Afstudeerverslag

Jesse van Assen

1-6-2012

Voorwoord

Dit verslag is geschreven door Jesse van Assen, vierdejaars informaticastudent aan de Haagse Hogeschool. In het derde en vierde kwartaal van het vierde jaar heb ik een afstudeerstage verricht bij het Haagse softwarebureau Cenosco B.V. Dit verslag is vooral bedoeld voor mijn eerste en tweede afstudeerbegeleiders, Tim Cocx en Nanny Jacobs, en de gecommitteerde die mijn afstudeerstage zal beoordelen. Daarnaast is het ook inzichtelijk voor andere eventueel geïnteresseerde lezers.

Ik wil deze gelegenheid gebruiken om een aantal mensen te bedanken. Ten eerste wil ik mijn collega's bij Cenosco hartelijk bedanken voor de leuke en vooral leerzame tijd die ik er heb doorgebracht. Hierbij wil ik mijn begeleider, Mischa Simonis, speciaal uitlichten. Hij heeft mij tijdens mijn afstudeerstage begeleid, feedback en adviezen gegeven, maar mij ook vrij gelaten om een aantal keuzes zelf te maken. Door deze combinatie van gefundeerde feedback en vrijheid heb ik enorm veel geleerd. Mischa, bedankt!

Ten tweede wil ik Tim Cocx en Nanny Jacobs bedanken voor de tijd die ze in het begeleiden en beoordelen van mijn afstudeerperiode hebben gestoken. Dit is een aanzienlijk document geworden, en het lezen hiervan moet een flinke klus geweest zijn. Ook wil ik de gecommitteerde bij voorbaat alvast bedanken, echter kan ik deze nog niet bij naam benoemen.

Ten derde wil ik een goede vriend en collega van mijn vorige werkgever Patrick Teerlink bedanken. Ook al had hij geen professioneel belang in mijn afstudeeropdracht, heeft hij altijd interesse getoond in de oplossingen waarvoor ik gekozen heb, en kon mij soms bij een probleem net dat duwtje in de rug geven wat nodig was om tot de oplossing te komen. Patrick, bedankt.

Ten slotte wil ik mijn medestudenten en vrienden Sander Benschop en Jeroen Niesen bedanken voor het geven van feedback tijdens mijn afstudeerstage, voor het lezen van mijn verslag, en de feedback die ze op dit verslag hebben gegeven. Jullie hebben mij geholpen om deze onderdelen naar dit niveau te tillen.

Inhoudsopgave

1 Inleiding

De opleiding informatica wordt in het vierde jaar afgesloten met een afstudeerstage, waarin de student kan laten zien dat hij kan functioneren in een bedrijf, en dat hij de competenties die hij op school geleerd heeft in de praktijk kan toepassen.

Dit document heeft als doel om de activiteiten die ik tijdens mijn afstuderen heb uitgevoerd te beschrijven, en de keuzes die ik tijdens deze periode heb gemaakt toe te lichten. Op basis van deze informatie kan er een oordeel worden gegeven over deze periode.

Dit document is globaal op te delen in drie onderdelen. In het eerste onderdeel wordt het bedrijf omschreven, en wordt de opdracht toegelicht. Dit onderdeel geeft voornamelijk achtergrondinformatie.

Het tweede deel beschrijft de werkzaamheden die ik tijdens mijn afstuderen heb uitgevoerd, en licht de keuzes die ik gemaakt heb toe.

In het derde deel kijk ik terug op mijn stageperiode. Eerst kijk ik terug op het testen binnen het project, en het projectmanagement met Scrum. Hierna evalueer ik zowel het eindproduct en het resultaat, als het gehele proces.

2 Het bedrijf

2.1 Het ontstaan van Cenosco

Cenosco is heel ruim gezien ontstaan vanuit de eenmanszaak van mijn begeleider, Mischa Simonis. Mischa werd in ’99 gebeld, dat een ander informaticabedrijf uit Den Haag met een probleem zat met mensen die dat bedrijf bij Shell gedetacheerd had. Dit project verliep niet heel goed, en het bedrijf wilde eigenlijk de gedetacheerden vervangen. Mischa is toen gebeld of hij een van de personen kon zijn waardoor deze mensen vervangen gingen worden.

Toen Mischa bij dit project betrokken raakte, verliep dit weer beter, en merkte hij dat er gelijk geld binnen begon te stromen. Hij heeft toen een oude schoolvriend waar hij nog regelmatig contact mee had, Marcel Edelmans, gevraagd of deze zijn huidige baan kon opzeggen om hem te ondersteunen bij de werkzaamheden die hiermee gepaard gingen.

Ook de detacheerder merkte dat het eind ’99 erg goed ging, en wilde eigenlijk groeien in zowel de grootte van zijn organisatie als de richtingen waarin deze werkzaam was. Mischa is hiervoor benaderd, of hij interesse had om directeur te worden van een maatwerksoftwarebedrijf. Deze interesse was er, en hij heeft vervolgens aan Marten en twee oud-collega’s, Marcel Sminia en Serge van Voorst, om hier onderdeel van te worden. Zo is op 1 januari 2000 Custom Productivity Tools (CPT) ontstaan, als onderdeel van de Metropol IT Holding.

Eind 2002 begon het slechter te gaan met de detachering, en merkte Mischa dat hij het niet altijd eens was met de visie van de holding. Als gevolg hiervan heeft CPT toen een buyout gedaan, waarbij ze de aandelen van CPT hebben uitgekocht van de holding. Ze zijn vervolgens in het huidige pand getrokken als alleenstaand bedrijf. In 2005 verhuisde Serge, en is toen ook uitgekocht. Sindsdien zijn Mischa, Marcel en Marten met zijn drieën.

In 2008 ging het steeds beter met CPT, en gingen ze ook steeds meer detacheren. De naam Custom Productivity Tools paste hier echter niet heel goed bij, omdat dit niet echt als een detacheernaam over kwam. Elke keer moest uitleg gegeven worden dat er náást maatwerk ook aan detachering werd gedaan.

De behoefte is toen ontstaan om holding te creëren waarin deze onderdelen beter gedefinieerd waren. Zo is Cenosco ontstaan, met hierbinnen de suborganisaties Custom, People en Products. Custom is hierin het maatwerkbedrijf, en People de detacheerder. Products is een tak die zich bezig houdt met de verkoop van software welke binnen een groot aantal organisaties kan draaien.

2.2 Labirint, het kantoor in Kroatië

In 2007 is er het bedrijf Labirint in Kroatië opgezet, wat als de nearshore-locatie dient voor Cenosco. Bijna alle programmeeropdrachten gaan hiernaartoe. Binnen Den Haag worden voornamelijk ontwerpen gemaakt, en worden de complexere opdrachten uitgevoerd. Ook is er in Den Haag een framework ontwikkeld, wat hier onderhouden wordt. Dit framework wordt in Kroatië gebruikt om de applicaties bovenop te bouwen. Ook geeft Labirint Cenosco ondersteuning bij systeembeheer.

2.3 De organisatie binnen Cenosco

Cenosco is een klein bedrijf. Hierdoor is de verdeling van de rollen binnen het bedrijf niet goed te definiëren. Mischa kon de rol van hem, Marcel en Marten het beste beschrijven als meelopend voorman. Dat betekend dat ze meehelpen en meedenken waar nodig is binnen de projecten, maar ook het klantencontact onderhouden, zorgen dat de planningen gehaald worden, er voor zorgen dat er overzicht gehouden wordt op de werkzaamheden, er een testplan wordt opgesteld, etc.

Wanneer je binnen een groot bedrijf werkt, zijn de rollen meestal smal en gefocust. In een klein bedrijf zijn deze veel breder, en minder concreet. De functies van het management zijn hierdoor niet goed in een hokje te plaatsen.

Binnen Cenosco houden twee medewerkers zich bezig met verkoop, en een iemand doet de boekhouding en vervult de rol van “manusje van alles”. De overige medewerkers zorgen allemaal voor inkomsten, door ontwikkeling of systeembeheer. In zowel Den Haag als Kroatië zijn ongeveer tien mensen aanwezig.

2.4 De werkzaamheden die Cenosco Custom verricht

Cenosco richt zich voornamelijk op twee branches: engineering en health care. Onder engineering valt onder andere manufactoring en proces & chemie. Dit zijn volgens Cenosco marktsegmenten waarin het grote geld te verdienen is.

Dit betekent dat ze als bedrijf een technisch profiel hebben. In de software die ze maken wordt veel gerekend, en is echt maatwerk. Rijke applicaties noemt Cenosco dat. Het is dus geen marketingbedrijf die hele strategische websites maakt om iets op de kaart te zetten.

De applicaties die Cenosco maakte waren vroeger altijd clientapplicaties in de vorm van een Windows- of Linuxapplicatie, omdat deze het beste bij de wensen van de klanten pasten. De benodigde interactie was gewoonweg nog niet mogelijk binnen een website. Dit is de laatste jaren steeds meer veranderd, en sinds een jaar of twee vindt Cenosco dat ze ook een webapplicatie aan klanten kunnen verkopen. De wens van de klant is hierin natuurlijk altijd leidend, maar met de nieuwe ontwikkelingen binnen de browsermarkt vindt Cenosco dat ze nu met vertrouwen een webapplicatie aan de klanten kunnen aanbevelen.

2.5 Klanten van Cenosco

Onder de klanten van Cenosco bevinden zich een aantal grote namen, waarbij Shell de grootste klant is. Ook vallen de Brink groep, Starbucks en een aantal uitgeverijen onder de klanten van Cenosco. Ook vervullen ze werkzaamheden voor Rijkswaterstaat.

Cenosco geeft zelf aan dat ze zich best gelukkig mogen prijzen dat ze veel grote organisaties hebben als klanten, waardoor ze redelijk met rust gelaten worden. Cenosco krijgt een klus, en na een aantal weken wil de klant iets moois zien, zonder veel administratie hier omheen. In de praktijk blijkt deze aanpak erg goed te werken.

2.6 Cenosco en stagiairs

Cenosco heeft al enige ervaring met stagiairs. Tussen 2000 en 2007 hebben ze eigenlijk elk jaar wel een stagiair aangenomen. Het is toen even rustig geweest, maar in 2010 hebben ze weer een stagiair aangenomen, en vorig jaar zelfs twee. Dit jaar ben ik vooralsnog de enige stagiair.

3 Het project

3.1 De opdrachtomschrijving

3.1.1 Probleemstelling

Wanneer er een conflict optreedt, en beide partijen hier samen niet uit kunnen komen, kan het oplossen van dit conflict een hoge kostenpost worden. Een voorbeeld hiervan is een echtscheiding. Wanneer beide partijen niet uit de verdeling van de boedel kunnen komen, komt hier vaak een advocaat aan te pas. Deze advocaat vervult dan een soort van scheidsrechterrol, waarbij zijn doel is dat de boedel goed verdeeld wordt. Door het niet geringe tarief van de advocaat kan dit tot flinke kosten leiden.

3.1.2 Doelstelling

Cenosco zag een mogelijkheid om dit soort conflicten op te lossen zonder dat hier een jurist bij aan de pas hoeft te komen. Hierbij kan er aan verschillende soorten conflicten worden gedacht, zoals het bovengenoemde echtscheidingconflict.

Dit is een groot project, wat veel groter is dan mijn afstudeeropdracht. Ik heb me dan ook op enkele subonderdelen gefocust. De gehele applicatie is in de vorm van een ASP.NET MVC webapplicatie.

Ten eerste heb ik een platform opgesteld, waarbinnen de conflicten kunnen worden opgelost. Binnen dit platform is het mogelijk om gebruikers aan te maken, en kunnen deze gebruikers zich aanmelden. Ook worden vanuit dit platform de conflictoplossingen gestart, en kan de gebruiker de conflictoplossingen inzien waarbij hij aangemeld is.

Ook worden in dit platform enkele beveiligingsaspecten opgenomen. Er moet voorkomen worden dat gebruikers de conflictoplossingen van anderen kunnen bekijken. Door hier op een abstract niveau naar te kijken, kan dit voor alle type conflictoplossingen gebruikt worden. Ook moet er een rollen- en rechtensysteem worden ontwikkeld waarmee de handelingen die een gebruiker mag verrichten op een hoger niveau worden geregeld, en hier niet binnen de conflictoplossing expliciet op gecontroleerd hoeft te worden.

Door deze onderdelen op een abstract niveau af te handelen, kost dit minder tijd bij het daadwerkelijke implementeren van de conflictoplossingen. De abstractie kan immers bij alle oplossingen worden gebruikt, en is ook voor alle van toepassing.

Ten tweede ga ik mij richten op het conflict boedelverdeling, wat onderdeel is van een echtscheiding. Gebruikers moeten op een snelle manier een inventaris van een of meerdere locaties kunnen uitvoeren, waardoor er snel structuur kan worden aangebracht in de spullen die verdeeld moeten worden. Gebruikers kunnen aangeven dat ze eigenaar van de spullen zijn, dat ze de spullen willen hebben, etc. Vervolgens kunnen de overige spullen met een algoritme over de betrokkenen worden verdeeld.

3.2 Afbakening van het project

Binnen de opdracht ga ik gebruik maken van Scrum. Ik heb hiervoor gekozen omdat bij de start van de opdracht de requirements verre van duidelijk waren. Het startpunt van mijn opdracht was een idee binnen Cenosco, waardoor er geen concrete wens of probleem ligt. De vele iteraties en

terugkoppelingen met mijn begeleider waar Scrum voor zorgt, zijn ideaal binnen dit soort project. De vele releases die Scrum met zich meebrengt zorgen voor veel feedback van mijn opdrachtgever, waardoor gaandeweg de requirements steeds verder duidelijk worden.

Scrum brengt ook een ander voordeel met zich mee, en dat is dat er geen vast releaseprogramma is vastgesteld. Er kan per sprint worden besproken welke onderdelen belangrijk zijn, en waar ik mijn aandacht aan ga besteden. Hierdoor komen er tijdens de opdracht requirements bij, en kan er met de prioriteit van requirements worden geschoven.

Door deze eigenschappen van het project, is dit eigenlijk niet geschikt om getimeboxed te zijn. Doordat de requirements regelmatig veranderen, is het erg lastig om het project af te bakenen. Een requirement die aan het begin van het project erg belangrijk lijkt, kan in de loop van het project minder relevant blijken, of kan worden verdrongen door andere requirements die een hogere prioriteit hebben gekregen.

Hierdoor is van te voren niet in details te treden welke concrete onderdelen ik op ga leveren. In de loop van het project komt er steeds meer duidelijkheid hierover, en tegen het einde van het project ga ik samen met mijn begeleider afbakenen welke onderdelen ik ga opleveren. Deze onderdelen zullen vervolgens als een afgerond geheel worden opgeleverd. Het feit dat de producten van mijn afstudeerstage onderdeel zijn van een groter geheel maakt dit mogelijk.

4 De opstart van het project

De eerste week van de afstudeerstage heb ik gebruikt om het project op te starten. Deze opstart is in drie onderdelen in te delen: Het inleidende gesprek met de opdrachtgever, het indelen van mijn werkplek en het opstellen van het plan van aanpak.

In eerste plaats zou het inleidende gesprek het eerste zijn wat bij het begin van het afstuderen plaats zou vinden. Echter was mijn opdrachtgever deze dag bezig met het afronden van een ander project wat voor Cenosco van groot belang was. Daarom is dit gesprek naar een later moment verplaatst.

Een ander lid van het management heeft mij toen een rondleiding binnen het bedrijf gegeven. Hierbij heeft hij mij onder andere verteld over de manier waarop Cenosco projecten oppakt. Ook heeft hij mij toen binnen bedrijfsnetwerk aangemeld zodat ik hierop kon inloggen en hiervan gebruik kon maken. Hierna kon mijn werkplek worden ingedeeld.

De volgende dag heeft het inleidende gesprek plaats gevonden. Hierbij is er gesproken over de manier waarop mijn opdrachtgever en ik de afstudeeropdracht voor ogen hadden. We hebben op een erg globaal niveau besproken welke activiteiten ik tijdens mijn afstudeerproject ga uitvoeren, en hebben we een globale planning opgesteld. Ook is er besproken van welke middelen ik bij het afstuderen gebruik ging maken.

4.1 Het opstellen van het plan van aanpak

Aan het begin van het project kon ik al een aantal dingen zeggen over het verdere verloop van het project. Dit heb ik in een plan van aanpak verwerkt, zodat er aan het begin van het project al enige houvast was aan de projectverloop.

4.1.1 Opdrachtomschrijving en afbakening

Als eerste heb ik in kaart gebracht wat de opdrachtomschrijving was, en heb ik de afbakening van het project aangegeven. Deze twee punten waren lastig, gezien het project nog redelijk vaag was. Omdat het voortgekomen is uit een idee, was er geen concreet probleem. Hierdoor kon de opdrachtomschrijving globaal worden opgesteld, maar nog niet in details.

Ook de afbakening was lastig, omdat er qua details nog niet heel veel bekend was. Ook de afbakening is hierdoor op een globaal niveau gebeurd, en is aan verandering onderhevig.

Echter ga ik binnen het project gebruik maken van een Agile-methodiek, en de Agile Manifesto[1] geeft duidelijk aan waar de focus moet liggen binnen een project: “Customer collaboration over contract negotiation” en “Responding to change over following a plan”. De focus moet dus liggen op de gesprekken tussen mij en de opdrachtgever en de veranderingen waar dit naar leidt, en niet op het vasthouden van de opdrachtomschrijving en de scoping van het project binnen het plan van aanpak. Daarom kon ik met een gerust hart deze onderdelen wat globaler houden.

4.1.2 Activiteiten

Binnen het project zal ik een aantal activiteiten gaan doorlopen, beginnend met het verzamelen van requirements. Binnen het project zijn er een aantal zaken waar aandacht aan besteed moet

[1] http://www.agilemanifesto.org/

worden, en deze onderdelen kunnen in een requirementsdocument in kaart worden gebracht. Wederom zijn deze door het gebruik van Agile niet bindend, maar zijn deze onderhevig aan veranderingen.

Voordat er ontwikkeld gaat worden, zal ik eerst aandacht besteden aan het uitdenken van de software, en hier een ontwerp bij maken van de software en de database. Zo kan ik voorkomen dat ik voor verassingen kom te staan tijdens het ontwikkelen, omdat er onverwachte afhankelijkheden opduiken, of dat er onderdelen uit te breiden moeten zijn waar ik bij het ontwikkelen geen rekening mee gehouden heb. Door van tevoren na te denken over de structuur van de software kan dit worden verminderd. Ook kan ik de modellen gebruiken bij het overleg met mijn opdrachtgever wanneer het over de interne werking van het systeem gaat.

Om het uiterlijk van de applicatie te tonen, maak ik gebruik van grafische ontwerpen. Dit kunnen schetsen zijn, screenshots van een mockup of screenshots van de ware applicatie. Zo kan ik met mijn opdrachtgever door de applicatie lopen zonder dat hier een draaiende versie voor nodig is.

Hierbij moet er rekening gehouden worden dat het maken van een grafische interface niet binnen mijn kwaliteiten aanwezig is. Binnen de informatica-opleiding wordt er aan het grafisch ontwerp geen aandacht besteed, en hierdoor heb ik weinig ervaring met het opzetten van een gebruiksvriendelijke interface.

Wel maak ik al jaren gebruik van het internet, en hier worden een aantal dingen standaard gedaan, zoals het plaatsen van een loginknop rechtsboven op het internet. Hierdoor heb ik een “gevoel” ontwikkeld waar bepaalde onderdelen geplaatst moeten worden. Daarbij kan mijn begeleider mij sturen wanneer er onderdelen misplaatst zijn.

Hierna kan er worden overgegaan in het bouwen van het systeem. Hierbij zal er gebruik gemaakt worden van de activiteiten ontwikkelen, testen en documenteren. Bij het bouwen zal er extra rekening gehouden worden met het schrijven van *clean code*, code die goed te begrijpen is voor de lezer en zichzelf documenteert. Door functies en variabelen gepaste namen te geven en te refactoren zal documentatie binnen de code een ondergeschikte rol krijgen. In het boek Clean Code[1] wordt hier veel aandacht aan besteed.

Documentatie binnen de code zal voornamelijk worden gebruikt om onderdelen te verduidelijken die niet binnen code kunnen worden uitgedrukt. Een voorbeeld hiervan kunnen een aantal regels zijn, die om een specifieke reden in een volgorde moeten staan, wat niet direct aan de code af te leiden is.

Tijdens het project wordt er gebruik gemaakt van Test Driven Development, waarbij de unit tests eerst worden geschreven, en daarna de code. Hierdoor is bijna alle code gedenkt door unit tests, en kan de kwaliteit beter worden gegarandeerd. Ook is de geproduceerde code testbaar, omdat de tests eerst worden geschreven en daarna pas de code. De code wordt dus niet geschreven als er geen tests voor zijn, waardoor de code wel testbaar *moet* zijn. Niet-testbare code is bijvoorbeeld code die veel afhankelijkheden heeft, welke niet vervangbaar zijn door testimplementaties. In het plan van aanpak ga ik dieper op TDD in.

[1] *Clean Code*, Robert C. Martin, Prentice Hall, 2008

Ten slotte moet het product aan het einde van het afstudeertraject worden opgeleverd. In welke vorm dit gebeurd, wordt aan het einde van mijn afstudeerstage met mijn opdrachtgever besproken.

4.1.3 Methodes en technieken

Binnen het project ga ik gebruik maken van een aantal methodes en technieken. Binnen Cenosco wordt er gewerkt met een beheertechniek die Project Delivery Framework heet. Deze techniek is een watervalmethodiek, echter zijn de eerste twee fasen, project setup en requirements, wél nuttig binnen dit project. De project setup wordt immers altijd uitgevoerd, en de requirements-fase kon mij veel inzicht geven in het project.

Het project is door de eerste twee fasen van het Project Delivery Framework te gebruiken opgestart volgens de methode die binnen Cenosco gebruikelijk is. Hierdoor is de manier waarop het project is gestart conform met de andere projecten binnen Cenosco. Ook kon ik bij de documenten die worden gebruikt voor de requirements afkijken om mijn non-functionele requirements in kaart te brengen.

Binnen het project ga ik gebruik maken van de Agilemethodiek Scrum. Hier heb ik voor gekozen omdat een Agiletechniek in mijn ogen het beste bij dit type project past. In paragraaf 3.2 ben ik hier al op ingegaan.

Voor het vastleggen van prioriteiten van requirements is de MoSCoW-methode een standaard, welke ik binnen school behandeld heb en binnen Cenosco ook wordt gebruikt. Daarom leek mij deze methode geschikt om te gebruiken bij het vastleggen van de requirements. Ook voor het in kaart brengen van de structuur van software is er een standaard die wij op school hebben gehad, namelijk UML. Deze standaard methode gebruik ik ook binnen mijn afstuderen.

Binnen het project maak ik ook gebruik van een aantal technieken en talen. Bij het aannemen van het project was het duidelijk dat het om een webapplicatie zou gaan, welke in ASP.NET mvc gemaakt zou gaan worden. Hierdoor kon ik ASP.NET mvc, C#, HTML, CSS en Javascript aan de te gebruiken technieken toevoegen.

Ik had al enkele ervaring met de Javascript-library jQuery en wist hierdoor dat deze het werken met Javascript enorm veel vergemakkelijkt. Veel browsers hebben net een wat andere implementatie van de manier waarop Javascript hierbinnen werkt, en jQuery slaat hier een brug tussen door een uniforme interface aan te bieden die in alle browsers werkt. Ook zitten hier een aantal handigheidjes in waardoor het werken met Javascript eenvoudiger wordt. Daarom heb ik jQuery ook aan het lijstje van technieken toegevoegd. Helemaal makkelijk is dat jQuery bij ASP.NET mvc wordt geleverd, waardoor ik deze library automatisch kon gebruiken bij het gebruik van ASP.NET mvc.

Een punt dat in het eerste gesprek naar voren kwam, was de uiteindelijke hosting van de applicatie. De opdrachtgever gaf aan dat hij hiervoor veel interesse had in het cloud-platform van Microsoft, Windows Azure. Cenosco heeft een MSDN-account bij Microsoft, en hier zit hosting binnen Windows Azure bij inbegrepen, waar tot dusverre nog geen gebruik van gemaakt is. Mijn project kon een mooie gelegenheid zijn om hier gebruik van te maken vanuit kostenperspectief gezien, maar ook omdat het een mooie eerste test was om te kijken of de cloud geschikt is voor vervolgprojecten binnen Cenosco.

Binnen Cenosco wordt er als version control system gebruik gemaakt van TFS (Team Foundation Server). Hier zal ik ook gebruik van maken, zodat alle sourcecode binnen het bedrijf op één plek aanwezig is.

Voor het testen zal er, zoals eerder al beschreven, gebruik gemaakt worden van Test Driven Development.

4.1.4 Risico's

Binnen het project zijn er een aantal risico's die er voornamelijk mee te maken hebben dat ik een afstudeerder ben, en geen fulltime medewerker. Doordat ik als afstudeerder nog een hoop praktijkervaring mis, zal het tempo waarin ik werk niet hetzelfde zijn als de fulltime medewerkers binnen Cenosco, en zal ik me ook in een aantal dingen moeten inlezen voordat ik deze kan gebruiken. Hierdoor zal ik langer over het project doen, dan wanneer er een fulltime medewerker op gezet zou zijn.

4.1.5 Team

Binnen het project zal ik in een klein team werken. De rol van ontwikkelaar zal ik hierin zelf vervullen, en hierbij zal ik alle activiteiten doorlopen die in het plan van aanpak zijn genoemd.

Mischa Simonis zal de rol van opdrachtgever vervullen, en kan mij helpen met vragen of problemen met betrekking tot de opdracht.

Ten slotte zullen er meerdere personen binnen Cenosco de mentorrol vervullen. Wanneer ik inhoudelijke problemen heb over de technieken, kan ik bij de mentor aankloppen die mij vervolgens verder kan helpen hierbij. Mischa kan de mentorrol vervullen, maar dit kan ook een andere medewerker zijn.

4.1.6 Planning

Met de data die ik heb verzameld kan ik een globale planning opstellen. De eerste twee weken heb ik gereserveerd voor het verzamelen en verwerken van requirements. Daarna ga ik in sprints van twee weken iteratief werken. In deze iteraties zal ik zowel gaan ontwerpen, gaan ontwikkelen en gaan testen.

Zoals in de planning te zien is, bestaan de sprints uit twee weken. Ik heb voor deze lengte gekozen omdat dit een goede balans geeft tussen het werk wat verzet moet worden binnen een sprint, en de lengte van de sprint.

Het opstarten van een sprint brengt overhead met zich mee, omdat de sprint moet worden ingepland met werkzaamheden, etc. Daarom zijn hele korte sprints niet heel erg handig, omdat de overhead hierin best groot is.

Aan de andere kant is er binnen het project nog niet heel veel duidelijkheid, en daarom is vaak contact met de opdrachtgever van belang zodat hij sturing kan geven, en aan kan geven of ik in de goede richting ga. Daarom is een hele lange sprint ook niet handig, omdat ik dan niet heel veel momenten heb waarop ik een sprint afrond en feedback kan krijgen. Omdat mijn afstudeerstage maar een aantal weken in beslag neemt, is het ook geen goed idee om hele lange sprints te gebruiken, omdat ik dan maar enkele iteraties zou doorlopen.

Binnen school heb ik al eens eerder met Scrum gewerkt. Binnen dit schoolproject hebben wij in iteraties van twee weken gewerkt, en dit bleek in de praktijk erg prettig te werken. De balans tussen opleveren en het te verzetten werk was goed in sprints met deze lengte. Daarom heb ik ervoor gekozen om in dit project ook in sprints van twee weken te werken.

Omdat er iteratief wordt gewerkt, zal er zoveel mogelijk worden ontworpen voor de huidige iteratie, en zo min mogelijk voor de daaropvolgende sprints. Natuurlijk moet er rekening gehouden worden met uitbreidbaarheid, maar als van tevoren het volledige systeem wordt ontworpen is de kans groot dat dit later weer moet worden gewijzigd, door veranderende requirements. Daarom zal het ontwerpen zoveel mogelijk binnen de huidige sprint worden gehouden.

Na het ontwerpen volgt het ontwikkelen en het testen, in dezelfde sprint. Omdat er met Test Driven Development wordt gewerkt, worden de tests gelijktijdig opgesteld met de code, waardoor deze in dezelfde sprint gepland zijn.

In de planning zijn de activiteiten in de volgorde ontwerpen, ontwikkelen en testen aangegeven. Dit is echter geen vaste volgorde die wordt gevolgd. Het testen en ontwikkelen wordt gelijktijdig uitgevoerd, waarbij het testen zelfs iets eerder uitgevoerd wordt doordat er eerst een test wordt geschreven. Ook wordt er niet enkel aan het begin van de sprint ontworpen. De ontwerpen-activiteit zal worden doorlopen wanneer deze nuttig is. Dit kan aan het begin van de sprint zijn, maar ook halverwege.

In de laatste anderhalve sprint zal ik daarnaast tijd besteden aan het documenteren van onduidelijkheden binnen de code, en in de laatste sprint zal ik me ook gaan bezighouden met het klaarmaken van de release van het product. Door de release ruim in te plannen wordt dit geen haastklus.

5 Opstellen van de requirements

Wanneer er een project gaat plaatsvinden, is het van belang om te weten wat er precies moet gebeuren. Het afstudeerproject is niet anders. Voordat er met het project gestart kan worden, moeten hiervoor de requirements worden verzameld, zodat ik en mijn opdrachtgever op één lijn zitten wat betreft het eindresultaat.

Het verzamelen van deze requirements is door middel van een interview gedaan. Ik heb voor deze methode gekozen, omdat ik verwacht hier het meeste informatie uit te verkrijgen. Op het moment dat het interview afgenomen werd, was er voor mij nog veel onduidelijk. Door een open interview te houden, kon mijn opdrachtgever mij veel informatie geven zonder dat ik veel voorkennis nodig had.

Voorafgaand aan het interview ben ik eerst zelf gaan kijken naar mogelijke eisen voor het product. Door eerdere ervaring en inzichten kon ik al een aantal eisen opstellen, die voor veel webapplicaties van toepassing zijn. Deze kon ik vervolgens met mijn opdrachtgever bespreken aan het einde van het interview. De gedetailleerde eisen over het product zelf heb ik tijdens het interview verzameld.

5.1 Requirements verzamelen voorafgaand van het interview

Bij mijn vorige werkgever en binnen school ben ik in aanraking gekomen met webapplicaties. Met de globale opzet hiervan ben ik hierdoor al bekend. Binnen veel webapplicaties zijn dezelfde elementen aanwezig, en deze applicatie is hier geen uitzondering op.

Een van de meest gebruikte elementen is het inloggen van gebruikers. Gebruikers kunnen inloggen binnen de applicatie, waardoor deze weet wie er met het systeem werkt. Een gebruiker kan conflicten aanmaken, en betrokkenen binnen het conflict ook bij het conflict binnen de applicatie aanmelden. Deze handelingen zijn privé voor de betrokkenen. Door in te loggen kan dit voor de buitenwereld worden afgeschermd.

Voor inloggen zijn verschillende middelen beschikbaar. Het bekendst is het inloggen met gebruikersnaam en wachtwoord. Een andere vorm van inloggen die tegenwoordig opkomt is het inloggen via een andere dienst, zoals Google. Hierdoor hoeft de gebruiker geen eigen inloggegevens te onthouden, en gebeurd het inloggen via een veilig kanaal. De keuze voor de manier van inloggen heb ik bij het interview aan de opdrachtgever voorgelegd.

Het product is commercieel van aard. Voordat gebruikers een conflict kunnen oplossen, moeten ze hier een betaling voor verrichten. Voor het verrichten van betalingen zijn verschillende mogelijkheden. Deze moeten echter wel aansluiten bij de applicatie. Als de software als een dienst wordt aangeboden, is de reactietijd van belang. Als een gebruiker een conflict wil oplossen, maar hiervoor eerst een week moet wachten voordat de betaling verwerkt is, is dit niet klantvriendelijk, en past dit ook niet goed binnen het type dienst wat we aanbieden. Een betaalmethode waarbij de betaling onmiddellijk verwerkt kan worden is dus van belang.

Ik heb twee methoden gevonden die hieraan voldoen, iDeal en Paypal. iDeal is een Nederlandse dienst die rechtstreeks aan de bank gekoppeld is. Betalingen worden dan ook gelijk afgeschreven. Paypal is meer te vergelijken met een creditcard. Op het moment van betalen schiet Paypal de betaling voor, en schrijft deze op een later moment af. Ook is het met paypal mogelijk

creditcardbetalingen uit te voeren. Bij beide diensten worden de betalingen binnen een beveiligde omgeving van de dienst uitgevoerd. Zowel iDeal als Paypal zijn geschikt voor de applicatie, en in het meest gunstige geval zal de applicatie ze ook beide ondersteunen.

Binnen de applicatie zijn er meerdere betaalmodellen te realiseren. Hierbij is er onderscheid te maken in aanschaf en abonnement. Bij aanschaf koopt de gebruiker bij wijze van spreken de mogelijkheid om conflicten op te lossen, bij een abonnement betaald hij een bedrag per periode om gebruik te kunnen maken van de dienst.

Binnen het aanschafmodel zijn er drie modellen te onderscheiden. Bij het eerste model moet de gebruiker een betaling verrichten bij het aanmaken van zijn account. Voordat hij een betaling verricht heeft kan hij dus nog geen gebruik maken van het systeem. Het tweede model laat de gebruiker wel een account aanmaken, maar voordat hij conflicten kan aanmaken, moet hij hiervoor het recht eerst kopen. Het derde model lijkt hierbij op het tweede model, wederom moet de gebruiker betalen voordat hij een conflict kan oplossen. Hierbij gaat het betalen per conflict, dus meer zoals het kopen van een product in een webwinkel. Welke betaalmethode voor de applicatie het meest interessant en van toepassing wordt met de opdrachtgever besproken.

Een punt wat voor gebruikers belangrijk is, is privacy. Het idee dat iemand over de verbinding kan meeluisteren naar wat de gebruiker doet is natuurlijk niet gewenst. Wanneer de gebruiker met privacygevoelige informatie werkt, zal dit over een beveiligde verbinding moeten gebeuren om te voorkomen dat het internetverkeer door een derde kan worden onderschept.

Ten slotte heb ik gekeken naar reclame. Binnen de applicatie zal een beperkte hoeveelheid reclame voorkomen. Deze is zo veel mogelijk gericht op de wensen van de gebruiker. Als er bijvoorbeeld uit de boedelverdeling blijkt dat de gebruiker zijn bank en eettafel moet afstaan, is een advertentie voor de uitverkoop bij de IKEA van relevantie voor de gebruiker.

De advertenties moeten dus helder, relevant en duidelijk herkenbaar zijn. Om de gebruiker zoveel mogelijk te ondersteunen moeten de advertenties binnen de stijl van de pagina vallen, zodat ze niet afleidend zijn. Ook moet het duidelijk zijn dat het om een advertentie gaat, om te voorkomen dat de gebruiker zich misleid voelt als hij op een link klikt en het een advertentie blijkt te zijn. Op deze manier kunnen we de gebruikers toch een goede ervaring bieden ondanks dat hij advertenties te zien krijgt. Welke aanbieder voor advertenties er gebruikt gaat worden, zal op een later moment worden vastgesteld, wanneer de advertenties daadwerkelijk gerealiseerd worden.

Na deze voorbereiding ben ik met de opdrachtgever in gesprek gegaan, om deze en overige requirements door te spreken en helder te krijgen.

5.2 Het interview

Halverwege de tweede week heb ik een interview uitgevoerd met de opdrachtgever, waarin de opdrachtgever verteld heeft wat zijn wensen bij de opdracht zijn. Hierbij is er eerst kort ingegaan op het platform, en hebben we vervolgens een conflict besproken.

De opdrachtgever gaf aan dat hij blij was met de voorbereiding van de requirements die ik had opgesteld, omdat dit in grote lijnen aangaf wat het platform moest kunnen. Tijdens het verloop van het project zou het kunnen dat er meer onderdelen naar dit algemene onderdeel van de applicatie

verschoven zouden worden, maar zijn eerste indruk was dat ik de basis van het platform in het rapport had omvat.

Ook is er gesproken over de manier waarop gebruikers kunnen inloggen. Binnen Cenosco was er al ervaring met het inloggen via een derde partij zoals Google. Hiervoor wordt Claim Based Identity gebruikt, en mijn opdrachtgever adviseerde mij sterk om naar deze manier van inloggen te kijken.

Hierna hebben we gesproken over de mogelijkheid tot het oplossen van een conflict. De opdrachtgever had de wens om het conflict boedelverdeling beschikbaar te maken binnen de applicatie, omdat dit een type conflict is waarbij software een goede ondersteuning kan bieden. Bij het verdelen van de boedel kan enkel een goede structuur al een enorme aanwinst zijn. Tijdens dit gesprek is de manier waarop de opdrachtgever het oplossen van dit type conflict voor ogen zag besproken, waardoor ik inzicht in het verloop hiervan heb gekregen.

5.3 Verwerken van de requirements

Na dit gesprek kon ik de requirements officieel gaan vastleggen. Hierbij heb ik zowel gekeken naar functionele als non-functionele requirements.

Ik had er in eerste instantie voor gekozen om de functionele requirements door middel van use cases in kaart te brengen, omdat deze in detail weergeven welke handelingen de gebruiker op het systeem kan uitvoeren.

Tijdens het opstellen van deze requirements kwam ik er echter achter dat de details die me in eerste instantie goed leken, toch een probleem bleken te zijn. Omdat de opdracht is voortgekomen uit een idee, en niet uit een concrete wens of probleem, waren deze details nog niet bekend. Er was wel te zeggen welke dingen de applicatie moest kunnen doen, alleen waren de precieze details hiervan niet bekend. Hierdoor heb ik ervoor gekozen om van use cases af te stappen, en ben ik overgestapt op user stories.

Deze user stories heb ik prioriteiten gegeven waarbij ik de MoSCoW-methode gebruik. MoSCoW staat voor de afkortingen ***M**ust have*, ***S**hould have*, ***C**ould have* en ***W**on't have*. De letter o is hiertussen geplakt als ezelsbruggetje.

Door gebruik te maken van deze afkortingen, is het duidelijk voor welk stadium de requirements bedoeld zijn. Requirements die gemarkeerd zijn met Must have, hebben een hoge prioriteit voor de initiële versie van het project, en requirements die een lagere MoSCoW-prioriteit hebben komen in een latere iteratie terug.

Vanaf mijn sollicitatie voor deze afstudeeropdracht was het al bekend dat het product veel groter was dan ik in mijn tijd bij Cenosco zou kunnen bouwen. Hier heb ik bij het verzamelen van de requirements rekening mee gehouden. Ik heb me niet enkel beperkt tot de requirements waar ik me mee bezig ging houden, maar heb de requirements breder dan dat verzameld.

Omdat de eerste versie van het product ook nog niet klaar is wanneer ik mijn afstudeerstage afrond, heb ik de MoSCoW-prioriteiten ook niet beperkt tot mijn project. Ik heb gekeken welke requirements minstens geïmplementeerd moeten zijn voordat er een eerste versie van het product kan worden gereleased. Deze eisen heb ik de Must have-prioriteit gegeven. Aan het einde van mijn afstudeerstage zijn dan ook niet alle Must have-requirements geïmplementeerd.

Voor non-functionele requirements heb ik gebruik gemaakt van een sjabloon welke binnen Cenosco ook gebruikt wordt voor dit type requirements. In dit sjabloon waren een groot aantal non-functionele requirements opgesomd, zoals maintainability. Een deel van deze non-functionele requirements waren voor mijn project niet van toepassing, door de manier van hosten in de cloud, of omdat deze niet van toepassing waren op een webapplicatie. Het sjabloon heeft me bij de overige non-functionele requirements wel veel houvast kunnen bieden.

Omdat ik zelf nog nooit gebruik gemaakt had van de cloud, stond ik hier kritisch tegenover. Dit klonk mij veelal als een hype in de oren, waarbij alle bedrijven tegenwoordig *in de cloud* willen zitten. Nadat ik me verder in Windows Azure verdiept had, werd ik hier echter steeds enthousiaster over. Voor onze applicatie is de cloud erg interessant, omdat deze makkelijk op te schalen is en niet vast zit aan vaste serverhardware.

Stel dat er geen gebruik gemaakt wordt van de cloud, en voor de applicatie een simpele server wordt aangeschaft. Als blijkt dat de applicatie meer wordt gebruikt dan in eerste instantie verwacht werd en te zwaar is voor de server, moet er een nieuwe server worden aangeschaft. Omgekeerd geldt hetzelfde. Als er een zware server wordt aangeschaft en de applicatie wordt nauwelijks gebruikt is de server een erg grote onnodige investering geweest.

Met de cloud is dit probleem er niet. De server is virtueel, wat inhoudt dat deze niet vast zit aan echte hardware, maar dat de server en zijn hardware worden gesimuleerd binnen een veel grotere server. Wanneer de applicatie te zwaar blijkt te zijn voor de huidige configuratie, kan deze worden opgeschaald, waardoor de virtuele server meer rekenkracht krijgt. Dit zou bij een klassieke server hetzelfde effect hebben als een zwaardere server neerzetten. Dit opschalen van de virtuele server wordt allemaal bij de hoster, in dit geval Microsoft, geregeld. Bij het opschalen is Cenosco dus nauwelijks betrokken wat veel tijd scheelt.

Een ander voordeel is dat er geen echte machines zijn die kunnen falen. De virtuele servers staan opgeslagen op meerdere plekken in het datacenter. Als er een server zou falen neemt een ander het over, zonder dat de klant daar iets van merkt. Door gebruik van de cloud wordt het gehele serverbeheer dus uit handen gegeven.

Een nadeel is echter wel dat je de hosting uit handen geeft, omdat je deze bij een andere partij plaatst. Microsoft geeft echter aan dat de data die in de cloud wordt geplaatst binnen de regio blijft waar deze geplaatst is. In ons geval zal de data Europa niet verlaten. Ook geeft Microsoft aan zich aan de Data Privacy Richtlijn te houden, die door het Europees Parlement is opgesteld om de privacy van gebruikers te waarborgen. [1]

In eerste instantie was ik kritisch over de cloud, en wist ik niet of deze wat kon bijdragen aan het project. Nu ik hier meer informatie over opgezocht heb, sta ik achter het besluit van de opdrachtgever om de applicatie in de cloud te hosten.

[1] http://www.windowsazure.com/en-us/support/trust-center/privacy/

6 De eerste sprint: inloggen en registreren

6.1 De opbouw van de sprint

Tijdens het interview met de opdrachtgever adviseerde deze mij om voor het inloggen van de gebruikers te kijken naar een techniek die Claim Based Identity heet. Microsoft heeft hiervoor een component gemaakt dat de authenticatie uit handen van de ontwikkelaars neemt, waardoor deze dit niet hoeven te implementeren. Dit component heet Windows Identity Foundation, veelal afgekort tot WIF.

Zelf had ik nog geen ervaring met dit component, of Claim Based Identity in het algemeen. Hierdoor kon ik zelf nog niet beoordelen of deze techniek te gebruiken was binnen de applicatie. Hiervoor moest ik me inlezen op de techniek en het component.

Omdat ik nog geen ervaring met dit component , of de techniek in het algemeen had, was het implementeren hiervan een hoog risico. Dit risico wilde ik zo snel mogelijk verwijderen. Ik kon op dit moment nog geen inschatting doen hoeveel tijd het implementeren zou kosten, en nu in het begin van het project is er nog veel ruimte voor uitloop. Dit is eventueel later te corrigeren.

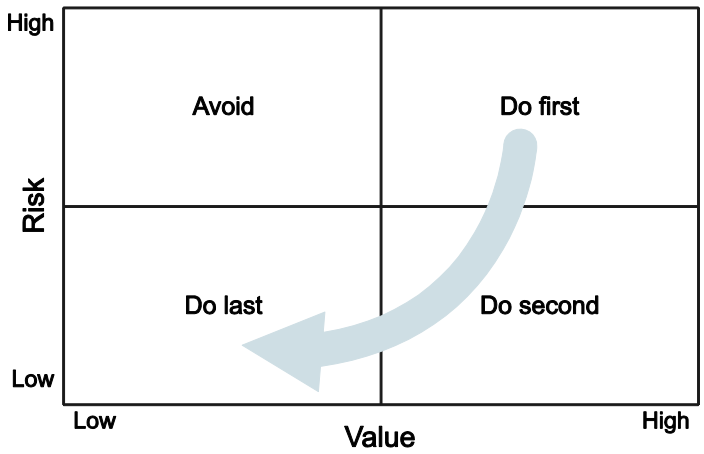
Ook is het inloggen van gebruikers een belangrijk onderdeel van de applicatie. Als gebruikers niet kunnen inloggen kunnen ze geen gebruik maken van het systeem. Daarnaast maakt het ontbreken van een inlogfunctionaliteit het ontwikkelen een stuk lastiger, omdat ik een stuk moeilijker kan testen met verschillende soorten gebruikers.

Binnen Agile-planningen is de relatie tussen risico en belang volgens het boek Agile Estimating and Planning[1] in vier onderdelen te verdelen:

- Hoog risico / laag belang
- Hoog risico / hoog belang
- Laag risico / laag belang
- Laag risico / hoog belang

Hierin wordt geadviseerd de requirements met een hoog risico en hoog belang als eerste op te pakken, daarna de requirements met een hoog belang en een laag risico, en ten slotte de requirements met een laag risico en een laag belang. Geadviseerd wordt om requirements met een hoog risico maar een laag belang over te slaan.

[1] *Agile Estimating and Planning*, Mike Cohn, Prentice Hall, 2006



Als ik dit toepas op de requirements binnen mijn project, valt het inloggen van gebruikers in de hoog risico/hoog belang-groep. Al deze redenen samen hebben ervoor gezorgd dat ik ervoor gekozen heb om de inlogfunctionaliteit helemaal aan het begin, in de eerste sprint in te plannen.

Deze sprint is in te delen in vier activiteiten:

1. Uitzoeken hoe Claim Based Identity en WIF werken, door middel van een prototype.
2. Het opzetten van een basis binnen de applicatie, waarin bijvoorbeeld databaseverbindingen kunnen worden gebruikt.
3. Het opzetten van gebruikers, rollen en privileges binnen de applicatie.
4. Inloggen en registreren implementeren binnen de applicatie.

Om er achter te komen hoe WIF en Claims Based Identity werken, heb ik ervoor gekozen om dit uitzoeken via een prototype te doen, zodat ik me minder zorgen hoefde te maken over de netheid waarop ik werkte. Het idee van het prototype is dat ik de inzichten mee neem naar de echte applicatie, maar de code niet.

Omdat de werking van WIF invloed heeft op het ontwerp voor het inloggen van gebruikers binnen de applicatie, heb ik ervoor gekozen om dit eerst te doen. Hierna heb ik een basis opgezet die ik voor de gehele applicatie gebruikt kan worden, waarbinnen dingen als de databaseverbindingen worden geregeld. Vervolgens kon ik een ontwerp voor de applicatie en de database maken, met betrekking tot het gebruikersonderdeel. Ten slotte kon ik Claim Based Identity implementeren binnen de applicatie.

6.2 Claim Based Identity

6.2.1 Wat is Claim Based Identity?

Het idee bij Claim Based Identity is dat het inloggen van gebruikers wordt uitbesteed naar een dienst die vertrouwd wordt door de applicatie. Inloggen gebeurd dus niet binnen onze applicatie, maar bij een externe dienst. Deze externe dienst doet vervolgens een aantal uitspraken (claims) over de gebruiker. Deze uitspraken samen worden een token genoemd. Hoe dit allemaal werkt kan ik het beste uitleggen aan de hand van een voorbeeld. Dit voorbeeld is niet informaticagericht, maar schetst wel goed de situatie. Het gaat over het inchecken op het vliegveld voor een vlucht.

Wanneer je een reis met een vliegtuig wilt gaan maken, kan je niet zomaar het vliegtuig in stappen. Voordat je dit kan doen, moet je eerst een ticket kopen. Dit gaat niet zo eenvoudig als het kopen van een kaartje voor de trein, omdat de identiteit van de reiziger gecontroleerd moet worden.

Het vliegveld kan zelf een gegevensbestand bijhouden, waar alle identiteiten van de reizigers in staan. Dit is echter totaal niet praktisch, omdat een reiziger zich eerst moet aanmelden bij het vliegveld als hij een vlucht wil maken. Vertrekt hij vanaf een andere luchthaven, dan moet hij zich ook daar aanmelden. Ook moeten de medewerkers op het vliegveld een manier hebben om te controleren dat de reiziger die het ticket wil kopen wel echt is wie hij zegt dat hij is.

Dit is voor zowel de luchthaven als de reiziger een enorm gedoe. De reiziger moet onthouden bij welke luchthavens hij zich heeft ingeschreven. Heeft hij zich nog niet ingeschreven, dan kan het luchthavensysteem de reiziger niet vinden en hem ook geen ticket uitreiken. Aan de andere kant, als hij zich al ingeschreven heeft maar dit niet meer weet, kost het inschrijven hem veel tijd en moeite.

Voor de luchthaven is het ook lastig, omdat deze een methode moet hebben om de reiziger te herkennen. Als dit op naam gebeurd, is dit erg fraudegevoelig. Een foto als identificatiemiddel is ook geen goede methode, omdat de reiziger sinds de foto gemaakt is naar de kapper geweest kan zijn, zijn baard heeft laten staan of er op de foto een stuk jonger uit ziet dan in het echt. Het identificeren is dus voor de luchthaven ook een complex karwei.

De oplossing voor dit identificatieprobleem is logisch binnen de context van dit probleem, de identificatie wordt namelijk niet door de luchthaven gedaan. Dit besteedt deze uit naar een andere organisatie, namelijk de overheid. Binnen de overheid zijn alle burgers bekend met hun geboortedatum, pasfoto, etc. Als de luchthaven wil weten wie de reiziger is, vragen ze hiervoor een identificatiemiddel die door de organisatie is uitgereikt, namelijk zijn paspoort of identiteitskaart.

Deze documenten zijn voor de luchthaven voldoende om te kunnen zeggen dat de reiziger is wie hij zegt dat hij is. Omdat het document eens in de vijf jaar ververst dient te worden, is deze recent. Ook is makkelijk te zien of de reiziger de eigenaar van het document is door de foto die erop staat. Door de watermerken die erop aanwezig zijn, en de chip die tegenwoordig in het document verwerkt zit, kan er ook worden gecontroleerd of het document echt is.

Nadat de controle van de reiziger is gebeurd, kan deze betalen voor zijn ticket, waarna deze aan hem wordt uitgereikt. Op dit ticket staat voor welke vlucht hij is geboekt, en dat hij hiervoor betaald heeft (zonder te betalen krijgt hij immers geen ticket). Het ticket kan hij vervolgens gebruiken om het vliegtuig in te stappen en zijn reis te maken.

Het is belangrijk dat de luchthaven de organisatie vertrouwt die het document dat de reiziger identificeert uitgereikt heeft, en dat de identiteit voldoende informatie bevat. In de overheid heeft de luchthaven veel vertrouwen omdat deze de reizigers goed kent, omdat deze hier al hun hele leven bekend zijn. Wanneer de reiziger zich echter probeert te identificeren met zijn bibliotheekkaart, zal de luchthaven hier geen genoegen mee nemen. Deze documenten zijn makkelijk te vervalsen door het ontbreken van watermerken, en het aanmelden bij de bibliotheek onder een valse naam is ook relatief gemakkelijk. De luchthaven heeft niet voldoende vertrouwen in de mogelijkheid tot identificatie van de gebruikers met het document van de bibliotheek, en de reiziger kan hier daardoor niet mee reizen. Ook is de informatie die de bibliotheekkaart biedt niet

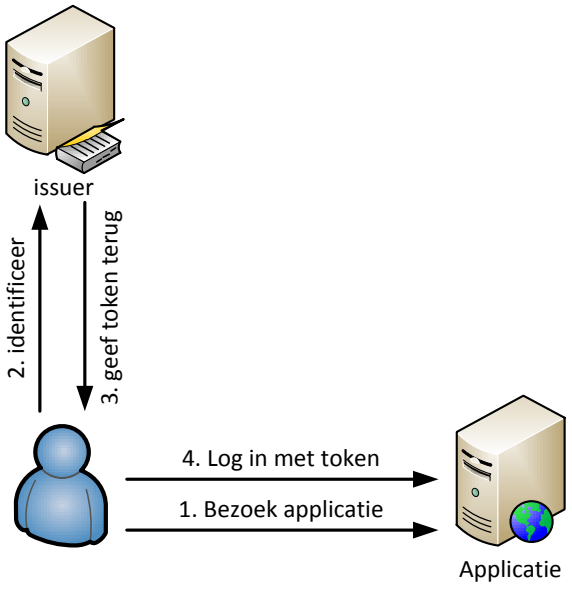
voldoende. Waar ze bij het vliegveld de naam van de reiziger, zijn geboortedatum, het land waar hij geregistreerd staat, etc. willen hebben, bevat de bibliotheekkaart deze informatie niet.

Het voorbeeld klinkt wellicht wat omslachtig, echter is dit wel hoe veel websites en webapplicaties werken. Voordat hier gebruik van gemaakt kan worden, moet er eerst een account aangemaakt worden. Dit leidt kan leiden tot twee scenario's: de gebruiker heeft heel veel wachtwoorden die hij waarschijnlijk niet allemaal kan onthouden waardoor hij ze vergeet, of ergens opslaat, of de gebruiker gebruikt het zelfde wachtwoord bij meerdere websites. Beide scenario's vormen risico's voor de veiligheid.

Bij het gebruik van Claim Based Identity, wordt er ingelogd via een andere dienst. Hierdoor hoeft de gebruiker geen nieuwe gebruikersnaam en wachtwoord aan te maken, omdat de inloggegevens van de dienst gebruikt worden. Het identificeren gebeurd bij deze dienst, waardoor er binnen onze applicatie geen gebruikersnamen en wachtwoorden opgeslagen hoeven te zijn. Het valideren van de gebruikersnaam en wachtwoord gebeurd ook bij de dienst, waardoor de beveiliging hiervan wordt gebruikt. Als ik kijk naar een dienst zoals Google, kan ik de aanname doen dat met het geld en de middelen die ze beschikken ze een betere beveiliging kunnen bieden dan ik zou kunnen in mijn afstudeerperiode.

Het voorbeeld en een echte applicatie die van claims gebruik maakt vertonen nog meer overeenkomsten. De luchthaven is de applicatie in het voorbeeld, de reiziger is een gebruiker binnen de applicatie, en de overheid en de bibliotheek zijn diensten die het inloggen beschikbaar maken, zoals de Microsoft Active Directory. Laatstgenoemde wordt de issuer genoemd, omdat deze tokens verstrekt aan de applicatie.

Wanneer de gebruiker zich aangemeld heeft bij de issuer, stuurt deze een token terug, met hierin een aantal claims over de gebruiker. In het voorbeeld is het token wat de reiziger van de overheid heeft gekregen een paspoort. Hierin staan een aantal claims, zoals de geboortedatum van de reiziger, zijn burgerservicenummer en zijn voor- en achternaam.



De tokens die een gebruiker terugkrijgt van een issuer hebben een zelfde soort informatie. Zo kan er een claim zijn met de naam van de gebruiker, een claim met zijn emailadres, een claim met de afdeling waar hij werkt, en een claim die aan geeft welke rol hij heeft. Welke claims er in de token zitten, hangen af van de issuer. De ene issuer kan het gehele adressenbestand van de gebruiker terugsturen, terwijl de andere issuer enkel een naam en rol terugstuurt. Dit is te vergelijken met het eerder genoemde paspoort en de bibliotheekkaart. Waar er in het paspoort een aantal persoonlijke gegevens staan, staat er op een bibliotheekkaart enkel een klantnummer.

6.2.2 Problemen bij het implementeren van WIF en Claims Based Identity

Tijdens het implementeren van Claims Based Identity binnen het prototype stuitte ik op problemen. Dit komt voornamelijk door de toepassingen van Claims Based Identity en WIF. Waar deze in uitblinken, is het uitbesteden van authenticatie en autorisatie buiten de applicatie. Hier wordt vervolgens binnen de applicatie geen aandacht aan besteed. Rollen en rechten komen van de derde partij af, en worden niet binnen de applicatie verstrekt.

Echter werkt de applicatie niet zo. Binnen de applicatie hebben de gebruikers rollen en rechten, maar buiten de applicatie hebben deze geen context. Wanneer Claims Based Identity gebruikt wordt om een verbinding te maken naar het bedrijfsnetwerk wat gebruik maakt van een Active Directory, kan hieruit gehaald worden dat de gebruiker bij de afdeling Sales werkt, en hier de rechten binnen de applicatie op aanpassen. Deze opzet is bij ons niet aanwezig.

De rollen en rechten van een gebruiker moeten namelijk binnen de applicatie worden bepaald, en hierbij werken Claims Based Identity en WIF niet lekker, omdat deze ervoor gemaakt zijn dit buiten de applicatie te doen.

Hierdoor ging het implementeren van Claims Based Identity erg moeizaam. Na twee weken ploeteren was er nog geen resultaat wat gebruikt kon worden, en stuitte is steeds weer op problemen. Ik vond dat er op dit moment maatregelen genomen moesten worden.

Ik had op dit moment drie mogelijkheden. Ik kon doorgaan met het implementeren van Claims Based Identity met WIF, ik kon op zoek gaan naar een andere bestaande manier van inloggen, of ik kon zelf inloggen implementeren door middel van een gebruikersnaam en wachtwoord.

Doorgaan met Claims Based Identity vond ik geen goede optie, omdat ik al een tijd vastzat, en ik niet verwachtte dat ik opeens een doorbraak zou krijgen en alles op zijn plek zou vallen. Deze optie viel dus af.

Een andere optie is om op zoek te gaan naar een andere bestaande manier van inloggen. Hier voorzag ik echter ook problemen. Ten eerste moest er een passende methode om in te loggen worden gevonden, en deze moest hierna worden geïmplementeerd. Bij het implementeren van de vorige bestaande oplossing, WIF, liep ik tegen problemen aan. Wanneer ik een andere bestaande methode zou gaan gebruiken, was de kans aanwezig dat ik tegen dezelfde problemen aan zou lopen. De zekerheid bij het implementeren van een bestaande oplossing ontbrak. Daarom heb ik hier ook niet voor gekozen.

De derde optie was om zelf het authentiseren van gebruikers te bouwen. Hier had ik al ervaring mee, en wist dus hoe dit geïmplementeerd moest worden. Hierdoor was er veel zekerheid en vertrouwen aanwezig dat dit tot een goed resultaat zou leiden. Doordat ik bij deze methode het authentiseren zelf gebouwd zou worden, was ik niet afhankelijk van externe componenten die niet mee werkten. Ook kon ik goed inschatten hoeveel tijd het implementeren hiervan zou gaan kosten.

Ik heb over deze kwestie een gesprek aangevraagd met mijn begeleider, en hem hierin uitgelegd tegen welke problemen ik stuitte, en mijn bevindingen dat deze techniek wellicht niet helemaal geschikt was voor de applicatie. Ik heb hierbij het advies uitgebracht om een andere manier van inloggen te implementeren, waarbij de gebruiker een gebruikersnaam en wachtwoord gebruikt om zich te identificeren.

Aan het einde van dit gesprek hebben mijn begeleider en ik besloten dat een eigen authenticatiecomponent de beste mogelijkheid was, om bovengenoemde redenen, en omdat ik al vrij veel tijd kwijt geraakt was aan de pogingen om WIF te implementeren.

De volgende stap is dus om een eigen authenticatiemodule te schrijven. Voordat dit echter mogelijk was, moest er eerst een basisopzet van de applicatie worden gemaakt waarbinnen deze module geplaatst kon worden.

6.3 De basisopzet van de applicatie

In dit onderdeel beschrijf ik de basisopzet van de applicatie. Hierin worden dingen geregeld zoals de databaseverbinding. Dit onderdeel staat los van het inloggen en registreren, alleen maken deze hier wel gebruik van.

6.3.1 Interface naar de database

De applicatie gaat veelvoudig gebruik maken van een database, waarin dingen als de klanten en de conflicten opgeslagen zijn. De database benaderen door middel van queries is echter geen goed idee.

Ten eerste kost deze manier van queries uitvoeren veel tijd. Voor elk type data wat moet worden opgevraagd moet een eigen query worden samengesteld. Deze moet handmatig worden uitgetypt, gecontroleerd, etc. Vervolgens moet het resultaat van de query geconverteerd worden naar objecten die binnen de applicatie te gebruiken zijn.

Ten tweede neemt deze manier van queries uitvoeren beveiligingsrisico's met zich mee. Een van de meest gebruikte methoden om een website te hacken is SQL-injectie. Wanneer er een query op een database wordt uitgevoerd, is hier meestal input voor nodig van een gebruiker. Als een gebruiker bijvoorbeeld een nieuwsbericht op nu.nl wil bekijken wordt dit bericht opgevraagd aan de hand van een uniek nummer. Dit nummer komt vervolgens in de query waarmee het nieuwsbericht wordt opgevraagd. Een gebruiker kan dit nummer in het adres van de pagina aanpassen. Als dit nummer, of andere invoer onvoldoende wordt gecontroleerd, is het mogelijk dat de gebruiker zelf commando's op de database kan uitvoeren. Zo is het mogelijk om bijvoorbeeld alle gebruikers die in de database staan op te vragen, of een tabel te legen.

Binnen de informaticawereld zijn er componenten beschikbaar, die deze twee problemen aanpakken. Deze worden ORMs (Object-Relational Mappers) genoemd. De taak van zo'n ORM is om een tussenlaag te vormen tussen de objecten binnen de applicatie, en de relationele database.

De ORM moet eerst worden geconfigureerd, zodat deze weet hoe de database en de objecten eruit zien. Deze worden op elkaar *gemapped*. Hierbij wordt aangegeven welk veld in het object overeen komt met welke kolom in de database, en welke tabel er bij een klasse hoort.

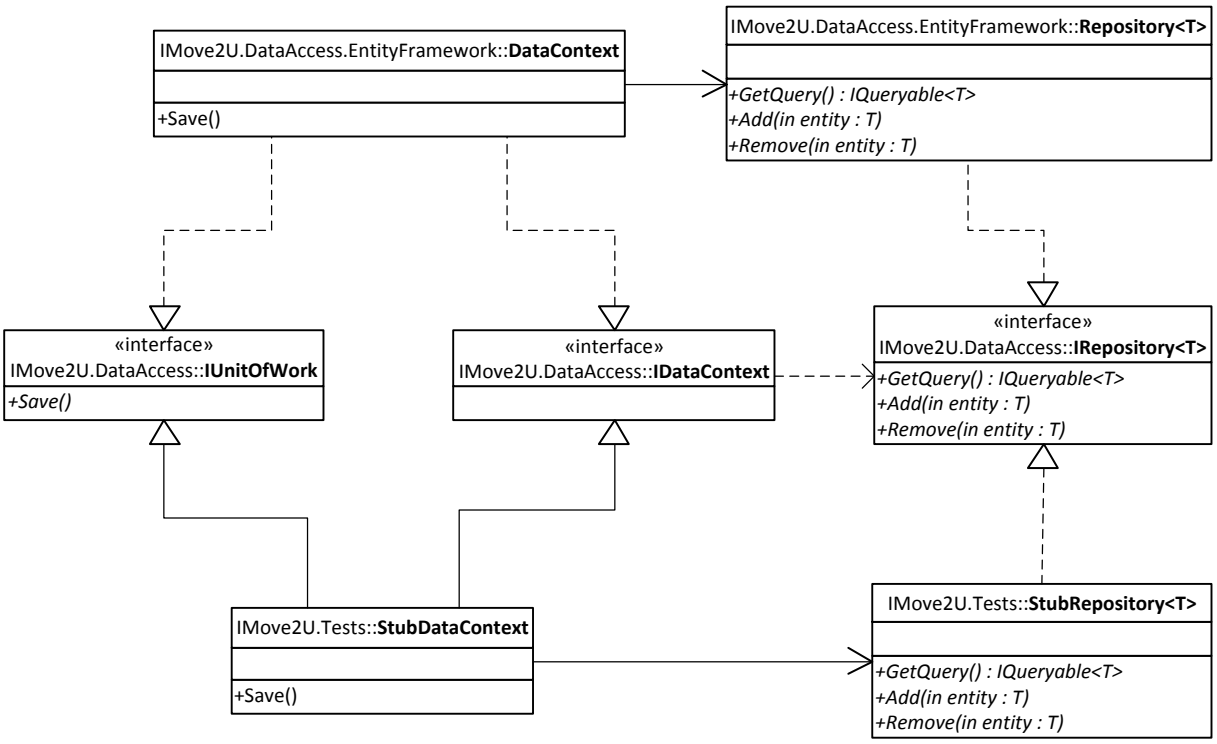
Als het configureren en mappen is gebeurd, is het mogelijk om via de ORM objecten op te halen uit de database. De objecten hoeven dus niet zelf in elkaar gezet te worden aan de hand van de rijen in de database, dit wordt namelijk automatisch door het ORM gedaan. Binnen de applicatie hoeft er hierdoor, buiten de configuratie om, geen rekening gehouden te worden met het type database, alles wordt namelijk als objecten opgehaald.

Er zijn verschillende van dit soort componenten beschikbaar. Ik heb ervoor gekozen om Entity Framework te gebruiken. Dit is een component wat Microsoft gemaakt heeft, wat goed gecombineerd kan worden met ASP.NET MVC. Ook had ik al ervaring met dit ORM, waardoor ik niet eerst de werking heb hoeven uitzoeken.

Het was wenselijk dat er een lage koppeling tussen het ORM en de applicatie aanwezig was, zodat het ORM kan worden vervangen voor een andere implementatie. Een toepassing waar dit wordt gebruikt is bij unit testen. Er wordt hierbij niet getest tegen een implementatie waarbij de database wordt gebruikt, maar tegen een stub.

Een stub is een soort van nepimplementatie van het object. Deze implementatie kan volledig worden geconfigureerd zoals de tester wil, zo kan er bijvoorbeeld precies worden ingesteld welke testdata de stub moet teruggeven. Op deze manier kunnen onderdelen worden getest, zonder dat er een afhankelijkheid aan de database is.

Om deze koppeling te realiseren, heb ik een abstractie gevormd bovenop het ORM door middel van een aantal interfaces. Deze implementatie ziet er zo uit:



Om een collectie van objecten in de database te beheren, heb ik ervoor gekozen om het Repository-pattern te gebruiken. Een repository is een object wat tussen de collectie van objecten in de database en de applicatie staat. Als de applicatie een of meerdere objecten uit de database wil hebben vraagt hij deze op uit de repository. Ook kunnen er middels de repository objecten aan de database worden toegevoegd, of hiervanuit verwijderd.

Microsoft heeft als onderdeel van het .NET framework een querytaal ontwikkeld die LINQ (Language-Integrated Query) heet. Met deze taal is het mogelijk om een soort van queries uit te

voeren op objecten. Deze taal is ook geïntegreerd in het Entity Framework, waardoor het mogelijk is met deze taal queries uit te voeren op de database.

Door gebruik te maken van generics is het mogelijk de repositories generiek op te zetten. Generics is een methode om de implementatie los te koppelen van een bepaalt type. Een voorbeeld waarbij dit veel wordt gebruikt is de List binnen .NET of de ArrayList binnen Java. Wanneer een ontwikkelaar gebruik wil maken van deze List, hoeft hij geen aparte implementatie te maken voor elk type object wat hij in de lijst wil plaatsen, maar kan hij gebruik maken van generics. In de praktijk zit dit er zo uit:

```
// Een lijst met personen.
List<Person> people = new List<Person>();
// Een lijst met bedrijven.
List<Company> companies = new List<Company>();
```

Entity Framework gebruikt ook generics voor zijn LINQ-queries en methoden. Door deze generic collecties te gebruiken in de implementatie van mijn repository, hoef ik maar een enkele repository-implementatie te programmeren, welke voor elk type entity gebruikt kan worden. Het gebruik van de repositories is hierdoor erg flexibel.

Entity Framework maakt gebruik van het Unit of Work-pattern. In dit pattern is er een soort van manager die alle repositories beheert. Objecten kunnen worden toegevoegd of verwijderd vanaf de repository. De Unit of Work houdt deze wijzigingen bij, en vervolgens kunnen al deze wijzigingen door middel van een Save-operatie worden weggeschreven naar de database toe. Dit heb ik in de abstractie verwerkt middels de IUnitOfWork-interface.

De DataContext wordt gebruikt om toegang te bieden tot de repositories. In de IDataContext worden de verschillende repositories beschikbaar gemaakt door middel van getters. Omdat de repositories gekoppeld zijn aan de Unit of Work die Entity Framework gebruikt, is het niet mogelijk om een simpele instantie aan te maken van de repository, omdat deze onderdelen van Entity Framework nodig hebben. Daarom worden deze repositories geïnstantieerd in de DataContext, en door middel van getters beschikbaar gesteld.

Zoals in het bovenstaande diagram te zien is, implementeren zowel de Entity Framework-versie van de data accesslaag als de stubversie de interfaces. Binnen de echte applicatie zal de Entity Framework-implementatie worden gebruikt, en binnen de tests de stubversie.

6.3.2 Inversion of Control & Dependency Injection

Zoals ik in het vorige onderdeel beschreef, heb ik een abstractie gebouwd die gebruikt wordt wanneer er uit de database moet worden opgevraagd moet worden toegevoegd of moet worden verwijderd. Door deze abstractie hoeft de applicatie niet te weten tegen welke implementatie hij praat, de databaseimplementatie, of de stubimplementatie die gebruikt wordt bij het testen.

Het probleem is echter dat er wel ergens een instantie van deze concrete implementatie gemaakt moet worden. Wanneer dit wordt gedaan op het moment dat bijvoorbeeld de repository gebruikt gaat worden, is de hele abstractielaag in principe overbodig, omdat door het aanmaken van de instantie deze alsnog sterk wordt gekoppeld aan een implementatie.

Daarom heb ik ervoor gekozen om gebruik te maken van Inversion of Control en Dependency Injection.

Het idee achter Inversion of Control is dat een component geen koppeling heeft met een concrete implementatie, maar enkel de abstractie hiervan kent. De koppeling naar de daadwerkelijke implementatie gebeurd niet in het component zelf, maar dit gebeurd elders. Hierdoor heeft het component geen afhankelijkheden naar de concrete implementatie, en kan deze makkelijk worden gewisseld.

Door de abstractie over de database te definiëren, is een deel van dit principe al vervuld. Binnen het component wordt er geen gebruik gemaakt van de concrete implementatie, maar wordt er tegen een abstractie gesproken. Het andere deel van het principe, het koppelen met de daadwerkelijke implementatie doe ik met behulp van Dependency Injection.

Dependency Injection is een manier om een implementatie bekend te maken bij het component. Wanneer een component zelf een instantie van een object aanmaakt, *vraagt* deze bij wijze van spreken om een concrete implementatie. Dependency Injection werkt andersom. Hierbij wordt de implementatie het component in geïnjecteerd. Vandaar ook de naam Dependency Injection. In plaats van de afhankelijkheid naar de concrete implementatie in het component te regelen, wordt de afhankelijkheid naar de concrete implementatie elders geregeld en het component in geïnjecteerd.

Hoe dit in de praktijk werkt, is het beste te tonen met een voorbeeld. Stel we hebben een klasse die gebruikt kan worden om alle personen in de database een mail te sturen. Deze klasse heeft een repository nodig waaruit de personen gehaald kunnen worden. Een deel van deze klasse staat hieronder.

```
public class Mailer
{
    private readonly IRepository<Person> personRepository;

    public Mailer()
    {
        this.personRepository = new DatabaseRepository<Person>();
    }
}
```

Zoals te zien is, heeft de klasse een variabele van het type `IRepository<Person>`. Er wordt dus gebruik gemaakt van een interface, en niet van een concrete implementatie. Er kan echter geen instantie van de interface worden gemaakt, dit moet van een concreet type zijn. Deze instantie wordt hier in de constructor gemaakt, op regel zeven.

Door het maken van deze instantie, wordt de klasse sterk gekoppeld aan de specifieke instantie, hier van het type `DatabaseRepository<Person>`. Er kan nu dus niet worden gewisseld naar een ander type repository, zonder de bestaande code aan te passen. Bijvoorbeeld testen met een stubrepository is hierdoor niet mogelijk.

```
public class Mailer
{
```

```
    private readonly IRepository<Person> personRepository;

    public Mailer(IRepository<Person> personRepository)
    {
        this.personRepository = personRepository;
    }
}
```

In dit voorbeeld wordt door middel van Dependency Injection de repository beschikbaar gemaakt aan de Mailer-klasse. De klasse weet niet wát voor repository dit is, echter is dit voor het functioneren ook niet van belang. De klasse hoeft enkel te weten dát er een repository is. Van welk type implementatie deze repository is, is voor de klasse niet van belang.

Bij het gebruik van de Mailer-klasse in de applicatie wordt de databaseimplementatie van de repository meegegeven via de constructor en wanneer de Mailer-klasse wordt getest, wordt de stubimplementatie van de repository meegegeven. Zo is er geen enkele afhankelijkheid naar de concrete repository toe.

Ditzelfde principe gebruik ik ook binnen de applicatie.

Om te ondersteunen bij Dependency Injection, kan er gebruik gemaakt worden van een Inversion of Control-container, kort een IoC-container. Deze containers bezitten kennis om concrete instanties te maken van een interface. Vanuit de code wordt er aan de container gevraagd om een instantie te maken van het type `IRepository<Person>`, en de container is intelligent genoeg om te weten dat hier de implementatie `DatabaseRepository<Person>` bij hoort.

Om dit te kunnen, moeten deze abstracte en concrete klassen eerst worden aangemeld bij de container. Hieronder staat een voorbeeld hoe dit in de praktijk werkt, met behulp van de IoC-container Unity:

```
IUnityContainer IoCContainer = new UnityContainer();
IoCContainer.RegisterType<IRepository<Person>, DatabaseRepository<Person>>();
```

```
IRepository<Person> repository = IoCContainer.Resolve<IRepository<Person>>();
```

In het eerste voorbeeld is te zien dat de interface `IRepository<Person>` binnen de container geregistreerd wordt aan het type `DatabaseRepository<Person>`. In het tweede voorbeeld wordt er een instantie van `IRepository<Person>` opgevraagd aan de container. De container weet dat bij deze interface de `DatabaseRepository<Person>`-implementatie hoort, en maakt een nieuwe instantie aan van dit type.

Dit werkt ook recursief. Stel dat we boven de `Mailer`-klasse uit het vorige voorbeeld de interface `IMailer` hangen, en ook deze op de container registreren. Vervolgens halen we dit type uit de container met de `Resolve`-methode. De container ziet bij het aanmaken van de Mailer-klasse dat deze in de constructor een argument van type `IRepository<Person>` wilt hebben. Om dit argument te leveren kijkt de container in zijn geregistreerde types of hier een `IRepository<Person>` in zit, en als dit het geval is geeft hij deze als parameter mee aan de constructor.

6.3.3 ASP.NET MVC

Binnen de applicatie wordt er gebruik gemaakt van ASP.NET MVC 3. Dit is een framework wat het Model View Controller-pattern implementeert, en is bovenop ASP.NET gebouwd. Binnen dit framework wordt er gebruik gemaakt van het .NET-framework.

Het MVC-pattern brengt een scheiding aan tussen de logica van de applicatie, en de grafische weergave hiervan. Doordat de logica losstaat van de grafische weergave is het bijvoorbeeld mogelijk om deze functionaliteit te testen.

Binnen het MVC-pattern zijn er drie rollen, het model, de view en de controller. Het model omvat een stukje aan logica binnen de applicatie. De controller bereidt het model voor, en geeft dit vervolgens aan de view. De view zorgt vervolgens voor de grafische weergave van het model.

Als voorbeeld kunnen we een webpagina nemen, die een lijst van medewerkers toont. Hier beginnen we bij de controller, waar het verzoek van de pagina binnen komt. De controller maakt verbinding met de database, en haalt de lijst van de medewerkers op. Deze plaatst hij vervolgens in het model. Vervolgens geeft de controller het model aan de view, en geeft opdracht deze te tekenen.

Op het moment dat een gebruiker op een pagina aankomt, wordt er automatisch een controller aangemaakt. Dit wordt gedaan op basis van het webadres van de pagina.

Een pagina binnen ASP.NET MVC heeft de volgende opbouw: `http://localhost/{controller}/{action}`. Hierin is {controller} de naam van de controller, en {action} de naam van de methode die op de controller wordt aangeroepen. Als voorbeeld nemen we de volgende controller:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Wanneer het adres `http://localhost/Home/Index` wordt bezocht, wordt er een instantie van de `HomeController` gemaakt, en vervolgens wordt hierop de `Index`-methode aangeroepen.

Binnen ASP.NET MVC is het resultaat van een action altijd een `ActionResult`. Dit is een pakketje waarin precies in aangegeven wat er moet gebeuren. Zo kan deze `ActionResult` een `ViewResult` zijn als er opdracht gegeven is om een view weer te geven. Ook is het mogelijk om vanuit een action opdracht te geven om te redirecten naar een andere pagina of internetadres. Er wordt dan een `RedirectResult` terug gegeven.

Het bovenliggende framework vertaald dit vervolgens naar iets concreets. Wanneer de `ActionResult` een instantie is van `ViewResult`, zal dit een webpagina opleveren, inclusief de nodige http-codes. Bij een `RedirectResult` worden de juiste http-codes meegegeven die nodig zijn om naar een andere pagina te verwijzen, etc.

Zoals ik hierboven schreef maak ik gebruik van dependency injection. Om deze dependencies in de controller te krijgen, moeten ze via de constructor worden meegegeven aan de controller, bij het aanmaken hiervan. Deze controllers worden echter automatisch opgebouwd aan de hand van het webadres, en bij de vorm van opbouwen die gebruikt wordt, kunnen er enkel controllers gebouwd worden die constructors zonder parameters hebben.

Gelukkig hebben ze bij de derde versie van ASP.NET MVC hier rekening mee gehouden, door middel van een `IDependencyResolver`-interface. Dit is een component wat gebruikt kan worden om een eigen controller mee op te bouwen.

Wanneer een controller opgebouwd wordt, wordt er aan de hand van het pad een typedefinitie opgesteld. Doordat controllers bij ASP.NET MVC in de *Controllers*-map in de hoofdmap van de webapplicatie zitten, kunnen de typedefinities aan de hand van deze conventies worden opgesteld.

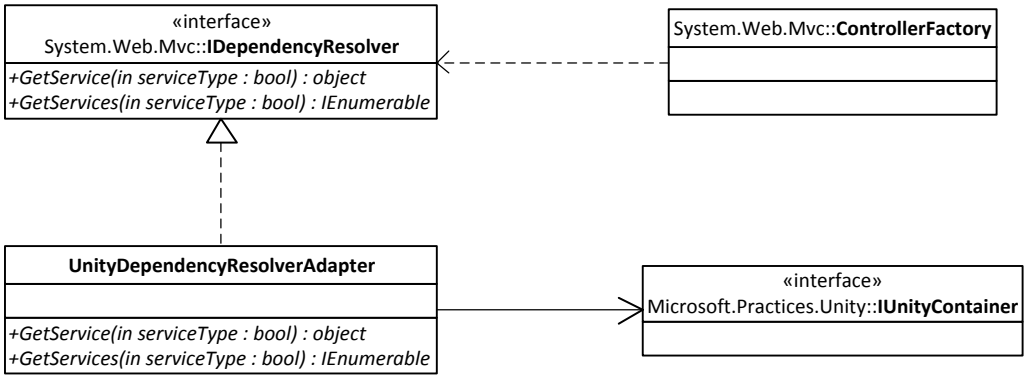
Vervolgens wordt er aan de `IDependencyResolver` om een instantie van de typedefinitie gevraagd. De `IDependencyResolver` maakt vervolgens het concrete type aan, en geeft dit terug.

Het is mogelijk om de standaard `IDependencyResolver` te overschrijven door een eigen type wat de interface implementeert. Dit was mijn ingang om controllers die gebruik maken van dependency injection te instantiëren.

Om zelf controllers op te bouwen aan de hand van een type, heb ik ervoor gekozen om een Inversion of Control-container te gebruiken, en gebruik te maken van het Adapter-pattern om deze IoC-container en de `IDependencyResolver` te combineren.

Ik heb voor Unity gekozen omdat ik al bekend was met deze IoC-container, en uit ervaring kon zeggen dat deze perfect geschikt was voor de toepassingen waarvoor ik hem wilde gebruiken. Ook is er op internet veel informatie beschikbaar over hoe Unity binnen ASP.NET MVC geïmplementeerd kon worden.

De implementatie van deze opzet ziet er als volgt uit:



Op het moment dat er nu een controller aangemaakt moet worden, wordt er aan Unity om een instantie gevraagd. Unity probeert het type aan te maken, en ziet dat deze constructorargumenten nodig heeft. Vervolgens kijkt hij in zijn eigen container of hij de beschikking heeft tot deze types, en haalt hij deze op. Ten slotte maakt unity de controller aan door de net opgehaalde types als constructorargumenten mee te geven.

Om dit te laten werken, moeten de typen wel bekend zijn binnen Unity. Dit is geen probleem, omdat deze bij het opstarten van de applicatie geregistreerd kunnen worden bij de Unity container. Om hierbij te helpen heb ik een helperklasse `UnityDependencyRegistrator` gemaakt, waarin de types worden geregistreerd op de Unity container. Op deze manier wordt het registreren van de types gescheiden gehouden van het opstarten van de applicatie.

6.4 Gebruikers, rollen en privileges

6.4.1 Rollen en privileges

Om gebruik te maken van de applicatie, moet een gebruiker eerst inloggen om zich binnen de applicatie kenbaar te maken. Zijn handelingen worden immers opgeslagen, en deze moeten aan een gebruiker worden gekoppeld.

Binnen de applicatie kan een gebruiker verschillende privileges hebben. Zo kan een gebruiker bijvoorbeeld de privilege hebben om een conflictoplossing op te starten. Deze privileges moeten aan de gebruikers worden gekoppeld.

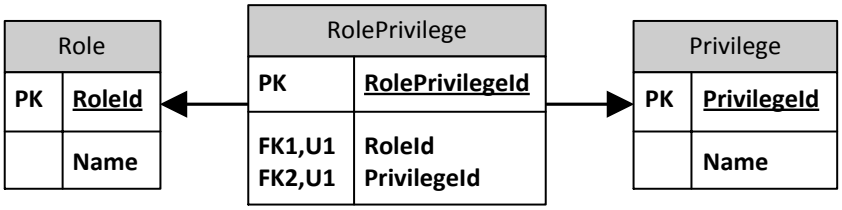
Een volledige set aan privileges koppelen is buiten dat het veel werk is, ook erg onoverzichtelijk. Daarom heb ik ervoor gekozen om hier een tussenlaag in te brengen, in de vorm van een rol.

Een rol is letterlijk wat het zegt, een rol binnen de applicatie. Zo kan een gebruiker bijvoorbeeld de rol *gebruiker* bezitten, of de rol *administrator*. Omdat deze rollen specifieke privileges binnen de applicatie met zich meebrengen, een administrator kan bijvoorbeeld geregistreerde gebruikers bekijken, vergemakkelijkt dit het beheer van de privileges enorm. Wanneer een nieuwe gebruiker zich aanmeldt, hoeft er geen collectie aan privileges worden gekoppeld, maar enkel een rol. Hiermee verdwijnt ook het probleem dat een nieuwe privilege aan alle gebruikers gekoppeld moet worden als deze voor iedereen binnen de rol *gebruiker* geldt. Deze privilege hoeft enkel aan de rol te worden toegekend om hem voor alle gebruikers te laten gelden.

Als ik kijk naar op welke manier privileges van elkaar verschillen, is dit enkel in het type privilege. Een privilege kan bijvoorbeeld zijn dat een gebruiker een conflict mag aanmaken, een conflict mag bekijken of gebruikersgegevens mag bekijken. Het type privilege veranderd dus, maar de handeling komt steeds terug. Als ik kijk naar de handelingen die mogelijk zijn, kan ik hier in vier handelingen in onderscheiden: creëren, bekijken, bewerken en verwijderen.

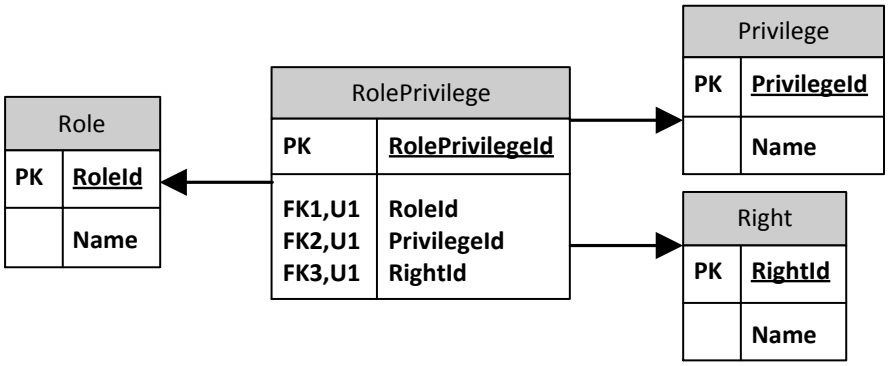
Deze privileges en handelingen kunnen op verschillende manieren worden opgeslagen binnen de database. Hierin heb ik vier mogelijkheden gevonden: de privilege en handeling samen opslaan als tekst, de handeling in een aparte tabel opnemen en koppelen, de verschillende handelingen als kolommen in een tabel opnemen of de handelingen als flags opslaan. Aan alle methoden zitten voor- en nadelen.

Als eerste heb ik gekeken naar de methode om de privilege en de handeling samen op te slaan als tekst. Een privilege kan dan zijn *CreateConflictResolution* of *ReadConflictResolution*.



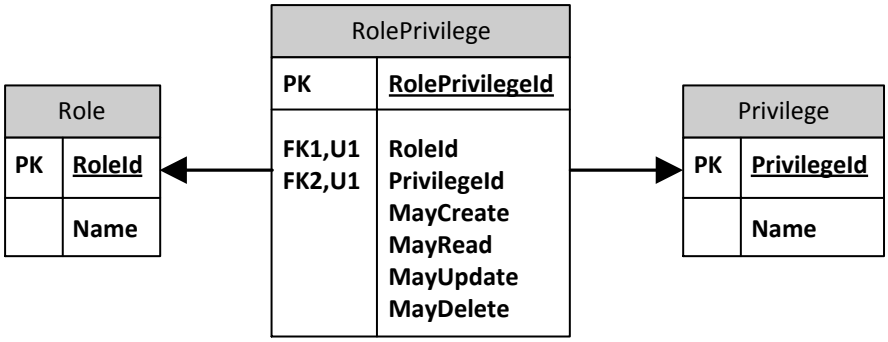
Het nadeel van deze methode is dat het heel gevoelig is voor typefouten. In zowel de naam van de privilege als de naam van de handeling kunnen makkelijk typefouten worden gemaakt. De privilege kan dan gelijk niet meer worden gevonden. Het gebruiken van een naam bij de privilege is onvermijdelijk, maar bij de handeling is dit niet zo. Daarom heb ik deze methode laten afvallen.

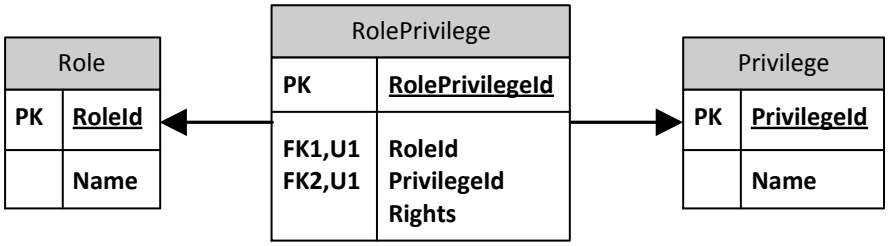
Vervolgens heb naar de andere methoden gekeken. Bij de eerste methode worden de handelingen in de derde normaalvorm opgeslagen in een aparte tabel, en aan de privilege gekoppeld. Dit ziet er als volgt uit.



Hier is de RolePrivilege-tabel een koppeltabel tussen de rol, de privilege en de handeling. Het nadeel hiervan is dat binnen de database op deze manier typefouten worden voorkomen, en enkel handelingen die binnen de tabel Right zijn opgenomen kunnen worden gekoppeld. Vanuit de applicatie moeten deze echter nog steeds met een string worden opgehaald. Hierdoor heb ik besloten ook deze methode niet te hanteren.

Ten slotte heb ik gekeken naar twee methoden om de handelingen strong typed binnen de applicatie beschikbaar te maken. Hiervoor heb ik twee mogelijkheden gezien: een tabel maken met de verschillende handelingen als kolommen binnen de tabel waar de handelingen als booleans worden opgeslagen of de handelingen opslaan als flags, binnen een numeriek veld. Deze twee mogelijkheden zien er als volgt uit.





De rechten zullen enkel vanaf de applicatie worden benaderd. Een oplossing waarbij de handelingen in een enum worden opgeslagen is daardoor prima mogelijk. Binnen de applicatie kan er worden gekeken of de waarde van de enum overeen komt met een flag die in de Rights-waarde zit.

Vanuit de database gezien is het nuttiger om meerdere kolommen te gebruiken, omdat hier indices op geplaatst kunnen worden. Wanneer er geselecteerd wordt op de waarde van MayCreate, kan een index hier een snelheidswinst bij opleveren.

Echter zullen de privileges niet op deze manier worden opgehaald. Bij het ophalen van een gebruiker zullen alle privileges worden opgehaald die voor hem van toepassing zijn, en niet andersom. Daarom maakt het vanuit de database gezien niet uit.

Ten slotte is er het probleem van onderhoud. Op het moment dat er een nieuwe handeling bij komt, zal er bij de methode waarbij de handelingen in kolommen wordt opgeslagen een nieuwe kolom in de database moeten worden gevoegd. Hiervoor moeten alle rijen in de tabel worden aangepast omdat de nieuwe kolom een waarde moet hebben, en eventuele indexen moeten worden toegevoegd.

Ook moeten er in de applicatie een aantal dingen gebeuren. Ten eerste moet de kolom worden toegevoegd in het domeinmodel, omdat deze vanuit de database wordt gevuld. Ook moet de mapping tussen het domeinmodel en de database worden aangepast, om de kolom vanuit de database in het domeinobject te krijgen.

Als er echter gekeken wordt naar de handelingen die nodig zijn om een extra waarde toe te voegen aan de enumeratie met handelingen, die in het laatste voorbeeld getoond zijn, is dat er eigenlijk maar één: daadwerkelijk de waarde aan de enumeratie toevoegen. Omdat de waarde numeriek opgeslagen wordt, hoeft er aan zowel de klasse als de databasetabel niets te worden gewijzigd. Qua onderhoud is deze methode dus erg flexibel.

Er is echter een belemmering. Wanneer de flags in een numerieke waarde worden opgeslagen, kunnen er maximaal 32 waarden worden opgeslagen. Elke flag komt overeen met een bit in de integer, en deze heeft er 32.

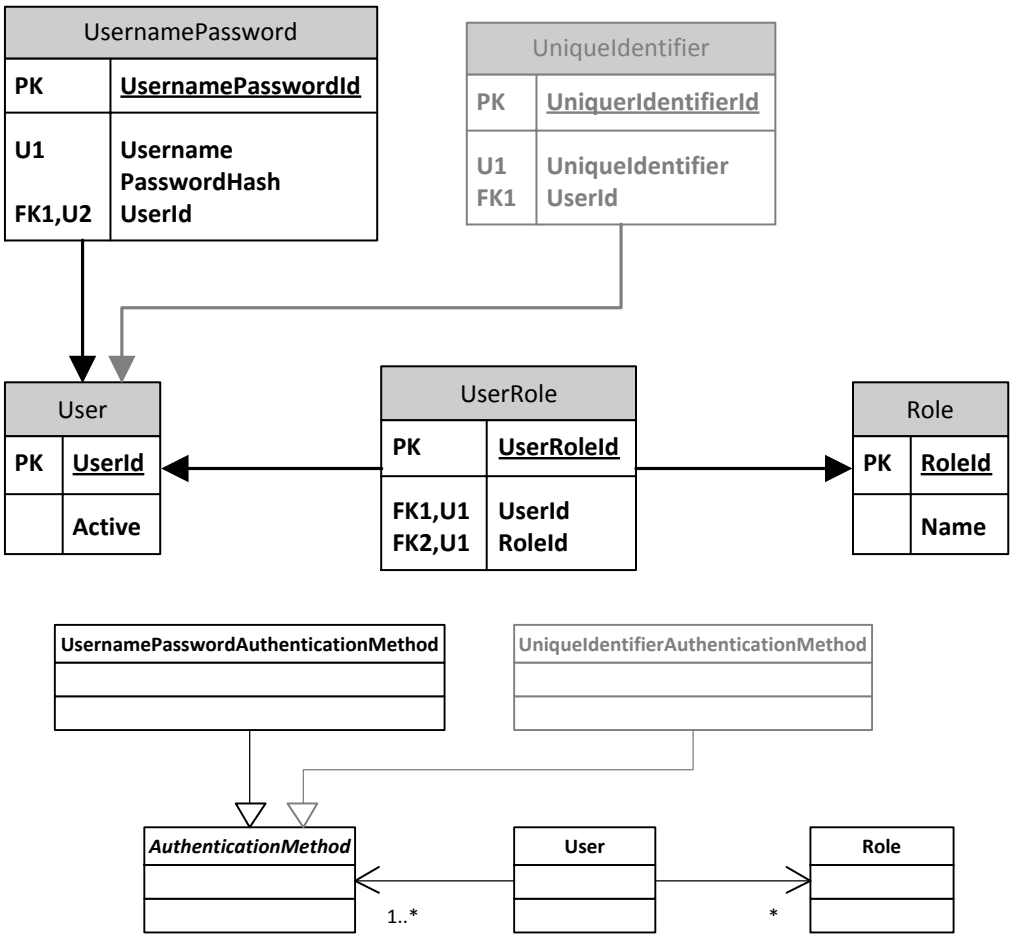
Het zal niet voorkomen dat er een wens is voor méér dan 32 verschillende handelingen. Daarom kan deze belemmering over het hoofd gezien worden in deze toepassing.

Als ik deze argumenten op een rij zet, is mijn keuze gevallen om de rechten in een numerieke waarde op te slaan. Het op kunnen vragen van een handeling door middel van een constante of variabele weegt hoog op tegenover het gebruiken van strings. Ook de flexibiliteit bij het onderhoud van de numerieke waarde weegt mee. Daarom zie ik dit als de beste mogelijkheid.

6.4.2 Gebruikers

Gebruikers moeten worden geïdentificeerd binnen het systeem. De keus momenteel is gevallen om dit door middel van een gebruikersnaam en een wachtwoord te doen. Echter zou in de toekomst er alsnog gebruik gemaakt kunnen worden van een Claim Based Identity-gerelateerde techniek, waarbij de gebruiker niet bekend is binnen het systeem, maar bij een extern systeem die de identiteit verzorgt.

Bij het ontwerpen van de gebruikers, wilde ik hier dus rekening mee houden, om te voorkomen dat bij het toevoegen van een inlogmethode er een grote verandering in de applicatie moet gebeuren. Ik ben tot het volgende ontwerp databaseontwerp en domeinontwerp gekomen om dit te bewerkstelligen.



Door op deze manier het systeem te ontwerpen, is het mogelijk om op een later moment een andere manier van authentiseren toe te voegen en is het ook mogelijk om meerdere manieren van authentiseren te koppelen. Zo zou een gebruiker meerdere `UniqueIdentifier`s kunnen instellen, zodat hij zowel kan inloggen met zijn Google-account als zijn Facebook-acccount

Dit kan handig zijn wanneer in de toekomst de gebruiker wil inloggen met een sociaal netwerk. Als de gebruiker vier sociale netwerken heeft kan het goed mogelijk zijn dat hij niet meer weet met welk netwerk hij heeft geregistreerd. Door ze allemaal te koppelen, maakt het niet uit met welke hij inlogt, want ze komen allemaal bij hetzelfde account uit.

In de bovenstaande diagrammen zijn de mogelijke andere methoden ook opgenomen in een grijze kleur, om te laten zien hoe een eventuele andere manier van inloggen in het systeem kan worden verwerkt.

Zoals te zien is, sla ik in de `UsernamePassword`-tabel geen wachtwoord op, maar slechts een hash. Dit doe ik om te voorkomen dat bij een eventuele hack op de database de inloggegevens van de gebruikers bij de hackers in handen komen. Hierbij maak ik gebruik van een salt. Ik kom later dit hoofdstuk terug op het gebruik van hashes en salt.

6.5 Registreren en inloggen

6.5.1 Registreren

Voordat een gebruiker kan inloggen bij de webapplicatie, moet hij eerst een account aanmaken. Dit doet hij door zich te registreren.

Voor het registreren is er momenteel één mogelijkheid voor de gebruiker om zich kenbaar te maken, door een gebruikersnaam en wachtwoord in te voeren. Zoals ik in het vorige onderdeel schreef, is er bij het ontwerpen rekening gehouden met mogelijke andere methoden voor de gebruikers om zich te identificeren. Op dit moment houd ik enkel rekening met identificatie middels een gebruikersnaam en wachtwoord.

Een gevaar bij het gebruik van wachtwoorden is dat een gebruiker vaak slechts enkele wachtwoorden gebruikt voor veel verschillende diensten. Wanneer een dienst gehacked zou worden, kunnen de gebruikersnamen en wachtwoorden van de gebruikers in handen vallen van de hackers. Wanneer de wachtwoorden plain text in de database opgeslagen zijn, geeft dit de hackers gelijk toegang tot een aantal andere systemen waar de gebruiker deze gebruikersnaam- en wachtwoordcombinatie ook gebruikt.

Om te voorkomen dat dit bij onze applicatie gebeurd, sla ik het wachtwoord niet als tekst op in de database, maar sla ik het wachtwoord op als een hash.

Een hash is een tekenreeks die wordt berekend vanuit de originele input. Een hashfunctie geeft altijd een hash terug met een specifieke lengte. Een hashfunctie is daarbij op een dergelijke manier opgezet dat input die ook maar een klein beetje verschilt, een compleet ander resultaat teruggeeft. Hierdoor wordt een hashfunctie vaak gebruikt om een wachtwoord onleesbaar op te slaan.

Wanneer een gebruiker een wachtwoord invult bij het inloggen, kan deze dus niet vergeleken worden met een wachtwoord wat is opgeslagen in de database. Wanneer van dit wachtwoord echter ook een hash wordt gemaakt, is deze hash identiek aan die van het wachtwoord, als het wachtwoord wat de gebruiker ingevuld heeft correct is. Zo kan er toch worden geverifieerd dat het wachtwoord klopt, zonder dat het wachtwoord opgeslagen hoeft te zijn.

Omdat de input in de hashfunctie geen vaste lengte heeft, is het mogelijk dat twee verschillende sets aan input hetzelfde resultaat geven. Dit wordt een collision genoemd. Voor gebruik bij hacken worden deze collisions opgeslagen in rainbow tables. Dit zijn tabellen met hashes en de collisions hierbij.

In tegenstelling tot het encrypten van een string, is het bij een hashfunctie niet mogelijk om het resultaat weer te decrypten, ofwel terug te gaan naar de originele input. Wat wel mogelijk is, is alle mogelijke combinaties van hashes te genereren, om zo een collision te vinden. Omdat de gehashte versie van de collision hetzelfde resultaat geeft als het originele wachtwoord, kan de collision worden gebruikt om in te loggen. Bij het inloggen wordt er immers niet gekeken of het wachtwoord overeen komt, maar of het resultaat van de hashfunctie overeen komt met de hash die opgeslagen is. Met de collision zou een hacker dus kunnen inloggen zonder dat hij het originele wachtwoord heeft.

Hiervoor is een trucje bedacht, wat *salt* wordt genoemd. Salt is een geheim stukje tekst wat in het wachtwoord wordt verweven. Stel dat we het wachtwoord *123456* nemen. Een voorbeeld van een wachtwoord waar salt aan is toegevoegd, is dan *DIT+IS+123456+MET+ZOUT*. Het wachtwoord wordt inclusief salt opgeslagen in de database.

Als een hacker vervolgens probeert een collision te vinden voor het gehashte wachtwoord in de database, zal hij een collision vinden voor *DIT+IS+123456+MET+ZOUT*, en niet voor *123456*. Omdat het resultaat van twee op elkaar lijkende strings na een hashfunctie volledig anders is, kan de collision voor 123456 niet worden afgeleid uit de hash.

Wanneer de hacker vervolgens met de gevonden collision probeert in te loggen, werkt dit niet. Wat er namelijk gebeurd, is dat bij het testen van de hashes wederom salt wordt toegevoegd. Stel dat de gebruiker de perfecte collision voor de hash van het wachtwoord heeft gevonden, namelijk het originele wachtwoord met salt *DIT+IS+123456+MET+ZOUT*. Wanneer hij hiermee probeert in te loggen, is het resultaat van de hashfunctie *DIT+IS+DIT+IS+123456+MET+ZOUT +MET+ZOUT*, namelijk het wachtwoord met salt, waar nogmaals salt overheen gedaan is. Ook al heeft hij een collision, kan hij nog niet inloggen omdat hij niet enkel het wachtwoord heeft.

Het enige wat de hacker in dit geval kan doen, is een nieuwe rainbow table genereren waar de wachtwoorden met salt in staan. Hiervoor moet hij echter wel weten wat de salt is, omdat hij deze moet gebruiken bij het genereren van de hash. Omdat de hash geheim blijft binnen de applicatie is dit in principe onmogelijk. Daarbij kost het genereren van een rainbow table veel tijd en geheugen, omdat een hash van 32 tekens lang al $3{,}4^{38}$ mogelijkheden heeft. Voor een hacker is het vinden van collisions dus niet interessant.

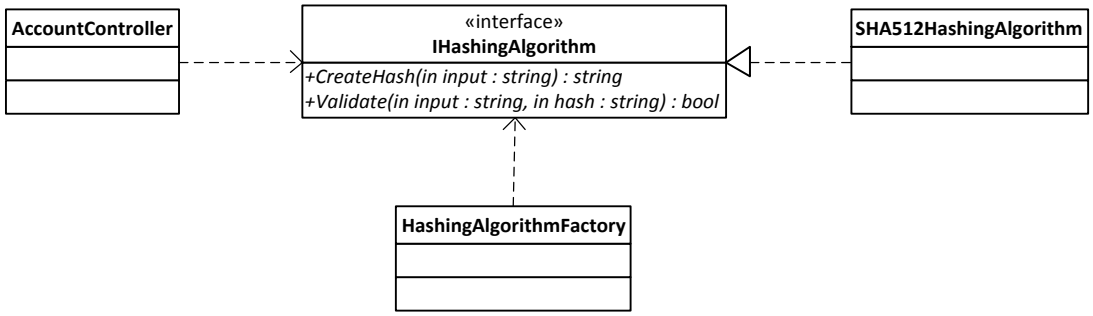
Om voor een salt te zorgen die niet makkelijk geraden kan worden, heb ik zestien willekeurig gekozen karakters voor, en zestien willekeurig gekozen karakters achter het wachtwoord gezet. Ook heb ik gekozen voor een hashingalgoritme wat een lange hash oplevert. Ik heb gebruik gemaakt van het SHA-512 algoritme, welke een hash van 512 bits of 64 bytes maakt.

Een hash wordt opgeslagen als hex. Een hexkarakter komt overeen met een getal tussen de 0 en de 15. Een byte heeft 256 verschillende waarden. Eén byte neemt hierdoor twee hexkarakters in (16 * 16 = 256). Een hash van 64 bytes krijgt hierdoor een lengte van 128 karakters. Een dergelijke hash heeft $1{,}34^{154}$ mogelijkheden, wat het voor hackers niet te doen maakt om hier een collision voor te vinden.

Omdat het mogelijk is dat het hashingalgoritme in de toekomst vervangen kan worden voor een betere, heb ik er rekening mee gehouden dat deze makkelijk kan worden vervangen. Dit heb ik gedaan door hier een interface boven te plaatsen, en gebruik te maken van het factory pattern.

In de configuratie van de applicatie kan worden aangegeven welke instantie van het hashingalgoritme moet worden gebruikt. Wanneer er binnen de applicatie gebruik gemaakt gaat worden van het hashingalgoritme, wordt er een instantie van dit algoritme gevraagd aan de factory. Deze kijkt vervolgens in de configuratie van welk type hashingalgoritme er een instantie gemaakt moet worden. Op deze manier is het specifieke algoritme makkelijk te wisselen wanneer deze noodzaak er is, zonder dat de onderliggende code gewijzigd hoeft te worden.

Omdat ik gebruik maak van het hashalgoritme in een controller binnen ASP.NET MVC, heb ik dit algoritme opgenomen in de Inversion of Control-container, waardoor deze automatisch bij het aanmaken van de controller hierin geïnjecteerd wordt. Het ontwerp van dit onderdeel ziet er als volgt uit.



6.5.2 Inloggen

Wanneer er op een site ingelogd kan worden, hoeft dit maar één keer te gebeuren. De gebruiker hoeft niet in te loggen bij elke pagina die hij opvraagt, maar hij logt één keer in, en welke gebruiker er ingelogd is wordt onthouden. Dit gebeurd met behulp van een cookie.

Een cookie is een hoeveelheid data die de server naar de browser stuurt, met het doel dat de browser deze opslaat, en bij de volgende request weer naar de server terugstuurt. Op deze manier is het voor de server niet alsof de gebruiker de site voor het eerst bezoekt, maar kan hij deze herkennen door het cookie.

Om dit te kunnen doen, heb ik een authenticatieklasse geschreven die deze cookies kan in- en uitlezen. Op deze klasse is het mogelijk om een gebruiker in te loggen, een gebruiker uit te loggen en om het gebruikersid van de ingelogde gebruiker op te vragen.

Als een gebruiker is geregistreerd, kan hij inloggen bij het systeem. Dit doet hij door de gebruikersnaam- en wachtwoordcombinatie in te vullen in het inlogscherm, die hij bij het registreren heeft ingegeven. Als hij dit gedaan heeft, worden deze in de controller gevalideerd. Dit wordt op de volgende manier gedaan.

Eerst wordt er door middel van het hashalgoritme wat in het vorige hoofdstuk besproken is een hash gemaakt van het door de gebruiker ingevulde wachtwoord. Vervolgens wordt er in de database gekeken of er hierin een gebruiker voorkomt met deze gebruikersnaam en wachtwoordhash. Als de gebruiker in de database voorkomt, wordt deze opgehaald. De gebruikersid van deze gebruiker wordt vervolgens in een cookie opgeslagen door de authenticatieklasse.

Wanneer het id van de gebruiker echter plain text wordt opgeslagen in een cookie, is het voor een gebruiker makkelijk om deze te wijzigen. De cookies in de browser kunnen immers gewoon bekeken en gewijzigd worden. Als een gebruikersid bijvoorbeeld 15 is, en een gebruiker maakt hier 10 van, denkt de server dat gebruiker 10 is ingelogd. Hierdoor kan de gebruiker inloggen onder het account van een andere gebruiker.

Om dit te voorkomen, heb ik het cookie geëncrypt. Waar bij het hashen van een string de originele inhoud niet kan worden teruggehaald, kan dit bij encrypten wel. De waarde wordt alleen op zo'n manier opgeslagen dat deze niet kan worden gelezen of gewijzigd zonder dat de gebruiker

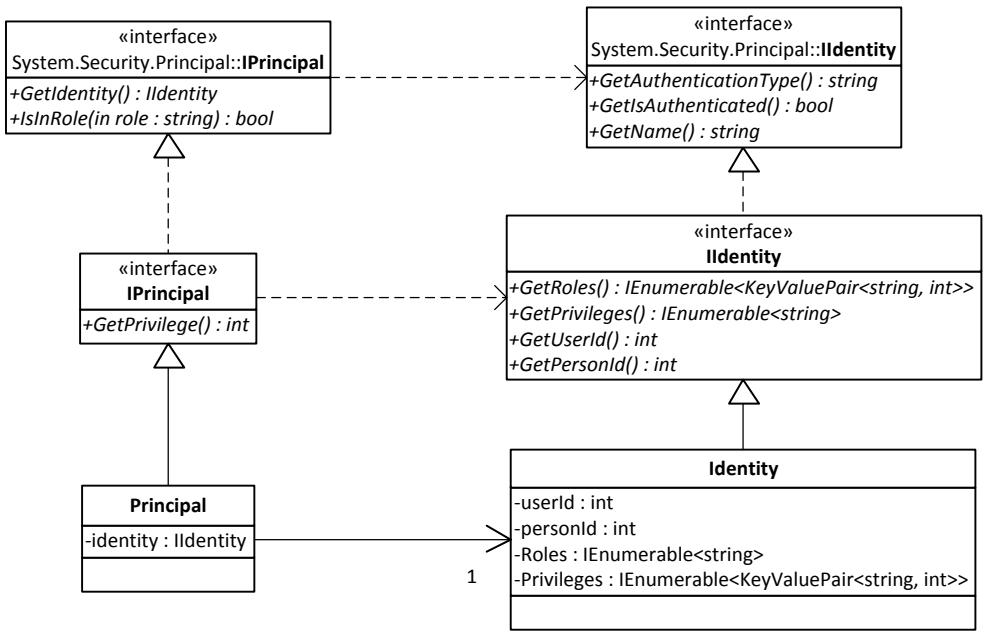
de sleutel weet waarmee deze versleuteld is. Op deze manier kan de gebruiker niet stiekem inloggen als een andere gebruiker.

Voor het versleutelen van gegevens zijn verschillende algoritmes beschikbaar. Een veelgebruikt algoritme is het AES-algoritme (advanced encryption standard). Binnen het .NET framework is een implementatie van dit algoritme beschikbaar, waardoor ik deze kon gebruiken en dit niet zelf hoefde te implementeren, of op zoek hoefde naar een goede implementatie.

Omdat ik zelf geen ervaring met het versleutelen van gegevens heb, kon ik zelf niet veel zeggen over de werking en kwaliteit hiervan. Ik kon enkel overnemen wat ik op internet las, en dat was dat AES een erg sterk algoritme is, en tot dusverre nog niet gekraakt.

Om toch de flexibiliteit te houden, heb ik ervoor gekozen een zelfde soort aanpak te gebruiken die ik ook bij het hashen gebruikt heb: Ik heb boven de implementatie van het algoritme een interface gezet. Ook heb ik een factory pattern gebruikt die uit de configuratie van de applicatie kan halen welke versie van het encryptie-algoritme er geïnstantieerd moet worden. Op deze manier kan het algoritme makkelijk worden vervangen, als dit nodig blijkt te zijn.

Als de gebruiker is ingelogd, moet er bekend zijn bij het opvragen van een pagina welke gebruiker de pagina opvraagt. Een gebruikersid is hierbij leuk, maar heel veel kan de server hier niet mee. Binnen de applicatie is het interessant om te weten tot welke rollen de gebruiker behoord, en welke privileges hij heeft. Wanneer een gebruiker een pagina opvraagt, zou deze informatie dus beschikbaar moeten zijn.



Het .NET framework biedt standaard een tweetal interfaces aan voor deze taak: de `IPrincipal` en de `IIdentity`. De `IIdentity` is via een getter beschikbaar op de `IPrincipal`. De `IPrincipal` heeft het doel om een context te bieden om autorisatie uit te voeren. De `IIdentity` representeert de huidige ingelogde gebruiker. De `IPrincipal` wordt per webrequest opgebouwd. Omdat vooral de `IPrincipal` binnen het .NET framework en ASP.NET MVC veel gebruikt worden om autorisatie uit te voeren die met de frameworks meegeleverd is, heb ik besloten om deze interfaces te

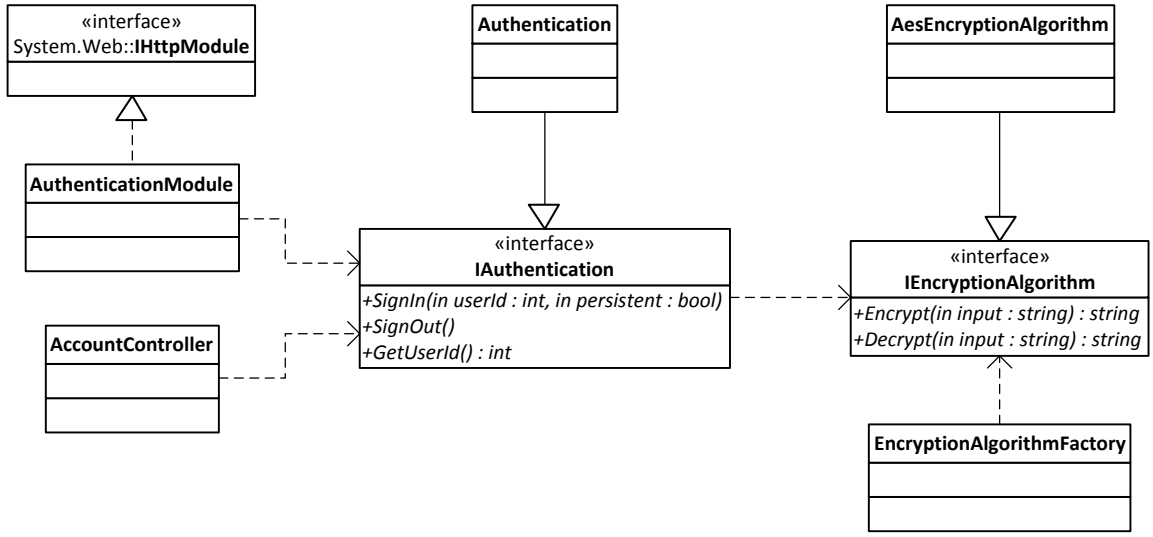
gebruiken. Hiervoor heb ik eigen interfaces en klassen ontwikkeld die de bestaande interfaces implementeren, en uitbreiden met de functionaliteit die ik binnen de applicatie nodig heb.

Binnen ASP.NET MVC zijn er verschillende events beschikbaar waar op geabonneerd kan worden. Eén van deze events is het `AuthenticateRequest` event. Dit treedt op wanneer de gebruiker geauthentiseerd moet worden binnen de applicatie. Dit is voor mij het perfecte moment om de gebruiker op te halen aan de hand van de id die in de cookie opgeslagen is. Vervolgens kan de opgehaalde gebruiker in de identity worden geplaatst, waarna deze binnen de applicatie beschikbaar is. Op deze manier is de gebruiker altijd beschikbaar binnen de applicatie, zonder dat het specifieke stuk code wat het gebruikersobject nodig heeft de gebruiker op hoeft te halen.

Het abonneren op zo'n event gaat door middel van een `HttpModule`. Dit is een module die binnen de applicatie geregistreerd kan worden, en vervolgens voor elke request kan worden aangeroepen. Zo hoeft er bij een specifieke pagina geen rekening gehouden te worden met inloggen.

Een ander voordeel van een `HttpModule` ten opzichte van normale code, is dat deze op een specifiek moment kan worden aangeroepen, afhankelijk van het gebruikte event. Het bovengenoemde `AuthenticateRequest` wordt aangeroepen vóórdat de pagina wordt opgebouwd. Zo worden er niet eens gegevens opgehaald als de gebruiker hier geen rechten toe heeft.

Hoe dit allemaal samen komt, is in onderstaand diagram te zien.



6.6 Vertraging

Doordat het implementeren van Claims Based Identity niet verlopen is zoals geplanned, heb ik een week vertraging opgelopen. Het was op twee manieren mogelijk om met deze vertraging om te gaan.

Ten eerste had ik zoals op de planning stond een nieuwe sprint kunnen starten, en de resterende openstaande punten voor het inloggen als hoogste prioriteit te behandelen. Ook was het mogelijk om de sprint door te laten lopen, zodat de sprint in plaats van twee weken, drie weken zou duren.

Ik heb voor dit laatste gekozen, omdat de volgende stappen op de planning ook groot waren. Zo groot, dat ze naar verwachting niet binnen een week afgerond zouden zijn. Als ik een nieuwe sprint had gestart, zou ik een week hebben voor de andere openstaande punten. Dit ging ik naar verwachting niet redden.

Als ik tóch een nieuwe sprint gestart had, was ik in dezelfde situatie uitgekomen als dat ik nu zat: aan het einde van de sprint géén release kunnen doen. Omdat ik nog erg in het begin van het project zat, waren er niet veel kleine stukjes waaraan ik kon werken, maar was het nog voornamelijk de fundering die ik aan het leggen was.

Dit alles is voor mij de doorslaggevende reden geweest om de sprint te verlengen. Wanneer ik aan het einde weer in dezelfde situatie terecht gekomen was, was ik nog verder van huis geweest. Op deze manier heeft de vertraging zich binnen één sprint geisoleerd.

7 Sprint twee: Betaalmethoden en veel itereren

7.1 Online betalen

Voordat er tot het implementeren van een betaalmethode kan worden overgegaan, moeten er eerst contracten worden afgesloten. Bij het implementeren van betalen is er immers een afhankelijkheid aan een externe dienst, en hiervoor moet een contract worden afgesloten. Omdat het hier over bankzaken gaat, worden er eerst dingen gecontroleerd, etc. waardoor het aanvragen van de dienst enkele weken kan duren. Een vriend van mij die eerder iDeal geïmplementeerd had binnen zijn eigen webwinkel kon mij bijvoorbeeld vertellen dat het afsluiten van het contract bij hem een ruime maand heeft gekost.

Daarom ben ik op dit moment gaan kijken naar de betaalmethoden en de contracten die hiervoor nodig zijn, zodat ik niet met een wachttijd te maken krijg op het moment dat ik daadwerkelijk op implementeren over ga.

In de voorbereiding voor het verzamelen van de requirements heb ik al kort gekeken naar de methoden waarmee online betaald kan worden. Directe response is hierbij het belangrijkst. Wanneer een gebruiker een betaling verricht, moet de applicatie hier direct een melding van krijgen.

Omdat de gebruiker betaald voor een digitale dienst, is het zeer wenselijk dat hij na de betaling gelijk gebruik kan maken van de dienst. Als een gebruiker een artikel uit een webwinkel besteld, maakt het minder uit dat de betaling bijvoorbeeld pas na een dag wordt verwerkt, omdat de gebruiker toch op zijn bestelling moet wachten.

Wanneer er echter voor een digitale dienst wordt betaald, is het veel wenselijker dat er gelijk gebruik gemaakt kan worden van de dienst. De gebruiker kan dan in één keer door met het gebruiken van de dienst, en hoeft niet te wachten op de betaling.

Eerder heb ik bij de voorbereiding hier twee methoden voor gevonden, namelijk iDeal en Paypal. Voor zowel Paypal als iDeal moeten abonnementen worden afgesloten. Paypal heeft hierbij geen maandelijkse kosten, maar iDeal wel. Paypal wordt bij Paypal zelf afgesloten, en iDeal gebruikelijk bij de bank waarvan de rekeninghouder (dus niet de klant) lid is. Ook zijn er externe diensten die PSP's (Payment Service Provider) worden genoemd. Deze vormen een tussenlaag tussen de applicatie en de bank. PSP's bieden vaak meerdere betaalmethoden, niet enkel iDeal.

Om te kijken welke dienst, iDeal bij de bank afsluiten, of een PSP, het voordeligst is, heb ik de tarieven hiervan tegenover elkaar gezet.

Bij het zoeken van verschillende PSP's werd ik positief verrast. De Rabobank, waar Cenosco haar zakelijke rekening heeft, had halverwege februari zijn eigen PSP gelanceerd, die ze OmniKassa noemen. Dit is nét na mijn eerste oriëntatie op de betaalmethoden geweest, waardoor ik deze gemist heb.

Binnen de OmniKassa is het mogelijk om iDeal-betalingen en creditcardbetalingen te ontvangen. Ook staat op de planning dat de Belgische variant van iDeal en Paypal in 2012 geïntegreerd gaan worden binnen de dienst.

Omdat de OmniKassa als PSP dient, vormen ze een tussenoplossing tussen de applicatie en de banken. Dit betekent dat er geen aparte implementatie voor creditcards en iDeal gemaakt hoeft te worden. Er wordt enkel een betaalverzoek naar de dienst verstuurd, en deze neemt vervolgens de betaalmethoden voor haar rekening. Wanneer de kassa een nieuwe betaalmethode implementeert hoeft deze enkel te worden aangezet in het portal van de kassa, maar hoeft deze niet te worden geïmplementeerd in de applicatie.

Ook het prijzenaspect van de kassa is interessant. Tot ongeveer honderd transacties per maand is de kassa zelfs goedkoper dan de iDeal-variant van de Rabobank. Zeker in het begin zullen er niet heel veel transacties worden verwacht, dus dit is ook voordelig.

Mijn bevindingen heb aan mijn opdrachtgever getoond, en deze was er ook voor om de OmniKassa aan te vragen en te implementeren binnen de applicatie.

Het aanvragen van de dienst kost echter tijd, zoals hierboven al vermeld. Dit was voorzien, dus binnen mijn planning heeft dit geen consequenties gehad. Echter betekende dit wel dat betalen nog niet gelijk ingebouwd kon worden, maar dat er eerst op goedkeuring gewacht moest worden. Nadat alles goedgekeurd is, kan er pas tot implementatie worden overgegaan. Het implementeren is daarom naar achteren geschoven in de planning.

7.2 Start van de boedelverdeling

Doordat het inloggen en registreren gebouwd was, en er op de bank gewacht moet worden voordat betalen geïmplementeerd kan worden, zijn er geen belangrijke punten voor het platform meer over waar er op dit moment aan gewerkt kan worden. Er kon nu een start worden gemaakt aan het daadwerkelijke conflict waar ik me tijdens mijn afstudeerperiode mee bezig zou gaan houden, namelijk het verdelen van spullen tijdens een scheiding.

Over dit conflict heb ik een gesprek gehad met mijn begeleider, waarbij we het voornamelijk over de implementatie gehad hebben. De eerste stap bij de implementatie van dit conflict is het in kaart brengen van de spullen van de betrokkenen bij het conflict. Wanneer de spullen immers niet in het systeem aanwezig zijn, kunnen ze ook niet verdeeld worden.

Zowel mijn begeleider als ik hadden nog geen concreet beeld van de manier waarop dit zou moeten gebeuren. Mijn begeleider vond wel dat het proces van het toevoegen van producten zo gemakkelijk mogelijk moest gebeuren voor de gebruiker. Deze moest als het ware door het proces van het toevoegen van producten heen geleid worden.

Om achter de juiste manier van het toevoegen van de producten te komen, hebben we besloten dat ik in iteraties prototypes zou gaan maken. Over het prototype kon er vervolgens gediscussieerd worden, en uit deze feedback kon er weer een nieuwe prototype worden gemaakt. Op deze manier kwamen we steeds dichter bij het doel, zonder dat alle bagage van het volledige product steeds meegenomen werd.

Wanneer er volgens Scrum wordt gewerkt, wordt er gewerkt in sprints die meestal twee weken duren. Aan het einde van de twee weken wordt er een werkend stuk software opgeleverd, waar de klant vervolgens feedback over kan geven. Deze feedback kan vervolgens worden meegenomen in volgende sprints. Deze opzet was echter niet goed toepasbaar voor mijn situatie.

Het enige punt wat ik op dat moment op mijn planning had staan was om samen met mijn opdrachtgever achter de wenselijke manier van producten toevoegen te komen. Aan het platform kon op dat moment niets gedaan worden, en de andere punten op de backlog waren afhankelijk van de implementatie van de productenlijst.

Wanneer deze opzet via Scrum wordt aangepakt, zou het maken van een prototype een van de punten op de backlog van de sprint zijn. De rest van de sprint wordt dan opgevuld met andere openstaande punten. Zoals ik al schreef, was dit in mijn situatie niet van toepassing.

Ook was het voor mij erg makkelijk om contact op te nemen met de “klant”. De klant was immers gewoon mijn stagebegeleider wiens kamer enkele meters van de mijne verwijderd was. Het contact tussen mij en mijn begeleider was dus erg kort.

Daarom hebben we ervoor gekozen om ietwat van Scrum af te wijken. In plaats van iteraties aan te houden van twee weken, hebben we hele korte iteraties gedaan, van een flexibele lengte. In de praktijk zijn dit iteraties van twee tot drie dagen geweest.

Deze iteraties waren als volgt opgebouwd. Op basis van het gesprek kon ik een prototype opbouwen, waarin de functionaliteit die ik voor ogen had geïmplementeerd werd. Vervolgens heb ik een kort gesprek gehad met mijn begeleider waarbij we samen door het prototype heen liepen, hier over discussieerde, en ik hier feedback over verzamelde. Op basis van deze feedback kon ik een nieuw prototype opstellen.

Deze opzet van korte iteraties maakte het mogelijk dat in een relatief korte periode er veel releases werden gedaan. Hierdoor kon de gewenste opzet voor het verzamelen van de producten snel kon worden achterhaald. Mijn begeleider stond hier ook voor open, en dit leek hem ook een goede methode.

Hoe het proces van de prototypes verlopen is, beschrijf ik hieronder.

7.2.1 Het eerste prototype: producten toevoegen middels een invoerscherm

In het gesprek met mijn begeleider was naar voren gekomen dat hij graag wilde dat het verzamelen van producten per kamer zou verlopen. Op deze manier is het voor gebruikers gemakkelijk om een mentaal beeld te maken van de spullen die op de lijst moeten komen.

Wanneer een gebruiker een aanzienlijke lijst met spullen moet gaan opstellen, kan het snel voorkomen dat hij enkele producten vergeet, gewoon omdat hij door de bomen het bos niet meer kan zien. Door het proces van spullen invoeren te verdelen over de kamers willen we dat voorkomen. Door in kamers te denken wordt er structuur aangebracht, en is het invoeren van de producten in één kamer geïsoleerd van het invoeren van producten in een andere kamer.

Ook hebben we bij het gesprek gesproken over het invoeren van de producten. Dit willen we niet heel erg gedetailleerd doen, maar wat meer globaler. Het overwegende hierbij is de prijs van de in te vullen producten. Wanneer er een product wordt toegevoegd aan de lijst, heeft dit geen nieuwwaarde meer. Het product is immers per definitie tweedehands. Heel precies nieuwwaarde van het product invullen heeft dan ook weinig zin, omdat de producten dit gewoonweg niet meer waard zijn.

Om toch een concrete prijs bij een product in te kunnen vullen, zou dit product eerst moeten worden getaxeerd, etc. Deze detaillering is niet waar mijn opdrachtgever naar toe wilde. Daarom hebben we een andere opzet gekozen, namelijk het opgeven van prijzen in prijsranges. Omdat er hierbij geen concrete prijzen komen kijken, hoeft deze niet precies te zijn, zoals dat bij een taxatie gebeurd. Zo kan bijvoorbeeld een televisie die twee jaar geleden gekocht is voor €1000 in de prijsrange €500-€700 vallen.

Omdat bepaalde artikelen een extra waarde kunnen krijgen als ze antiek zijn, zoals een antieke servieskast, was het ook gewenst om een soort van tags aan de producten te hangen die deze eigenschappen representeerde.

Ten slotte wilden we voorkomen dat de gebruiker alle producten met de hand in moest vullen. Er hoeft geen concreet product te worden uitgetypt, maar de gebruiker kiest het product uit een lijst met producten. Het toevoegen van een product gaat zo sneller, en typefouten worden voorkomen.

Ook kan een lijst met producten gebruikers helpen bij het toevoegen van deze producten aan de inventarislijst. Wanneer er een groot aantal producten moet worden geïnventariseerd, kan het snel gebeuren dat er producten worden vergeten. Het product in de lijst terugzien kan als een herinnering dienen dat ook dit product aan de inventaris moet worden toegevoegd.

Al deze informatie heb ik gebruikt om een eerste prototype op te stellen.

Ik ben begonnen met kijken naar het datamodel wat voor de applicatie nodig is, om het mogelijk te maken producten toe te voegen. Omdat we willen indelen per kamer, is er ten eerste een `Room` nodig. Ook is de `ProductList` nodig om de producten aan toe te voegen.

Het moet mogelijk zijn om producten uit een lijst te selecteren. Hiervoor gebruik ik de tabel `ProductType`. Om deze beter in te kunnen delen heb ik de tabel `ProductCategory` geïntroduceerd. Zo is een product type een televisie, die binnen de categorie elektronica valt.

Om de prijsranges van de producten op te slaan heb ik de tabel `PriceRange` gemaakt. In deze tabel staat de minimumprijs en de maximumprijs waartussen de prijs ligt. Bij een prijsrange van ‘onder de 300 euro’ is de minimumprijs `NULL`. Dit vond ik een nettere oplossing door de minimumprijs de concrete waarde 0 te geven.

Bij een prijs die onder een bepaald bedrag ligt is een minimumbedrag van 0 nog wel te doen, omdat een prijs van nul euro de laagste prijs kan zijn die aan een product kan worden gegeven, omdat het product dan niets kost. Bij een prijs die ‘boven de 300 euro’ ligt wordt het heel smerig. De maximumprijs zou dan de maximum waarde van het veld moeten worden wat ergens in de miljoenen ligt. Het verschil tussen een daadwerkelijke prijsrange en een prijs boven een bepaald bedrag is dan moeilijk meer af te leiden.

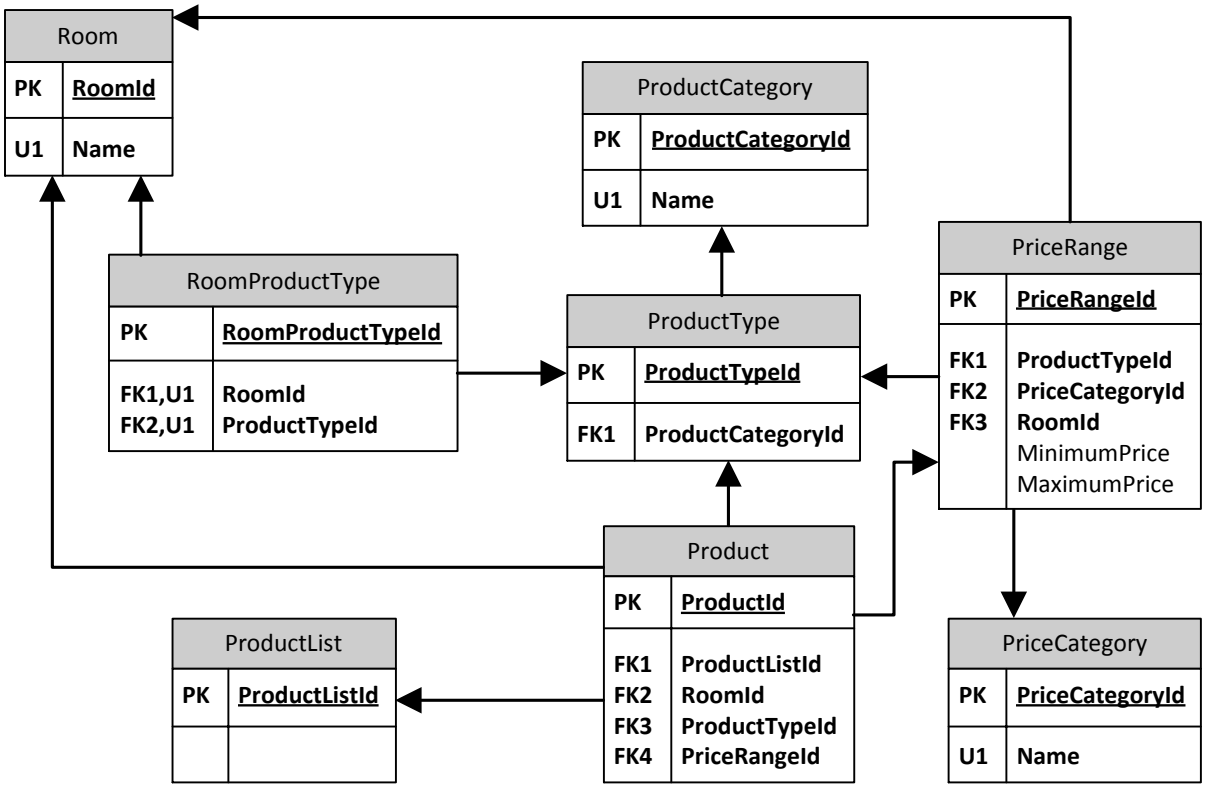
Om dit op te lossen, heb ik zowel het minimumbedrag als het maximumbedrag in de prijsrange nullable gemaakt. Wanneer er een prijs onder de 300 euro is, is het minimumbedrag `NULL` en het maximumbedrag 300. Wanneer het om een prijs boven de 300 euro gaat, is het minimumbedrag 300 en het maximumbedrag `NULL`. Zo is binnen de applicatie goed duidelijk of het om een prijsrange tussen twee bedragen gaat, of een ‘groter dan’-prijsrange.

Aan een specifieke prijsrange kan een tag worden gehangen zoals ‘antiek’. Deze prijsrange zal dan vermoedelijk hoger uitvallen dan de andere prijsranges. Het is een manier om context te bieden binnen de verschillende prijsranges.

Prijzen kunnen afhangen van de kamer waarin de producten aanwezig zijn. Wanneer we het over een televisie hebben die in de woonkamer staat, zal hier over het algemeen een hoger prijskaartje aan hangen dan de televisie in de slaapkamer. De televisie in de woonkamer is meestal een grote flatscreen, en de televisie in de slaapkamer een kleine flatscreen of een televisie met beeldbuis. Wanneer er een prijs aan de televisie in de slaapkamer wordt gehangen, is het niet erg nuttig om prijzen als ‘tussen de 1200 en 1600’ weer te geven, omdat niemand een televisie van die waarde in de slaapkamer heeft. Daarom heb ik een koppeling gelegd tussen de prijsrange en de kamer waarbinnen deze zich bevindt. Zo kunnen de prijzen worden gefilterd op de kamer waaraan het product wordt toegevoegd.

Ten slotte moet het product zelf nog worden toegevoegd. Dit product wordt toegevoegd aan een kamer, heeft een concreet product type, en heeft een prijsrange. Een voorbeeld kan zijn een televisie, die wordt toegevoegd aan de slaapkamer en een waarde heeft tussen de 200 en 400 euro.

De opbouw van de tabellen is als volgt.



Vervolgens kon ik gaan kijken naar de manier waarop producten toegevoegd konden worden, binnen de webinterface.

Om een product toe te voegen, moet er dus een kamer bekend zijn, het daadwerkelijke product, en de prijsrange. Om dit in het systeem te zetten, heb ik een invoerscherm gemaakt waarmee dit mogelijk was. Dit invoerscherm zag er als volgt uit.

Op het moment dat er een kamer wordt geselecteerd, worden alle producttypes opgehaald die aan de gekozen kamer gekoppeld zijn. Hetzelfde gebeurd met de prijs als de gebruiker een producttype selecteert.

Op deze manier kunnen de keuzes voor de gebruiker tot een minimum worden beperkt. Als een gebruiker de woonkamer selecteert, ziet hij enkel relevante producten voor de woonkamer. Het is immers niet van belang om bijvoorbeeld een koelkast of een fiets als keuze beschikbaar te maken in de woonkamer. Ruis wordt hierdoor verminderd. Wanneer er een producttype is gekozen, worden ook enkel de prijsranges weergegeven die van toepassing zijn op de gekozen kamer en het gekozen product.

Naar aanleiding van dit prototype heb ik het eerste feedbackgesprek gehad met mijn begeleider. In dit gesprek was hij erg direct, de manier van producten toevoegen was té omslachtig. Het was de wens om in een paar minuten een heel huis in kaart te kunnen brengen, en wanneer er per product een scherm doorlopen moest worden, kostte dit teveel tijd.

Een voorstel wat hij deed, was om te kijken naar een lijst, waar de gebruiker vinkjes voor kon zetten. Op deze manier kan hij door de lijst lopen, en vinkjes zetten voor de producten die hij in zijn huis heeft, en op deze manier snel een complete lijst opstellen.

Een ander punt wat hij aangaf, is dat het wellicht nuttig was om een aantal op te kunnen geven bij een product, omdat het goed mogelijk is dat er meerdere dezelfde spullen in huis zijn. Bij de eetkamertafel staan bijvoorbeeld vaak meerdere stoelen.

Over de tags gaf hij de opmerking dat deze niet direct aan de prijs gekoppeld hoefden te zijn. waar nu een prijs altijd aan een tag hangt, zoals antiek, was dit niet wenselijk. Het idee was om de tags als een soort van modifiers te zien. Wanneer de tag aan een product hangt, krijgt dit product binnen het algoritme een andere waarde, maar hoeft dit niet direct aan de prijs gekoppeld te zijn. De `PriceRange` en `PriceCategory` moesten dus van elkaar gescheiden worden.

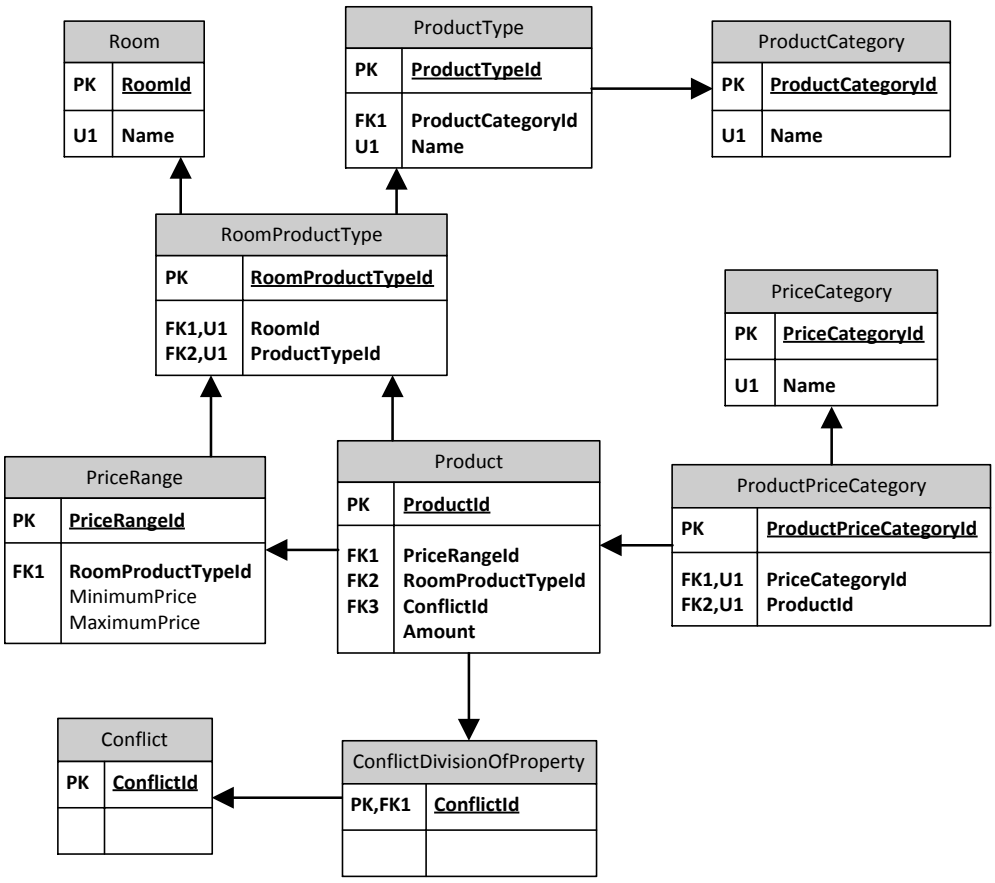
Hij gaf ook aan dat er een paar onderdelen in het datamodel verbeterd konden worden. Zoals in het bovenstaande diagram te zien is had ik zowel `PriceRange` als `Product` aan `Room` en `ProductType` gekoppeld had. Deze tabellen konden aan de koppeltabel `RoomProductType` gekoppeld worden om hetzelfde resultaat te bereiken, zonder dat er twee foreign keys aanwezig hoefden te zijn.

Ten slotte hebben we over een ander punt gesproken, namelijk templates. Wanneer er voor het eerst een huis opgebouwd wordt, kunnen we voorspellingen doen over de spullen waarvan we verwachten dat ze in het huis aanwezig zijn. Ieder huis heeft bijvoorbeeld een koelkast in de keuken staan, een bed in de slaapkamer en een bank en televisie in de woonkamer. Door gebruik te maken van een template zouden we gebruikers werk uit handen kunnen nemen, door deze producten standaard aan het huis toe te voegen.

Met deze feedback en de nieuwe ideeën die uit het gesprek voorgekomen waren, kon ik een tweede prototype opzetten.

7.2.2 Het tweede prototype: producten aanvinken in een lijst

Om producten in een lijst te tonen, hoefde er niet veel aan het datamodel veranderd te worden. Enkel de PriceCategory moest van de PriceRange worden losgekoppeld, er moest een aantal worden toegevoegd aan `Product` en de koppelingen die aan de verkeerde tabel gekoppeld waren moesten worden gewijzigd. Dit zag er als volgt uit in het databasemodel:



Voor het toevoegen van templates moest er meer gebeuren. Omdat een template in principe een lijst is met producten die aan het conflict toegevoegd moeten worden op het moment dat dit wordt aangemaakt, moet er binnen de template een collectie van producten bekend zijn.

Voor templates waren er meerdere mogelijkheden. Zo kan een template in de database worden opgenomen, maar kan de template ook in de vorm van een xml-bestand worden geüpload. Wanneer er dan gebruik gemaakt wordt van de template, wordt het bestand ingelezen, en worden de producttypes hieruit gehaald.

Binnen de producttypes in de database en de producten in de templates zit echter een harde koppeling, naar de primary key van het producttype. Wanneer er gebruik gemaakt wordt van een medium buiten de database, zijn er twee mogelijkheden: Of de primary key wordt in het bestand opgeslagen, of andere onderdelen om de producttypes mee te identificeren.

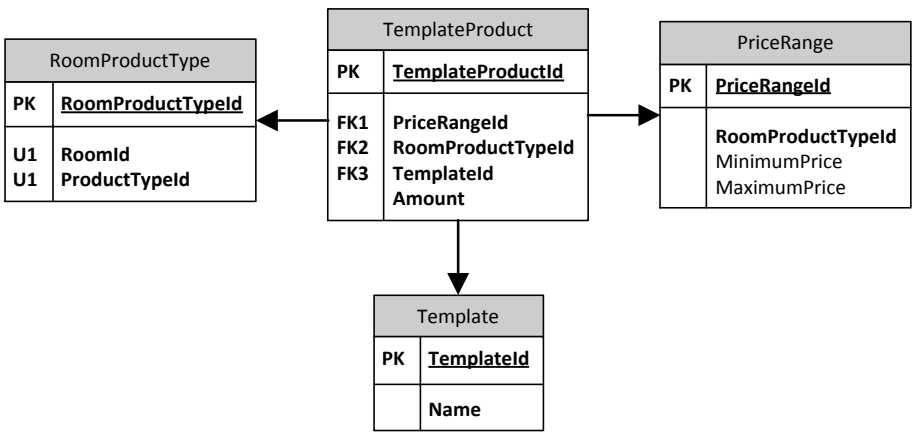
Wanneer een primary key in het bestand wordt opgeslagen, verliest deze alle significantie. Wanneer het bestand wordt bekeken, kan er aan de keys niet worden gezien welk product hierbij hoort. Voor het aanmaken of bewerken van het bestand moet dus altijd de database beschikbaar zijn om de keys hieruit over te nemen.

Andere manieren die gebruikt kunnen worden om het product te identificeren zijn bijvoorbeeld de naam van het producttype. Het nadeel hiervan is dat het producttype in de database en het producttype in de template ongelijk kunnen gaan lopen. Stel dat het producttype eerst *TV* was, maar later veranderd is naar *Televisie*. Het templatebestand is dan niet meer geldig.

Om deze redenen heb ik ervoor gekozen om ook de templates op te nemen in de database. Door de relationele integriteit is het niet mogelijk om producttypes toe te voegen die niet in de database aanwezig zijn. Bij het bouwen van de templates hoeft er ook geen gebruik gemaakt te worden van keys, omdat dit door de applicatie wordt geregeld.

Een nadeel van deze opzet is dat de templates enkel kunnen worden opgebouwd bínnen de applicatie. Een xml-bestand bijvoorbeeld kan ook worden opgebouwd buiten de applicatie om. Dit nadeel weegt echter niet op tegen de voordelen van de database. Daarbij zal de wens niet snel ontstaan om templates te kunnen bouwen buiten de applicatie om.

Om templates binnen de database mogelijk te maken, heb ik twee extra tabellen in leven geroepen: `Template` en `TemplateProduct`.



Door deze opzet te gebruiken kunnen er meerdere templates worden aangemaakt. Zo zou er onderscheid gemaakt kunnen worden tussen verschillende prijsklassen. In een huis in de dure prijsklasse is het logischer om duurder meubilair te plaatsen, en in een huis in een lagere prijsklasse goedkoper meubilair. Zo kan het huis standaard zo relevant mogelijk worden ingedeeld.

Omdat een product nu aan een `RoomProductType` en aan een specifieke `PriceRange` hangt, en een aantal ook moeten deze ook bekend zijn bij de template. Hiervoor is de tabel `TemplateProduct` bestemd. Hierin staan alle producten die bij de specifieke template horen.

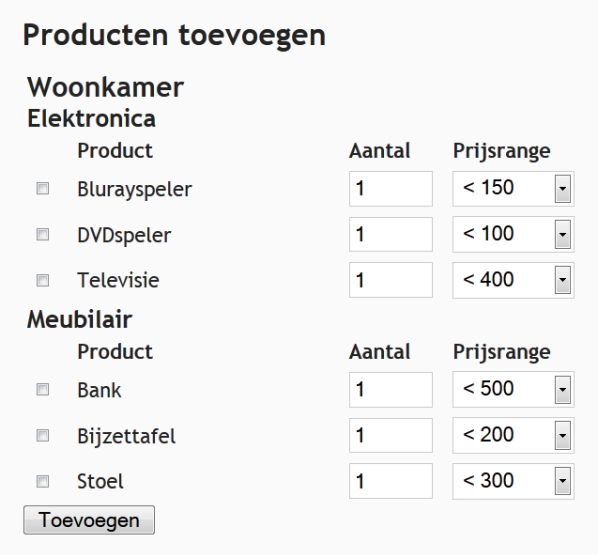
Wanneer er een conflict wordt aangemaakt en er voor een template gekozen is, moeten de producten uit de template in de `Product`-tabel worden toegevoegd. Omdat in de `TemplateProduct`-tabel alle nodige informatie zit om de producten bij het conflict te vullen, is het een kwestie van over de producten in deze tabel te itereren, en deze aan de `Product`-tabel toe te voegen.

Door deze opzet is mogelijk om via een managementonderdeel de templates te beheren. Wanneer er producten in de database bijkomen kunnen ze makkelijk aan de templates worden toegevoegd door een rij in de tabel aan te maken, en kunnen er ook nieuwe templates worden aangemaakt. Hierdoor zijn de templates flexibel en voorbereid op wijzigingen binnen de producttypes.

Bij het gesprek met mijn begeleider was aan bod gekomen dat hij graag zag dat een gebruiker een lijst met producten voorgeschoteld krijgt, waarbij hij kan aanvinken welke hij wel en welke hij niet aan het conflict wil toevoegen.

In het vorige prototype gebruikte ik voor de verschillende kamers en producttypes een dropdownlist. Deze twee dropdownlists lieten zich goed vertalen naar een lijst. In deze lijst wordt er gegroepeerd op kamer en productcategorie, waarna de producttypes die onder deze groeperingen hangen in een lijst worden weergegeven. In deze lijst kunnen de gebruikers vinkjes zetten bij de producten die ze aan de inventaris willen toevoegen. Hier kunnen ze vervolgens een aantal en een prijsrange bij aangeven.

Het prototype wat hieruit volgde zag er als volgt uit. Hierin zijn voor de leesbaarheid enkel wat producten toegevoegd aan de woonkamer. In de praktijk zullen er echter meerdere kamers in deze lijst voorkomen.



Bij het bouwen hiervan stuitte ik echter op twee problemen met dit prototype. Ten eerste kan de lijst snel onoverzichtelijk worden door de grote lijst met vinkjes, en kan er snel uit het oog worden verloren welke producten wel, en welke producten niet zijn geselecteerd.

Ten tweede kunnen er op deze manier meerdere van dezelfde producten worden geselecteerd, door het aantal aan te passen. Wanneer er in de woonkamer zes eetkamerstoelen aanwezig zijn, kan de prijsrange worden aangegeven, en zes bij het aantal.

Wanneer er echter zes eetkamerstoelen in de woonkamer aanwezig zijn, én stoelen die onderdeel zijn van het interieur, treedt er een probleem op met deze opzet. Het is immers niet mogelijk om twee type stoelen toe te voegen. In de situatie dat je zes eetkamerstoelen en twee luie stoelen hebt, is er dus een probleem, omdat de luie stoelen een stuk duurder zijn dan de eetkamerstoelen.

Dit kan op een extreem hacky manier worden opgelost door de gebruiker door acht bij het aantal in te vullen en de prijsrange iets hoger te kiezen, maar dit is zo'n beetje de minst wenselijke situatie om gebruikers in te brengen. Om deze problemen op te lossen, moest ik dus op zoek naar een andere manier op producten aan het conflict toe te voegen.

Hiervoor heb ik inspiratie gevonden bij de manier waarop een webwinkel werkt. Een gebruiker bladert eerst door de catalogus en voegt artikelen toe aan zijn winkelmandje. Vervolgens rekent hij alle artikelen af.

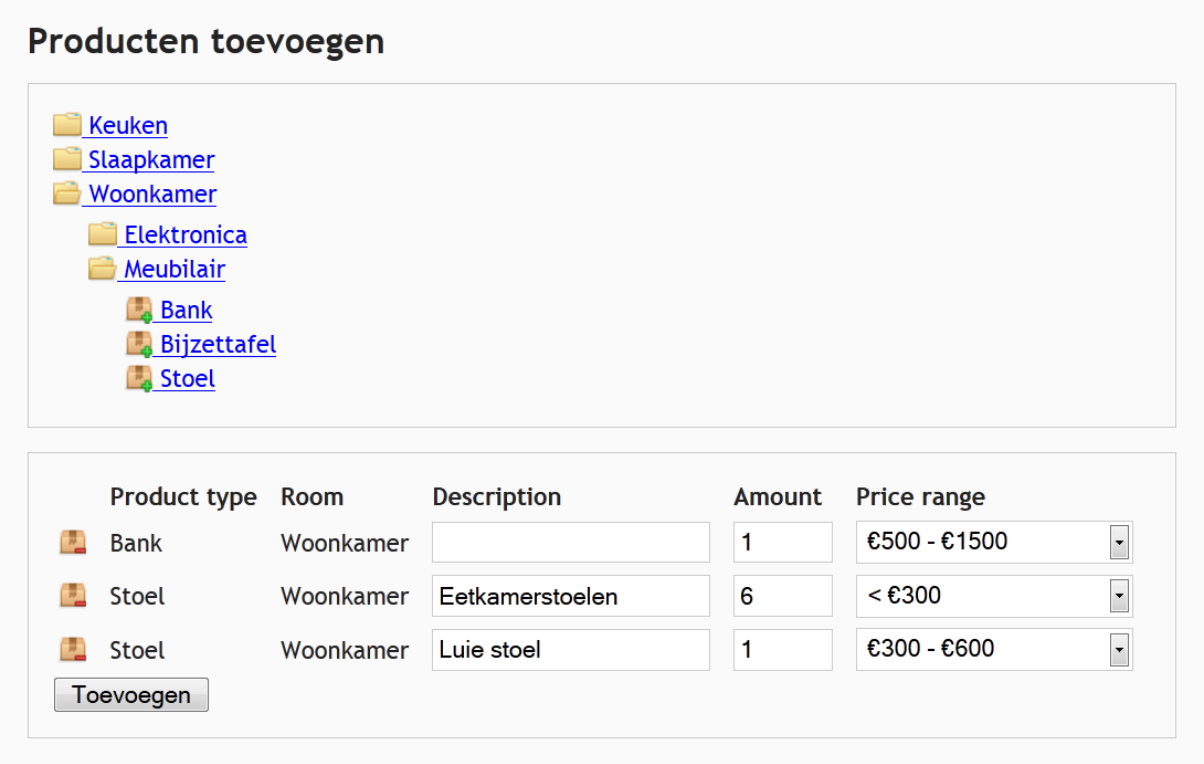
Een zelfde soort opzet wilde ik ook gebruiken voor het toevoegen van producten aan het conflict. Een gebruiker kiest uit producttypen uit de catalogus, voegt deze toe aan een tijdelijke lijst, en voegt deze vervolgens allemaal toe aan het conflict. In de tijdelijke lijst kunnen vervolgens ook de aantallen en de prijsrange worden opgegeven.

Door deze opzet zit de gebruiker niet meer vast aan één producttype, maar kan hij meerdere producten van hetzelfde type toevoegen aan de lijst. Ook is de lijst een stuk overzichtelijker, omdat hier enkel producten in staan die de gebruiker wil toevoegen, en niet enkel de catalogus.

Om de producten in de catalogus te groeperen heb ik gebruik gemaakt van een tree, welke ongeveer gelijk werkt als die binnen de Windows Explorer. Op het bovenste niveau staan de kamers, en deze kunnen uitgeklapt worden om de categorieën te tonen, welke op hun beurt ook weer uitgeklapt kunnen worden op de producttypes te laten zien. Wanneer op een van deze producttypes wordt geklikt, wordt deze aan de lijst toegevoegd.

Omdat het zo mogelijk is om meerdere van dezelfde producten toe te voegen, wordt het hiermee ook wenselijk dat er een beschrijving aan het product toegevoegd kan worden. Wanneer een gebruiker bijvoorbeeld twee bankstellen heeft, een tweezitter en een driezitter, is het voor het overzicht handig dat hij bij elk bankstel kan aangeven welke dit is.

Dit prototype zag er als volgt uit.



Het idee van de omschrijvingen sloeg aan bij mijn begeleider om dezelfde reden als bij mij, omdat het meer structuur in de lijst geeft. De tree sloeg helaas minder aan. Hij gaf aan dat hij in het verleden slechte ervaringen gehad heeft met klanten en trees, namelijk dat klanten hier niet goed mee om kunnen gaan. Trees zijn bij ontwikkelaars bekend en kunnen ermee werken, maar de normale computergebruiker heeft hier moeite mee. Naar aanleiding van dit gesprek kon er met het derde prototype worden gestart.

7.2.3 Het derde prototype

Mijn begeleider gaf aan om toch echt naar een lijst te kijken. Binnen een lijst zijn ook mogelijkheden tot filteren, bijvoorbeeld door te filteren op kamer of producttype. Het probleem dat er meerdere verschillende producten van hetzelfde type kunnen zijn, had hij in eerste instantie niet voorzien, maar erkende hij wel. De oplossing die hij hiervoor had was een knopje waarmee de rij in de lijst gedupliceerd kan worden. Op die manier konden er toch meerdere producten van hetzelfde type in de lijst weergegeven worden.

Mijn begeleider had ook zelf nog verder nagedacht over de implementatie van dit type conflict, en kwam met een aantal andere punten. Ten eerste kunnen we bij het verdelen van de spullen ook te maken krijgen met producten die niet in ons systeem staan. Deze producten moeten toch toegevoegd kunnen worden.

Ook de manier waarop er nu met kamers wordt omgegaan moest wat verder worden uitgebreid. Nu werd een kamer geïdentificeerd door de producten die hierin aanwezig zijn. Wanneer een huis echter drie slaapkamers heeft, is dit niet voldoende. Het moest dus mogelijk zijn om meerdere instanties van een specifieke kamer aan te maken, waar producten in geplaatst konden worden.

Door deze nieuwe inzichten moest de database worden aangepast. Omdat er instanties van kamers gemaakt moesten worden, komt hier een extra tabel in het spel die de instantie van de kamer voorstelt, `RoomInstance` genaamd. Hieraan kan een beschrijving worden gehangen om de kamer te identificeren, zoals *slaapkamer van Henk*. Ook moeten de producten nu aan deze tabel worden gekoppeld om aan te geven in welke kamer ze aanwezig zijn.

Binnen het conflict zijn er nu twee type producten waarmee rekening gehouden moet worden, een product wat binnen ons systeem aanwezig is, en een product wat de gebruiker zelf toegevoegd heeft. Deze heb ik respectievelijk `CatalogProduct` en `CustomProduct` genoemd. De producten delen echter ook een aantal eigenschappen. Beide typen producten hebben een beschrijving, en belangrijker nog, hebben ze allebei een koppeling naar de `RoomInstance`. Ook zijn er een tweetal koppelingen die naar beide type producten lopen, namelijk die van het product naar het conflict lopen, en de koppeltabel tussen `ProductPriceCategory` die het product en de `PriceCategory` koppelt.

Hiervoor heb ik in overervingsrelatie gemaakt in de database. Zowel `CatalogProduct` als `CustomProduct` erven over van `Product`.

Wanneer er een overervingsrelatie in een database moet worden gemodelleerd, zijn hier drie mogelijkheden voor: table per hierarchy, table per concrete type en table per type.

In het eerste type relatie, table per hierarchy, wordt er per hiërarchie een tabel gemaakt. `Product`, `CatalogProduct` en `CustomProduct` komen dan in dezelfde tabel terecht. Op basis van de waarde in een extra kolom wordt er aangegeven welk type product er in de tabel aanwezig is. Omdat óf de set aan waarden van het ene type product, óf de andere set in de tabel aanwezig is, moeten alle kolommen `NULL` toestaan. Eventueel kan met een check toch worden afgedwongen dat de kolommen gevuld zijn, op basis van de ingevoerde waarde in de extra kolom.

Het tweede type relatie is van het type table per concrete type. Elk concreet type krijgt hierbij een eigen tabel, in mijn geval zou dat twee tabellen opleveren, de tabellen `CatalogProduct` en `CustomProduct`. De kolommen die bij `Product` horen zullen in beide tabellen aanwezig zijn, beide tabellen zullen zo de kolom `Description` krijgen.

Het derde type relatie, table per type, lijkt het meeste op de relaties zoals ze in Object-Oriented Programming aanwezig zijn: een tabel voor het base type, in dit geval Product, en een tabel voor elk subtype. Dit levert dus drie tabellen op. De primary key van de tabellen voor de subtypes zijn in dit geval ook de foreign key naar het base type.

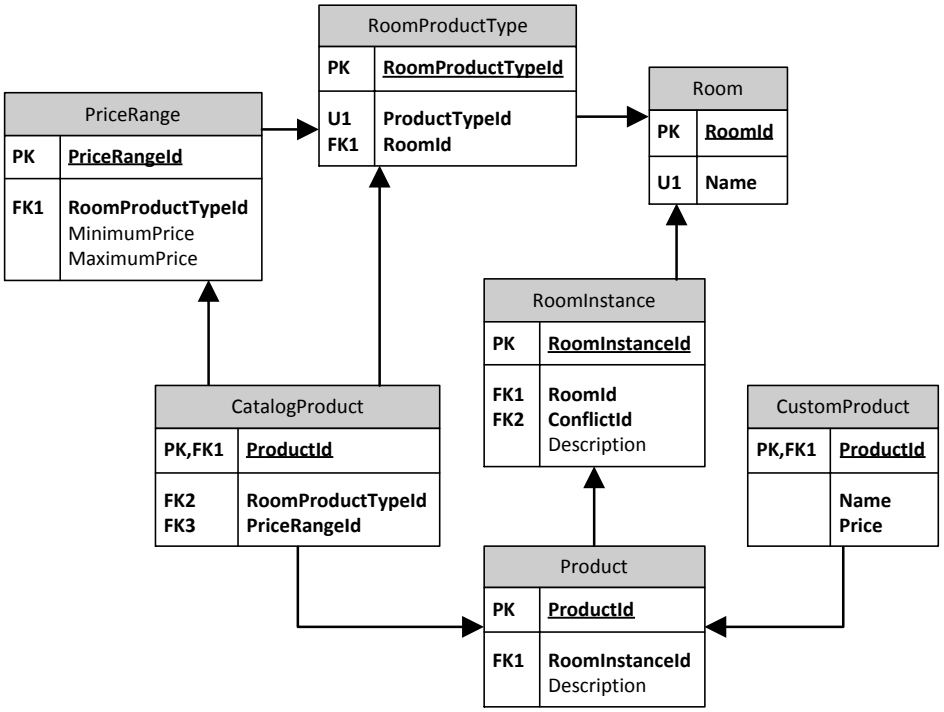
Omdat er een koppeling tussen `PriceCategory` en `Product`, en tussen `Product` en `Conflict` aanwezig is, valt het tweede type relatie af. Hiervoor zou het namelijk vereist zijn om een koppeling naar allebei de type producten te leggen, wat het beheren van de koppelingen lastiger maakt. Bij de andere twee type relaties hoeft er maar een enkele relatie te worden gelegd.

Zowel table per type als table per hierarchy waren mogelijk bij dit type probleem. Wanneer er gebruik gemaakt wordt van een table per type, kost dit een extra join. Echter wordt de referentiele integriteit wel beter gewaarborgd. Binnen de database kan worden aangegeven dat de kolommen die naar andere tabellen verwijzen niet `NULL` mogen zijn, bij table per hierarchy kan dit alleen op een omslachtige manier middels een check.

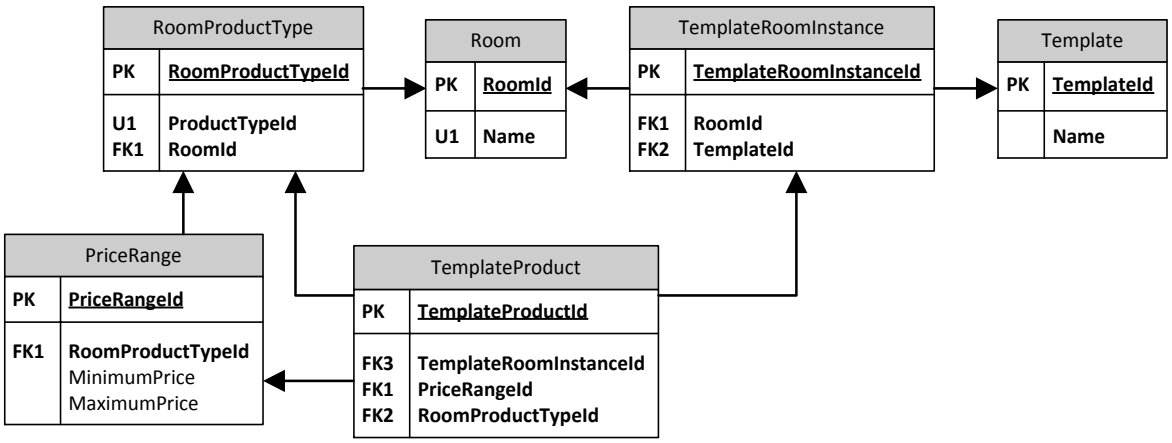
Ook is een table per type-relatie beter bestendig tegen eventuele veranderingen. Als er een tabel bijkomt die enkel naar een `CustomProduct` kan verwijzen, kan dit makkelijk wanneer er een concrete tabel is voor dit type. Wanneer er een derde type product bij zou komen, kan dit ook makkelijker met een table per type-relatie, omdat er dan enkel een tabel toegevoegd moet worden. Bij table per hierarchy moet de tabel worden aangepast, en groeit het aantal kolommen ook per type wat erbij wordt opgeslagen.

Ten slotte komt deze manier van relatie ook het meest overeen met de manier waarop het in code werkt, in code is een base class voor `Product`, en subclasses voor `CatalogProduct` en `CustomProduct`. Deze drie classes zijn ook in de database waar de nemen.

Om deze redenen heb ik ervoor gekozen om als relatie tussen deze tabellen een table per type-relatie te gebruiken. Dit onderdeel van de database ziet er als volgt uit.



Doordat er nu een instantie van een kamer wordt gemaakt waar producten aan gehangen worden, moet dit ook in de templates gebeuren. Een template voor een huis met drie slaapkamers is anders niet mogelijk. De templates in de database zien er nu als volgt uit.



Omdat het uiterlijk van de nieuwe lijst erg veel op het uiterlijk van het eerdere prototype lijkt, heb ik hier geen prototype van gemaakt, maar heb ik in dit geval gebruik gemaakt van enkele snelle schetsen om het concept met mijn begeleider kort te sluiten.

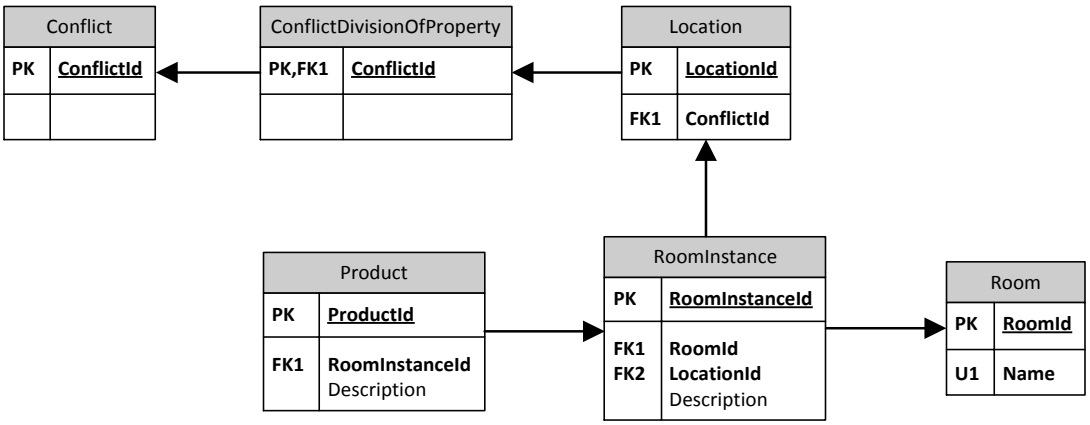
Mijn begeleider was blij met de voortgang die ik gemaakt had, en vond de database en de schetsen er goed uitzien. Volgens hem waren we bijna op het punt aangebroken waarop we konden beginnen met het implementeren “voor het echie”. Er waren nog enkele details waarover wij gesproken hebben, welke tot enkele wijzigingen leidde.

7.2.4 Het vierde prototype

Er waren nog een tweetal punten waar mijn begeleider en ik over gesproken hebben. Het eerste punt was iets waar hij nog niet aan had gedacht, en dat was dat er binnen een verdeling sprake kan zijn van meerdere locaties. Een locatie kan het huis zijn waar de spullen van worden verdeeld, maar ook een boot of een vakantiehuisje. De kamers moeten dus nog gegroepeerd worden per locatie. Om dit te doen heb ik de tabel `Location` geïntroduceerd.

Waar voorheen de kamers gekoppeld waren aan het conflict, zit daar nu dus de tabel Location tussen. Op deze manier kan een conflict meerdere locaties hebben en de locaties kunnen meerdere kamers hebben. De spullen hangen vervolgens aan een kamer.

Dit ziet er in de database als volgt uit.



Ten tweede hebben we het gehad over de rol die prijzen spelen binnen de applicatie. In mijn ontwerp spelen deze een redelijk belangrijke rol. Wanneer er een product wordt toegevoegd, moet

hier ook een prijsrange bij gegeven worden. Mijn begeleider gaf aan dat hij dit wat minder belangrijk wilde maken.

Het opstellen van de lijst met producten moet in eerste plaats voor structuur zorgen en moet duidelijk maken welke producten er zijn. Wanneer er een conflict is over het verdelen van producten, komt de prijs pas om de hoek kijken, omdat deze gebruikt wordt bij het maken van een verdeling. In eerste plaats wordt er dus geen prijs opgegeven, en kon de `PriceRange` bij `CatalogProduct` dus nullable worden.

Met deze wijzigingen is het prototypestadium afgesloten, en kon er overgegaan worden naar de daadwerkelijke bouw van dit onderdeel. Het uiteindelijke prototype zag er als volgt uit:

8 Sprint drie: begin van de implementatie van het conflict

In de vorige sprint is er veel aandacht besteed aan de manier waarop het conflict boedelverdeling moet werken. Met deze sprint begint het implementeren van dit conflict.

Bij het oplossen van een conflict wilden we dat alles zo vloeiend mogelijk zou verlopen. We wilden voorkomen dat gebruikers door een eindeloos aantal schermen moest navigeren om zijn huis in kaart te brengen, en de spullen te inventariseren.

Om dit te waarborgen wilden we veel dynamisch opbouwen. Neem het voorbeeld van het toevoegen van een locatie. Hierbij moeten een aantal kamers worden toegevoegd. Een methode hiervoor is een knop maken waarmee een kamer kan worden toegevoegd, waarna er naar een nieuw scherm wordt genavigeerd. Hierin wordt gekozen om welke kamer het gaat, en eventueel kan er een omschrijving worden opgegeven om bijvoorbeeld drie slaapkamers uit elkaar te houden. Hierna slaat de gebruiker de kamer op, en wordt hij weer teruggestuurd naar het eerste scherm.

Hiervoor moeten er veel handelingen worden verricht, die de applicatie niet erg dynamisch maken. Bij het toevoegen van een locatie wordt een gebruiker door twee schermen gestuurd, en moet hij ook twee keer wachten tot de pagina verzonden is. Dit kon beter.

Om dit aan te bieden, wilden we één scherm gebruiken waarin een gebruiker onderdelen kon toevoegen, kon wijzigen én kon verwijderen. Alle handelingen die de gebruiker kan verrichten, gebeuren dus in één scherm, waardoor het beheren van een onderdeel veel minder tijd kost.

Om dit te doen, moest er gebruik gemaakt worden van client side scripting. Dit is scripting die gedaan wordt in de browser, en dus niet op de server. De meest gebruikte en ondersteunde taal hiervoor is javascript. Met client side scripting is het mogelijk om de anders statische pagina's dynamisch te maken. Zo is het mogelijk om onderdelen op een pagina toe te voegen, te verwijderen, etc. zonder dat de pagina opnieuw opgebouwd hoeft te worden door de server. Omdat de communicatie met de server uitgespaard wordt, draait het script enkel op de client, en is hierdoor razendsnel. Met client side scripting kan meer de *feel* van een desktopapplicatie worden bereikt door de snelle response en de dynamische opbouw.

8.1 De eerste opzet: locaties toevoegen

Omdat het scherm waarmee producten kunnen worden toegevoegd aan een kamer veel afhankelijkheden hebben, zo moeten alle producten worden opgehaald, gekeken worden of ze wel of niet moeten worden aangevinkt, etc, heb ik eerst als test het scherm om kamers toe te voegen aan een locatie gebouwd. Dit scherm heeft veel minder afhankelijkheden waardoor het opzetten van een eerste versie een stuk makkelijker was.

Zoals ik al schreef, was het de bedoeling dat in één scherm er kamers konden worden toegevoegd, konden worden gewijzigd en konden worden verwijderd. Hoe ik dit in de user interface verwerkt heb, is te zien in onderstaande afbeelding.

Het scherm ziet er zo uit op het moment dat er al kamers zijn toegevoegd, en de locatie opnieuw wordt geopend. Wanneer de locatie aangemaakt wordt, zijn er immers nog geen kamers aan toegevoegd.

Wanneer er op de knop “Kamer toevoegen” wordt gedrukt, komt er een nieuwe rij in de tabel bij, die er hetzelfde uit ziet als de bestaande rijen. Het type kamer moet worden gekozen uit de lijst, en er kan eventueel een omschrijving worden toegevoegd zoals hierboven bij de slaapkamers is gedaan.

De rijen die al in de tabel stonden, kunnen worden aangepast. Zo kan het type kamer worden gewisseld, of kan de omschrijving worden toegevoegd of worden gewijzigd. Ook kan de kamer volledig worden verwijderd, door het vinkje voor de rij weg te halen.

Op het moment dat er op de “Opslaan”-knop wordt gedrukt, moeten al deze handelingen worden verwerkt. Hiervoor moeten een aantal dingen worden bijgehouden.

Ten eerste moet er worden bijgehouden of een rij in de tabel gevuld is vanuit de database, of dat deze door de gebruiker toegevoegd is. Op het moment dat de rij door de gebruiker is toegevoegd, moet de kamer worden toegevoegd aan de database als het vinkje voor de rij gevuld is.

Ten tweede moet bij de rijen die uit de database komen worden bijgehouden wat het ID in de database is. Als het vinkje voor een rij is weggehaald moet de kamer met dat ID worden verwijderd, en als er iets is gewijzigd aan de rij, moet de kamer met dat ID worden geüpdate.

Ter optimalisatie moet er ook worden bijgehouden bij bestaande rijen of ze gewijzigd zijn. Standaard is dit niet zo, maar als een vinkje wordt omgezet, of het type kamer of beschrijving worden gewijzigd, is dit wel het geval. Bij de kamers zal deze optimalisatie niet heel veel uitmaken, maar bij de producten wel. Als de tabel met producten straks bijvoorbeeld 200 producten bevat, en er zijn er drie gewijzigd, wordt het versturen en updaten van 197 producten uitgespaard, wat een aanzienlijke winst oplevert in zowel de data die verstuurd moet worden, als het werk wat de server moet verrichten.

In eerste instantie had ik deze eigenschappen opgeslagen als attributen in de rij van de tabel. Voor dit soort toepassingen zijn de `data-(…)`-attributen bestemd, hiermee kunnen namelijk eigen attributen worden gedefineerd. Een rij waarin een bestaande kamer staat, zag er als volgt uit.

```
<tr data-existing="true" data-rowId="1" data-existing="false">
```

Nadat ik hier een voorbeeld van geïmplementeerd had, heb ik een voortgangsgesprek met mijn begeleider gehad. Hij gaf aan dat hetgene wat ik probeerde te bereiken, Op een makkelijkere en nettere manier te bereiken is via een javascript-library die KnockoutJS heet. Omdat mijn methode niet zo heel netjes was (ik moest zelf alle events in de gaten houden of er iets was gewijzigd en dit in de code updaten, etc.) ben ik hiernaar gaan kijken, en kon ik ook concluderen dat KnockoutJS precies was wat ik in deze situatie nodig had.

KnockoutJS is een javascript-library waarmee een variant op het Model View ViewModel-pattern (MVVM) kan worden geïmplementeerd. Dit is een pattern waarmee code van de user interface kan worden gescheiden. Hierbinnen is er een Model, een View en een ViewModel, zoals de naam al zegt.

De Model representeert de business logic binnen de applicatie. Omdat de business logic niet in javascript geschreven is, maar in C#, komt dit in dit geval niet terug. Binnen de applicatie is een Model bijvoorbeeld een gebruiker of een product in de productlijst.

De View is de user interface, welke in dit geval in HTML is gemaakt. Deze bestaat uit verschillende elementen als textboxes, checkboxes, radiobuttons, etc.

De ViewModel is een datarepresentatie van de data in de View. Wanneer er bijvoorbeeld een lijst van locaties wordt getoond in de view, zit de data hierachter (een collectie van kamers) in de ViewModel.

KnockoutJS neemt al het updatewerk van de View en de ViewModel uit handen van de programmeur. De programmeur hoeft enkel het ViewModel op te stellen, en op een element in de View aan te geven op welk dataelement binnen de ViewModel dit correspondeert. Wanneer een gebruiker de waarde van een tekstveld in de View wijzigt wat door KnockoutJS is gekoppeld, wordt de waarde automatisch in de ViewModel aangepast. Dit werkt ook andersom, als de waarde in de ViewModel wordt aangepast wordt de View ook automatisch geüpdate.

KnockoutJS kan ook met arrays werken. De programmeur definieert een template[1] waarin staat hoe de rij eruit komt te zien. KnockoutJS genereert vervolgens zelf het benodigde aantal rijen, en vult deze met de data uit de array.

Dit maakt het werken met de lijst voor mij een stuk eenvoudiger. Om de lijst te tonen hoef ik nu enkel een ViewModel te maken met hierin een array waarin de kamers staan, en deze te vullen, waarna KnockoutJS zelf de lijst in de View opbouwt. Als er een kamer toegevoegd moet worden aan de locatie, hoeft deze ook enkel aan de array in de ViewModel te worden toegevoegd.

[1]Template is een veelgebruikte naam voor sjabloon. Hierin wordt de opmaak en indeling van de data bepaald, en deze wordt later gevuld.

Javascript is een dynamische taal. Dit maakt het mogelijk om “on the fly” objecten op te bouwen. Ook maakt het de array niet uit welk type data erin staat, alle type objecten kunnen erin geplaatst worden. KnockoutJS is hier iets strikter in, omdat de elementen binnen de template worden gekoppeld aan members van de objecten binnen de array. Op het moment dat KnockoutJS probeert om een element binnen de template te koppelen aan de member van een object binnen de array, en deze member kan niet worden gevonden, geeft KnockoutJS een foutmelding. KnockoutJS wil dus de members hebben die in de template gedefinieerd worden. Hoe de objecten verder zijn opgebouwd maakt het niet uit.

Door deze dynamische eigenschap, was het erg makkelijk om de array binnen de ViewModel op te bouwen. Voor alle bestaande kamers die uit de database kwamen, heb ik een object aangemaakt waarin staat dat de kamer al bestaat in de database, wat de ID van de rij in de database is, en of de data in het object is gewijzigd. Wanneer er een nieuwe kamer wordt toegevoegd, maak ik een object aan waarin staat dat de kamer nog niet bestaat in de database. De members van deze objecten komen overeen met de attributen op de rij in de tabel, wat ik in de eerste opzet had bedacht.

Op het moment dat er op de “Opslaan”-knop wordt gedrukt, moeten de rijen naar de server worden gestuurd. Zoals ik al schreef, worden ter optimalisatie niet alle rijen naar de database gestuurd, maar enkel de rijen waar iets mee gebeurd is. Hiervoor worden de volgende stappen doorlopen voor elke rij.

Eerst wordt er gekeken of de rij überhaupt is gewijzigd. Om dit te controleren heb ik een extensie op KnockoutJS gemaakt. Standaard biedt KnockoutJS het datatype `Observable`. Hier zitten een aantal events op, die worden afgevuurd op het moment dat de waarde wijzigt. Dit wordt normaal gesproken door de view bijgehouden zodat deze kan worden geupdate.

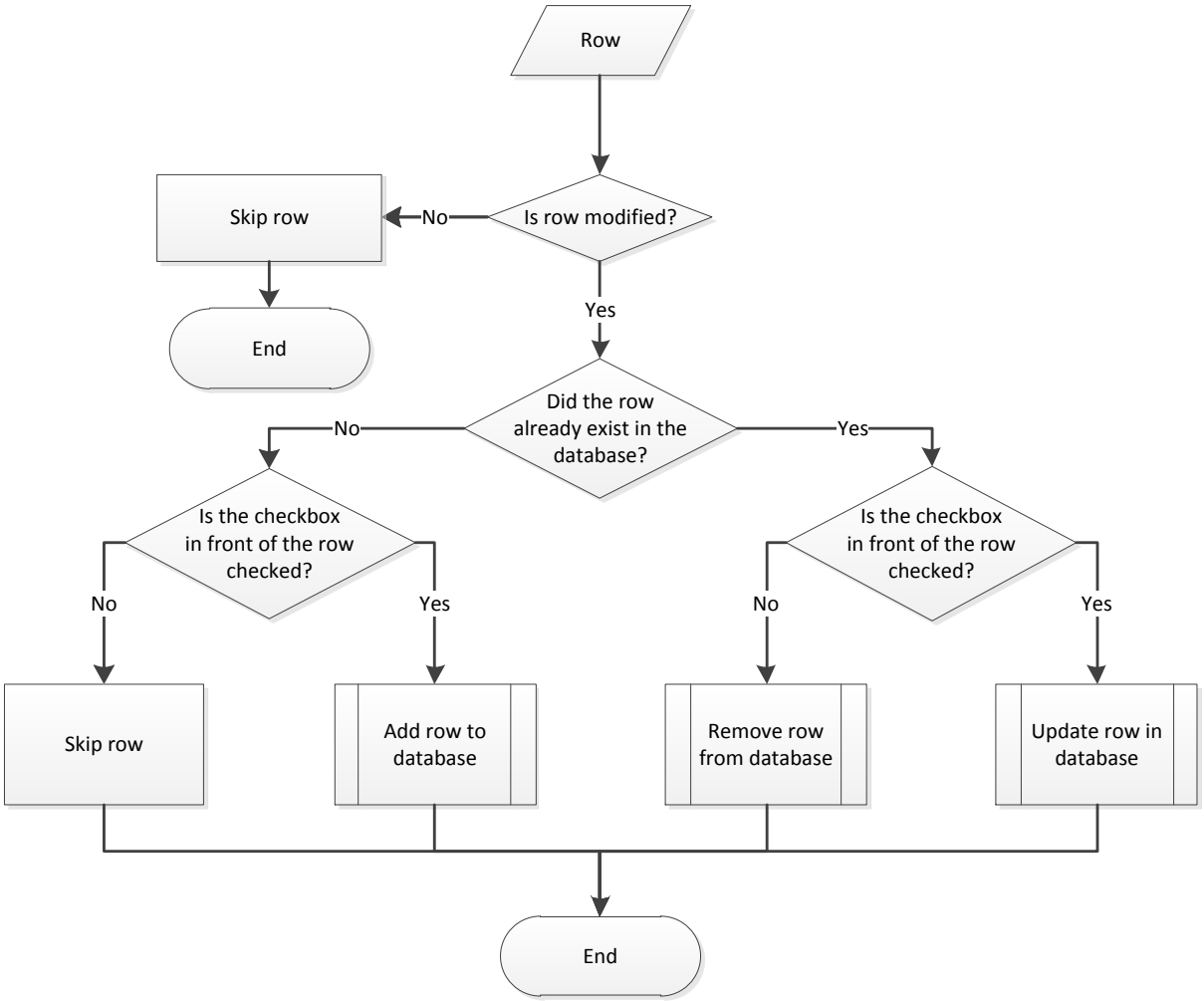
In mijn geval heb ik een eigen subklasse gemaakt van de `Observable`, `TrackedObservable` genaamd, die ook naar deze events luistert. Op het moment dat het event vuurt, en de waarde is veranderd, kan ik in de klasse aangeven dat de waarde veranderd is.

Wanneer er door de waarden van de rijen wordt gelopen, is te controleren of de waarden zijn veranderd, door op de `TrackedObservable`s in de rij te vragen of ze gewijzigd zijn. is dit bij allen false, dan zijn de waarden binnen de rij ongewijzigd, en kan de rij worden overgeslagen. Wanneer één van de waarden gewijzigd is, gaat de rij mee naar de volgende ronde.

Vervolgens wordt er op het object gecontroleerd of de rij al in de database aanwezig was. Wanneer dit het geval is, kan de rij worden geüpdate of worden verwijderd, op basis van de toestand van het vinkje voor de rij. Wanneer het vinkje voor de rij is aangevinkt, blijft de rij aanwezig in de database, maar moeten de gegevens worden geüpdate omdat er een waarde in de rij is aangepast. Als het vinkje voor de rij niet is aangevinkt, wordt de rij uit de database verwijderd. Het is dan ook niet meer van belang of er wat gewijzigd is, want de rij gaat toch weg.

Als de rij nog niet in de database aanwezig was, is wederom afhankelijk van het vinkje wat er gebeurd. Als het vinkje niet is aangevinkt, kan de rij worden overgeslagen. Een niet-aangevinkte rij komt immers niet voor in de database. Als het vinkje wel is aangevinkt, moet de rij in de database worden toegevoegd.

Om deze stappen duidelijker te maken, heb ik onderstaande flowchart opgesteld, die de stappen in een grafische weergave toont.



Om de verschillende waarden naar de server te sturen, heb ik drie wachtrijen opgebouwd, een wachtrij voor rijen die aan de database moeten worden toegevoegd, een wachtrij met rijen die gewijzigd moeten worden, en een wachtrij met rijen die moeten worden verwijderd. Ik heb hiervoor gekozen omdat alle wachtrijen andere data bevatten.

Voor de rijen die worden verwijderd, is enkel het ID van de rij nodig. Wanneer er een nieuwe rij aangemaakt moet worden, is de data die in deze rij hoort benodigd. Als er ten slotte een rij moet worden geüpdate, is naast de data die in de rij hoort ook het ID van de rij benodigd, omdat dit de rij die geüpdate moet worden identificeert.

De wachtrijen worden vervolgens naar de server gestuurd door middel van een http post. Deze wordt op de server afgevangen, en hier kan er door de wachtrijen worden gelopen. Op basis van de rij kan deze vervolgens worden toegevoegd, worden geüpdate en worden verwijderd.

8.2 Producten beheren

Door het toevoegen van locaties heb ik een idee gekregen hoe ik een dynamische layout kan opbouwen. Nu is het tijd voor het echte werk, het beheren van producten. Het beheren hiervan gaat

verder met de ideeën die ik heb opgedaan bij het volgende onderdeel, maar de basis is hetzelfde. Daarom bespreek ik de basis niet.

Bij het beheren van locaties, is er sprake van een klein aantal, en deze zijn allemaal relevant. Bij producten is dit anders. Hierbij worden er een groot aantal producten aangeboden, waarvan er maar een deel van relevantie is voor de gebruiker. Hierdoor moet er kunnen worden gefilterd, om de lijst met producten overzichtelijk te houden.

Wanneer een huis in kaart gebracht wordt, zal dit ten eerste per kamer gebeuren. Om de producten vervolgens nog verder op te delen, heb ik dit gedaan op categorie. Gebruikers kunnen zo eerst het meubilair in kaart brengen, vervolgens de elektronica, etc. Standaard worden alle categorieën weergegeven, maar gebruikers kunnen deze verder beperken.

Om goed te kunnen testen, had ik een aantal producten nodig in de database, die ik kon toevoegen. Bij het invoeren hiervan liep ik tegen een beperking van het huidige ontwerp aan. Hoe ik het systeem ontworpen had, hoort een product bij een of meerdere kamers. Een oven hoort in de keuken, en een televisie in de woon- of slaapkamer.

Echter zie je ook dat mensen producten in kamers plaatsen waar je ze in eerste instantie niet verwacht. Zo staat een wasmachine soms op zolder, soms in de keuken, of een koelkast in de schuur. Om toe te staan dat er een product wat eigenlijk niet voor de kamer bestemd is, toch in de kamer toe te staan, moest er een aanpassing in het ontwerp komen.

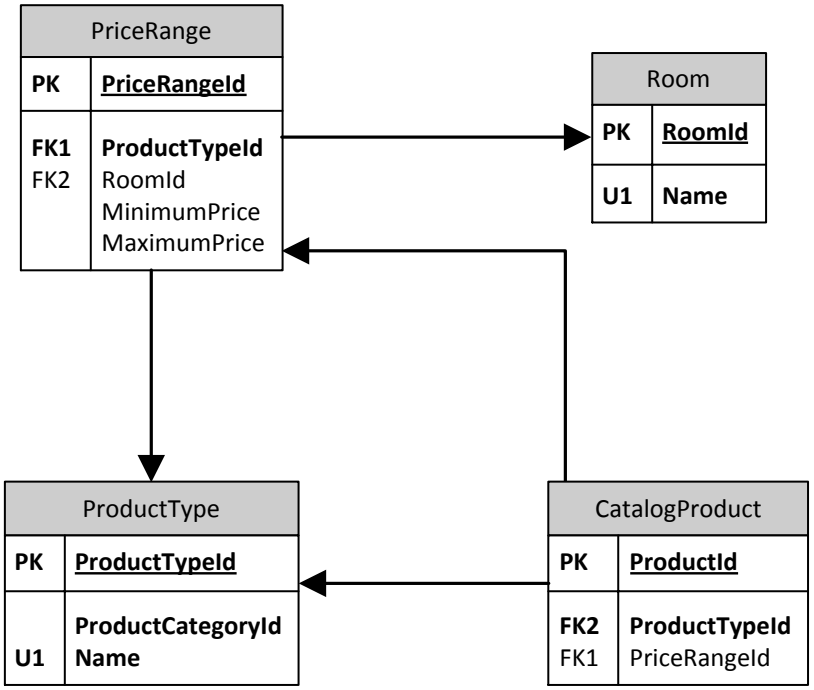
Het huidige ontwerp maakt het mogelijk om een product aan meerdere kamers te koppelen, echter wordt er bij het toevoegen van het product een harde koppeling gelegd naar de kamer-producttypecombinatie. Wanneer de gebruiker een televisie in de badkamer wil plaatsen, komt hierbij de koppeling naar de woonkamer of naar de slaapkamer mee. De koppeling bezit in deze situatie geen enkele context, en maakt het erg verwarrend.

De producten veranderen in principe niet met de kamer waarin ze geplaatst zijn. Enkel de `PriceRange`s zijn gekoppeld aan een specifieke `RoomProductType`. Dit heb ik gedaan omdat een prijs kan veranderen met de kamer waarin het product is geplaatst. Een bekend geval hiervan is een televisie die in de slaapkamer geplaatst is. Deze is over het algemeen minder duur dan de televisie in de woonkamer.

Toen ik in de gedachtegang van dit probleem zat, kwam ik er achter dat producten als een televisie meer een uitzondering zijn. Als mensen een koelkast in de schuur willen plaatsen zullen de `PriceRange`s minder verschillen, als ze niet helemaal verdwijnen. De afwijkende `PriceRange`s moeten dus meer als een uitzondering worden behandeld, dan als een regel.

Daarom heb ik deze koppeling losgemaakt. Wanneer een `CatalogProduct` wordt toegevoegd, wordt deze gelinkt aan een `ProductType`, en niet aan een `RoomProductType`. Ook wordt een `PriceRange` niet meer gekoppeld aan de `RoomProductType`, maar aan de `ProductType` met een koppeling naar een `Room` die null mag zijn. Wanneer de `PriceRange` aan een product wordt gekoppeld, is de koppeling naar `Room` null voor de “standaardprijzen”. Voor de uitzonderingen kan de koppeling naar `Room` worden gevuld.

De nieuwe situatie ziet er als volgt uit.



Wanneer de producten worden geladen, worden standaard alle categorieën getoond. Wanneer de gebruiker meer overzicht wil hebben, kan hij de producten filteren op een specifieke categorie. Meestal zal dit echter niet nodig zijn. Omdat alle producten geladen worden, is filteren op de server ook niet interessant, omdat hier meer tijd overheen gaat dan filteren op de client door middel van client side scripting.

Producten in een kamer plaatsen die eigenlijk niet bij de kamer horen is een ander verhaal. Omdat dit een uitzonderingssituatie is, heb ik ervoor gekozen om de producten bij deze kamers niet in te laden. Producten inladen waar in de normale gevallen niet eens naar gekeken worden, is zonde van zowel het geheugen op de client, als van de extra serverlasten die dit met zich meebrengt.

Daarom heb ik ervoor gekozen om enkel de producten in te laden die bedoeld zijn voor de kamer die ingedeeld wordt. Wanneer een gebruiker producten uit een andere kamer wilt toevoegen, worden deze producten alsnog ingeladen, via Ajax.

Ajax is een techniek waarmee informatie kan worden opgehaald, wanneer de pagina al volledig is geladen. Wanneer gebruik gemaakt wordt van traditionele webtechnieken, moet bij het ophalen van nieuwe informatie de pagina worden ververst. Dit komt omdat de verbinding bij een website niet wordt opengehouden nadat de pagina is geladen. Om nieuwe data op te halen moet er opnieuw een verbinding worden geopend.

Wanneer er gebruik gemaakt wordt van Ajax, wordt er ook een nieuwe verbinding geopend. Dit wordt echter niet gedaan door een nieuwe pagina op te vragen binnen de browser, maar door een pagina op te halen via javascript. De pagina is hierna niet veranderd, maar de data van de nieuwe pagina is binnen javascript beschikbaar. Vervolgens kan met javascript de pagina worden aangepast, en kan de nieuwe data hierin worden verwerkt.

Deze opzet heb ik hier ook gekozen. Op het moment dat een gebruiker een andere kamer selecteert, worden de producten voor deze pagina via Ajax opgehaald, en worden deze op de pagina getoond in plaats van de vorige producten. KnockoutJS kon hier mooi bij helpen, omdat ik hiermee enkel de producten in het ViewModel hoefde aan te passen. KnockoutJS Zorgde er vervolgens voor dat de user interface op de pagina automatisch aangepast werd.

Deze opzet van producten laden bracht een interessant probleem met zich mee. Doordat een product losgekoppeld is van een kamer, is een televisie die in de woonkamer geplaatst is in wezen dezelfde soort televisie die in de slaapkamer is geplaatst. Hierdoor maakt het niet uit of de televisie is toegevoegd vanuit de woonkamer of de slaapkamer, het is dezelfde televisie en het toevoegen heeft in de database ook hetzelfde resultaat.

Er treedt een probleem op, wanneer een gebruiker een product aanpast, en vervolgens naar een andere kamer wisselt die het product ook bezit. Dit is het beste uit te leggen door middel van een voorbeeld.

Stel dat de gebruiker een aantal producten in de woonkamer heeft ingedeeld, en dat deze al zijn opgeslagen. De gebruiker opent vervolgens het scherm waarin producten beheerd kunnen worden. We zien hier de bekende televisie terug, met een omschrijving. De gebruiker heeft meer details over de televisie opgezocht, en past de beschrijving aan. Vervolgens wisselt hij de productenfilter naar de slaapkamer.

De producten die bij de slaapkamer geclassificeerd zijn worden nu opgehaald, inclusief de televisie. Dit is dezelfde televisie als bij de woonkamer. Omdat de gebruiker de wijzigingen bij de televisie nog niet heeft opgeslagen, wordt echter de oude beschrijving opgehaald. In de woonkamer heeft de televisie de nieuwe beschrijving, maar in de slaapkamer de oude. De televisie is dus niet goed gesynchroniseerd tussen de kamers.

Hetzelfde treed op als een gebruiker in de ene kamer de televisie heeft verwijderd door het vinkje ervoor weg te halen, en vervolgens naar de andere kamer wisselt. Omdat in de database het product nog niet verwijderd is, staat hier het vinkje nog gewoon voor het product.

Ook met het toevoegen van producten is dit probleem aanwezig, omdat in de andere kamer de producten nog niet toegevoegd zijn. Waar er in de woonkamer bijvoorbeeld drie televisies aanwezig kunnen zijn, is dit bij de slaapkamer dan niet het geval.

Ik heb over dit probleem gesproken met mijn begeleider en zijn oplossing was eenvoudig. Voordat er naar een andere kamer gewisseld wordt, de opslaan-functie gebruiken waardoor de producten worden opgeslagen. Als de producten vervolgens van de server worden gehaald, zijn hier de wijzigingen in doorgevoerd omdat deze voor het ophalen zijn verwerkt.

Ik vond dit echter geen optie. Producten moeten pas opgeslagen worden op het moment dat de gebruiker hier expliciet opdracht voor geeft, door op de opslaan-knop te drukken. Tussendoor opslaan zonder dat de gebruiker weet dat dit is gebeurd is zeer ongebruikersvriendelijk, omdat er recht tegen het mentale model van de werking van de applicatie ingegaan wordt. De producten mogen pas opgeslagen worden op het moment dat de gebruiker wil dat ze worden opgeslagen, en niet op het moment dat het systeem wil dat de producten opgeslagen dienen te worden.

Om dit probleem op te lossen zonder dat er tussentijds opgeslagen wordt op de server, moest ik op zoek naar een andere oplossing. Hierbij heb ik voor ideeën gekeken naar de manier waarop een ORM (Object Relational Mapper), zoals Entity Framework wat ik binnen de applicatie gebruik, omgaat met objecten die uit de database zijn opgehaald.

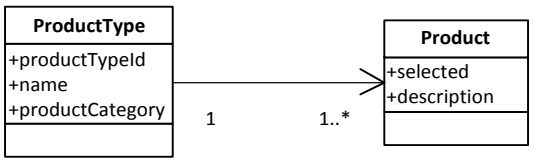
Een ORM heeft namelijk hetzelfde probleem met objecten die uit sync kunnen lopen als ik. Wanneer twee keer hetzelfde object uit de database wordt gehaald, moeten beide calls hetzelfde object terug geven. Wanneer dit niet gebeurd, treden er vreemde dingen op wanneer beide objecten worden gewijzigd. Het wordt immers erg lastig om de wijzigingen van beide objecten door te voeren in de database, zonder dat de tweede update de wijzigingen van de eerste update overschrijft.

Een ORM maakt hiervoor gebruik van het Identity Map pattern. De Identity Map houdt alle objecten bij die uit de database zijn opgehaald. Wanneer er een object opgehaald wordt, wordt er in de Identity Map gekeken of het object hier al in zit. Als dit het geval is wordt het object hieruit gehaald. Als het object nog niet geladen is, wordt deze vanuit de database opgehaald, en aan de Identity Map toegevoegd.

Ditzelfde pattern heb ik op de client in javascript geïmplementeerd. Wanneer er een kamer wordt opgehaald, wordt er voor elk product in de kamer gekeken of deze al in de Identity Map aanwezig is. Dit product wordt dan teruggegeven, en anders wordt er een nieuw product aangemaakt.

Een probleem bij het dupliceren van een product is echter dat er een kopie van het product wordt gemaakt. Omdat de Identity Map de producten opslaat op basis van de unieke primary key uit de database, konden deze gedupliceerde producten hier niet aan worden toegevoegd.

Om dit op te lossen moest ik de objecten lostrekken. Waar voorheen de objecten platgeslagen werden op de server, wordt er nu een hiërarchie in Javascript opgebouwd. De waarden die door de gebruiker ingevuld kunnen worden, staan in het type `Product`, en de gegevens die voor alle producten overeenkomen staan in het type `ProductType`.



De Identity Map houdt de producttypes bij. Wanneer een producttype al bestaat, geeft hij deze, plus alle bijbehorende producten terug. Bestaat hij nog niet, wordt er een nieuw producttype en het vereiste aantal producten aangemaakt, en wordt het producttype in de Identity Map opgenomen. Als een product nu gedupliceerd wordt, komt er een nieuw product bij in het producttype, maar veranderd het producttype niet. De Identity Map kan zo gebruikt worden, onafhankelijk van het aantal duplicaten van het product er aanwezig zijn.

9 Sprint vier: het achterliggende management van de boedelverdeling

Binnen de conflictoplossing voor de boedelverdeling is het mogelijk om een inventaris van het huis op te bouwen. Hierbij bouwt de gebruiker eerst een locatie op, en vult deze locatie vervolgens met producten. Hierbij kan hij vanaf nul beginnen, of hij kan een template kiezen. In de template zijn een aantal kamers en bijbehorende producten opgenomen, die automatisch in het conflict worden ingevuld.

Deze catalogus van producten, de type kamers, de categorieën, etc. moeten echter wel in systeem aanwezig zijn voordat de productenlijst kan worden gemaakt door de gebruiker, en voordat er een template kan worden aangemaakt. Voordat er een template kan worden gebruikt om een locatie aan te maken en de producten te vullen, moet deze template ook worden opgebouwd.

In deze sprint ben ik bezig geweest met het implementeren van de functionaliteiten die nodig zijn om deze dingen te vullen. Dit onderdeel is enkel beschikbaar voor de beheerders van de applicatie, en zal niet beschikbaar zijn voor het grote publiek.

Omdat enkel de beheerders bij deze beheeronderdelen kunnen, en ze hierdoor veel minder gebruikt worden, kon ik enkele shortcuts nemen. Waar ik bij voorgaande delen er erg op gelet heb dat deze snel laden, en er niet teveel geladen werd, hoefde ik er hierbij minder rekening te houden. Het inladen van wat langere lijsten was hierdoor mogelijk, en doordat ik geen lazy loading hoefde te implementeren kon ik hier een kleine tijdswinst op behalen.

De implementatie van deze onderdelen lijkt qua technieken en opbouw erg op de onderdelen die ik al gebouwd had. De implementaties van het toevoegen van kamers, categorieën, etc. waren redelijk eenvoudige CRUD-operaties, waarbij ik net als bij de vorige onderdelen gebruik gemaakt heb van KnockoutJS. Het opbouwen van een template en het toevoegen van producten hieraan leek erg op het inventariseren van een huis.

Aan de database die ik in de vorige sprints had opgebouwd, hoefde niets te worden veranderd. Ik had immers een groot deel al getest door gebruik te maken van testdata die ik hard in de database gezet had. De praktijktest had deze dus al overleeft.

Omdat ik in deze sprint geen nieuwe keuzes gemaakt heb, maar de technieken uit de vorige sprints heb hergebruikt, ga ik niet verder in op de implementaties die ik deze sprint heb gedaan. Dit zou geen relevante informatie toevoegen, omdat de keuzes al benoemd zijn in de vorige hoofdstukken.

10 Sprint vijf: koppelen van gebruikers en conflictoplossingen

In de eerste sprint heb ik aandacht besteed aan het toevoegen van gebruikers en het inloggen. In de tweede, derde en vierde sprints heb ik mij gericht op het conflicttype boedelverdeling. Tussen deze twee was tot dusverre echter geen koppeling. Wanneer er een conflict aangemaakt werd, werden er nog geen gebruikers aan gekoppeld, en waren de conflictoplossingen dus ook niet persoonsgebonden. In deze sprint heb ik me gericht op deze koppeling tussen de gebruikers en de conflictoplossingen.

10.1 Gebruikers koppelen aan een conflict

Op het moment dat een conflict aangemaakt wordt, kan er gelijk een persoon worden gekoppeld, namelijk de eigenaar van de conflictoplossing. Deze kan vervolgens een of meerdere mensen uitnodigen voor het conflict.

Binnen de conflictoplossing is er sprake van een verschil in rollen en rechten. Zoals hierboven geschreven kan de eigenaar hiervan een andere gebruiker uitnodigen, hij heeft dus meer rechten dan een normale betrokkene. De rollen van de gebruikers zijn dus ook specifiek voor een instantie van een conflictoplossing. Een persoon kan de eigenaar van een conflictoplossing zijn, maar ook de betrokkene bij een andere conflictoplossing.

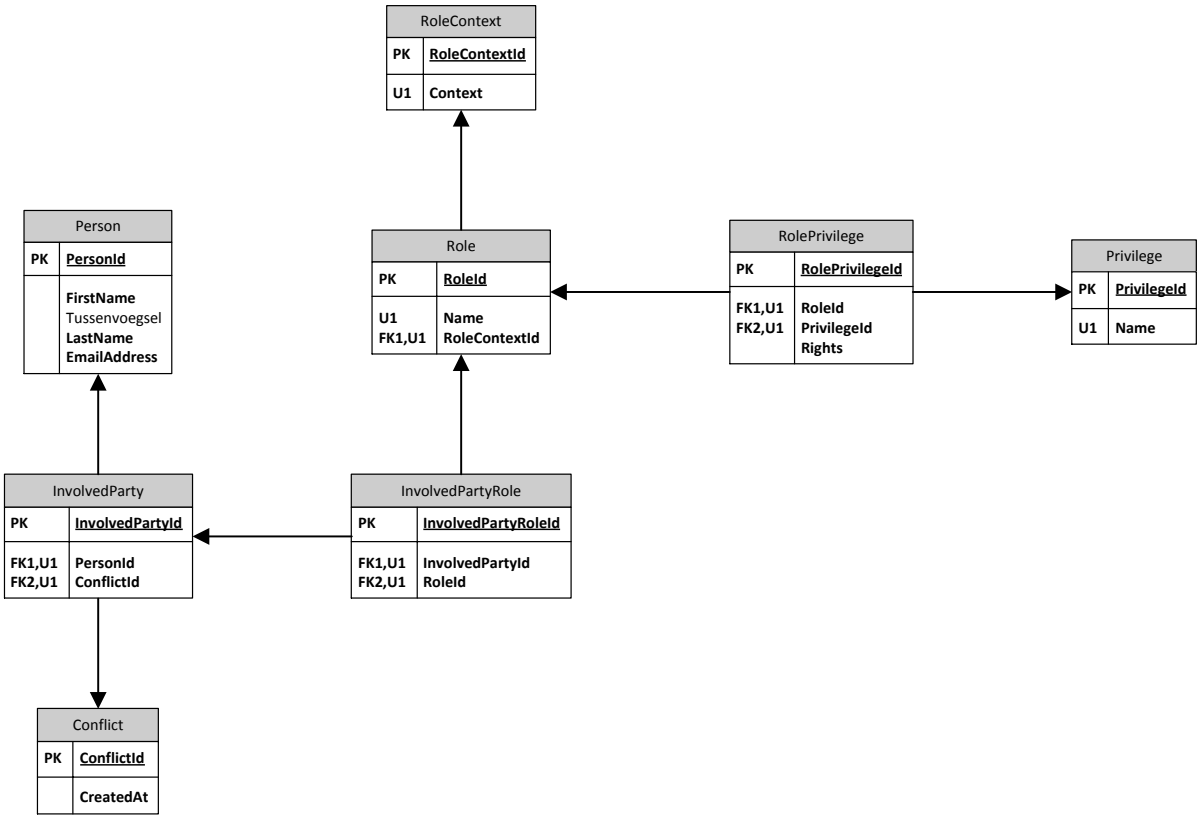
In de eerste sprint heb ik me al bezig gehouden met de manier waarop gebruikers rechten hebben binnen het systeem. De manier van rechten binnen een conflict werken in principe hetzelfde, alleen zitten deze aan een conflict gekoppeld in plaats van dat ze algemeen zijn voor het gehele systeem.

Vanuit de database gezien zijn het echter allemaal rollen en rechten, waaraan deze dan ook gekoppeld zitten. Daarom heb ik ervoor gekozen om de rollen voor het gehele systeem en de conflict specifieke rollen in dezelfde tabellen op te slaan.

De rollen die de gebruiker in het conflict heeft, zijn conflictspecifiek. Vanuit dat oogpunt is het logisch om de rollen te koppelen aan de koppeltabel tussen het conflict en de persoon. Zo kunnen er meerdere verschillende rollen zijn binnen de conflictoplossing, en kan een persoon ook verschillende rollen per conflict hebben.

Een rol met dezelfde naam kan een verschillende context hebben, afhankelijk van de entiteit waaraan hij gekoppeld is. Een gebruiker die de eigenaar van een conflict is, kan gebruikers uitnodigen, etc. maar iemand die de eigenaar van de conflictoplossing van de boedelverdeling is, kan daar weer rechten bovenop krijgen, zoals bijvoorbeeld het kunnen toevoegen van locaties. Zo kunnen rollen verschillende niveaus hebben: rollen binnen de applicatie, rollen binnen het abstracte conflict, en rollen voor concrete conflicten zoals de boedelverdeling. Hierbij kunnen er dubbele namen ontstaan, zoals de eigenaar van een conflict. Aan een rol moet dus ook een context worden gekoppeld, die aangeeft voor elk niveau.

Deze nieuwe onderdelen gecombineerd met de eerder gedefinieerde gebruikers en rollen ziet er als volgt uit.



10.2 Conflicten afschermen

Het moet natuurlijk voorkomen worden dat een gebruiker de conflicten van een ander kan bekijken. Wanneer hier geen preventie voor ingebouwd is, is dit erg makkelijk. Het adres van de pagina bevat namelijk het ID van het conflict.

Om nu een andere conflictoplossing te bekijken, kan de vier in de bovenstaande link worden vervangen voor een ander cijfer, en wordt de conflictoplossing met de bijbehorende ID getoond. Zonder filtering kan dit elke conflictoplossing zijn, ook waar de gebruiker niet bij betrokken is.

Om te voorkomen dat andermans conflictoplossingen kunnen worden bekeken, moet er filtering worden ingebouwd. Dit kan binnen de controller van het conflict gebeuren, alleen moet dit dan bij élke query die naar de database gaat worden gedaan. Hierdoor krijg ik dus heel veel duplicate code, en is de kans groot dat ik de filtering ergens vergeet, met alle gevolgen van dien. Om deze beide problemen aan te pakken heb ik de filtering op een hoger niveau gedaan.

In paragraaf 6.3.1 heb ik beschreven hoe de interface naar de database eruit ziet. Voor de filtering gebruik ik deze weer.

Voor het opvragen van rijen uit de database roep ik de Query-property op de `IRepository<T>` aan, welke een `IQueryable<T>` terug geeft. Dit is een type collectie die rijen in de database representeert. Het mooie aan deze interface, is dat deze als het ware de filters, sorteringen, etc.

opstapelt, en pas uitvoert op het moment dat er daadwerkelijk wat met de rijen wordt gedaan, zoals door de rijen heen lopen of een specifiek element uit de collectie halen.

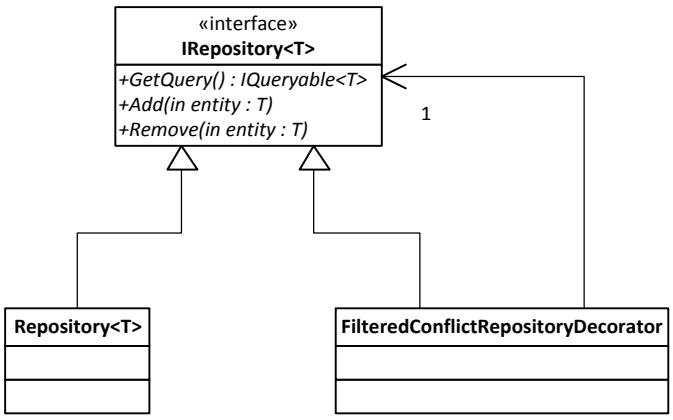
Doordat filters niet direct worden uitgevoerd, maar pas op een later moment, kon ik deze perfect gebruiken om de conflicten te filteren. Ik kan immers de filter op het conflict leggen, zonder dat gelijk alle rijen worden opgehaald die aan de filter voldoen. De `IQueryable<T>` mét het filter kan vervolgens door de overige delen in het systeem worden gebruikt, en deze kunnen hier ook andere filters, etc. overheen leggen.

In mijn controllers worden door middel van dependency injection de `IRepository<T>` de constructor in geschoten. Omdat het hier om een interface gaat, kan ik hier ook een andere implementatie onder hangen en deze kan dan op dezelfde manier door de controller worden gebruikt. Dit is wat ik heb gedaan.

Ik heb hier gebruik gemaakt van het Decorator pattern. Dit is een pattern waarbij er als het ware functionaliteit om een component heen gelegd wordt. De decorator implementeert dezelfde interface als het component, en bezit een referentie naar het component. Wanneer er een functie op de decorator wordt aangeroepen, roept de decorator dezelfde functie aan op het component. Voordat hij het resultaat van deze functie zelf teruggeeft, kan hij hier nog extra operaties op uitvoeren.

In mijn geval implementeren zowel de concrete `Repository<T>` als de `RepositoryDecorator<T>` de interface `IRepository<T>`. Wanneer op de `RepositoryDecorator<T>` de `Add`- of `Remove`-functies worden aangeroepen, speelt hij deze ongewijzigd door naar de `Repository<T>`. Wanneer de Query wordt aangeroepen, vraagt de `RepositoryDecorator<T>` de `IQueryable<T>` op van de `Repository<T>`, maar geeft deze nog niet gelijk terug. Eerst legt hij de filter eroverheen, die alle conflicten eruit filtert waar de huidige gebruiker geen betrokkene bij is. Daarna geeft hij de `IQueryable<T>` pas terug.

In UML ziet dit er als volgt uit.



Binnen de code van de controllers is er geen verschil in het gebruik tussen de `Repository<T>` en de `RepositoryDecorator<T>`, omdat binnen de code enkel de interface gebruikt wordt. Binnen de code is er dus geen weet van dat de conflicten zijn gefilterd. Enkel in de configuratie moet worden ingesteld dat de `Repository<T>` niet direct de controllers wordt ingeschoten, maar dat deze eerst moet worden “gedecorate” met een `RepositoryDecorator<T>`.

Doordat de conflicten al gefilterd zijn in de IRepository<T>, wanneer deze de controller binnen komt, hoeft er binnen de controllers en overige delen van het systeem geen rekening gehouden te worden met filtering, dit wordt immers door de RepositoryDecorator<T> gedaan. Binnen de controllers hoeft dus niet extra gefilterd te worden waardoor er geen duplicate code optreedt, en de filtering ook niet vergeten kan worden. De filtering wordt op één plek uitgevoerd, en de rest van het systeem hoeft hier geen rekening mee te houden, wat tot een erg eenvoudig en clean resultaat leidt.

10.3 Aanmaken van conflicten

Tot dusverre heb ik, wanneer ik een conflict nodig had, hier gewoon keihard een nieuwe van aangemaakt, door bijvoorbeeld `new ConflictDivisionOfProperty();` aan te roepen. Voor testscenario's is dit prima, maar dit werkt niet meer wanneer er meerdere verschillende soorten conflicten in het systeem aanwezig zijn. Met oog voor de toekomst ben ik gaan kijken naar een oplossing die schaalbaar is.

In de toekomst zal het aanmaken van conflicten samen moeten werken met een catalogus, omdat er voor het oplossen van een specifiek conflict moet worden betaald. Dit is vergelijkbaar met een webwinkel. De gebruiker bladert in de catalogus, kiest het conflict wat hij wil oplossen, en rekent dit af. Bij een webwinkel krijgt de gebruiker vervolgens het product toegestuurd, in onze situatie wordt het product (de conflictoplossing) aan zijn account toegevoegd.

ConflictTypeCatalogItem	
PK	**ConflictTypeCatalogItemId**
U1	**Name** **Description** **ConflictType**

Er moet dus een methode zijn om de verschillende conflicttypen te koppelen aan deze catalogus. Hiervoor heb ik elk type conflict een unieke naam gegeven, waarmee deze geïdentificeerd kan worden. Deze naam is in het catalogusitem opgenomen, naast andere zaken zoals de naam van het type conflict zoals deze in de catalogus zichtbaar is, een korte beschrijving, etc. Wanneer de gebruiker een conflicttype in de catalogus heeft uitgekozen, kan het type binnen het systeem worden geïdentificeerd en worden aangemaakt.

Voor het aanmaken van een instantie van de specifieke conflictoplossing heb ik gebruik gemaakt van een combinatie van Design Patterns.

Voor het in elkaar zetten van het conflict heb ik een shortlist gemaakt van twee Design Patterns, het Abstract Factory-pattern en het Builder-pattern. Deze twee patterns lijken erg op elkaar en hebben hetzelfde doel, namelijk een instantie van een object maken. Het grootste verschil is dat het Builder-pattern dit in meerdere stappen doet, en een Abstract Factory in één stap.

Een Abstract Factory definieert een methode, zoals `CreateConflict` in mijn situatie zou heten, en deze bouwt het hele conflict op. Alles wat hierbij nodig is, wordt via deze methode als parameters meegegeven. Het Builder-pattern doet dit in stapjes. Eerst worden alle benodigdheden aan de Builder gegeven door middel van setters, en vervolgens wordt op het moment dat de Build-methode wordt aangeroepen, wordt het object opgebouwd.

In mijn geval moeten er bij het opbouwen van een conflict meerdere dingen gebeuren. Zo moet de eigenaar van het conflict worden gekoppeld, er moeten een aantal rollen worden gekoppeld, zoals *Owner*, etc. Wanneer dit allemaal aan de CreateConflict-methode meegegeven moet worden,

wordt dit erg onoverzichtelijk en rommelig. Door de verschillende stappen is het Builder-pattern veel makkelijker te volgen.

In het boek Clean Code van Robert C. Martin[1] wordt er veel ingegaan op de nadelen van meerdere parameters op een functie. Onderstaand is een quote die de problemen met meerdere parameters kort duidelijk maakt.

"The ideal number of arguments for a function is zero (niladic). Next comes one (monadic), followed closely by two (dyadic). Three arguments (triadic) should be avoided where possible. More than three (polyadic) requires very special justification - and then shouldn't be used anyway.

A function with two arguments is harder to understand than a monadic function. For example, `writeField(name)` *is easier to understand than* `writeField(output-Stream, name)`*. Though the meaning of both is clear, the first glides past the eye, easily depositing its meaning. The second requires a short pause until we learn to ignore the first parameter. And that, of course, eventually results in problems because we should never ignore any part of code. The parts we ignore are where the bugs will hide.*

Functions that take three arguments are significantly harder to understand than dyads. The issues of ordering, pausing, and ignoring are more than doubled. I suggest you think very carefully before creating a triad."

Elk type conflict wordt op een andere manier opgebouwd, waardoor er voor elk type conflict een eigen Builder nodig is. Om deze specifieke builder te initialiseren maak ik gebruik van het Factory-pattern. Dit pattern weet welke Builder er bij welk type conflictoplossing hoort. De Factory heeft een Create-methode waaraan het type conflictoplossing wordt meegegeven. Hij kijkt vervolgens welk type Builder bij dit type conflictoplossing hoort, en geeft hier een instantie van terug.

Door deze methode te gebruiken, hoeft er binnen de code geen gebruik gemaakt te worden van het new-keyword, maar wordt de factory gebruikt om een instantie te maken. Dit verwijdert de koppeling tussen de code en de verschillende instanties van de builders.

Binnen de builders zijn er echter een aantal gelijkenissen bij het aanmaken van een conflictoplossing. Zo moet bij elke conflictoplossing de persoon die het conflict aanmaakt worden toegevoegd en moet deze rollen toegewezen krijgen. Omdat dit in elke builder moet gebeuren is het wenselijk om hier een abstractie boven te maken, die dit doet.

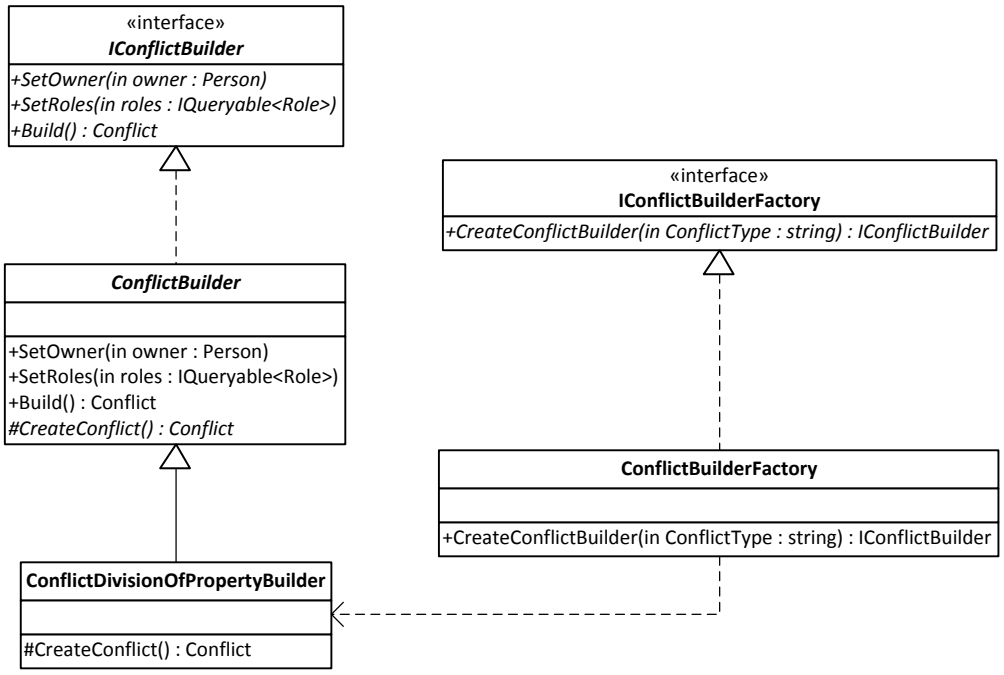
Hiervoor heb ik gebruik gemaakt van het Factory Method-pattern binnen de Builder. Dit pattern definieert een abstracte methode die gebruikt wordt om een concrete instantie te maken van een object. De subklassen implementeren vervolgens deze methode.

In de subklassen wordt dus de concrete instantie van de conflictoplossing aangemaakt, dus bijvoorbeeld `ConflictDivisionOfProperty` in mijn situatie, en deze wordt teruggegeven. Op deze manier kan de Builder het aanmaken van een specifieke instantie van `Conflict` uitbesteden aan zijn

[1] *Clean Code*, Robert C. Martin, Prentice Hall, 2008

subklassen, maar de algemene handelingen, zoals het toewijzen van de eigenaar, wel zelf uitvoeren. Hierdoor treedt er geen duplicate code op.

In UML ziet deze structuur er als volgt uit. Enkel de relevante onderdelen voor de context van het probleem zijn in dit diagram opgenomen om het overzicht te bewaren.



10.4 Koppelen van betrokkenen

Op dit moment is het mogelijk om een conflictoplossing aan te maken, alleen is er op dit moment nog maar één persoon bij betrokken, namelijk de persoon die de conflictoplossing aangemaakt heeft. Omdat bij een conflict altijd meer dan één persoon betrokken is, moet het mogelijk zijn om anderen uit te nodigen om bij het conflict betrokken te zijn.

Hierbij is er een belangrijke eis. Er moet namelijk rekening gehouden worden met het aantal personen die uitgenodigd kunnen worden. Wanneer het type conflictoplossing waarmee er we te maken hebben iets is als een familieprobleem zit er niet direct een limiet op het aantal personen die betrokken zijn binnen het conflict.

Wanneer we echter met een conflict als een scheiding te maken hebben, kan de conflictoplossing maar twee betrokkenen hebben, gezien een scheiding tussen twee personen verloopt. Wanneer er een betrokkene uitgenodigd gaat worden, moet er dus rekening gehouden worden of er nog een persoon kan worden uitgenodigd.

Voor het uitnodigen van personen zijn verschillende mogelijkheden. Zo kunnen de gebruikers door een lijst bladeren met personen die geregistreerd zijn binnen het systeem, zoals dit bijvoorbeeld ook bij Facebook kan, of kan er worden gezocht op gebruikersnaam.

Geen van deze twee methoden vonden mijn begeleider en ik acceptabel, omdat dit doorzichtig maakt wie er binnen het systeem geregistreerd zijn. Bij Facebook is het handig dat je vrienden kan zoeken, omdat het doel hiervan is om connecties tussen personen te leggen, en is positief van aard.

Wanneer een persoon binnen ons systeem geregistreerd is, betekend dit dat deze persoon bij een conflict betrokken is geweest, en dit is negatief. Daarom hebben we ervoor gekozen om het niet inzichtelijk te maken welke personen er binnen het systeem geregistreerd zijn, om de privacy van deze personen te bewaren.

We moesten dus een andere opzet verzinnen die niet doorzichtig is met betrekking tot de gebruikers in het systeem, en zijn hierbij uitgekomen op een uitnodiging per email. Wanneer een gebruiker een persoon wil uitnodigen, vult hij bij de uitnodiging het emailadres in van diegene die hij wil uitnodigen, en deze krijgt vervolgens een mail binnen met een uitnodiging.

Door deze manier te gebruiken worden er twee problemen opgelost. Doordat er een emailadres wordt gebruikt en de uitnodiging per email wordt gestuurd, is het niet doorzichtig of de persoon al in het systeem aanwezig is, omdat de persoon een bericht krijgt en niet geïdentificeerd wordt.

Een ander probleem wat opgelost wordt, is dat gebruikers personen kunnen uitnodigen die nog niet in het systeem beschikbaar zijn. Wanneer er een specifieke persoon gekozen moet worden, moet deze al wel in het systeem staan. Bij het sturen van een email met een link waarmee de uitnodiging kan worden geaccepteerd is dit geen vereiste, omdat de gebruiker zich kan registreren voordat hij de uitnodiging accepteert.

Door deze argumenten leek een uitnodiging middels een emailbericht de beste optie. Deze email bevat een link die de gebruiker moet bezoeken. In de link is de ID van de uitnodiging aanwezig, waardoor het systeem weet aan welke conflictoplossing de gebruiker gekoppeld moet worden.

Enkel een link met een ID is echter niet heel erg veilig, omdat de ID een oplopend nummer is. Als de gebruiker dat patroon doorheeft, zou hij zichzelf ook bij andere conflictoplossingen aan kunnen melden zonder dat dit de bedoeling is. Daarom moet er nog een andere vorm van beveiliging bij komen.

Dit heb ik gedaan door middel van een token, in de vorm van een hash welke random gegenereerd wordt. Alleen als de token in de link in het emailbericht overeen komt met de token opgeslagen bij de uitnodiging, wordt de uitnodiging “geaccepteerd”. Door het vele aantal mogelijkheden die de hash kan aannemen, is het gokken van de juiste hash praktisch onmogelijk.

Nadat de gebruiker de uitnodiging geaccepteerd heeft, wordt de uitnodiging voor het conflict verwijderd, waardoor deze maar door één persoon gebruikt kan worden. Elke uitnodiging is uniek, dus als er nog een ander persoon uitgenodigd dient te worden, krijgt deze een eigen uitnodiging.

11 De oplevering van het product

Nu ik aan het einde van mijn afstudeerperiode zit, moet het product worden opgeleverd. Deze oplevering is niet naar een klant, maar naar Cenosco zelf. Omdat het product nog niet is afgerond, moet dit in een vorm gebeuren waar Cenosco het meeste aan heeft.

Daarom heb ik ervoor gekozen om deze release op een informele manier te doen, via het version control system. Wanneer een medewerker binnen Cenosco, of een mogelijke toekomstige stagiair verder gaat werken aan het product, moet dit zo snel mogelijk kunnen gebeuren.

Wanneer ik een release zou doen op een cd of dvd is er een fysieke release, echter levert dit voor de toekomstige ontwikkelaar enkele problemen op. Wanneer hij verder gaat werken aan de code, moet hij de broncode van de schijf naar zijn computer kopiëren, en deze opnemen in een version control system. De hele geschiedenis van het product is nu niet meer beschikbaar, omdat de ontwikkelaar met een schone lei begint. Ook kost het opzetten van de version control tijd.

Wanneer de release echter wordt gedaan bínnen het version control system, hoeft de ontwikkelaar enkel de code uit te checken, en heeft hij de laatste versie ter beschikking. De geschiedenis van mijn werk is voor de ontwikkelaar beschikbaar, en ook hoeft hij de version control niet meer op te zetten.

Met oog op de toekomst, en hoe er verder wordt gewerkt aan het product, leek deze vorm van releasen mij hier het beste bij passen. In het version control system is aangegeven door middel van een tag dat deze versie de release is waardoor het later makkelijk kan worden teruggevonden, de ontwikkelaar kan makkelijk verder waar ik gebleven ben, en binnen Cenosco staat alles op één plek. Deze manier van opleveren leek mij hierdoor het beste.

Naast het opleveren van de broncode heb ik twee andere onderdelen opgeleverd, ook via het version control system. Ten eerste heb ik de ontwerpdocumentatie die ik tijdens mijn afstudeertraject heb opgesteld in de release opgenomen. Dit kan voor de ontwikkelaar handig zijn om de structuur van het systeem te doorgronden, zonder dat er in de code of de database moet worden gedoken.

Ten tweede heb ik de huidige versies van de libraries die ik gebruikt heb opgenomen in de release. Binnen Visual Studio is een package manager beschikbaar, waarmee de laatste versies van de libraries automatisch gedownload kunnen worden.

Het gevaar zit hem hier in het automatisch downloaden van de *laatste* versie. Als er een nieuwe ontwikkelaar verder gaat en een nieuwe versie van de library download, kan het zijn dat de interface is aangepast. Het project kan dan niet meer gecompileerd worden omdat de functieaanroepen vanuit de broncode naar de library niet overeen komen met de functies die de library blootstelt.

In dit geval kan het erg wenselijk zijn om de libraries waarmee het project wel compileert beschikbaar te hebben, zodat het project kan worden gecompileerd en uitgevoerd. Als alles werkt kan het project worden gemigreerd om de nieuwe versie van de library te ondersteunen.

12 Unittesten

Binnen mijn project heb ik gebruik gemaakt van Test Driven Development. Dit is een techniek waarbij de tests voor een component eerst worden geschreven, en daarna pas de daadwerkelijke implementatie. Dit zorgt voor een aantal voordelen.

Het grootste voordeel is dat doordat de tests eerst worden geschreven, en daarna pas de code, dat de code testbaar is. Wanneer eerst de code en later pas de tests worden geschreven, zijn er meestal teveel afhankelijkheden om goed te unit testen. Zo kan bijvoorbeeld de implementatie van de database sterk zijn gekoppeld waardoor er enkel getest kan worden waarbij er ook daadwerkelijk naar de database wordt geschreven. Dit is geen unit test.

Met Test Driven Development worden de tests eerst geschreven. Omdat er letterlijk geen test geschreven kan worden als er harde koppelingen liggen, de implementaties kunnen immers niet vervangen worden door stubs en mocks, wordt er bij het schrijven van de tests al rekening gehouden dat de code testbaar moet zijn.

Doordat de tests eerst worden opgesteld, zijn deze er dus van bijna alle onderdelen aanwezig, en is de code coverage van de tests ook erg hoog, in de negentig procent. Ik heb er bewust voor gekozen om een aantal onderdelen niet te testen, waardoor de code coverage niet tegen de honderd procent aanzit.

Ten eerste heb ik door Test Driven Development enkel unit tests gemaakt. Een unit test is een test die zich richt op een enkele unit, waarbij een unit het kleinst mogelijke testbare onderdeel is. De code die getest wordt hoort hierbij geen afhankelijkheden te hebben op andere componenten.

Een ander type test is een integratietest. Bij dit type test worden de componenten met elkaar verbonden, en worden als een geheel getest. Zo kan er getest worden of de verschillende componenten samen werken zoals verwacht.

Ik heb ervoor gekozen om geen integratietests op te stellen. Doordat de componenten individueel grondig worden getest door de unit tests, is de kans zeer klein dat ze gecombineerd niet werken, terwijl de individuele componenten wel werken. Het opstellen van integratietests is veel werk, en daarom heb ik een afweging gemaakt of de tijd die het kost om de tests op te stellen afweegt tegen het risico dat de componenten niet juist samen werken. Het risico vond ik te laag om de tests te rechtvaardigen. Wanneer de componenten niet samen blijken te werken, komt dit bij het praktijktesten van het systeem aan het licht.

Onder integratietests vallen een aantal dingen. Ten eerste is er in het ASP.NET MVC-project een hoop initialisatiecode aanwezig. Deze code initialiseert het daadwerkelijke webproject, registreert onderdelen bij de IoC-container, etc. Deze code is onderdeel van de opstart van ASP.NET MVC, en niet van de functionaliteit die ik opbouw.

Ten tweede heb ik enkele implementaties van algoritmen gedaan, waaronder SHA512 voor hashing en AES voor encryptie. Deze algoritmen zijn al onderdeel van het .NET framework, en mijn implementatie is een adapter tussen de interfaces die ik binnen mijn project definieer, en de implementaties binnen het framework van Microsoft. Niet alleen zouden ook deze tests

integratietests worden, maar ook enkel code testen die Microsoft heeft opgeleverd. Ik heb hier de aanname gedaan dat de code van Microsoft al goed getest was.

Ten derde heb ik een project, waarbinnen de mappings naar de database worden geregeld. Ook dit project zou enkel de implementatie testen van het ORM-framework en zouden integratietests worden. Daarom heb ik ook voor dit project geen unit tests opgesteld.

Ten slotte heb ik in de views, welke de vorm van webpagina's hebben, gebruik gemaakt van Javascript. Dit wordt veel gebruikt om de pagina dynamisch aan te passen waardoor deze niet opnieuw hoeft te worden opgehaald. Deze scripts hebben sterke afhankelijkheid van de browser, en zijn hierdoor ook slecht individueel te testen.

Het tweede onderdeel wat ik bewust niet heb getest zijn een aantal methoden binnen de controllers. ASP.NET MVC verplicht dat voor elke pagina die de gebruiker kan bekijken er een methode binnen de controller is. Een methode die enkel een pagina zonder afhankelijkheden naar de database, etc. heeft, ziet er als volgt uit.

```
public ActionResult Index()
{
    return View();
}
```

Deze code doet letterlijk niets, behalve aangeven dat er een pagina moet worden getoond. Een test hiervoor zou makkelijk op te stellen zijn, maar helemaal niets testen. Daarom heb ik voor dit soort methoden zonder afhankelijkheden of logica geen tests opgesteld.

Een overzicht van de tests die ik heb opgesteld en de bijbehorende resultaten heb ik in bijlage C opgenomen. Ook is hier een overzicht van de code coverage te zien.

13 Projectmanagement met Scrum

Binnen dit project heb ik gebruik gemaakt van Scrum. Ik ben hier echter op twee momenten van afgeweken.

In de eerste sprint waarin ik daadwerkelijk ging ontwikkelen, sprint 1, moest ik heel veel uitzoeken over de werking van de manier van inloggen. Omdat ik nog geen ervaring had met deze techniek, kon ik geen enkele inschatting maken over de tijd die ik hieraan zou gaan besteden. Hoe complex en uitgebreid het is wordt immers pas duidelijk tijdens dit uitzoeken. Het proberen te plannen van iets waarvan de grootte niet bekend is, is gedoemd te falen.

In dit geval kon ik een schatting van een groot aantal units maken waarbij de kans heel erg klein is dat deze in de buurt van de daadwerkelijke grootte van het onderdeel zou komen, óf ik kon dit onderdeel niet volgens Scrum doen, waarbij ik het uitzoeken zonder planning zou gaan doen en wanneer er duidelijkheid was over de tijd en complexiteit die het implementeren zou gaan kosten overschakelen naar Scrum. Ik heb voor dit laatste gekozen.

In de tweede sprint heb ik, zoals ik al geschreven heb in paragraaf 7.2, gebruik gemaakt van een aantal kleine iteraties van een paar dagen om kleine prototypes van de productenlijst uit te werken. Ook al werd er op dit moment gewerkt met iteraties en contact met mijn begeleider, heb ik deze prototypes niet in units geschat, en heb ik geen gebruik gemaakt van een vaste iteratielengte. De argumenten hiervoor heb ik reeds genoemd in bovengenoemde paragraaf.

De backlog met requirements en bijbehorende units heb ik in de bijlagen opgenomen als bijlage D. Ook zijn hier de burndown charts van de sprints te zien.

14 Evaluatie

14.1 Procesevaluatie

Het proces is al met al goed verlopen, ondanks dat ik in het begin een grote tegenslag had. Ik wilde het inloggen van gebruikers opzetten via een externe dienst, waardoor deze konden inloggen met bijvoorbeeld hun Googleaccount of hun Facebook. Na twee weken uitzoeken, liep ik nog steeds tegen veel problemen aan, en kreeg ik sterk het idee dat ik beter een andere manier van inloggen kon gaan gebruiken, omdat ik bang was dat ik anders nog veel langer vast zou komen te zitten en hierdoor in een serieuze tijdsnood zou komen.

Achteraf gezien heb ik hiermee de juiste keuze gemaakt. De andere manier van inloggen kon ik, samen met alle overige randzaken die hiervoor nodig waren zoals het opzetten van databaseverbindingen, etc. in een week implementeren. Het inbouwen van inloggen heeft hierin niet meer dan twee dagen gekost. Hierbij heb ik er wel in mijn implementatie rekening gehouden met het feit dat er eventueel op een later moment voor een andere manier van inloggen gekozen gaat worden. Hierdoor kon ik nu voor de traditionele gebruikersnaam-wachtwoordcombinatie gaan, zonder dat ik hiermee andere implementaties uitsloot.

Als ik terugkijk naar de keuze om deze manier van inloggen te gebruiken, moet ik concluderen dat ik deze keuze beter niet had kunnen maken. Door gebruik te maken van een onbekend component was er geen zekerheid in de haalbaarheid en was dit ook moeilijk te plannen. Ik ben hierin te overmoedig geweest. De keuze om deze manier van inloggen te proberen ben ik dan ook op de ergste manier mogelijk tegengekomen, namelijk het implementeren van het component is niet gelukt. Als ik de keuze wederom zou moeten maken zou ik eieren voor mijn geld kiezen en een methode gebruiken waarbij zekerheid aanwezig is. Dit zou in mijn project de manier van inloggen met gebruikersnaam en wachtwoord zijn.

Bij het daadwerkelijk bouwen van de applicatie heeft hij me erg vrij gelaten. Hij heeft mij geen keuzes opgelegd, maar enkel af en toe tips gegeven. Zo gaf hij aan dat er de javascript-library KnockoutJS was, die erg goed paste bij de dingen die ik probeerde te doen. Het is echter nooit meer dan een advies geweest, waardoor hij het daadwerkelijke implementeren aan mij heeft overgelaten.

Binnen mijn project waren de opdrachtgever en begeleider dezelfde persoon. Hierdoor is er veel overlap geweest tussen de opdrachtgever- en de begeleiderrol. Wanneer ik een onderdeel aan de opdrachtgever opleverde, kon hij eventueel in zijn begeleiderrol springen om mij feedback te geven. Dit korte schakelen is binnen mijn project erg prettig geweest, omdat ik niet met twee personen hoefde te overleggen. Hierdoor heb ik ook het probleem dat de begeleider altijd achter loopt met betrekking tot de projectverloop en elke keer hierover geïnformeerd moet worden niet gehad. Ook is er geen belangenverstrengeling tussen de opdrachtgever en de begeleider.

De opdrachtgever was bij dit project een van de eigenaars van Cenosco, en hierdoor had hij al veel kennis en inzicht in de gebruikte technieken. Hierdoor kon hij mij binnen het project een aantal waardevolle tips geven. Omdat ik niet alle technieken die de opdrachtgever had aanbevolen kende, heb ik hier eerst kritisch naar gekeken. Zoals ik bijvoorbeeld al in paragraaf 5.3 schreef, had ik nog geen ervaring met de cloud, en heb ik hier eerst informatie over verzameld, en heb ik gekeken of dit

binnen het project bruikbaar was. Met uitzondering van het inloggen wat ik hierboven beschreven heb, zijn eigenlijk alle tips van de opdrachtgever bruikbaar geweest.

Scrum bleek voor dit project een perfecte keuze te zijn. Omdat het project in het begin slechts een niet-uitgespecifieerd idee was, was er toen niet veel duidelijkheid wat het product allemaal moest kunnen, enzovoorts. Ik heb aan het begin requirements opgesteld, maar hier zijn eigenlijk elke sprint wel requirements bijgekomen, en is er flink met de prioriteiten geschoven.

Doordat er met Scrum gewerkt is, kon dit zonder veel problemen, omdat deze projectmethode erop gebouwd is dat de sprints flexibel zijn in punten. Vooraf wordt er geen planning gemaakt wat er in een sprint opgenomen gaat worden, maar dit wordt aan het begin van de sprint gedaan. Hierdoor konden elke keer de punten met de hoogste prioriteit worden bepaald en worden ingedeeld in de sprint.

Ik heb veel contact gehad met mijn begeleider, en hier is veel feedback uit voortgekomen. Ook dit sloot goed aan bij Scrum, omdat naar aanleiding van deze gesprekken er nieuwe requirements boven de tafel kwamen, en punten die nog niet naar wens waren geïmplementeerd konden worden geïdentificeerd. Hierdoor zijn de onderdelen die ik opgeleverd heb in lijn met de verwachtingen van mijn begeleider.

Een bijproduct van het schuiven met de requirements zoals dat is gebeurd, is dat niet alles wat ik in eerste instantie gepland had om te doen afgekomen. Voor deze onderdelen zijn echter andere onderdelen in de plaats gekomen, die tijdens het project een hogere prioriteit bleken te hebben. Dat deze onderdelen niet geïmplementeerd zijn is hierdoor niet slecht, maar zelfs goed, omdat er anders wel onderdelen geïmplementeerd zouden zijn die een minder hoge prioriteit hadden, en de onderdelen met een hogere prioriteit zouden zijn komen te vervallen. Deze manier van werken is voor de opdrachtgever het meest gunstig geweest.

Als ik kijk naar de rol die de opdrachtgever binnen het project heeft gespeeld, vind ik deze goed. Hij had een aantal wensen, zoals het hosten van het project in de Windows Azure-cloud, maar gezien het feit dat hij de opdrachtgever is, en kennis heeft in dit gebied staat hij hiermee goed in zijn recht.

Wel kon ik tijdens mijn afstudeerstage merken dat ik nog niet heel erg veel ervaring heb met plannen. Een aantal dingen had ik onderschat, of ik had geen rekening gehouden met benodigde gerelateerde onderdelen die veel tijd vergden. Dit is vooral te zien in de burndown chart van sprint 3. Door het zwaar onderschatten van de benodigde functionaliteiten heb ik hier veel minder units kunnen afronden dan in eerste instantie gepland. Deze sprint gaf mij echter wel veel inzicht in de complexiteit van de nog te implementeren requirements en hierdoor is het plannen van de overige sprints beter verlopen. Dit liet mij wel zien dat praktijkervaring erg belangrijk is bij het maken van een planning, doordat de schattingen hier erg vanaf hangen. Door meer praktijkervaring op te doen zullen mijn planningen in de toekomst ook steeds beter worden.

De lengte twee weken per sprint is mij goed bevallen. De sprints zouden zeker niet langer moeten zijn geweest. Een andere mogelijkheid was een sprint van één week. Ik heb wekelijks een gesprek met mijn opdrachtgever gehad, waarbij nu één gesprek over de oplevering ging, en het andere gesprek over de tussentijdse voortgang. Wanneer ik in sprints van één week had gewerkt was dit goed samen gegaan met deze gesprekken.

Aan de andere kant heb ik ook veel gerelateerde werkzaamheden kunnen groeperen in de sprints van twee weken. Wanneer ik in sprints van één week zou werken, zouden deze over twee sprints worden verdeeld. Hierdoor worden niet alle onderdelen opgeleverd die eigenlijk wel samen horen. Dit is niet per definitie fout, maar gerelateerde onderdelen samen opleveren is netter.

Ik vind dat zowel een lengte van één week als twee weken in mijn project mogelijk waren. De lengte van twee weken is me goed bevallen, en sta ook nog steeds achter deze keuze.

Nu aan het einde van mijn afstudeerperiode kan ik tevreden terugkijken op het proces, en vind ik dat dit succesvol is verlopen.

14.2 Productevaluatie

Ik kan achteraf zeggen dat ik blij ben met het eindresultaat wat ik heb behaald. Het product is mooi geworden, en er zit redelijk wat complexiteit in.

Ondanks dat in het begin niet helemaal duidelijk was wat het resultaat van het product zou zijn, heb ik toch een afgebakend deel kunnen opleveren. Naarmate het project verder vorderde was dit steeds verder te specificeren samen met mijn begeleider. Hierdoor heb ik, ondanks dat er van tevoren al gespecificeerd was dat ik het gehele product niet af zou krijgen, toch het idee dat ik iets afgeronds heb opgeleverd.

Door dingen als beveiliging, het aanmaken van conflictoplossingen en het uitnodigen van betrokkenen, etc. op een abstract niveau heb opgezet, is het eindproduct hierdoor goed uitbreidbaar geworden. Wanneer er een nieuwe vorm van conflictoplossing moet worden opgelost, kunnen deze onderdelen worden gebruikt hoeven deze niet gekopieerd of opnieuw geschreven te worden. Bij het toevoegen van een nieuwe vorm van conflictoplossing hoeft er hierdoor enkel naar de daadwerkelijke implementatie hiervan te worden gekeken, en niet naar alle randzaken.

Binnen het project heb ik gebruik gemaakt van Test Driven Development. Voordat er een stuk functionaliteit wordt gemaakt, worden eerst de tests hiervoor geschreven, en daarna pas de implementatie. Het resultaat hiervan is dat er van bijna alle onderdelen tests aanwezig zijn, en de code coverage die de unit tests dekken erg hoog is.

Aan het gebruiken van Test Driven Development hangt wel een nadeel. Omdat er van elk onderdeel tests worden opgesteld, kost dit meer tijd. De tijd die ik aan het schrijven van tests en het schrijven van functionaliteit heb besteed is ongeveer 40/60 geweest. Het schrijven van tests is dus een aanzienlijk deel van het project geweest.

Wanneer ik geen gebruik gemaakt had van TDD en enkel de belangrijkste dingen getest had, was er meer tijd over geweest voor daadwerkelijke ontwikkelingen aan het product. Deze manier van werken heeft er echter wel voor gezorgd dat ik met zekerheid kon zeggen dat een stuk functionaliteit werkte.

Deze manier van werken is een afweging tussen kwaliteit en kwantiteit geweest. Hierbij heb ik voor kwaliteit gekozen, om twee redenen. Ten eerste was er geen harde deadline aanwezig wanneer het product opgeleverd moest worden. Hierdoor was er geen tijdsdruk aanwezig waarbinnen ik de functionaliteit moest opleveren, maar kon ik veel tijd in het testen steken.

De tweede en belangrijkste reden, is dat ik (nog) een student ben, en veel praktijkervaring mis. Hierdoor is de kans dat ik fouten die als beginnersfouten kunnen worden gekenmerkt groot. Wanneer er veel tests aanwezig zijn, geven deze een goede controle waardoor vele van dit soort fouten kunnen worden voorkomen, of tot een minimum kunnen worden beperkt.

Deze manier van testen heeft mij meer vertrouwen gegeven in het resultaat wat ik heb opgeleverd. Hierdoor ben ik achteraf gezien blij dat ik dit heb gebruikt, omdat ik hierdoor beter kan garanderen dat ik kwaliteit opgeleverd heb, en met vertrouwen achter mijn eigen werk kan staan.

Binnen het product heb ik veel gebruik gemaakt van externe libraries. Voor de communicatie met de database heb ik gebruik gemaakt van het ORM-framework Entity Framework en voor de dependency injection heb ik Unity gebruikt. Ook heb ik aan de browserkant gebruik gemaakt van een aantal libraries, waaronder jQuery en KnockoutJS. Het gebruik van deze libraries is mij zeer goed bevallen. Het gebruik hiervan scheelt ten eerste tijd, omdat het niet gebouwd hoeft te worden maar omdat er een bestaand component wordt gebruikt. Deze libraries zijn van bekende spelers binnen de informaticawereld en deels ook Open Source, waardoor de kwaliteit kan worden gegarandeerd.

Ik had enkel met jQuery veel ervaring, en met Entity Framework een klein beetje. De andere libraries waren voor mij nieuw. Ik merkte dat Unity een goede keuze was als IoC-container, omdat hier veel over te vinden was op internet in combinatie met ASP.NET MVC. Dit heeft mij erg geholpen bij de implementatie. Ook was ik erg tevreden over de API, omdat deze perfect aan bleek te sluiten op mijn wensen.

Vooral over KnockoutJS was ik positief verrast. Deze library kende ik nog niet, maar het bleek een puzzelstukje te zijn wat bijna onmisbaar was binnen de applicatie die ik gebouwd heb. Het loskoppelen van de user interface en de achterliggende data gebeurde zo mooi, dat er bij het ontwikkelen eigenlijk geen aandacht besteed hoefde te worden aan de interactie tussen de klant en de browser, dit werd allemaal uit handen genomen door KnockoutJS. Hierdoor is de code heel erg versimpelt, en is de ontwikkeltijd ook drastisch afgenomen. Bij een volgend project zou ik zeker KnockoutJS weer overwegen. Dit kan ik ook voor de andere bovengenoemde libraries zeggen.

Binnen het product is er echter één ding waar ik gemengde gevoelens over heb. Bij de achterliggende administratie zijn er een aantal handelingen die erg op elkaar lijken, zo lijkt het toevoegen van een locatie erg op het toevoegen van een productcategorie. Ik heb geprobeerd hier een abstractie van te maken, alleen hebben al deze onderdelen eigen domeinmodellen binnen het systeem. Hierdoor is het niet gelukt om de functionaliteit te abstraheren.

Om de functionaliteit toch te bouwen, heb ik een aantal stukken code moeten kopiëren en heb ik daarin kleine stukken moeten aanpassen, wat veel duplicate code opgeleverd heeft. Het werkt wel, het is alleen niet heel erg netjes. Omdat dit in nadele is voor de netheid en onderhoudbaarheid van het systeem, vind ik het jammer dat het niet gelukt is om deze functionaliteiten te generaliseren.

14.3 Evaluatie van de verrichtte competenties

Als ik kijk naar de competenties die ik bij het indienen van mijn afstudeeropdracht heb gedefinieerd, zie ik dat ik al deze competenties behandeld heb binnen mijn afstudeerstage.

Ten eerste heb ik de competentie *uitvoeren analyse door definitie van requirements* behaald door aan het begin van mijn opdracht een interview met mijn begeleider te houden, waardoor ik meer

informatie over de opdracht gekregen heb. Met deze informatie kon ik een lijst met requirements opstellen. Deze lijst is tijdens de opdracht verder uitgebreid doordat er meer over de opdracht bekend werd.

De requirements heb ik opgesteld met de intentie dat deze niet enkel voor mijn opdracht te gebruiken waren. In het begin was duidelijk dat het gehele project groter zou zijn dan enkel mijn afstudeerstage. Ik kon de requirements dan opstellen zodat ze enkel voor mijn project te gebruiken waren, maar dit zou meer werk opleveren voor Cenosco wanneer er verder aan het project gewerkt zou gaan worden. Hierom heb ik de requirements ruim opgesteld, waardoor deze niet alleen de eisen die aan mijn afstudeeropdracht gingen bevatte, maar ook overige requirements die al aan het begin duidelijk waren.

De database die bij deze applicatie aanwezig is, is redelijk complex. Voorafgaand moest deze ontworpen worden, en tijdens het project is deze steeds verder geëvalueerd tot de database die het nu is. In deze database zijn natuurlijk ook gegevens opgeslagen, zoals klantgegevens, gegevens voor de conflictoplossingen, etc. Hiermee heb ik de competenties *opstellen gegevensmodel voor een database* en *ontwerpen, bouwen en bevragen van een database* vervuld.

De applicatie zelf moest natuurlijk ook worden ontwikkeld. Bij het ontwerpen van de applicatie moest er rekening gehouden worden met het feit dat deez uitbreidbaar moest zijn, zodat er later zonder veel bijkomende randzaken conflictoplossingen konden worden toegevoegd. Bij het ontwerpen en ontwikkelen van de applicatie zijn er enkele design patterns gebruikt, zoals het Adapter pattern, het Factory Method pattern en het Repository pattern. Ook heb ik gebruik gemaakt van Dependency Injection. Al deze onderdelen zorgen ervoor dat de applicatie uitbreidbaar en testbaar is. Hier komen de competenties *ontwerpen systeemdeel* en *bouwen applicatie* terug.

Tot slot heb ik veel aandacht besteed aan het opzetten van unit tests. Deze heb ik volgens de Test Driven Development-techniek opgezet, waardoor er van bijna alle onderdelen tests beschikbaar zijn, en de code coverage erg hoog ligt. Uit deze tests volgt automatisch een rapportage die de slagende tests en de code coverage die uit de unit tests volgt weergeeft. Bij het opstellen van de tests kwam de competentie *uitvoeren van, en rapporteren over het testproces* terug.

Bijlage A: Plan van aanpak

Inleiding

In dit plan zal beschreven worden wat de strategie wordt om tot een succesvolle afronding van het afstudeerproject te komen. Het plan is opgedeeld in de volgende hoofdstukken:

Opdrachtomschrijving

In dit hoofdstuk wordt kort uitgelegd hoe het project tot stand is gekomen, en wat het doel van het project is.

Afbakening project

In dit hoofdstuk worden de grenzen van het project afgebakend.

Activiteiten

In dit hoofdstuk wordt beschreven welke activiteiten er tijdens het afstuderen uitgevoerd gaan worden.

Methoden en technieken

In dit hoofdstuk wordt beschreven welke methoden en technieken er tijdens het project gebruikt gaan worden.

Planning

In dit hoofdstuk is er een globale planning voor de looptijd van het afstudeertraject opgesteld.

Opdrachtomschrijving

Wanneer er een conflict optreedt, en beide partijen hier samen niet uit kunnen komen, kan het oplossen van dit conflict in hoge kosten oplopen.

Een voorbeeld hierbij is een echtscheiding. Als beide partijen niet uit het verdelen van de boedel komen, komt hier vaak een advocaat aan te pas. Deze advocaat vervult een soort van "scheidsrechterrol", waarbij zijn doel is dat de boedel goed verdeeld wordt. Voor €150-200 per uur kan dit een flinke kostenpost worden.

Cenosco ziet een mogelijkheid om conflicten zonder juridische kosten op te lossen via een ict-oplossing. Deze applicatie kan verschillende conflicten aan en biedt algoritmes om ze op te lossen. Er kunnen verschillende profielen worden gebruikt zoals conflicten binnen een echtscheiding of een levering. Het doel van de webapplicatie is om een eerlijke scheidsrechter te zijn.

De applicatie zal gebouwd worden in de vorm van een commerciële webapplicatie. Voor een klein bedrag kan de gebruiker de applicatie gebruiken om een conflict op te lossen. Financieel is dit voor de klant een stuk voordeliger gezien de kosten voor het oplossen van een conflict via de website vele malen voordeliger is, dan dit conflict oplossen langs een juridische weg. De gebruikskosten van de applicatie zullen voor inkomsten voor Cenosco zorgen.

Afbakening project

Binnen het project wordt er gefocust op een applicatie waarbinnen conflicten kunnen worden opgelost. Dit is op te splitsen in twee onderdelen:

- Een platform waarbinnen conflicten opgelost kunnen worden
- Profielen om conflicten mee op te lossen

Mijn focus binnen het project ligt op het platform. Binnen dit platform moeten er een aantal dingen mogelijk zijn die voor elk type conflict van toepassing zijn. Ik ga me onder andere bezig houden met het inloggen en registreren van gebruikers en het verrichten van betalingen. Ook ga ik kijken naar beveiligde verbindingen.

Binnen het onderdeel conflicten oplossen is mijn rol klein. De mogelijkheden die nodig zijn om dit te doen moeten in het platform aanwezig zijn, maar in een abstracte vorm, zodat ze voor alle mogelijke conflicten bruikbaar zijn.

Bij het implementeren van de conflicten is mijn rol om de algoritmes die gebruikt kunnen worden om deze conflicten op te lossen te vertalen naar code. De algoritmes die hiervoor worden gebruikt worden aangeleverd door Cenosco.

Activiteiten

Tijdens het afstudeerproject worden er een aantal activiteiten doorlopen. Deze worden hieronder beschreven.

Requirements verzamelen

Om een goed beeld te krijgen wat de applicatie precies moet gaan doen, worden er requirements opgesteld. Dit zal gedaan worden aan de hand van een interview wat er met de opdrachtgever wordt gehouden. De requirements zijn in te delen in twee categorieën:

- Functionele requirements
- Technische requirements

Functionele requirements geven aan wat de applicatie moet doen, zoals “een gebruiker moet een account kunnen aanmaken”. Technische requirements geven aan hoe de applicatie dit gaat doen, zoals “de applicatie zal in de cloud draaien”.

De requirements zullen in kaart gebracht worden door middel van use cases en user stories. Een aantal requirements zullen erg simpel zijn, zoals “een gebruiker moet kunnen inloggen”. Dit soort requirements laten zich hierdoor beter vastleggen in enkele user stories dan in een use case.

Bij een requirement als “een gebruiker moet een account kunnen aanmaken” komen meer dingen kijken. Er moet bijvoorbeeld worden gecontroleerd of er al een account is met de inlognaam. Requirements die dit soort uitzonderingssituaties hebben, of op een andere manier complex zijn, zullen door middel van use cases vastlegt worden omdat deze meer informatie kunnen geven.

De requirements zullen worden ingedeeld door middel van de MoSCoW-methode. Op basis van de categorieën waarin de requirements terecht komen, kunnen deze in verdere prioriteiten worden ingedeeld.

Ontwerpen

Op basis van de requirements kunnen er ontwerpen voor het project worden gemaakt. Er zullen drie soorten ontwerpen worden gemaakt:

- Softwareontwerp
- Databaseontwerp
- Grafisch ontwerp (GUI)

Voor het softwareontwerp zal gebruik gemaakt worden van UML-diagrammen. Voor het databaseontwerp van klassediagrammen en ERDs.

Ontwikkelen

Op basis van het ontwerp wat in de vorige stap is gemaakt, kan het systeem worden ontwikkeld. Voorafgaand aan de opdracht is vastgesteld dat dit een ASP.NET mvc-website gaat worden, in combinatie met de taal c#.

Testen

Tijdens het ontwikkelen van de applicatie zal er gebruik gemaakt worden van TDD (Test Driven Development). Hieruit volgen unit tests.

Documenteren

In deze fase van het project wordt er documentatie geschreven om de gebruikers van het systeem hier wegwijs in te maken.

Klaarmaken release

Wanneer de afstudeerstage is afgerond, is het volledige product niet af, er zal dus nog verder aan gewerkt gaan worden door een andere ontwikkelaar. Om dit op een goede manier te doen, moet er worden afgebakend welke requirements er wel en welke er niet zijn geïmplementeerd, en op welke manier deze geïmplementeerd zijn. Hiervoor kan de eerder opgestelde ontwikkeldocumentatie, en de lijst met geïmplementeerde requirements gebruikt worden. Eventueel moeten hier nog aanpassingen in gemaakt worden, en zal deze informatie gebundeld moeten worden in een overdraagbaar product.

De release die uitgevoerd gaat worden zal dus geen release naar een klant zijn, maar een release binnen Cenosco naar een andere ontwikkelaar.

Wanneer er een project volledig wordt doorlopen, worden er nog twee andere activiteiten gedaan, namelijk de Transition to Support (de applicatie overgeven naar de supportafdeling die eventuele bugs hieruit verwijderen) en de Roll out (de applicatie uitrollen bij de klant). Ik ga deze activiteiten niet doorlopen, omdat ik niet het gehele traject ga doorlopen. Deze twee activiteiten worden binnen Cenosco op een later moment uitgevoerd.

Methodes en technieken

Methodes

Project Delivery Framework

Binnen Cenosco wordt er gebruik gemaakt van de projectbeheermethode Project Delivery Framework (PDF).

PDF onderscheidt verschillende fases. Een project wordt opgestart in de “Project Setup”-fase. Hierin wordt een plan van aanpak opgesteld. Hierop volgt de “Requirements”-fase, waarin de Requirements voor het project worden verzameld.

De derde fase is de “Design”-fase, hierin wordt de software, de database en de GUI ontworpen. Hierna komt de “Development”-fase waarin dit alles wordt geprogrammeerd. De vijfde fase is de “Deploy & Stabilize”-fase, waarin de applicatie in een testomgeving wordt gedraaid om te kijken hoe de applicatie met meerdere gebruikers functioneert. Ten slotte komt de applicatie in de “Run & Maintain”-fase, waarin de applicatie door de klant in gebruik genomen wordt, en bugs, etc. worden opgelost.

Omdat er bij het ontwikkelen van het project gebruik gemaakt gaat worden van iteraties, zal mijn ontwikkeltraject niet geheel volgens de PDF-methode verlopen. De Design- en Developmentfase zullen gecombineerd worden en in iteraties worden uitgevoerd. Ook is er in gesprekken met de opdrachtgever naar voren gekomen dat mijn project misschien net de “Run & Maintain”-fase zal aantikken, maar niet verder dan dat zal gaan, omdat aan het einde van mijn afstudeerperiode het product nog niet klaar is om in de markt gezet te worden.

Om mijn project toch zoveel mogelijk binnen de standaarden van het bedrijf te houden, zullen de eerste twee fasen wel doorlopen worden.

Scrum

Binnen het design- en ontwikkelstuk zal er in iteraties worden gewerkt. Deze iteraties zullen via de ontwikkelmethode Scrum worden uitgevoerd.

Omdat het product wat ontwikkelt gaat worden is ontstaan vanuit een idee, is het lastig om alle Requirements vanaf het begin van het project rond te hebben. Deze zullen hoogstwaarschijnlijk tijdens het project worden bijgesteld. Een iteratieve methode kan hierbij van pas komen, omdat deze hierop inspelen.

Ook mis ik als afstudeerder zijnde nog veel kennis over het opzetten van een volledig systeem. Ik zal me moeten verdiepen en inlezen op een aantal aspecten die voor mij nieuw zijn. Dit uitzoeken is lastig te plannen, en kan hierdoor langer duren dan verwacht. Als blijkt dat ik een achterstand op loop, kan er per sprint worden gespeeld met de prioriteiten van de eisen, zodat de onderdelen met het grootste belang zeker in het eindproduct aanwezig zijn.

MoSCoW

MoSCoW is een methode waarmee requirements kunnen worden ingedeeld. De hoofdletters zijn de eerste letter van een categorie waar de requirements in onderverdeeld kunnen worden, de o's zijn er enkel zodat het woord als een ezelsbruggetje gebruikt kan worden. MoSCoW onderscheidt de volgende categorieën:

- **Must have** De requirement moet in het uiteindelijke product aanwezig zijn, anders is het product niet bruikbaar.
- **Should have** Deze requirement is zeer gewenst, maar zonder functioneert het product wel
- **Could have** Deze requirement wordt enkel geïmplementeerd als er tijd over is
- **Won't have** Deze requirement wordt niet geïmplementeerd, maar is wel verzameld voor een eventueel vervolgproject

Als niet alle requirements die als must have zijn ingedeeld zijn geïmplementeerd, is het project gefaald.

UML

Om het softwareontwerp te maken zal er gebruik gemaakt worden van de modelleertechniek Unified Modeling Language (UML). Hierbij worden er domeinmodellen en klassediagrammen gemaakt. Tevens zal er voor de gewenste onderdelen een sequencediagram worden gemaakt.

Technieken

ASP.NET mvc 3

ASP.NET mvc 3 is een webframework dat op het .NET framework en ASP.NET draait, en zoals de naam al zegt, het Model-View-Controller (mvc) pattern implementeert. Met dit framework kunnen webapplicaties worden ontwikkeld, waarbij er gebruik gemaakt kan worden van het .NET framework.

C# & .NET

ASP.NET mvc draait bovenop het .NET framework. Dit is een volwassen framework, ontwikkeld door Microsoft. Het .NET framework heeft een groot aantal klassen ter beschikking waar veel

functionaliteit al aanwezig is, zoals klassen om het filesystem aan te spreken, webrequests uit te voeren en grafische interfaces op te stellen. Door deze toevoegingen kan er sneller met het daadwerkelijke project worden gestart, omdat deze randfunctionaliteiten niet ontwikkeld hoeven te worden.

Wanneer een .NET-programma wordt gecompileerd, wordt dit niet naar machinecode gedaan, zoals bij C++ gebeurd, maar wordt er gecompileerd naar een soort van tussentaal, welke de Microsoft Intermediate Language heet. Wanneer het programma wordt uitgevoerd, wordt deze code door middel van just in time compilation omgezet naar machinecode. Zoals de naam al zegt, wordt de code net op tijd gecompileerd, namelijk voordat het programma opstart. Java gebruikt een soortgelijk principe.

Er zijn meerdere talen beschikbaar voor het .NET framework, waarvan C# en Visual Basic.NET de bekendste zijn. Al deze talen worden naar Microsoft Intermediate Language gecompileerd. Als twee dezelfde functionaliteiten in twee verschillende talen worden geprogrammeerd, zouden de libraries die hieruit volgen theoretisch gezien identiek zijn. Hierdoor is het mogelijk om binnen een project meerdere talen die op het .NET framework gebouwd zijn te gebruiken, bijvoorbeeld Visual Basic.NET voor de grafische user interface, en C# voor de achterliggende functionaliteit.

Binnen het project wordt er enkel gebruik gemaakt van C#, omdat deze taal de standaard is binnen de organisatie en ik hier ook de meeste ervaring mee heb. Om de consistentie te bewaren worden er geen talen binnen het project combineert.

HTML, CSS en Javascript

De applicatie die ontwikkeld gaat worden is een webapplicatie. Om een website of webapplicatie structuur te geven wordt HTML gebruikt. HTML is een opmaaktaal die erg op XML lijkt. HTML specificeert een aantal elementen waarmee de structuur van de pagina gespecificeerd kan worden.

HTML-elementen hebben van zichzelf weinig tot geen opmaak. Om deze elementen opmaak te geven worden Cascading Stylesheets (CSS) gebruikt. Deze opmaaktaal is er speciaal voor ontwikkeld om webpagina, van layout te voorzien.

Ten slotte is het mogelijk om binnen de browser scripts te draaien door middel van Javascript. Hiermee is het mogelijk om de pagina aan te passen nadat de pagina geladen is. Voor deze aanpassingen hoeft de pagina dus niet wederom opgehaald te worden vanaf de server.

Een veelgebruikte toepassing voor Javascript is Ajax. Hiermee is het mogelijk om een verzoek te doen naar de server voor data, zonder dat de pagina hiervoor ververst hoeft te worden. Hierdoor kan er een ervaring worden geboden die meer lijkt op een desktopapplicatie dan een website.

Een simpel voorbeeld van een Ajax-request kan een twitter-feed op een website zijn. Het script kijkt automatisch via Ajax of er nieuwe tweets zijn. Als er nieuwe berichten zijn, worden deze automatisch in de lijst met tweets geplaatst zonder dat de gebruiker de pagina hoeft te verversen.

jQuery

jQuery is een javascript-library die gemaakt is om de interactie met de browser gemakkelijker te maken. Een voorbeeld is Ajax. Dit is in verschillende browsers net iets anders geïmplementeerd. jQuery zorgt voor een overkoepelende functionaliteit die in al deze browsers functioneert.

Daarnaast wordt jQuery ook gebruikt om makkelijker de pagina aan te spreken via Javascript. Om een element op de pagina, bijvoorbeeld een afbeelding, te selecteren, moest de Document Object Model (DOM) worden gebruikt. De DOM is een boomstructuur waar alle elementen binnen de pagina in staan. jQuery maakt het mogelijk om elementen te selecteren op de manier waarmee dit ook in CSS gebeurd, wat het gebruik een stuk eenvoudiger en flexibeler maakt.

Microsoft Windows Azure

Windows Azure is het cloud-platform van Microsoft, waar de uiteindelijke applicatie op gaat draaien. Hoewel ik niet zal zorgen voor het directe eindresultaat, zal er wel rekening gehouden moeten dat de applicatie in de cloud komt te staan. Bij het ontwikkelen zullen hier enkele keuzes anders door moeten worden genomen.

Microsoft SQL Azure

Binnen de webapplicatie wordt er ook gebruik gemaakt van een database. De keuze voor de database-engine is gevallen op Microsoft SQL Azure, omdat deze database binnen Azure wordt ondersteund.

Team Foundation Server (TFS)

Microsoft heeft een eigen pakket ontwikkeld voor softwareontwikkelprojecten, Team Foundation Server (TFS), wat ook binnen Cenosco wordt gebruikt. Van de mogelijkheden die het pakket biedt, gaat er gebruik gemaakt worden van de source control en de project tracking.

Het voordeel wat TFS biedt ten opzichte van andere producten, is de integratie van de bovengenoemde onderdelen. Er is een directe koppeling tussen de requirements die zijn ingevoerd binnen het project tracking-onderdeel, en de check-ins van de code die zijn gedaan. Hierdoor is achteraf te zien welke code op welk moment bij welke requirement ingechecked is.

Test Driven Development (TDD)

Test Driven Development (TDD) is een softwareontwikkelmethode waarbij eerst de tests worden geschreven, en daarna pas de code. Bij TDD wordt er gebruik gemaakt van Red/Green/Refactor. Dit geeft de drie stappen aan die gedaan worden binnen TDD:

- Schrijf eerst de test. Omdat de code die de test moet gaan uitvoeren nog niet bestaat, zal de test falen. Dit wordt Red genoemd, omdat dit overeen komt met de kleur die falende tests hebben in de tools die unit tests uitvoeren. Slaagt de test op dit moment wél, dan geeft dit over het algemeen aan dat de test fouten bevat, of dat niet het juiste wordt getest.
- Schrijf de code. Wanneer de test nu slaagt, weet je dat de functionaliteit doet wat je verwacht dat het doet. Slaagt de test niet, zal of de code, of de test niet goed zijn. Het slagen van de test wordt in verband gebracht met de kleur groen, oftewel Green, omdat geslaagde tests doorgaans deze kleur krijgen.
- Nu je weet dat de code doet wat je verwacht dat hij doet, kan de code worden opgeschoond. Overbodige stukken kunnen worden verwijderd, functies kunnen worden opgesplitst om de leesbaarheid te vergroten, etc. Dit wordt refactoren genoemd. Hierbij dient de test als een soort van vangnet. Op het moment dat een test slaagt voordat er gerefactored wordt, en na het refactoren falen deze tests, weet je dus dat bij het refactoren wat fout gegaan is. Zo kan je dus code aanpassen waarbij je weet dat je niets kapot maakt.

Test Driven Development heeft een aantal voordelen ten opzichte van de tests aan het einde van het ontwikkelen opstellen.

Ten eerste worden de tests tegelijk met de code geschreven. Hierdoor zijn er dus tests van alle code aanwezig. Op deze manier wordt voorkomen dat tests aan het einde van het project worden opgesteld. Het komt niet zelden voor dat een project uitloop heeft, en om toch op tijd op te kunnen leveren kan het zijn dat de tests worden overgeslagen. De kwaliteit kan hierdoor niet gegarandeerd worden.

Ten tweede moet de code die geschreven wordt wel testbaar zijn. Omdat eerst de test geschreven wordt, is de code testbaar, omdat anders de test nooit geschreven kan worden. Het effect hiervan is ook positief voor het ontwerp van het systeem. Bij een unit test is het de bedoeling dat er een zo klein mogelijk deel wordt getest. Op het moment dat er wordt getest of er een gebruiker aangemaakt kan worden, is het niet de bedoeling dat de gebruiker aan de database wordt toegevoegd. Er worden dan immers twee dingen getest, namelijk het aanmaken van de gebruiker en het toevoegen in de database. Om dit te voorkomen wordt er naar gestreefd om de componenten die je test een lage afhankelijkheid te geven: het moet in hier bijvoorbeeld mogelijk zijn binnen de testcontext de echte database te vervangen voor een “nepdatabase”, die relevant is binnen de test. Door hier naar te streven is niet alleen de code testbaar, maar wordt er ook “loose coupling” bereikt.

Ten slotte zorgt TDD voor een grotere zekerheid bij het ontwikkelen. Als er iets fout gaat, wordt dit afgevangen in de specifieke test. De debugger hoeft dus minder te worden gebruikt om er achter te komen op welk punt in de applicatie er iets fout gaat.

Risico’s

Bij het project moet er rekening gehouden worden met het feit dat ik als afstudeerder zijnde praktijkervaring mis. Hierdoor zal ik niet met dezelfde snelheid werken als ontwikkelaars die deze praktijkervaring wel hebben. Daarbij zal het voorkomen dat ik beginnersfouten maak. Ook zal ik me moeten verdiepen in de onderdelen waar ik nog geen ervaring heb. Door deze punten zal de tijdsduur van het project bovengemiddeld zijn.

Team

Het team voor het project ziet er als volgt uit:

Opdrachtgever	Mischa Simonis
Ontwikkelaar	Jesse van Assen
Mentor	contextafhankelijk

De opdrachtgever hierin is Mischa Simonis. Hij representeert hierin volledig Cenosco.

De rol van ontwikkelaar zal ik zelf innemen. Dit is in de breedste vorm, omdat ik niet enkel zal gaan ontwikkelen. Ik zal alle activiteiten uit hoofdstuk 0 gaan doorlopen.

De mentorrol zal contextafhankelijk worden toegewezen. Zijn rol is hierbij om mij te helpen bij eventuele technische vraagstukken. Binnen Cenosco hebben de medewerkers verschillende

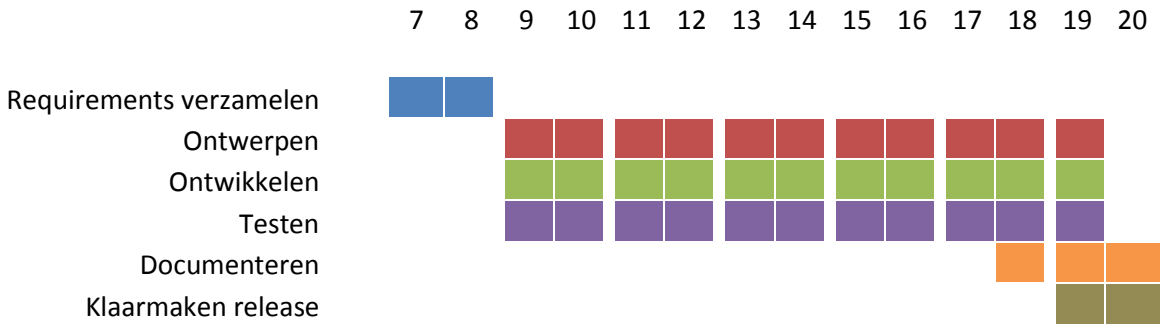
expertises. De mentorrol zal bij een probleem worden aangenomen door de medewerker die hier het meeste verstand van heeft.

Ondanks dat de mentor onderdeel van Cenosco is, zal zijn rol niet zijn om inhoudelijk dingen over de opdracht te verduidelijken. Dit blijft de taak van de opdrachtgever.

Planning

De activiteiten die verricht gaan worden zijn hieronder in een globale planning verwerkt. De planning is op weekniveau. Omdat mijn afstudeerstage begonnen is op 6 februari, en de eerste week besteed is aan het kennis maken en bekend worden in het bedrijf, en het opstellen van dit plan van aanpak, begint de planning bij week 7. Omdat mijn afstuderen 70 dagen in beslag neemt, en tot 18 mei loopt, eindigt de planning bij week 20.

Ik ga het project in sprints van twee weken uitvoeren. Deze sprints zullen door middel van Scrum worden uitgevoerd. Het verzamelen van requirements zal ook de duur hebben van een sprint, maar niet via Scrum worden uitgevoerd. In totaal zal ik 6 sprints gaan doorlopen.



Bijlage B: Requirements

Introduction

In this document, the functional and non-functional requirements will be described.

In the first part of the document, the functional requirements will be described. Functional requirements indicate what the application should do. This section is divided in three parts. The first part describes the actors of the system. The second part describes the requirements for the platform wherein conflict can be resolved. The third part describes the requirements for the division of property conflict. The last part defines the priorities of the requirements.

In the second section of the document, the non-functional requirements will be described. These requirements describe how the application should do things.

Functional requirements

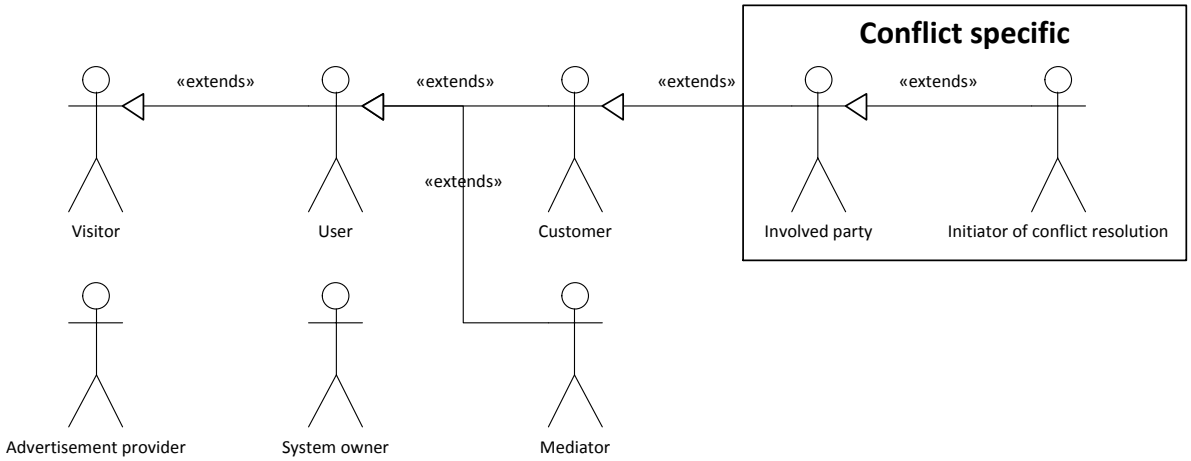
Functional requirements define the things an application needs to accomplish from a customer perspective. This chapter begins by defining the types of actors in the system. In the second part, the requirements will be defined in the form of user stories. The user stories will have one of the following formats:

<identifier> *<title>* As a *<role>*, I want *<goal/desire>*.

Or

<identifier> *<title>* As a *<role>*, I want *<goal/desire>* so that *<benefit>*.

Actors



Visitor

A visitor is an actor who visits the site. This can be anyone who visits, from a real person to a search engine spider. Visitors can have an account, but when they are in the visitor state, they haven't logged in. When they try to access a restricted part of the web application, they are prompted to log in.

User

A user is a visitor who has an account and has been logged in. Users have access to the restricted areas of the web application.

Customer

A customer is a user that is able to initiate conflicts or can be involved in them. These actors are the primary target for the application.

Involved party

The involved party is the party with whom the initiator of the conflict has the conflict. In a divorce conflict for example, this is the former spouse. The involved party can perform interactions with the conflict.

A customer becomes an involved party after he has been invited by the initiator of the conflict resolution.

Initiator of the conflict resolution

The initiator of the conflict resolution is the involved party which started the resolution of the conflict. The initiator has the most rights within the conflict for editing the conflict, and making important choices, like marking the conflict as resolved.

The role of initiator is specific for each conflict. In one conflict he can be the initiator, but in another he can be the involved party.

Mediator

A mediator is a user who can view conflicts on which he is invited. The role of the mediator is to provide a second opinion, when the parties involved in the conflict aren't able to resolve it by themselves. In these situations, the mediator can provide help with resolving the conflict from an independent standpoint. The mediator can only advise the involved parties, he cannot interact with the conflict, apart from viewing it.

The mediator can't create conflicts himself. He can only view conflicts on which he is invited. The invitation to view the conflict comes from the initiator of the conflict resolution.

Advertisement provider

The advertisement provider provides context relevant advertisements when asked.

We don't want to flood the customers with advertisements, because the advertisements won't be taken serious this way. If the advertisements are shown to the customers everywhere, they recognize them as advertisements and won't notice them anymore, even when they are relevant. This is called banner blindness (Nielsen, 2007).

We also only want to show context relevant advertisements. When a customer loses all the furniture in a divorce, he probably isn't interested in buying ringtones for his phone, but he most likely is interested in an advertisement for a sale in a local furniture store. Apart from making a profit from the advertisements, we want to help the customers, not annoy them.

System owner

In some cases, the product owner wants a feature from his own standpoint, and not from the user of the system. An example for this is the ability for the users to make a payment. This is at the benefit of the system owner, and not of the customer.

User stories for the platform by actor

Within the system, there are components which are shared between the conflict profiles. In this section the shared components will be treated.

Visitor

1 Landing page
As a visitor, I want to see a landing page when I visit the application so that I know what the application is, and what it can do.

2 View conflict profiles
As a visitor, I want to be able to view a list of profiles which the application provides so that I know if the application can help me resolving the conflict that I have.

3 Register
As a visitor, I want to be able to register so that the system will know who I am, and I am able to use the restricted areas of the application.

4 Log in
As a visitor, I want to be able to log in so that the system knows who I am, and I can use the restricted areas of the application.

5 Log in using OpenID
As a visitor, I want to be able to login with, and register using an OpenID provider, so that I don't have to remember another username and password combination.

User

6 Log out
As a user, I want to be able to log out.

7 View help page
As a user, I want to be able to view a help page, so that I can find the answer to my questions if I don't understand the system.

8 Receive notification
As a user, I want to be able to receive a notification if something important happened, so that I won't miss anything.

9 How to receive notifications
As a user, I want to be able indicate how I want to receive notifications (like email, sms, etc.), so that I won't miss anything, but also don't get spammed if I don't want to.

Customer

10 Initiate resolution of conflict
As a customer, I want to be able to initiate the resolution of a conflict by picking it from the catalog and making a payment.

11 Carry out iDeal payments
As a customer, I want to be able to carry out payments via iDeal, so that I can use my own bank to transfer the money and the transaction is carried out in a secure environment.

12 Carry out credit card payments
As a customer, I want to be able to carry out credit card payments.

13 View relevant advertisement
As a customer, I only want to see relevant advertisements, so that they don't bug me but help me.

Involved party

14 View list of active conflicts
As an involved party, I want to be able to view the list of active conflicts that I am a part of, so that I can see the status of the conflicts, and I can quickly navigate to it.

15 Be notified of changes
As an involved party, I want to be notified when a change happens to a conflict that I am involved in.

16 View changes
As an involved party, I want to view the changes which are made to the conflict that I haven't accepted or rejected yet.

17 Accept change
As an involved party, I want to be able to accept a change that another involved party made, so that that party knows that I approve of the change, and the conflict can move on.

18 Reject change
As an involved party, I want to be able to reject a change that another involved party made when I disagree with the changes, so that the involved party knows that he needs to modify the change and the conflict can't be resolved.

19 Leave message with rejected change
As an involved party, I want to be able to leave a message with a rejected change, so that the party that made the change knows why I don't agree with his change.

Initiator of the conflict resolution

20 Invite involved party
As an initiator of the conflict resolution, I want to be able to invite an involved party.

21 Invite mediator
As an initiator of the conflict resolution, I want to be able to invite a mediator when I'm unable to resolve the conflict, so that the mediator can give me a second opinion which can help me resolve the conflict.

22 Mark conflict as resolved
As an initiator of the conflict resolution, I want to be able to mark the conflict as resolved, so that I and all other involved parties know that the conflict is resolved.

23 Modify conflict
As an initiator of the conflict resolution, I want to be able to modify the conflict after it has been created, so that I can update things that have changed or are incorrect.

Mediator

24 View list of conflicts
As a mediator, I want to view the list of conflicts that I have been invited to.

25 View conflict
As a mediator, I want to view the conflict.

26 Give second opinion
As a mediator, I want to give a second opinion on a conflict, so that I can make a little profit.

System owner

27 Carry out payment
As a system owner, I want customers to be able to carry out payments, so that they can use the services that I offer, and I can increase my revenue.

28 Show advertisements
As a system owner, I want to show advertisements to my customers so that I can increase my revenue

29 Use secure connection
As a system owner, I want private data to be sent using a secure connection, so the data the users fill in can't be overheard by third parties.

User stories for the conflict "division of property" by actor

Within the scope of the project, one profile to resolve a conflict with will be built. This is the division of property conflict, which happens during a divorce. The owned property needs to be divided over both the parties, and this profile can be utilized to help with this task. In this section, the user stories for this profile will be treated.

Involved party

30 Add product to product list
As an involved party, I want to be able to add a product to the product list, so that this product is included in the division.

31 Use wizard to add products
As an involved party, I want to be able to use a wizard that leads me through all the rooms to add products, so that adding products happens in small, overseeable steps.

32 Group products per room
As an involved party, I want to be able to group the products in the product list by room, so that the list is better structured, and it is easier to add products because I can make a mental picture of the room.

33 Add ownership to product
As an involved party, I want to be able to add the ownership property to a product to indicate whose it is, so that this can be taken into consideration during the division.

34 Add divide preference to product
As an involved party, I want to be able to indicate who prefers to have the product in the division, so that this can be taken into consideration during the division.

35 Add price category to product
As an involved party, I want to be able to add a price category property to the product, so that I don't have to fill in the specific price, and the price indication can be taken into consideration during the division.

36 Add age to product
As an involved party, I want to be able to add an age to the product so that during the division the current value can be estimated.

37 View list of mandatory investments
As an involved party, after the division of products I want to be able to view a list of primary needs I need to invest, so that I know what I need to buy.

38 Graphical view of progress
As an involved party, I want to be able to view a graphical representation of the progress of the resolution of the conflict, so I can quickly see how far it has been progressed.

39 Download product list
As an involved party, I want to be able to download the divided product list, so that I can print it, or send it via an email message.

Initiator of the conflict resolution

40 Fill product list on creation
As an initiator of the conflict resolution, I want that the list with product is filled on creation time with default products so that I don't have to fill them in manually.

41 Automatically divide product list
As an initiator of the conflict resolution, I want to be able to automatically divide the remaining products in the product list based upon the current division and the properties, so I don't have to do it myself.

System owner

42 Show advertisements
As a system owner, I want to show advertisements for stores after the products have been divided, so that users can click them to buy products and I get a profit from the click.

Priority of requirements

Id	Title	Priority (M o S C o W)
1	Landing page	Must have
3	Register	Must have
4	Log in	Must have
5	Log in using OpenID	Must have
6	Log out	Must have
7	View help page	Must have
8	Receive notification	Must have
10	Initiate resolution of conflict	Must have
11	Carry out iDeal payments	Must have
12	Carry out credit card payments	Must have
14	View list of active conflicts	Must have
20	Invite involved party	Must have
22	Mark conflict as resolved	Must have
23	Modify conflict	Must have
29	Use secure connection	Must have
30	Add product to product list	Must have
31	Use wizard to add products	Must have
33	Add ownership to product	Must have
34	Add divide preference to product	Must have
35	Add price category to product	Must have
36	Add age to product	Must have
2	View conflict profiles	Should have
9	How to receive notifications	Should have
15	Be notified of changes	Should have
16	View changes	Should have
17	Accept change	Should have
18	Reject change	Should have
19	Leave message with rejected change	Should have
27	Carry out payment	Should have
32	Group products per room	Should have
40	Fill product list on creation	Should have
41	Automatically divide product list	Should have
13	View relevant advertisement	Could have
21	Invite mediator	Could have
24	View list of conflicts	Could have
25	View conflict	Could have
26	Give second opinion	Could have
28, 42	Show advertisements	Could have
37	View list of mandatory investments	Could have

38	Graphical view of progress	Could have
39	Download product list	Could have

Non-functional requirements

In this part, the non-functional requirements are specified. These requirements describe how the system should operate, rather than what the system should do.

Functionality

Language requirements

The languages used in this project are:

- C#
- Javascript / jQuery
- HTML & CSS
- SQL

Interoperability

The interface of the application will be a web application which can be accessed from an internet browser.

The application should work on all mayor browsers. According to the website W3Counter (Global web stats - January 2012), these are, of February 1st:

- Microsoft Internet Explorer
- Mozilla Firefox
- Google Chrome
- Apple Safari

Opera has a usage percentage of 2.5%. This won't fit the terms of a major browser.

Lower versions of Internet Explorer are known for messing up the layout and aren't working according to the web standards. However, more than a fifth of the users today still use these versions (15.3% uses Internet Explorer 8, 5.9% uses Internet Explorer 7). Because of this, extra considerations and efforts will be made to support these versions of the browser. Internet Explorer 6 will not be supported.

Security

All personal areas of the application will be secured by user authentication.

To prevent listening in by other parties, all activity where the user needs to fill in private information will be send using a secured connection. This way, private data like user names and passwords, or personal information can't be retrieved by analyzing the data connection.

Reliability (Availability)

The application will be hosted in the Windows Azure cloud, which guarantees an uptime of 99.95% in their service level agreement.

The cloud provides an almost infinite amount of virtual machines. The traffic is automatically loadbalanced between the machines. When the traffic becomes too large for the active machines, an extra virtual machine can be switched on. This can be done manually, or automatically by Azure.

Windows Azure automatically monitors virtual machines, and when a machine fails, it gets replaced automatically by a new one. This way, high uptime is guaranteed without the need for monitoring.

Windows Azure automatically creates backups from the files and databases stored in the cloud.

Usability

Understandability

In the effort of making the application as natural to use as possible, we will look at existing systems to preserve the user's existing mental models.

Learnability

In developing the application, we will aim for making it as easy and natural to use as possible. We will also offer a help page to clarify functionalities if they aren't understood by the user.

Accessibility

The application will provide a web interface for providing access.

Efficiency (Performance)

The cloud is highly scalable. When the need for a faster server arises, this can be done through the Azure configuration portal. This can also be done automatically, although this may lead to unforeseen costs.

This upscale can be done in two ways. Azure provides different sizes for virtual machines, ranging from extra small to extra large. In the extra small size, azure provides one shared processor core, and 768 megabytes memory. The extra large size offers eight dedicated cores, and 14 gigabytes of memory. One way to upscale is to migrate to a larger size virtual machine.

Another possibility is to add another virtual machine. Windows Azure provides a nearly unlimited amount of servers. The traffic is automatically loadbalanced between the machines when a new machine is powered on.

Portability(Flexibility)

Adaptability

The application needs to be flexible in multiple points. The most important being the possibility to add new profiles for conflict resolutions. This will be the main extensibility point of the application.

Also, the application might be transferred from the cloud to a local server at a later moment. Everything that uses cloud specific features should be abstracted so it is possible to replace the implementation if the need arises.

Installability

The cloud makes it very easy to install the software. Publishing the app can be done from within Visual Studio.

Deploying an application to the cloud happens in a couple of steps:

- Create a new virtual machine where the new package will be staged
- Deploy the new package to this virtual machine
- Swap the new virtual machine for the old one
- Shut down the old virtual machine

When done this way, there isn't any downtime for the application.

Replaceability

Components which need to be flexible should be identified.

An example is cloud specific elements. When switching to a different cloud host, or to a regular server, this should be possible without modifying the internal structure of the application

Maintainability (Supportability)

How easy it is to support and maintain the solution once it has been delivered in the operational environment.

Analyzability

When an exception occurred, it is important that the cause of the exception can be found. When the exception resulted in a crash, a log file is of the upmost importance, so the exception can be reproduced and resolved.

There are components especially designed for this task. During the project, a comparison of logging frameworks will be made, and one will be chosen.

Changeability (Configurability)

To avoid having to recompile for every little change, things that could be modified by configuration should implement this.

Deployability

Deploying to the live environment is easy. To deploy, it only needs to be copied to the cloud, and all users will use the new version automatically. Deploying is done through the Azure SDK, integrated in Visual studio.

Bijlage C: Resultaten unit tests en code coverage

Testresultaten

Result	Class name	Test name
Passed	MenuControllerTests+RenderTopMenuTests	UserIsNotAuthenticated_ShowDashboardLinkIsFalse
Passed	TemplateControllerTests+CreateTests	TemplateDoesNotYetExist_SuccessIsTrue
Passed	LocationExporterFactoryTests+GetLocationExporterTests	AskForUnknownExporter_ExceptionIsThrown
Passed	DelegateEqualityComparerTests+GetHashCodeTests	GetHashCodeIsCalled_ParameterIsPassedCorrectly
Passed	ConflictControllerTests+CreatePostTests	ConflictIdIsGiven_AllRolesIsPassedToBuilder
Passed	ConflictDivisionOfPropertyTests+GetTypeTests	GetTypeCalled_ReturnsDivisionOfPropertyTypeConstant
Passed	TemplateControllerTests+UpdateGetTests	ActionCalled_TemplateNameAddedToViewModel
Passed	EmailInvitationSenderTests+CreateMailMessageTests	MailMessageCreated_SubjectIsSet
Passed	ConflictControllerTests+CreatePostTests	NonExistingIdIsGiven_RedirectedToCreate
Passed	PersonTests+FullName	PersonDoesntHaveTussenvoegsel_ReturnFullName
Passed	ConflictInvitationControllerTests+CreateTests	InvitationIsCreated_RedirectedToInvolvedParty
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_CustomProductsAreAddedToViewModel
Passed	ConflictDivisionOfPropertyTests+DetailsTest	ActionCalled_ViewIsReturned
Passed	LocationControllerTests+CreateTests	FormIsFilledOut_SaveHasBeenCalled
Passed	ConflictDivisionOfPropertyBuilderTests	MethodCalled_ConflictDivisionOfPropertyIsReturned
Passed	LocationControllerTests+ExportTests	FormatIsNull_ExceptionIsThrown
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_RoomInstanceIsAddedToViewModel
Passed	ProductTypeControllerTests+IndexGetTests	ActionCalled_RoomsAreInsertedInViewModel
Passed	ConflictInvitationControllerTests+RevokeTests	CurrentUserIsntOwner_UnauthorizedExceptionIsThrown
Passed	LocationControllerTests+ExportTests	LocationIdIsNull_LocationWithIdIsSelected
Passed	DataContextTests	GetDataContextValue
Passed	TemplateRoomInstanceControllerTests+DetailsPostTests	ActionCalled_SaveHasBeenCalled
Passed	PersonTests+FullName	PersonHasTussenvoegsel_ReturnFullName
Passed	TemplateControllerTests+CreateTests	NameIsNull_ResultIsFalse
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_SaveIsCalled
Passed	Base64EncodingAlgorithmTests+EncodeFromStringTests	KnownInputAndOutput_ExpectedOutputMatchesActualOutput
Passed	ConflictControllerTests+CreatePostTests	ConflictIdIsGiven_SaveIsCalled
Passed	TemplateRoomInstanceControllerTests+DetailsGetTests	ActionCalled_MultipleProductPerProductType
Passed	TemplateControllerTests+UpdateTests	ActionIsCalled_SaveIsCalled
Passed	ConflictInvitationControllerTests+AcceptTests	InvitationIsntFound_ShowNotFoundView
Passed	RoomControllerTests+IndexGetTests	ActionCalled_ListRoomsOrderedByName
Passed	AuthenticationTests+SignOut	CookieDoesntExist_ValuelessCookieIsInserted
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	KnownUser_DuplicatePrivilegesAreMerged
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_RoomsAreAddedToViewModel
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_ViewModelIsReturned
Passed	RoomInstanceControllerTests+DetailsGetTests	MultipleProductsWithSameRooms_RoomsAreDistinct
Passed	ConflictInvitationControllerTests+CreateTests	ConflictUnknown_ViewIsReturned
Passed	IEnumerableExtensionsTests+IsNullOrEmptyTests	IEnumerableIsNotNullAndNotEmpty_ReturnsFalse
Passed	TemplateControllerTests+UpdateTests	ActionIsCalled_RedirectToIndex
Passed	ProductTypeControllerTests+IndexPostTests	CreateInViewModel_AddedToRepository
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_EachRoomIsIncluded
Passed	RoomInstanceTests+GetDescriptionTests	DescriptionIsNotNull_RoomAndDescriptionAreReturnedSeparatedByDash
Passed	RoomControllerTests+IndexPostTests	ActionCalled_EmptyResultIsReturned
Passed	FilteredConflictRepositoryDecoratorTests+QueryTests	UserIsNotAnInvolvedPartyInConflict_ConflictIsNotInRepository
Passed	TemplateRoomInstanceControllerTests+DetailsGetTests	ActionCalled_ProductsAreAdded
Passed	TemplateControllerTests+DeleteTests	DeleteIsCalledWithValidId_ViewDataIsTrue
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_RoomMarkedForUpdate_RoomIsUpdated
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_DescriptionIsUpdated
Passed	LocationControllerTests+ExportTests	ExportActionIsCalled_ExportLocationsIsCalledOnLocationExporter
Passed	RoomInstanceControllerTests+GetCatalogProductsTests	MethodCalled_ProductsAreReturned
Passed	AccountControllerTests+LogOffTests	ActionCalled_SignOutHasBeenCalled
Passed	ProductCategoryControllerTests+IndexPostTests	CreateInViewModel_AddedToRepository
Passed	TemplateControllerTests+CreateTests	TemplateAlreadyExists_SuccessIsFalse
Passed	InvolvedPartyTests+AddRolesTests	AddUnknownRoleName_RoleIsntAdded
Passed	RoomControllerTests+IndexPostTests	UpdateInViewModel_UpdatedInRepository
Passed	ConflictControllerTests+CreatePostTests	ConflictIdIsGiven_ConflictTypeIsPassedToFactory
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	KnownUser_AllRolesAreAdded
Passed	CsvLocationExporterTests+ExportLocationsTests	ParameterIsNull_ArgumentNullExceptionIsThrown
Passed	DelegateEqualityComparerTests+EqualsTests	EqualsIsCalled_DelegateIsCalled
Passed	TemplateControllerTests+UpdateGetTests	ActionCalled_RoomsAreSortedByName
Passed	IdentityTests+UserIdTests	UserIdCalled_ReturnsUserId
Passed	AccountControllerTests+IsUsernameAvailableTests	UsernameNotInRepository_ReturnTrue
Passed	TemplateControllerTests+UpdateTests	DeleteIsGiven_RoomInstancesAreRemoved
Passed	ConflictDivisionOfPropertyBuilderTests	AllRolesIsNull_ArgumentNullExceptionIsThown
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredCorrectData_AccountIsActivated
Passed	PrincipalTests+IsInRole	IdentityDoesntHaveRole_ReturnsFalse
Passed	AesEncryptionStrategyTests	EncryptedThenDecrypted_ResultEqualsOriginal
Passed	TemplateControllerTests+UpdateGetTests	ActionCalled_RoomInstancesAreSortedByRoom
Passed	RoomInstanceControllerTests+DetailsPostTests	CustomProductsMarkedForCreationButProductCategoryIsntInRepository_DontAddCustomProducts
Passed	DetailsTests+GetDescriptionTests	DescriptionIsNull_OnlyNameIsReturned
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_RoomsAreSortedByName
Passed	HtmlConflictExtensionsTests+ConflictLinkTests	UnknownConflictType_ThrowsException
Passed	ConflictInvitationControllerTests+CreateTests	InvitationIsCreated_FormatHasBeenCalled
Passed	RoomInstanceControllerTests+DetailsPostTests	ModelStateIsInvalid_RedirectToView
Passed	AuthenticationTests+SignIn	SignInIsPersistent_ExpirationDateIsAdded
Passed	ConflictInvitationControllerTests+AcceptTests	InvitationIsFoundAndTokenMatches_InvolvedPartyRolesAreAddedToParty

Passed	AuthenticationTests+SignIn	SignInCalled_EncryptIsCalled
Passed	TemplateControllerTests+IndexTests	ActionCalled_RoomsAreSortedByType
Passed	Sha512HashingAlgorithmTests+CreateHashTests	InputIsNull_ArgumentNullExceptionIsThrown
Passed	ConflictDivisionOfPropertyBuilderTests	MethodCalled_OwnerIsAddedToInvolvedParties
Passed	TemplateRoomInstanceControllerTests+DetailsPostTests	ProductsInCreate_MultipleDuplicateProductsAreAdded
Passed	TemplateControllerTests+IndexTests	ActionCalled_RoomsAreAddedToTemplate
Passed	TemplateRoomInstanceControllerTests+DetailsGetTests	ActionCalled_ProductTypesAreSortedOnCategoryThenOnType
Passed	DelegateEqualityComparerTests+GetHashCodeTests	GetHashCodeIsCalled_ReturnsResultOfDelegate
Passed	RoomControllerTests+IndexGetTests	RoomDoesntHaveChildren_HasChildrenIsFalse
Passed	ProductTypeControllerTests+IndexGetTests	ActionCalled_ProductTypesAreSortedByCategoryThenByType
Passed	TemplateControllerTests+CreateTests	TemplateDoesNotYetExist_ResultHasTemplateId
Passed	InvolvedPartyControllerTests+IndexGetTests	ActionCalled_InvitationsAreAdded
Passed	DashboardControllerTests+IndexTests	ActionCalled_ConflictsAreAddedToViewModel
Passed	LocationControllerTests+ExportTests	ExportActionIsCalled_ResultContainsCorrectContentType
Passed	AuthenticationTests+GetUserIdFromAuthCookie	CookieDoesntHaveIntegerValue_ReturnsNull
Passed	TemplateControllerTests+UpdateTests	NameIsntNull_NameIsModified
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_BothCustomProductsAndCatalogProductsAreIncluded
Passed	InvolvedPartyControllerTests+IndexGetTests	LoggedInPersonIsntOwner_ShowInvitationsIsFalse
Passed	IdentityTests+PersonIdTests	PersonIdCalled_ReturnsPersonId
Passed	ConflictInvitationControllerTests+CreateTests	InvitationIsCreated_MessageIsSent
Passed	InvolvedPartyTests+AddRolesTests	AddOwnerRole_BothConflictAndTestConflictRolesAreAdded
Passed	ConflictBuilderFactoryTests+CreateConflictBuilderTests	NotRegisteredType_ThrowsInvalidOperationException
Passed	PrincipalTests+GetPrivilegeRights	IdentityHas_Result
Passed	ProductCategoryControllerTests+IndexPostTests	ModelStateIsInvalid_SaveIsntCalled
Passed	RoomInstanceControllerTests+GetCatalogProductsByRoomIdTests	MethodCalled_TwoCatalogProductsWithSameProductTypeBothAreAdded
Passed	AuthorizeAttributeTests	IdentityHasRight_ReturnsTrue
Passed	MockUnitOfWorkTests+TimesSaveHasBeenCalledTests	SaveCalled_TimesSaveHasBeenCalledIncremented
Passed	ProductCategoryControllerTests+IndexPostTests	ModelStateIsvalid_SaveIsCalled
Passed	AuthenticationTests+SignIn	SignInIsNotPersistent_ExpirationDateIsMinDateTimeValue
Passed	ConflictUtilitiesTests+ControllerNameTests+ConflictControllerTests	UnknownConflictType_ThrowsException
Passed	TemplateControllerTests+UpdateTests	NameIsNull_NameIsntModified
Passed	IdentityTests+AuthenticationTypeTests	AuthenticationTypeCalled_ReturnsCustomAuthenticationType
Passed	FilteredConflictRepositoryDecoratorTests+AddTests	AddIsCalledOnDecorator_AddIsCalledOnDecorated
Passed	ConflictTests+MoreInvolvedPartiesMayBeInvitedTests	MaxNumberOfPartiesIsTwoAndOnePartyAndOneInvitationAreAdded_DontShowAddPartyLink
Passed	AuthenticationTests	GetHashingStrategyValue
Passed	MockRepositoryTests	ItemIsAdded_EntityInArgsIsAddedEntity
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredCorrectData_UserIsAssignedUserRole
Passed	ConflictInvitationControllerTests+AcceptTests	PersonIsAlreadyAttachedToConflict_NotFoundViewIsReturned
Passed	CsvLocationExporterTests+ExportLocationsTests	NoLocationsInParameter_OnlyContainsHeadersLine
Passed	ProductCategoryControllerTests+IndexGetTests	ActionCalled_ListCategoriesOrderedByName
Passed	ConflictTests+MoreInvolvedPartiesMayBeInvitedTests	MaxNumberOfPartiesIsTwoAndOnePartyIsInvolved_ShowAddPartyLink
Passed	RoomInstanceControllerTests+DetailsPostTests	CatalogProductsAreMarkedForCreation_AddCatalogProducts
Passed	LocationControllerTests+IndexTests	IndexIsAccessed_TemplatesAreInserted
Passed	TemplateRoomInstanceControllerTests+DetailsPostTests	ProductsInCreate_AddedToTemplate
Passed	Sha512HashingStrategyTests	ValidateHashAgainstDifferentSource_ReturnsFalse
Passed	ProductCategoryControllerTests+IndexPostTests	ActionCalled_EmptyResultIsReturned
Passed	InvolvedPartyTests+AddRolesTests	AddInvolvedPartyRole_BothConflictAndTestConflictRolesAreAdded
Passed	AccountControllerTests+LogOnTests	UsernameAndPasswordAreCorrect_UserIsSignedIn
Passed	InvolvedPartyControllerTests+IndexGetTests	MaxNumberOfPartiesIsTwoAndTwoPartiesAreAlreadyInvolved_DontShowAddPartyLink
Passed	FilteredConflictRepositoryDecoratorTests+AddTests	AddIsCalledOnDecorator_ParameterIsPassed
Passed	ConflictDivisionOfPropertyTests+DetailsTest	ActionCalled_ConflictIdIsInViewModel
Passed	ProductTypeControllerTests+IndexPostTests	TypeAlreadyInDatabase_TypeIsntInserted
Passed	HtmlConflictExtensionsTests+ConflictLinkTests	DivisionOfConflictType_ReturnsLinkToDivisionOfPropertyController
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_ProductCategoriesAreAddedToViewModelOrderedByName
Passed	LocationControllerTests+CreateTests	ConflictIsntFound_EmptyResultIsReturned
Passed	ProductTypeControllerTests+IndexGetTests	ActionCalled_ProductTypesAreReturned
Passed	LocationControllerTests+CreateTests	ModelBindingFailed_NoExceptionIsThrown
Passed	ProductTypeControllerTests+IndexPostTests	ModelStateIsInvalid_SaveIsntCalled
Passed	ProductTypeControllerTests+IndexPostTests	ModelStateIsvalid_SaveIsCalled
Passed	AccountControllerTests+LogOnTests	UsernameUnknown_ReturnView
Passed	MenuControllerTests+RenderTopMenuTests	UserIsNotAdmin_DontShowAdminMenu
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredDoesntValidate_ShowView
Passed	TemplateControllerTests+UpdateTests	TemplateWithIdDoesntExist_RedirectToIndex
Passed	MenuControllerTests+RenderTopMenuTests	UserIsNotAuthenticated_ShowCreateConflictLinkIsFalse
Passed	LocationControllerTests+UpdatePostTests	ModelBindingFailed_NoExceptionIsThrown
Passed	PrincipalCreatorTest+CreateUnauthenticatedPrincipal	MethodCalled_ReturnsUnauthenticatedPrinciple
Passed	AuthorizeAttributeTests	IdentityHasRightButFailsAtPrevious_ReturnsFalse
Passed	Base64EncodingAlgorithmTests+DecodeToStringTests	KnownInputAndOutput_ExpectedOutputMatchesActualOutput
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_ProductsAreOrderedByCategoryThenByTypeThenByDescription
Passed	AccountControllerTests+LogOnTests	UsernameAndPasswordAreCorrect_Redirect
Passed	EmailInvitationSenderTests+CreateMailMessageTests	MailMessageCreated_BodyIsSet
Passed	TemplateControllerTests+UpdateTests	CreateIsGiven_RoomInstancesAreAdded
Passed	TemplateControllerTests+DeleteTests	DeleteIsCalledWithValidId_TemplateIsRemovedFromRepository
Passed	ConflictInvitationControllerTests+RevokeTests	InvitationIsFound_InvitationIsRemovedFromInvitationRepository
Passed	RoomInstanceControllerTests+DetailsPostTests	ActionCalled_SaveHasBeenCalled
Passed	PrincipalTests+IsInRole	IdentityHasRole_ReturnsTrue
Passed	ConflictControllerTests+CreateGetTests	ActionCalled_CatalogItemsAreReturned
Passed	TemplateControllerTests+DeleteTests	DeleteIsCalledWithValidId_SaveIsCalled
Passed	TemplateControllerTests+UpdateTests	UpdateIsGiven_RoomsAreUpdated
Passed	ConflictDivisionOfPropertyBuilderTests	OwnerIsNull_ArgumentNullExceptionIsThown
Passed	AuthenticationTests+GetUserIdFromAuthCookie	CookieValueExists_DecryptIsCalled
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	KnownUser_PersonIdIsSet

Passed	AuthenticationTests+GetUserIdFromAuthCookie	CookieDoesntHaveValue_ReturnsNull
Passed	HtmlEmailInvitationFormatterTests+FormatInvitationTests	FormatInvitationCalled_SenderIsIncluded
Passed	Sha512HashingStrategyTests	GenerateHashAndValidateAgainstOriginal_ReturnsTrue
Passed	RoomInstanceControllerTests+DetailsPostTests	CustomProductsMarkedForUpdate_UpdateCustomProducts
Passed	AuthenticationTests	AuthenticationConfigurationIsLoaded
Passed	ProductCategoryControllerTests+IndexPostTests	CategoryAlreadyInDatabase_CategoryIsntInserted
Passed	TemplateRoomInstanceControllerTests+DetailsGetTests	ActionCalled_AllProductTypesAreInserted
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_LocationsAreSortedByDescription
Passed	EmailInvitationSenderTests+CreateMailMessageTests	MailMessageCreated_ReceiverIsSet
Passed	InvolvedPartyControllerTests+IndexGetTests	MaxNumberOfPartiesIsTwoAndOnePartyIsInvolved_ShowAddPartyLink
Passed	ConflictDivisionOfPropertyBuilderTests	MethodCalled_ConflictOwnerRoleIsAssigned
Passed	TemplateRoomInstanceControllerTests+DetailsPostTests	ProductsInDelete_RemovedFromRepository
Passed	LocationControllerTests+DetailsTests	ActionIsCalled_ViewModelIsFilled
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredCorrectData_UserIsAddedToRepository
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredCorrectData_RedirectToDashboard
Passed	RoomInstanceControllerTests+DetailsPostTests	ModelBindingFailed_NoExceptionIsThrown
Passed	ProductCategoryControllerTests+IndexGetTests	CategoryHasChildren_HasChildrenIsTrue
Passed	MenuControllerTests+RenderTopMenuTests	UserIsAuthenticated_ShowDashboardLinkIsTrue
Passed	ConflictInvitationControllerTests+CreateTests	PersonIsntOwner_ExceptionIsThrown
Passed	TemplateControllerTests+CreateTests	NameIsWhitespace_ResultIsFalse
Passed	PrincipalTests+GetPrivilegeRights	IdentityDoesntHaveRight_ReturnsZero
Passed	FilteredConflictRepositoryDecoratorTests+RemoveTests	RemoveIsCalledOnDecorator_ParameterIsPassed
Passed	CsvLocationExporterTests+ExportLocationsTests	FileIsBuilt_HeaderColumnsAreCorrect
Passed	ConflictDivisionOfPropertyBuilderTests	MethodCalled_ConflictDivisionOfPropertyOwnerRoleIsAssigned
Passed	RoomInstanceTests+GetDescriptionTests	DescriptionIsNull_OnlyNameIsReturned
Passed	DashboardControllerTests+IndexTests	ActionCalled_ConflictsAreSortedByCreationDate
Passed	AccountControllerTests+LogOnTests	PasswordIsIncorrect_ReturnView
Passed	FilteredConflictRepositoryDecoratorTests+QueryTests	UserIsAnInvolvedPartyInConflict_ConflictIsInRepository
Passed	ConflictInvitationControllerTests+CreateTests	InvalidEmailAddress_ViewIsReturned
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_LineForEachProduct
Passed	AccountControllerTests+LogOffTests	ActionCalled_RedirectedToLogOn
Passed	TemplateControllerTests+UpdateTests	DuplicateRoomsAreInserted_TwoRoomInstancesAreAdded
Passed	ConflictInvitationControllerTests+RevokeTests	InvitationIsFound_SaveIsCalled
Passed	LocationControllerTests+CreateTests	TemplateIdIsGiven_ProductsAreInsertedFromTemplate
Passed	InvolvedPartyControllerTests+IndexGetTests	ActionCalled_InvitationsAreSortedByAddress
Passed	ProductCategoryControllerTests+IndexPostTests	CategoryAlreadyInDatabase_CategoryIsntUpdated
Passed	NoEncryptionStrategyTests	EncryptIsCalled_ReturnsInput
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_RedirectedToDetails
Passed	ConflictInvitationControllerTests+CreateTests	ConflictHasMaxParties_ViewIsReturned
Passed	RoomInstanceControllerTests+DetailsPostTests	ProductsAreMarkedForDeletion_DeleteProductsFromRepository
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserIsKnown_RedirectToDashboard
Passed	RoomInstanceControllerTests+DetailsGetTests	ActionCalled_CatalogProductsAreAddedToViewModel
Passed	ConflictInvitationControllerTests+CreateTests	ConflictAlreadyHasInvitationForAddress_ViewIsReturned
Passed	InvolvedPartyControllerTests+IndexGetTests	MaxNumberOfPartiesIsInfinite_ShowAddPartyLink
Passed	ProductTypeControllerTests+IndexPostTests	DeleteInViewModel_DeletedFromRepository
Passed	MenuControllerTests+RenderTopMenuTests	UserIsAuthenticated_ShowCreateConflictLinkIsTrue
Passed	TemplateControllerTests+CreateTests	TemplateAlreadyExists_ErrorIsAdded
Passed	NoEncryptionStrategyTests	DecryptIsCalled_ReturnsInput
Passed	RoomControllerTests+IndexPostTests	ModelStateIsInvalid_SaveIsntCalled
Passed	Sha512HashingAlgorithmTests+CreateHashTests	KnownInput_OutputEqualsExpectedOutput
Passed	ControllerTests+GetPersonIdTests	GetPersonIdIsCalled_ReturnsPersonIdFromIdentity
Passed	ConflictInvitationControllerTests+CreateTests	InvitationIsCreated_SaveIsCalled
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	UnknownUser_ReturnsUnauthenticatedPrincipal
Passed	Sha512HashingStrategyTests	KnownInputAndOutput_HasherOutputShouldCorrespond
Passed	ProductCategoryControllerTests+IndexGetTests	CategoryDoesntHaveChildren_HasChildrenIsFalse
Passed	AccountControllerTests+IsUsernameAvailableTests	UsernameInRepository_ReturnFalse
Passed	ProductCategoryControllerTests+IndexPostTests	UpdateInViewModel_UpdatedInRepository
Passed	ProductTypeControllerTests+IndexGetTests	ActionCalled_CategoriesAreInsertedInViewModel
Passed	ConflictInvitationControllerTests+AcceptTests	InvitationIsFoundAndTokenMatches_InvitationIsRemovedFromRepository
Passed	InvolvedPartyControllerTests+IndexGetTests	MaxNumberOfPartiesIsTwoAndOnePartyAndOneInvitationAreAdded_DontShowAddPartyLink
Passed	ProductTypeControllerTests+IndexPostTests	UpdateInViewModel_RoomsAreUpdated
Passed	InvolvedPartyTests+AddRolesTests	AddInvolvedPartyRole_RoleWithUnknownContextIsntAdded
Passed	IEnumerableExtensionsTests+IsNullOrEmptyTests	IEnumerableIsNull_ReturnsTrue
Passed	ConflictInvitationControllerTests+RevokeTests	InvitationIsntFound_SaveIsntCalled
Passed	RoomInstanceControllerTests+DetailsGetTests	RoomInstanceIsntFound_RedirectToConflictIndex
Passed	AuthenticationTests	GetEncryptionStrategyValue
Passed	RoomInstanceControllerTests+DetailsPostTests	CatalogProductsAreMarkedForUpdateButArentInRoomInstance_DontUpdateProducts
Passed	ConflictInvitationControllerTests+RevokeTests	InvitationIsntFound_NotFoundViewIsReturned
Passed	RoomControllerTests+IndexGetTests	RoomHasChildren_HasChildrenIsTrue
Passed	ConflictDivisionOfPropertyBuilderTests	MethodCalled_ConflictInvolvedPartyRoleIsAssigned
Passed	MockRepositoryTests	ItemIsAdded_EventIsFired
Passed	EmailInvitationSenderTests+CreateMailMessageTests	MailMessageCreated_SenderIsSet
Passed	AccountControllerTests+UserWidgetTests	UserIsntAuthenticated_ReturnsFalse
Passed	RoomInstanceControllerTests+DetailsPostTests	CatalogProductsMarkedForCreation_CatalogProductDoesntExist_ProductIsntAdded
Passed	ConflictInvitationControllerTests+AcceptTests	InvitationIsFoundAndTokenMatches_RedirectedToConflict
Passed	LocationControllerTests+IndexTests	IndexIsAccessed_TemplatesAreSortedByName
Passed	IEnumerableExtensionsTests+IsNullOrEmptyTests	IEnumerableIsNotNullButIsEmpty_ReturnsTrue
Passed	ConflictInvitationControllerTests+AcceptTests	InvitationIsFoundAndTokenMatches_UserIsAddedToInvolvedParties
Passed	RoomControllerTests+IndexPostTests	DeleteInViewModel_DeletedFromRepository
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredCorrectData_RegistrationDateIsSet
Passed	RoomInstanceControllerTests+DetailsPostTests	CustomProductsMarkedForUpdateButProductCategoryIdIsntInRepository_SkipCustomProduct
Passed	RoomControllerTests+IndexPostTests	RoomNameAlreadyInDatabase_RoomIsntUpdated

Passed	InvolvedPartyControllerTests+IndexGetTests	ActionCalled_ConflictIdIsAdded
Passed	ConflictInvitationControllerTests+AcceptTests	InvitationIsFoundAndTokenMatches_SaveIsCalled
Passed	AuthenticationTests+GetUserIdFromAuthCookie	CookieDoesntExist_ReturnsNull
Passed	LocationControllerTests+ExportTests	ExportActionIsCalled_ResultContainsCorrectFilename
Passed	RoomInstanceControllerTests+DetailsPostTests	CustomProductsMarkedForCreation_AddCustomProducts
Passed	TemplateControllerTests+UpdateTests	RoomNotInRepository_RoomInstanceNotCreated
Passed	RoomControllerTests+IndexPostTests	CreateInViewModel_AddedToRepository
Passed	LocationControllerTests+IndexTests	IndexIsAccessed_RoomsAreSortedByRoomName
Passed	MockRepositoryTests	ItemIsRemoved_EntityInArgsIsRemovedEntity
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	KnownUser_UserIdIsSet
Passed	DelegateEqualityComparerTests+EqualsTests	EqualsIsCalled_ParametersArePassedCorrectly
Passed	IdentityTests+RolesTests	RolesCalled_ReturnsRoles
Passed	TemplateRoomInstanceControllerTests+DetailsPostTests	UnknownProductInCreate_ProductIsntAdded
Passed	TemplateControllerTests+IndexTests	ActionCalled_TemplatesAreSortedByName
Passed	RoomInstanceControllerTests+GetCatalogProductsByRoomIdTests	MethodCalled_CatalogProductsAreAddedToViewModel
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_RoomsMarkedForCreateAreInserted
Passed	MenuControllerTests+RenderTopMenuTests	UserIsAdmin_ShowAdminMenu
Passed	IdentityTests+NameTests	NameCalled_ReturnsName
Passed	ConflictControllerTests+CreateGetTests	ActionCalled_CatalogItemsAreSortedByName
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_RoomMarkedForUpdate_DescriptionIsUpdated
Passed	InvolvedPartyControllerTests+ListTests	ActionCalled_InvolvedPartiesAreAddedToViewModel
Passed	HtmlEmailInvitationFormatterTests+FormatInvitationTests	FormatInvitationCalled_UrlIsInserted
Passed	Base64EncodingAlgorithmTests+EncodeFromByteArrayTests	KnownInputAndOutput_ExpectedOutputMatchesActualOutput
Passed	ConflictTests+MoreInvolvedPartiesMayBeInvitedTests	MaxNumberOfPartiesIsInfinite_ShowAddPartyLink
Passed	RoomInstanceControllerTests+DetailsPostTests	CatalogProductsAreMarkedForUpdate_UpdateProducts
Passed	MockRepositoryTests	ItemIsNotRemoved_EventIsntFired
Passed	LocationControllerTests+CreateTests	FormIsFilledOut_LocationIdIsReturned
Passed	Base64EncodingAlgorithmTests+DecodeToByteArrayTests	KnownInputAndOutput_ExpectedOutputMatchesActualOutput
Passed	LocationControllerTests+CreateTests	FormIsFilledOut_DescriptionIsAdded
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserIsUnknown_ShowView
Passed	AccountControllerTests+UserWidgetTests	UserIsAuthenticated_ReturnsTrue
Passed	LocationControllerTests+CreateTests	TemplateIdIsGiven_RoomInstancesAreInsertedFromTemplate
Passed	TemplateRoomInstanceControllerTests+DetailsGetTests	ActionCalled_OnlyProductTypesFromThisRoomAreInserted
Passed	FilteredConflictRepositoryDecoratorTests+RemoveTests	RemoveIsCalledOnDecorator_RemoveIsCalledOnDecorated
Passed	DelegateEqualityComparerTests+EqualsTests	EqualsIsCalled_ComparerReturnsDelegateResult
Passed	LocationExporterFactoryTests+GetLocationExporterTests	AskForCsvExporter_CsvExporterIsReturned
Passed	LocationControllerTests+UpdateGetTests	ActionCalled_LocationIsReturned
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserEnteredCorrectData_SaveMethodIsCalled
Passed	ProductTypeControllerTests+IndexPostTests	UpdateInViewModel_UpdatedInRepository
Passed	TemplateControllerTests+UpdateGetTests	ActionCalled_RoomsAreAdded
Passed	RoomInstanceControllerTests+DetailsPostTests	CustomProductsMarkedForUpdateButProductIsntInRoomInstance_SkipCustomProduct
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	KnownUser_ReturnsAuthenticatedPrincipal
Passed	ConflictDivisionOfPropertyBuilderTests	MethodCalled_ConflictDivisionOfPropertyInvolvedPartyRoleIsAssigned
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UsernameIsTaken_AddError
Passed	LocationControllerTests+UpdatePostTests	FormIsSubmitted_RoomsMarkedForDeletionAreDeleted
Passed	LocationControllerTests+CreateTests	FormIsFilledOut_RoomsAreAddedToLocation
Passed	AccountControllerTests+RegisterWithUsernameAndPasswordTests	UserIsAlreadyAuthenticated_RedirectToDashboard
Passed	LocationControllerTests+IndexTests	IndexIsAccessed_LocationsAreSortedByDescription
Passed	AuthenticationTests+GetUserIdFromAuthCookie	CookieValueExists_ReturnsValue
Passed	CsvLocationExporterTests+ExportLocationsTests	ValidInputIsGiven_ContentTypeIsCsv
Passed	ConflictControllerTests+CreatePostTests	ConflictIdIsGiven_OwnerIsPassedToBuilder
Passed	InvolvedPartyControllerTests+ListTests	ActionCalled_InvolvedPartiesAreSortedByName
Passed	LocationControllerTests+IndexTests	IndexIsAccessed_RoomsAreSortedByRoomNameThenByDescription
Passed	TemplateControllerTests+DeleteTests	DeleteIsCalledWithNonExistingId_ViewDataIsTrue
Passed	DelegateEqualityComparerTests+GetHashCodeTests	GetHashCodeIsCalled_DelegateIsCalled
Passed	LocationControllerTests+CreateTests	FormIsFilledOut_NewLocationIsAddedToConflict
Passed	DetailsTests+GetDescriptionTests	DescriptionIsNotNull_RoomAndDescriptionAreReturnedSeparatedByDash
Passed	RoomInstanceTests+GetDescriptionTests	RoomIsNull_ThrowsNullReferenceException
Passed	ProductTypeControllerTests+IndexPostTests	ActionCalled_EmptyResultIsReturned
Passed	InvolvedPartyControllerTests+IndexGetTests	LoggedInPersonIsntOwner_DontAddInvitations
Passed	InvolvedPartyTests+AddRolesTests	AddBothInvolvedRoleAndOwnerRole_AllRolesAreAdded
Passed	MockRepositoryTests	ItemIsRemoved_EventIsFired
Passed	InvolvedPartyControllerTests+IndexGetTests	LoggedInPersonIsOwner_ShowInvitationsIsTrue
Passed	InvolvedPartyControllerTests+ListTests	ActionCalled_LoggedInPersonHasLoggedInBoolean
Passed	RoomControllerTests+IndexPostTests	RoomAlreadyInDatabase_RoomIsntInserted
Passed	TemplateRoomInstanceControllerTests+DetailsGetTests	TemplateIdUnknown_RedirectedToIndex
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_RoomsAreSortedByRoomTypeThenByDescription
Passed	TemplateControllerTests+CreateTests	TemplateDoesNotYetExist_TemplateIsAddedToRepository
Passed	ConflictDivisionOfPropertyTests+GetTypeTests	GetMaximumInvolvedParties_ReturnsTwo
Passed	AuthenticationTests+SignOut	CookieExists_ExistingCookieIsDeletedAndValuelessCookieIsInserted
Passed	ConflictInvitationControllerTests+CreateTests	MailerThrowsException_SaveIsntCalled
Passed	TemplateControllerTests+CreateTests	TemplateIsInserted_SaveHasBeenCalled
Passed	RoomInstanceControllerTests+DetailsPostTests	ProductsAreMarkedForDeletionButArentInRoomInstance_DontDeleteFromRepository
Passed	ProductTypeControllerTests+IndexGetTests	ProductTypeInCatalogProduct_HasChildrenIsTrue
Passed	ConflictTests+MoreInvolvedPartiesMayBeInvitedTests	MaxNumberOfPartiesIsTwoAndTwoPartiesAreAlreadyInvolved_DontShowAddPartyLink
Passed	ConflictControllerTests+CreatePostTests	ConflictIdIsGiven_EmptyConflictIsAddedToRepository
Passed	IdentityTests+PrivilegesTests	PrivilegesCalled_ReturnsPrivileges
Passed	ConflictUtilitiesTests+ControllerNameTests+ConflictControllerTests	DivisionOfConflictType_Returns
Passed	LocationExporterFactoryTests+GetLocationExporterTests	AskForAnotherInstance_InstancesArentEqual
Passed	RoomInstanceControllerTests+GetCatalogProductsByRoomIdTests	MethodCalled_OnlyCatalogProductsInRoomAreAdded
Passed	TemplateControllerTests+IndexTests	ActionCalled_TemplatesAreAddedToViewModel
Passed	ConflictInvitationControllerTests+RevokeTests	RevokeCalled_RedirectToInvolvedPartyController

Passed	ConflictInvitationControllerTests+CreateTests	InvitationIsCreated_InvitationIsAddedToConflict
Passed	ConflictControllerTests+CreatePostTests	ConflictIdIsGiven_BuildConflictIsCalled
Passed	LocationControllerTests+IndexTests	IndexIsAccessed_ConflictIdIsInserted
Passed	ConflictBuilderFactoryTests+CreateConflictBuilderTests	ConflictDivisionOfPropertyAsType_ReturnsConflictDivisionOfPropertyBuilder
Passed	AuthenticationTests+SignIn	SignInCalled_CookieIsAdded
Passed	ProductCategoryControllerTests+IndexPostTests	DeleteInViewModel_DeletedFromRepository
Passed	PrincipalCreatorTest+CreateAuthenticatedPrincipal	KnownUser_AllPrivilegesAreAdded
Passed	CsvLocationExporterTests+ExportLocationsTests	LocationsArePassed_EachLocationIsIncluded
Passed	LocationControllerTests+ExportTests	LocationIdIsNull_AllLocationsAreSelected
Passed	RoomInstanceControllerTests+DetailsPostTests	RoomInstanceIsntFound_RedirectToConflictIndex
Passed	RoomInstanceControllerTests+DetailsPostTests	ActionCalled_RedirectedToDetailsView
Passed	RoomControllerTests+IndexPostTests	ModelStateIsvalid_SaveIsCalled
Passed	AuthorizeAttributeTests	IdentityDoesntHaveRight_ReturnsFalse
Passed	TemplateControllerTests+UpdateGetTests	ActionCalled_RoomInstancesAreAdded
Passed	ConflictControllerTests+CreatePostTests	ConflictIsAdded_RedirectToDashboard
Passed	ProductTypeControllerTests+IndexPostTests	CreateInViewModel_RoomsAreAdded
Passed	TemplateControllerTests+UpdateTests	UnknownRoomIsGiven_NotUpdated

Code coverage

Projecten

Project	Not covered	Covered
IMove2U.Business.Creational.dll	0.00%	100.00%
IMove2U.Business.Domain.dll	0.00%	100.00%
IMove2U.Business.Export.dll	0.00%	100.00%
IMove2U.DataAccess.dll	0.00%	100.00%
IMove2U.Tests.Shared.dll	0.00%	100.00%
IMove2U.WebUI.dll	10.93%	89.07%

Namespaces in project 'IMove2U.WebUI'

Namespace	Not covered	Covered
IMove2U.WebUI	100.00%	0.00%
IMove2U.WebUI.Attributes	0.00%	100.00%
IMove2U.WebUI.Attributes.ValidationAttributes	100.00%	0.00%
IMove2U.WebUI.Authentication	17.17%	82.83%
IMove2U.WebUI.Authentication.Encryption	25.00%	75.00%
IMove2U.WebUI.Authentication.Hashing	40.91%	59.09%
IMove2U.WebUI.Code	0.00%	100.00%
IMove2U.WebUI.Configuration	40.00%	60.00%
IMove2U.WebUI.Controllers	11.57%	88.43%
IMove2U.WebUI.Controllers.DivisionOfProperty	1.45%	98.55%
IMove2U.WebUI.Encoding	0.00%	100.00%
IMove2U.WebUI.Extensions	0.00%	100.00%
IMove2U.WebUI.Hashing	0.00%	100.00%
IMove2U.WebUI.InvitationSender	38.46%	61.54%
IMove2U.WebUI.Models.DivisionOfProperty.Room	0.00%	100.00%
IMove2U.WebUI.Unity	100.00%	0.00%
IMove2U.WebUI.Utilities	0.00%	100.00%

Bijlage D: Scrum backlogs en burndowns

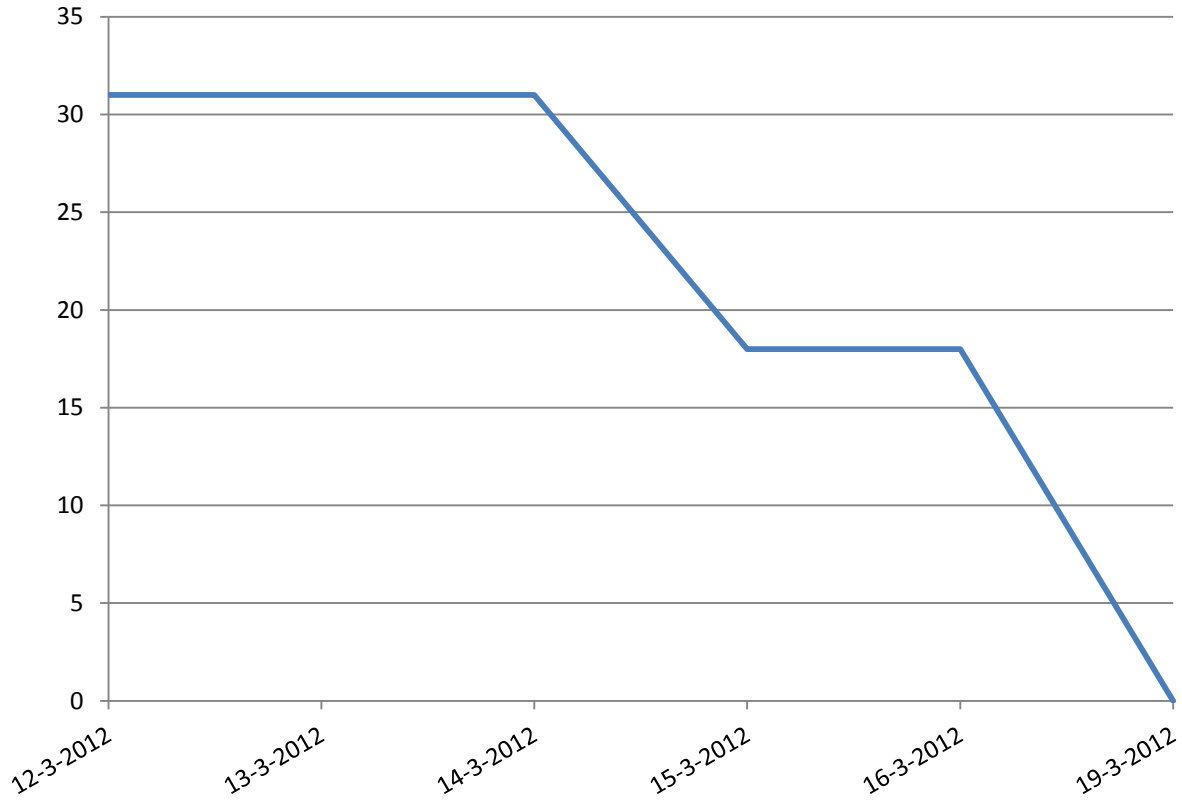
Product backlog

Subject	User story	Status	Story points	Sprint
Application	Initial setup of the project	Done	13	1
Application	As a user, I want to be able to log in	Done	13	1
Application	As a user, I want to be able to register	Done	5	1
Division of property	As an involved party, I want to be able to add a location, and separate it into rooms.	Done	8	3
Division of property	As an involved party, I want to be able to add products from the catalog to a room.	Done	13	3 / 4
Division of property	As an involved party, I want to be able to add custom products to a room.	Done	8	3
Division of property	As an involved party, I want to be able to modify the description and price range of a product added to a room from the catalog.	Done	3	3 / 4
Division of property	As an involved party, I want to be able to modify the description, name and price of a custom product added to a room.	Done	3	3
Division of property	As an involved party, I want to be able to remove a product from the product list.	Done	3	3
Division of property	As an involved party, I want to be able to download the product list so that I can save it, print it or email it.	Done	5	4
Division of property	As the system owner, I want to be able to manage the rooms.	Done	3	4
Division of property	As the system owner, I want to be able to manage the product categories in the catalog	Done	3	4
Division of property	As the system owner, I want to be able to manage the product types in the catalog.	Done	8	4
Division of property	As the system owner, I want to be able to create, modify and delete a template.	Done	13	4

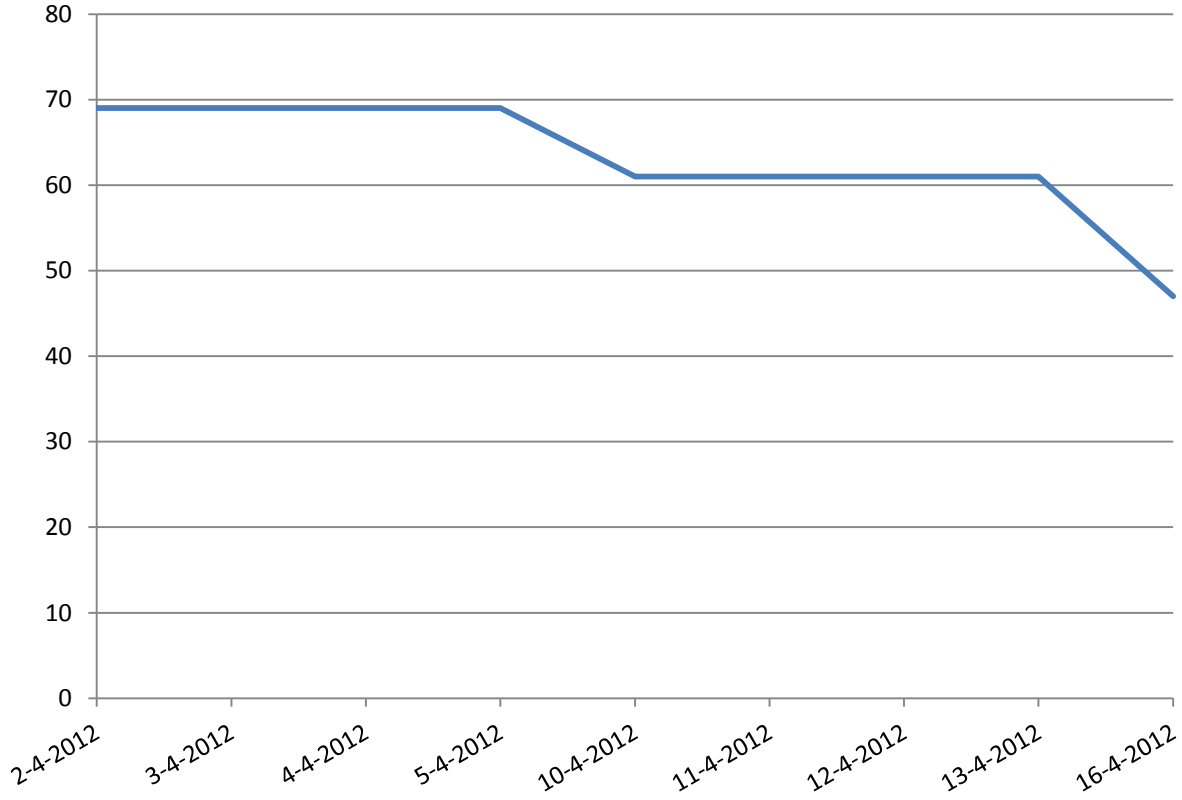
Division of property	As an involved party, I want to be able to automatically generate a location and fill it with products based on a template.	Done	5	4 / 5
Conflicts in general	As the owner of the conflict, I want to be able to add one or multiple involved parties, based on the type of conflict.	Done	13	5
Conflicts in general	As the system owner, I only want users to be able to view conflicts that they are part of	Done	13	5
Application	As a user, I want to be able to initiate a conflict.	Done	13	5
Application	As a user, I want to be able to see all the conflicts that I am a part of.	Done	3	5
Application	As the system owner, I want that the permissions that the user has are defined by the role that they have in both the application and the conflict.	Done	8	5
Application	As the system owner, I want the menu to change according to the roles and rights that the user has, so that only relevalt links are shown	Done	8	5

Burndown charts

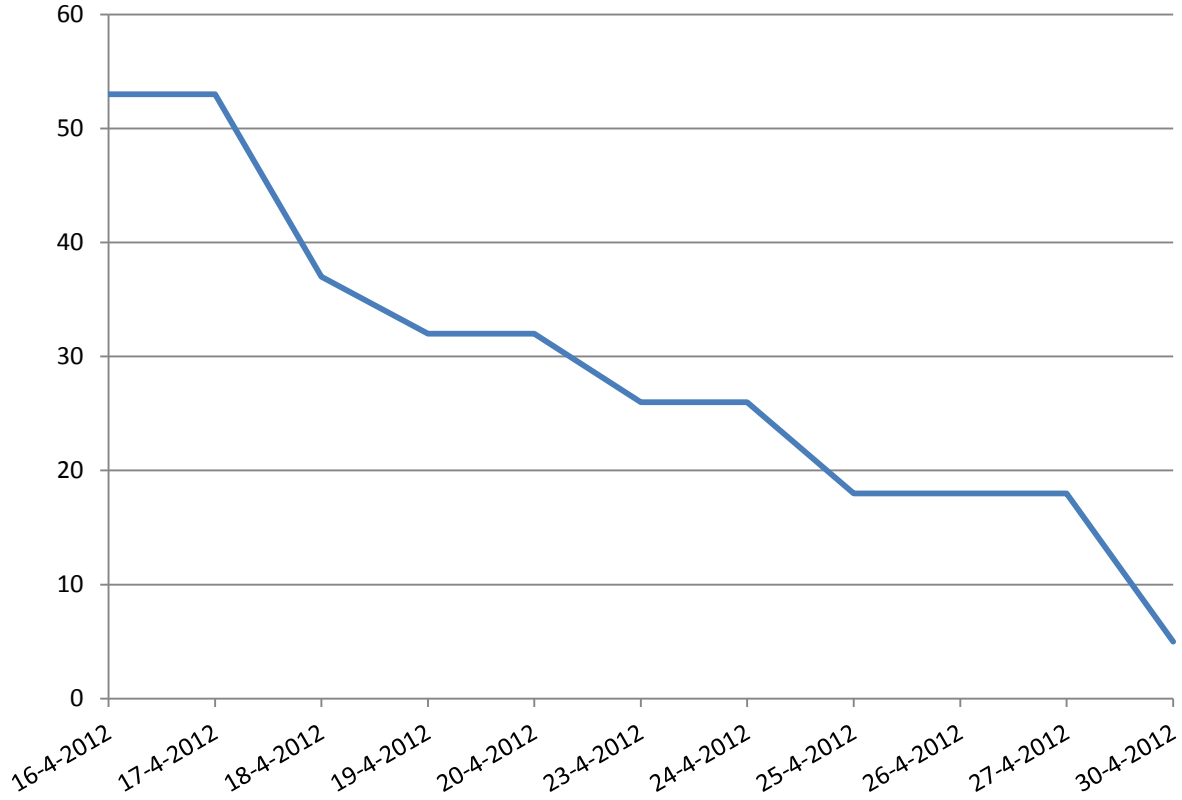
Burndown sprint 1



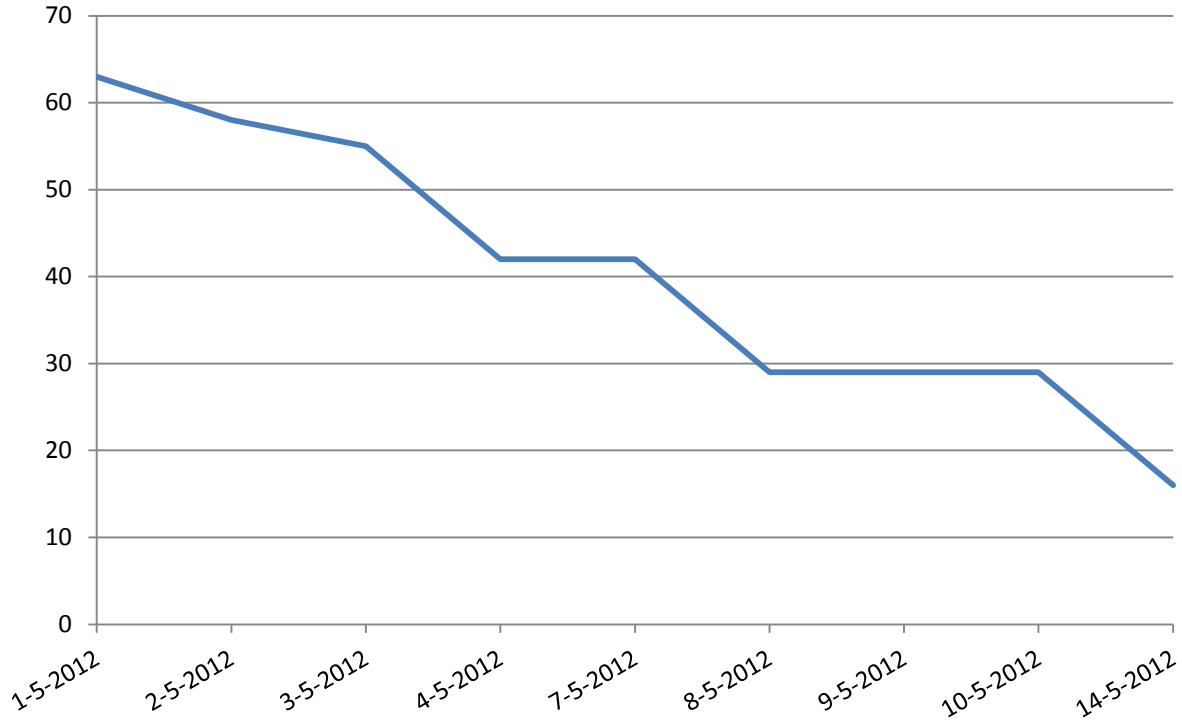
Burndown sprint 3



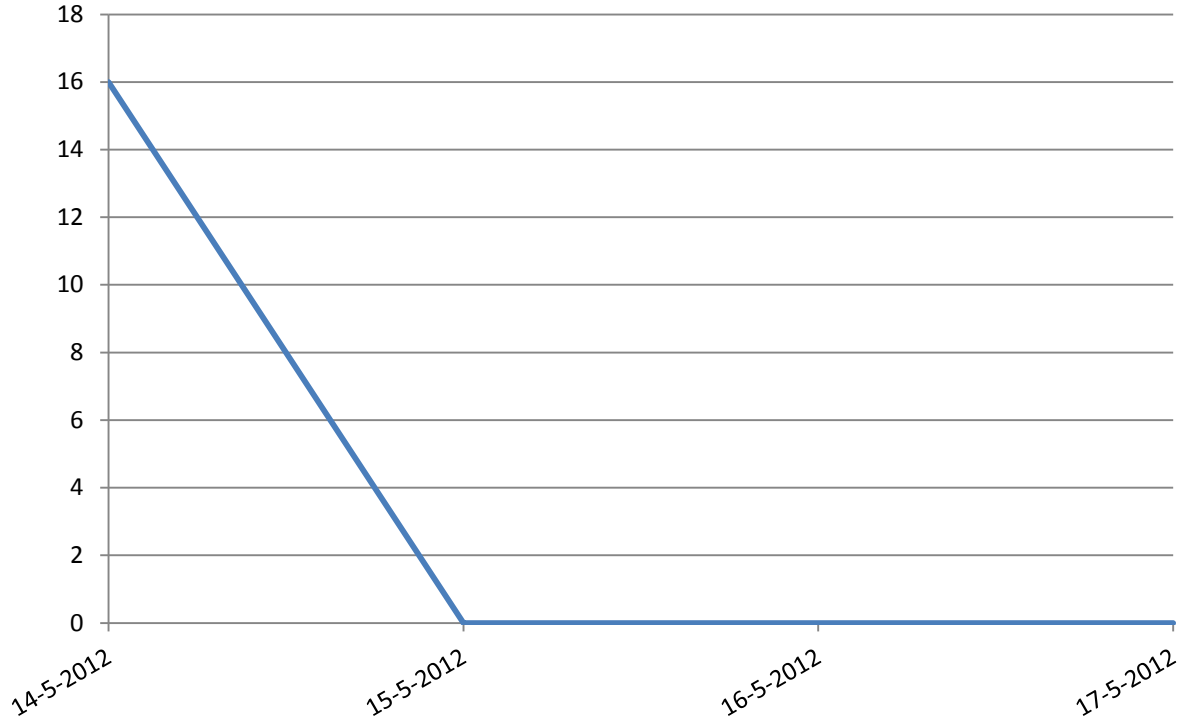
Burndown sprint 4



Burndown sprint 5



Burndown sprint 6



Bijlage E: Formulier tussentijds assessment

Student: Jesse van Assen **Studentnummer:** 08014353

Datum: 11 mei 2012 **eerste / tweede TTA:** eerste

Tijdens het tussentijds assessment is het volgende geconstateerd:		**Ja**	**Nee**
a	Het voortgangsverslag is ontvangen	x	
b	Het afstudeerdossier is digitaal beschikbaar	x	
c	Het afstudeerdossier is opgebouwd conform de richtlijnen	x	
d	Het goedgekeurde afstudeerplan is aanwezig	x	
e	Het plan van aanpak is aanwezig	x	
f	Reeds geleverd commentaar is aanwezig	x	
g	Het afstudeerdossier geeft voldoende inzicht in de stand van zaken	x	
h	De afstudeeropdracht is tot nu toe naar behoren uitgevoerd	x	

Aanpak	**O**	**T**	**V**	**G**
Passend			x	
Theoretisch verantwoord			x	
Samenhang uitvoering beroepstaken			x	

Beroepstaken op afgesproken niveau uitgevoerd?		**O**	**T**	**V**	**G**
1	Uitvoeren analyse door definitie van requirements		x		
2	Opstellen gegevensmodel voor een database			x	
3	Ontwerpen, bouwen en bevragen van een database			x	
4	Ontwerpen systeemdeel			x	
5	Bouwen applicatie			x	
6	Uitvoeren van en rapporteren over het testproces			x	

Producten	O	T	V	G
Tussenproducten			x	
Eindproducten			x	

Effectief communiceren	O	T	V	G
Binnen afstudeerbedrijf			x	
Afstudeerdossier			x	

Reflectie	O	T	V	G
Inzicht in eigen functioneren			x	
Inzicht in eigen leerproces			x	

14.4 Toelichting per beoordelingscriterium

Aanpak
Student heeft gegeven het project een aanpak beschreven. De verdeling in een sprints en het verloop daarvan. Op een aantal punten kan het nog verder uitgewerkt en verbeterd worden. Het verslag zou wel iets compacter mogen.

Beroepstaken op afgesproken niveau uitgevoerd?
Met name de wijze waarop de requirements tot stand zijn gekomen behoeft aandacht. De overige lijken redelijk uitgevoerd. Het database model is op gezet en er is een applicatie ontwikkeld die gebruik maakt van deze database. Verder is het systeem ontworpen en zijn er delen gerealiseerd. Daarnaast is het testen uitgevoerd en een code coverage.

Producten
De producten die geleverd zijn lijken van voldoende kwaliteit.

Effectief communiceren
Het verslag is goed leesbaar. Het zou iets compacter mogen. Verder communiceert Jesse op een goede manier met de examinatoren.

Reflectie
De evaluatiehoofdstukken lijken voldoende alleen zou het goed zijn om nog eens goed naar de rol van de opdrachtgever te kijken.

14.5 Advies

x	**Positief** (bindend advies)
	Verlengen (vrijblijvend advies)
	Negatief (vrijblijvend advies)

14.6 Besluit student

Aankruisen welke beslissing de student heeft genomen (alleen na vrijblijvend advies)

x	**Afstudeerdossier wordt op afgesproken datum ingeleverd**	Inleverdatum: 1 juni 2012
	Afstudeerperiode wordt verlengd	Inleverdatum:
	Student stopt met afstudeeropdracht	

Naam begeleidend examinator: Tim Cocx

Naam tweede examinator: Nanny Jacobs

Datum: 24 mei 2012