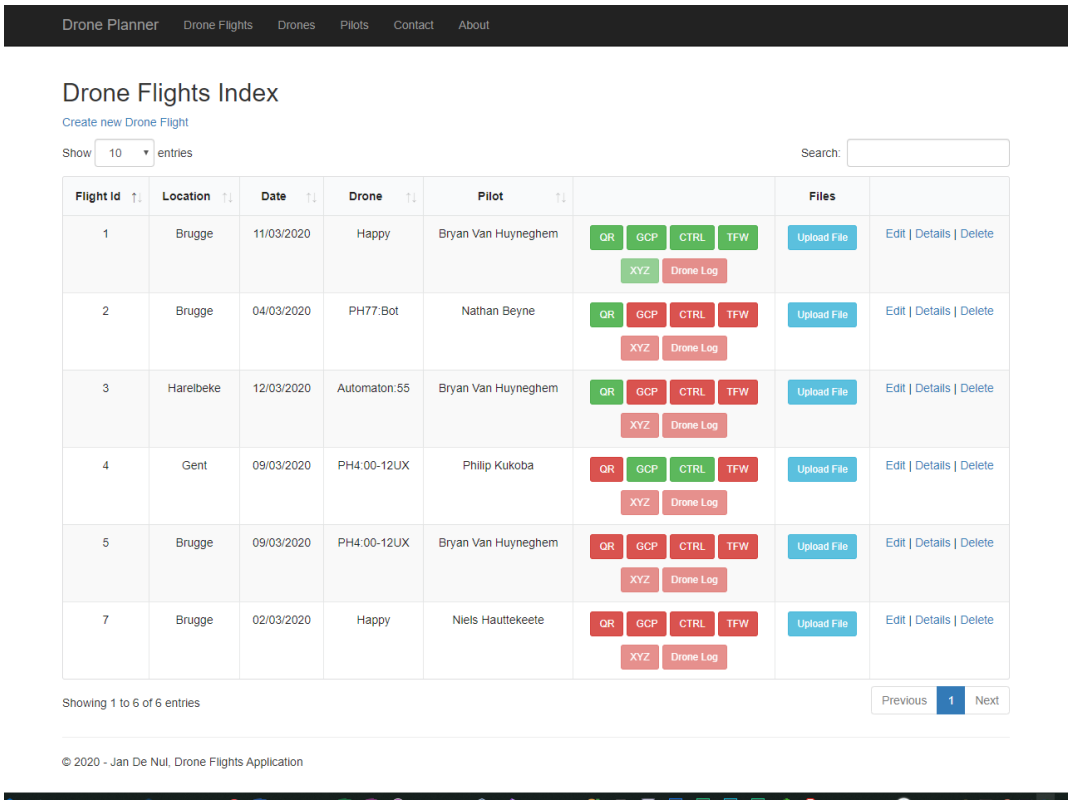
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Droneplanning-tool

Sprint 1: analyseverslag



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

Bryan Van Huyneghem
Philip Kukoba
Nathan Beyne
Niels Hauttekeete

Promotoren: Prof. Helga Naessens, Prof. dr. 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

UNIVERSITEIT
GENT

Inhoudsopgave

Lijst van figuren

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 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 aan 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 classes* 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.

(3) 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.

(4) 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.

(5) 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.

1. 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.

1.1 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.

1.2 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.

1.3 Use case-diagrammen

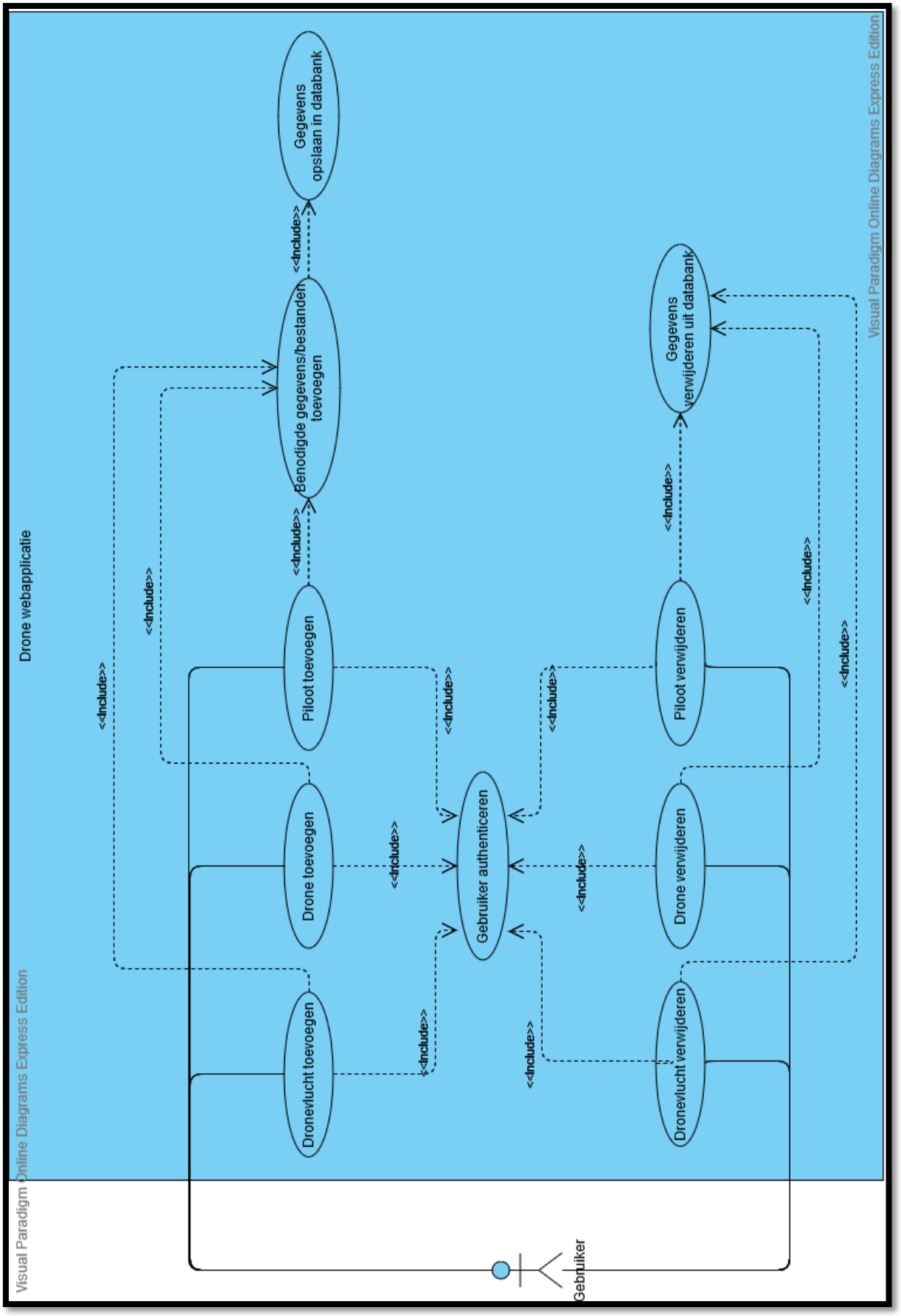
Het gebruik van de webapplicatie wordt verduidelijkt in **Figuren 1 en 2** op pagina's 10 en 11 via twee use case-diagrammen. Hierop is te zien hoe de actor (de gebruiker) kan interageren met de verschillende functionaliteiten van het systeem. Een gebruiker beschikt over CRUD (*Create, Read, Update,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

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

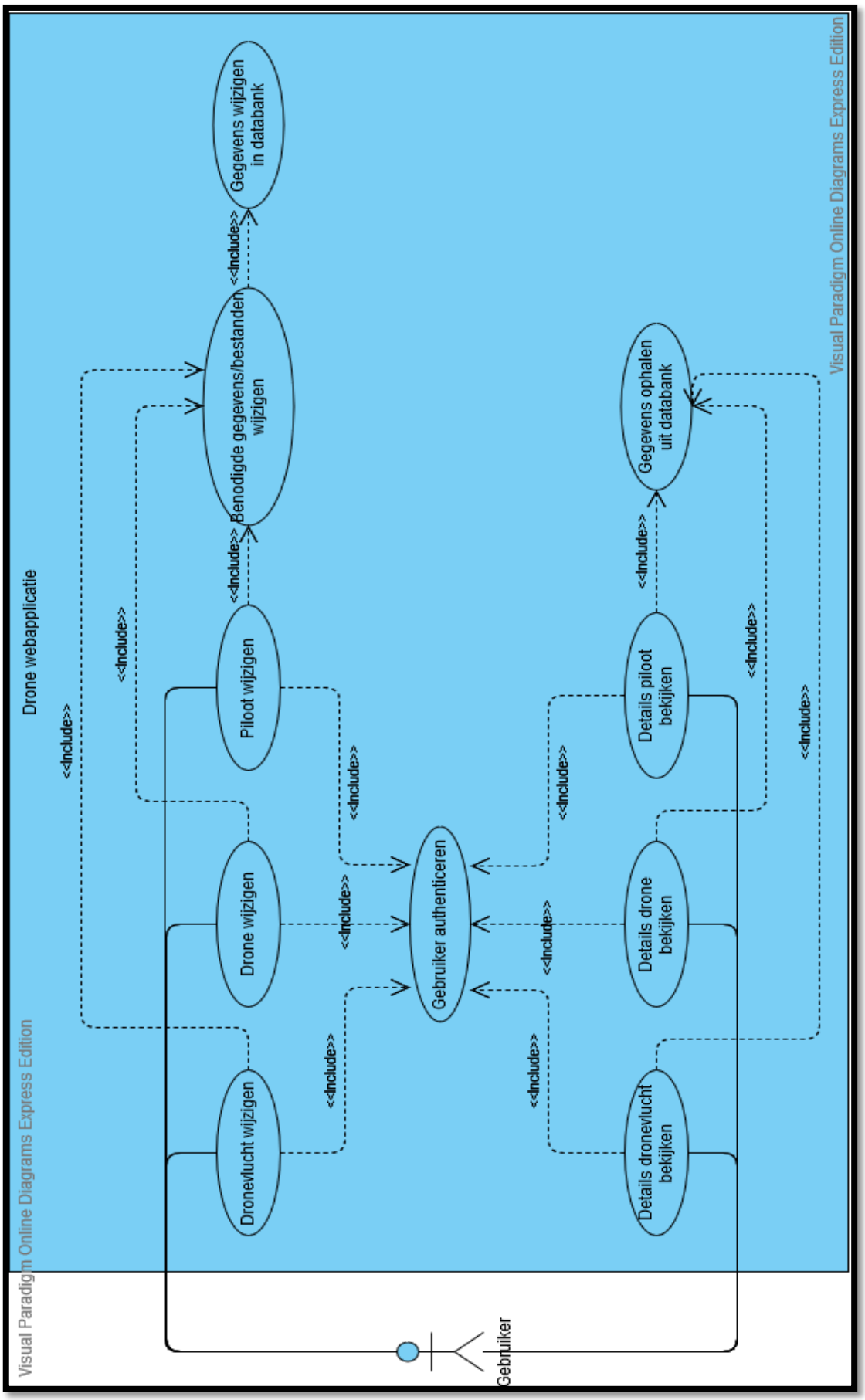
Een ingelogde gebruiker kan ook een dronevlucht, drone of piloot verwijderen uit het systeem. Verder kan een gebruiker de bijhorende en beschrijvende informatie van dronevluchten, drones of piloten wijzigen. Deze aanpassingen hebben onmiddellijk effect op de databank aan server-sidekant.

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.

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

1.4 Featurelijst

Hieronder volgt een opsomming van de alle features in dit project, inclusief hun complexiteit en prioriteit, per sprint.

1.4.1 Sprint 1

(to-do)

1.4.2 Sprint 2

1.4.3 Sprint 3

2. Systeemarchitectuur

In dit hoofdstuk wordt besproken hoe het programma doelstellingen 1, 2, 3 en 4 realiseert en hoe deze opgebouwd zijn. Aan de hand van een databankdiagram en een klassendiagram voor de *parser classes* zal de opbouw van de webapplicatie beschreven worden en getoond worden op welke manier de verschillende onderdelen samenwerken.

2.1 High-levelsysteemmodel

Deployment-diagram

2.2 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 15, 17 en 18.

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`.

2.2.1 DroneFlight tabel

De eerste en tevens ook belangrijkste tabel in het databankdiagram is de `DroneFlight` tabel, te zien op **Figuur 3** op pagina 15.

Figuur 3: de DroneFlight tabel 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 twee *Foreign Keys*: een `DroneId` en een `PilotId`. Op deze manier kunnen drones en piloten informatie opvragen over hun vluchten. Merk op dat een drone en een pilot met een dronevlucht een één-op-veel relatie beschrijven. Een drone en een pilot kunnen immers meerdere dronevluchten uitvoeren.

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.

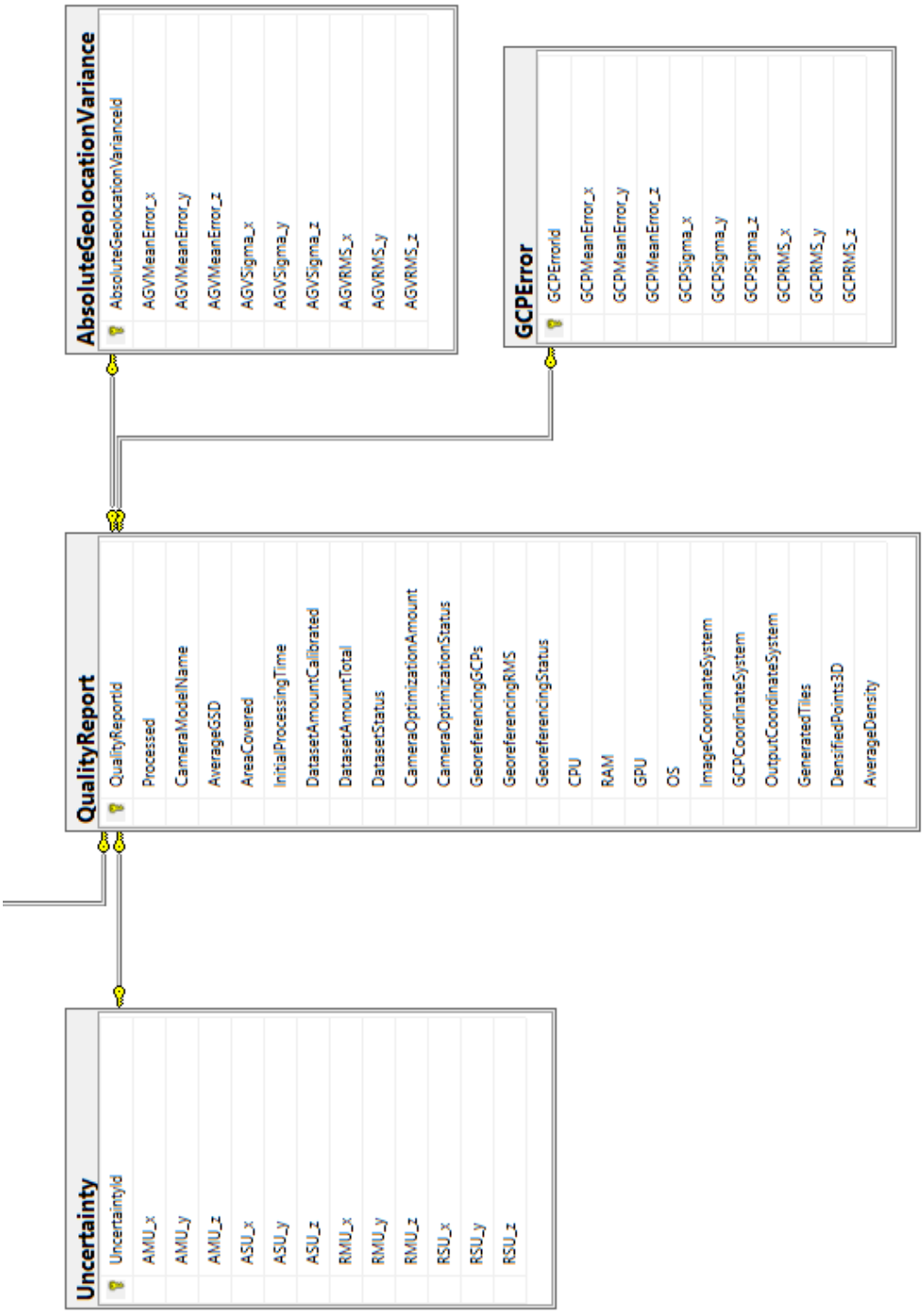
De tabel `TFW` bevat rotatie- en translatie informatie, alsook wereldcoördinaten die gebruikt worden bij een *tiff image* bestand in een GIS applicatie. Deze informatie laat toe een andere, externe databank aan te spreken en de juiste resultaten te verkrijgen in de vorm van een kaart.

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

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

2.2.2 QualityReport tabel

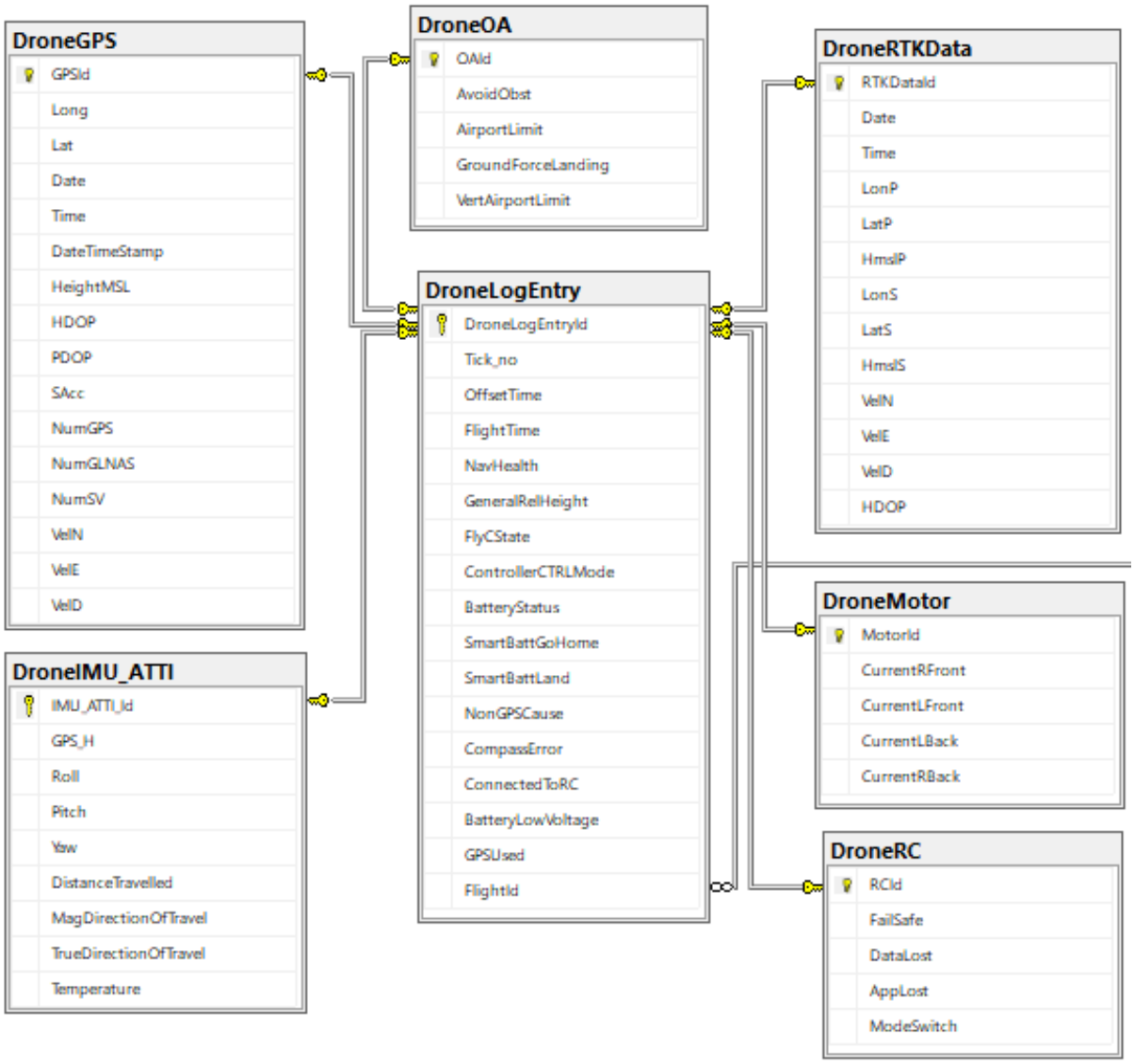
In de `QualityReport` tabel in **Figuur 4** wordt alle data bijgehouden die ingelezen wordt uit het kwaliteitsrapport. Dit is een pdf-bestand dat de output beschrijft van een analyse in het programma Pix4D. Tabellen die een relationeel verbonden zijn met deze tabel zijn: `Uncertainty`, `AbsoluteGeolocationVariance` en `GCPError`. Op deze manier is de ingelezen informatie uit het kwaliteitsrapport op een logische manier gegroepeerd.



Figuur 4: de QualityReport tabel met al haar relaties

2.2.3 DroneLogEntry tabel

In **Figuur 5** staat de `DroneLogEntry` tabel 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 `DroneFlight` tabel, aangezien elke dronevlucht meerdere *dronelog entries* heeft.



Figuur 5: de DroneLogEntry tabel met al haar relaties

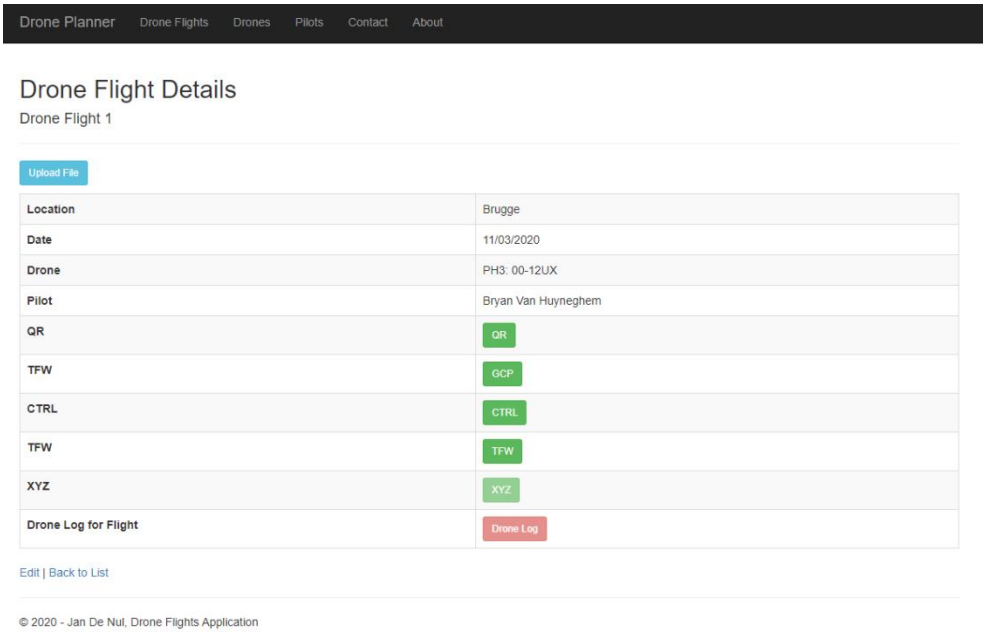
2.3 MVC

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.

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.

Vervolgens worden enkele *Controllers* (de C in het MVC pattern) aangemaakt die instaan voor navigatie (routing): de `DroneFlightsController`, de `DronesController`, de `PilotsController`, de `FilesController` en de `HomeController`. De drie 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).

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 6** 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 6: De detailpagina van een Drone Flight

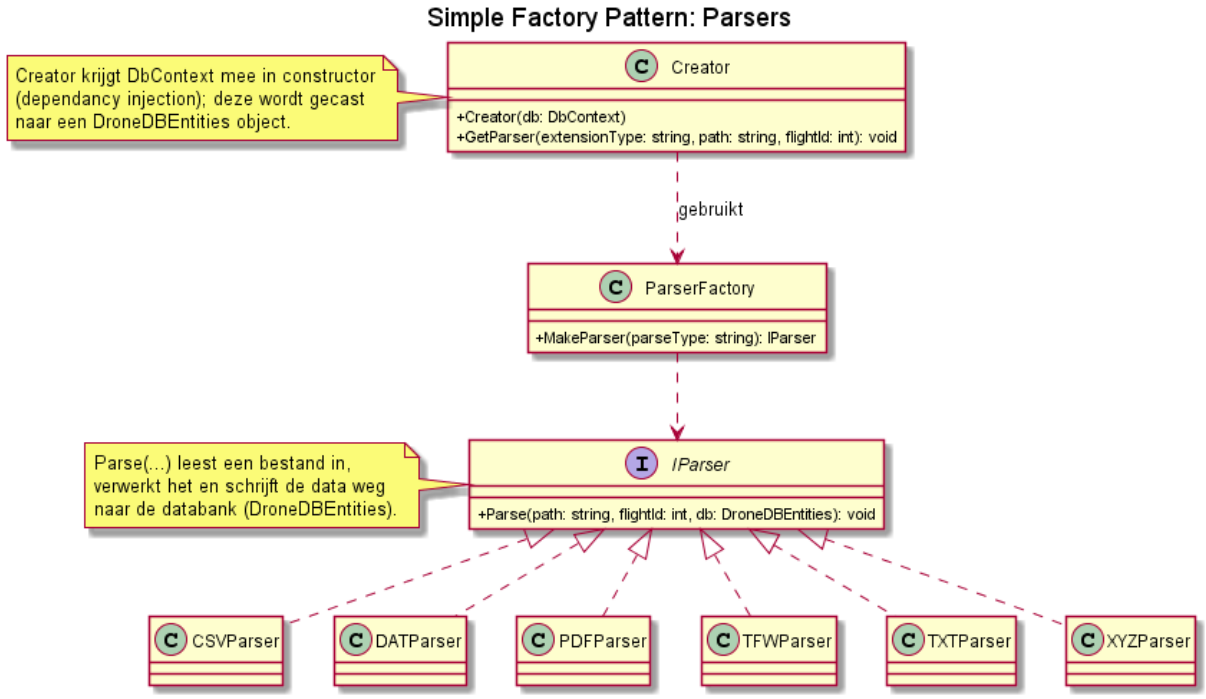
2.4 Klassendiagrammen

In dit onderdeel worden de klassendiagrammen beschreven die gebruikt werden in deze applicatie: een simple factory pattern voor het parsen van de bestanden, alsook javascriptklassen voor het visualiseren van data.

2.4.1 Simple Factory pattern voor parserklassen

Het parsen van data uit bestanden gebeurt via parserklassen. 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 parserklasse aanmaakt en teruggeeft aan de `Creator`. Daarna vindt het parsen plaats met de methode `Parse()` van de aangemaakte parserklasse.

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.



Figuur 7: het simple factory pattern voor parsers

2.4.2 Javascriptklassen

(Sprint 2)

Referenties

ArcGIS for Developers. (z.j). Geraadpleegd op 13 februari 2020 via https://developers.arcgis.com/labs/

ASP.NET. (z.j). Geraadpleegd op 15 februari 2020 via https://dotnet.microsoft.com/apps/aspnet

Understanding world files. (z.j.). Geraadpleegd op 13 februari 2020 via http://webhelp.esri.com/arcims/9.3/General/topics/author_world_files.htm

Appendix A: use case-diagrammen stories