UNIT4 SOFTWARE B.V.

Afstudeerscriptie

Fabric web applicatie

Raymond Meerburg

9/23/2012

Student	:	Raymond Meerburg
Studentnummer	:	08056757
Opdrachtgever	:	UNIT4 Software B.V.
Bedrijfsmentoren	:	Dhr. Y. Nijsen Dhr. K. Klumpers
School	:	Haagse Hogeschool, Academie voor ICT & Media te Delft
Opleiding	:	Technische Informatica
Periode	:	2012-1.2
Examinatoren	:	Dhr. J. Visser Dhr. H. van den Bosch
Datum	:	23 september 2012
Versie	:	1.0 (definitief)

REVISIE GESCHIEDENIS

Datum	Opmerking	Versienummer
23-04-2012	Initiële versie	0.1
11-07-2012	Inleiding, aanpak, opdrachtdefinitie, eerste drie sprints, opmaak	0.7 1e concept
26-08-2012	Indeling aangepast, algemene verbeteringen, sprints aangevuld, evaluatie	0.8 2e concept
23-09-2012	Meer algemene verbeteringen, alle sprints aangevuld, Fabric rapport opgenomen, bijlagen in orde gemaakt.	1.0

MANAGEMENT SAMENVATTING

UNIT4 Software B.V., onderdeel van de UNIT4 holding, maakt een boekhoudpakket gericht op het MKB. Het standaard pakket is niet altijd toereikend, soms is er maatwerk nodig voordat een klant het kan gebruiken. Dit maatwerk wordt uitgevoerd door partners van UNIT4, deze verkopen het aangepaste pakket dan aan klanten. Om partners in staat te stellen dit maatwerk uit te voeren wordt er een framework aangeboden dat in Visual Basic is geschreven en dat aan vernieuwing toe is. De opvolger hiervan is Fabric, deze moet het voor partners een stuk makkelijker maken om hun maatwerk te leveren, bovendien kan Fabric ook andere dingen die in het oude framework niet mogelijk zijn.

Fabric is na anderhalf jaar ontwikkeling in een stabiel stadium. Het is nu de wens van UNIT4 om het framework in te zetten voor een concept-applicatie. De ontwikkeling van dit concept moet mogelijke problemen met Fabric aan het licht brengen. Niet alleen is het belangrijk dat er fouten worden opgespoord, ook moet het werken met Fabric vlot verlopen. Obstakels of irritaties moeten aangemerkt worden. De uitvoering van deze opdracht kan het beste gebeuren door een ontwikkelaar die nog niet bekend is met Fabric want deze zal dezelfde problemen tegenkomen als partners later tegen zullen komen.

Het doel van deze concept-applicatie is niet alleen om Fabric te verfijnen. De applicatie moet ook op twee manieren kunnen dienen als demo: de applicatie zelf moet partijen geïnteresseerd kunnen krijgen in het werken met Fabric. Daarnaast moet de source code van het project kunnen dienen als voorbeeld voor andere ontwikkelaars.

De aard van de applicatie wordt een eenvoudig order verwerking en CRM systeem. Het moet hiermee mogelijk zijn om klanten en leveranciers aan te maken en hiervoor orders op te stellen. Alle aangemaakte objecten moeten ook gewijzigd en verwijderd kunnen worden. De applicatie moet goed kunnen schalen, dat wil zeggen dat het om moet kunnen gaan met grote datasets en meerdere gebruikers.

Aan het begin van dit project zijn een groot deel van de te gebruiken technologieën, methoden en frameworks uitgekozen. Er is een gedetailleerde werkwijze opgesteld en een globale planning.

Een initieel concept waarin klanten en leveranciers aangemaakt en gewijzigd kunnen worden wordt goed ontvangen. Deze functionaliteit wordt uitgebreid en verfijnd door invoervalidatie toe te passen en zoeken mogelijk te maken. Een order scherm breidt dit uit. Hiermee is een lijst orders op te vragen en kan van elke order de details bekeken worden. Er worden uitgebreidere zoekcriteria toegevoegd zodat orders nauwkeurig gefilterd kunnen worden. Fabric heeft nog problemen met het leveren van data, dit gebeurt soms op een onhandige manier en de performance van Fabric is verbeterd. De efficiëntie van het framework is daardoor ver vooruitgegaan.

Door het systeem op een modulaire manier te ontwerpen is het nu zeer gemakkelijk om additionele schermen aan de applicatie toe te voegen.

Om de performance van de applicatie verder te verbeteren is er een caching systeem toegevoegd. Deze moet een zo groot mogelijk deel van het werk van de webserver overnemen. Hierdoor reageert de applicatie nog beter.

Een bevindingenrapport over Fabric brengt alle gevonden problemen in kaart en toont ook de onhandigheden. Met dit rapport kan Fabric op meer punten worden verbeterd waardoor het makkelijker en intuïtiever is om mee te werken.

De uiteindelijke applicatie geeft de mogelijkheden van Fabric goed weer. Daarnaast kan het door de duidelijke documentatie door externe ontwikkelaars worden gebruikt als voorbeeld voor het programmeren voor Fabric.

REFERAAT

Afstudeerverslag van Raymond Meerburg. Bouwen van een web applicatie t.b.v. het terugkoppelen van problemen met een nieuw framework.

Dit verslag beschrijft het proces en de activiteiten die plaats hebben gevonden in de periode van 23 april 2012 tot 3 september 2012 in het kader van bovengenoemde afstudeeropdracht. De opdracht is uitgevoerd door een student Technische Informatica aan De Haagse Hogeschool te Delft.

De opdracht betreft het ontwikkelen van een concept web applicatie op een nieuw framework. Problemen die optreden m.b.t. het nieuwe framework moeten teruggekoppeld worden, en indien mogelijk, zelf opgelost worden. Het concept moet later inzetbaar zijn als demo voor geïnteresseerden en de code als voorbeeld en referentiemateriaal voor andere ontwikkelaars die het framework gaan gebruiken.

Descriptoren:

- .NET Framework
- Afstudeeropdracht
- Agile
- ASP .Net MVC
- C#
- Caching
- HHS
- HTML
- JavaScript
- LINQ
- Model-View-Controller
- Performance
- Scrum
- UML
- Unit testing
- Visual Studio 2010

COLOFON

Afstudeerder

Naam	Raymond Meerburg
Straat / Nummer	IJsvogel 8
Postcode / Plaats	3362 KC, Sliedrecht
Telefoonnummer	06 11 444 701
E-mailadres	rmeerburg@gmail.com; Raymond.Meerburg@unit4.com
Onderwijsinstelling	Haagse Hogeschool
Afstudeerrichting	Technische Informatica
Studentnummer	08056757

Eerste examinator

Naam	J.J. Visser
E-mailadres	j.j.visser@hhs.nl
Telefoonnummer	015-2606281

Tweede examinator

Naam	H.C. van den Bosch
E-mailadres	h.c.vandenbosch@hhs.nl
Telefoonnummer	-

Bedrijfsgegevens

Naam	UNIT4 Software B.V.
Straat / Nummer	Stationsweg 1000
Postcode / Plaats	3364 DA, Sliedrecht

Opdrachtgever

Naam	Kuno Klumpers
E-mailadres	k.klumpers@unit4.nl
Telefoonnummer	31 184 44 4452

Begeleider

Naam	Yuri Nijsen
E-mailadres	y.nijsen@unit4.nl
Telefoonnummer	31 184 44 4457

VOORWOORD

Dit verslag is geschreven in het kader van mijn afstudeerstage bij Unit 4 Software B.V. te Sliedrecht. Het afstuderen is de laatste fase van mijn opleiding Technische Informatica aan de Haagse Hogeschool, Academie ICT & Media te Delft. De stage had een doorlooptijd van 18 weken en liep in de periode van 23 april 2012 tot en met 24 augustus 2012. De inleverdatum van het eindverslag ligt op 24 september.

Dit verslag is bestemd voor de examinatoren van de Haagse Hogeschool die mijn afstudeerwerk beoordelen, de afdeling MKB ontwikkeling van UNIT4 Software B.V. en andere geïnteresseerden. Hierin vertel ik over mijn werkzaamheden en ervaringen tijdens het afstuderen. Je vindt hier achtereenvolgens een beschrijving van de organisatie, UNIT4, waar ik met genoegen vier maanden heb gewerkt. Daarop volgt een definitie van de afstudeeropdracht, het plan van aanpak wat hieruit is voortgekomen en de inrichting van het project. In de kern van het verslag worden de uitgevoerde werkzaamheden in detail beschreven. Tot slot volgt een evaluatie en een conclusie.

De afstudeerperiode is voor mij erg leerzaam en ook zeer leuk geweest. Ik heb mijn kennis van programmeren en ontwerpen uitgebreid en ook kennis gemaakt met veel nieuwe onderwerpen. Maar het leren gaat niet alleen over kennis van systemen en talen, ook het werken in een professionele omgeving en zelfstandig een project uitvoeren dragen bij aan mijn ontwikkeling. Ik heb ook meer geleerd over mezelf, hoe ik het beste werk en op welke vlakken ik mezelf nog kan beteren.

Ik sluit dit voorwoord af met een dankbetuiging aan een aantal personen, dat mij heeft begeleid en ondersteund tijdens de stage en zonder wie de afstudeerperiode er heel anders uit had gezien.

Mijn dank gaat uit naar de opdrachtgever en mijn manager: Kuno Klumpers die mij goed thuis heeft laten voelen bij UNIT4 en de voortgang van mijn opdracht heeft bewaakt. Mijn technisch begeleider Yuri Nijsen, van wie ik veel heb geleerd over het ontwerpen en implementeren van oplossingen en over inrichting van een groot framework en wat daarbij komt kijken. Ik wil al mijn naaste collega's bij Unit 4 bedanken voor het delen van hun kennis en ervaring en het zorgen voor een geweldige werksfeer. In het bijzonder Bernardo voor het beantwoorden van al mijn vragen over het dagelijks leven bij UNIT4 (maar ik bedankt hem niet voor het altijd verslaan van iedereen tijdens het gamen in de pauze). Tot slot bedank ik mijn begeleidend examinator, Dhr. Visser voor zijn begeleiding tijdens het afstudeerproces.

Dank jullie wel!

Raymond Meerburg,
Sliedrecht, 18 september 2012

INHOUD

Figuren referentie

Code referentie

Tabellen referentie

1 INLEIDING

Dit verslag is opgesteld naar aanleiding van de afstudeerstage, welke het laatste onderdeel is van de opleiding Technische Informatica aan de Haagse Hogeschool, Academie voor ICT & Media te Delft[1].

De opdracht is uitgevoerd voor de afdeling MKB ontwikkeling van Unit 4 Software B.V.[2] Hier is een nieuw framework in ontwikkeling wat onder andere moet kunnen dienen als web platform. Er is onzekerheid over de paraatheid van het framework, daarom moet er een concept web applicatie op gebouwd worden. Een gedetailleerde beschrijving van de opdracht is te vinden in hoofdstuk 3. De uitvoering van deze opdracht wordt in dit verslag beschreven.

Dit verslag moet inzicht geven in de gebruikte aanpak en uitwerking van de opdracht en in de denk- en werkwijze van de afstudeerder. Op basis hiervan kunnen de examinatoren en de gecommitteerde beoordelen of de afstudeerder de opdracht op HBO niveau heeft uitgevoerd en hierbij gebruik heeft gemaakt van de vaardigheden en kennis die hij tijdens de opleiding heeft opgedaan.

1.1 INDELING

Het afstudeerverslag is als volgt in drieën opgedeeld:

Deel I: Oriëntatie

Dit eerste deel bestaat uit twee hoofdstukken en dient als randinformatie bij het verslag. Hoofdstuk 2 beschrijft het bedrijf en de organisatie. In het volgende hoofdstuk wordt de opdracht toegelicht.

Deel II: Werkzaamheden

Dit deel omvat de kern van het verslag. Hierin worden alle werkzaamheden beschreven. In hoofdstuk 4 wordt het plan van aanpak en de totstandkoming daarvan beschreven. Hoofdstuk 5 beschrijft de uitwerking van de analysefase. De hoofdstukken 6 tot en met 13 beschrijven de implementatiefase. Tot slot wordt in hoofdstuk 14 de deploymentfase beschreven.

Deel III: Uitkomst

In het laatste deel wordt in hoofdstuk 15 de uitvoering van het project, en de oplevering van het product geëvalueerd. In hoofdstuk 16 wordt een conclusie getrokken en aanbevelingen gedaan. Daarop volgt een begrippenlijst en literatuurverwijzing. Voor alle schuingedrukte woorden en alle afkortingen in dit verslag staat in de begrippenlijst een verklaring. Het verslag sluit af met een bibliografie.

Achter in het verslag staat een opgave van de bijlagen die bij dit rapport horen.

[1] http://www.dehaagsehogeschool.nl/
[2] http://www.unit4.nl/

DEEL I: ORIËNTATIE

Dit eerste deel van het verslag geeft inzicht in de initiële situatie van het project en er wordt ingegaan op de organisatie waar de stage heeft plaatsgevonden: wat voor een bedrijf is Unit 4 Software B.V. en hoe is deze organisatie ingericht. Ook wordt het probleem beschreven en aangegeven wat de oplossingen kunnen zijn.

Hoofdstuk 2 Het bedrijf

Hierin wordt kort de geschiedenis van het bedrijf beschreven en haar kernactiviteiten. Hierin komt ook naar voren de organisatiestructuur en de plaats van de afstudeerder.

Hoofdstuk 3: Opdrachtdefinitie

Dit gaat over de opdrachtomschrijving. Hierin wordt de definitieve opdracht toegelicht, in dit hoofdstuk komt aan bod: de probleemstelling, doelstelling, opdracht, werkzaamheden, op te leveren producten en de initiële globale planning.

2 HET BEDRIJF

De afstudeeropdracht is uitgevoerd bij UNIT4 Software B.V. te Sliedrecht. Dit hoofdstuk biedt achtergrond informatie over het bedrijf en de plaats van de afstudeerder binnen de organisatie.

2.1 DE ORGANISATIE[3]

Unit 4 werd in 1980 opgericht onder de naam Unit Four International. In februari 1998 kreeg Unit 4 een beursnotering aan de AEX. Zo werd kapitaal aangetrokken om Acoso (software voor accountancy en gezondheidszorg), X-info en X-International (software voor groothandels) te kunnen overnemen. Het begin van het millennium was een mijlpaal in de geschiedenis van het Nederlandse softwarebedrijf. Unit 4 nam het Noorse Agresso over en werd in één klap een internationale speler waardoor het vestigingen in verschillende landen in Europa krijgt. In de daaropvolgende jaren worden er nog meer overnames gedaan waardoor Unit 4 in veel Europese landen een grote speler wordt op de softwaremarkt. In 2010 verandert Unit 4 Agresso haar naam wereldwijd in UNIT4.

UNIT4 is een internationaal opererend bedrijf welke zich bezig houdt met het ontwikkelen, verkopen, implementeren en ondersteunen van oplossingen waarmee organisaties gemakkelijk aan hun veranderende bedrijfsbehoeften kunnen voldoen. Dit houdt in dat het bedrijf software ontwikkelt om bedrijfsprocessen te automatiseren. Hierbij kan gedacht worden aan ERP, HRM, ICT, accounting, finance etc.

Wereldwijd is UNIT4 de zesde ERP leverancier voor middelgrote bedrijven.
UNIT4 N.V. is sinds 1998 genoteerd aan de Nederlandse aandelenbeurs, heeft meer dan 4.200 medewerkers in dienst en realiseerde in 2011 een omzet van € 454,7 miljoen[4].

UNIT4 Software B.V. is onderdeel van de UNIT4 holding. Deze organisatie houdt zich bezig met het ontwikkelen van administratieve software voor midden- en klein bedrijf. Hun product, Multivers[5], is een totaaloplossing voor de administratie in het MKB. Software B.V. is een *Microsoft gold certified partner*. Deze hoogste status binnen het Microsoft Partner Network wordt toegekend aan partners die voldoen aan de zware toelatingseisen op het gebied van klantreferenties met een aantal succesvolle implementaties, specialisatie en certificering van medewerkers en/of gecertificeerde producten. Ook onder nieuwe aangescherpte regels heeft UNIT4 deze status behouden.

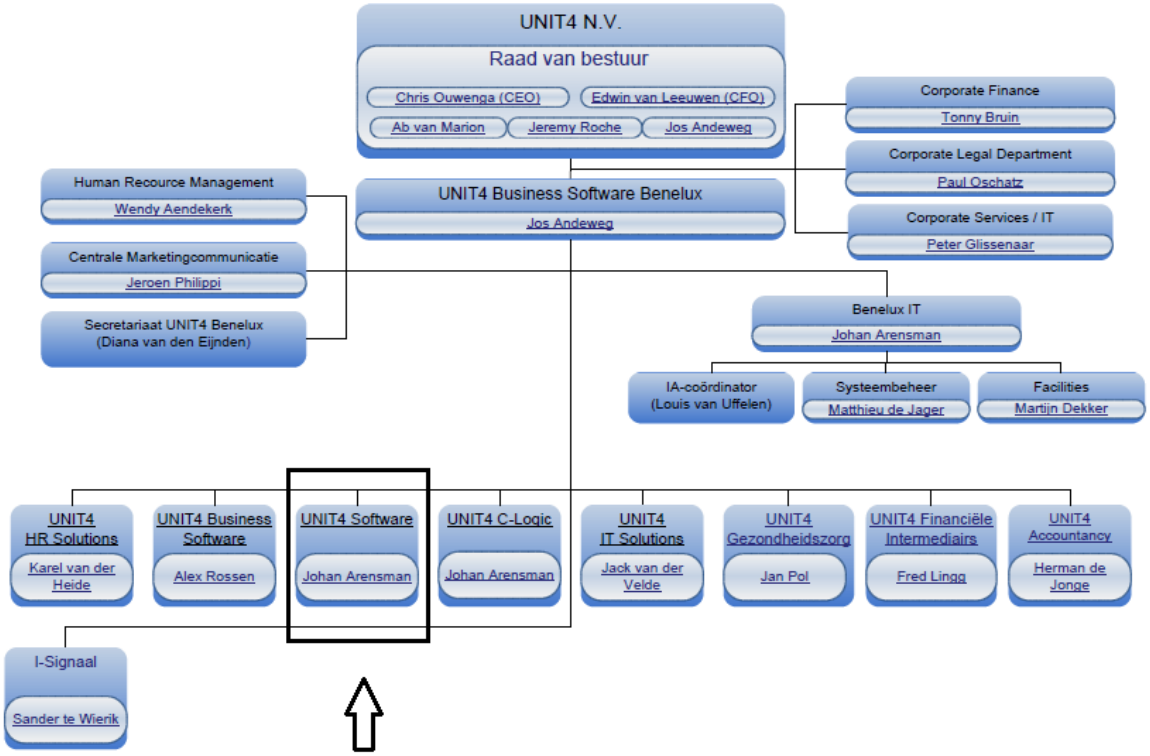
Het hoofdkantoor van UNIT4 staat in Sliedrecht.

2.2 PLAATS IN DE ORGANISATIE

[3] De eerste alinea van deze paragraaf is overgenomen van http://www.unit4.nl/over-unit4/geschiedenis
[4] http://www.unit4.nl/over-unit4
[5] http://www.unit4multivers.nl/

UNIT4 Software B.V. is ingedeeld in een aantal afdelingen: verkoop, support, administratie, facilities, en ontwikkeling. De opdracht komt voort uit de ontwikkelafdeling, deze is opgedeeld in een nieuwbouw team en een maintenance team. Ik maak deel uit van het nieuwbouw team, mijn manager is Kuno Klumpers en mijn begeleider is Yuri Nijsen.

De indeling van de complete organisatie is te zien in Figuur 1.



FIGUUR 1 ORGANISATIESTRUCTUUR UNIT4

3 OPDRACHTDEFINITIE

Dit hoofdstuk gaat in op de opdracht, en de totstandkoming daarvan.

3.1 PROBLEEMSTELLING

Naast het product Multivers maakt UNIT4 ook een framework (Universal Business Connector) welke gebruikt kan worden om applicaties te schrijven die samenwerken met Multivers. Dit is het framework dat partners kunnen gebruiken om Multivers uit te breiden met branche-specifieke uitbreidingen en onderdelen. Sinds anderhalf jaar wordt er gewerkt aan een nieuw framework genaamd Fabric. Met Fabric moet het veel makkelijker worden om met applicaties over het internet te communiceren met Multivers.

Via Fabric kunnen gegevens zoals sales, logistics, CRM, financial etc. benaderd worden. Fabric maakt hiervan business objects aan die gebruikt kunnen worden in applicaties. Error handling, authenticatie, verificatie etc. wordt allemaal door Fabric geregeld. Op dit moment is Fabric nog in ontwikkeling.

Er is steeds meer vraag naar web applicaties. Het is wenselijk om de business logic zo in te richten dat er zowel desktop applicaties als web applicaties op kunnen draaien. Het is hierbij belangrijk dat beide applicaties kunnen berusten op dezelfde implementatie en dat er geen gebruikt wordt gemaakt van web of desktop specifieke functionaliteit.

Partners van UNIT4 willen graag Fabric gebruiken om services via het web aan te bieden, maar UNIT4 weet niet of Fabric daar op dit moment gereed voor is en zij willen daarom meer zekerheid hebben dat Fabric er klaar voor is.

3.2 DOELSTELLING

Het doel van de opdracht is enerzijds het uitbrengen van een bevindingenrapport over de paraatheid van Fabric m.b.t. het dienen als framework voor web applicaties. Anderzijds moeten die delen die nog niet in orde zijn, aangepast worden. De concept-applicatie moet kunnen dienen als demo voor geïnteresseerden. De source code van het programma moet ingezet kunnen worden als voorbeeld code voor een Fabric applicatie.

3.3 RESULTAAT

Bij succesvolle afronding van het project heeft UNIT4 meer zekerheid dat Fabric voldoende paraat is voor de ontwikkeling van web applicaties die erop berusten. Dit betekent dat UNIT4 Fabric als ontwikkelplatform voor web applicaties kan aanbieden aan partners. Ze kunnen het concept gebruiken om een demo te geven van de mogelijkheden die Fabric biedt. Partners die met het nieuwe framework gaan werken kunnen de source code van het concept bekijken om zo te leren hoe Fabric het beste gebruikt kan worden. Dit maakt het ontwikkelen makkelijker.

3.4 UIT TE VOEREN WERKZAAMHEDEN

Het bewijzen van en onderzoeken naar de paraatheid van Fabric zal gebeuren door een proof-of-concept web applicatie te bouwen welke enkele financiële bedrijfsprocessen kan beheren. De aard van de web applicatie wordt een overzicht systeem van een boekhoudpakket. De belangrijkste informatie moet opgevraagd kunnen

worden en het moet tevens mogelijk zijn om zelf objecten zoals opdrachten, debiteuren en artikelen aan te maken. De web applicatie moet wel kunnen dienen als voorbeeld voor partners, het moet daarom een oplossing zijn die goed samenhangt en niet zomaar een verzameling webpagina's. Omdat deze code als voorbeeld moet dienen is het ook van belang om alle source code goed te documenteren.

Enkele aspecten die belangrijk zijn bij de web applicatie:

- Configureerbaar en flexibel.
- Goede performance op de server.
- Caching van data bij de client en server om performance te verbeteren.
- Om kunnen gaan met foute input.

3.5 Globale planning

Een globale fasering van het traject is als volgt.

- Week 1: Plan van aanpak schrijven, oriënteren op Fabric. Details opdracht verduidelijken. Project plannen.
- Week 2: Opstellen requirements web app. Ontwerp maken. Functionele ontwerpen maken, eisen bepalen. Implementatie plannen.
- Week 3-13: Implementatie web applicatie tegelijk met het aanpassen van Fabric.
- Week 13-15: client- en serverside caching implementeren.
- Week 15-16: Ontwikkelaars tests, gebruikerstests, laatste bugs fixen.
- Week 17-18: schrijven rapport, afmaken afstudeerdossier.

3.6 Op te leveren (tussen)producten

Als eerst zal een plan van aanpak opgeleverd worden waarin staat beschreven op welke manier het project wordt beheerd, welke technieken en methoden er gebruikt gaan worden en wat de werkwijze wordt. Alle keuzes worden toegelicht en onderbouwd. Hier hoort ook een requirements analyse bij.

Tijdens de opdracht worden er zowel functionele als technische ontwerpen gemaakt. Deze worden opgeleverd aan de opdrachtgever.

De complete opdracht levert enerzijds een rapport op over de paraatheid van Fabric. Dit bestaat uit een beschrijving van welke aspecten er veranderd zijn, en of er nog meer aanpassingen nodig zijn. Een klein rapport zal beschrijven hoe het werken met Fabric is verlopen. Hierin wordt aangegeven welke onderdelen vlot verliepen en welke er voor vertraging zorgden. Moeilijkheden en onverwachte problemen worden toegelicht zodat Fabric hier mogelijk op aangepast kan worden.

De volledige source code van de applicatie wordt opgeleverd samen met documentatie. Ook de verbeteringen en aanpassingen die in Fabric worden doorgevoerd worden opgeleverd.

3.7 Leerdoelen

Het project wordt uitgevoerd als afstudeeropdracht, dit betekent dat een belangrijk doel van het project is om kennis op te doen. De volgende leerdoelen zijn opgesteld:

- Uitbreiden programmeer en ontwerp kennis.
- Zelfstandig een compleet project kunnen plannen en uitvoeren.
- Leren over nieuwe methoden en technieken.
- Ervaring opdoen over het werken in een professionele werkomgeving.

DEEL II: WERKZAAMHEDEN

Dit deel vormt de kern van het verslag, hier wordt diep ingegaan op alle uitgevoerde werkzaamheden. Als eerste wordt het opstellen van het projectplan toegelicht, daarna komen de werkzaamheden per sprint aan bod.

Hoofdstuk 4

Plan van aanpak

Dit hoofdstuk gaat in op de aanpak van het project en vertelt hoe de opdracht uit hoofdstuk 3 Opdrachtdefinitie zal worden uitgevoerd. Er wordt ingegaan op projectmanagement, beheer, technieken, ontwikkelmethode en de werkwijze. Daarnaast wordt er een initiële planning voorgelegd.

Hoofdstuk 5

Sprint 1 “Analyse”

In dit hoofdstuk worden de werkzaamheden tijdens de eerste sprint beschreven. In deze periode is het project verder uitgewerkt en zijn de nodige voorbereidingen gemaakt voordat aan het eindproduct van het project gewerkt kan worden. Dit houdt in dat de projectinrichting volledig is uitgewerkt en er alvast enkele ontwerpen zijn gemaakt.

Hoofdstuk 6 - 13 Sprints

De kern van het verslag gaat over de werkzaamheden tijdens de sprints. Aan elke sprint wordt een hoofdstuk toegewijd. De eerste sprint heeft een andere aard en wordt daarom apart opgenoemd. De overige sprints gaan voornamelijk over het analyseren, ontwerpen, implementeren en testen van features uit de sprint backlog. De meest interessante onderwerpen worden besproken, dit betekent dat sommige fasen van user stories niet aan bod komen.

In deze hoofdstukken wordt per sprint diep ingegaan op de uitgevoerde werkzaamheden. De meeste sprints hebben zich gehouden aan de volgende opzet:

1. Analyseren van de items in de product backlog.
2. Sprint indelen.
3. Functionele ontwerpen.
4. Technische ontwerpen.
5. Per sprint item: implementatie, afgewisseld met testen.
6. Unit tests schrijven voor items waar mogelijk.
7. Alle nieuwe tests doorlopen.
8. Complete testsuite uitvoeren.
9. Documentatie volledig in orde maken (source code).
10. Eventueel demo geven van de voortgang aan de product owner.
11. Evaluatie sprint, niet behaalde items terugplaatsen in product backlog.

Hoofdstuk 14 Deployment:

De laatste fase van het project is de deployment. Hier wordt het product onderworpen aan acceptatie tests en uitgeleverd.

4 PLAN VAN AANPAK

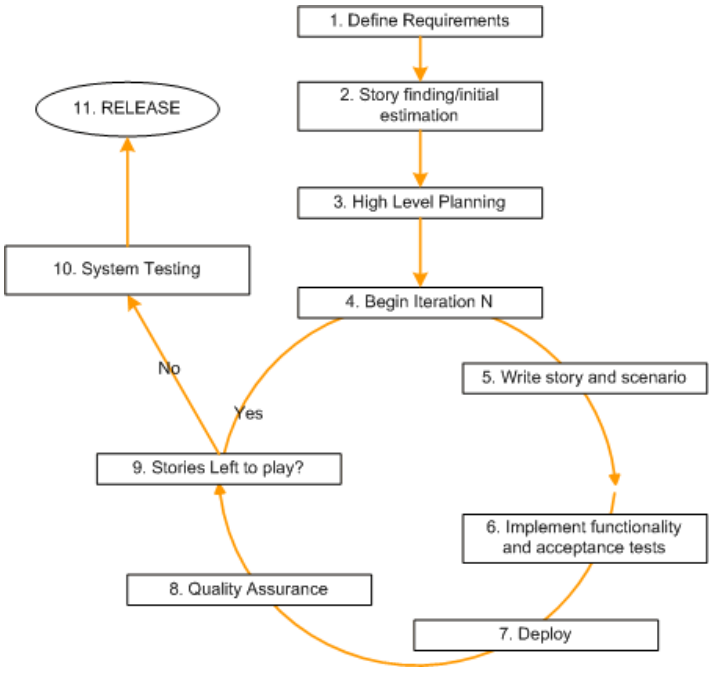
In het plan van aanpak worden doorslaggevende keuzes gemaakt die veel invloed hebben op het verloop van het project, het plan van aanpak geeft achtergrond informatie over het bedrijf, de opdrachtgever en de opdrachtnemer. De beginsituatie wordt geschetst, evenals het gewenste eindresultaat. Een plan om dit eindresultaat te bereiken staat centraal in het plan van aanpak. Verdere informatie over de werkzaamheden, planning, projectinrichting, risicofactoren en kosten en baten moeten de opdrachtgever inzicht geven in de geplande uitvoering van het project.

4.1 AANPAK

4.1.1 PROJECT MANAGEMENT

Project management is de discipline van het plannen, organiseren, beheren en controleren van een project. Het kiezen van een methode hiervoor kan grote impact hebben op het verloop van het project. Er zijn veel verschillende methoden specifiek voor software ontwikkeling. Het kiezen van de juiste hangt af van de sterke en zwakke punten van de methode.

De precieze aard van het eindproduct staat niet aan het begin van de opdracht vast. Gedurende de ontwikkeling ervan worden er vrijwel zeker nieuwe wensen ingediend en/of bestaande gewijzigd. Er is dus ruimte nodig voor veranderende requirements. Dit sluit simpele methoden zoals waterval[6] uit. Het herhaaldelijk ontwerpen, bouwen, testen en integreren van nieuwe onderdelen sluit aan bij de aard van de opdracht. Een ontwikkelmethode die hier goed bij past is 'incrementeel'. De opdrachtgever kan na ieder increment het resultaat bekijken en eventuele nieuwe inzichten of eisen doorvoeren in volgende iteraties. Specifieke implementaties van incrementeel[7] ontwikkelen zijn RUP[8] en agile[9] methoden. RUP is toe te passen op kleine projecten, maar past het best bij grote software projecten. Veel agile methoden daarentegen kunnen zowel voor grote als kleine projecten ingericht worden. Agile is een



FIGUUR 2 AGILE DEVELOPMENT CYCLE

[6] http://en.wikipedia.org/wiki/Waterfall_model
[7] Boek: User Stories Applied: For Agile Software Development
[8] www.ibm.com/software/awdtools/rup/
[9] http://www.agilealliance.org/

Engels woord voor ‘lenig’ of ‘behendig’. Deze methodiek is gericht op het snel kunnen reageren op verandering, feedback en input in een project. Agile ontwikkelen wordt niet alleen gedreven door planning, maar ook door feedback van tests. Zie Figuur 2.

Het kan voordelig zijn om dezelfde methode te gebruiken die het bedrijf voor haar eigen projecten gebruikt. Er is dan kennis en ervaring paraat, wat goed uitkomt als er problemen optreden. Bij UNIT4 worden projecten gemanaged met Scrum. Scrum[10] is een vorm van agile ontwikkelen die goed past bij projecten waar het moeilijk is om ver vooruit te plannen. De Scrum methode heeft een aantal doelstellingen:

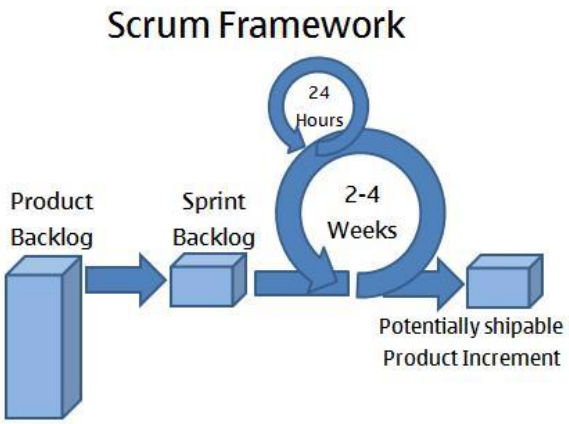
- Verhogen van de effectiviteit van het team.
- Het bewaken van de vooruitgang van het team.
- Het oplossen van blokkades.
- Het bewaken van de projectvoortgang.
- In kaart brengen en minimaliseren van de risico’s.

Hoewel Scrum is gericht op het aansturen van een team van ontwikkelaars, is het ook mogelijk de methode alleen te gebruiken. Scrum past goed bij het project en ook vanuit het bedrijf is er kennis van, er is daarom gekozen voor Scrum.

4.1.2 Ontwikkel Methode

De ontwikkelmethode bepaald op welke manier de ontwikkeling van software verloopt. Scrum definieert drie kern rollen. De ‘product owner’: deze vertegenwoordigt de belanghebbenden en oordeelt over de kwaliteit van wat opgeleverd wordt. Deze rol wordt vervuld door mijn manager. Het ‘development team’ is verantwoordelijk voor het implementeren van functionaliteit, zo mogelijk in compleet afgeronde subsystemen. De leden van het team analyseren, ontwerpen, implementeren en testen de software, de organisatie hiervan wordt intern geregeld. Het team bestaat uit één iemand die de complete uitvoering van het project doet. De ‘Scrum master’ is degene die moet zorgen dat het ontwikkelteam niet gestoord wordt door afleidingen, deze rol wordt in dit project niet vervuld.

Scrum deelt het werk op in sprints. Een sprint duurt tussen de één en de vier weken, zie Figuur 3. Voorafgaand aan elke sprint wordt er gekeken welke en/of hoeveel functionaliteit er in deze sprint geïmplementeerd kan worden. Er wordt vooraf zo veel mogelijk functionaliteit bedacht, een subset daarvan wordt in een sprint geplaatst. De gekozen items (functionaliteit) komen voort uit de *sprint backlog*, dit is een verzameling *user stories*. Een user story beschrijft functionaliteit vanuit de gebruiker. Een user story zou kunnen zijn “Ik wil resultaten kunnen sorteren op naam of datum.” Aan elke user story wordt een prioriteit toegekend. De user stories met hoge prioriteiten worden in de sprint geplaatst. Het ontwikkelteam moet zelf schatten hoeveel er gehaald kan worden in één sprint. Aan het eind van de sprint wordt er gekeken welke functionaliteit er af is, als er iets niet af is wordt het terug geplaatst in de product backlog. Het kan dan in een volgende sprint opnieuw worden aangepakt.



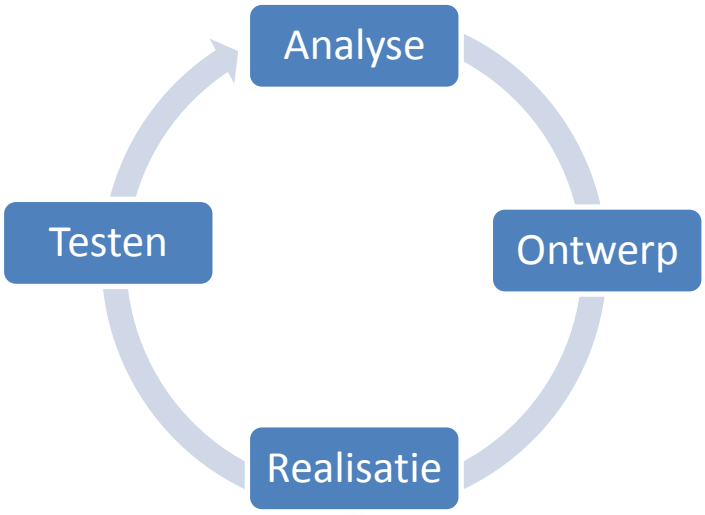
FIGUUR 3 SCRUM CYCLUS

[10] http://en.wikipedia.org/wiki/Scrum_(development)

Een user story wordt volgens het SMART-principe[11] opgesteld. Dit houdt in dat een user story moet voldoen aan de volgende voorwaarden:

- **Specifiek**; De doelstelling moet eenduidig zijn.
- **Meetbaar**; Onder welke (meetbare/observeerbare) voorwaarden of vorm is het doel bereikt.
- **Aanvaardbaar**; Is deze acceptabel genoeg voor de doelgroep en/of management.
- **Realistisch**; De doelstelling moet haalbaar zijn.
- **Tijdgebonden**; Wanneer (in de tijd) moet het doel bereikt zijn.

User stories zijn inherent tijdgebonden, ze moeten binnen de sprint afgemaakt worden. Met de overige parameters moet wel expliciet rekening gehouden worden. Het is vooral belangrijk dat er overeenstemming is tussen de product owner en het ontwikkelteam wanneer een user story is 'af' is, d.w.z. het is correct geïmplementeerd. Zie Figuur 4.



FIGUUR 4 PROCESS VOOR ELK USER STORY

4.2 TECHNIEKEN

Het kiezen van de juiste technieken om het probleem mee op te lossen is van groot belang. Er moeten technieken uitgezocht en gekozen worden die goed aansluiten bij de taken die voltooid moeten worden, maar ook bij het bestaande software systeem.

4.2.1 FRAMEWORK

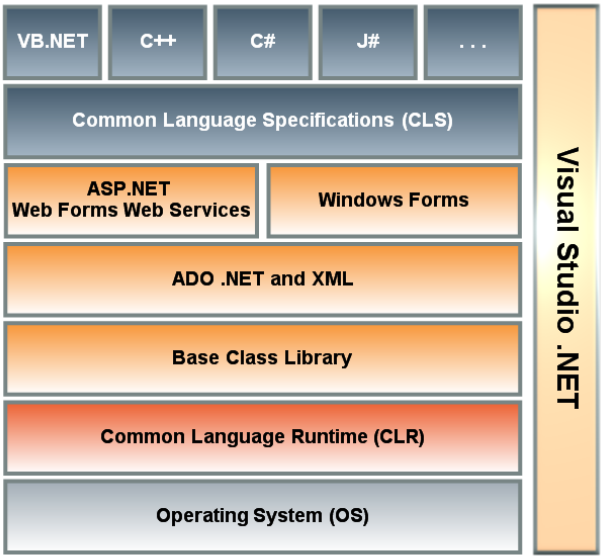
Een totaaloplossing zoals een web applicatie bestaat uit veel verschillende componenten. Niet alleen moet de interface en de logica gemaakt worden, ook het afhandelen van web-requests en communiceren met de database vormen barrières. Deze laatste twee zijn niet specifiek aan de opdracht en kunnen opgelost worden door een *framework* te gebruiken. Een framework verzorgt een abstractie om generieke problemen op te lossen. Er is een set generieke functionaliteit die gebruikt en uitgebreid kan worden door de ontwikkelaar. Op die manier kan de ontwikkelaar het afhandelen van web-requests overlaten aan het framework, terwijl hij zelf

[11] http://nl.wikipedia.org/wiki/SMART-principe

kan werken aan de daadwerkelijke opdracht. Ontwikkelaars maken programma's door hun eigen code te combineren met code uit een framework.

4.2.1.1 .NET

Het .NET framework[12] (uitspraak: dotNET) staat aan de basis van alle software die binnen UNIT4 Software B.V. geschreven wordt. Het .NET framework, ontwikkeld door Microsoft, bestaat uit twee delen, enerzijds uit een *runtime* (zie Figuur 5) die code kan uitvoeren en anderzijds uit de *Base Class Library (BCL)*. De runtime verzorgt belangrijke services zoals proces management. Beveiliging, memory management etc. De BCL is een verzameling generieke klassen (vergelijkbaar met de C++ STL[13], maar uitgebreider). Deze klassen bieden generieke functionaliteit voor onder andere user interface, data access, cryptografie, web development, netwerk communicatie en meer. Daarnaast zijn er veel verschillende typen datastructuren beschikbaar. Het framework wordt neergezet als concurrent op Java, het idee is om het gemakkelijk te maken voor ontwikkelaars om programma's te schrijven die communiceren met andere .NET software.



FIGUUR 5 .NET FRAMEWORK STACK

Het .NET framework is gericht op Windows. Dit betekent dat er officieel alleen support is voor Windows. Onofficieel is het met het Mono framework mogelijk om .NET applicaties ook op andere systemen draaien, waaronder Unix en Mac OSX.

4.2.1.2 FABRIC

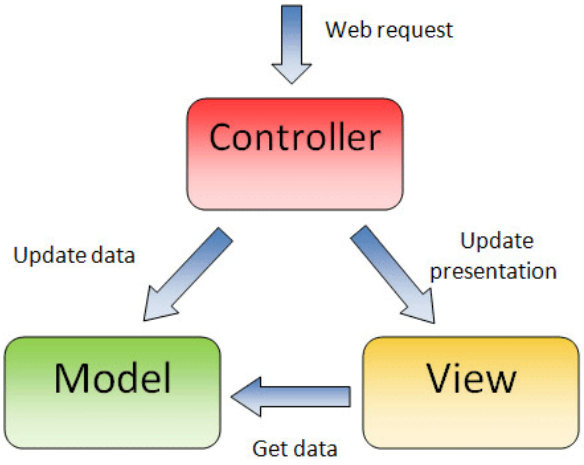
Fabric is de naam van het nieuwe framework van UNIT4. Dit framework verzorgt de koppeling tussen database en *business objects*[14] van UNIT4. Een business object is een entiteit welke zich bevindt in de *business-layer* van een stuk software. Een business object zelf is meestal beperkt in functionaliteit, maar bezit een hele reeks eigenschappen. Een order object heeft bijvoorbeeld onder andere de eigenschappen 'orderdatum' en 'customerId', de customerId verwijst naar het customer business object die de order geplaatst heeft. Een customer bezit vervolgens gegevens over personalia etc. Op deze manier worden entiteiten uit de echte wereld vertaald naar business objects. De objecten en hun samenhang wordt volledig gemodelleerd in software. Informatie over entiteiten staat opgeslagen in een database, wanneer er een business object opgevraagd wordt, wordt de informatie uit de database vertaald naar een business object. De business object worden gebruikt om acties uit de echte wereld, zoals een geplaatste order, in het systeem te verwerken.

[12] http://www.microsoft.com/net
[13] http://www.sgi.com/tech/stl/table_of_contents.html
[14] http://en.wikipedia.org/wiki/Business_object

Fabric is gebouwd bovenop CSLA (Component-based Scalable Logical Architecture)[15]. CSLA is een open-source framework geschreven in C#. Het framework definieert een architectuur waarmee robuuste object-georiënteerde software geschreven kan worden die werkt met business objects. CSLA biedt base-classes waarmee zelf business objects geïmplementeerd kunnen worden, faciliteiten voor database access, vertaling van data naar business object, het afdwingen van *business rules* en meer.

4.2.1.3 ASP .NET MVC

Een statische website kan gewoon met html gebouwd worden, maar zodra er meer nodig is wordt het steeds wenselijker om een *web application framework*[16] te gebruiken. Een web application framework is een framework dat ontworpen is om het maken van dynamische websites en web applicaties te ondersteunen door onder andere overhead te verminderen. Het framework biedt dan vaak oplossingen voor database toegang, sessie management en meer. Hergebruik van onderdelen wordt gestimuleerd. Een website pagina kan beschreven worden als een set subonderdelen die door het framework als één geheel worden gepresenteerd aan de gebruiker.



FIGUUR 6 MODEL-VIEW-CONTROLLER

Een aantal populaire frameworks zijn: *Ruby on Rails*[17], *Django*[18], *Zend*[19] en *ASP.NET MVC*[20]. Het kiezen van een framework betekent vaak ook direct kiezen voor de programmeertaal waar het framework op gericht is. Ruby on Rails applicaties worden geschreven in Ruby, Django maakt gebruik van Python, Zend van PHP en ASP .NET MVC applicaties kunnen geschreven worden in C# of Visual Basic .NET.

De laatst genoemde sluit verreweg het beste aan bij Fabric, wat compleet in C# is geschreven. Het ontwikkelen zal makkelijker zijn als de verschillende componenten goed met elkaar integreren. De parate kennis van C# is ook veruit het best van de beschikbare talen. Ook wordt verwacht dat partners hun oplossingen zullen implementeren met ASP.NET MVC.

ASP .NET MVC is een open source web application framework gebouwd door Microsoft. Het framework staat ontwikkelaars toe om een website te beschrijven als een samenstelling van drie rollen: Model, View en Controller, ASP .NET MVC berust op het *Model-View-Controller pattern*[21]. Zie Figuur 6.

4.2.2 PROGRAMMEERTAAL

Deze applicatie bestaat uit een server-gedeelte en een client-gedeelte. Op de server wordt alle input verwerkt en het resultaat teruggestuurd. Aan de client-zijde wordt het resultaat grafisch weer gegeven. De gebruiker

[15] http://lhotka.net/cslanet/
[16] http://en.wikipedia.org/wiki/Web_application_framework
[17] http://rubyonrails.org/
[18] https://www.djangoproject.com/
[19] http://www.zend.com/en/
[20] http://www.asp.net/mvc
[21] http://en.wikipedia.org/wiki/Model–view–controller

communiceert via de gebruikers interface met de server. De twee platformen worden gedreven door verschillende omgevingen en programmeertalen. De meest geschikte voor dit project moet bepaald worden.

4.2.2.1 SERVER SIDE

De logica op de server is verreweg het meest uitgebreid en ingewikkeld. Hier wordt de input geanalyseerd en naar de juiste plaats gestuurd. Hierop wordt alle logica uitgevoerd en het resultaat wordt in elkaar gezet. Dit resultaat wordt teruggestuurd naar de gebruiker.

De keuze van de programmeertaal wordt zeer beperkt door het gebruik van ASP .NET MVC. Deze laat alleen C# en Visual Basic .NET toe. C# sluit goed aan bij het Fabric framework, het bedrijf en ook mijn eigen kennis. De programmeertaal voor dit project is daarom C#, versie 4.0[22].

C# is een objectgeoriënteerde programmeertaal ontwikkeld door Microsoft als deel van het .NET-initiatief. Het is bedoeld als een simpele, moderne taal voor algemene doeleinden. In 2002 verscheen de eerste versie en deze heeft sindsdien drie grote updates gehad, een vierde is in ontwikkeling.

Een belangrijk onderdeel van dit project is het query-en van data. Gebruikers willen kunnen filteren, sorteren enz. Hiervoor biedt het .Net framework een oplossing: LINQ[23] (Language Integrated Query). Hiermee kan op een intuïtieve manier data opgevraagd worden. Zie Code 1.

```
var customerIdList = Fabric.CRM.Lists.CustomerNVL.Fetch(); // haal lijst id's op.
var customerSubset = customerIdList.Take(10); // pak 10 resultaten.
var customerIds = customerSubset.Select(customerEntry => customerEntry.Key); // selecteer
alleen het key veld
var Customers = customerIds.Select(id => Fabric.CRM.Customer.Fetch(id)); // haal customers
op uit de database
var CustomerInfos = Customers.Select(customer => new { name = customer.Name, city =
customer.City }); // selecteer alleen de velden die je wilt.
```

CODE 1 OPHALEN VAN NAAM EN WOONPLAATS VAN TIEN CUSTOMERS

De verschillende LINQ methoden kunnen ook aan elkaar gekoppeld worden door de *fluent-interface*[24] te gebruiken. Het gehele statement wordt daardoor groter en minder leesbaar, maar er worden minder tijdelijke variabelen aangemaakt. Zie Code 2.

```
var customerIdList = Fabric.CRM.Lists.CustomerNVL.Fetch().Take(10).Select(entry =>
Fabric.CRM.Customer.Fetch(entry.Key)).Select(customer => new { name = customer.Name, city
= customer.City });
```

CODE 2 GEBRUIK VAN FLUENT NOTATIE

4.2.2.2 CLIENT SIDE

De client-zijde van de applicatie is enerzijds verantwoordelijk voor het verwerken en doorgeven van input van de gebruiker en het terugkoppelen (meestal weergeven) van de resultaten. Aan de andere kant moet deze de antwoorden van de server ontvangen en juist weergeven. Hiervoor moet een *gebruikers interface* gemaakt worden. Aangezien het gaat om een web applicatie is er weinig vrijheid in programmeertaal. Voor de opmaak

[22] http://msdn.microsoft.com/en-us/vstudio/hh388566.aspx
[23] http://msdn.microsoft.com/en-us/library/bb397926.aspx
[24] http://www.martinfowler.com/bliki/FluentInterface.html

zal HTML5 gebruikt worden maar er is wel backwards-compatibiliteit met HTML4. Geavanceerde interactie met de gebruiker wordt gedaan met JavaScript. De client zijde van de web applicatie zicht zich volledig op web browsers.

4.3 BEHEERS ASPECTEN

De uitvoerbaarheid van het project zoals beschreven in het plan van aanpak moet gecontroleerd worden om er zeker van te zijn dat het project gehaald kan worden. Als aan de haalbaarheid wordt getwijfeld moet het plan aangepast worden. Het modelleren van de haalbaarheid kan gedaan worden aan de hand van TGKIO[25], hierbij wordt Tijd, Geld, Kwaliteit, Informatie en Organisatie gekwantificeerd. Er is geen budget nodig voor dit project, dus het geld onderdeel wordt niet beschreven. De organisatie van het project is aan het begin van dit hoofdstuk besproken, Project Management.

4.3.1 TIJD

De doorlooptijd van het project wordt vanuit de Haagse Hogeschool vastgezet op minimaal zeventien weken, en maximaal twintig. Indien het project niet succesvol binnen deze tijd afgerond kan worden, kan er een verlenging van vijf weken worden aangevraagd. Het project is op 23 april van start gegaan. Als uitgegaan wordt van zeventien weken moet het project op 18 augustus afgerond zijn. Er is geen harde deadline voor de oplevering, dit betekent dat de planning opgeschoven kan worden indien nodig.

4.3.2 KWALITEIT

Het meten van de kwaliteit van software kan moeilijk zijn, er spelen veel factoren mee. Vooral de robuustheid is belangrijk, andere punten zijn betrouwbaarheid, uitbreidbaarheid, onderhoudbaarheid, testbaarheid, flexibiliteit en efficiëntie. Ook systeemdocumentatie is belangrijk, hoewel het geen onderdeel is van de software zelf, is het wel belangrijk de software goed te beschrijven. Het objectief beoordelen van kwaliteit kan gedaan worden met behulp van de *Software engineering — Product quality*[26] standaard. Hoewel een volledig assessment van de gebouwde software buiten de scope van het project valt, kan er wel gebruik gemaakt worden van een aantal orderdelen van de standaard. Zie Tabel 1 voor de kwaliteitseisen van dit project.

TABEL 1 SUBSET VAN KWALITEITSEISEN VOLGENS ISO 9126-4

Kwaliteitseis	Acceptatie criteria
Functionaliteit	
Geschiktheid	Het product voldoet aan de gestelde requirements.
Veiligheid	De applicatie controleert of de gebruiker toegang heeft tot bepaalde data, alvorens deze beschikbaar te stellen. Gebruikers moeten zich authentiseren.
Betrouwbaarheid	
Stabiliteit	De applicatie kan onbepaalde tijd draaien, met meerdere gebruikers zonder vast te lopen.
Correctheid	De bron code bevat geen lekken of bugs, er treedt ook geen *undefined behaviour* op.

[25] http://www.pmwiki.nl/kennis/tgkio
[26] http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749

Onderhoudbaarheid	
Testbaarheid	De verschillende componenten van het programma hebben waar mogelijk bijbehorende unit tests.
Veranderbaarheid	Onderdelen van het programma maken zo veel mogelijk gebruik van interfaces. Het abstractieniveau wordt hoog gehouden en er bestaat geen sterke koppeling tussen niet verwante onderdelen.
Overdraagbaarheid	
Code standaard	Er wordt zo veel mogelijk gehouden aan de code standaarden van UNIT4.
Begrijpelijkheid	De code is voorzien van documentatie. Daar waar complexe constructies worden gebruikt wordt dit toegelicht in de source file.
Efficiëntie	
Tijd gebonden	De applicatie reageert vlot.
Processor gebonden	Er treden geen situaties op waarbij het programma langdurig zeer veel processortijd gebruikt.
Schaalbaarheid	De *look and feel* van de applicatie gaat niet achteruit wanneer er veel gebruikers bezig zijn, of er grote datasets aangesproken moeten worden.

De kwaliteitseisen van dit rapport zelf zijn opgesteld door de Haagse Hogeschool.

4.3.3 INFORMATIE

Tijdens de uitvoering van het project moet er geregeld gecommuniceerd worden over de huidige stand van zaken. De opdrachtgever moet goed geïnformeerd blijven en ook de Haagse Hogeschool wil graag op de hoogte blijven. Er wordt zowel schriftelijk als mondeling verslag uitgebracht. Er vindt elke twee weken een voortgangsoverleg plaats met de opdrachtgever. In dit overleg wordt besproken wat er in de afgelopen periode is gedaan, wat de planning is voor de komende periode en of er nog eventuele problemen zijn, of verwacht worden.

Na ongeveer zes weken doorlooptijd vindt er een bedrijfsbezoek plaats door de examinator. Deze heeft dan de kans om te controleren of het project een goede start heeft. Er vindt ook overleg plaats tussen de manager, de examinator en de student. Indien er belangrijke aspecten van het project niet in orde zijn, kan er in een vroeg stadium worden ingegrepen.

Een logboek zal zorgen dat er later altijd terug geblikt kan worden op de werkzaamheden. Dit komt van pas tijdens de verslaglegging en kan ook helpen als de planning herzien moet worden.

4.3.4 ORGANISATIE

Het uitvoeren van de werkzaamheden binnen het afstudeerproject komt allemaal aan de afstudeerder toe. Maar er zijn wel diverse andere mensen bij betrokken. De structuur binnen het project is als volgt:

Opdrachtnemer → Raymond Meerburg (Afstudeerder)

Zelf speel ik de rol van opdrachtnemer in het project. Ik ben de eindverantwoordelijke voor het succesvol afronden van het project. Ik hou me bezig met het analyseren, ontwerpen, implementeren en testen van de applicatie. Daarnaast handel ik alle communicatie met andere partijen af. Binnen Scrum ben ik het enige lid van het development team, en neem ik ook de rol van Scrum master op me.

Opdrachtgever → Kuno Klumpers (Product development manager)

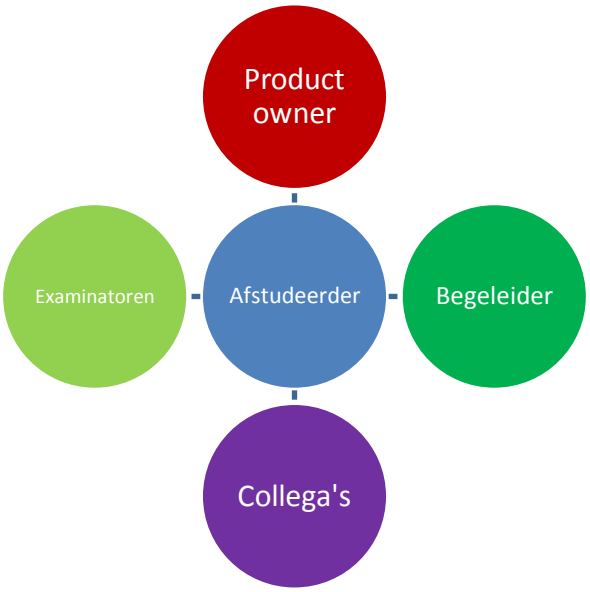
Dhr. Klumpers is binnen het project de opdrachtgever en binnen de organisatie mijn manager. Hij speelt ook de rol van product owner.

Begeleider → Yuri Nijsen (Software architect)

Dhr. Nijsen is de architect van Fabric, hij kent de ins en outs van het framework en zal mij begeleiden tijdens de ontwikkeling van de nieuwe applicatie. Yuri zal adviseren en kan ook assisteren tijdens de ontwikkelfase.

Examinatoren → John Visser, Henk van den Bosch (docenten technische informatica aan de HHS Delft)

De examinatoren beoordelen het uiteindelijke resultaat. De eerste examinator controleert het plan van aanpak en het eerste concept. Beide examinatoren voeren het tussentijds assessment uit en geven advies over het inleveren van het verslag.



FIGUUR 7 OMGEVING VAN DE AFSTUDEERDER

4.4 TIJDSPLANNING

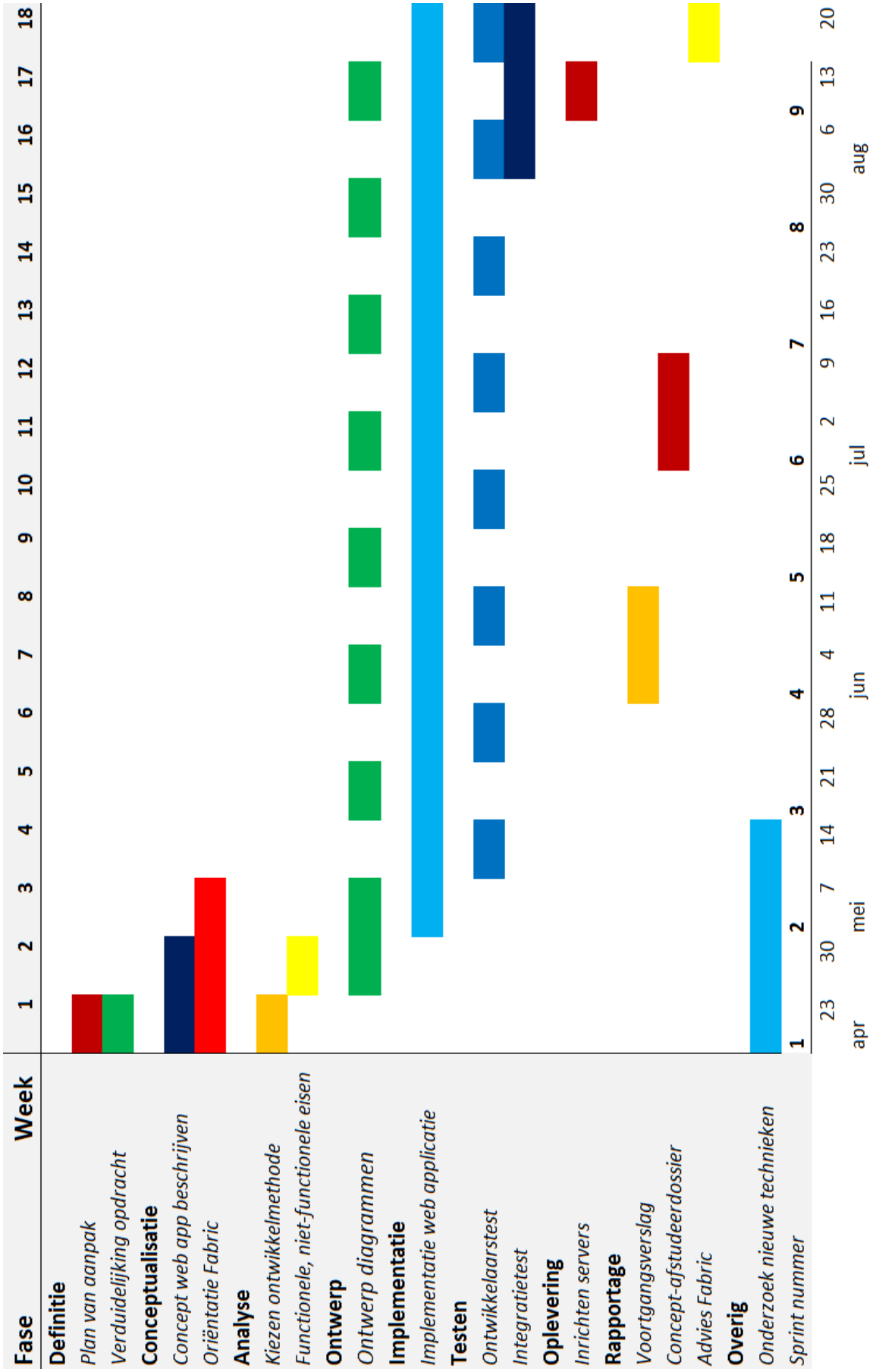
Het maken van de planning begint bij het indelen van de uit te voeren activiteiten. Er kan daarna een schatting gemaakt worden van de hoeveelheid tijd nodig is per activiteit. Het samenkomen hiervan levert een planning op. Tijdens elke fase van het project worden er documenten of software opgeleverd. Aan het begin van een fase wordt gedefinieerd welke items er opgeleverd moeten worden in die fase, deze items worden *deliverables* genoemd. Zodra alle items zijn opgeleverd en zo nodig goedgekeurd kan het project doorgaan naar de volgende fase.

Om de verschillende fasen van het project in te delen lenen we een aantal termen van de RUP ontwikkelmethode, er kunnen analogen worden getrokken tussen de fasen uit RUP, en de indelingen van de sprints in dit project. De eerste sprint die uitgevoerd wordt gaat over analyse, deze bevat elementen uit de *inception* fase en de *elaboration* fase. Dit zijn twee relatief korte fasen die in dit project in één sprint worden gecombineerd. In deze fasen worden alle parameters van het project zelf bepaald, er wordt gekeken naar de haalbaarheid, risico's, en aanpak. Bij de aanvang van het project wordt er dus nog niet meteen ontwikkeld, het opstellen van een plan van aanpak en het oriënteren op de opdracht heeft voorrang. In deze eerste fase worden de functionele en niet-functionele eisen opgesteld en omgezet in user stories. Er worden ook al enkele ontwerpen gemaakt voor het systeem.

De *construction* fase is verreweg het langst, de meeste sprints vallen in deze fase. Hierin wordt het product gerealiseerd en getest. Dit gebeurt in sprints (iteraties). Elke sprint begint met het bedenken van de technische architectuur en het maken van technische ontwerpen van de use cases. Per sprint vindt er een analyse plaats, gevolgd door een ontwerp en daarna de implementatie, tijdens het implementeren moet er ook getest worden. Het implementeren zal de meeste tijd kosten. In de eerste week van de sprint vindt het ontwerpen plaats, dit levert verschillende UML diagrammen op, deze zouden rond halverwege de week af moeten zijn. Dan wordt er direct geïmplementeerd. In de tweede week vindt er naast het implementeren ook testen plaats. Het ontwikkelen vindt eigenlijk constant plaats in de construction fase, terwijl het ontwerpen en testen afgewisseld wordt.

Aan het einde van het project is er tijd nodig voor afronding en oplevering. Er vindt dan ook een uitgebreide systeem plaats. Verslaglegging zal dan meer naar voren komen, alle projectdocumentatie moet dan in orde zijn. Door middel van een Gantt Chart[27] kan een planning duidelijk gevisualiseerd worden. Hier is goed zichtbaar hoe de sprints zijn ingedeeld (zie Figuur 8)

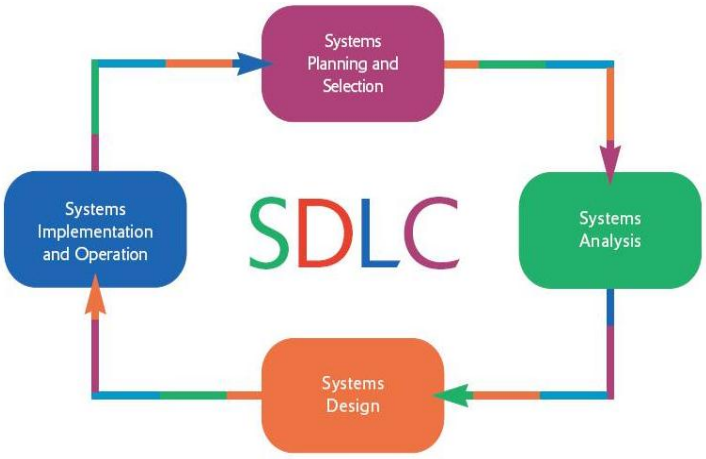
[27] H.L. Gantt, *Work, Wages and Profit,* 1910



FIGUUR 8 GANNT CHART VAN DE PLANNING

4.5 WERKWIJZE

Zonder een vaste werkwijze is het moeilijker om grip te bewaren op het project. Er ontstaat dan sneller een onzekere situatie en het is lastiger om aan de planning te houden. De sprints die Scrum definieert helpen al met een werkwijze te bepalen, maar ook binnen de sprints kan opnieuw een werkwijze bepaald worden. Tijdens dit project wordt per sprint achtereenvolgens een analyse en een ontwerp gemaakt. Daarop volgt de implementatie, het testen gebeurt ook tijdens de implementatie. Deze cyclus kijkt af bij SDLC (Systems development life cycle)[28]. Zie Figuur 9.



FIGUUR 9 SCHEMATISCHE WEERGAVE VAN EEN SDLC CYCLUS

4.5.1 REQUIREMENTS ANALYSE

In een requirements analyse worden de eisen van het product vastgesteld, de eisen vormen de basis voor user stories en kunnen opgedeeld worden in functionele en niet functionele eisen. Een functionele eis geeft het gewenste gedrag van het systeem aan; wat moet het systeem kunnen bereiken. Een niet-functionele eis is een kwaliteitseis waar het systeem aan moet voldoen, deze gaan vaak over het gebruik of uiterlijk. Functionele eisen worden vertaald naar user stories en in de product backlog geplaatst. Niet-functionele eisen hebben meestal geen plaats in de product backlog en worden apart bijgehouden.

Voor een sprint van start kan gaan moeten eerst items uit de product backlog geanalyseerd worden. Deze eerste analyse kijkt naar de omvang en complexiteit van de features. Dit is nodig om te schatten hoeveel tijd het implementeren van de verschillende items zal kosten. Met deze informatie kunnen er items in de sprint worden geplaatst, het is hierbij belangrijk om de sprint zo in te delen dat de features voor die sprint aan het eind af zijn. Indien alle geplande items zijn afgerond voordat de sprint is afgelopen kunnen er extra items uit de backlog gehaald worden. Deze tijd kan ook gebruikt worden extra kwaliteitscontrole uit te voeren of aan andere zaken te werken die toekomstig werk verlichten.

Voor het prioriteren van de items word gebruik gemaakt van de MoSCoW methode[29]. Deze methode kent een prioriteit toe in de vorm van categorieën. Elk item komt in één van de volgende vier categorieën:

- **M**ust have. Deze functie MOET in het product zitten. Dit is een vereiste functie, het product is geen succes tot alle Must have's erin zitten.
- **S**hould have. Hoge prioriteit, dit is een belangrijke vereiste, maar kan op andere manier ingevuld worden indient strikt noodzakelijk.
- **C**ould have. Deze vereist is gewenst, maar niet noodzakelijk. Deze items worden ingevuld als de tijd het toe laat.
- **W**on't have. Er is overeenstemming dat dit item niet in het eindproduct zal zitten.

De must have items worden als eerste in de sprints geplaatst. Het is belangrijk om deze features eerst af te werken. Dit betekent niet dat een 'should have' of zelfs 'could have' item niet in de sprint kan komen, dit

[28] http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle
[29] http://www.coleyconsulting.co.uk/moscow.htm

gebeurt juist wel als er een samenhang is met een must have item. Zodra een feature daadwerkelijk ontworpen moet worden zal er opnieuw een analyse plaats vinden. Hiervoor moet er eerst informatie verzameld worden. Dit keer is het doel van de analyse om de feature goed te begrijpen en een ontwerp te maken voor de implementatie. Als er meerdere implementaties mogelijk zijn moet onderzocht worden welke het beste past.

Een volledig overzicht van alle functionele en niet functionele eisen is te vinden in bijlage [Analyse document].

4.5.2 ONTWERP

Items uit de product backlog moeten ontworpen worden, dit gebeurd per sprint opnieuw. Er kunnen twee typen ontwerpen worden gemaakt. Een functioneel ontwerp en een technisch ontwerp. Een functioneel ontwerp zegt iets over *wat* het systeem kan, een technisch ontwerp specificeert *hoe* dit bereikt wordt. Een functioneel ontwerp gaat altijd vooraf aan een technisch ontwerp. De ontwerpen zijn een model van de voorgenomen werkelijkheid.

Het maken van modellen kan helpen bij de implementatie indien deze niet triviaal is. Het gebruik van een feature kan bijvoorbeeld gemodelleerd worden, of de samenhang met andere klassen. Zulke modellen worden gemaakt volgens de UML (unified modelling language)[30] specificatie. UML definieert een hele reeks standaard modellen voor verschillende situaties. Met een zodanig model kan een deel van het systeem formeel worden uitgedrukt. UML is onafhankelijk van de ontwikkelmethode maar is wel meest geschikt voor use-case gedreven, iteratief en incrementeel gedreven ontwikkeling.

Er wordt in dit project gebruik gemaakt van de volgende vijf typen UML diagrammen:

- Use case diagram
- Sequence diagram
- Activity diagram
- Class diagram
- Package diagram

Deze diagrammen zijn voldoende om de componenten die in dit project gemaakt worden te beschrijven.

4.5.2.1 FUNCTIONEEL ONTWERP

Een functioneel ontwerp geeft weer welke functies het systeem bezit en op welke manier de dialoog tussen gebruiker en systeem verloopt. Dit type ontwerp is bestemd voor de gebruikers van het systeem (product owner in dit geval). Zij kunnen zien hoe hun eisen zijn vertaald naar een systeem en kunnen aangeven of zij het hiermee eens zijn.

Voor een functioneel ontwerp worden use case en activity diagrams gemaakt. Zo nodig kunnen er ook ontwerpen voor user interfaces gemaakt worden.

Een use case diagram beschrijft op een zo simpel mogelijke manier wat een gebruiker met het systeem kan bereiken. De user stories kunnen direct vertaald worden naar use case diagrams.

Een activity diagram geeft een flow, of volgorde, weer. Dit is een stapsgewijs schema met activiteiten en keuzes. Hierin wordt als het ware een ‘pad’ gevolgd van begin situatie naar eindresultaat. Een activity diagram komt voort uit één case van een use case diagram.

[30] http://www.uml.org/

Alle use case diagrammen zijn te vinden in bijlage [Analyse document].

4.5.2.2 TECHNISCH ONTWERP

Een technisch ontwerp is nodig om de implementatiefase goed te laten verlopen. Het doel van een technisch ontwerp is vooraf vaststellen wat de beste aanpak is om deze feature te implementeren. Hiervoor moet opnieuw informatie ingewonnen worden over de aard van de feature is, welke onderdelen van het systeem er bij betrokken zijn en hoe de nieuwe feature geïntegreerd zal worden. Niet alleen helpt dit bij de implementatie, maar dient ook later als referentie materiaal als er wijzigingen doorgevoerd moeten worden, of introductie materiaal voor nieuwe teamleden.

Een technisch ontwerp beschrijft hoe een functioneel ontwerp wordt verwezenlijkt. Deze ontwerpen zijn bestemd voor ontwikkelaars, gebruikers hebben niks met deze diagrammen te maken.

Technische ontwerpen worden in dit project gemaakt d.m.v. class, sequence en package diagrams.

Een class diagram is bedoeld om de structuur van de software te beschrijven. Van de relevante klassen wordt opgenomen welke functies en variabelen ze bezitten en nog belangrijker; hoe zij samenhangen met andere klassen. Dit is een belangrijk hulpmiddel voor de implementatie, er wordt zo een goed overzicht behouden.

Sequence diagrams modelleren hoe een methode samenwerkt met andere methoden. Dit diagram laat interactie zien tussen objecten en welke communicatie er plaats vind. Een sequence diagram laat ook zien in welke volgorde alle operaties plaatsvinden. Dit kan opnieuw bijdragen aan het overzicht tijdens de implementatie fase.

4.5.3 IMPLEMENTATIE

In de implementatiefase wordt het technisch ontwerp gerealiseerd, deze fase kost gewoonlijk de meeste tijd. Hierin worden de componenten van het systeem geprogrammeerd en tevens getest.

Hoewel de complete analyse-ontwerp-realisatie cyclus voor ieder user story opnieuw wordt doorlopen is er tijdens de implementatiefase vaak de meeste overlap, er wordt dan vaak aan meerdere user stories tegelijk gewerkt. Tijdens de bouw van componenten moet er namelijk al veel rekening gehouden worden met de functionaliteit die andere user stories ook aan het systeem gaan toevoegen.

5 SPRINT 1 "ANALYSE"

In de analyse fase moet het project gedefinieerd worden. Het oorspronkelijke idee wordt omgezet in een *productvisie.* Dit betekent dat het probleem geanalyseerd moet worden en er een oplossing voorgesteld moet worden. Er wordt gekeken naar de inhoud, de haalbaarheid, projectgrenzen, risico's e.a. Doorgaans is budgettering een belangrijk onderwerp tijdens de analyse, maar daar is in dit geval geen sprake van. Als alle project parameters zijn ingevuld worden deze opnieuw geanalyseerd en aangepast tot het project een vorm heeft waar alle betrokken partijen het mee eens zijn. In deze fase wordt nog weinig aan het product gewerkt, het gaat hier om het volledig definiëren van de opdracht. De werkzaamheden van de volgende sprints hangen af van deze eerste sprint. De analyse wordt in één sprint afgerond.

De volgende criteria zijn opgesteld voor de eerste sprint.

- Bepalen van de software die gebruikt zal worden.
- Er is overeenstemming over de projectinrichting en aanpak. Alle onzekerheden zijn zo mogelijk weggewerkt.
- Er is overeenstemming over terugkoppeling van de voortgang.
- De omvang van het project is gedefinieerd.
- De risico's van het project zijn in kaart gebracht.
- Alle deliverables zijn opgeleverd en zo nodig goedgekeurd.

De volgende delivarables zijn achtereenvolgens in de analyse opgeleverd.

- De product backlog met geprioriteerde user stories.
- De planning.
- Het plan van aanpak.
- Er zijn functionele ontwerpen voor de belangrijkste use cases.
- Een analyse document met daarin de uitgewerkte requirements analyse

5.1 SOFTWARE

Het Windows 7 Enterprise besturingssysteem wordt gebruikt. Het ontwikkelen wordt gedaan met Microsoft Visual Studio 2010 Premium[31], het testen van de applicatie gebeurt hier ook mee. Hiernaast worden de ASP .NET MVC libraries geïnstalleerd. Voor version control wordt Microsoft Team Foundation Server[32] gebruikt. Verslaglegging, documentatie en projectdocumenten worden gemaakt met behulp van de Microsoft Office Professional Plus 2010. De planning wordt in Excel gemaakt. Communicatie en agendering verlopen via Microsoft Outlook 2010. Het testen van de client interface van de applicatie gebeurt hoofdzakelijk met Opera, daarnaast wordt Firefox ingezet wanneer nodig. Voor het debuggen van de JavaScript code en de opmaak van de interface zal Opera Dragonfly[33] worden gebruikt. UML diagrammen worden gemaakt met Software Ideas Modeler[34]. Een installatie van Unify SQLBase[35] zorgt voor communicatie met de testdatabase. Met SQLTalk kan de database direct benaderd worden, dit kan van pas komen om te bevestigen dat bepaalde data bestaat, is verwijderd, opgeslagen enz.

[31] http://www.microsoft.com/visualstudio/
[32] http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx
[33] http://www.opera.com/dragonfly/
[34] http://www.softwareideas.net/
[35] http://www.unify.com/Products/Data_Management/SQLBase/default.aspx

5.2 PROJECTINRICHTING

Alle parameters van de projectinrichting moeten op orde zijn voordat de uitvoering van het project kan beginnen. Tijdens de analyse fase wordt het project management bepaald, de ontwikkelmethode gekozen en de geschikte technieken voor het probleem uitgekozen. Er worden methoden gekozen voor risico analyse, kwaliteitscontrole en het testen.

5.2.1 TERUGKOPPELING VOORTGANG

Door het terugkoppelen van voortgang kan de opdrachtgever inzicht behouden in het verloop van het project. Omdat de sprints binnen Scrum telkens functionele iteraties opleveren is het voor de opdrachtgever mogelijk om gaandeweg te zien hoe het eindproduct zich ontwikkeld. Indien er nieuwe inzichten of ideeën opgedaan worden kunnen deze door de flexibiliteit van Scrum altijd binnen twee weken opgenomen worden in het product. De terugkoppeling is niet alleen bedoeld om de opdrachtgever te informeren over het product. Ook (technische) belemmeringen en eventuele oplossingen kunnen voorgelegd worden zodat de opdrachtgever mee kan besluiten welke weg er ingeslagen moet worden.

De terugkoppeling met de opdrachtgever gebeurt formeel elke twee weken door middel van een vergadering en een demo van de laatst opgeleverde iteratie. Informeel kan er ook tussendoor overlegd worden. Indien gewenst wordt ook inzicht gegeven in de technische aspecten van het product. Dit kan door technische ontwerpen voor te leggen.

5.2.2 OMVANG PROJECT

Scrum maakt het mogelijk om te beginnen aan het ontwerp en zelfs realisatie van het product zonder dat de totale omvang van het product vastgelegd is. Strikt genomen hoeft er alleen een eerste sprint gespecificeerd te worden waarna de werkzaamheden kunnen beginnen. Toch is het voordelig om vooraf al zo veel mogelijk te definiëren. Hierdoor kan er al een globale indeling worden gemaakt voor de sprints. Juist omdat er onzekerheid is over de werking van Fabric is het moeilijk te schatten hoe lang het duurt om bepaalde features te implementeren. Met de product owner is afgesproken dat er vooraf een set features in de product backlog worden geplaatst en dat er gedurende het traject nog meer bijgeplaatst kunnen worden.

5.2.3 RISICOMANAGEMENT

Het vooraf vaststellen van de risico's heeft als voordeel dat er al vanaf het begin rekening mee gehouden kan worden. Als er grote risico's worden gevonden kan het project aangepast worden om deze risico's uit de weg te gaan, of er kunnen maatregelen worden opgesteld om de gevolgen tegen te gaan.

5.2.3.1 ONZEKERHEID

Een eenvoudige manier om de onzekerheid van het project te schatten is door een mate van onzekerheid toe te kennen aan een aantal factoren. In Tabel 2 is een beknopte contingentieanalyse opgesteld.

TABEL 2 CONTINGENTIEANALYSE

Situationele factoren	Aandeel in totale onzekerheid

Soort	Gradatie	
Projectgrootte	Klein	-
Mate van gestructureerdheid	Gestructureerd	-
Taakinzicht ontwikkelaars	Onvolledig	+
Deskundigheid ontwikkelaars	Gemiddeld	+/-

Er is aan het begin van het project tijd besteed aan het bepalen van de werkwijze, de planning en methoden. Alle uit te voeren taken zijn gestructureerd, maar er is geen inzicht in welke taken er precies bestaan. De deskundigheid van de ontwikkelaar is gemiddeld. Het project kent maar één ontwikkelaar en neemt zeventien weken in beslag, dit betekent dat het om een klein project gaat. De grootste onzekerheid binnen het project is het taakinzicht, voor de rest is het project relatief zeker.

5.2.3.2 RISICOANALYSE

Een risicoanalyse geeft weer welke risico's er bestaan met een schatting van de kans dat het risico optreedt en een schatting van de impact. Aan de hand hiervan kan worden bepaald welke maatregelen er genomen moeten worden om het risico te beperken. Als de risicoanalyse slecht uitvalt, moet het project aangepast worden, of kan het zelfs helemaal niet doorgaan. Aan elk risico wordt een kans en een impact parameter gekoppeld, zie Tabel 3. Beiden uit een schaal van [1-10], deze worden met elkaar vermenigvuldigd en dit getal stelt het risicofactor voor.

TABEL 3 PRIORITERING RISICO'S

Risicofactor	Prioriteit
1-15	Zeer laag
16-30	Laag
31-45	Normaal
46-60	Hoger dan normaal
61-75	Hoog
76-100	Kritiek

Er zijn een aantal risico's geïdentificeerd, in Tabel 4 staan de belangrijkste twee aangegeven. In bijlage [Analyse document] staan alle risico's verder uitgewerkt.

TABEL 4 RISICOFACTOREN

Onervarenheid	
Omschrijving	Er wordt gebruik gemaakt van een aantal nieuwe technieken waar ik nog niet mee bekend ben, deze onervarenheid kan voor vertraging zorgen.
Prioritering	Kans: 10, Impact: 4, Risicofactor: **40** **Normaal**
Toelichting	De Scrum methode en de Fabric, CSLA en ASP.NET MVC frameworks zijn bij mij nog niet volledig bekend.
Preventie	Onderzoek doen naar alle onbekende technieken en opzoeken wat de 'best practices' zijn.
Repressie	Extra tijd steken in leren nieuwe techniek, zo nodig planning opschuiven.

Onduidelijke scope	
Omschrijving	De scope van het project is niet goed vastgesteld.
Prioritering	Kans: 7, Impact: 7, Risicofactor: **49**

	Hoger dan normaal
Toelichting	Enerzijds bestaat het project uit het 'bewijzen' van het Fabric framework, anderzijds moet er wel een volledig werkende web applicatie opgeleverd worden. Er moet hiertussen een goede balans worden gevonden.
Preventie	Vroegtijdig met de opdrachtgever afspreken hoe de opdracht wordt ingedeeld.
Repressie	Opdracht herdefiniëren, disciplines beter afbakenen.

5.3 DELIVERABLES

5.3.1 PRODUCT BACKLOG

Om de eerste sprint in te vullen zijn er items nodig in de product backlog. Deze items komen voort uit de wensen van de gebruikers, die door de product owner worden omgezet in user stories. Hoe meer items er aan het begin van het project in de product backlog worden gezet, hoe meer inzicht er is in de totale omvang van het project.

Aangezien er geen sprake is van een groep gebruikers moet de product owner zelf wensen bedenken. Dit is gedaan tijdens een brainstorm sessie met de product owner en opdrachtnemer. De bedachte wensen zijn door vervolgens omgezet in user stories. De opdrachtgever heeft aangegeven dat de invulling van de applicatie ook door de opdrachtnemer verder ingevuld moet worden. Een kort onderzoek naar de features van andere CRM oplossingen heeft inzicht gegeven in de wenselijke features. Veel geavanceerde features zijn erg complex en kosten daardoor te veel tijd om te implementeren. Toch heeft het onderzoek ervoor gezorgd dat er nu meer duidelijkheid bestaat over de essentiële onderdelen van een dergelijke applicatie. Dit heeft geholpen bij het prioriteren van de user stories.

Uiteindelijk is er een eerste sprint ingedeeld met een klein aantal kern features die als eerst afgemaakt moeten worden voordat de andere user stories aangepakt kunnen worden. Zie Tabel 5.

TABEL 5 USER STORIES VOOR DE EERSTE SPRINT

User story	Prioriteit
Ik kan een totaaloverzicht opvragen van alle debiteuren.	**Must**
Ik kan gegevens van een debiteur bekijken.	**Must**
Ik kan gegevens van een debiteur wijzigen.	**Must**
Ik kan een nieuwe debiteur toevoegen.	**Must**
Ik kan een debiteur verwijderen.	**Must**

Een user story wordt verkort opgeschreven in de sprint indeling. Het eerst punt van de lijst komt bijvoorbeeld voort uit de user story "Ik kan een lijst bekijken van alle debiteuren die in het systeem zijn opgeslagen." Aan een user story kunnen nog niet-functionele eisen gekoppeld zijn. Zoals hoe het totaaloverzicht eruit ziet, en dat aan het verwijderen van een debiteur een bevestiging vooraf gaat.

5.3.2 PLANNING

De initiële planning zoals opgesteld in hoofdstuk **3.5 Globale planning** moet verfijnd worden. Doordat er nu meer informatie over de aard van het project bekend is, kan er specifieker gepland worden. Omdat er tijdens het ontwikkeltraject telkens nieuwe user stories worden toegevoegd aan de backlog wordt er in de planning niet gesproken over deadlines. Het ontwikkeltraject beslaat een periode van maximaal veertien weken, wanneer individuele user stories af moeten zijn wordt niet gespecificeerd. De uiteindelijke planning is te vinden in Figuur 8.

5.3.3 PLAN VAN AANPAK

Het plan van aanpak is af zodra alle hiervoor besproken punten zijn uitgewerkt en vastgelegd. Voordat het project naar de volgende fase kan gaan moet het plan nog door de opdrachtgever goedgekeurd worden. Tijdens een bespreking worden alle onduidelijkheden voor zover mogelijk opgehelderd. Het plan van aanpak voor dit project hoefde niet gereviseerd te worden en is direct goedgekeurd.

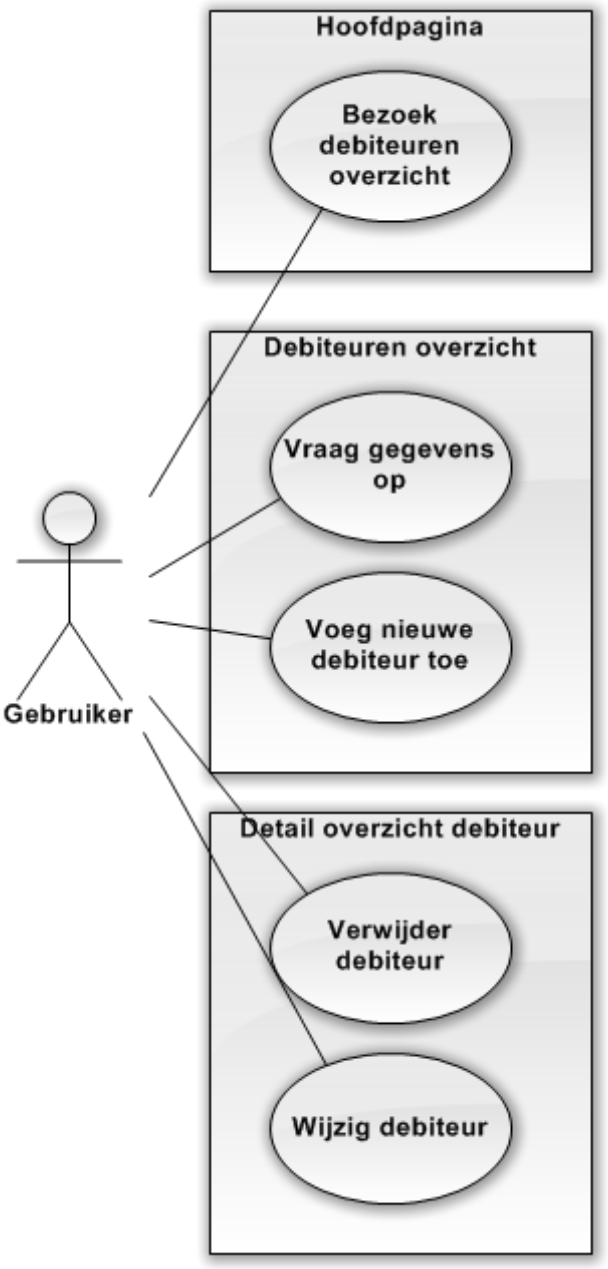
5.3.4 FUNCTIONELE ONTWERPEN

5.3.4.1 USE CASES

De eerste use cases gaan allemaal over het debiteuren overzicht. Er worden relatief eenvoudige operaties in opgenomen die nodig zijn voordat er complexe operaties toegevoegd kunnen worden. Deze use case vormt de basis voor volgende use cases. Er worden later nieuwe use cases toegevoegd, maar deze kan ook uitgebreid worden met meer cases.

Een desktop applicatie heeft altijd een beginscherm, vanaf dit scherm kunnen alle andere schermen worden bereikt (direct of indirect). Voor een web applicatie geldt dit niet, een gebruiker kan direct naar alle schermen navigeren, doorgaans begint wel op de hoofdpagina, maar het hoeft niet. Vanaf de hoofdpagina kan het debiteuren overzicht worden bezocht. Dit overzicht staat op een nieuwe pagina. Hier wordt een lijst weergegeven van alle debiteuren met daarbij wat algemene informatie. Wanneer de gebruiker een debiteur uitkiest krijgt hij op een nieuwe pagina gedetailleerde informatie te zien. Op deze pagina is het mogelijk om de debiteur te wijzigen of deze te verwijderen. Voor de verwijdering uitgevoerd wordt moet eerst een bevestiging worden gegeven.

In bijlage 8 zijn alle functionele diagrammen opgenomen.

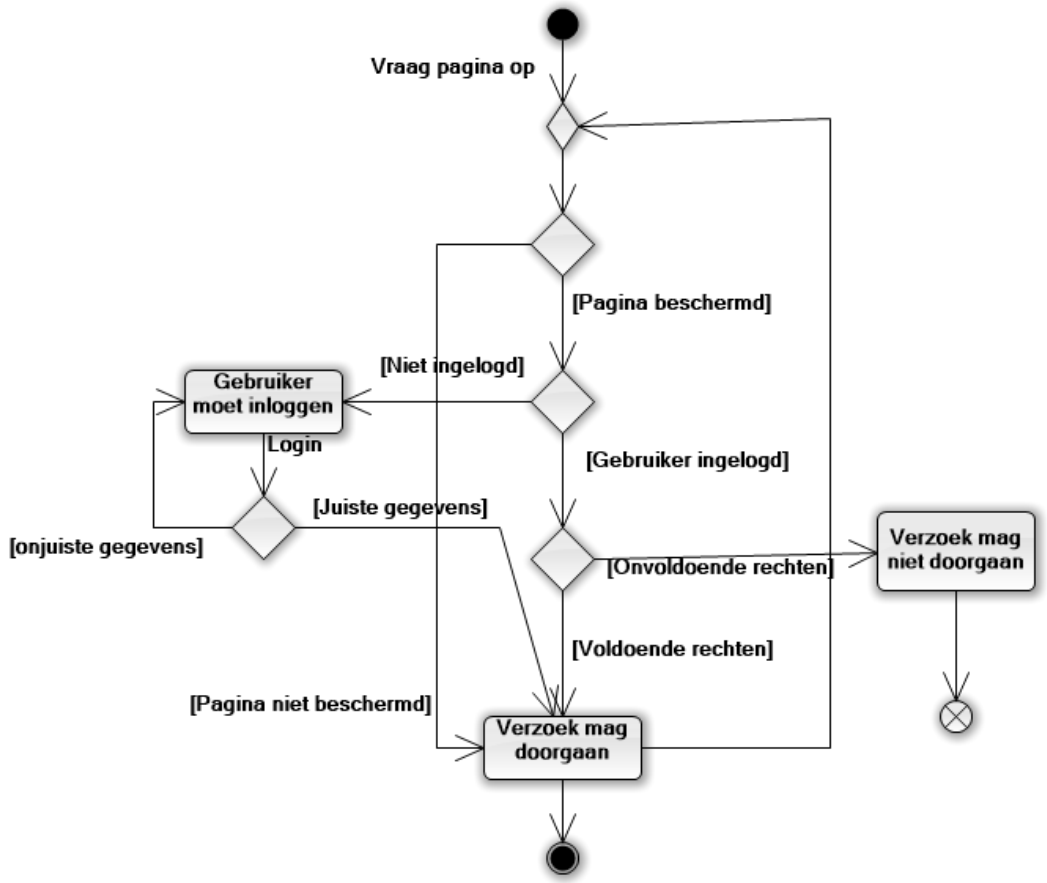


FIGUUR 10 USE CASES VOOR DE EERSTE SPRINT

5.3.4.2 ACTIVITY DIAGRAMS

Het eerste activity diagram geeft weer hoe het opvragen van een pagina verloopt. Er is in dit diagram (zie figuur 11) al informatie opgenomen over een gebruikers login en rechten systeem. Dit wordt nog niet in de eerste sprint geïmplementeerd. Het is wel belangrijk dat er tijdens het maken van het technisch

ontwerp rekening gehouden wordt met deze functionaliteit zodat deze later zonder problemen ingebouwd kan worden.



FIGUUR 11 ACTIVITY DIAGRAM PAGINA OPVRAGEN

Als een gebruiker een pagina opvraagt wordt eerst gekeken of er authenticatie nodig is voor deze pagina. Dit is voor pagina's zoals de hoofdpagina niet nodig, maar wel voor het opvragen van gevoelige data. Als de pagina beschermd is en de gebruiker is niet ingelogd worden deze naar de inlog pagina verwezen. Hier moet hij inloggen met geldige *credentials.* Wanneer de gebruiker ingelogd heeft worden de rechten van die gebruiker gecontroleerd. Aan de hand hiervan wordt bepaald of de pagina getoond mag worden of niet. Indien de gebruiker onvoldoende rechten heeft wordt deze verwezen naar de fout pagina, vanaf hier kan opnieuw een pagina worden opgevraagd.

5.3.5 TEST PLAN

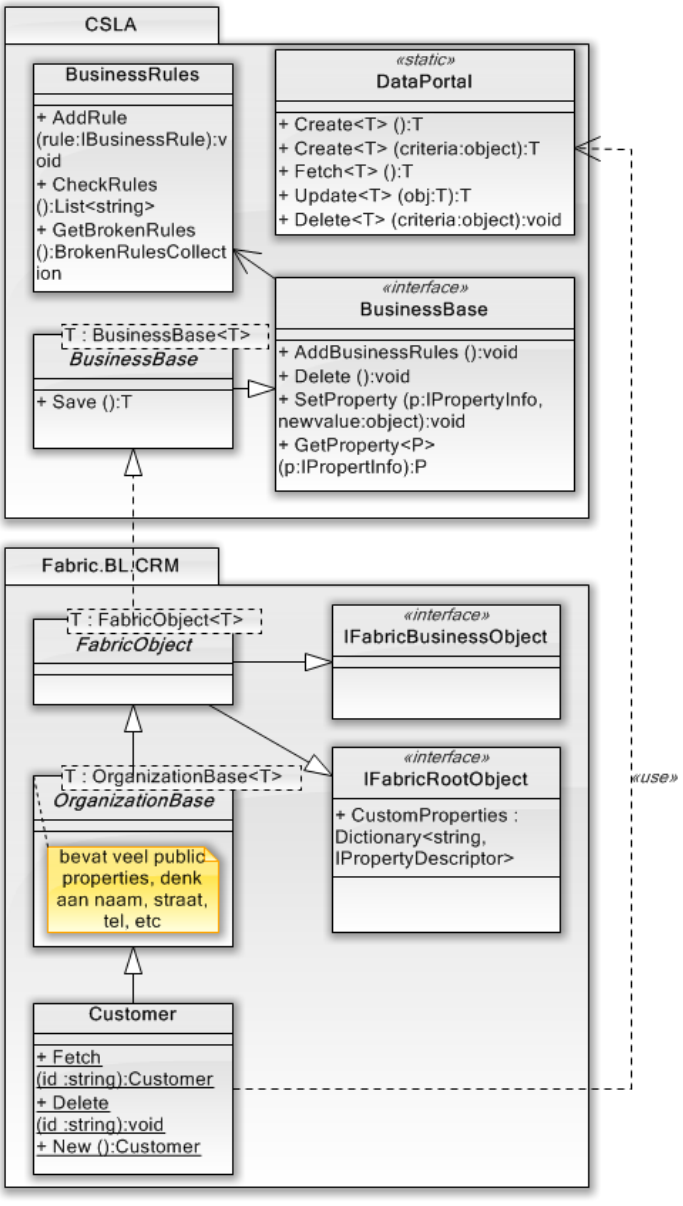
Een test plan geeft aan op welke manier(en) de applicatie getest zal worden. Dit project zal gebruik maken van unit tests[36]. Unit testing heeft zich bewezen als een zeer robuuste manier van het continu controleren van de correcte werking van een software systeem. Unit testing houdt in dat alle units (vrijwel altijd functies) van de code in afzonderlijke tests worden getest. Als alle units in een systeem hun unit test succesvol doorlopen is er veel zekerheid over de correcte werking. Het slagen van alle tests betekent nog niet dat het systeem gegarandeerd juist werkt. Unit tests moeten afzonderlijk en in willekeurige volgorde uitgevoerd kunnen

[36] http://www.extremeprogramming.org/rules/unittests.html

worden. Om dit te faciliteren wordt er gebruik gemaakt van *mocks*[37]. Een mock is een nep implementatie van een afhankelijkheid. Het is bijvoorbeeld onhandig dat sommige tests niet zouden werken als er geen verbinding is met een database. De database afhankelijkheid kan in dat geval ge-mocked worden. Er wordt dan een simpele nep implementatie voor in de plaats gezet die goed genoeg is om tests mee uit te voeren. Als alle componenten worden gesteund door een unit test kan bestaande code ook met meer vertrouwen worden aangepast. Vaak bestaat er de angst om een fout te introduceren door een bepaalde wijziging te maken. Als alle unit tests nog slagen na het wijzigen van een component is er zekerheid dat het systeem nog naar behoren werkt.

Unit tests dragen bij aan het verzekeren van de correctheid van individuele componenten van een systeem. Om zekerheid te krijgen over de werking van het *totale* systeem moeten er andere soort tests worden ingezet. Een systeemtest kan dit soort zekerheden geven. Een systeemtest combineert de werking van verschillende componenten, meestal om een bepaalde taak te volbrengen. Hiermee wordt de samenwerking van het systeem getest en niet alleen de onderdelen apart.

De vorige twee test typen richten zich op de software, een acceptatietest[38] richt zich op de gebruiker. Eindgebruikers gaan hierbij zelf aan de slag met het systeem en proberen taken te volbrengen. De gebruiker moet zelf aangeven of het systeem de gewenste en juiste uitvoer geeft.



FIGUUR 12 CLASS DIAGRAM VAN CUSTOMER EN RELEVANTE KLASSEN

5.4 VOORONDERZOEK TECHNIEKEN

Voordat de implementatie kan beginnen moet er eerst onderzoek worden gedaan naar de technieken die gebruikt gaan worden. Het is belangrijk om een goed idee te hebben van zowel de werking van als het werken met Fabric. Daarnaast is het ook nodig om te verdiepen in het ontwerp van een idiomatische ASP .NET MVC applicatie.

5.4.1 FABRIC

De functionaliteit die Fabric biedt is bekend. Het is nu zaak om te onderzoeken *hoe* er met Fabric gewerkt moet worden. Omdat we als eerst gaan werken met het Debiteur object gaat het vooronderzoek uit naar de implementatie van de Debiteur en de samenhang ervan met andere klassen. De begeleider van het project heeft de interne werking van Fabric

[37] http://en.wikipedia.org/wiki/Mock_object
[38] http://www.extremeprogramming.org/rules/functionaltests.html

toegelicht. In Figuur 12 staat een overzicht van de hiërarchie van een Debiteur object (Customer). Van de beschreven klassen zijn alleen relevante methoden opgenomen.

De Customer klasse bevindt zich in de CRM module van Fabric (zie Figuur 12). Hier staan klassen die te maken hebben met *customer relations.* De Customer leidt af van *OrganizationBase,* deze bevat velden voor onder andere adres gegevens en persoonsgegevens. De Customer specialiseert deze abstracte klasse en voegt daarbij bankgegevens en betalingsgevens toe. Voor het aanmaken van nieuwe business objects wordt het Factory Method[39] pattern toegepast. Het ophalen, wijzigen en verwijderen van instanties gebeurt ook via *static* methoden. Deze methoden maken op zich weer gebruik van de static DataPortal klassse. Het DataPortal abstraheert communiceert met de database.

De *BusinessBase* staat aan de basis van de Customer klasse. Hierin worden *BusinessRules* opgenomen. Een BusinessRule forceert contracten, zo mag het id van een customer bijvoorbeeld geen `null` zijn en zijn er ook beperkingen op andere velden.

In zogenaamde NVL's (Name-Value List) worden lijsten bijgehouden van de id's van instanties van objecten. Zo kan de CustomerNVL gebruikt worden om een lijst van alle id's van Customers op te halen.

In C# is het mogelijk om klassen en algoritmen te schrijven die opereren op nader-te-bepalen datatypen. Een dergelijke klasse of methode heet een *generic* (template in C++). In tegenstelling tot C++ is het mogelijk om constraints te stellen aan het *generic* argument van een klasse, de compiler controleert dan of het generic argument voldoet aan de gestelde constraints. Dit maakt het programma robuuster. Er wordt in Fabric veel gebruik gemaakt van de volgende constructie:

```
class Derived<T> : Base<T> where T : Derived<T> { /* ... */ }
```

CODE 3 GENERIC TYPE CONSTRAINT

Dit patroon, waar de *generic parameter* van een klasse hetzelfde type moet zijn als die klasse zelf wordt ook wel CRTP (Curiously recurring template pattern) genoemd[40]. Het gebruik van dit patroon zorgt voor een restrictievere implementatie van een base class maar gaat ten koste van de abstractie.

5.4.2 ASP .NET MVC

ASP .NET MVC is een van de steunpalen van dit project. Een web applicatie wordt hierin beschreven als een groep van combinaties van een model, een view en een controller (MVC). Elk onderdeel (kleine groep samenhangende pagina's) van de applicatie wordt aangedreven door een MVC architectuur. Een model is een representatie van een bepaald aspect van de applicatie, vaak zijn het modellen van dingen in de echte wereld, zoals een product of klant. Een controller verzorgt de communicatie over en weer met de gebruiker. Een gebruiker kan aanpassingen maken in het model, welke door de controller worden doorgevoerd. Een view is de terugkoppeling naar de gebruiker. In essentie communiceert de gebruiker met de controller. Deze laadt modellen en past deze zo nodig aan. Het resultaat, of antwoord, wordt naar een view vertaald. De gebruiker krijgt deze view te zien. Zie Figuur 13.

Er wordt gewerkt met *routes,* dit is een beschrijving van de logische layout van de web applicatie. In eerste instantie wordt de volgende route gebruikt voor de eerste controllers.

`{websitenaam}/{controllernaam}/{operatienaam}/{parameters}`

[39] Design Patterns: Blz. 107.
[40] http://en.wikipedia.org/wiki/Curiously_recurring_template_pattern

Het bekijken van de gegevens van debiteur nummer 123 kan bijvoorbeeld op de volgende manier:

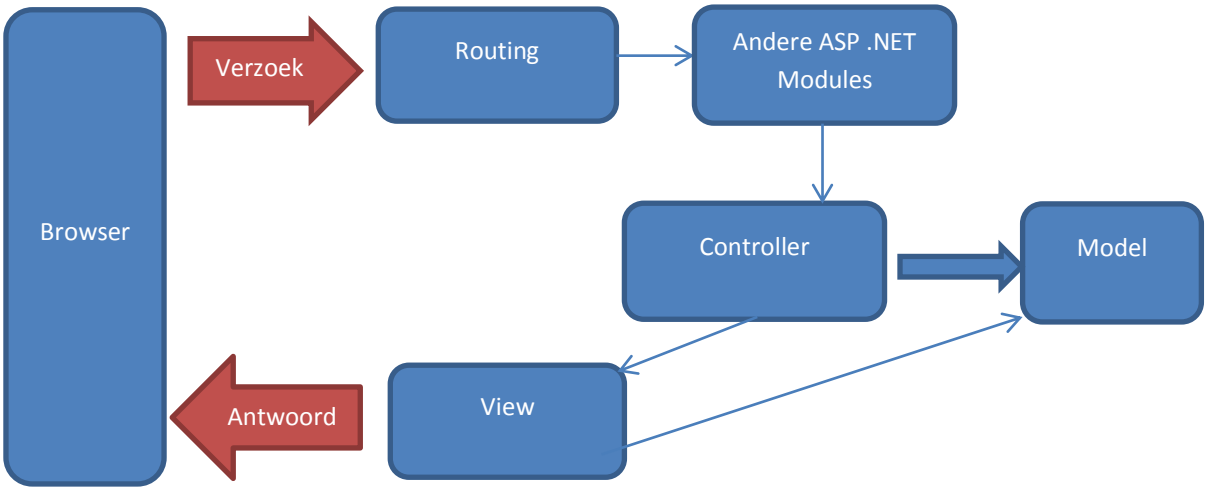
fabric.nl/customer/details/123

Een lijst opvragen van alle facturen zou op deze manier kunnen:

fabric.nl/factuur/index

Er vindt in het laatste geval geen operatie plaats op een specifiek item, er hoeft dus geen id meegegeven te worden. De routes worden zo veel mogelijk volgens de REST[41] principes ingedeeld.

Verschillende onderdelen van de applicatie worden door andere Controllers afgehandeld. Het is gebruikelijk dat elke operatie een eigen View heeft. Een web applicatie heeft meerdere Controllers, en elke Controller kan meerdere Views aansturen. Figuur 13 toont een schematische weergave van de communicatie tussen te verschillende onderdelen van ASP .NET MVC.



FIGUUR 13 COMMUNICATIE BINNEN ASP .NET MVC

Een HTML file heeft geen betekenis in dit framework. Pagina's bestaan alleen in de vorm van een View in het geheugen van de applicatie en worden dynamisch opgebouwd. Meestal is er per View een apart Model. Een View bestaat uit een mix van code en HTML elementen (zie Code 4) en wordt geïmplementeerd op basis van een specifiek Model. De Razor View Engine[42] voert de code uit en bouwt een pagina. Deze pagina wordt naar de gebruiker gestuurd. Het framework is zeer uitbreidbaar, er bestaan talloze hooks en er is veel customization mogelijk.

```
<table>
@foreach (var customer in Model)
{
       <tr class="index-row">
           <td>@customer.id</td>
           <td>@customer.Name</td>
           <td>@customer.Adres.Street1</td>
           <td>@customer.Adres.Street2</td>
           <td>@customer.Adres.City</td>
       </tr>
}
</table>
```

CODE 4 TONEN VAN CUSTOMERS IN EEN <TABLE>

[41] http://www.ibm.com/developerworks/webservices/library/ws-restful/
[42] http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx

6 SPRINT 2 "INZET"

In deze sprint wordt voor het eerst daadwerkelijk geprogrammeerd. In de vorige sprint is een werkwijze bepaald die nu aangehouden wordt. Er zijn ook al enkele diagrammen gemaakt die nu gebruikt worden. Er wordt ook voor het eerst *hands-on* gewerkt met Fabric.

6.1 SPRINT INDELING

Omdat er nog onzekerheid bestaat over de precieze werking van Fabric en ASP .NET MVC is de sprint conservatief ingedeeld. De applicatie krijgt nu voor het eerst vorm, dus het is belangrijk dat er voldoende tijd is om een goed ontwerp te maken. Als deze basis staat is het in volgende sprints mogelijk meer in te delen omdat er dan meer zekerheid is over de technieken waar de applicatie op berust en er is al een basis waar op verder gebouwd kan worden. Er is gekozen een kleine set van kern functionaliteit in de eerste sprint zetten, namelijk het debiteuren subsysteem. De indeling van sprint 2 is te zien op Tabel 6.

TABEL 6 USER STORIES VOOR DE TWEEDE SPRINT

User story	Prioriteit
Ik kan een totaaloverzicht opvragen van alle debiteuren	**Must**
Ik kan gegevens van een debiteur bekijken.	**Must**
Ik kan gegevens van een debiteur wijzigen.	**Must**
Ik kan een nieuwe debiteur toevoegen.	**Must**
Ik kan een debiteur verwijderen.	**Must**

Aan sommige items zijn nog niet-functionele eisen gekoppeld. Zo moeten de gegevens van een debiteur op een bepaalde manier zijn ingedeeld en moet er om een bevestiging worden gevraagd bij het verwijderen van een debiteur.

6.2 ANALYSE

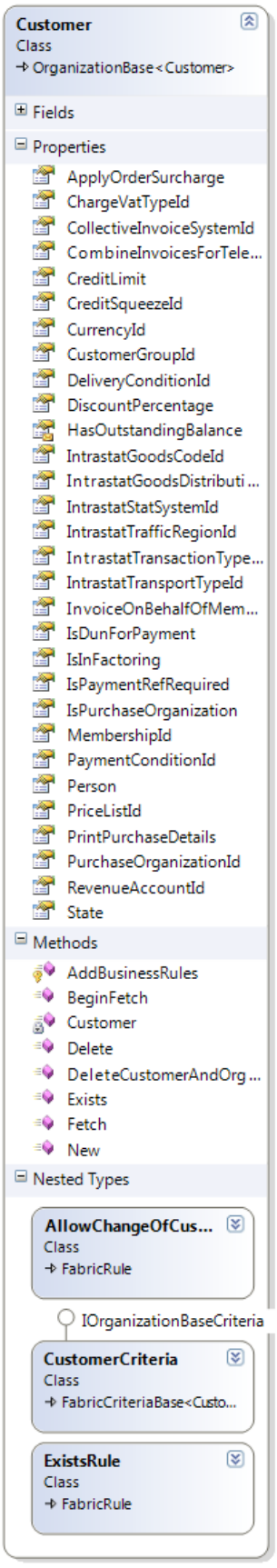
In de vorige sprint is de algemene publieke interface van Fabric onderzocht. In deze sprint worden de details van de Customer klasse onderzocht. Een algemeen begrip van de verschillende velden en functionaliteit is nodig voordat hier een web interface voor gemaakt kan worden. Om het systeem te kunnen testen is er test data nodig. Het is mogelijk om met het net ontwikkelde systeem nieuwe debiteuren toe te voegen en deze ook te gebruiken als test data voor de wijzig en verwijder functionaliteit. Een nadeel hieraan is dat het voor iemand zonder ervaring of expertise op gebied van boekhouding moeilijk zal zijn om data in te voeren op een reële manier. Het is wenselijk om het systeem te testen met de soort waarden waar het later mee om moet gaan en niet waarden die willekeurig zijn ingevuld. Dit betekent niet dat het systeem niet getest wordt met ongeldige input maar het is voor test doeleinden beter om specifieke test data te gebruiken. Er wordt daarom een relatief kleine database aangesloten die onder andere en verzameling verschillende Customers bevat. Het systeem wordt op correctheid getest met deze database. Wanneer in de toekomst de schaling van het systeem onderzocht moet worden zal een grotere database worden gebruikt.

6.2.1 FUNCTIONEEL ONTWERP

De use cases voor deze items zijn in de vorige sprint al gemaakt (zie Figuur 10). Een algemeen activity diagram voor het opvragen van pagina's is ook al gemaakt (zie Figuur 11). Er kan een activity diagram gemaakt worden

voor het uitvoeren van de items in de backlog, maar in dit diagram zou geen workflow naar voren komen, de taken zijn te elementair. Er is daarom gekozen dit activity diagram achterwege te laten.

De indeling van de weergave een debiteur is een niet-functionele eis. Deze wordt pas later in het project, wanneer er tijd voor is, afgemaakt. Vooralsnog is het voldoende als de gegevens van de debiteur achtereenvolgens op de pagina staan. Als een debiteur gewijzigd moet worden zijn alle velden bewerkbaar en staat er onderin een submit knop.



FIGUUR 14 KLASSE OVERZICHT VAN EEN DEBITEUR

6.3 TECHNISCH ONTWERP

Een debiteur wordt in Fabric gerealiseerd door de Customer klasse. Vanaf nu zal een debiteur Customer genoemd worden. Een Customer kan opgevraagd worden door de static `Fetch` methode (zie 5.4.1) aan te roepen en het id van de Customer mee te geven. De id's van alle Customers worden bijgehouden in de `CustomerNVL`, hier staan alle id's gekoppeld aan de naam van de Customer. Deze lijst kan vrij opgevraagd worden door opnieuw de `Fetch` methode te gebruiken.

In eerste instantie is het doel gesteld om zo snel mogelijk een toonbaar resultaat te bouwen. Het mag geen *proof of concept* heten, want alleen het totaaloverzicht en detail overzicht worden geïmplementeerd, de meest simpele items uit de sprint. Deze twee items zijn gekozen omdat het hier gaat om lees acties. Het wijzigen, toevoegen of verwijderen van een Customer brengt ongetwijfeld extra complexiteit met zich mee omdat het systeem dan aangepast wordt, een lees actie heeft (doorgaans) geen neven effecten. Daarnaast is het voor het wijzigen van een instantie hoe dan ook nodig om data op te halen.

Op Figuur 12 in het vorige hoofdstuk staat de samenhang van het het Customer object in Fabric. Figuur 14 (hiernaast) geeft weer welke methoden en velden de Customer *zelf* bezit, hierin zijn dus niet de methoden van base classes opgenomen.

Figuur 15 (volgende pagina) geeft het eerste klasse diagram weer van de customer controller. De `CustomerIndexController` zal het ophalen van álle `Customers` verzorgen, in de weergave worden slechts enkele gegevens weer gegeven. De `CustomerDetailsController` kan alle gegevens tonen van één Customer. Figuur 16 toont een sequentie diagram bij het opvragen van een customer.

6.4 IMPLEMENTATIE

De eerste implementatie van het ophalen van gegevens gebeurt buiten ASP .NET. Het is namelijk van belang om éérst de data op te kunnen halen en verwerken, pas daarna hoeft het getoond te worden in een web pagina. Door de werking van de controller te controleren in een unit test hoeft er geen view aangemaakt te worden. Dit zorgt er ook voor dat het testen sneller is, want de webserver hoeft niet telkens

geladen te worden. Zodra deze werking is getest *kan* het opnieuw getest worden in een daadwerkelijke view. In de praktijk is dit niet nodig want een view zou nooit iets moeten kunnen veranderen aan de weer te geven data.

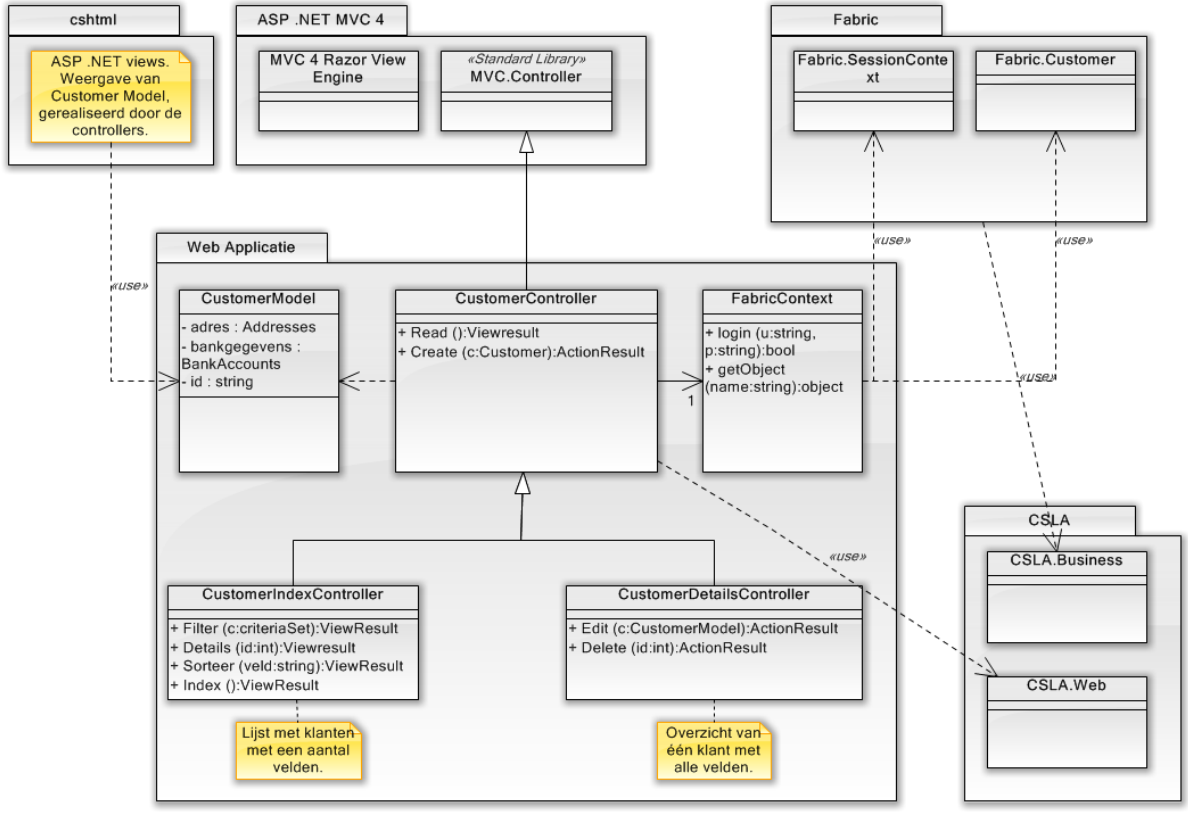
6.4.1 CUSTOMER

Zowel de Details als Index controller moeten een customer uit kunnen lezen, het is ook gewenst dat op beide pagina's een nieuwe customer aangemaakt kan worden. Hiervoor is een abstract base class met deze twee methoden gemaakt. De index controller voegt een Index methode toe. Deze haalt informatie op via Fabric en maakt daar een model van. Het model wordt doorgegeven aan de view. Voor het ophalen van de data wordt veel gebruik gemaakt van LINQ (zie 4.2.2.1). De volgende unit test haalt alle customers op en zet deze om naar een `CustomerModel`. Buiten deze code wordt er eerst via de FabricContext klasse ingelogd met een geldige gebruiker. Er is een kleine test database aangesloten.

```
IEnumerable<CustomerModel> Results;
var customerIds = Fabric.CRM.Lists.CustomerNVL.Fetch().Select(nvlPair => nvlPair.Key);
var customers = customerIds.Select(id => Fabric.CRM.Customer.Fetch(id));
Results = customers.Select(customer => new CustomerModel(customer));
Assert.IsTrue(Results.Any());
```

CODE 5 UNIT TEST: CUSTOMERS OPHALEN

Een `CustomerModel` geeft alleen een deelverzameling weer van alle velden uit een Customer, sommige velden zijn namelijk alleen bedoeld voor intern gebruik en hebben voor een gebruiker geen betekenis.

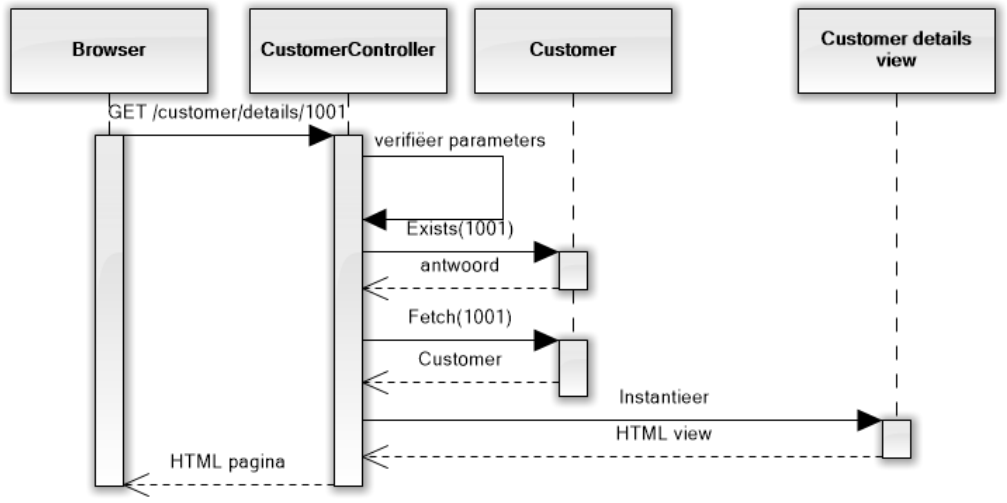


FIGUUR 15 KLASSEDIAGRAM VAN EERSTE ONTWERP CUSTOMER CONTROLLER

De unit test bewijst dat het ophalen en omzetten van Customer objecten werkt. Dit is nog maar het halve deel van de oplossing want de gegevens moeten ook getoond worden, hiervoor moet een View gemaakt worden. Deze het viewmodel hiervoor wordt een IEnumerable<CustomerModel>. Door een loop kunnen de velden van alle models worden weergegeven (zie Code 4).

Voordat deze pagina via een webbrowser opgevraagd kan worden moet het eerst aangesloten worden op het grotere geheel van de web applicatie. De code bestaat nu nog alleen als compleet losstaande unit test. Er moet een route aangemaakt worden die web-requests doorstuurt naar de juiste controller. De controller zelf moet ook gemaakt worden. Het gaat in dit geval om een relatief eenvoudige controller.

Wanneer alles is aangesloten wordt de lijst Customers goed getoond. Een probleem is dat er geen gebruik wordt gemaakt van opmaak. Dit zorgt ervoor dat de lijst Customers er onaantrekkelijk uitziet, er wordt alleen een lange lijst getoond. De data die nu getoond wordt is nog simpel, maar later in het project wordt de functionaliteit van het scherm telkens uitgebreid. Bovendien bestaat er een algemene niet-functionele wens dat de web applicatie er enigszins goed uit moet zien. Vroeg of laat moet er dus een opmaak worden gemaakt in HTML. Dit maakt het ook makkelijker om een demo te geven van het product aan het einde van de sprint. Het zelf vormgeven en bouwen van het uiterlijk van de website valt buiten de scope van het project, bovendien is er niet genoeg kennis paraat om de voor web applicatie een moderne vorm te geven. Daarom is er gekozen een bestaand template te gebruiker. Een *Masterpage* samen met een stylesheet verzorgen de layout en opmaak van de website. De masterpage is de pagina waarbinnen andere pagina's geladen worden, op de masterpage staat bijvoorbeeld de header en footer en andere sitenavigatie. De kern van de masterpage is leeg, in dit gebied komt de content van views te staan. Individuele views hoeven zich daarna alleen bezig te houden met content. Voor de detailpagina is hetzelfde proces gevolgd. Hierbij is het opgehaalde Customer object niet vertaald naar een CustomerModel, maar is de Customer zelf direct als model doorgegeven. Dit is nodig want (vrijwel) alle velden moeten getoond worden. Aan elke rij is nu een 'details' knop toegevoegd die naar de detailview leidt.



FIGUUR 16 SEQUENCE DIAGRAM BIJ HET OPVRAGEN VAN EEN CUSTOMER

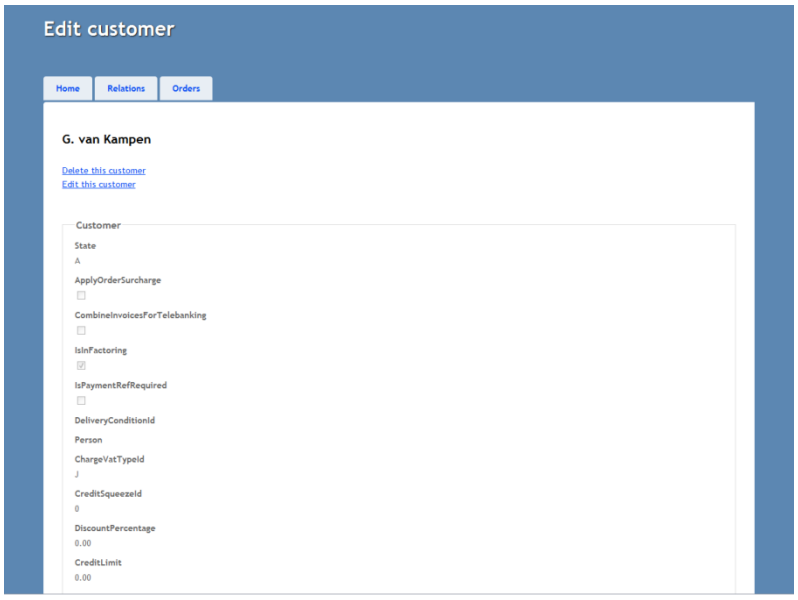
Nu de lijst customers getoond kan worden en ook de detailview functioneert, kan complexere functionaliteit worden toegevoegd. Het aanmaken van een nieuwe Customer wordt als eerst aangepakt. De gebruiker moet op de web pagina waarden invullen in een *webform,* deze worden verstuurd komen ze bij de Controller aan in de vorm van een `formCollection`, dit is een verzameling waarden uit het webform. De controller kan via de static `Create` methode een nieuwe `Customer` aanmaken en de waarden uit de `formCollection` toewijzen

aan het object. Dit heeft als gevolg dat de code bestaat uit een lange lijst toewijzingen. Als later namen van velden veranderen binnen een Customer werkt de code niet meer. Dit is een reële mogelijkheid want Fabric is nog in ontwikkeling. Doordat de web applicatie gebruik maakt van de DLL's die Fabric uitmaken en niet gebouwd wordt op de *source code* van Fabric kan de applicatie crashen wanneer deze DLL's laadt die conflicteren met het verwachte gedrag van Fabric. Een betere oplossing is om het form direct te laten *binden* aan het Customer object. Het ASP .NET framework kent een ModelBinder die dit doet. Deze kan in de volgende gevallen de waarden van een form direct aan een object toewijzen:

1. De velden van de klasse zijn public.
2. De namen van de velden in de klasse komen overeen met de namen van de formvelden.
3. Er bestaat een parameter loze constructor.

Aan nummer 1 en 2 wordt voldaan, maar er bestaat geen parameter loze constructor. De standaard modelbinder of CSLA modelbinder kan daardoor niet werken. Er moet een nieuwe modelbinder gemaakt worden die het probleem oplost, dit zal later ook nodig zijn voor het *wijzigen* van Customers. De nieuwe binder leidt af van de `CslaModelBinder` en overschrijft de `CreateModel` functie. Daarnaast wordt er een nieuwe interface gedefinieerd die afdwingt dat een controller een instantie kan aanmaken van het object waar die controller over gaat (Customer in dit geval). De nieuwe modelbinder controleert of de controller waar het form voor bestemd is deze interface implementeert. Zo ja wordt het verzoek doorgestuurd naar die controller, hier wordt gecontroleerd of of er al een Customer bestaat met het meegegeven id, zo niet wordt er een nieuwe aangemaakt. Dit nieuwe object wordt dan meegegeven aan de `Create` methode. Deze geeft uiteindelijk de View terug. Eenzelfde proces wordt gevolgd bij het bekijken van een bestaande customer. Het customerId wordt ingegeven, de controller haalt de customer op uit de database en de relevante velden worden geladen in een `CustomerModel` welke door de view getoond wordt (zie Figuur 17).

Deze nieuwe modelbinder is ook nodig voor het aanpassen van Customers. Het principe is hetzelfde maar zodra het meegegeven id overeenkomt met een bestaand id wordt de daarbij horende Customer opgehaald, en de formvalues toegekend. Het toekennen van waarden aan de velden slaagt altijd, maar zodra de veranderingen opgeslagen worden vindt er eerst validatie plaats. Het valideren van input behoort nog niet aan deze sprint en wordt later behandeld. Het verwijderen van een Customer brengt ook geen problemen met zich mee en kan zo gebouwd worden. Na een heroverweging is besloten de Index en Details controllers weg te laten en de functionaliteit in één Controller te stoppen. Het onderverdelen in subklassen brengt niet genoeg voordeel met zich mee.



FIGUUR 17 EERSTE IMPLEMENTATIE DEBITEUREN DETAILS

6.4.2 SUPPLIER

De user stories voor deze sprint zijn gematigd ingedeeld. Doordat de implementatie voorspoedig is verlopen is er tijd over, er zijn daarom additionele user stories toegevoegd aan de sprint. Zie Tabel 7.

TABEL 7 TWEEDE SET USER STORIES VOOR SPRINT 2

User story	Prioriteit
Ik kan een totaaloverzicht opvragen van alle crediteuren	**Must**
Ik kan gegevens van een crediteur bekijken.	**Must**
Als ik een foutieve waarde invul in een veld, levert dit een foutmelding op.	**Must**

Dit zijn dezelfde operaties als eerder zijn gemaakt voor de Customer, alleen nu voor crediteuren. Een crediteur wordt in Fabric gerealiseerd door de Supplier klasse.

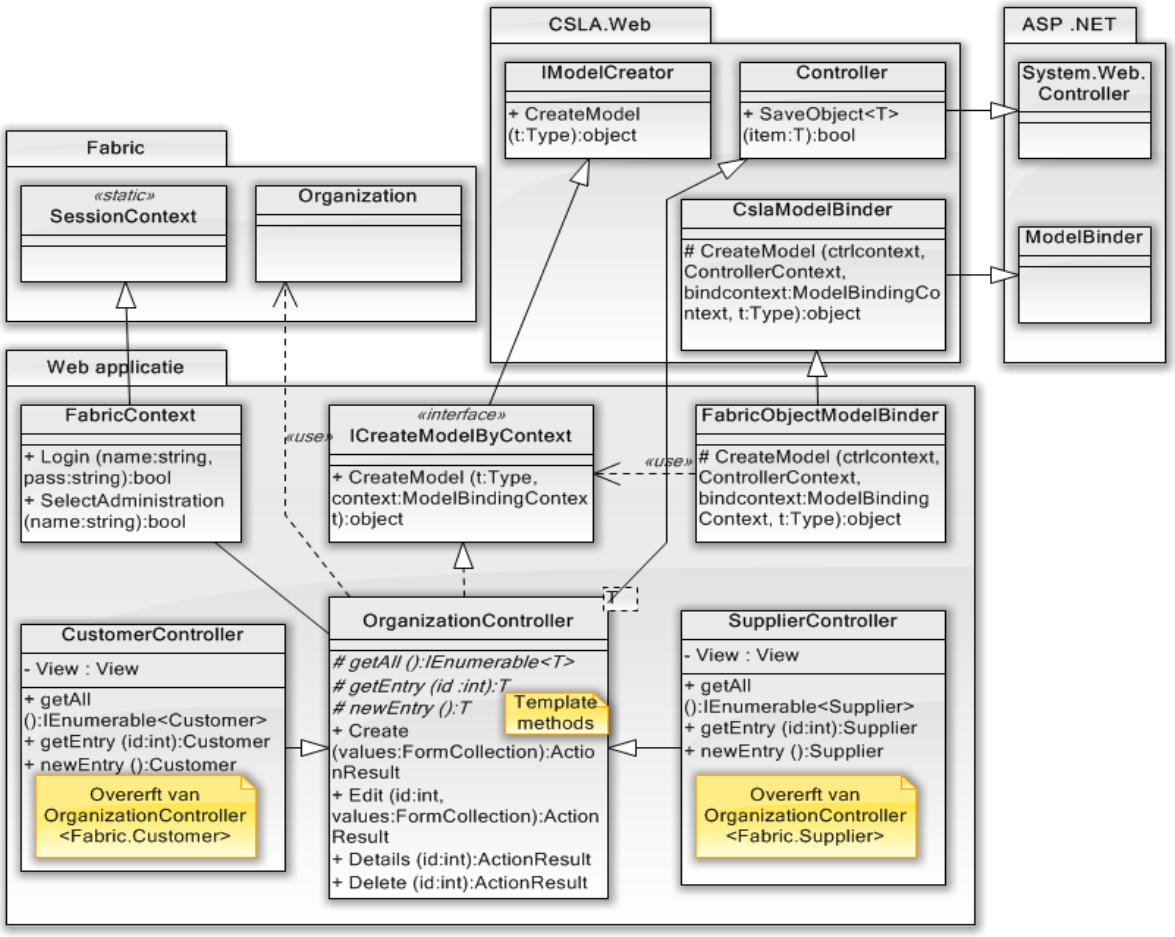
Op dit moment is een heroverweging gemaakt over het ontwerp van de controllers. De architectuur bevindt zich in een vroeg stadium, dus de impact van een verandering is klein. De Supplier en Customer klassen hebben veel gemeen, beiden ervan van de OrganizationBase. Het zou hierom mogelijk moeten zijn om een generieke OrganizationController te maken die beide klassen kan afhandelen.

Het uitvoeren van *CRUD*[43] functionaliteit is uitsluitend mogelijk via static methoden. Alleen een 'delete' operatie kan op een *instance* worden aangeroepen. Een standaard generic aanpak is daardoor erg moeilijk Het is namelijk niet mogelijk een static functie aan te roepen op een generic parameter zonder uitvoerig gebruik te maken van reflectie[44]. Het gebruik van reflectie is afgewezen om de volgende redenen:

- Reflectie kost relatief veel performance, het uitvoeren van een functie call die door middel van reflectie is bepaald kan een orde van grootte langer duren.
- Zeer complex.
- Static type checking op functie calls die via reflection zijn bepaald is niet mogelijk.

Er is daarom een ontwerp gemaakt dat gebruik maakt van het *template method* patroon[45]. Dit houdt in dat een base class zowel abstracte (puur virtueel) als geïmplementeerde methoden heeft. De geïmplementeerde methoden roepen abstracte methoden aan, deze moeten door subklassen geïmplementeerd worden. Op die manier kan een base class het algemene verloop van het algoritme bepalen terwijl subclasses hun eigen invulling geven. In dit geval moet het aanmaken van nieuwe, en het opvragen van bestaande Customers en Suppliers door subklassen ingevuld worden. De methoden van de base class handelen de rest van het werk af. Het ontwerp van het systeem is te zien op Figuur 18.

[43] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[44] http://en.wikipedia.org/wiki/Reflection_(computer_programming)
[45] Design Patterns pagina 325 en http://en.wikipedia.org/wiki/Template_method



FIGUUR 18 KLASSEDIAGREAM TWEEDE SPRINT

6.4.3 VALIDATIE

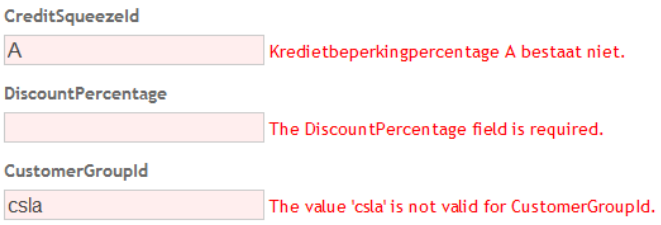
Business objecten definiëren een of meerdere business rules. Deze rules zeggen iets over de velden van het business object en moeten ervoor zorgen dat het object niet in een invalide staat komt. Sommige velden zijn vrij, maar anderen zijn begrensd. Dit kan betekenen dat er simpele beperkingen zijn op de lengte van de input, maar ook dat er uitvoerige controles worden gedaan op de samenhang van input van meerdere velden. De input van gebruikers moet dus gevalideerd worden. Het 'krediet' veld mag bijvoorbeeld uitsluitend nummers bevatten en is ook een verplicht veld.

Er is in eerste instantie onderzoek gedaan naar het valideren van input waarden. ASP .NET biedt geen hapklare oplossing voor het controleren van input, maar velden van een klasse kunnen wel aangepast worden zodat er alleen valide waarden aan toegewezen kunnen worden. Deze oplossing werkt door middel van een attribuut.

```
[Required]
[StringLength(100, MinimumLength = 4)]
[Display(Name = "Customer id")]
public string customerId { get; set; }
```

CODE 6 HET CUSTOMERID VELD MOET INGEVULD WORDEN, EN DE LENGTE IS TUSSEN DE 4 EN 100

Een nadeel van deze oplossing is dat de klasse waar de velden op van toepassing zijn aangepast moet worden. Dit betekent dat er in Fabric aanpassingen gemaakt moeten worden. Na overleg hierover bleek dat de validatie die in de business rules wordt uitgevoerd ook gebruikt kan worden. Hiervoor moet het systeem op de juiste manier worden ingericht zodat de validatieberichten uit Fabric op de goede manier door de controller worden geïnterpreteerd en daarna door de view op een duidelijke wijze worden teruggekoppeld. Het uiteindelijke resultaat is te zien in Figuur 19.



FIGUUR 19 VALIDATIE INVOERVELDEN

6.5 TERUGKOPPELING

6.5.1 VOORTGANG

Al een aantal dagen nadat de tweede sprint sprint begonnen was is er al kort een demo gegeven van de bereikte functionaliteit. Op dat moment was het totaaloverzicht en detailoverzicht van de debiteur al functioneel. De opdrachtgever heeft positief gereageerd op de functionaliteit en de algemene voortgang. Aan het eind van de sprint is er opnieuw een demo gegeven van de applicatie. Er is extra functionaliteit getoond, maar veel van de veranderingen sinds de eerste demo waren onzichtbaar voor de gebruiker, omdat het vooral veranderingen in de interne architectuur betrof.

6.5.2 FABRIC

De eerste sprint heeft een aantal belangrijke functies van Fabric gebruikt: het ophalen, tonen, aanmaken, wijzigen en verwijderen van business objects. Daarnaast is het resultaat van de validatie van business rules doorgegeven aan de Controller die ze daarna doorspeelt aan de View.

Er zijn een paar problemen met Fabric naar voren gekomen. Dit ging vooral om het implementeren van de web applicatie op de manier waarop Rockford Lhotka[46] (de ontwikkelaar van Csla) het beschrijft in zijn boek over Csla. Het is verstandig om de adviezen van Lhotka in acht te nemen bij het maken van de applicatie omdat Fabric erg nauw verbonden is met Csla. Door zijn adviezen op te volgen heb je meer vertrouwen in het ontwerp van je applicatie en kun je er zekerder van zijn dat er zich later geen onverwachtse incompatibiliteiten voordoen.

[46] http://lhotka.net/Area.aspx?id=1

7 SPRINT 3 "VERBREDING"

De derde sprint breidt uit op de vorige werkzaamheden. Het orderscherm wordt hier voor het eerst aangemaakt. Validatie van input wordt aangepakt, resultaten kunnen gesorteerd worden en er wordt zoekfunctionaliteit toegevoegd.

7.1 SPRINT INDELING

Er is nu meer vertrouwen in de kennis van Fabric en ASP .NET. Ook bestaat er nu een basis die uitgebreid kan worden. Het is aannemelijk dat het wijzigen/verwijderen/aanmaken van Supplier op dezelfde manier verloopt als een Customer. Zo ook zal een order object op een vergelijkbare manier werken. Daardoor zijn er in deze sprint meer user stories. In Tabel 8 User stories voor de derde sprint staan de user stories voor deze sprint.

TABEL 8 USER STORIES VOOR DE DERDE SPRINT

User story	Prioriteit
Ik kan een crediteur wijzigen.	**Must**
Ik kan een crediteur aanmaken.	**Must**
Ik kan een crediteur verwijderen.	**Must**
Ik kan een overzicht opvragen van orders.	**Must**
Ik kan een order aanmaken.	**Must**
Ik kan een order verwijderen.	**Must**
Ik kan een order wijzigen.	**Must**
Als de resultaten set groot is, wordt deze verspreid over meerdere pagina's.	**Must**
Ik kan de lijst debiteuren en crediteuren sorteren op naam.	**Must**
Ik kan het orderoverzicht sorteren op prijs, datum en debiteurnaam.	**Must**
Ik kan een debiteur op naam zoeken.	**Must**
Ik kan een crediteur op naam zoeken.	**Must**
Ik kan een lijst opvragen van alle orders bij een naam	**Must**
Ik kan via het debiteuren detail overzicht zien welke orders er bij deze debiteur horen.	**Must**
Wanneer ik begin met typen in het zoek veld van pagina, wordt er autocomplete ingezet.	**Should**

7.2 ANALYSE

7.2.1 CREDITEUR EN ORDER

De `SupplierController` moet uitgebreid worden zodat de functionaliteit ervan gelijk is aan de `CustomerController`. De `OrganizationController` base class is hiervoor al ingericht wat betekent dat alleen de nieuwe Views gemaakt hoeven worden.

De `OrderController` kan geen gebruik maken van de `OrganizationController` en moet nieuw ontworpen worden, op Figuur 20 staat een use case diagram voor deze controller.

7.2.2 PAGING

Er kunnen duizenden, zo niet tien- of honderdduizenden debiteuren, crediteuren en orders in de database zijn opgeslagen, wanneer deze opgevraagd worden is het praktisch gezien onmogelijk om deze allemaal tegelijk te tonen. Het is voor de gebruiker makkelijker als de resultaten over meerdere pagina's worden verspreid, bovendien wordt het systeem hierdoor ook véél minder belast. Het is hierbij belangrijk dat de resultaten per

pagina worden opgehaald uit de database, dat wil zeggen dat er alleen wordt opgehaald wat nodig is. De paging wordt dus *niet* toegepast door grote stukken van de resultaat set af te knippen, dit zou zeer inefficiënt zijn.

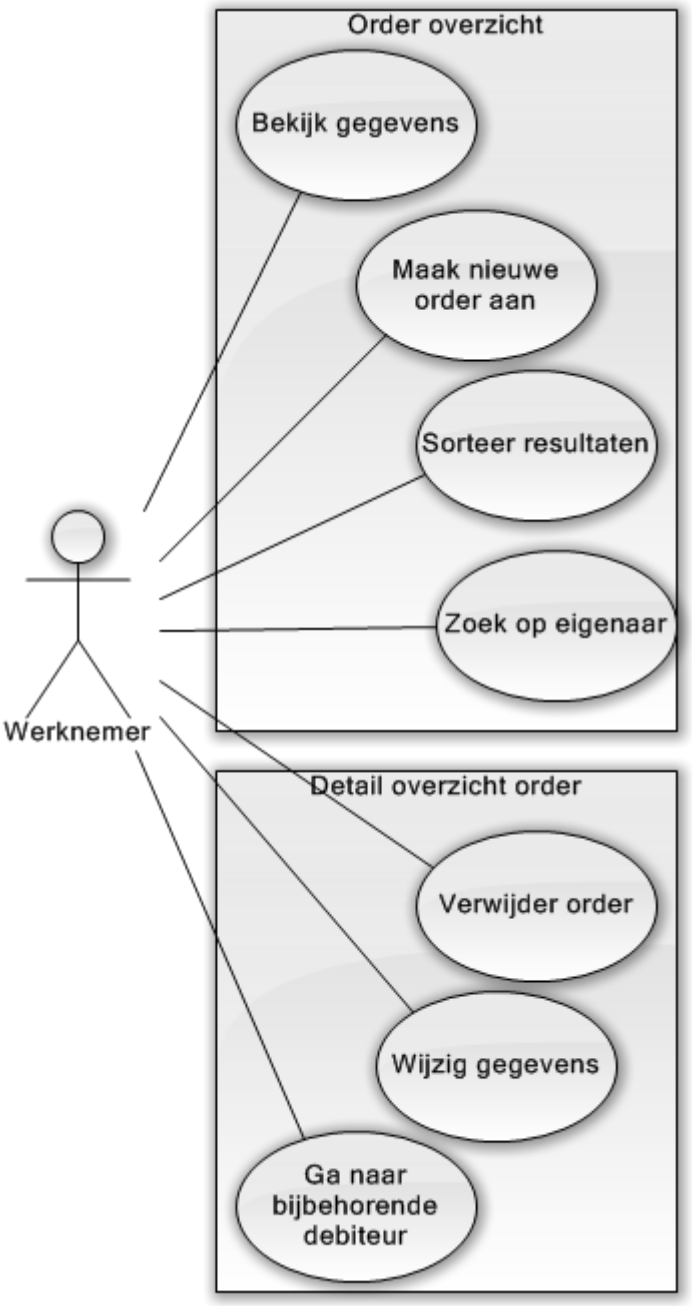
7.2.3 SORTEREN

Een belangrijk punt dat de implementatie moeilijk maakt is dat de ingevulde zoekopties en sorteeropties onthouden moeten worden tussen pagina's. Ook moet er een afweging gemaakt worden over het gedrag van de sorteer functionaliteit, moet de gebruiker teruggestuurd worden naar de eerste pagina als hij de sortering aanpast?

7.2.4 AUTOCOMPLETE

Autocomplete [47] helpt gebruikers bij het invullen van waarden in velden. Zodra een gebruiker begint met typen verschijnen er suggesties die de gebruiker kan selecteren.

Het implementeren van autocomplete vereist voor het eerst een stuk JavaScript. Het volledig implementeren van een nieuwe autocomplete oplossing valt buiten de scope van het project. Alleen het onderliggende systeem wat zorgt voor communicatie tussen het zoek veld en de database moet gebouwd worden. Het zelf realiseren van het visuele aspect is onnodig want hier bestaan al meerdere oplossingen voor. *Jquery* [48] wordt ingezet als JavaScript library om de transformatie van data naar autocomplete suggesties te faciliteren. JQuery is een uitgebreide JavaScript library.



FIGUUR 20 USE CASES BIJ HET ORDER SCHERM

7.2.5 PERFORMANCE

Tot nu toe werd het volledige `Customer` object doorgespeeld aan de Index View van de `CustomerController`. Dit object bevat veel meer data dan nodig is en duurt daarom langer dan noodzakelijk om op te halen. Fabric kent ook een `CustomerInfo` object, dit is een soort samenvatting van een `Customer` die alleen de belangrijkste velden bevat. In de sprint wordt de Controller en View aangepast om hiermee om te

[47] http://en.wikipedia.org/wiki/Autocomplete
[48] http://jquery.com/

gaan. Ditzelfde geldt voor de `Supplier`- en `OrderController`. Dit beperkt te hoeveelheid data die telkens uit de database gelezen moet worden en dat betekent een betere performance.

7.3 FUNCTIONEEL ONTWERP

De use cases voor deze sprint hebben vooral te maken met het order scherm. De `CustomerController` wordt uitgebreid met een zoekfunctie en een functie om orders van de debiteur op te vragen. De crediteuren use case lijkt erg op de debiteuren versie, alleen ontbreekt hier het opvragen van orders. Het use case diagram voor de order schermen is te zien in Figuur 20. In de bijlage zijn alle diagrammen te zien die in deze sprint zijn gemaakt.

Er kan op twee manieren gezocht worden. Er is een aparte zoekpagina, hier kan gekozen worden waarin gezocht moet worden en op welke criteria. Sommige velden zijn vrij, bij anderen moet er een optie uit een drop-down menu worden gekozen. Daarnaast is er op de pagina's van Customer, Supplier en Order ook een zoek veld aanwezig. Een query die hier wordt ingevuld leidt uiteindelijk naar het resultatenscherm van de algemene zoekpagina.

De gebruikers kunnen resultaten of overzichten sorteren door te klikken op de header van de kolom. Een tweede klik zorgt ervoor dat de resultaten de andere kant op worden gesorteerd.

Het opvragen van orders bij een Customer gebeurt via een link die wijst naar een order zoek opdracht met als enige criteria het customer id van de huidige Customer.

Een autocomplete veld vult de ingevulde tekst aan met een suggestie. Dit maakt het invullen van het veld makkelijker. Zodra de gebruiker begint met typen moet er een drop down menu verschijnen met suggesties. De gebruiker dan deze aanklikken of er met de cursor toetsen naartoe navigeren. Bij iedere nieuwe knop die wordt ingedrukt worden de suggesties opnieuw bepaald. Het aantal suggesties dat getoond wordt moet te beperken zijn. Het is niet overzichtelijk als er meer dan tien of vijftien suggesties verschijnen.

7.4 TECHNISCH ONTWERP

De belangrijkste nieuwe toevoeging in deze sprint is de Ordercontroller. Deze moet onder andere dezelfde functionaliteit van de Customer en de Supplier krijgen, maar de View ervan zal compleet anders zijn. Voor alle Controllers moet er zoekfunctionaliteit worden toegevoegd en de resultaten moeten gesorteerd kunnen worden.

In de vorige sprint is al een gedeelte van de Supplier geïmplementeerd. Voor de implementatie van de rest kan afgekeken worden bij de `CustomerController`. Er is geen nieuw technisch ontwerp nodig voor de `SupplierController`, deze kan 'afkijken' bij het ontwerp van de `CustomerController`.

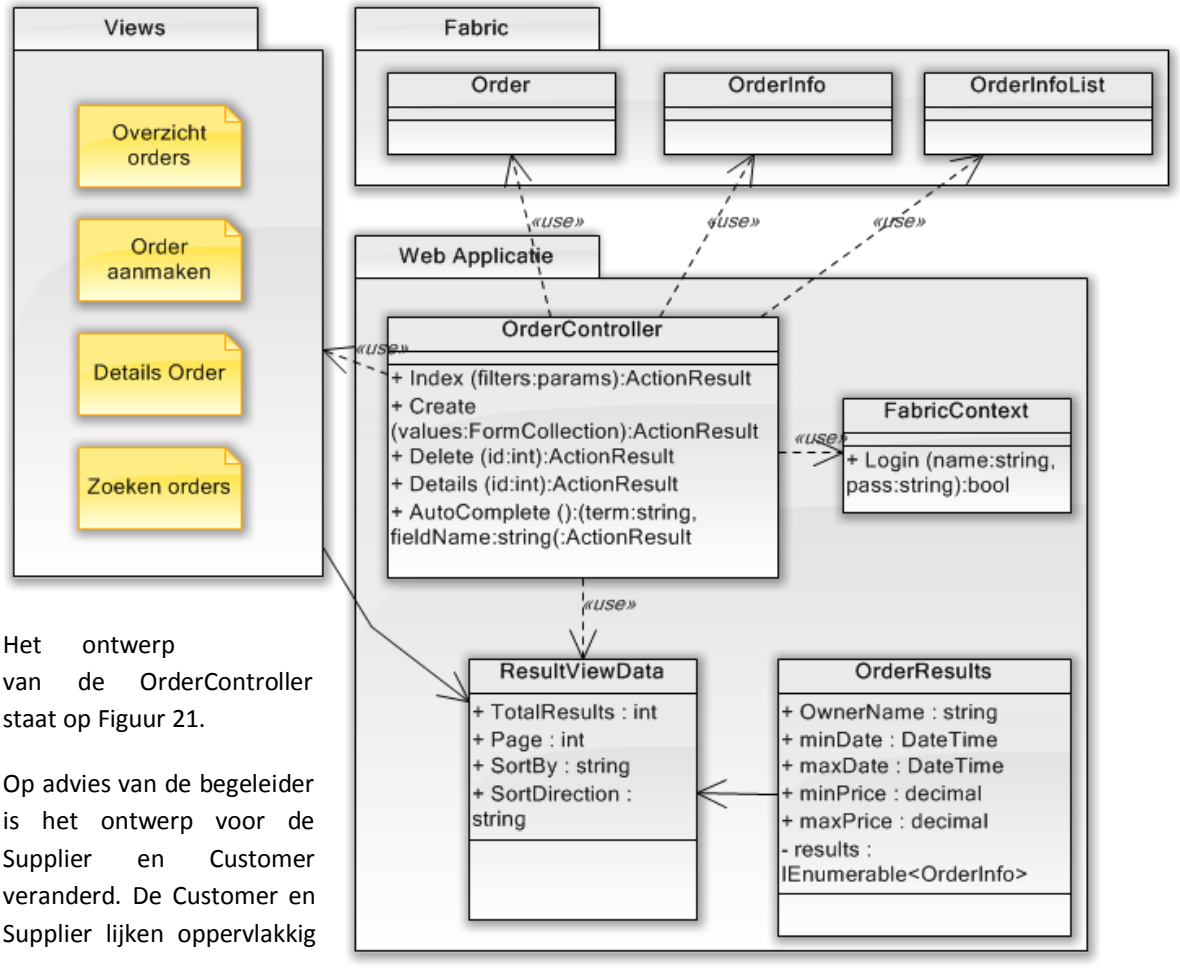
Het sorteren behoeft geen uitgebreid technisch ontwerp, een gebruiker selecteert een veld om op te sorteren en dit wordt in de code met behulp van LINQ uitgevoerd. Zoeken verloopt op een zelfde manier, de gebruiker voert waarden in bij het zoek veld, deze waarde wordt in de code toegevoegd in een LINQ Where clausule.

7.4.1 ORDER CONTROLLER

De OrderController is nieuw en anders dan de twee voorgaande Controllers. Hiervoor wordt een nieuw ontwerp gemaakt. Er is voor deze view een specifiek model nodig, In de vorige sprint werd de complete lijst

Customer objecten gewoon doorgespeeld aan de View, nu word het model anders ingericht, er is namelijk meer informatie nodig in de View dan alleen de data van de orders zelf. De lijst met orders moet verspreid worden over meerdere pagina's indien deze groter is dan een bepaald aantal. Dit is nodig om te voorkomen dat er duizenden orders op één pagina gezet worden. Het model wat nu aan de View wordt aangeboden bevat nu alleen een deelverzameling van alle resultaten. Zodra de gebruiker op 'volgende' o.i.d. klikt, moeten de volgende set resultaten worden weergegeven. Op een of andere manier moet de server weten welke resultaten de gebruiker nu bekijkt, en wat er de volgende keer gestuurd moet worden. Bij dit probleem kunnen twee wegen ingeslagen worden; er moet *metadata* over de resultaat set meegegeven worden aan het model, deze data wordt bij volgende verzoeken teruggegeven. Een andere oplossing is de server zelf metadata laten onthouden in de vorm van sessie variabelen.

Het zelf bewaren van sessie variabelen brengt wat problemen met zich mee. Dit schaalt slecht want voor elke gebruiker moeten deze gegevens afzonderlijk bewaard worden, daarnaast moet de data bewaard worden zolang de gebruiker op de pagina blijft. Het detecteren wanneer een gebruiker de website verlaat kan niet waterdicht geïmplementeerd worden. Het HTTP protocol[49] schrijft namelijk niets voor over het melden wanneer een gebruiker de pagina verlaat. Sterker nog, HTTP is expliciet *stateless.* Het bijhouden van een sessie is een manier om toch een state te introduceren. Een sessie kan om performance redenen niet onbeperkt lang bijgehouden worden door een server. Dit betekent dat er een 'houdbaarheidsdatum' bij de sessie opgeslagen moet worden en deze automatisch opgeruimd moeten worden wanneer de houdbaarheid verstrijkt. Als de data lang bewaard moet worden kost dat performance, maar als het snel wordt opgeruimd krijgen gebruikers de melding dat hun sessie is verlopen wat een slechte gebruikerservaring is. Het sturen van metadata heeft dus de preferentie, dit heeft nog een aantal andere voordelen die later naar voren komen.



Het ontwerp van de OrderController staat op Figuur 21.

Op advies van de begeleider is het ontwerp voor de Supplier en Customer veranderd. De Customer en Supplier lijken oppervlakkig

FIGUUR 21 ONTWERP ORDERCONTROLLER

[49] http://www.w3.org/Protocols/

gezien erg op elkaar, maar later kunnen er verschillen optreden waardoor de implementatie dusdanig van elkaar gaat afwijken dat ze geen base class meer kunnen delen. Er wordt daarom geen gebruik meer gemaakt van een `OrganizationController` als base class. De functionaliteit van de `OrganizationController` is daarom bij beide klassen ondergebracht.

7.4.2 AUTOCOMPLETE

Voor de autocomplete functie moet de View weten welke waarden er allemaal ingevuld *kunnen* worden in een veld. In dit geval moeten de namen van alle Customers worden meegestuurd naar de Zoek View, dit is geen kunst als het maar een klein aantal namen betreft, maar wanneer er meer dan duizend Customers bestaan wordt het moeilijker. Telkens moeten dan alle namen worden opgehaald uit de database wat veel tijd kost, daarna moet de lijst doorgestuurd worden naar de View. Dit is geen goede oplossing, het zou beter zijn als de browser dynamisch een lijst zou ophalen zodra de gebruiker begint met typen. Achtergrondcommunicatie tussen browser en server kan gedaan worden met *AJAX*[50]. AJAX maakt het mogelijk om berichten naar de server te sturen en de antwoorden op te halen zonder dat er een nieuwe pagina geladen hoeft te worden. De controller krijgt de door de gebruiker ingevoerde waarde en geeft aan de hand hiervan een lijst suggesties terug. Dit beperkt de communicatie.

De suggesties worden geladen uit een NVL (zie 5.4.1). Als er bijvoorbeeld gezocht wordt op het veld "`orderType`" wordt de bestaande `OrderTypeNVL` geraadpleegd, de waarden die een overeenkomst hebben met de door de gebruiker ingevoerde waarde worden geselecteerd. Deze selectie moet teruggestuurd worden in een formaat wat daarna door een stuk Javascript in de browser begrepen kan worden.

7.5 IMPLEMENTATIE

7.5.1 ORDER CONTROLLER

Het zoeken op klantnaam wordt als volgt gedaan:

ASP .NET koppelt waarden uit webform aan argumenten voor de Controller methoden als deze in naam overeenkomen. Er moet een form gemaakt waar de gebruiker een naam kan invullen. De Controller krijgt deze naam binnen als parameter. Vervolgens wordt de `CustomerNVL` opgehaald en de juiste Customer eruit gehaald. Het id van deze customer wordt doorgespeeld aan de `Fetch` functie van een `OrderInfoList`. Een `OrderInfoList` is een lijst Orders die voldoet aan bepaalde criteria. Vooralsnog kan er alleen een CustomerId of een boekjaar als criterium opgegeven worden. Dit levert een set orders op in de vorm van een `OrderInfo`. Dit is net als een `CustomerInfo` een samenvatting van het object. De `OrderInfo's` worden in de `OrderResults` geplaatst en aan een `ResultViewData` gekoppeld welke daarna door de View wordt afgehandeld.

De `OrderResults` zegt iets over de resultaat set die hij bevat, namelijk welke filtering er is toegepast. De `ResultViewData` zegt iets over de weergave ervan. De filters en weergave opties die de gebruiker heeft aangegeven worden dus weer gewoon teruggestuurd. Zodra de gebruiker naar de volgende pagina wil gaan worden die waarden allemaal opnieuw naar de server gestuurd, alleen de *page* wordt opgehoogd. De server ziet welke pagina er opgevraagd wordt en knipt de juiste resultaten uit de totale set, dit heet toepasselijk *paging* en wordt op deze manier geïmplementeerd door de Controller:

[50] http://www.dmoz.org/Computers/Programming/Languages/JavaScript/Ajax/

```
private IEnumerable<T> paginate<T>(IEnumerable<T> entries, ResultViewData viewdata)
{
    viewdata.TotalPages = (int)(1 + Math.Floor((double)((entries.Count() - 1) /
viewdata.PageSize)));
    return entries.Skip((viewdata.Page - 1) * viewdata.PageSize).Take(viewdata.Page);
}
```

CODE 7 PAGING VAN DATA

De LINQ methoden `Skip` en `Take` worden gebruikt om éérst een reeks over te slaan, en daarna een reeks eruit te pakken. `Skip` slaat een gegeven aantal items over, en `Take` neemt en gegeven aantal items over uit de set.

Het filteren en sorteren van data gebeurt met een combinatie van de LINQ methoden `Where` en `OrderBy`. Hiermee kan een filter worden toegepast en de resultaat set gesorteerd worden. Zie Code 8.

```
string CustomerId = "1001";
decimal minAmount = 100m;
int page = 2;
int pageSize = 10;
var ordersForSpecificCustomer = Fabric.Sales.Lists.OrderInfoList.Fetch(CustomerId);
var filteredOrders = ordersForSpecificCustomer.Where(order => order.AmountExclVat >
minAmount);
var resultset = filteredOrders.OrderBy(order => order.AmountExclVat);
var pagedResultset = resultset.Skip(page * pageSize).Take(pageSize);
```

CODE 8 FILTEREN EN PAGING VAN ORDERS

7.5.2 CUSTOMER EN SUPPLIER

Op de Customer en Supplier moet dezelfde paging functionaliteit worden toegepast. Om dit te faciliteren wordt er een ResultsBase<T> base class toegevoegd. De OrderResults leiden hiervan af. Deze base class kan dan ook voor de Customer en Supplier worden geïmplementeerd.

7.5.3 SORTEREN

Het sorteren van data kan met behulp van de LINQ functie `OrderBy`, hierbij wordt een veldnaam opgegeven. Met de functie ThenBy kan een tweede veld worden opgegeven waarop wordt gesorteerd wanneer het eerste veld gelijk is. Een groot voordeel van LINQ queries is dat ze *statically typed*[51] zijn, dit betekent dat de compiler de juistheid van de query al tijdens compile time kan controleren. Bij het sorteren op een veld is het makkelijker om gewoon een string op te geven die de naam van het sorteerveld voorstelt. Dit is standaard niet mogelijk met LINQ. Er moet daarom een switch statement komen die alle mogelijke sorteeropties uit kan voeren. Er bestaat een uitbreiding op LINQ genaamd Dynamic LINQ, hiermee is het wel mogelijk een string in te geven, een voordeel hiervan is dat de code ook niet uitgebreid hoeft te worden wanneer er extra sorteer opties worden toegevoegd. Een nadeel is dat het programma minder stabiel is omdat er ook ongeldige waarden opgegeven kunnen worden, omdat het hier gaat om een web applicatie komen alle waarden toch binnen in de vorm van een string dus is dit geen probleem. Het gebruik van Dynamic LINQ en standaard LINQ naast elkaar zorgt voor problemen, omdat de prioriteit van het sorteren niet hoog is wordt er daarom gekozen om af te zien van Dynamic LINQ en de switch methode te gebruiken.

[51] http://en.wikipedia.org/wiki/Type_system#Static_typing

7.5.4 ZOEK FUNCTIONALITEIT

De waarden die een gebruiker invoert in het zoekveld worden net als de sorteeropties als parameters aangeleverd in de functie. De zoekwaarde wordt in een Where clausule gezet die de lijst met crediteuren of debiteuren vernauwt. Een voordeel van zoeken op naam is dat er nu in de NVL gezocht kan worden, de NVL staat in het geheugen van de applicatie. De database hoeft hiervoor niet doorzocht te worden.

De aparte zoekpagina is achterwege gelaten, dit scherm voegt niet veel functionaliteit toe aan de applicatie maar is wel relatief complex. Er kan nu alleen per scherm gezocht worden.

7.5.5 AUTOCOMPLETE

De autocomplete functionaliteit moet algemeen inzetbaar zijn. Het is hiervoor belangrijk dat deze generiek wordt geïmplementeerd zodat de implementatie opnieuw gebruikt kan worden door andere Controllers. Hiervoor wordt een methode aangemaakt die een zoekterm en een categorie als input neemt. De categorie bepaald in welke lijst er gezocht wordt, de input wordt gebruikt als filtering. Een JSON object wordt terug gegeven welke met behulp van een stuk JavaScript op de browser weergegeven wordt. Om de AJAX calls te vergemakkelijken wordt hier opnieuw JQuery voor gebruikt, deze biedt een abstractie laag om de XMLHttpRequest[52] API heen.

De controller class die hoort bij de View krijgt een AutoComplete functie. Deze bekijkt de meegegeven categorie en bepaalt in welke NVL gezocht moet worden. De waarden uit de NVL worden gematched met de meegegeven string (zie Code 9). Indien er méér dan een ingesteld aantal velden overeenkomen wordt de lijst beperkt. Het resultaat wordt teruggestuurd naar de browser.

```
case "CustomerName":
{
    var names = Fabric.CRM.Lists.CustomerNVL.Fetch() // haal de lijst klantnamen op
        .Where(c => c.Value.ToUpper().Contains(term.ToUpper())) // filter op naam
        .Select(c => new AutocompleteResult(c.Value)) // verpak in datastructuur
        .OrderBy(c => c.label).Take(returnsize); // sorteer en beperk de waarden
    return Json(names, JsonRequestBehavior.AllowGet); // converteer resultaat naar JSON
}
```

CODE 9 SELECTEREN VAN SUGGESTIES BIJ HET INVOEREN VAN EEN CUSTOMER NAME

Aan de browser kant is het mogelijk om te subscriben aan het ‘keyup’ event van een HTML input veld. Dit houdt in dat er een gegeven stuk javascript wordt uitgevoerd zodra een gebruiker een letter heeft ingevoerd in een veld. Zodra een ‘keyup’ event afgaat (triggered) wordt er een AJAX request gestuurd met daarin de categorie van het veld, en de op dit moment ingevoerde waarde. Op de achtergrond wacht de browser op antwoord, zodra dit aankomt wordt de ontvangen lijst suggesties aan de gebruiker getoond.

7.5.6 LOCALIZATION

Tijdens de sprint is er een user story bijgekomen. De gebruiker wil kunnen kiezen tussen de taal die op de website wordt gebruikt. Hoewel het geen ‘must have’ prioriteit bezit, is het toch belangrijk deze vroeg te implementeren anders wordt het later extra moeilijk. Dit komt omdat alle output dan als *hardcoded* strings in de source code moet staan. Dit maakt het onmogelijk om de berichten dynamisch aan te passen. Als de output

[52] http://www.w3.org/TR/XMLHttpRequest/

al vanaf het begin dynamisch wordt geladen hoeft later niet de hele source code doorgezocht te worden en alle hardcoded string te vervangen. Het is nog niet nodig om gelijk alle output in meerdere talen te zetten, zolang de mogelijkheid er maar wel is. De website aanpassen zodat deze meerdere talen support heet 'localization'.

Er zijn meerdere manieren om dit te implementeren, uit een onderzoek komen twee voornaamste methoden naar voren: de huidige taal wordt in de URL geplaats (website.com/**nl**/Order/Index) of de gekozen taal wordt in een sessie variabele opgeslagen. Al eerder in het project is onderzocht dat het werken met sessie moeilijk(er) te implementeren is en daarom ook vatbaar is voor bugs. De localization wordt in de URL opgeslagen.

Een ontwerp hiervoor is niet nodig, want de feature is eenvoudig genoeg. Hiervoor werd gebruik gemaakt van de standaard ASP .NET RouteHandler, er moet een nieuwe klasse komen die hiervan afleid en functionaliteit toevoegt voor het kiezen van de localization. Deze wordt dan ingesteld als de routehandler voor pagina's die er anders uit zien in andere talen (het is nog steeds mogelijk om geen andere taal te implementeren voor delen van de applicatie).

Standaard wordt Engels uitgekozen, maar als de gebruiker op een link klikt bovenaan de pagina wordt de website in het Nederlands getoond, pagina's die hierna bezocht worden zijn ook in het Nederlands. Dit moet vroeg geïmplementeerd worden want deze aanpak vereist dat alle strings niet direct in de code staan, maar in een aparte resource file. In plaats van hardcoded strings wordt er nu verwezen naar een *resource file.* Per Controller per View is er een aparte resource file gemaakt. Welke file er gebruikt wordt kan tijdens runtime bepaald worden. Dit maakt het niet alleen makkelijk om de applicatie later te vertalen, alle strings staan nu ook op één plaats en niet verspreid door de code heen.

7.6 TERUGKOPPELING

Er is aan het einde van de sprint een demopresentatie gegeven van het product aan het management, de ontwikkel- en testafdeling van UNIT4 Software B.V. Tijdens een vergadering over onder andere de voortgang met Fabric is kort de tot dan toe gerealiseerde functionaliteit getoond. Daarnaast is er gesproken over de ervaring bij het ontwikkelen met Fabric, deze is voor het grootste deel goed geweest. Vanuit het perspectief van een partner is het werken met het framework geprezen.

8 SPRINT 4 "VERDIEPING"

Waar in de vorige sprint vooral functionaliteit werd toegevoegd, wordt in deze sprint gericht op het verbeteren en uitbreiden van de bestaande functies. De zoekfunctionaliteit wordt flink uitgebreids, er wordt op meer plekken het drop-down menu ingevoerd en het sorteren wordt verbeterd. In deze sprint wordt ook voor het eerst gewerkt met een grote database, hiervoor werd er getest met een database van ongeveer tien debiteuren en 30 orders, de nieuwe testdatabase bevat 200.000+ orders en 160.000+ debiteuren. De niet-functionele eisen over performance komen nu aan bod.

8.1 SPRINT INDELING

In Tabel 9 staan de user stories voor deze sprint. Deze sprint bestaat opnieuw voornamelijk uit **must have** stories. Het item over paging komt opnieuw in de backlog omdat de performance nog niet naar wens is.

TABEL 9 USER STORIES VOOR DE VIERDE SPRINT

User story	Prioriteit
Ik kan orders zoeken die tussen een bepaalde datum vallen.	**Must**
Ik kan order zoeken die binnen een bepaald prijsbereik vallen.	**Must**
Ik kan orders zoeken die in een bepaald boekjaar vallen.	**Must**
Ik kan orders zoeken met een bepaalde status.	**Must**
Ik kan meerdere zoekopties tegelijk invullen.	**Must**
Ik wordt bij het aanmaken van een klant geholpen door auto-complete.	**Must**
Ik kan alleen die acties uitvoeren waar ik recht toe heb.	**Must**
Als de resultaten set groot is, wordt deze verspreid over meerdere pagina's. (verbeteren)	**Must**
Ik kan zien hoeveel resultaten er in totaal zijn gevonden, ongeacht paging.	**Should**
Bij het kiezen van een datum wordt ik geholpen door een datepicker.	**Should**
Resultaten worden dynamisch bijgeladen terwijl ik naar beneden scrol.	**Could**

8.2 ANALYSE

8.2.1 ZOEKEN OP CRITERIA

Fabric staat op dit moment alleen toe om orders te filteren op boekjaar. Dit betekent dat er een gigantische dataset wordt opgehaald uit de database en daarna pas wordt er door de web applicatie filtering toegepast op prijs, datum enz. Dit zorgt voor onnodig veel vertraging. De filter parameters moeten doorgegeven kunnen worden aan Fabric. Deze haalt dan alleen de juiste resultaten op. Dit betekent dat er véél minder data uit de database gelezen hoeft te worden. Het complete activity diagram voor een zoekactie is te zien op Figuur 23. Deze data hoeft dan ook niet door Fabric verwerkt te worden. Fabric moet aangepast worden zodat het deze additionele criteria opties kan verwerken.

8.2.2 PERFORMANCE VAN FETCHING

Voor een performance test is het nodig om een grotere database aan te sluiten. De tot nu toe gebruikte test database telt slechts 35 orders, dit is te weinig om de schaalbaarheid van de applicatie te meten. Er wordt daarom een veel grotere database aangesloten. Een eenvoudige eerste test (tonen lijst orders) met de grote database wijst uit dat er de nodige performance problemen spelen die niet van toepassing zijn op een kleine database. Zo worden nu nog vaak naïef álle resultaten opgehaald waar er maar een klein aantal nodig zijn. Er

zijn een aantal oplossingen geprobeerd om de duur van de query's te verkorten, bijvoorbeeld door parallellisatie toe te passen of door op de achtergrond alvast data te laden voordat de gebruiker erom vraagt.

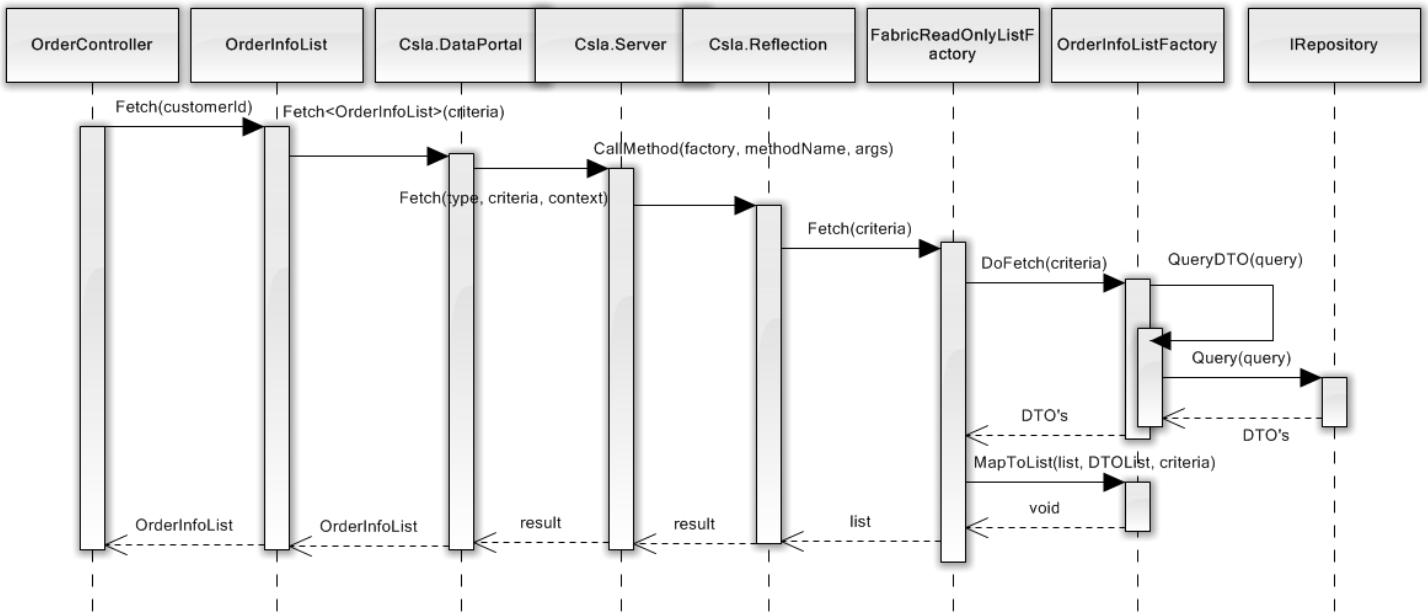
In eerste instantie zijn de query's op verschillende manieren herschreven om te controleren dat het probleem niet ligt bij een onhandige query. Het verschil in tijdsduur tussen de verschillende query's blijkt minimaal. Een mogelijk probleem is dat het proces op een inefficiënte manier is ingedeeld. Er wordt eerst een lijst van `OrderInfo` objecten opgehaald. Een `OrderInfo` object is een soort samenvatting van het volledige `Order` object. Uit deze lijst wordt alleen het id van de objecten bewaard. Voor al deze id's wordt de 'echte' Order opgehaald. Er worden dus heel veel query's in serie uitgevoerd. Door de query's parallel uit te voeren zou er veel winst behaald kunnen worden. Helaas hanteert deze database een globale lock op het uitvoeren van query's. Er kan dus maar één query tegelijk worden uitgevoerd. Een andere insteek is om de query zelf te wijzigen. Op dit moment haalt elke query één record op, het is met SQL mogelijk om meerdere records tegelijk op te halen. Ook deze oplossing blijkt onmogelijk, Fabrc biedt niet de mogelijkheid om zelf query's op te geven. Het ophalen van data verloopt via de Fetch functie van business objecten, de precieze indeling en opbouw van de query's schuilt achter een ontoegankelijke data access layer[53]. Het herinrichten van de query's lijkt geen mogelijke oplossing te zijn. Wél is het mogelijk om de query's te veranderen zodat er andere data opgehaald wordt. Het ophalen van het volledige `Order` object is alleen noodzakelijk wanneer alle data nodig is, een simpele lijst met Orders heeft genoeg aan de basis data die een `OrderInfo` biedt. Door met een kritisch oog te kijken naar welke data er daadwerkelijk gebruikt wordt is het mogelijk geweest om op een aantal plaatsen te 'besparen'.

Deze besparing is niet genoeg om de applicatie vlot te laten reageren, het duurt zelfs nog tientallen seconden om een lijst orders te tonen. Een mogelijke voorziening hierin is om de lijst alvast op de achtergrond te laden voordat de gebruiker erom vraagt. Dit is een naïeve oplossing want de lijst is niet meer geldig zodra er een wijziging wordt aangebracht in één order. Alleen het eerste verzoek om de lijst zou hiermee versneld worden.

Deze opties hebben niet voor voldoende performance winst gezorgd, de volgende stap is het aanpassen van Fabric. Het ophalen van data moet dichter bij de kern van het probleem aangepakt worden. Vooral het ophalen van een lijst Orders moet véél sneller kunnen.

Aan de `Fetch` methoden van een business object kunnen criteria parameters worden meegegeven. Deze parameters worden door de `Fetch` methode verpakt in een criteria klasse. Deze criteria wordt via een aantal Csla modules doorgegeven aan de factory klasse die het gevraagde object kan bouwen (zie Figuur 22). Het opbouwen van een lijst business objects (in dit geval de Orders) gebeurt in twee stappen. Als eerste wordt de criteria klasse omgezet naar een LINQ query in de `DoFetch` functie. Deze query wordt doorgegeven aan de `Repository` interface welke in dit geval een SQL database is. Hier wordt de query omgezet in SQL en uitgevoerd op de database. De overeenkomende rijen worden geladen in *Data transfer objects (DTO)*[54]. De tweede stap is het omzetten van data transfer objects naar de desbetreffende business objects. De totaalprijs van een order object is niet opgeslagen in de database, dit is een *berekende* waarde die afhangt van andere waarden zoals het btw percentage, kortingen e.a. Deze waarde wordt pas bepaald wanneer het DTO wordt omgezet in het business object. Indien er in de criteria beperkingen zijn opgegeven aan de totaalprijs is het dus onvermijdelijk dat er te veel data wordt opgehaald uit de database. De prijscriteria kan pas toegepast worden wanneer de data is opgehaald en niet al in de SQL query. Zodra deze tweede stap is afgerond kan de resultaat set worden geretourneerd.

[53] http://msdn.microsoft.com/en-us/library/ee658127.aspx
[54] http://msdn.microsoft.com/en-us/library/ff649585.aspx



FIGUUR 22 SEQUENTIEDIAGRAM VAN OPHALEN ORDERINFOLIST

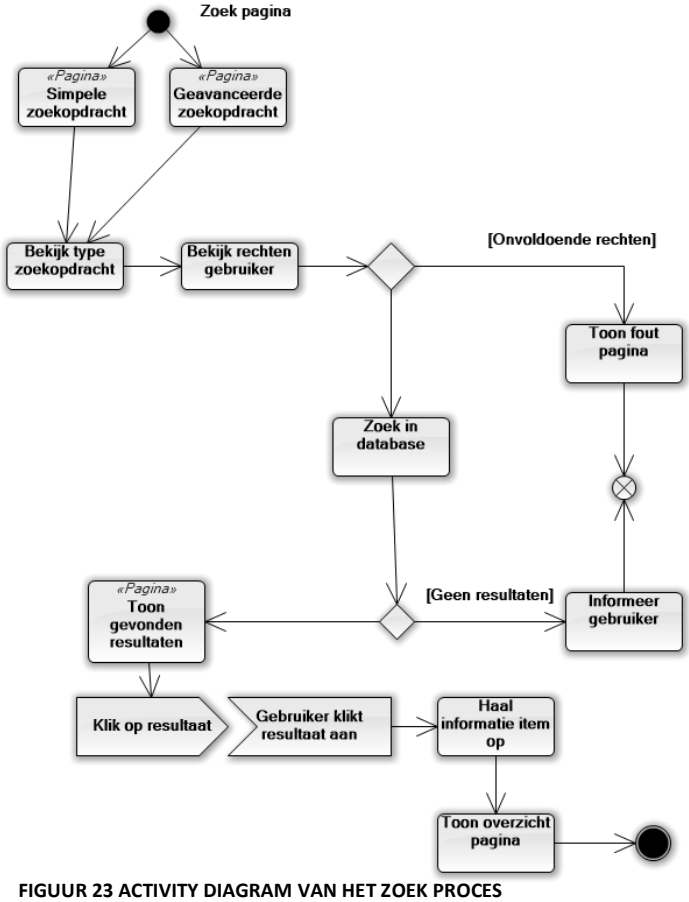
8.2.3 AUTOCOMPLETE

Autocomplete moet nu ook toegepast worden op het klant aanmaak scherm. Hiervoor kan functionaliteit overgenomen worden uit het zoekscherm. Het toevoegen van autocomplete functionaliteit aan velden kost nu relatief veel werk. Het is wenselijk om een oplossing te maken waarbij alléén aangegeven hoeft te worden dat een veld autocomplete ondersteund, de rest zou vanzelf moeten gaan.

8.3 FUNCTIONEEL ONTWERP

8.3.1 ZOEKEN OP CRITERIA

Voor het inbouwen van extra criteria parameters is geen nieuw ontwerp nodig. Elke lijst van business objecten heeft een geassocieerde criteria klasse. Hierin staan de mogelijke criteria waar de lijst op gefilterd kan worden. Deze criteria klasse moet uitgebreid worden, daarnaast moet de `Factory` klasse van de list uitgebreid worden zodat het de nieuwe criteria op de juiste manier kan verwerken.



FIGUUR 23 ACTIVITY DIAGRAM VAN HET ZOEK PROCES

8.3.2 PAGING

Het inbouwen en het toepassen van paging heeft verschillende doelgroepen. Het aanbieden van paging functionaliteit moet op een manier gaan die voor ontwikkelaars makkelijk werkt. Het makkelijkst zou zijn om opties voor paging op te geven voordat je de `fetch` methode aanroept, een nog betere oplossing is om een

`PageOptions` parameter mee te geven aan de `fetch` operatie, dit parameter object[55] bezit de benodige informatie om paging toe te passen.

Na het invoeren van een zoekopdracht wordt de gebruiker naar de eerste pagina gestuurd, er zijn dan onderaan de pagina knoppen beschikbaar om naar de volgende of vorige pagina te navigeren. Het is ook mogelijk om direct naar een bepaalde pagina te gaan door op het nummer te klikken. De gebruiker wil ook graag kunnen zien hoeveel resultaten de zoekopdracht in totaal heeft opgeleverd. De paging functionaliteit wordt op een voor de gebruiker bekende manier aangeboden.

8.3.3 AUTOCOMPLETE

De Autocomplete functionaliteit moet enigszins worden verbeterd. De implementatie werkt naar behoren, maar de inzetbaarheid is niet optimaal omdat er relatief veel gedaan moet worden om Autocomplete toe te voegen aan een veld (zie Code 10) en het is niet flexibel.

```
@Html.TextBox("OrderState") // normaal veld

@Html.TextBoxFor(model => model.State, new { data_autocomplete_url =
      @Url.Action("AutoComplete", "OrderController", new { fieldName = "State" }) })
// autocomplete veld
```

CODE 10 NORMAAL INVOER VELD T.O.V. AUTOCOMPLETE VELD

Er staan in het autocomplete veld drie hardcoded strings, waarvan alleen de laatste variabel is. Het zou makkelijker in gebruik zijn om deze complexiteit te verstoppen achter een eenvoudige functie, die functie wordt dan aangeroepen met alléén het `fieldname`, de rest wordt automatisch ingevuld.

8.4 TECHNISCH ONTWERP

8.4.1 ZOEKEN OP CRITERIA

Het invullen van criteria gebeurt via een webform, hierin staan een aantal tekst velden waarin een gebruiker kan typen. De velden waarop gezocht kan worden staan vast, er kan niet op alle velden worden gezocht. Wanneer de gebruiker waarden heeft ingevuld en het form submit worden alle waarden naar de web server opgestuurd. Het sturen van deze waarden kan op twee manier gebeuren, een GET request of een POST request. Omdat de staat van de applicatie niet mag veranderen door een zoek actie worden de waarden door middel van GET verstuurd. ASP .NET MVC haalt de waarden uit het webform op en zet deze om in parameters die naar de opgegeven functie van de opgegeven controller worden gestuurd.

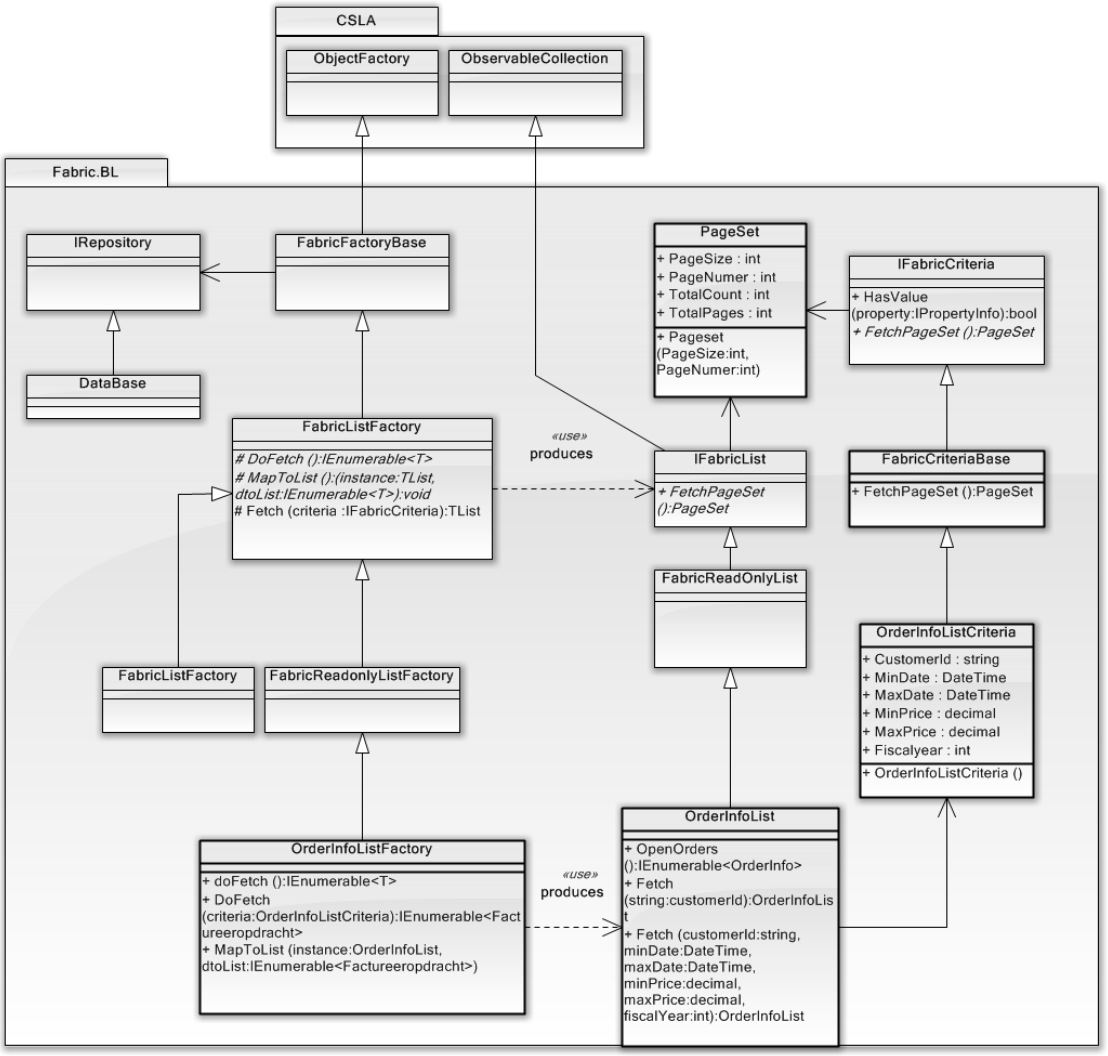
De controller is nu in staat om parameters door te geven aan Fabric, maar deze moet eerst uitgebreid worden om de nieuwe parameters te kunnen verwerken. Als eerst moet de OrderCriteria klasse worden uitgebreid, deze slaat de ingegeven parameters op. Een instantie van deze klasse wordt doorgegeven aan de klasse die uiteindelijk verantwoordelijk is om de zoekopdracht uit te voeren, in dit geval de OrderInfoListFactory. In deze klasse worden de individuele criteria parameters omgezet in één LINQ query, deze query wordt daarna door een ander component omgezet in SQL, welke vervolgens de database aanspreekt en het resultaat terug geeft.

8.4.2 PAGING

[55] http://c2.com/cgi/wiki?ParameterObject

In de vorige sprint zijn de paging klassen al iets aangepast zodat deze generiek zijn. De `OrderResults` klasse moet uitgebreid worden om de nieuwe opties toe te staan (Zie Figuur 25). De belangrijkste toevoeging tijdens deze sprint is een `PageSet` klasse in Fabric. Een `PageSet` past paging toe in Fabric zelf. De `PageSet` wordt door de ontwikkelaar gebruikt om te specificeren hoe paging toegepast moet worden. De `PageSet` wordt daarna meegegeven aan een `Fetch` aanroep. De `PageSet` wordt daarna gekoppeld aan de criteria klasse. De hoogste `IFabricCriteria` interface moet uitgebreid worden zodat deze een `PageSet` kan aannemen. Dit is een stap in de richting van het paging algemeen toepasbaar maken. Alle criteria klassen kunnen dan gebruik maken van paging. De `FabricListFactory` moet ook aangepast worden zodat deze controleert of er paging toegepast moet worden.

Het ontwerp van het sorteren van waarden moet ook herzien worden. Om resultaten te sorteren moeten álle waarden bekend zijn. Door gebruik te maken van een clustered index[56] op een kolom kan het sorteren snel uitgevoerd worden op die ene kolom, maar er kan maar één clustered index zijn per tabel. Het sorteren van andere kolommen zal bij een grote dataset te veel tijd kosten. Het sorteren van data op arbitraire velden wordt uitgezet ten behoeve van performance. Gebruikers kunnen nu alleen de waarden van één page sorteren op alle kolommen door middel van een stuk javascript.



FIGUUR 24 PAGING IN FABRIC

[56] http://msdn.microsoft.com/en-us/library/aa933131(v=sql.80).aspx

8.4.3 AUTOCOMPLETE

In plaats van een TextBox waaraan een Action wordt meegegeven is het ook mogelijk om zelf een andere implementatie van een TextBox te maken waar autocomplete standaard bij aan staat. De Html klasse, waar de TextBox functie toe behoord kan niet uitgebreid worden omdat dit een klasse uit de BCL is, het is wel mogelijk om af te leiden van de klasse en de functionaliteit op die manier uit te breiden. Een beter alternatief is om gebruik te maken van Extension Methods[57], hiermee kunnen er publieke functies aan een klasse worden toegevoegd zonder de source code van die klasse aan te passen. Op die manier kan er een Html.AutoCompleteBox o.i.d. gemaakt worden.

8.5 IMPLEMENTATIE

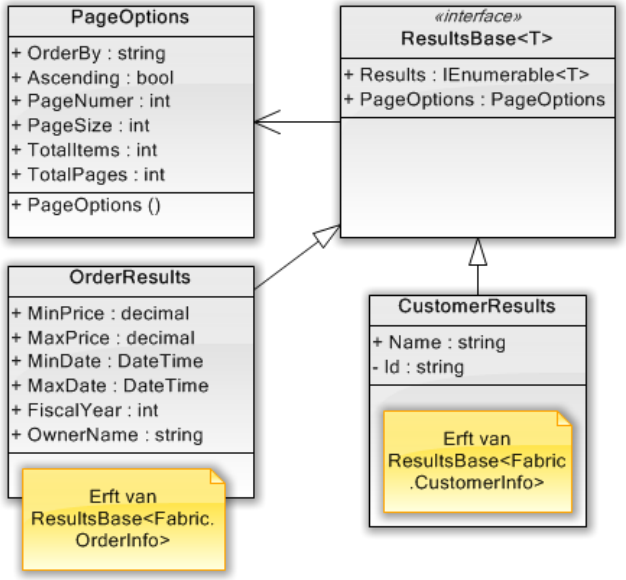
8.5.1 ZOEKEN OP CRITERIA

De criteria klasse die hoort bij het Order object is uitgebreid zodat deze met additionele parameters om kan gaan. De OrderInfoListFactory is ook uitgebreidt zodat de extra parameters op de juiste wijze worden verwerkt in de query die uiteindelijk door de database uitgevoerd zal worden.

8.5.2 PAGING

In de `OrderInfoListFactory` wordt de criteria set dynamisch in een LINQ `Expression` omgezet. Deze expression wordt vervolgens omgezet naar SQL en naar de database gestuurd.

Het prijs component van een Order staat niet rechtstreeks opgeslagen in de database, dit is een *berekende waarde.* In de MapToList functie kan deze waarde pas berekend worden, er moet dus een tweede ronde van filtering toegepast worden, dit keer op prijs. Pas nadat deze filtering is uitgevoerd kan paging worden toegepast. Dit gebeurt door de LINQ methoden `Skip` en `Take`, op dezelfde manier als eerder in de `OrderController` zelf werd gedaan.



FIGUUR 25 PAGING KLASSEN

Om de PageSet in Fabric te bouwen moeten alle ListFactory klassen worden aangepast zodat zij paging kunnen toepassen, in dit geval hoeft van alle klassen alleen de *signature* van een overload te worden aangepast.

Omdat het implementeren van paging complexer is dan verwacht kan deze user story niet volledig worden afgerond in deze sprint. In de volgende sprint moet deze user story opnieuw worden ingedeeld.

8.5.3 AUTO-COMPLETE

[57] http://msdn.microsoft.com/en-us/library/bb383977.aspx

Door een Extension Method toe te passen is het werken met autocomplete velden niet alleen makkelijker in gebruik maar ook beter onderhoudbaar en flexibeler, zie Code 11. Een wijziging in de afhandeling van autocomplete kan nu snel worden doorgevoerd en er is minder herhaling van code in de views.

Gebruikers van een autocomplete veld hoeven nu alleen nog op te geven uit welke lijst de suggesties geladen moeten worden, de rest is geabstraheerd.

```
private static MvcHtmlString AutocompleteFieldFor<TModel, TProperty>(this
HtmlHelper<TModel> html, System.Linq.Expressions.Expression<Func<TModel, TProperty>>
expression, string fieldName, string actionName, string controllerName,
RouteValueDictionary htmlAttributes)
{
    if (htmlAttributes == null)
        htmlAttributes = new RouteValueDictionary();

    string url = UrlHelper.GenerateUrl(null, actionName, controllerName, new
RouteValueDictionary(new {fieldName }), html.RouteCollection,
html.ViewContext.RequestContext, true);
    htmlAttributes.Add(AutocompleteUrlFormat, url);
    return System.Web.Mvc.Html.InputExtensions.TextBoxFor<TModel, TProperty>(html,
expression, htmlAttributes);
}

public static MvcHtmlString FabricAutocompleteFieldFor<TModel, TProperty>(this
HtmlHelper<TModel> html, System.Linq.Expressions.Expression<Func<TModel, TProperty>>
expression, string fieldName, object htmlAttributes)
{
    var attributes = new RouteValueDictionary(htmlAttributes);
    return AutocompleteFieldFor<TModel, TProperty>(html, expression, fieldName,
"AutoComplete", "FabricAPI", attributes);
}
```

CODE 11 NIEUWE FABRIC AUTOCOMPLETE, ALLEEN HET VELDNAAM HOEFT NOG OPGEGEVEN TE WORDEN.

9 SPRINT 5 "VERSNELLING"

In de vorige sprint is er veel tijd besteed aan het implementeren van de `PageSet`. Het is daardoor nu mogelijk om data in pages op te halen. Er moet deze sprint onderzocht worden of de performance van de applicatie door deze oplossing daadwerkelijk genoeg is verbeterd. Hiervoor zal er op een aantal manier de performance van de applicatie gemeten worden. Rapporten hiervan kunnen vergeleken worden met eerdere rapporten, hieruit moet blijken hoe veel invloed de paging oplossing heeft.

Als de huidige performance nog steeds beneden het gewenste niveau is worden de nieuw rapporten gebruikt om meer bottlenecks te ontdekken, een analyse moet daarna uitwijzen of deze problemen opgelost kunnen worden en hoeveel tijd dat zal kosten.

Naast deze activiteit is het ook de bedoeling om de interface van de applicatie te verbeteren. Met de `PageSet` zou het mogelijk moeten zijn om resultaten op de achtergrond te laden terwijl de gebruiker scrolt.

Aan het eind van deze sprint bestaat er al een redelijke applicatie. Om de werking te bewijzen wordt de applicatie gedraaid op een volwaarde webserver (IIS) in plaats van de Visual Studio development server. Dit moet meer zekerheid geven over de correcte werking van de applicatie.

9.1 SPRINT INDELING

In deze sprint zijn een aantal items ingedeeld die niet direct overeenkomen met een user story. Deze gaan over het verbeteren of meten van componenten en worden aangegeven met de prioriteit 'n.v.t.'.

TABEL 10 SPRINT ITEMS VOOR DE VIJFDE SPRINT

User story	Prioriteit
PageSet in Fabric verbeteren	**N.v.t.**
Performance controleren	**N.v.t.**
Bevestigen werking Supplier.	**N.v.t.**
Onderzoek PageSet bij overige klassen, indien nodig: inbouwen.	**N.v.t.**
Localization verbeteren.	**N.v.t.**
Als ik heb gezocht op criteria en er zijn meer dan twee pagina's aan resultaten, wordt de criteria onthouden als ik naar de volgende pagina navigeer.	**Must**
Ik moet inloggen met geldige credentials voordat ik het systeem kan benaderen.	**Must**
Ik kan zelf aangeven hoeveel resultaten ik per pagina wil zien.	**Should**
Als ik een object probeer te wijzigen controleert het systeem of ik daarvoor bevoegd ben.	**Should**
Tijdens de uitvoeringen van een zoekopdracht wordt ik geïnformeerd over de voortgang	**Could**

9.2 ANALYSE

Tot nu toe is de Customer veel meer getest van de Supplier. Dit is mogelijk omdat ze veel op elkaar lijken. Als de customer controller nog naar behoren werkt is er weinig kans dat de Supplier controller dit niet zal doen. Toch is het verstandig om dit te controleren. De controllers verschillende een klein beetje en in de views is er een nog groter verschil.

Het onthouden van de ingegeven criteria blijkt een groter probleem dan gedacht. Een 'volgende pagina' link moet opnieuw alle ingegeven criteria versturen. HTTP is immers stateless en het bijhouden van een sessie wordt zo veel mogelijk vermeden. De criteria worden via *querystring* parameters meegegeven. Op de

resultatenpagina moet de 'volgende pagina' link gelijk zijn aan de link waarmee de huidige pagina is bezocht met als enige verschil het pagina nummer, welke opgehoogd moet worden.

9.2.1 PAGING

Een eerste indruk van de nieuwe `PageSet` oplossing is goed. De applicatie reageert vele malen sneller bij grote databases. In de meeste gevallen is er vrijwel geen vertraging bij het zoeken op criteria. Het springen naar de laatste pagina van resultaten is net zo snel als de eerste, er wordt niet meer onnodig veel data opgehaald. Maar zelfs als de applicatie snel reageert, is het nog steeds mogelijk dat de code intern inefficiënt werkt. Het is in dat geval belangrijk om de aard en oorzaak van deze inefficiëntie te achterhalen, en zo mogelijk te verbeteren.

Er is een merkbare vertraging bij het navigeren tussen pagina´s van de resultaat set. De eerste pagina laadt snel, maar iedere volgende pagina duurt langer dan de vorige. Het direct navigeren naar pagina 20 (met een pagina grootte van 50) zorgt al voor een vertraging van gemiddeld 6.3 seconden. Uit onderzoek blijkt dat web gebruikers zijn in het algemeen niet bereid om langer te wachten dan 2 seconden (Nah). Bij het invoeren van zulke query's is een wat grotere vertraging te verwachten, maar vijf seconden is aangehouden als een limiet. Een resultaten set kan tienduizenden orders bevatten, deze vertraging treedt al op na 1.000 orders. Er is er verbetering nodig.

Als deze performance problemen zijn onderzocht en aangepakt moet gecontroleerd worden of paging ook in andere klassen ingebouwd kan worden. De insteek is algemene toepasbaarheid, maar dit moet nog getest worden.

9.2.2 AUTHENTICATIE

Tot nu toe is er 'achter de schermen' ingelogd door de controller zelf nadat een gebruiker een request heeft ingediend. Dit is nodig voordat er verder met Fabric gecommuniceerd kan worden. De login functionaliteit heeft een minder hoge prioriteit dan de items uit de eerdere sprints. Nu verandert dat en zal de gebruiker zelf moeten inloggen. Met deze verandering is al rekening gehouden waardoor het nu niet nodig is om de bestaande klassen drastisch aan te passen.

Het inloggen verloopt via een aparte pagina en hiervoor moet een nieuwe controller worden aangemaakt, deze controller houdt zich bezig met het communiceren van het authenticatieproces aan de gebruiker. De daadwerkelijke authenticatie wordt afgehandeld door onderliggende klassen. Deze splitsing moet ervoor zorgen dat de verantwoordelijkheden van de klassen goed zijn verdeeld. Er zijn twee vormen van web authenticatie die wijd verspreid gebruikt worden, dit zijn Basic Authentication[58] en Forms Authentication. De eerstgenoemde is een gestandaardiseerd challenge-response protocol wat verweven is in HTTP, Forms Authentication is een niet-standaard aanpak die bovenop HTTP draait. Met Basic Authentication moet de aanvrager zich authentiseren door in de HTTP header te reageren op een challenge. Voor iedere pagina die opgevraagd wordt moet deze challenge-response opnieuw worden doorlopen. In Forms Authentication logt een gebruiker één keer in waarna de server een cookie toestuurt die in het vervolg gebruikt kan worden om te authentiseren. Op het web wordt verreweg het meest gebruik gemaakt van Forms Authentication, hiermee is het mogelijk om een eigen login pagina te bouwen, Basic Authentication staat dit niet toe, hier vraagt de browser zelf om authenticatie zodat deze daarna bij de HTTP header kunnen worden ingevuld. Dit is bijvoorbeeld nodig bij web API's omdat er daarbij geen sprake is van een gebruiker die inlogt.

[58] http://www.ietf.org/rfc/rfc2617.txt

9.2.3 INDICATIE VOORGANG

Als een zoekopdracht enige tijd (meer dan 2 seconden) kan duren is het gebruiksvriendelijk om de voortgang van de opdracht terug te koppelen in de vorm van een voortgangsbalk (progressie indicator). De terugkoppeling kan op twee manieren worden geïmplementeerd:

- Een echte voortgang indicatie
- Een 'nep' indicatie

Het meten van de voortgang van een algoritme kan zeer complex zijn, in sommige gevallen zelfs onmogelijk (halting problem van Turing[59]). Fabric biedt geen functionaliteit die de voortgang van een operatie kan terugkoppelen. Het is evenmin onmogelijk om zelf de voorgang van de operatie te meten. Het is dus onmogelijk om een nauwkeurige meting te verkrijgen van de voortgang van een operatie. Er moet een schatting gemaakt worden door de ingegeven parameters van een opdracht te analyseren. Wanneer er een enigszins accurate schatting kan worden afgeleid van de parameters blijft alleen het probleem van het ontwerpen van de oplossing. HTTP is een stateless protocol dus de server kan de browser niet blijven updaten met de huidige voortgang. Zelfs met AJAX is dit niet mogelijk, AJAX kan alleen gebruik worden om de server te *pollen*, elke operatie moet een uniek nummer krijgen en elk voortgangsverzoek moet dit nummer meegeven. Dit lijkt een complex systeem te worden waarvan het maar de vraag is of de opgegeven voortgang nauwkeurig genoeg is om nuttig te zijn.

Een tweede oplossing is het inbouwen van een niet functionele voortgangsindicatie. Dit is een techniek die veel wordt toegepast. Meestal wordt er dan een procentuele of visuele indicatie gegeven die aanvankelijk snel verloopt maar steeds langzamer naar het einde gaat. Wanneer de operatie klaar is verdwijnt de voorgangsindicatie en wordt het resultaat getoond. De nep voortgang kan worden geïmplementeerd met een eenvoudig stukje JavaScript. De voordelen van een nauwkeurige voortgangsindicatie wegen niet op tegen de hoge kosten om het te bouwen. Omdat een nep indicatie niet veel daadwerkelijke functionaliteit toevoegt wordt deze user story uitgesteld.

9.3 FUNCTIONEEL ONTWERP

Voor de veranderingen aan de `PageSet` is geen nieuw functioneel ontwerp nodig, het gaat hier alleen om veranderingen 'onder de motorkap'. Gebruikers merken dit alleen in hun gebruikerservaring, niet in de aangeboden functionaliteit. De zoek interface en mogelijkheden blijven hetzelfde.

De functionele diagrammen voor het navigeren tussen pagina's van de resultaten set zijn te vinden in bijlage **VI.**

9.3.1 AUTHENTICATIE

De applicatie moet een eigen login pagina hebben. Gebruikers zijn vaak niet goed bekend met de authentication prompt die verschijnt bij gebruik van Basic Auth, het is niet onmogelijk om toch een eigen inlogpagina te bouwen, maar wel een stuk moeilijker. Er wordt daarom gebruik gemaakt van Forms Authentication. Het ontwerp voor het inloggen is grotendeels weergegeven op Figuur 11. Een gebruiker vraagt een pagina op; als deze is ingelogd wordt Het verzoek verder behandeld. Als de gebruiker niet is ingelogd wordt deze eerst naar de inlogpagina gestuurd. Hier kan de gebruiker zich authentiseren. Het systeem controleert de

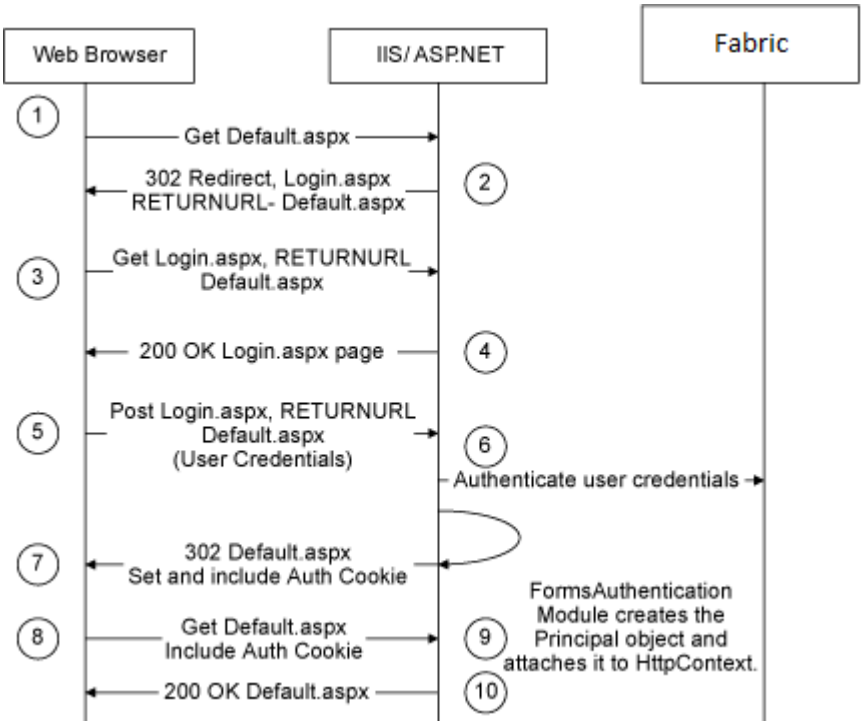
[59] http://mathworld.wolfram.com/HaltingProblem.html

ingevoerde gegeven, als deze juist zijn wordt het eerste verzoek alsnog afgehandeld, zo niet wordt de inlog pagina opnieuw getoond. Een gebruiker authentiseert zich met een gebruikersnaam en wachtwoord. Deze moeten overeenkomen met de gegeven die in Fabric zijn opgeslagen. Het toevoegen van nieuwe gebruikers is via Fabric nog niet mogelijk, dit wordt daarom niet aangeboden in de web applicatie maar er wordt al wel met de mogelijkheid rekening gehouden door de code op een uitbreidbare manier in te richten.

9.4 TECHNISCH ONTWERP

9.4.1 AUTHENTICATIE

De nieuwe `AccountController` serveert de login pagina en ontvangt de login gegevens van een gebruiker. De `AccountController` zal gebruik maken van een interface die de gegevens kan controleren, er moet een `FabricAuthenticator` komen die Fabric user kan authentiseren. Als de gegevens juist zijn moet de gebruiker worden ingelogd in de database. Figuur 26 geeft het technisch ontwerp weer van het opvragen van een beschermde pagina.



FIGUUR 26 AUTHENTICATIE VAN EEN GEBRUIKER

Wanneer er Basic Auth gebruikt wordt onthoudt de browser de opgegeven naam en wachtwoord, Forms Auth staat volledig log van HTTP, er moet op een andere manier worden onthouden dat een gebruiker is ingelogd. De `AccountController` is verantwoordelijk voor het behouden van een sessie zodat de gebruiker ingelogd kan blijven. De gebruiker blijft ingelogd door een authentication ticket, dit is een cookie waarmee de webserver kan zien welke gebruiker een verzoek doet. De controle op de authentication ticket vindt als allereerste plaats zodra de webserver een verzoek ontvangt, dit is nodig zodat de gebruiker ingelogd wordt voordat deze om data vraagt die beschermd is.

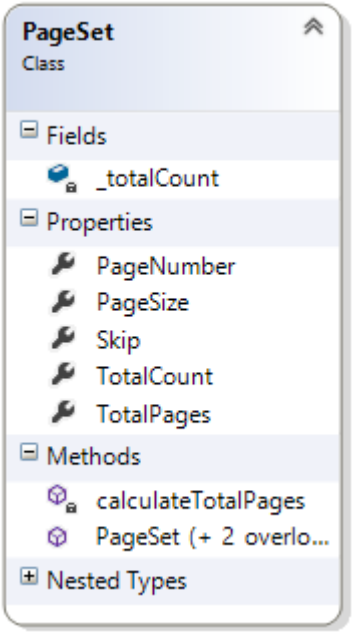
Om informatie uit Fabric op te vragen moet een gebruiker niet alleen zijn ingelogd maar in de meeste gevallen moet er ook een administratie zijn geselecteerd. Deze informatie moet ook met elk verzoek worden meegestuurd, dit moet in een aparte cookie staan. Omdat het meegeven van de administratie niet eigen is aan de authenticatie wordt het implementeren van deze cookie uitgesteld. In de tussentijd wordt de te gebruiken administratie uit een configuratiebestand gelezen.

Alle controllers worden aangepast zodat zij controleren of de gebruiker is ingelogd voordat ze een pagina serveren. Als de gebruiker niet is ingelogd wordt deze naar de login pagina verwezen.

9.5 IMPLEMENTATIE

9.5.1 PAGING

Als eerst wordt de vertraging bij het navigeren van pages onderzocht. Visual Studio heeft een ingebouwde performance analyzer. Deze kan aan het proces gekoppeld worden en meet daarna hoeveel tijd het programma besteedt in de verschillende functies die uitgevoerd worden. Het eerste performance rapport (zie Figuur 28) wijst al op een opmerkelijk probleem. De Count functie (uit de BCL) wordt gebruikt om het aantal gevonden *records* te tellen. Dit wordt daarna aan de gebruiker gecommuniceerd. Het tellen van de rijen kost relatief extreem veel tijd. Dit heeft te maken met *Deferred Execution*[60] (ook wel Lazy Loading genoemd). Dit is een techniek die probeert het onnodig berekenen of laden van gegevens te minimaliseren, LINQ pat dit toe waar mogelijk. In geval van Fabric betekent dit dat de objecten van een verzameling pas uit de database geladen worden zodra het object daadwerkelijk opgevraagd wordt. Dit zorgt voor véél meer aparte *database hits* maar in verreweg de meeste gevallen wordt er in totaal veel minder data verstuurd (omdat niet alle elementen uit de resultaat set gebruikt worden). Pas wanneer de elementen expliciet geteld worden moeten ze allemaal opgehaald worden, dit zorgt voor veel database hits en kost veel tijd. Het resultaat hiervan is dat 90% van de tijd die het ophalen van de lijst kost zit in het tellen van de elementen.



FIGUUR 27 DIAGRAM VAN DE PAGESET KLASSE

Function Name	Inclusive Samples %	Exclusive Samples %
Csla.Reflection.MethodCaller.CallMethod(object,string,bool,object[])	97,04	0,00
Csla.Reflection.MethodCaller.CallMethod(object,class Csla.Reflection.DynamicMetho...	97,04	0,00
UNIT4.MKB.Fabric.Server.FabricReadOnlyListFactory`3.Fetch(!2)	97,04	0,00
System.Linq.Enumerable.Count(class System.Collections.Generic.IEnumerab...	**90,98**	**0,13**
UNIT4.MKB.Fabric.BL.Factory.Sales.Lists.OrderInfoListFactory.MapToList(cla...	**5,38**	**0,00**

FIGUUR 28 DEEL VAN HET INITIËLE PERFORMANCE RAPPORT

Dit kan tegen gegaan worden door te tellen tot een specifiek aantal, en daarna op een of andere manier te laten weten dat er niet meer verder geteld wordt. Er kan bijvoorbeeld een limiet worden ingesteld van 5.000 items. Als de uitgevoerde query meer rijen telt wordt er in de PageSet een flag gezet die aangeeft dat er meer dan x aantal elementen zijn. De gebruiker krijgt dat alleen te zien dat er minstens x aantal elementen zijn, dit draagt minder informatie over maar levert wel een betere performance.

Een nóg betere oplossing zou zijn om een COUNT[61] SQL query uit te voeren. Hierbij wordt de database gevraagd te tellen hoeveel records er voldoen aan de opgegeven query, er wordt dan geen data getransporteerd en het kost slechts één database hit. De structuur van het toepassen van paging moest hiervoor enigszins gewijzigd worden maar het resultaat is zeer merkbaar.

Een tweede ogenschijnlijk simpele optimalisatie is het berekenen van de totaalprijzen niet uitvoeren wanneer er geen criteria aan zijn gesteld. Het kost relatief veel tijd om deze waarden te bepalen, het is onnodig om deze voor alle orders te berekenen, dit is alleen nodig wanneer er orders weggelaten moeten worden als zij niet

[60] http://blogs.msdn.com/b/charlie/archive/2007/12/09/deferred-execution.aspx
[61] http://msdn.microsoft.com/en-us/library/ms175997.aspx

voldoen aan de gestelde totaalprijscriteria. Als de prijzen daar niet worden berekend gebeurd dit pas later wanneer de prijs opgevraagd word om te tonen op het scherm (opnieuw lazy loading).

Het omzetten van een DTO in een business object gebeurt sequentieel. Dit kan parallel uitgevoerd worden omdat het omzetten verder geen *side effects*[62] heeft. Een gevolg hiervan is wel dat de data enigszins in een andere volgorde in het uiteindelijke resultaat *kan* zitten. Dat is in dit geval geen probleem. Er moet wel opgelet worden dat er géén parallellisatie wordt toegepast wanneer er een prijscriteria is ingevoerd, dit kan door *race condities*[63] zorgen voor te veel resultaten. Dit is af te vangen door *locking*[64] in te bouwen, maar dit heft het voordeel van parallellisatie op.

Er bestaat nog een hardnekkig probleem bij het opvragen van 'hoge' pagina's. Gewoonlijk worden in dat geval de eerste x aantal DTO's overgeslagen en dan een klein aantal verwerkt. Deze oplossing is niet mogelijk bij een zoekopdracht met prijscriteria. Er moeten dan namelijk x aantal resultaten overgeslagen worden *waarvan de prijs binnen de criteria valt.* Dit laatste deel betekent dat de prijs eerst berekend moet worden om te bepalen of de order wél in de resultaten set behoord maar overgeslagen moet worden omdat het op een eerdere pagina staat. Een DTO valt in één van drie categorieën:

- Afleggen: Voldoet niet aan de criteria.
- Overslaan: Voldoet aan de criteria, maar zou niet op de gevraagde pagina staan.
- Bewaren: Voldoet aan de criteria en zou op de gevraagde pagina staan.

We zijn alleen geïnteresseerd in de laatste categorie, maar het is onvermijdelijk om eerst net zo lang DTO's te verwerken tot het aantal orders dat overgeslagen moet worden (door paging) is bereikt.

9.5.2 AUTHENTICATIE

Alle controllers moeten voordat ze een operatie uitvoeren controleren of de aanvrager zich heeft geauthentiseerd. Dit kan gedaan worden door in de functie zelf de gebruiker te controleren, dit betekent wel dat alle functies deze controle moeten uitvoeren en dat is niet handig. Het is ook mogelijk om hiervoor een Attribute[65] te gebruiken. Een Attribute voegt metadata toe aan het programma, een voordeel van het gebruik van een attribuut is dat het op klasse-niveau ingezet kan worden, het is daardoor niet meer nodig om alle functies de controle zelf uit te laten voeren. In dit geval moet de `ActionFilterAttribute` interface geïmplementeerd worden, hiermee kan code worden uitgevoerd vóórdat de aangeroepen functie uitgevoerd wordt. Er kan op dat moment gecontroleerd worden of de gebruiker ingelogd is, als dit niet het geval is zal er een redirect plaatsvinden naar de login pagina.

De `AccountController` is zelf niet verantwoordelijk voor het authentiseren van een gebruiker. Hiervoor maakt de controller gebruik van een losstaande service. Deze service interface moet gebruikers kunnen valideren, om voorbereid te zijn op de toekomst worden er ook functies opgenomen om gebruikers toe te voegen en wachtwoorden te wijzigen. De Fabric implementatie van deze service kan alleen gebruikers authentiseren, dit wordt bereikt door de `SessionContext` klasse uit Fabric te gebruiken.

Wanneer de gebruiker ingelogd heeft wordt er een sessie-id opgeslagen op de web server, deze identificeert de gebruiker. Een verwijzing naar deze sessie wordt opgenomen in de Authentication Ticket die de gebruiker als cookie meekrijgt. De gebruiker kan zelf aangeven of dit ticket moet verlopen nadat de website verlaten is, of deze voor onbepaalde tijd bewaard moet blijven.

[62] http://en.wikipedia.org/wiki/Side_effect_(computer_science)
[63] http://support.microsoft.com/kb/317723
[64] http://msdn.microsoft.com/en-us/library/c5kehkcz.aspx
[65] http://msdn.microsoft.com/en-us/library/z0w1kczw(v=vs.80).aspx

10 SPRINT 6 "CONCURRENCY"

Er is gekozen om deze sprint een week langer te laten duren. Dit is nodig omdat er tijdens deze sprint een concept versie van het eindrapport ingeleverd moet worden. Er is een week tijd uitgetrokken om dit te realiseren. Wanneer dit afgerond is zal de sprint zoals normaal verlopen.

In deze sprint wordt de authenticatie functionaliteit uitgebreid en verbeterd, dit is nodig om meerdere verschillende gebruikers te ondersteunen. Het volgende punt van deze sprint is namelijk het ondersteunen van meerdere gebruikers die in verschillende administraties werken.

10.1 SPRINT INDELING

Deze sprint focust zich weer op het toevoegen van nieuwe functionaliteit. Er is nog één item opgeschoven uit de vorige sprint, namelijk de authenticatie, hier moeten een paar dingen aan gewijzigd worden voordat het overweg kan met meerdere gebruikers die tegelijk zijn ingelogd. Daarnaast zal het maken van het concept verslag ook relatief veel tijd kosten. Deze sprint telt niet veel verschillende items, dat is zo ingedeeld omdat er onzekerheid is over de duur van het maken van het concept rapport. Bovendien lijkt het implementeren van *Multi-Tenancy*[66] complex. Deze user story moet niet onderschat worden. De complexiteit die geïntroduceerd wordt door meerdere gebruikers op meerdere verschillende (of dezelfde) administraties te laten werken kan moeilijk zijn om te implementeren. Indien alles wel voorspoedig gaat kan het werk al beginnen aan het onthouden van gebruikers instellingen.

TABEL 11 USER STORIES VOOR DE ZESDE SPRINT

User story	Prioriteit
Concept afstudeer dossier opleveren.	**N.v.t.**
Ik kan normaal doorwerken zonder storing als er een tweede gebruiker inlogt.	**Must**
Ik kan tegelijk met een andere gebruiker in dezelfde administratie werken.	**Must**
Ik kan zelf vrij wisselen tussen welke administratie ik gebruik.	**Must**
De administratie die ik heb uitgekozen blijft onthouden als ik de pagina verlaat.	**Should**
De taal die ik heb uitgekozen blijft onthouden als ik de pagina verlaat.	**Should**
Ik kan een business object alleen wijzigen als ik de daarvoor vereiste rol bezit.	**Could**

10.2 ANALYSE

10.2.1 ADMINISTRATIE WIJZIGEN

Een administratie is een groep instellingen en data die op zichzelf staat, voor klanten is het gebruikelijk om meerdere administraties te beheren. Voorheen is er altijd met één administratie gewerkt, nu moet het mogelijk worden om meerdere administraties te gebruiken. De gebruiker kiest zelf in welke administratie hij wil werken en er moet op ieder moment gewisseld kunnen worden. Het zou gebruiksvriendelijk zijn als de gekozen administratie onthouden zou blijven tussen sessies.

Het aanmaken van een nieuwe administratie d.m.v. Fabric is nog niet mogelijk.

10.2.2 MULTI USER

[66] http://msdn.microsoft.com/en-us/library/aa479086.aspx

Meerdere gebruikers moeten de applicatie tegelijk kunnen gebruiken, zelfs als zij in dezelfde administratie werken. Zowel het opvragen als wijzigen van business objects moet tegelijk kunnen. Problemen met gebruikers die elkaar storen kunnen alleen optreden wanneer de applicatie is ingericht op een verkeerde manier. Dit kan namelijk alleen optreden wanneer er binnen de applicatie gebruik wordt gemaakt van een globale staat waarvan elke gebruiker afhankelijk is. De applicatie mag niet op een andere manier functioneren wanneer er meerdere gebruikers tegelijk in werken. Als eerste moet daarom alle afhankelijkheden van een globale staat worden geëlimineerd. Testing moet daarna uitwijzen of de applicatie hetzelfde reageert onafhankelijk van het aantal gebruikers. Bij elk verzoek moet er opnieuw verbinding worden gemaakt met de database, dit kost performance, maar het openhouden en beheren van tientallen zo niet honderden openstaande verbindingen is evenwel kostbaar.

Het is mogelijk om te tonen hoeveel gebruikers en zelfs welke er nog meer ingelogd zijn en/of in de huidige administratie werken. Hier komt opnieuw een probleem naar voren wat al eerder is opgedoken, namelijk dat er niet met zekerheid gezegd kan worden wanneer een gebruiker een pagina verlaat. Het is daardoor moeilijk om nauwkeurig te bepalen welke gebruikers er nog bezig zijn. Omdat deze functionaliteit geen hoge prioriteut heeft wordt dit niet geïmplementeerd.

10.2.3 Beveiliging cookies

Op dit moment staat nog alleen de gekozen administratie en de gekozen taal in een cookie. Later kunnen er meer waarden aan toegevoegd worden en dit kunnen ook gevoelige gegevens zijn. De waarden moeten daarom niet in plaintext worden neergezet. Als een aanvaller de cookie bemachtigd mag hij niet weten wat erin staat. De data in de cookie moet daarom versleuteld worden. Dit betekend dat ook de ‘echte’ eigenaar van de cookie niet kan lezen wat erin staat.

10.3 Functioneel ontwerp

Omdat het voor de gebruiker onzichtbaar is dat er ook andere gebruikers zijn ingelogd hoeft hier geen functioneel ontwerp voor gemaakt te worden. Voor de gebruiker verandert er niets. Ook het versleutelen van de cookies behoeft geen functioneel ontwerp.

10.3.1 Administratie wijzigen

De gebruiker moet op iedere pagina kunnen zien in welke administratie hij werkt, deze informatie kan daarom het beste op dezelfde plaats worden getoond als de naam van de huidige gebruiker. Wanneer de gebruiker op de naam van de administratie klikt, moet er een nieuwe pagina worden geopend waarin de administratie gewijzigd kan worden. Omdat de applicatie weet tot welke administraties de ingelogde gebruiker toegang heeft kan hier ook autocomplete worden toegepast.

10.4 Technisch ontwerp

10.4.1 Administratie wijzigen

Voordat een gebruiker kan wisselen tussen administraties moet het systeem eerst aangepast worden zodat het om kan gaan met een door een gebruiker opgegeven administratie. Voorheen werd de gebruikte administratie uitgelezen uit een configuratiebestand. Deze moet nu meegegeven worden aan elk verzoek aan de server. Er zijn twee manieren om dit te bereiken:

- De indeling van de standaard route (zie 5.4.2) wordt aangepast zodat de naam van de administratie hierin wordt meegenomen. (`website.com/{administratie}/{controller}/{actie}/{id}`).
- De naam van de administratie wordt in een cookie geplaatst.

Volgens REST voorschriften zou de naam van de administratie in de URL thuishoren. Het is voor de toegankelijkheid van de applicatie het beste om niet afhankelijk te zijn van een verborgen interne staat, een cookie is zo een verborgen staat. Welke oplossing het best is hangt van het perspectief af. Door gebruik te maken van een cookie is het onmogelijk om een overzicht pagina van een business object op te vragen zonder eerst een administratie geselecteerd te hebben (of handmatig een cookie aan te maken). Een gemiddelde gebruiker zal niet handmatig URL's invullen maar navigeren door op links te klikken. Het is mogelijk om tijdens de opbouw van de pagina de links zo in te richten dat de administratie hier in staat. Voordat dit gedaan kan worden moet er wel een administratie zijn opgegeven, het probleem van een sessie komt hier opnieuw naar voren. Moet de webserver de gekozen administratie zelf onthouden? Zoals eerder aangegeven heeft een oplossing zonder sessie variabelen de preferentie, de gekozen administratie wordt daarom opgeslagen in een cookie.

10.4.2 Multi user

De login functionaliteit is nu nog niet goed geïntegreerd in Csla. Csla biedt al functionaliteit om gebruikers voor ieder verzoek te authentiseren. Het gebruik maken van Csla functionaliteit zou het een stuk makkelijker moeten maken om meerdere gebruikers tegelijk te ondersteunen. Hiervoor moet elk inkomend verzoek geauthentiseerd worden door Csla, deze vult dan het 'User' object waarmee de rest van de applicatie kan controleren welke gebruiker het verzoek heeft gedaan, welke rechten die gebruiker heeft etc. Zo nodig kan de 'User' klasse worden afgeleid om meer functionaliteit toe te voegen. Zodra het binnen Fabric mogelijk is om met een sessie id in te loggen kan ook de geselecteerde administratie aan de User worden gekoppeld.

10.4.3 Beveiliging cookies

De cookie moet versleuteld worden met een symmetrische sleutel. Deze sleutel moet buiten de applicatie om bewaard worden. Als de applicatie zelf de sleutel zou genereren en het programma loopt vast is de sleutel verloren gegaan en kunnen de cookies die in omloop zijn niet meer gelezen worden. De encryptiesleutel wordt daarom uit het configuratie bestand gelezen. Voordat de cookie wordt overgedragen aan de browser wordt de inhoud versleuteld. Wanneer de webserver de cookie weer ontvangt wordt deze ontsleuteld zodat de waarden uitgelezen kunnen worden. Voor de versleuteling wordt AES[67] gebruikt, deze encryptie standaard is geschikt voor algemene doeleinden.

10.5 Implementatie

10.5.1 Administratie wijzigen

ASP .NET MVC biedt een groot aantal *hooks* waarmee de afhandeling van een verzoek aangepast kan worden. een van deze hooks is de *Application_AuthenticeRequest,* hiermee kan de authenticatie van een verzoek worden aangepast. In dit geval moeten we de gebruiker inloggen in Fabric aan de hand van de authentication ticket (zie 9.4.1), daarna moet de juiste administratie worden geopend.

[67] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Wanneer een gebruiker zijn Account pagina bezoekt staat daar aangegeven welke administratie er is geopend. Hier kan een andere administratie worden ingevuld, de applicatie overschrijft dan de waarde in de cookie en opent de nieuwe administratie direct.

Zodra dit is opgezet moet de oude implementatie worden weggehaald, hiermee lazen de controllers de administratienaam uit de configuratiefile.

10.5.2 MULTI USER

Door gebruik te maken van de Csla ApplicationContext kan het afhandelen van meerdere gebruikers relatief gemakkelijk worden opgelost. Evenals het uitlezen van de administratie wordt ook de authenticatie met Csla gecontroleerd in de *Application_AuthenticeRequest*. Op dit moment zou Csla de gebruiker al geauthentiseerd moeten hebben indien deze een authentication ticket heeft meegestuurd.

Omdat elk verzoek naar de webserver in een apart thread wordt afgehandeld en er geen afhankelijkheid van globale staat is moet de applicatie correct om kunnen gaan met meerdere gebruikers. Het simuleren en geautomatiseerd testen van tientallen tot duizenden gebruikers die de applicatie tegelijk gebruiken kost meer tijd dan er beschikbaar is. Om die reden is er getest door in meerdere verschillende browsers verschillende users in te loggen en daarmee acties uit te voeren, de applicatie heeft zich tijdens deze tests gedragen zoals verwacht.

10.5.3 BEVEILIGING COOKIES

In een cookie kunnen meerdere waarden staan. Het is daarom niet handig om elke waarde apart te versleutelen, de gehele cookie moet als één alfanumerieke reeks worden beschouwd en op die manier worden versleuteld. Het toepassing van de versleuteling door middel van AES kan gedaan worden door gebruik te maken van de RijndaelManaged klasse, deze kan arbitraire byte sequenties versleutelen. Voor het versleutelen met AES is zowel een sleutel nodig als een *initialization vector,* deze vector stelt de initiële staat van het versleutelingsalgoritme voor. Voor optimale veiligheid moet de initialization vector voor iedere versleuteling anders zijn. Dit wordt gedaan door de vector iedere keer op te hogen.

Er moet rekening gehouden worden met een scenario waarin een cookie niet kan worden ontsleuteld. Dit komt voor wanneer de cookie beschadigd is, wellicht door toedoen van de gebruiker zelf of wanneer de sleutel verloren gaat. Wanneer dit optreedt moet de cookie verwijderd worden en een nieuwe aangemaakt. Het is niet mogelijk om een cookie direct te verwijderen, alleen door de verloopdatum op een moment in het verleden te zetten zal de browser de cookie verwijderen. Het is daarnaast ook mogelijk om de gehele cookie te overschrijven met nieuwe data.

11 SPRINT 7 "TOEPASSING"

Het concept bestaat al lange tijd uit de Customer, Supplier en Order schermen. Dit is namelijk de basis functionaliteit, dit is uitgebreid met paging, authenticatie, localization, autocomplete en andere functies die algemeen toepasbaar zijn. Voordat er nog meer schermen toegevoegd kunnen worden moeten er nog een aantal dingen gebeuren. Deze sprint richt zich op het verbeteren van de kern van de applicatie. Zodra deze verbetering is doorgevoerd kunnen er nieuwe schermen worden toegevoegd.

11.1 SPRINT INDELING

Op dit moment is er in het project te veel 'dubbele code'. Er is in de verschillende controllers overlap in functionaliteit. Het toevoegen van nieuwe controllers zou een stuk flexibeler zijn als deze niet opnieuw functionaliteit hoeven te implementeren die andere controllers ook al aanbieden.

De voorkeursinstellingen die een gebruiker opgeeft worden onbeveiligd in een cookie opgeslagen. Op dit moment is die data nog niet gevoelig maar dat zou later wel kunnen. Het is daarom nodig om de opgeslagen gegevens te beveiligen.

Er bestaat ook een wens om gegevens of gebeurtenissen op te kunnen slaan in een log. In deze sprint wordt er veel gewerkt aan functionaliteit die niet direct door de gebruiker te zien is, maar die het verder ontwikkelen van de applicatie vergemakkelijken.

TABEL 12 USER STORIES VOOR DE ZEVENDE SPRINT

User story	Prioriteit
Het toevoegen van nieuwe controllers moet makkelijker worden	**N.v.t.**
Het moet mogelijk zijn om overal in de applicatie gebeurtenissen te kunnen loggen.	**N.v.t.**
De onderliggende communicatie met Fabric moet geabstraheerd worden.	**N.v.t.**
Ik kan een product aanmaken.	**Should**
Ik kan een product opvragen.	**Should**
Ik kan een product verwijderen.	**Should**
Ik wil niet dat gegevens uit cookies leesbaar zijn door derden.	**Should**

11.2 ANALYSE

11.2.1 HERSTRUCTURERING CONTROLLERS

De order, customer en supplier controller bieden allemaal functionaliteit voor het aanmaken, ophalen, wijzigen en verwijderen van business objecten. Elke controller heeft een `Details` functie met vrijwel dezelfde implementatie. Het zou mooi zijn om dit op een generieke manier op te lossen. Deze functionaliteit moet buiten de klasse worden gehaald en in een herbruikbaar component worden gestopt. Het is niet mogelijk om zomaar alle controllers te combineren in één generic klasse o.i.d. Het verwijderen van een Customer wijkt af bijvoorbeeld af van de rest, het wijzigen en creëren van een order gebeurt ook op een andere manier dan bij customer of suppliers, etc. Er moet een oplossing komen die voor zover mogelijk functionaliteit hergebruikt.

11.2.2 PRODUCT CONTROLLER

De Product controller moet informatie over artikelen kunnen weergeven, aanmaken en verwijderen. Tevens moeten er artikelen gezocht kunnen worden op hun beschrijving. In deze sprint wordt de product controller gemaakt zodra de herstructurering van de controllers is voltooid. Het zou daarna namelijk een stuk makkelijker moeten zijn om nieuwe controllers toe te voegen.

11.2.3 REPOSITORY

Op dit moment is de web applicatie nauw verbonden met Fabric, dit is logisch maar het zou mooi zijn om sommige onderdelen beter herbruikbaar te maken. Dit is mogelijk door de communicatie met Fabric abstracter te maken. Hiervoor kan het Repository pattern gebruikt worden, de applicatie communiceert dan met een abstracte data-access laag waarvoor een Fabric implementatie wordt gemaakt. Het is op die manier mogelijk om de applicatie ook met een andere dataprovider te laten communiceren. Hier is dan alleen een implementatie van de repository interface nodig.

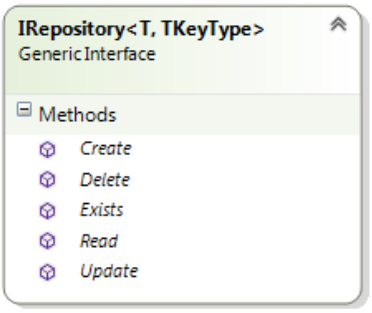
De verschillende controllers communiceren nu direct met Fabric bij het implementeren van CRUD functionaliteit. Fabric is nog in ontwikkeling dus het is verstandig om de communicatie voor zover mogelijk te laten verlopen via een repository interface. De controllers maken gebruik van een implementatie van deze interface om te communiceren met Fabric. Mocht Fabric later veranderen waardoor clients niet meer op de hiervoor gebruikte manier kunnen communiceren hoeft alléén deze implementatie aangepast te worden (of een nieuwe implementatie aangemaakt te worden). De controllers hangen af van deze interface en niet direct van Fabric. Deze ontkoppeling splitst het systeem beter op in modules en zorgt dat de afhankelijkheid ligt bij interfaces en niet implementaties.

11.2.4 LOGGING

Tijdens het testen van de applicatie met de debugger is het makkelijk om te zien waarom er in bepaalde situaties fouten optreden. Wanneer de applicatie los van de debugger draait is het onmogelijk om de interne staat van het programma op te vragen, en is er dus geen informatie beschikbaar over de toedracht van de fout. In een productie omgeving worden namelijk doorgaans niet de complete exception en stacktrace getoond. Door de applicatie gebeurtenissen te laten loggen naar een bestand is het altijd mogelijk om na een fout te onderzoeken wat er is gebeurt. Om dit te implementeren moeten er twee dingen gebeuren: de logger moet gebouwd worden, en alle klassen moeten aangepast worden zodat zij belangrijke gebeurtenissen en fouten via de logger vastleggen.

11.3 FUNCTIONEEL ONTWERP

Deze sprint is opnieuw gericht op interne uitbreidingen en verbeteringen, alleen voor het Product scherm moet er een extern functioneel ontwerp gemaakt worden. Voor de repository en de logging module wordt alleen een beschrijving gemaakt.



FIGUUR 29 KLASSE DIAGRAM VAN DE IREPOSITORY

11.3.1 REPOSITORY

De repository moet zo abstract mogelijk zijn en kent een minimaal aantal database operaties (alle CRUD operaties, zie figuur Figuur 29), er kan

alleen op primary key worden uitgelezen. Dit is precies wat de meeste controllers in de meeste gevallen doen. Het type van deze key is generic wat betekent dat de implementatie dit zelf kan bepalen. De lees operaties geven ook een generic type terug. Wanneer er nauwkeuriger gezocht moet worden moet Fabric direct worden benaderd.

11.3.2 LOGGING

Met de logs krijgt de beheerder van de applicatie meer inzicht in de gebeurtenissen die zich hebben voorgedaan. Hiermee moet het onder andere makkelijker worden om te ontdekken waarom er fouten optreden in de applicatie. (zie figuur Figuur 30).

```
PS C:\Users\Rmeerbur> get-content "C:\Users\Rmeerbur\Documents\rommel\Fabric web app.log"
Info     18-09-12 16:25:33: Application started!
Info     18-09-12 16:27:19: User "SYSTEEM" signed in.
Info     18-09-12 16:27:33: User "SYSTEEM" changed Administration to MVL99999
Info     18-09-12 16:28:09: User SYSTEEM is fetching a Customer with id 1001.
Info     18-09-12 16:30:37: User SYSTEEM is fetching a Order with id 20124011.
Info     18-09-12 16:30:54: User "Raymond" signed in.
Info     18-09-12 16:31:22: User Raymond is fetching a Customer with id .
Error    18-09-12 16:31:22: Exception occurred: Customer.Details: id is null or emptry, Parameter name: id
Info     18-09-12 16:31:31: Application shut down.
```

FIGUUR 30 LOG ENTRIES TIJDENS EEN KORTE RUN VAN DE APPLICATIE

Er moet naar verschillende bestanden tegelijk gelogd kunnen worden. In de toekomst is het wellicht wenselijk om de status van de caching in een ander log bij te houden dan de status van de rest van het programma. Er moet dus per logbericht aangegeven worden naar welk log het bericht hoort. Verder moet elk logbericht voorzien zijn van een type (zoals informatie, waarschuwing, fout etc.) er kan dan in de applicatie worden aangegeven wat voor type berichten er daadwerkelijk in de log moeten komen. Als er wordt ingesteld dat alleen errors in de log moeten komen zullen alle andere typen berichten genegeerd moeten worden.

11.4 TECHNISCH ONTWERP

In deze sprint is er veel tijd besteed aan het (her)ontwerpen van componenten. Er komt een belangrijke nieuwe base class bij en de communicatie met Fabric wordt geabstraheerd.

11.4.1 REPOSITORY

Controllers halen op dit moment objecten op door ze op te vragen aan Fabric. Het komt de indeling van de applicatie ten goede als hier een interface wordt tussen geplaatst. Als de interface met Fabric verandert, of er wordt een ander framework dan Fabric gebruikt (wellicht de voorgangen van Fabric) moet de code in alle controllers worden aangepast. Dit is veel werk. De controllers moeten onafhankelijk zijn van de manier waarop zij de data (business objects) ontvangen. Dit kan bereikt worden door hier een interface tussen te plaatsen. Deze interface kan gebruikt worden om business objecten op te halen. Een implementatie die specifiek is aan Fabric wordt in dit project gebruikt, maar het is mogelijk een andere implementatie te maken als dat nodig is.

De communicatie met Fabric zal gebruik maken van het *Repository* design pattern[68]. Hiermee wordt de data access logica gescheiden van de van de business logica. De repository kan aangesproken worden om CRUD functionaliteit uit te voeren voor business objects (zie Figuur 31).

Er moet voor ieder business object een aparte repository implementatie gemaakt worden. Hoe moet de implementatie anders weten op wélk type business object hij zijn operaties moet uitvoeren. Een oplossing

[68] http://msdn.microsoft.com/en-us/library/ff649690.aspx

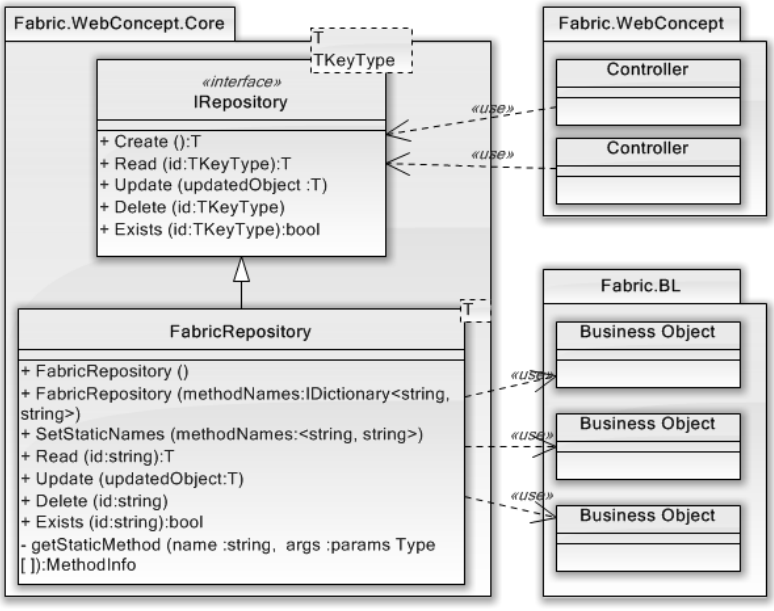
hiervoor is de naam van het business object mee te geven in de functie aanroep. De functie kan daardoor (door middel van een `switch`[69] constructie o.i.d.) het gewenste business object aanspreken. Een belangrijk nadeel hiervan is dat de repository implementatie aangepast moet worden voor elk nieuw business object waar de repository mee om moet gaan. Als er later een nieuw business object wordt toegevoegd aan Fabric moet de repository implementatie eerst uitgebreid worden voordat deze overweg kan met het nieuwe object.

Het zou ideaal zijn om een generic klasse te maken die operaties uitvoert op het meegegeven type parameter. Bij deze constructie komt er opnieuw een probleem om de hoek wat eerder heeft gezorgd voor moeilijkheden bij het generiek implementeren van de Customer en Supplier controllers. Het is niet mogelijk een `static` functie aan te roepen op een type, alleen op een instantie van een type (zie Code 12). Dit probleem kan opgelost worden door gebruik te maken van reflectie. Deze optie is eerder afgewezen (zie 6.4.2), voornamelijk omdat het te complex is. Deze conclusie is getrokken uit eerdere ervaring met reflectie in C++, hier zijn de mogelijkheden zeer beperkt en is het ook erg moeilijk om een robuuste oplossing te bouwen. Voor dit probleem is er opnieuw gekeken naar reflectie, dit keer specifiek naar C#. Reflectie wordt hierin volledig ondersteund in tegenstelling tot C++ waar alleen de meest basis informatie beschikbaar is. Het is hierdoor een stuk aantrekkelijker geworden om gebruik te maken van reflectie. De performance zal iets lager zijn, maar de extra flexibiliteit die hiermee beschikbaar is zorgt voor een veel betere uitbreidbaarheid van het programma.

```
public interface GenericRepository<T> { T GetFabricObject(string id); }
public interface FabricRepository<T> : GenericRepository <T> where T : FabricObject<T>
{
    public T GetFabricObject(string id)
    {
        return T.Fetch(id); // compilatie fout
    }
}
```

CODE 12 GEWENSTE OPLOSSING GENERIC REPOSITORY

De repository interface heeft twee type arguments. Het type waarop deze repository opereert (T) en een key type (TKeyType). In Fabric worden voor alle business objects strings gebruik als id. Door de repository onafhankelijk te maken van het key type is het mogelijk later een andere implementatie te gebruiken die integers of andere datastructuren als key gebruikt. Strikt gezien is het noodzakelijk om een generic constraint toe te voegen die verplicht dat het key type argument een `ToKey` functie implementeert. In de praktijk is dit niet nodig omdat alle objecten impliciet afleiden van de `Object`



FIGUUR 31 KLASSE DIAGRAM FABRICREPOSITORY

[69] http://en.wikipedia.org/wiki/Switch_statement

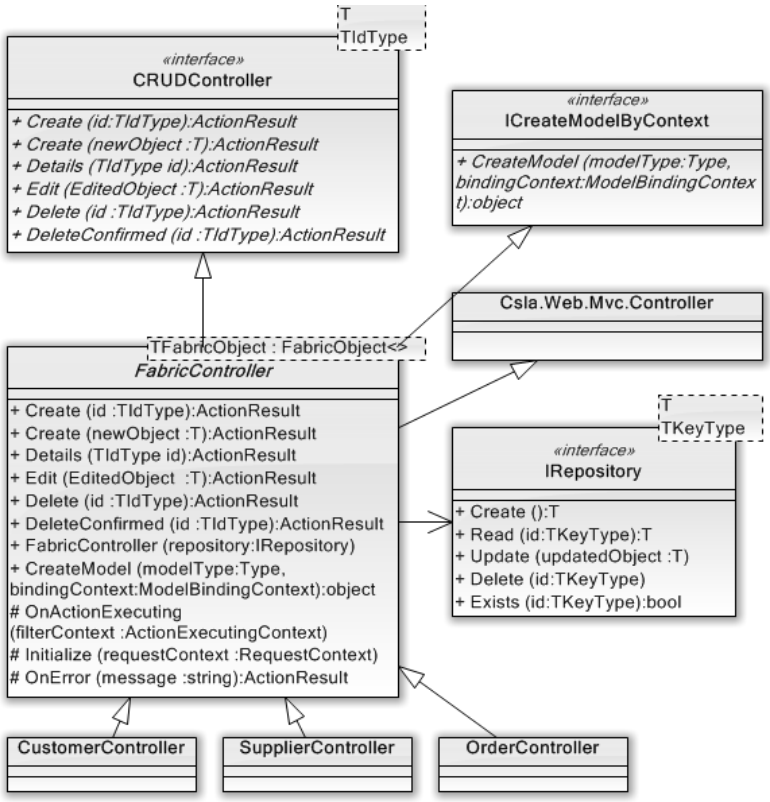
klasse, deze definieert een `GetHashCode`[70] functie die gebruikt kan worden om een object om te zetten in een key.

Intern maakt Fabric al gebruik van zijn eigen repository, dit betekent niet dat het opnieuw toevoegen van een repository dubbel werk is. Het is hierdoor mogelijk om de web applicatie gebruik te laten maken van een andere bron van data. Door de toegang tot data te abstraheren is het ook makkelijker om de controllers te testen. Er kan dan een *mock* (nep) repository gebruikt worden voor testgevallen.

Het gebruiken van deze repository zal later ook een groot voordeel zijn bij het loggen.

11.4.2 FABRICCONTROLLER

De in de vorige paragraaf beschreven Repository moet aan elke controller toegevoegd worden. Dit is op zich geen probleem maar het zou een betere oplossing zijn om de controllers deze functionaliteit te laten delen. Eén oplossing daarvoor zou een generieke '`FabricController`' abstract base class zijn. Deze base class implementeert functionaliteit voor bepaalde Views. Specifieke controllers leiden van deze base class af en kunnen de functionaliteit overschrijven indien nodig. Dit elimineert de noodzakelijkheid om veel code te dupliceren.



FIGUUR 32 KLASSE DIAGRAM VAN FABRICCONTROLLER

De `CRUDController` is een interface met methoden voor controllers. De verschillende functies hiervan worden indirect aangeroepen door de gebruiker. De `FabricController` is een abstracte base class die al deze functies implementeert. De verschillende 'echte' controllers leiden hiervan af en kunnen zo nodig de gegeven functionaliteit overschrijven. Het doel van deze veranderingen is om het zo makkelijk mogelijk te maken om nieuwe controllers toe te voegen.

11.4.3 LOGGING

De logging module zal bestaan uit een tweetal interfaces en twee implementaties. Eén logschrijver die berichten kan schrijven naar een bestand en een `LogManager` die de verschillende logschrijvers beheert en die dient als 'aanspreekpunt' voor de rest van de applicatie. De logberichten worden naar de `LogManager` gestuurd die ze daarna naar de juiste schrijver stuurt, de logmanager kijkt eerst of het type bericht wel . De verschillende schrijvers schrijven het bericht weg naar een filestream.

[70] http://msdn.microsoft.com/en-us/library/system.object.gethashcode.aspx

Alle klassen in de applicatie moeten gebruik kunnen maken van de `LogManager`. Het gaat tegen het doel van de `LogManager` in als alle klassen zelf een instantie van de `LogManager` maken, het moet juist zo zijn dat alle klassen dezelfde `LogManager` gebruiken. Dit kan bereikt worden door gebruik te maken van het Singleton pattern[71], hiermee wordt constructie van een klasse worden beperkt tot één instantie. Via een static methode kan deze instantie opgehaald worden waarna er functies op aangeroepen kunnen worden.

11.5 Implementatie

11.5.1 Repository

De `FabricRepository` is een implementatie van de `IRepository` waarbij het return type en key type zijn ingevuld met respectievelijk `FabricObject` en string. De Fabric implementatie werkt door via reflectie de juiste methode op te zoeken. Deze methode wordt uitgevoerd en het resultaat wordt teruggegeven.

Omdat de Fabric methode wordt gezocht op naam kunnen er problemen optreden wanneer de naam van een van deze operaties in het framework verandert. Dit is iets wat waarschijnlijk niet zal gebeuren, maar het is toch verstandig om ertegen bestendig te zijn. Om het systeem hiermee om te kunnen laten gaan worden de namen van de functies uit een configuratiebestand gelezen. Er wordt dus niet automatisch rekening mee gehouden maar hierdoor hoeft het project niet opnieuw gecompileerd te worden.

11.5.2 FabricController

De `FabricController` probeert zo veel mogelijk functionaliteit uit de verschillende controllers algemeen toepasbaar te maken. Er zijn faciliteiten voor fout afhandeling die door alle subklassen gebruikt kunnen worden. Op die manier worden fouten altijd op dezelfde manier af gehandeld en hoeven subklassen niet zelf te kiezen hoe zijn omgaan met een onherstelbare foutsituatie.

In theorie hoeft een business object niet per se de `FabricController` te subklassen, als het scherm geen functionaliteit biedt die specifiek is aan dat business object kan de `FabricController` zelf gebruikt worden. Dit levert problemen op met de `ControllerFactory` die gebruikt wordt, deze instantieert namelijk een controller die hoort bij de ingegeven naam. Wanneer een gebruiker een URL bezoekt zit de naam van de controller hierin verwerkt, dit kan niet werken als er direct gebruik gemaakt wordt van de base class. Deze heeft generic type arguments nodig en dit wordt niet ondersteund door de gebruikte `ControllerFactory`, daarnaast zou het ook een vreemde URL opleveren. Om dit te laten werken moet er een andere `ControllerFactory` gebouwd worden, het probleem heeft geen hoge prioriteit en bovendien is er een eenvoudige omweg: een klasse ervoor definiëren.

11.5.3 Logging

Er moet rekening gehouden worden met verschillende typen loggers. Doorgaans worden de logberichten naar een bestand geschreven, maar het kan ook wenselijk zijn om de berichten op het beeld te tonen. De `LogManager` werkt daarom met een ILogWriter interface. Via deze interface kunnen er berichten worden doorgeven. De implementatie van de interface kan zelf bepalen of de berichten naar het scherm moeten worden gestuurd of in een bestand terecht komen o.i.d.

De `LogManager` is in principe slechts een wrapper om de ILogWriter, hij houdt bij welke writers er zijn en geeft het logbericht door aan de goede writer indien de gebruiker (in dit geval de beheerder van de server) het type

[71] Design patterns: Blz. 127

logbericht heeft ingeschakeld. Het instellen welke logberichten er doorgelaten mogen worden gebeurd via de configuratiefile.

Het is niet verplicht om de LogManager te gebruiken, logwriters kunnen ook direct worden aangemaakt en gebruikt. Om de writer via de LogManager te benaderen moet deze eerst via een functie aan de LogManager worden toegevoegd, deze kan daarna op eenzelfde manier worden verwijderd. Op die manier is het mogelijk om verschillende logfiles te laten aanmaken op verschillende momenten door verschillende klassen. Via de LogManager kunnen de logbestanden ook weer geleegd worden. Tevens is

12 SPRINT 8 "CACHING"

Performance overwegingen zijn tijdens de ontwikkeling van de applicatie continu belangrijk geweest. Er is geprobeerd winst te boeken door de code te optimaliseren en de database zo min mogelijk aan te spreken. In deze sprint wordt er onderzocht op welke manier er caching geïmplementeerd kan worden. Dit zou de vertraging tijdens het laden van pagina's flink kunnen verminderen. De gebruiker hoeft dan minder lang te wachten en de applicatie heeft het minder zwaar.

12.1 SPRINT INDELING

Caching is het belangrijkste onderdeel tijdens deze sprint. Wanneer dit afgerond is, is de applicatie klaar om weer nieuwe controllers toe te voegen. Caching kan op veel manieren toegepast worden, in deze sprint wordt er daarom relatief veel tijd besteed aan het onderzoeken van de beste toepassing van paging voor deze applicatie.

User story	Prioriteit
Als een pagina gecached kan worden zal dit gebeuren.	**Should**
Als een pagina wordt opgevraagd wordt eerst bekeken of deze uit de cache gehaald kan worden	**Should**
Een pagina wordt uit de cache verwijderd wanneer de pagina niet meer geldig is.	**Should**
Het oplossen (resolving) van afhankelijkheden tijdens het uitvoeren van de applicatie gebeurt op een flexibele manier.	**Should**
Ik kan van een order de individuele order regels bekijken	**Should**
Ik kan een opdrachtregel van een order aanpassen of verwijderen	**Should**
Ik kan een nieuwe opdrachtregel toevoegen aan een order	**Should**
Ik kan statistieken bekijken van een debiteur	**Could**
Ik kan facturen van een debiteur bekijken	**Could**

FIGUUR 33 USER STORIES VOOR DE ACHTSTE SPRINT

12.2 ANALYSE

12.2.1 CACHING

Caching is een universele maatregel voor het verminderen van de *server load*[72]. Als er geen cache wordt gebruikt wordt er voor elk request een nieuwe HTML pagina opgebouwd (View). Een pagina kan niet zomaar als file worden geserveerd omdat de pagina stukken code bevat die de pagina invulling geven. De pagina moet dus eerst opgebouwd (ook *gerendered* genoemd). Zelfs als er een tweede keer dezelfde pagina wordt opgevraagd kan de applicatie niet aannemen dat dit dezelfde pagina oplevert. De waarde van parameters kan veranderd zijn door een externe partij, of de interne staat van het programma is anders. Er moet dus expliciet aangegeven worden wanneer een pagina bewaard kan worden. Als dit het geval is kan de web applicatie de opgebouwde HTML bewaren en direct serveren bij volgende requests (zie Figuur 34). Dit levert een grote performance winst op omdat de server minder werk hoeft uit te voeren. Pagina's laden hierdoor ook sneller.

Sommige pagina´s zijn statisch, andere zijn dynamisch. Het detailoverzicht van een debiteur is bijvoorbeeld een statische pagina, iedereen die deze pagina opvraagt krijgt hetzelfde te zien. Dit maakt deze pagina een uitstekende kandidaat voor caching. De account pagina daarentegen is dynamisch, iedere gebruiker heeft zijn eigen account en de account pagina ziet er daarom voor iedereen anders uit. Deze pagina kan ook gecached worden, maar dit zal minimale winst opleveren omdat het voor elke gebruiker een andere pagina is.

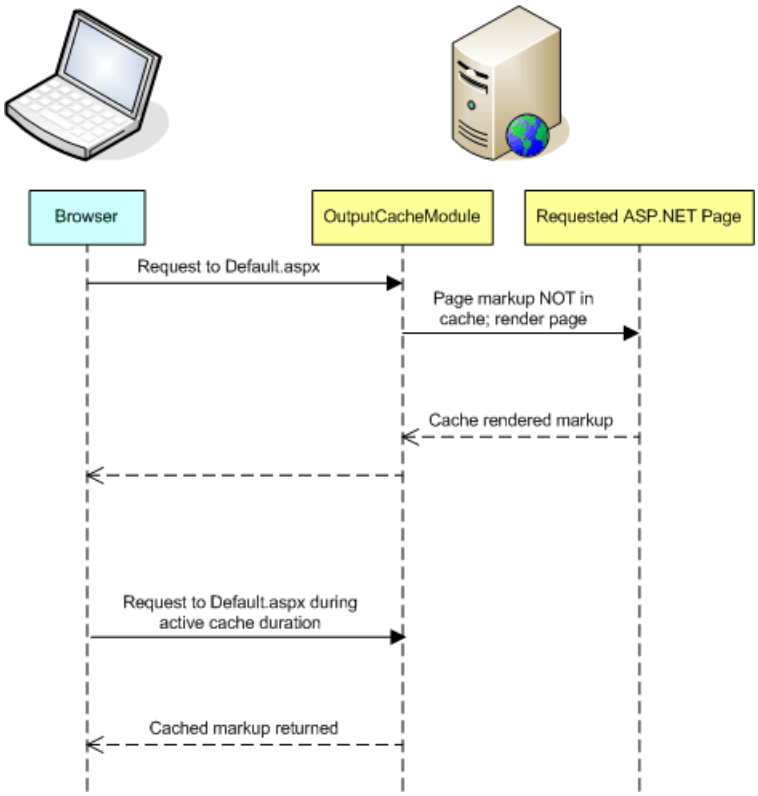
[72] http://en.wikipedia.org/wiki/Load_(computing)

Voordat het ontwerp van de caching oplossing gemaakt kan worden, moet eerst gekeken worden wat de *applicatie* van de cache is; hoe wordt de cache ingezet:

- Welke pagina's kunnen er wel en niet gecached worden.
- Hoe lang moet een pagina in de cache blijven.
- Welke gebeurtenissen kunnen er voor zorgen dat een cache item niet meer geldig is.
- Wat wordt er precies opgeslagen in de cache.
- Waar bevindt de cache zich in de structuur van het programma (logisch)
- Waar bevindt de cache zich fysiek (op de webserver, of een aparte server)

Deze vragen moeten beantwoord worden voordat er een technisch ontwerp gemaakt kan worden.

Er is net al toegelicht welke soorten pagina's er geschikt zijn voor caching, alle pagina's behalve de account overzicht pagina zijn geschikt voor caching. In theorie is zelfs mogelijk om ook deze te cachen, in dat geval moet de webserver eerst controleren wíe de pagina opvraagt voordat er gekeken kan worden of er voor die specifieke gebruiker een cache pagina bestaat. In de praktijk is dit nadelig, er komen dan zeer veel cache items van verschillende account pagina´s. Hoe meer items er in de cache staan, hoe langer het duurt om één item te vinden en hoe meer opslagruimte er nodig is. De cache voor deze applicatie beperkt zich daarom tot alle pagina´s behalve de account overzicht pagina. In de cache komen de volledig opgebouwde pagina's te staan.



FIGUUR 34 SEQUENCE DIAGRAM VAN CACHING

Er zijn twee manieren om de geldigheid van een cache item te specificeren (*cache invalidation*[73]). Eén manier is door een tijdslimiet te koppelen aan het cache item. De tweede manier is om item voor onbepaalde tijd in de cache te plaatsen, en deze expliciet te verwijderen wanneer ze niet meer geldig zijn. Voor deze web applicatie kunnen items er voor onbepaalde tijd ingezet worden. Zodra een gebruiker een business object wijzigt moet het bijbehorende cache item verwijderd worden.

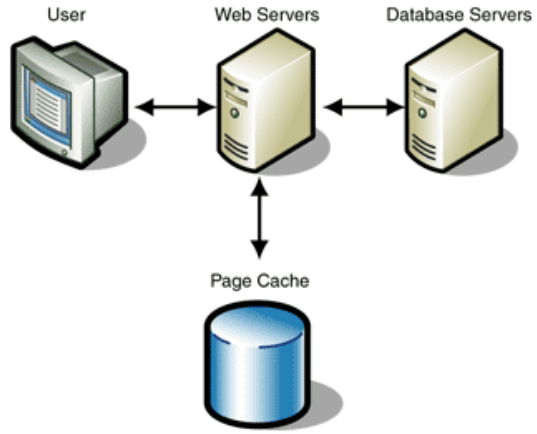
Caching kan op meerdere plaatsen worden toegepast, ook tegelijk. Het is mogelijk om de web browser van de gebruiker pagina´s te laten cachen[74]. Dit zorgt ervoor dat de browser de pagina uit zijn eigen cache haalt wanneer een pagina voor de tweede keer wordt opgevraagd. De webserver wordt hiervoor niet eens benaderd. Dit is qua performance de beste oplossing, maar er zit een belangrijk nadeel aan: als een pagina veranderd is de browser hier niet van op de hoogte. Er kan enkel een verloop tijd worden gekoppeld aan een pagina uit de browser cache, de verloop tijd wordt gespecificeerd door de web server. Als de gebruiker een business object wijzigt zou de webserver in zijn reply moeten specificeren dat de pagina met het

[73] http://en.wikipedia.org/wiki/Cache_invalidation
[74] http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

detailoverzicht nu niet meer geldig is. HTTP biedt hier geen mogelijkheden voor aan, bovendien zou dit een naïeve oplossing zijn omdat het niet werkt bij meerdere gebruikers. Dit betekent niet dat de browser cache niet gebruikt kan worden. Het is namelijk de perfecte cache voor JavaScript files, CSS files of afbeeldingen. Door een verloop tijd van een uur in te stellen hoeft de webserver al een stuk minder data te versturen. Een gebruiker moet dan wel maximaal een uur wachten als er veranderingen worden doorgevoerd aan files van dit type, maar dat zou alleen zeer sporadisch voorkomen. Alles wat niet in de browser gecached kan worden moet op de webserver gebeuren.

Het bepalen of er een cache item bestaat voor het gegeven request moet gebeuren vóórdat de controller het request gaat afhandelen.

De fysieke plaatsing van de cache is ook belangrijk. Het zou ideaal zijn om de cache te implementeren als dedicated cache server. Het is binnen dit project niet mogelijk die opstelling te bouwen. Wel wordt er rekening gehouden met deze implementatie en moet het mogelijk zijn om het cache systeem op deze manier in te zetten. Een belangrijke overweging bij het inrichten van de cache oplossing is de schaalbaarheid. Als de cache items op de lokale schijf van de webserver worden opgeslagen levert dit problemen op wanner het systeem op twee of meer webservers wordt gedraaid. Dit is een zeer reële opstelling wat kan zorgen voor redundantie of *load balancing*[75]. De cache zou in dat geval tegelijk dubbel en voor de helft opgeslagen worden. Dit probleem kan opgelost worden door de cache te verplaatsen naar een specifiek daarvoor bedoelde machine, of door de caches tussen de servers te coördineren (Eisley, Peh en Shang). Het robuust implementeren van deze tweede optie is zeer complex, er is daarom gekozen om de cache op een andere server te plaatsen. Een nadeel hiervan in dat het zoeken in een database op een aparte server een stuk langzamer zal zijn dan het houden van een cache in het geheugen van de webserver.



FIGUUR 35 MOGELIJKE PLAATSING CACHE

Om een andere server te simuleren tijdens het ontwikkelen wordt de cache in een ander proces gedraaid, communicatie verloopt via een lokale socket. Het redundant uitvoeren van de cache server valt buiten de scope van de opdracht, maar er is geen reden om aan te nemen dat dit niet triviaal is om toe te passen.

De invulling van de daadwerkelijke manier waarop de cache items opgeslagen zullen worden kan grote implicaties hebben op de performance. Het is mogelijk zelf een simpele applicatie te maken die cache items simpelweg als files op het filesysteem wegschrijft. Een betere oplossing zou zijn om een bepaalde database structuur te gebruiken.

Een database kan voor verschillende doeleinden zijn ingericht. Over het algemeen zijn er vier categorieën:

- Relational
- Key-value
- Document-Oriented

Elke architectuur is gericht op een ander type data en heeft voor- en nadelen. Relationele data betekent dat er veel data is in verschillende categorieën die op een bepaalde manier samenhangen. Het is met dit type database mogelijk om data op veel verschillende manieren te queryen. Bij een cache is geen sprake van

[75] http://en.wikipedia.org/wiki/Load_balancing_(computing)

relationele data, er is slechts één type data en één tabel. Zelfs als er later verschillende typen items gecached moeten worden zullen de cache items onderling (waarschijnlijk) geen relatie hebben. Items die in deze database opgeslagen moeten worden zijn exclusief files met een geassocieerde key en een verloop datum. Het gaat hier niet om relationele data, een relationele database valt daarom af. Een key-value store is het eenvoudigste type database, deze kent slechts één tabel met twee kolommen, een primary key en een waarde. Dit type database is een geschikte kandidaat. Een Document-Oriënted database lijkt op een key-value store, een belangrijk verschil is dat de value in dit geval is geoptimaliseerd voor 'documenten'. Deze documenten zijn doorgaans van het XML, JSON, YAML etc. formaat. Een HTML document komt goed overeen met deze formaten. Een document-store is een vorm van een NoSQL[76] database. NoSQL is een term voor databases die niet SQL gebruiken als query taal, en zich ook niet houden aan ACID[77] garanties. Dit houdt kort gezegd in dat de database minder robuust is en ook minder bestendig tegen onverwachte situaties. Tegenover deze beperkingen staat een betere performance. NoSQL databases zijn vrijwel altijd key-value stores, of document stores. Onderzoek wijst uit dat document-stores zich goed lenen voor het implementeren van een webpage-cache. Er is daarom gekozen om de document-cache te gebruiken.

12.2.2 Dependency resolution

Een groot deel van de interfaces binnen dit project heeft slechts één implementatie, toch is het verstandig om de code goed bestendig te maken voor het gebruiken van nieuwe implementaties. Alle klassen in het project hangen af van interfaces in plaats van implementaties. Wanneer een instantie van een klasse aangemaakt wordt moet er een implementatie (dependency) van de vereiste interface worden meegegeven aan de constructor. Traditioneel wordt de gebruikte implementatie (subtype van het meegegeven constructor argument) bepaald tijdens *compile time*[78], Dit houdt in dat er is de keuze van implementatie al in de source code vast ligt. Het is ook mogelijk om de implementatie pas te kiezen tijdens *runtime*[79], dit is veel flexibeler. Een nadeel hiervan is dat mogelijke problemen pas later aan het licht komen. Het dynamisch bepalen van de afhankelijkheden wordt ook *Inversion of control*[80] genoemd. Er zijn meerdere patronen om inversion of control toe te passen.

- Service Locator pattern[81]
- Dependency Injection pattern[82]

Een derde oplossing is het gebruik van het Abstract Factory pattern[83] (Gamma, Helm en Johnson). Dit patroon wordt al toegepast in dit project maar is niet krachtig genoeg om alle problemen op te lossen.

12.2.2.1 Service Locator

Een Service Locator is een verzameling van koppelingen tussen interfaces en implementaties. De koppelingen moeten tijdens de initialisatie van het programma kenbaar worden gemaakt aan de Service Locator. Klassen die constructor argumenten nodig hebben passen zich aan zodat ze alleen een Service Locator nodig hebben, deze wordt dan gebruikt om de 'echte' afhankelijkheden van het object binnen te halen (zie Code 13 en Figuur 36).

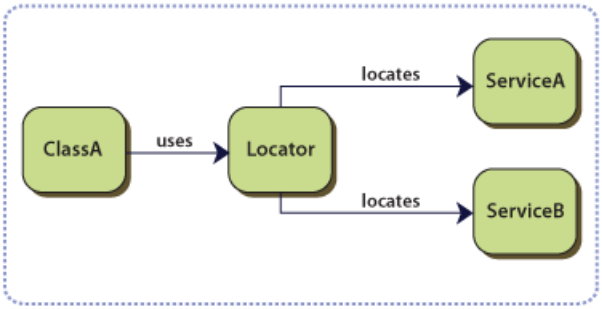
[76] http://nosql-database.org/
[77] http://en.wikipedia.org/wiki/ACID
[78] http://en.wikipedia.org/wiki/Compile_time
[79] http://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)
[80] http://en.wikipedia.org/wiki/Inversion_of_control
[81] http://msdn.microsoft.com/en-us/library/ff921142(v=pandp.20).aspx
[82] http://msdn.microsoft.com/en-us/magazine/cc163739.aspx
[83] Design Patterns blz. 87

Intern kijkt de locator welke implementatie er is gespecificeerd bij de opgegeven interface. Hier wordt dan een instantie van aangemaakt en terug gegeven.

```
interface Engine {}
class V8Turbo : Engine {}

class Car
{
    private Engine _engine;
    public Car(ServiceLocator locator)
    {
        _engine = locator.Resolve<Engine>();
    }
}
```

CODE 13 IMPLEMENTATIE VAN EEN SERVICE LOCATOR

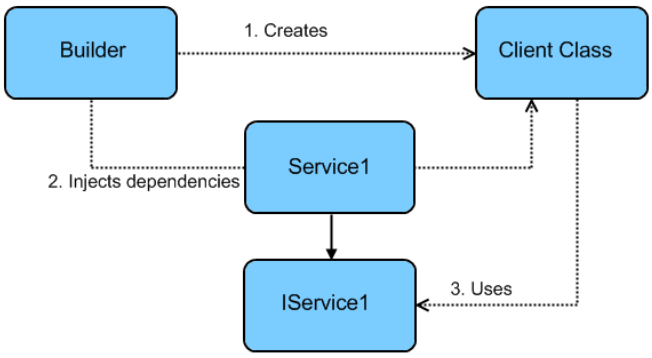


FIGUUR 36 SERVICE LOCATOR

Er zijn een aantal grote nadelen aan het gebruik van een Service Locator, het wordt door sommigen zelfs gezien als een *anti-pattern*[84]. Het belangrijkste probleem is dat alle klassen aangepast moeten worden om deze locator te gebruiken. Daarnaast is het voor de gebruiker van de klasse niet langer duidelijk wat de daadwerkelijke afhankelijkheden van de klasse zijn. Dit maakt de code complexer en moeilijker te doorgronden.

12.2.2.2 Dependency Injection

Het dependency injection patroon werkt op een andere manier. De klasse houdt gewoon zijn constructor en hoeft verder niet gewijzigd te worden. Op een vergelijkbare manier als bij de Service Locator wordt tijdens initialisatie een koppeling gemaakt tussen interface en implementatie. In het programma verloopt instantiëren van objecten gaan dan niet meer door `new()` te gebruiken maar door een `activator` te gebruiken. Er wordt aan de activator gevraagd om een nieuwe instantie van een gegeven object type aan te maken. De activator maakt gebruik van de opgegeven koppelingen om uit te vinden welk type er geïnstantieerd moet worden. Het oplossen van afhankelijkheden is nu niet gekoppeld aan een service locator. Er wordt voor dit project gebruik gemaakt van Dependency Injection.



FIGUUR 37 DEPENDENCY INJECTION

12.3 Functioneel ontwerp

12.3.1 Dependency Injection

Controllers kunnen op dit moment tekst naar de logger schrijven. Deze logger is een singleton waardoor hij vanaf overal benaderd kan worden. Een belangrijk nadeel aan singletons is dat zij een globale staat in het programma introduceren. Bovendien is het mogelijk dat er later een wens ontstaat om twee verschillende loggers tegelijk te gebruiken (een log naar file, en een naar scherm o.i.d.). Er kan per definitie maar één instantie zijn van een singleton en daarom is dit in zo een geval onmogelijk. Een betere oplossing zou zijn om een log klasse mee te geven in de constructor van een controller.

[84] http://blog.ploeh.dk/2010/02/03/ServiceLocatorIsAnAntiPattern.aspx

Op dit moment worden de controller klassen die de web applicatie gebruiken geïnstantieerd door een `ControllerFactory`. Deze factory vertrouwt erop dat de controllers een parameter loze constructor hebben. De factory moet dus aangepast of uitgebreid wordt voordat deze controllers de nieuwe controllers kan bouwen. Hier speelt dependency injection een grote rol. De factory maakt gebruik van een `ControllerActivator`. Deze activator maakt gebuik van een `IDependencyResolver` interface. Hier komt Ninject aan te pas, in de applicatie is Ninject toegewezen als de `IDependencyResolver` implementatie. Ninject zoekt dat de constructor van de klasse op en maakt de gespecificeerde afhankelijkheden aan.

12.4 Technisch ontwerp

12.4.1 Caching

ASP .NET MVC biedt al mogelijkheden om caching toe te passen. Om de cache oplossing goed in het framework te integreren moet deze de `OutputCacheProvider` klasse implementeren. Hiervoor moeten de volgende vier functies worden geïmplementeerd:

```
public abstract object Add(string key, object entry, DateTime utcExpiry);
public abstract object Get(string key);
public abstract void Remove(string key);
public abstract void Set(string key, object entry, DateTime utcExpiry);
```

De `Add` en `Set` functies lijken erg op elkaar. Het verschil is dat de Add functie eerst kijkt of de gegeven key al in de cache bestaat, als dit het geval is wordt het bestaande item opgehaald en terug gegeven, de bestaande waarde wordt dus niet overschreven met de nieuwe waarde. De Set functie zet altijd een waarde in de cache, als het item al bestaat wordt deze overschreven.

Het integreren van een cache oplossing en het implementeren van communicatie met een externe database zijn relatief complex. Als deze twee taken opgesplitst zouden kunnen worden is het makkelijker om te ontdekken waar problemen zitten. Wanneer beide onderdelen werken kunnen deze worden samengevoegd en toegepast. Omwille hiervan is er eerst een cache provider gebouwd die cache items wegschrijft naar het lokale filesysteem. Hiermee kan de correctheid van de cache oplossing worden getest zonder dat er mogelijke fouten met database communicatie of configuratie in de weg zitten. Om de correctheid te bewijzen worden de cache acties opgeslagen in een logfile, het is namelijk relatief moeilijk om dit via unit tests te testen. Als tweede wordt er in een test klasse geprobeerd te verbinden met de database en hierin objecten op te slaan en uit te lezen. Unit tests zullen bewijzen dat de database klasse werkt.

De specifieke document-database die gebruikt wordt moet nog bepaald worden. Hieraan zijn de volgende eisen gesteld:

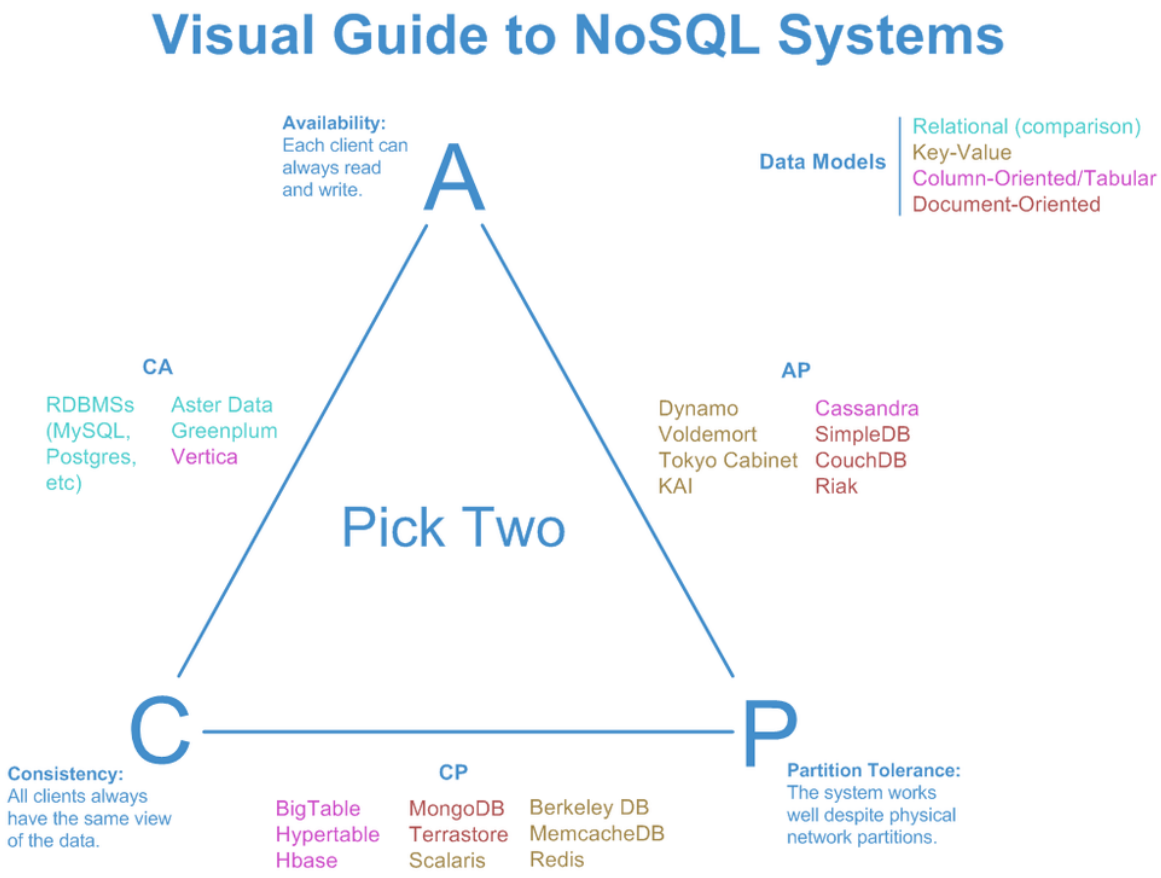
- NoSQL document database structuur.
- Relatief weinig complexiteit nodig.
- Moet aanstuur baar zijn vanuit .NET.
- Een externe database browser tool is een pré.
- Kan goed schalen.

Een belangrijk punt voor een cache database is schaling. Hoe meer administraties er aan de applicatie verbonden worden hoe meer pagina´s er in de cache opgeslagen moeten worden. Wanneer de data of load te groot wordt moet de cache database opgeschaald kunnen worden door er twee of meer aan te sluiten die redundant zijn. De database structuur moet goed blijven werken ook al is deze verspreid over meerdere fysieke locaties.

De CAP theorie (Lynch en Gilbert) (zie Figuur 38) stelt dat het voor een gedistribueert systeem onmogelijk is om tegelijk de volgende drie garanties te geven:

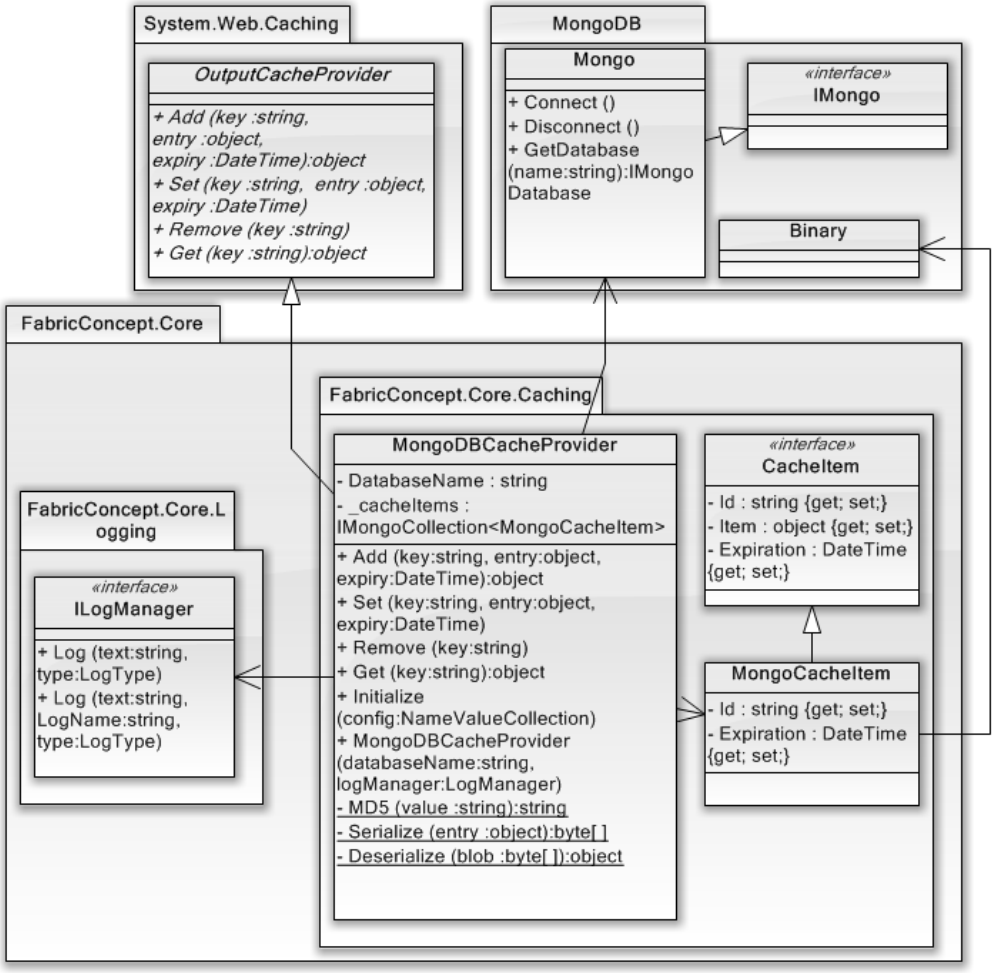
- Consistentie: alle databases bevatten te allen tijde dezelfde data.
- Beschikbaarheid: alle query's krijgen een antwoord over of de query is geslaagd of niet.
- Partitie tolerantie: het systeem blijft werken ondanks verlies van berichten of het falen van een onderdeel van het systeem.

De consistentie is voor een cache systeem belangrijker dan de beschikbaarheid. Er blijven daardoor twee database systemen over: MongoDB en Terrastore. Er is voor Terrastore geen .NET driver beschikbaar. Een driver verzorgt de communicatie tussen applicatie en database. Het is mogelijk er zelf een te schrijven, maar dit valt buiten de scope van de opdracht. MongoDB kent wel een C# driver, er is daarom gekozen voor MongoDB als database voor het caching component van de applicatie.



FIGUUR 38 UITEENZETTING CAP THEORIE

De cache oplossing die naar het lokale filesysteem schrijft is slechts een test systeem, het technisch ontwerp daarvan wordt niet beschreven. De uiteindelijke implementatie wel: de MongoDBCacheProvider leidt af van de standaard OutputCacheProvider van het .NET framework. De C# driver voor MongoDB biedt het Mongo object aan, hiermee kan verbinding gemaakt worden met de database. Daarnaast biedt het een IMongoCollection aan waar de cache items in opgeslagen zullen worden. De ImongoCollection staat op de database. Om het debuggen van het systeem te vergemakkelijken en het ook later mogelijk te maken om gebeurtenissen in de cache te onderzoeken wordt de `LogManager` ook aan deze klasse aangesloten.



FIGUUR 39 KLASSEDIAGRAM CACHEPROVIDER

12.4.2 DEPENDENCY INJECTION

Er bestaan al meerdere Dependency Injection (DI) frameworks die direct ingezet kunnen worden. Het zou onzinnig zijn om dit zelf te implementeren, bovendien valt dat buiten de scope van de opdracht. Het doel is om de *tight-coupling*[85] van bepaalde componenten te reduceren. Voor het .NET framework zijn een aantal gevestigde DI frameworks beschikbaar:

- Castle Windsor
- StructureMap
- Autofac
- Unity
- Ninject

Elke oplossing heeft sterke en zwakke punten. Initieel onderzoek wijst uit dat de verschillen vooral gaat over specifieke situaties en implementatie details. Er wordt verwacht dat voor dit project alleen de gebruikelijke standaard features van een DI framework nodig zijn. Er is geen expertise in huis over het werken met een DI framework en daarom weegt het gemak van gebruik zwaar in de keuze. Een groot voordeel van Ninject is dat

[85] http://en.wikipedia.org/wiki/Coupling_(computer_programming)

deze zeer goed integreert in ASP .NET MVC. Er lijkt ook voldoende documentatie en hulp beschikbaar te zijn op internet. De keuze is daarom gevallen op Ninject.

12.5 IMPLEMENTATIE

12.5.1 CACHING

Per functie van een controller kan aangegeven worden of het resultaat gecached mag worden. Net als het beveiligen van een pagina is de cache instelling een stukje metadata. Dit kan door middel van een attribuut bij een functie worden aangegeven.

Een hash van de opgevraagde URL dient als primary key voor cache items. Voordat het verzoek wordt doorgegeven aan de controller wordt er met behulp van de CacheProvider gekeken of de verzochte pagina in de cache staat, zo niet wordt het verzoek doorgegeven aan de controller en het resultaat in de cache geplaatst. Wanneer hetzelfde verzoek een tweede keer binnenkomt, kan de gecachede pagina geretourneerd worden.

De administratie waarin gewerkt wordt is geen onderdeel van de URL. Alle Id's die aan business objects worden toegekend zijn uniek *binnen die administratie.* Het is mogelijk, en zelfs zeer waarschijnlijk, dat er twee business objects zijn met hetzelfde id, alleen in een andere administratie. Een gebruiker zou dan een gecachede pagina kunnen krijgen van een business object wat het gevraagde id heeft, maar die niet in de huidige administratie zit. De geselecteerde administratie moet dus meegenomen worden in de beslissing of een gecachede pagina getoond kan worden. Dit kan op twee manieren opgelost worden:

- De indeling van de standaard route (zie 5.4.2) wordt aangepast zodat de naam van de administratie hierin wordt meegenomen.
- De logica van de cache oplossing wordt uitgebreid zodat deze controleert welke administratie geselecteerd is deze waarde meeneemt bij het genereren van de key voor het cache item.

De voor en nadelen van het toevoegen van de administratie in de route zijn in 10.4.1 al overwogen. Aan dit besluit wordt gehouden, de cache logica moet dus uitgebreid worden. Dit wordt gedaan door een callback te specificeren die de naam van de huidige administratie terug geeft. De CacheProvider roept deze aan wanneer er een key gemaakt moet worden om een item toe te voegen of te zoeken. Voorheen bestond de key uit een hash van de URL, nu wordt de naam van de administratie hierin meegenomen. Op die manier blijven cache items met dezelfde URL van elkaar gescheiden als zijn aan een andere administratie behoren.

Een pagina kan opgebouwd zijn uit meerdere sub pagina's (ook wel partial views genoemd). De login control die in dit project gebruikt wordt is een partial view. Hier staat de naam van de ingelogde gebruiker. Wanneer een pagina gecached wordt moet dit hier niet in meegenomen worden, anders krijgen alle gebruikers de naam te zien van de éérste gebruiker die de pagina opvroeg. Om dit tegen te gaan wordt er *donut caching*[86] ingezet. Dit houdt in de het grootste gedeelte van de pagina wordt gecached, behalve een klein stukje (het gat van de donut). Op de login control wordt dan geen caching toegepast.

Wanneer een business object is gewijzigd moeten de cache items van de edit en details pagina geïnvalideerd worden. In de `Edit` functie moet de CacheProvider worden opgedragen om de relevante cache items te verwijderen. Een onhandigheid in het huidige ontwerp is dat clients die de `Edit` functie van de `FabricController` overschrijven zelf moeten letten op het invalideren van de cache. Deze waarschuwing kan gedocumenteerd worden maar het is goed mogelijk dat deze waarschuwing over het hoofd wordt gezien. Zeker als het om een nieuwe ontwikkelaar gaat die nog onbekend is met het systeem kan het voorkomen dat deze de caching regels niet in acht neemt. Dit is eigenlijk juist goed, clients zouden onafhankelijk moeten zijn

[86] http://mvcdonutcaching.codeplex.com/

van het feit dat bepaalde pagina's gecached worden. Het zou daarom beter zijn om het ontwerp aan te passen zodat clients niet met het invalideren van de cache te maken hebben. Om dit te realiseren wordt er gebruik gemaakt van de template method. Een niet virtuele `Edit` functie roept een virtuele `Edit` aan. Zodra deze `Edit` klaar is worden de relevante items uit de cache geplaatst en de pagina geretourneerd (zie Code 14).

```
public ActionResult MasterEdit(TFabricObject EditedFabricObject)
{
    var result = Edit(EditedFabricObject);
    var id = GetIdPropertyValue(EditedFabricObject);
    RemoveFromCache(id);
    return result;
}
```

CODE 14 TOEPASSING VAN TEMPLATE METHOD OM CACHE INVALIDATIE TE VERPLICHTEN

12.5.2 DEPENDENCY INJECTION

Om gebruik te maken van Ninject moeten de benodigde libraries aan het project worden toegevoegd. Voor elke interface moet een implementatie gespecificeerd worden. Dit gaat door een nieuwe klasse aan te maken die afleidt van `NinjectModule`. Hierin moet de functie `Load` worden overschreven, hierin kan een interface aan een implementatie worden gekoppeld (zie Code 15). Voor elke interface kan een aparte module worden aangemaakt. Deze modules worden tijdens initialisatie van de applicatie geladen.

```
this.Bind<Core.Logging.ILogManager>()
    .To<Core.Logging.LogManager>()
    .InSingletonScope()
    .WithConstructorArgument("path", LogPath)
    .WithConstructorArgument("file", "Default.log")
    .WithConstructorArgument("logLevel", logLevelEnum);
```

CODE 15 KOPPELING VAN INTERFACE EN IMPLEMENTATIE VIA NINJECT

Klassen die voorheen hun eigen dependencies aanmaakten worden aangepast zodat deze dependencies via de constructor binnen komen. De `ControllerFactory` kan nu controllers aanmaken die constructor argumenten hebben. In de praktijk wordt dit gebruikt om onder andere de `ILogManager` en de `IRepository` klasse door te geven aan de controllers. Daarnaast kunnen ook andere dependencies tussen objecten op een abstracte manier worden opgelost. Dit ontkoppelt de afhankelijkheden van de applicatie en zorgt voor meer flexibiliteit.

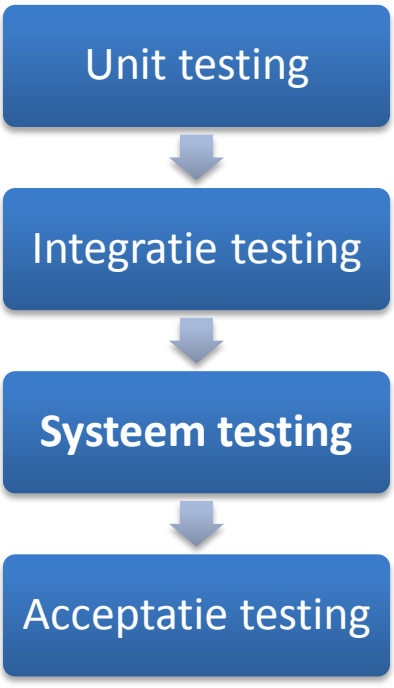
13 Sprint 9 "Afronding"

De laatste sprint is gericht op het opnieuw testen van alle code en daarna het verbeteren van de kwaliteit van de code. Er zijn een aantal verschillende methoden waarmee de kwaliteit van code onderzocht kan worden en verbeterd. De code is éérst verbeterd, daarna zijn alle klassen onderworpen aan een abstractie en afhankelijkheid analyse. De uitkomst van deze analyse zegt iets over de indeling en samenhang van de complete oplossing.

Naar het verbeteren van de code is tijdens deze sprint ook het bevindingenrapport van Fabric geschreven.

13.1 Testing

Gedurende het project zijn alle klassen getest door middel van unit tests. Een groep klassen vormt samen een component, de componenten zijn getest met integratie tests. Het complete systeem is nu af wat betekent dat de volgende stap in het testproces uitgevoerd kan worden, namelijk de systeem test. Een systeem test controleert de werking van het totale systeem door meerdere componenten die invloed op elkaar hebben tegelijk te testen. Systeem testing kan bij dit project het beste gedaan worden door het systeem daadwerkelijk te gebruiken. Doordat het niet geautomatiseerd is kan het wel langer duren, maar de ontwikkelaar van het product heeft op die manier wel meer zekerheid over de werking ervan.



FIGUUR 40 STAPPEN IN HET TESTPROCES

13.2 Kwaliteitscontrole

De kwaliteit van de ontwikkelde software moet constant gecontroleerd worden, aan het einde van het project is er nog eens extra tijd genomen om bepaalde onderdelen van de applicatie opnieuw te bekijken met als doel het controleren van de oplossing. In sommige gevallen is het mogelijk geweest om kleine problemen te verhelpen maar veelal heeft de kwaliteitscontrole geleidt tot een aanpassing ten behoeve van de uitbreidbaarheid of flexibiliteit van de code.

13.2.1 Static code analysis

Het controleren of het programma naar behoren werkt is tot nu toe gedaan door runtime tests uit te voeren. Het programma wordt hierbij opgestart en een actie wordt uitgevoerd, als het verwachte resultaat gelijk is aan het daadwerkelijke resultaat werkt dat deel van het programma correct. De runtime tests kunnen zowel unit tests als integratietests zijn. Er is ook nog een andere methode om de correctheid van het programma te meten die problemen aan het licht kan brengen die door middel van unit tests en dergelijke erg moeilijk, of zelfs onmogelijk zijn om te ontdekken. Dit is static code analysis[87], hierbij wordt niet het programma, maar de source code geanalyseerd. Deze analyse kan wijzen op subtiele bugs of problemen met bepaalde constructies die niet zichtbaar zijn tijdens door alleen te kijken naar de uitvoering van het programma. Visual Studio heeft een ingebouwde code analyzer. Deze kan geconfigureerd worden met verschillende rulesets die zich richten op

[87] http://en.wikipedia.org/wiki/Static_program_analysis

andere gebieden. Het uitvoeren van deze analyse en het nakijken van vermeende problemen of waarschuwingen heeft de kwaliteit van de code verbeterd.

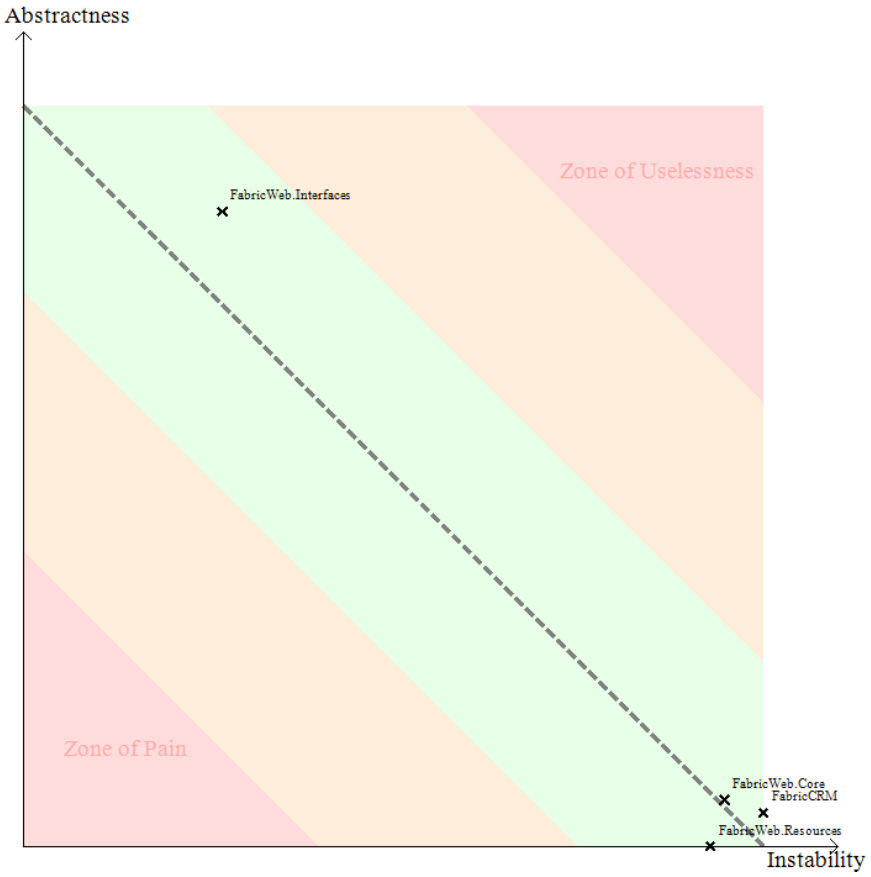
13.2.2 REFACTORING

Refactoring[88] is het herstructureren van een stuk code, waarbij de interne structuur wordt verbeterd zonder dat de externe werking verandert. Er bestaande verscheidene plugins voor Visual Studio die assisteren bij het refactoren van code. Een van deze plugins is ReSharper[89], deze geeft aanwijzingen en suggesties over het herstructureren van de code. Veelal komt dit de leesbaarheid ten goede maar ook wordt er zo veel mogelijk geprobeerd om de code te reduceren naar het minst nodige. Er is op zo veel mogelijk plaatsen code ingekort waardoor het juist makkelijker te lezen en te begrijpen is. Met ReSharper zijn er nog een aantal kleine foutjes ontdekt die in de static code analysis van Microsoft niet naar boven kwamen.

13.3 PACKAGING

Er is tijdens het project veel gelet op het indelen en ontwerpen van de componenten zodat zij voor zover mogelijk onafhankelijk van elkaar kunnen werken. Daarnaast zijn alle componenten die logisch bij elkaar horen in groepen ingedeeld. Het is wenselijk om alle klassen op de juiste manier in te delen zodat de onderlinge afhankelijkheden zo veel mogelijk één kan op wijzen.

Zodra alle code is afgerond is er opnieuw gekeken naar de indeling van het systeem. Hierbij zijn er opnieuw componenten verplaatst om de samenhang van het systeem te verbeteren. Met behulp van het programma Ndepend[90] kan er inzicht verkregen worden in verscheidene *metrics* van de code, waaronder de abstractheid, instabiliteit, cohesie etc.



FIGUUR 41 GRAFISCHE WEERGAVE VAN DE INSTABILITEIT UITGEZET TEGEN DE ABSTRACTIE

Tijdens het ontwerp van alle componenten is er al rekening gehouden met de ontwerp parameters. Dit zijn een aantal getallen die per klasse een indicatie geven hoe die klasse samenhangt met het systeem. Met deze getallen kan gerekend worden en de uitkomsten hiervan kunnen geplot worden op een grafiek. Er bestaat een ideale lijn waar deze punten op moeten vallen. Op Figuur 41 is een grafische weergave van een van de belangrijkste metrics te zien. Tabel 13 geeft de waarden weer waaruit de grafiek is voortgekomen.

[88] http://refactoring.com/
[89] http://www.jetbrains.com/resharper/
[90] http://www.ndepend.com/

TABEL 13 CODE METRIEKEN VAN DE PACKAGES

Package	Afferente koppelingen	Efferente koppelingen	Cohesie	Instabiliteit	Abstractheid	Afstand van lijn
FabricWeb	0	240	1.2	1.2	0.05	0.03
FabricWeb.Interfaces	19	7	0.43	0.27	0.86	0.09
FabricWeb.Core	6	108	0.69	0.95	0.06	0.01
FabricWeb.Resources	1	13	0.06	0.93	0	0.05

13.4 FABRIC BEVINDINGENRAPPORT

Gedurende de loop van het project zijn alle bevindingen met betrekking tot Fabric (d.w.z. mankementen, gemissen etc.) bijgehouden. Voor elke bevinding is een beknopte beschrijving opgenomen en verwijzingen naar de relevante stukken code. Wanneer een bevinding de voortgang van het project blokkeert is deze direct gemeld zodat er gewerkt kan worden aan een mogelijke oplossing. Daarnaast zijn kleine foutjes ook direct gemeld, hiervoor is geen formele beschrijving nodig omdat het om een relatief eenvoudig probleem gaat.

In deze laatste sprint zijn alle bevindingen verzameld en omgezet naar een standaard formaat. Alle bevindingen hebben een prioriteit en een status, toegewezen gekregen. Daarnaast wordt de bevinding kort beschreven, er wordt, indien bekend, een mogelijke oorzaak gegeven en daarbij ook een mogelijke oplossing.

De verzameling bevindingen is onderzocht en daaruit is een conclusie getrokken m.b.t. de kwaliteit en paraatheid van Fabric.

Alle bevinding zijn terug te vinden in het Fabric Bevindingenrapport (zie bijlage IX).

14 DEPLOYMENT

De deployment is de laatste fase van het project, hierin worden de op te leveren producten overgedragen aan de opdrachtgever. De meeste tijd en moeite binnen dit project is besteed aan het realiseren van de web applicatie, dit is het hoofdproduct. Hiermee kunnen de mogelijkheden van Fabric gelijk worden gedemonstreerd. De aanpassingen aan Fabric en het bevindingen rapport hebben een heel ander doeleinde, namelijk het verbeteren van de kwaliteit van Fabric.

14.1 INTERNE OPLEVERING

De functionaliteit van de applicatie is tijdens het verloop van het project constant gedemonstreerd aan de opdrachtgever. De applicatie draaide hierbij altijd op de relatief eenvoudige development server van Visual Studio. De uitlevering van de applicatie vraagt om een volledige webserver.

De web applicatie is intern al succesvol gedemonstreerd aan een andere afdeling, hierbij is de algemene functionaliteit gedemonstreerd en dit is goed ontvangen.

14.1.1 ACCEPTATIE

Nadat de systeemtest is afgerond is het product klaar om overgedragen te worden. Voordat de opdrachtgever akkoord gaat met het opgeleverde product vindt er eerst een acceptatie test plaats. De acceptatietest wordt vaak door de opdrachtgever zelf gespecificeerd, maar in dit geval wordt de acceptatietest zelf aangeleverd. Een acceptatie test bestaat doorgaans uit het uitvoeren van alle user stories, de user stories met een Must prioriteit moeten geïmplementeerd zijn. Van de user stories met een lagere prioriteit dan Must kan de opdrachtgever zelf beslissen of het ontbreken ervan belangrijk genoeg is om het product daarom te weigeren.

Ten tijde van schrijven is de acceptatietest door de opdrachtgever nog niet uitgevoerd en/of goedgekeurd. Alle user stories met een Must prioriteit werken naar behoren, het grootste deel van de Should have's zijn ook met succes in het product opgenomen.

14.2 DOCUMENTATIE

14.2.1 SOURCE CODE

Documentatie kan op twee manieren worden toegepast. Statements of blokken code kunnen van een stuk commentaar worden voorzien of een klasse of functie kunnen gedocumenteerd worden. Deze twee vormen van documentatie hebben verschillende doeleinden.

Het documenteren van een stuk code heeft als doel uitleg geven en inzicht bieden in de oplossing. Hier kan kort worden aangegeven waarom de oplossing is ingericht zoals het is, of waarom een meer voor de hand liggende oplossing niet mogelijk was. Het doel hiervan is om andere ontwikkelaars inzicht te geven in de totstandkoming van de code. Vragen die zij zouden kunnen stellen bij een stuk code moeten door dit soort commentaar beantwoord worden. Het is hierbij belangrijk om uit te leggen *waarom* de oplossing zo is, en niet *hoe* het werkt, dit zou voor een competente ontwikkelaar duidelijk moeten zijn.

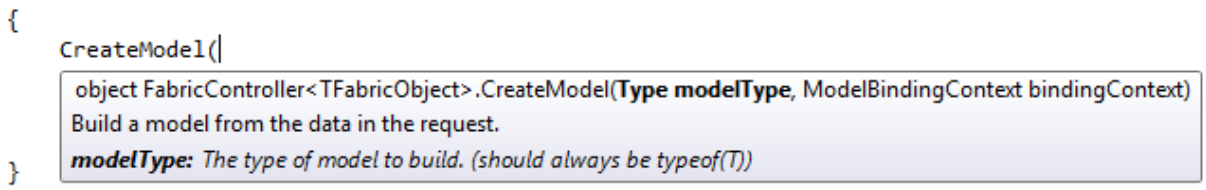
Een tweede manier van documenteren is het toelichten van het doel van klassen, functies en membervariabelen. Dit geeft inzicht in het doel van componenten en hoe deze zich zullen gedragen. Visual Studio toont automatisch het commentaar bij een functie op de plek waar deze wordt aangeroepen. Hiervoor

moet het commentaar wel aan een paar eisen voldoen. Er zijn door Microsoft een aantal richtlijnen opgesteld waar de XML documentatie aan moet voldoen, zoals welke tags en attributen gewenst zijn (zie Figuur 42).

```
/// <summary>
/// Build a model from the data in the request.
/// </summary>
/// <param name="modelType">The type of model to build. (should always be
typeof(T))</param>
/// <param name="bindingContext">The bindingContext</param>
/// <returns>The constructed object</returns>
public virtual object CreateModel(Type modelType, ModelBindingContext bindingContext)
```

FIGUUR 42 XML STIJL COMMENTAAR BIJ EEN FUNCTIE

Dit commentaar wordt door Intellisense[91] automatisch aan de ontwikkelaar getoond wanneer hij deze functie aanroept, zie hieronder.



FIGUUR 43 COMMENTAAR WORDT GETOOND TIJDENS INVULLEN VAN EEN FUNCTIE

Alle XML commentaar is met behulp van tools worden omgezet in HTML pagina's met een nette opmaak, deze tools lezen de broncode in en verwerken alle XML commentaar. Alle publieke functies zijn voorzien van XML commentaar. In bijlage VIII staat de documentatie van de applicatie.

[91] http://msdn.microsoft.com/en-us/library/hcw1s69b(v=vs.110).aspx

DEEL III: UITKOMST

Dit is het laatste deel van het verslag. Hierin worden eerst een aantal evaluaties uitgevoerd, daarop volgt een aanbeveling en een conclusie. Een korte begrippenlijst gaat vooraf aan het laatste hoofdstuk waarin de gebruikte literatuur wordt opgegeven.

Hoofdstuk 15 Evaluatie:

Als eerst staat hier een evaluatie van het uiteindelijke product, gevolgd door een evaluatie van het proces dat naar dit product heeft geleid. Hierbij wordt gekeken wat er goed ging en wat minder goed is gegaan. Een zelfreflectie geeft inzicht in de ontwikkeling die de afstudeerder heeft doorgemaakt en hoe hij het traject heeft ervaren.

Hoofdstuk 0 Aanbevelingen en conclusies:

De bevindingen van het Fabric bevindingenrapport worden hierin samengevat en er worden aanbevelingen gedaan m.b.t. de verdere ontwikkeling van Fabric.

Hoofdstuk 17 Begrippenlijst en **Hoofdstuk 18 Bibliografie:**

Behoeven geen verdere toelichting.

15 EVALUATIE

In dit hoofdstuk worden een aantal verschillende aspecten van het project geëvalueerd. Als eerst wordt het opgeleverde product kritisch bekeken, daarop volgt een evaluatie van het proces, hierbij wordt bekeken of de methoden en technieken goed zijn uitgekozen en correct zijn toegepast. De planning van het project en de uitgevoerde werkzaamheden worden bekeken, de procesevaluatie sluit af door te kijken welke aspecten er goed gingen en wat er minder goed verliep. Als laatst komt een zelfreflectie aan bod, de afstudeerder blikt dan terug op zijn persoonlijke ontwikkeling.

15.1 PRODUCT EVALUATIE

Dit project heeft drie belangrijke artikelen opgeleverd:

- De demo web applicatie.
- Aanpassingen en verbeteringen in het Fabric framework.
- Een rapport over het werken met Fabric met daarin vanuit mij gezien tekortkomingen, problemen en fouten in het framework.

De demo web applicatie heeft tijdens het project veel feedback ontvangen van zowel collega´s als de opdrachtgever. Mede door deze constante feedback is het mogelijk geweest om problemen al in een vroeg stadium aan te pakken. Dat is de applicatie ten goede gekomen. De feedback is veelal positief geweest wat voor meer zekerheid heeft gezorgd tijdens de ontwikkeling ervan.

De web applicatie stelt een aantal mogelijkheden van Fabric ten toon. Het proces van het toevoegen, wijzigen en beheren van orders is in duidelijke stukken onderverdeeld die goed samenwerken. De applicatie kan ingezet worden als demo en ook als voorbeeld voor andere ontwikkelaars. De code is goed gedocumenteerd en de implementatie probeert zich te houden aan de voorschriften van een Csla web applicatie zoals bedoeld door de ontwikkelaar van Csla. Ook wordt er goed gebruik gemaakt van de verschillende componenten van ASP .NET MVC, opnieuw op een manier die overeenkomt met de insteek van het web framework.

In de applicatie is caching toegepast, hoewel niet direct onderdeel van Fabric of strikt noodzakelijk voor een demo komt het wel van pas wanneer de code gebruikt wordt als *voorbeeld.* Het cachen van data is tegenwoordig zo belangrijk om de performance van web servers te verbeteren dat dit een vrijwel noodzakelijk onderdeel van een web applicatie is. Ditzelfde geldt voor de vrij hoge configureerbaarheid van de applicatie. Waar meerdere opties mogelijk zijn wordt meestal de configuratiefile geraadpleegd om te controleren of er een voorkeur is ingegeven. Ook aan de compile time flexibiliteit is gedacht door gebruik te maken van een dependency injection framework. Dit maakt het opnieuw makkelijker voor ontwikkelaars om delen van de demo-code te kopiëren maar er een andere invulling aan te geven.

De functionaliteit van de applicatie is beknopt, het beperkt zich voornamelijk tot de factuur en orderverwerking kant van een boekhoudpakket. Omdat het hier om een demo gaat is dit juist wenselijk, de code en de functionaliteit is goed te begrijpen en daarom voldoet het product aan de gestelde eisen.

De aanpassingen aan het framework zijn in overleg, en samen met de begeleider gemaakt. Bij de nieuwe `PageSet` klasse zijn unit tests gemaakt die de werking ervan bevestigen. Voordat de nieuwe code is toegevoegd aan Fabric is eerst zeker gemaakt dat deze geen incompatibiliteiten introduceert. De `PageSet` is algemeen toepasbaar en inzetbaar voor alle klassen die lijsten van data bijhouden. De implementatie probeert op een overzichtelijke en efficiënte manier lijsten op te splitsen in kleine stukken. Het ophalen van data is hierdoor in veel gevallen vele malen sneller.

Het rapport over Fabric gaat niet alleen over delen die niet, of slecht werken, maar ook over dingen die wel werken maar niet intuïtief zijn. Het is voor een framework van groot belang dat ontwikkelaars er gemakkelijk mee kunnen werken. Tijdens het ontwikkeltraject is de *mindset* van een partner aangehouden. Als er bepaalde componenten niet handig, of op een onverwachte of vervelende manier werken is hier melding van gemaakt. In veel gevallen wordt er ook een beter alternatief gegeven of zelfs een implementatie voorgesteld. UNIT4 kan voor elk item uit dit rapport bepalen of ze het eens zijn met de kritiek en of het überhaupt mogelijk is een alternatief te implementeren. Dit rapport stelt UNIT4 in staat om de publieke interface van Fabric vriendelijker te maken voor ontwikkelaars.

15.2 Proces evaluatie

In deze paragraaf wordt ingegaan op het traject dat is afgelegd om tot het eindresultaat te komen. Er wordt terug geblikt op de effectiviteit en doeltreffendheid van de gekozen methoden en technieken, de accuraatheid en haalbaarheid van de planning en er volgt een uiteenzetting van de belangrijkste dingen die goed en minder goed zijn gegaan.

15.2.1 Methoden en technieken

Het gebruik van scrum om de realisatie van het product te beheren lijkt een juiste geweest te zijn. De flexibiliteit met betrekking tot het toevoegen van additionele user stories, of het doorschuiven van niet afgemaakte stories is goed benut. De onzekerheid aan het begin van het project was relatief groot, scrum kan hier goed mee overweg.

De gebruikte frameworks hebben allemaal de juiste hoeveelheid werk uit handen kunnen nemen. Vooral de flexibiliteit van ASP .NET MVC heeft veel voordelen gehad, het biedt al standaard functionaliteit voor routing, caching, authenticatie en anderen die uitgebreid kan worden waar nodig.

Een techniek die zich niet helemaal heeft kunnen bewijzen is MongoDB, deze is zeker toereikend voor de gestelde doeleinden, maar juist daar ligt de twijfel. De keuze om deze database te gebruiken heeft eigenlijk te veel geleund op het feit dat het de meest toegankelijke keuze was. Er waren een aantal beknopte tutorials beschikbaar en ook de C# driver ervoor was al kant-en-klaar. Er zijn geen klachten over de performance van de database, maar het is zeker mogelijk dat er een betere keuze bestaat voor de gestelde doeleinden. De applicatie is wel zo ingericht dat het wijzigen van de onderliggende database zeer makkelijk is.

15.2.2 Planning

De planning zoals opgesteld aan het begin van het project heeft al twee keer een wijziging doorgemaakt. Het betrof hier geen grote wijzigingen, maar dit geeft wel aan dat de initiële planning niet goed genoeg doordacht was. Het project heeft goed gelopen en de sprints zijn op uitzondering van één allemaal in twee weken doorlopen.

De indeling van de werkzaamheden heeft meer om de sprint planning gedraaid dan de globale planning. Het is bij de aanvang van het project niet mogelijk geweest een gedetailleerde planning te maken. Voor de sprint planning is dit wel mogelijk geweest maar hier is van af gezien omdat het als onnodig werd geacht. Achteraf gezien had het wellicht wel toegevoegde waarde kunnen hebben. Op enkele momenten is de vereiste tijd om een user story te implementeren overschat (initiële concept) of juist grof onderschat (paging). Een gedetailleerde planning waarin per activiteit de tijdsduur wordt geschat en ook onderlinge afhankelijkheden

tussen activiteiten zijn aangegeven had deze inschattingsfouten mogelijk kunnen voorkomen, of in ieder geval de impact verzachten.

15.2.3 WAT GING ER GOED

Het hele project is over het algemeen goed verlopen. In de eerste sprint zijn er veel dingen onderzocht en geanalyseerd over welke technieken er het beste bij het probleem passen en op welke manier ze het beste gebruikt kunnen worden. Daardoor is er in de tweede sprint snel een werkend concept opgeleverd. Dit heeft al vroeg vertrouwen gegeven in de aanpak. Tijdens het project is er nog veel tijd besteed aan het onderzoeken van de mogelijkheden van ASP .NET MVC4, de sterke punten van het framework zijn hierdoor zo veel mogelijk toegepast.

Het beheer van het project is ook goed verlopen, in de meeste sprints zijn alle user stories in de opgegeven tijd geïmplementeerd. Omdat je met scrum kunt afzien van een lange termijn planning is het mogelijk geweest om tijdens het project goed om te gaan met de veranderende requirements.

De implementatie van de applicatie is over het algemeen voorspoedig gegaan, in een aantal gevallen is het ontwerp van een component herzien. Dit is meestal ten behoeve van de uitbreidbaarheid geweest. De unit tests hebben tijdens de ontwikkeling zekerheid gegeven dat de applicatie constant naar behoren werkt.

15.2.4 WAT GING ER MINDER GOED

De kwaliteit van het bevindingen rapport heeft enigszins geleden onder de te beknopte documentatie van gevonden problemen in Fabric. Er is vaak niet genoeg vast gelegd over een probleem om er zonder moeite later een rapport over de schrijven. Het is af en toe nodig geweest om terug in de code te duiken om erachter te proberen te komen hoe het probleem precies zat. Dit vooral vervelend wanneer het probleem ondertussen al is opgelost. Dit had voorkomen kunnen worden door de problemen onmiddellijk volledig te documenteren zodra ik ze tegenkwam, in plaats van alleen de kern van het probleem te noteren.

De performance problemen, met name het paging, is een groter probleem geweest dan aanvankelijk geschat. Dit heeft betekent dat er eerst tijd verloren is gegaan aan het maken van een oplossing die al snel ontoereikend bleek te zijn.

15.3 ZELFREFLECTIE

15.3.1 PERSOONLIJKE ONTWIKKELING

Ik heb tijdens het afstuderen heel veel geleerd. Niet alleen op gebied van programmeren maar ook het werken in een professionele omgeving en het zelf beheren van een project. Ik heb nieuwe ontwikkelpatronen toegepast en constructies gemaakt waar ik voorheen nog niet mee bekend was. Hierdoor heb ik ook veel specifieke .NET kennis opgedaan. Over het algemeen heb ik veel geleerd als software ontwikkelaar. Doordat ik iedere dag heb meegedaan aan de daily scrum met de hele afdeling heb ik ook meegemaakt hoe het ontwikkel-, onderhoud- en testproces bij een groot bedrijf verloopt.

Voordat ik begon aan deze opdracht wist ik absoluut zeker dat ik zou willen verder studeren. Naarmate ik langer aan de opdracht bezig was begon ik hier steeds meer te twijfelen. Aan het einde van de opdracht heeft UNIT4 mij een baan aangeboden als software ontwikkelaar en deze heb ik met veel plezier aangenomen.

16 CONCLUSIE EN AANBEVELINGEN

De aanbevelingen en conclusies die dit project heeft opgeleverd komen voor een groot deel voort uit het Fabric bevindingenrapport (zie bijlage X).

16.1 CONCLUSIE

16.1.1 FABRIC

Het ontwikkelen met Fabric is over het algemeen voorspoedig verlopen, het framework is netjes ingedeeld en is strak opgedeeld in lagen. Er is veel flexibiliteit, doordat alle query's in een LINQ formaat worden ingevoerd kan de onderliggende database in een handomdraai worden gewijzigd in een ander type database. De complete keten van een Fetch aanroep naar een lijst objecten is gemakkelijk te volgen, wat ervoor zorgt dat het debuggen van problemen minder moeilijk is. De business objects zitten logisch in elkaar en kunnen daardoor ingezet worden zonder documentatie te hoeven lezen, vrijwel alle objecten zitten consistent in elkaar en houden zich aan dezelfde conventies en stijlen.

Fabric kent nog een aantal kleine problemen, deze zijn over het algemeen alleen merkbaar in bepaalde situaties en in sommige gevallen is er een omweg mogelijk om toch het gewenste resultaat te bereiken. Daarnaast spelen er echter wel een aantal grotere problemen die flink in de weg zitten bij het bouwen van een applicatie. Die problemen gaan voornamelijk over het ondersteunen van meerdere gebruikers, dit is een belangrijk probleem welke complex kan zijn om op te lossen.

De functionaliteit van Fabric is verbeterd door onder andere paging toe te voegen en de Fetch operaties uit te breiden waardoor er nauwkeuriger gezocht kan worden. Tijdens het project zijn er problemen aan het licht gebracht die direct opgelost konden worden, de andere problemen zijn beschreven in het bevindenrapport (bijlage X), het oplossen van deze problemen zal de kwaliteit van Fabric ten goede komen.

16.1.2 WEB APPLICATIE

De web applicatie geeft de verschillende functionaliteiten van Fabric goed weer, het grootste deel van de business objects maken een verschijning in de applicatie. De applicatie kan zonder meer dienen als proof of concept van een web applicatie die op Fabric draait, wanneer deze demo aan niet-ontwikkelaars wordt gegeven is het wellicht gunstig om de interface op in ieder geval een aantal plaatsen te verbeteren. De focus van dit project lag op het implementeren van de functionaliteit, er is daarom minder tijd besteed aan het opmaken van de interface. Geïnteresseerden kunnen niet altijd het onderscheid maken tussen interface en functionaliteit en dit kan afdoen aan het product.

Doordat de code goed gedocumenteerd is kan het project ingezet worden als referentie materiaal voor ontwikkelaars die voor het eerst met Fabric gaan werken.

16.2 AANBEVELINGEN

Wanneer de problemen met globaal cachen opgelost zijn en er is ondersteuning voor meerdere gebruikers door middel van sessie id's kan er een eerste versie van Fabric worden opgeleverd. Er moet echter nog wel veel functionaliteit worden toegevoegd voordat Fabric zijn voorloper kan vervangen.

17 Begrippenlijst

17.1 Afkortingen

Afkorting	Betekenis
AJAX	Asynchronous JavaScript and XML
BCL	Base Class Library
CRM	Customer Relationship Management
CSLA	Component-based Scalable Logical Architecture
JSON	JavaScript Object Notation
MVC	Model View Controller
RUP	Rational Unified Process
SDLC	Systems Development Life-cycle
SMART	Specifiek, Meetbaar, Aanvaardbaar, Realistisch, Tijdgebonden
UML	Unified Modeling Language

17.2 Begrippen

Begrip	Uitleg
Base Class Library	Dit is de standaard library die bij het .NET framework is geleverd. Vergelijkbaar met de STL, maar biedt meer functionaliteit
Binding	Het maken van een verbinding tussen twee datatypen.
Business layer	De abstractie laag in een software architectuur waar de business objects en business rules zich bevinden.
Business object	Een object dat zich in de business-layer bevindt. Vaak een representatie van een 'echt' object zoals een klant of factuur. Bevat meestal veel velden, maar is beperkt in functionaliteit.
Business rules	Regels waar een business object zich aan moet houden. Voorbeeld: bepaalde velden moeten ingevuld worden of aan voorwaarden voldoen.
Compile time	Het moment wanneer een project gecompileerd wordt.
Credentials	Authenticatie gegevens die opgegeven worden om toegang tot een systeem te verkrijgen.
CRUD	*Create, Update, Read, Delete.* Simpele operaties om data aan te maken, uit te lezen, aan te passen of te verwijderen.
Customer relations	Klant relaties
Database hit	Één query die op de database wordt uitgevoerd.
Deferred execution	Uitstellen van uitvoering tot het resultaat opgevraagd wordt.
Deliverables	Op te leveren items.
Framework	Letterlijk: geraamte. Een set libraries die samen een universeel, herbruikbaar software platform vormen waar applicaties mee gebouwd kunnen worden.
Gebruikers interface	Hetgeen waar de gebruiker communiceert met het systeem. Veelal een grafisch interface.

Generic	Letterlijk: generiek. Datastructuren die opereren op typen die later pas gespecificeerd hoeven te worden. Bijvoorbeeld een List<T>.
Hardcoded	Geprogrammeerd op een niet-flexibele manier, meestal bij ogenschijnlijk simpele dingen. De source moet aangepast worden en opnieuw gecompileerd voor een verandering.
Hooking	Optineel het gedrag van een operatie aanpassen door een extra functie te laten aanroepen. Kan alleen indien de originele functie dit expliciet ondersteunt.
Library	Een serie datastructuren en algoritmen die samen een concept omvangen en door ontwikkelaars gebruikt kan worden om software te ontwikkelen.
Load balancing	Het verspreiden (balanceren) van het werk over meerdere uitvoerders. Deze uitvoerders kunnen computers zijn, maar ook cores op een CPU of netwerk links.
Locking	Het afschermen van gedeelde variabelen om race condities uit de weg te gaan.
Look and feel	Kwalificatie voor het ontwerp en gebruik van een interface.
Masterpage	Hoofdpagina binnen een ASP .NET applicatie waar de layout van de website wordt vastgelegd.
Metadata	'Data over data', gegevens die gaan over andere gegevens. De grootte van een set items.
Multi-Tenancy	Het behandelen en verwerken van meerdere clients door één softwaresysteem.
Product backlog	Lijst waarin alle user stories zich bevinden.
Productvisie	Een abstract idee van wat een product moet kunnen en hoe het eruit ziet.
Proof of concept	Een applicatie die een bepaald concept ten toon stelt, bewijzen dat een idee (op kleine schaal) werkt.
Querystring	Een van twee manieren om parameters door te geven aan een webserver. Dit gebeurt door parameters op de volgende manier in de link te zetten. 'website.com/customer/index?Name=piet&city=rotterdam'.
Race conditie	Een stuk code waarbij het resultaat onverwacht af kan hangen van de volgorde of timing van andere gebeurtenissen.
Record	Een rij in een database.
Requirements	Eisen van een stuk software.
Resource file	Een bestand dat een 'hulpbron' bevat. Dit kan een serie strings zijn, maar ook een afbeelding.
Run time	De tijd wanneer het programma uitgevoerd wordt.
Runtime	De omgeving waar software in uitgevoerd wordt. Voor .NET code is dat de CLR (Common Language Runtime), C en C++ worden niet uitgevoerd in een runtime maar draaien direct op de processor.
Server load	De mate van belasting van de server. Als er veel requests binnen komen zal de load hoog zijn, als de load te hoog wordt kan de server het werk niet aan.
Side effect	Een bijwerking van een operatie die het systeem in een andere staat brengt. Side effects zijn meestal ongewenst omdat het de complexiteit van een operatie verhoogd.
Software engineering — Product quality	Een standaard waarmee de kwaliteit van software gekwantificeerd kan worden.
Source code	De broncode van een programma in de taal waar het oorspronkelijk in is geschreven.
Sprint backlog	Een lijst met sprints, waar bij elke sprint is aangegeven welke items uit de product backlog in die sprint aangepakt worden.

Stateless	Zonder state; er wordt niets bijgehouden nadat de communicatie is afgelopen. Elke vorm van communicatie staat op zich en berichten kunnen niet betrouwen op eerdere berichten.
Static (methode)	Een methode die niet op een instantie van een klasse, maar op de klasse in zijn algemeenheid van toepassing is. Deze functies kunnen dan ook aangeroepen worden zonder dat het via een instantie van de klasse moet.
Tight coupling	Wanneer een groep modules zeer afhankelijk is van elkaar, een kenmerk van tight coupling is dat een verandering in één klasse (nadelige, overwachtse) gevolgen heeft voor andere modules.
Undefined behaviour	Resultaat van een operatie in een programma waarvan de uitkomst niet wordt gespecificeerd in de standaard.
User story	Een of meer zinnen die een stuk gewenste functionaliteit van een programma uitdrukken.
Web application framework	Een software framework dat ontworpen is om het ontwikkelen van dynamische web applicaties te ondersteunen.
Webform	Een serie velden op een web pagina waar waarden ingevuld worden die naar de server worden opgestuurd.

18 BIBLIOGRAFIE

Albahari en Albahari. *C# in a nuthsell*. O'Reilly, 2002.

Cohn, Mike. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, 2004.

Eisley, Noel, Li-Shiuan Peh en Li Shang. „In-Network Cache Coherence." (2007).

Elling, de Jong en Swankhuisen. *Rapportagetechniek*. Noordhoff, 2004.

Gamma, et al. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

Kleppe en Warmer. *Praktisch UML 4e editie*. Pearson Education, 2007.

Lhotka. *Using CSLA 4, ASP .NET MVC*. 2011.

Lynch, Nancy en Seth Gilbert. „Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." *ACM SIGACT News* (2002): 51-59.

Nah, Fiona Fui-Hoon. „A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait?" *Behavior and Information Technology* (2004): 153-163.

18.1 REFERENTIE MATERIAAL

Website	Opmerking
http://en.wikipedia.org/	Algemene informatie over technieken, methoden en technologieën.
http://msdn.microsoft.com/	Essentiële informatie over verschillende Microsoft tools, frameworks, programmeertalen en meer.
http://www.lhotka.net/cslanet/	Informatie en forum over CSLA
http://www.uml-diagrams.org/	Referentie materiaal over UML diagrammen, complete met voorbeelden en uitleg.

19 BIJLAGEN

Niet alle bijlagen zijn afgedrukt omdat dit in sommige gevallen onhandig zou zijn door de hoeveelheid papier die hiervoor nodig is. Deze bijlagen zijn digitaal bijgevoegd. De sprint en product backlog is ook alleen digitaal bijgevoegd omdat dit slechts een andere weergave is van de requirements uit het analyse document en de user stories die in elke sprint zijn aangegeven.

Hieronder staat per categorie aangegeven welke bijlagen dit document heeft.

Documenten m.b.t. informatievoorziening omtrent het project en de voortgang ervan tegenover de school:

I. Afstudeerplan
II. Competentie verantwoording
III. Tussentijdse voortgangscontroles (bespreking concept en tussentijds assessment)

Documenten over de projectinrichting, projectspecificatie en aanpak:

IV. Requirements analyse
V. Product backlog [digitaal bijgevoegd]

Diagrammen en andere producten die voortkomen uit het ontwerp:

VI. UML Diagrammen [digitaal bijgevoegd]

Op te leveren producten:

VII. Source code [digitaal bijgevoegd]
VIII. Documentatie [digitaal bijgevoegd]
IX. Fabric bevindingenrapport