. Voorbeelden hiervan zijn: het logboek van een drone, een TFW-bestand, het kwaliteitsrapport, het XYZ-bestand, foto’s van de vlucht en CSV-bestanden. De eis van doelstelling 2 (cf. Inleiding) stelt dat de ingelezen bestanden moeten worden opgeslagen in de databank na het verwerken (*parsen*) ervan. Deze verwerking gebeurt, zoals eerder vermeld in 2.4.1, met *parser*-klassen die elk een bepaald type bestand afhandelen.

De controller FilesController staat in voor het juist afhandelen van de aanvraag van de gebruiker om bestanden te uploaden. De gebruiker navigeert naar de uploadsectie van de webapplicatie via het pad /Files/Index/{DroneFlightId}, waarbij DroneFlightId aangeeft voor welke dronevlucht de gebruiker bestanden wil uploaden. Er is gepaste foutenafhandeling bij het opgeven van een onbestaande DroneFlightId of het niet opgeven van een DroneFlightId. De gebruiker wordt op een gepaste wijze geïnformeerd wat er precies fout gaat.

Aan de *client-side* verzorgt een stuk javascriptcode, het bestand **fileUpload.js**, de communicatie met de *server-side*. Dit bestand vervult meerdere functies:

* Het controleert of de gebruiker een bestand meegeeft bij het uploaden. Het zorgt er met andere woorden voor dat de gebruiker niet zomaar op de uploadknop kan klikken, zonder één of meerdere bestanden toe te voegen. Dit wordt bereikt door een abort uit te voeren op wat men probeert te versturen (**Figuur 11**). Aan server-side wordt er nogmaals gecontroleerd of er weldegelijk een bestand werd meegegeven; dit als dubbele controle.



Figuur 13: code snippet: abort-functionaliteit

* Het controleert de toegestane bestandsextensies, zoals weergegeven in **Figuur 12**. Indien een gebruiker een bestand probeert te uploaden dat niet is toegestaan, dan geeft de webapplicatie de gebruiker hier een melding van. Er wordt een lijst getoond met extensies die wel zijn toegelaten. De uploadknop wordt vergrendeld tot wanneer de gebruiker geldige bestanden opgeeft. De gebruiker kan nu opnieuw proberen uploaden. Aan server-side wordt er nogmaals gecontroleerd of de bestanden die werden meegegeven een geldige extensie hebben; dit als dubbele controle.



Figuur 14: code snippet: controle op bestandsextensie

* Het uploaden van de bestanden via jQuery’s AjaxForm. Dit bevat 3 *callback*-functies: beforeSend, uploadProgress en success.
  + beforeSend voert de controle uit op het aantal bestanden dat werd toegevoegd. Indien dit 0 is, dan wordt er een abort uitgevoerd. In de andere gevallen werden er één of meerdere bestanden meegegeven. Resultaatvelden worden op *hidden* gezet, zoals in **Figuur 13**, aangezien de gebruiker reeds in een tweede rond van uploaden kan zitten, zonder de pagina te *refreshen*.



Figuur 15: code snippet: verbergen van div-elementen

In **Figuur 14** worden de velden met resultaten verborgen en worden variabelen, zoals firstCheck en filesLeftToParse, gereset naar hun initiële default-waarde of geüpdatet naar een nieuwe waarde, zoals in het geval van de variabele totalFilesToParse.

Figuur 16: code snippet: beforeSend callback bij uploaden van één of meerdere bestanden

* + uploadProgress verzorgt het eigenlijke uploaden van de bestanden (zie **Figuur 15**). De progress bar wordt periodiek geüpdatet door de AjaxForm en van zodra het uploaden voltooid is (percentComplete == 100), wordt de gebruiker hiervan op de hoogte gebracht. Vervolgens wordt alles in gereedheid gebracht om de gebruiker de voortgang van het parsen te kunnen tonen en wordt de <div> met id progressField getoond. Het eigenlijke parsen wordt gestart met de functie startParsing.

Figuur 17: code snippet: uploadProgress callback bij het uploaden van één of meerdere bestanden

De functie startParsing (zie **Figuren 16**, **17** en **18** op deze pagina, en pagina 31) reset de waarde van de progress bar terug naar 0 en roept met de functie setTimeout de functie parse op. Deze laatstgenoemde functie wordt blijft opnieuw en opnieuw afgevuurd worden zolang het aantal verwerkte bestanden niet gelijk is aan het aantal verzonden bestanden. Van zodra dit wel het geval is, zal het laatst ontvangen HTTP-bericht ook de lijst met eventueel gefaalde bestanden bevatten. Alles wordt in gereedheid gebracht om de gebruiker de juiste velden te tonen (via de functies show of hide op de nodige <div>-elementen). Er wordt gecontroleerd of er effectief bestanden zijn die niet werden ingelezen; in elk van de twee gevallen (ja of neen) wordt de gepaste informatie (feedback) aan de gebruiker getoond.

Figuur 18: code snippet van de functie startParsing

Figuur 19: code snippet: helper-functie updatet de progress bar via aan HTTP get call



Figuur 20: code snippet: instellen en tonen van alle resultaatvelden na afronden parsen

* + success wordt uitgevoerd nadat het uploaden werd voltooid. In deze *callback*-functie wordt de gepaste *error*-boodschap getoond, indien dit nodig is. Dit is weergegeven in **Figuur 19** op pagina 32.



Figuur 21: code snippet: de verschillende error codes bij het uploaden van bestanden

* Het tonen van een *progress bar* voor het uploaden en het parsen van de bestanden. Er wordt periodiek gecommuniceerd met de server-side of een status op te halen van het parsen. Op deze manier kan aan de gebruiker een voortgang worden getoond van het parsen van zijn bestanden.
* Indien er iets misloopt aan server-side, dan wordt de gebruiker hier ook van op de hoogte gebracht. Hiervoor zijn zelfgemaakte errorcodes in het leven geroepen:
  + 1: *succes*;
  + 0: *no files submitted*;
  + 2: *no drone flight specified*;
  + 3: *drone flight does not exist*;
  + 4: *someone else is already uploading*;
  + 5: *invalid file type*.
* Eén limitatie van de *file uploader* is dat deze slechts één gebruiker per keer toelaat om te uploaden. Dit zou verholpen kunnen worden door een unieke id toe te kennen aan een gebruiker. Heden wordt een boodschap weergegeven indien een andere gebruiker reeds aan het uploaden is.
* De gebruiker kan gedurende het parsen van de bestanden zien welk bestand de FilesController aan het verwerken is. Hiervoor wordt periodiek een *call* gemaakt naar de server-side. Deze antwoordt met een *HTTP-response* waarvan de inhoudt gemaakt wordt met de methode GetStatus van de FilesController. Deze methode maakt een anoniem object dat naar JSON geconverteerd wordt. De inhoud bevat steeds:
  + De huidige *progress* van een bestand;
  + Het parse-resultaat van een parser;
  + De bestandsnaam van het bestand dat op dit moment geparset wordt;
  + Het aantal bestanden dat nog geparset moet worden;
  + (Een lijst met de namen van de bestanden die niet geparset werden).

De allerlaatste call naar GetStatus gebeurt nadat alle bestanden geparset zijn. In dit geval staat de waarde van het aantal bestanden dat nog geparset moet worden op 0 en wordt een lijst gebouwd met de namen van alle bestanden die niet geparset werden (het laatste puntje van bovenstaande lijst).

* Aan client-side wordt dan een *unordered list* gebouwd met bovenvermelde bestandsnamen, zodat de gebruiker kan zien welke bestanden niet geparset werden (doorgaans omwille van duplicaten, d.w.z. dat deze bestanden reeds aanwezig waren voor deze vlucht).
* De gebruiker kan kiezen om nogmaals bestanden te uploaden of om terug te keren naar de lijst van dronevluchten of de detailpagina van de dronevlucht waarvoor hij zojuist bestanden heeft geüpload/proberen uploaden.

Het uploaden van bestanden via de webapplicatie is onderhevig aan enkele beperkingen:

* Zoals eerder vermeld: slechts één gebruiker per keer kan uploaden.
* Er kan maximaal 2.147 GB aan bestanden geüpload worden. Deze beperking wordt opgelegd door ASP.NET zelf in het bestand Web.config. De gebruiker kan daarom op dit moment enkel bestanden kleiner dan (of in totaal) 2.147 GB uploaden.
* Bij het uploaden van zeer veel afbeeldingen kan de server mogelijks een ‘not enough memory’ fout opgooien. Dit is sterk afhankelijk van wat er op de server qua geheugen beschikbaar is voor de webapplicatie. Indien hij of zij afbeeldingen wil uploaden, dan is deze limiet voorlopig ingesteld op 500 MB. Dit kan mogelijks verhoogd worden, indien meer geheugen kan worden toegekend aan de applicatie. Het geheugen wordt terug vrijgegeven na het *parsen* van de afbeeldingen. Een test gaf aan dat het uploaden van 70 afbeeldingen de applicatie tot zijn ingestelde limiet van 3 GB liet gaan.
* Aan de hand van een lus wordt de totale som gemaakt van de bestandsgroottes, zoals weergegeven in Figuur 20 op pagina 34. Er wordt vervolgens gecontroleerd op de bestandsgroottes (Figuur 21 op pagina 34). Indien deze overschreden worden, dan wordt een gepaste boodschap getoond aan de gebruiker.



Figuur 22: berekenen van totale bestandsgroottes



Figuur 23: controle bestandsgroottes

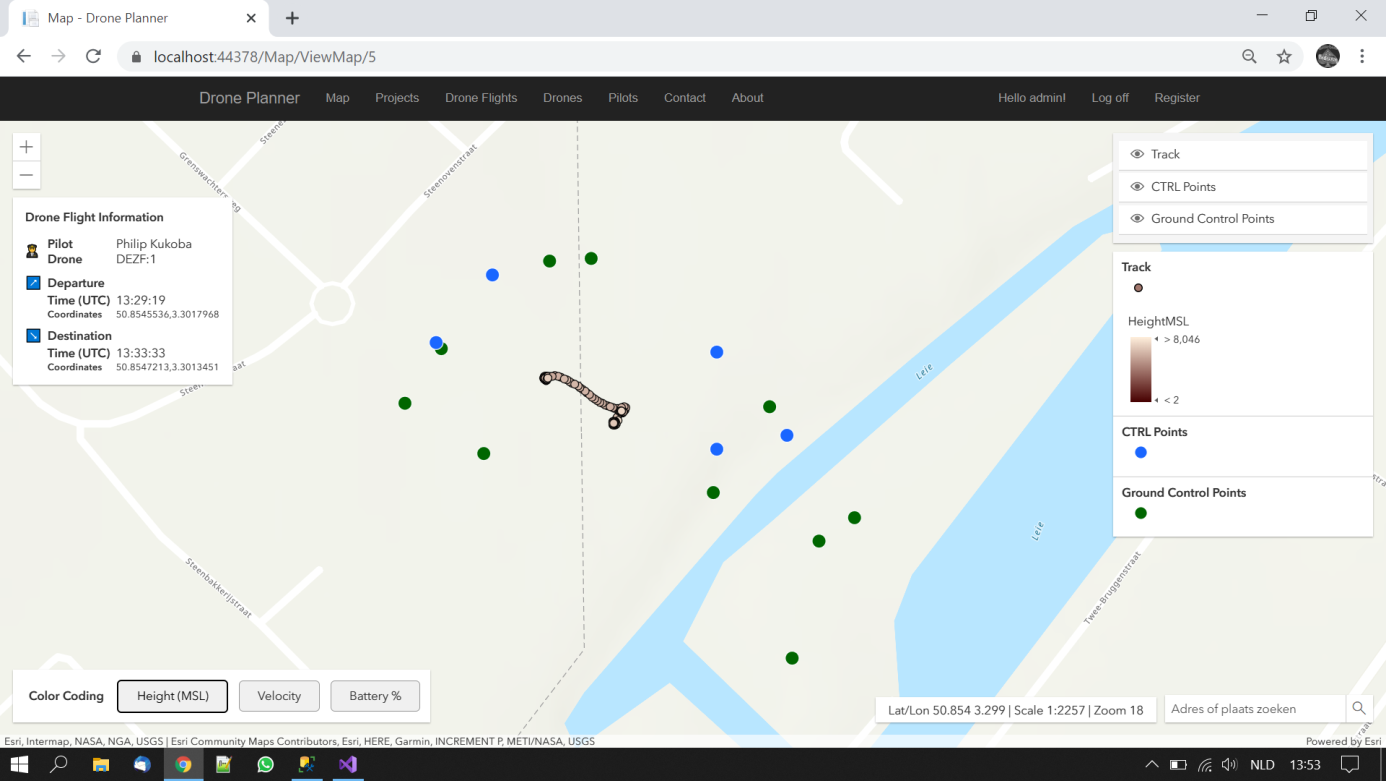
## Visualisatie met ArcGIS JS API

De gebruiker kan via de navigatiebalk navigeren naar 'Map'. Deze *map view* biedt een overzicht van alle dronevluchten op een kaart. Voor elke vlucht komt er een icoon op de startplaats van deze vlucht. Er kan op het icoon klikken om een *pop-up* te zien met informatie over deze vlucht. Deze informatie bevat de naam van de piloot, de naam van de drone en informatie over start- en eindtijd van de vlucht. Deze *pop-up* bevat ook een knop om deze specifieke vlucht te bekijken in een eigen *map view*.



Figuur 24: pop-up voor een vlucht in een map overview

De front-end bestaat uit verschillende klassen uit de ArcGIS Javascript API.



Figuur 25: de map overview met verschillende datapunten van een dronevlucht

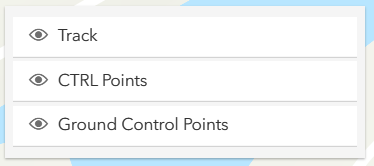
**Map**:De Map klasse is een container voor de *layers* van de applicatie en heeft bijhorende methodes voor deze *layers*.

**MapView**: De MapView toont een 2D view van de Map.De layers worden gerenderd door deze klasse. Om de *MapView* zichtbaar te maken heeft deze een referentie nodig naar de *Map* en naar het *Document Object Model* (DOM) element waar deze in hoort.

**Graphic:** Een Graphic stelt een visuele figuur voor op de *map*. Deze heeft typisch de velden *geometry* (*point*, *polyline* of *polygon*), een referentie naar de bijhorende *layer*, een eventuele PopupTemplate en een aantal zelfgemaakte attributen (vooral van belang bij de *track* van de vlucht).

**SpatialReference**: Deze klasse definieert de ruimtelijke referentie van de te visualiseren data. Het geeft aan welk geografisch projectiesysteem wordt gebruikt om geografische punten op de kaart te situeren. Deze ruimtelijke referentie kan het gemakkelijkst gespecifieerd worden met behulp van het *well-known ID (WKID)* veld van de SpatialReference klasse. Voor dit project wordt het coordinatensysteem "Belgian Lambert 1972" gebruikt met WKID 31370.

**LayerList**: Een *widget* die een legende toont van alle layers van de map met opties om elke layer te verbergen of te tonen. De LayerList heeft een referentie nodig naar de MapView.



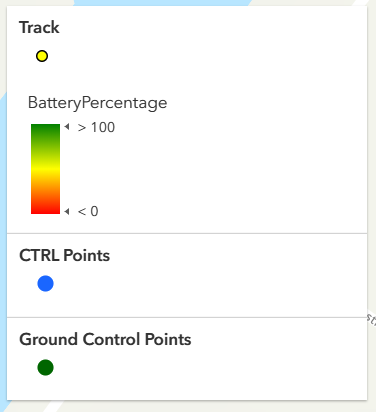
Figuur 26: LayerList widget

**Search**: Een zoekbalk met suggesties (zoals bij Google Maps).

****

Figuur 27: search widget

**Legend**: Toont een legende van alle *layers* van de MapView met een voorbeeld van de visuele figuren in elke *layer*. De Legend updatet automatisch als een *layer* verborgen of getoond wordt, of als de highlighte attribuut van de track verandert.



Figuur 28: legend klasse

**PopupTemplate**: Deze klasse definieert de inhoud van de *pop-up* van een specifieke Graphic. Deze popup wordt gevuld met de coördinaten en alle andere attributen van de Graphic.



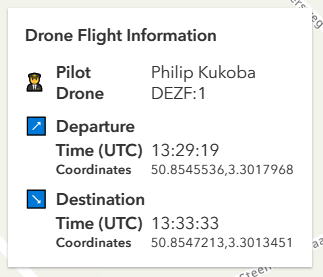
Figuur 29: PopupTemplate

**FeatureLayer:** Een FeatureLayer stelt éen enkele *layer* voor. Deze heeft verschillende velden die ingesteld moeten worden. Het *source* veld krijgt een *array* van Graphic objecten mee die bij deze layer horen. Het *fields* veld bevat een array van alle mogelijk attributen bij de Graphic objecten, nodig voor de PopUpTemplate. Waarop volgt: het *popUpTemplate* veld moet een referentie naar het gepaste PopUpTemplate object bevatten.

Het *renderer* veld bepaalt hoe de Graphic objecten gevisualiseerd moeten worden (zoals kleur, grootte en *opacity*). Bij de *layer* van de *track* krijgt de FeatureLayer een *custom renderer* mee die een attribuut van de *track* punten, bv. hoogte of batterijstand, visualiseert met een *visual variable* object*.* Deze *visual variable* heeft een gepaste *color ramp* voor een specifiek attribuut. Bijvoorbeeld: als de *track* gevisualiseerd wordt op basis van de batterijstand van de drone kleurt een punt rood als de batterijstand dicht bij 0% is, kleurt het geel bij 50% en kleurt het groen bij 100%. Dit kleurenverloop is een continu spectrum *(color ramp).*

Als laatste moet het *objectIdField* ingesteld worden op het unieke ID attribuut van de Graphic objecten.

**Feature:** Deze *widget* toont data volgens zijn *popUpTemplate* veld. Deze klasse wordt gebruikt wanneer informatie constant moet getoond worden, in tegenstelling tot pop-ups. De Feature bevat een tabel met informatie over de dronevlucht.



Figuur 30: Feature klasse

Daarnaast is er een *color coding* menu aanwezig links onderaan om de *track* te visualiseren op basis van hoogte, snelheid of batterijstand. Deze menu is gebaseerd op de feature klasse. De acties worden geïmplementeerd met jQuery.



Figuur Color coding menu

## Data Tables

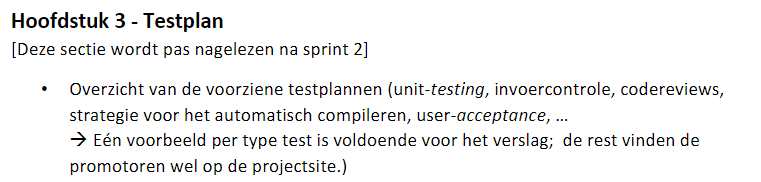
Het script datatabel\_script.js voorziet de nodige functionaliteit om gemarkeerde HTML-tabellen om te zetten naar zogenaamde *data tables*. Deze tabellen laten *paging*, *searching* en *sorting* toe op HTML-tabellen.

In totaal worden 10 tabellen omgezet naar dit genre tabellen. Het gaat om:

* de index van de *drone flights*;
* de index van de *projects*;
* de index van de *drones*;
* de index van de *pilots*;
* de index van een *pilot* zijn of haar *drone flights*;
* de index van een *drone* zijn *drone flights*;
* de index van een *project* zijn *drone flights*;
* de tabel voor de *CTRLs*;
* de tabel voor de *GCPs*;
* de tabel voor de *Images*.

## Sequence diagrams

# Testplan



Een mock-databank moeten worden gemaakt om Unit en Integration tests te kunnen uitvoeren, zodat de controllers getest kunnen worden. Dit werd geprobeerd met Moq (een *mocking framework* in C#), maar dit lukte, helaas, niet.

## Web API Controllers

De Web API Controllers, op basis van een Flight ID, geven een JSON-object terug met de opgevraagde data uit de databank. Hierbij moeten drie voorwaarden gecontroleerd worden:

1) Als een vlucht niet bestaat in de databank, moet de controller HttpResponseMessage 404 teruggeven.

2) De *data projection* van een C#-klasse naar JSON moet juist gebeuren. Er moet gecontroleerd worden op de juiste waarden van de attributen van de teruggegeven data.

3) De JSON moet alle opgevraagde data bevatten.

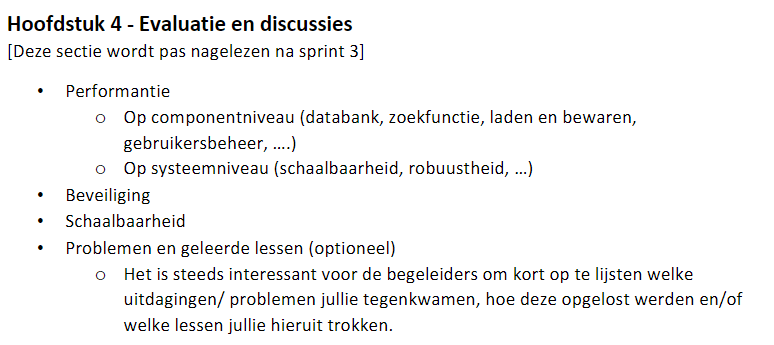
## “View” Controllers

1) Bij de gewone controllers moet de Index-methode getest worden op het teruggeven van een View met een *dependency injection*.

2) Gelijkaardige testing voor QualityReport, CTRLPoints etc. waarbij een View teruggegeven wordt met bijhorende data (een lijst of een object).

3) CRUD functionaliteit uittesten van de controllers.

# Evaluaties en discussies



# Handleidingen

## Inhoud van de distributie

In de distributie van dit project bevinden zich volgende bestanden:

* de **DroneWebApp**-map met alle code voor de webapplicatie;
* het SQL-script **DroneDB.sql** om de databank aan te maken;
* **ChangeDataSourceName\_Script.exe**, een Perl-executable;
* dit **verslag** met handleidingen en documentatie.

## Installatiehandleiding voor ontwikkelaar

De installatiehandleiding is opgedeeld in drie delen:

* Deel 1: Vereiste software
* Deel 2: Aanmaken van de databank
* Deel 3: Opstarten van de webapplicatie

## Vereiste software

In dit deel installeert u alle benodigde software om de webapplicatie te laten werken op **Windows 7 en hoger**.

1. Installeer **SQL Server 2019** op uw machine. U kan deze software [hier](https://www.microsoft.com/en-us/sql-server/sql-server-downloads) downloaden op de website van Microsoft.
   1. Scrol naar beneden en kies de versie die u verkiest (Developer of Express).
   2. Klik ‘Download now’. Het programma downloadt.
   3. Volg na het uitvoeren van het gedownloade bestand de instructies op het scherm.
2. Installeer **SQL Server Management Studio (18.4) (SSMS)** op uw machine. U kan deze software [hier](https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?redirectedfrom=MSDN&view=sql-server-ver15) downloaden.
   1. Klik op ‘Download SQL Server Management Studio (SSMS)’. Het programma downloadt.
   2. Volg na het uitvoeren van het gedownloade bestand de instructies op het scherm.
3. Installeer **Visual Studio 2019** op uw machine. U kan deze software [hier](https://visualstudio.microsoft.com/vs/) downloaden.
   1. Klik op ‘Download Visual Studio’ en kies de Community 2019 of de Professional 2019 versie naargelang uw eigen voorkeur. Het programma downloadt.
   2. Volg na het uitvoeren van het bestand de instructies op het scherm. Kies tijdens de installatie om de volgende *workloads* te installeren: ‘ASP.NET and web development*’* en ‘Data storage and processing*’*.
4. Surf naar de website van IvyTools.
   1. Op deze webpagina kunt u de gratis *personal license key* verkrijgen die u zult nodig hebben om de IvyTools software te activeren. Klik hiervoor op ‘Click here to get your free personal license key’.
   2. Kopieer deze sleutel.
   3. Navigeer in de distributie naar de map ‘drone1\IvyPdf\_1.62’.
   4. Voer het bestand **IvyTemplateEditor.exe** uit.
   5. Navigeer via de balk bovenaan het programma naar ‘Help > About > Apply License Code’.
   6. Plak de eerder gekopieerde sleutel in het veld en druk op OK.
   7. Sluit IvyTemplateEditor af.
   8. U heeft nu toegang tot de IvyParser dll-bestanden in de webapplicatie. Deze worden gebruikt bij het inlezen van een pdf.

## Aanmaken van de databank

In dit deel maakt u de SQL-Serverdatabank aan.

1. Start **SQL Server Management Studio** op en verbind met uw machine.
   1. Het veld ‘Server name’ wordt automatisch ingevuld.
   2. Noteer deze naam, want u heeft deze later nodig in een volgend deel (opstarten van de webapplicatie).
   3. Klik op ‘Connect’.
2. Ga naar het menu ‘File*’* bovenaan links.
3. Kies ‘Open’ en ga naar ‘File’.
4. Navigeer in de distributie naar het script **DroneDB.sql** en open dit.
5. Klik op ‘Execute’ om het script uit te voeren.
6. In het ‘Messages*’*-venster verschijnt “Commands completed successfully”. U kan verifiëren dat de databank is aangemaakt met volgende stappen:
   1. Klik in het ‘Object Explorer’-venster op ‘refresh*’*.
   2. Vouw de Machinenaammap en Databasesmap open. Hierin bevindt zich nu de nieuwe database **DroneDB**. Merk op dat de Machinenaammap dezelfde naam heeft als de eerder genoteerde ‘Server name’.
7. Een lege databank is nu aangemaakt en klaar voor gebruik.
8. Sluit SQL Server Management Studio.
9. In de distributie bevindt zich op het pad ‘drone1\DroneWebApp\Scripts\Perl’ een bestand genaamd **ChangeDataSourceName\_Script**. Voer dit bestand uit om de juiste *connection strings* in te vullen in **Web.config**. Deze leggen de verbinding tussen de databank en de webapplicatie.

## Opstarten van de webapplicatie (beter: deployen)

1. Voer de webapplicatie uit met **F5**.
2. De allereerste keer kan een venster verschijnen dat u vraagt om het ‘IIS Express SSL certificate’ te vertrouwen.
   1. Klik ‘yes’.
3. Er verschijnt een ‘security warning*’*.
   1. Klik ‘yes*’*.
4. U kunt nu aan de slag met de dronewebapplicatie.

## Gebruikershandleiding

Deze sectie wordt pas gepubliceerd op het einde van sprint 3.

## Problemen die zich kunnen voordoen

* De webapplicatie start op, maar hij kan niet connecteren met de databank

Er is hoogstwaarschijnlijk een probleem met de *connection string* die de brug legt tussen de webapplicatie en de databank.

1. Noteer de naam waarmee u in SQL Server Management Studio ingelogd hebt. Dit is doorgaans de computernaam.
2. In de distributie bevindt zich onder het pad ‘drone1\DroneWebApp\DroneWebApp’ een bestand **Web.config**. Open dit bestand met een editor naar keuze, zoals Visual Studio, Sublime Text of Notepad.
3. Gebruik de toetsencombinatie **CTRL + F** om te zoeken in dit bestand. Zoek naar ‘data source’. U vindt twee gevallen.
4. Vervang de *string* tussen het gelijkheidsteken en de puntkomma door uw computernaam. Dit is tevens de naam waarmee u in SQL Server Management Studio ingelogd hebt. Onderstaande afbeelding geeft u een voorbeeld van de *strings* die u moet vervangen, met als voorbeeld de computernaam ‘Predator’.



* De databank werd aangepast en nu klopt het model niet meer.

U zal de webapplicatie moeten openen in Visual Studio en volgende stappen ondernemen:

1. Start **Visual Studio**.
2. Indien er een **DroneDBModel.edmx** aanwezig is in het project, dan verwijdert u deze.
   1. Klik hiervoor rechts op ‘DroneDBModel.edmx > Delete > OK’.
3. Navigeer naar het bestand **Web.config**. Gebruik **CTRL + F** om te zoeken op ‘connectionStrings’. Verwijder de tag <add name=“DroneDBEntities” … />.
4. Sla dit bestand op.
5. Navigeer in het project naar de map ‘**Models**’.
6. Klik rechts op deze map en selecteer ‘Add > New Item’.
7. Zoek in de zoekbalk bovenaan rechts naar “model”.
8. Er verschijnt een project item genaamd **ADO.NET Entity Data Model**. Klik deze eenmalig aan om hem te selecteren.
9. Geef dit ADO.NET Entity Data Model de naam **DroneDBModel** en klik op ‘Add’.
10. Een nieuw venster verschijnt. Selecteer ‘EF Designer from database’ en klik op ‘Next’.
11. Er verschijnt een nieuw venster. Klik op ‘New Connection’.
12. Controleer of ‘Data source’ van het type ‘Microsoft SQL Server (SqlClient)’ is.
    1. Indien dit niet het geval is, klik dan op ‘Change’ en selecteer daar ‘Microsoft SQL Server (SqlClient)’.
13. Klik op ‘OK’.
14. Klik op ‘Refresh’ en wacht tot de lijst van servers opgehaald is.
15. De naam van uw computer verschijnt. Dit is dezelfde naam als de ‘Server name’ die u noteerde tijdens het aanmaken van de databank.
    1. Indien er geen naam verschijnt, dan kan u op het pijltje naar beneden klikken om uw computer uit de lijst te selecteren.
16. Indien deze lijst leeg is, geef dan uw computernaam (‘server name’) manueel in.
17. In de sectie ‘Connect to a database’, onder ‘Select or enter a database name’, selecteert u ‘**DroneDB’**. Klik op OK.
18. Controleer of ‘Save connection settings in Web.Config as’ aangevinkt is en ‘DroneDBEntities’ heet.
19. Klik op ‘Next’.
20. Selecteer ‘Tables’ in het veld ‘Which database objects do you want to include in your model?’.
21. Vink ‘Pluralize or singularize generated object names’ aan.
22. Vink ‘Include foreign key columns in the model’ aan.
23. Klik op ‘Finish’.
24. Na enige tijd zijn de modelklassen van de databank aangemaakt.

* Waarom heb ik IvyTools nodig? / Ik krijg een fout omtrent IvyTools in de webapplicatie.

IvyTools is een hulp-parser die gebruikt wordt om PDF-bestanden in te lezen. Het ingeven van een sleutel geeft u gedurende twee maanden toegang tot de dll-bestanden die in de webapplicatie aanwezig zijn.

Indien deze sleutel verloopt, dan kan u zoals in 5.2.1, punt 4, naar de website surfen en een nieuwe sleutel generen. U volgt de stappen die daar vermeldt staan om deze sleutel te activeren.

# Besluit



# Future work

In dit onderdeel worden beperkingen van de drone-planningtool belicht, alsook enkele functionaliteiten die tijdens een *future work* toegevoegd zouden kunnen worden.

* Beperkingen betreffende het uploaden van bestanden

De eerste (en onmiddellijk de grootste) beperking van de tool betreft het feit dat slechts één gebruiker per keer bestanden kan uploaden. Dit zou opgelost kunnen worden door aan iedere gebruiker een unieke id toe te kennen tijdens het uploaden, zodat de *server-side* weet welke *progress* hij naar welke gebruiker moet sturen. Op dit moment krijgen de andere gebruikers die trachten om bestanden te uploaden een melding indien er reeds een gebruiker bestanden aan het uploaden is. De andere gebruikers krijgen dan onmiddellijk de *progress bar* van de gebruiker die aan het uploaden is te zien. Op deze manier kunnen zij ongeveer inschatten wanneer zij kunnen om beurt kunnen uploaden.

Een tweede beperking betreft het feit dat de som van alle bestandsgroottes van de bestanden die een gebruikers probeert te uploaden niet groter kan zijn dan 2.147 GB (een *signed int*). Deze beperking wordt opgelegd in het bestand **Web.config**. Dit bestand laat geen grotere bestandsgroottes toe. Dit kan een probleem geven voor zeer grote XYZ-bestanden. Mogelijks moeten deze in twee stukken gehakt worden en na elkaar geüpload worden. In het geval dat de totaalsom van een reeks bestanden groter is dan 2.147 GB, dan moet de gebruiker zijn bestanden in twee of meerdere keren uploaden.

Een derde beperking betreft het feit dat de huidige implementatie (met Entity Framework 6) van de RawImageParser tijdens het aanmaken van de *thumbnails* van de afbeeldingen plots zeer veel geheugen na een tijd meer en meer geheugen gaat gebruiken indien zeer veel afbeeldingen tegelijk geüpload worden. Dit kan mogelijks te maken hebben met de *memory cap* van de programmeeromgeving (Visual Studio 2019) die een *out of memory* geeft indien meer dan 3 GB aan memory gebruikt wordt. Dit geheugengebruik is van vrij korte duur en kan dus mogelijks op een server helemaal geen probleem geven. Het geheugen wordt immers snel weer vrijgegeven na het *parsen* van de geüploade afbeeldingen. Hoe dan ook zou het interessant kunnen zijn om de RawImageParser te herschrijven zodat hij gebruikmaakt van ADO.NET (analoog aan de DATParser en XYZParser). Dit zou het *parsen* van de afbeeldingen ook sneller laten verlopen.

* Verwijderen van bestanden behorende bij een dronevlucht

In een *future work* zou het ook mogelijk moeten kunnen zijn voor een *admin* om specifieke data per dronevlucht te verwijderen, indien bijvoorbeeld een foutief bestand geüpload zou zijn. In de betreffende tabellen zou gezocht kunnen worden met de FlightId van een dronevlucht, om zo alle informatie die te maken heeft met die dronevlucht uit de databank te verwijderen. Dit zou voor de data van elk bestand moeten kunnen.

* Visualisatie
* De mogelijkheid voorzien om kleuren van *track* dynamisch te laten aanpassen naargelang de keuze van de gebruiker.
* Een *differential view* implementeren om snel een situatie met een vorige situatie te kunnen vergelijken. Dit zou een nieuw raster maken waarbij de waarde gelijk is aan het verschil tussen de vorige en de laatste meting.
* Via een KML een polygoon tekenen voor de dronevluchten (d.i. een *flight plan*).
* De grenzen (uiterste limieten) van alle afbeeldingen tonen op de kaart.
* De TIFF-afbeelding van een dronevlucht projecteren op de kaart.
* De track van een dronevlucht aanpassen naar een polyline in plaats van verscheidene, aparte punten.
* De puntenwolk visualiseren met LAS/LASZ. XYZ punten projecteren op de *map* is op dit moment niet haalbaar. Er zijn teveel individuelen punten die opgehaald moeten worden met een HTTP GET call en vervolgens getekend moeten worden op de kaart. Mogelijks kan een LAS-bestand hier soelaas brengen.

# Referenties

# Appendix A: use case-diagrammen stories

**Naam**

Dronevlucht toevoegen

**Doelstelling**

Een gebruiker voegt een dronevlucht toe aan de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De dronevlucht is opgeslagen in de databank

**Successcenario**

1. De gebruiker geeft aan dat hij een dronevlucht wil toevoegen.
2. De gebruiker voert de gegevens van de dronevlucht in.
3. De gebruiker geeft aan dat hij het toevoegen van de dronevlucht wil afronden.
4. Het systeem voegt de dronevlucht toe aan de databank.
5. Het systeem geeft een overzicht van de dronevluchten.

**Alternatieve scenario’s**

3.a. De gebruiker geeft aan dat hij het toevoegen van de dronevlucht wil afbreken, ga naar stap 5.

**Naam**

Drone toevoegen

**Doelstelling**

Een gebruiker voegt een drone toe aan de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De drone is opgeslagen in de databank.

**Successcenario**

1. De gebruiker geeft aan dat hij een drone wil toevoegen.
2. De gebruiker voert de gegevens van de drone in.
3. De gebruiker geeft aan dat hij het toevoegen van de drone wil afronden.
4. Het systeem voegt de drone toe aan de databank.
5. Het systeem geeft een overzicht van de drones.

**Alternatieve scenario’s**

3.a. De gebruiker geeft aan dat hij het toevoegen van de drone wil afbreken, ga naar stap 5.

**Naam**

Piloot toevoegen.

**Doelstelling**

Een gebruiker voegt een piloot toe aan de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De piloot is opgeslagen in de databank.

**Successcenario**

1. De gebruiker geeft aan dat hij een piloot wil toevoegen.
2. De gebruiker voert de gegevens van de piloot in.
3. De gebruiker geeft aan dat hij het toevoegen van de piloot wil afronden.
4. Het systeem voegt de piloot toe aan de databank.
5. Het systeem geeft een overzicht van de piloten.

**Alternatieve scenario’s**

3.a. De gebruiker geeft aan dat hij het toevoegen van de piloot wil afbreken, ga naar stap 5.

**Naam**

Dronevlucht verwijderen

**Doelstelling**

Een gebruiker verwijdert een dronevlucht uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De dronevlucht is verwijderd uit de databank.

**Successcenario**

1. De gebruiker duidt aan welke dronevlucht hij wil verwijderen.
2. De gebruiker bevestigt dat hij deze dronevlucht wil verwijderen.
3. Het systeem verwijdert de dronevlucht uit de databank.
4. Het systeem geeft een overzicht van de overige dronevluchten.

**Alternatieve scenario’s**

2.a. De gebruiker geeft aan dat hij het verwijderen van de dronevlucht wil afbreken, ga naar stap 4.

**Naam**

Drone verwijderen

**Doelstelling**

Een gebruiker verwijdert een drone uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De drone is verwijderd uit de databank.

**Successcenario**

1. De gebruiker duidt aan welke drone hij wil verwijderen.
2. De gebruiker bevestigt dat hij deze drone wil verwijderen.
3. Het systeem verwijdert de drone uit de databank.
4. Het systeem geeft een overzicht van de overige drones.

**Alternatieve scenario’s**

2.a. De gebruiker geeft aan dat hij het verwijderen van de drone wil afbreken, ga naar stap 4.

**Naam**

Piloot verwijderen

**Doelstelling**

Een gebruiker verwijdert een piloot uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De piloot is verwijderd uit de databank.

**Successcenario**

1. De gebruiker duidt aan welke piloot hij wil verwijderen.
2. De gebruiker bevestigt dat hij deze piloot wil verwijderen.
3. Het systeem verwijdert de piloot uit de databank.
4. Het systeem geeft een overzicht van de overige piloten.

**Alternatieve scenario’s**

2.a. De gebruiker geeft aan dat hij het verwijderen van de piloot wil afbreken, ga naar stap 4.

**Naam**

Dronevlucht wijzigen

**Doelstelling**

Een gebruiker wijzigt een dronevlucht uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De dronevlucht is gewijzigd in de databank.

**Successcenario**

1. De gebruiker duidt aan welke dronevlucht hij wil wijzigen.
2. De gebruiker wijzigt de gegevens van de dronevlucht.
3. De gebruiker geeft aan dat hij het wijzigen van de dronevlucht wil afronden.
4. Het systeem wijzigt de dronevlucht in de databank.
5. Het systeem geeft een overzicht van de dronevluchten.

**Alternatieve scenario’s**

3.a. De gebruiker geeft aan dat hij het wijzigen van de dronevlucht wil afbreken, ga naar stap 5.

**Naam**

Drone wijzigen

**Doelstelling**

Een gebruiker wijzigt een drone uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De drone is gewijzigd in de databank.

**Successcenario**

1. De gebruiker duidt aan welke drone hij wil wijzigen.
2. De gebruiker wijzigt de gegevens van de drone.
3. De gebruiker geeft aan dat hij het wijzigen van de drone wil afronden.
4. Het systeem wijzigt de drone in de databank.
5. Het systeem geeft een overzicht van de drones.

**Alternatieve scenario’s**

3.a. De gebruiker geeft aan dat hij het wijzigen van de drone wil afbreken, ga naar stap 5.

**Naam**

Piloot wijzigen

**Doelstelling**

Een gebruiker wijzigt een piloot uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Postcondities**

De piloot is gewijzigd in de databank.

**Successcenario**

1. De gebruiker duidt aan welke piloot hij wil wijzigen.
2. De gebruiker wijzigt de gegevens van de piloot.
3. De gebruiker geeft aan dat hij het wijzigen van de piloot wil afronden.
4. Het systeem wijzigt de piloot in de databank.
5. Het systeem geeft een overzicht van de piloten.

**Alternatieve scenario’s**

3.a. De gebruiker geeft aan dat hij het wijzigen van de piloot wil afbreken, ga naar stap 5.

**Naam**

Details dronevlucht bekijken

**Doelstelling**

Een gebruiker bekijkt de details van een dronevlucht uit de webapplicatie.

**Actor**

Gebruiker

**Successcenario**

1. De gebruiker duidt aan welke dronevlucht hij wil bekijken.
2. Het systeem haalt de dronevlucht op uit de databank.
3. Het systeem toont de details van de dronevlucht.
4. De gebruiker geeft aan dat hij het bekijken van de details wil afronden.
5. Het systeem geeft een overzicht van de overige dronevluchten.

**Naam**

Details drone bekijken

**Doelstelling**

Een gebruiker bekijkt de details van een drone uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Successcenario**

1. De gebruiker duidt aan welke drone hij wil bekijken.
2. Het systeem haalt de drone op uit de databank.
3. Het systeem toont de details van de drone.
4. De gebruiker geeft aan dat hij het bekijken van de details wil afronden.
5. Het systeem geeft een overzicht van de overige drones.

**Naam**

Details piloot bekijken

**Doelstelling**

Een gebruiker bekijkt de details van een piloot uit de webapplicatie.

**Actor**

Gebruiker

**Precondities**

De gebruiker is ingelogd.

**Successcenario**

1. De gebruiker duidt aan welke piloot hij wil bekijken.
2. Het systeem haalt de piloot op uit de databank.
3. Het systeem toont de details van de piloot.
4. De gebruiker geeft aan dat hij het bekijken van de details wil afronden.
5. Het systeem geeft een overzicht van de overige piloten.

1. Makkelijk, vrij gemakkelijk, gemiddeld, vrij moeilijk, moeilijk [↑](#footnote-ref-1)
2. Onbelangrijk, redelijk belangrijk, belangrijk, zeer belangrijk [↑](#footnote-ref-2)