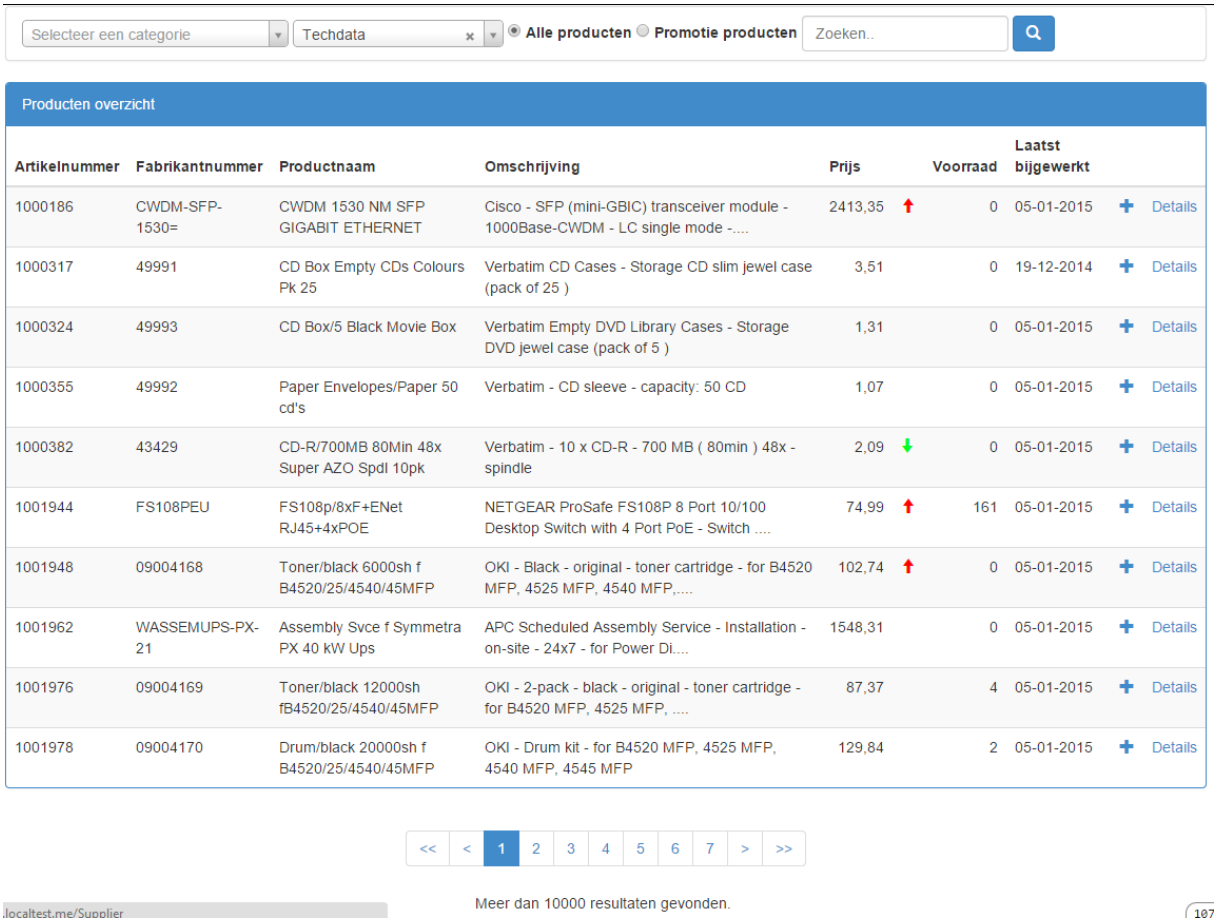
Afstudeerverslag

Hardware offerte tool



Student: Donny Roelen

Afstudeerbedrijf: Interpulse Automatisering B.V.

1e examinator: Gerard Mijnarends

2e examinator: Juul Maas

Bedrijfsmentor: Arjan Maagdelijn

Datum: 8 januari 2015

Afstudeerperiode: 1 september 2014 – 9 januari 2014

Versie: 1.0

Voorwoord

Voor u ligt het afstudeerverslag van Donny Roelen. Het afstudeerverslag is geschreven ter afronding van de opleiding informatica aan de Haagse Hogeschool Den Haag. Als basis van het afstudeertraject ligt de afstudeeropdracht. De afstudeeropdracht is uitgevoerd bij Interpulse automatisering B.V. te Leiden. Het doel van de afstudeeropdracht was het ontwerpen en ontwikkelen van een tool die Interpulse helpt bij het maken van offertes voor klanten.

Dit verslag is een beschrijving van de uitgevoerde afstudeeropdracht en bevat daardoor technische en vak gerelateerde termen. Het rapport is dan ook gericht op lezers die basiskennis hebben van informatica en de bijbehorende trajecten.

Ik wil bij dezen mijn bedrijfsmentor Arjan Maagdelijn bedanken voor de begeleiding tijdens het afstudeertraject. Ook wil ik de opdrachtgever Marcel van Elsacker bedanken voor zijn flexibiliteit. Daarnaast wil ik nog de hele afdeling ontwikkeling bedanken voor de ondersteuning tijdens het afstudeertraject.

Ook wil ik graag mijn dankwoord uitspreken naar de begeleiding vanuit school. Mijn dank gaat hierbij uit naar Gerard Mijnarends en Juul Maas.

In zijn geheel wil ik Interpulse bedanken voor een leuke, gezellige en leerzame afstudeer periode.

Leiden, 8 januari 2015

Donny Roelen

Inhoudsopgave

1 Inleiding

Dit document is geschreven door Donny Roelen, afstudeerder van de opleiding Informatica aan de Haagse Hogeschool. Het project duurt 17 weken en wordt uitgevoerd bij Interpulse Automatisering B.V. Het onderwerp van de afstudeeropdracht is “het ontwerpen en ontwikkelen van een hardware offerte tool”.

Het doel van de afstudeeropdracht is het aanzienlijk terugbrengen van de benodigde tijd voor het samenstellen van hardware offertes.

Het afstudeerverslag is geschreven als eindverslag voor de afstudeerperiode. Het afstudeerverslag bevat een uitgebreide beschrijving van het proces. Hoe er te werk is gegaan en tot welke resultaten dit heeft geleid. In het afstudeerverslag wordt meerdere malen gerefereerd naar andere documenten. Deze documenten zijn toegevoegd in de bijlage en dienen als extra informatie.

2 Organisatie en omgeving

Dit hoofdstuk beschrijft de organisatie waarin ik werkzaam ben geweest. Onderwerpen die aan bod komen zijn: achtergrond informatie, de organisatie structuur van het bedrijf, beschrijving van mijn werkplek en de rolverdeling tijdens het afstudeertraject.

2.1 Interpulse Automatisering B.V.

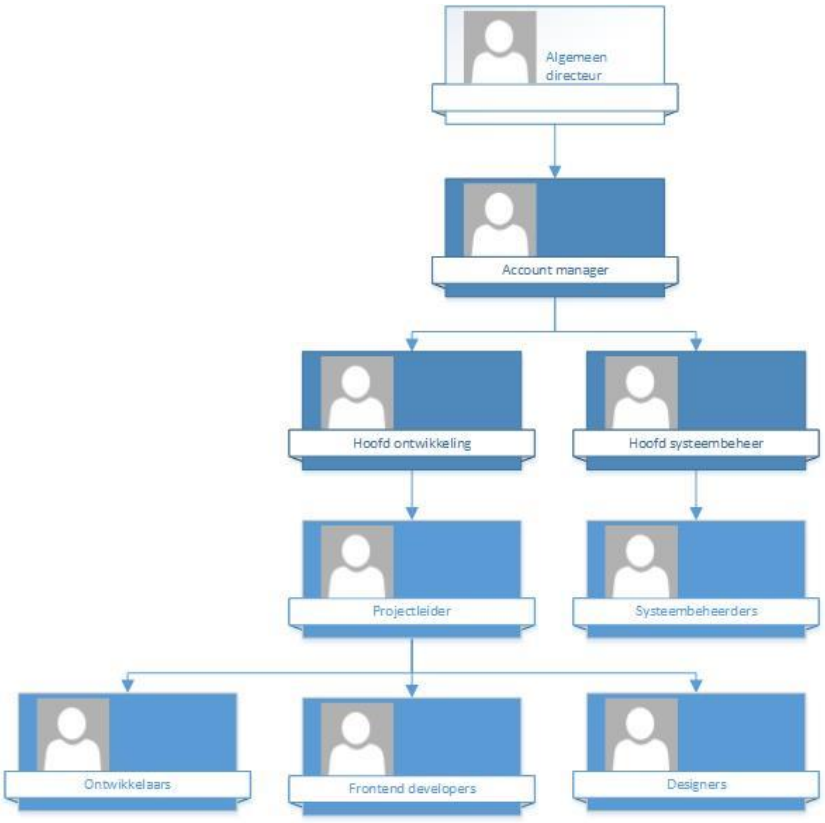
De afstudeeropdracht wordt uitgevoerd bij Interpulse Automatisering B.V. Interpulse Automatisering B.V. is een full-service internet en automatiseringsbedrijf. Het bedrijf is begonnen als een ICT bedrijf met 7 werknemers in 2000. 14 jaar later telt Interpulse 19 werknemers en blijft het aantal klanten nog steeds groeien. Interpulse is gevestigd in het centrum van Leiden. Interpulse biedt de volgende diensten aan:

- Systeembeheer
- Systeemontwikkeling
- Internetdiensten
- Consultancy

Deze diensten worden geleverd namens de afdelingen; directie, management, beheer en ontwikkeling. Er is 1 algemeen directeur, met daaronder het managementteam. Daaronder vallen de 2 werkvelden Ontwikkeling en Beheer.

2.2 Organisatie cultuur

Figuur 1 geeft de structuur van de organisatie weer.



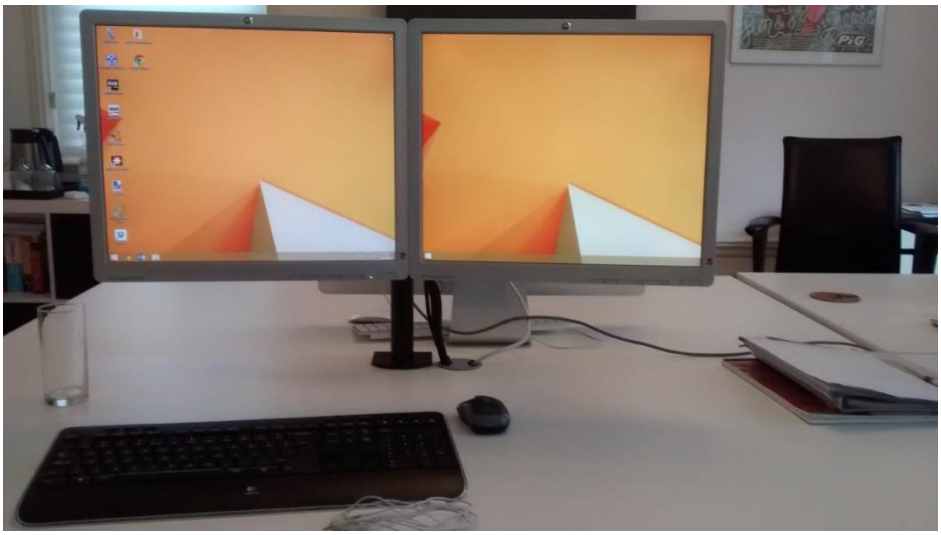
Figuur 1: Organisatie structuur van interpulse.

2.3 Werkplek

De afstudeeropdracht is uitgevoerd binnen de afdeling ontwikkeling op de 1e verdieping. Hier zat ik samen met vijf andere ontwikkelaars en twee designers. Mijn werkstation is ook uitgerust over de benodigde software, onderstaand een lijst met beschikbare software:

- Microsoft Office 2013 pakket
- Visual studio 2013
- MS SQL Server 2012
- Microsoft Visual Studio 2013
- GIT

In figuur 2 ziet u mijn werkplek voor de afstudeerperiode.



Figuur 2: Mijn werkplek tijdens de afstudeerperiode bij Interpulse.

2.4 Rolverdeling

Om de opdracht succesvol uit te voeren zijn er enkele rollen benoemd. De rollen zijn opgesteld in het belang van de opdracht en de begeleiding hiervan.

Rol	Wie	Omschrijving
Bedrijfsmentor	Arjan Maagdelijn	Begeleidde de afstudeerder en functioneerde als algemeen aanspreekpunt
Opdrachtgever	Marcel van Elsacker	Naast opdrachtgever ook eindgebruiker. Bij de opdrachtgever werden de wensen en eisen voor het te bouwen systeem opgevraagd. Tevens is Marcel hoofd manager systeembeheer binnen Interpulse.
Projectleider	Ik	De projectleider is verantwoordelijk voor het gehele project, zowel het opstellen van de planning als het uitvoeren van de werkzaamheden.
Overige ondersteuning	Collega ontwikkelaars	Collega ontwikkelaars zijn soms geraadpleegd voor advies of hulp.

3 Definitie afstudeeropdracht

In dit hoofdstuk wordt de opdracht beschreven. De aanleiding, probleemstelling, doestelling en het resultaat komen hier aanbod.

3.1 Aanleiding

Interpulse Automatisering B.V. is een full service automatiseerder, naast software en diensten levert Interpulse ook hardware aan klanten. Interpulse is zelf geen fabrikant, importeur of groothandelaar op het gebied van hardware, maar functioneert als tussenpersoon tussen leverancier en klant. Interpulse verkrijgt informatie over de producten via de websites van de leveranciers. Er zijn 6 leveranciers waar op dit moment bestellingen worden gedaan:

- Techdata
- Aces Direct
- Copaco
- Xeptor
- Central Point BV
- Travion

De belangrijkste leveranciers zijn Techdata, Aces Direct, Copaco en Xeptor.

3.2 Probleemstelling

Het samenstellen van hardware offertes, waarbij de prijzen dagelijks wijzingen is een tijdrovende klus. Er is geen compleet beeld van alle prijzen. Bestellingen plaatsen bij leveranciers moet voldoen aan bepaalde regels, zoals minimum orderbedrag. Het verzamelen van meerdere orders is op dit moment niet goed mogelijk en foutgevoelig.

In de huidige situatie bestaat het samenstellen van hardware offertes voornamelijk uit handwerk. Op de website van de leveranciers worden producten met elkaar vergeleken op inkoopprijs en voorraad. De gevonden artikelen worden vervolgens handmatig in een offerte geplaatst. Deze handelingen worden in principe altijd twee keer per week gedaan. Het samenstellen van de offertes kost ongeveer 2 uur per week en wordt gedaan door de hoofd afdeling ICT. Dat is elke week 2 uur die beter besteed had kunnen worden. De hoofd afdeling ICT heeft namelijk al erg druk.

Het is de bedoeling dat er een tool wordt ontwikkeld dat producten kan vergelijken bij de groothandelaars (soort van tweakers pricewatch). Je wilt bijvoorbeeld van een bepaalde I-7 processor weten waar deze het goedkoopst verkrijgbaar is of waar je deze het snelst geleverd kan krijgen. Door het automatiseren van dit proces wordt er veel tijd bespaard. Daarnaast is het de bedoeling dat deze tool in de toekomst ook offertes kan maken aan de hand van gekozen artikelen bij de groothandelaars. Een verkoper kan dan bijvoorbeeld bij de klant gelijk een offerte opmaken, dan weet de klant precies wat een bepaald systeem kost. De tool zal in eerste instantie vooral veel tijd besparen, maar mogelijk in de toekomst ook extra inkomsten bezorgen.

3.3 Doelstelling

De doelstelling van de opdracht is een duidelijk overzicht genereren van het artikelenassortiment van de groothandelaars. Inzicht in de inkoop prijzen en voorraad status is hierbij van belang.

De koppelingen met de leveranciers stelt mij in staat binnen één systeem alle product informatie beschikbaar te stellen. Aan de hand van een overzicht moet de gebruiker snel producten kunnen opzoeken. Doordat alle informatie binnen één systeem beschikbaar wordt gesteld hoeft de gebruiker

niet meer informatie op verschillende websites op te zoeken. Dit zorgt ervoor dat de benodigde tijd voor inkoop proces aanzienlijk wordt teruggebracht.

3.4 Resultaat

Het uiteindelijk resultaat van de opdracht bestaat uit de hardware offerte tool. Met deze tool kan de opdrachtgever veel tijd besparen bij het opmaken van hardware offertes. Er hoeft namelijk niet meer handmatig vergeleken te worden, alle prijsinformatie is nu beschikbaar binnen één tool.

4 Aanpak

Dit hoofdstuk beschrijft de aanpak van het project. De gekozen methoden en de invulling hiervan komen hier aan bod.

4.1 Vergelijking methoden

In de beginfase van het project heb ik een klein onderzoek uitgevoerd om een keuze te maken voor de te gebruiken ontwikkelmethode. Ik heb er voor gekozen om dit onderzoek uit te voeren omdat ik in eerste instantie zat te twijfelen tussen de te gebruiken methodes. Ik leunde in eerste instantie naar de methode Scrum, maar ik vroeg me af of het verstandig was om deze methode zelfstandig uit te voeren. Om ervoor te zorgen dat ik een goede keuze zou maken met betrekking tot de te gebruiken methode heb ik de eerste week besteed aan het onderzoek naar de methoden.

De volgende vijf methoden heb ik onderzocht:

- Scrum
- Waterval
- RUP
- RAD
- XP

Bij het maken van het afstudeerplan is gekozen om mijn onderzoek enkel te richten op de vijf bovengenoemde methoden. De keuze om alleen deze vijf methoden te onderzoeken had ik al genomen voordat de afstudeerperiode was begonnen. Ik heb ervoor gekozen om maximaal vijf methoden te onderzoeken om het onderzoek niet te groot te maken. Ik heb gekeken naar welke methoden ik kende en daarnaast heb ik nog gekeken welke ik verwachte toe te kunnen passen binnen mijn project. Scrum heb ik in mijn onderzoek meegenomen omdat ik daar in eerste instantie gebruik van wilde maken. Daarnaast maakt Interpulse ook standaard gebruik van Scrum als ontwikkelmethode. Waterval en RUP heb ik meegenomen omdat ik hier in aanraking ben mee gekomen tijdens mijn studie. Met RAD en XP was ik al enigszins bekend en ik dacht dat deze toe te passen zouden zijn binnen mijn afstudeerproject.

Methode	**Scrum**	**Waterval**	**RUP**	**RAD**	**XP**
Kenmerken					
Iteratief	Ja	Nee	Ja	Ja	Ja
Is individueel uit te voeren	Redelijk	Ja	Redelijk	Ja	Nee
Documentatie / modellering	Redelijk	Ja	Ja	Nee	Nee
Feedback opdrachtgever	Ja	Ja	Ja	Ja	Ja
Duidelijk structuur in het traject	Redelijk	Ja	Ja	Nee	Nee
Omgaan met wijzigende wensen en eisen	Ja	Nee	Ja	Ja	Ja

Ik heb voor mijn onderzoek zelf een aantal kenmerken opgesteld die ik belangrijk vond voor mijn project. Iteratief vind ik belangrijk, omdat ik de klant kort wil houden. Doordat ik de opdrachtgever kort heb gehouden kan deze ook feedback op het product geven. Hierdoor krijgt hij niet opeens aan het eind van het ontwikkeltraject een product opgeleverd wat niet binnen zijn verwachtingen valt. Aangezien ik het project alleen uitvoer moet het een methode zijn de individueel uitgevoerd kan worden. Een belangrijk onderdeel van mijn afstudeeropdracht was het maken van ontwerpen van het te bouwen systeem. Een methode die mij hier ook sturing aangaf was dan ook erg belangrijk. Feedback opdrachtgever had ik al vermeld. Een duidelijk structuur binnen het project vond ik ook erg

belangrijk. Hiermee bedoel ik dat het project duidelijk is opgedeeld in bepaalde onderdelen. Volgens deze onderdelen moet er te werk gegaan worden. Als laatste kenmerk vond ik het belangrijk dat de methode om kon gaan met wijzigende requirements. Het was namelijk voordat het ontwikkeltraject was begonnen nog duidelijk wat voor informatie de leveranciers ons allemaal konden leveren. Aan de hand van de beschikbare informatie komen er ook weer nieuwe requirements naar boven. Hier moet de methode dus mee om kunnen gaan.

4.1.1 Conclusie

Aan de hand van de bovenstaande vergelijking valt duidelijk te zien welke methoden de meeste raakvlakken heeft met de wensen voor mijn project. Dit is namelijk RUP. Na RUP heeft Scrum de meeste raakvlakken.

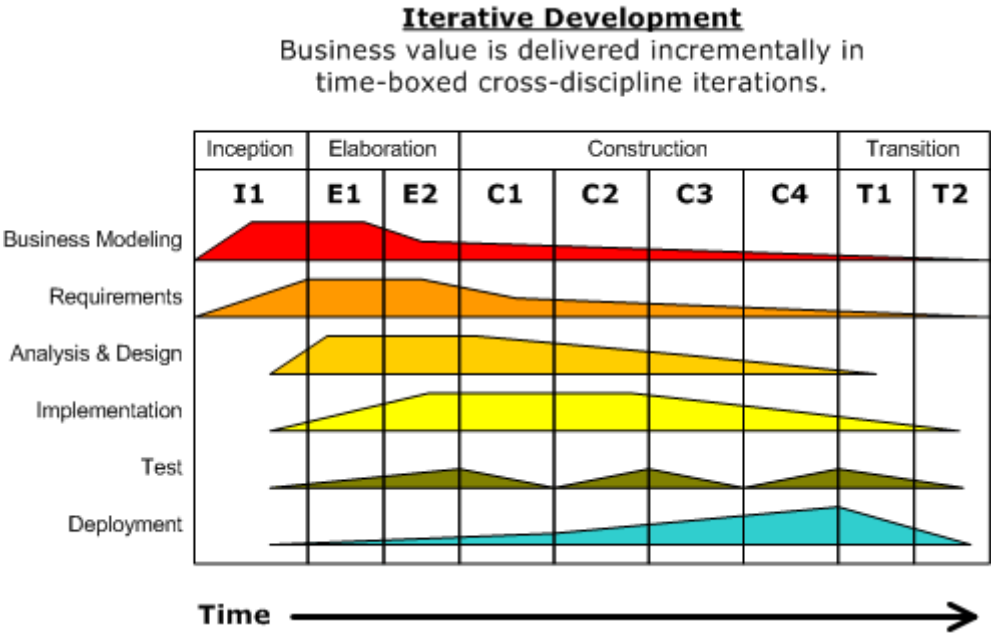
Zoals ik al eerder aangaf ging mijn voorkeur in eerste instantie uit naar Scrum. Scrum wordt namelijk binnen Interpulse als standaard methode gebruikt voor het ontwikkelen van software. Het werken volgens de Scrum sprints was iets wat mij erg aantrok. Per sprint kan je een duidelijke planning maken en een stuk software opleveren bij de opdrachtgever. Op het gemaakte stukje software kan je feedback vragen en deze feedback kan je vervolgens in de volgende sprint meenemen. Helaas is Scrum meer bedoeld voor projectgroepen van 5-10 projectleden. Alhoewel er genoeg voorbeelden zijn waarin het in kleinere projectgroepen ook succesvol is uitgevoerd. Het in mijn eentje uitvoeren van de Scrum methode zou betekenen dat ik bepaalde onderdelen zou moeten laten vallen. Zoals de bijvoorbeeld de dagelijkse standup meetings en het planningspoker. Verder heeft Scrum een beperkte hoeveelheid aan documentatie, een echt technisch ontwerp ontbreekt bijvoorbeeld. Al deze nadelen hebben mij uiteindelijk van Scrum doen afwijken.

Zo komen we bij RUP, ik heb uiteindelijk gekozen om volgens RUP te gaan werken. RUP voldoet in principe aan al mijn opgestelde eisen, alleen heb ik hier en daar terugzien komen dat het individueel uitvoeren lastig kan zijn. RUP geeft mij een duidelijke houvast voor mijn afstudeerproject. RUP is iteratief, hierdoor kan ik telkens een aantal modules per iteratie gaan ontwikkelen in plaats van alles in één keer. Aan het eind van de iteratie kan ik feedback vragen en de feedback kan ik meenemen in de volgende iteratie. Ook is bij RUP een uitgebreide documentatie vereist, wat een belangrijk onderdeel is van mijn afstudeerproject.

Voor een uitgebreide beschrijving van de voor en nadelen van de onderzochten methoden verwijs ik u naar mijn onderzoeksrapport.

4.2 RUP

RUP is opgedeeld in vier fasen, inception, elaboration, Construction en transition. In figuur 3 ziet u hoe een RUP proces normaal is opgebouwd.



Figuur 3: Overzicht algemeen RUP proces. [http://en.wikipedia.org/wiki/Rational_Unified_Process]

4.2.1 Inception fase

Het project begint met de inception fase, in de inception fase wordt het project voorbereid. Het plan van aanpak wordt opgesteld en de planning voor de iteraties wordt opgesteld. Aan de hand van interviews worden de wensen en eisen van de opdrachtgever opgehaald. Vervolgens worden er requirements opgesteld en worden de use cases gemaakt en beschreven. Het laatste onderdeel van de inception fase is het opstellen van een eerste versie van een klassendiagram. Het onderdeel business modeling heb ik uiteindelijk laten vallen. In eerste instantie had ik het wel meegenomen in mijn inception fase, alleen het heeft naar mijn mening geen meerwaarde voor de uitvoering van mijn project. De software wordt namelijk ontwikkelt voor een interne opdrachtgever die deze alleen voor intern gebruik zal gebruiken.

4.2.2 Elaboration fase

Vervolgens komen we bij de elaboration fase. In de elaboration fase worden de ontwerpen opgesteld. Het klassendiagram wordt verder uitgewerkt, er worden sequentie en activiteiten diagrammen opgesteld en er wordt ook een database model opgesteld. Naast de ontwerpen wordt er ook nog een analyse uitgevoerd voor de koppelingen met de leveranciers. Er wordt op de website van de leverancier gekeken wat voor mogelijkheden er zijn voor een koppeling. Mocht er geen informatie te vinden vallen op de website wordt er contact op genomen met de desbetreffende leverancier.

4.2.3 Construction fase

De constructie fase is de fase waar het meeste tijd aan zal worden besteed. De construction bestaat uit het bouwen van de software en het maken van testscripts. De applicatie wordt in twee iteraties opgebouwd, de eerste iteratie bestaat uit de basis en de tweede vooral uit uitbreidingen die de applicatie nog bruikbaarder maken.

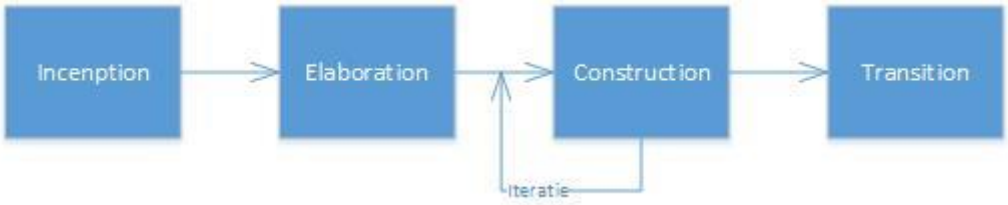
4.2.4 Transition fase

De transition fase bestaat uit het uitrollen van de software. Daarnaast zullen er tijdens de transition fase nog gebruikers testen worden uitgevoerd.

4.2.5 Iteraties

Ik heb het project opgedeeld in twee iteraties. In de eerste iteratie wordt de basis van het systeem gebouwd. De inception en elaboration fase voor zover mogelijk geheel doorgelopen. In mijn ontwerp zitten zo ook al onderdelen voor iteratie 2. Iteratie 1 is klaar wanneer de constructie fase is doorlopen. Bij iteratie 2 ga ik niet helemaal terug naar de inception fase, maar ik begin gelijk bij de construction fase. Ik ga namelijk niet opnieuw alle activiteiten uitvoeren die bij de inception en elaboration fase horen. Wel wordt de bestaande documentatie bijgewerkt met de nieuwe functionaliteiten. Zo worden bijvoorbeeld de requirements bijgewerkt en worden er enkele nieuwe sequentiediagrammen gebouwd. In figuur 3 kan je ook terugzien dat tijdens de construction fase er nog activiteiten worden uitgevoerd voor de requirements en analysis en design.

Mijn RUP proces ziet er uit zoals in figuur 4 weergeven.



Figuur 4: Mijn RUP proces

4.3 UML

RUP maakt standaard gebruik van UML voor de ontwerpen. UML staat voor Unified Modeling Language. Ik heb binnen mijn project ook gebruik gemaakt van UML voor de ontwerpen van het systeem. UML biedt meerdere diagrammen aan waarmee ontwerpen worden gemaakt. Elk van deze diagrammen is bedoeld om een bepaald onderdeel van het systeem te beschrijven:

Ik heb gebruik gemaakt van de volgende diagrammen:

- Klassendiagram
- Sequentiediagram
- Toestandsdiagram
- ERD

4.4 Frameworks

In dit hoofdstuk kunt u informatie vinden over de frameworks die ik gebruikt het tijdens het ontwikkelen van de software.

4.4.1 ASP.NET MVC 5

Het systeem wordt ontwikkeld op het ASP.NET MVC 5 framework. ASP.NET is gebaseerd op het 3 lagen systeem Model, View, Controller. Enkele functionaliteiten die gebruikt worden van ASP.NET MVC 5 zijn ASP.NET Identity, Bootstrap en Scaffolding.

De keuze om met dit framework aan de slag te gaan had ik eigenlijk al gemaakt voordat ik op zoek ben gegaan naar een afstudeeropdracht. In het blok I-7 van de opleiding Informatica heb ik onderzocht welke ontwikkelaars functie het meest wordt gevraagd. Hierin kwam de functie als ASP.NET ontwikkelaar als winnaar naar boven. Doordat ik tijdens het afstudeertraject ervaring opdoe met het ASP.NET framework, hoop ik in de toekomst meer kans te maken op een leuke baan als ASP.NET ontwikkelaar. Daarnaast maak ik gebruik van de meest recente versie van ASP.NET MVC, namelijk versie 5. Binnen Interpulse wordt er ook standaard gebruikt van ASP.NET MVC.

4.4.1.1 Entity framework

De software wordt ontwikkeld met behulp van het entity framework in combinatie met code first. Het Microsoft Entity Framework is een ORM framework dat ontwikkelaars in staat stelt om te werken met relationele data als domein specifieke objecten. Door gebruik te maken van het Entity Framework kunnen ontwikkelaars data opvragen aan de hand van LINQ queries, vervolgens krijg je strongly typed objecten terug. Het Entity Framework stelt ontwikkelaars in staat zich meer te richten op de business logica in plaats van de data access laag.

De keuze voor het gebruik maken van code first is pas gemaakt nadat ik al een database ontwerp had opgesteld. Dit is enigszins onhandig, aangezien er onnodig tijd is besteed aan het opstellen van een database ontwerp. In eerste instantie twijfelde ik nog of het verstandig was om code first te gaan gebruiken. Maar de voordelen die mijn collega's noemde hebben me uiteindelijk overtuigd.

Door gebruik te maken van code first binnen mijn project hoef ik niet zelfstandig mijn database aan te maken en bij te werken. Deze worden gegenereerd aan de hand van de model classes. Het voordeel hiervan is dat de aanpassingen in de model klassen ook worden doorgevoerd naar de database. Het enige wat ik dan moet doen is een nieuwe migratie aanmaken. De migratie zoekt de verschillen tussen het model van de vorige migratie en het model van de huidige migratie. Vervolgens wordt er een script aangemaakt voor het bijwerken van de database. De scripts staan gewoon in een standaard class bestandje en kunnen dus worden geopend en gewijzigd mocht je dit willen.

De database kan je vervolgens bijwerken door in de package manager console het commando "update-database" uit te voeren. Het handige van deze migraties is dat je ook nog terug kan naar een vorige database versie. Je hoeft enkel aan te geven naar welke migratie je de database wilt bijwerken en de database wordt weer teruggezet zoals het model er tijdens de gekozen migratie uitzag. De aanwezige data in de database wordt niet aangepast. Tenzij je een kolom weghaalt, dan zal alle data in deze kolom worden verwijderd.

4.4.2 Bootstrap

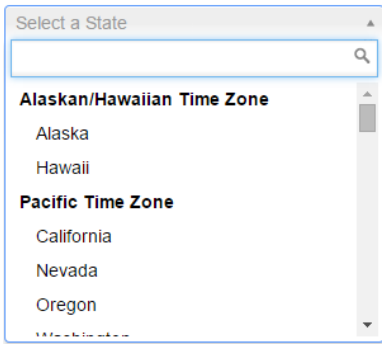
Voor de opmaak van de applicatie maak ik gebruik van de meeste recente versie van Bootstrap, namelijk v3.2.0 anno oktober 2014. Bootstrap is een open-source framework gemaakt door Twitter. Bootstrap beschikt over HTML, CSS en Javascript code die mij helpt bij het ontwikkelen van de front-end van het systeem. Door gebruik te maken van de beschikbare content van Bootstrap ziet de website al goed uit zonder dat ik daar zelf veel tijd aan hoef te besteden. Daarnaast wordt er voor de meeste projecten bij Interpulse gebruikt gemaakt van Bootstrap.

4.4.3 Javascript en jQuery

Binnen mijn applicatie wordt er veelvuldig gebruik gemaakt van javascript en jQuery. Naast de basis functionaliteiten die bootstrap, javascript en jQuery standaard bieden maak ik ook nog gebruik van een aantal externe libaries. Tijdens het ontwikkelen heb ik pas de keuze gemaakt om hier gebruik van te maken. Onderstaand zal ik de gebruikte externe libaries kort beschrijven.

Select2

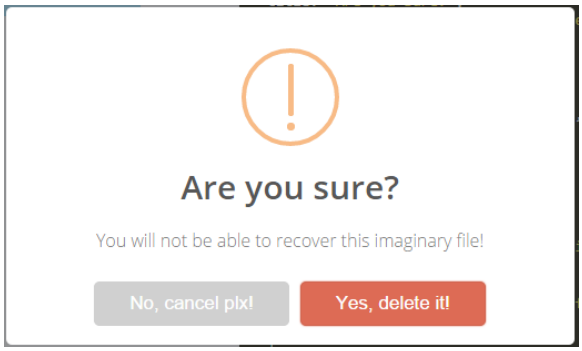
Ik gebruik select2 voor alle html selectboxes binnen de applicatie. Select2 zorgt ervoor dat de selectbox er beter uit ziet. Maar belangrijker is dat select2 het mogelijk maakt om te zoeken in de waardes van een selectbox. In figuur 5 kunt u zien hoe een select2 selectbox eruit ziet.



Figuur 5: Voorbeeld select2 [http://ivaynberg.github.io/select2/]

Sweetalert

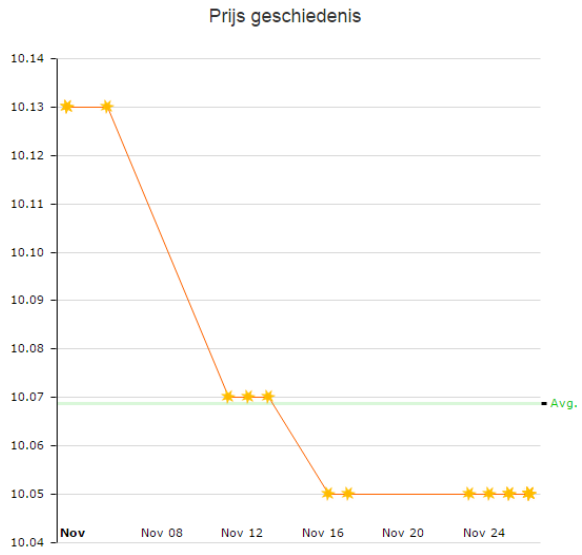
Sweetalert heb ik veelvoudig gebruikt binnen mijn applicatie. Ik gebruik sweetalert voor het tonen van verschillende meldingen. Maar ook heb ik sweetalert gebruikt voor confirm meldingen. Sweetalert zorgt ervoor dat de standaard alert en confirm er een stuk beter uitzien. In figuur 6 kunt u zien hoe een sweet alert confirm message wordt getoond.



Figuur 6: Voorbeeld sweetalert [http://tristanedwards.me/sweetalert]

Amcharts

Ik gebruik amcharts voor het weergeven van grafieken binnen de applicatie. Amcharts maakt het mogelijk om dynamisch verschillende soorten grafieken te genereren. Ik figuur 7 ziet u een voorbeeld van een amchart grafiek die ik heb gebruikt voor de prijsgeschiedenis.



Figuur 7: Voorbeeld amcharts [http://www.amcharts.com/demos/]

5 Uitvoering: Inception Fase

In dit hoofdstuk wordt de inception fase besproken. De inception fase is de eerste fase van een RUP proces.

Tijdens de inception fase zijn de volgende activiteiten uitgevoerd:

- Opstellen plan van aanpak
- Wensen en eisen in kaart brengen
- Eerste ontwerp van het systeem

5.1 Plan van aanpak

De eerste activiteit tijdens de inception fase was het opstellen van een plan van aanpak. Het plan van aanpak is als geschreven in een los document. In het plan van aanpak valt onder andere een planning te vinden voor het gehele project en ook detail planningen voor de twee iteraties. Een uitgebreide omschrijving van het plan van aanpak kunt u terugvinden in mijn bijlage.

5.2 Requirements

De requirements zijn opgesteld aan de hand van de eerste wensen en eisen die aan bod kwamen tijdens het eerste interview voor het afstudeerplan. In een vervolg gesprek in eind week 1 zijn de wensen en eisen geverifieerd en zijn er weer nieuwe wensen en eisen boven tafel gekomen.

De requirements worden verdeeld in groepen volgens het FURPS+ model, dit model is ontwikkeld door Hewlett-Packard. FURPS+ verdeelt de requirements in de volgende groepen:

- Functionality - funtioneel
- Usability - gebruikbaarheid
- Reliability - betrouwbaarheid
- Performance - werking
- Supportability - onderhoudbaarheid

De plus staat voor enkele overige wensen en eisen die de gebruiker kan hebben:

- Ontwerp – hoe wordt de software gebouwd.
- Implementatie – moeten de programmeurs bepaalde standaarden aanhouden.
- Interface – hoe ziet de applicatie eruit.
- Fysiek – op wat voor hardware het systeem draait.

Ik heb uiteindelijk alleen gebruik gemaakt van de in het groen gekleurde type requirements. Van de overige requirements typen is geen gebruik gemaakt.

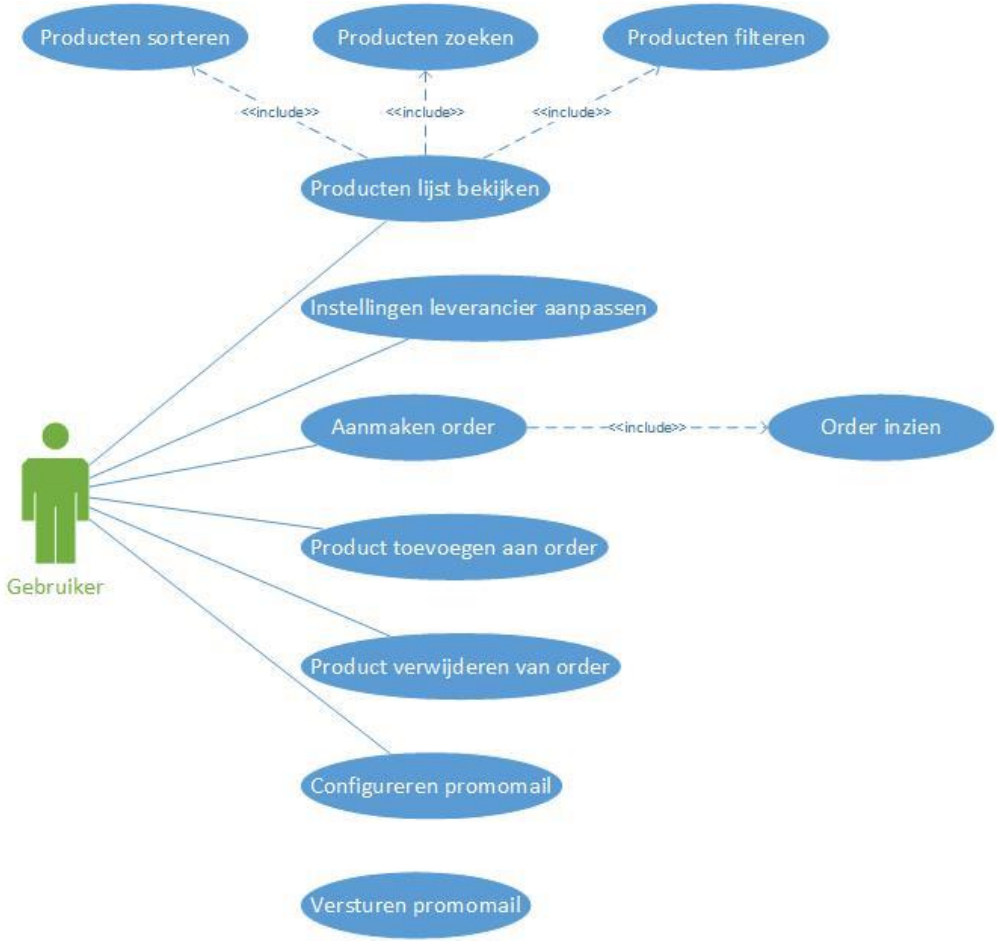
Daarnaast worden de requirements geprioriteerd aan de hand van MoSCoW. Ik hier voor de MoSCoW methoden gekozen omdat ik dit heb meegekregen vanuit school. Ik kan dit dus gemakkelijk binnen mijn project toepassen zonder daar eerst vooronderzoek naar te hoeven doen. De Must have requirement komen voor zover mogelijk in iteratie 1 terug. Iteratie 2 bestaat voornamelijk uit de should en could have requirements.

- Must have – moet in het eindresultaat komen, zonder is het product niet bruikbaar.
- Should have – is zeer gewenst, maar zonder is het product bruikbaar.
- Could have – komen aanbod als er tijd genoeg is.
- Won’t have – komen in dit project niet aanbod, maar misschien wel in een vervolgproject.

In de hoofdstuk 5 van het Inception rapport vindt u een complete lijst met de opgestelde requirements. De requirements zijn opgesteld aan de hand van de afgelegde interviews met de opdrachtgever. Het eerste interview vond al plaats in de zoektocht naar een afstudeeropdracht. Tijdens de eerste week van de afstudeerperiode zijn de vooraf opgestelde wensen en eisen geverifieerd en daarnaast zijn er nog enkele nieuwe wensen en eisen boven tafel gekomen. De requirements zijn gedurende het project constant bijgewerkt en uitgebreid aan de hand van de ontvangen feedback.

5.3 Use cases

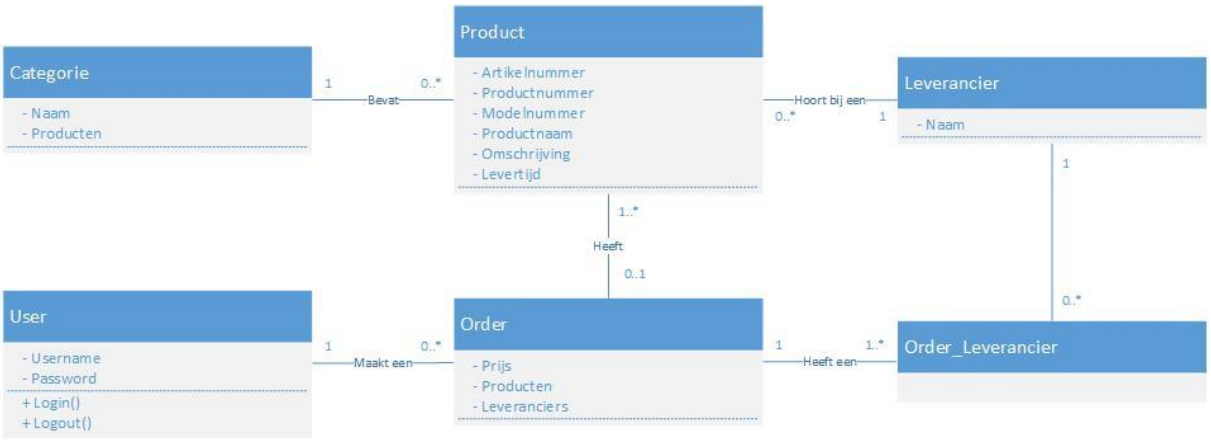
Met behulp van een use case diagram wilde ik de wensen en eisen van de opdrachtgever beter in kaart brengen. Verder heb ik de use case gemaakt als houvast voor mijzelf tijdens het afstudeertraject. In figuur 8 vindt u de uitwerkingen van mijn use case diagram.



Figuur 8: Use case diagram.

5.4 Eerste versie technisch ontwerp

De eerste versie van het klassendiagram is opgebouwd op een ontwerp van hoe ik in week 2/3 dacht hoe het systeem er uit kwam te zien. Dit is dus niet de definitieve versie, de definitieve versie kunt u vinden in de elaboration fase. In figuur 9 ziet u de eerste versie van mijn klassendiagram.



Figuur 9: Eerste opzet klassendiagram.

Klasse	**Omschrijving**
Product	De klasse product bevat alle informatie over een product. Een product hoort altijd bij 1 categorie. Een product hoort ook altijd bij 1 leverancier. Verder kan een product bij 0 of 1 orders horen.
Order	De klasse order bevat alle informatie met betrekking tot één order. Een order hoort altijd bij 1 user. Verder kan een order meerdere producten bevatten. Een order kan producten van meerdere leveranciers bevatten.
Order_Leverancier	De klasse order_leverancier is de tussentabel voor order en leverancier. Naast een combinatie van een leverancierId en orderId bevat deze geen extra informatie.
Leverancier	De klasse leverancier bevat alle informatie over een leverancier. Een leverancier heeft meerdere producten. Daarnaast kan een leverancier meerdere orders tegelijk open hebben staan.
User	De klasse user bevat informatie van de ingelogde gebruiker. Een user kan meerdere order plaatsen.
Categorie	De klasse categorie bevat informatie over een categorie. Een categorie kan meerdere producten bevatten.

6 Uitvoering: Elaboration Fase

In dit hoofdstuk wordt de elaboration fase besproken. Tijdens de elaboration fase zijn de volgende activiteiten uitgevoerd:

- Analyse koppelingen leveranciers
- Ontwerpen design klassen diagram
- Ontwerpen producten import
- Ontwerpen sequentie diagrammen
- Ontwerpen database model

6.1 Analyse koppelingen leveranciers

De elaboration fase ben ik begonnen met een analyse van de mogelijkheden die de leveranciers bieden tot het opvragen van product data. Per leverancier ben ik de website afgegaan en heb ik gekeken of er informatie viel te vinden over het leggen van een zogenaamde koppeling.

In de onderstaande tabel worden mijn bevindingen van deze analyse kort weergegeven. Per leverancier wordt er weergegeven welke data beschikbaar wordt gesteld via een koppeling. Een “Y” betekend dat de data beschikbaar is en een “?” betekend dat er geen informatie is gevonden over dit onderdeel.

	Techdata	**Aces Direct**	**Copaco**	**Xeptor**	**Central Point**	**Travion**
Producten lijst	Y	?	Y	?	?	Y
Prijzen updates	Y	?	Y	?	?	Y
Levertijd	Y	?	Y	?	?	Y
Aantal producten beschikbaar	Y	?	Y	?	?	Y
Promo prijzen	Y	?	?	?	?	?
Bestand formaat	CSV	?	CSV / XML	?	?	?

Techdata voldoet aan al mijn opgestelde eisen. Daarnaast biedt Techdata ook nog een uitgebreide omschrijving met het leggen tot de koppeling. Deze omschrijving kunt u terugvinden in bijlage Techdata handleiding. Techdata biedt zijn informatie aan via meerdere CSV bestanden. Op de website van Copaco heb ik ook informatie kunnen vinden met betrekking tot het leggen van een koppeling. Aan de hand van de informatie die zij beschikbaar stellen kan ik opmaken dat ze geen promotie prijzen hanteren. Copaco geeft aan dat ze zowel CSV als XML bestanden kunnen aanleveren. Op de website van Travion heb ik ook informatie kunnen vinden met betrekking tot het leggen van een koppeling. Net als Copaco zegt ook Travion niks over promotie prijzen. Travion geeft aan de bestanden via diverse formaten te kunnen leveren, maar welke dit precies zijn staat daar niet bij.

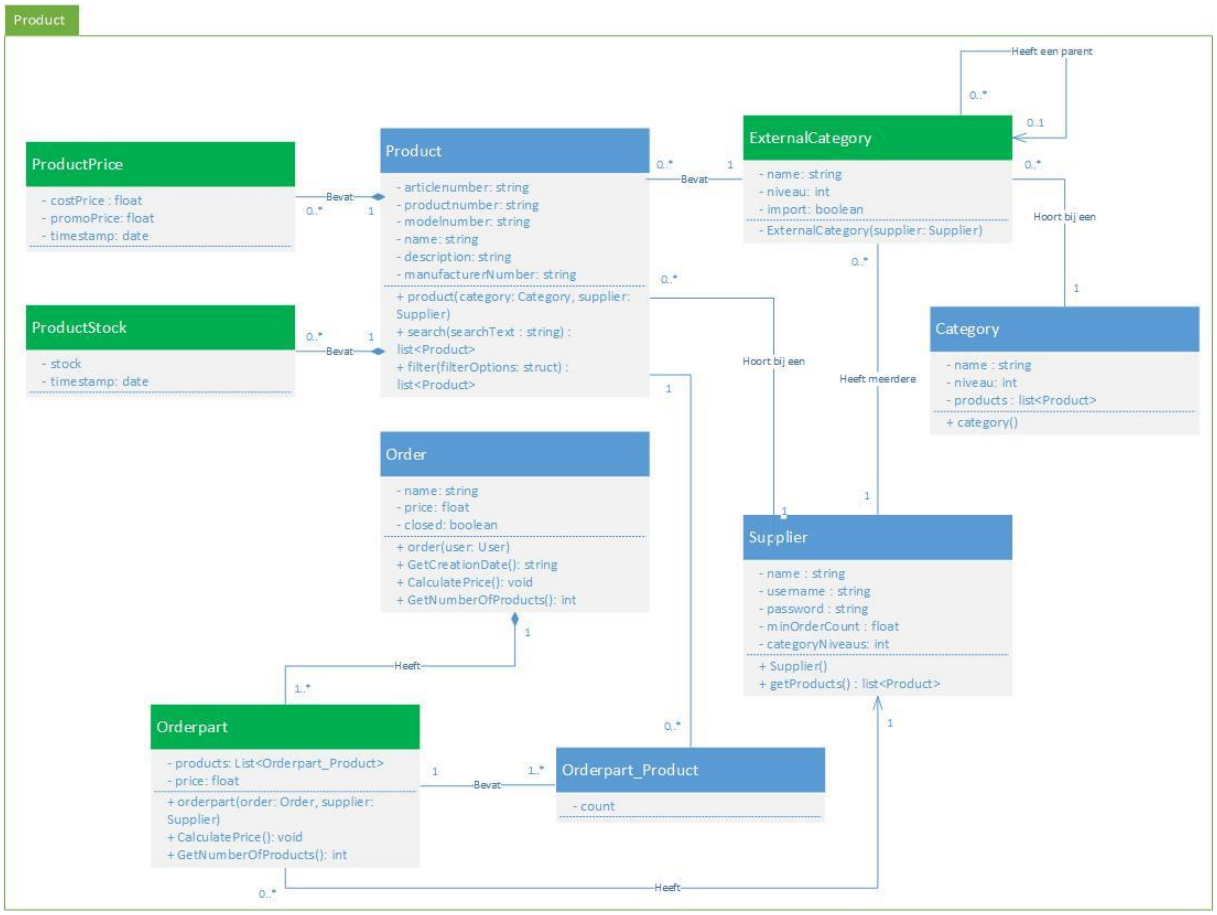
Ik heb mijn bevindingen vervolgens met de opdrachtgever besproken. De opdrachtgever is vervolgens meer informatie gaan opvragen bij de leveranciers via zijn vaste contactpersoon. Met elke leverancier hebben we dus contact opgenomen aan de hand van de bevindingen. Verder weet ik aan de hand van mijn bevindingen dat waarschijnlijk niet elke leverancier promotieprijzen hanteert. Hier houdt ik dus rekening mee in mijn ontwerpen.

6.2 Ontwerpen design klassendiagram

Het design klassendiagram omschrijft hoe de model classes met elkaar verbonden zijn. Ik heb in mijn ontwerp onderscheid gemaakt tussen twee packages. De package product en de package user.

6.2.1 Product

De package Product bevat alle belangrijkste componenten van het te bouwen systeem. In figuur 10 vindt u de uitwerking van de package product. De klassen met de groene kopjes zijn nieuw toegevoegd en de klassen met de blauwe kopjes bestonden al uit het eerste concept klassendiagram.



Figuur 10: Design klassendiagram, package: product

Product

De relatie tussen product en categorie is gewijzigd. Er is namelijk een externe categorie tussen gekomen. Een product heeft een relatie met een externe categorie. Een product hoort altijd bij maar één externe categorie. Verder wordt van een product nu prijs en voorraad geschiedenis bijgehouden. Een product wordt nooit geheel verwijderd uit de database, maar wel wordt er bijgehouden wanneer deze verwijderd is. Dit heb ik zo ontworpen omdat de opdrachtgever verwijderde producten nog wel terug wilt zien in het producten overzicht. Daarnaast zouden de aangemaakte orders worden aangepast als een product geheel wordt verwijderd en dit is niet de bedoeling.

ProductPrice

De klasse productprice bevat prijsinformatie van een Product. Met behulp van deze klasse kan er gemakkelijk een prijsgeschiedenis worden bijgehouden. Er wordt namelijk bijgehouden wanneer de prijsinformatie is toegevoegd. Door het scheiden van de prijsinformatie van de product hoef ik niet voor elke prijswijziging de bijbehorende Product aan te passen, er wordt enkel een nieuwe record toegevoegd aan de tabel productprice.

ProductStock
De klasse productstock bevat informatie over de voorraad van de voorraad van een Product. Deze klasse werkt in principe hetzelfde als de productprice klasse. We kunnen nu de gemakkelijk een voorraadgeschiedenis gaan bijhouden. De belangrijkste reden voor het scheiden van de voorraad van de Product is zodat we de product niet voor elke voorraad wijziging hoeven aan te passen, er wordt enkel een nieuwe record toegevoegd aan de tabel productstock.

Order
De klasse order is sinds het vorige ontwerp enigszins aangepast. Een order heeft niet langer een relatie met de klassen product. Wel heb ik een nieuwe relatie toegevoegd, namelijk een relatie met de klasse orderpart. Een order bestaat uit 1 of meer orderparts.

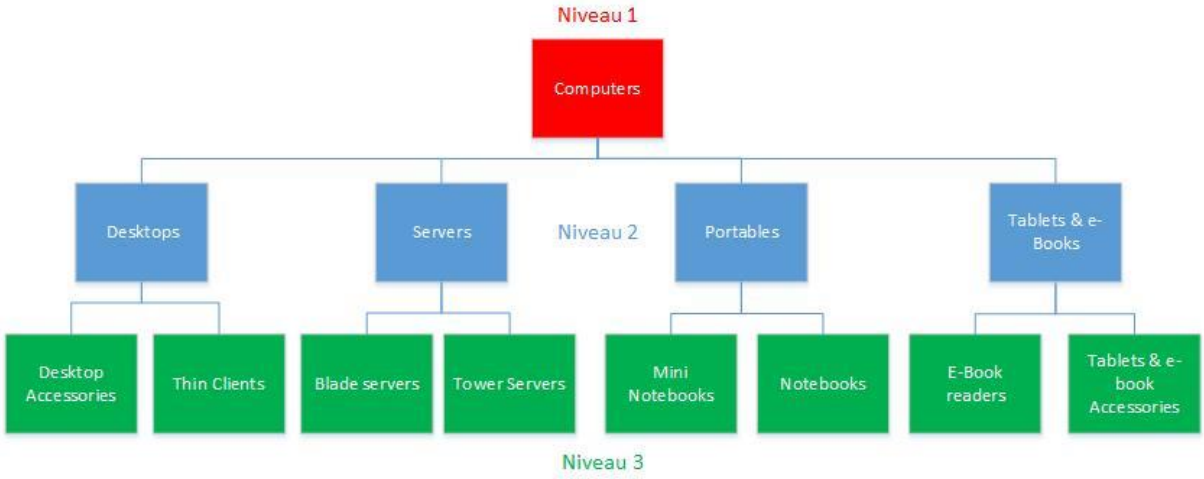
Orderpart
De klasse orderpart bevat alle informatie met betrekking tot een deel van een order. Een orderpart is eigenlijk een order, maar dan maar voor één leverancier. Een orderpart hoort dus altijd bij één leverancier. Verder kan een orderpart één of meer producten bevatten. Een orderpart hoort altijd bij maar één order. Zodra de bijhorende order ophoudt met bestaan heeft het bestaan van de orderpart geen meerwaarde meer.

Orderpart_Product
De klasse orderpart_product is de tussentabel tussen klasse product en orderpart. Deze klasse houdt bij welke producten aan welke orderpart zijn gekoppeld. Daarnaast wordt met de attribuut count bijgehouden hoevaak één product voorkomt in één orderpart.

Category
De klasse categorie is ook aangepast sinds het eerste ontwerp. Een categorie heeft niet langer een directe relatie met een product. Een categorie heeft nu een relatie met een externe categorie gekregen. Bij 1 categorie kunnen 0 of meer externe categorieën horen. Aan de hand van een categorie kunnen ook producten worden opgezocht. Een categorie is ook wel een interne categorie genoemd.

ExternalCategory
De klasse externalcategory bevat informatie over een categorie zoals de leveranciers deze aanleveren. De leveranciers gebruiken helaas allemaal net iets andere categorieën. Daarvoor is deze klasse aangemaakt. Een externalcategory wordt gekoppeld aan één (interne) categorie. Door het koppelen van de door de leverancier aangeleverde categorieën aan onze eigen categorieën kunnen we in het producten overzicht zoeken op interne categorieën. Zonder deze koppeling is het voor mij lastig om te bepalen waar een product nou onder valt. Ook heb ik deze koppeling nodig voor het bepalen van bij welke categorie een product hoort. De leveranciers gebruiken natuurlijk allemaal andere id’s voor de koppelingen. Daarnaast kan een externalcategory ook nog gekoppeld worden aan een andere externalcategory. Een externalcategory kan maar één parent externalcategory hebben. Wel kan een externalcategory meerdere sub externalcategories bevatten.

Een externalcategory heeft altijd een niveau. Er zijn in totaal 3 niveaus, waarvan niveau 1 het hoogste niveau is en niveau 3 het laagste niveau. In figuur 11 kunt u zien hoe ik deze structuur binnen mijn applicatie toepas.



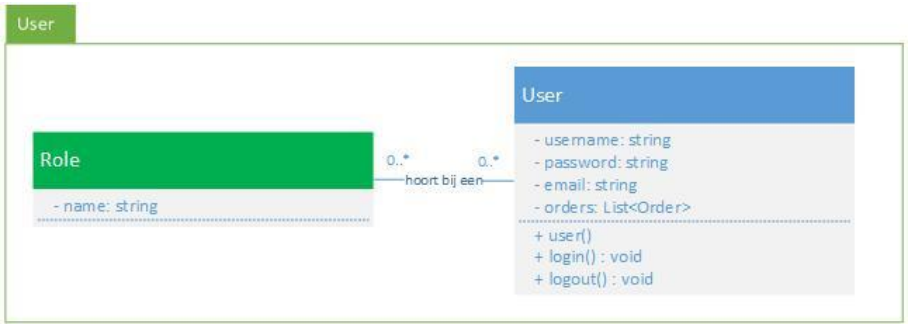
Figuur 11: Voorbeeld structuur externe categorieën

De leveranciers koppelen hun producten altijd aan een categorie met het niveau 3. Het lastige in dit verhaal was dat niet elke leverancier dezelfde structuur aanhield voor hun categorieën. Zo heeft de ene leverancier 3 categorie niveaus en de andere maar 2. Door gebruik te maken van deze boomstructuur maakt dit in principe niet heel erg veel uit. Zolang de producten maar kunnen worden gekoppeld aan de categorie met het laagste niveau. Vervolgens kan ik van deze categorie opvragen of hij een bovenliggende categorie heeft. Voor de bovenstaande categorie geldt weer precies hetzelfde. Dit zorgt ervoor dat het systeem gemakkelijk uit te breiden is naar een structuur met nog een extra laag niveaus.

6.2.2 User

De package user bestaat uit de klasse User en Role. Beide klassen zijn automatisch gegenereerd door het Entity Framework. Het Entity Framework biedt standaard de mogelijkheid om authenticatie in een applicatie toe te voegen. Ik heb ervoor gekozen om hier gebruik van te maken. Er is in eerste instantie maar één gebruiker. Wel is het de bedoeling dat in de toekomst meer gebruikers met het systeem gaan werken, hier wil ik dus alvast rekening mee gaan houden.

In figuur 12 vindt u de uitwerking van de package user.



Figuur 12: Design klassendiagram, package: User

User

De klasse User bevat informatie over de ingelogde gebruiker. Daarnaast wordt een user gekoppeld aan een Order. Je kan van een User dus opvragen welke Orders hij allemaal heeft aangemaakt.

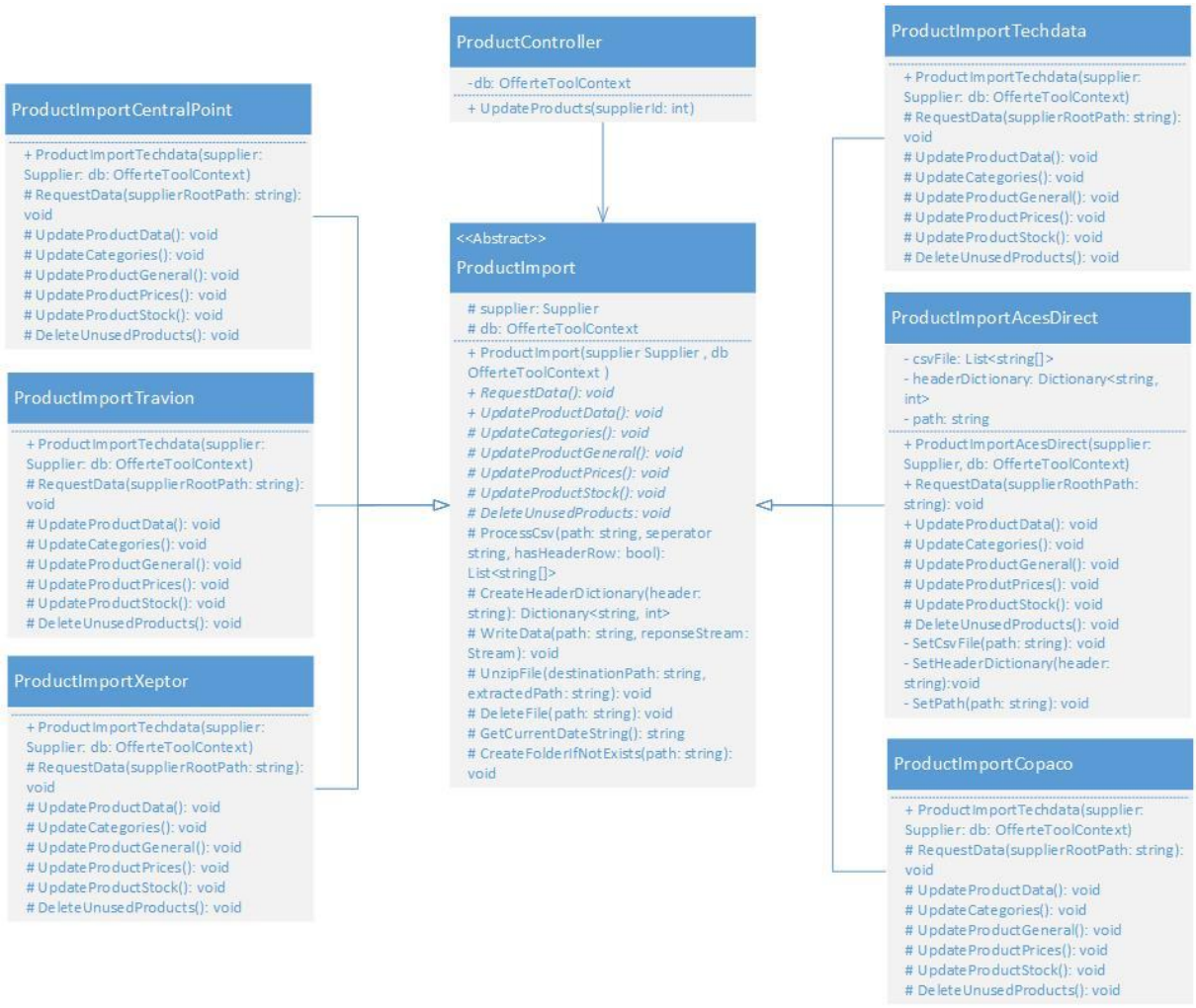
Role

De klasse Role bevat informatie over de aanwezige Rollen in de applicatie. Eén Role kan bij meerdere Users horen. Ook kan een User meerdere Rollen hebben.

6.3 Ontwerpen producten import

Elke leverancier levert zijn data net weer iets anders aan, maar er zijn zeker ook overeenkomsten. Om dit alles zo goed mogelijk te implementeren heb ik ervoor gekozen om dit met behulp van een design pattern te implementeren. De twee design patterns die mij gelijk het meeste aanspraken waren het strategy pattern en de template method pattern. Na wat vergelijkingen tussen deze twee ben ik tot de conclusie gekomen dat de template method pattern de beste keuze is voor deze situatie. Ik wil namelijk verplichten dat de overervende klassen bepaalde functies bevatten, maar daarnaast wil ik ook in de base klasse een aantal functies uitwerken die vervolgens door de subklassen worden gebruikt. Het uitwerken van functies is niet mogelijk in een interface, vandaar dat de template method pattern de voorkeur krijgt.

In figuur 13 heb ik mijn uitwerking van het template method pattern weergegeven in een klassendiagram.



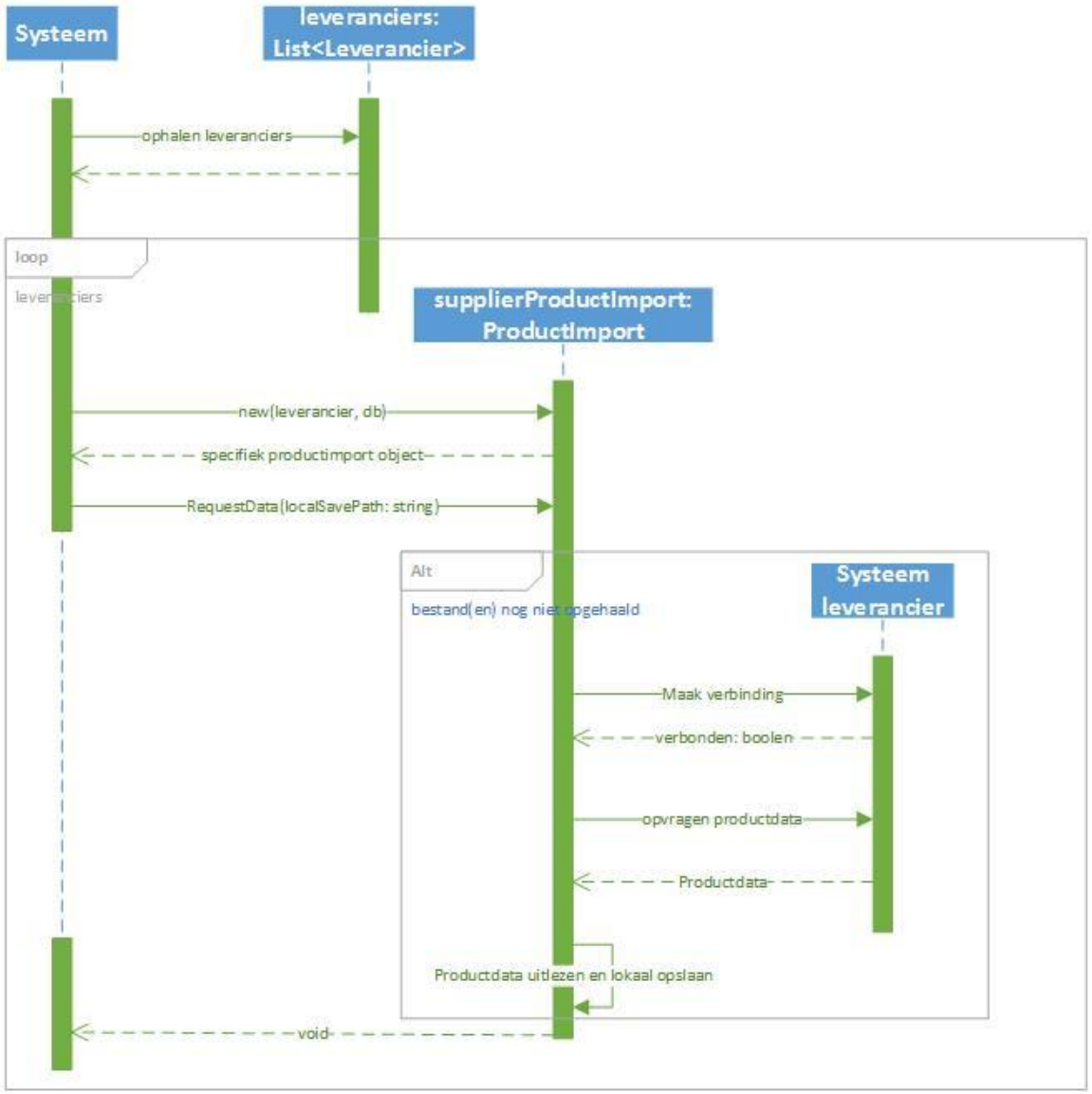
Figuur 13: Product import klassendiagram

De geïmplementeerde functies in de subklassen zien er allemaal anders uit qua uitwerking. Wel roepen ze allemaal dezelfde manager functies aan. De subklassen zorgen ervoor dat de data hetzelfde aankomt bij de manager, de logica staat dus in de uitwerking van de subklassen. De manager zorgt er alleen voor dat de data in de database wordt toegevoegd.

6.4 Ontwerpen sequentie diagrammen

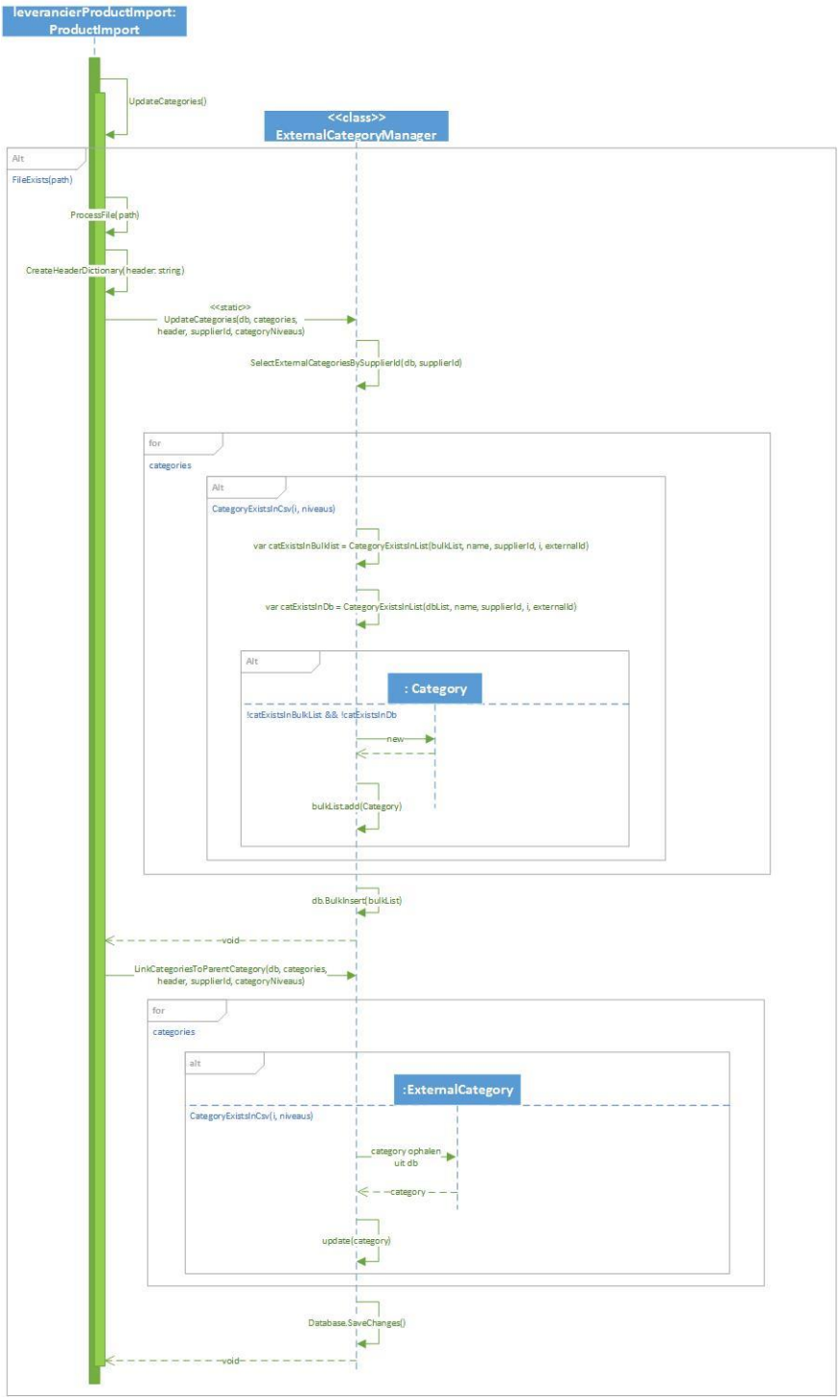
Ik zal niet alle sequentie diagrammen doornemen, aangezien dit een hele lijst is, maar ik bespreek hier de naar mijn mening interessantste twee.

De eerste sequentie diagram beschrijft het proces van het ophalen van de productdata bij de leveranciers. De product data zal 1x per dag worden opgehaald op een vastgesteld tijdstip. Het proces begint met het ophalen van de aanwezige leveranciers. Vervolgens wordt er per leverancier de volgende acties uitgevoerd. Er wordt eerst een nieuw ProductImport object aangemaakt van de desbetreffende leverancier. Vervolgens wordt de functie RequestData() aangeroepen, deze functie heeft als parameter het pad waar bestanden moet worden opgeslagen. Er wordt daarna gecontroleerd of er al bestanden aanwezig zijn op het meegegeven pad, als dit niet het geval is wordt er verbinding gemaakt met de leverancier. Zodra de verbinding tot stand is gekomen worden de gegevens opgehaald en lokaal weggeschreven. Het proces is voor alle leveranciers vergelijkbaar, alleen de manier van verbinden en de ontvangen datatype zal verschillen. In figuur 14 is de sequentie diagram weergegeven.



Figuur 14: Ophalen product informatie bij leveranciers

Het tweede sequentie diagram beschrijft het bijwerken van de categorieën aan de hand van de opgehaalde product data. Er wordt eerst gecontroleerd of er een bestand aanwezig is op het aangegeven pad. Als dit het geval is wordt de file uitgelezen en omgezet naar een lijst met objecten. Vervolgens wordt de functie CreateHeaderDictionary(string) aangeroepen, deze functie maakt van de meegegeven string een dictionary met als key een string en als value een integer. Dit zorgt ervoor dat ik mijn objecte kan uitlezen met string keys in plaats van “magic numbers”. Vervolgens wordt de functie UpdateCategories() van ExternalCategoryManager aangeroepen. Binnen deze functie wordt geloopt over de ExternalCategory objecten en een object wordt aan de database toegevoegd als deze nog niet bestond. Nadat de objecten zijn toegevoegd kunnen de ExternalCategories gelinkt worden aan parents als ze tenminste een parent hebben. Dit wordt gedaan door de functie LinkCategoriesToParentCategory(). In figuur 15 is dit weergegeven in een sequentie diagram.



Figuur 15: Categorieën bijwerken

Voor de volledige lijst van sequentie diagrammen en de daar bijbehorende omschrijvingen verwijs ik u door naar hoofdstuk 4 in mijn elaboration rapport.

6.5 Ontwerpen database model

Aangezien ik er voor heb gekozen om gebruik te maken van het entity framework in combinatie met code first was mijn database ontwerp overbodig geworden. Deze heb ik dan ook na mijn eerste ontwerp niet verder bijgewerkt. Het leek mij dan ook niet verstandig om deze in mijn verslag te plaatsen. Mocht u mijn ontwerp toch nog willen zien kunt u deze vinden in het elaboration rapport onder het hoofdstuk “database ontwerp”.

7 Uitvoering: Construction Fase (de basis)

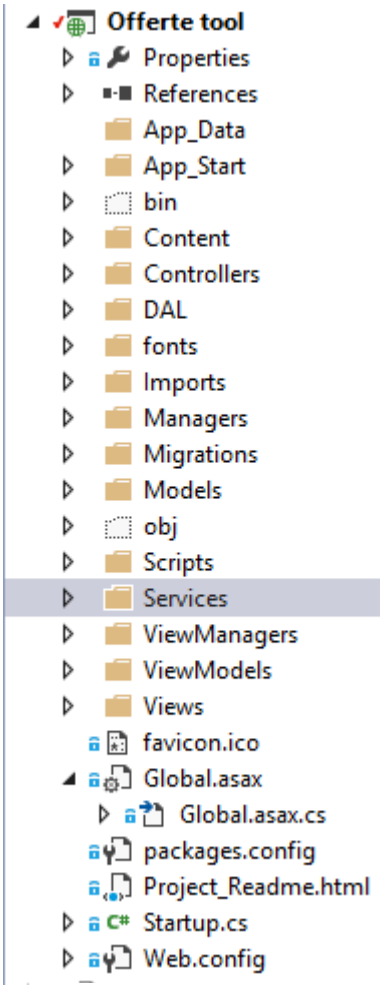
De eerste keer dat de construction fase wordt doorgelopen wordt “de basis” van het systeem gebouwd. In de eerste construction fase komen vooral de must have requirements aan bod. Aan het eind van de construction fase heeft de opdrachtgever al een bruikbaar product. De basis van het systeem bestaat uit de volgende onderdelen:

- Opzetten van het visual studio project met code first.
    - Mvc structuur aanmaken
    - Code first configureren
    - Bootstrap configureren
    - Git configureren
- Koppelingen maken met Techdata en AcesDirect.
    - Product gegevens voor de eerste keer ophalen
    - Product prijzen bijwerken
    - Product voorraad bijwerken
    - Categorieën ophalen en koppelen aan hoofd categorieën
    - Leverancier categorieën koppelen aan interne categorieën
- Een overzicht van producten
    - Zoeken op producten
    - Filteren op categorie en product
    - Filteren op leverancier
    - Product details
- Interne categorie module
    - Met CRUD functionaliteiten
- Interne categorieën koppelen aan externe categorieën.
- Prijs geschiedenis
- Voorraad geschiedenis

7.1 Opzetten Project in Visual studio 2013

Ik ben het software ontwikkeltraject begonnen met het opzetten van mijn Visual studio project. Ik ben begonnen met het aanmaken van de structuur. In figuur 16 ziet u de mappen van mijn project.

Map	Omschrijving
App_Data	Wordt standaard aangemaakt, bevat geen bestanden.
App_Start	Bevat informatie over de mapping van models naar viewsmodels en andersom.
Content	Bevat de CSS bestanden van onder andere Bootstrap.
Controllers	Hier staan de uitwerkingen van de controllers
DAL	DAL staat voor Data Access Layer, in dit mapje staat de DbContext klasse met de bijbehorende tabellen en mappings.
Fonts	Plaatjes van Bootstrap worden hier opgeslagen.
Imports	Wordt gebruikt om de product data bestanden van de leveranciers in op te slaan. Per leverancier komt er een apart mapje in de map Imports.
Managers	De functies die de database aanspraken worden hier geplaatst. Bijvoorbeeld de CRUD functies komen hier.
Migrations	Het mapje migrations is aangemaakt door het Entity framework voor de code first migraties. De aangemaakte migratie bestanden komen in deze map terecht.
Models	Bevat de model klassen.
Scripts	Javascripts bestanden worden hier opgeslagen.
Services	Hier komen de klasse die betrekking hebben tot het ophalen van de data van de leveranciers.
ViewManagers	Services is een mapje die ik zelf heb aangemaakt, ik ben van plan om hier de code te plaatsen voor het maken van de koppelingen naar de leveranciers.
ViewModels	Zie het kopje viewmanagers
Views	Zie het kopje viewmodels



Figuur 16: Mappenstrctuur

7.1.1 Viewmanagers

In het mapje ViewManager komen de ViewManagers, een ViewManager maakt de Viewmodels aan en zorgt voor een minder sterke koppeling tussen de ViewModel objecten (loose coupling). Wanneer Object A bijvoorbeeld een instantie van Object B moet aanmaken dan doet niet Object A dit maar de ViewManager van Object A. Dit wordt ook wel een vorm van het Mediator pattern genoemd.

7.1.2 Viewmodels

In het mapje ViewModels komen de ViewModel klassen. Een ViewModel is een extra laag tussen de View en de Model. Een ViewModel hoort altijd bij maar één model klasse. Door het toevoegen van de ViewModel klassen haal ik de logica uit mijn views. Daarnaast is het werken met ViewModels veiliger, je stuurt alleen nog maar de data die je wilt tonen naar de View en geen onnodige dingen die een Model object normaal wel mee zou sturen. Daarnaast is de koppeling tussen model en de

view minder sterk geworden. Ik kan nu mijn Model aanpassen zonder dat mijn Views ook aangepast hoeven te worden.

In figuur 17 wordt de communicatie tussen de views, viewsmodels en model klassen weergegeven.

Figuur 17: ViewModel [http://en.wikipedia.org/wiki/Model_View_ViewModel]

Het laatste mapje is Views, in dit mapje komen natuurlijk de views die de website zullen weergeven.

Ik maak binnen mijn project eigenlijk gebruik van het MVVM (Model View ViewModel) pattern. Interpulse gebruikt dit binnen al haar nieuwe ASP.net MVC projecten als standaard. Het MVVM pattern is gebaseerd op het MVC pattern. Het verschil is echter dat bij het MVVM pattern een view niet meer direct de data van een model object uitleest maar van een viewmodel. Met andere worden er is dus een extra laag tussen gekomen.

7.2 Code first configureren

Nadat de structuur van het project was neer gezet ben ik verder gegaan met het configureren van code first. Voordat ik mijn database kan laten aanmaken met code first moest ik eerst al mijn model klassen aanmaken. Ik heb dus al mijn model klassen aangemaakt zoals ik ze had ontworpen in de elaboration fase. Vervolgens heb ik migrations aangezet, zodat ik met het entityframework de eerste migratie kan aanmaken.

Het toevoegen van de eerste migratie gaf echter problemen, de migratie kan niet worden aangemaakt vanwege een cascade delete error tussen de tabel Product en Orderpart_Product. Waarom ik deze error kreeg is mij niet geheel duidelijk, maar ik heb dit opgelost door handmatig een aanpassing te maken in de migratiefile. Nadat ik on delete cascade uit had gezet voor ProductId in Orderpart_Product werkte de migratie wel. Deze aanpassing zorgt echter weer voor een ander probleem. De tussentabel wordt niet meer geleegd wanneer een product wordt verwijderd. Dit heb ik opgelost door een trigger te maken die na het verwijderen van een product ook de tussentabel leegt.

7.3 Bootstrap configureren

Het in eerste instantie configureren van Bootstrap is vrij eenvoudig. Ik heb deze gewoon toegevoegd met behulp van de NuGet package manager binnen Visual Studio. NuGet is een package management systeem voor .NET. Met NuGet kan je gemakkelijk externe libaries aan een Visual Studio project toevoegen. Daarnaast kan je van geïnstalleerde libaries snel zien of er updates voor beschikbaar zijn. Als je ervoor kiest om een update binnen te halen worden de bestanden automatisch bijgewerkt.

7.4 GIT configureren

Binnen Interpulse wordt sinds kort gebruik gemaakt van GIT als standaard versiebeheer tool. Ik heb met behulp van een collega voor mijn afstudeerproject een GIT repository opgezet. Zodra een stuk functionaliteit klaar was zorgde ik ervoor dat dit naar de GIT server werd gepushed. Hierdoor staat het grootste deel van mijn code altijd veilig. Mocht mijn werk computer ermee ophouden heb ik het grootste deel van mijn werk in ieder geval nog op de GIT server staan.

7.5 Bouwen eerste functionaliteit

Voor de eerste opzet van de views en de controllers binnen de applicatie heb ik gebruik gemaakt van scaffolding. Het entity framework maakt dan de CRUD views en controller functies voor je aan. De aangemaakte functies en views zijn echter wel beperkt, deze heb ik dan ook in de loop van het project verder uitgewerkt. Ik heb er meer gebruik van gemaakt als eerste opzet en om een beetje aan het MVVM pattern te wennen.

Het plan was om vervolgens verder te gaan met het bouwen van de eerste koppeling, namelijk de koppeling met Techdata. Dit kon alleen nog niet, omdat we moesten wachten op de FTP login gegevens van Techdata. In plaats daarvan ben ik begonnen met het aanmaken van dummy data om hier vervolgens mee aan de slag te gaan. De dummy data liet ik aanmaken door de Seed methode. De Seed methode is een methode van de klasse DbContext die wordt aangeroepen bij het aanmaken of wijzigen van de database.

7.6 Leveranciers module

Ik ben begonnen met de kleinste module namelijk die van de leveranciers. Ik heb de basis functionaliteiten die aanwezig waren na het scaffolding aangepast en verder uitgebreid naar mijn eigen wens. De leverancier module bestaat uit een overzicht, toevoegen, bewerk en detail scherm. Verder niet heel erg bijzonder, wat wel interessant is dat je per leverancier kan zien hoeveel producten deze in de database heeft staan. Dit is niet iets wat je uit het model object kan halen, maar dit heb ik zelf als property aan de viewmodel voor leverancier overzicht toegevoegd. Vervolgens heb ik dit aantal in de viewmanager opgehaald en aan de viewmodel toegevoegd. In figuur 18 zie je de functie die de lijst van leveranciers opbouwt.

```
public List<SupplierViewModelGrid> CreateGrid(OfferteToolContext db)
{                                                    1
    var suppliers = SupplierManager.Instance.SelectAll(db).ToList();
    var model = suppliers.Select(a => new SupplierViewModelGrid()
    {                                                    2
        Id = a.Id,
        Name = a.Name,                                   3
        NumberOfProducts = ProductManager.Instance.SelectBySupplierId(db, a.Id).Count(),
        CategoryNiveaus = a.CategoryNiveaus,
        MinimumOrderAmount = a.MinimumOrderAmount
    }).ToList();
                                                         4
    return model;
}
```

Figuur 18: Leveranciers overzicht, viewmanger code

Ik haal eerst de beschikbare leverancier uit de database, dit doe ik via de SupplierManager. Zoals u kunt zien roep ik ook “Instance”aan (bij nummertje 1). Dit stukje zorgt ervoor dat er een statische instantie van de klasse SupplierManager wordt opgehaald. Via deze statische instantie kan ik vervolgens alle functies van de betreffende klasse aanroepen zonder een object aan te maken. Dit concept gebruik ik voor mijn manager en viewmanager klassen. In figuur 19 kunt u zien hoe ik deze constructie heb toegepast voor de manager klassen. In de BaseManager wordt deze constructie opgezet, vervolgens overerven allen subklassen van de BaseManager. In de BaseManager is alles op een generieke manier opgezet zodat alle subklassen er gebruik van kunnen maken.

1. Bij punt 1 wordt de juiste tabel gezet voor de betreffende entiteit.
2. Hier wordt de bijbehorende model klassen gezet.
3. De entiteit wordt gezet.

4. Maakt een nieuwe instantie aan van de desbetreffende klasse.
5. Return de aangemaakte instantie.

```
public abstract class BaseManager<TEntities, TEntity, TSelf>
    where TEntities : DbContext 1
    where TEntity : class 2
    where TSelf : BaseManager<TEntities, TEntity, TSelf>, new()
{       3
    private static TSelf _instance = new TSelf(); 4


    #region Properties

    public static TSelf Instance
    {
        get
        {
            return _instance; 5
        }
    }

    #endregion Properties
```

Figuur 19: Gebruik instance

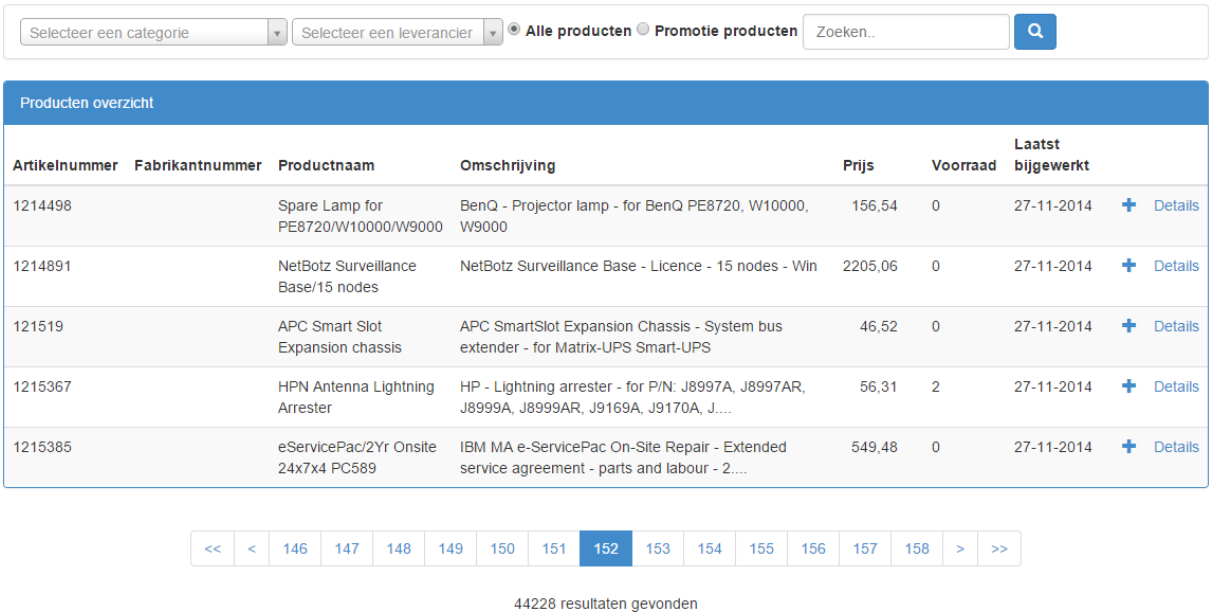
Als we weer verder gaan met figuur 18 zie je dat ik bij punt 2 een SupplierViewModelGrid object aanmaak. Dit is een viewmodel klassen voor het overzicht van de leveranciers. Voor elke aanwezige leverancier wordt dit viewmodel object aangemaakt en gevuld. Vervolgens wordt dit ViewModel object gebruikt om informatie van een leverancier weer te geven in de tabel. Eén SupplierViewModelGrid staat gelijk aan één regel in de tabel van leveranciers. Bij punt 3 haal ik via een andere manager klasse het aantal bijbehorende producten van een leverancier op. Punt 4 is bevat de LINQ query die ik al gedeeltelijk had omschreven, ik loop dus over de lijst van leveranciers. Vervolgens wordt er per leverancier een viewmodel object aangemaakt.

Wat nog wel interessant is om te melden is dat ik de lijst van leveranciers (1^{e} regel) gelijk omzet naar een lijst. Normaal wil je dat niet doen, omdat dit slecht is voor de performance. Elke regel uit de query moet namelijk toegevoegd worden aan een lijst. Maar aangezien het maar een lijst is van enkele leveranciers maakt dit niet echt uit. Maar dat is niet de reden dat ik het zo gedaan heb, de reden is namelijk de aanroep van een functie in een lambda expressie. De regel bij nummer 3 had een error gegeven als ik de leveranciers niet van een IQueryable had omgezet naar een list of array. De reden voor deze error is dat het entity framework probeert de LINQ Query om te zetten naar SQL. Alleen hij kent de methode die aangeroepen wordt niet. Dit los je dus op door eerst de data in een list te zetten en vervolgens een LINQ query op de list uitvoeren.

7.7 Producten overzicht

Vervolgens ben ik begonnen met een overzicht op te zetten voor producten. Het producten overzicht bestaat uit een tabel met de producten, een pager, een zoekveld, een filter voor categorieën en een filter op leverancier.

In figuur 20 kunt u zien hoe het producten overzicht eruit ziet, voor dit voorbeeld toon ik echter maar 5 records in plaats van 20 records per pagina, anders wordt het plaatje wel erg groot.



Figuur 20: Producten overzicht

De zoek functie voor producten heb ik als volgt opgebouwd. Ik haal eerst alle producten met het entity framework op. Vervolgens ga ik per filteroptie een LINQ where query uitvoeren. Ik gebruik eigenlijk door mijn hele applicatie LINQ queries met daarin lambda expressie. Ik vind de lambda expressies fijner lezen dan de standaard LINQ expressie.

De zoekfunctie voor producten ziet er uit zoals in figuur 21 weergegeven.

```
public IQueryable<Product>  SelectBySearch(OfferteToolContext db, ProductFilterOptions filterOptions)
{
    var products = this.SelectSecure(db);

    if (filterOptions.CategoryId > 0)
    {
        products = SelectByCategoryId(db, filterOptions.CategoryId, products); 1
    }
    if (filterOptions.SupplierId > 0)
    {
        products = products.Where(a => a.SupplierId == filterOptions.SupplierId);
    }
    if (!String.IsNullOrEmpty(filterOptions.Q))
    {
        int unusedInt;
        if (Int32.TryParse(filterOptions.Q, out unusedInt))
        {
            products = products.Where(a => a.Articlenumber.Contains(filterOptions.Q));
        }
        else
        {
            products = products.Where(a => a.Manufacturernumber.Contains(filterOptions.Q) ||
                a.Productname.Contains(filterOptions.Q) || a.Description.Contains(filterOptions.Q));
        }

    }
    if (filterOptions.PromotionPrices == 1)
    {
        products = products.Where(a => a.ProductPrices.OrderByDescending(b => b.Id).Take(1).FirstOrDefault
    }

    return products;
}
```

Figuur 21: Products zoek functie

Er is nog wel iets bijzonders aan deze functie, bij nummertje 1 wordt er nog een andere functie aangeroepen, namelijk SelectByCategoryId(). Met deze functie haal ik alle producten op die zijn gekoppeld aan de geselecteerde categorie. Dit zit echter wel wat lastiger in elkaar dan het in eerste instantie klinkt. Een product is namelijk niet direct gekoppeld aan een categorie maar aan een externe categorie met niveau 3. Ik zal dit iets verder in het verhaal nog verder toelichten.

In figuur 22 vind u de uitwerking van de SelectByCategoryId methode. Zoals u kunt zien roept deze methode weer een andere methode aan, namelijk SelectChildExternslCategoryIds(). Ik heb de logica van het ophalen van de externe categorieën die horen bij een categorieën gescheiden. Ik verwacht namelijk dat ik dit nog wel vaker nodig zal hebben, daarnaast is de code op deze manier ook beter te overzien.

1. Ik geef een optionele parameter aan de functie mee. Products wordt op null gezet wanneer deze niet is meegegeven in de arguments.
2. Ik check met de short if notatie of de producten zijn meegestuurd in de arguments, als dit niet het geval is haal ik ze op.
3. Er wordt een functie aangeroepen die van een categorie al zijn gekoppelde externe categorieën ids ophaalt. Deze functie zal ik zo nog verder uitwerken en uitleggen.
4. Filtert de productenlijst aan de hand van de opgehaalde externe categorie ids.
5. Dit is een fallback voor wanneer er geen ids gekoppeld waren aan de gekozen categorie.

```
public IQueryable<Product> SelectByCategoryId(OfferteToolContext db, int categoryId,
    IQueryable<Product> products = null) 1
{
    // use the existing products query or create a new one if there is no existing one.
    IQueryable<Product> productsQuery = (products != null) ? (IQueryable<Product>)products : this.SelectAll(db); 2

    var externalCategoryIds = ExternalCategoryManager.Instance.SelectChildExternalCategoryIds(db, categoryId); 3

    if (externalCategoryIds != null)
    {
        productsQuery = productsQuery.Where(a => externalCategoryIds.Contains(a.ExternalCategoryId)); 4
    }
    else // The received category has no linked external categories, return an empty set.
    {
        productsQuery = productsQuery.Where(a => a.ExternalCategoryId == -1); 5
    }

    return productsQuery;
}
```

Figuur 22: Selecteer producten aan de hand van de gekozen categorie

In figuur 23 ziet u de uitwerking voor de methode SelectChildExternalCategoryIds. Ik begin bij nummer 1 met het ophalen van alle externe categorieën uit de database. Vervolgens wordt aan de hand van het meegegeven categoryId bepaald om welk niveau het gaat (nummer 2). Aan de hand van het niveau worden vervolgens weer een aantal stappen doorgelopen. Bij het zoeken op een categorie met niveau 1 worden de onderstaande stappen doorlopen.

Stap 1: Opvragen welke externe categorieën met niveau 1 er allemaal gekoppeld zijn aan de geselecteerde categorie.
Stap 2: Van de opgehaalde externe categorieën met niveau 1 opvragen welke externe categorieën met niveau 2 daar weer aan gekoppeld zijn.
Stap 3: Van alle opgehaalde externe categorieën met niveau 2 opvragen welke externe categorieën met niveau 3 daar weer aan gekoppeld zijn.
Stap 4: Nu heb je een lijst met de benodigde externe categorie ids. Alleen die met niveau 3 dus.
Stap 5: Vervolgens wordt de producten lijst doorgelopen en worden alle producten die gekoppeld zijn aan één van de opgehaalde externe categorieën opgehaald.

Er worden echter niet alleen gefilterd op een categorie met niveau 1, de eindgebruiker wil ook kunnen filteren op categorieën met niveau 2 of 3. Er veranderd verder weinig aan de stappen als we gaan zoeken op een andere niveau. Voor categorieën met niveau 2 hoef je alleen nog de subcategorieën met niveau 3 op te halen. Voor categorieën met niveau 3 hoef je helemaal geen subcategorieën meer op te halen.

```
public IQueryable<int> SelectChildExternalCategoryIds(OfferteToolContext db, int categoryId)
{
    IQueryable<int> externalCategoryIds = null;
    var externalCategories = ExternalCategoryManager.Instance.SelectAll(db);  1
    var niveau = CategoryManager.Instance.SelectById(db, categoryId).Niveau;  2

    switch (niveau)
    {
        case 1:
            externalCategoryIds = externalCategories
                .Where(a => a.Niveau == 3
                    && externalCategories.Any(b => b.Id == a.ParentExternalCategoryId && b.Niveau == 2
                    && externalCategories.Any(c => c.Id == b.ParentExternalCategoryId && c.Niveau == 1
                    && c.CategoryId == categoryId)))
                .Select(a => a.Id);
            break;
        case 2:
            externalCategoryIds = externalCategories
                .Where(a => a.Niveau == 3
                    && externalCategories.Any(b => b.Id == a.ParentExternalCategoryId && b.Niveau == 2
                    && b.CategoryId == categoryId))
                .Select(a => a.Id);
            break;
        case 3:
            externalCategoryIds = externalCategories
                .Where(a => a.Niveau == 3 && a.CategoryId == categoryId)
                .Select(a => a.Id);
            break;
    }
    return externalCategoryIds;
}
```

3

Figuur 23: Ophalen van de externe categorieën die zijn gekoppeld aan een categorie

7.8 Koppeling Techdata

De koppeling met Techdata wordt gelegd via een FTP verbinding. Techdata Nederland gaf aan dit de beste manier was om dagelijks product gegevens bij te werken. Er was voor mij dus niet echt een keuze mogelijkheid om dit op te gaan lossen. Dit is de manier hoe Techdata Nederland informatie aanbiedt aan haar klanten. Techdata zet elke ochtend rond 06:00 een map met product gegevens voor ons klaar. Techdata levert een CSV bestand op voor product gegevens, product prijs, product voorraad, categorieën en producten die buiten gebruik zijn gesteld. We moeten dus in totaal 5 bestanden gaan inlezen om onze product data bij te werken.

Ik heb het koppelen met Techdata opgedeeld in een aantal onderdelen:

- Ophalen van de bestanden op de FTP server
- Bestanden uitlezen
- Gegevens opslaan in de database

7.8.1 Ophalen bestanden op de FTP server

Het ophalen van de bestanden wordt gedaan doormiddel van een FTP URL, username en wachtwoord. Doormiddel van de combinatie van deze 3 kunnen we de bestanden inzien op de FTP server van Techdata. Er wordt verbinding naar de FTP server gemaakt door middel van een

FtpWebRequest. FtpWebRequest is standaard beschikbaar in asp.net MVC 5. In figuur 24 kunt u hoe ik gebruik hem gemaakt van de FtpWebRequest.

```
FtpWebRequest request = (FtpWebRequest)WebRequest.Create(new Uri(fullPath));
request.Credentials = new NetworkCredential(ftpUsername, ftpPassword);
request.UseBinary = true;
request.Method = WebRequestMethods.Ftp.DownloadFile;
```

Figuur 24: FtpWebRequest

Het bovenstaande gedeelte zorgde ervoor dat we konden verbinden met de FTP server, maar we moeten natuurlijk de data er ook nog afhalen. Dit heb ik gedaan door middel van een FtpWebReponse, ook dit object is standaard beschikbaar in asp.net MVC 5. In figuur 25 valt te zien hoe ik hiervan gebruik maak.

```
using (FtpWebResponse reponse = (FtpWebResponse)request.GetResponse())
{
    Stream responseStream = reponse.GetResponseStream();

    // read and write the data to a local directory
    WriteData(destinationPath, responseStream);
}
```

Figuur 25: FtpWebReponse

Zodra de data van het externe systeem naar ons systeem is gedownload moeten we de data nog lokaal gaan opslaan. Dit doe ik met behulp van een FileStream object.

7.8.2 Bestanden uitlezen

Aangezien Techdata 5 losse bestanden aanlevert met betrekking tot de productdata, heb ik ook 5 functies geschreven die de bestanden uitlezen. Een functie voor het bijwerken van de algemene product gegevens, product prijs gegevens, product voorraad gegevens, categorie gegevens en verwijderde producten. In deze functie staat het pad naar de specifieke bestanden. Vervolgens wordt aan de hand van dit pad een bestand opgehaald. Daarna wordt de CSV uitgelezen en in een list gezet. In figuur 26 kunt u zien hoe ik de bestanden uitlees en omzet na een list met array objecten. Met deze functie kan ik verschillende typen CSV uitlezen. Zoals valt te zien bij nummer 1 kan mijn methode met verschillende separators omgaan. Ook houdt de methode er rekening mee of er een header is meegegeven in de CSV of niet. De header haal ik bij nummer 2 eruit mocht deze aanwezig zijn. De list wordt vervolgens doorgegeven naar de manager en de manager zorgt ervoor dat de desbetreffende objecten aan de database worden toegevoegd.

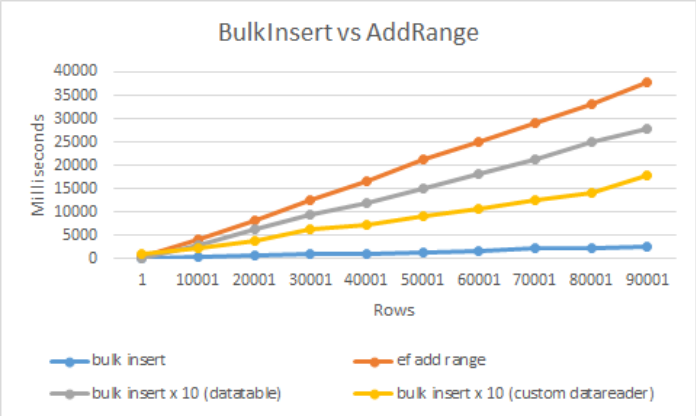
```
protected List<string[]> ProcessCsv(string path, string separtor = "\t", bool hasHeaderRow = false)
{
    List<string[]> values = new List<string[]>();

    using (TextFieldParser parser = new TextFieldParser(path))
    {
        parser.TextFieldType = FieldType.Delimited;
        parser.SetDelimiters(separtor); 1

        if (hasHeaderRow) // Skip first line if csv has a header row
        {
            parser.ReadLine();
        }

        while (!parser.EndOfData)
        {
            string[] fields = parser.ReadFields();
            values.Add(fields);
        }
    }
    return values;
}
```

1

2

Figuur 26: CSV bestanden inlezen

Tijdens het bijwerken van mij data liep ik tegen een bottleneck van het entity framework aan. Ik heb op het internet gezocht, ook daar kwam ik veel mensen tegen met hetzelfde probleem. Het entity framework is niet ideaal als het gaat om een groot aantal inserts of updates tegelijk. Tijdens het zoeken kwam ik een extensie tegen genaamd BulkInsert. BulkInsert voegt de gehele data set in een keer toe in plaats van één voor één zoals de standaard insert. In figuur 27 valt te zien wat de snelheid is van BulkInsert met vergelijking tot andere mogelijkeden.



Figuur 27: BulkInsert [https://efbulkinsert.codeplex.com/]

Zoals valt te zien in figuur 27 is de BulkInsert (blauwe lijn) vele malen sneller dan de standaard range insert (oranje lijn). Door gebruik te maken van de BulkInsert voor het toevoegen van de productdata was de functie al sneller geworden, alleen nog lang niet snel genoeg.

Er werden nog steeds één voor één update queries aangeroepen voor de product prijs en product voorraad. Om dit op te kunnen lossen heb ik mijn eerste ontwerp moeten aanpassen. De prijs en voorraad van een product wordt niet langer bewaard in een product zelf, maar in een losse tabel. Hierdoor hoef ik niet langer een product te updaten, maar er wordt voor elke prijs en voorraad wijziging een nieuwe record in de desbetreffende tabel toegevoegd. Het voordeel hiervan is dat ik ook voor de prijs en voorraad van een product gebruik kan gaan maken van de BulkInsert extensie. Nog een bijkomend voordeel is dat ik op deze manier per dag de prijs en voorraad geschiedenis kan opvragen. Dit kan ik vervolgens weer gebruiken in mijn prijsgeschiedenis en voorraadgeschiedenis feature.

Een nadeel is dat op deze manier de database wel erg snel gevuld wordt. Als we ervan uitgaan dat we 200k records in de database hebben staan, dan wordt elke dag 200k X 2 records toegevoegd. Dit betekend dat ik na een maand al 12 miljoen records in de database heb staan voor alleen de prijs en voorraad van een product. Dit gaat natuurlijk voor problemen zorgen, daarom neem ik dit mee in een performance onderzoek in iteratie 2. In iteratie 1 wil ik me vooral focussen op het resultaat en minder op de performance.

Ik voeg nu algemene product gegevens, prijs, voorraad en categorieën via de BulkInsert toe, het enige wat nog per record wordt bijgewerkt is het verwijderen van buitengebruik gestelde producten, alleen om hier veel aandacht aan te besteden vond ik zonde van mijn tijd. Dit zullen misschien enkele producten per week zijn.

Na alle aanpassingen was de functie al zeker 10 keer sneller geworden. Ik moest alleen nog een check inbouwen die controleerde of de toe te voegen record niet al bestond in de database. Hierbij kwam ik weer bij hetzelfde probleem uit, dat dus voor elk product een aanvraag naar de database gedaan zo moeten worden. Het wegschrijven van 1000 records naar de database op deze manier duurde 21 seconden. Dit is natuurlijk belachelijk lang, helemaal als je nagaat dat ik uiteindelijk nog zeker 200 keer zoveel producten erbij krijg. Hiervoor moest dus een oplossing worden gevonden. Dit heb ik opgelost door eerst een lijst met artikelnummers op te halen uit de database van de leverancier waarvan we producten aan het toevoegen zijn. Aangezien artikkelnummers voor een leverancier uniek moeten zijn kan ik doormiddel van deze lijst controleren of een product al in de database staat en hoef ik niet meer voor elke record de database aan te spreken. Door het doorvoeren van deze wijziging duurde het toevoegen en updaten van 1000 records nog maar 239 ms. Dat is een performance verbeting van ruim x100 ten opzichte van eerst.

Ditzelfde concept heb ik ook gebruikt bij het opzoeken van de bijbehorende categorie van een product. Ik haal eerst een lijst op met categoriegegevens, in plaats van voor elke record een query naar de database te doen. In figuur 28 ziet u hoe ik de data eerst ophaal, inplaats van per record een database aanvraag te doen. Hetzelfde concept heb ik op overig vervolgens op diverse plaatsen in de applicatie toegepast.

```
var productBulkList = new List<Product>();                    Ophalen artikelnummers
var articleNumbers = SelectArticleNumbersBySupplier(db, supplierId).ToArray();
var externalCategories = ExternalCategoryManager.Instance.SelectAll(db)
    .Select(a => new { id = a.Id, name = a.Name, niveau = a.Niveau, supplierId = a.SupplierId })
    .Where(a => a.niveau == 3)
    .ToArray();                    Externe categorieën met niveau 3 eenmalig ophalen
```

Figuur 28: UpdateProductGeneral, Eerst ophalen data

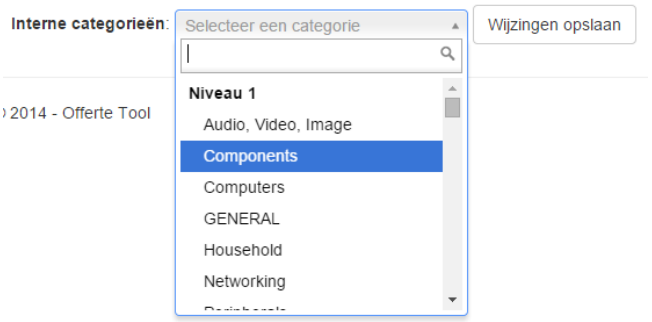
7.8.3 Gegevens opslaan in de database

Vervolgens worden de producten opgeslagen in de database met behulp van de bulkinsert. Er wordt een lijst aangemaakt voor alle producten die moeten worden toegevoegd. Zodra alle producten aan de lijst zijn toegevoegd, kan ik met slechts één regel alle producten toevoegen aan de database. Dit doe ik met het volgende stukje code: “databaseTabel.BulkInsert(List<Product>)”.

7.9 Categorieën module

De categorie module bestaat uit twee onderdelen, het eerste onderdeel bestaat uit interne categorieën. De interne categorieën worden weergegeven in een overzicht. Verder is het mogelijk om nieuwe interne categorieën aan te maken of bestaande categorieën te wijzigen of verwijderen. Van een interne categorie wordt ook weergegeven hoeveel producten hieraan zijn gekoppeld.

Het tweede onderdeel bestaat uit het koppelen van externe categorieën aan interne categorieën. Je selecteert eerst de interne categorie waar je externe categorieën aan wilt koppelen. Dit valt te zien in figuur 29.



Figuur 29: Keuze interne categorie

Zodra de interne categorie is geselecteerd wordt een nieuw overzicht opgehaald en ingeladen met jQuery load. Aan de linkerzijden vindt u een tabel met externe categorieën die zijn gekoppeld aan de gekozen interne categorie. Aan de rechterzijden vindt u een tabel met externe categorieën die nog niet aan een interne categorie zijn gekoppeld. In figuur 30 ziet u een voorbeeld voor de interne categorie “Audio, Video, Image”. Zoals valt te zien is er op dit moment één externe categorie aangekoppeld, namelijk “Audio, Video, Image”. Deze heeft toevallig dezelfde naam. Dit komt doordat ik de interne categorieën heb aangemaakt aan de hand van de categorieën van Techdata. Vervolgens heb ik deze 1 op 1 gekoppeld met behulp van een SQL query.

Figuur 30: Koppelen categorieën

Op dit moment is er één externe categorie gekoppeld aan de geselecteerde interne categorie. Deze koppeling kan je verwijderen door de checkbox uit te vinken en vervolgens kies je voor wijzigingen opslaan. De lijst van nog te koppelen externe categorieën is een stuk groter, ik heb voor dit voorbeeld alleen de eerste 3 weergegeven. Je kan een externe categorie koppelen aan een interne categorie door het vinkje aan te zetten en vervolgens voor wijzigingen opslaan te kiezen. Het is ook mogelijk om beide tabellen tegelijk te bewerken. Verder is voor het gemak van de gebruiker een zoekveld toegevoegd. Met dit zoekveld worden alleen de records in de tabel weergegeven die overeenkomen met de zoekterm. Het zoeken op velden in de tabel is gemaakt met javascript en jQuery. Er wordt met behulp van een keylistener gekeken of er aanslagen zijn, als dit het geval is worden alleen nog de rows in de tabel weergeven waarin de waarde overeenkomt met de zoekterm.

7.10 Koppeling AcesDirect

De tweede leverancier waar we een koppeling mee konden maken was AcesDirect. AcesDirect levert hun productdata aan ons via een URL. Zodra de URL wordt aangeroepen begin je gelijk met het ophalen van de productdata. Ze leveren de productdata in CSV en XML formaat. Ik heb beide formaten bekeken en aan de hand van een korte inspectie is gebleken dat de CSV file een stuk sneller wordt opgehaald. De XML feed die werkt eigenlijk helemaal niet, mijn browser loopt in ieder geval altijd vast als ik deze probeer op te halen. Ik heb er mede hierdoor voor gekozen om gekozen om gebruik te maken van de CSV file. Daarnaast kan ik nu veel bestaande code hergebruiken aangezien de koppeling van Techdata ook gebruik maakt van bestanden in CSV formaat.

De feed die AcesDirect in eerste instantie voor ons klaar heeft gezet bevat producten uit hun gehele assortiment. Dit zijn in totaal ongeveer 500.000 producten, die CSV file is dan ook erg groot qua formaat, ruim 150mb. De grootte van de CSV file zorgt ervoor dat het ophalen van de productdata wel enige tijd kan duren, meestal duurt het opvragen een minuut lokaal. Dit is geen ideale situatie, we hebben dan ook gevraagd of het mogelijk is om alleen de benodigde productdata op te vragen. AcesDirect had aangegeven dat dit tot de mogelijkheid behoorde, het CSV bestandje wordt hierdoor aanzienlijk kleiner. Dit zorgt er gelijk ook voor dat het ophalen van de CSV binnen enkele seconden gebeurd.

Het ophalen van de productdata van AcesDirect wordt gedaan met een WebClient object. In figuur 31 kunt u zien hoe ik dit heb uitgewerkt.

```
CreateFolderIfNotExists(supplierRootPath);

var url = Supplier.RequestUrl;
var destinationPath = supplierRootPath + "/" + GetCurrentDateString() + ".csv";

if(!File.Exists(destinationPath)){
    WebClient wc = new WebClient();
    wc.DownloadFile(url, destinationPath);
}
```

Figuur 31: Download productdata AcesDirect

De data wordt opgehaald in gelijk weggeschreven naar een lokale directory. Zodra lokaal het bestandje is weggeschreven kan deze worden uitgelezen. In tegenstelling tot Techdata levert AcesDirect al hun productdata in één CSV bestand. Dit is op zich geen probleem, hier had ik binnen mijn ontwerp al rekening mee gehouden. Wat wel voor een probleem zorgde was het feit dat AcesDirect product had gekoppeld in hun CSV file aan Categorie namen en niet aan categorie Ids. Hierdoor heb ik de uitwerking voor het toewijzen van een categorie aan een product iets moeten aanpassen. Ik heb een aanpassing gemaakt die de bijbehorende categorie nu altijd opzoekt aan de hand van een categorie naam. In het geval van Techdata moet de naam wel eerst worden opgehaald aan de hand van de meegegeven categorieId in de productenlijst.

De functies voor het aanmaken van de Categorieën heb ik ook enigszins moeten bijwerken, in het geval van AcesDirect zijn er namelijk maar 2 categorie niveaus. Het laagste niveau voor AcesDirect is ook gewoon 3, AcesDirect heeft alleen geen niveau 1 categorieën. Ook moest ik er rekening mee houden dat de categorieën van Techdata wel een zelf een id meegeven en AcesDirect niet. Dit id heb ik nog wel nodig voor Techdata, aangezien ik anders niet kan opvragen welke id bij welke naam hoort tijdens het toewijzen van een categorie aan een product.

De functies voor het bijwerken van de product prijs en voorraad konden zonden aanpassing weer worden gebruikt voor AcesDirect.

7.11 Prijs en voorraad geschiedenis

De prijs en voorraad geschiedenis zou ik eigenlijk pas in de tweede iteraties implementeren, maar door de keuzes in mijn technisch ontwerp was dit qua backend werkzaamheden gemakkelijk te implementeren. De geschiedenis hield ik namelijk voor zowel de prijs en voorraad al bij in mijn database. Het enige wat ik nog moest doen is een geschikte grafiek zoeken om de data in weer te kunnen geven. Zoals u al eerder had kunnen lezen maak voor de grafieken binnen de applicatie gebruik van Amcharts.

7.12 Unit tests

Ik had voorafgaand aan de iteratie mijzelf voorgesteld dat ik alle belangrijke functionaliteiten zou gaan testen met een unit test. Tijdens het schrijven van de unit test kwam ik er echter achter dat dit wat lastiger ging worden dan ik in eerste instantie had gedacht. Het grootste gedeelte van mijn applicatie is gebaseerd op het ophalen en aanpassen van data uit de database. Daarnaast werk ik ook nog eens met het entity framework. Dit alles maakt het lastig om bepaalde functionaliteiten te testen aangezien er meestal een database aanvraag wordt gedaan. Ik vind het zelf niet handig als mijn live database wordt bevuild met testdata. Ik denk zelf dat de beste oplossing hiervoor is een testdatabase opzetten, die qua data er ongeveer hetzelfde uitziet als de live database. Anders heeft het testen op performance bijvoorbeeld weinig zin. In iteratie 1 test ik alleen de functies die geen gebruik maken van de database. De functies die wel gebruik maken van de database neem ik mee in de tests van iteratie 2.

Een van de functionaliteiten die ik wel heb getest in iteratie 1 is testen of het ophalen van product gegevens van de servers van de leveranciers goed gaat. Voor de beiden gemaakte koppeling heb ik een test opgesteld die probeert om data van de server van de leverancier op te halen. Vervolgens wordt er gekeken of de bestanden ook echt worden weggeschreven naar onze lokale directory. In figuur 32 vindt u de unit test voor het controleren op opgehaalde product data voor de leverancier Techdata.

```
[TestMethod]
public void TestRequestDataTechdata_FilesCreated()
{
    var supplier = GetSupplier();
    var supplierImportPath = GetImportPath(supplier.Name);
    var filePath = supplierImportPath + "\\" + DateTime.Now.ToString("yyyyMMdd");

    // Controleer eerst of de files al niet zijn opgehaald, zoja verwijder deze.
    if (Directory.Exists(filePath))
    {
        Directory.Delete(filePath, true);
    }

    ProductImportTechdata productImportTechdata = new ProductImportTechdata(supplier, new OfferteToolContext
    productImportTechdata.RequestData(supplierImportPath);
    var result = Directory.Exists(filePath);

    Assert.IsTrue(result, "Fout bij het downloaden van de productdata van Techdata");
}
```

Figuur 32: Unit test voor het controleren van de verbinding met Techdata

Ik vraag eerst de bijbehorende leverancier op, daarna haal ik een string op met een referentie naar de import folder van de leverancier. Voordat ik een data aanvraag ga doen controleer ik of de data al niet is opgehaald, mocht de data al zijn opgehaald zou de test altijd slagen en dat is niet de bedoeling. Als de data al was opgehaald wordt deze eerst verwijderd. Vervolgens wordt de data opgehaald en gekeken of deze lokaal is weggeschreven. De test voor AcesDirect ziet er ongeveer hetzelfde uit, alleen wordt er gekeken of er een file is aangemaakt in plaats van een directory. Daarnaast wordt er natuurlijk ook een ander import object aangemaakt.

Ik heb ook nog tests gemaakt voor functies in de abstracte klasse. Om de abstracte klasse te kunnen testen heb ik binnen mijn testproject een nieuwe klasse aangemaakt die overerft van de abstracte ProductImport klasse. Deze test klasse heb ik aangemaakt omdat ik van een abstracte klasse geen object kan aanmaken. Wat ik wel had kunnen doen was een object aanmaken van één van de subklassen. Alleen ik vond dit zelf geen nette oplossing, het aanmaken van een testklasse die overerft

van de abstracte klasse en de functies volgens teruggeeft met base.functie() vond ik de beste oplossing. In figuur 33 kunt u zien wat ik bedoel met het teruggeven van de base.functie().

```
public List<string[]> ProcessCsv(string path, string separtor = "\t", bool hasHeaderRow = false)
{
    return base.ProcessCsv(path, separtor, hasHeaderRow);
}
```

Figuur 33: Return base functie van abstracte klas voorbeeld

Ik heb testen geschreven voor het uitlezen van CSV bestandje. Er wordt onder andere gecontroleerd wat er gebeurt wanneer het ingelezen bestand niet van een CSV formaat is. Ook worden verschillende scheidingstekens getest en wordt er getest met en zonder een header regel.

Daarnaast heb ik nog enkele andere kleinere functies getest die me helpen bij het ophalen en toevoegen van data aan de database na het importeren van productgegevens van een leverancier.

8 Uitvoering: Construction fase 2 (uitbreidingen)

Construction fase twee zal bestaat uit uitbreidingen op de basis. Tijdens deze fase komen vooral de Should en de Could have's requirements aan bod. De uitbreidingen bestaan uit de volgende onderdelen:

- Kleine aanpassingen aan de hand van feedback opdrachtgever
- Koppeling met Overige leveranciers
- Order en ShoppingCart module
- User module
- Externe categorieën kiezen voor de producten import
- Promo prijzen module
- Performance verbeteren
- Unit test scripts

8.1 Demo

Na een demo van de huidige applicatie heeft de opdrachtgever mij feedback gegeven op de huidige functionaliteiten. Hieruit zijn een aantal kleine wijzigingen gekomen, maar ook een paar nieuwe wensen die wat meer tijd gaan kosten.

Kleine punten die naar voren gekomen zijn:

- Extra kolommen in het producten overzicht.
- Filter / zoek module iets uitbreiden.
- Aangeven of een prijs is gestegen of gedaald.
- Aangeven of de prijs een promotie prijs of normale prijs is.
- Gemiddelde in de prijs en voorraad geschiedenis.
- Verwijderde producten wel weergeven in het overzicht, maar met een rode kleur.
- Product omschrijving inkorten als deze langer is als 2 regels in de tabel.

Deze kleinere punten kosten naar schatting bij elkaar ongeveer een dag om door te voeren. Dit kon ik dus gemakkelijk bij mij huidige planning erbij zetten.

Grotere punten die naar voren gekomen zijn:

- Het moet mogelijk zijn om meerdere winkelmandjes tegelijk aan te maken.
- De opdrachtgever wil zelf aan kunnen geven van welke externe categorieën producten worden geïmporteerd.

Ik verwachte ongeveer een week tijd nodig te hebben om de bovenstaande nieuwe functionaliteiten door te kunnen voeren. Normaal had ik hier geen tijd voor gehad, maar doordat de koppeling met Xeptor niet doorging is er extra tijd vrij gekomen. Xeptor wilde alleen product informatie beschikbaar stellen als Interpulse maandelijks een minimaal bedrag aan orders zou afnemen. Dit zag de opdrachtgever niet zitten en daarom hebben we besloten de koppelingen voor nu links laten liggen. Daarnaast gaf de opdrachtgever ook aan dat we de koppelingen met de secundaire leveranciers (Central Point en Travion) voor dit project nog links zouden laten liggen. De bestellingen bij deze leveranciers zijn namelijk minimaal en hij zag liever de nieuwe wensen en eisen terugkomen in de applicatie. Hierdoor bleef er tijd over om de grotere punten door te voeren in de planning.

8.2 Bijwerken requirements en diagrammen

Ik ben iteratie 2 begonnen met het bijwerken van alle documentatie. Zo zijn er weer nieuwe requirements naar voren gekomen. Ik heb mijn klassendiagram bijgewerkt en ook heb ik nieuwe sequentiediagrammen opgesteld.

8.3 Order en ShoppingCart module

De eerste functionaliteit die ik in iteratie 2 ben gaan bouwen was die van de winkelwagen en order module. De gebruiker maakt eerst een winkelwagen aan, vervolgens worden hier producten aan toegevoegd. Zodra de gebruiker klaar is met winkelen kiest hij ervoor om de winkelwagen om te zetten naar order. De producten uit de actieve winkelwagen worden omgezet naar een order en de winkelwagen wordt vervolgens weer leeg gemaakt. Het is niet meer mogelijk om een order te wijzigen, deze is definitief.

In eerste instantie had ik er voor gekozen om een winkelwagen op te slaan in het huidige sessie object. Zodra de sessie is verlopen worden de aangemaakte winkelwagens dus ook weer verwijderd. Hiervoor maakte ik gebruik van het standaard aanwezige Session object. Ik roep echter niet door mijn hele applicaties de sessie direct aan, maar doet die via een ShoppingCart object in de BaseController. Deze constructie heb ik in figuur 34 weergegeven.

```
public ShoppingCart SelectedShoppingCart
{
    get
    {
        if (this.Session["SelectedShoppingCart"] == null)
        {
            this.Session["SelectedShoppingCart"] = new ShoppingCart();
        }

        return (ShoppingCart)this.Session["SelectedShoppingCart"];
    }
    set
    {
        this.Session["SelectedShoppingCart"] = value;
    }
}
```

Figuur 34: Ophalen huidige winkelwagen

Alleen na een feedback moment met de opdrachtgever kwam ik tot de conclusie dat dit niet de juiste oplossing was. De opdrachtgever wilde namelijk een winkelwagen de volgende dag ook nog kunnen bijwerken. Hierdoor moest mijn ontwerp weer enigszins worden aangepast.

Om ervoor te zorgen dat een winkelwagen ook wordt bewaard voor later gebruik heb ik wijzigingen in mijn ontwerp en code moet maken. De klasse shoppingcart heb ik verwijderd en aan de klasse order heb ik een nieuwe attribuut toegevoegd die bepaald of het order of een winkelwagen is. De klasse order is een winkelwagen wanneer de nieuwe attribuut false en een order wanneer de nieuwe attribuut true is. Producten worden nu direct toegevoegd aan een order en dus ook opgeslagen in de database voor later gebruik.

Zoals al eerder was vermeld wilde de opdrachtgever met meerdere winkelwagens tegelijk kunnen werken. Ik heb het zo gemaakt dat alle aangemaakte winkelwagentjes in een selectlist terecht komen. Zodra uit deze selectlist een winkelwagen wordt geselecteerd haal ik met jQuery een nieuwe view op waarin de geselecteerde winkelwagen wordt getoond. In figuur 35 kunt u zien hoe ik dit heb gedaan. Ik zal per nummer even kort uitleggen waar het voor is en wat ik ermee doe.

1. Nummer 1 is de target element waarin de opgehaalde view in wordt weergegeven.
2. De methode die wordt uitgevoerd
3. Welk element uit de op te halen view moet er worden ingeladen

4. De data die ik meestuur met de methode
5. Een zelf geschreven functie die zorgt dat je een product aantal kan bijwerken. Ook deze functie werkt met jQuery. In dit geval maak ik gebruik van een ajax request om de actieve winkelwagen bij te werken.

```
function LoadShoppingCartProducts(name)
{
    $("#cartGrid").load("/ShoppingCart/Detail .row", { name: name }, function () {
        BindOnCountItemsChange();
    });
}
```

Figuur 35: Ophalen winkelwagen

Tijdens het omzetten van de winkelwagen naar een order liep ik tegen een probleem aan. Het was namelijk niet mogelijk om één product meer dan 1 keer aan een orderpart toe te voegen. Dit kwam doordat de tussentabel Orderpart_Product automatisch was gegeneerd door het entityframework en de combinatie van orderpartId en productId moest uniek zijn. Dit heb ik opgelost door een model object van deze tussentabel aan te maken met een extra attribuut aantal. Wanneer een product vaker voorkomt in een orderpart verhoog ik gewoon het aantal.

Van een order wordt per orderpart weergegeven welke producten daarin voorkomen. De opdrachtgever kan zo zien welke producten er bij welke leverancier moeten worden besteld. Er is uiteraard ook per leverancier een totaal prijs. Het is ook mogelijk om een order te downloaden als pdf bestand. Voor het omzetten van een html pagina naar een pdf document heb ik gebruik gemaakt van de externe libary Rotativa. Rotativa is een gratis .NET libary voor het omzetten van html views naar pdf documenten.

8.4 Externe categorieën kiezen voor producten import

In de huidige situatie werden alle producten die aangeleverd worden door de leverancier ingelezen in opgeslagen in onze database. Hierdoor krijgen we ook producten in ons systeem waarvan we eigenlijk nooit gebruik van zullen maken. De opdrachtgever had tijdens een feedback moment aangegeven dat hij graag zelf wilde bepalen van welke externe categorieën er producten worden geïmporteerd.

8.4.1 Overzicht externe categorieën

Ik ben begonnen met het maken van een nieuwe overzicht. In dit overzicht worden alle externe categorieën per leverancier weergegeven. Hier kan de gebruiker per externe categorie aangeven of deze moet worden genomen in de import. Dit kan voor alle niveaus, als de gebruiker kiest om een externe categorie van niveau 1 mee te nemen worden automatisch alle sub externe categorieën ook meegenomen in de import. Hetzelfde geld bij een niveau 2 externe categorie. Bij een externe categorie van niveau 3 wordt alleen de gekozen externe categorie meegenomen. Het selecteren van deze externe categorieën gaat door middel van een html checkbox. Ik heb een “on click” event listener gezet op deze checkbox. Zodra deze wordt geactiveerd word er een ajax request uitgevoerd. De ajax request wordt in figuur 36 weergegeven. Daarnaast kunt u hier gelijk zien hoe ik gebruik maak van de sweet alert libary. In dit geval ging het enkel om een normale alert zonder confirm knoppen.

```
function SaveExternalCategory(data, checked)
{
    $.ajax({
        url: "/ExternalCategory/Save/",
        type: "POST",
        data: data
    }).done(function () {
        swal({
            title: "Externe categorie opgeslagen.",
            text: checked
                ? "Producten uit deze categorie worden nu meegenomen in de import."
                : "Producten uit deze categorie worden niet meer meegenomen in de import.",
            type: "success",
            timer: 5000,
            allowOutsideClick: true
        });
    });
}
```

Aan de hand van de short if notatie de tekst voor de alert zetten.

Sweet alert success box

Figuur 36: Externe categorie meenemen in import

Nu we weten welke externe categorieën de gebruiker mee wilt nemen in de import moeten we hier ook nog rekening mee gaan houden. Dit uitwerking hiervan verschilt per leverancier, aangezien leveranciers data verschillend aanleveren.

8.4.2 Techdata import bijwerken

Techdata levert al haar data aan in losse bestandjes, dat is voor dit onderdeel niet heel erg handig. Ik heb een nieuwe functie toegevoegd die de producten lijst filter aan de hand van de gekozen externe categorieën. Het filteren van de lijst doe ik met behulp van een LINQ query. Hierdoor worden alleen de producten geïmporteerd waarvan de bijbehorende externe categorie mag worden geïmporteerd. Voor de prijs en voorraad heb ik geen wijziging door hoeven voeren. Ik had namelijk al een check staan die controleerde of de bijbehorende product bestond in de database. Als dit niet het geval was werd de prijs en voorraad ok niet toegevoegd.

8.4.3 AcesDirect import bijwerken

AcesDirect levert al haar data aan in één CSV bestand. Dit was in dit geval erg handig, hierdoor kan ik namelijk in 1x de producten, prijzen en voorraad filteren aan de hand van de gekozen externe categorieën. Ook hier filter ik de lijst door middel van een LINQ query.

8.5 Koppeling met Copaco

De koppeling met Copaco heb ik nog niet kunnen realiseren. Dit kwam doordat Copaco zelf moeilijk deed over het vrijgeven van hun productdata. Copaco wilt namelijk niet dat wij deze data publiek beschikbaar stellen. De opdrachtgever heeft meerdere malen contact met Copaco gehad hierover, alleen we hebben tot op heden geen data van ze ontvangen.

Ik had al wel voorbereidingen getroffen voor de import van Copaco. In de toekomst zou dit dus binnen een dag gekoppeld kunnen worden. De tijd die vrij kwam door het niet aangeleverd krijgen van de productdata heb ik vooral gebruikt bij mijn performance onderzoek.

8.6 Promo prijzen module

De promotieprijzen module bestaat uit twee onderdelen. Het eerste onderdeel hoort bij het producten overzicht. Er is een extra filter toegevoegd die alleen producten weergeeft waarvan een promotieprijs bekend is. Daarnaast worden de prijzen van producten waarvan een promo bekend is ook vetgedrukt en in het groen weergegeven.

Het tweede onderdeel bevindt zich in het begin scherm. Ik heb een extra kader toegevoegd die nieuwe promotie prijzen van de huidige dag weergeeft. De opdrachtgever kan hierdoor in één oogopslag zien welke producten er die dag in de aanbieding zijn.

8.7 Performance

Zoals ik al eerder had aangegeven zou ik nog een “performance onderzoekje” houden. Dit onderzoek richt zich op de performance binnen het producten overzicht. Ik merkte dat het zoeken naarmate het aantal producten groeide steeds trager werd. Het ophalen van alle data gaat gewoon snel, alleen zodra er ook gefilterd moest worden werd het overzicht aanzienlijk trager. Zoeken op een tekst met meerdere worden kon soms wel 20 seconden duren en dat was naar mijn mening onacceptabel.

Na he bestuderen van de queries kwam ik er al gauw achter er veel performance verloren ging op SQL niveau. Het uitvoeren van een zoekactie met een “%tekst%” wildcard kost SQL enkele seconden. Dit komt doordat SQL 4 kolommen keer het aantal producten in de database moet doorzoeken.

8.7.1 Blitzindex

Een collega vertelde mij over blitz_index, je kan met behulp van een blitz_index script zien waar binnen je database problemen zijn met bijvoorbeeld ongebruikte indexen of juist ontbrekende indexen. Ik heb deze scripts op mijn database gedraaid en geanalyseerd, hieruit kwamen eigenlijk maar 2 problemen naar voren. Er zat een index op productnaam die nooit werd gebruikt en artikelnummer was van het type string, terwijl dat altijd een integer waarde bevatten. De index heb ik verwijderd van de productnaam en artikelnummer heb ik omgezet naar een integer waarde. Verder heb ik geen problemen kunnen ondervinden en de performance boost door deze wijzigingen was dan ook niet echt merkbaar.

8.7.2 FULLTEXT SQL

Ik weet nu dat het probleem hem niet zozeer zit in het wel of niet hebben van indexen, maar het probleem zit ergens anders. Doordat ik gebruik maak van een full wildcard search kan SQL geen gebruik maken van de indexen. Dit moet ik dus op een andere manier oplossen.

Ik heb verder onderzoek gedaan wat er nog meer voor mogelijkheden zijn en zo kwam ik FULLTEXT SQL tegen. Dit zou vele male sneller moeten zijn dan de standaard LIKE query en het is al beschikbaar vanaf SQL server 2005. Om dit te gebruiken moet je zogenaamde FULLTEXT indexen aanmaken voor de kolommen waarop je wil gaan zoeken. Hierbij liep ik al gauw tegen een probleem aan, een FULLTEXT index moet altijd uniek en mag nooit NULL zijn. Het uniek maken van de kolommen is voor mij niet mogelijk. Een product kan namelijk meerdere keren bestaan met dezelfde naam of frabrikantcode, alleen dan van een andere leverancier. Ik kan dus helaas geen gebruik maken van FULLTEXT SQL.

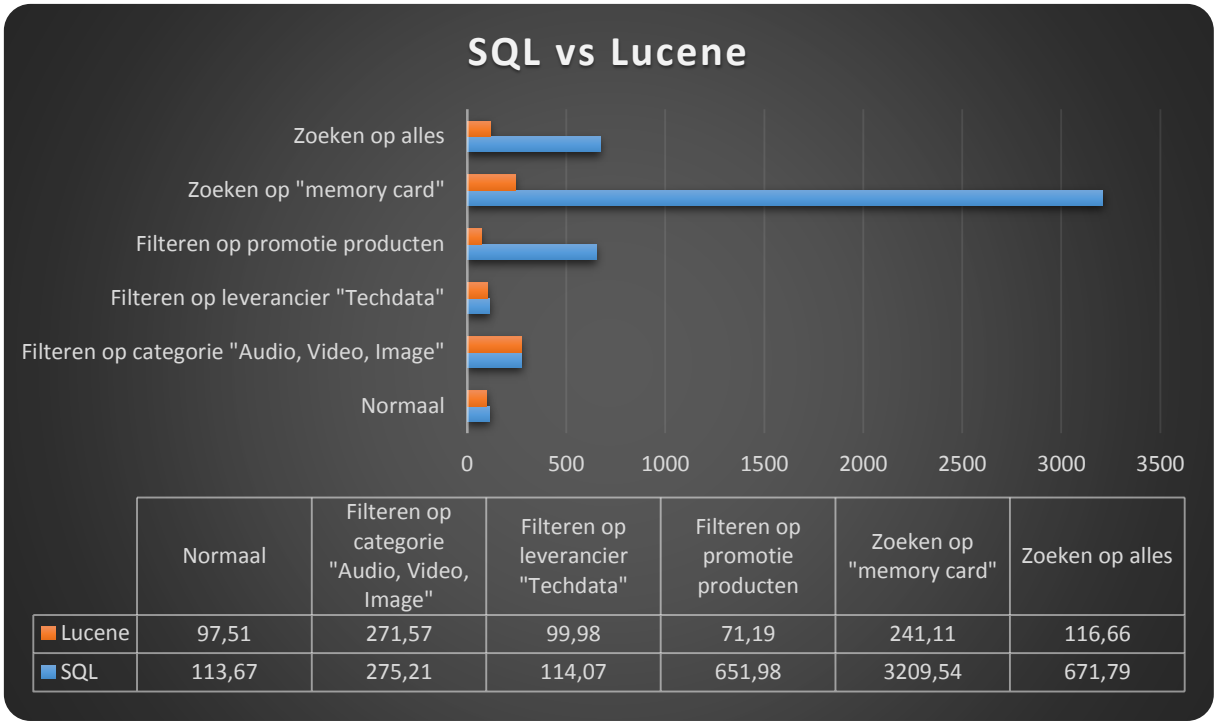
8.7.3 Lucene search

Tijdens mijn zoektocht naar een oplossing ben ik meerdere keren de term Lucene tegengekomen. Lucene is opensource zoekengine die geschreven is in Java. Lucene werkt volgens tekstbestandjes met indexen en claimt vele malen sneller te zijn dan de standaard SQL search. Daarnaast is Lucene ook beschikbaar voor het .NET framework. Na het doorlezen van verschillende artikelen over Lucene heb ik genoeg informatie gevonden om het zelf toe te kunnen passen binnen de applicatie.

Om gebruik te maken van Lucene heb ik eerst een aantal kleine wijzigingen moeten doorvoeren, Lucene kan namelijk geen NULL waardes in een tekst document opslaan. Verder heb ik special voor de Lucene search een nieuw model object aangemaakt, namelijk ProductLucene. Een ProductLucene object wordt niet opgeslagen in de database en bevat informatie die nodig is voor het zoeken en tonen van het productenoverzicht. Een ProductLucene object wordt aangemaakt aan de hand van

een Product object, het grootste verschil is echter dat een ProductLucene object alleen de laatste prijs een voorraad bevat. Ik kan namelijk geen relaties opslaan in de Lucene tekstbestanden. Het is namelijk geen relationele database.

Ik heb een vergelijking gemaakt qua performance met het filteren op producten op SQL niveau en op Lucene niveau. Wat nog wel belangrijk is om te vermelden is dat de SQL filter was gebaseerd op 46.000 producten en de Lucene filter op 106.000 producten. De resultaten van deze vergelijking wordt weergegeven in figuur 37. Ik heb deze vergelijking gemaakt in mijn controller met behulp van de StopWatch klasse. De resultaten zijn inclusief het aanmaken van enkele viewmodel objecten die worden getoond in het producten overzicht. Zoals valt te zien in figuur 37 is alles eigenlijk in het voordeel van de Lucene search. Vooral bij het zoeken op een stuk tekst valt het enorme verschil in performance op, de Lucene search is op dit vlak zeker 10 keer zo snel.



	Normaal	Filteren op categorie "Audio, Video, Image"	Filteren op leverancier "Techdata"	Filteren op promotie producten	Zoeken op "memory card"	Zoeken op alles
Lucene	97,51	271,57	99,98	71,19	241,11	116,66
SQL	113,67	275,21	114,07	651,98	3209,54	671,79

Figuur 37: SQL vs Lucene search

Waar het zoeken met Lucene veel tijd wint verloor ik weer tijd bij het aanmaken van de indexen. Ik moet namelijk bij elke producten import de indexen opnieuw aanmaken. De prijs of voorraad kan namelijk gewijzigd zijn, daarnaast kunnen er ook nieuwe producten zijn bijgekomen. Het aanmaken van de indexen zelf is echter niet het probleem, maar het aanmaken van de ProductLucene objecten. Ik maak namelijk voor elk product object aanwezig in de database een productLucene object aan. Het ophalen van alle relaties van een product kost gewoon wat tijd. Ik heb de query wil iets sneller kunnen maken door lazy loading te overriden met de include query, maar alsnog kost het aanmaken van al deze objecten enkele minuten. In figuur 38 kunt u zien wat ik hiermee bedoel. Door gebruik te maken van de include query wordt de gehele relatie onmiddellijk opgehaald in plaats van alleen de benodigde onderdelen. Dit scheelt uiteindelijk qua performance doordat het entity framework alles in één query ophaalt in plaats van telkens delen in kleinere queries.

```
var luceneProducts = ProductManager.Instance.SelectAll(db)
    .OrderBy(a => a.Id)
    .Include(a => a.ProductPrices)
    .Include(a => a.ProductStock)
    .ToList();
```

Figuur 38: Include query

8.8 User module

De gebruiker module is nog redelijk beperkt. Het is mogelijk gebruikers aan te maken en ermee in te loggen. Wachtwoord vergeten en wachtwoord wijzigen zit er ook in. Verder heb ik alvast een opgezet gemaakt voor de relatie user – order. Wanneer er een gebruiker is ingelogd wordt deze ook gelijk gekoppeld zodra er een order wordt aangemaakt. Maar voor nu blijft het hier ook bij, de opdrachtgever vond user module op dit moment nog niet belangrijk, voorlopig wordt de applicatie toch maar door één gebruiker gebruikt. De user module is leuk voor in de toekomst, maar valt op dit moment nog buiten de scope van de opdracht.

8.9 Unit test scripts

Voor de applicatie heb ik diverse unit test scripts geschreven. De keuze welke functionaliteiten ik zou gaan testen heb ik in eerste instantie gemaakt aan de hand van mijn opgestelde requirements. Ik ben alle must en should have requirements doorgelopen en heb gekeken welke requirements zinnig zijn om te testen. Daarnaast heb ik zelf nog gekeken wat vind ik daarnaast nog belangrijk om te testen. Hieruit zijn uiteindelijk tussen de 40 en 50 testscripts uitgekomen.

Het testen op zich was best een uitdaging, aangezien het grootste deel van de applicatie afhankelijk is van de data uit de database. Hier heb ik wel een oplossing voor kunnen vinden namelijk Effort in memory database. De naam zegt het eigenlijk al, met behulp van Effort kan je een tijdelijk een database aanmaken in je geheugen voor de unit test scripts. Helaas werkt Effort niet in combinatie met de BulkInsert module, het bulk inserten heb ik dus ook niet kunnen testen. Wel heb ik de logica die daaraan vooraf gaat getest.

Het aanmaken van de tijdelijke database in het geheugen is redelijk simpel. Wat heb ik gedaan is een 2de constructor aangemaakt voor me context klassen, deze 2de constructor heeft als parameter een connectie string. Vervolgens maak ik een connectie aan via effort, dit valt te zien in de eerste regel van de functie CreateTestDatabase in figuur 39. Zodra ik deze connectie meegeef aan mijn context constructor is de context klassen gelinkt aan de in memory database in plaats van de fysieke database zoals in de werkelijke applicatie. Hierdoor kan ik binnen mijn unit test scripts ook functies testen die gebruik maken van de database.

```
[AssemblyInitialize]
public static void CreateTestDatabase(TestContext context)
{
    DbConnection connection = Effort.DbConnectionFactory.CreateTransient();
    Db = new OfferteToolContext(connection);

    CreateTestSuppliers();
    CreateTestCategories();
    CreateTestExternalCategories();
}
```

Figuur 39: In memory connectie aanmaken

Zoals u kunt zien maak ik gebruik van een keyword “[AssemblyInitialize]”, deze functie wordt één keer uitgevoerd voordat alle test scripts worden gedraaid. Ik maak hiervan gebruik, omdat ik voor sommige functionaliteiten testdata nodig heb in mij database. Hierdoor hoef ik het maar éénmalig aan te maken in niet telkens handmatig in de test scripts zelf. Naast de “[AssemblyInitialize]” maak ik ook nog gebruik van het keyword “[ClassInitialize]”. ClassInitialize wordt altijd uitgevoerd voor alle test functies van de test klasse. Dit is erg handig als je specifiek voor één test klassen bepaalde objecten wilt aanmaken.

Zoals ik al eerder had genoemd heb ik een waslijst een test scripts geschreven, deze ga ik natuurlijk niet allemaal laten zien. Ik ga 1 test script laten zien die gebruik maakt van de in memory database die ik in de bovenstaande alinea had besproken.

In figuur 40 kunt u mijn testscript voor het linken van externe categorieën aan een interne categorie zien. Om de testscript begrijpbaar te houden heb ik deze opgedeeld in 5 stukken.
Stap 1: het aanmaken van de interne categorie en deze toevoegen aan de test database.
Stap 2: het aanmaken van een aantal externe categorieën en deze toevoegen aan de test database.
Stap 3: het ophalen van de externe categorie Ids zodat ik deze kan gaan linken aan de interne categorie. Daarnaast bepaal ik gelijk het verwachte resultaat met de variabele expectedResult.
Stap 4: het uitvoeren van de functie de getest moet worden. Nadat dit gedaan is slaan we het resultaat op in de variabele result;
Stap 5: controleren over het verwachte resultaat overeenkomt met de werkelijke uitkomst.

```
[TestMethod]
public void TestLinkCategories()
{
    var category = new Category() { Id = 10, Name = "Test", Niveau = 1 };
1   Db.Category.Add(category);
    Db.SaveChanges();

    var extCategories = new List<ExternalCategory>()
    {
        new ExternalCategory(){ Id = 10, SupplierId = 1, Name = "ExternalCategory10", Niveau = 1, ParentE:
        new ExternalCategory(){ Id = 11, SupplierId = 1, Name = "ExternalCategory11", Niveau = 2, ParentE:
        new ExternalCategory(){ Id = 12, SupplierId = 1, Name = "ExternalCategory12", Niveau = 3, ParentE:
2       new ExternalCategory(){ Id = 13, SupplierId = 2, Name = "ExternalCategory13", Niveau = 1, ParentE:
        new ExternalCategory(){ Id = 14, SupplierId = 2, Name = "ExternalCategory14", Niveau = 2, ParentE:
        new ExternalCategory(){ Id = 15, SupplierId = 2, Name = "ExternalCategory15", Niveau = 3, ParentE:
    };
    Db.ExternalCategory.AddRange(extCategories);
    Db.SaveChanges();

3   var externalCategorieIds = extCategories.Select(a => a.Id.ToString()).ToList();
    var expectedResult = externalCategorieIds.Count;

4   ExternalCategoryManager.Instance.LinkCategories(Db, category.Id, externalCategorieIds);
    var result = Db.Category.First(a => a.Name == "Test").ExternalCategories.Count;

5   Assert.AreEqual(expectedResult, result);
}
```

Figuur 40: TestLinkCategories

De overige test scripts zijn redelijk vergelijkbaar opgebouwd, ik maak altijd eerst testdata aan in de database. Vervolgens voor ik de functie uit de moet worden getest en als laatste controleer ik of het resultaat overeenkomt met de verwachting.

8.10 Smoke testen

Naast de unit test heb ik ook nog verder gekeken naar testmogelijkheden voor de applicatie. Zo kwam ik smoke tests tegen. Met een smoke test klik je als het ware door de applicatie heen en controleer je of er niks breekt. Wanneer iets breekt komt er als het ware rook uit.

Voor het schrijven van de smoke test maak ik gebruik van het Robot Framework. Het Robot Framework is een open source testautomatisering framework voor acceptatie testen en acceptatie test-driven development (ATDD). Het Robot Framework is verder uit te breiden met diverse Java of Python libaries.

Ik gebruik de externe python libary Selenium2Libary in combinatie met het Robot Framework. De Selenium2Libary voert de testen uit in een web browser. De meeste moderne webbrowsers worden ondersteund, ik maak voor mijn testen gebruik van de Firefox webbrowser.

Voor de smoke testen heb ik een nieuwe folder binnen mijn project map aangemaakt. Vervolgens heb ik voor elke test module ook een aparte folder aangemaakt. Een test module is in principe een verzameling van testen voor één model object. Elke module bestaat uit een resource tekst bestand en een tekst bestand waarin de testen worden uitgevoerd. In het resource tekst bestand staan keywords specifiek voor de module. Ik heb ook nog een globale resource waarin keywords staan die door meerdere test modules worden aangeroepen.

De testen zijn allemaal geschreven in standaard tekst bestandjes. Binnen het tekst bestand wordt als scheidingsteken gebruik gemaakt van een tab. De tekst bestanden bestaan vervolgens weer uit variabelen en keywords die kunnen worden uitgevoerd. Aan de hand van deze keywords wordt er binnen de webbrowser acties uitgevoerd. Wanneer één zo een keyword niet met succes wordt afgerond komt er “rook” uit test en is deze gefaald. De beschikbare keywords zijn te vinden in de documentatie van Selenium2Libary.

Tot op heden heb ik 8 smoke tests geschreven voor het testen van de applicatie. Onderstaand zal ik één smoke test doorlopen en toelichten. De smoke test die ik ga toelichten is de smoketest voor het toevoegen van een product aan de winkelwagen.

In figuur 41 kunt u zien hoe de test is opgebouwd. De eerste regel beschrijft de naam van de test. De tweede regel met [Tags] beschrijft de tags die zijn aangemaakt voor het aanroepen van deze test. Vanaf regel 3 t/m regel 9 staan keywords die worden uitgevoerd voor deze test, deze zal ik zo verder toelichten. De laatste regel met de [Teardown] bevat een keyword die wordt uitgevoerd aan het eind van de test. In dit geval wilde ik dat de browser weer werd afgesloten zodra de test klaar is.

```
Add_product_to_shoppingcart
    [Tags]    Shoppingcart    Add_product_to_shoppingcart
    Open Browser To Start Page
    Open Shoppingcart Page
    Create Shoppingcart
    Open Product Page
    Add Product To Shoppingcart
    Open Shoppingcart Page
    Delete Shoppingcart
    [Teardown]    Close Browser
```

Figuur 41: Smoke test - add product to shoppingcart

De commando's die worden aangeroepen in de test zijn gedefinieerd in de resource bestanden of zijn standaard beschikbaar door Selenium2Libary. Figuur 42 bevat het resource bestand specifiek voor shoppingcart en figuur 43 bevat de base resource die wordt gebruikt door diverse test modulen. Ik laat niet alle keywords zien, anders werden de plaatjes te groot.

```
*** Settings ***
Library           Selenium2Library

*** Keywords ***
Add Product To Shoppingcart
    Click Element    //tr[1]/td[9]/form/a/span
    Page Should Contain    Product toegevoegd
```

Figuur 42: Smoke test - shoppingcart resource

```
*** Settings ***
Library           Selenium2Library

*** Variables ***
${SERVER}         offertetool.localtest.me
${START URL}      http://${SERVER}/
${BROWSER}        Firefox
${DELAY}          0.5

*** Keywords ***
Open Browser To Start Page
    Open Browser    ${START URL}    ${BROWSER}
    Maximize Browser Window
    Set Selenium Speed    ${DELAY}

Open Shoppingcart Page
    Click Link    id=popOverProductAdded
```

Figuur 43: Smoke test - Base resource

Als we kijken naar het eerste keyword bij figuur 43 "Open Browser To Start Page", zien we dat binnen dit keyword weer andere keywords worden aangeroepen. Eerst wordt de browser opgestart, vervolgens wordt de browser gemaximaliseerd en als laatste wordt de delay per commando aangegeven. In dit geval 0,5 seconden.

In figuur 42 staat het keyword voor het toevoegen van een product aan een winkelwagen. Dit wordt gedaan met het keyword Click Element, wat standaard door Selenium2Libary wordt meegeleverd. Een Click Element heeft als parameter een html element waarop geklikt moet worden. In dit geval is de eerste rij uit de tabel en vervolgens de 9^{e} kolom. Vervolgens controleer ik met het keyword "Page Should Contain" of er ook echt een product is toegevoegd aan de winkelwagen. Er moet namelijk een pop up zijn getoond met de tekst "Product toegevoegd".

Zoals u al had kunnen zien bevatten de test in totaal7 keywords die achter elkaar worden uitgevoerd. Onderstaand zal ik per keyword globaal beschrijven wat er gebeurt.

1. Open Browser To Start Page; Start Firefox op en navigeert naar de home pagina.
2. Open Shoppingcart Page; Navigeert naar het winkelwagen overzicht.
3. Create Shoppingcart; Maakt een test winkelwagen aan.
4. Open Product Page; Navigeert naar het producten overzicht.
5. Add Product To Shoppingcart; Voegt een product toe aan de test winkelwagen.
6. Open Shoppingcart Page; Navigeert naar het winkelwagen overzicht.
7. Delete Shoppingcart; Verwijderd de test winkelwagen.

De overige testen zijn op een zelfde manier opgebouwd. Aan het eind van de test ruim ik altijd de aangemaakte test data op zoals hierboven bij stap 7.

9 Uitvoering: Transition fase

De transition fase had ik eigenlijk al deels tijdens de laatste fase van de construction fase iteratie 1 uitgevoerd. Ik heb namelijk na de demo de opdrachtgever ook toegang geven tot de applicatie. Vervolgens heeft de gebruiker regelmatig gebruik gemaakt van de applicatie en zo dus ook een soort van gebruikerstesten uitgevoerd.

De database stond al vanaf begin af op een lokale Interpulse server. Deze staat dus voorlopig nog goed, aangezien de applicatie in eerste instantie alleen voor eigen gebruik is. De website is op dit alleen nog bereikbaar via mijn werkcomputer, deze zal dus nog wel op een Interpulse server moeten worden geplaatst. Maar dit wordt gedaan buiten de afstudeerperiode om.

10 Evaluatie

In dit hoofdstuk vindt u de evaluatie van de afstudeerperiode. Ik begin met de productevaluatie, in de productevaluatie beoordeel ik de tussentijdse opleveringen en het eindproduct. Vervolgens ga ik verder met de procesevaluatie, in de procesevaluatie geef ik mijn oordeel over de totstandkoming van de producten, gehanteerde methoden en technieken en hoe ik ben omgegaan met bepaalde problemen en keuzes. Ik sluit af met een evaluatie van de beroepstaken, ik geef per uitgevoerde beroepstaak aan waarom ik vind dat ik de beroepstaak op een bepaald niveau heb uitgevoerd.

10.1 Productevaluatie

Het doel van het afstudeerproject is naar mijn mening behaald. De opdrachtgever heeft een product opgeleverd gekregen wat voldoet aan zijn verwachtingen. Daarnaast heeft de applicatie de geschreven testen doorstaan. De opdrachtgever had de applicatie al enkele keren gebruikt en gaf aan dat hij er erg blij mee was, vooral het zoeken ging lekken snel. Het enige minpunt is dat de koppeling met Copaco nog niet afgerond is, maar zoals ik al eerder had aangegeven lag dat niet aan onze kant.

10.1.1 Onderzoeksrapport

Het onderzoeksrapport heeft mij geholpen om een goede keuze met de te gebruiken ontwikkel methode voor het afstudeerproject. Zonder dit vooronderzoek zou mijn keuze gevallen zijn op Scrum.

10.1.2 Plan van aanpak

Het plan van aanpak heeft niet alleen mij, maar ook mijn begeleider en opdrachtgever inzicht geven in de aanpak en planning van het project. Doordat het van te voren duidelijk was wanneer een iteratie zou worden afgerond kon ik afspraken met de opdrachtgever maken over een feedback moment. Daarnaast kon mijn begeleider aan de hand van de opgestelde planning zien of dit ook haalbaar was.

10.1.3 Inception rapport

Ik ben tevreden over het inception rapport, door de opgestelde requirements kon ik duidelijk zien wat de opdrachtgever wel en niet belangrijk vond. Daarnaast heeft het maken van de eerste versie van het klassendiagram me geholpen goed na te denken over hoe de applicatie er onder de motorkap uit komt te zien.

10.1.4 Elaboration rapport

Ik denk zelf dat het elaboration rapport waardevolle informatie geeft aan ontwikkelaars van de applicatie. Ontwikkelaars kunnen van de belangrijkste functionaliteiten duidelijk zijn hoe ik dit heb opgebouwd in een sequentie diagram. Daarnaast geeft het klassendiagram informatie over de applicatie is opgebouwd.

10.1.5 Construction rapport

Van het construction rapport heb ik zelf eigenlijk weinig gebruik gemaakt, omdat het voor mij al duidelijk is wat ik precies heb gemaakt. Aan de hand van het construction rapport heb ik wel feedback gehad op mijn geschreven code en dit zal er volgens hun goed uit.

10.2 Procesevaluatie

Ik vind zal dat proces over het algemeen goed is verlopen. De gebruikte methode sloot goed aan bij het project. Ik denk wel dat ik tijdens een vervolg project eerder zou kiezen voor Scrum dan voor RUP. Dit omdat bij een RUP traject ook veel documentatie hoort, bij sommige projecten is de tijd er gewoon niet voor om al deze documentatie te maken.

Ik heb gebruik gemaakt van enkele nieuwe technieken, zo heb ik binnen mijn project het MVVM design pattern toegepast. In eerste instantie vond ik het veel extra werk, ik had namelijk zo'n beetje het dubbele aantal klassen vergeleken met normaal. Maar nadat ik er even mee heb gewerkt begon ik zeker ook de voordelen ervan in te zien. Ik zou dan ook zeker bij een volgend vergelijkbaar project weer gebruik maken van dit design pattern.

Ik heb ook nog gebruik gemaakt van entity framework 6 en hier heb ik eigenlijk een beetje gemengde gevoelens bij. Ik zou het zeker weer gebruiken als het gaat om een applicatie waarbij er geen grote hoeveelheid aan data te pas komt. Maar als het gaat om een soortgelijke applicatie, dan zou ik er nog is goed over nadenken. Naast de vele voordelen heb ik namelijk ook zeker een aantal nadelen ondervonden en hiermee bedoel ik het omgaan met grote hoeveelheid aan data. Doordat het entity framework zelf traag was met bijvoorbeeld het toevoegen van een grote hoeveel records tegelijk heb ik tijd moeten besteden aan het zoeken naar een oplossing hiervoor. Uiteindelijk denk ik dat het wisselen van techniek meer tijd gaat kosten dan het omgaan met enkele nadelen van het entity framework. Een aantal problemen heb ik namelijk al kunnen tackelen. Mede hierdoor zou ik er dan voor kiezen om er weer gebruik van te gaan maken. Maar voordat het zover wil ik het eerst nog wel met mijn collega ontwikkelaars over hebben en vragen wat hun hiervan vinden. Want je bent vaak niet de enige die aan een project werkt.

Naast het entity framework heb ik ook gebruik gemaakt van code first. Zonder enige twijfel zou ik hiervan bij een soortgelijk project weer gebruik van maken. Er zitten zoveel voordelen aan, hier kunnen de enkele nadelen niet tegen opwegen. Ik zou dit dus zonder te aarzelen weer gebruiken in een vervolg project. Een nadeel is wel dat dit alleen gebruikt kan worden in combinatie met het entity framework.

Als ik naar de planning kijk in het grote geheel dan komt dit redelijk overeen met de uitgevoerde werkzaamheden. Wel heb ik bij sommige onderdelen meer of minder tijd besteed dan in eerste instantie geplant. Het performance onderzoek is er hier één van. Ik had hiervoor een halve week ingeplant, uiteindelijk ben ik er zeker een anderhalve week mee bezig geweest. Dit kwam doordat meerdere technieken heb moeten onderzoeken voordat ik echt tot een oplossing kwam. Tijdens het maken van de planning had ik het idee dat ik de performance kon verbeteren door enkel indexen toe te voegen aan de database. Dit was helaas niet het geval. Het koppelen met de leveranciers heeft uiteindelijk minder tijd gekost dan in eerste instantie verwacht. Dit kwam doordat er uiteindelijk maar twee leveranciers zijn gekoppeld aan het systeem in plaats van 6. De tijd die hierdoor vrij kwam ik heb voor het grootste gedeelte gebruikt in mijn performance onderzoek.

Het lastigste onderdeel vond ik het oplossen van de trage query's. Het performance onderzoek dus. Ik heb meerdere technieken geprobeerd, met helaas wisselend succes. Ook het toepassen van Lucene binnen mijn project heeft mij aardig wat tijd gekost, bij elkaar uiteindelijk ongeveer een week. Dit kwam doordat er niet echt een duidelijke handleiding was voor te toepassen van Lucene binnen het .NET framework. Ook zijn er verschillende Lucene versies in omgang, dit maakte het ook niet echt makkelijker. Mijn collega ontwikkelaars hadden allen ook nog niet eerder gebruikt gemaakt van Lucene, dus ook hier kan ik geen hulp bij vragen. Na het doorlezen van diverse artikelen het zelf gewoon te proberen ben ik er wel uitgekomen. Maar zoals ik aangaf, koste het wel enige tijd.

In het vervolg zou enkele dingen anders aanpakken. Eén hiervan is het opdelen van construction fase in meer iteraties. Hierdoor kan ik ook weer sneller feedback van de opdrachtgever vragen. Een probleem wat ik tegenkwam was namelijk dat bij de winkelwagen module flinke wijzigingen moesten worden doorgevoerd in het technisch ontwerp door enkele nieuwe requirements. Hierdoor is er in eerste instantie onnodig tijd verspilt aan het bouwen van de winkelwagen module. Verder zou ik

meer demo's willen geven aan de opdrachtgever, maar dit hoort in principe bij het bovenstaande probleem.

10.3 Evaluatie beroepstaken

Onderstaand zal ik per beroepstaken toelichten op welk niveau ik vind dat ik deze heb uitgevoerd.

10.3.1 2.1 Opstellen gegevensmodel voor een database

Ik vind zelf dat ik deze beroepstaak gedeeltelijk op niveau lastig heb uitgevoerd. Ik had namelijk wel een database ontwerp gemaakt voor de eerste versie van mijn applicatie. Ik heb deze alleen niet meer bijgewerkt, aangezien ik gebruik heb gemaakt van code first. Maar desalniettemin vind ik nog wel dat ik dit voor een deel op niveau lastig heb uitgevoerd.

10.3.2 2.2 Ontwerpen, bouwen en bevragen van een database

Ik vind zelf dat ik deze beroepstaak voor een deel op het niveau complex heb uitgevoerd. Het ontwerp van mijn database is eigenlijk gewoon mijn model klasse, bij het maken van het model is ook rekening gehouden met de database. Zie bijvoorbeeld de tussen tabel tussen orderpart en product. Deze is special voor het database ontwerp aangemaakt. Daarnaast bevraag ik de database constant, alleen niet via een SQL query maar via een LINQ query. Als laatste heb ik rekening gehouden met de performance, zie hiervoor hoofdstuk 8.7. De oplossing is dan uiteindelijk niet op SQL niveau uitgevoerd, maar dat was in eerste instantie wel de bedoeling.

10.3.3 3.2 ontwerpen systeemdeel

Ik vind zelf dat ik deze beroepstaak op niveau 3 heb uitgevoerd. Ik heb namelijk gebruik gemaakt van design patterns in mijn ontwerp (template). Ik heb rekening gehouden met de samenhang van objecten. Er is een duidelijke scheiding aanwezig tussen het model en de view (MVVM pattern). Ook is alles object georiënteerd opgebouwd.

10.3.4 3.3 bouwen applicatie

Ik vind zelf dat ik deze beroepstaak ook op niveau complex heb uitgevoerd. Ik heb namelijk verschillende design patterns toegepast. Ik heb diverse voor mij nieuwe technieken goed kunnen toepassen binnen de applicatie, een voorbeeld hiervan is code first. Ik heb de standaarden van Interpulse aangehouden tijdens het bouwen van de applicatie. Ik heb gebruik gemaakt van diverse externe libaries zoals BulkInsert. Ook het onderdeel versie beheer is aan de pas gekomen, ik heb namelijk gebruik gemaakt van GIT.

10.3.5 3.5 uitvoeren van en rapporteren over het testproces

Ook deze beroepstaak is op niveau complex uitgevoerd. De belangrijkste functionaliteiten van mijn applicatie worden getest met unit testscripts. Ik maak ik gebruik van het Effort framework om code die gebruik maakt van het entity framework te kunnen testen. De resultaten van de testen worden beschreven in het testrapport welke te vinden is in hoofdstuk 9 van het construction rapport van iteratie 2. Daarnaast heb ik voor mijn applicatie ook smoke testen geschreven. Voor het schrijven van deze smoke testen heb ik gebruik gemaakt van het Robot Framework in combinatie met de libary Selenium2Libary.

11 Geraadpleegde bronnen

Algemeen

https://www.google.nl/

http://msdn.microsoft.com/

http://stackoverflow.com/

Entity framework

http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

Code first

http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application

Bootstrap

http://getbootstrap.com/

Select2

https://select2.github.io/

Amcharts

http://www.amcharts.com/

Sweetalert

http://tristanedwards.me/sweetalert

Bulkinsert

https://efbulkinsert.codeplex.com

Blitz index

http://www.brentozar.com/blitzindex/

Full-tekst search

http://msdn.microsoft.com/en-us/library/ms142571.aspx

http://msdn.microsoft.com/en-us/library/ms187317.aspx

http://sysmagazine.com/posts/189806/

Lucene

http://lucenenet.apache.org/

http://lucene.apache.org/core/2_9_4/queryparsersyntax.html

http://www.codeproject.com/Articles/320219/Lucene-Net-ultra-fast-search-for-MVC-or-WebForms#Step1

Effort

http://effort.codeplex.com/

http://www.codeproject.com/Articles/460175/Two-strategies-for-testing-Entity-Framework-Effort

Smoketesten

http://robotframework.org/

https://github.com/rtomac/robotframework-selenium2library

https://bitbucket.org/robotframework/webdemo/wiki/Home#rst-header-downloading-demo-package

http://rtomac.github.io/robotframework-selenium2library/doc/Selenium2Library.html#Page%20Should%20Contain

12 Bijlage

Afstudeerplan

Plan van aanpak

Onderzoeksrapport

Inception rapport

Elaboration rapport

Techdata pdf voor het downloaden van de producten

Construction rapport

Construction rapport 2

Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2014-2.1 (start uiterlijk 1 september 2014)
Startdatum uitvoering afstudeeropdracht: 1 september 2014
Inleverdatum afstudeerdossier volgens jaarrooster: 9 januari 2015

Studentnummer: 10049754
Achternaam: dhr Roelen
Voorletters: D
Roepnaam: Donny
Adres: Fonteinkruidhof 8
Postcode: 2215GS
Woonplaats: Voorhout
Telefoonnummer: 0252 235495
Mobiel nummer: 0624563854
Privé emailadres: donny_roelen@hotmail.com

Opleiding: Informatica
Locatie: Den Haag
Variant: voltijd

Naam studieloopbaanbegeleider: Anneke Wieman
Naam begeleidend examinator: Gerard Mijnarends
Naam tweede examinator: Juul Maas

Naam bedrijf: Interpulse
Afdeling bedrijf: Back-end
Bezoekadres bedrijf: Zoeterwoudsesingel 56
Postcode bezoekadres: 2313 EK
Postbusnummer:
Postcode postbusnummer:
Plaats: Leiden
Telefoon bedrijf: 071 - 5665282
Telefax bedrijf: 071 - 5665283
Internetsite bedrijf: http://www.interpulse.nl/

Achternaam opdrachtgever: dhr van Elsacker
Voorletters opdrachtgever: M
Titulatuur opdrachtgever:
Functie opdrachtgever: Hoofd manager ICT
Doorkiesnummer opdrachtgever:
Email opdrachtgever: m.vanelsacker@interpulse.nl

Achternaam bedrijfsmentor: dhr Maagdelijn
Voorletters bedrijfsmentor: A
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: Projectleider ontwikkeling
Doorkiesnummer bedrijfsmentor:
Email bedrijfsmentor: A.Maagdelijn@Interpulse.nl

NB: bedrijfsmentor mag dezelfde zijn als de opdrachtgever

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:

Ontwerpen en ontwikkelen van een hardware offerte tool.

Opdrachtomschrijving

1. Bedrijf

Interpulse Automatisering B.V. is een full-service internet en automatiseringsbedrijf. Het bedrijf is begonnen als een ICT bedrijf met 7 werknemers in 2000. 14 jaar later telt Interpulse 19 werknemers en blijft het aantal klanten nog steeds groeien. Interpulse is gevestigd in het centrum van Leiden.

Interpulse biedt de volgende diensten aan:

- Systeembeheer
- Systeemontwikkeling
- Internetdiensten
- Consultancy

Deze diensten worden geleverd namens de afdelingen: directie, management, beheer en ontwikkeling. Er is 1 algemeen directeur, met daaronder het managementteam. Daaronder vallen de 2 werkvelden Ontwikkeling en Beheer. De opdracht wordt uitgevoerd binnen de afdeling ontwikkeling.

2. Probleemstelling

Het samenstellen van hardware offertes, waarbij de prijzen dagelijks wijzingen is een tijdrovende klus. Er is geen compleet beeld van alle prijzen. Bestellingen plaatsen bij leveranciers moet voldoen aan bepaalde regels, zoals minimum orderbedrag. Het verzamelen van meerdere orders is op dit moment niet goed mogelijk een foutgevoelig.

In de huidige situatie bestaat het samenstellen van hardware offertes voornamelijk uit handwerk. Producten worden bij verschillende leveranciers vergeleken op de kosten en of deze op voorraad is. Het opstellen van deze offertes wordt meerdere keren per week gedaan.

Het is de bedoeling dat er een tool wordt ontwikkeld die producten kan vergelijken bij de groothandelaars (soort van tweakers pricewatch). Je wilt bijvoorbeeld van een bepaalde I-7 processor weten waar deze het goedkoopst verkrijgbaar is of waar je deze het snelst geleverd kan krijgen. Door het automatiseren van dit proces wordt er veel tijd bespaard. Daarnaast is de het de bedoeling dat deze tool in de toekomst ook offertes kan maken aan de hand van gekozen artikelen bij de groothandelaars. Een verkoper kan dan bijvoorbeeld bij de klant gelijk een offerte opmaken, dan weet de klant precies wat een bepaald systeem kost. Dus naast de tijd die het zal gaan besparen kan het ook extra inkomsten gaan opleveren.

3. Doelstelling van de afstudeeropdracht

Het doel van de opdracht is de benodigde tijd voor het samenstellen van hardware offertes aanzienlijk terugbrengen.

4. Resultaat

Het uiteindelijk resultaat van de opdracht zal bestaan uit de hardware offerte tool. Met deze tool kan de opdrachtgever veel tijd besparen bij het opmaken van hardware offertes. Er hoeft namelijk niet meer handmatig te worden vergeleken, al deze informatie kan snel worden gevonden met de tool.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

De student begint zijn afstudeerperiode met een soort van onderzoekfase, hierin wordt informatie verzameld om de software te kunnen ontwerpen en ontwikkelen. De volgende activiteiten komen aan bod tijdens deze fase:

- Plan van aanpak
  - Interviewen opdrachtgever
  - Uitwerken onderdelen plan van aanpak
- Huidige situatie / gewenste situatie onderzoeken en in kaart brengen
  - Interviewen opdrachtgever
  - Hoe ziet de gewenste situatie eruit?
- Onderzoek software ontwikkelmethode
  - Welke methode past het best bij het ontwikkelen van de offerte tool?
    - Scrum, Waterfall, RUP, RAD, XP

Zodra de onderzoeksfase is afgerond begint de ontwerpfase, in deze fase worden de volgende activiteiten uitgevoerd:

- Requirements
  - Opstellen requirements
  - Prioriteren requirements met opdrachtgever.

- Ontwerpen
  - Technisch ontwerp
    - Koppeling met SOAP of rest API
    - UML modellen
      - Analysis en design klassendiagram
      - Sequence diagrammen
  - Database ontwerp
    - Entity-relationship diagram (ERD) maken
    - Het gaat om veel data (Big data)
    - Datastructuur zo opbouwen zodat de performance hoog blijft
      - Gebruik maken van indexes

Zodra de ontwerpfase is afgerond kan de student beginnen met het bouwen van de software, de volgende activiteiten komen nu aan bod:

- Ontwikkelen / bouwen software
  - Opstarten Visual studio project
  - Bootstrap configureren voor de opmaak
  - Database bouwen
  - Koppeling maken met leveranciers
    - Eenmalig data ophalen van alle aanwezige producten
    - Dagelijks data ophalen (de wijzigingen)
  - Zoek / filter module bouwen

        - Goedkoopste
        - Snelste levertijd
    - Ontwikkelen beheer module
        - Artikel categorieën per leverancier
        - Minimum order bedrag leverancier
    - Prijs geschiedenis (grafieken)
- Testen
    - Gebruikerstest
    - Unit test

Mocht tijdens de afstudeerperiode blijken dat de student nog tijd over heeft kan ervoor worden gekozen om één van de onderstaand modules op te nemen in het ontwikkeltraject :

- Email alerts
    - Grootste prijsstijgers en dalers
- Direct bestellen bij de leverancier via de offerte tool
    - Onderzoeken naar de mogelijkheden hiervan
- Integreren in een offerte document, bijvoorbeeld Word
    - Onderzoek doen naar de mogelijkheden en beste oplossing
- Koppeling maken met B2B webshop interpulse op basis van NopCommerce
    - Onderzoek doen naar de mogelijkheden en beste oplossing
    - Direct bestellen door klanten interpulse

Planning:

Activiteit	Werkdagen
Onderzoeksfase	**13**
Plan van aanpak	3
Huidige / gewenste situatie onderzoeken	5
Onderzoek software ontwikkelmethode	5
Ontwerpfase	**17**
Requirements opstellen en prioriteren	3
Technisch ontwerp	10
Database ontwerp	3
Ontwikkelfase	**40**
Opstarten Visual Studio project en configureren bootstrap	2
Bouwen database	2
Koppeling maken met leveranciers	10
Bouwen zoek en filter module	7
Beheer module	6
Prijs geschiedenis	6
Testen	7
Overig	**15**
Afstudeerverslag	15

6. Op te leveren (tussen)producten

De volgende producten worden opgeleverd tijdens de afstudeerperiode:

- Plan van aanpak
- Rapport voor de vergelijkingen van de software ontwikkelmethode ’s
- Technisch ontwerp rapport
- Databaseontwerp rapport
- Database
- Sourcecode
- Hardware offerte tool

7. Te demonstreren competenties en wijze waarop

Tijdens de afstudeerperiode worden de volgende competenties uitgevoerd op minimaal niveau 3:

- *2.1 Opstellen gegevensmodel voor een database*
  Er wordt een gegevensmodel opgesteld, namelijk een ERD diagram.

- *2.2 Ontwerpen, bouwen en bevragen van een database*
  Het gaat om een database met heel veel data (big data). Een belangrijk onderdeel hiervan is zorgen dat de performance hoog blijft. Hier wordt rekening mee gehouden bij het ontwerpen van de database en er zal gebruik worden gemaakt van indexes. Ook moet de huidige data van de leveranciers in onze database komen te staan, echter komen de artikel namen niet altijd overeen bij de leveranciers, hier moet dus rekening mee worden gehouden.

- *3.2 Ontwerpen systeemdeel*
  Voordat er wordt ontwikkelt wordt het systeem eerst ontworpen. Dit wordt gedaan in de UML tool Sparx enterprise architect. In het ontwerp wordt rekening gehouden toekomstige uitbreidbaarheid.

- *3.3 Bouwen applicatie*
  De tool wordt ontwikkeld in ASP.NET MVC 5, als editor wordt gebruik gemaakt van Visual Studio 2013 en er wordt gewerkt op Windows 8. Daarnaast wordt er gewerkt aan de hand van OTP (Ontwikkel, Test en Productie) omgevingen. Verder wordt er gebruik gemaakt van GIT als versiebeheer tool.

- *3.5 Uitvoeren van en rapporteren over het testproces*
  Aan het eind van het ontwikkel traject worden de belangrijkste functionaliteiten getest met unittests. De unit tests worden geschreven met behulp van één van de beschikbare test frameworks binnen ASP.NET. Daarnaast worden er nog een gebruikerstest uitgevoerd.

Onderzoeksplan

Softwareontwikkelmethoden vergelijken

Datum: 2 september 2014

Plaats: Leiden

Auteur: Donny Roelen

Versie: 1.1

Versiebeheer

Versie	Wijzigingen	Datum
0.1	Eerste opzet	1 September 2014
0.2	Hoofdvraag en deelvragen opgesteld	1 september 2014
0.5	Uitwerken methoden en aanpak	1 september 2014
0.6	Opstellen planning	1 september 2014
1.0	Eerste versie onderzoeksplan	1 september 2014
1.1	Deelvragen en aanpak bijgewerkt.	2 september 2014

Inhoudsopgave

1 Inleiding

Dit document betreft het onderzoeksplan voor het onderzoek naar de meeste geschikte softwareontwikkelmethode tijdens de afstudeerperiode van de afstudeerder. De wijze waarop het onderzoek zal worden uitgevoerd wordt beschreven in het onderzoeksplan.

2 Onderzoekskader

In dit hoofdstuk zal het onderwerp van het onderzoek worden beschreven en zullen de probleemstelling, hoofdvraag en deelvragen worden vastgelegd. Om te voorkomen dat het onderzoek te breed wordt, wordt het onderzoek afgebakend. De afbakening van het onderzoek wordt beschreven onder het kopje 'Afbakening'.

2.1 Probleemstelling

Er zijn zoveel verschillende methodes beschikbaar, dagelijks kom je termen als Scrum en Agile tegen, maar wat is wanneer nou echt in welke situatie geschikt. Dit zijn vragen die mij bezighouden. Tijdens mijn afstudeerperiode moet ik gaan werken volgens één van deze methoden. Op school hebben we vooral gewerkt met de waterval methode en het bedrijf maakt gebruik van Scrum. Is het verstandig om tussen één van deze methoden te kiezen, of is een ander methode nog geschikter voor mijn afstudeeropdracht.

2.2 Hoofdvraag

Welke softwareontwikkelmethode kan ik als afstudeerder van de Haagse Hogeschool het best gebruiken bij mijn afstudeeropdracht?

2.3 Doelstelling

Het doel van het onderzoek is om aan de hand van de resultaten de meest geschikte softwareontwikkelmethode te kiezen voor de afstudeeropdracht.

2.4 Deelvragen

De volgende deelvragen worden opgesteld om de hoofdvraag te kunnen beantwoorden:

- Wat zijn de kenmerken van de afstudeeropdracht?
- Wat zijn de voordelen van deze 5 methoden?
- Wat zijn de nadelen van deze 5 methoden?
- Voor wat voor soort projecten wordt deze 5 methoden vooral gebruikt?
- Welke methode heeft de meeste overeenkomsten met de kenmerken van mijn afstudeeropdracht?
- Welke methode heeft de minste overeenkomsten met de kenmerken van mijn afstudeeropdracht?

2.5 Afbakening

Om ervoor te zorgen dat het onderzoek binnen het kader blijft, baken ik het onderzoek af. Het onderzoek wordt afgebakend op de onderstaande punten:

- Er zijn vele softwareontwikkelmethoden beschikbaar, voor dit onderzoek worden alleen de onderstaande methoden meegenomen:
    - Scrum
    - Waterval
    - RUP (Rational Unified Process)

    - RAD (Rapid Application Development)
    - XP (Extreme Programming)
- Het onderzoek heeft betrekking tot de situatie van deze afstudeeropdracht, voor andere afstudeeropdrachten is een andere methoden misschien geschikter.

3 Methoden

Om de deelvragen en hoofdvraag te kunnen beantwoorden wordt gebruik gemaakt van verschillende onderzoeksmethoden. In dit hoofdstuk worden deze onderzoeksmethoden verder toegelicht.

3.1 Kwalitatief onderzoek

Voor het verzamelen van de data ga ik gebruik maken van een kwalitatief methoden. De data zal worden verzameld aan de hand van beschikbare literatuur over de desbetreffende methoden op het internet en daarnaast wordt er nog gebruik gemaakt van de aanwezige literatuur binnen het bedrijf.

3.1.1 Literatuuronderzoek

Een literatuuronderzoek maakt over het algemeen deel uit van elk onderzoek en is een vast onderdeel van een goede opzet. Met behulp van het literatuuronderzoek kan ik onderzoek doen naar de softwareontwikkelmethoden. Voor het uitvoeren van het literatuuronderzoek wordt voornamelijk gebruik gemaakt van het internet.

4 Aanpak

In dit hoofdstuk beschrijf ik hoe ik het onderzoek ga aanpakken. Om het onderzoek in goed te laten verlopen ga ik gebruik maken van de volgende fases:

- Voorbereiding
- Uitvoering
- Rapportage en presentatie

4.1 Voorbereiding

De eerste fase is de voorbereidingsfase, in deze fase ga ik me verkennen binnen het onderwerp. Daarnaast bestaat deze fase uit het uitwerken van het onderzoeksplan.

4.2 Uitvoering

In deze fase worden de deelvragen uitgewerkt. Onderstaand ga ik per deelvraag beschrijven hoe ik deze ga beantwoorden.

- Wat zijn de kenmerken van de afstudeeropdracht?
- Wat zijn de voordelen van deze 5 methoden?
- Wat zijn de nadelen van deze 5 methoden?
- Voor wat voor soort projecten worden deze 5 methoden vooral gebruikt?
- Welke methode heeft de meeste overeenkomsten met de kenmerken van mijn afstudeeropdracht?
- Welke methode heeft de minste overeenkomsten met de kenmerken van mijn afstudeeropdracht?

4.2.1 Wat zijn de kenmerken van de afstudeeropdracht?

De eerste deelvraag kan ik zelf beantwoorden door na te denken over mijn afstudeerplan. Ik ga na wat mijn afstudeeropdracht kenmerkt en beschrijf dit.

4.2.2 Wat zijn de voordelen van deze 5 methoden?

De voordelen van de desbetreffende methoden zijn gewoon op het internet beschikbaar. Ik ga bij minimaal 5 verschillenden bronnen de voordelen bekijken. Vervolgens analyseren ik voor de verschillende bronnen de overeenkomen en verschillen. Mocht het overeenkomen kunnen we ervanuit gaan dat de informatie klopt. Als er verschillen zijn of er is tegenstijdige informatie, wordt er gekeken hoe dit komt.

4.2.3 Wat zijn de nadelen van deze 5 methoden?

De nadelen van de desbetreffende methoden zijn gewoon op het internet beschikbaar. Ik ga bij minimaal 5 verschillenden bronnen de nadelen bekijken. Vervolgens analyseren ik voor de verschillende bronnen de overeenkomen en verschillen. Mocht het overeenkomen kunnen we ervanuit gaan dat de informatie klopt. Als er verschillen zijn of er is tegenstijdige informatie, wordt er gekeken hoe dit komt.

4.2.4 Voor wat voor soort projecten worden de methoden vooral gebruikt?

Ik ga na bij wat voor soort projecten de betreffende methoden vooral gebruikt worden. Dus wordt het gebruikt voor kleine of juist grotere projecten. Hoeveel tijd is er voor deze projecten en nog meer.

4.2.5 Welke methode heeft de meeste overeenkomsten met de kenmerken van mijn afstudeeropdracht?

Zodra alle bovenstaande deelvragen zijn beantwoord kan ik deze deelvraag gaan beantwoorden. Ik ga na welke methoden de meeste overeenkomsten heeft met de kenmerken van mijn afstudeeropdracht. Ik ga kijken welke methode de meeste raakvlakken heeft met de kenmerken van mijn afstudeeropdracht.

4.2.6 Welke methode heeft de minste overeenkomsten met de kenmerken van mijn afstudeeropdracht?

Zodra alle bovenstaande deelvragen zijn beantwoord kan ik deze deelvraag gaan beantwoorden. Ik ga na welke methoden de meeste overeenkomsten heeft met de kenmerken van mijn afstudeeropdracht. Ik ga kijken welke methode de minste raakvlakken heeft met de kenmerken van mijn afstudeeropdracht.

4.3 Rapportage en presentatie

De laatste fase is de fase rapportage en presentatie, deze fase bestaat uit het uitwerken van de onderzoeksresultaten in het onderzoeksrapport.

5 Planning

Fase	Werkzaamheden	Datum
Voorbereiding	Verdiepen in het onderwerp	1 september 2014
	Eerste opzet onderzoeksplan	1 september 2014
	Definitief onderzoeksplan	2 september 2014
Uitvoering	Zoeken naar data	2 september 2014
	Analyseren data	2 september 2014
Rapportage	Eerste opzet onderzoeksrapport	2 september 2014
	Definitieve versie onderzoeksrapport	3 september 2014

Onderzoeksrapport

Softwareontwikkelmethodes vergelijken

Datum: 3 september 2014

Plaats: Leiden

Auteur: Donny Roelen

Versie: 1.0

Versiebeheer

Versie	Wijzigingen	Datum
0.1	Eerste opzet	1 September 2014
0.2	Uitwerken onderzoekskader	1 September 2014
0.6	Uitwerken resultaten	2 September 2014
0.8	Uitwerken literatuurlijst	2 September 2014
1.0	Conclusie uitwerken en afmaken eerste versie	2 September 2014

Inhoudsopgave

1 Inleiding

Dit document bevat het onderzoeksrapport van de afstudeerder. De afstudeerder doet onderzoek naar de meest geschikte softwareontwikkelmethode voor de afstudeerperiode. De wijze waarop het onderzoek wordt uitgevoerd en de resultaten worden in dit onderzoek beschreven.

2 Onderzoekskader

2.1 Aanleiding

Er zijn vele methodes beschikbaar, dagelijks kom je termen als Scrum en Agile tegen, maar wat is nou echt in welke situatie geschikt. Dit vind ik als afstudeerder lastig te beantwoorden. Daarom heb ik ervoor gekozen om dit onderzoek te doen.

2.2 Doelstelling

Door het uitvoeren van dit onderzoek kan ik de voor mij meest geschikte softwareontwikkelmethode kiezen. Dit voorkomt dat ik er later achter kom dat de door mij gekozen softwareontwikkelmethode toch niet geschikt was voor deze specifieke afstudeeropdracht.

2.3 Methoden

Tijdens het onderzoek is er gebruikt gemaakt van verschillende methoden en technieken. Als belangrijkste onderdeel is er een literatuuronderzoek uitgevoerd.

Voor een volledige beschrijving van de methode verwijs ik u naar mijn onderzoeksplan.

2.4 Aanpak

Om mijn hoofdvraag te kunnen beantwoorden heb ik de volgende deelvragen opgesteld:

- Wat zijn de kenmerken van de afstudeeropdracht?
- Wat zijn de voordelen van deze 5 methoden?
- Wat zijn de nadelen van deze 5 methoden?
- Voor wat voor soort projecten worden deze 5 methoden vooral gebruikt?
- Welke methode heeft de meeste overeenkomsten met de kenmerken van mijn afstudeeropdracht?
- Welke methode heeft de minste overeenkomsten met de kenmerken van mijn afstudeeropdracht?

3 Uitvoering onderzoek

In dit hoofdstuk worden de resultaten van het onderzoek gepresenteerd aan de hand van de opgestelde deelvragen.

3.1 Wat zijn de kenmerken van de afstudeeropdracht

Onderstaand benoem ik de kenmerken van mijn afstudeeropdracht.

- Eenmalig
- Project wordt zelfstandig uitgevoerd
- Er is één stakeholder
- Einddatum is bekend

- Er staan 17 weken voor geplant
- Bij voorkeur een iteratieve methode
- Feedback opdrachtgever
- Er moet uitgebreid gedocumenteerd worden
    - Wensen en eisen
    - Functioneel ontwerp
    - Technisch ontwerp
- De software wordt ontwikkel in asp.net
- Software moet worden getest
- Het project is relatief klein.
- De scope is vanaf het begin af aan duidelijk.

3.2 Wat zijn de voordelen van deze 5 methoden

Onderstaand beschrijf ik per methode de naar mij mening belangrijkste kenmerken.

3.2.1 Scrum

Onderstaand een lijst met voordelen van het toepassen van Scrum:

- Verhoogt de effectiviteit van het team.
    - Bij scrum werken verschillende disciplines gelijktijdig samen. Dit bevordert het overleg, waardoor de kwaliteit enorm toeneemt.
    - Delen van kennis
    - Dagelijkse meetings zorgen ervoor dat je de productiviteit van de teamleden in de gaten kan worden gehouden.
- Biedt een optimale Return On Investment (ROI).
- Iedere 2-4 weken is er sprake van een oplevering van een stuk werkende software.
- Biedt duidelijk inzicht in de voortgang van het project/software development.
- Scrum biedt een hoge leverzekerheid. Bij traditionele methodes bestaat altijd het risico dat de oplossing bij oplevering niet (meer) voldoet aan de wensen van de markt. De flexibele scope van scrum maakt het mogelijk om snel bij te sturen en dit soort problemen te voorkomen.
- Scrum is een snelle methode. In sommige gevallen zijn ze zelfs twee keer sneller dan bij traditionele ontwikkelmethodes. Scrum heeft dus een korte time-to-market.
- Klantbetrokkenheid, de klant maakt iedere dag de voortgang mee en kan er direct op reageren.
- Bespaart het bedrijf tijd en geld
    - Minder overhead
- Er kan makkelijk worden omgegaan met wijzigingen door de korte sprints en constante feedback.

3.2.2 Waterval

Onderstand de voordelen van het gebruik van waterval:

- Wanneer in het begin van het project fouten worden ontdekt, kost het minder inspanning (en dus minder tijd en geld) om deze fouten te herstellen. Bij de watervalmodel wil men alle fasen eerst goed afsluiten voordat men verder gaat met de volgende fase. Men gaat ervan uit dat de fasen altijd goed zijn voordat men verder gaat met een volgende fase.
- Het is een rechttoe-rechtaan methode. De manier van werken zorgt ervoor dat er zich concrete fasen vormen. Hierdoor weet men in welke fase men zit.

- Men kan in deze methode gebruikmaken van mijlpalen. Mijlpalen kunnen worden gebruikt om de voortgang van het project in te schatten.
- Nieuwe mensen die bij een project komen kunnen door middel van de nadruk die op de documentatie wordt gelegd, beter en sneller betrokken worden bij het proces.
- Doordat de methode zeer bekend is en mensen er ervaring mee hebben, is er veel draagvlak en is gemakkelijker om met de methode te werken.
- Waterval werk goed voor kleine projecten waar de wensen en eisen duidelijk zijn.
- Waterval legt de nadruk op kwaliteit en niet op planning en kosten.

3.2.3 RUP

Onderstaand de voordelen van het gebruik van RUP:

- Het is een complete methode die de nadruk legt op documentatie.
- Het is een proactieve methode, de methode kan goed omgaan met veranderende requirements.
- Minder tijd nodig voor de integratie, aangezien de integratie gedurende het gehele software ontwikkel traject aan de gang is.
- Je bent minder tijd kwijt aan het ontwikkelen door het hergebruiken van componenten.
- Reguliere feedback van en naar de stakeholders.
- Efficiënt gebruik van middelen.
- Je levert precies op wat de klant wilt.
- Problemen komen vroeg in het project naar boven.
- Open en publiek

3.2.4 RAD

Onderstaand de voordelen van het gebruik van RAD:

- Online maatwerkapplicaties kunnen zeer snel worden ontwikkeld.
- Ontwikkelingskosten liggen vele malen lager dan bij andere methoden.
- Veelvoorkomende functies zijn al aanwezig.
- Koppelingen met andere online en offline systemen zijn eenvoudig tot stand te brengen.
- Geschikt voor zowel klein maatwerk als voor complexere applicaties.
- Minder ontwikkeltijd nodig.
- Verhoogt de herbruikbaarheid van componenten.
- Aanmoedigt de klant om feedback te geven.
- Integratie vanaf het begin lost veel integratie problemen op.
- Alle software prototypes kunnen worden bewaard in de repository voor toekomstig gebruik.
- De projectmanager kan duidelijk de kosten voor een project inschatten.
- Flexibel en kan omgaan met veranderingen.
- Moedigt de eindgebruiker aan om betrokken te zijn.
- Minder kans op fouten door de prototypes.

3.2.5 XP

Onderstaand de voordelen van het gebruik van XP:

- Flexibel
- Snel (deel) resultaten.
- De kosten nemen niet exponentieel toe, maar blijven lineair toenemen. Dit komt omdat er constant word aangepast in plaats van alles van tevoren te plannen.

- Je krijgt veel feedback van de klant op je kleine opleveringen, hierdoor voldoet het product beter aan de eisen van de klant.
- Door het simpele ontwerp van Extreme Programming kunnen er makkelijk dingen toegevoegd worden.
- Extreme Programming vind veranderingen in het project geen probleem.
- De klant kan beslissen welke eisen als eerst aan bod komen
- Bespaart kosten, ontwikkelaars houden zich bezig met ontwikkelen en verspelen geen tijd aan documentatie.
- Verminderd risico's, door de taken op te delen in modules is niet langer één iemand verantwoordelijk voor een bepaalde taak.
- Er wordt minder tijd verspeeld aan onnodige features.
- Minder giswerk in de planning.
- Grote klantwaardering voor de kosten van een feature.

3.3 Wat zijn de nadelen van deze 5 methoden

3.3.1 Scrum

Onderstaand de nadelen van het gebruik van Scrum:

- Je komt redelijk snel buiten de scope, doordat de stakeholders constant nieuwe functionaliteiten willen.
- Is vooral geschikt voor kleine snel bewegende projecten, aangezien kleine projectteams het beste werken.
- Je hebt veel mensen met ervaring nodig, met veel beginners komt het project vaak niet op tijd af.
- Het team bepaald zelf wat er wordt gedaan, niet de manager.
- Iedereen in een Scrum team moet meerdere rollen kunnen vertegenwoordigen.
- Overcapaciteit
    - Niet iedereen is altijd vol aan het werk
- Scrum biedt geen zekerheid over tijd, kosten en functionaliteit
    - Dit verschilt eigenlijk niet veel met andere methodieken
- Weinig voorbereiding en planning, je kan een sprint beginnen zonder een echte planning te hebben gemaakt.
- Product owner moet aanwezig zijn bij alle meetings.

3.3.2 Waterval

Onderstaand de nadelen van het gebruik van waterval:

- De opdrachtgever wilt soms in de loop van het project iets anders. De watervalmethode gaat er echter vanuit dat de eisen/wensen van de klant niet veranderen. Het aanpassen kost tijd en geld.
- Vanwege de omvang van de fases is het moeilijk in te schatten hoeveel tijd en geld elke fase kost.
- Er werken verschillende mensen aan de fases. Zo werken de ontwerpers aan de ontwerpfase en de bouwers aan de realisatiefase. Tussen deze partijen kunnen verschillende verwachtingen ontstaan over het product.
- Het testen gebeurt pas later in het project. Wanneer de resultaten moeten worden doorgevoerd, kost dit tijd en geld om het product aan te passen.

- De hoeveelheid documentatie kan bij kleine project ervoor zorgen dat er onnodig veel tijd aan verspeeld wordt, wat ten koste gaat van het product zelf.
- Er wordt vaak veel tijd verspeeld doordat de bouwers wachten totdat de ontwerpers klaar zijn.
- Doordat er zoveel nadruk wordt gelegd op documentatie, is de watervalmethode niet efficiënt voor kleinere projecten.
- Het model werkt niet goed bij lange doorgaande projecten.
- Niet geschikt bij projecten waar de wensen en eisen kunnen veranderen.
- Teruggaan naar een vorige fase kost veel geld en tijd.
- Kleine wijzigingen of fouten kunnen voor grote problemen zorgen.

3.3.3 RUP

Onderstaand de nadelen van het gebruik van RUP:

- Teamleden moeten expert zijn in hun gebied om software te ontwikkelen met deze methode.
- Het ontwikkel proces is complex en soms chaotisch.
- Het proces kan te complex zijn om te implementeren.
- Integratie tijdens het gehele proces kan voor problemen zorgen.
- Veel documentatie

3.3.4 RAD

Onderstaand de nadelen van het gebruik van RAD:

- Afhankelijk van een sterk team en individuele performances voor het identificeren van businessrequirements.
- Alleen systemen die in modules worden opgebouwd kunnen worden gebouwd in combinatie met RAD.
- Erg afhankelijk van modeleervaardigheden.
- De methode is niet geschikt voor kleine, unieke of zeer complexe projecten.
- Het team moet als een goed geheel samenwerken, anders werkt deze methode niet.
- Erg afhankelijk van de technische vaardigheden van ontwikkelaars.
- Minder schaalbaarheid en weinig functionaliteiten.

3.3.5 XP

Onderstaand de nadelen van het gebruik van XP:

- Niet elke ontwikkelaar is een teamspeler. Pair programming ligt niet iedereen.
- Het invoeren van XP is redelijk lastig, omdat de verschillende elementen van XP zo op elkaar lijken. Er wordt alleen goed resultaat geboekt als XP volledig wordt toegepast.
- Extreme Programming kan alleen makkelijk worden toegepast indien het een proces is waar weinig uitzonderingen op zijn.
- Extreme Programming eist van de teamleden dat ze over goede sociale vaardigheden bezitten, terwijl in ICT-branche niet de sociaal sterkste personen werken.
- De code moeten steeds op elkaar worden aangesloten. Dit is een hoop werk en misschien werkt het niet samen.
- De klant moet constant beschikbaar zijn, dit kan voor problemen zorgen.
- Vereist veel plannen.
- De scope van het project is vaak in het begin nog onduidelijk.

3.4 Voor wat voor soort projecten worden de methoden vooral gebruikt

3.4.1 Scrum

Scrum wordt gebruikt in allerlei soorten projecten, niet alleen voor software ontwikkelprojecten. Scrum kan worden toegepast in projecten waarvan de requirements regelmatig wijzigen. Scrum werkt het best in projectgroepen van ongeveer 5 t/m 10 projectleden.

3.4.2 Waterval

De waterval methode wordt vooral gebruikt bij projecten waarvan de requirements aan het begin van het proces duidelijk zijn. Mochten er in een later stadium nog eventuele wijzingen of fouten optreden kost het veel tijd om dit op te lossen. Verder wordt de waterval methode vooral gebruikt bij wat kleinere projecten die geen lange doorloop tijd hebben.

3.4.3 RUP

RUP is een softwareontwikkelmethode die vooral wordt toegepast bij wat grote projecten. De kosten voor kleinere projecten reizen vaak de pan uit door de vele documentatie die aan de pas komt.

3.4.4 RAD

RAD wordt vooral gebruikt bij kleine projecten, waarvan je snel resultaat wilt zien. Hoe groter het project wordt, hoe lastiger het wordt om deze in kleine stukjes op te splitsen. Verder is RAD ook geschikt voor projecten waarvan de stakeholder nog niet precies weet wat hij wilt, afgezien van een aantal basis requirements.

3.4.5 XP

XP wordt vaak gebruikt voor projecten waar het risico hoog is. Waar andere methoden falen wordt XP gebruikt. XP is bijvoorbeeld geschikt voor projecten waarin de requirements vaak wijzigen. Een project groep bestaat meestal over uit ongeveer 2 t/m 10 projectleden, maar er zijn genoeg projecten geslaagd met grotere projectgroepen. Nog een belangrijk gegeven van XP is de betrokkenheid van de klant bij het ontwikkelproces, de klant moet dus constant beschikbaar zijn.

3.5 Welke methode heeft de meeste overeenkomsten met de kenmerken van mijn afstudeeropdracht

Als ik de kenmerken van mijn afstudeeropdracht vergelijk met de voor en nadelen van de methoden kom ik tot de conclusie dat RUP de meeste overeenkomsten heeft. Zo focust RUP zich op documentatie, wat voor mij een belangrijk onderdeel is. Daarnaast geeft RUP de mogelijkheid tot input van de opdrachtgever, wat in mijn geval ook van toepassing gaat komen.

3.6 Welke methode heeft de minste overeenkomsten met de kenmerken van mijn afstudeeropdracht

Naar mijn mening is Extreme Programming totaal niet geschikt als softwareontwikkelmethode voor mijn afstudeeropdracht. Bij Extreme Programming is het van belang dat het gehele proces wordt toegepast, mochten er enkele onderdelen van het proces los worden toegepast krijg je niet dezelfde resultaten. Dit wordt in mijn geval erg lastig, aangezien ik de opdracht alleen ga uitvoeren. Ik kan namelijk geen gebruik maken van het zogenaamde “pair programming”. Verder is het voor mijn afstudeeropdracht van belang dat er voor het ontwikkel traject is wordt gedocumenteerd. Dit gaat niet werken met Extreme Programming, aangezien bij Extreme Programming geen documentatie hoort. Al met al is Extreme Programming niet geschikt voor het project wat ik ga uitvoeren.

4 Conclusie

Uit mijn onderzoek is gebleken dat RUP de meest geschikte softwareontwikkelmethode is in mijn situatie. Ik twijfelde zelf in eerste instantie tussen Scrum, waterval en RUP. Van deze drie viel de waterval methode voor mij eigenlijk het eerste af. De waterval methode gaat erg slecht om met wijzigende requirements of fouten die laten in het proces naar boven komen. Verder gaat mijn voorkeur uit naar een iteratieve methode.

Daarna zat ik nog te twijfelen tussen Scrum en RUP, Scrum wordt hier binnen de organisatie ook gebruikt als standaard ontwikkelmethode, dus mijn collega's zijn ermee bekend. Scrum heeft vele voordelen, vooral de Sprints vind ik erg aantrekkelijk. Alleen Scrum had ook een aantal nadelen, Scrum is eigenlijk niet bedoelt voor projecten die maar worden uitgevoerd door één persoon. Scrum is meer bedoelt voor projectgroepen, vandaar ook de vele projectrollen. Daarnaast is Scrum ook vrij open, dit heeft natuurlijk zo zijn voordelen, maar ik zie dit toch ook wel als een nadeel. Ik wil een methode die mij echt structuur geeft en naar mijn mening gaf Scrum dit niet genoeg.

Mijn keuze is dus uiteindelijk gevallen op RUP, RUP geeft mij duidelijke een duidelijke structuur voor de gehele afstudeeropdracht. Ik kan duidelijk zijn hoe ik ervoor sta qua planning. RUP is iteratief, hierdoor kan ik ervoor kiezen om telkens modules van de software te ontwerpen en te ontwikkelen. Aan het eind van een iteratie kan ik feedback vragen aan mijn opdrachtgever en dit meenemen in de volgende iteratie.

5 Literatuurlijst

Voor Scrum heb ik onder andere gebruik gemaakt van de Scrum Guide die beschikbaar wordt gesteld op https://www.scrum.org/

Verder heb ik gebruik gemaakt van vele bronnen die beschikbaar werden gesteld op het internet. Onderstaand per methode een lijst me bronnen waarvan ik gebruik van heb gemaakt:

Scrum

http://www.scrum.nl/site/Wat-is-Scrum-agile-scrum

http://www.inspire.nl/aanpak/scrum/

http://www.jeroenhulscher.nl/2010/waterval-vs-agile-scrum/

http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-agile-scrum-software-development.html

http://blog.belatrixsf.com/benefits-pitfalls-of-using-scrum-software-development-methodology/

http://www.websitedesignandsupport.com/software-design-scrum.html

http://beleidenmeerit.wordpress.com/2012/04/16/scrum-de-voor-en-nadelen/

http://www.agileadvice.com/2011/12/05/referenceinformation/24-common-scrum-pitfalls-summarized/

Waterval

http://nl.wikipedia.org/wiki/Watervalmethode

http://pc-en-internet.infonu.nl/diversen/97898-de-watervalmethode.html

http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/

http://www.ianswer4u.com/2011/11/advantages-and-disadvantages-of.html#axzz3C9378sxk

http://www.buzzle.com/articles/waterfall-model-advantages-and-disadvantages.html

RUP

http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-rup-software-development.html

http://www.slideshare.net/ERICEV/rup-2248862

http://www.scribd.com/doc/120243377/Rational-Unified-Process
http://businessanalysttrainingonline.com/index.php/ba-training-sessions/230-rational-unified-process-rup.html

https://www.cs.odu.edu/~zeil/cs350/latest/Public/processModels/index.html

RAD

http://www.actiview.nl/expertise-producten/web-based-applicaties/rapid-application-development.html

http://istqbexamcertification.com/what-is-rad-model-advantages-disadvantages-and-when-to-use-it/

http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-rad-software-development.html

http://sameeradilhan.com/advantages-and-disadvantages-of-rad-rapid-application-development

http://ecomputernotes.com/software-engineering/rapid-application-development

http://en.wikipedia.org/wiki/Talk%3ARapid_application_development

XP

http://nl.wikipedia.org/wiki/Extreme_programming

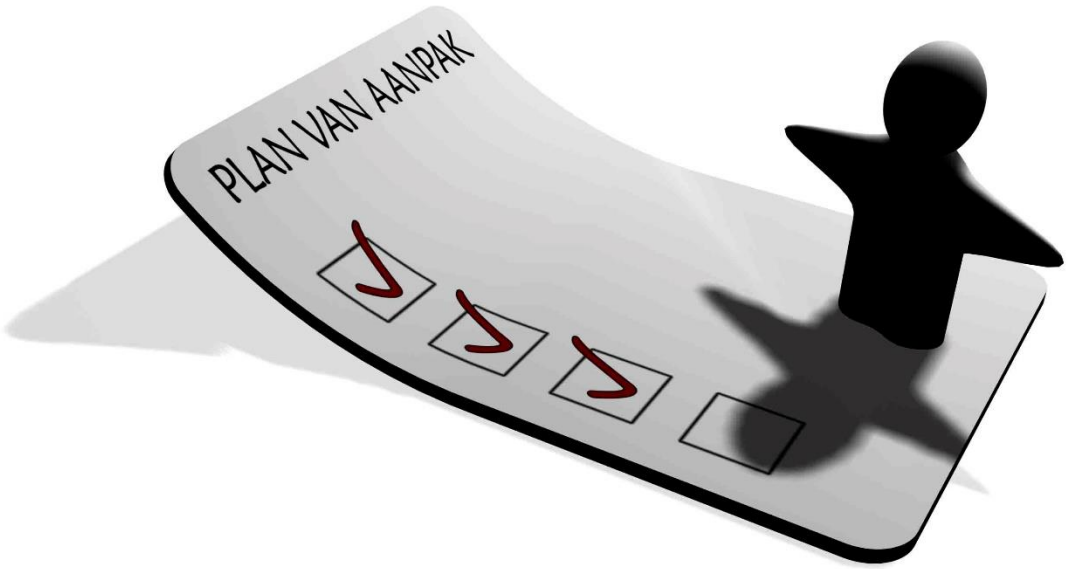
http://www.ukessays.com/essays/information-technology/software-ontwikkeling-methoden-van-agile.php

http://www.brighthubpm.com/methods-strategies/87839-advantages-of-extreme-programming/

http://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages

Plan van aanpak

Interpulse Automatisering



Auteur: Donny Roelen

Datum: 10 september 2014

Versie: 1.1

Inhoudsopgave

1 Inleiding

In dit rapport wordt het plan van aanpak van het afstudeerproject beschreven. Het doel van het plan van aanpak is duidelijk afspraken vastleggen over de inhoud en de omvang van de op te leveren producten. De opdracht wordt in zijn geheel globaal beschreven. Het plan van aanpak zal als richtlijn worden aangehouden gedurende het project.

2 Opdrachtomschrijving

Dit hoofdstuk wordt gebruikt om een beter beeld van de opdracht te krijgen. In dit hoofdstuk vindt u o.a. informatie over de organisatie, probleemstelling en de doestelling.

2.1 Organisatie

De afstudeeropdracht wordt uitgevoerd bij Interpulse Automatisering B.V. Onderstaand zal ik in het kort wat vertellen over het bedrijf.

Interpulse Automatisering B.V. is een full-service internet en automatiseringsbedrijf. Het bedrijf is begonnen als een ICT bedrijf met 7 werknemers in 2000. 14 jaar later telt Interpulse 19 werknemers en blijft het aantal klanten nog steeds groeien. Interpulse is gevestigd in het centrum van Leiden.

Interpulse biedt de volgende diensten aan:

- Systeembeheer
- Systeemontwikkeling
- Internetdiensten
- Consultancy

Deze diensten worden geleverd namens de afdelingen: directie, management, beheer en ontwikkeling. Er is 1 algemeen directeur, met daaronder het managementteam. Daaronder vallen de 2 werkvelden Ontwikkeling en Beheer. De afstudeeropdracht wordt uitgevoerd binnen de afdeling ontwikkeling.

2.2 Opdrachtgever

De opdrachtgever van de opdracht is Marcel van Elsacker. Marcel is hoofd manager ICT van Interpulse. De opdracht wordt dus uitgevoerd voor een interne opdrachtgever.

2.3 Probleemstelling

Het samenstellen van hardware offertes, waarbij de prijzen dagelijks wijzingen is een tijdrovende klus. Er is geen compleet beeld van alle prijzen. Bestellingen plaatsen bij leveranciers moet voldoen aan bepaalde regels, zoals minimum orderbedrag. Het verzamelen van meerdere orders is op dit moment niet goed mogelijk een foutgevoelig.

In de huidige situatie bestaat het samenstellen van hardware offertes voornamelijk uit handwerk. Producten worden bij verschillende leveranciers vergeleken op de kosten en of deze op voorraad is. Het opstellen van deze offertes wordt meerdere keren per week gedaan.

Het is de bedoeling dat er een tool wordt ontwikkeld die producten kan vergelijken bij de groothandelaars (soort van tweakers pricewatch). Je wilt bijvoorbeeld van een bepaalde I-7

processor weten waar deze het goedkoopst verkrijgbaar is of waar je deze het snelst geleverd kan krijgen. Door het automatiseren van dit proces wordt er veel tijd bespaard. Daarnaast is de het de bedoeling dat deze tool in de toekomst ook offertes kan maken aan de hand van gekozen artikelen bij de groothandelaars. Een verkoper kan dan bijvoorbeeld bij de klant gelijk een offerte opmaken, dan weet de klant precies wat een bepaald systeem kost. Dus naast de tijd die het zal gaan besparen kan het ook extra inkomsten gaan opleveren.

2.4 Doestelling

De doestelling van het project is het aanzienlijk terugbrengen van de benodigde tijd voor het samenstellen van hardware offertes.

2.5 Op te leveren producten

Tijdens het project zullen verschillende mijlpaal producten moeten worden opgeleverd. Onderstaand een lijst met op te leveren mijlpaalproducten:

- Onderzoek naar software ontwikkelmethoden
- Inception rapport
    - Plan van aanpak
    - Business modeling
    - Use case diagram
    - Use case beschrijvingen
    - Storyboard
- Elaboration rapport
    - Analyse leveranciers
    - Klassendiagram
    - Sequentie diagrammen
    - Activiteit diagrammen
    - Database ontwerp (ERD)
- Construction rapport
    - Testontwerpen
    - Testuitwerkingen
- Opleveren product
    - Database
    - Sourcecode
    - De web applicatie

2.6 Afbakening

Om ervoor te zorgen dat de scope van de opdracht niet groter wordt dan in eerste instantie de bedoeling was ga ik de opdracht afbakenen. De software die uiteindelijk gemaakt gaat worden kan opgedeeld worden in een aantal modules. De modules die binnen de scope vallen zijn:

- Koppeling maken met leveranciers
    - Er zijn 4 primaire en 2 secundaire leveranciers. Er moeten in totaal 6 koppelingen komen dus.
- Zoek ,filter en sorteer module
- Beheer module voor leveranciers
- Email alert met promo prijzen
- Order module

De volgende modules zouden een meerwaarde voor de applicatie zijn maar vallen buiten de scope:

- Direct bestellen bij de leverancier via de offerte tool
- Integreren in een offerte document (bijvoorbeeld word)
- Koppeling maken met B2B webshop van interpulse
- Prijs geschiedenis van producten

2.7 Randvoorwaarden

Dit project zal worden gerealiseerd in het kader van het afstuderen. De focus van het project ligt dan ook op het proces. Dit betekend dat er op een verantwoorde manier gebruik gemaakt wordt van methoden en technieken. Het project zal worden uitgevoerd binnen een periode van 17 weken, met als startdatum 1 september 2014 en als einddatum 9 januari 2014.

2.8 Risicofactoren

Aan elk project zitten risico's verbonden, dit is bij mijn afstudeeropdracht niet anders. Om ervoor te zorgen dat deze mogelijke risico's het project niet belemmeren worden er voorzorgsmaatregelen genomen. In de onderstaande tabel vind u de risico's en preventies:

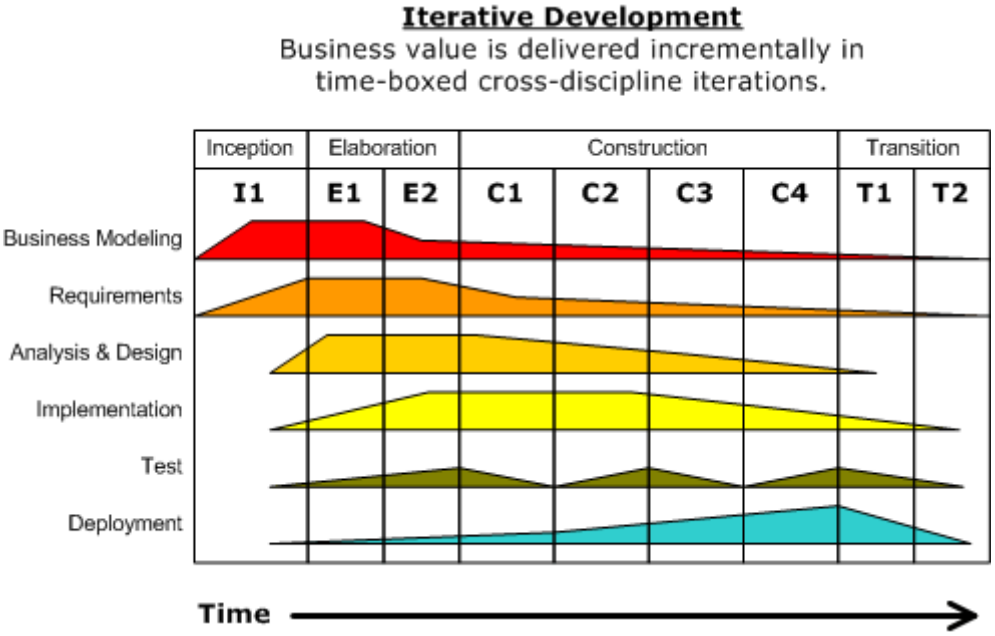
Risico	**Preventies**
Documenten raken beschadigd of gaan verloren.	Alle documentatie wordt online opgeslagen in dropbox. De source code van het project wordt opgeslagen met behulp van GIT. Hierdoor zijn alle bestanden geback-upt in de Cloud en zouden er dus geen bestanden verloren kunnen gaan.
Onvoldoende kennis	Bij onvoldoende kennis wordt in eerste instantie geprobeerd om de desbetreffende informatie te vinden in beschikbare boeken of op internet. Mocht dit niet voldoende zijn wordt er hulp gevraagd aan één van de medewerkers.
Ziekte	Bij het geval van ziekte kan er voor gekozen worden om thuis verder te werken. Bij langdurige ziekte wordt in overleg met de bedrijfsmentor en begeleider vanuit school besproken hoe we nu verder moeten gaan.

3 Aanpak

In dit hoofdstuk wordt de opdracht verder besproken. Er wordt besproken welke methoden en technieken in het project worden gebruikt. Daarnaast worden de uit te voeren activiteiten besproken en is er een planning.

3.1 Methoden en technieken

Voor het project wordt gebruikt gemaakt van de ontwikkelmethode RUP. RUP bestaat uit vier fases namelijk inception, elaboration, construction en transition. Ik heb voor RUP gekozen omdat ik met RUP een duidelijk structuur in mijn project krijg. Ik kan duidelijk zien hoe ik ervoor sta qua planning. Verder kan ik iteratief werken, dit geeft mij de mogelijkheid om telkens een module van de software te ontwerpen en te implementeren. Elke iteratie moet een stuk werkende software opleveren. Dit geeft mij de mogelijkheid om feedback van de opdrachtgever te krijgen. De feedback kan zorgen voor veranderingen in bijvoorbeeld de requirements, dit kan ik vervolgens meenemen in de volgende iteratie.



Voor het opstellen van de ontwerp modellen wordt gebruik gemaakt van de modelleringstechniek UML.

De software wordt ontwikkeld in de laatste versie van asp.net MVC.

Voor de versiebeheer van de sourcecode wordt gebruikt er gemaakt van het versiebeheersysteem GIT.

Verder wordt er tijdens het project nog gebruik gemaakt van rapportagetechnieken en interviewtechnieken.

3.2 Activiteiten

Onderstaand geef ik per fase aan welke activiteiten er worden uitgevoerd. De activiteiten worden verdeeld onder de vier fases van RUP.

Inception

- Plan van aanpak

- Analyseren huidige en gewenste situatie
- Requirements opstellen en prioriteren
- Use case diagram en beschrijvingen
- Storyboard
- Document inception rapport

Elaboration

- Klassendiagram
- Sequentie diagrammen
- Activiteit diagrammen
- Analyseren koppelingen leveranciers
- Database model (ERD)
- Elaboration rapport

Construction

- Bouwen software
- Opzetten database
- Unit test maken voor software
- Construction rapport

Transition

- Gebruikerstest
- Overdragen software

3.3 Planning

Het project heeft als startdatum 1 september 2014 en de einddatum is 9 januari 2015. Er staan in totaal 17 weken geplant voor het project, 3 weken hiervan zijn gereserveerd voor het afstudeerverslag. Dit betekend dat er in totaal 14 weken voltijd aan het project gewerkt kan worden.

Het project wordt opgedeeld in twee grote iteraties. De eerste iteratie zal bestaan uit de basis, de basis van het systeem zal hier worden gelegd. De tweede iteraties bestaat vooral uit uitbreidingen, zonder deze uitbreidingen is het systeem nog steeds bruikbaar, alleen in mindere mate.

Onderstaand vind u een gefaseerde planning die het gehele project beschrijft. Naast het gefaseerde projectplan wordt er voor elke iteratie nog een gedetailleerd plan opgesteld.

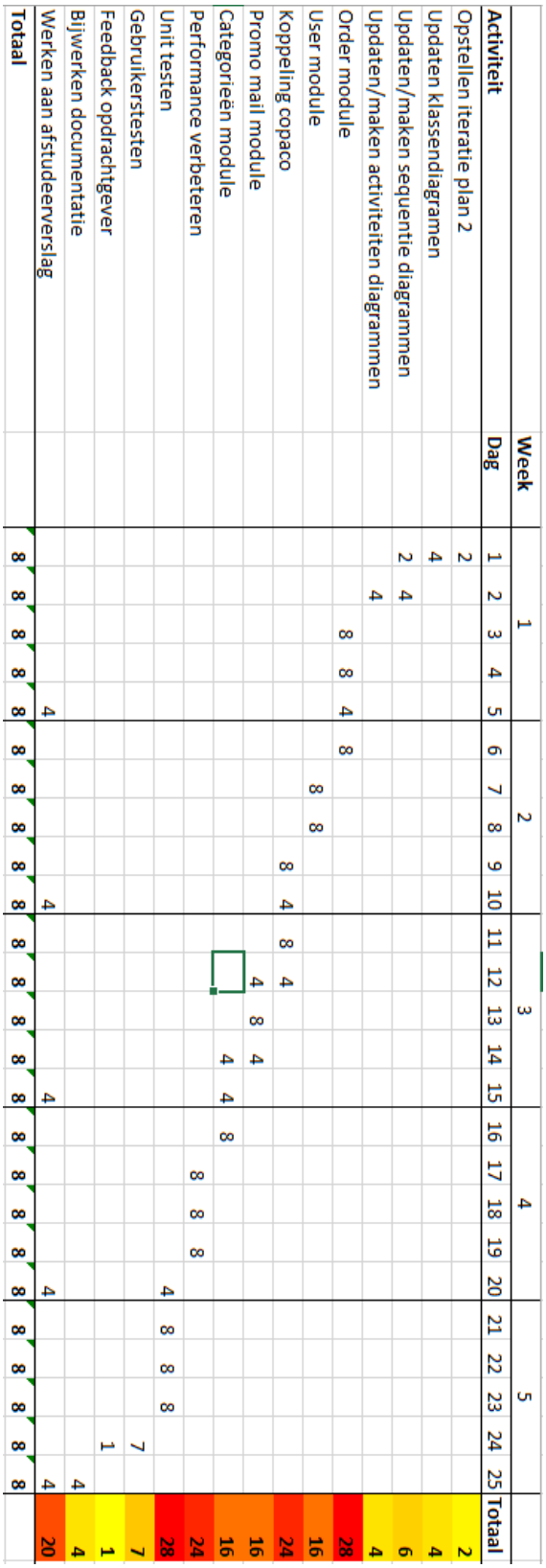
		November																									December																			
		9					10					11					12					13					14					15					16					17				
	Activiteit	27	28	29	30	31	3	4	5	6	7	10	11	12	13	14	17	18	19	20	21	24	25	26	27	28	1	2	3	4	5	8	9	10	11	12	15	16	17	18	19	22	23	24	25	26
	Onderzoek SOM																																													
Iteratie 1	Plan van aanpak																																													
	Interview opdrachtgever																																													
	Business analyse																																													
	Requirements opstellen																																													
	Requirements prioriteren																																													
	Use case diagram																																													
	Use case beschrijvingen																																													
	Storyboard																																													
	Inception rapport																																													
	Opstellen iteratie plan																																													
	Analyseren koppeling mogelijkheden leveranciers																																													
	Ontwerpen klassendiagram																																													
	Ontwerpen sequentie diagrammen																																													
	Ontwerpen activiteit diagrammen																																													
	Ontwerpen database model																																													
	Elaboration rapport																																														
	Opstarten Visual studio project																																													
	Configureren bootstrap																																													
	Opzetten database																																													
	Bouwen koppelingen en overzicht producten	X	X	X	X	X	X	X	X																																					
	Unit testen uitschrijven									X	X																																			
	Feedback opdrachtgever											X																																		
	Construction rapport												X																																	
Iteratie 2	Opstellen iteratie plan													X																																
	Klassendiagram uitbreiden														X	X																														
	Sequentie diagrammen															X	X																													
	Database ontwerp uitbreiden																	X																												
	Bouwen uitbreidingen iteratie 2																		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X													
	Unit testen uitschrijven																																	X	X											
	Overige testen																																			X	X									
	Feedback opdrachtgever																																					X								
	Bijwerken documentatie																																						X							
	Oplevering software																																							X	X					
	Werken aan afstudeerverslag					X					X					X					X					X					X					X						X	X	X	X	X

		Maand	September																						Oktober																			
		Projectweek	1					2					3					4					5			6					7					8								
	Activiteit	Dag	1	2	3	4	5	8	9	10	11	12	15	16	17	18	19	22	23	24	25	26	29	30	1	2	3	6	7	8	9	10	13	14	15	16	17	20	21	22	23	24
	Onderzoek SOM		X	X	X																																							
Iteratie 1	Plan van aanpak				X	X	X																																					
	Interview opdrachtgever						X																																					
	Business analyse						X	X	X																																			
	Requirements opstellen									X																																		
	Requirements prioriteren									X																																		
	Use case diagram										X	X																																
	Use case beschrijvingen											X	X																															
	Storyboard												X																															
	Inception rapport													X																														
	Opstellen iteratie plan														X																													
	Analyseren koppeling mogelijkheden leveranciers														X	X	X																											
	Ontwerpen klassendiagram																X	X	X																									
	Ontwerpen sequentie diagrammen																			X	X																							
	Ontwerpen activiteit diagrammen																					X																						
	Ontwerpen database model																						X	X																				
	Elaboration rapport																								X																			
	Opstarten Visual studio project																									X																		
	Configureren bootstrap																										X																	
	Opzetten database																											X																
	Bouwen koppelingen en overzicht producten																												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Unit testen uitschrijven																																											
	Feedback opdrachtgever																																											
	Construction rapport																																											
Iteratie 2	Opstellen iteratie plan																																											
	Klassendiagram uitbreiden																																											
	Sequentie diagrammen																																											
	Database ontwerp uitbreiden																																											
	Bouwen uitbreidingen iteratie 2																																											
	Unit testen uitschrijven																																											
	Overige testen																																											
	Feedback opdrachtgever																																											
	Bijwerken documentatie																																											
	Oplevering software																																											
	Werken aan afstudeerverslag						X					X					X					X				X					X					X							X

Onderstaan de planning voor de eerste iteratie:

	Week	1					2					3					4					5					6					7					8					9					10					
Activiteit	**Dag**	1	2	3	4	5	6	7	8	9	10	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	**Totaal**
Plan van aanpak		8	8																																																**16**
Interview opdrachtgever						1																																													**1**
Business analyse				8	8	5																																													**21**
Requirements opstellen							8	4																																											**12**
Requirements prioriteren								4																																											**4**
Use case diagram									6																																										**6**
Use case beschrijvingen									2	4																																									**6**
Storyboard										4																																									**4**
Inception rapport											4																																								**4**
Opstellen iteratie plan												2																																								**2**
Analyseren koppelingen leveranciers												6	8	8																																						**22**
Analyse SOAP of REST															8																																					**8**
Analysis klassendiagram																4																																				**4**
Design klassendiagram																	8																																			**8**
Sequentie diagrammen																		8	8																																	**16**
Activiteiten diagrammen																				8																																**8**
Ontwerpen database model																					4	4																														**8**
Uitwerken elaboration rapport																						4																														**4**
Opstarten Visual studio project																							4																													**4**
Configureren bootstrap																							4																													**4**
Database bouwen																								8																												**8**
Bouwen eerste opzet classes																									8																										**8**
Koppelingen module																										4	8	4																								**16**
Koppeling Techdata																												4	8	8	4																				**24**
Koppeling Aces Direct																																8	8	8																	**24**
Zoeken en filter module																																			8	4	8	4												**24**
Select 2 dropdowns																																					4														**4**
Leveranciers module																																						4	4												**8**
Categorieën module																																								4	8	8	8								**28**
Prijs en voorraad geschiedenis																																											8							**8**
Performance																																												4						**4**
Unit tests																																													8	8	8	8		**32**
Rapportage iteratie 1																																																3	**3**
Feedback iteratie 1																																																1	**1**
Werken aan afstudeerverslag						2					4					4					4					4					4					4					4					4				4	**38**
Totaal		8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	4	4	8	8	8	8	8	8	8	8	8	8	8	

Onderstaande planning voor de tweede iteratie:



	Week	1					2					3					4					5					
Activiteit	**Dag**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	**Totaal**
Opstellen iteratie plan 2		2																									2
Updaten klassendiagramen		4																									4
Updaten/maken sequentie diagrammen		2	4																								6
Updaten/maken activiteiten diagrammen			4																								4
Order module				8	8	4	8																				28
User module								8	8																		16
Koppeling copaco										8	4	8	4														24
Promo mail module													4	8	4												16
Categorieën module															4	4	8										16
Performance verbeteren																		8	8	8							24
Unit testen																					4	8	8	8			28
Gebruikerstesten																									7		7
Feedback opdrachtgever																									1		1
Bijwerken documentatie																										4	4
Werken aan afstudeerverslag						4				4						4					4					4	20
Totaal		8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	

4 Projectinrichting

In dit hoofdstuk wordt de projectinrichting besproken. De projectinrichting is opgebouwd uit de onderdelen projectorganisatie en de benodigde mensen en middelen.

4.1 Projectorganisatie

De projectgroep bestaat uit één persoon, dit is Donny Roelen. Het project wordt dus alleen uitgevoerd. Daarnaast is de opdrachtgever nog onderdeel van de projectorganisatie, de opdrachtgever wordt aan het begin van het proces geraadpleegd en aan het eind van elke iteratie. Het project wordt uitgevoerd bij Interpulse op de afdeling ontwikkeling.

4.2 Benodigde mensen en middelen

In dit kopje beschrijf ik de benodigde mensen en middelen om het project succesvol uit te kunnen voeren.

4.2.1 Mensen

Het is belangrijk dat ik als opdrachtnemer regelmatig feedback krijg van de opdrachtgever. In de begin fase van het project heb ik de opdrachtgever nodig om de wensen en eisen op te stellen. Daarnaast heb ik de opdrachtgever nodig om feedback te geven op de resultaten van een uitgevoerde iteratie.

4.2.2 Hardware

De werkzaamheden voor het project worden uitgevoerd op de door Interpulse beschikbaar gestelde computer. De werkplek voldoet om alle werkzaamheden op uit te voeren

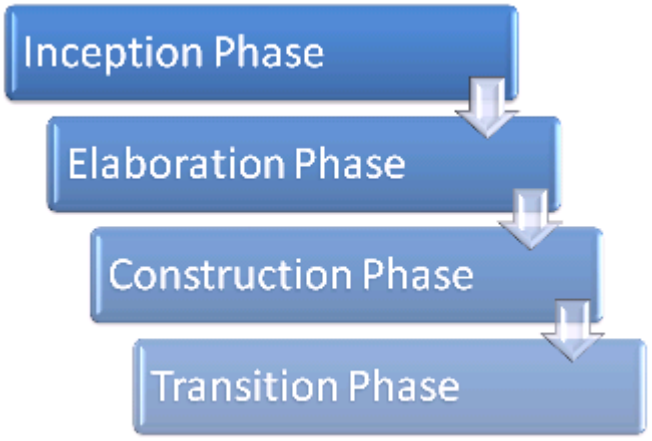
4.2.3 Software

Om het project succesvol uit te kunnen voeren wordt er gebruik gemaakt van de volgende softwarepakketten:

- Microsoft office 2013
- Visio studio 2013
- Dropbox
- GIT
- Microsoft SQL server 2012

Inception Rapport

Hardware offerte tool



Auteur: Donny Roelen

Datum: 5 januari 2015

Versie: 1.1

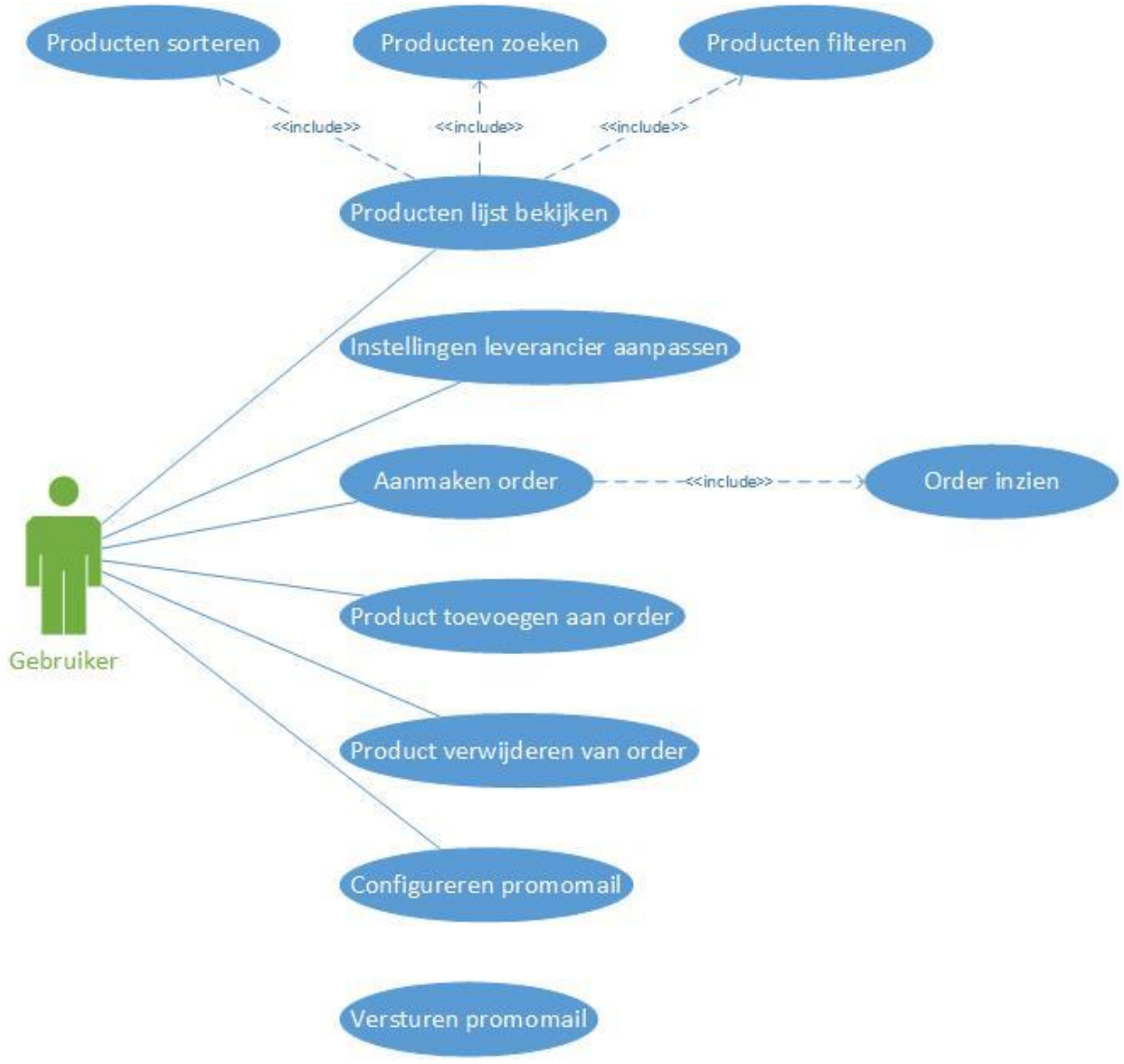
Inhoudsopgave

1 Inleiding

In dit hoofdstuk worden de artefacten die tijdens de inception fase aan bod komen besproken. Dit document bestaat uit de use cases, de beschrijvingen, een storyboard, de requirements en een eerste opzet van het analysis klassendiagram.

2 Use case diagram

In dit hoofdstuk wordt de use case diagram van ons systeem besproken. Met behulp van de use case diagram worden de wensen en eisen van het te bouwen systeem in kaart gebracht. In figuur 1 vindt u mijn use case diagram.



Figuur 1: Use case diagram

3 Use case beschrijvingen

In dit hoofdstuk worden de use cases die we eerder hebben opgesteld verder besproken. Elke case wordt uitgewerkt en beschreven in een use case beschrijving.

3.1 Product lijst bekijken

Naam	Producten lijst bekijken
ID	Products_view.1
Samenvatting	Het overzicht van producten bekijken
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Geen
Normale flow	1. De gebruiker opent het producten overzicht. 2. De producten worden weergegeven in een tabel.
Postconditie	Geen
Alternatieve flow	Geen

3.2 Producten zoeken

Naam	Producten zoeken
ID	Product_search.1
Samenvatting	Het zoeken van een product in het overzicht van producten.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Producten lijst wordt weergegeven.
Normale flow	1. De gebruiker voert een zoekterm. 2. De gebruiker drukt op de knop zoeken. 3. De gevonden resultaten worden getoond.
Postconditie	Geen
Alternatieve flow	1. De gebruiker voert een zoekterm in. 2. De gebruiker drukt op de zoeken. 3. De ingevoerde zoekterm levert geen resultaten op. 4. Melding “geen producten gevonden”.

3.3 Producten filteren

Naam	Producten filteren
ID	Product_filter.1
Samenvatting	Het instellen van de filter om enkel bepaalde producten te tonen.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Producten lijst wordt weergegeven.
Normale flow	1. De gebruiker stelt de filter in. 2. De producten lijst wordt bijgewerkt en de resultaten worden weergegeven.
Postconditie	Geen
Alternatieve flow	1. De gebruiker stelt de filter in. 2. De ingestelde filter opties leveren geen resultaten op.

3.4 Producten sorteren

Naam	Producten sorteren
ID	Product_sort.1
Samenvatting	Het sorteren van producten op een attribuut waarde zodat deze in de gekozen volgorde worden weergegeven.

Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Producten lijst wordt weergegeven.
Normale flow	1. De gebruiker kiest ervoor om een attribuut oplopend of aflopend te sorteren. 2. De resultaten worden bijgewerkt aan de hand van de geselecteerde sorteer optie.
Postconditie	Geen
Alternatieve flow	Geen

3.5 Product toevoegen aan order

Naam	Product toevoegen aan order
ID	Product_add_to_order.1
Samenvatting	Een product toevoegen aan een order.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	1. De producten lijst wordt weergegeven. 2. Er is een order aangemaakt om producten aan toe te voegen.
Normale flow	1. De gebruiker selecteert een product. 2. De gebruiker drukt op de knop “product toevoegen”. 3. Product wordt toegevoegd aan de order.
Postconditie	Order wordt bijgewerkt aan de hand van het geselecteerde product.
Alternatieve flow	1. De gebruiker selecteert een product. 2. De gebruiker drukt op de knop “product toevoegen”. 3. Het systeem geeft een melding dat het geselecteerde product niet op voorraad is. 4. Het geselecteerde product wordt niet toegevoegd aan de order.

3.6 Product verwijderen van order

Naam	Product verwijderen van order
ID	Product_del_from_order.1
Samenvatting	Een product van een order verwijderen.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Er is een order aangemaakt
Normale flow	1. De gebruiker selecteert de order die hij wilt aanpassen. 2. De gebruiker selecteert het product die hij wilt verwijderen. 3. De gebruiker selecteert de knop “product verwijderen”. 4. Product wordt verwijdert van de order.
Postconditie	Order wordt bijgewerkt.
Alternatieve flow	Geen

3.7 Order aanmaken

Naam	Order aanmaken
ID	Order_create.1
Samenvatting	Het aanmaken van een order zodat hier producten aan toegevoegd kunnen worden.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Geen
Normale flow	1. De gebruiker gaat naar het order overzicht. 2. De gebruiker kiest ervoor om een nieuwe order aan te maken.

	3. Nieuwe order is aangemaakt.
Postconditie	De gebruiker kan nu producten toevoegen aan de aangemaakte order.
Alternatieve flow	Geen

3.8 Order inzien

Naam	Order inzien
ID	Order_view.1
Samenvatting	Het inzien van een eerder aangemaakte order.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Er is een order aangemaakt
Normale flow	1. De gebruiker navigeert naar het order overzicht. 2. De gebruiker selecteerde een order. 3. De geselecteerde order wordt weergegeven.
Postconditie	Geen
Alternatieve flow	Geen

3.9 Promomail versturen

Naam	Promomail vesturen
ID	Promo_send.1
Samenvatting	Het versturen van een email naar de gebruiker(s) die dit ingesteld hebben.
Primaire actors	Systeem
Secundaire actors	Gebruiker
Pre condities	De gebruiker heeft aangegeven promomail te willen ontvangen voor de geselecteerde categorieën.
Normale flow	1. Het systeem werkt de producten en prijzen bij. 2. Het systeem “ziet” dat er promo prijzen zijn voor bepaalde producten. 3. De promo producten worden toegevoegd aan een email. 4. De email wordt verzonden naar gebruiker(s).
Postconditie	Geen
Alternatieve flow	1. Het systeem werkt de producten en prijzen bij. 2. Er zijn geen promo prijzen gevonden. 3. Er wordt geen promomail verzonden.

3.10 Configureren promomail

Naam	Promomail configureren
ID	Promomail_config.1
Samenvatting	Het configureren van de promomail voor de gebruiker zodat deze wel of niet promomailtjes ontvangt van een bepaalde categorie.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Geen
Normale flow	1. De gebruiker selecteert de optie “configureren promomail”. 2. De gebruiker selecteert de categorieën waarvan hij/zij promomails wilt ontvangen. 3. De gebruiker kiest voor opslaan.
Postconditie	De wijzigingen worden opgeslagen en de gebruiker krijgt voortaan promomails voor de door hem geselecteerde categorieën.
Alternatieve flow	Geen

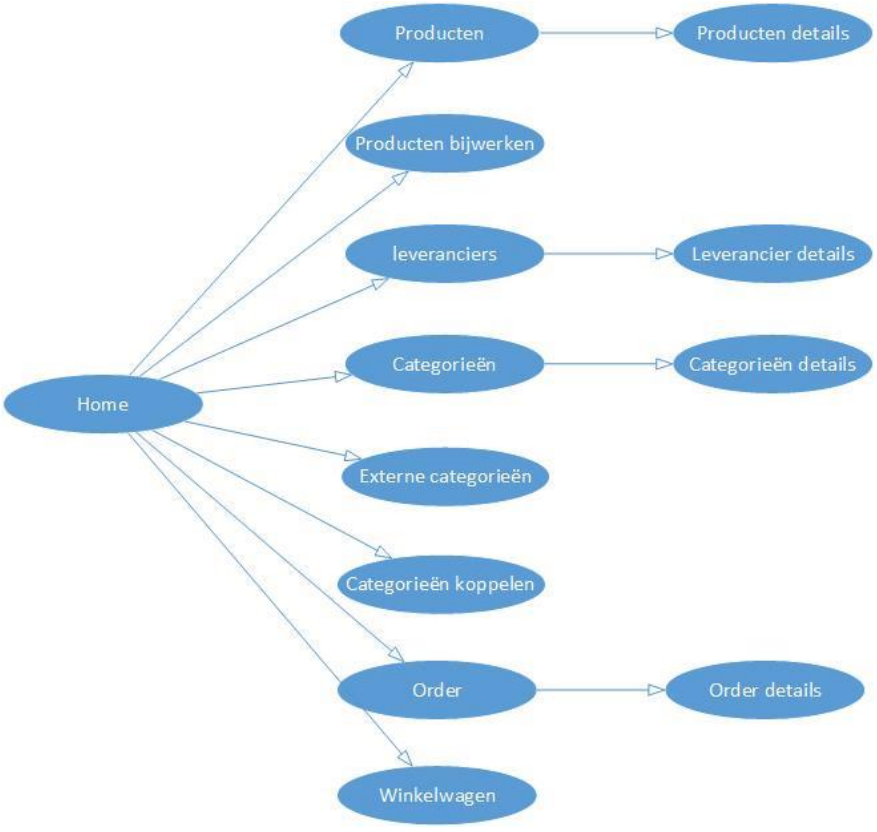
3.11 Leverancier instellingen aanpassen

Naam	Leverancier instellingen aanpassen

ID	Supplier_edit.1
Samenvatting	Het aanpassen van de voorkeursinstellingen voor de leveranciers.
Primaire actors	Gebruiker
Secundaire actors	Geen
Pre condities	Geen
Normale flow	1. De gebruiker gaat naar de beheer module voor leveranciers. 2. De gebruiker selecteert de leverancier waarvoor hij aanpassingen voor wilt maken. 3. De gebruiker maakt wijzingen aan de leverancier. 4. De gebruiker kiest ervoor om de wijzigingen op te slaan.
Postconditie	De wijzingen worden opgeslagen en de gebruiker keert terug naar het hoofdscherm.
Alternatieve flow	Geen

4 Storyboard

In dit hoofdstuk wordt de storyboard behandeld, de storyboard wordt gebruikt om interacties tussen de interfaces van het systeem weer te gegeven. In figuur 2 vindt u een uitwerkingen van de storyboard.



Figuur 2: Storyboard

5 Requirements

De requirements zijn opgesteld aan de hand van de eerste wensen en eisen die aan bod kwamen tijdens het eerste interview voor het afstudeerplan. In een vervolg gesprek in eind week 1 zijn de wensen en eisen geverifieerd en zijn er weer nieuwe wensen en eisen boven tafel gekomen.

De requirements worden verdeelt in groepen volgens het FURPS+ model, dit model is ontwikkeld door Hewlett-Packard. FURPS+ verdeelt de requirements in de volgende groepen:

- Functionality - funtioneel
- Usability - gebruikbaarheid
- Reliability - betrouwbaarheid
- Performance - werking
- Supportability - onderhoudbaarheid

De plus staat voor enkele overige wensen en eisen die de gebruiker kan hebben:

- Ontwerp – hoe wordt de software gebouwd
- Implementatie – moeten de programmeurs bepaalde standaarden aanhouden
- Interface – hoe ziet de applicatie eruit
- Fysieke – op wat voor hardware voor het systeem draaien

Daarnaast worden de requirements geprioriteerd aan de hand van MoSCoW.

- Must have – moet in het eindresultaat komen, zonder is het product niet bruikbaar
- Should have – is zeer gewenst, maar zonder is het product bruikbaar
- Could have – komen aanbod als er tijd genoeg is
- Won't have – komen in dit project niet aanbod, maar misschien wel in een vervolgproject

Onderstaand vindt u de requirements die op dit moment zijn opgesteld voor het systeem. Deze lijst kan in de loop van het project wijzigen door nieuwe wensen en eisen van de opdrachtgever met betrekking tot het systeem.

Code	**Omschrijving**	**Requirements volgens FURPS+**	**Prioriteit met MoSCoW**
F-07	Het systeem moet producten ophalen van leveranciers	Funtioneel	Must
F-08	De primaire leveranciers moeten worden gekoppeld aan het systeem	Funtioneel	Must
F-09	Er moet een overzicht komen met alle producten	Funtioneel	Must
F-10	Producten moeten gefilterd kunnen worden op leverancier en categorie	Funtioneel	Must
F-11	Producten moeten gezocht kunnen worden op artikelnummer, fabrikantnummer, naam en omschrijving	Funtioneel	Must
F-12	Producten moeten gesorteerd kunnen worden op prijs en voorraad	Funtioneel	Must

F-13	De prijzen moeten iedere dag worden bijgewerkt	Funtioneel	Must
F-14	De productenlijst moet iedere week worden bijgewerkt	Funtioneel	Must
F-15	Wanneer een product is verwijderd moet deze nog wel blijven bestaan	Funtioneel	Must
IMLP-01	De software wordt ontwikkel op het asp.net framework	Implementatie	Must
INT-02	De productenlijst moet overzichtelijk zijn	Interface	Must
O-01	Er wordt gebruikt gemaakt van visual studio 2013 als ontwikkel tool	Ontwerp	Must
O-02	Het DBMS is microsoft SQL server 2012	Ontwerp	Must
F-16	De secundaire leveranciers moeten worden gekoppeld aan het systeem	Funtioneel	Should
F-17	Het moet mogelijk zijn om een minimum order bedrag in te voeren bij een leverancier	Funtioneel	Should
F-18	Producten worden verdeeld in categorieën	Funtioneel	Should
F-19	De gebruiker moet op de hoogte worden gesteld van promo prijzen	Funtioneel	Should
F-20	Het moet mogelijk zijn om in te stellen waarvan je promo prijzen wilt ontvangen	Funtioneel	Should
F-21	Er moet een overzicht komen voor de orderkosten	Funtioneel	Should
F-22	Eerder aangemaakt order moeten later nog ingezien kunnen worden	Funtioneel	Should
F-23	Een product moet kunnen worden toegevoegd aan een winkelwagen	Funtioneel	Should
F-24	Een product moet kunnen worden verwijderd van een winkelwagen	Funtioneel	Should
F-25	Er moet een order aangemaakt kunnen worden van een winkelwagen	Funtioneel	Should
F-26	Zelf kunnen kiezen van welke categorieën producten worden geïmporteerd	Funtioneel	Should
F-27	Weergeven wanneer een product het laatst is bijgewerkt	Funtioneel	Should
F-28	Aangeven op de prijs van een product is gestegen of gedaald	Funtioneel	Should
F-29	Een filter toevoegen voor promotieprijzen	Funtioneel	Should
F-30	Bij de prijs in het overzicht weergeven of het om een normale of promotie prijs gaat	Funtioneel	Should

F-31	Het moet mogelijk zijn om meerdere winkelmandjes aan te maken	Funtioneel	Should
F-32	Een winkelmandje moet je een naam kunnen geven	Funtioneel	Should
F-33	Een winkelwagen moet worden opgeslagen om op een later tijdstip weer te gebruiken	Funtioneel	Should
F-34	Een winkelwagen moet kunnen worden verwijderd	Funtioneel	Should
INT-03	Verwijderde producten moeten in het rood worden weergegeven	Interface	Should
P-01	Een zoekresultaat moet binnen 5 seconden worden getoond	Performance	Should
F-01	Er moet een prijsgeschiedenis voor producten komen	Funtioneel	Could
F-02	Het moet mogelijk zijn om bestellingen te maken bij de leverancier met de offerte tool	Funtioneel	Could
F-03	Het moet mogelijk worden een offerte op te maken	Funtioneel	Could
F-04	Gemiddelde prijs weergeven in de prijsgeschiedenis grafiek	Funtioneel	Could
F-05	3 maanden geschiedenis bijhouden van een product	Funtioneel	Could
F-06	Er moet een overzicht komen met meest bestelde producten (favorieten)	Funtioneel	Could
INT-01	De prijsgeschiedenis wordt weergegeven in een grafiek	Interface	Could
F-35	Er moet een koppeling komen met de B2B webshop van interpulse	Funtioneel	Won't

6 Analysis klassendiagram

In dit hoofdstuk wordt het analysis klassendiagram besproken. De analysis klassendiagram dient als eerste opzet van het klassendiagram. Deze zal verder worden uitgewerkt in de elabaration fase.

6.1 Product

De klassen product bevat alle informatie over een product. Een product hoort altijd bij 1 categorie. Een product hoort ook altijd bij 1 leverancier. Verder kan een product bij 0 of 1 orders horen.

6.2 Order

De klassen order bevat alle informatie met betrekking tot één order. Een order hoort altijd bij 1 user. Verder kan een order meerdere producten bevatten. Een order kan producten van meerdere leveranciers bevatten.

6.3 Leverancier

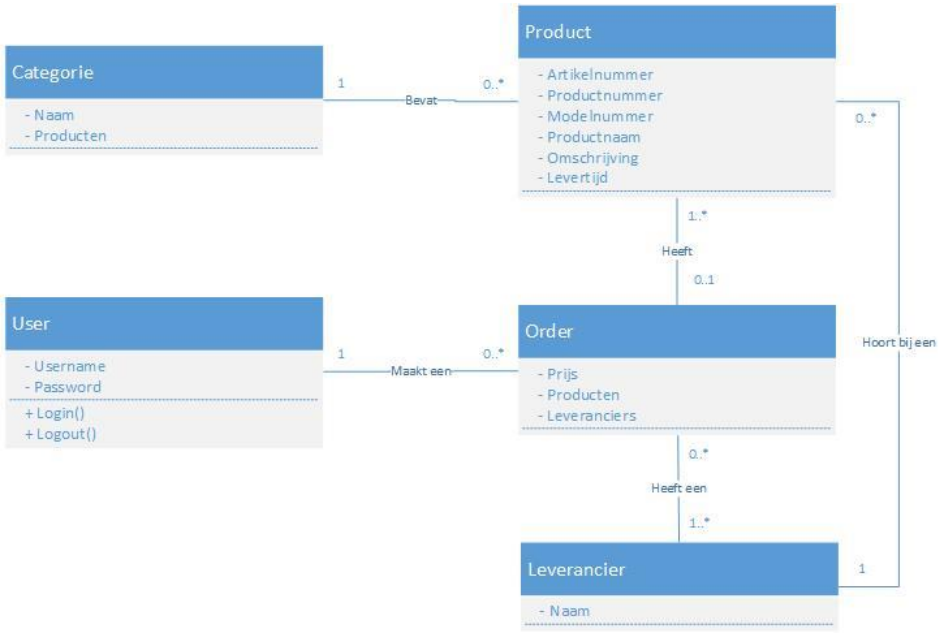
De klassen leverancier bevat alle informatie over een leverancier. Een leverancier heeft meerdere producten. Daarnaast kan een leverancier meerdere orders tegelijk open hebben staan.

6.4 User

De klassen user bevat informatie van de ingelogde gebruiker. Een user kan meerdere order plaatsen.

6.5 Categorie

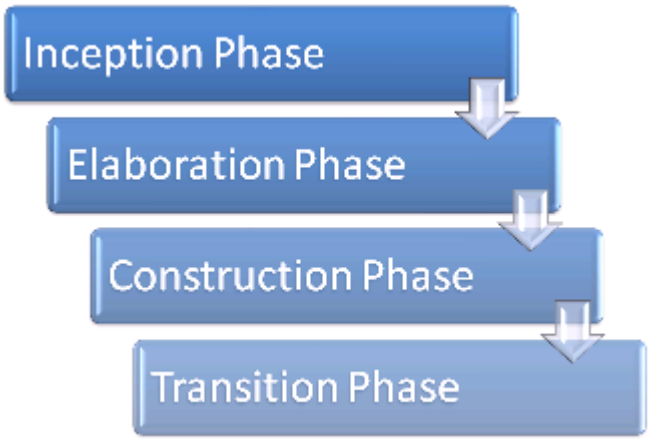
De klassen categorie bevat informatie over een categorie. Een categorie kan meerdere producten bevatten.



Figuur 3: Eerste concept klassendiagram

Elaboration rapport

Hardware offerte tool



Auteur: Donny Roelen

Datum: 5 januari 2015

Versie: 1.2

Inhoudsopgave

1 Inleiding

In dit hoofdstuk worden de artefacten die tijdens de elaboration fase aan bod komen besproken. Dit document bestaat uit een design klassen diagram, sequentie diagrammen, activiteiten diagrammen en een analyse over de leverancier koppelingen en een database ontwerp.

2 Analyse koppelingen leveranciers

In dit hoofdstuk beschrijf ik mijn bevingen over de mogelijkheden die de leveranciers bieden met betrekking tot de koppelingen die gemaakt moeten worden voor het ophalen van product gegevens. Er wordt een onderscheid gemaakt tussen primaire en secundaire leveranciers.

2.1 Primaire Leveranciers

De primaire leveranciers zijn de leveranciers waar de meeste bestellingen worden geplaatst. Met deze leveranciers zal in eerste instantie een koppelingen worden gemaakt. De koppelingen naar de primaire leveranciers is het belangrijkste onderdeel van het systeem dat zal worden gebouwd.

2.1.1 Techdata

Techdata is de eerste leverancier die ik ga analyseren. Daarnaast is Techdata ook de leverancier waar de meeste bestellingen worden geplaatst. Techdata biedt een uitgebreide documentatie over het koppelen aan het systeem. Dit kan doormiddel van XML. Deze documentatie is te vinden op http://www.techdata.com/Content/XMLGuide/Chap1.htm. Techdata biedt een lijst aan mogelijkheden:

- Product beschikbaarheid bij elk warenhuis
- Levertijden
- Price updates
- Promotie prijzen
- Order status en meer

Om gebruik te maken van deze service moet er wel eerst een XML Trading Partner Agreement worden aangevraagd bij het EC Implementations Team. Zodra dit is gedaan moet je bij elke aanvraag de username en password meesturen. Verder is het belangrijk dat je deze username en password dynamisch kan aanpassen, aangezien het wachtwoord naar verloop van tijd vervalt.

2.1.2 Aces Direct

Nog geen informatie kunnen vinden. Wel zie ik dat tweakers.net Aces direct op heeft genomen in de pricewatch, dus een connectie moet mogelijk zijn.

Email: info@acesdirect.nl

2.1.3 Copaco

Copaco biedt ook de mogelijkheid om bepaalde data op te vragen. Copaco geeft de mogelijk om deze bestanden op te sturen in XML of CSV formaat. Het opvragen van de productenlijst kan via de mail of via een FTP verbinding. Bij het aanvragen van de FTP verbinding moet wel eerst een aanvraag gedaan worden bij e-business@copaco.nl. Vervolgens moet je bij de aanvraag keuzes maken uit de onderstaande mogelijkheden:

Opties	Voorbeeld
Versie	Light / Normal / Plus
Heffingen	Prijs incl heffingen / Heffingen in losse velden
Wanneer	Bepaalde dag(en) of iedere dag
Inhoud	Alles, bepaalde merken of alleen telecom
Assortiment	Standaard en / of Selectief [3]
Format	XML of CSV
Verzendwijze	FTP Copaco of uw FTP

Naast de productenlijst biedt Copaco uit de mogelijkheid tot het real time opvragen van product prijzen en product beschikbaarheid. Deze aanvraag kan je doen via een commando via het internet, vervolgens krijg je een XML bestand terug van Copaco.

2.1.4 Xeptor

Nog geen informatie over webservices kunnen vinden. Wel heb ik kunnen vinden dat Xeptor en Infotheek hetzelfde bedrijf zijn.

Email: info@xeptor.nl en info@infotheek.nl

2.2 Secundaire leveranciers

De secundaire leveranciers zijn leveranciers waar af en toe besteld wordt. Deze zijn minder belangrijk dan de primaire leveranciers. De koppelingen met deze leveranciers hebben minder prioriteit van de koppeling van de primaire leveranciers.

2.2.1 Central Point

Nog geen informatie kunnen vinden. Ook Central point is opgenomen in de tweakers pricewatch, dus er moet een mogelijkheid zijn om producten te vergelijken.

Email: info@centralpoint.nl

2.2.2 Travion

Travion biedt aan volume klanten de mogelijkheid om hun e-services te gebruiken. Via een XML koppelingen kunnen de volgende gegevens worden gekoppeld:

- Productprijs
- Voorraadcheck
- Order aanmaken
- XML orderbevestiging
- XML verzendgegevens
- XML pakbonnen
- XML facturen

Om gebruik te maken van deze services moet contact worden opgenomen met de accountmanager. Meer informatie is te vinden op http://travion.nl/?e-catalogue.

2.3 Overzicht beschikbare functionaliteiten leveranciers

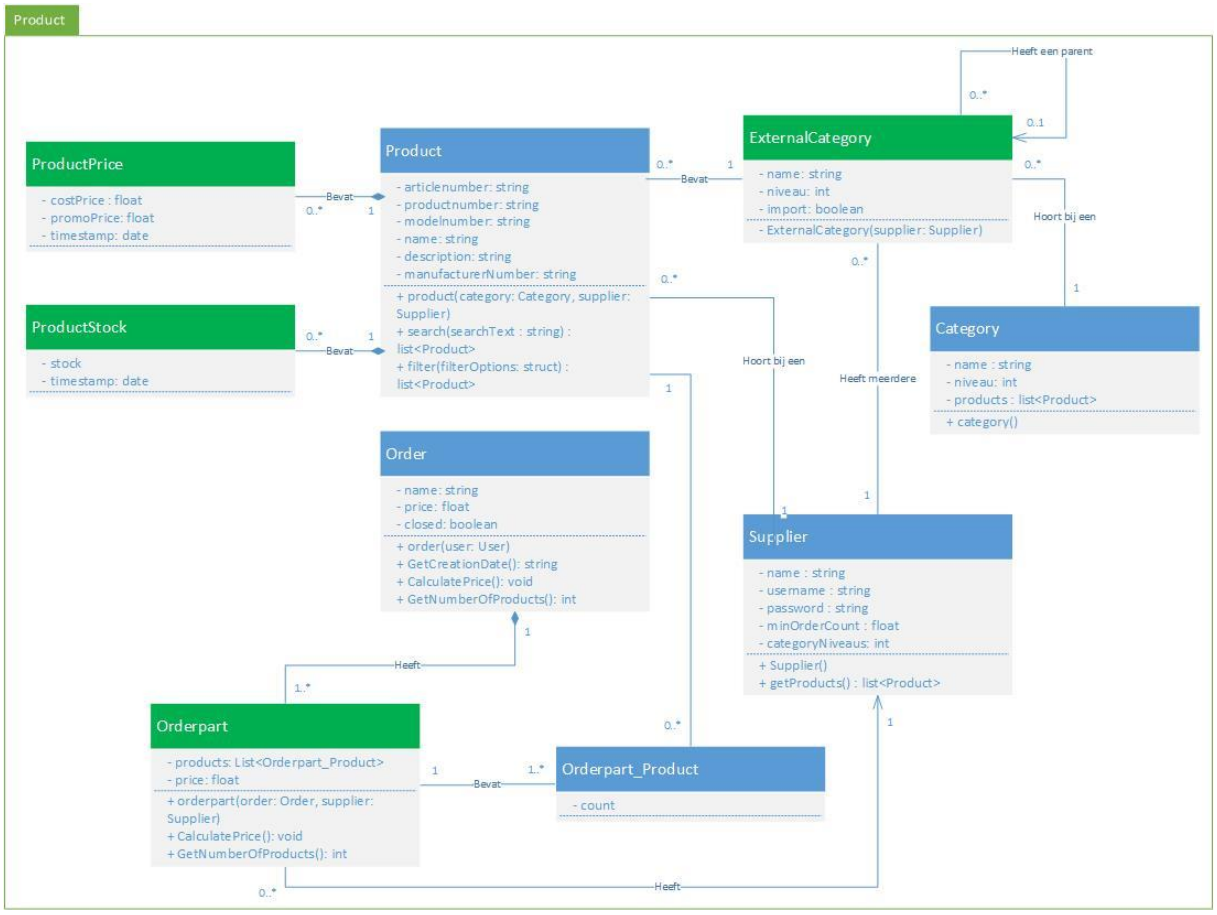
	Techdata	**Aces Direct**	**Copaco**	**Xeptor**	**Central Point**	**Travion**
Producten lijst	Y	?	Y	?	?	Y
Prijzen updates	Y	?	Y	?	?	Y
Levertijd	Y	?	Y	?	?	Y
Aantal producten beschikbaar	Y	?	Y	?	?	Y
Promo prijzen	Y	?	?	?	?	?

3 Design klassendiagram

In dit hoofdstuk wordt het design klassendiagram besproken. Het klassendiagram wordt getoond en de klassen worden verder toegelicht. Het design klassendiagram omschrijft hoe de model classes in elkaar met verbonden zijn. Ik heb in mijn ontwerp onderscheid gemaakt tussen twee packages. De package product en de package user.

3.1 Product

De package Product bevat alle belangrijkste componenten van het te bouwen systeem. In figuur 1 vindt u de uitwerking van de package product. De klassen met de groene kopjes zijn nieuw toegevoegd en de klassen met de blauwe kopjes bestonden al uit het eerste concept klassendiagram.



Figuur 1: Klassendiagram product package

Product

De relatie tussen product en categorie is gewijzigd. Er is namelijk een externe categorie tussen gekomen. Een product heeft een relatie met een externe categorie. Een product hoort altijd bij maar één externe categorie. Verder wordt van een product nu prijs en voorraad geschiedenis bijgehouden. Een product wordt nooit geheel verwijderd uit de database, maar wel wordt er bijgehouden wanneer deze verwijderd is. Dit heb ik zo ontworpen omdat de opdrachtgever verwijderde producten nog wel terug wilt zien in het producten overzicht. Daarnaast zouden de aangemaakte orders worden aangepast als een product geheel wordt verwijderd en dit is niet de bedoeling.

ProductPrice

De klasse productprice bevat prijsinformatie van een Product. Met behulp van deze klasse kan er gemakkelijk een prijsgeschiedenis worden bijgehouden. Er wordt namelijk bijgehouden wanneer de

prijsinformatie is toegevoegd. Door het scheiden van de prijsinformatie van de product hoef ik niet voor elke prijswijziging de bijbehorende Product aan te passen, er wordt enkel een nieuwe record toegevoegd aan de tabel productprice.

ProductStock
De klasse productstock bevat informatie over de voorraad van de voorraad van een Product. Deze klasse werkt in principe hetzelfde als de productprice klasse. We kunnen nu de gemakkelijk een voorraadgeschiedenis gaan bijhouden. De belangrijkste reden voor het scheiden van de voorraad van de Product is zodat we de product niet voor elke voorraad wijziging hoeven aan te passen, er wordt enkel een nieuwe record toegevoegd aan de tabel productstock.

Order
De klasse order is sinds het vorige ontwerp enigszins aangepast. Een order heeft niet langer een relatie met de klassen product. Wel heb ik een nieuwe relatie toegevoegd, namelijk een relatie met de klasse orderpart. Een order bestaat uit 1 of meer orderparts.

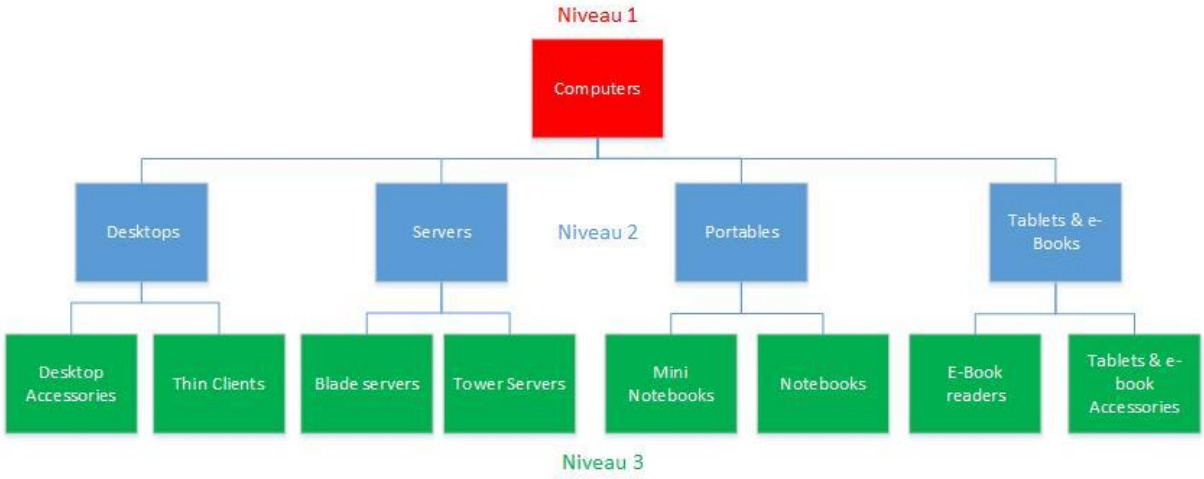
Orderpart
De klasse orderpart bevat alle informatie met betrekking tot een deel van een order. Een orderpart is eigenlijk een order, maar dan maar voor één leverancier. Een orderpart hoort dus altijd bij één leverancier. Verder kan een orderpart één of meer producten bevatten. Een orderpart hoort altijd bij maar één order. Zodra de bijhorende order ophoudt met bestaan heeft het bestaan van de orderpart geen meerwaarde meer.

Orderpart_Product
De klasse orderpart_product is de tussentabel tussen klasse product en orderpart. Deze klasse houdt bij welke producten aan welke orderpart zijn gekoppeld. Daarnaast wordt met de attribuut count bijgehouden hoevaak één product voorkomt in één orderpart.

Category
De klasse categorie is ook aangepast sinds het eerste ontwerp. Een categorie heeft niet langer een directe relatie met een product. Een categorie heeft nu een relatie met een externe categorie gekregen. Bij 1 categorie kunnen 0 of meer externe categorieën horen. Aan de hand van een categorie kunnen ook producten worden opgezocht. Een categorie is ook wel een interne categorie genoemd.

ExternalCategory
De klasse externalcategory bevat informatie over een categorie zoals de leveranciers deze aanleveren. De leveranciers gebruiken helaas allemaal net iets andere categorieën. Daarvoor is deze klasse aangemaakt. Een externalcategory wordt gekoppeld aan één (interne) categorie. Door het koppelen van de door de leverancier aangeleverde categorieën aan onze eigen categorieën kunnen we in het producten overzicht zoeken op interne categorieën. Zonder deze koppeling is het voor mij lastig om te bepalen waar een product nou onder valt. Ook heb ik deze koppeling nodig voor het bepalen van bij welke categorie een product hoort. De leveranciers gebruiken natuurlijk allemaal andere id’s voor de koppelingen. Daarnaast kan een externalcategory ook nog gekoppeld worden aan een andere externalcategory. Een externalcategory kan maar één parent externalcategory hebben. Wel kan een externalcategory meerdere sub externalcategories bevatten.

Een externalcategory heeft altijd een niveau. Er zijn in totaal 3 niveaus, waarvan niveau 1 het hoogste niveau is en niveau 3 het laagste niveau. In figuur 2 kunt u zien hoe ik deze structuur binnen mijn applicatie toepas.



Figuur 2: Voorbeeld structuur externe categorieën

De leveranciers koppelen hun producten altijd aan een categorie met het niveau 3. Het lastige in dit verhaal was dat niet elke leverancier dezelfde structuur aanhield voor hun categorieën. Zo heeft de ene leverancier 3 categorie niveaus en de andere maar 2. Door gebruik te maken van deze boomstructuur maakt dit in principe niet heel erg veel uit. Zolang de producten maar kunnen worden gekoppeld aan de categorie met het laagste niveau. Vervolgens kan ik van deze categorie opvragen of hij een bovenliggende categorie heeft. Voor de bovenstaande categorie geldt weer precies hetzelfde. Dit zorgt ervoor dat het systeem gemakkelijk uit te breiden is naar een structuur met nog een extra laag niveaus.

3.2 User

De package user bestaat uit de klasse User en Role. Beide klassen zijn automatisch gegenereerd door het Entity Framework. Het Entity Framework biedt standaard de mogelijkheid om authenticatie in een applicatie toe te voegen. Ik heb ervoor gekozen om hier gebruik van te maken. Er is in eerste instantie maar één gebruiker. Wel is het de bedoeling dat in de toekomst meer gebruikers met het systeem gaan werken, hier wil ik dus alvast rekening mee gaan houden.

In figuur 3 vindt u de uitwerking van de package user.

Figuur 3: Klassendiagram user package

User

De klasse User bevat informatie over de ingelogde gebruiker. Daarnaast wordt een user gekoppeld aan een Order. Je kan van een User dus opvragen welke Orders hij allemaal heeft aangemaakt.

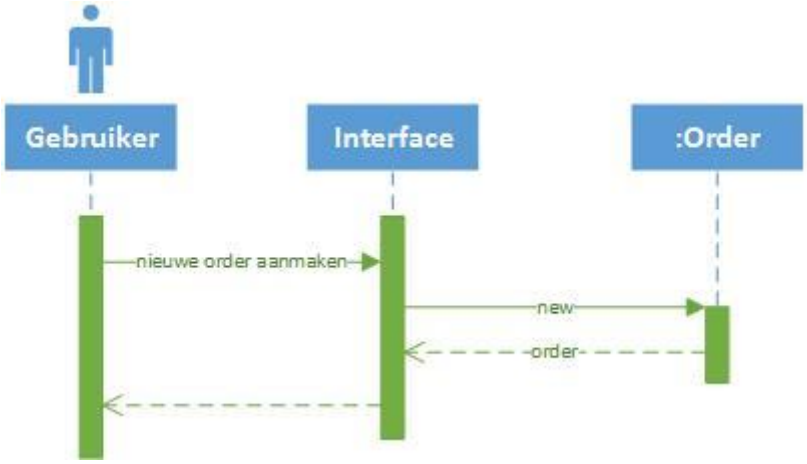
Role
De klasse Role bevat informatie over de aanwezige Rollen in de applicatie. Eén Role kan bij meerdere Users horen. Ook kan een User meerdere Rollen hebben.

4 Sequentie diagrammen

In dit hoofdstuk worden de sequentie diagrammen besproken. De sequentie diagrammen zijn opgemaakt aan de hand van de use cases. Niet voor elke use case is er een sequentie diagram opgesteld, aangezien sommige erg vergelijkbaar zijn. Daarnaast zijn er ook sequentie diagrammen gemaakt van het ophalen van producten en product prijzen bij de leveranciers.

4.1 Aanmaken order

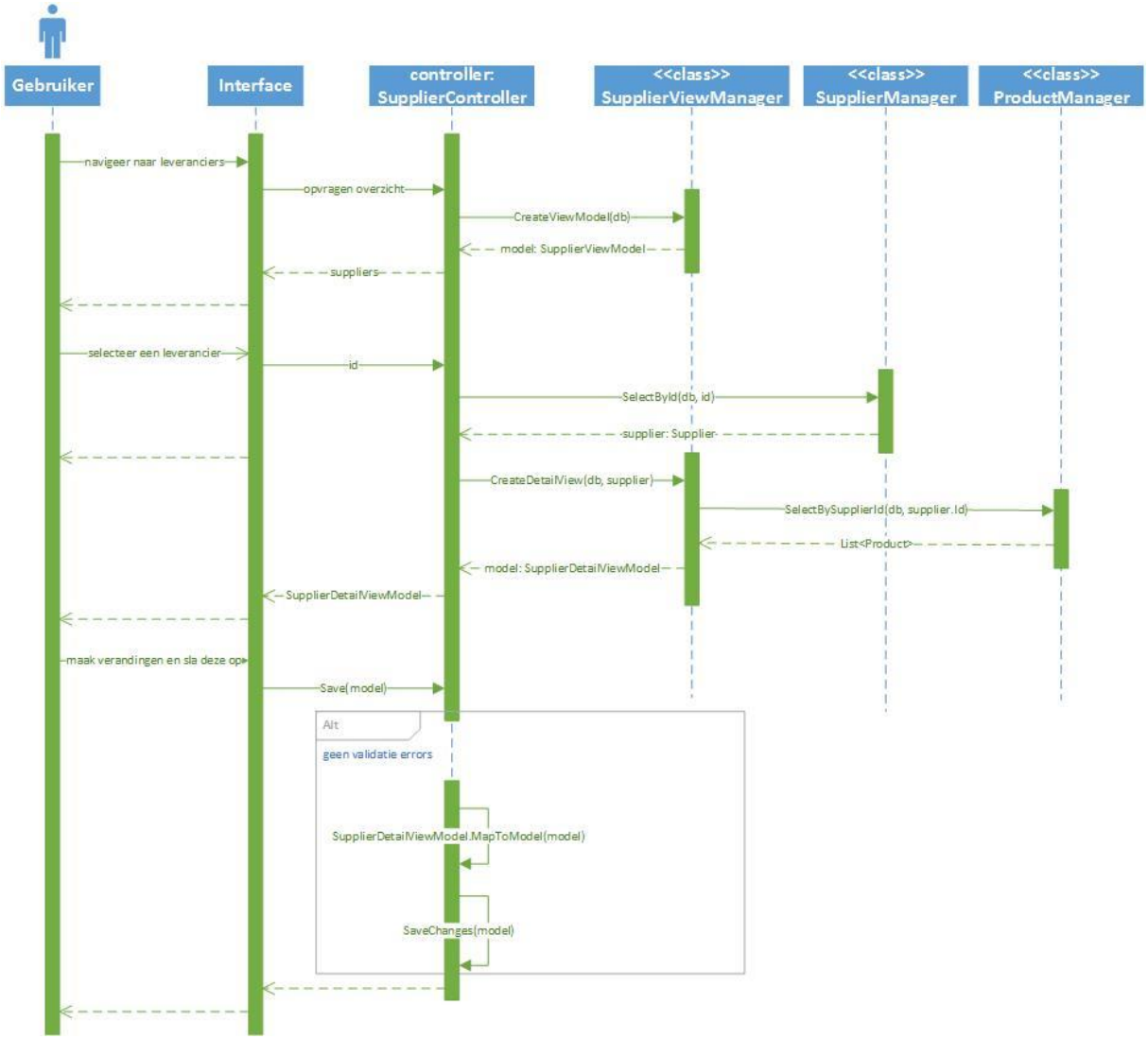
De onderstaande sequentie diagram beschrijft het aanmaken van een order. De gebruiker navigeert naar het order overzicht en selecteert de knop aanmaken nieuwe order. De order wordt vervolgens aangemaakt en er kunnen nu producten aan de order worden toegevoegd.



Figuur 4: Sequence diagram - aanmaken order

4.2 Instellingen leverancier aanpassen

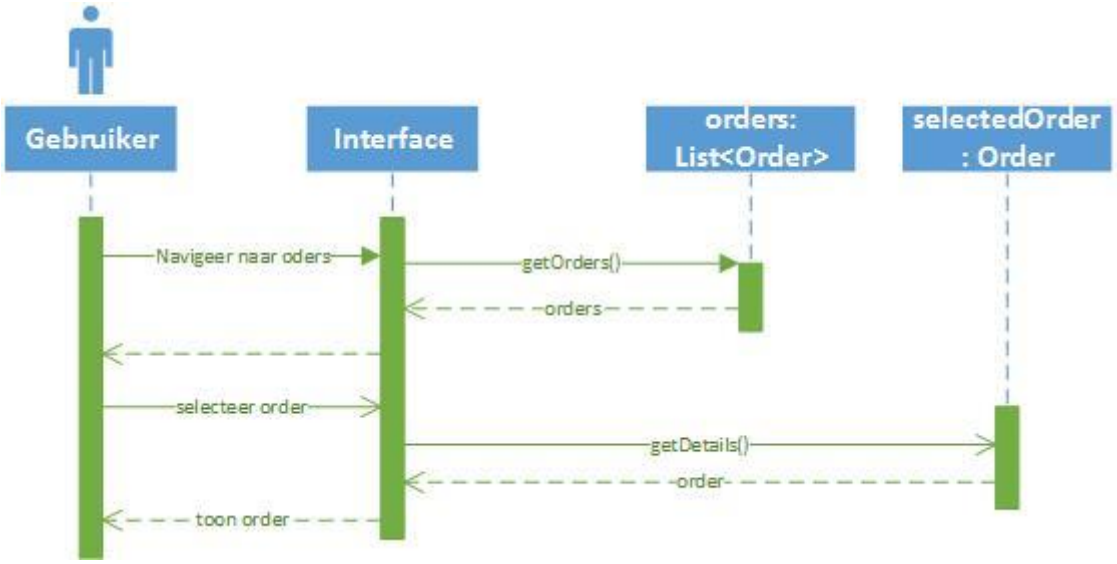
De onderstaande sequentie diagram beschrijft het aanpassen van leveranciers instellingen. De gebruiker navigeert eerst naar het overzicht van leveranciers. Vervolgens selecteerde de gebruiker de leverancier waarvan hij aanpassingen wilt maken. Deze leverancier wordt nu in een detail view getoond. De gebruiken maakt zijn aanpassingen en kiest voor opslaan. Als er geen validatie fouten zijn gevonden worden de wijzigingen opgeslagen in de database. Als er wel validatie fouten zijn gevonden wordt de huidige view weergegeven met de validatie errors.



Figuur 5: Sequence diagram – instellingen leverancier aanpassen

4.3 Order inzien

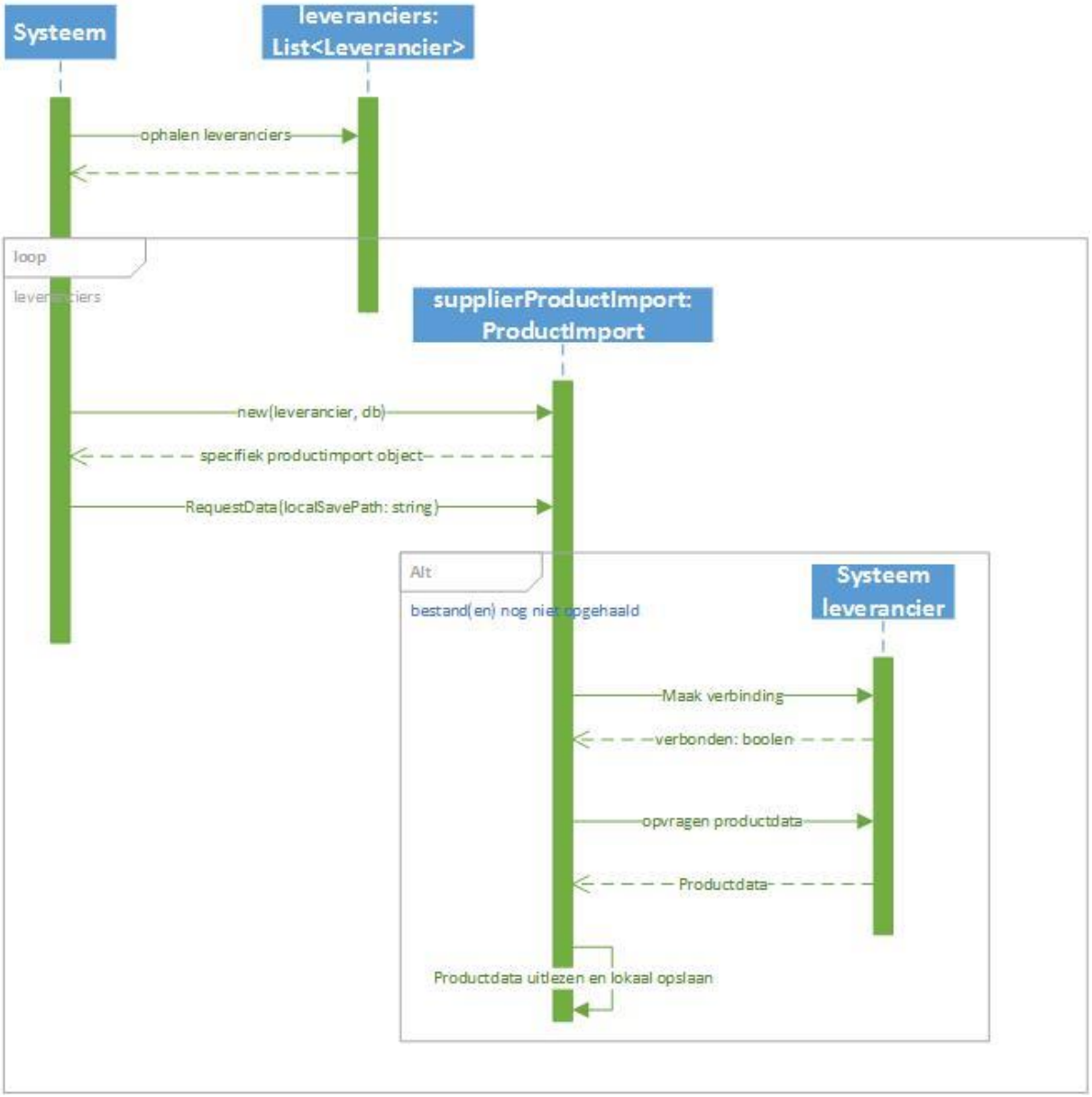
De onderstaande sequentie diagram beschrijft het inzien van een order. De gebruiker navigeert naar het overzicht voor orders. Vervolgens selecteert de gebruiker de order die hij wilt inzien. De gegevens van de geselecteerde order worden opgehaald en weergegeven.



Figuur 6: Sequence diagram - Order inzien

4.4 Product data ophalen

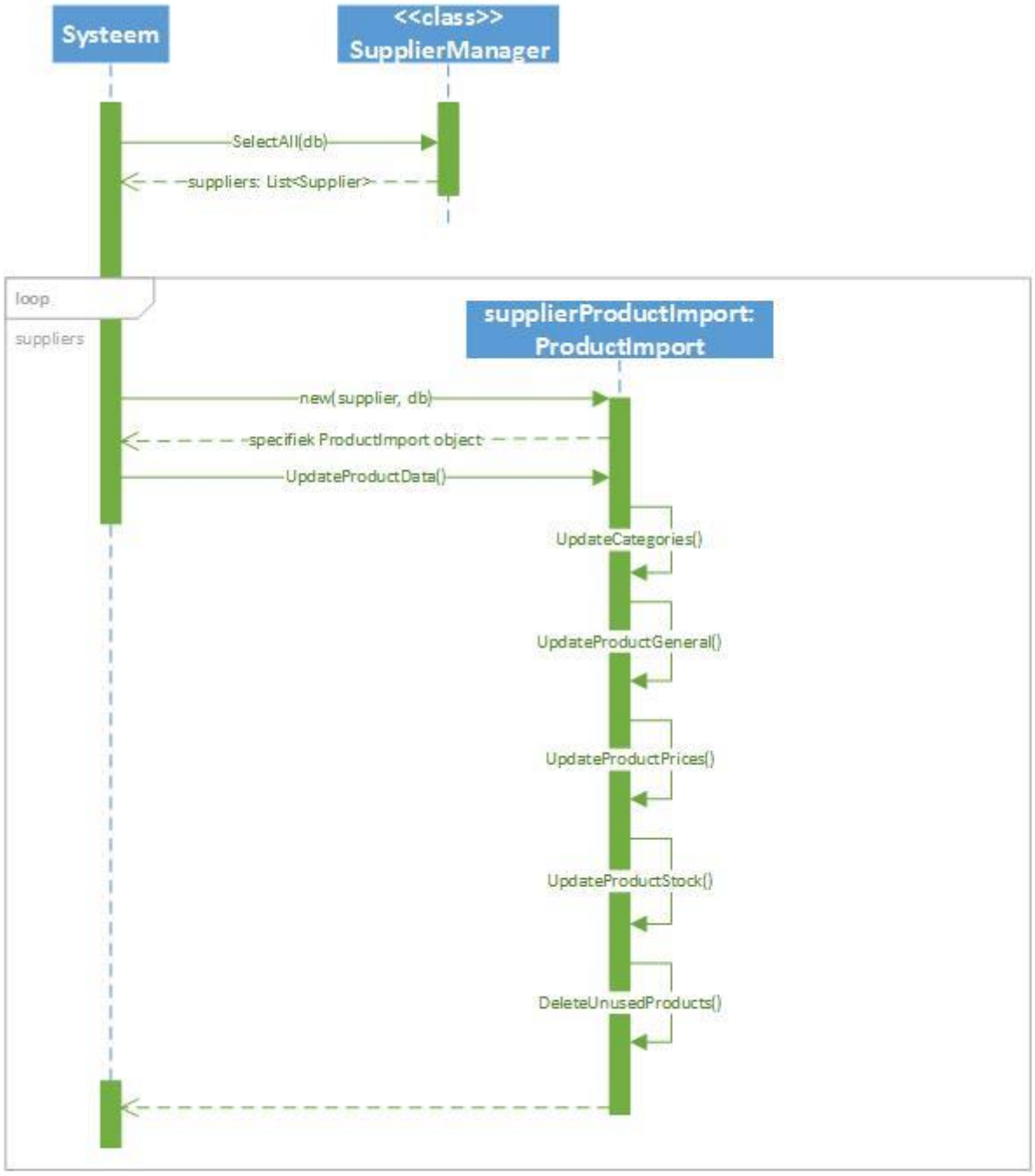
De onderstaande sequentie diagram beschrijft het proces van het ophalen van de productdata bij de leveranciers. De product data zal 1x per dag worden opgehaald op een vastgesteld tijdstip. Het proces begint met het ophalen van de aanwezige leveranciers. Vervolgens wordt er per leverancier de volgende acties uitgevoerd. Er wordt eerst een nieuw ProductImport object aangemaakt van de desbetreffende leverancier. Vervolgens wordt de functie RequestData() aangeroepen, deze functie heeft als parameter het pad waar bestanden moet worden opgeslagen. Er wordt daarna gecontroleerd of er al bestanden aanwezig zijn op het meegegeven pad, als dit niet het geval is wordt er verbinding gemaakt met de leverancier. Zodra de verbinding tot stand is gekomen worden de gegevens opgehaald en lokaal weggeschreven. Het proces is voor alle leveranciers vergelijkbaar, alleen de manier van verbinden en de ontvangen datatype zal verschillen.



Figuur 7: Sequence diagram – Product data ophalen

4.5 Product data bijwerken

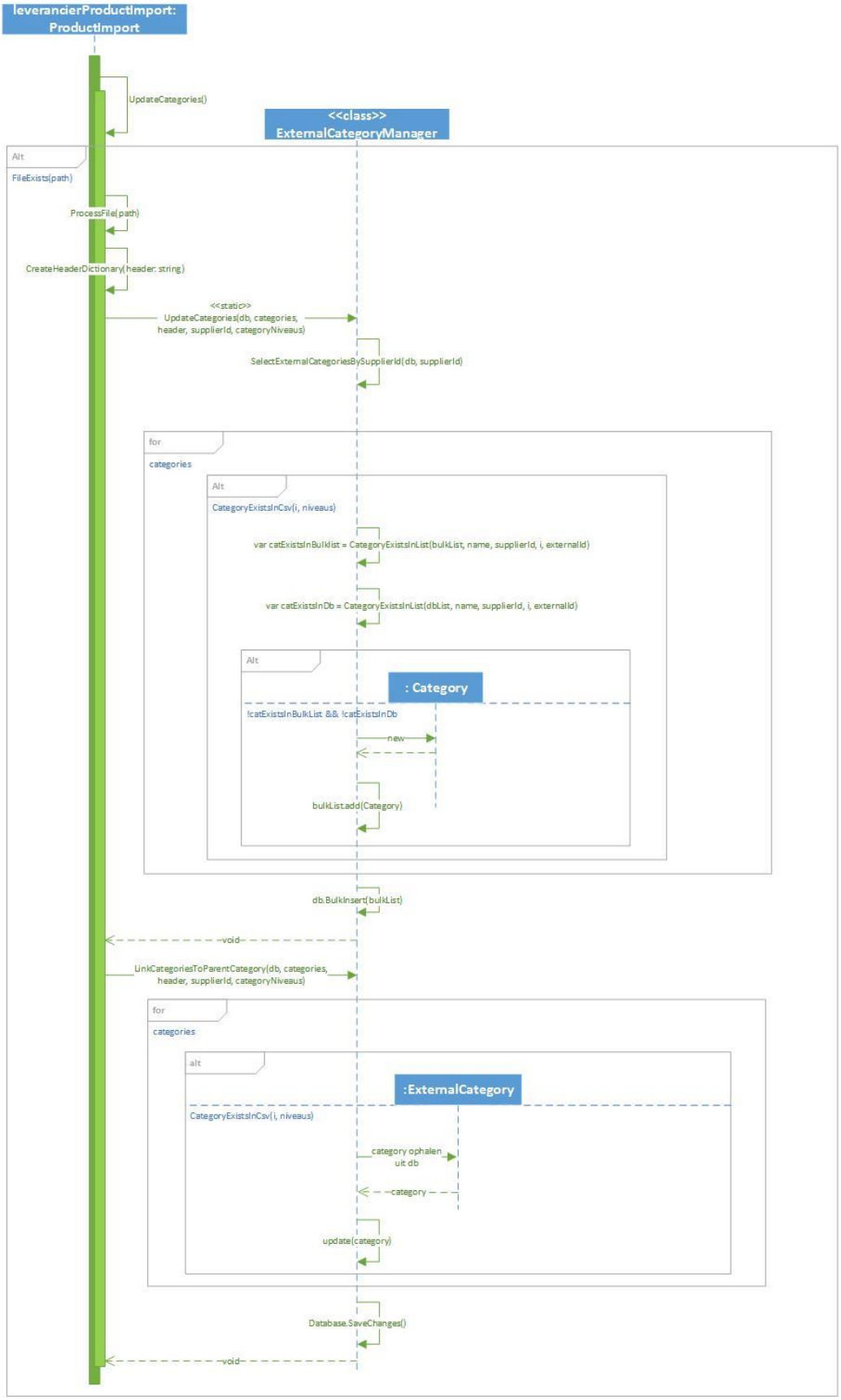
De onderstaande sequentie diagram beschrijft het bijwerken van de product data. Het bijwerken van de product data wordt in principe altijd gedaan na het ophalen van de productdata bij de leveranciers. Er wordt geloopt over de leveranciers en per leverancier wordt de desbetreffende ProductImport object aangemaakt. Vervolgens wordt alle informatie van de producten bijgewerkt. Zoals al valt te zien wordt die gedaan in 5 verschillende functies. Ik heb deze functies in een eigen sequentie diagram uitwerkt, omdat het anders wel erg onoverzichtelijk zou worden.



Figuur 8: Sequence diagram – product data bijwerken

4.6 Categorieën bijwerken

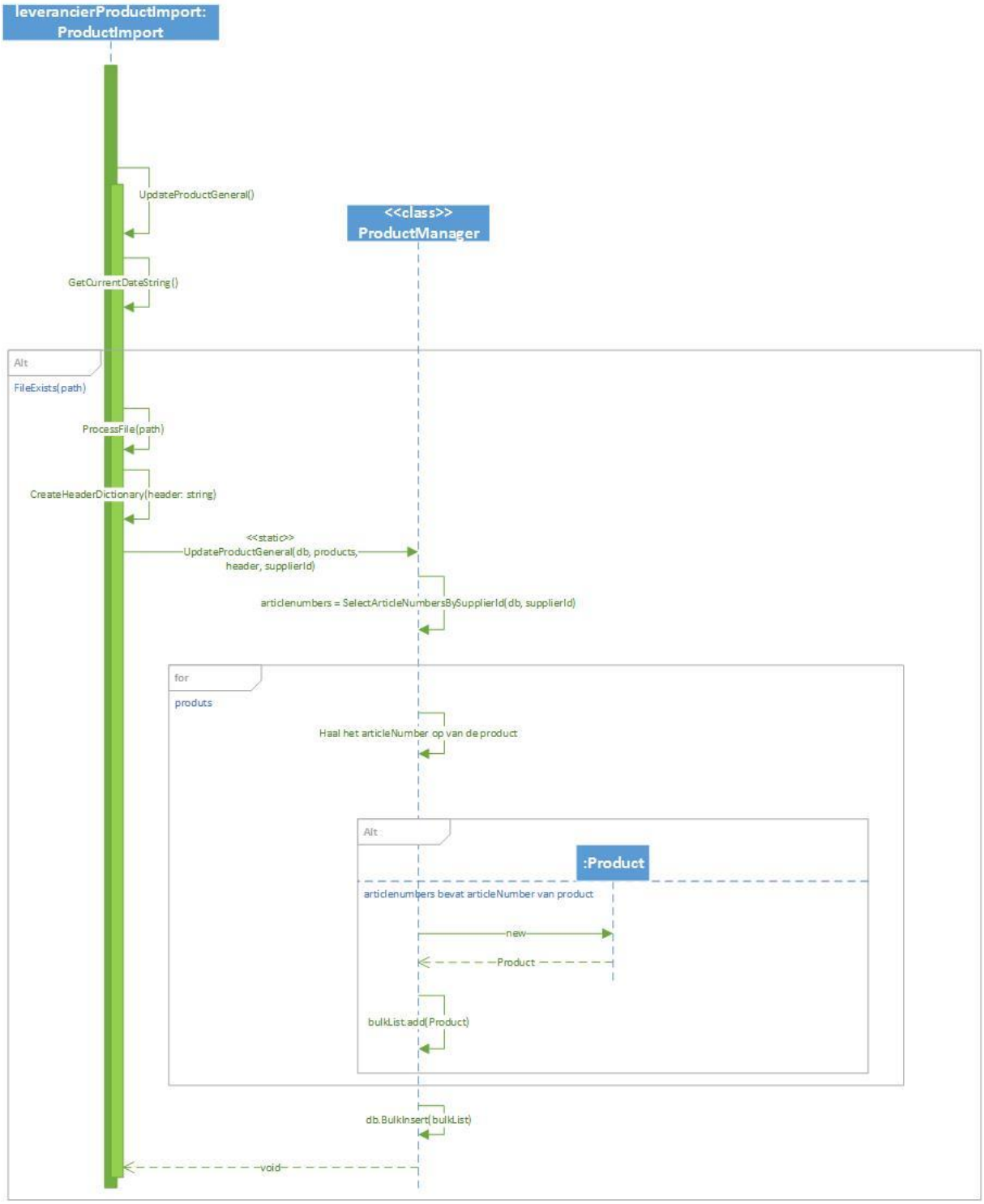
De onderstaande sequentie diagram beschrijft het bijwerken van de categorieën aan de hand van de opgehaalde product data. Er wordt eerst gecontroleerd of er een bestand aanwezig is op het aangegeven pad. Als dit het geval is wordt de file uitgelezen en omgezet naar een lijst met objecten. Vervolgens wordt een de functie CreateHeaderDictionary(string) aangeroepen om lijst met objecten met dictionary keys aan te roepen in plaats van “magic numbers”. Vervolgens wordt de functie UpdateCategories() van ExternalCategoryManager aangeroepen. Binnen deze functie wordt geloopt over de ExternalCategory objecten en een object wordt toegevoegd aan de database als deze nog niet bestond. Nadat de objecten zijn toegevoegd kunnen de ExternalCategories gelinkt worden aan parents als ze aan parent hebben. Dit wordt gedaan door de functie LinkCategoriesToParentCategory().



Figuur 9: Sequence diagram - categorieën bijwerken

4.7 Product bijwerken

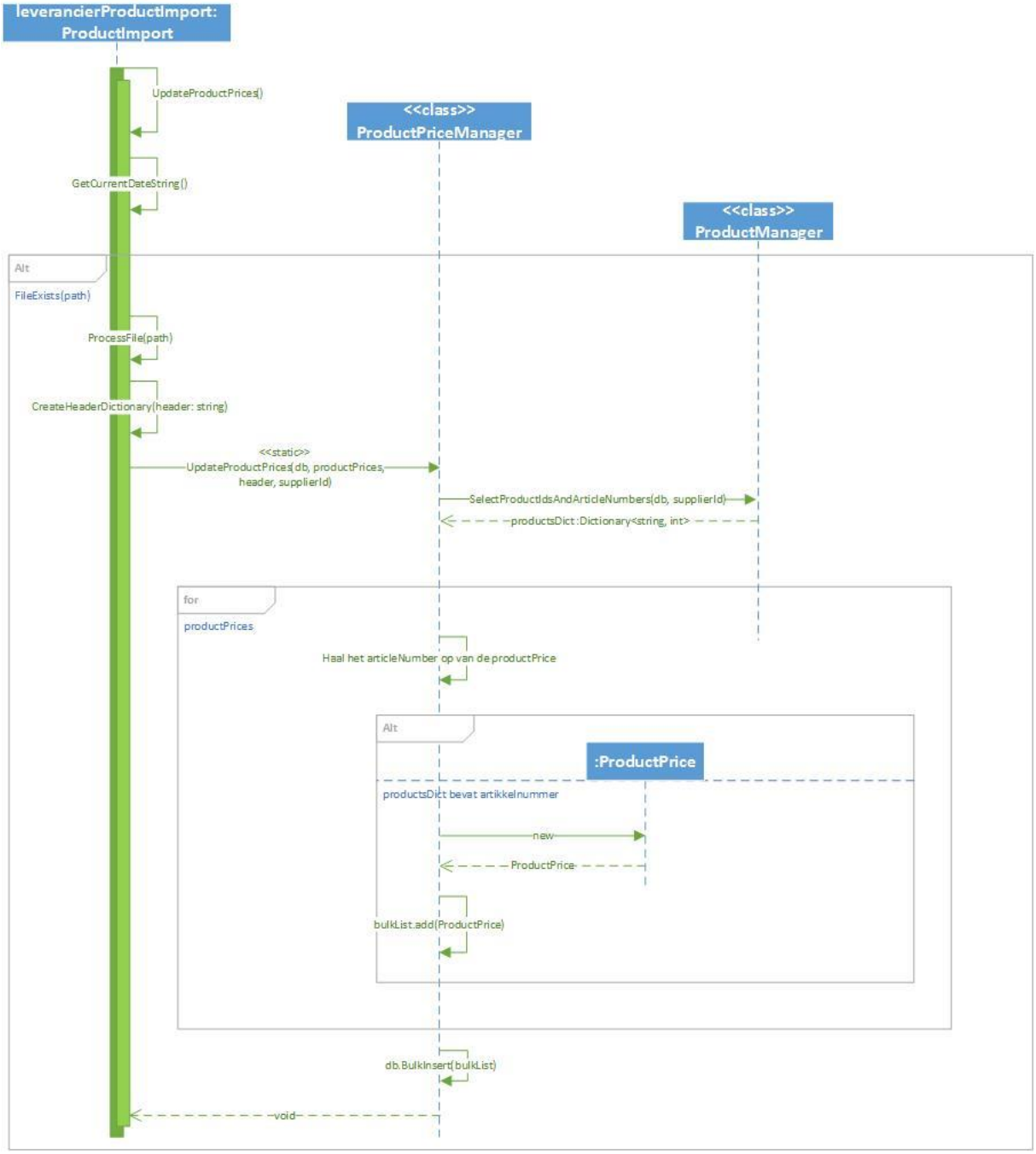
Het bijwerken van de product data werkt ongeveer hetzelfde als het bijwerken van de externe categorieën. Er wordt eerst gekeken of er een bestand is gevonden, vervolgens wordt de uitgelezen en omgezet. De headers worden aangemaakt en de Manager wordt aangeroepen voor het bijwerken van de data. In de manager wordt geloopt over de objecten en als een object nog niet bestaat wordt deze aan de database toegevoegd.



Figuur 10: Sequence diagram - product bijwerken

4.8 Product prijs bijwerken

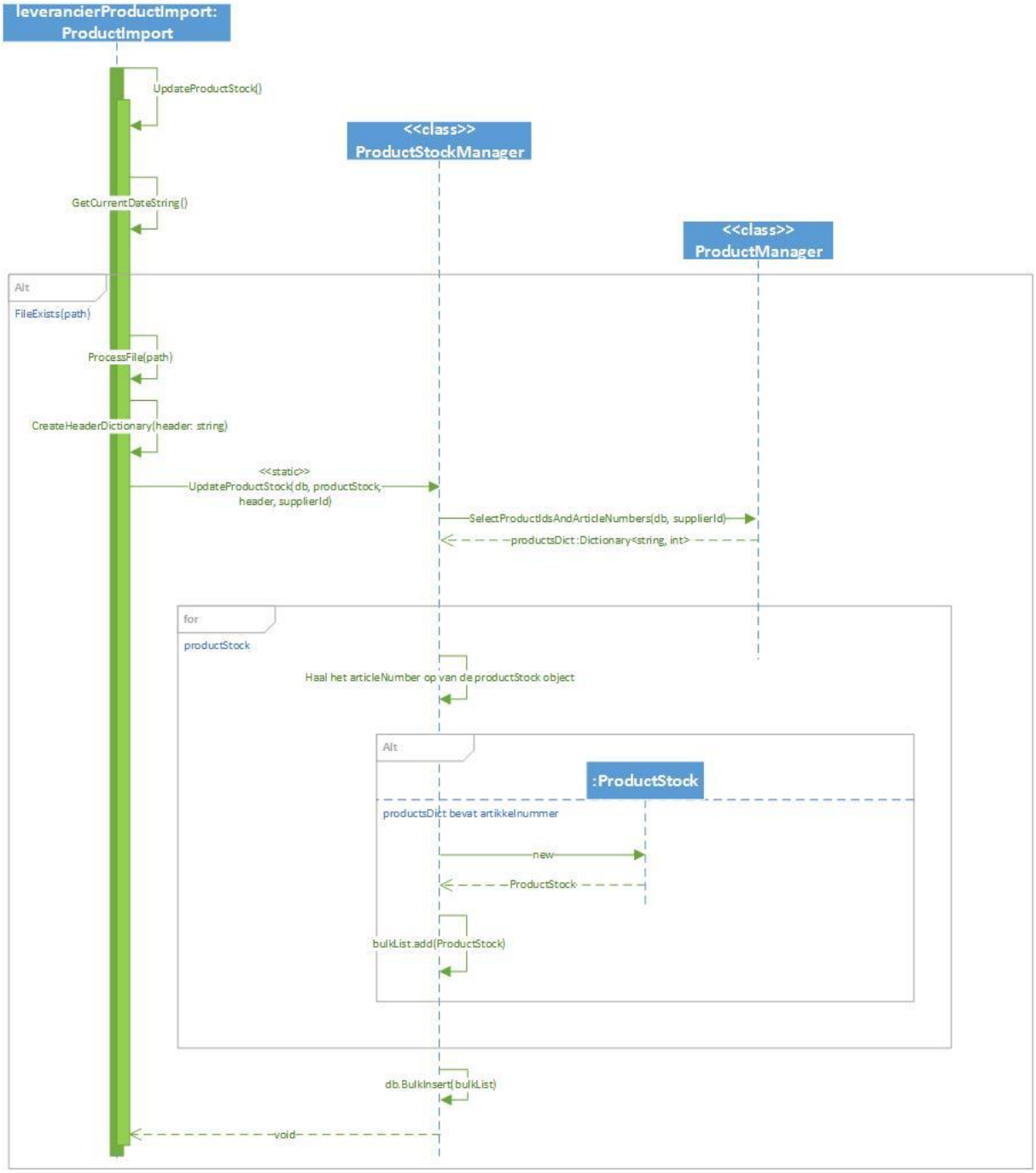
Het bijwerken van de productprijs werkt vergelijkbaar als de 2 bovenstaande bijwerkingen. Er wordt weer gecontroleerd of een bestand is gevonden, vervolgens wordt het bestand uitgelezen en omgezet. De headers worden weer aangemaakt en daarna wordt de Manager van ProductPrice aangeroepen. Elke ProductPrice object wordt vervolgens aan de database toegevoegd.



Figuur 11: Sequence diagram - product prijs bijwerken

4.9 Product voorraad bijwerken

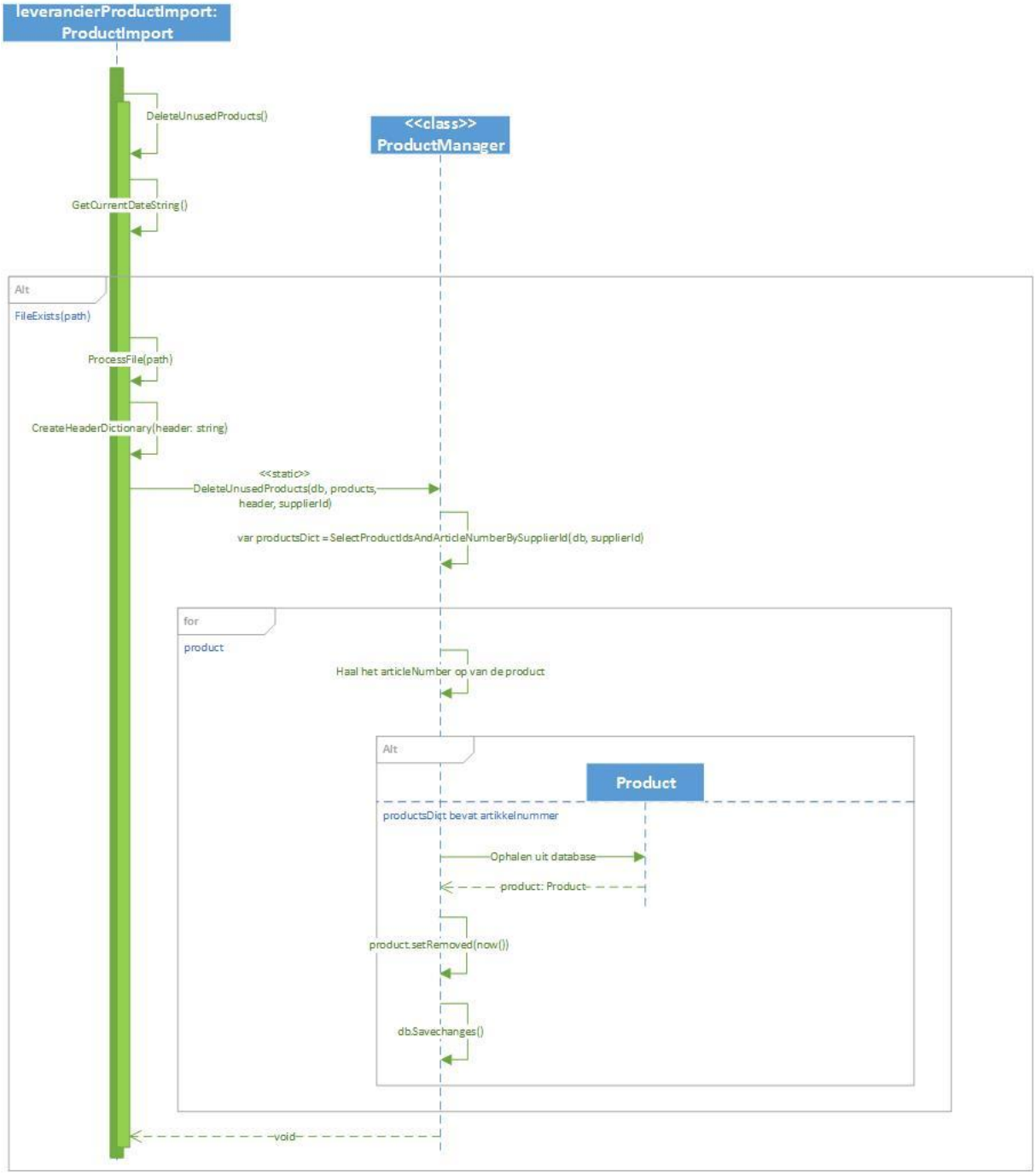
Het bijwerken van de voorraad werkt precies hetzelfde als het bijwerken van de productprijzen. Het enige verschil is de file die wordt uitgelezen.



Figuur 12: Sequence diagram - product voorraad bijwerken

4.10 Ongebruikte producten verwijderen

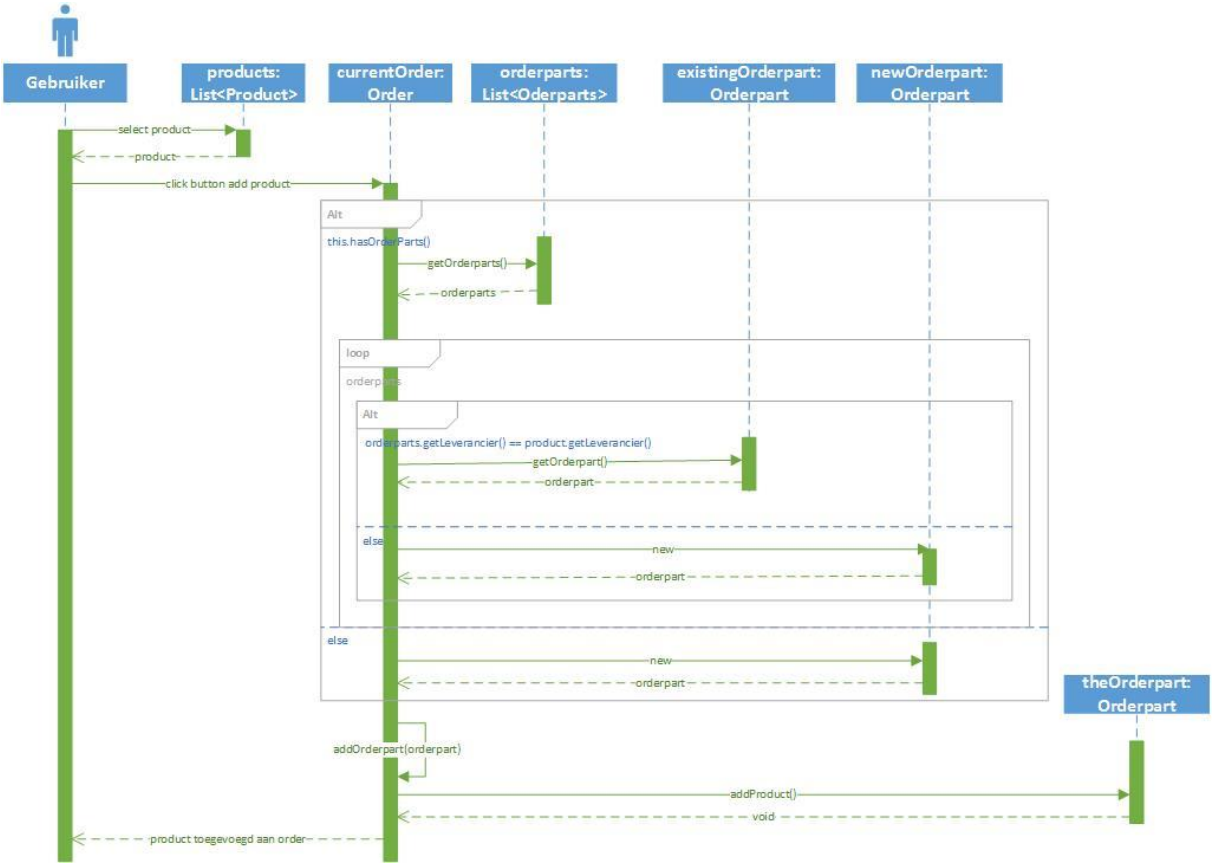
Het verwijderen van producten lijkt we erg op de bovenstaande sequentie diagrammen. Eerst wordt gekeken of er een bestand is gevonden, vervolgens wordt het bestand omgezet naar objecten en wordt de header weer aangemaakt. Daarna wordt de functie DeleteUnusedProducts van ProductManager aangesproken en wordt er geloopt over de aanwezige Product objecten. Per object wordt de bijbehorende Product opgezocht en hier wordt de attribuut removed op de huidig datum gezet. Wanneer de attribuut removed in ingevuld betekend het dat een product is verwijderd.



Figuur 13: Sequence diagram - ongebruikte producten verwijderen

4.11 Product toevoegen aan order

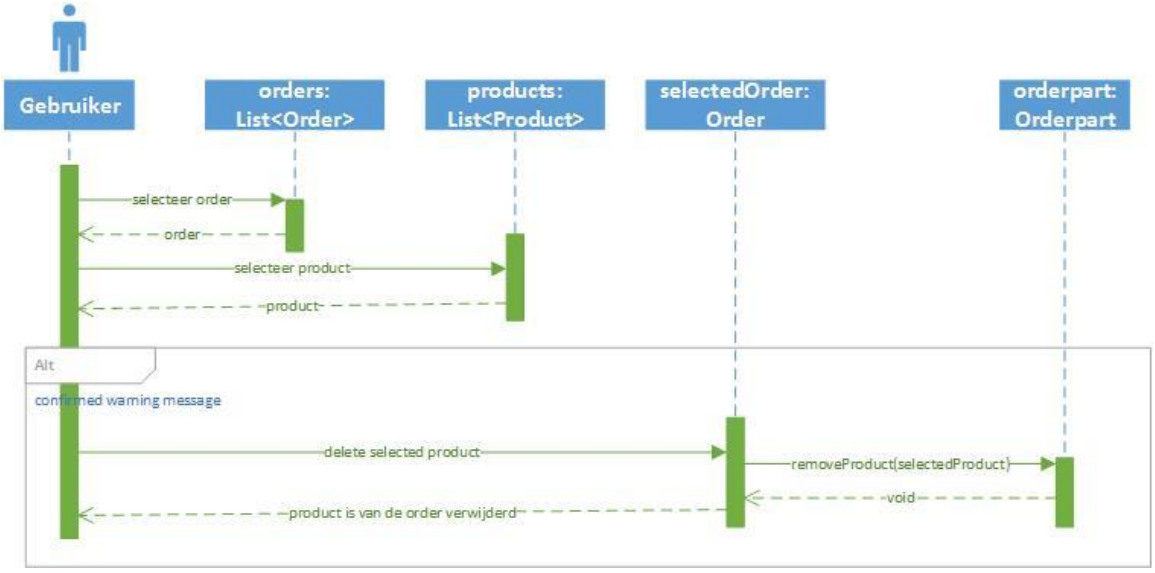
De onderstaande sequentie diagram beschrijft het toevoegen van een product aan een order. De gebruiker navigeert eerst naar het overzicht met producten. Vervolgens selecteert hij een product en kiest ervoor om deze toe te voegen aan de openstaande order. Vervolgens worden de bestaande orderparts van een order opgehaald, mocht de order nog geen orderparts bevatten wordt en een nieuwe aangemaakt. Mocht de order wel een orderpart bevatten wordt er gekeken of de leverancier overeenkomt met de leverancier van het geselecteerde product. Mocht dit niet het geval zijn wordt er alsnog een nieuwe orderpart aangemaakt. De orderpart wordt gekoppeld aan de order en de geselecteerde product(en) worden gekoppeld aan de orderpart(s).



Figuur 14: Sequence diagram - product toevoegen aan order

4.12 Product verwijderen van order

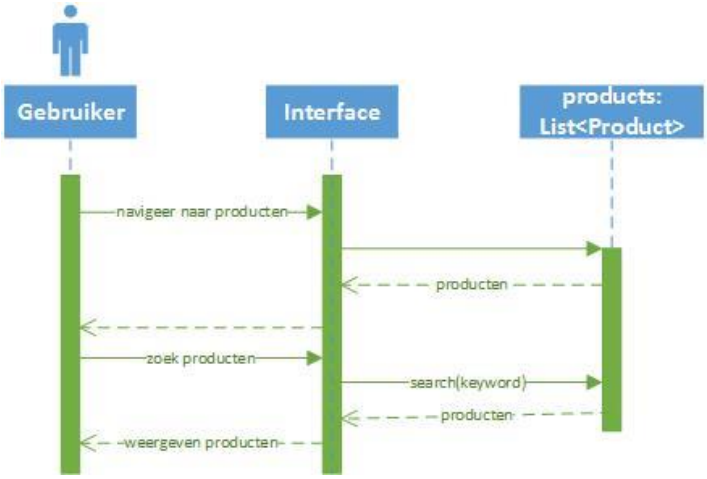
De onderstaande sequentie diagram beschrijft het verwijderen van een product van een order. De gebruiker navigeert naar het overzicht met orders en selecteert de order die hij wilt aanpassen. Vervolgens worden de gegevens van de geselecteerde order worden ingeladen. De gebruiker selecteert het product die hij van de order wilt verwijderen. Vervolgens krijgt de gebruiker een bevestigingsmelding, als deze wordt geaccepteerd wordt het product van de order verwijdert. In alle andere gevallen blijft het product gekoppeld aan de order.



Figuur 15: Sequence diagram - product verwijderen van order

4.13 Producten zoeken

De onderstaande sequentie diagrammen beschrijft het zoeken naar een product. De gebruiker navigeert eerst naar het producten overzicht. Vervolgens typt de gebruiker zijn toekterm in en kiest hij voor zoeken. De zoekactie wordt nu uitgevoerd en er wordt een nieuwe producten overzicht getoond.



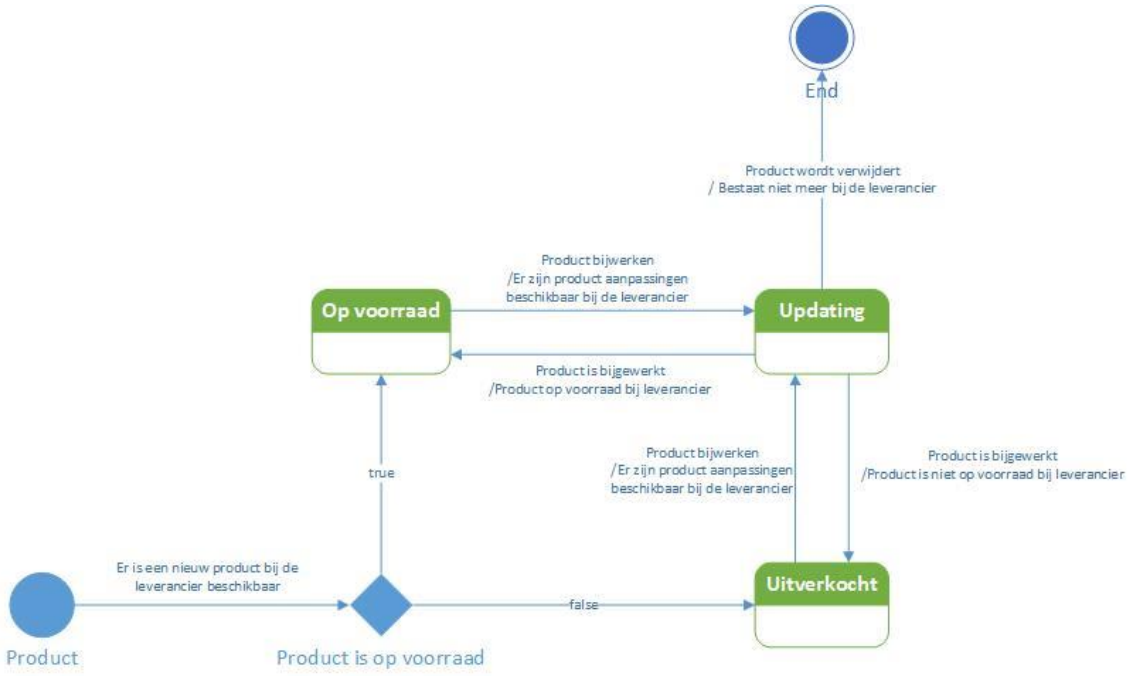
Figuur 16: Sequence diagram - producten zoeken

5 Activiteiten diagram

In dit hoofdstuk wordt de activiteiten diagram voor product besproken.

5.1 Product

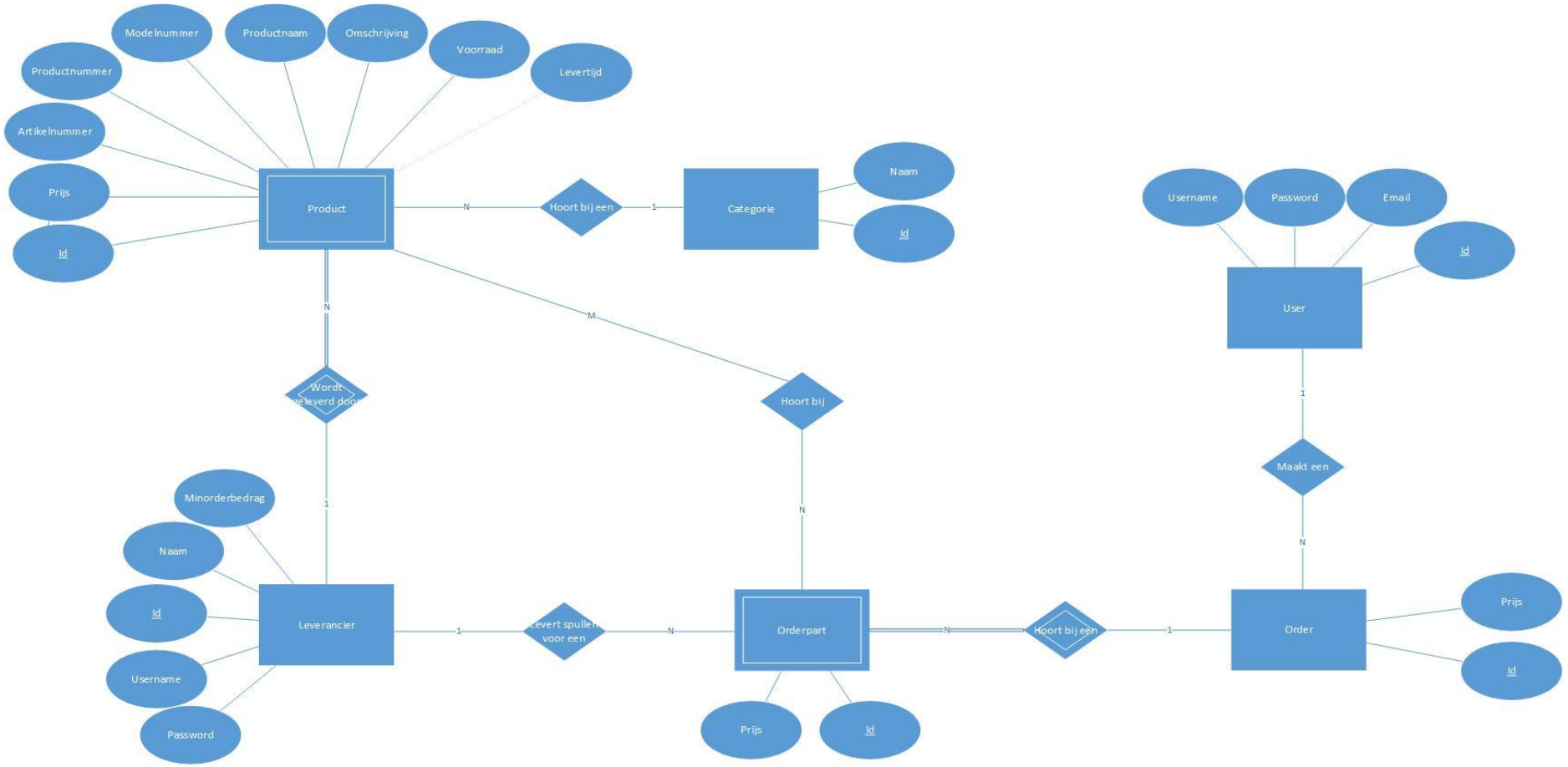
Het onderstaande activiteiten diagram beschrijft de verschillende staten waarin een product kan verkeren.



Figuur 17: Activiteiten diagram product

6 Database ontwerp

In dit hoofdstuk wordt het database ontwerp besproken. Er is in dit gebruik gemaakt van een entity-relationship model. Onderstaand vind u de uitwerking hiervan. Dit ontwerp is uitgewerkt op basis van het eerste ontwerp van het klassendiagram. Zoals u kunt zien missen er ten opzichte van mijn huidige klassendiagram wat entiteiten aan. Aangezien ik gebruik maak van het entity framework in combinatie met code first zag ik de toegevoegde waarde niet echt meer om dit diagram nog verder bij te werken.



Figuur 18: Entity relatie model

7 Relational Representation Model

In dit hoofdstuk wordt het RRM besproken. Net zoals bij het bovenstaande ontwerp is deze gebaseerd op het eerste ontwerp van het klassendiagram.

Product(id, artikelnummer, productnummer, modelnummer, omschrijving, voorraad, levertijd, prijs, *leverancierId, categorieId*)
LeverancierId vreemde sleutel naar id in Leverancier, NULL niet toegestaan.
CategorieId vreemde sleutel naar id in Categorie, NULL niet toegestaan

Categorie(id, naam)

Leverancier(id, naam, username, password, minimunOrderbedrag)

Order(id, prijs *userId*)
Userid vreemde sleutel naar id in User, NULL niet toegestaan

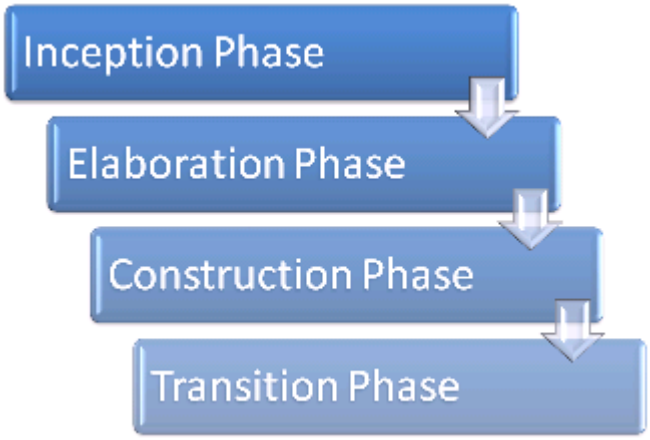
Orderpart(id, prijs, *leverancierId*)
Leverancierid vreemde sleutel naar id in Leverancier, NULL niet toegestaan

Orderpart_product(*orderpartId*, *productId*)
OrderpartId vreemde sleutel naar id in Order, NULL niet toegestaan
ProductId vreemde sleutel naar id in Product, NULL niet toegestaan

User(id, username, password, email)

Construction rapport 1

Hardware offerte tool



Auteur: Donny Roelen

Datum: 05-1-2015

Versie 1.1

Inhoudsopgave

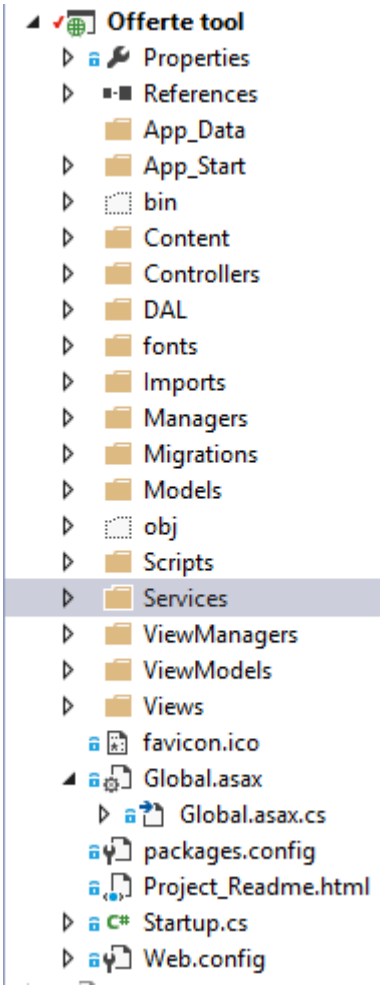
1 Inleiding

In dit hoofdstuk worden de artefacten die tijdens de construction fase van iteratie 1 aan bod zijn gekomen besproken. Keuzes met betrekking tot de bouw van de software van de eerste iteratie zullen in dit document worden besproken. De eerste iteratie bestaat uit de basis van het systeem en zal bestaan uit voornamelijk must have requirements. Het is de bedoeling dat de eerste iteratie een bruikbaar product oplevert voor de opdrachtgever.

2 Opzetten project

Ik ben het software ontwikkeltraject begonnen met het opzetten van mijn Visual studio project. Ik ben begonnen met het aanmaken van de structuur. In figuur 1 ziet u de mappen van mijn project.

Map	**Omschrijving**
App_Data	Wordt standaard aangemaakt, bevat geen bestanden.
App_Start	Bevat informatie over de mapping van models naar viewsmodels en andersom.
Content	Bevat de CSS bestanden van onder andere Bootstrap.
Controllers	Hier staan de uitwerkingen van de controllers
DAL	DAL staat voor Data Access Layer, in dit mapje staat de DbContext klasse met de bijbehorende tabellen en mappings.
Fonts	Plaatjes van Bootstrap worden hier opgeslagen.
Imports	Wordt gebruikt om de product data bestanden van de leveranciers in op te slaan. Per leverancier komt er een apart mapje in de map Imports.
Managers	De functies die de database aanspraken worden hier geplaatst. Bijvoorbeeld de CRUD functies komen hier.
Migrations	Het mapje migrations is aangemaakt door het Entity framework voor de code first migraties. De aangemaakte migratie bestanden komen in deze map terecht.
Models	Bevat de model klassen.
Scripts	Javascripts bestanden worden hier opgeslagen.
Services	Hier komen de klasse die betrekking hebben tot het ophalen van de data van de leveranciers.
ViewManagers	Services is een mapje die ik zelf heb aangemaakt, ik ben van plan om hier de code te plaatsen voor het maken van de koppelingen naar de leveranciers.
ViewModels	Zie het kopje viewmanagers
Views	Zie het kopje viewmodels



Figuur 1: Mappen structuur

2.1 Viewmanagers

In het mapje ViewManager komen de ViewManagers, een ViewManager maakt de Viewmodels aan en zorgt voor een minder sterke koppeling tussen de ViewModel objecten (loose coupling). Wanneer Object A bijvoorbeeld een instantie van Object B moet aanmaken dan doet niet Object A dit maar de ViewManager van Object A. Dit wordt ook wel een vorm van het Mediator pattern genoemd.

2.2 Viewmodels

In het mapje ViewModels komen de ViewModel klassen. Een ViewModel is een extra laag tussen de View en de Model. Een ViewModel hoort altijd bij maar één model klasse. Door het toevoegen van de ViewModel klassen haal ik de logica uit mijn views. Daarnaast is het werken met ViewModels veiliger, je stuurt alleen nog maar de data die je wilt tonen naar de View en geen onnodige dingen die een Model object normaal wel mee zou sturen. Daarnaast is de koppeling tussen model en de view minder sterk geworden. Ik kan nu mijn Model aanpassen zonder dat mijn Views ook aangepast hoeven te worden.

In figuur 17 wordt de communicatie tussen de views, viewsmodels en model klassen weergegeven.

Figuur 2: ViewModel [http://en.wikipedia.org/wiki/Model_View_ViewModel]

Het laatste mapje is Views, in dit mapje komen natuurlijk de views die de website zullen weergeven.

Ik maak binnen mijn project eigenlijk gebruik van het MVVM (Model View ViewModel) pattern. Interpulse gebruikt dit binnen al haar nieuwe ASP.net MVC projecten als standaard. Het MVVM pattern is gebaseerd op het MVC pattern. Het verschil is echter dat bij het MVVM pattern een view niet meer direct de data van een model object uitleest maar van een viewmodel. Met andere worden er is dus een extra laag tussen gekomen.

2.3 Code first configureren

Nadat de structuur van het project was neer gezet ben ik verder gegaan met het configureren van code first. Voordat ik mijn database kan laten aanmaken met code first moest ik eerst al mijn model klassen aanmaken. Ik heb dus al mijn model klassen aangemaakt zoals ik ze had ontworpen in de elaboration fase. Vervolgens heb ik migrations aangezet, zodat ik met het entityframework de eerste migratie kan aanmaken.

Het toevoegen van de eerste migratie gaf echter problemen, de migratie kan niet worden aangemaakt vanwege een cascade delete error tussen de tabel Product en Orderpart_Product. Waarom ik deze error kreeg is mij niet geheel duidelijk, maar ik heb dit opgelost door handmatig een aanpassing te maken in de migratiefile. Nadat ik on delete cascade uit had gezet voor ProductId in Orderpart_Product werkte de migratie wel. Deze aanpassing zorgt echter weer voor een ander probleem. De tussentabel wordt niet meer geleegd wanneer een product wordt verwijderd. Dit heb ik opgelost door een trigger te maken die na het verwijderen van een product ook de tussentabel leegt.

2.4 Bootstrap configureren

Het in eerste instantie configureren van Bootstrap is vrij eenvoudig. Ik heb deze gewoon toegevoegd met behulp van de NuGet package manager binnen Visual Studio. NuGet is een package management systeem voor .NET. Met NuGet kan je gemakkelijk externe libaries aan een Visual Studio project toevoegen. Daarnaast kan je van geïnstalleerde libaries snel zien of er updates voor beschikbaar zijn. Als je ervoor kiest om een update binnen te halen worden de bestanden automatisch bijgewerkt.

2.5 GIT configureren

Binnen Interpulse wordt sinds kort gebruik gemaakt van GIT als standaard versiebeheer tool. Ik heb met behulp van een collega voor mijn afstudeerproject een GIT repository opgezet. Zodra een stuk functionaliteit klaar was zorgde ik ervoor dat dit naar de GIT server werd gepushed. Hierdoor staat het grootste deel van mijn code altijd veilig. Mocht mijn werk computer ermee ophouden heb ik het grootste deel van mijn werk in ieder geval nog op de GIT server staan.

2.6 Bouwen eerste functionaliteit

Voor de eerste opzet van de views en de controllers binnen de applicatie heb ik gebruik gemaakt van scaffolding. Het entity framework maakt dan de CRUD views en controller functies voor je aan. De aangemaakte functies en views zijn echter wel beperkt, deze heb ik dan ook in de loop van het project verder uitgewerkt. Ik heb er meer gebruik van gemaakt als eerste opzet en om een beetje aan het MVVM pattern te wennen.

Het plan was om vervolgens verder te gaan met het bouwen van de eerste koppeling, namelijk de koppeling met Techdata. Dit kon alleen nog niet, omdat we moesten wachten op de FTP login gegevens van Techdata. In plaats daarvan ben ik begonnen met het aanmaken van dummy data om hier vervolgens mee aan de slag te gaan. De dummy data liet ik aanmaken door de Seed methode. De Seed methode is een methode van de klasse DbContext die wordt aangeroepen bij het aanmaken of wijzigen van de database.

3 Leveranciers module

Ik ben begonnen met de kleinste module namelijk die van de leveranciers. Ik heb de basis functionaliteiten die aanwezig waren na het scaffolding aangepast en verder uitgebreid naar mijn eigen wens. De leverancier module bestaat uit een overzicht, toevoegen, bewerk en detail scherm. Verder niet heel erg bijzonder, wat wel interessant is dat je per leverancier kan zien hoeveel producten deze in de database heeft staan. Dit is niet iets wat je uit het model object kan halen, maar dit heb ik zelf als property aan de viewmodel voor leverancier overzicht toegevoegd. Vervolgens heb ik dit aantal in de viewmanager opgehaald en aan de viewmodel toegevoegd. In figuur 3 zie je de functie die de lijst van leveranciers opbouwt.

```
public List<SupplierViewModelGrid> CreateGrid(OfferteToolContext db)
{                                                    1
    var suppliers = SupplierManager.Instance.SelectAll(db).ToList();
    var model = suppliers.Select(a => new SupplierViewModelGrid()
    {                                                    2
        Id = a.Id,
        Name = a.Name,                                   3
        NumberOfProducts = ProductManager.Instance.SelectBySupplierId(db, a.Id).Count(),
        CategoryNiveaus = a.CategoryNiveaus,
        MinimumOrderAmount = a.MinimumOrderAmount
    }).ToList();
                                                         4
    return model;
}
```

Figuur 3: Leveranciers overzicht, viewmanger code

Ik haal eerst de beschikbare leverancier uit de database, dit doe ik via de SupplierManager. Zoals u kunt zien roep ik ook “Instance”aan (bij nummertje 1). Dit stukje zorgt ervoor dat er een statische instantie van de klasse SupplierManager wordt opgehaald. Via deze statische instantie kan ik vervolgens alle functies van de betreffende klasse aanroepen zonder een object aan te maken. Dit concept gebruik ik voor mijn manager en viewmanager klassen. In figuur 4 kunt u zien hoe ik deze constructie heb toegepast voor de manager klassen. In de BaseManager wordt deze constructie opgezet, vervolgens overerven allen subklassen van de BaseManager. In de BaseManager is alles op een generieke manier opgezet zodat alle subklassen er gebruik van kunnen maken.

1. Bij punt 1 wordt de juiste tabel gezet voor de betreffende entiteit.
2. Hier wordt de bijbehorende model klassen gezet.
3. De entiteit wordt gezet.
4. Maakt een nieuwe instantie aan van de desbetreffende klasse.
5. Return de aangemaakte instantie.

```
public abstract class BaseManager<TEntities, TEntity, TSelf>
    where TEntities : DbContext 1
    where TEntity : class 2
    where TSelf : BaseManager<TEntities, TEntity, TSelf>, new()
{       3
    private static TSelf _instance = new TSelf(); 4


    #region Properties

    public static TSelf Instance
    {
        get
        {
            return _instance; 5
        }
    }

    #endregion Properties
```

Figuur 4: Gebruik instance

Als we weer verder gaan met figuur 3 zie je dat ik bij punt 2 een SupplierViewModelGrid object aanmaak. Dit is een viewmodel klassen voor het overzicht van de leveranciers. Voor elke aanwezige

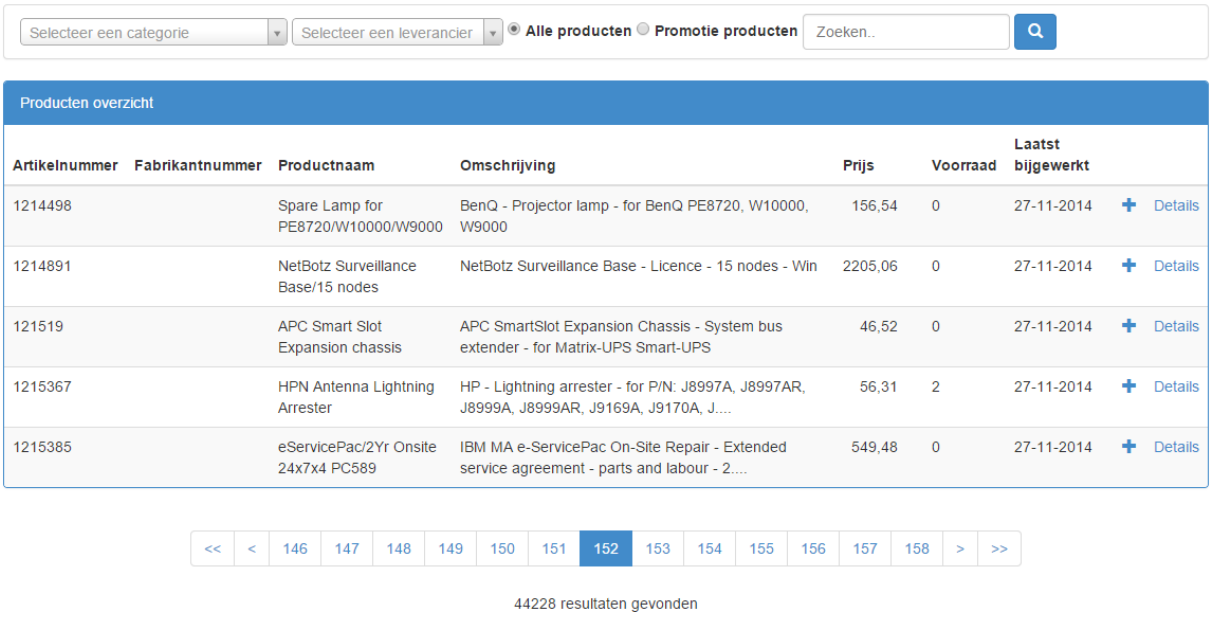
leverancier wordt dit viewmodel object aangemaakt en gevuld. Vervolgens wordt dit ViewModel object gebruikt om informatie van een leverancier weer te geven in de tabel. Eén SupplierViewModelGrid staat gelijk aan één regel in de tabel van leveranciers. Bij punt 3 haal ik via een andere manager klasse het aantal bijbehorende producten van een leverancier op. Punt 4 is bevat de LINQ query die ik al gedeeltelijk had omschreven, ik loop dus over de lijst van leveranciers. Vervolgens wordt er per leverancier een viewmodel object aangemaakt.

Wat nog wel interessant is om te melden is dat ik de lijst van leveranciers (1e regel) gelijk omzet naar een lijst. Normaal wil je dat niet doen, omdat dit slecht is voor de performance. Elke regel uit de query moet namelijk toegevoegd worden aan een lijst. Maar aangezien het maar een lijst is van enkele leveranciers maakt dit niet echt uit. Maar dat is niet de reden dat ik het zo gedaan heb, de reden is namelijk de aanroep van een functie in een lambda expressie. De regel bij nummer 3 had een error gegeven als ik de leveranciers niet van een IQueryable had omgezet naar een list of array. De reden voor deze error is dat het entity framework probeert de LINQ Query om te zetten naar SQL. Alleen hij kent de methode die aangeroepen wordt niet. Dit los je dus op door eerst de data in een list te zetten en vervolgens een LINQ query op de list uitvoeren.

4 Producten overzicht

Vervolgens ben ik begonnen met een overzicht op te zetten voor producten. Het producten overzicht bestaat uit een tabel met de producten, een pager, een zoekveld, een filter voor categorieën en een filter op leverancier.

In figuur 5 kunt u zien hoe het producten overzicht eruit ziet, voor dit voorbeeld toon ik echter maar 5 records in plaats van 20 records per pagina, anders wordt het plaatje wel erg groot.



Figuur 5: Producten overzicht

De zoek functie voor producten heb ik als volgt opgebouwd. Ik haal eerst alle producten met het entity framework op. Vervolgens ga ik per filteroptie een LINQ where query uitvoeren. Ik gebruik eigenlijk door mijn hele applicatie LINQ queries met daarin lambda expressie. Ik vind de lambda expressies fijner lezen dan de standaard LINQ expressie.

De zoekfunctie voor producten ziet er uit zoals in figuur 6 weergegeven.

```
public IQueryable<Product>  SelectBySearch(OfferteToolContext db, ProductFilterOptions filterOptions)
{
    var products = this.SelectSecure(db);

    if (filterOptions.CategoryId > 0)
    {
        products = SelectByCategoryId(db, filterOptions.CategoryId, products); 1
    }
    if (filterOptions.SupplierId > 0)
    {
        products = products.Where(a => a.SupplierId == filterOptions.SupplierId);
    }
    if (!String.IsNullOrEmpty(filterOptions.Q))
    {
        int unusedInt;
        if (Int32.TryParse(filterOptions.Q, out unusedInt))
        {
            products = products.Where(a => a.Articlenumber.Contains(filterOptions.Q));
        }
        else
        {
            products = products.Where(a => a.Manufacturernumber.Contains(filterOptions.Q) ||
                a.Productname.Contains(filterOptions.Q) || a.Description.Contains(filterOptions.Q));
        }

    }
    if (filterOptions.PromotionPrices == 1)
    {
        products = products.Where(a => a.ProductPrices.OrderByDescending(b => b.Id).Take(1).FirstOrDefault
    }

    return products;
}
```

Figuur 6: Products zoek functie

Er is nog wel iets bijzonders aan deze functie, bij nummertje 1 wordt er nog een andere functie aangeroepen, namelijk SelectByCategoryId(). Met deze functie haal ik alle producten op die zijn gekoppeld aan de geselecteerde categorie. Dit zit echter wel wat lastiger in elkaar dan het in eerste instantie klinkt. Een product is namelijk niet direct gekoppeld aan een categorie maar aan een externe categorie met niveau 3. Ik zal dit iets verder in het verhaal nog verder toelichten.

In figuur 7 vind u de uitwerking van de SelectByCategoryId methode. Zoals u kunt zien roept deze methode weer een andere methode aan, namelijk SelectChildExternslCategoryIds(). Ik heb de logica van het ophalen van de externe categorieën die horen bij een categorieën gescheiden. Ik verwacht namelijk dat ik dit nog wel vaker nodig zal hebben, daarnaast is de code op deze manier ook beter te overzien.

1. Ik geef een optionele parameter aan de functie mee. Products wordt op null gezet wanneer deze niet is meegegeven in de arguments.
2. Ik check met de short if notatie of de producten zijn meegestuurd in de arguments, als dit niet het geval is haal ik ze op.
3. Er wordt een functie aangeroepen die van een categorie al zijn gekoppelde externe categorieën ids ophaalt. Deze functie zal ik zo nog verder uitwerken en uitleggen.
4. Filtert de productenlijst aan de hand van de opgehaalde externe categorie ids.
5. Dit is een fallback voor wanneer er geen ids gekoppeld waren aan de gekozen categorie.

```
public IQueryable<Product> SelectByCategoryId(OfferteToolContext db, int categoryId,
    IQueryable<Product> products = null) 1
{
    // use the existing products query or create a new one if there is no existing one.
    IQueryable<Product> productsQuery = (products != null) ? (IQueryable<Product>)products : this.SelectAll(db); 2

    var externalCategoryIds = ExternalCategoryManager.Instance.SelectChildExternalCategoryIds(db, categoryId); 3

    if (externalCategoryIds != null)
    {
        productsQuery = productsQuery.Where(a => externalCategoryIds.Contains(a.ExternalCategoryId)); 4
    }
    else // The received category has no linked external categories, return an empty set.
    {
        productsQuery = productsQuery.Where(a => a.ExternalCategoryId == -1); 5
    }

    return productsQuery;
}
```

Figuur 7: Selecteer producten aan de hand van de gekozen categorie

In figuur 8 ziet u de uitwerking voor de methode SelectChildExternalCategoryIds. Ik begin bij nummer 1 met het ophalen van alle externe categorieën uit de database. Vervolgens wordt aan de hand van het meegegeven categoryId bepaald om welk niveau het gaat (nummer 2). Aan de hand van het niveau worden vervolgens weer een aantal stappen doorgelopen. Bij het zoeken op een categorie met niveau 1 worden de onderstaande stappen doorlopen.

Stap 1: Opvragen welke externe categorieën met niveau 1 er allemaal gekoppeld zijn aan de geselecteerde categorie.
Stap 2: Van de opgehaalde externe categorieën met niveau 1 opvragen welke externe categorieën met niveau 2 daar weer aan gekoppeld zijn.
Stap 3: Van alle opgehaalde externe categorieën met niveau 2 opvragen welke externe categorieën met niveau 3 daar weer aan gekoppeld zijn.
Stap 4: Nu heb je een lijst met de benodigde externe categorie ids. Alleen die met niveau 3 dus.
Stap 5: Vervolgens wordt de producten lijst doorgelopen en worden alle producten die gekoppeld zijn aan één van de opgehaalde externe categorieën opgehaald.

Er worden echter niet alleen gefilterd op een categorie met niveau 1, de eindgebruiker wil ook kunnen filteren op categorieën met niveau 2 of 3. Er veranderd verder weinig aan de stappen als we gaan zoeken op een andere niveau. Voor categorieën met niveau 2 hoef je alleen nog de subcategorieën met niveau 3 op te halen. Voor categorieën met niveau 3 hoef je helemaal geen subcategorieën meer op te halen.

```
public IQueryable<int> SelectChildExternalCategoryIds(OfferteToolContext db, int categoryId)
{
    IQueryable<int> externalCategoryIds = null;
    var externalCategories = ExternalCategoryManager.Instance.SelectAll(db);  1
    var niveau = CategoryManager.Instance.SelectById(db, categoryId).Niveau;  2

    switch (niveau)
    {
        case 1:
            externalCategoryIds = externalCategories
                .Where(a => a.Niveau == 3
                    && externalCategories.Any(b => b.Id == a.ParentExternalCategoryId && b.Niveau == 2
                    && externalCategories.Any(c => c.Id == b.ParentExternalCategoryId && c.Niveau == 1
                    && c.CategoryId == categoryId)))
                .Select(a => a.Id);
            break;
        case 2:
            externalCategoryIds = externalCategories
                .Where(a => a.Niveau == 3
                    && externalCategories.Any(b => b.Id == a.ParentExternalCategoryId && b.Niveau == 2
                    && b.CategoryId == categoryId))
                .Select(a => a.Id);
            break;
        case 3:
            externalCategoryIds = externalCategories
                .Where(a => a.Niveau == 3 && a.CategoryId == categoryId)
                .Select(a => a.Id);
            break;
    }
    return externalCategoryIds;
}
```

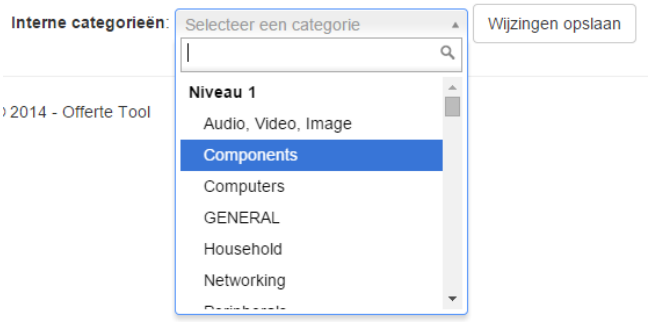
3

Figuur 8: Ophalen van de externe categorieën die zijn gekoppeld aan een categorie

5 Categorieën module

De categorie module bestaat uit twee onderdelen, het eerste onderdeel bestaat uit interne categorieën. De interne categorieën worden weergegeven in een overzicht. Verder is het mogelijk om nieuwe interne categorieën aan te maken of bestaande categorieën te wijzigen of verwijderen. Van een interne categorie wordt ook weergegeven hoeveel producten hieraan zijn gekoppeld.

Het tweede onderdeel bestaat uit het koppelen van externe categorieën aan interne categorieën. Je selecteert eerst de interne categorie waar je externe categorieën aan wilt koppelen. Dit valt te zien in figuur 9.



Figuur 9: Keuze interne categorie

Zodra de interne categorie is geselecteerd wordt een nieuw overzicht opgehaald en ingeladen met jQuery load. Aan de linkerzijden vindt u een tabel met externe categorieën die zijn gekoppeld aan de gekozen interne categorie. Aan de rechterzijden vindt u een tabel met externe categorieën die nog niet aan een interne categorie zijn gekoppeld. In figuur 10 ziet u een voorbeeld voor de interne categorie “Audio, Video, Image”. Zoals valt te zien is er op dit moment één externe categorie aangekoppeld, namelijk “Audio, Video, Image”. Deze heeft toevallig dezelfde naam. Dit komt doordat ik de interne categorieën heb aangemaakt aan de hand van de categorieën van Techdata. Vervolgens heb ik deze 1 op 1 gekoppeld met behulp van een SQL query.

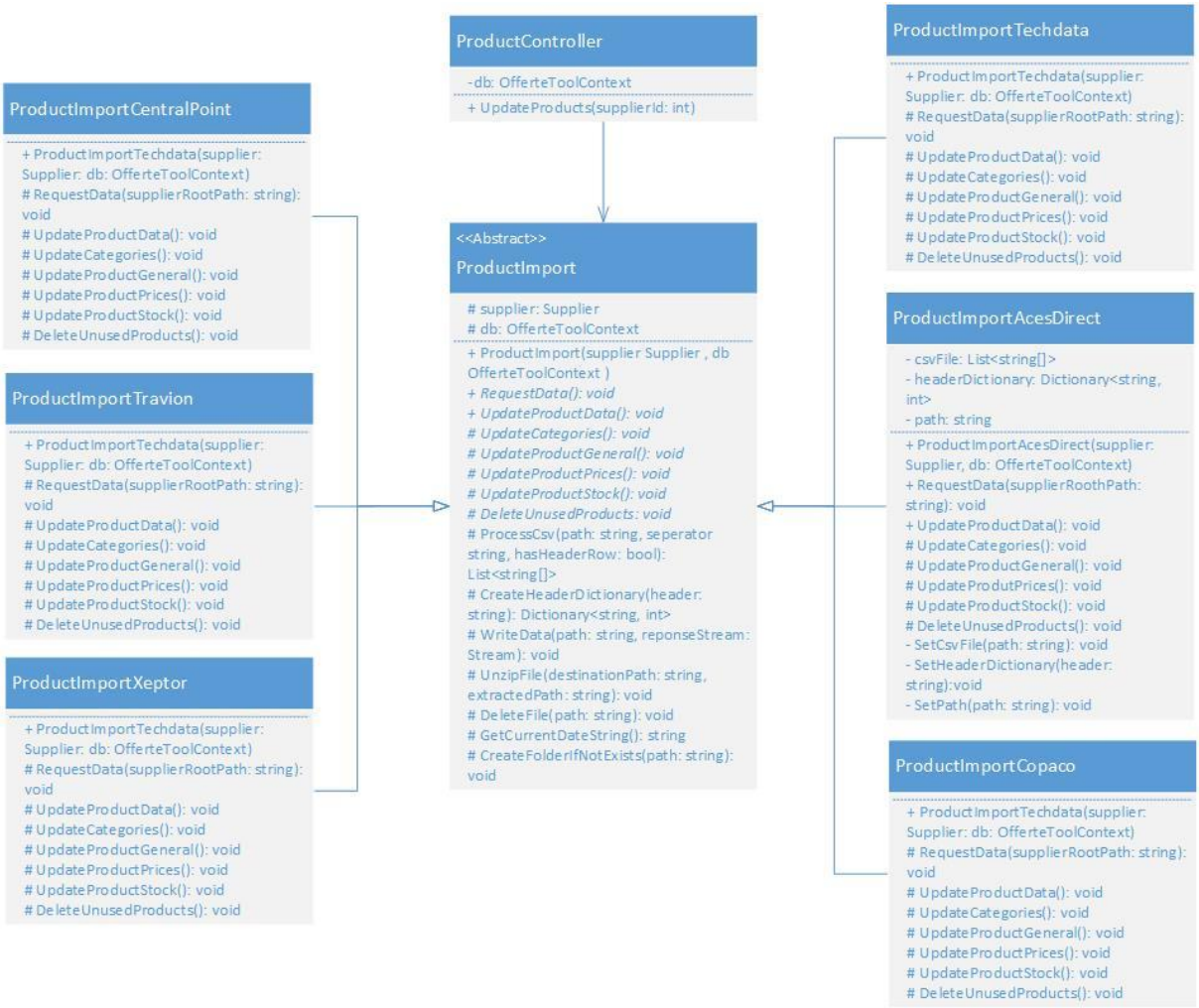
Figuur 10: Koppelen categorieën

Op dit moment is er één externe categorie gekoppeld aan de geselecteerde interne categorie. Deze koppeling kan je verwijderen door de checkbox uit te vinken en vervolgens kies je voor wijzigingen opslaan. De lijst van nog te koppelen externe categorieën is een stuk groter, ik heb voor dit voorbeeld alleen de eerste 3 weergegeven. Je kan een externe categorie koppelen aan een interne categorie door het vinkje aan te zetten en vervolgens voor wijzigingen opslaan te kiezen. Het is ook mogelijk om beide tabellen tegelijk te bewerken. Verder is voor het gemak van de gebruiker een zoekveld toegevoegd. Met dit zoekveld worden alleen de records in de tabel weergegeven die overeenkomen met de zoekterm. Het zoeken op velden in de tabel is gemaakt met javascript en jQuery. Er wordt met behulp van een keylistener gekeken of er aanslagen zijn, als dit het geval is worden alleen nog de rows in de tabel weergeven waarin de waarde overeenkomt met de zoekterm.

6 Product Import

Elke leverancier levert zijn data net weer iets anders aan, maar er zijn zeker ook overeenkomsten. Om dit alles zo goed mogelijk te implementeren heb ik ervoor gekozen om dit met behulp van een design pattern te implementeren. De twee design patterns die mij gelijk het meeste aanspraken waren het strategy pattern en de template method pattern. Na wat vergelijkingen tussen deze twee ben ik tot de conclusie gekomen dat de template method pattern de beste keuze is voor deze situatie. Ik wil namelijk verplichten dat de overervende klassen bepaalde functies bevatten, maar daarnaast wil ik ook in de base klasse een aantal functies uitwerken die vervolgens door de subklassen worden gebruikt. Het uitwerken van functies is niet mogelijk in een interface, vandaar dat de template method pattern de voorkeur krijgt.

In figuur 11 heb ik mijn uitwerking van het template method pattern weergegeven in een klassendiagram.



Figuur 11: Product import klassendiagram

De geïmplementeerde functies in de subklassen zien er allemaal anders uit qua uitwerking. Wel roepen ze allemaal dezelfde manager functies aan. De subklassen zorgen ervoor dat de data hetzelfde aankomt bij de manager, de logica staat dus in de uitwerking van de subklassen. De manager zorgt er alleen voor dat de data in de database wordt toegevoegd.

7 Koppeling Techdata

De koppeling met Techdata wordt gelegd via een FTP verbinding. Techdata Nederland gaf aan dit de beste manier was om dagelijks product gegevens bij te werken. Er was voor mij dus niet echt een keuze mogelijkheid om dit op te gaan lossen. Dit is de manier hoe Techdata Nederland informatie aanbiedt aan haar klanten. Techdata zet elke ochtend rond 06:00 een map met product gegevens voor ons klaar. Techdata levert een CSV bestand op voor product gegevens, product prijs, product voorraad, categorieën en producten die buiten gebruik zijn gesteld. We moeten dus in totaal 5 bestanden gaan inlezen om onze product data bij te werken.

Ik heb het koppelen met Techdata opgedeeld in een aantal onderdelen:

- Ophalen van de bestanden op de FTP server

- Bestanden uitlezen
- Gegevens opslaan in de database

7.1 Ophalen bestanden op de FTP server

Het ophalen van de bestanden wordt gedaan doormiddel van een FTP URL, username en wachtwoord. Doormiddel van de combinatie van deze 3 kunnen we de bestanden inzien op de FTP server van Techdata. Er wordt verbinding naar de FTP server gemaakt door middel van een FtpWebRequest. FtpWebRequest is standaard beschikbaar in asp.net MVC 5. In figuur 12 kunt u hoe ik gebruik hem gemaakt van de FtpWebRequest.

```
FtpWebRequest request = (FtpWebRequest)WebRequest.Create(new Uri(fullPath));
request.Credentials = new NetworkCredential(ftpUsername, ftpPassword);
request.UseBinary = true;
request.Method = WebRequestMethods.Ftp.DownloadFile;
```

Figuur 12: FtpWebRequest

Het bovenstaande gedeelte zorgde ervoor dat we konden verbinden met de FTP server, maar we moeten natuurlijk de data er ook nog afhalen. Dit heb ik gedaan door middel van een FtpWebReponse, ook dit object is standaard beschikbaar in asp.net MVC 5. In figuur 13 valt te zien hoe ik hiervan gebruik maak.

```
using (FtpWebResponse reponse = (FtpWebResponse)request.GetResponse())
{
    Stream responseStream = reponse.GetResponseStream();

    // read and write the data to a local directory
    WriteData(destinationPath, responseStream);
}
```

Figuur 13: FtpWebReponse

Zodra de data van het externe systeem naar ons systeem is gedownload moeten we de data nog lokaal gaan opslaan. Dit doe ik met behulp van een FileStream object.

7.2 Bestanden uitlezen

Aangezien Techdata 5 losse bestanden aanlevert met betrekking tot de productdata, heb ik ook 5 functies geschreven die de bestanden uitlezen. Een functie voor het bijwerken van de algemene product gegevens, product prijs gegevens, product voorraad gegevens, categorie gegevens en verwijderde producten. In deze functie staat het pad naar de specifieke bestanden. Vervolgens wordt aan de hand van dit pad een bestand opgehaald. Daarna wordt de CSV uitgelezen en in een list gezet. In figuur 14 kunt u zien hoe ik de bestanden uitlees en omzet na een list met array objecten. Met deze functie kan ik verschillende typen CSV uitlezen. Zoals valt te zien bij nummer 1 kan mijn methode met verschillende separators omgaan. Ook houdt de methode er rekening mee of er een header is meegegeven in de CSV of niet. De header haal ik bij nummer 2 eruit mocht deze aanwezig zijn. De list wordt vervolgens doorgegeven naar de manager en de manager zorgt ervoor dat de desbetreffende objecten aan de database worden toegevoegd.

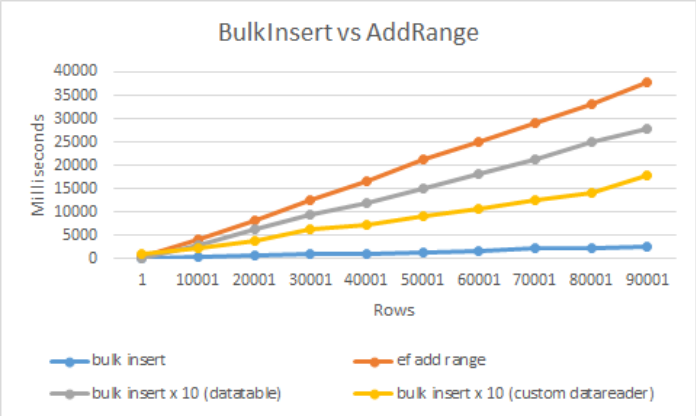
```
protected List<string[]> ProcessCsv(string path, string separtor = "\t", bool hasHeaderRow = false)
{
    List<string[]> values = new List<string[]>();

    using (TextFieldParser parser = new TextFieldParser(path))
    {
        parser.TextFieldType = FieldType.Delimited;
        parser.SetDelimiters(separtor); 1

        if (hasHeaderRow) // Skip first line if csv has a header row
        {
            parser.ReadLine();
        }

        while (!parser.EndOfData)
        {
            string[] fields = parser.ReadFields();
            values.Add(fields);
        }
    }
    return values;
}
```

1

2

Figuur 14: CSV bestanden inlezen

Tijdens het bijwerken van mij data liep ik tegen een bottleneck van het entity framework aan. Ik heb op het internet gezocht, ook daar kwam ik veel mensen tegen met hetzelfde probleem. Het entity framework is niet ideaal als het gaat om een groot aantal inserts of updates tegelijk. Tijdens het zoeken kwam ik een extensie tegen genaamd BulkInsert. BulkInsert voegt de gehele data set in een keer toe in plaats van één voor één zoals de standaard insert. In figuur 15 valt te zien wat de snelheid is van BulkInsert met vergelijking tot andere mogelijkeden.



Figuur 15: BulkInsert [https://efbulkinsert.codeplex.com/]

Zoals valt te zien in figuur 15 is de BulkInsert (blauwe lijn) vele malen sneller dan de standaard range insert (oranje lijn). Door gebruik te maken van de BulkInsert voor het toevoegen van de productdata was de functie al sneller geworden, alleen nog lang niet snel genoeg.

Er werden nog steeds één voor één update queries aangeroepen voor de product prijs en product voorraad. Om dit op te kunnen lossen heb ik mijn eerste ontwerp moeten aanpassen. De prijs en voorraad van een product wordt niet langer bewaard in een product zelf, maar in een losse tabel. Hierdoor hoef ik niet langer een product te updaten, maar er wordt voor elke prijs en voorraad wijziging een nieuwe record in de desbetreffende tabel toegevoegd. Het voordeel hiervan is dat ik ook voor de prijs en voorraad van een product gebruik kan gaan maken van de BulkInsert extensie. Nog een bijkomend voordeel is dat ik op deze manier per dag de prijs en voorraad geschiedenis kan opvragen. Dit kan ik vervolgens weer gebruiken in mijn prijsgeschiedenis en voorraadgeschiedenis feature.

Een nadeel is dat op deze manier de database wel erg snel gevuld wordt. Als we ervan uitgaan dat we 200k records in de database hebben staan, dan wordt elke dag 200k X 2 records toegevoegd. Dit betekend dat ik na een maand al 12 miljoen records in de database heb staan voor alleen de prijs en voorraad van een product. Dit gaat natuurlijk voor problemen zorgen, daarom neem ik dit mee in een performance onderzoek in iteratie 2. In iteratie 1 wil ik me vooral focussen op het resultaat en minder op de performance.

Ik voeg nu algemene product gegevens, prijs, voorraad en categorieën via de BulkInsert toe, het enige wat nog per record wordt bijgewerkt is het verwijderen van buitengebruik gestelde producten, alleen om hier veel aandacht aan te besteden vond ik zonde van mijn tijd. Dit zullen misschien enkele producten per week zijn.

Na alle aanpassingen was de functie al zeker 10 keer sneller geworden. Ik moest alleen nog een check inbouwen die controleerde of de toe te voegen record niet al bestond in de database. Hierbij kwam ik weer bij hetzelfde probleem uit, dat dus voor elk product een aanvraag naar de database gedaan zo moeten worden. Het wegschrijven van 1000 records naar de database op deze manier duurde 21 seconden. Dit is natuurlijk belachelijk lang, helemaal als je nagaat dat ik uiteindelijk nog zeker 200 keer zoveel producten erbij krijg. Hiervoor moest dus een oplossing worden gevonden. Dit heb ik opgelost door eerst een lijst met artikelnummers op te halen uit de database van de leverancier waarvan we producten aan het toevoegen zijn. Aangezien artikkelnummers voor een leverancier uniek moeten zijn kan ik doormiddel van deze lijst controleren of een product al in de database staat en hoef ik niet meer voor elke record de database aan te spreken. Door het doorvoeren van deze wijziging duurde het toevoegen en updaten van 1000 records nog maar 239 ms. Dat is een performance verbeting van ruim x100 ten opzichte van eerst.

Ditzelfde concept heb ik ook gebruikt bij het opzoeken van de bijbehorende categorie van een product. Ik haal eerst een lijst op met categoriegegevens, in plaats van voor elke record een query naar de database te doen. In figuur 16 ziet u hoe ik de data eerst ophaal, inplaats van per record een database aanvraag te doen. Hetzelfde concept heb ik op overig vervolgens op diverse plaatsen in de applicatie toegepast.

```
var productBulkList = new List<Product>();                    Ophalen artikelnummers
var articleNumbers = SelectArticleNumbersBySupplier(db, supplierId).ToArray();
var externalCategories = ExternalCategoryManager.Instance.SelectAll(db)
    .Select(a => new { id = a.Id, name = a.Name, niveau = a.Niveau, supplierId = a.SupplierId })
    .Where(a => a.niveau == 3)
    .ToArray();                    Externe categorieën met niveau 3 eenmalig ophalen
```

Figuur 16: UpdateProductGeneral, Eerst ophalen data

7.3 Gegevens opslaan in de database

Vervolgens worden de producten opgeslagen in de database met behulp van de bulkinsert. Er wordt een lijst aangemaakt voor alle producten die moeten worden toegevoegd. Zodra alle producten aan de lijst zijn toegevoegd, kan ik met slechts één regel alle producten toevoegen aan de database. Dit doe ik met het volgende stukje code: “databaseTabel.BulkInsert(List<Product>)”.

8 Koppeling AcesDirect

De tweede leverancier waar we een koppeling mee konden maken was AcesDirect. AcesDirect levert hun productdata aan ons via een URL. Zodra de URL wordt aangeroepen begin je gelijk met het ophalen van de productdata. Ze leveren de productdata in CSV en XML formaat. Ik heb beide

formaten bekeken en aan de hand van een korte inspectie is gebleken dat de CSV file een stuk sneller wordt opgehaald. De XML feed die werkt eigenlijk helemaal niet, mijn browser loopt in ieder geval altijd vast als ik deze probeer op te halen. Ik heb er mede hierdoor voor gekozen om gekozen om gebruik te maken van de CSV file. Daarnaast kan ik nu veel bestaande code hergebruiken aangezien de koppeling van Techdata ook gebruik maakt van bestanden in CSV formaat.

De feed die AcesDirect in eerste instantie voor ons klaar heeft gezet bevat producten uit hun gehele assortiment. Dit zijn in totaal ongeveer 500.000 producten, die CSV file is dan ook erg groot qua formaat, ruim 150mb. De grootte van de CSV file zorgt ervoor dat het ophalen van de productdata wel enige tijd kan duren, meestal duurt het opvragen een minuut lokaal. Dit is geen ideale situatie, we hebben dan ook gevraagd of het mogelijk is om alleen de benodigde productdata op te vragen. AcesDirect had aangegeven dat dit tot de mogelijkheid behoorde, het CSV bestandje wordt hierdoor aanzienlijk kleiner. Dit zorgt er gelijk ook voor dat het ophalen van de CSV binnen enkele seconden gebeurd.

Het ophalen van de productdata van AcesDirect wordt gedaan met een WebClient object. In figuur 17 kunt u zien hoe ik dit heb uitgewerkt.

```
CreateFolderIfNotExists(supplierRootPath);

var url = Supplier.RequestUrl;
var destinationPath = supplierRootPath + "/" + GetCurrentDateString() + ".csv";

if(!File.Exists(destinationPath)){
    WebClient wc = new WebClient();
    wc.DownloadFile(url, destinationPath);
}
```

Figuur 17: Download productdata AcesDirect

De data wordt opgehaald in gelijk weggeschreven naar een lokale directory. Zodra lokaal het bestandje is weggeschreven kan deze worden uitgelezen. In tegenstelling tot Techdata levert AcesDirect al hun productdata in één CSV bestand. Dit is op zich geen probleem, hier had ik binnen mijn ontwerp al rekening mee gehouden. Wat wel voor een probleem zorgde was het feit dat AcesDirect product had gekoppeld in hun CSV file aan Categorie namen en niet aan categorie Ids. Hierdoor heb ik de uitwerking voor het toewijzen van een categorie aan een product iets moeten aanpassen. Ik heb een aanpassing gemaakt die de bijbehorende categorie nu altijd opzoekt aan de hand van een categorie naam. In het geval van Techdata moet de naam wel eerst worden opgehaald aan de hand van de meegegeven categorieId in de productenlijst.

De functies voor het aanmaken van de Categorieën heb ik ook enigszins moeten bijwerken, in het geval van AcesDirect zijn er namelijk maar 2 categorie niveaus. Het laagste niveau voor AcesDirect is ook gewoon 3, AcesDirect heeft alleen geen niveau 1 categorieën. Ook moest ik er rekening mee houden dat de categorieën van Techdata wel een zelf een id meegeven en AcesDirect niet. Dit id heb ik nog wel nodig voor Techdata, aangezien ik anders niet kan opvragen welke id bij welke naam hoort tijdens het toewijzen van een categorie aan een product.

De functies voor het bijwerken van de product prijs en voorraad konden zonden aanpassing weer worden gebruikt voor AcesDirect.

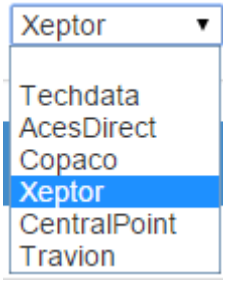
9 Prijs en voorraad geschiedenis

De prijs en voorraad geschiedenis zou ik eigenlijk pas in de tweede iteraties implementeren, maar door de keuzes in mijn technisch ontwerp was dit qua backend werkzaamheden gemakkelijk te implementeren. De geschiedenis hield ik namelijk voor zowel de prijs en voorraad al bij in mijn database. Het enige wat ik nog moest doen is een geschikte grafiek zoeken om de data in weer te kunnen geven. Zoals u al eerder had kunnen lezen maak voor de grafieken binnen de applicatie gebruik van amcharts.

10 SELECT 2

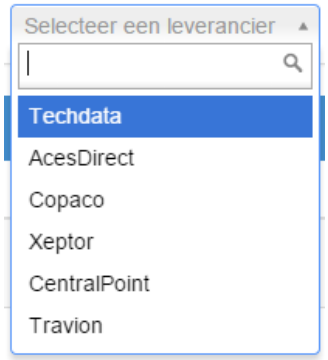
Voor de select boxes binnen mijn applicatie maak ik gebruik van de bootstrap select2 plug-in. Dit zorgt ervoor dat de select box mooier wordt weergeven, maar belangrijker is dat je ook kan zoeken binnen de select box. Om select2 binnen een project te gebruiken moeten eerst een aantal javascript en css bestandjes worden gedownload. Zodra alle documentatie goed is ingeladen kun je met slechts één javascript regel een standaard saaie select box omzetten in een select2 select box. Een flinke verbetering vind ik zelf.

Een normale select box:



Figuur 18: Regular selectbox

Een select2 select box:



Figuur 19: Select2 selectbox

11 Unit tests

Ik had voorafgaand aan de iteratie mijzelf voorgesteld dat ik alle belangrijke functionaliteiten zou gaan testen met een unit test. Tijdens het schrijven van de unit test kwam ik er echter achter dat dit wat lastiger ging worden dan ik in eerste instantie had gedacht. Het grootste gedeelte van mijn applicatie is gebaseerd op het ophalen en aanpassen van data uit de database. Daarnaast werk ik ook nog eens met het entity framework. Dit alles maakt het lastig om bepaalde functionaliteiten te testen aangezien er meestal een database aanvraag wordt gedaan. Ik vind het zelf niet handig als mijn database wordt bevuild met testdata. Ik denk zelf dat de beste oplossing hiervoor is een testdatabase opzetten, die qua data er ongeveer hetzelfde uitziet als de live database. Anders heeft het testen op performance bijvoorbeeld weinig zin. In iteratie 1 test ik alleen de functies die geen gebruik maken van de database. De functies die wel gebruik maken van de database neem ik mee in de tests van iteratie 2.

Een van de functionaliteiten die ik wel heb getest in iteratie 1 is testen of het ophalen van product gegevens van de servers van de leveranciers goed gaat. Voor de beiden gemaakte koppeling heb ik een test opgesteld die probeert om data van de server van de leverancier op te halen. Vervolgens wordt er gekeken of de bestanden ook echt worden weggeschreven naar onze lokale directory. In

figuur 20 vindt u de unit test voor het controleren op opgehaalde product data voor de leverancier Techdata. Ik vraag eerst de bijbehorende leverancier op, daarna haal ik een string op met een referentie naar de import folder van de leverancier. Voordat ik een data aanvraag ga doen controleer ik of de data al niet is opgehaald, mocht de data al zijn opgehaald zou de test altijd slagen en dat is niet de bedoeling. Als de data al was opgehaald wordt deze eerst verwijderd. Vervolgens wordt de data opgehaald en gekeken of deze lokaal is weggeschreven. De test voor AcesDirect ziet er ongeveer hetzelfde uit, alleen wordt er gekeken of er een file is aangemaakt in plaats van een directory. Daarnaast wordt er natuurlijk ook een ander import object aangemaakt.

```
[TestMethod]
public void TestRequestDataTechdata_FilesCreated()
{
    var supplier = GetSupplier();
    var supplierImportPath = GetImportPath(supplier.Name);
    var filePath = supplierImportPath + "\\" + DateTime.Now.ToString("yyyyMMdd");

    // Controleer eerst of de files al niet zijn opgehaald, zoja verwijder deze.
    if (Directory.Exists(filePath))
    {
        Directory.Delete(filePath, true);
    }

    ProductImportTechdata productImportTechdata = new ProductImportTechdata(supplier, new OfferteToolContext
    productImportTechdata.RequestData(supplierImportPath);
    var result = Directory.Exists(filePath);

    Assert.IsTrue(result, "Fout bij het downloaden van de productdata van Techdata");
}
```

Figuur 20: Unit test voor het controleren van de verbinding met Techdata

Ik heb ook nog tests gemaakt voor functies in de abstracte klassen. Om de abstracte klassen te kunnen testen heb ik binnen mijn testproject een nieuwe klas aangemaakt die overerft van de abstracte ProductImport klas. Deze test klas heb ik aangemaakt omdat ik van een abstracte klas geen object kan aanmaken. Wat ik wel had kunnen doen was een object aanmaken van één van de subklassen. Alleen ik vond dit zelf geen netten oplossing, het aanmaken van een testklassen die overerft van de abstracte klas en de functies volgens teruggeeft met base.functie() vond ik de beste oplossing. In figuur 21 kunt u zien wat ik bedoel met het teruggeven van base.functie().

```
public List<string[]> ProcessCsv(string path, string separtor = "\t", bool hasHeaderRow = false)
{
    return base.ProcessCsv(path, separtor, hasHeaderRow);
}
```

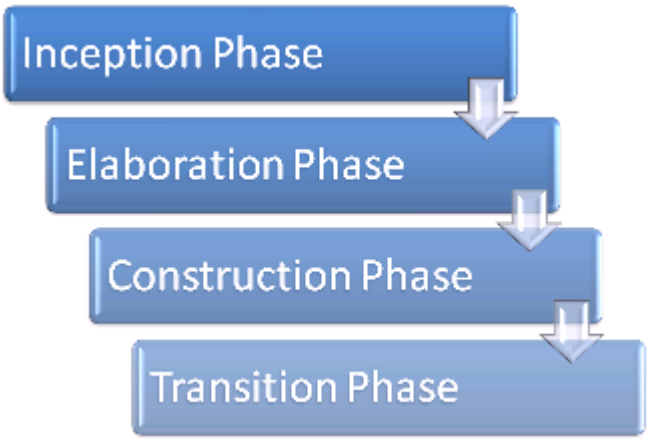
Figuur 21: Return base functie van abstracte klas voorbeeld

Ik heb testen geschreven voor het uitlezen van CSV bestandje. Er wordt onder andere gecontroleerd wat er gebeurd wanneer het ingelezen bestand niet van een CSV formaat is. Ook worden verschillende scheidingstekens getest en wordt er getest met en zonder een header regel.

Daarnaast heb ik nog enkele andere kleinere functies getest die me helpen bij het ophalen en toevoegen van data aan de database na het importeren van productgegevens van een leverancier.

Construction rapport 2

Hardware offerte tool



Auteur: Donny Roelen

Datum: 08-1-2015

Versie 1.1

Inhoudsopgave

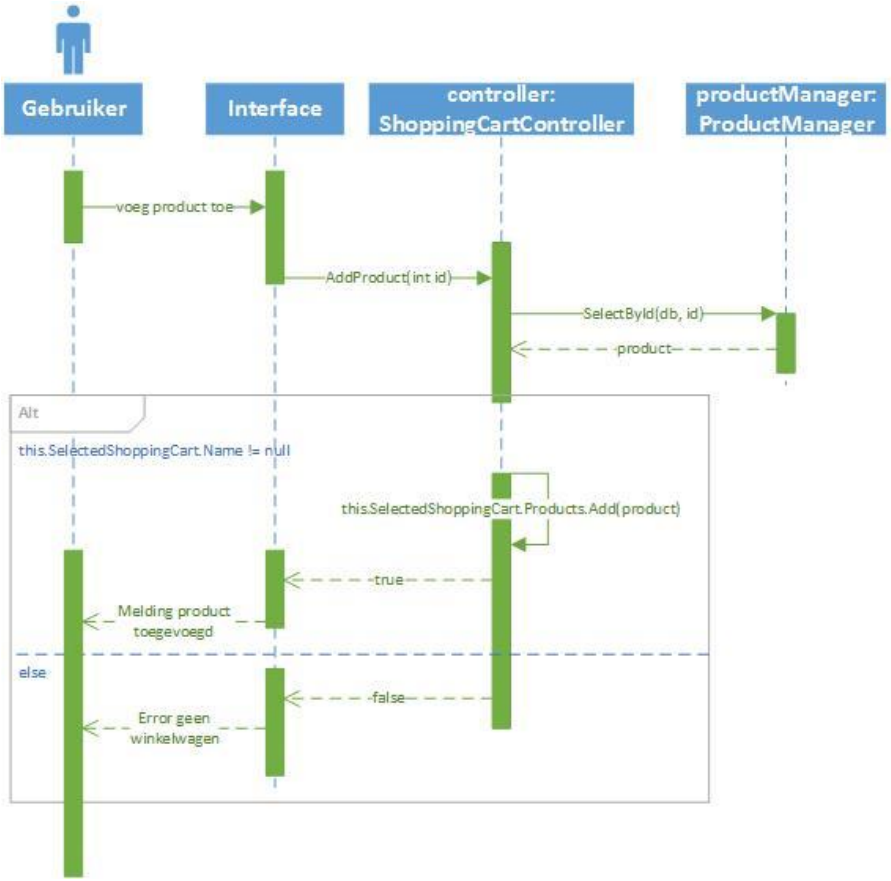
1 Inleiding

In dit hoofdstuk worden de artefacten die tijdens de construction fase van iteratie 2 aan bod zijn gekomen besproken. Keuzes met betrekking tot de bouw van de software van de eerste iteratie zullen in dit document worden besproken. De tweede iteratie bestaat uit aanpassingen aan de hand van feedback van de opdrachtgever en uitbreiding op de basis versie.

2 Bijwerken huidige documentatie

Ik ben iteratie 2 begonnen met het bijwerken van de documentatie. Aan de hand van een gegeven demo aan de opdrachtgever zijn er weer nieuwe wensen en eisen naar voren gekomen. De nieuwe wensen en eisen heb ik verwerkt in zowel de inception als de elaboration rapporten.

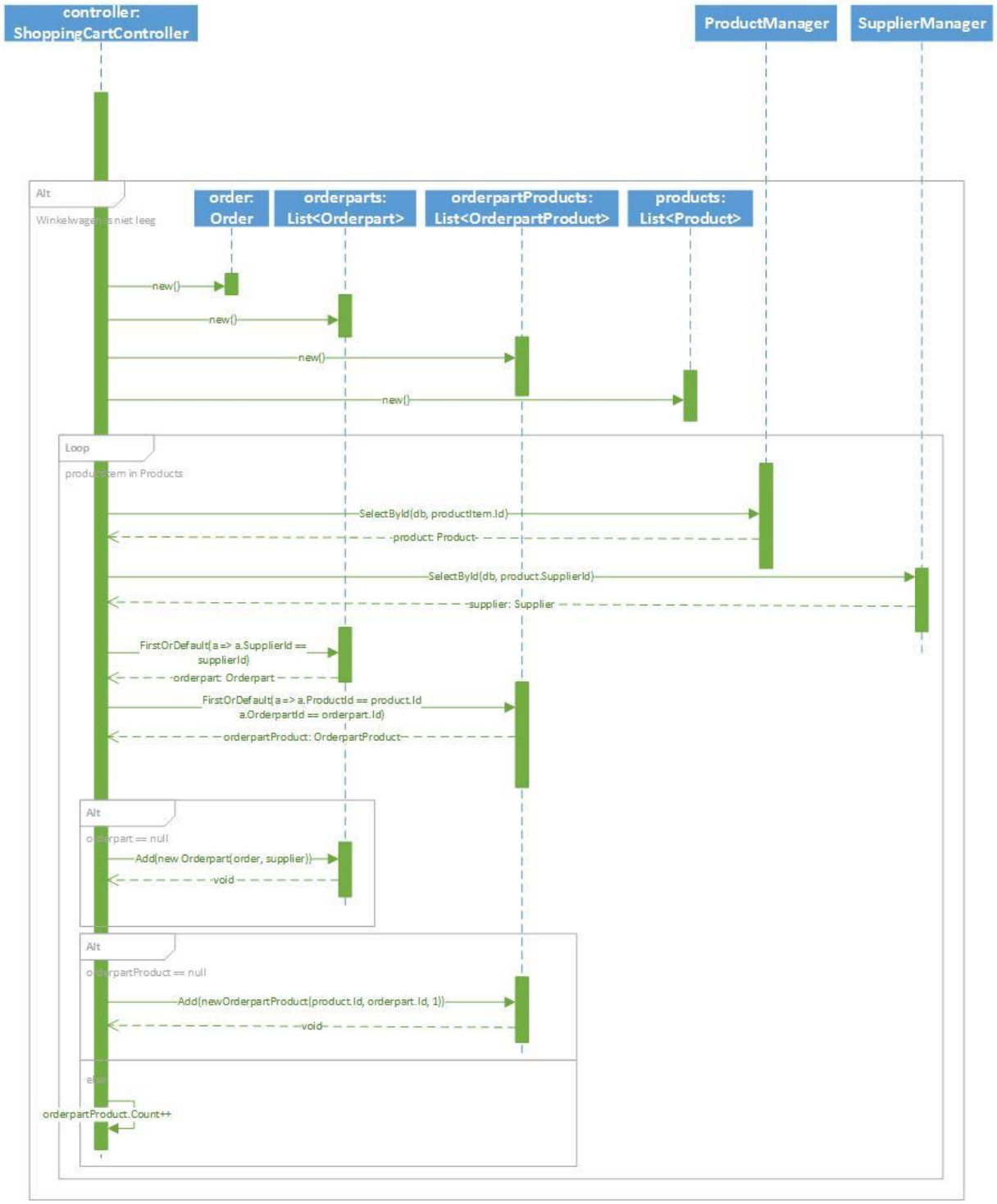
Daarnaast heb twee nieuwe sequentie diagrammen aangemaakt welke ik hieronder ga bespreken. De sequentie diagrammen waren gebaseerd op mijn eerste ontwerp van de winkelwagen module. Deze is ondertussen enigszins gewijzigd, alleen ik had geen tijd meer op de diagrammen opnieuw te maken. De eerste sequentie diagram beschrijft het toevoegen van een product aan de winkelwagen.



Figuur 1: Sequentie diagram - product toevoegen aan order

De gebruiker kiest ervoor om een product toe te voegen aan de winkelwagen. Vervolgens wordt er gecontroleerd of er een winkelwagen is aangemaakt. Als dit het geval is wordt de product toegevoegd aan de winkelwagen en wordt er een melding getoond dat de product is toegevoegd. Wanneer er nog geen winkelwagen bestond wordt er een foutmelding getoond met de melding dat er nog geen winkelwagen is aangemaakt.

De tweede sequentie diagram beschrijft het aanmaken van een order van een winkelwagen. Ik heb maar ervoor gekozen om maar een deel van de sequentie diagram te omschrijven, omdat de gehele sequentie diagram erg groot was. Het deel wat ik eruit het gelaten is de interactie van de gebruiker met de interface en het opslaan van de data in de database via de manager klassen.



Figuur 2: Sequentie diagram - aanmaken order

Allereerst wordt er gecontroleerd of er wel producten aanwezig zijn in de winkelwagen. Als dit het geval is gaan we verder. Vervolgens worden er een aantal nieuwe lege objecten aangemaakt. Nu

loop ik over alle aanwezige producten in de winkelwagen en per product worden de volgende acties uitgevoerd:

1. Eerst haal de betreffende product op uit de database aan de hand van zijn id
2. Daarna haal ik de bijbehorende leverancier op van de product.
3. Vervolgens haal ik een orderpart op die hoort bij de leverancier.
4. Daarna haal ik de orderpartProduct op uit de tussentabel van product en orderpart
5. Ik controleer vervolgens of de orderpart niet null is, als dit wel het geval is worden er een nieuwe orderpart aangemaakt.
6. Daarna controleer of de orderpartProduct al bestond. Als deze al bestond wordt de count van deze met 1 opgehoogd. Als deze nog niet bestond wordt er een nieuwe object aangemaakt.
7. Vervolgens wordt alles opgeslagen in de database, maar dit deel paste er dus niet meer op.

3 Order en ShoppingCart module

De eerste functionaliteit die ik in iteratie 2 ben gaan bouwen was die van de winkelwagen en order module. De gebruiker maakt eerst een winkelwagen aan, vervolgens worden hier producten aan toegevoegd. Zodra de gebruiker klaar is met winkelen kiest hij ervoor om de winkelwagen om te zetten naar order. De producten uit de actieve winkelwagen worden omgezet naar een order en de winkelwagen wordt vervolgens weer leeg gemaakt. Het is niet meer mogelijk om een order te wijzigen, deze is definitief.

In eerste instantie had ik er voor gekozen om een winkelwagen op te slaan in het huidige sessie object. Zodra de sessie is verlopen worden de aangemaakte winkelwagens dus ook weer verwijderd. Hiervoor maakte ik gebruik van het standaard aanwezige Session object. Ik roep echter niet door mijn hele applicaties de sessie direct aan, maar doet die via een ShoppingCart object in de BaseController. Deze constructie heb ik in figuur 3 weergegeven.

```
public ShoppingCart SelectedShoppingCart
{
    get
    {
        if (this.Session["SelectedShoppingCart"] == null)
        {
            this.Session["SelectedShoppingCart"] = new ShoppingCart();
        }

        return (ShoppingCart)this.Session["SelectedShoppingCart"];
    }
    set
    {
        this.Session["SelectedShoppingCart"] = value;
    }
}
```

Figuur 3: Ophalen huidige winkelwagen

Alleen na een feedback moment met de opdrachtgever kwam ik tot de conclusie dat dit niet de juiste oplossing was. De opdrachtgever wilde namelijk een winkelwagen de volgende dag ook nog kunnen bijwerken. Hierdoor moest mijn ontwerp weer enigszins worden aangepast.

Om ervoor te zorgen dat een winkelwagen ook wordt bewaard voor later gebruik heb ik wijzigingen in mijn ontwerp en code moet maken. De klasse shoppingcart heb ik verwijderd en aan de klasse order heb ik een nieuwe attribuut toegevoegd die bepaald of het order of een winkelwagen is. De klasse order is een winkelwagen wanneer de nieuwe attribuut false en een order wanneer de nieuwe

attribuut true is. Producten worden nu direct toegevoegd aan een order en dus ook opgeslagen in de database voor later gebruik.

Zoals al eerder was vermeld wilde de opdrachtgever met meerdere winkelwagens tegelijk kunnen werken. Ik heb het zo gemaakt dat alle aangemaakte winkelwagentjes in een selectlist terecht komen. Zodra uit deze selectlist een winkelwagen wordt geselecteerd haal ik met jQuery een nieuwe view op waarin de geselecteerde winkelwagen wordt getoond. In figuur 4 kunt u zien hoe ik dit heb gedaan. Ik zal per nummer even kort uitleggen waar het voor is en wat ik ermee doe.

1. Nummer 1 is de target element waarin de opgehaalde view in wordt weergegeven.
2. De methode die wordt uitgevoerd
3. Welk element uit de op te halen view moet er worden ingeladen
4. De data die ik meestuur met de methode
5. Een zelf geschreven functie die zorgt dat je een product aantal kan bijwerken. Ook deze functie werkt met jQuery. In dit geval maak ik gebruik van een ajax request om de actieve winkelwagen bij te werken.

```
function LoadShoppingCartProducts(name)
{
    $("#cartGrid").load("/ShoppingCart/Detail .row", { name: name }, function () {
        BindOnCountItemsChange();
    });
}
```

Figuur 4: Ophalen winkelwagen

Tijdens het omzetten van de winkelwagen naar een order liep ik tegen een probleem aan. Het was namelijk niet mogelijk om één product meer dan 1 keer aan een orderpart toe te voegen. Dit kwam doordat de tussentabel Orderpart_Product automatisch was gegeneerd door het entityframework en de combinatie van orderpartId en productId moest uniek zijn. Dit heb ik opgelost door een model object van deze tussentabel aan te maken met een extra attribuut aantal. Wanneer een product vaker voorkomt in een orderpart verhoog ik gewoon het aantal.

Van een order wordt per orderpart weergegeven welke producten daarin voorkomen. De opdrachtgever kan zo zien welke producten er bij welke leverancier moeten worden besteld. Er is uiteraard ook per leverancier een totaal prijs. Het is ook mogelijk om een order te downloaden als pdf bestand. Voor het omzetten van een html pagina naar een pdf document heb ik gebruik gemaakt van de externe libary Rotativa. Rotativa is een gratis .NET libary voor het omzetten van html views naar pdf documenten.

4 Externe categorieën kiezen voor producten import

In de huidige situatie werden alle producten die aangeleverd worden door de leverancier ingelezen in opgeslagen in onze database. Hierdoor krijgen we ook producten in ons systeem waarvan we eigenlijk nooit gebruik van zullen maken. De opdrachtgever had tijdens een feedback moment aangegeven dat hij graag zelf wilde bepalen van welke externe categorieën er producten worden geïmporteerd.

4.1 Overzicht externe categorieën

Ik ben begonnen met het maken van een nieuwe overzicht. In dit overzicht worden alle externe categorieën per leverancier weergegeven. Hier kan de gebruiker per externe categorie aangeven of

deze moet worden genomen in de import. Dit kan voor alle niveaus, als de gebruiker kiest om een externe categorie van niveau 1 mee te nemen worden automatisch alle sub externe categorieën ook meegenomen in de import. Hetzelfde geld bij een niveau 2 externe categorie. Bij een externe categorie van niveau 3 wordt alleen de gekozen externe categorie meegenomen. Het selecteren van deze externe categorieën gaat door middel van een html checkbox. Ik heb een “on click” event listener gezet op deze checkbox. Zodra deze wordt geactiveerd word er een ajax request uitgevoerd. De ajax request wordt in figuur 5 weergegeven. Daarnaast kunt u hier gelijk zien hoe ik gebruik maak van de sweet alert libary. In dit geval ging het enkel om een normale alert zonder confirm knoppen.

```
function SaveExternalCategory(data, checked)
{
    $.ajax({
        url: "/ExternalCategory/Save/",
        type: "POST",
        data: data
    }).done(function () {
        swal({
            title: "Externe categorie opgeslagen.",
            text: checked
                ? "Producten uit deze categorie worden nu meegenomen in de import."
                : "Producten uit deze categorie worden niet meer meegenomen in de import.",
            type: "success",
            timer: 5000,
            allowOutsideClick: true
        });
    });
}
```

Aan de hand van de short if notatie de tekst voor de alert zetten.

Sweet alert success box

Figuur 5: Externe categorie meenemen in import

Nu we weten welke externe categorieën de gebruiker mee wilt nemen in de import moeten we hier ook nog rekening mee gaan houden. Dit uitwerking hiervan verschilt per leverancier, aangezien leveranciers data verschillend aanleveren.

4.2 Techdata import bijwerken

Techdata levert al haar data aan in losse bestandjes, dat is voor dit onderdeel niet heel erg handig. Ik heb een nieuwe functie toegevoegd die de producten lijst filter aan de hand van de gekozen externe categorieën. Het filteren van de lijst doe ik met behulp van een LINQ query. Hierdoor worden alleen de producten geïmporteerd waarvan de bijbehorende externe categorie mag worden geïmporteerd. Voor de prijs en voorraad heb ik geen wijziging door hoeven voeren. Ik had namelijk al een check staan die controleerde of de bijbehorende product bestond in de database. Als dit niet het geval was werd de prijs en voorraad ok niet toegevoegd.

4.3 AcesDirect import bijwerken

AcesDirect levert al haar data aan in één CSV bestand. Dit was in dit geval erg handig, hierdoor kan ik namelijk in 1x de producten, prijzen en voorraad filteren aan de hand van de gekozen externe categorieën. Ook hier filter ik de lijst door middel van een LINQ query.

5 Koppeling met Copaco

De koppeling met Copaco heb ik nog niet kunnen realiseren. Dit kwam doordat Copaco zelf moeilijk deed over het vrijgeven van hun productdata. Copaco wilt namelijk niet dat wij deze data publiek beschikbaar stellen. De opdrachtgever heeft meerdere malen contact met Copaco gehad hierover, alleen we hebben tot op heden geen data van ze ontvangen.

Ik had al wel voorbereidingen getroffen voor de import van Copaco. In de toekomst zou dit dus binnen een dag gekoppeld kunnen worden. De tijd die vrij kwam door het niet aangeleverd krijgen van de productdata heb ik vooral gebruikt bij mijn performance onderzoek.

6 Promo prijzen module

De promotieprijzen module bestaat uit twee onderdelen. Het eerste onderdeel hoort bij het producten overzicht. Er is een extra filter toegevoegd die alleen producten weergeeft waarvan een promotieprijs bekend is. Daarnaast worden de prijzen van producten waarvan een promo bekend is ook vetgedrukt en in het groen weergegeven.

7 Performance

Zoals ik al eerder had aangegeven zou ik nog een “performance onderzoekje” houden. Dit onderzoek richt zich op de performance binnen het producten overzicht. Ik merkte dat het zoeken naarmate het aantal producten groeide steeds trager werd. Het ophalen van alle data gaat gewoon snel, alleen zodra er ook gefilterd moest worden werd het overzicht aanzienlijk trager. Zoeken op een tekst met meerdere worden kon soms wel 20 seconden duren en dat was naar mijn mening onacceptabel.

Na he bestuderen van de queries kwam ik er al gauw achter er veel performance verloren ging op SQL niveau. Het uitvoeren van een zoekactie met een “%tekst%” wildcard kost SQL enkele seconden. Dit komt doordat SQL 4 kolommen keer het aantal producten in de database moet doorzoeken.

7.1 Blitzindex

Een collega vertelde mij over blitz_index, je kan met behulp van een blitz_index script zien waar binnen je database problemen zijn met bijvoorbeeld ongebruikte indexen of juist ontbrekende indexen. Ik heb deze scripts op mijn database gedraaid en geanalyseerd, hieruit kwamen eigenlijk maar 2 problemen naar voren. Er zat een index op productnaam die nooit werd gebruikt en artikelnummer was van het type string, terwijl dat altijd een integer waarde bevatten. De index heb ik verwijderd van de productnaam en artikelnummer heb ik omgezet naar een integer waarde. Verder heb ik geen problemen kunnen ondervinden en de performance boost door deze wijzigingen was dan ook niet echt merkbaar.

7.2 FULLTEXT SQL

Ik weet nu dat het probleem hem niet zozeer zit in het wel of niet hebben van indexen, maar het probleem zit ergens anders. Doordat ik gebruik maak van een full wildcard search kan SQL geen gebruik maken van de indexen. Dit moet ik dus op een andere manier oplossen.

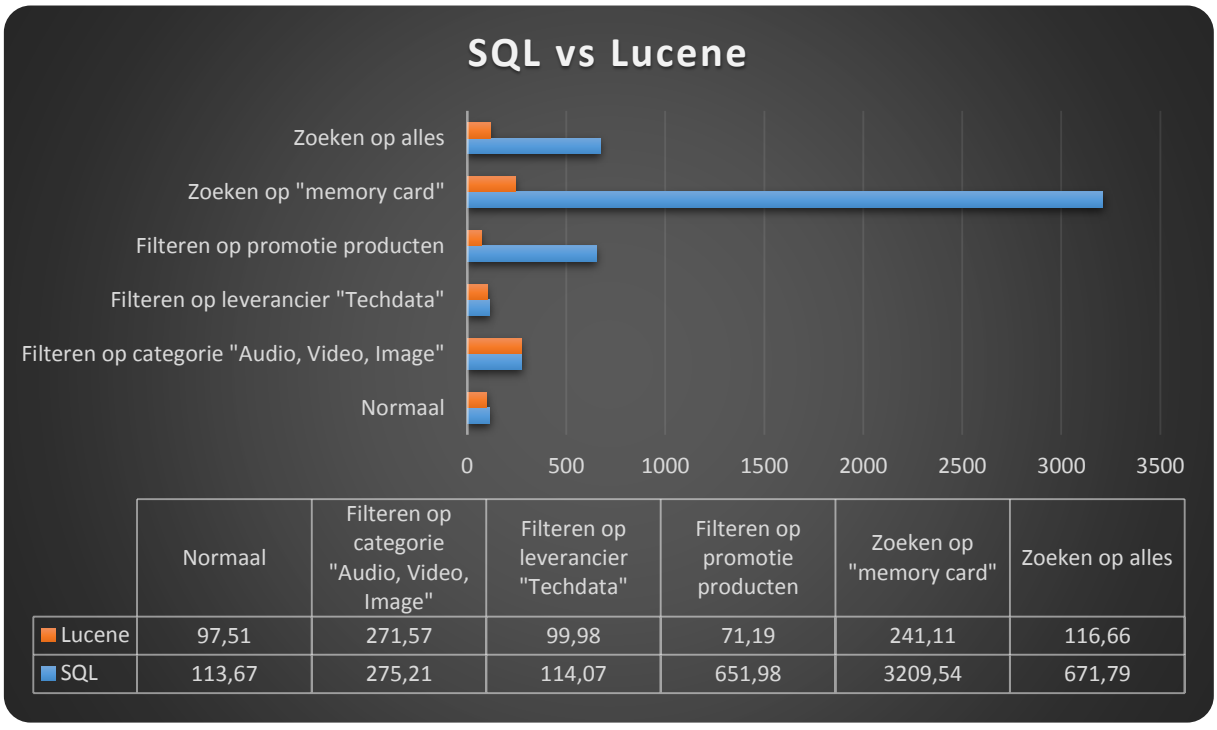
Ik heb verder onderzoek gedaan wat er nog meer voor mogelijkheden zijn en zo kwam ik FULLTEXT SQL tegen. Dit zou vele male sneller moeten zijn dan de standaard LIKE query en het is al beschikbaar vanaf SQL server 2005. Om dit te gebruiken moet je zogenaamde FULLTEXT indexen aanmaken voor de kolommen waarop je wil gaan zoeken. Hierbij liep ik al gauw tegen een probleem aan, een FULLTEXT index moet altijd uniek en mag nooit NULL zijn. Het uniek maken van de kolommen is voor mij niet mogelijk. Een product kan namelijk meerdere keren bestaan met dezelfde naam of frabrikantcode, alleen dan van een andere leverancier. Ik kan dus helaas geen gebruik maken van FULLTEXT SQL.

7.3 Lucene search

Tijdens mijn zoektocht naar een oplossing ben ik meerdere keren de term Lucene tegengekomen. Lucene is opensource zoekengine die geschreven is in Java. Lucene werkt volgens tekstbestandjes met indexen en claimt vele malen sneller te zijn dan de standaard SQL search. Daarnaast is Lucene ook beschikbaar voor het .NET framework. Na het doorlezen van verschillende artikelen over Lucene heb ik genoeg informatie gevonden om het zelf toe te kunnen passen binnen de applicatie.

Om gebruik te maken van Lucene heb ik eerst een aantal kleine wijzigingen moeten doorvoeren, Lucene kan namelijk geen NULL waardes in een tekst document opslaan. Verder heb ik special voor de Lucene search een nieuw model object aangemaakt, namelijk ProductLucene. Een ProductLucene object wordt niet opgeslagen in de database en bevat informatie die nodig is voor het zoeken en tonen van het productenoverzicht. Een ProductLucene object wordt aangemaakt aan de hand van een Product object, het grootste verschil is echter dat een ProductLucene object alleen de laatste prijs een voorraad bevat. Ik kan namelijk geen relaties opslaan in de Lucene tekstbestanden. Het is namelijk geen relationele database.

Ik heb een vergelijking gemaakt qua performance met het filteren op producten op SQL niveau en op Lucene niveau. Wat nog wel belangrijk is om te vermelden is dat de SQL filter was gebaseerd op 46.000 producten en de Lucene filter op 106.000 producten. De resultaten van deze vergelijking wordt weergegeven in figuur 6. Ik heb deze vergelijking gemaakt in mijn controller met behulp van de StopWatch klasse. De resultaten zijn inclusief het aanmaken van enkele viewmodel objecten die worden getoond in het producten overzicht. Zoals valt te zien in figuur 37 is alles eigenlijk in het voordeel van de Lucene search. Vooral bij het zoeken op een stuk tekst valt het enorme verschil in performance op, de Lucene search is op dit vlak zeker 10 keer zo snel.



	Normaal	Filteren op categorie "Audio, Video, Image"	Filteren op leverancier "Techdata"	Filteren op promotie producten	Zoeken op "memory card"	Zoeken op alles
Lucene	97,51	271,57	99,98	71,19	241,11	116,66
SQL	113,67	275,21	114,07	651,98	3209,54	671,79

Figuur 6: SQL vs Lucene search

Waar het zoeken met Lucene veel tijd wint verloor ik weer tijd bij het aanmaken van de indexen. Ik moet namelijk bij elke producten import de indexen opnieuw aanmaken. De prijs of voorraad kan namelijk gewijzigd zijn, daarnaast kunnen er ook nieuwe producten zijn bijgekomen. Het aanmaken

van de indexen zelf is echter niet het probleem, maar het aanmaken van de ProductLucene objecten. Ik maak namelijk voor elk product object aanwezig in de database een productLucene object aan. Het ophalen van alle relaties van een product kost gewoon wat tijd. Ik heb de query wil iets sneller kunnen maken door lazy loading te overriden met de include query, maar alsnog kost het aanmaken van al deze objecten enkele minuten. In figuur 7 kunt u zien wat ik hiermee bedoel. Door gebruik te maken van de include query wordt de gehele relatie onmiddellijk opgehaald in plaats van alleen de benodigde onderdelen. Dit scheelt uiteindelijk qua performance doordat het entity framework alles in één query ophaalt in plaats van telkens delen in kleinere queries.

```
var luceneProducts = ProductManager.Instance.SelectAll(db)
    .OrderBy(a => a.Id)
    .Include(a => a.ProductPrices)
    .Include(a => a.ProductStock)
    .ToList();
```

Figuur 7: Include query

8 User module

De gebruiker module is nog redelijk beperkt. Het is mogelijk gebruikers aan te maken en ermee in te loggen. Wachtwoord vergeten en wachtwoord wijzigen zit er ook in. Verder heb ik alvast een opgezet gemaakt voor de relatie user – order. Wanneer er een gebruiker is ingelogd wordt deze ook gelijk gekoppeld zodra er een order wordt aangemaakt. Maar voor nu blijft het hier ook bij, de opdrachtgever vond user module op dit moment nog niet belangrijk, voorlopig wordt de applicatie toch maar door één gebruiker gebruikt. De user module is leuk voor in de toekomst, maar valt op dit moment nog buiten de scope van de opdracht.

9 Unit test scripts

Voor de applicatie heb ik diverse unit test scripts geschreven. De keuze welke functionaliteiten ik zou gaan testen heb ik in eerste instantie gemaakt aan de hand van mijn opgestelde requirements. Ik ben alle must en should have requirements doorgelopen en heb gekeken welke requirements zinnig zijn om te testen. Daarnaast heb ik zelf nog gekeken wat vind ik daarnaast nog belangrijk om te testen. Hieruit zijn uiteindelijk tussen de 40 en 50 testscripts uitgekomen.

Het testen op zich was best een uitdaging, aangezien het grootste deel van de applicatie afhankelijk is van de data uit de database. Hier heb ik wel een oplossing voor kunnen vinden namelijk Effort in memory database. De naam zegt het eigenlijk al, met behulp van Effort kan je een tijdelijk een database aanmaken in je geheugen voor de unit test scripts. Helaas werkt Effort niet in combinatie met de BulkInsert module, het bulk inserten heb ik dus ook niet kunnen testen. Wel heb ik de logica die daaraan vooraf gaat getest.

Het aanmaken van de tijdelijke database in het geheugen is redelijk simpel. Wat heb ik gedaan is een 2de constructor aangemaakt voor me context klassen, deze 2de constructor heeft als parameter een connectie string. Vervolgens maak ik een connectie aan via effort, dit valt te zien in de eerste regel van de functie CreateTestDatabase in figuur 8. Zodra ik deze connectie meegeef aan mijn context constructor is de context klassen gelinkt aan de in memory database in plaats van de fysieke database zoals in de werkelijke applicatie. Hierdoor kan ik binnen mijn unit test scripts ook functies testen die gebruik maken van de database.

```
[AssemblyInitialize]
public static void CreateTestDatabase(TestContext context)
{
    DbConnection connection = Effort.DbConnectionFactory.CreateTransient();
    Db = new OfferteToolContext(connection);

    CreateTestSuppliers();
    CreateTestCategories();
    CreateTestExternalCategories();
}
```

Figuur 8: In memory connectie aanmaken

Zoals u kunt zien maak ik gebruik van een keyword “[AssemblyInitialize]”, deze functie wordt één keer uitgevoerd voordat alle test scripts worden gedraaid. Ik maak hiervan gebruik, omdat ik voor sommige functionaliteiten testdata nodig heb in mij database. Hierdoor hoef ik het maar éénmalig aan te maken in niet telkens handmatig in de test scripts zelf. Naast de “[AssemblyInitialize]” maak ik ook nog gebruik van het keyword “[ClassInitialize]”. ClassInitialize wordt altijd uitgevoerd voor alle test functies van de test klasse. Dit is erg handig als je specifiek voor één test klassen bepaalde objecten wilt aanmaken.

Zoals ik al eerder had genoemd heb ik een waslijst een test scripts geschreven, deze ga ik natuurlijk niet allemaal laten zien. Ik ga 1 test script laten zien die gebruik maakt van de in memory database die ik in de bovenstaande alinea had besproken.

In figuur 9 kunt u mijn testscript voor het linken van externe categorieën aan een interne categorie zien. Om de testscript begrijpbaar te houden heb ik deze opgedeeld in 5 stukken.
Stap 1: het aanmaken van de interne categorie en deze toevoegen aan de test database.
Stap 2: het aanmaken van een aantal externe categorieën en deze toevoegen aan de test database.
Stap 3: het ophalen van de externe categorie Ids zodat ik deze kan gaan linken aan de interne categorie. Daarnaast bepaal ik gelijk het verwachte resultaat met de variabele expectedResult.
Stap 4: het uitvoeren van de functie de getest moet worden. Nadat dit gedaan is slaan we het resultaat op in de variabele result;
Stap 5: controleren over het verwachte resultaat overeenkomt met de werkelijke uitkomst.

```
[TestMethod]
public void TestLinkCategories()
{
    var category = new Category() { Id = 10, Name = "Test", Niveau = 1 };
1   Db.Category.Add(category);
    Db.SaveChanges();

    var extCategories = new List<ExternalCategory>()
    {
        new ExternalCategory(){ Id = 10, SupplierId = 1, Name = "ExternalCategory10", Niveau = 1, ParentE:
        new ExternalCategory(){ Id = 11, SupplierId = 1, Name = "ExternalCategory11", Niveau = 2, ParentE:
        new ExternalCategory(){ Id = 12, SupplierId = 1, Name = "ExternalCategory12", Niveau = 3, ParentE:
2       new ExternalCategory(){ Id = 13, SupplierId = 2, Name = "ExternalCategory13", Niveau = 1, ParentE:
        new ExternalCategory(){ Id = 14, SupplierId = 2, Name = "ExternalCategory14", Niveau = 2, ParentE:
        new ExternalCategory(){ Id = 15, SupplierId = 2, Name = "ExternalCategory15", Niveau = 3, ParentE:
    };
    Db.ExternalCategory.AddRange(extCategories);
    Db.SaveChanges();

3   var externalCategorieIds = extCategories.Select(a => a.Id.ToString()).ToList();
    var expectedResult = externalCategorieIds.Count;

4   ExternalCategoryManager.Instance.LinkCategories(Db, category.Id, externalCategorieIds);
    var result = Db.Category.First(a => a.Name == "Test").ExternalCategories.Count;

5   Assert.AreEqual(expectedResult, result);
}
```

Figuur 9: TestLinkCategories

In de onderstaande tabel beschrijf ik voor welke must en should have requirement ik een unit test script heb geschreven. Per functionaliteit heb ik aangegeven of deze gedekt wordt met een unit test. Voor sommige functionaliteiten kan ik geen nuttige unittest kunnen schrijven. Deze heb ik met een streepje weergegeven. Bijvoorbeeld het aangeven of een prijs is gestegen of gedaald. Dit valt voor mij niet te testen, aangezien dit in de view wordt geregeld.

Functionaliteit	MoSCoW prioriteit	Getest Y/N
Het systeem moet producten ophalen van leveranciers	Must	Y
De primaire leveranciers moeten worden gekoppeld aan het systeem	Must	Y
Er moet een overzicht komen met alle producten	Must	-
Producten moeten gefilterd kunnen worden op leverancier en categorie	Must	Y
Producten moeten gezocht kunnen worden op artikelnummer, fabrikantnummer, naam en omschrijving	Must	Y
Producten moeten gesorteerd kunnen worden op prijs en voorraad	Must	Y
De prijzen moeten iedere dag worden bijgewerkt	Must	Y
De productenlijst moet iedere week worden bijgewerkt	Must	Y
Wanneer een product is verwijderd moet deze nog wel blijven bestaan	Must	-
De secundaire leveranciers moeten worden gekoppeld aan het systeem	Should	N
Het moet mogelijk zijn om een minimum order bedrag in te voeren bij een leverancier	Should	-
Producten worden verdeeld in categorieën	Should	-
De gebruiker moet op de hoogte worden gesteld van promo prijzen	Should	-
Het moet mogelijk zijn om in te stellen waarvan je promo prijzen wilt ontvangen	Should	N
Er moet een overzicht komen voor de orderkosten	Should	-
Eerder aangemaakt order moeten later nog ingezien kunnen worden	Should	-
Een product moet kunnen worden toegevoegd aan een winkelwagen	Should	Y
Een product moet kunnen worden verwijderd van een winkelwagen	Should	Y

Er moet een order aangemaakt kunnen worden van een winkelwagen	Should	-
Zelf kunnen kiezen van welke categorieën producten worden geïmporteerd	Should	Y
Weergeven wanneer een product het laatst is bijgewerkt	Should	-
Aangeven op de prijs van een product is gestegen of gedaald	Should	-
Een filter toevoegen voor promotieprijzen	Should	-
Bij de prijs in het overzicht weergeven of het om een normale of promotie prijs gaat	Should	-
Het moet mogelijk zijn om meerdere winkelmandjes aan te maken	Should	-
Een winkelmandje moet je een naam kunnen geven	Should	Y
Een winkelwagen moet worden opgeslagen om op een later tijdstip weer te gebruiken	Should	Y
Een winkelwagen moet kunnen worden verwijderd	Should	Y

In de onderstaande tabellen ga ik per model klasse aangeven wat voor unit test ik heb geschreven en wat de unit test precies test. Hiermee krijgt u inzicht in wat er precies wordt gedekt met de unit test scripts.

Category

Test functie	**Doel van de test?**
TestLinkCategories	Er wordt getest of externe categorieën met succes kunnen worden gekoppeld aan een interne categorie. Er wordt voor de test gebruik gemaakt van een lijst van externe categorieën die wordt gekoppeld aan één interne categorie.
TestRemoveLinkedCategories	Doet het tegenovergestelde van de bovenstaande test functie. Deze test controleert of de link van externe categorieën naar interne categorieën met succes wordt verwijderd.

ExternalCategory

Test functie	**Doel van de test?**
TestUseExternalCategoryInImport	Deze test controleert of de functie die ervoor zorgt dat een externe categorie wordt meegenomen in de import nog werkt. De import zelf wordt dus niet getest, alleen het gedeelte wat de boolean voor het meenemen van de external categorie op true of false zet.

Product

Test functie	**Doel van de test?**
TestSearchByArticleNumber	Testen of het zoeken op een artikelnummer werkt.
TestSearchByManufacturerNumber	Testen of het zoeken op een fabrikantnummer werkt.
TestSearchByName	Testen of het zoeken op de naam van een product werkt.
TestSearchByDescription	Testen of het zoeken op de omschrijving van een product werkt.
TestFilterByCategory	Test of het filteren van producten op een categorie nog werkt.
TestFilterBySupplier	Test of het filteren van producten op een leverancier nog werkt.

TestFilterByPromotionPrice	Test of het filteren van producten op alleen promotie producten nog werkt.
TestSortByPriceAscending	Test of het sorteren op prijs oplopend nog werkt.
TestSortByPriceDescending	Test of het sorteren op prijs aflopend nog werkt.
TestSortByStockDescending	Test of het sorteren op voorraad aflopend nog werkt.
TestSortByStockAscending	Test of het sorteren op voorraad oplopend nog werkt.
TestProductSearchSpeed	Testen of een zoekactie op producten binnen 1000 milliseconden wordt uitgevoerd. Het zoeken wordt getest of een dataset van 100k producten.

ShoppingCart

Test functie	**Doel van de test?**
TestAddProductToShoppingCart	Deze test controleert of het toevoegen van een product aan een winkelwagen nog functioneert.
TestRemoveProductFromShoppingCart	Deze test controleert of het verwijderen van een product van een winkelwagen nog goed functioneert.
TestupdateProductCountShoppingCart	Deze test controleert of de functie die ervoor zorgt dat een product vaker in een winkelwagen voor kan komen nog goed functioneert. Dus of de count bij een orderpart_product nog werkt.

ProductImport

Test functie	**Doel van de test?**
TestProcessCSV_WrongFileType	Deze test kijkt wat er gebeurd wanneer ik een producten bestand probeer uit te lezen die niet van een CSV formaat is. Voor dit test geval wordt gebruik gemaakt van een XML bestand.
TestProcessCSV_WithHeaderRow	Testen of het uitlezen van een CSV met een header row nog functioneert.
TestProcessCSV_WithoutHeaderRow	Testen of het uitlezen van een CSV zonder een header row nog functioneert. Dus zonder een eerste regel die de kolom namen definieert.
TestProcessCSV_WithSeperator	Testen of het uitlezen van een CSV met een seperator werkt. Voor deze test heb ik gebruik gemaakt van een komma als seperator.
TestProcessCSV_WithOutSeperator	Testen of het uitlezen van een CSV zonder een meegegeven separator nog werkt. De standaard separator staat in ingesteld op een tab.
TestCreateHeaderDictionary	Controleert of het aanmaken van de headers voor de CSV files nog werkt.
TestGetCurrentDateString	Test of de naam van de aan te maken import mapjes nog overeenkomt met de datum van vandaag.

ProductImportAcesDirect

Test functie	**Doel van de test?**

TestRequestDataAcesDirect_FilesCreated	Deze test functie controleert of de product data nog kan worden gedownload van de door AcesDirect aangeleverde download link.
TestUpdateCategoriesAcesDirect	Deze test functie controleert of de functie die categorieën aanmaakt van AcesDirect nog goed functioneert. Het deel wat ervoor zorgt dat de categorieën ook echt worden opgeslagen wordt niet getest. Dit komt doordat bulkinsert niet wordt ondersteund binnen Effort. Hetzelfde geldt ook voor de onderstaande test scripts.
TestUpdateProductGeneralAcesDirect	Deze functie controleert of de functie die algemene product gegevens aanmaakt voor AcesDirect nog werkt.
TestUpdateProductPricesAcesDirect	Deze functie controleert of de functie die de product prijzen aanmaakt voor AcesDirect nog werkt.
TestUpdateProductStockAcesDirect	Deze functie controleert of de functie die de product voorraad aanmaakt voor AcesDirect nog werkt.

ProductImportTechdata

Test functie	**Doel van de test?**
TestRequestDataTechdata_FilesCreated	Test of het ophalen van product data bij Techdata nog werkt. Mocht de ftp verbinding om één of andere reden niet meer werken. Merkt je dat door deze test uit te voeren.
TestUpdateCategoriesTechdata	Test of de functie die categorieën voor Techdata aanmaakt nog werkt.
TestUpdateProductGeneralTechdata	Test of de functie de algemene product informatie aanmaakt voor Techdata nog werkt.
TestUpdateProductPricesTechdata	Test of de functie die de product prijzen aanmaakt voor Techdata nog werkt.
TestUpdateProductStockTechdata	Test of de functie die de product voorraad aanmaakt voor Techdata nog werkt.
TestDeleteProductsTechdata	Test of de functie die producten verwijderd die door Techdata uit de import lijst zijn gehaald nog werkt.

10 Smoketest

Naast de unit test heb ik ook nog verder gekeken naar testmogelijkheden voor de applicatie. Zo kwam ik smoke tests tegen. Met een smoke test klik je als het ware door de applicatie heen en controleer je of er niks breekt. Wanneer iets breekt komt er als het ware rook uit.

Voor het schrijven van de smoke test maak ik gebruik van het Robot Framework. Het Robot Framework is een open source testautomatisering framework voor acceptatie testen en acceptatie test-driven development (ATDD). Het Robot Framework is verder uit te breiden met diverse Java of Python libaries.

Ik gebruik de externe python libary Selenium2Libary in combinatie met het Robot Framework. De Selenium2Libary voert de testen uit in een web browser. De meeste moderne webbrowsers worden ondersteund, ik maak voor mijn testen gebruik van de Firefox webbrowser.

Voor de smoke testen heb ik een nieuwe folder binnen mijn project map aangemaakt. Vervolgens heb ik voor elke test module ook een aparte folder aangemaakt. Een test module is in principe een verzameling van testen voor één model object. Elke module bestaat uit een resource tekst bestand en een tekst bestand waarin de testen worden uitgevoerd. In het resource tekst bestand staan keywords specifiek voor de module. Ik heb ook nog een globale resource waarin keywords staan die door meerdere test modules worden aangeroepen.

De testen zijn allemaal geschreven in standaard tekst bestandjes. Binnen het tekst bestand wordt als scheidingsteken gebruik gemaakt van een tab. De tekst bestanden bestaan vervolgens weer uit variabelen en keywords die kunnen worden uitgevoerd. Aan de hand van deze keywords wordt er binnen de webbrowser acties uitgevoerd. Wanneer één zo een keyword niet met succes wordt afgerond komt er “rook” uit test en is deze gefaald. De beschikbare keywords zijn te vinden in de documentatie van Selenium2Libary.

Tot op heden heb ik 8 smoke tests geschreven voor het testen van de applicatie. Onderstaand zal ik één smoke test doorlopen en toelichten. De smoke test die ik ga toelichten is de smoketest voor het toevoegen van een product aan de winkelwagen.

In figuur 10 kunt u zien hoe de test is opgebouwd. De eerste regel beschrijft de naam van de test. De tweede regel met [Tags] beschrijft de tags die zijn aangemaakt voor het aanroepen van deze test. Vanaf regel 3 t/m regel 9 staan keywords die worden uitgevoerd voor deze test, deze zal ik zo verder toelichten. De laatste regel met de [Teardown] bevat een keyword die wordt uitgevoerd aan het eind van de test. In dit geval wilde ik dat de browser weer werd afgesloten zodra de test klaar is.

```
Add_product_to_shoppingcart
    [Tags]    Shoppingcart    Add_product_to_shoppingcart
    Open Browser To Start Page
    Open Shoppingcart Page
    Create Shoppingcart
    Open Product Page
    Add Product To Shoppingcart
    Open Shoppingcart Page
    Delete Shoppingcart
    [Teardown]    Close Browser
```

Figuur 10: Smoke test - add product to shoppingcart

De commando’s die worden aangeroepen in de test zijn gedefinieerd in de resource bestanden of zijn standaard beschikbaar door Selenium2Libary. Figuur 11 bevat het resource bestand specifiek voor shoppingcart en figuur 12 bevat de base resource die wordt gebruikt door diverse test modulen. Ik laat niet alle keywords zien, anders werden de plaatjes te groot.

```
*** Settings ***
Library           Selenium2Library

*** Keywords ***
Add Product To Shoppingcart
    Click Element    //tr[1]/td[9]/form/a/span
    Page Should Contain    Product toegevoegd
```

Figuur 11: Smoke test - shoppingcart resource

```
*** Settings ***
Library           Selenium2Library

*** Variables ***
${SERVER}         offertetool.localtest.me
${START URL}      http://${SERVER}/
${BROWSER}        Firefox
${DELAY}          0.5

*** Keywords ***
Open Browser To Start Page
    Open Browser    ${START URL}    ${BROWSER}
    Maximize Browser Window
    Set Selenium Speed    ${DELAY}

Open Shoppingcart Page
    Click Link    id=popOverProductAdded
```

Figuur 12: Smoke test - Base resource

Als we kijken naar het eerste keyword bij figuur 12 “Open Browser To Start Page”, zien we dat binnen dit keyword weer andere keywords worden aangeroepen. Eerst wordt de browser opgestart, vervolgens wordt de browser gemaximaliseerd en als laatste wordt de delay per commando aangegeven. In dit geval 0,5 seconden.

In figuur 11 staat het keyword voor het toevoegen van een product aan een winkelwagen. Dit wordt gedaan met het keyword Click Element, wat standaard door Selenium2Libary wordt meegeleverd. Een Click Element heeft als parameter een html element waarop geklikt moet worden. In dit geval is de eerste rij uit de tabel en vervolgens de 9^{e} kolom. Vervolgens controleer ik met het keyword “Page Should Contain” of er ook echt een product is toegevoegd aan de winkelwagen. Er moet namelijk een pop up zijn getoond met de tekst “Product toegevoegd”.

Zoals u al had kunnen zien bevatten de test in totaal7 keywords die achter elkaar worden uitgevoerd. Onderstaand zal ik per keyword globaal beschrijven wat er gebeurt.

1. Open Browser To Start Page; Start Firefox op en navigeert naar de home pagina.
2. Open Shoppingcart Page; Navigeert naar het winkelwagen overzicht.
3. Create Shoppingcart; Maakt een test winkelwagen aan.
4. Open Product Page; Navigeert naar het producten overzicht.
5. Add Product To Shoppingcart; Voegt een product toe aan de test winkelwagen.
6. Open Shoppingcart Page; Navigeert naar het winkelwagen overzicht.
7. Delete Shoppingcart; Verwijderd de test winkelwagen.

De overige testen zijn op een zelfde manier opgebouwd. Aan het eind van de test ruim ik altijd de aangemaakte test data op zoals hierboven bij stap 7.