Visual Deployment Pipeline

Afstudeerverslag

M.R. de Meza, 10043659

Info Support
Kruisboog 42
3905 TG Veenendaal
Tel. +31(0)318 - 55 20 20
Fax +31(0)318 - 55 23 55

De Haagse Hogeschool
Johanna Westerdijkplein 75
2521 EN Den Haag
Tel. +31(0)704 - 45 88 88
Fax +31(0)704 - 45 88 25

Eerste examinator
G.A. Mijnarends

Tweede examinator
J.J. van der Hoek

Opdrachtgever
P. Borgeld

Technisch begeleider
F. Sedney

Proces begeleider
L. van de Hoef

Datum
03-06-2016

Referaat

De Meza, Marc

Analyseren, ontwerpen, realiseren en kwaliteitsbewaking van een dashboard dat gebruikt kan worden om visueel inzicht te krijgen in een deploymentstraat van Microsoft Release Management(MRM) 2015.

Afstudeerverslag van Marc de Meza, geschreven in het kader van afstuderen bij de opleiding Informatica aan de academie voor ICT & Media aan De Haagse Hogeschool.

Het verslag behandelt het ontwikkelproces dat is doorlopen tijdens het afstuderen van Marc de Meza bij Info Support BV.

Descriptoren

- Scrum
- C#
- Microsoft Release Management
- Dashboard
- Visual deployment pipeline
- .Net
- Webapi
- AngularJS
- Test Driven Development
- SpecFlow
- Protractor
- MongoDB
- HTML5
- CSS3

Voorwoord

Voor u ligt het afstudeerverslag van Marc de Meza wat geschreven is voor de opleiding Informatica aan De Haagse Hogeschool. In dit verslag geef ik inzicht in de werkzaamheden die ik heb uitgevoerd tijdens het afstuderen bij Info Support BV van 01 februari 2016 tot 03 juni 2016.

Graag wil ik het bedrijf Info Support bedanken voor de mogelijkheid die mij is aangeboden om bij hun af te studeren. Bij Info Support zijn er vier medewerkers die ik persoonlijk wil bedanken voor de begeleiding die ik van hun heb gekregen. Deze medewerkers zijn L. van de Hoef, P. Borgeld, F. Sedney en G. Timmerman.

Verder wil ik G..A Mijnarends en J.J. van der Hoek, mijn examinatoren van De Haagse Hogeschool, bedanken voor de feedback die zij hebben gegeven tijdens mijn afstuderen.

Marc de Meza
Leiden, 01 juni 2016

Inhoud

1. Inleiding

Microsoft Release Management(MRM) is een tool van Microsoft dat gemaakt is om applicaties op verschillende omgevingen geautomatiseerd te installeren. Tijdens de installatie voert MRM verschillende acties uit op de desbetreffende omgeving. Er kunnen verschillende acties worden uitgevoerd, zoals het aanpassen van een database of het herstarten van een webserver. In MRM is het mogelijk om verschillende gebruikers rechten te geven om deze acties te verifiëren. Deze gebruikers hebben toestemming om de installatie op bepaalde punten te stoppen of door te laten gaan. In de 2015 update 1 versie van de applicatie zijn er enkele tekortkomingen in het inzicht krijgen van het gehele installatieproces.

Het doel van dit verslag is om duidelijk te maken hoe mijn afstudeeropdracht bij Info Support BV tot stand is gekomen. Info Support maakt gebruik van MRM om applicaties automatisch te installeren. Aangezien Info Support gebruik maakt van de 2015 update 1 versie van MRM is het momenteel niet mogelijk om duidelijke te zien hoe het installatieproces is verlopen. Gedurende mijn afstuderen ga ik op zoek naar de mogelijkheden om gegevens uit MRM te verkrijgen.

Nadat ik de mogelijkheden heb afgewogen en een keuze heb gemaakt zal ik een dashboard realiseren waarin de verloop van het installatieproces duidelijk weergegeven zal worden. Met dit dashboard zal Info Support duidelijk kunnen zien hoe de installaties van hun applicaties verlopen.

In dit verslag zal ik beginnen met een beschrijving van het bedrijf waar ik mijn opdracht heb uitgevoerd. Hierna zal ik toelichten hoe mijn opdracht tot stand is gekomen. Vervolgens ga ik verder met het behandelen van de aanpak van mijn opdracht. Hierin zal duidelijk worden welke ontwikkelmethodiek is gekozen ter aanvulling van mijn opdracht. Verder zal ik mijn werkzaamheden, per sprint, beschrijven om duidelijk te maken welke beslissingen ik heb moeten nemen tijdens mijn opdracht. Als laatste geef ik nog een evaluatie over de beroepstaken die ik voor de opdracht heb vastgelegd. In mijn evaluatie zal duidelijk worden hoe ik heb voldaan aan deze beroepstaken.

2. Context en opdracht

In dit hoofdstuk zal ik het bedrijf beschrijven om de context van de opdracht te verduidelijken. In dit hoofdstuk kan de lezer terug zien hoe de organisatie, waar de opdracht voor is uitgevoerd, precies in elkaar zit. Verder zal ik ook mijn werkomgeving en opdracht beschrijven.

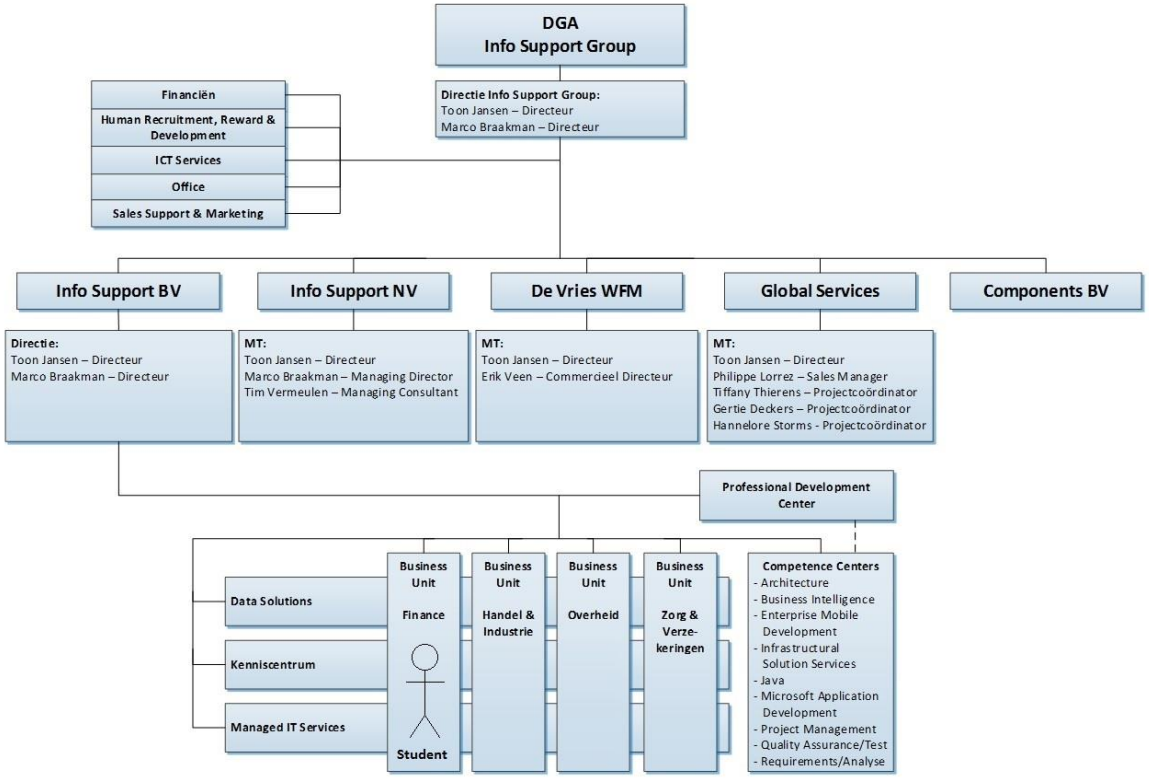
2.1 Organisatieomschrijving

Info Support is in 1986 opgericht door huidige directeur Toon Jansen. Toon Jansen had de visie dat de “personal computer” een grote verschuiving zou beleven in het IT-landschap. Info Support had daarom in het begin veel focus gelegd op de Client/Server Architectuur.

Het bedrijf is sinds die tijd uitgegroeid tot een volwaardig IT-dienstverlener met meer dan 400 medewerkers. Info Support richt zich nu vooral op het ontwikkelen van software op maat, grote administratieve bedrijfsapplicaties en integratie- en business intelligence projecten. De opdrachten die er worden uitgevoerd, zijn voornamelijk gemaakt met Microsoft Technologieën waaronder C#, SQL Server, ASP.NET en Xamarin. Naast kennis van Microsoft Technologieën heeft het bedrijf ook ruim expertise in Oracle en Java.

Info Support hecht veel waarde aan het opleiden van haar medewerkers. Daarom is er een kenniscentrum opgericht. Het kenniscentrum biedt alle medewerkers toegang tot opleidingen die onder andere nodig zijn bij het uitvoeren van hun werkzaamheden. Tijdens mijn afstuderen heb ik ook gebruik kunnen maken van deze opleidingsmogelijkheden.

Aan de top van Info Support staat het Info Support Group. In figuur 1 is weergegeven welke bedrijven onder Info Support Group vallen. Verder is mijn positie als afstudeerder binnen Info Support ook te zien in de figuur.



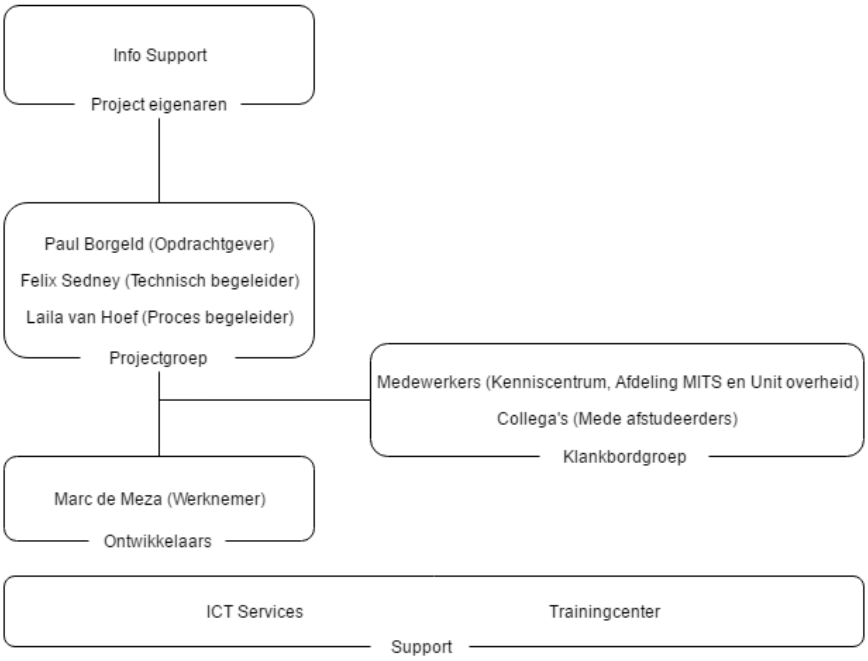
Figuur 1 organogram Info Support[RF06]

De 400 medewerkers van Info Support zijn verdeeld over de bovenstaande bedrijven. Als afstudeerder in Nederland is mijn positie weergegeven onder Info Support BV. Info Support BV is verdeeld in vier business units. Deze units bestaan uit Finance, Handel & Industrie, Overheid en Zorg & Verzekeringen. Alle afstudeerders zijn verdeeld over deze units. Ik zat tijdens het afstuderen in de unit Finance.

Tijdens het afstuderen maak ik niet alleen deel uit van unit Finance, maar ook van een projectgroep waarbinnen mijn opdracht uitgevoerd moet worden. In figuur 2 is er een overzicht weergegeven van de volledige groep waar ik mee te maken heb tijdens het project.

Project eigenaar
Het project eigenaar is in dit geval Info Support BV.

Projectgroep
De projectgroep bestaat uit mijn begeleiders en opdrachtgever binnen Info Support. Deze begeleiders hou ik op de hoogte van mijn voortgang zodat dit te controleren is.



Figuur 2 contactpunten van student

Klankbordgroep
Deze groep bestaat uit de medewerkers met wie ik op kantoor zit. Deze groep ondersteunt mij tijdens de opdracht door vragen te beantwoorden en mee te denken over problemen.

Ontwikkelaars
Mijn functie binnen het project is ontwikkelaar. Het team van ontwikkelaars bestaat alleen uit mezelf. Ik ben verantwoordelijk voor het ontwerpen, realiseren en testen van mijn opdracht. Verder ben ik verantwoordelijk voor de communicatie richting alle groepen over mijn voortgang.

Support
ICT Services faciliteert mij tijdens de opdracht en het Kenniscentrum biedt opleidingen aan voor onderwerpen waar ik nog niet genoeg kennis van heb.

2.2 Werkomgeving

Tijdens mijn afstuderen bij Info Support had ik beschikking over verschillende middelen in mijn werkomgeving. Deze middelen hebben het mogelijk gemaakt voor mij om mijn afstudeeropdracht uit te kunnen voeren.

Werkplek
In Info Support worden afstudeerders in een kantoor geplaatst met andere afstudeerders en medewerkers. Ik zat bij de afdeling SQL-Beheer. Gedurende mijn afstuderen heb ik ook kunnen meekijken met de werkzaamheden van deze afdeling.

Verder had ik op mijn kantoor beschikking over een werkplek. Op deze werkplek had ik toegang tot een computer die ik mocht gebruiken bij het uitvoeren van mijn opdracht. Op deze computer was ik administrator en had ik daarom alle rechten om applicaties te installeren. Verder was het mogelijk om van een grote selectie softwarepakketten gebruik te maken. Indien ik nog aanvullende software nodig had voor mijn opdracht kon dit ook geregeld worden. Het inrichten van mijn werkplek had ik helemaal zelf in handen.

Tools
Zoals in hoofdstuk 1.1 is vermeld maakt Info Support voornamelijk gebruik van Microsoft technologieën. Hierdoor heb ik gedurende mijn afstuderen vaak Microsoft applicaties, talen en frameworks gebruikt. Voor versiebeheer heb ik gebruik gemaakt van Team Foundation Server(TFS) wat ook een bekende applicatie is van Microsoft.

Bij Info Support wordt er voornamelijk gewerkt met Scrum. Dit valt ook op binnen het bedrijf, omdat er overal in het kantoor Scrumborden hangen met de voortgang van een project. In het bedrijf loop je ook vaak toevallig langs een “daily standup”. Voor mijn opdracht bestond de mogelijkheid om zelf een ontwikkelmethodiek te kiezen, mits deze voldeed aan een aantal eisen. Dit wordt in het hoofdstuk Aanpak verder beschreven.

OTAP-omgevingen
Tijdens mijn afstuderen had ik beschikking over een ontwikkelomgeving. Mijn ontwikkelomgeving was gelijk ook mijn test omgeving. Hiermee bestond de mogelijkheid om mijn project gelijk te testen. Zodra mijn opdracht voldoende functionaliteit bevatte kwam er een echte test en acceptatie omgeving. Een productie omgeving was er niet tijdens mijn opdracht. De bedoeling was dat bij het opleveren van mijn product deze in productie genomen zou worden.

2.3 Opdracht omschrijving

'DevOps' is een term dat steeds vaker gebruikt wordt de afgelopen jaren. Deze term gaat voornamelijk over het samenwerken en voortdurende communicatie tussen software ontwikkelaars en software beheerders. Met deze samenwerking is het mogelijk om software en infrastructuren sneller op te leveren naar de klant[RF11]. Een belangrijk aspect van 'DevOps' is 'continuous integration and delivery'. 'Continuous integration and delivery' is een belangrijke term bij het werken met Agile projecten. Als er meerdere teams werken op een deel van een systeem worden er elke keer kleine stukjes van het systeem opgeleverd, met elkaar getest en in productie genomen. Indien de nieuwe opleveringen niet door de tests komen wordt de applicatie niet in productie genomen.

Voordat de nieuwe versie van het systeem uitgerold kan worden naar de productie omgeving is het nodig dat het op verschillende omgevingen getest wordt. Het uitrollen van nieuwe versies kan lang duren als het niet geautomatiseerd is. Voor het automatiseren van dit proces biedt Microsoft een tool aan, namelijk MRM 2015.

Aanleiding
Sinds TFS 2013 heeft Microsoft het mogelijk gemaakt om geautomatiseerd applicaties te kunnen installeren door gebruik te maken van MRM. In MRM is het mogelijk om een deploymentstraat te definiëren. Een deploymentstraat is een soort template waarin is aangegeven hoe een applicatie op verschillende omgevingen geïnstalleerd moet worden. In de deploymentstraat is het mogelijk om verschillende stappen te definiëren die uitgevoerd moeten worden tijdens het installeren. In MRM is het mogelijk om tijdens en na de installatie te zien of er fouten zijn opgetreden.

Probleemstelling
MRM heeft momenteel geen mogelijkheid om in één oogopslag te zien welke versies van applicaties op een bepaalde omgeving zijn geïnstalleerd. Verder is het niet gamakkelijk te zien welke stappen zijn misgegaan tijdens het uitrollen.

Momenteel is er een eenvoudig dashboard gemaakt door Info Support waarin een deel van deze informatie wordt weergegeven. Dit dashboard geeft nog lang niet alle gewenste informatie weer. Als er meer informatie nodig is moet een gebruiker in MRM hiernaar op zoek gaan.

Doelstelling
Er moet een analyse uitgevoerd worden naar de mogelijkheden om op basis van gegevens uit MRM een dashboard te realiseren die een deployment pipeline visualiseert. Verder moet er ook een applicatie gerealiseerd worden waarin deze visualisatie getoond worden. In de visualisatie van de deployment pipeline moet er per applicatie te zien zijn op welke omgevingen het is geïnstalleerd. Verder moet er in dit overzicht ook gegevens zoals versie nummers en eventueel fouten tijdens installatie getoond worden. De gegevens moeten niet alleen per applicatie getoond worden, maar ook per omgeving.

Resultaat
Als de opdracht is uitgevoerd zal het bedrijf beschikken over een dashboard waarop diverse overzichten weergegeven worden. De informatie van de deployment pipeline, waar normaal gesproken op verschillende locaties in MRM naar gezocht moet worden, zal in één overzicht weergegeven worden. Het dashboard zal, in de vorm van een webapplicatie, overal binnen het domein te benaderen zijn. Het dashboard zal weergegeven worden op pc schermen en iPad's .

Effect
De opdracht zal ervoor zorgen dat het bedrijf tijd bespaart bij het vinden van informatie over releases. Ten slotte komt er ook meer inzicht in het deployment proces. Dit zal ervoor zorgen dat fouten eerder worden opgespoord.

2.4 Microsoft Release Management

Om mijn afstudeerverslag te kunnen begrijpen is het belangrijk dat MRM uiteen wordt gezet. Zoals in paragraaf 2.3 is beschreven wordt MRM gebruikt voor het automatiseren van 'continuous integration and delivery'. In MRM is het mogelijk om een volledige deploymentstraat in te voeren. In deze paragraaf wordt er kort tijd besteed aan alle stappen hiervoor en wat ze precies inhouden.

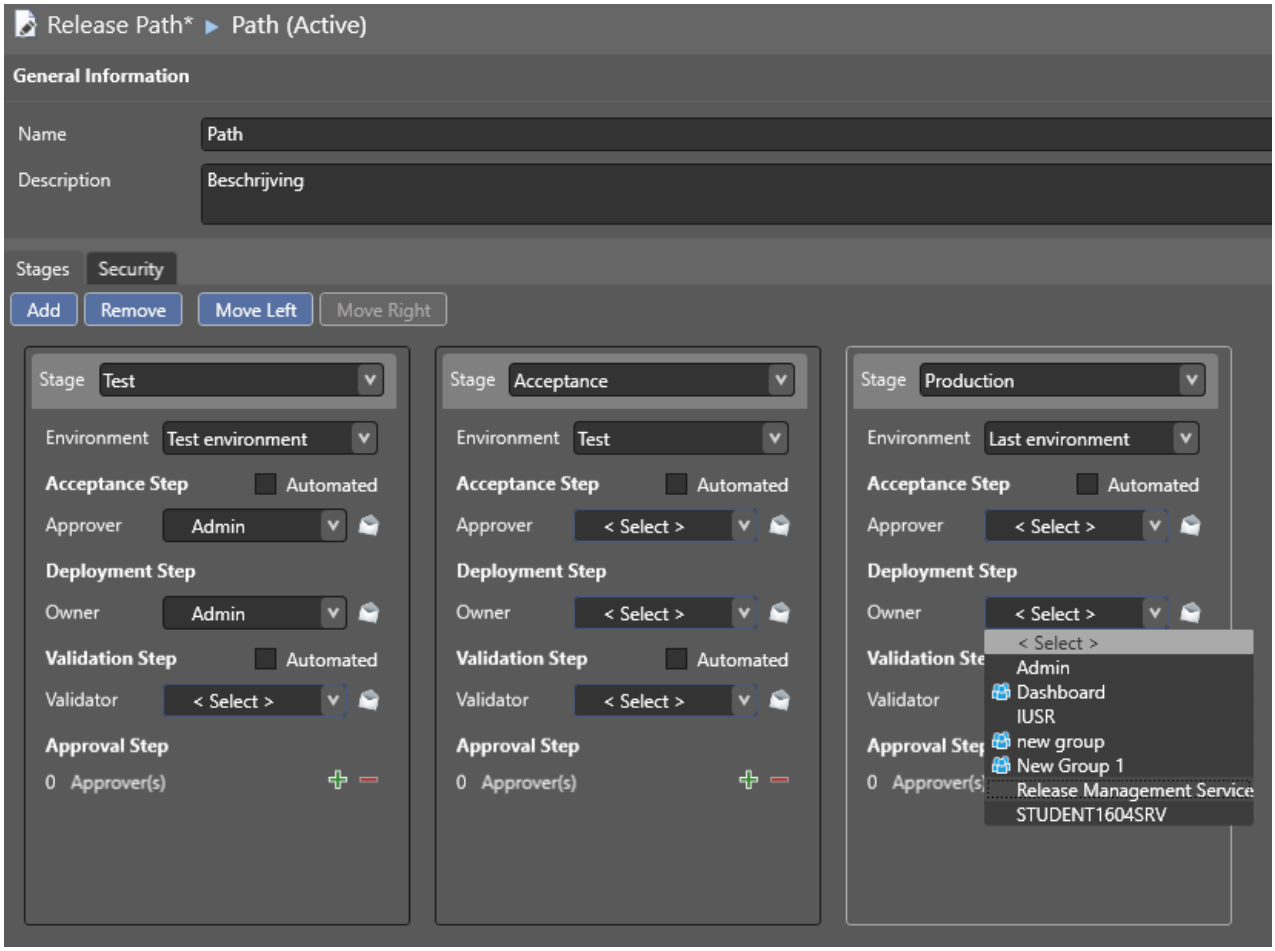
Een deploymentstraat begint bij het bepalen welke applicatie gebruikt zal worden. Voor deze applicatie moet bepaald worden waar die geïnstalleerd gaat worden en welke stappen daarbij moeten worden uitgevoerd. Applicaties kunnen namelijk op verschillende omgevingen geïnstalleerd worden. Nadat dit is bepaald, is het mogelijk om de deploymentstraat in MRM te configureren. In MRM is een deploymentstraat te zien als een ReleasePath. Het is gebruikelijk om per applicatie een aparte ReleasePath te maken.

De source code van de applicatie kan uit de GIT van TFS gehaald worden. Dit is mogelijk omdat er in MRM verschillende TFS projecten gekoppeld kunnen worden waarin de code voor projecten staan. In MRM is het mogelijk om twee verschillende ReleasePaths aan te maken. Deze ReleasePaths bestaan uit AgentBasedPaths en een vNext paths. Het verschil tussen deze twee Paths is dat de AgentBasedPaths worden geïnstalleerd door gebruik te maken van Deployment Agents. Deployment Agents zijn applicaties die geïnstalleerd worden waarmee MRM verbinding kan maken. Als er een verbinding is gemaakt kan MRM door de Deployment Agent installaties uitvoeren. vNext Paths maken gebruik van Powershell scripts. Powershell is een command line en script tool van Microsoft. Met Poweshell is het mogelijk om bepaalde scripts te maken met instructies over hoe de installatie moet plaats vinden. MRM maakt vervolgens gebruik van deze scripts bij het installeren. Bij het maken van een nieuwe ReleasePath kan de gebruiker een naam en een beschrijving invoeren. Naast het toevoegen van deze gegevens is het ook mogelijk om een aantal Stages te koppelen aan de Path. Het koppelen van de Stages is te zien in figuur 3. Wat een Stage precies inhoudt wordt later duidelijk.

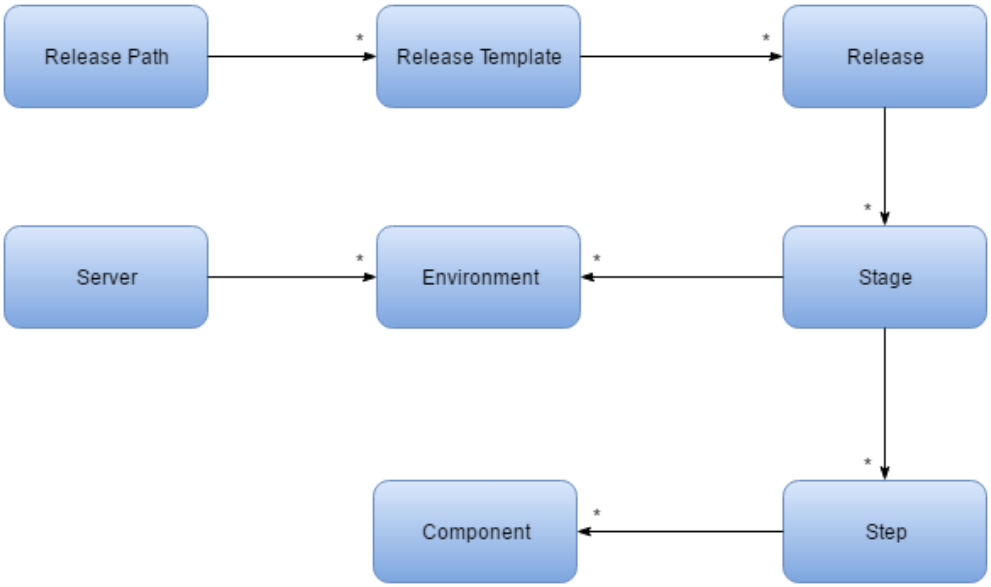
Nadat een ReleasePath is aangemaakt komen we bij de volgende stap, namelijk het maken van een ReleaseTemplate. Een ReleaseTemplate is een template van een ReleasePath die gebruikt wordt bij het uitvoeren van een Release. Alles wat is beschreven in je ReleasePath wordt gekopieerd naar de Template. De Stages die zijn aangegeven bij de ReleaseTemplate worden gebruikt. Nadat de Template is aangemaakt is het mogelijk Steps per omgeving te definiëren. Wat een Step precies inhoudt wordt later duidelijk. Templates zijn ook verdeeld in AgentBased en vNext.

Zoals eerder vermeld is het mogelijk om Stages te definiëren en te gebruiken bij het maken van een ReleasePath. Een stage is een fase waarin de deploymentstraat zich kan bevinden. Voorbeelden hiervan zijn acceptatie, ontwikkeling of productie. Per Stage is ook aan te geven wie moet controleren of het uitrollen naar deze Stage goed is verlopen. In alle Stages is het mogelijk om een aantal Steps aan te geven. Deze steps zijn acties die uitgevoerd worden tijdens het uitrollen in een bepaalde Stage. Er wordt ook bijgehouden of Steps succesvol worden uitgevoerd of niet. Bij het niet succesvol afronden van een Step is het uitrollen mislukt. Dit betekent dat er geen andere stappen uitgevoerd worden en de applicatie ook niet op de volgende omgevingen wordt geïnstalleerd. Tijdens elke Step is het ook mogelijk voor een gebruiker om aan te geven of het naar de volgende Stap mag.

Bij het instellen van een Stage in een ReleaseTemplate is het mogelijk om een aantal Components toe te voegen. Deze Components zijn applicaties die eventueel gekoppeld worden aan een Stage.



Figuur 3 toevoegen van Stages in een ReleasePath



Figuur 4 klasse diagram van MRM zonder attributen

De laatste twee onderdelen waar wij naar gaan kijken zijn de Servers en de Environments. Servers zijn fysieke machines waarop applicaties geïnstalleerd kunnen worden. Tijdens het uitvoeren van een Release worden de Servers dus werkelijk gebruikt. Servers hebben ook een collectie van Environments. Als een bepaalde applicatie geïnstalleerd wordt op bijvoorbeeld Environment 1 kan er gezien worden op welke Server dit komt te staan. Een Environment is een soort virtuele omgeving waar een Release op wordt uitgevoerd. Bij het aanmaken van een Environment kan je naast een naam en beschrijving ook aangeven welke technologieën er in die omgeving aanwezig zijn. In figuur 4 is een opsomming te zien van alle onderdelen waarmee je te maken hebt tijdens het configureren van een deploymentstraat in MRM.

2.4.1 Resultaat

Nadat de hele deploymentstraat is ingesteld is het mogelijk deze te gebruiken voor ‘continuous integration and delivery’. Elke keer dat de applicatie in de GIT op TFS wijzigt kan een gebruiker een Release laten uitvoeren met de ReleasePath die voor die applicatie is ingesteld. De applicatie gaat langs alle omgevingen in de Path en alle stappen zoals databases vullen en tests uitvoeren worden uitgevoerd. Indien de Path succesvol wordt doorlopen is de gewijzigde applicatie in productie genomen. Dit kan bij de kleinste wijzigingen uitgevoerd worden en is volledig automatisch.

3. Aanpak

In dit hoofdstuk beschrijf ik mijn aanpak voor dit project. Eerst beschrijf ik welke onderdelen in mijn plan van aanpak terecht zijn gekomen. Vervolgens beschrijf ik hoe ik van plan was mijn taken uit te voeren.

3.1 Plan van aanpak

De eerste twee weken hield ik mijzelf bezig met het maken van mijn plan van aanpak. In mijn plan van aanpak heb ik verschillende aspecten over het project beschreven. Het gehele plan van aanpak is te raadplegen in bijlage I. In deze paragraaf beschrijf ik de belangrijkste punten van het plan van aanpak.

3.1.1 Afbakening

De focus van de opdracht is:

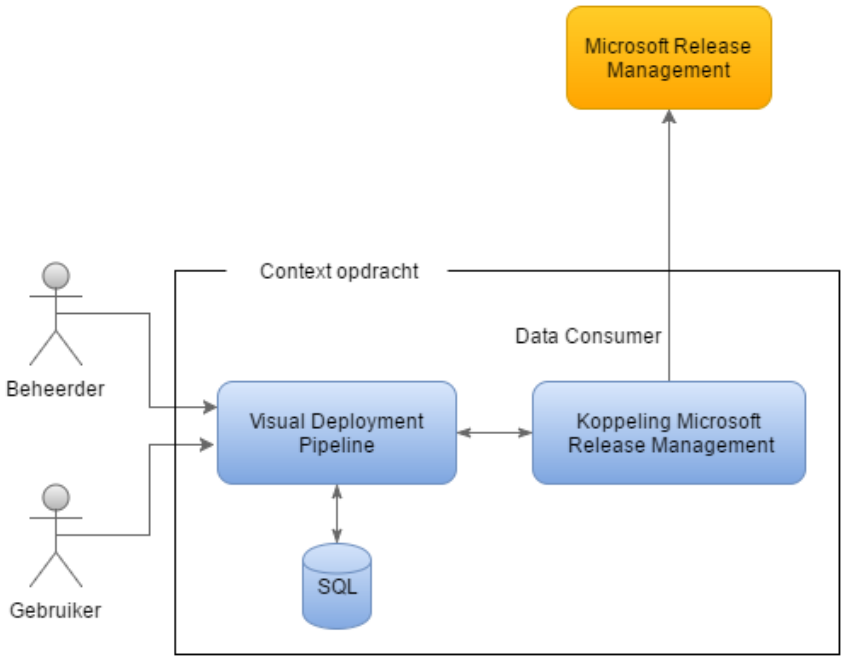
1. Analyse naar mogelijkheden om gegevens uit MRM te halen om een dashboard te realiseren.
2. Realiseren van het dashboard.
3. Koppeling met MRM realiseren.

In figuur 5 is een context diagram van de opdracht te zien. Hierin is te zien dat MRM buiten de context staat. Dit betekent dat het dashboard data vanuit MRM gaat verzamelen, opslaan en weergeven. Alles in het diagram wat blauw is zal ik realiseren.

Alles buiten de rechthoek in het context diagram valt buiten de scope van de opdracht. Dit betekent dat er geen wijzigingen zullen plaats vinden aan componenten buiten de context. Een opsomming van de grenzen van de opdracht zijn:

1. Het dashboard gaat alleen data weergeven van MRM en geen data in MRM aanpassen. (Alleen een consumer en geen provider);
2. Om informatie te zien van het dashboard hoeft de gebruiker zich niet te authentiseren. De informatie is voor iedereen binnen het domein toegankelijk.

In figuur 5 is er verder ook te zien dat er een dashboard gerealiseerd gaat worden. Dit is het dashboard dat is beschreven in de opdracht omschrijving. Hiermee kunnen gebruikers inzicht krijgen in de deployment pipeline.



Figuur 5 context diagram van opdracht

3.1.2 Ontwikkelmethodiek

Aan het begin van dit project had de opdrachtgever een aantal eisen en wensen ten aanzien van het proces. Deze eisen en wensen waren:

1. Overzicht tijdens het gehele project.
2. Snel kunnen ingrijpen als iets mis gaat.
3. Goed kunnen omgaan met wijzigende requirements.
4. Korte tussentijdse opleveringen.
5. Tijdens het project prioriteren van requirements.
6. “Requirements as a Scenario”.

Aan de hand van deze eisen en wensen was het duidelijk dat er niet met een waterval methodiek gewerkt kon worden. Hiermee is het niet mogelijk om tussentijdse opleveringen te doen en ook is het niet mogelijk om snel in te kunnen grijpen als iets mis gaat. Met een iteratieve ontwikkelmethode kan er wel voldaan worden aan deze eisen en wensen. Het is namelijk na elke iteratie mogelijk om in te grijpen voordat er teveel werk is verricht. Als er tijdens het project ook iets moet wijzigen kan dit gedaan worden voordat de volgende iteratie begint.

De opdrachtgever had verder ook een aantal voorkeuren aangegeven:

1. Sprints van twee weken.
2. Keuze tussen Scrum en Kanban.

Info Support voert opdrachten voornamelijk uit met Scrum. Verder had ik als afstudeerder meer ervaring met Scrum. De opdrachtgever en ik hadden daardoor besloten met Scrum te werken.

Met Scrum is het gemakkelijk om een overzicht te houden over het project, omdat er per sprint een contactmoment is met de opdrachtgever (Sprint review). Hier kan de opdrachtgever gelijk ingrijpen als het nodig is. Bij het plannen van een sprint kan de opdrachtgever ook gelijk aangeven welke user story de hoogste prioriteit heeft. Hier kan hij gelijk requirements wijzigen als dat nodig is. Als laatste kan de opdrachtgever ook “Requirements as a Scenario” opstellen. Dit worden de user stories.

Scrum past goed bij de eisen en wensen van de opdrachtgever en voldoen ook aan zijn voorkeuren. Daarom is er gekozen om tijdens dit project met Scrum te werken. Elke sprint in dit project zal twee weken duren.

3.1.3 Aangepaste Scrum

Het is niet mogelijk om Scrum correct uit te voeren in een project waar er alleen één persoon in het ontwikkelteam zit. Scrum is op de volgende wijze aangepast:

Rollen

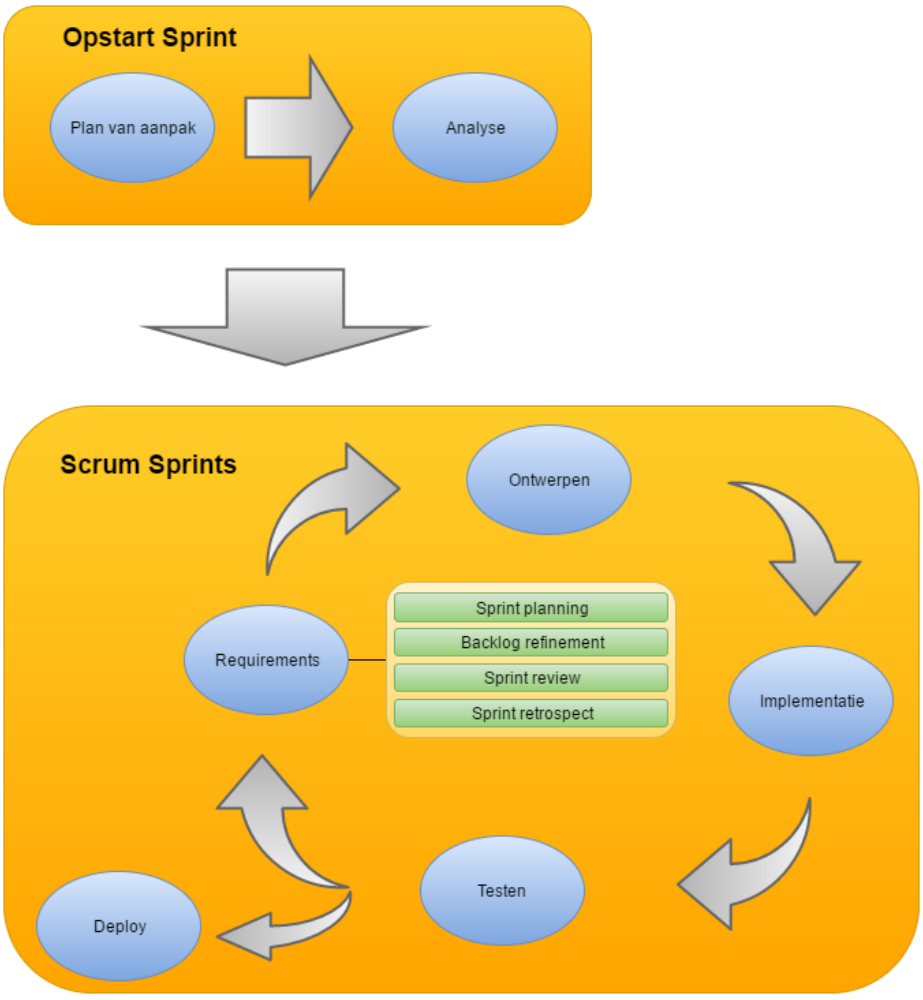
Rol	Wie
Product owner	Paul Borgeld
Ontwikkelteam	Marc de Meza
Scrummaster	Marc de Meza

Het scrumteam bestaat alleen uit Paul Borgeld en Marc de Meza.

Daily stand ups
Daily stand ups worden niet door mij uitgevoerd. Als ik alleen daily stand ups ga houden kan dat leiden tot een situatie waarin ik een onterecht vertrouwen krijg in de voortgang van het project. Dit kan gebeuren omdat ik snel positief ga denken over mijn eigen voortgang, maar in werkelijkheid het project vertraging kan oplopen. De voortgang van de opdracht zal duidelijk bijgehouden worden in TFS.

3.1.4 Fasering

Aan het begin van het project zal er sprake zijn van een sprint 0, oftewel een “opstart sprint”. In deze “opstart sprint” zullen verschillende documenten worden opgeleverd zoals een plan van aanpak en de analyse naar de mogelijkheden om gegevens uit MRM te halen. De rest van de sprints zullen bestaan uit de volgende taken: Requirements, Ontwerpen, Implementatie en Test & verificatie. De volgorde van deze taken kunnen afwijken bij gebruik van technieken zoals Test Driven Development (TDD). In figuur 6 is dit weergegeven.



Figuur 6 sprint faseringen

In figuur 6 is er verder ook te zien dat er vier gesprekken met de product owner horen bij requirements, namelijk sprint planning, backlog refinement, sprint review en sprint retrospect. Deze gesprekken vinden plaats op dezelfde dag met de product owner.

Sprint planning
Tijdens de sprint planning wordt samen met de product owner besproken welke backlogs gerealiseerd gaan worden in de sprint. De product owner geeft ook gelijk de prioriteit weer voor de gekozen backlogs.

Sprint review
Het product dat in de sprint is afgerond wordt gepresenteerd aan de product owner. Tijdens de sprint review kan de opdrachtgever feedback geven op het product. Dit gebeurt aan het einde van een sprint.

Sprint retrospect
Tijdens de sprint retrospect kan de product owner feedback geven over de sprint. Hier kan besproken worden wat goed of fout is gegaan. De retrospect wordt uitgevoerd aan het einde van een sprint.

Back log refinement
Back log items die niet specifiek zijn worden samen met de product owner specifiek gemaakt. Dit vindt plaats aan het begin van een sprint.

3.1.5 Doelstelling en op te leveren producten

In mijn plan van aanpak worden de doelstellingen en op te leveren producten duidelijk beschreven. Deze momenten staan beschreven in Bijlage I.

Product	Beschrijving
Plan van aanpak	In de “opstart sprint” moet een plan van aanpak voor de opdracht gemaakt worden. In dit document komen organisatorische aspecten van het project aan de orde. Aan het einde van deze fase wordt het plan van aanpak opgeleverd.
Analyse	Om de opdracht te kunnen realiseren moet er een analyse uitgevoerd worden. In deze analyse moeten de mogelijkheden om data uit MRM te halen onderzocht worden. Het resultaat kan vervolgens gebruikt worden bij het realiseren van het dashboard. Met dit document kan de opdrachtgever zien welke mogelijkheden er bestaan voor het ophalen van gegevens uit MRM. De motivatie van de gekozen mogelijkheid zal ook beargumenteerd worden, zodat de opdrachtgever terug kan zien waarom het gekozen is. Voor de analyse worden ‘proof of concepts’ (POC) gebruikt.
Requirements	Tijdens de requirements fase worden nieuwe requirements opgehaald voor het project. Na de requirements fase zal het functioneel ontwerp worden opgeleverd met daarin de “Requirements as a Scenario”. Verder wordt er ook gebruik gemaakt van epics, user stories en acceptatie criteria.
Ontwerpen	De nieuwe functionaliteiten voor de huidige sprint worden ontworpen. Dit wordt gedaan met UML. Na het ontwerp fase worden de beslissingen van het project opgeleverd in een technisch ontwerp.
Implementatie	De nieuwe functionaliteiten worden gerealiseerd door het “ontwikkelteam”.
Test & verificatie	Tijdens het realiseren van de nieuwe functionaliteiten zal er gewerkt worden met Test Driven Development(TDD). Hiermee zullen de unit en integratietests gemaakt worden. Voor de acceptatie tests wordt SpecFlow gebruikt. Met SpecFlow worden de requirements overgezet naar Specifications by Example. Met deze specifications worden er automatisch tests gegenereerd door SpecFlow.
Deployment	Nadat de nieuwe functionaliteit is gerealiseerd en getest zal er een kleine demonstratie zijn tijdens de sprint review. Dit zal plaats vinden op de omgeving van de opdrachtnemer zelf. Naarmate het project vordert zal de opdrachtgever mogelijk de applicatie in gebruik willen nemen. In dat geval zal de deployment plaats vinden op de productie omgeving.
Back log refinement	Back log items die niet specifiek zijn worden samen met de product owner specifiek gemaakt. Dit vindt plaats aan het begin van een sprint.

Product	Beschrijving
Sprint planning	Tijdens de sprint planning wordt samen met de product owner besproken welke user story gebouwd gaan worden in de sprint. De product owner geeft ook gelijk de prioriteit weer voor de gekozen user story.
Sprint review	Het product dat in de sprint is afgerond wordt gepresenteerd aan de product owner. Tijdens de sprint review kan de opdrachtgever feedback geven op het product. Dit gebeurt aan het einde van een sprint.
Sprint retrospect	Tijdens de sprint retrospect kan de product owner feedback geven over de sprint. Hier kan besproken worden wat goed of fout is gegaan. De retrospect wordt uitgevoerd aan het einde van een sprint.

3.1.6 Globale planning en mijlpaalproducten

Om een overzicht te krijgen van wanneer bepaalde taken worden uitgevoerd heb ik een globale planning gemaakt. In deze planning heb ik aangegeven wanneer ik aan bepaalde onderdelen moet werken. Elke sprint duurt twee weken en eindigt met een Sprint review/Sprint retrospect/Sprint planning.

Mijlpaal	Begin	Eind	Duur
Plan van aanpak	01-02-2016	12-02-2016	2 Weken
Analyse mogelijkheden Microsoft Release Management	15-02-2016	09-03-2016	$3^{1/2}$ Weken
Sprints	11-03-2016	19-05-2016	10 weken
Sprint review/Sprint retrospect/Sprint planning	-	-	1 dag
Afstudeerverslag	01-02-2016	03-06-2016	18 Weken

In elke sprint zijn er een aantal taken die opnieuw uitgevoerd worden. In de volgende lijst zijn de taken opgenomen die gedurende de sprint worden uitgevoerd. De begin datum van deze taken is 14-03-2016 en de einddatum is 19-05-2016. De totale duur is 10 weken.

- Requirements analyseren
- Functioneel ontwerp
- Technisch ontwerp
- Applicatie realiseren
- Tests documenteren
- Tests automatiseren

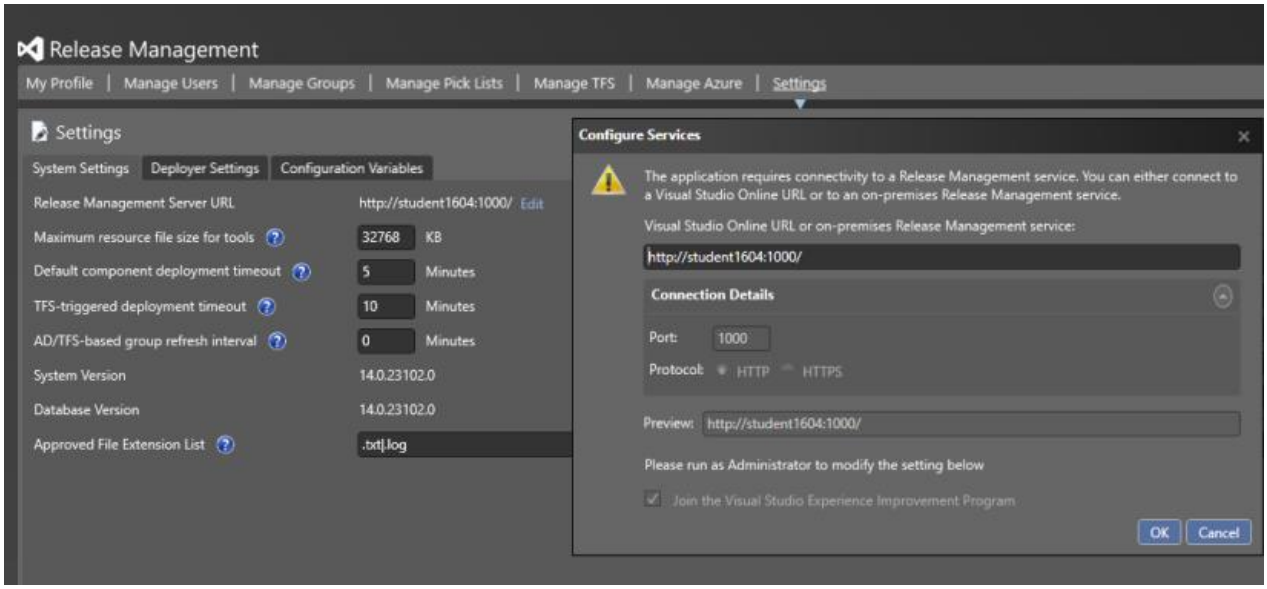
4. Uitvoering Analyse

In dit hoofdstuk ga ik duidelijk beschrijven hoe ik mijn analyse heb uitgevoerd en wat de conclusie hiervan was. Mijn volledige analyse is te raadplegen in Bijlage II.

4.1 Vooronderzoek

Voordat ik aan mijn analyse begon heb ik navraag gedaan naar de versie van MRM waar ik een verbinding mee moest maken. Er zijn namelijk verschillende versies op de markt. Deze versies zijn MRM 2013 update 6, MRM 2015 update 1 en MRM 2015 update 2. De MRM 2015 update 2 was nog in bèta versie toen ik aan mijn opdracht begon. Mijn opdrachtgever had besloten dat ik mijn analyse en opdracht volledig moest richten op 2015 update 1. Verder moest ik tijdens het realiseren ook rekening houden dat de MRM bron kon wijzigen.

Tijdens mijn analyse had ik besloten kort een vooronderzoek te doen over wat er precies allemaal al bekend was over de verbinding naar MRM. Ik begon met het bestuderen van de officiële documentatie van Microsoft. Uit de documentatie van MRM kwam ik te weten dat er gebruik wordt gemaakt van een desktop client en een server[RF04]. In figuur 7 is de desktop client te zien. Deze client en server communiceren door middel van HTTP. Er stond ook beschreven dat er een API is, maar deze niet gedocumenteerd is[RF03]. Verder las ik dat in toekomstige versies hier wel aandacht aan besteed gaat worden[RF03].



Figuur 7 desktop client van MRM

Hierna heb ik interne bronnen van Info Support geanalyseerd. Het bleek dat in het verleden al een verbinding met MRM gemaakt was. Dit was gedaan in de 2013 update 6 versie door direct met de database te verbinden[RF01]. In een ander artikel was het iemand gelukt om de API van MRM 2015 update 1 aan te spreken met een Powershell script[RF02].

4.2 Werkwijze

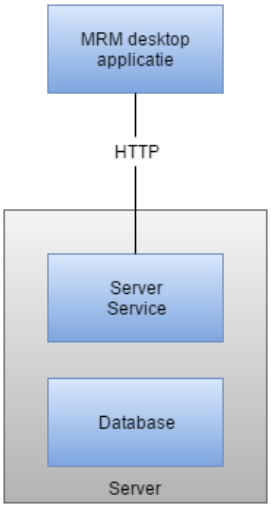
Na afronding van mijn vooronderzoek wist ik wat ik precies wilde analyseren. Mijn eerste stap was om te kijken of ik verbinding kon maken met de database van MRM 2015 update 1 en of ik hier toegang had tot informatie. Verder wilde ik ook analyseren hoe de API precies werkte en of ik hier ook informatie uit kon halen. De vraagstukken die ik wilde analyseren zijn hier beschreven:

Api mogelijkheden
Zoals eerder vermeldt maakt MRM 2015 update 1 gebruik van een desktop client en een server. Verder is er ook aangegeven dat deze twee door middel van een API communiceren. Voor het analyseren van de API mogelijkheden ga ik alle netwerk verkeer “sniffen”. “Sniffen” is de term die gebruikt wordt wanneer iemand met behulp van bepaalde software data verkeer tussen een client en server onderschept. Verder ga ik kijken welke gegevens hier vrij worden gegeven en hoe dit gebruikt kan worden.

Database mogelijkheden
Voor het ophalen van de gegevens wordt er ook gekeken naar het direct uitlezen van de database van MRM 2015 update 1. Alle benodigde informatie is in de database te vinden. Het direct uitlezen van de database brengt problemen met zich me die ik paragraaf 4.3 behandel. Verder wordt er ook gekeken of het uitlezen van de database nog mogelijk zal zijn in MRM 2015 update 2.

4.3 Microsoft Release Management API

De eerste mogelijkheid die ik heb geanalyseerd was de API. MRM 2015 update 1 maakt gebruik van twee componenten die met elkaar communiceren, namelijk de client en de server. De server biedt een API aan waar de client mee kan communiceren. Met deze API is de MRM database volledig afgeschermd van de client. Dit betekent dat de client nooit wijzigingen kan doorvoeren zonder dat de MRM API weet wat er gewijzigd wordt.



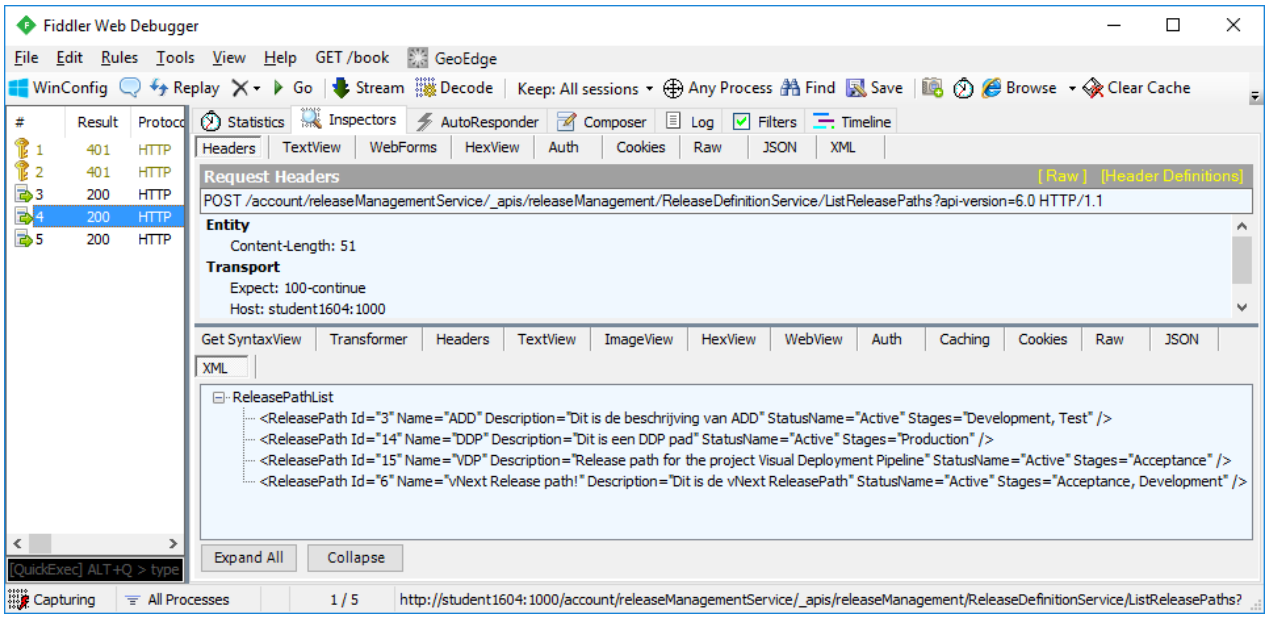
Figuur 8 architectuur MRM

Tijdens het installeren van de MRM server wordt er toestemming gevraagd om een webservice aan de Internet Information Services (IIS) toe te voegen. IIS is de webserver die gebruikt wordt op Windows. Dit betekent dat MRM een API in de vorm van een webservice gebruikt voor alle communicatie. De API die wordt aangeboden door de server is momenteel niet gedocumenteerd. Dit brengt als probleem mee dat niet duidelijk is welke gegevens wel of niet uitgelezen kunnen worden.

Voor het onderscheppen van de communicatie tussen de client en server heb ik de tool Fiddler gebruikt. Fiddler had ik geïnstalleerd en geconfigureerd om alleen verkeer van MRM te onderscheppen. Dit had ik gedaan zodat Fiddler overzichtelijk bleef. Vervolgens ben ik in de client van MRM verschillende pagina’s gaan inladen.

De eerste keer toen ik dit had uitgevoerd kreeg ik geen resultaten in Fiddler te zien. Dit kwam door een foute configuratie in Fiddler. Fiddler luistert niet automatisch naar verbindingen op localhost. Na enkele kleine aanpassingen in Fiddler kreeg ik alle verkeer tussen de client en server te zien.

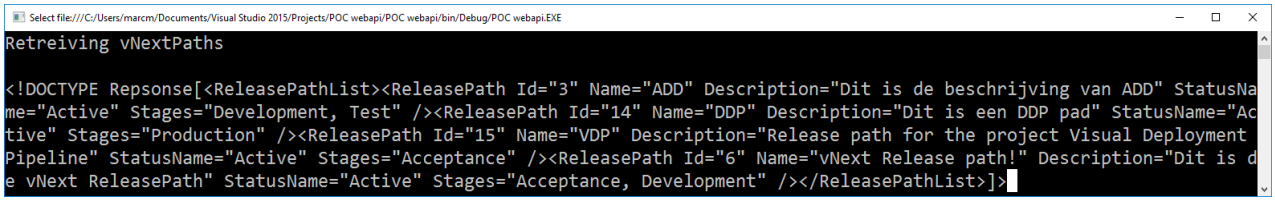
In figuur 9 is te zien hoe ik naar een pagina van de Releases navigeerde en alle gegevens in XML formaat kon uitlezen. Tijdens mijn opdracht moest ik dus voor alle informatie eerst opzoeken waar in MRM die stond en vervolgens vertalen naar entiteiten in C#.



Figuur 9 XML data afkomstig van MRM

Proof of concept

Om zeker te zijn dat ik de gegevens ook daadwerkelijk kon uitlezen in een C# applicatie heb ik een POC gemaakt waarin ik wilde weten of alle ReleasePaths uitgelezen konden worden. De applicatie die ik had gemaakt voerde een HTTP request naar de server uit. Het resultaat van de request is te zien in figuur 10.



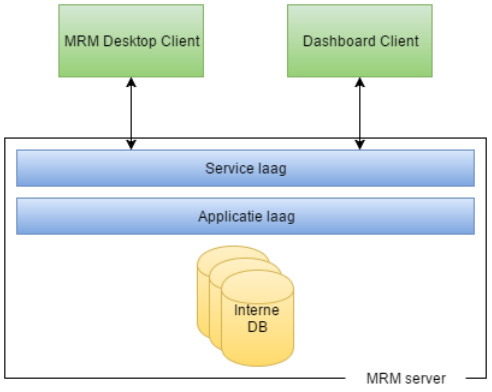
Figuur 10 ReleasePath Informatie uit POC

Voordelen gebruik van API

Bij het gebruik van de API is er geen kennis nodig over de structuur van de database. Het is niet nodig om te kijken hoe en waar MRM de data opslaat. Met deze abstractie laag boven de database is het mogelijk dat de interne opslag van MRM kan wijzigen zonder dat er wat verandert aan alle applicaties die verbonden zijn.

De gegevens die uit de database worden opgehaald door MRM gaan ook door de applicatie (business) laag. Hierdoor zorgen ervoor dat MRM de data kan presenteren zoals bedoeld was.

In figuur 11 is te zien dat het dashboard zal aansluiten op een bestaande API. De MRM client maakt al gebruik van deze API om



Figuur 11 API aansluiting lagenmodel

precies dezelfde informatie op te halen. Door ook gebruik te maken van de API zorgen we ervoor dat alle communicatie alleen op één manier beschikbaar is

Encapsulatie van data
Het encapsuleren van data zorgt ervoor dat de gebruiker niet zomaar acties kan uitvoeren om de data zonder dat de applicatie weet wat aan het wijzigen is. Door de API te gebruiken zorgen we ervoor dat de data nog steeds geëncapsuleerd is.

Voordelen veiligheid
Het laatste voordeel bij het gebruiken van de API is dat er geen gevoelige informatie kan uitlekken. MRM bepaalt of informatie verzonden mag worden vanuit de API. Als MRM bepaalt dat informatie gevoelig is, dan wordt deze informatie niet over de API verstuurd.

Algemene nadeel toekomst
Een niet gedocumenteerde API betekent dat het bedrijf die de API heeft gemaakt geen garantie biedt dat die API niet gaat wijzigen. Bij de volgende update is het mogelijk dat er een nieuwe structuur in de data zit of dat bepaalde http requests niet meer zullen werken. Verder is het niet mogelijk om de API in update 1 te benaderen. Applicaties die zich hebben aangesloten bij de service laag van de server zullen niet meer te gebruiken zijn. Voor deze update zal er een alternatief bedacht moeten worden totdat de update 3 uit is.

Geen documentatie
Het laatste nadeel dat veel invloed kan hebben is dat de API niet gedocumenteerd is. Tijdens het ontwikkelen van een koppeling met de service laag zal het niet mogelijk zijn om documentatie te raadplegen. De verbinding tussen de client en server zal afgeluisterd moeten worden en deze gegevens moeten onderzocht worden om te kijken of ze gebruikt kunnen worden.

4.4 Database

De tweede mogelijkheid die ik heb geanalyseerd was het direct verbinden met de database. MRM maakt gebruik van een SQL database waarin alle gegevens over de installatie worden opgeslagen. Deze gegevens kunnen gemakkelijk uitgelezen worden, maar hoe dit precies gedaan wordt en wat de gevolgen hiervan zijn heb ik geanalyseerd.

Tijdens het installeren van MRM 2015 update 1 server wordt er gevraagd om een SQL database op te geven. Nadat de installatie is afgerond is het mogelijk om te navigeren naar de SQL server en daarin alle MRM tabellen te benaderen. Dit kan gebruikt worden om alle gegevens die nodig zijn uit te lezen. Het direct aanspreken van een database brengt enkele risico's met zich mee. Ten eerste is de structuur van de data niet overzichtelijk. Het zal veel tijd kosten om precies te achterhalen waar bepaalde data staat. Ten tweede is het ook mogelijk dat de database bij kleine updates verandert. In dat geval moeten alle queries herschreven worden. Ten derde kunnen er te veel rechten worden toegekend aan het account waarmee de tabellen worden benaderd en is het mogelijk data te wijzigen in MRM zonder validatie te hebben op je invoer. Deze aspecten worden in dit hoofdstuk besproken.

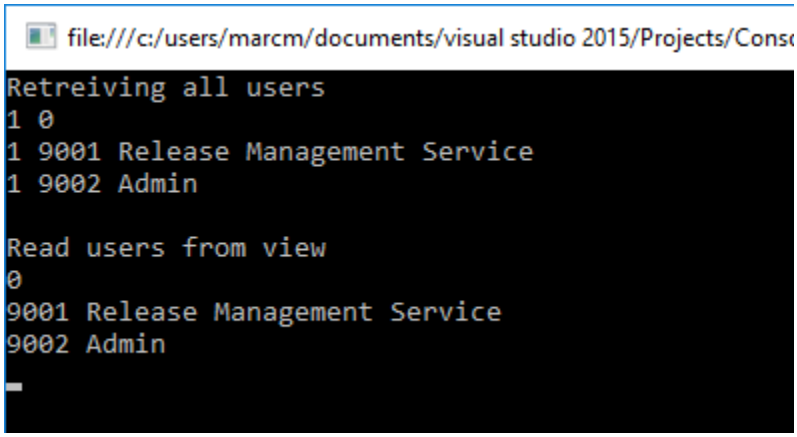
Abstractie met views
In plaats van het verstrekken van rechten om bepaalde tabellen direct uit te lezen is het ook mogelijk views te maken in een SQL database. Views zijn afgeleide tabellen die gegenereerd worden bij het opvragen van de view. Met deze views is het mogelijk om de gegevens die wij nodig hebben uit de database eerst te definiëren. Na het definiëren van de views moet de nieuwe database gebruiker alleen maar rechten krijgen voor het lezen van deze views.

Door views te gebruiken zorgen wij ervoor dat het afgeven van rechten niet onoverzichtelijk wordt bij ingewikkelde tabellen. Een database administrator hoeft niet per tabel en per kolom te gaan kijken of de database gebruiker hier recht op heeft.

Het voordeel van het gebruiken van views is dat wij een abstractie laag boven de database realiseren waarmee gecommuniceerd kan worden. Indien de database wijzigt moeten alleen de views worden aangepast. In de views moeten de tabellen die worden bevraagd aangepast worden zonder de namen en kolommen van de views aan te passen. Hierdoor is het niet nodig om een applicatie die gebruik maakt van deze views aan te passen.

Een risico bij het gebruiken van views is dat een view snel complex kan worden. Tijdens het maken van de views kan het mogelijk zijn dat er overbodige tabellen worden bevraagd en hierdoor de database wat langzamer wordt.

Proof of concept
Om te bewijzen dat het mogelijk is om gegevens uit de database van MRM 2015 update 1 te lezen is er een POC gemaakt. In deze POC wordt er een verbinding met de database gelegd. Met deze verbinding worden alle gebruikers in MRM opgehaald. Het ophalen van de gebruikers wordt eerst gedaan door het uitlezen van de User tabel. Vervolgens worden alle gegevens, met behulp van een view, ingeladen om te bewijzen dat het ook mogelijk is om de gegevens uit een view te halen.

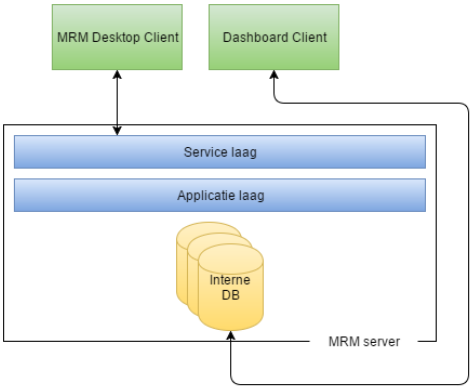


Figuur 12 Uitlezen gegevens van de database

Voordelen bij het gebruiken van de database
Eén van de voordelen van het direct verbinden met de database is dat al de informatie die erin zit ter beschikking wordt gesteld. Bij een API zal het nodig zijn om te werken met de gegevens die je kunt verkrijgen. Als de gegevens van een API niet voldoende zijn is er geen alternatief. Bij het gebruiken van de database zijn alle tabellen te benaderen en dus ook al de informatie. Verder is het verbinden met een database een bekende techniek en brengt dit ook geen moeilijkheden met zich mee. Informatie vinden over hoe er verbonden wordt met een SQL database is gemakkelijk te vinden.

Als laatste is de toekomst van de verbinding duidelijk. Bij elke nieuwe versie van MRM moet alleen de database geanalyseerd worden om te kijken hoeveel het is veranderd. Alle versies van MRM maken gebruik van een database en zijn daarom allemaal te benaderen.

Nadelen van het gebruik van de database
Ook al is het een voordeel dat alle informatie ter beschikking wordt gesteld wordt het ook gelijk genoteerd als een nadeel. Door de database te kunnen benaderen is er een groot verlies op het gebied van encapsulatie. MRM bepaalt niet meer zelf hoe er wordt omgegaan met de gegevens die het intern opslaat. Verder kunnen er zelfs aanpassingen in de database plaatsvinden zonder dat MRM daar wat van af weet. In figuur 13 is te zien hoe een applicatie direct verbinding maakt en alle lagen van MRM overslaat.



Figuur 13 overslaan lagenmodel database

Verder bestaat de database van MRM 2015 update 1 uit meer dan 100 tabellen. Na verloop van tijd is het niet meer duidelijk welke tabellen bevraagd worden. Dit maakt het moeilijker om wijzigingen door te voeren in de applicatie. Bij het analyseren van de data is de structuur ook niet helemaal logisch opgebouwd. Een voorbeeld hiervan is dat ReleasePaths zijn opgedeeld in twee verschillende tabellen.

De rechten van de database gebruiker spelen ook een belangrijke rol. De gebruiker die de database benadert moet duidelijk worden opgesteld en bijgewerkt. Als er teveel rechten worden weggegeven kan een applicatie data gaan wijzigen of zelfs verwijderen.

Als laatste moet er goed worden omgegaan met wijzigende structuur. Na een update in MRM worden er vaak nieuwe tabellen toegevoerd of worden ze verwijderd. Na zo een update moet de data laag van een applicatie volledig aangepast worden.

4.5 Conclusie

De opties die uit mijn analyse zijn gekomen heb ik besproken met de opdrachtgever. Uiteindelijk heb ik als advies gegeven om gebruik te maken van de API. Mijn advies heb ik gebaseerd op de volgende punten:

- Data is geëncapsuleerd;
- Sla geen lagen van MRM over zoals de applicatie (business) laag;
- Uitzoeken van database structuur duurt, volgens mij, langer dan het uitzoeken van een niet gedocumenteerde API.

Zoals reeds vermeld, is het niet nodig om de database uitgebreid te bestuderen voordat er geprogrammeerd kan worden. Hiermee is het mogelijk om tijd te besparen tijdens het ontwikkelen van het dashboard.

Het enige probleem is dat de API niet gedocumenteerd is. Dit betekent dat ik alle requests zelf moest uitzoeken. Het uitlezen van requests kost, volgens mij, minder tijd dan het uitlezen van de database. Verder kan er meer fout gaan bij het uitlezen van de database dan bij het uitlezen van een API, zoals beschreven in de nadelen.

Indien het nodig is om nog meer gegevens uit te lezen die niet door de API te benaderen zijn, is het altijd mogelijk om de database aan te spreken.

5. Uitvoering sprints

In dit hoofdstuk ga ik per sprint uitleg geven over hoe mijn afstuderen is verlopen. In deze uitvoering probeer ik duidelijk te maken hoe ik bepaalde taken heb uitgevoerd en welke beslissingen ik hierbij heb genomen. Verder zal ik beschrijven hoe ik ben omgegaan met (mogelijke) moeilijke situaties.

5.1 Opstart sprint

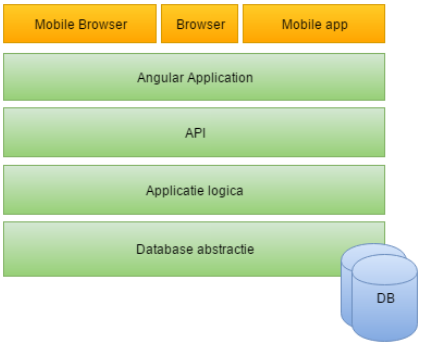
In deze paragraaf beschrijf ik belangrijke beslissingen die ik heb genomen voordat mijn sprint was begonnen. Deze beslissingen waren niet gedurende een sprint genomen en krijgen daarom een aparte paragraaf.

5.1.1 Realiseren

Voor mijn project moet ik een dashboard maken die in een web- en mobile omgeving moet werken. De beslissing over hoe deze twee omgevingen moesten werken mocht ik zelf bepalen. Eén van de eisen aan mijn project was dat de back-end ASP.NET als framework moest gebruiken. Alle functionele en niet-functionele eisen en wensen zijn te vinden in bijlage III.

Mobile first HTML5 & CSS3
Het was mogelijk om een webapplicatie te maken en vervolgens ook tijd te besteden aan het maken van een mobiele applicatie die dezelfde functionaliteit aanbood. Het probleem hiermee is dat de logica van de applicatie op twee plekken geschreven moet worden. Verder moet bij het toevoegen van functionaliteit ook op twee plekken een wijziging plaats vinden.

Een andere optie was om een “Application Programming Interface” (API) aan te bieden waarmee een mobiele applicatie en een website zoals een “single page application” mee konden communiceren. Dit ging de goede kant op, omdat de logica van de applicatie maar op één locatie aangepast moest worden. Het enige probleem dat wij nog overhielden was dat bij een update de mobiele applicatie en webapplicatie aangepast moest worden met nieuwe functionaliteit[RF07].



Figuur 14 lagenmodel van applicatie

Uiteindelijk is er voor gekozen om een “single page applicatie” te maken die op een mobiel ook goed te gebruiken is. Deze aanpak heet “Mobile First”. Bij Mobile First wordt de user interface eerst ontworpen voor een mobiele browser. Als wij eerst rekening houden met de krachten van een mobiel apparaat, die normaal minder krachtig zijn, weten wij dat het op een normale browser ook goed zal werken.

Uiteindelijk is er voor gekozen om dus een website te maken waarmee de verschillende apparaten gaan communiceren. Indien er een mobiele applicatie moet komen kan deze applicatie de webapplicatie ook inladen. Deze lagen zijn duidelijk te zien in figuur 14.

AngularJS
Nadat de keuze was gemaakt om de website een single page applicatie te maken heb ik gekeken naar welke frameworks ik hier voor kon gebruiken. Info Support geeft cursussen in het programmeren in AngularJS. Verder heeft Info Support veel kennis in het ontwikkelen en onderhouden van AngularJS applicaties. Daarom is er besloten om te werken met AngularJS

AngularJS 2.0 was ook een optie bij de overweging van framework. AngularJS 2.0 was aan het begin van mijn project nog in een bèta versie. Daarom besloot ik om geen gebruik te maken van het nieuwe framework. Bèta versies kunnen in de toekomst nog wijzigen. Dit betekent dat het framework zonder waarschuwing aangepast kan worden.

Webapi
De keuze voor het maken van een single page applicatie heeft invloed gehad op de back-end van de applicatie. AngularJS verwacht namelijk om door middel van HTTP requests te sturen naar een back-end. Ik besloot om Webapi te gebruiken als webservice voor AngularJS aangezien de eis werd gesteld om Microsoft technologieën te gebruiken in dit project.

Git
Voor het beheren van mijn versies bood TFS de mogelijkheid aan om een git te gebruiken. Als versiebeheertool heb ik dus gebruik gemaakt van git.

5.2 Sprint 1 – ReleasePaths

In deze paragraaf beschrijf ik de eerste sprint die ik heb uitgevoerd. Eerst zal ik beschrijven hoe ik mijn eerste sprint had gepland. Vervolgens zal ik beschrijven hoe ik mijn analyse op de eisen en wensen van de opdrachtgever heb uitgevoerd. Verder zal ik mijn ontwerpdocumenten beschrijven en welke beslissingen ik hierbij heb genomen. Hierna zal ik aantonen wat er allemaal met deze sprint is gerealiseerd en als laatste beschrijf ik hoe ik heb getest.

5.2.1 Planning en Analyse

Mijn eerste sprint begon met een sprint planning met de opdrachtgever. In dit gesprek hebben wij globaal gekeken welke functionaliteiten allemaal in de applicatie moesten komen. Nadat wij een globaal gesprek hadden gehad, hebben wij bepaald welke taken wij in die sprint wilde uitwerken. Wij begonnen met het benoemen van een epic. Een epic is een bepaalde functionaliteit die de opdrachtgever terug wilt zien in het project, maar te groot is om in één sprint te realiseren. Deze epic is vervolgens opgedeeld in meerdere user stories. De user stories beschrijven een bepaalde actie die een gebruiker van het systeem moet kunnen uitvoeren. Het is mogelijk om een user story te hebben die niet verbonden is aan een epic. Dit betekent dat de user story klein genoeg is om gelijk toe te voegen.

De epic van deze sprint was ReleasePath informatie. De bedoeling was dat alle functionaliteiten van de ReleasePaths van deze sprint af moest komen. Nadat deze epic was bedacht hebben wij user stories gemaakt. Al mijn user stories en de uitwerkingen hiervan zijn te raadplegen in Bijlage III. De user stories die waren bedacht zijn in de volgende tabel te zien:

Nummer	Story	Prioriteit	TFS id
US1	Als gebruiker wil ik een overzicht van Release Paths zodat ik een Release Path kan selecteren	1	9096
US2	Als gebruiker wil ik een detail scherm van een Release Path zien zodat ik alle informatie van die Release Path kan zien	2	9071

De volgende user stories had ik toegevoegd aan mijn TFS omgeving. Hierin kon ik taken koppelen aan mijn user stories en de voortgang bijhouden. Om een duidelijk beeld te krijgen van hoe ik van user story tot analyse was gekomen ga ik een user story uitwerken. Hiervoor gebruik ik US1.

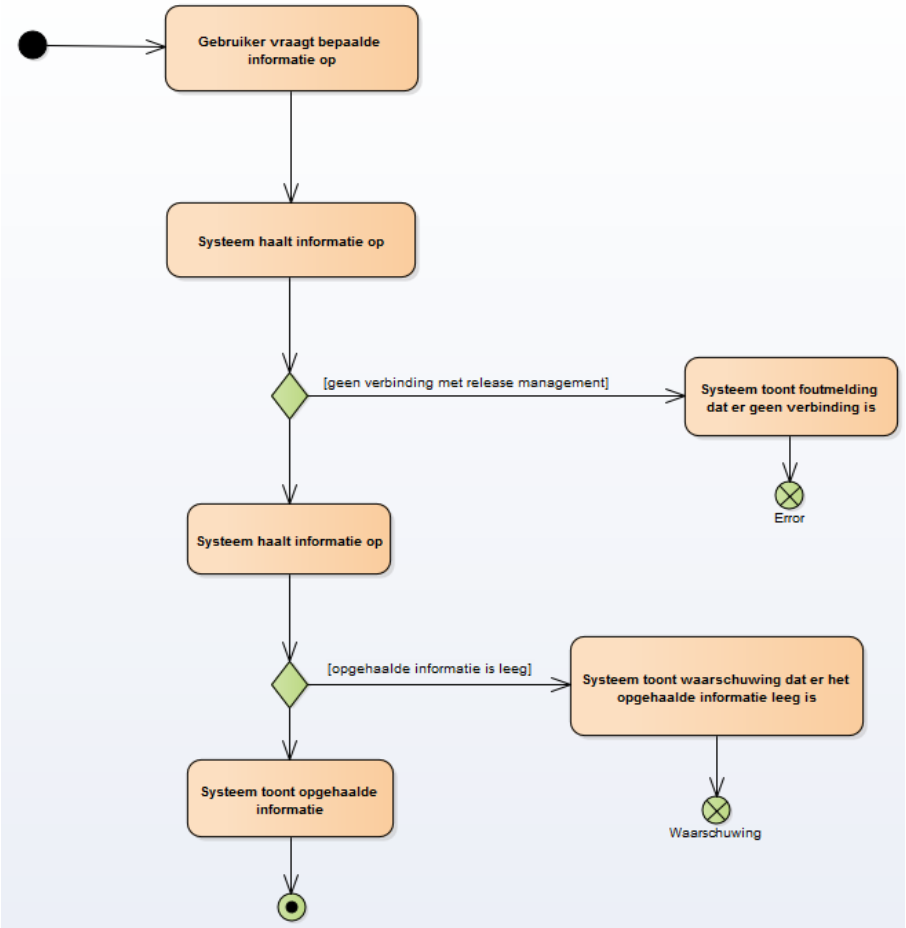
Nadat ik US1 had ingevoerd in TFS was het mijn taak om acceptatiecriteria te definiëren. Acceptatie criteria worden gebruikt om te valideren of gebouwd zal worden wat de opdrachtgever nodig heeft. Verder kunnen deze criteria ook gebruikt worden als input voor het testen. Aan de hand van de criteria weet ik of de user story is afgerond of niet. Eerst had ik alle criteria genoteerd die ik belangrijk vond. Vervolgens kon mijn opdrachtgever criteria toevoegen en/of verwijderen. De criteria die ik had verzameld voor US1 zijn in de volgende tabel te zien:

Nummer	Criteria
US1	**Functioneel** 1. Alle release paths in MRM moeten getoond worden 2. Van alle release paths moet de naam en beschrijving er bij staan **Niet-functioneel** 1. Als er geen release paths aanwezig zijn moet er een melding komen 2. Als er geen verbinding is met MRM moet er een waarschuwing zijn 3. Alleen MRM versie 2015 update 1 wordt ondersteund 4. Alleen Internet Explorer versie 10 en hoger worden ondersteund 5. Op tablets wordt alleen de safari browser ondersteund 6. De aantal verzoeken naar MRM moeten minimaal blijven(cache) 7. Er moeten Microsoft Technologieën gebruikt worden 8. De API is geschreven met in C# m.b.v. Webapi 9. De Front-end is geschreven met AngularJs

Het prioriteren van de user stories en epics werd door de opdrachtgever uitgevoerd. De enige vorm van prioritering was de volgorde van de user stories. Ik heb hiervoor gekozen om de opdrachtgever te dwingen een keuze te maken welke user story belangrijker is. Hierdoor zijn er gedurende de sprints geen twijfels over welke user story belangrijker is, in tegenstelling tot bijvoorbeeld MOSCOW, waar na het prioriteren er alsnog een aantal user stories zijn met dezelfde prioriteit. In sommige gevallen waren user stories afhankelijk van elkaar. Een voorbeeld hiervan was met US1 en US2. Het is niet mogelijk om US2 te realiseren zonder US1 af te ronden. Hiermee hield ik rekening tijdens het prioriteren met de opdrachtgever.

Voor het bijhouden van alle epics en user stories is er gebruik gemaakt van TFS. In TFS heb ik ook beschikking over een Scrum bord waarin ik al mijn taken kan zien voor de sprint. Verder heb ik ook een grafiek waarin ik mijn sprint burndown kan zien.

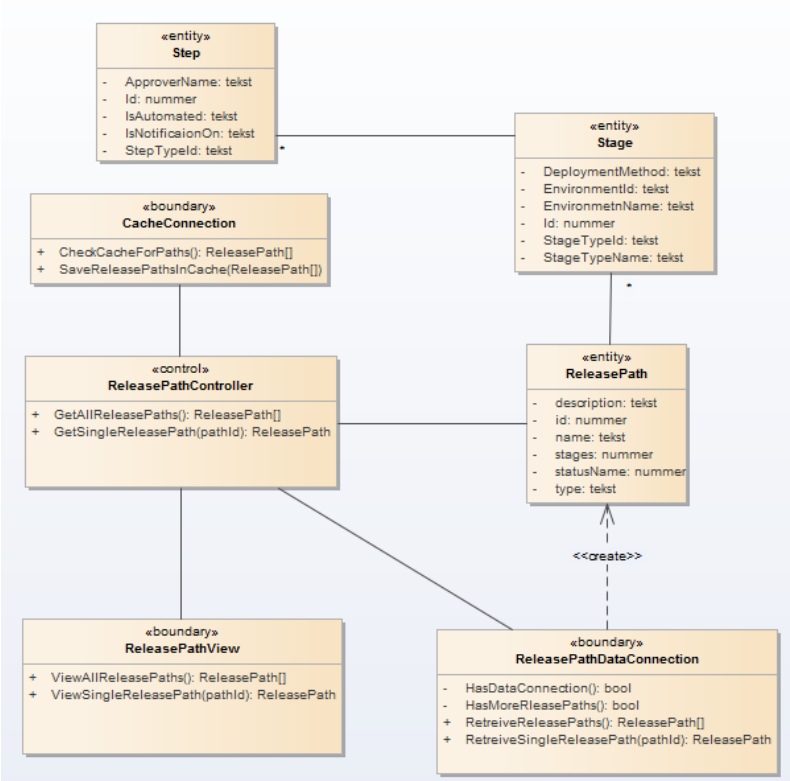
In sommige gevallen was het voor mij niet helemaal duidelijk hoe een bepaalde functionaliteit moest verlopen. Voor het tonen van waarschuwingen en foutmeldingen wilde ik meer duidelijkheid creëren. Om een duidelijk inzicht te krijgen in hoe dit in de applicatie moest verlopen had ik een activity diagram gemaakt. Dit diagram is te zien in figuur 15. Ik heb een globale diagram gemaakt die bij alle user stories past die waarschuwingen en foutmeldingen moeten geven. Nadat ik dit diagram had gemaakt kon ik de opdrachtgever ook duidelijk maken hoe deze functionaliteit te werk moest gaan.



Figuur 15 verloop van waarschuwingen en foutmeldingen

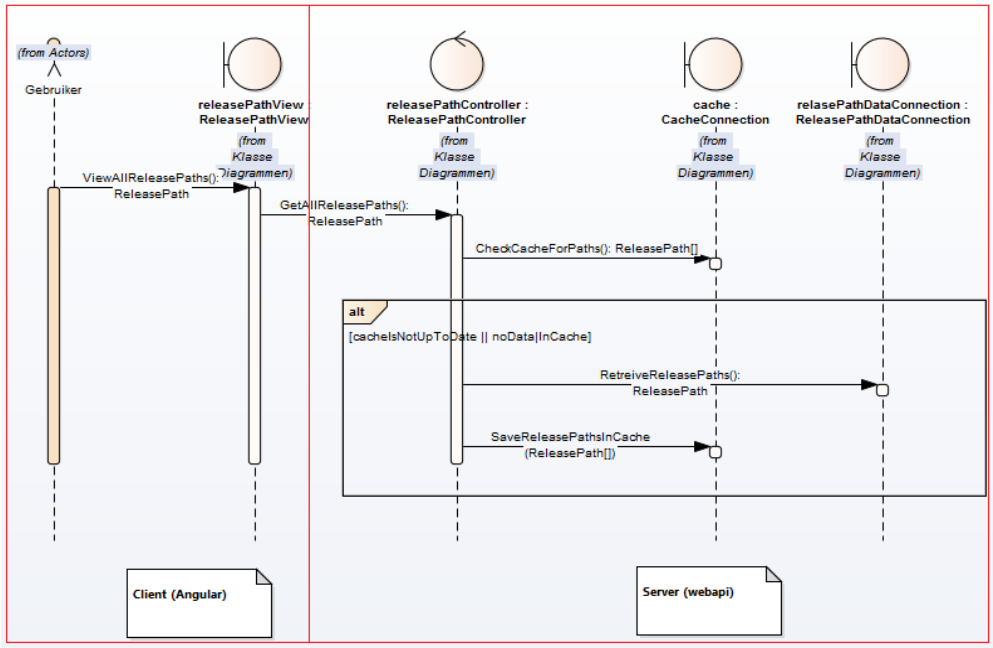
Voordat ik begon met ontwerpen en realiseren wilde ik nog een analyse klasse diagram maken. Met een analyse diagram wilde ik de functionaliteiten met de opdrachtgever bespreken. Hieruit kon ik afleiden of ik de functionaliteiten ook goed had begrepen. Indien ik het fout had begrepen was het mogelijk voor de opdrachtgever om in te grijpen. Mijn analyse klasse diagram is te zien in figuur 16. In mijn analyse klasse diagram is ook te zien dat alle klassen een beschrijving boven hun naam hebben. Deze beschrijving is een stereotype. Met een stereotype is het mogelijk om aan te geven welke taak deze klasse zal hebben. Bijvoorbeeld de klasse Step is een ‘entity’. Dit betekent dat hij alleen verantwoordelijk is voor data. Verder is ook te zien dat er een ‘boundary’ is. Met deze stereotype is aan te geven waar precies de applicatie verbinding maakt met een externe gebruiker. In mijn geval is dat de view en de klasse die verbinding maakt met MRM. Als laatste hebben wij een ‘control’. Deze stereotype is verantwoordelijk voor het koppelen van alle boundaries en entiteiten. Ik had besloten om dit in mijn diagram weer te geven zodat er een duidelijk beeld is waar de gebruiker of MRM interactie heeft met het systeem.

De entiteiten in figuur 16 zijn naar aanleiding van de XML van MRM opgesteld. De attributen die ik heb opgenomen in het diagram zijn letterlijk terug te vinden in de XML. De gegevens die uit MRM zijn niet vaak genormaliseerd. Dit is te zien aan de attributen EnvironmentId en EnvironmentName in de klasse Stage. Normaal gesproken zou dit in een andere klasse moeten staan. Aangezien dit alleen maar ging om een aantal attributen had ik er geen aandacht aan besteed.



Figuur 16 analyse klasse diagram

Als laatste wist ik nog niet precies hoe de communicatie moest verlopen in mijn applicatie. Daarom had ik besloten een sequentie diagram te maken van US1. Met mijn sequentie diagram hoopte ik duidelijk te zien hoe informatie binnen de applicatie moest verlopen. Dit had ik alleen gedaan voor US1, omdat na mijn eerste diagram ik al genoeg inzicht had. Deze sequentie is te zien in figuur 17.



Figuur 17 ophalen gegevens uit MRM of cache analysis sequentie

Alle sprints die hierna volgden was ik van plan op deze wijze aan te pakken. Op deze manier was het mogelijk snel feedback te krijgen van de opdrachtgever.

5.2.2 Ontwerpen en Realiseren

Tijdens het ontwerpen en realiseren had ik besloten mijn beslissingen goed te ontwerpen. Al mijn ontwerp beslissingen zijn terug te zien in Bijlage IV. De Webapi die ik realiseer is een object georiënteerde applicatie dit blijkt ook uit mijn ontwerpen die hier volgen.

XML – Webapi
Het eerste wat ik tijdens deze sprint heb gerealiseerd is het ophalen van gegevens uit MRM. Door mijn analyse wist ik dat alle requests een XML document terug gaven. Dit betekent dat ik een tool moet vinden waarmee het mogelijk is om alle XML over te zetten naar C# objecten. Het eerste waar ik aan dacht was om alle nodes in het XML document langs te gaan en hier handmatig objecten van te maken. Bij een eenvoudige XML document is dit voldoende, maar de XML documenten van MRM zijn ingewikkeld.

Uiteindelijk had ik de officiële documentatie van Microsoft doorgenomen en had ik meer informatie gelezen over de XmlSerializer. Met deze serializer was het mogelijk om aan de hand van een XML document een object te maken. Het enige waar ik rekening mee moest houden tijdens her deserialiseren was dat lijsten in XML niet één op één overkomen met C# modellen. Een voorbeeld van een stukje XML waar ik mee te maken had is hier te zien:

```
<ReleaseList>
        <Release id='1' />
</ReleaseList
```

Om dit te kunnen deserialiseren moet er een klasse ReleaseList zijn die een lijst of array bevat van Releases. Hierdoor had ik besloten klassen te maken die alleen gebruikt werden tijdens het deserialiseren. Een voorbeeld hiervan is te zien in figuur 18. Het stukje XML hierboven wordt door de serializer vertaald naar de diagrammen in dit figuur.

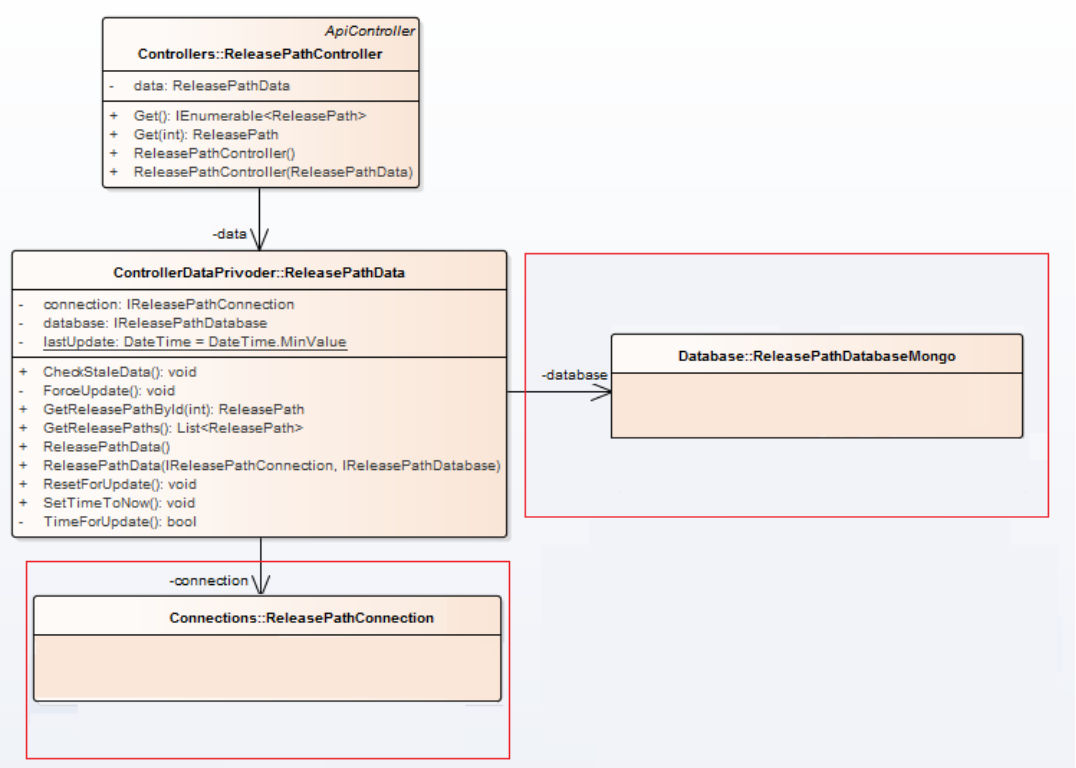
Figuur 18 XML relatie dat uit MRM afkomstig is

Als laatste had ik nog geprobeerd een methode te maken die zo generiek mogelijk was. De XmlSerializer verwacht namelijk een Type als parameter. Met dit Type weet de XmlSerializer welke attributen en relaties hij uit het XML document moet halen. In C# is het mogelijk gebruik te maken van Generics. Generics maken het mogelijk om methoden en klassen te maken die niet vast staan aan een bepaalde Type. De return Type of parameters staan dus niet vast voor een bepaalde klasse. Om Generics toe te passen in een methode maak je gebruik van "<T>" na de methode naam. Deze "T" kan vervolgens gebruikt worden in de return type of in de parameterlijst. Om mijn generieke methode te maken heb ik gebruik gemaakt van Generics. In mijn geval was het alleen nodig om de "T" te gebruiken als return Type. Mijn methode ziet er als volgt uit:

```
private T DeserializeDataFromRequest<T>(HttpWebRequest request)
{
	Try
	{
		HttpWebResponse response = request.GetResponse() as HttpWebResponse;
		XmlSerializer serializer = new XmlSerializer(typeof(T));
		return (T)serializer.Deserialize(response.GetResponseStream());
	}
	catch(Exception e){ return default(T); }
}
```

Klasse diagram ReleasePath – Webapi

In figuur 19 is de initiële klasse diagram te zien. In de klasse zijn de methoden van de database klasse en MRM verbinding wegelaten voor de duidelijkheid. Nadat ik mijn analyse had uitgevoerd, was ik ervan bewust dat de verbinding van MRM op elk moment kon wijzigen door een update. Verder was het ook duidelijk dat de database verbinding ook gewijzigd kon worden. De twee klassen in het rood aangegeven in figuur 19 konden in de toekomst dus mogelijk aangepast worden. Met de implementatie van figuur 19 kost het uitvoeren van wijzigingen veel moeite, omdat er geen abstractie is tussen die klassen. De klasse ReleaseDataProvider is dus te zwaar gekoppeld aan twee klassen in het rood gemarkeerd. Om ervoor te zorgen dat de applicatie niet veel aangepast moet worden bij een wijziging had ik besloten om een aantal design patterns te overwegen. Voor dit probleem had ik een Factory-method en Dependancy Injection overwogen.



Figuur 19 oude situatie ReleasePathData

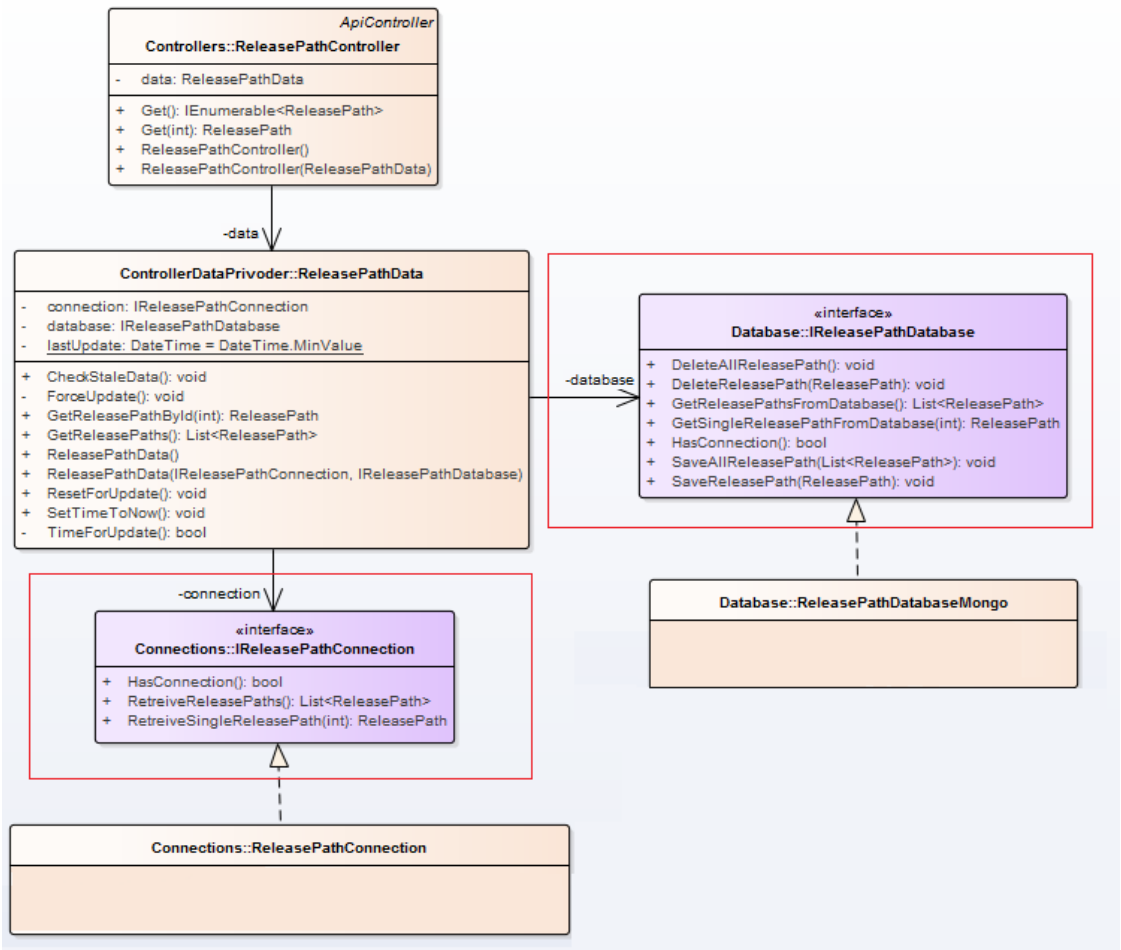
De Factory-method pattern zorgt ervoor dat het mogelijk is om objecten te creëren zonder dat de exacte klasse van het object gespecificeerd wordt[RF09]. De creatie van het verwachte object wordt gedaan door een Factory-object die instanties van het verwachte object kan creëren. Met deze oplossing is het mogelijk om de klasse ReleasePathData een Factory-method te laten aanspreken die de benodigde ReleasePathConnection object creëert. De ReleasePathConnection wordt vervolgens een abstracte klasse of interface. De subklasse of realisatie van

ReleasePathConnection kunnen dan concrete implementaties bevatten van de verschillende implementaties van MRM. Deze concrete implementaties worden door de Factory-method gecreëerd. Dit kan ook gedaan worden voor de ReleasePathDatabase klasse.

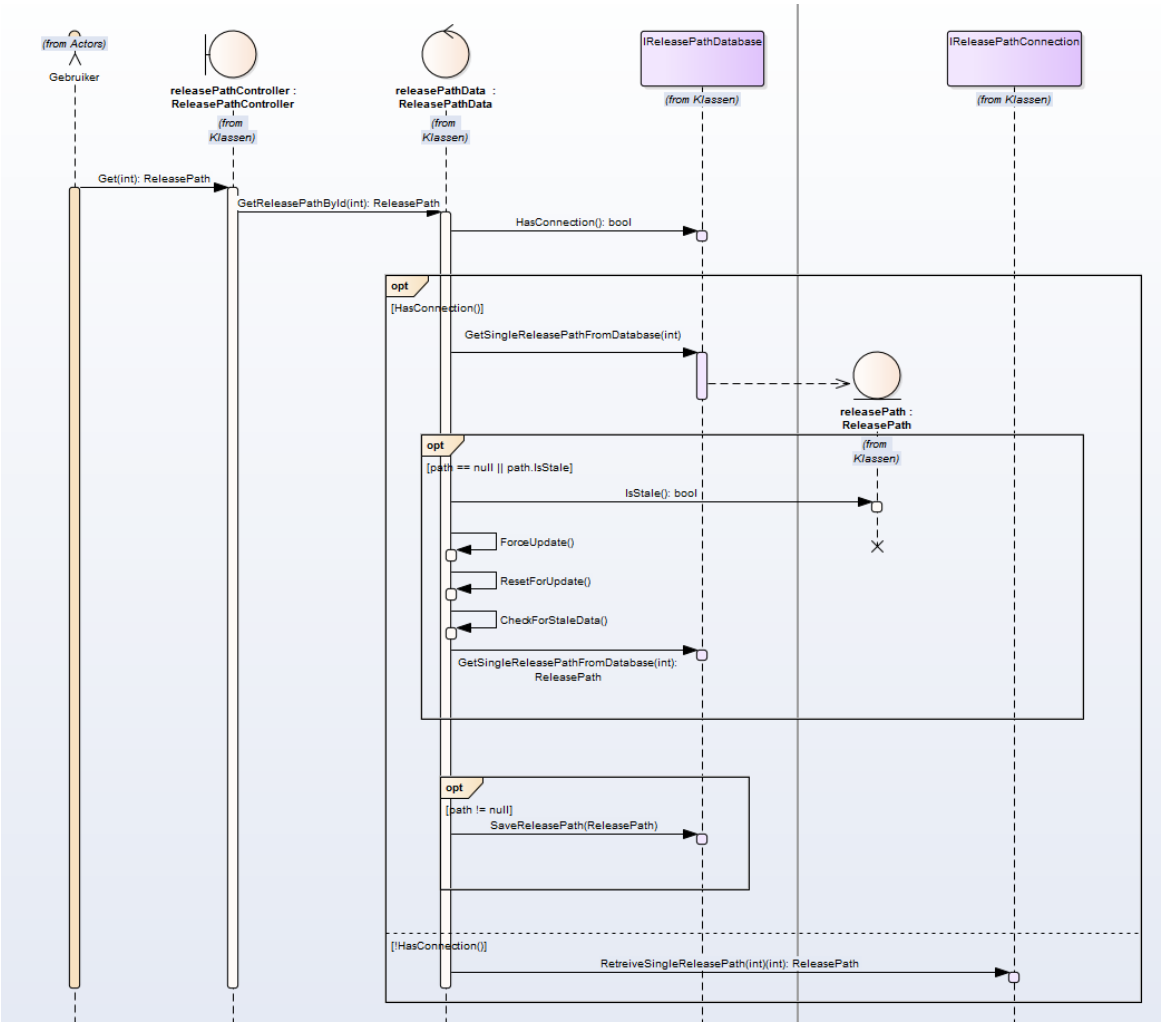
De volgende optie die ik had overwogen was Dependency Injection. Dependency Injection is het voorkomen van zware koppeling tussen klassen[RF09]. Dit wordt gedaan door de relatie tussen twee klassen te vervangen met een relatie naar een interface. Vervolgens kunnen er verschillende implementaties van de interface uitgewisseld worden zonder dat er veel code gewijzigd moet worden. In dit geval krijgt de klasse ReleasePathController een relatie met een IReleasePathConnection interface. Vervolgens krijgen we een nieuwe klasse ReleasePathConnection die alle methoden van de interface realiseert. De klasse ReleasePathController maakt nu gebruik van een van de realisaties van de interface IReleasePathConnection. In de toekomst moet er alleen een nieuwe realisatie van de interface gemaakt worden en kan de oude vervangen worden met de nieuwe.

Uiteindelijk heb ik besloten om gebruik te maken van Dependency Injection. Het probleem met de Factory-Method pattern is dat er veel complexiteit bij komt. Per nieuwe implementatie moet er een Factory-Method gemaakt worden. Met Dependency Injection blijft de implementatie simpel en gemakkelijk uit te breiden.

Met deze oplossing kon ik voorkomen dat de applicatie sterk afhankelijk werd van de verbinding met MRM. Als er nu een nieuwe versie van MRM wordt uitgebracht moet deze interface alleen geïmplementeerd te worden. De uiteindelijke implementatie is te zien in figuur 20. Hierin is met rood aangegeven waar de interfaces staan. Verder zijn de methoden en attributen van de implementaties niet weergegeven om ruimte te besparen.



Figuur 20 uiteindelijke implementatie van interfaces

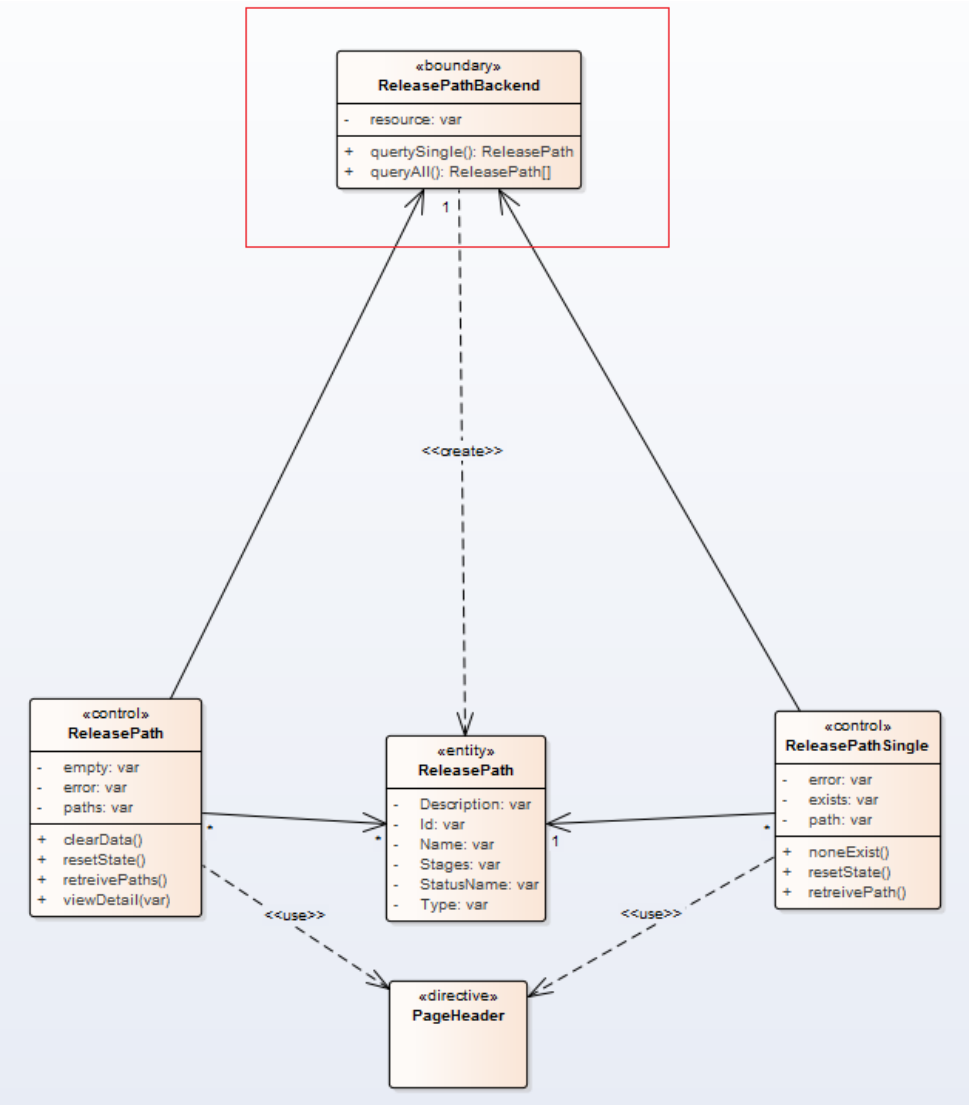


Figuur 21 implementatie van ophalen één Release Path tussen gebruiker en interface

In figuur 21 laat ik ook zien hoe de interactie tussen de klassen van figuur 20 verloopt. Bij het ophalen van ReleasePaths aan de hand van een Id wordt er uiteindelijk gecommuniceerd met de abstractielagen IReleasePathDatabase en IReleasePathConnection. Deze abstractie bestaat om de applicatie later te kunnen uitbreiden. Nieuwe implementaties van de interface zullen hiervoor gebruikt worden.

Services – AngularJS client

Tijdens het realiseren van de AngularJS front-end heb ik besloten gebruik te maken van services. Services in AngularJS zijn Singleton objecten die vanuit je gehele project gebruikt kunnen worden. Een Singleton object is een object waar er alleen één instantie in de gehele applicatie van bestaat. Wanneer AngularJS een service aanmaakt injecteert hij het in al je instanties waar je gebruik van de Service wilt maken. De beslissing om een service te gebruiken heb ik gedaan, omdat bepaalde functionaliteiten vaker gebruikt werden in de applicatie. Het ophalen van een specifieke ReleasePath moest op meerdere locaties gedaan worden. De klasse diagram van de AngularJS code is te zien in figuur 22. Normaal worden er geen UML diagrammen gemaakt voor Javascript, maar om mijn applicatie te verduidelijken heb ik dat wel gedaan.



Figuur 22 AngularJS code met Service

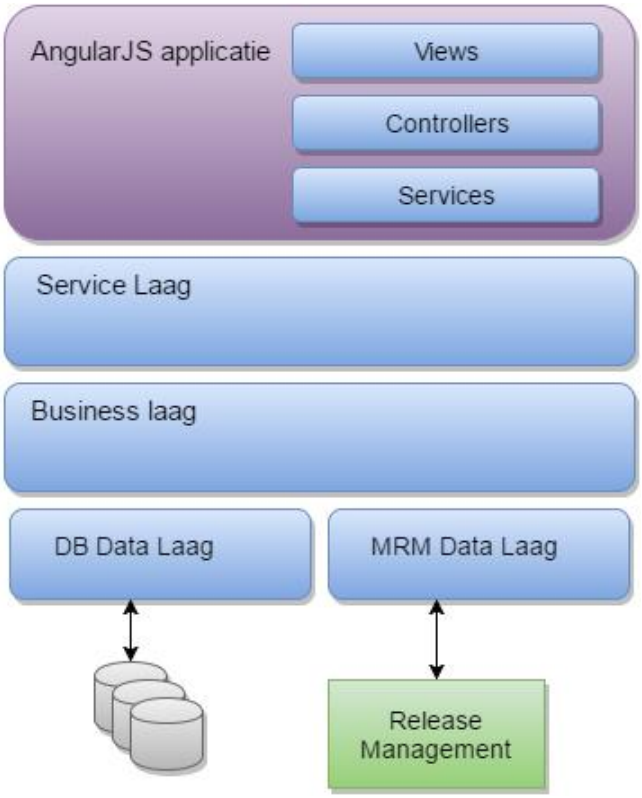
Cache database – Webapi
Tijdens mijn analyse merkte ik snel dat het ophalen van grote hoeveelheden data met de MRM API lang duurde. Bij het ophalen van alle ReleasePaths moest ik namelijk eerst VNext Paths ophalen en vervolgens alle AgentBased Paths. Hierna moest ik voor elke Path nog meer informatie ophalen met andere requests. Het probleem hiermee is dat als er meer dan één gebruiker hetzelfde informatie wilt zien de MRM server snel belast kan worden. Deze belasting op de MRM server is ook niet nodig, omdat er niet snel wijzigingen zijn in de applicatie. Wegens dit probleem had ik besloten een cache database te maken. Met een cache database zijn er diverse concurrency problemen die kunnen ontstaan. Het is namelijk mogelijk dat twee mensen tegelijk de cache proberen te verversen en daardoor twee keer de data wordt opgeslagen. Ik was mij wel bewust van deze problemen en wilde dit in toekomstige sprints oplossen.

Aan het begin van mijn opdracht was er gezegd dat dit SQL gebruikt kon gebruiken voor eventuele opslag van gegevens. In sprint 1 had ik besloten om gebruik te maken van MongoDB als cache database. Om MongoDB in C# te integreren heb ik de officiële MongoDB driver gebruikt. Verder heb ik queries uitgevoerd met Linq. De beslissing om MongoDB te gebruiken had ik genomen, omdat bij een SQL database de structuur van de gegevens niet gemakkelijk te wijzigen is. Dit betekent dat bij de kleinste wijzigingen in structuur, de database aangepast moet worden. Het nadeel hiervan is dat je veel tijd kwijt bent aan het wijzigen van de database.

Het voordeel van het gebruik van MongoDB is dat de structuur niet hard vast zit. Bij het wijzigen van de database of de structuur van de entiteiten hoeven er geen aanpassingen uitgevoerd te worden op de database. Dit komt omdat in MongoDB er geen vaste schema is vastgelegd. Verder is het ook niet nodig om documenten zoals een relationele implementatie model bij te houden. Aangezien de hoeveelheden data die gecached moeten worden niet groot zijn, is performance niet meegenomen in de overweging.

5.2.3 Architectuur van de Webapi en AngularJS

Het systeem dat ik ga ontwerpen bestaat uit meerdere lagen. Met een lagen model zal er een duidelijk onderscheid zijn van verantwoordelijkheden in de applicatie. De lagen bestaan uit een service laag, business laag en verschillende data lagen. Alle delen die met de service laag willen communiceren, zullen dit moeten doen door middel van het HTTP protocol. Dit is nodig voor een AngularJS applicatie. Dit protocol wordt gebruikt voor het versturen van een verzoek om data op te halen. Indien dit verzoek correct is verstuurd zal de service laag de gewenste gegevens terugsturen. De API die ik ga realiseren communiceert altijd over HTTP. Bij het uitvoeren van een HTTP request moet er altijd een method worden meegegeven. Deze methods bestaan uit GET, POST, PUT en DELETE. De GET wordt gebruikt bij het ophalen van gegevens. De POST, PUT en DELETE worden gebruikt om wijzigingen uit te voeren op de gegevens. Aangezien het systeem alleen gegevens toont wordt alleen de GET method ondersteund. De verbinding met Release Management wordt ook gedaan door middel van het HTTP protocol. Elke request die naar Release Management gaat wordt beantwoord met XML gegevens.



Figuur 23 Lagen model met componenten erin

De AngularJS applicatie wordt ook verdeeld in een aantal lagen. Deze verdeling wordt deels gemaakt omdat AngularJS al werkt met een controllers en views. De service laag is erbij gezet om ervoor te zorgen dat alle communicatie met de API

op één locatie zit. Hiermee is de communicatie van de AngularJS applicatie gemakkelijk uit te wisselen.
De verbinding met de database server wordt gedaan over TCP/IP. Momenteel gaat dit om de MongoDB implementatie. Het is niet noodzakelijk dat de database op een andere server is geïnstalleerd. Deze kan ook geïnstalleerd zijn op de server waar het systeem op zit.

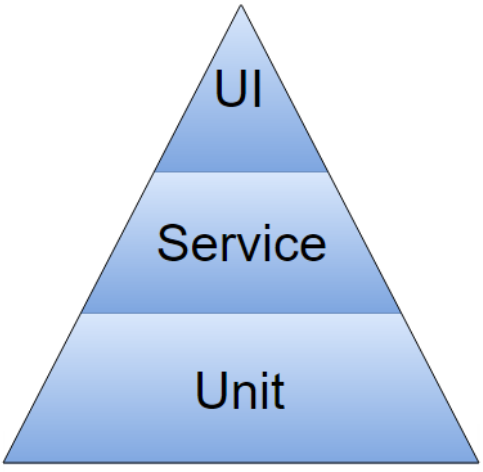
Als laatste had ik in deze sprint ook aandacht besteed aan het analyseren van de architectuur van de applicatie. De applicatie die ik heb gebouwd maakt gebruik van een lagen model. Deze lagen model is te zien in figuur 23. Deze lagen zorgen voor een scheiding tussen bepaalde functionele elementen. Het is namelijk niet gewenst dat het dashboard element ook direct de database kan aanspreken zonder dat de business laag hier van af weet. Als dit mocht dan was er geen manier om ervoor te zorgen dat het informatie die opgeslagen was correct was.

Met dit lagen model encapsuleren wij bepaalde gegevens en bieden we een interface aan om deze gegevens op te halen. Bij het wijzigen van één van de lagen is het ook niet noodzakelijk om het systeem helemaal aan te passen. Zolang de nieuwe laag dezelfde interface aanbiedt zal het systeem blijven werken zoals verwacht.

In de toekomst is het ook mogelijk nieuwe clients toe te voegen aan de back-end van het dashboard. Als deze kunnen communiceren met de service laag is het mogelijk nieuwe clients toe te voegen aan het systeem zonder dat er wijzigingen aangebracht moeten worden. De service laag zal worden gerealiseerd door gebruik te maken van WebApi. Dit is een framework van Microsoft om API's mee te schrijven. Het dashboard moet op verschillende toestellen bekeken kunnen worden. De meest gebruikte toestellen voor deze applicatie zijn iPad'-s en Internet Explorer browsers op beeldschermen. Momenteel is ervoor gekozen om een AngularJS applicatie te schrijven die op beide toestellen goed te zien is. Voor mijn complete architectuur beslissingen zie bijlage IV.

5.2.4 Testen

Voor het testen van een applicatie kan er onderscheid worden gemaakt tussen unit-, service- en user interface (UI) tests. Deze vormen van testen kunnen worden weergegeven in een piramide. In figuur 24 is dit weergegeven. Hoe hoger je in de piramide komt hoe langer het maken van de test duurt. Een UI-test duurt veel langer om te maken of herschrijven dan een Unit test. De vorm van de piramide geeft de verhouding van hoeveelheid aan tussen de tests. Een applicatie moet voornamelijk getest worden met Unit tests. Vervolgens wordt er minder tijd besteed aan het maken van Service tests. Uiteindelijk worden er weinig UI-tests geschreven. De testpiramide beschrijft dus hoe er meer tijd besteed moet worden aan het schrijven van unit tests en minder aan UI tests. Het testen vanuit de UI kan meer tijd kosten en is soms niet te automatiseren. Hierdoor is een programmeur veel tijd kwijt aan het UI-testen.



Figuur 24 testpiramide

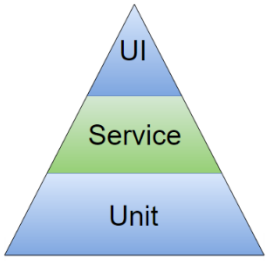
Voor de opdrachtgever is het van belang dat alle tests geautomatiseerd uit te voeren zijn. Dit is nodig om de applicatie met een "continous integration and delivery" tool zoals MRM uit te rollen. Voor alle drie lagen worden tools gekozen waarmee snel geautomatiseerde tests uitgevoerd kunnen worden. Verder wordt er per laag een korte beschrijving gegeven wat het is en hoe het uitgevoerd wordt.

Test driven development
Gedurende het project heb ik gebruik gemaakt van TDD. TDD gaat ervan uit dat de programmeur eerst testen opstelt die falen en vervolgens pas functionaliteit gaat toevoegen. Het proces van testen gaat als volgt:

- Test maken.
- Test uitvoeren om te kijken dat het faalt.
- Code schrijven totdat de test slaagt.
- Code van de test en functionaliteit refactoren.
- Beginnen bij de eerste stap totdat alle functionaliteit af is.

SpecFlow
Gherkin is een taal waarmee een tester of productowner test scenario's kan opschrijven. Deze scenario's zijn gemakkelijk te lezen en begrijpen vanuit de business, omdat er geen technische aspecten aan verbonden zijn. De Gherkin taal kan door bepaalde tools worden vertaald naar geautomatiseerde tests. De tool die ik hiervoor gebruik heet SpecFlow. SpecFlow genereert de structuur van de unit tests. Nadat de structuur is gegenereerd moet de logica nog geïmplementeerd worden. Vervolgens is het mogelijk om je specificaties aan te passen of nog meer specificaties toe te voegen. SpecFlow gebruikt dezelfde structuur en logica die je hebt geïmplementeerd en kan daarom gemakkelijk nieuwe tests genereren. Ik zal SpecFlow gebruiken tijdens mijn afstuderen, omdat Info Support hiermee werkt.

Webapi Service (Integratie) test
Een service test is een test die in aanraking komt met externe systemen tijdens het uitvoeren van de test. Bij het testen van een functionaliteit die bijvoorbeeld een verbinding met je database of filesysteem maakt en gegevens ophaalt is een service test. Bij het slagen van een verbinding of het ophalen van data is een service test eigenlijk al geslaagd. Het gaat dus niet om de juistheid van je gegevens of van de logica dat erop wordt uitgevoerd, maar om de verbinding en foutgevoeligheid.



Figuur 25 service test

Alle service test zijn alleen uitgevoerd op de webapi en niet op de AngularJS code. Voor het uitvoeren van service tests wordt er een combinatie van SpecFlow en NUnit gebruikt.

Door een user story als voorbeeld te nemen zal ik verduidelijken hoe ik testen heb geïmplementeerd. Aangezien ik mijn project met TDD heb uitgevoerd zal ik ook beschrijven hoe dit is verlopen. Voordat ik ging programmeren had ik twee verschillende test projecten aangemaakt. Eén van deze projecten was voor het uitvoeren van mijn integratie tests en de andere was voor mijn unit tests.

Voor het uitvoeren van mijn unit tests was ik begonnen met het opstellen van SpecFlow scenario's. Deze scenario's heb ik opgesteld aan de hand van de user story en acceptatie criteria. De scenario's van US1 zijn hieronder te zien:

```
Feature: ReleasePathConnections
       In order to get an overview of all ReleasePaths
       As a user
       I want to be able to retrieve all ReleasePaths from Release Management

@mytag
Scenario: Retrieve active ReleasePaths
       Given I have chosen to make a connection with Release Management
       When I press connect to ReleasePaths
       Then the result should be a connection with Release Management

Scenario: Retrieve a vNext Release Path
       Given I have a vNext ReleasePath in Release Management
       When I press select all Release Paths
       Then the result should be a vNext Release Path
```

Nadat ik deze scenario's had gemaakt moest ik alle stappen die in de scenario's waren beschreven koppelen aan code die kon worden uitgevoerd. Visual Studio genereert zelf een bestand waar je alle stappen kan koppelen aan code. Een voorbeeld van deze code is hieronder te zien.

```
[Given(@"I have chosen to make a connection with Release Management")]
public void GivenIHaveChosenToMakeAConnectionWithReleaseManagement()
{
	Requester r = new Requester("http://student1604:1000/");
	connection = new ReleasePathConnection(r);
}

[When(@"I press connect to ReleasePaths")]
public void WhenIPressConnectToReleasePaths()
{
	connectionMade = connection.HasConnection();
}

[Then(@"the result should be a connection with Release Management")]
public void ThenTheResultShouldBeAConnectionWithReleaseManagement()
{
	Assert.IsTrue(connectionMade);
}
```

In dit stukje code zijn er drie stappen gekoppeld aan de scenario's. Nadat al deze stappen zijn ingevuld kunnen de testen worden uitgevoerd. Indien je nog meer testen wilt uitvoeren is het mogelijk om een nieuwe scenario te maken met dezelfde stappen, maar in een andere volgorde.

Het testen met SpecFlow had als nadeel dat het veel tijd in beslag nam. Daarom had ik ervoor gekozen om niet alle stukken code hiermee te testen. Om de andere functionaliteiten te realiseren met TDD had ik gebruik gemaakt van NUnit. In NUnit had ik ook getest, zoals TDD dit voorschrijft.

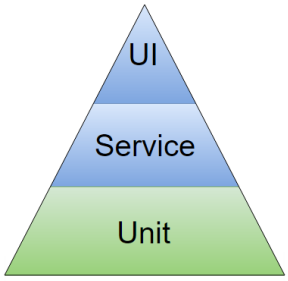
Webapi Unit test

Met TDD is het ook noodzakelijk unit tests uit te voeren. Bij functionaliteiten die geen aanraking hadden met externe systemen had ik unit tests geschreven. Het schrijven van deze unit tests heb ik ook gedaan door eerst de test op te stellen en vervolgens pas de functionaliteit die daarbij hoorde op te stellen. In sommige gevallen hadden moest ik functionaliteiten testen die uiteindelijk in aanraking kwamen met externe systemen zoals MRM. Aangezien ik eerst mijn testen opstelde kon ik in mijn code hier rekening mee houden. Een voorbeeld hiervan is het testen van de ReleaseController klasse. De ReleaseController haalt uiteindelijk gegevens uit MRM.

Bij het uitvoeren van een unit test wil je niet dat externe componenten invloed kunnen hebben op de test. Bijvoorbeeld de beschikbaarheid van een database kan een test wel of niet laten falen. Daarom is het nodig om bepaalde componenten van je applicatie te mocken. Bij het mocken van deze componenten laat je de applicatie denken dat er wel een verbinding is en voorzie je de applicatie ook van test gegevens. Zoals beschreven in de paragraaf Ontwerpen en Realiseren heb ik gebruik gemaakt van 'dependency injection'. Met deze 'dependency injection' is het nu mogelijk zonder de verbinding van MRM te vervangen met een Mock object. In het voorbeeld hieronder is te zien hoe de klasse ReleaseController een Mock object meekrijgt:

```
private Mock<ReleasePathData> mockConnection;

public ReleaseControllerUnit()
{
	mockConnection = new Mock<ReleasePathData>(null, null);
	controller = new ReleasePathController(mockConnection.Object);
}
```



Figuur 26 Unit tests

Vervolgens is het mogelijk om te bepalen wat het Mock object moet returnen als zijn methoden worden aangeroepen. Dit is te zien in het voorbeeld hieronder:

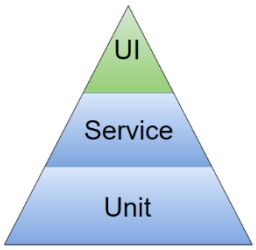
```
[TestMethod]
public void
GivenIHaveReleasesPaths_WhenIWantToViewThem_ThenThePathsShouldBeInOrderFromMostRecentToLat
est()
{
	mockConnection.Setup(m => m.GetAllReleasePaths(1)).Returns(() => paths);

	List<ReleasePath> paths = controller.Get(1).ToList();

	Assert.AreEqual(paths [0].Id, 3);
	Assert.AreEqual(paths [1].Id, 6);
	Assert.AreEqual(paths [2].Id, 1);
}
```

AngularJS UI test

De user interface tests gaan vooral over het testen van wat de gebruiker ziet op het scherm. Aangezien ik een AngularJS applicatie ga realiseren, maak ik gebruik van de tooling die door AngularJS zelf wordt aangeraden. Voor het testen van de user interface heb ik gebruik gemaakt van Protractor als framework. Met Protractor is het mogelijk om tests te schrijven die interactie hebben met een AngularJS website.

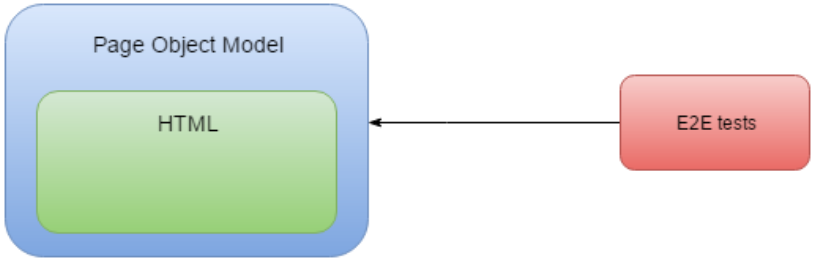


Figuur 27 UI tests

De user interface tests zullen worden gemaakt met SpecFlow en worden aangestuurd door Protractor. Met deze combinatie is het mogelijk de AngularJS website te testen.

Voor het testen van de UI had ik gebruik gemaakt van Protractor. Alle Protractor tests die ik in deze sprint had geschreven waren uitgevoerd door gebruik te maken van NodeJS. Ik had hiervoor gekozen, omdat dit de standaard manier was om Protractor tests te runnen volgens AngularJS. Het was dus niet mogelijk om deze tests uit te voeren binnen Visual Studio.

Alle UI tests die ik had geschreven spraken de HTML pagina eerst direct aan. Dit betekende dat deze tests sterk afhankelijk waren van de HTML. Bij de kleinste wijziging in de HTML was het mogelijk dat verschillende tests herschreven moesten worden. Om ervoor te zorgen dat ik nog wijzigingen kon uitvoeren op de HTML zonder grote wijzigingen te moeten doorvoeren ging ik op zoek naar een oplossing. Mijn eerste gedachte was om de HTML te encapsuleren en ervoor te zorgen dat mijn tests geen directe contact hadden met de HTML. Mijn oplossing voor dit probleem stond beschreven onder het Page Object Model. Met het Page Object Model wordt de HTML pagina geëncapsuleerd. De UI-tests (E2E tests) communiceert nooit direct met HTML, maar met het Page Object Model. Bij wijzigingen in het HTML moet alleen het Page Object Model gewijzigd te worden. In figuur 28 is dit in een figuur weergegeven.



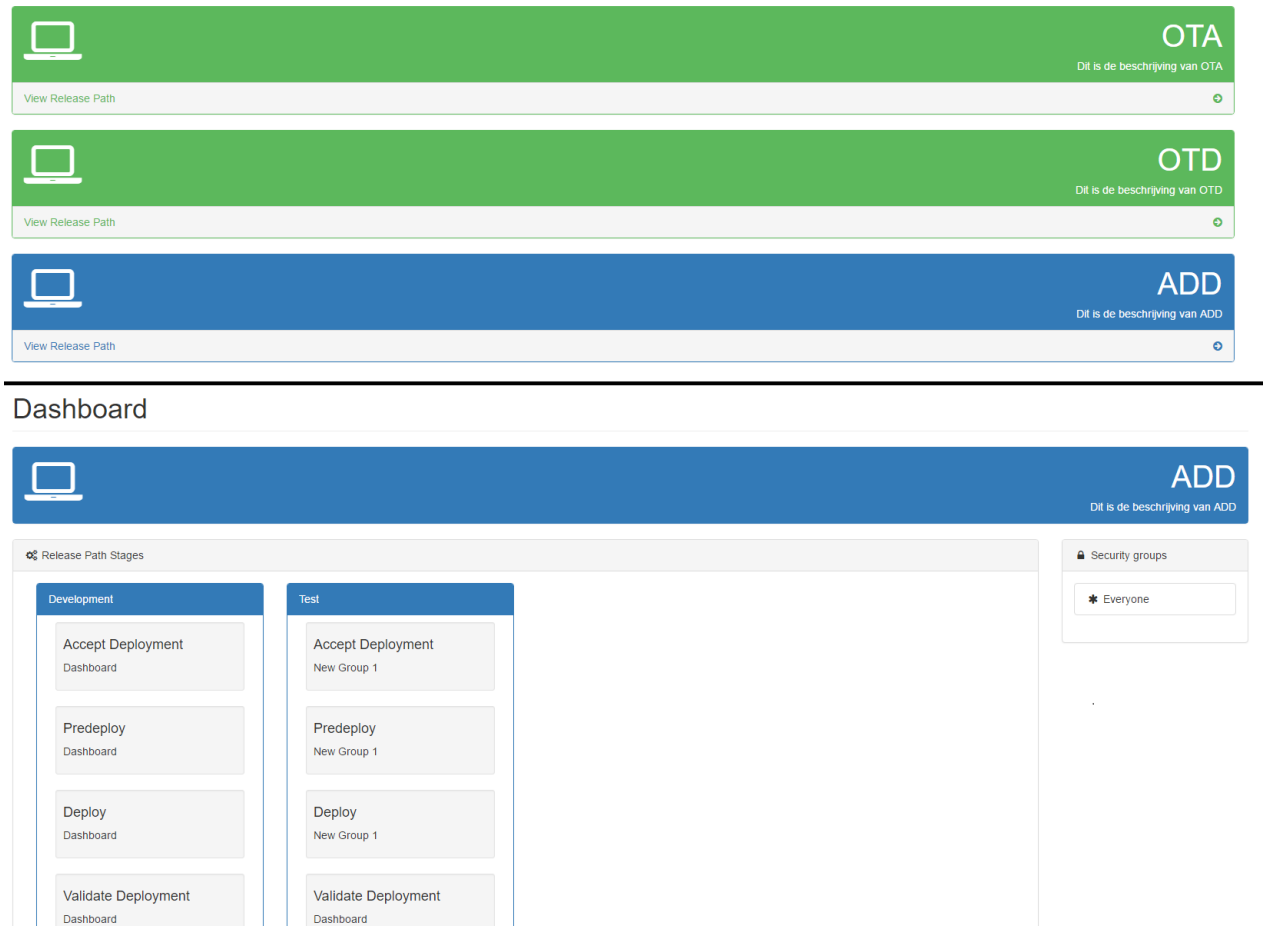
Figuur 28 page object model weergegeven

5.2.5 Sprint review en retrospect

Op het einde van de sprint was het mij gelukt om alle functionaliteiten te realiseren. Deze sprint waren er veel taken die ik had uitgevoerd. Aangezien het mijn eerste sprint was nadat ik mijn analyse had uitgevoerd wilde ik zoveel mogelijk programmeren.

Tijdens de sprint review had ik de nieuwe functionaliteit aan de opdrachtgever gepresenteerd. Over de functionaliteit was hij tevreden. Voor de verslaglegging van mijn tests had ik besloten elke sprint mijn tests te laten draaien. Hiermee kon de opdrachtgever gelijk zien welke test slaagden en welke niet. Over de UI tests met Protractor twijfelde de opdrachtgever. De tests moesten namelijk uitgevoerd worden binnen Visual Studio. Er werd mij verzocht om uit te zoeken of ik, mogelijk met Coded Ui, alles binnen Visual Studio kon testen.

Resultaten
De resultaten van deze sprint zijn te zien in figuur 29. In het eerste scherm is te zien hoe alle ReleasePaths worden getoond. Nadat er een selectie is gemaakt wordt het tweede scherm getoond met daarin meer informatie over die ReleasePath.



Figuur 29 schermprintjes van gerealiseerde functionaliteit

5.3 Sprint 2 – Release informatie

In deze paragraaf beschrijf ik de tweede sprint van mijn project. Eerst zal ik beginnen met het beschrijven van mijn planning en analyse van de sprint. Vervolgens zal ik uitleggen welke beslissingen ik heb genomen tijdens het ontwerpen en realiseren. Als laatste, beschrijf ik welke bijzonderheden ik heb ervaren tijden het testen.

5.3.1 Planning en Analyse

Aan het begin van de sprint was het belangrijk dat ik wist hoeveel een story point eigenlijk waard was. Na mijn eerste sprint kon ik terug zien hoeveel tijd ik had besteed aan het uitvoeren van bepaalde taken en hoeveel story points ik daarvoor had ingeschat. Voor de eerste sprint had ik mijn werkzaamheden ingeschat op 60 story points in een sprint. Dit betekende dat ik schatte per week ongeveer 30 story aan werk te kunnen verrichten. Aan het einde van de eerste spring had ik meer tijd over. Hierdoor had ik besloten meer storypoints voor deze sprint in te plannen. Voor deze sprint had ik besloten dat ik bepaalde werkzaamheden sneller kon uitvoeren en meer tijd moest inplannen. Daarom had ik besloten 90 storypoints in te schatten.

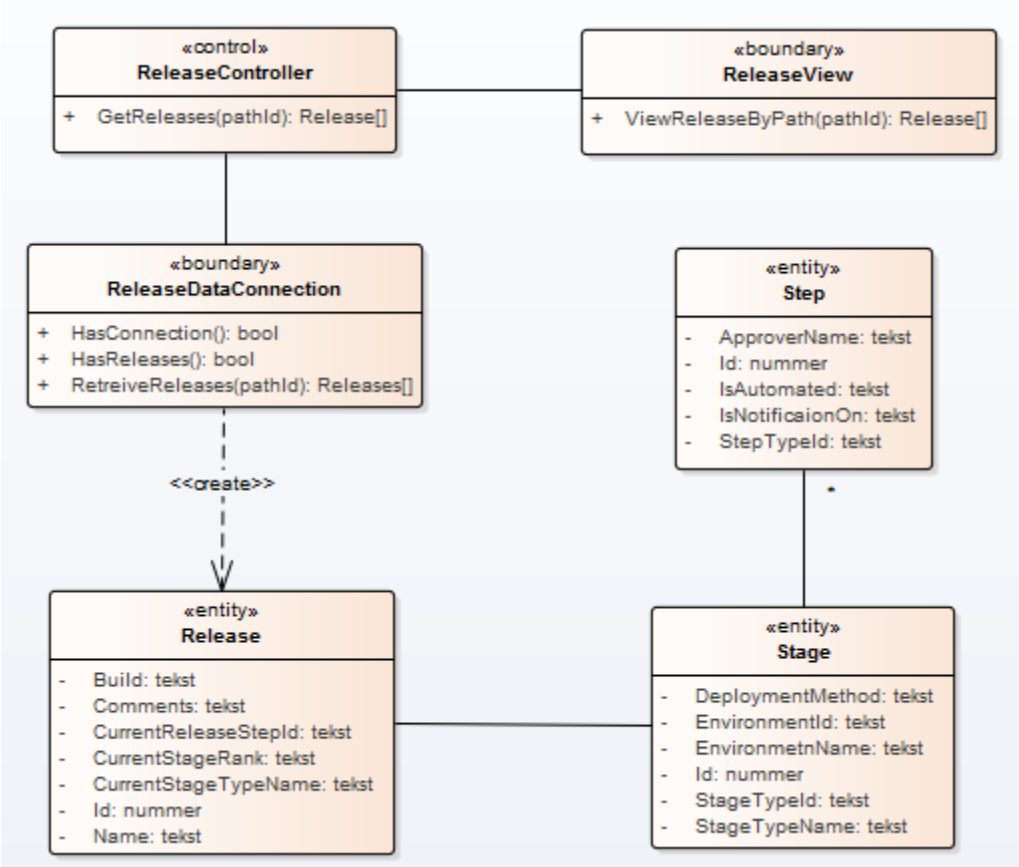
Voordat de sprint planning van deze sprint begon had ik een epic met een lijst van user stories bedacht die tijdens deze sprint uitgevoerd konden worden. Dit had ik gedaan om ervoor te zorgen dat ik een voorstel kon doen bij de opdrachtgever. Mijn voorstel was om alle Release informatie te tonen van bepaalde ReleasePaths. Verder wilde ik ook dat er inzicht was in welke ReleaseTemplates bij een bepaalde ReleasePath hoorde. De opdrachtgever was met mijn idee akkoord gegaan. De user stories die ik had ingepland voor deze sprint waren de volgende:

Nummer	Story	Prioriteit	TFS id
US3	Als gebruiker wil ik een overzicht van ReleaseTemplates die zijn gekoppeld aan een ReleasePath zodat ik kan zien welke applicaties bij welke pad horen	3	9477
US4	Als gebruiker wil ik een overzicht van alle Releases die bij één ReleaseTemplate horen, zodat ik een overzicht kan hebben welke releases zijn uitgebracht van een bepaalde Release Path	4	9255
US5	Als gebruiker wil ik een overzicht van alle Stages die bij een Release horen zodat ik kan zien welke Stages bij de Release betrokken zijn	5	9444

Na de sprint planning merkte ik dat er een user story in de backlog was die paste bij de sprint van deze week. Ik had ervoor gekozen om deze user story ook in deze sprint uit te voeren. Indien ik niet genoeg tijd had kon ik het naar de volgende sprint verplaatsen. Deze user story is de volgende:

Nummer	Story	Prioriteit	TFS id
US6	Als gebruiker wil ik een overzicht van Steps die horen bij een Stage, zodat ik duidelijk kan zien welke stappen zijn ondernomen in een bepaalde Stage	6	9445

Nadat ik alle user stories had ingevoerd ging ik verder met het toevoegen van acceptatie criteria en het verder uitwerken van mijn analyse klasse diagram. In mijn analyse klasse diagram heb ik voor mezelf en de opdrachtgever proberen duidelijk te maken welke wijzigingen en nieuwe functionaliteiten in deze sprint gerealiseerd zouden worden. Mijn analyse diagram voor deze sprint is te zien in figuur 30. Alle klassen in dit figuur zijn nieuw.



Figuur 30 analyse diagram van sprint 2

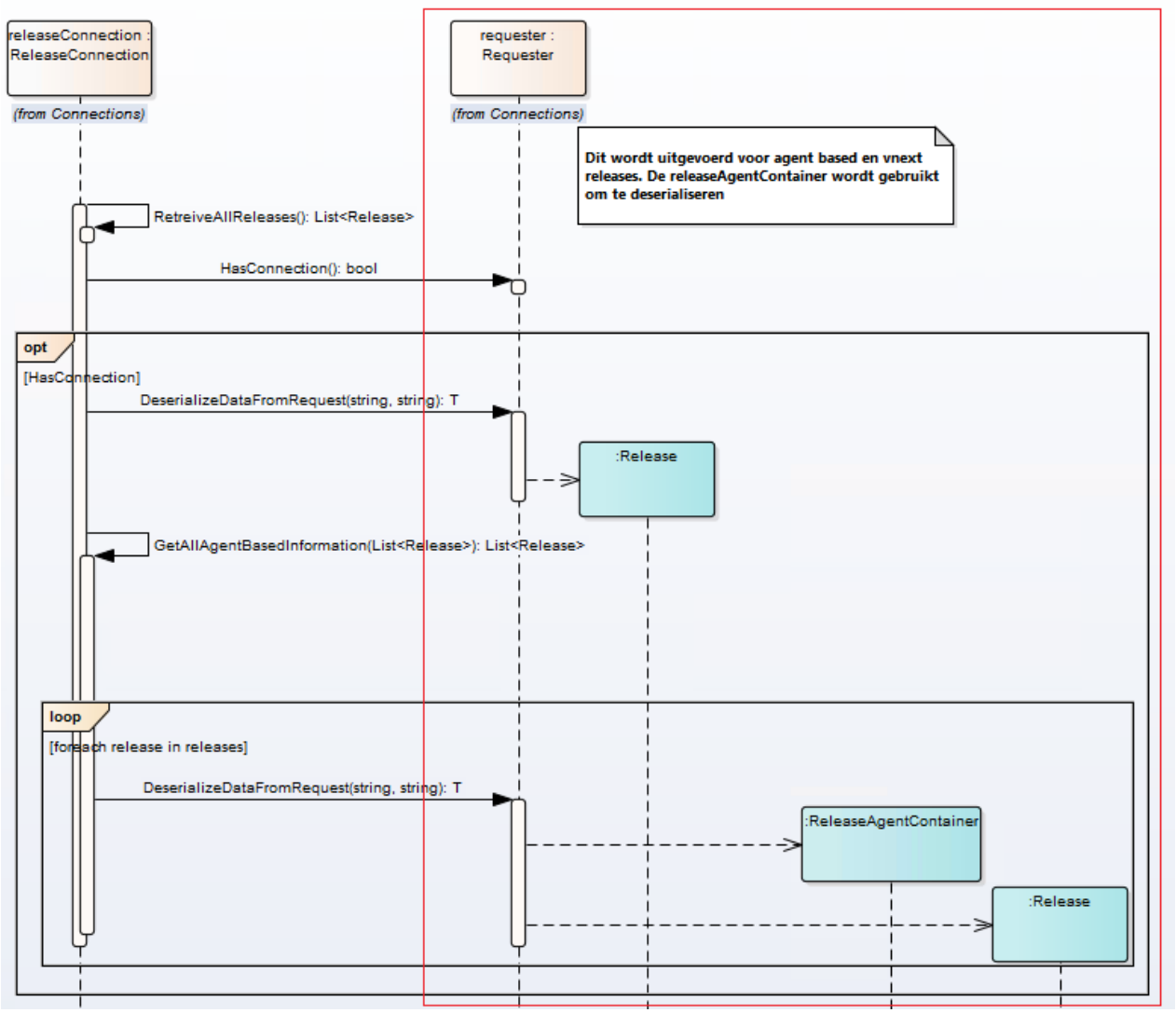
5.3.2 Ontwerpen en Realiseren

Analyseren van requests – Webapi
Toen ik begon met het ontwerpen en realiseren moest ik eerst weten welke gegevens ik uit MRM kon halen en welke requests ik daarvoor moest aanspreken. Ik was begonnen met het opstarten van MRM en Fiddler. In MRM ben ik op zoek gegaan naar de pagina's waar al het informatie over Releases staan aangegeven. Er was een verschil tussen overzichten van alle Releases van de afgelopen paar dagen en uitgebreide informatie over één Release. Verder merkte ik ook dat er verschillende requests werden uitgevoerd voor het ophalen van AgentBased Releases en vNext.

Om al de informatie van alle Releases op te halen was het nodig om eerst alle vNext en AgentBased Releases op te halen. Vervolgens kon ik met deze Releases alle details ophalen. Hiervoor had ik dus drie requests nodig. Al snel merkte ik op dat dit lang kon duren als er teveel Releases in het systeem waren. Daarom had ik ervoor gekozen om met de opdrachtgever te bespreken tot hoever de geschiedenis van Release moest gaan. De opdrachtgever had besloten om alleen de laatste 28 dagen te tonen. Om het ophalen van Releases met een bepaalde datum te realiseren ging ik kijken of de datum werd verstuurd als filter naar de API in MRM. Uit de requests kwam ik te weten dat er ook een datum wordt opgestuurd in XML formaat. Hiermee kon ik bepalen hoe recent de Releases moesten zijn. De filter data dat werd verstuurd zag er als volgt uit:

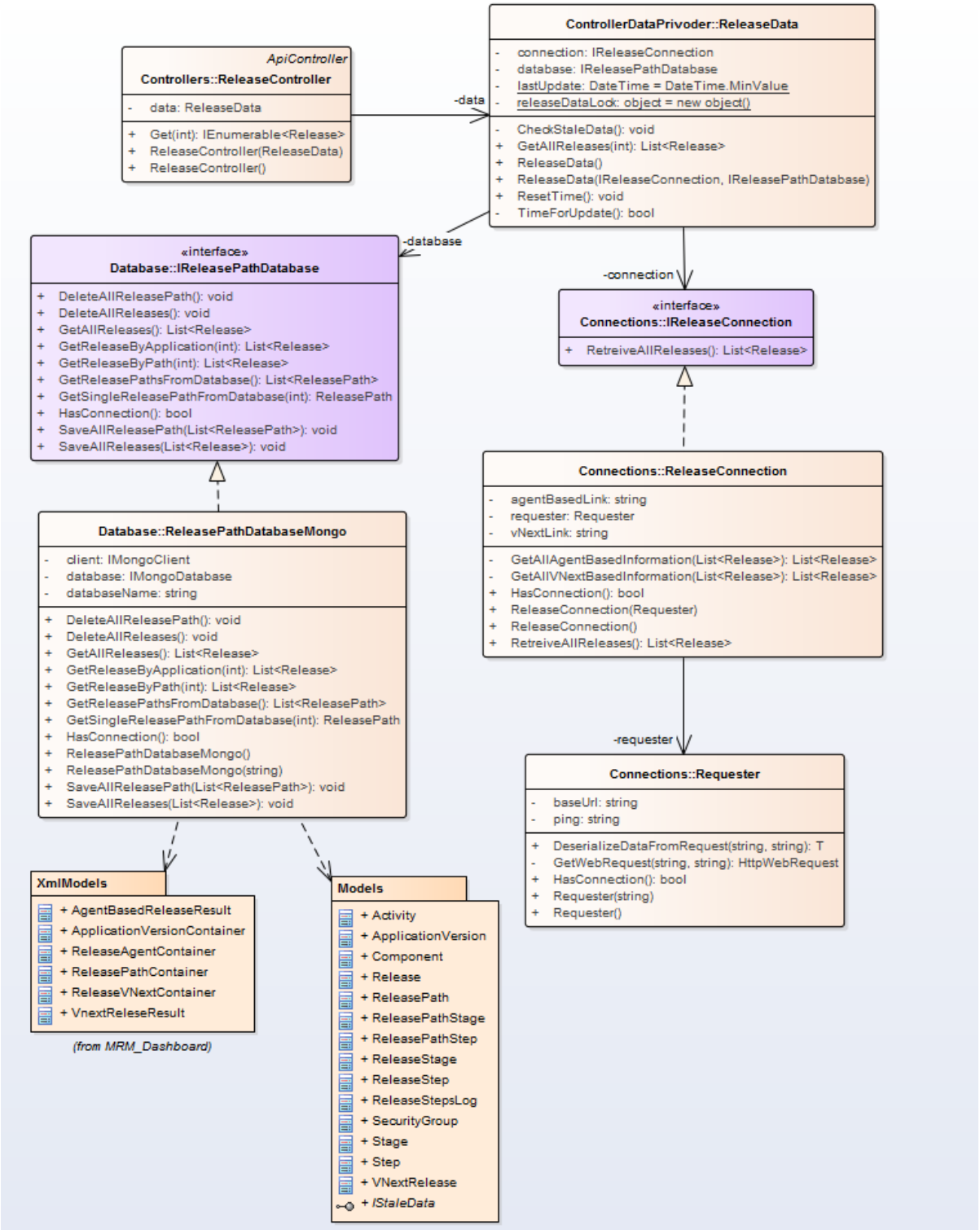
<Filter StatusId="0" **DateStatusId="6"** ApplicationVersionId="0" StageId="0" IsDeferred="0" UserId="9002" />

Ontkoppelen van uitvoeren requests – Webapi
Verder had ik in deze sprint mijn klasse diagram uitgebreid met de nieuwe functionaliteit. Al snel merkte ik dat voor het uitvoeren en deserialiseren van requests veel code gekopieerd werd. Om te voorkomen dat mijn code niet meer te onderhouden was had ik besloten een refactor slag te maken. Voor het werkelijk uitvoeren van de requests heb ik een nieuwe klasse gemaakt. Nu was het mogelijk om verschillende klassen te maken die gegevens van MRM moesten ophalen die voor het opsturen van requests allemaal gebruik maakte van één klasse. De sequentie voor het ophalen is te zien in figuur 31. In deze figuur heb ik met rood aangegeven welke nieuwe klasse erbij is gekomen.



Figuur 31 sequentie diagram van refactorslag

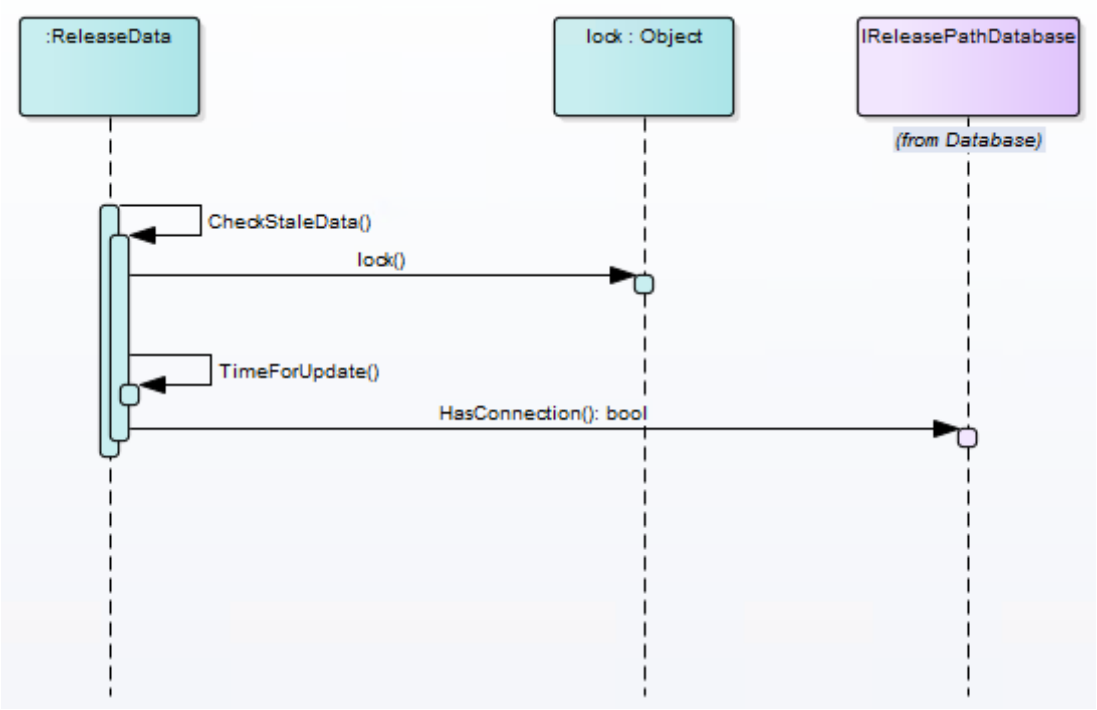
De rest van functionaliteiten van de sprint zijn terug te zien in het klasse diagram in figuur 32. Hierin is ook te zien hoe ik dezelfde manier van werken had aangehouden zoals in de vorige sprint. Voor alle communicatie met externe bronnen zoals MRM en MongoDB maakte ik gebruik van interfaces. Als laatste had ik ook gemerkt dat de database klasse teveel verantwoordelijkheden had. Dit wilde ik in de volgende sprint aanpakken.



Figuur 32 klasse diagram van gerealiseerde functionaliteit

Locken threads – Webapi
Gedurende de sprint had ik een probleem opgespoord tijdens het realiseren. Voor het ophalen van ReleasePaths en Releases werd er gekeken wanneer de informatie was opgehaald. Als de informatie oud was moest de cache worden vervangen. Het probleem was dat wanneer twee gebruikers tegelijk informatie probeerden op te halen er ook twee keer gekeken werd of er al nieuwe gegevens ingeladen moesten worden. Het was hierdoor mogelijk om twee keer dezelfde gegevens op te halen en op te slaan in de cache. Op het scherm kreeg de gebruiker dan twee keer hetzelfde te zien.

Om dit probleem op te lossen had ik besloten de methode waar alle gegevens mee worden opgehaald te locken. Het was niet meer mogelijk voor twee mensen om een controle te doen of de informatie oud was. Eén persoon moest wachten totdat de andere klaar was met deze actie. Als de eerste persoon klaar was met het controleren en ophalen van alle nieuwe gegevens uit MRM kreeg de tweede persoon de nieuwe gegevens te zien. Hoe ik het locken heb geïmplementeerd is te zien in figuur 33.

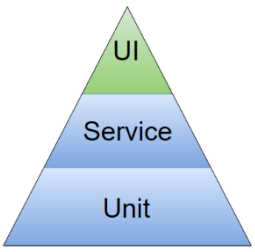


Figuur 33 sequentie diagram van locken

Bij het locken van een thread is het mogelijk er bij het optreden van een fout de lock niet meer wordt vrijgegeven. Dat betekent dat alle andere threads die dezelfde code willen uitvoeren blijven wachten. Dit heeft tot gevolg dat de applicatie niet verder kan functioneren. In C# wordt in de code `lock (lock1) { };` automatisch de lock vrijgegeven bij het optreden van een fout. Hierdoor hoeft niets geïmplementeerd te worden[FR05].

5.3.3 Testen AngularJS

Tijdens deze sprint ging ik op zoek naar een alternatief voor UI tests met Node en Protractor. Mijn opdrachtgever had aangegeven dat het mogelijk was om met behulp van CodedUI mijn applicatie te testen. Verder was ik ook naar andere mogelijkheden op zoek gegaan. Ik had een C# package gevonden die gebruik maakte van Protractor en een webdriver om alle tests uit te voeren. Met beide mogelijkheden kon ik een AngularJS website testen.



Figuur 34 UI tests

Het eerste wat ik besloot te doen was beide testtools uit proberen. Nadat ik was begonnen met CodedUI merkte ik al snel dat er een steile leercurve was. Ik moest veel vooronderzoek doen voordat ik de tool kon begrijpen. Uiteindelijk kreeg ik foutmeldingen waardoor ik geen acties op een Chrome of Internet Explorer browser kon uitvoeren. Na CodedUI ging ik verder met de Protractor package. Hier merkte ik al snel op dat het dezelfde structuur had van de Protractor van Node. Deze package was ook gericht op het testen van AngularJS websites en wist wanneer alle AngularJS bestanden waren ingeladen.

Uitgaande van mijn bevindingen heb ik de opdrachtgever geadviseerd om van Protractor gebruik te maken. Mijn opdrachtgever vond het een goede keuze. Tijdens het overzetten van alle vorige tests had ik ook besloten om gebruik te maken van de Page Object Model, die ik heb beschreven in sprint 1. De Protractor package maakt namelijk ook gebruik van HTML elementen om gegevens op te halen. Het model zag er bijna hetzelfde uit als in Node. Het was nu mogelijk om alle tests uit te voeren in Visual Studio.

De Protractor Package was goed te combineren met SpecFlow. Alle scenario's die met SpecFlow bedacht waren konden direct worden vertaald naar interacties die uitgevoerd konden worden met Protractor. Samen met de opdrachtgever had ik besloten om alle UI tests uit te voeren met SpecFlow en Protractor. Voor de integratie en unit tests werd SpecFlow niet meer gebruikt. De opdrachtgever vond het namelijk niet nodig dat de unit en integratie tests werden beschreven vanuit de gebruikers perspectief. Hieronder volgt een voorbeeld waaruit blijkt dat Protractor goed te combineren was met SpecFlow:

Eerst werden de scenario's opgesteld:

```
Feature: Release
	In order to view all releases of a release path
	As a user
	I want overviews of releases

@view all releases
Scenario: view releases of a release path that does not exist
	Given I do not have releases of a release path
	When I choose to view the releases of the release path that doesnt exist
	Then I want a warning that the release path doesnt exist
```

Vervolgens konden de scenario's worden gekoppeld aan stukken code:

```
[Given(@"I do not have releases of a release path")]
public void GivenIDoNotHaveReleasesOfAReleasePath()
{
	ngDriver = new NgWebDriver(wrapper.GetChromeDriver());
}

[When(@"I choose to view the releases of the release path that doesnt exist")]
public void WhenIChooseToViewTheReleasesOfTheReleasePathThatDoesntExist()
{
	ngDriver.Url = wrapper.GetReleasesOfPathThatDoesNotExist();
}

[Then(@"I want a warning that the release path doesnt exist")]
public void ThenIWantAWarningThatTheReleasePathDoesntExist()
{
	NgWebElement element = .FindElement(wrapper.WarningReleasePathDoesNotExist());
	Assert.IsTrue(element.Displayed);
}
```

Als laatste werd het Page Object Model nog overgezet:

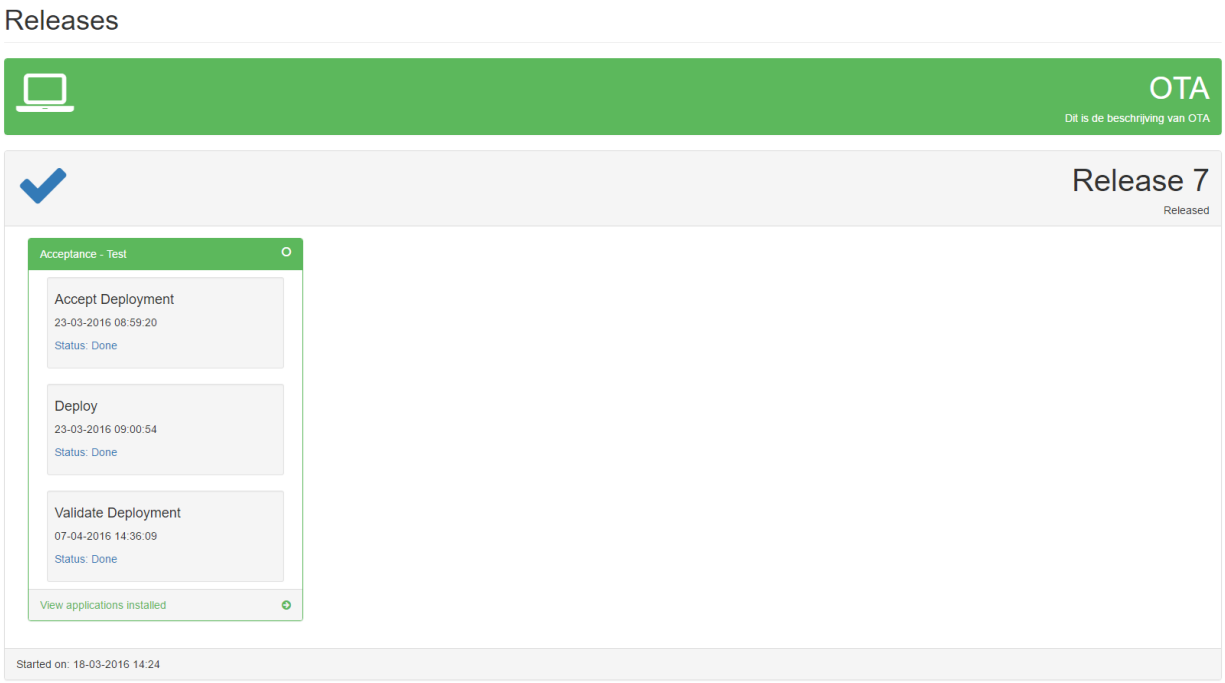
```
internal string GetReleasesOfPathThatDoesNotExist()
{
	return "http://localhost:24065/#/releasepath/500/releases";
}

internal By WarningReleasePathDoesNotExist()
{
	return By.CssSelector(".releasepath-warning");
}
```

5.3.4 Sprint review en retrospect

Nadat de sprint was afgerond had ik een sprint review en een sprint retrospect met de opdrachtgever. Tijdens de sprint review had ik de opdrachtgever alle nieuwe functionaliteiten laten zien. Met de resultaten was de opdrachtgever tevreden. In dit gesprek had ik ook aangegeven dat er een aantal stukken code waren die ik nog wilde aanpassen. Dit ging voornamelijk over de database verbinding.

Resultaten
Het resultaat van deze sprint is te zien in figuur 35. In dit figuur is te zien hoe de gebruiker de Releases van een bepaalde ReleasePath te zien krijgt. Verder zijn alle Stages en Steps ook zichtbaar. Als laatste kan de gebruiker ook zien bij welke Stage de Release is door een icoontje rechts boven de Stage.



Figuur 35 schermprint van gerealiseerde functionaliteit

5.4 Sprint 3 – Omgevingen

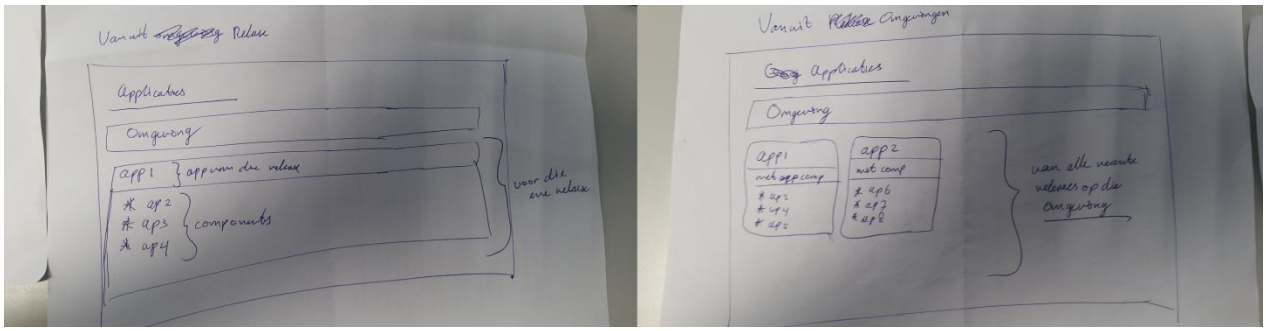
In deze paragraaf beschrijf ik de belangrijke punten van mijn derde sprint. In deze sprint stond al de informatie over de applicaties en omgevingen centraal. Als eerste zal ik beschrijven hoe ik in deze sprint mijn planning en analyse had uitgevoerd. Verder zal ik een aantal problemen beschrijven die ik was tegen gekomen tijdens mijn sprint. In Deze sprint beschrijf ik niet hoe ik heb getest, omdat het precies op dezelfde wijze was verlopen als voorgaande sprints. Er waren dus geen bijzonderheden.

5.4.1 Planning en Analyse Webap

Voordat mijn sprint begon had ik met de opdrachtgever een sprint planning. Vóór het gesprek had ik al een aantal user stories bedacht waaraan ik kon werken in deze sprint. Ik had besloten om user stories te maken waarmee ik de kern van de opdracht kon afsluiten. In de voorgaande sprints had ik gewerkt aan het visualiseren van de pipeline. In deze sprint wilde ik het mogelijk maken om verschillende versies van applicaties te tonen. Verder had ik ook besloten om een taak aan te maken waar ik de database verbinding zou herschrijven. De user stories van deze sprint waren de volgende:

Nummer	Story	Prioriteit	TFS id
US7	Als gebruiker wil ik een overzicht van Omgevingen zodat ik een keuze kan maken van welke omgeving ik een detail scherm te zien krijg	7	9446
US8	Als gebruiker wil ik een overzicht van applicaties die op een Omgeving zijn geïnstalleerd zodat ik duidelijk kan zien welke applicatie waar is geïnstalleerd	8	9447
US9	Als gebruiker wil ik een overzicht van applicaties en de versies die betrokken waren bij een omgeving van een bepaalde release zodat ik inzicht kan krijgen in welke applicaties hier bij betrokken waren	9	9831

Nadat ik alle acceptatie criteria had opgesteld en taken had aangemaakt ging ik voorbeelden schetsen van hoe ik de functionaliteit visualiseerde. Dit had ik gedaan om aan de opdrachtgever een voorstel te doen over het uiterlijk. Samen met de opdrachtgever heb ik, aan de hand van mijn tekeningen, besloten een aantal acceptatie criteria aan te passen. De opdrachtgever wilde namelijk naar deze schermen navigeren vanuit een Stage van een Release. Mijn wireframes zijn te zien in figuur 36.



Figuur 36 wireframes van omgevingen

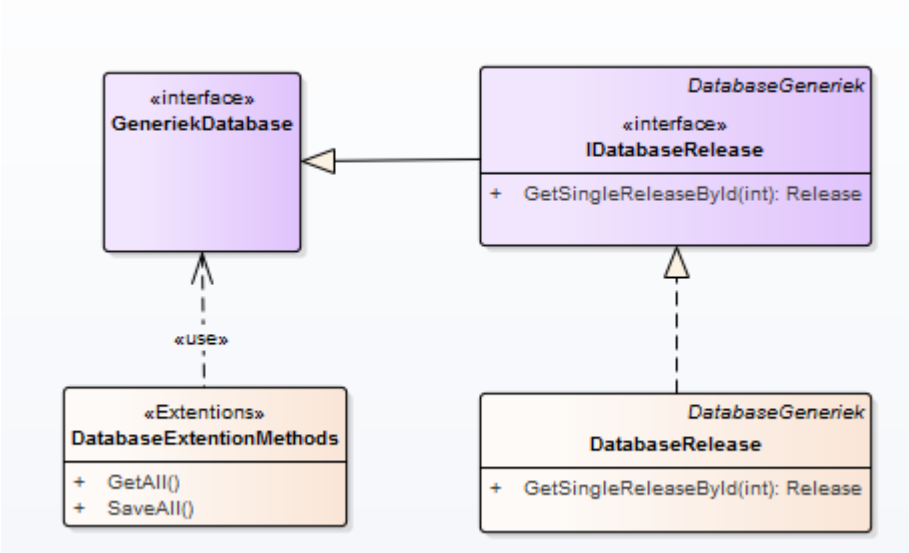
5.4.2 Ontwerpen en Realiseren

Aan het begin van mijn sprint wist ik dat ik aandacht moest besteden aan het herschrijven van mijn database verbinding van de Webapi. Het was namelijk zo dat ik in één klasse alle methoden had gezet voor de database.

Verder had ik ook geen generieke methoden gemaakt voor het opslaan of ophalen van alle gegevens. Hierdoor wilde ik een refactorslag maken. Ik wilde dit zo snel mogelijk aanpassen om ervoor te zorgen dat mijn code nog te onderhouden was. Het eerste probleem dat ik wilde oplossen was dat de verantwoordelijkheid van de klasse opgesplitst moest worden. Hiermee bedoel ik dat een klasse die verantwoordelijk is voor het ophalen van Releases niet ook verantwoordelijk is voor het ophalen van ReleasePaths. Verder moest het nog steeds mogelijk zijn om zonder teveel moeite een andere database te gebruiken. Dus de interface tussen de data laag en controllers moest blijven bestaan.

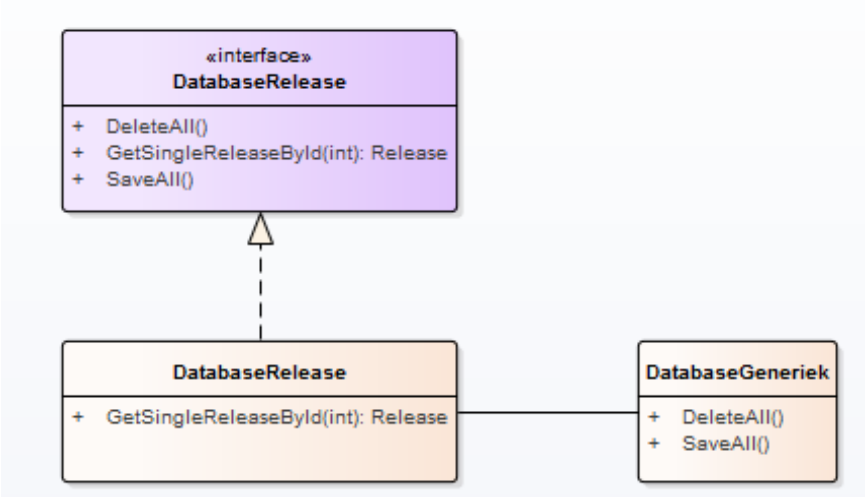
Het eerste wat ik had bedacht om dit op te lossen was het maken van verschillende interfaces waarmee controllers konden communiceren. Voor de ophaal acties van ReleasePath werd er één interface gemaakt. Verder werd er voor alle ophaal acties van een Release een andere interface gemaakt. Hiermee was het mogelijk om per onderdeel een interface implementatie te maken voor een bepaalde database.

Nadat ik dit probleem had opgelost merkte ik al snel op dat er veel methoden waren die bijna hetzelfde deden. Dit waren methoden die gebruikt werden voor het ophalen van alle Releases of ReleasePaths. Het enige verschil was dat er gebruik werd gemaakt van andere klasse en collectienamen. Om dit probleem op te lossen wilde ik de methoden generiek maken en ervoor zorgen dat alles op één plek stond. Na een tijdje een aantal ontwerpen te hebben gemaakt had ik drie mogelijkheden. Deze mogelijkheden worden hieronder besproken:



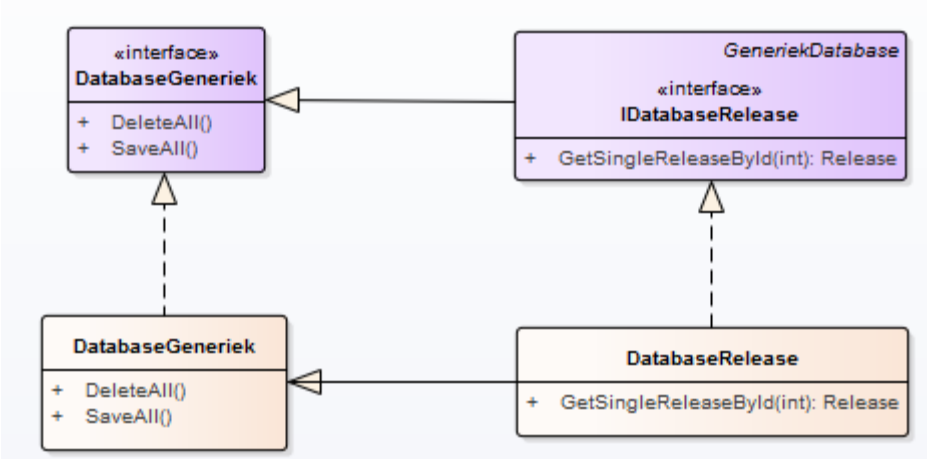
Figuur 37 mogelijkheid 1

Mogelijkheid 1 bestond uit het maken van een interface waarin alle specifieke methoden stonden en een interface die van een andere interface overerft. In de lege interface zouden alle generieke methoden worden geschreven met behulp van extention methods. Met deze extention methods is het mogelijk om methoden aan deze interface te koppelen. Door de interface overerving was het mogelijk voor de DatabaseRelease klasse om ook gebruik te maken van deze extention methods. Het probleem met dit model is dat de extention methods eigenlijk statische methoden zijn die worden toegevoerd aan een andere klasse. Hierdoor was het niet mogelijk om deze methoden te mocken tijdens het testen. Dit bracht al een groot probleem in het mocken van mijn database verbinding. Het was namelijk niet meer mogelijk om de verbinding te mocken. Deze manier viel hierdoor gelijk af.



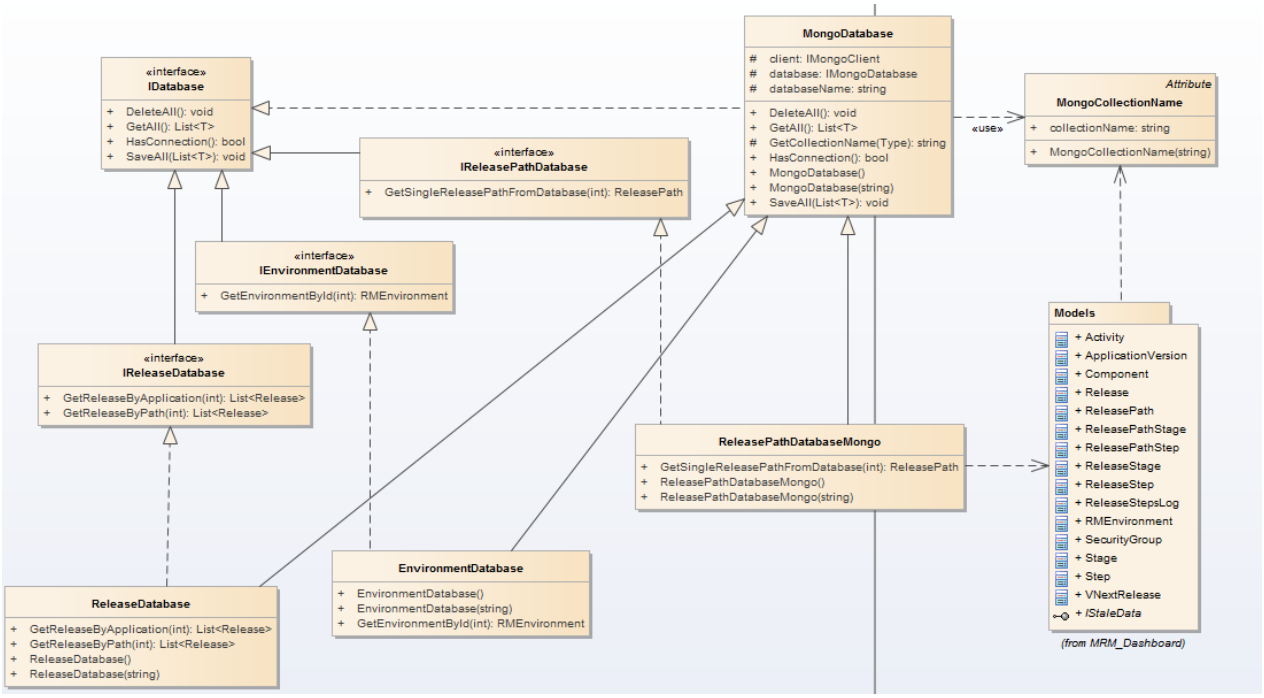
Figuur 38 mogelijkheid 2

Mogelijkheid 2 had ik bedacht met als achterliggende gedachte dat er een aparte klasse bestond waarin alle generieke methoden moesten staan. Verder moest elke database interface deze methoden beschrijven. Voor de implementatie van deze methoden kon een klasse gebruik maken van een relatie naar de klasse met de generieke methoden. Hierdoor was mijn logica op één plek en was het mogelijk alles nog te testen. Met deze oplossing was het mogelijk om alle generieke methoden één keer te schrijven. Ik was niet helemaal tevreden met deze implementatie, omdat alle database interfaces vol zouden raken met de generieke methoden. De implementatie zou alleen op één plek staan, maar alle interfaces moesten alsnog verwijzen naar de implementatie. Om ervoor te zorgen dat dit niet gebeurde had ik ook niet voor deze manier gekozen.



Figuur 39 mogelijkheid 3

Mijn laatste optie was mogelijkheid 3. In mogelijkheid 3 maak ik een twee interfaces. Eén interface voor alle generieke methoden en een andere interface voor specifieke methoden. De interface met specifieke methoden overerft alles van de interface met generieke methoden. Hierdoor is het af te dwingen dat een klasse die specifieke methoden implementeert ook generieke methoden moet implementeren. Verder heb ik ook twee klassen in het diagram die elk een interface implementeren. Eén van deze klassen implementeert alle generieke methoden en de andere alle specifieke methoden. Om ervoor te zorgen dat de specifieke klasse ook voldoet aan de overerving van interfaces moet deze klasse alleen nog een subklasse worden van de Database klasse. Dit ontwerp was ook gemakkelijk te mocken na de implementatie. Verder loste dit ook al de problemen, die eerder zijn beschreven, op. Uiteindelijk heb ik ook gekozen voor mogelijkheid 3. In figuur 40 is de implementatie hiervan te zien.



Figuur 40 klasse diagram uiteindelijke implementatie mogelijkheid 3

Toen ik alle generieke methoden aan het schrijven was merkte ik op dat ik alle collectienamen nodig had voor elk object die ik wilde opslaan. Het probleem hiermee was dat ik niet aan de hand van de klasse kon weten in welke collectie die zat zonder meer informatie. Een voorbeeld van het verwijderen zag er zo uit:

```
public void DeleteAll<T>()
{
        var collection = database.GetCollection<T>("collectionnaam");
        collection.DeleteManyAsync(_ => true).Wait();
}
```

In de voorbeeld is te zien dat de collectienaam nog opgehaald moet worden. Er waren verschillende manieren waarop ik dit kon uitvoeren. Eén van de manieren was om op elk object een interface te laten implementeren waar ik afdwing dat er een collectienaam wordt meegegeven. Het enige probleem hiermee was dat het specifiek voor MongoDB was en niet voor bijvoorbeeld een MySQL implementatie. Er was nog een andere optie die ik had overwogen en dat was het maken van een custom attribute. Een custom attribute is een bepaald stuk code waarmee je een klasse metadata kan geven. Met een custom attribute is het ook mogelijk om een foutmelding naar de gebruiker te sturen als een gebruiker een object zonder metadata probeert op te slaan. Uiteindelijk had ik een custom attribute gemaakt die er als volgt uitzag:

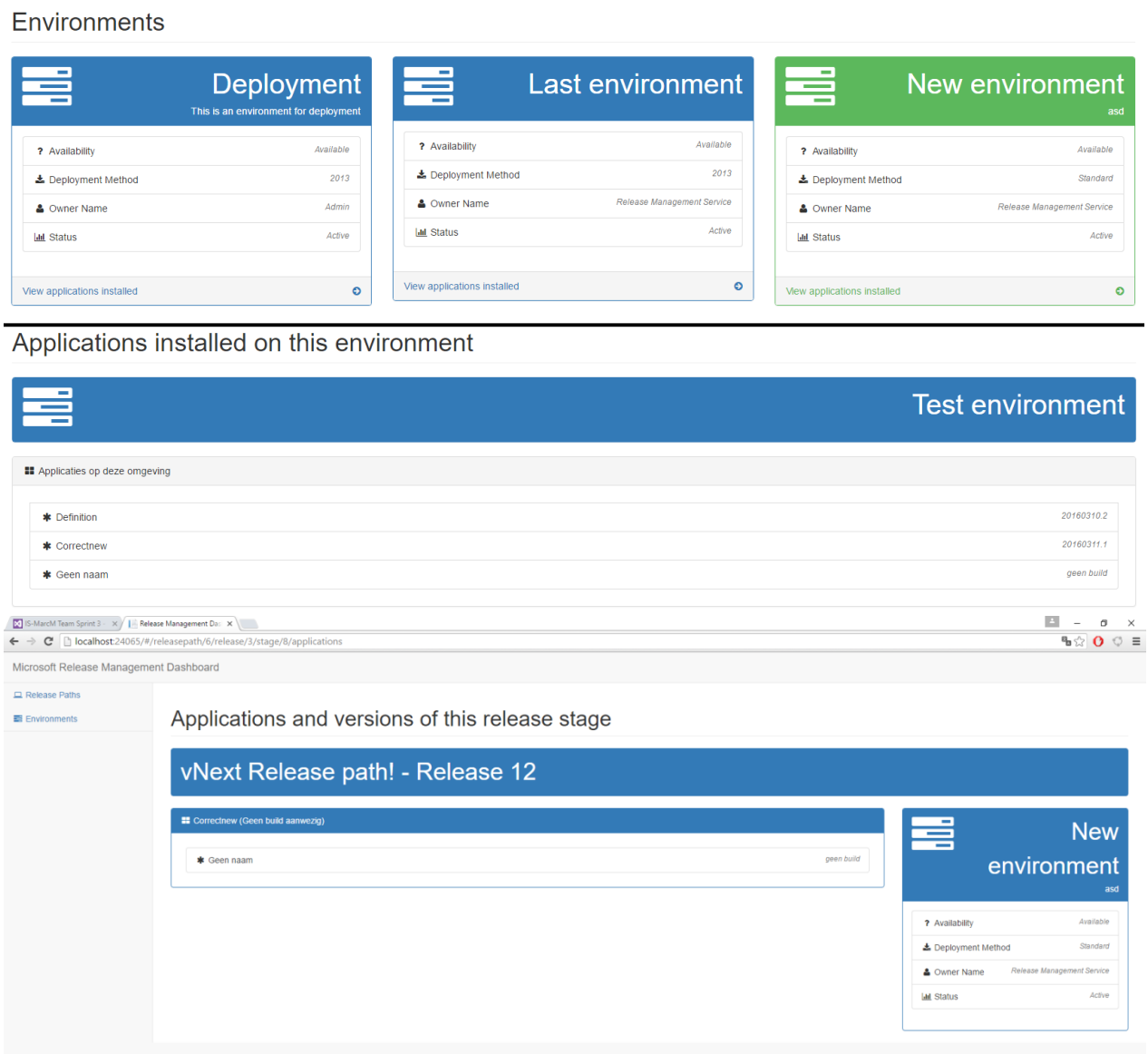
```
[MongoCollectionName("Release")]
```

Nadat ik een methode had geschreven die de MongoCollectionName uitlas was het mogelijk om deze methode te gebruiken bij het verwijderen van entiteiten die deze custom attribute hadden. De code om te verwijderen zag er als volgt uit:

```
var collection = database.GetCollection<T>(GetCollectionName(typeof(T)));
collection.DeleteManyAsync(_ => true).Wait();
```

Resultaten

De resultaten van de sprint zijn in figuur 41 te zien. Hierin zijn de overzichten van alle omgevingen en de applicaties die daarop geïnstalleerd zijn te zien. Verder is er ook een detail scherm van een stage te zien:



Figuur 41 schermprintjes van gerealiseerde functionaliteit

5.5 Sprint 4 – Instellingen, componenten en live updates

In deze paragraaf beschrijf ik eerst aan welke onderdelen ik in deze sprint ga werken. Hierna leg ik uit welke beslissingen ik tijdens het realiseren van deze functionaliteiten heb gemaakt.

5.5.1 Planning en analyse

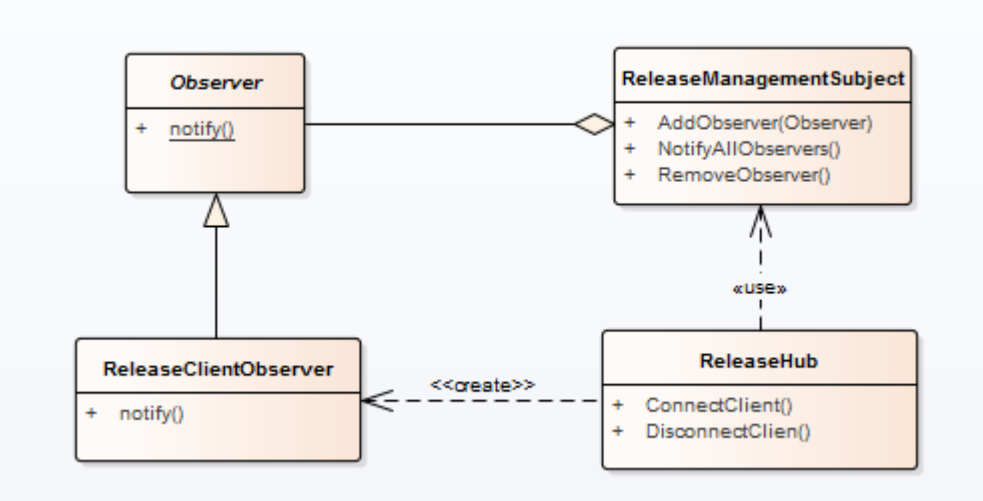
Gedurende de sprint planning met de opdrachtgever hadden wij besloten te werken aan een aantal instellingen op de AngularJS client. De opdrachtgever wilde namelijk dat het mogelijk was om voor bepaalde ReleasePaths te kiezen. Verder wilde hij ook live updates op de AngularJS client. Hiervoor waren er ook instellingen voor nodig. De user stories van deze sprint zijn hier te zien:

Nummer	Story	Prioriteit	TFS id
US10	Als gebruiker wil ik dat er nieuwe data ingeladen wordt in mijn schermen als er wijzigingen hebben plaats gevonden in release management	10	10065
US11	Als gebruiker wil ik kunnen instellen welke Release Paths ik te zien krijg in mijn overzichten, zodat ik alleen informatie zie dat voor mij belangrijk is	11	10069
US12	Als gebruiker wil ik in een Release per Step in elke Stage meer informatie zien over welke stappen daarin zijn uitgevoerd en welke status ze hebben	12	10068

5.5.2 Ontwerpen en Realiseren

Als het dashboard alle Releases toont en er een nieuwe Release wordt uitgevoerd met MRM moet een gebruiker eerst het dashboard verversen voordat alle nieuwe gegevens zichtbaar zijn. Dit verlaagt de gebruikersvriendelijkheid, omdat iemand voortdurend moet controleren of het scherm is ververst. Het is mogelijk om het scherm zelf verantwoordelijk te maken voor het verversen van gegevens. Voor dit probleem heb ik twee oplossingen bedacht. De eerste oplossing was om gebruik te maken van websocket en de tweede oplossing was om gebruik te maken van long polling.

Oplossing 1 websocket op de Webapi server
Websockets is een protocol dat gebruikt kan worden voor communicatie in twee richtingen over een TCP verbinding. Moderne web browsers implementeren deze protocol en maken het dus mogelijk voor servers en clients om direct met elkaar te communiceren. Het .Net framework heeft een package dat gebruikt kan worden voor het snel realiseren van een verbinding met websockets. Deze package heet SignalR. Met SignalR is het mogelijk om een centrale punt te maken waar alle clients naar toe kunnen verbinden. Deze clients kunnen met elkaar en de server communiceren zonder dat er requests op de server worden uitgevoerd. Dit kan gebruikt worden om live updates te realiseren door de server naar de clients een bericht te laten sturen zodra er nieuwe wijzigingen zijn in release management. In figuur 42 is een implementatie te zien van websockets met SignalR.



Figuur 42 klasse diagram van SignalR implementatie

In dit ontwerp is de ReleaseHub verantwoordelijk voor het afhandelen van alle verbindingen van de clients. Dit betekent dat de ReleaseHub ook de observers moet aanmaken. Nadat een Observer is aangemaakt kan deze door de ReleaseHub worden toegevoegd of verwijderd uit de ReleaseManagementSubject (de observerable). De ReleaseManagementSubject is verantwoordelijk voor het afluisteren naar MRM. Bij wijzigingen in MRM wordt dit doorgegeven aan alle Observers. Het nadeel van deze oplossing is dat het de complexiteit van de applicatie sterk verhoogt.

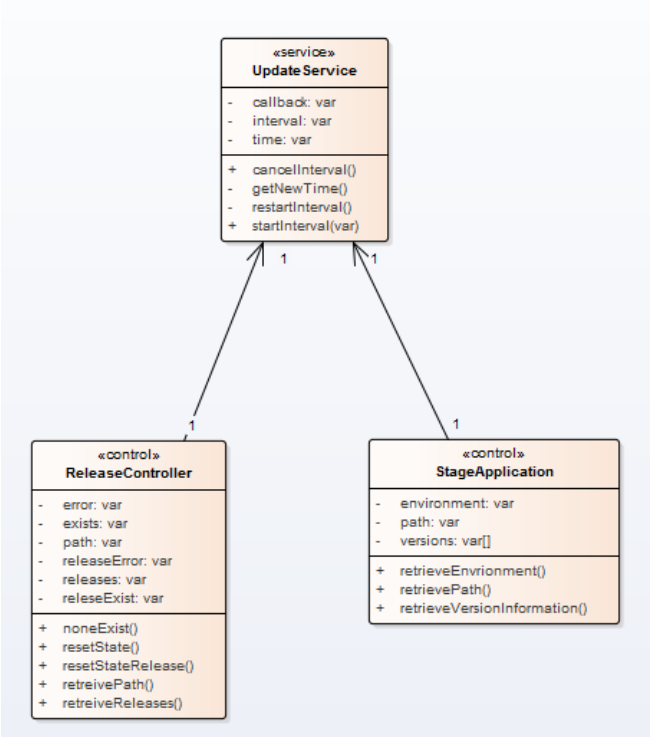
Oplossing 2 long polling op de AngularJS client

Long polling is een term dat gebruikt wordt voor het beschrijven van een techniek waar een website met gebruik van Javascript requests doet naar een server[RF08]. Met deze requests ververst de website de data die wordt getoond aan de gebruiker. Officieel gezien is dit geen vorm van een push notificatie, maar het effect lijkt er wel op. Met long polling is het belangrijk om te weten dat alle requests asynchroon worden uitgevoerd. De gebruiker merkt dus niet aan de website dat het bezig is met nieuwe gegevens inladen. Dit is goed voor het dashboard, omdat een gebruiker nog verder informatie kan bekijken terwijl het dashboard data ververst.

Long polling kan op verschillende wijze geïmplementeerd worden in het dashboard. Niet alle schermen moeten een live update krijgen. Het is noodzakelijk dat de schermen, waar alle Releases en Stages, op staan live updates krijgen. Dit betekent dat de methode die gebruikt wordt om deze gegevens in te laten met een bepaalde interval aangeroepen moet worden.

Beslissing

Aangezien de eerste oplossing meer tijd kost en de applicatie complexer maakt had ik deze beslissing aan de opdrachtgever voorgelegd. De opdrachtgever had gekozen voor de tweede oplossing. In figuur 43 is te zien hoe ik dit heb geïmplementeerd. Voor het bijhouden van de timer heb ik een service gebruikt. Deze service roept met een interval een bepaalde callback aan die je moet meegeven bij het aanroepen van de timer. Een callback is een referentie naar een functie in Javascript. Zoals eerder vermeld in voorgaande sprints zijn services Singleton's in AngularJS. Hierdoor maakt de applicatie op de client nooit meer dan één timer aan.



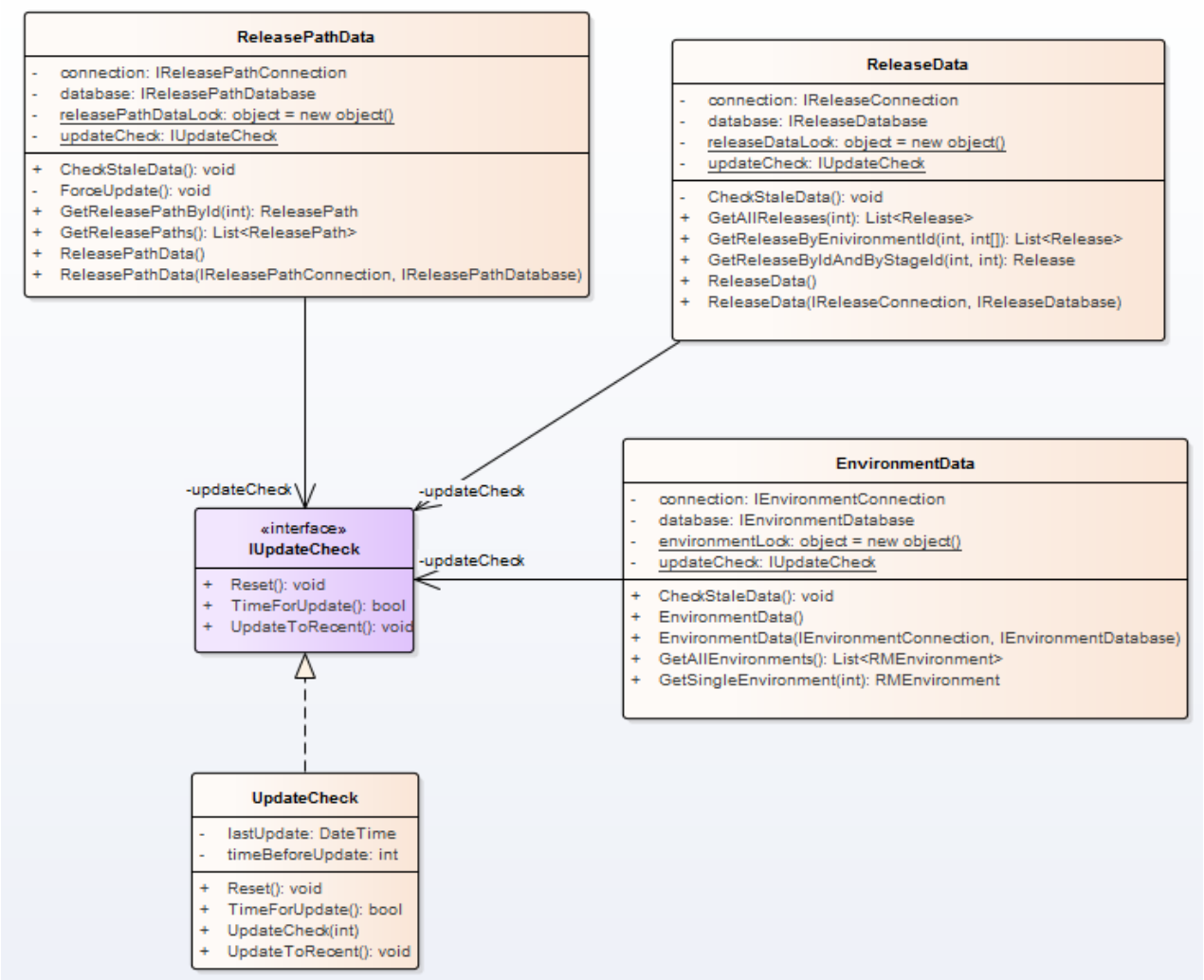
Figuur 43 implementatie van long polling AngularJS

Refactor Update

In de voorgaande sprints was er geen aandacht besteed aan het verversen van gegevens in de cache. In deze sprint wil ik daar aandacht aan besteden. Daarom wordt dit stukje in de code gerefactored. Momenteel zijn er drie klasse in de applicatie die de cache met een interval verversen. Om ervoor te zorgen dat er geen dubbele code ontstaat, is er een interface gemaakt met de methoden die nodig zijn voor het updaten. Vervolgens heb ik een implementatie gemaakt die met een interval aangeeft wanneer er een update uitgevoerd moet worden.

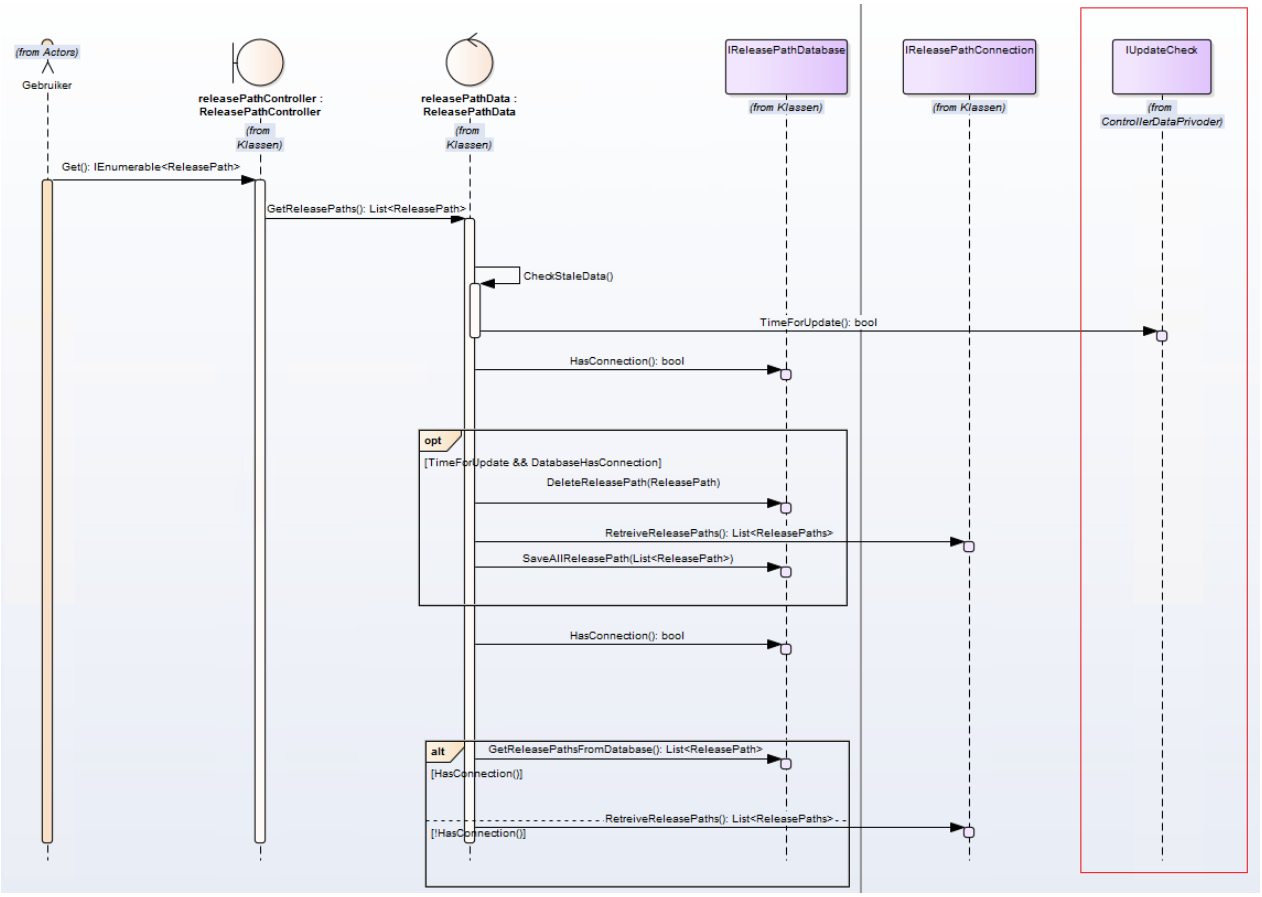
Ik heb besloten om een interface te maken, zodat er op elk moment een andere manier van updaten gerealiseerd kan worden. Als MRM in de toekomst events kan versturen bij een update is het mogelijk om hier een andere implementatie van de interface voor te maken.

In figuur 44 is te zien hoe ik dit heb geïmplementeerd. In de klasse die de interface gebruiken is te zien dat deze statisch wordt bijgehouden. Bij elke request naar de server maakt webapi een nieuwe instantie aan van de controller. Dit betekent dat er ook nieuwe instanties van de data klassen gemaakt wordt. Als er nieuwe instanties van implementaties van de IUpdateCheck gemaakt worden betekent het dat de cache elke keer opnieuw ververst wordt. Met een statische referentie betekent het dat dit object altijd hetzelfde zal zijn voor alle instanties van die klasse.



Figuur 44 nieuwe interface voor het updaten

In figuur 45 is te zien hoe dit de sequentie diagram beïnvloed. De nieuwe interface is aangegeven in het rood. In plaats van alle logica intern aan te roepen kan de klasse ReleasePathData deze interface aanroepen.



Figuur 45 aanpassing aan de sequentie van updaten

Opslag instellingen op de AngularJS client

Wanneer een gebruiker het dashboard gebruikt wilt hij niet alle gegevens zien die zijn medegebruiker ook ziet. Hij wilt alleen informatie zien waar hij bij betrokken is. Daarom is het nodig dat de instellingen op de client worden opgeslagen. Om gegevens op een client op te slaan zijn er twee mogelijkheden. Deze mogelijkheden bestaan uit het opslaan in cookies of in localstorage. In het volgende tabel is er informatie te zien over cookies en localstorage.

Overweging	Localstorage	Cookie
Doel	Gegevens opslaan op de client en door de client te laten gebruiken	Gegevens opslaan op de client en uitwisselen tussen client en server
Elke request verstuurd	Nee	Ja
Vervaltijd	Geen	Aangeven
Opslag	5MB	4069B

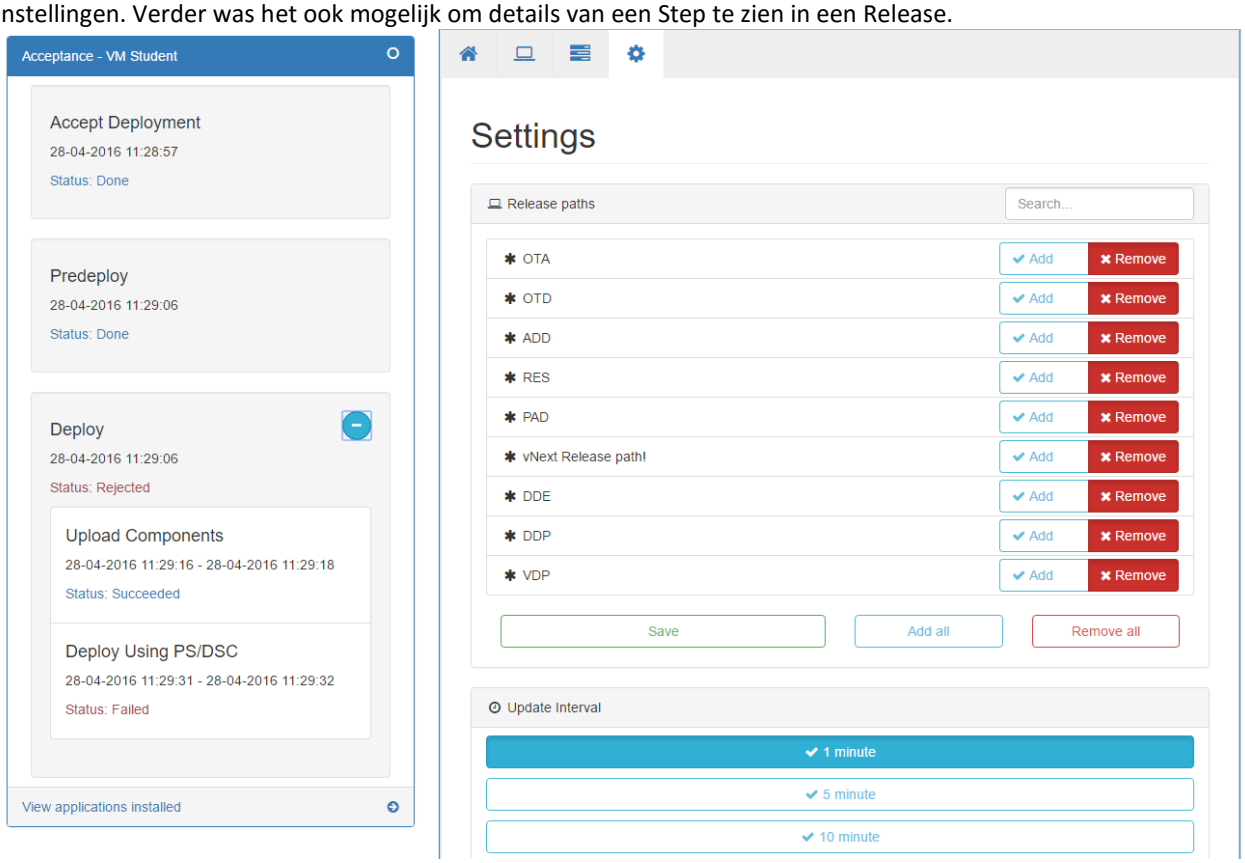
Uiteindelijk is er gekozen om gebruik te maken van localstorage. De laatste twee punten, namelijk vervaltijd en opslag zijn niet doorslaggevend voor het kiezen van een opslag methode. Het aangeven van een vervaltijd op een cookie is niet moeilijk en indien het nodig is om meer dan 4069B op te slaan kunnen er meerdere cookies worden opgeslagen.

Bij deze beslissing gaat het voornamelijk om de functionaliteit die gerealiseerd wordt. Ten eerste willen wij een interval kunnen kiezen die de client kan gebruiken om te bepalen wanneer het een verzoek naar de server moet doen. Dit betekent dat deze gegevens nooit naar de server verstuurd moeten worden. De overige instelling is het

bepalen welke ReleasePath je te zien krijgt in het ReleasePath overzicht. In AngularJS is het mogelijk om alle gegevens van ReleasePaths in te laden en deze vervolgens te filtreren aan de hand van gegevens in de instellingen. Een AngularJS filter is een klasse die je data kan filtreren aan de hand van logica die je zelf schrijft. Hierdoor is het ook niet nodig om deze instellingen te versturen naar de server.

Uiteindelijk heb ik gebruik gemaakt van localstorage. De enige functionaliteit waarvoor een instelling naar de server moet gaan is het tonen van alle applicaties en versies op een omgeving. Om deze overzichten te tonen is het nodig dat er op de server alle ReleasePaths gesorteerd worden voordat er bepaald wordt welke applicaties zijn geïnstalleerd. Aangezien dit de enige keer is dat er een instelling naar de server moet gaan is het niet nodig cookies te gebruiken. Cookies zullen alleen maar onnodig informatie tussen de client en server versturen.

Resultaten
De resultaten van de sprint zijn te zien in figuur 46. Hierin is te zien dat ik het mogelijk was om te kiezen tussen instellingen. Verder was het ook mogelijk om details van een Step te zien in een Release.



Figuur 46 schermprintjes van gerealiseerde functionaliteit

5.6 Sprint 5 – Recente releases en KnowNow koppeling

In deze paragraaf beschrijf ik de laatste sprint. Gedurende deze sprint heb ik niet veel nieuwe functionaliteiten ingebouwd. Deze sprint was vooral gericht op het afronden van mijn opdracht. Verder moest mijn applicatie ook gekoppeld worden aan een MRM server binnen Info Support.

5.6.1 Planning en analyse

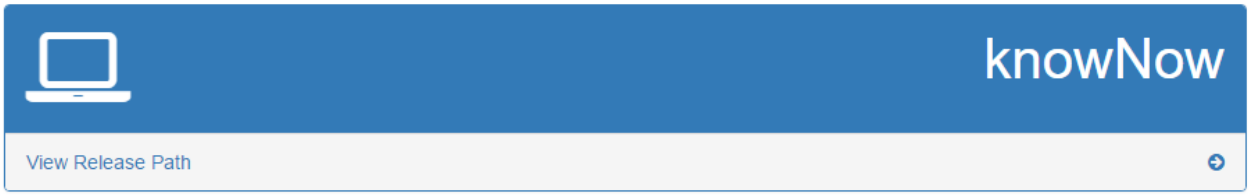
Gedurende deze sprint hadden de opdrachtgever en ik besloten om twee kleine functionaliteiten te realiseren. Deze beslissing hadden we genomen, omdat dit de laatste sprint was en wij het product wilde afronden. Verder moest ik een handleiding schrijven voor de installatie van het dashboard. De handleiding van het dashboard is te vinden in bijlage V. De user stories waar ik deze sprint aan had gewerkt zijn hieronder te zien:

Nummer	Story	Prioriteit	TFS id
US13	Als gebruiker wil een scherm waarin de meest recente releases staan	13	10366
US14	Als gebruiker wil ik een bij elke step in een stage van een release een waarschuwing krijgen als de status van de step 'failed' is	14	10385

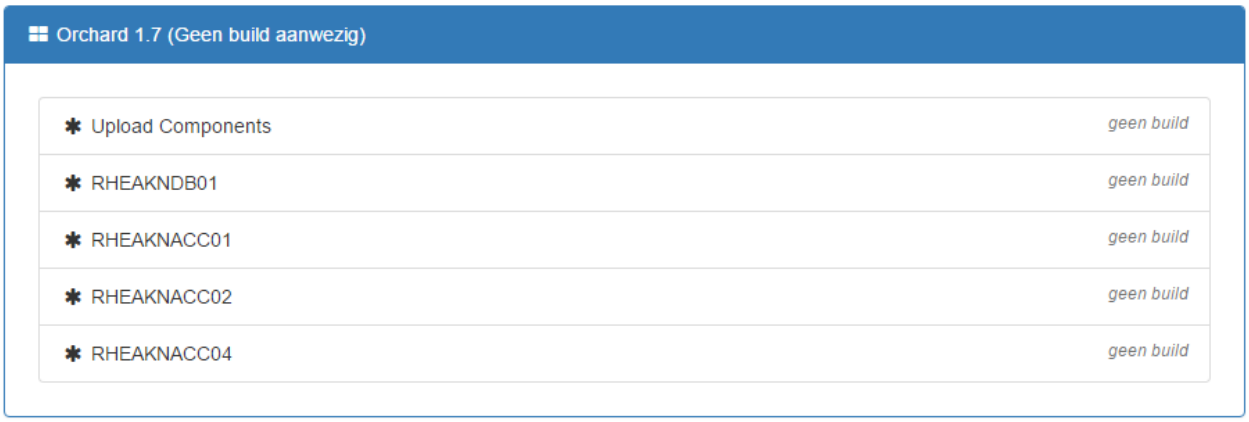
5.6.2 Koppelen dashboard aan KnowNow

KnowNow is een interne kennisbank applicatie van Info Support. Hierop kunnen alle medewerkers artikelen en presentaties delen met elkaar. De KnowNow applicatie is ook in MRM ingericht om op verschillende omgevingen automatisch geïnstalleerd te worden.

In mijn laatste sprint moest ik het dashboard koppelen aan de MRM server waar KnowNow op is ingericht. Met deze koppeling kon de opdrachtgever zien hoe de applicatie in een echte omgeving te werk ging. Het resultaat van de koppeling met de MRM server is te zien in figuur 47 en 48.



Figuur 47 KnowNow Release path in het keuze menu



Figuur 48 alle componenten die zijn geïnstalleerd in Release 506 van KnowNow op de acceptatie omgeving

Backwards compatible

De MRM server waar KnowNow op geconfigureerd is, was nog niet geüpdatet naar de versie waar ik rekening mee had gehouden tijdens het analyseren en realiseren. Op de server was nog versie 2013 update 4 aanwezig. Het was niet noodzakelijk dat het dashboard met deze versie werkte, maar dit was de MRM server waar ik het snelst toegang tot kon krijgen. Het dashboard werkte niet gelijk met deze versie, maar na onderzoek met Fiddler was ik te

weten gekomen dat er verschillende versies van de API bestaan. In het voorbeeld hieronder is te zien hoe de API versie wordt meegestuurd met een request naar de MRM API:

/account/releaseManagementService/_apis/releaseManagement/ConfigurationService/GetReleasePath?id=2**&api-version=3.0**

De installatie van MRM 2013 update 4 werkte op API versie 3.0 en de MRM API van 2015 update 1 werkte op API versie 6.0. Door de API versie variabel te maken heb ik met een kleine wijziging ervoor gezorgd dat het dashboard backwards compatible is.

6. Evaluatie

In dit hoofdstuk geef ik een evaluatie over mijn producten, proces en beroepstaken.

6.1 Productevaluatie

In mijn productevaluatie zal ik beginnen met het verduidelijken van hoe mijn eindresultaat het doel van het project heeft behaald. Hiervoor zal ik terugverwijzen naar mijn afstudeerplan. Verder heb ik gedurende mijn afstuderen aan diverse producten gewerkt. Voor elk product beschrijf ik hoe deze haar doel heeft gediend en voor welke doelgroepen dit belangrijk was.

6.1.1 Eindresultaat

Het einderesultaat dat beschreven is in mijn afstudeerplan is behaald. In mijn afstudeerplan heb ik aangegeven dat mijn opdracht als doel had om een dashboard te maken. In dit dashboard moest er informatie van MRM getoond worden. Hiermee moet duidelijk zijn welke versies van applicaties op welke omgevingen geïnstalleerd zijn. Verder moest de geschiedenis van elke installatie van een applicatie ook duidelijk weergegeven worden. Hierbij is het belangrijk dat alle stappen van de installaties ook te zien zijn.

In mijn eerste twee sprints heb ik een aantal functionaliteiten moeten realiseren die nodig waren om het eindresultaat te bereiken. Nadat mijn vierde sprint was afgerond had ik de functionaliteiten die in mijn afstudeerplan waren beschreven behaald. Het is nu mogelijk voor de gebruiker om de gehele geschiedenis van de installaties te zien. Verder is het ook mogelijk om per omgeving te kijken welke applicaties daarop zijn geïnstalleerd.

Als ik terugkijk naar het eindresultaat merk ik wel dat er geen extra functionaliteiten gerealiseerd zijn buiten de kern van de applicatie.

6.1.2 Plan van aanpak

Het plan van aanpak heeft ervoor gezorgd dat de afspraken tussen de opdrachtgever, technisch begeleider en proces begeleider duidelijk waren. In mijn plan van aanpak had ik beschreven hoe ik van plan was het project aan te pakken en hoe ik alle communicatie richting de genoemde groepen van plan was uit te voeren. Als laatste was het ook mogelijk om een aantal beheer aspecten te benoemen die belangrijk waren voor het project, zoals overlegvormen en middelen die nodig waren.

Gedurende het schrijven van mijn plan van aanpak merkte ik wel dat ik in sommige gevallen eerst beslissingen nam voordat ik dit met de opdrachtgever had besproken. Hierdoor moest ik soms mijn teksten herschrijven om het kloppend te maken met wat de opdrachtgever verwachte. In de toekomst zal minder beslissingen alleen nemen en vaker met een voorstel komen bij de opdrachtgever.

6.1.3 Microsoft Release Management Analyse

Mijn analyse over de mogelijkheden voor het verkrijgen van gegevens uit MRM heeft mijn opdracht duidelijker gemaakt. In de eerste instantie wist ik niet wat mijn opties waren voor het verkrijgen van gegevens. Na deze analyse

kon ik mijn opdrachtgever uitleggen welke voor- en nadelen er waren per mogelijkheid en heb ik mijn keuze kunnen onderbouwen.

In paragraaf 5.6.2 staat er dat ik te weten was gekomen dat mijn code ook backwards compatible was. Ik vind het jammer dat ik dit pas gedurende mijn laatste sprint te weten ben gekomen. Als ik mijn analyse opnieuw had kunnen doen had ik meer onderzoek gedaan op de verbinding in oudere versies. Dit was niet noodzakelijk voor mijn analyse, maar ik had er mogelijk meer van kunnen leren.

6.1.4 Requirements Analyse

Mijn requirements analyse heb ik uitgevoerd en verwerkt in een functioneel ontwerp. In dit document heb ik al mijn user stories, wireframes en diagrammen per sprint beschreven. Dit document heb ik gebruikt om samen met de opdrachtgever te valideren of het duidelijk is wat er gerealiseerd moet worden. Mijn opdrachtgever was tevreden met dit document, omdat het mogelijk voor hem was om in te grijpen als iets fout ging voordat het gerealiseerd werd.

6.1.5 Ontwerpen

Gedurende mijn project heb ik een technische ontwerp document gemaakt. In dit document heb ik al mijn beslissingen kunnen documenteren. Mijn technisch begeleider was tevreden met al mijn beslissingen en vond dat deze op een goed niveau waren. Mijn enige feedback was dat ik soms moest letten op mijn schriftelijke vaardigheden.

In mijn ontwerpen had ik achteraf gezien meer tijd moeten besteden aan mijn datamodel. Met de gegevens uit MRM heb ik bepaalde modellen opgesteld. Als ik vergelijkbaar werk in de toekomst moet doen zal ik mijn datamodel met meer detail uitwerken.

6.1.6 Applicatie

Over de code die ik heb geschreven ben ik tevreden. Gedurende het programmeren heb ik ervoor gezorgd dat de verantwoordelijkheden in de applicatie goed verdeeld waren. Dit is te zien in de refactor slagen die ik in sprint 3 en 4 heb gemaakt. Van mijn technisch begeleider heb ik ook een code review gehad. Na de review had hij alleen positief commentaar.

6.1.7 Tests

Mijn volledig geautomatiseerde tests hebben ervoor gezorgd dat er gedurende het geheel project vaak gekeken kon worden of de applicatie nog werkte of niet. Na wijzigingen en/of refactor slagen heb ik snel fouten kunnen vinden in mijn applicatie.

SpecFlow combineren met TDD was voor mij aan het begin van mijn project een hindernis. Naarmate ik gewend was aan het werken met SpecFlow ging dit wel beter. In de toekomst zal ik SpecFlow niet snel combineren met TDD, omdat ik vind dat het teveel tijd kost om SpecFlow in te richten voordat je kan programmeren.

6.2 Procesevaluatie

Zoals in mijn aanpak is vermeld heb ik een aangepaste versie van Scrum gebruikt. Het werken met deze aangepaste versie van Scrum beviel mij goed. Het niet uitvoeren van daily standups had naar mijn mening geen negatieve invloed op het project. Het toepassen van deze manier van werken zal ik in de toekomst weer overwegen als ik Scrum alleen moet gebruiken. De tweewekelijkse activiteiten van Scrum, zoals sprint review en sprint planning, heb ik ook goed kunnen uitvoeren.

In de eerste sprint had ik het uitvoeren van mijn tests te kort ingepland. Dit bleek veel langer te duren, omdat ik nog niet gewend was om met TDD te werken. In de sprints die hierna volgde ging het testen sneller. Uiteindelijk heb ik tijd kunnen besparen bij het debuggen en opsporen van fouten door mijn bestaande tests. In een toekomstige project zal ik deze manier van werken zeker aanhouden.

Tijdens mijn proces merkte ik wel dat ik soms niet veel communiceerde naar de opdrachtgever. Ik voerde mijn taken uit en nam beslissingen over implementatie. Na twee sprints merkte ik dat ik dit aan het doen was en had ik besloten meer documenten naar de opdrachtgever te sturen. Hierdoor nam ik niet alleen beslissingen. In toekomstige projecten zal ik vaker en sneller contact opnemen met de opdrachtgever om hem de voortgang van het project te wijzen.

Over de meeste van mijn keuzes ben ik tevreden geweest tijdens dit project. Mijn belangrijkste keuze was het gebruiken van de niet gedocumenteerde API van MRM. Dit koste namelijk veel tijd aan het begin van het project om uit te zoeken hoe de API precies werkte. Volgens de bevindingen van mijn analyse over de complexiteit van de MRM database had het uitzoeken en bevragen van de database meer tijd gekost.

Wat ik moeilijk vond tijdens het project was het werken met SpecFlow om unit en integratie tests uit te voeren. Voor mij waren de SpecFlow tests te veel gezien vanuit de gezichtsvlak van de gebruiker. Hierdoor was het voor mij moeilijk om het voor unit- en integratietests te gebruiken. Zoals in mijn uitvoering is beschreven heb ik dit alleen gebruikt voor UI tests. In de toekomst zal ik SpecFlow alleen gebruiken om een deel functionaliteit in grote lijnen te testen. Het uitvoeren van TDD zal ik er nog naast doen.

Als laatste heb ik tijdens het project ook gekozen om te werken met MongoDB als database voor het dashboard. Met deze beslissing ben ik tevreden. Aangezien MongoDB flexibel is met dataopslag heb ik geen implementatie modellen moeten maken en bijwerken na elke sprint zoals in sprint 1 is beschreven.

6.3 Evaluatie beroepstaken

In mijn evaluatie van beroepstaken geef ik aan in welke mate elke beroepstaak is behaald. Om dit goed te kunnen beargumenteren zal ik terug verwijzen naar mijn sprints.

6.3.1 1.4 Uitvoeren analyse door definitie van requirements

Voor mijn opdracht heb ik requirements moeten ophalen bij de opdrachtgever. Het ging om een applicatie die geen vaste eisen en wensen hadden aan het begin van het project. Er waren meerdere stakeholders die betrokken waren bij het project, maar deze werden allemaal vertegenwoordigd door de opdrachtgever. Tijdens de sprint planning en analyse waren er een aantal requirements gewijzigd, zoals beschreven in sprint 3.

Verder is er in sprint 1 te zien dat user stories worden gebruikt voor het bijhouden van eisen en wensen. Per user story wordt ook het acceptatie criteria opgesteld. Indien de user stories te groot waren was er ook gebruik gemaakt van epics. Deze technieken zijn allemaal gebruikt om bij de opdrachtgever te valideren of het juist systeem gebouwd zal worden. In paragraaf 5.2.4 beschrijf ik ook dat ik de acceptatie criteria van user stories gebruik als input voor het opstellen van de tests. Hierdoor hebben user stories ook invloed op wat er gerealiseerd wordt.

Deze beroepstaak heb ik geheel uitgevoerd op niveau 3. Zelfstandig en lastig.

6.3.2 3.1 Ontwerpen softwarearchitectuur

In eerste instantie had ik deze beroepstaak niet opgenomen in mijn afstudeerplan. Echter heb ik hier wel aandacht aan besteed tijdens mijn project. In sprint 1 is te zien hoe ik een deel van het architectuur heb ontworpen. De rest van mijn ontwerpen zijn in bijlage III te raadplegen.

Deze beroepstaak heb ik deels uitgevoerd op niveau 3. Zelfstandig en lastig.

6.3.3 3.2 Ontwerpen systeemdeel

De webapi die ik heb ontwerpen is object georiënteerd. Dit is duidelijk te zien in mijn ontwerp diagrammen die ik in elke spring heb beschreven. Voor het ontwerpen heb ik gebruik gemaakt van UML en als tool Enterprise Architect. Bij het ontwerpen heb ik, zoals in sprint 1 en 2, beschreven hoe ik de testbaarheid en hergebruik heb verhoogd door gebruik te maken van interfaces en dependancy injection. Verder heb ik in sprint 3 ook beschreven hoe ik mijn database laag heb gerefactored. Dit ontwerp zorgde ervoor dat mijn database laag beter uit te breiden was.

Deze beroepstaak heb ik geheel uitgevoerd op niveau 3. Zelfstandig en lastig.

6.3.4 3.3 Bouwen applicatie

De applicatie die ik heb gerealiseerd is, zoals in paragraaf 5.2.2 vermeld, object georiënteerd. Verder heb ik geavanceerde concepten van de programmeertaal gebruik. Voorbeelden hiervan zijn beschreven in sprint 3 met generics. Verder heb ik ook gebruik gemaakt van Linq queries voor het ophalen van gegevens uit mijn database. Verder heb ik, zoals in mijn analyse is beschreven, webrequests ‘gesniffed’ en de XML antwoorden gedeserialiseerd naar C# modellen. Hierdoor is de applicatie die ik heb gerealiseerd ook aangesloten op een extern systeem, namelijk MRM.

Al mijn programmeer werk heb ik uitgevoerd op een ontwikkelomgeving die tot mijn beschikking werden gesteld. Voor een versiebeheertool heb ik gebruik gemaakt van GIT op TFS, zoals vermeld in mijn aanpak.

Als laatste heb ik ook twee programmeertalen gebruikt. Deze talen zijn C# en Javascript. Voor beide talen heb ik gebruik gemaakt van een framework. In C# was dat het .Net framework en voor Javascript was dat AngularJS. In sprint 1 en 2 is te zien hoe ik ervoor heb gezorgd dat mijn code uit te breiden en testbaar was. Voorbeelden hiervan zijn gebruik van services in AngularJS en het gebruiken van dependency injection in mijn C# code.

Deze beroepstaak heb ik geheel uitgevoerd op niveau 4. Zelfstandig en complex.

6.3.5 3.5 Uitvoeren van en rapporteren over het testproces

Voor het testen van mijn applicatie heb ik gebruik gemaakt van TDD als test ontwerptechniek. Verder heb ik ook aandacht besteed aan het herhaalbaar maken van mijn tests door alle tests te automatiseren, zoals in sprint 1 is beschreven. Mijn tests bestaan uit integratie, unit en UI tests. Hiermee dek ik ook de volledige applicatie af. Alle tests heb ik tijdens de sprint reviews laten draaien zodat de opdrachtgever de resultaten kon zien. Dit is in sprint 1 beschreven. Hiermee heb ik ook mijn tests richting de opdrachtgever gecommuniceerd.

Deze beroepstaak heb ik geheel uitgevoerd op niveau 3. Zelfstandig en lastig.

7. Referenties

Code	Bron
RF01	Dijkstra, R. (2016). *Uitlezen gebruikers, groepen en rechten van Release Management 2015 met PowerShell.* https://knownow.infosupport.com/knowledge/uitlezen-gebruikers-groepen-en-rechten-van-release-management-2015-met-powershell
RF02	Van Wijk, E. (2016). *Microsoft Release Management Dashboard*. https://knownow.infosupport.com/knowledge/microsoft-release-management-dashboard
RF03	(2016). *Visual Studio Team Services Features Timeline.* https://www.visualstudio.com/en-us/news/release-archive-vso.aspx
RF04	*Release Management 2015 (server and client version)* https://msdn.microsoft.com/library/vs/alm/release/overview-rm2015
RF05	*The lock statement.* https://msdn.microsoft.com/en-us/library/aa664735(v=vs.71).aspx
RF06	De Penning, M (2016). *Info Support – Organogram.* https://knownow.infosupport.com/knowledge/info-support-organogram-introductiefilmpjes
RF07	Wroblewski, L. (2011). *Mobile first*. New York: Jeffrey Zeldman
RF08	Hanson, J. (2014, 12 01). *What is HTTP Long Polling*. Opgehaald van PubNub: https://www.pubnub.com/blog/2014-12-01-http-long-polling/
RF09	Shalloway, M. & Trott (2005). J.R. *Design Patterns Explained.* Boston: Addison-Wesley
RF10	Fowler, M. (2013). *PageObject.* http://martinfowler.com/bliki/PageObject.html
RF11	Loukides, M. (2012). *What is DevOps?* http://radar.oreilly.com/2012/06/what-is-devops.html