Aenova B.V.

OVInfo

De dood van een god class

Thom Schanzleh 10067523
3-26-2016

DE HAAGSE HOGESCHOOL

aenova®

Voorwoord

Het lijkt wel of iedereen die ik de laatste tijd spreek als eerste genoemd wil worden in het voorwoord van mijn verslag. Blijkbaar is onsterfelijkheid verkrijgen door genoemd te worden in het verslag van één of andere student een voorrecht. Hoe het ook zij, er zijn zeker mensen geweest die me veel hebben geholpen met het schrijven van dit verslag, en die wil ik toch bedanken.

Ten eerste bedank ik Edwin Delsman, die de oorspronkelijke code heeft geschreven waaraan ik tijdens dit project heb gesleuteld en die mij daardoor inzicht kon geven in de redenatie achter veel van de functionaliteiten van OVInfo en die mij gedurende het project heeft voorzien van feedback.

Vervolgens wil ik mijn andere collega's bij Aenova bedanken voor hun inzichten en hulp bij het uitvoeren van dit project, en voor de prettige werksfeer waarin ik dagelijks werk. Wat? Een slijmbal? Ik? Hoe kom je erbij!

Uiteraard mag ook een bedankje voor mijn begeleiders op de Haagse Hogeschool niet ontbreken.

Als laatste (en je mag zelf weten of je dat de place of honor of een achterafje vindt…) wil ik Dylan de Wolff bedanken, omdat hij de tijd heeft genomen dit verslag door te lezen en er feedback op te geven.

Den Haag, 26-3-2016

DE HAAGSE HOGESCHOOL

aenova®

Inhoudsopgave

Voorwoord 1

1. Inleiding 5

1.1 Over de opbouw van dit verslag 5

2. Over Aenova B.V. 6

2.1 Belangrijkste werkzaamheden 6

2.2 Bedrijfscultuur en organisatie 6

3. Opdracht 8

3.1 Huidige situatie 8

3.2 Probleemstelling 8

3.3 Werkzaamheden 8

4. Methodes en technieken 9

4.1 Programmeertaal 9

4.2 Ontwikkelmethode 9

4.3 Frameworks 9

4.4 Tools 10

5. Over TimEnterprise 11

5.1 Het weekbriefje 11

5.2 De dagplanning 12

5.3 De budgetmatrix 13

6. Voorbereidend werk: Leren kennen van de codebase 18

6.1 Doel van de sprint 18

6.2 Werkzaamheden 18

6.3 Evaluatie 19

7. Sprint 1: Vastleggen dependencies en verantwoordelijkheden 20

7.1 Doel van de sprint 20

7.2 Werkzaamheden 20

7.4 Evaluatie 24

8. Sprint 2: Main Loop 25

8.1 Doel van de sprint 25

8.2 Werkzaamheden 25

8.3 Evaluatie 28

9. Sprint 3: Adapter 30
9.1 Doel van de sprint 30
9.2 Werkzaamheden 30
9.3 Evaluatie 33
10. Sprint 4: Serialisatie van OVInfo 35
10.1 Doel van de sprint 35
10.2 Werkzaamheden 35
10.3 Evaluatie 37
11. Sprint 5: REST service en repareren serialisatie 38
11.1 Doel van de sprint 38
11.2 Werkzaamheden 38
11.3 Evaluatie 43
12. Sprint 6: Integratietests schrijven 44
12.1 Doel van de sprint 44
12.2 Werkzaamheden 44
12.3 Evaluatie 47
13. Evaluatie 49
13.1 Terugblik 49
13.2 Geleerde lessen 49
14. Beroepstaken 50
14.1 Ontwerpen systeemdeel 50
14.2 Bouwen systeemdeel 50
14.3 Initiëren en plannen van het testproces 50
14.4 Uitvoeren van en rapporteren over het testproces 50
15. Glossary 51
Bijlagen 54
Opdrachtomschrijving 55
Bedrijf 55
Probleemstelling 55
Doelstelling opdracht 55
Scope van het project 56
Binnen de scope 56

Buiten de scope 56
Risicofactoren 57
Verlies van oorspronkelijke functionaliteit 57
Projectorganisatie 58
Programmeertaal 58
Ontwikkelmethode 58
Mijlpaalproducten 59
Sequence diagram buildMultipleRoots 74
Class diagram 75

1. Inleiding

God class. Bij het horen van die woorden zullen de meeste programmeurs huiveren. Toch komen dit soort objecten, die veel te veel verantwoordelijkheden hebben, nog vaak voor. Met name in *legacy code* zie je vaak dergelijke classes. In dit verslag beschrijf ik de werkzaamheden die ik de afgelopen 17 weken heb uitgevoerd aan zo'n god class, bij Aenova B.V. in Delft. Het doel van dit verslag is het voor mijn begeleiders en gecommitteerde inzichtelijk maken van de werkwijze die ik heb gevolgd en de beslissingen die ik heb genomen om dit project uit te voeren. Verder zal ik de problemen bespreken waar ik tijdens dit project tegenaan ben gelopen en hoe ik deze heb opgelost.

1.1 Over de opbouw van dit verslag

Dit verslag is verdeeld in vier delen. In het eerste deel, bestaande uit hoofdstuk 1 tot en met 5, beschrijf ik achtereenvolgens het bedrijf, de opdracht, de gebruikte methodieken en tools, en zal ik informatie geven over TimEnterprise, het programma waarbinnen dit project plaatsvindt.

In het tweede deel van het verslag, bestaande uit hoofdstuk 6 tot en met 12, beschrijf ik de sprints (zie Hoofdstuk 4: Methodieken en Tools) die tijdens dit project zijn uitgevoerd. Om alles overzichtelijk te houden volgt elk van deze hoofdstukken de volgende opbouw:

- Doel van de sprint
- Werkzaamheden
- Evaluatie

Het derde deel van het verslag, bestaande uit hoofdstuk 13 tot en met 15, vormt de afronding, waarin ik het project als geheel evalueer en de beroepstaken bespreek. Ook bevindt zich in dit deel een glossary waarin gebruikte termen worden verklaard. Als in het verslag een woord *cursief* is gedrukt, is dit opgenomen in de glossary.

Het vierde en laatste deel van het verslag bestaat uit bijlagen. Hierin zijn onder andere het Plan van Aanpak en ontwerp opgenomen.

2. Over Aenova B.V.

In dit hoofdstuk beschrijf ik het bedrijf waar ik de opdracht heb uitgevoerd. Eerst zal ik het hebben over de belangrijkste werkzaamheden van het bedrijf. Daarna zal ik ingaan op de bedrijfscultuur en organisatie. Hierbij zal ik ook vermelden op welke afdeling ik zelf werkzaam ben.

2.1 Belangrijkste werkzaamheden

Aenova is een bedrijf dat zich richt op het samenbrengen van de manager en de werknemer. De laatste jaren is flexwerken steeds meer in opkomst, waardoor medewerkers niet altijd even gemakkelijk hun uren kunnen verantwoorden.

Het antwoord dat Aenova hierop biedt is TimEnterprise. Dit softwarepakket stelt werknemers in staat uren te verantwoorden vanaf elke locatie en geeft zo managers inzicht in de voortgang van het project zonder dat er extensief micro-management nodig is. Naast het schrijven van uren is TimEnterprise ook geschikt om te koppelen met veelgebruikte backoffice systemen zodat verlofregistraties uit bijvoorbeeld ADP eenvoudig kunnen worden overgenomen in Tim, zodat alle informatie altijd bij de hand is.

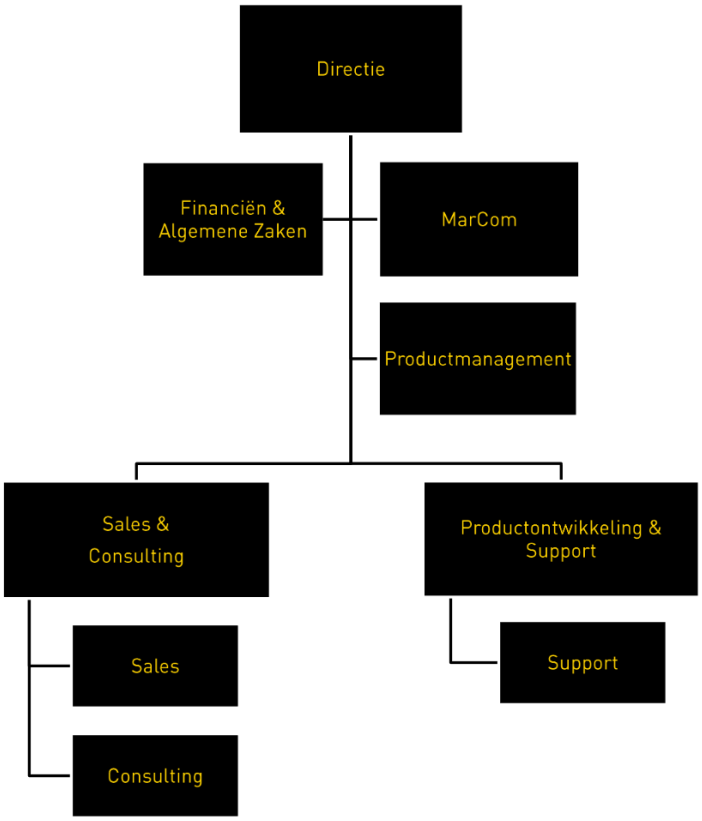
TimEnterprise wordt in-house ontwikkeld, maar Aenova biedt nog een ander product aan om flexwerken makkelijker te maken: TotalMobile. TotalMobile heeft als doel bedrijfsprocessen te automatiseren door gebruikers digitaal toegang te geven tot hun werkomgeving. Ze kunnen snel achtergrondinformatie opvragen, maar ook zelf nieuwe informatie vastleggen en deze indien nodig aanvullen met foto’s, handtekeningen, of andere zaken. Net als TimEnterprise biedt TotalMobile volledige ondersteuning voor alle grote backoffice systemen, zodat de tijd die nodig is om de administratie op orde te krijgen aanzienlijk daalt.

2.2 Bedrijfscultuur en organisatie

Aenova heeft 19 medewerkers en daardoor heerst er een informele werksfeer. Omdat de afdelingen binnen het bedrijf veel met elkaar communiceren is kent iedereen elkaar goed en is de drempel laag om vragen te stellen of gewoon een praatje te maken.

Dankzij de informele sfeer is het ook mogelijk eerlijk te zijn over verwachtingen tijdens een sprint. Als er bijvoorbeeld een beslissing wordt genomen vanuit de directie die om wat voor reden dan ook niet haalbaar is, wordt het van de werknemers verwacht hier iets over te zeggen en uit te leggen wat het probleem is. Communicatie werkt altijd twee kanten op, waardoor developers goed inzicht hebben in het werk dat Productmanagement van hen verwacht, maar andersom Productmanagement ook weet welke problemen er spelen onder de developers en altijd up to date zijn met de voortgang.

Figuur 1 op de volgende pagina geeft het organogram van de organisatie weer. Zelf ben ik werkzaam op de afdeling Productontwikkeling.



Figuur 1: Organogram Aenova B.V.

3. Opdracht

In dit hoofdstuk beschrijf ik de opdracht waaraan ik heb gewerkt. Eerst beschrijf ik de huidige situatie en het probleem daarmee. Vervolgens beschrijf ik de inhoud van de opdracht. Om het hoofdstuk af te sluiten beschrijf ik de gewenste eindsituatie.

3.1 Huidige situatie

Het belangrijkste product van Aenova heet TimEnterprise. Dit is in de eerste plaats een urenregistratiesysteem, maar door de jaren heen is het uitgegroeid tot een systeem waarin niet alleen uren geschreven kunnen worden, maar ook vakanties worden aangevraagd en ziekmeldingen kunnen worden geregistreerd. Verder kan TimEnterprise koppelen met verschillende HRM systemen, om op die manier gegevens over werknemers te kunnen registreren. Het systeem werd al vroeg in de jaren 90 ontwikkeld, maar pas overgezet naar Java in 1999. Meer over TimEnterprise vindt u in Hoofdstuk 5.

Sinds die tijd zijn er veel features bij gekomen en is de manier van programmeren sterk veranderd. Dit heeft tot gevolg dat veel van de *core business logic* binnen Tim geschreven is op een manier die niet voldoet aan moderne standaarden en niet onderhoudbaar is. Zo is het in veel gevallen onmogelijk om *unittests* uit te voeren omdat veel methods bestaan uit honderden regels code, waar vaak over de jaren heen specifieke condities zijn bijgeschreven om te zorgen dat gespecialiseerde code goed werkt.

3.2 Probleemstelling

Zoals in de vorige paragraaf al werd aangegeven, bevinden zich in TimEnterprise oude classes die niet binnen een moderne softwareomgeving passen. Met het oog op de toekomst richt Aenova zich steeds meer op het aanbieden van TimEnterprise als een webservice. Op dit moment wordt dat tegengehouden door een class, genaamd OVInfo, die zich bezighoudt met het uitzoeken van rechten en selecties voor onder andere overzichten, maar die ook intern wordt gebruikt om bijvoorbeeld weekbriefjes en dagplanningen op te zetten.

Om de gewenste informatie te kunnen tonen worden er door deze class soms vragen gesteld aan de gebruiker door middel van zogenaamde dialogables. Deze dialogables blokkeren de server totdat de gebruiker input heeft gegeven. Dit is een model dat in een webomgeving niet bruikbaar is, en daarom moet OVInfo herschreven worden. Het is daarbij de bedoeling dat er een geheel nieuwe implementatie komt, die ook de oude manier van werken nog ondersteunt voor de desktop client.

3.3 Werkzaamheden

Mijn opdracht is het vastleggen van de verantwoordelijkheden en dependencies van OVInfo, en gebaseerd daarop met een nieuw ontwerp te komen waarin de *separation of concerns* goed is geregeld en dit ontwerp vervolgens te bouwen. Daarnaast moet er voor de nieuwe webclient ook een *REST* service gebouwd worden. Hierbij is het van belang dat de afhankelijkheid van dialogables verdwijnt, maar dat er geen bestaande functionaliteit verloren gaat. Daarnaast is het zeer belangrijk dat het nieuwe ontwerp goed testbaar is, zodat toekomstige uitbreidingen veilig ingebouwd kunnen worden.

4. Methodes en technieken

In dit hoofdstuk beschrijf ik de methodes en technieken die ik tijdens dit project heb gebruikt. Eerst zal ik de programmeertaal beschrijven. Daarna volgt een beschrijving van de ontwikkelmethode. Vervolgens beschrijf ik de gebruikte frameworks. Ik sluit het hoofdstuk af met een beschrijving van de gebruikte tools.

4.1 Programmeertaal

De taal waarin dit project wordt uitgevoerd is Java. In 1999 werd Tim hiernaar geport vanuit C en sindsdien is de programmeertaal niet meer veranderd. De applicatie wordt gebouwd met Java 7 in plaats van 8, omdat er oude onderdelen van Tim zijn die methods gebruiken die in Java 8 niet meer bestaan.

4.2 Ontwikkelmethode

Bij het volbrengen van deze opdracht heb ik gebruik gemaakt van de ontwikkelmethode Scrum. Dit lijkt wellicht op het eerste gezicht vreemd voor een opdracht waarbij voor het grootste deel geen nieuwe functionaliteit wordt gemaakt, maar omdat ik zou meedraaien in reguliere sprints tijdens dit project en de taken die ik zou uitvoeren goed in stories verdeeld konden worden zou ik toch deze methode gebruiken.

De sprintduur was twee weken. Twee weken wordt door het bedrijf gezien als een goede lengte omdat het voldoende tijd biedt functionaliteit af te krijgen, maar ook nog kort genoeg is dat er tijdig feedback gegeven kan worden door de Product Owner.

Voorgaande aan de sprint vindt er een sprintplanning plaats waarin wordt bepaald welke stories op de sprint zouden komen. Het maximaal aantal te behalen punten wordt vastgesteld door te kijken naar vorige sprint en aanwezigheid van de developers.

Tijdens de sprint is er elke dag een daily standup, waarbij elke developer vertelt wat hij die dag heeft gedaan. Hierbij is Productmanagement ook aanwezig om eventuele vragen te kunnen stellen en op de hoogte te blijven van de voortgang. Tijdens de daily standup wordt ook gekeken naar de burndown chart.

De sprint wordt afgesloten met een retrospective, waarin wordt besproken waarom bepaalde stories niet af zijn, wat er fout ging, en wat er goed ging. Ook wordt hier al een schatting gemaakt voor het aantal punten voor de volgende sprint.

De Product Owner en Scrum Master zijn dezelfde persoon in dit geval. Beide rollen worden vervuld door Eva van der Last, directeur van Aenova.

4.3 Frameworks

Voor de REST kant van het project wordt gebruik gemaakt van het Spring Framework. Dit framework biedt eenvoudige configuratie waarmee men een class door middel van annotaties kan aanmerken als een controller, service, repository of entity. Verder zijn er annotaties beschikbaar om URLs te kunnen mappen op zowel methods als classes. Tim maakt geen gebruik van de repository en entity annotaties, omdat Spring niet vanaf het begin van de ontwikkeling is gebruikt. Vanuit de Spring services worden de classes met daarin de bestaande business logic aangeroepen.

De geschreven tests maken gebruik van het jUnit framework en Mockito, zodat unittests met bijvoorbeeld controllers niet afhankelijk zijn van het bestaan van een echte service. Meer over de werking van mocks vindt u in Hoofdstuk 12.

4.4 Tools

Bij het uitvoeren van het project heb ik gebruikt gemaakt van verschillende tools. In Tabel 1 zijn deze tools en hun gebruik opgenomen.

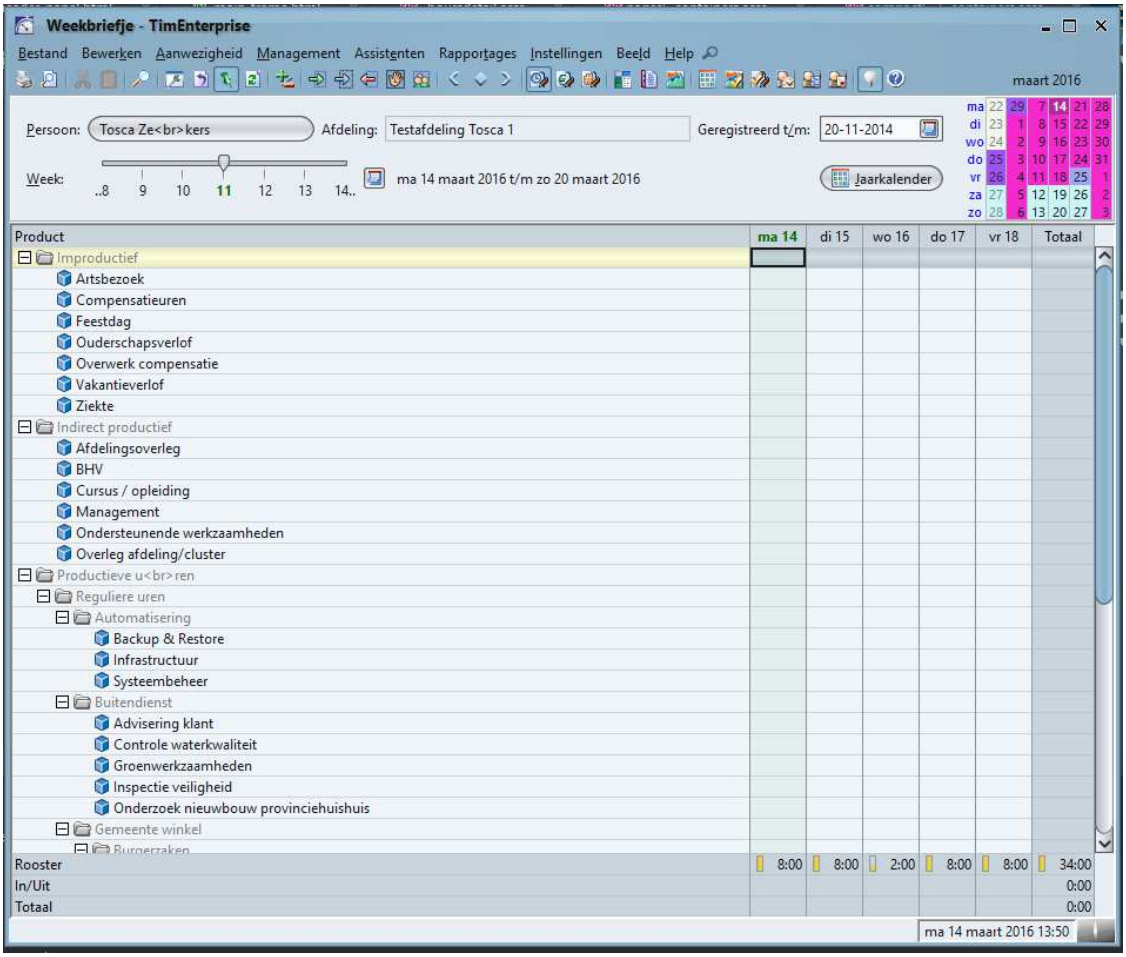
IntelliJ IDEA	De Integrated Developer Environment waarmee ik heb gewerkt. IntelliJ biedt breed support voor frameworks (waaronder Spring), versiebeheer, en build tools en is de bedrijfsstandaard bij Aenova.
Subversion	Subversion is het versiebeheersysteem dat in gebruik is bij Aenova. Er bestaan verschillende branches voor verschillende incarnaties van Tim. Er is een wens over te stappen naar Git vanwege de betere support voor branches, maar dit heeft geen hoge prioriteit.
Maven	Maven is een build tool die vooral geschikt is om externe dependencies te beheren, zodat men geen losse libraries hoeft te bewaren op het versiebeheersysteem.
Confluence	Dit is een bedrijfswiki waar informatie over bepaalde delen van het project gevonden kan worden. Ook zijn hier de coding standards en architectuurrichtlijnen vastgelegd.
JIRA	Dit is het systeem waarin user stories worden bijgehouden. Elke user story

Tabel 1: Gebruikte tools

5. Over TimEnterprise

In dit hoofdstuk ga ik dieper in op het hoofdproduct van Aenova, TimEnterprise. Hiermee wil ik u inzicht geven in de omgeving waarin het project wordt uitgevoerd, om zo een beter beeld te kunnen schetsen van de omvang ervan. Omdat Tim een zeer uitgebreide applicatie is zal ik niet elke functionaliteit in detail bespreken. Hieronder volgt een overzicht van de belangrijkste functionaliteiten van Tim die te maken hebben met OVInfo.

5.1 Het weekbriefje



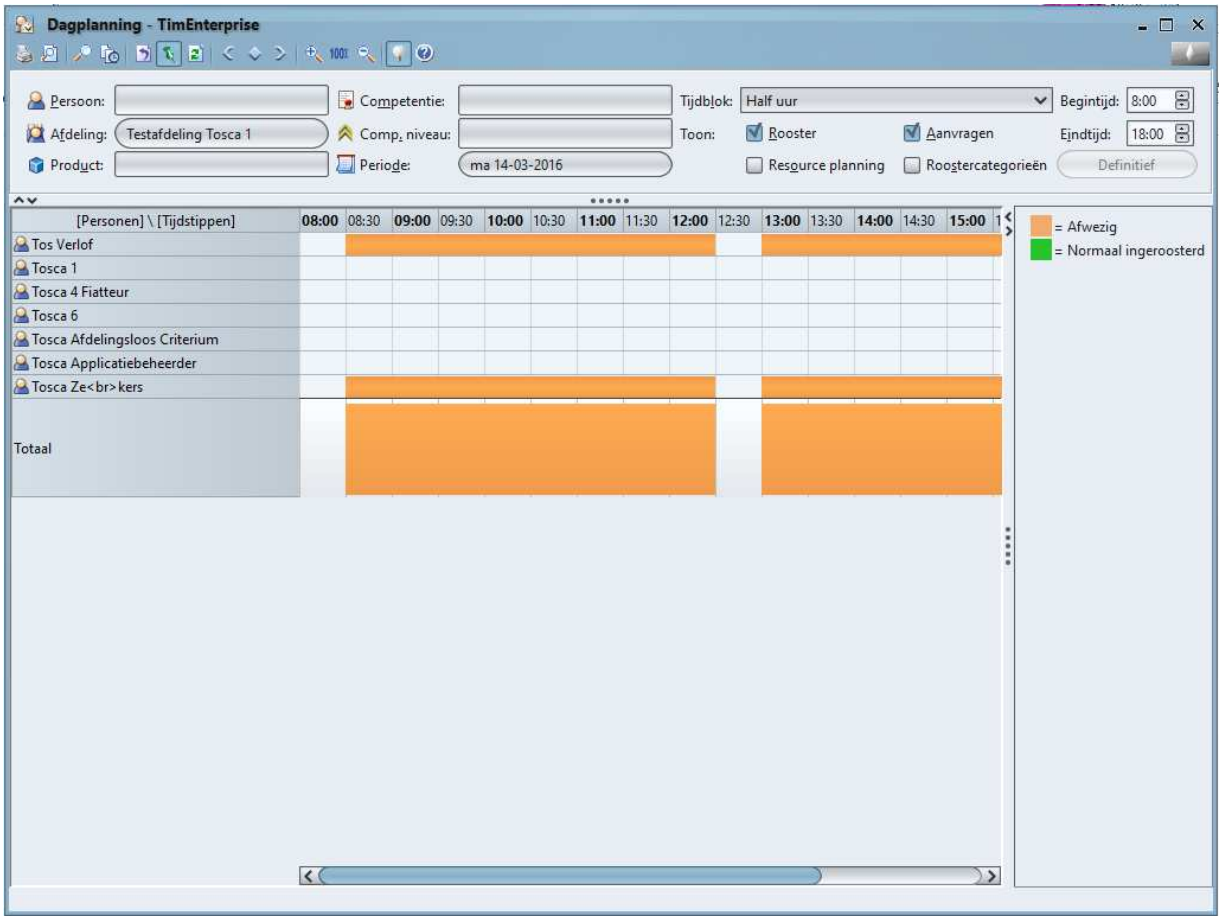
Figuur 2: Het weekbriefje in TimEnterprise

Figuur 2 toont het basisscherm van TimEnterprise, het weekbriefje. In dit scherm kan men uren schrijven op de producten waarop men recht heeft. Onderaan het scherm ziet men het aantal ingeroosterde uren, het aantal uren dat men is ingeklokt en het totaal aantal geregistreerde uren. Dit is één van de schermen die onder water gebruik maakt van OVInfo, ondanks het feit dat hier geen gebruikersinput voor vereist is.

Boven de tabel ziet men een slider waarmee de week kan worden gekozen, met daarnaast een optie om snel een datum te kiezen. Boven deze slider bevindt zich de naam van de ingelogde persoon met

daarnaast de afdeling waarop deze persoon werkzaam is. Als men de rechten heeft om de weekbriefjes van andere werknemers in te zien is de box waarin de naam van de werknemer wordt weergegeven ovaal in plaats van rechthoekig. Dit is in Figuur 2 het geval. Door op de naam te klikken krijgt men een lijst met werknemers te zien, van wie men de weekbriefjes kan bekijken. Dit is onder andere van toepassing op afdelingsleiders.

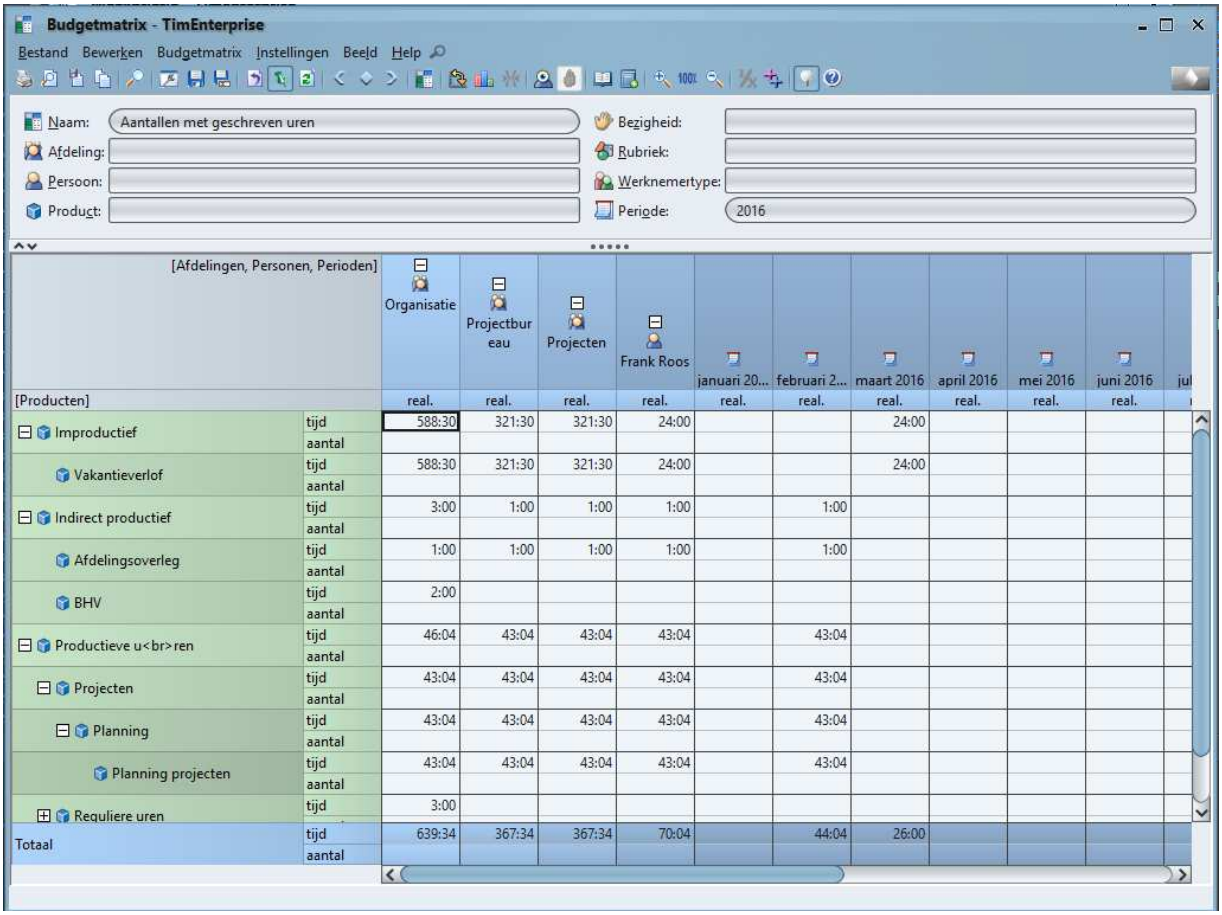
5.2 De dagplanning



Figuur 3: Dagplanningscherm

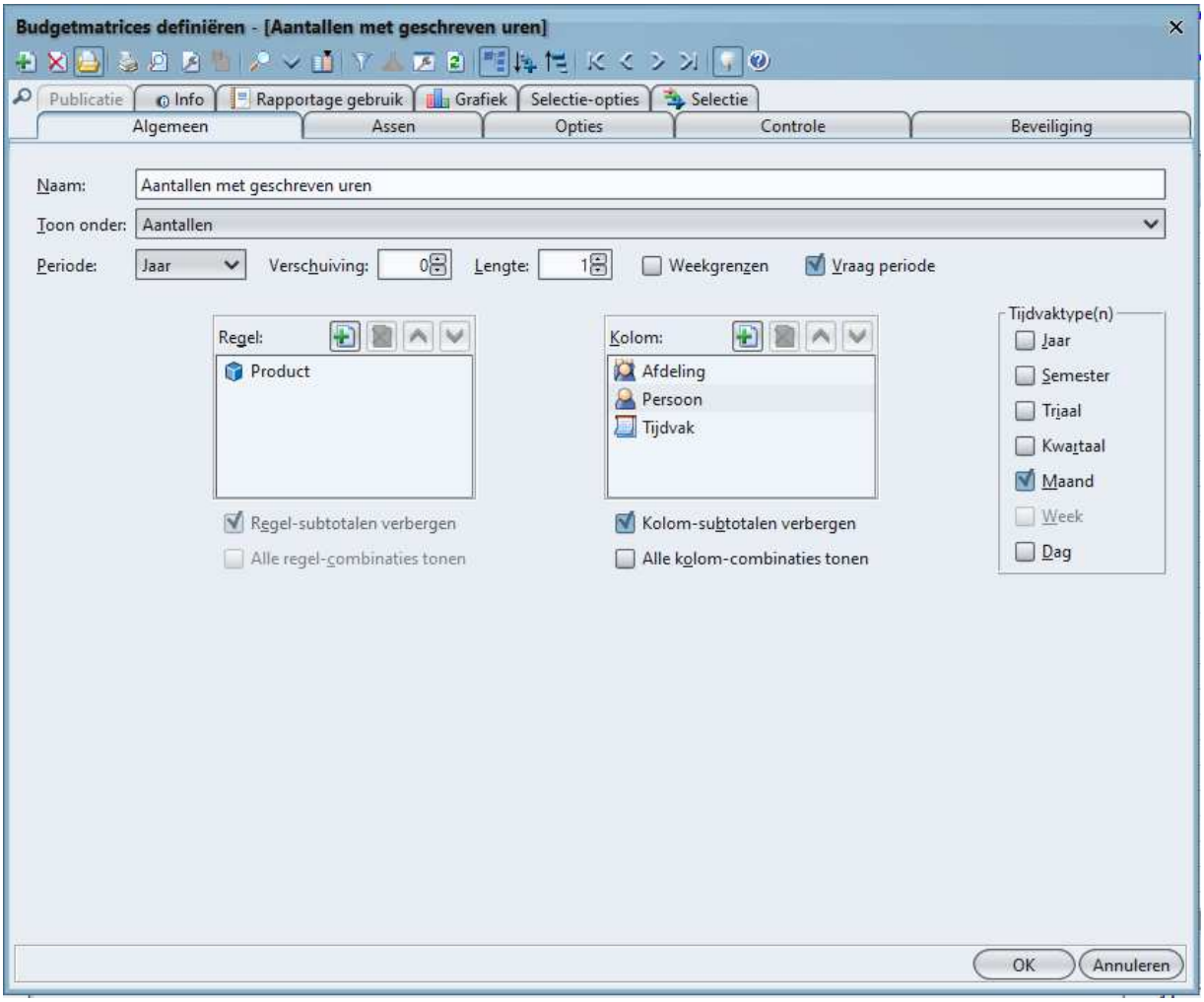
Figuur 3 toont het tweede scherm waarbij gebruik wordt gemaakt van OVInfo zonder hier eerst input voor nodig te hebben, het dagplanningscherm. In dit scherm ziet men als normale medewerker zonder speciale rechten de eigen planning van de huidige week. In Figuur 3 ziet men dat in plaats daarvan de planning van de gehele afdeling over een enkele dag wordt getoond. De knoppen bovenin het scherm laten verfijningen hierop toe, om bijvoorbeeld de planning van een enkel afdelingslid over een hele maand te bekijken.

5.3 De budgetmatrix



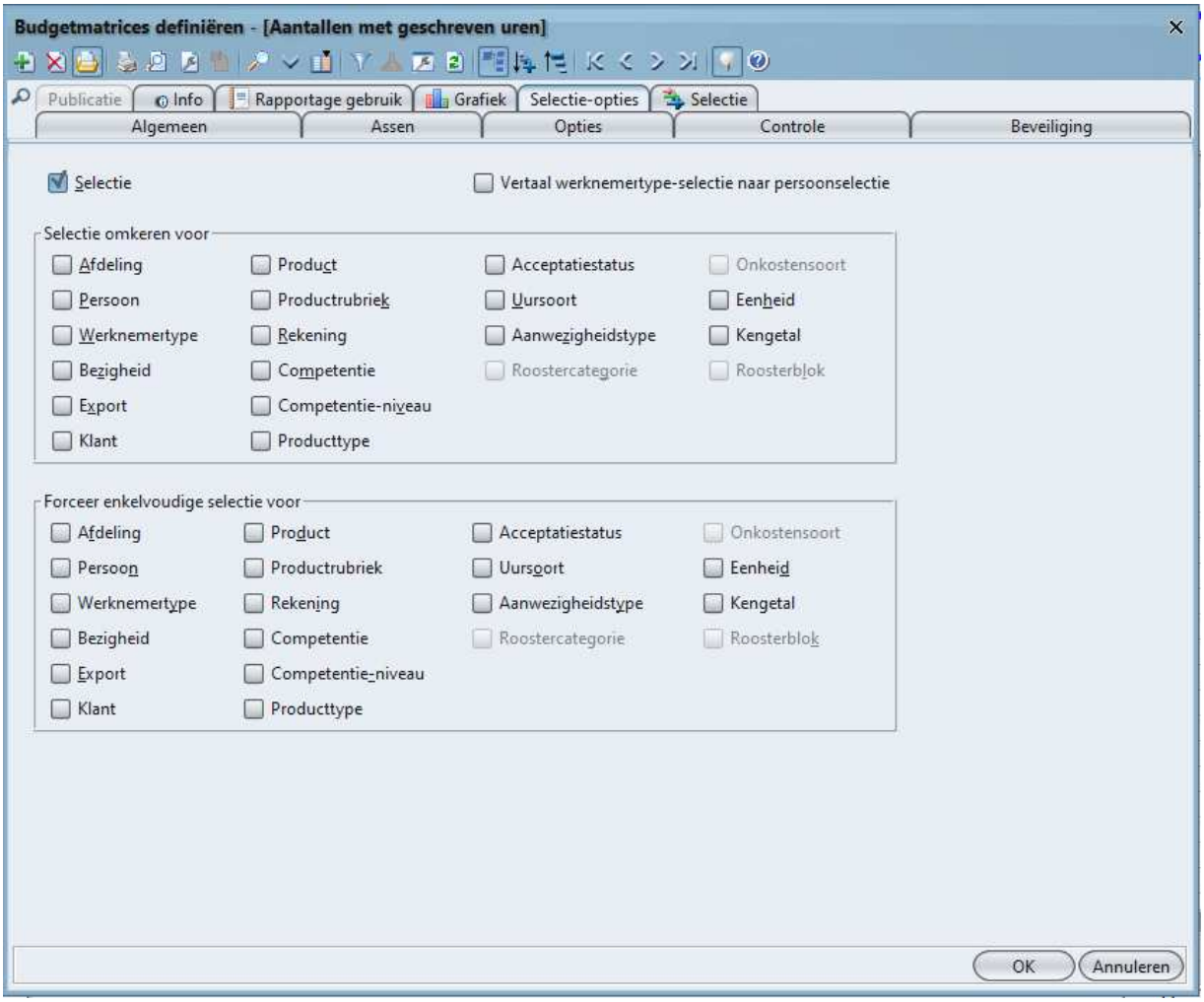
Figuur 4: Een standaard budgetmatrix

Figuur 4 toont een budgetmatrix waarin weergegeven wordt op welk product er in welk tijdvak door welke persoon is geschreven. Hierbij ziet men dat er een hiërarchische structuur aanwezig is die men open of dicht kan klappen om op die manier een gedetailleerder of minder gedetailleerd beeld te krijgen. Net als bij de dagplanning kan er met de knoppen bovenin het scherm worden aangegeven van wie (of van welke product, of welke afdeling) men informatie wil zien. Om een matrix als deze te kunnen opbouwen moet daarvoor eerst een definitie zijn vastgesteld. Het vaststellen van overzichtdefinities kan niet worden gedaan door gewone werknemers, maar omdat de definitie van groot belang is voor het project zal ik deze nader bespreken.



Figuur 5: Hoofdscherm budgetmatrix definiëren

Figuur 5 toont het basisscherm budgetmatrix definiëren. In dit scherm bepaalt men welke waardes er op de assen van de matrix komen. De volgorde waarin de opties staan bepalen de hiërarchie bij het tonen van de matrix. In dit geval worden dus de afdelingen getoond, met daaronder de persoon, en daaronder het tijdvak. De grootte van het tijdvak wordt bepaald door de lijst met checkboxes. Als er geen tijdvak op één van de assen voorkomt is zijn deze checkboxes wel zichtbaar, maar niet selecteerbaar. Voor OVInfo is uit dit scherm eigenlijk alleen de checkbox 'Vraag periode' van belang, omdat deze bepaalt of er al dan niet een vraag aan de gebruiker wordt gesteld. De assen worden wel opgeslagen in het overzichtobject dat onder andere door OVInfo wordt gebruikt, maar ze worden niet in de workflow van die class gebruikt.



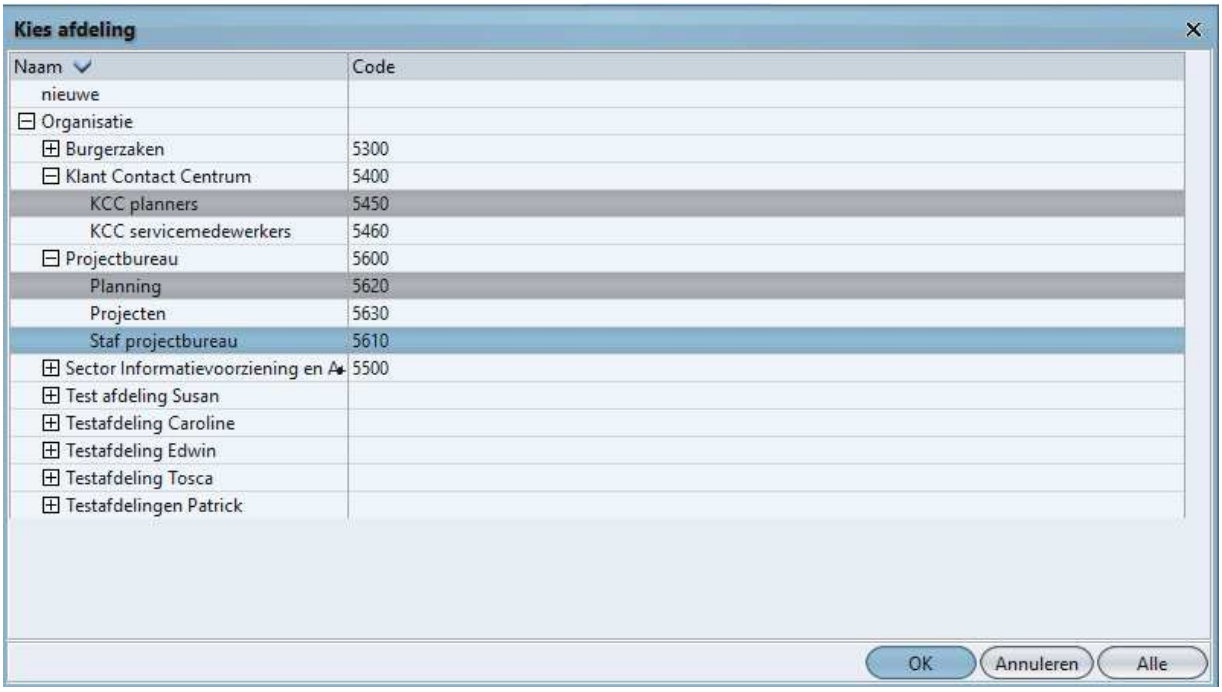
Figuur 6: Selectie-opties van een budgetmatrix

Figuur 6 geeft de selectie-opties van een budgetmatrix weer. Als het vinkje ‘Selectie’ aanstaat wordt dit scherm relevant voor OVInfo, omdat het aangeeft dat er mogelijk input van de gebruiker nodig zal zijn. De groep vinkjes met de titel ‘Selectie omkeren voor’ is bedoeld om een inverse selectie te kunnen maken. Als men hier bijvoorbeeld ‘Afdeling’ aanvinkt, en men kiest vervolgens de afdeling ‘Projecten’ in het dialoogvenster (zie verderop in dit hoofdstuk), dan krijgt men een budgetmatrix van alle afdelingen behalve Projecten.

De tweede groep vinkjes bepaalt dat er van een bepaalde categorie slechts één waarde geselecteerd mag worden. Normaal gesproken zou men bijvoorbeeld een matrix over meerdere afdelingen kunnen maken, maar met dit vinkje aan zou dit niet mogelijk zijn.

Figuur 7: Selectiekeuze budgetmatrix

In Figuur 7 ziet men de daadwerkelijke categoriëen die gevraagd zullen worden. In deze budgetmatrix zal er bijvoorbeeld gevraagd worden om een persoon en een afdeling. De linker kolom 'waarde' kan naast 'Vraag' ook de opties 'Vaste waarde' of 'Huidige' hebben. Bij de optie 'Vaste waarde' geeft men in de overzichtdefinitie zelf al aan welke waarde er gebruikt dient te worden, terwijl 'Huidige' de waarde kiest die van toepassing is op de persoon die het overzicht opent. In het geval van deze matrix zouden dus de persoon zelf en diens afdeling worden gekozen.



Figuur 8: Dialoog van budgetmatrix met meervoudige selectie

Figuur 8 (Dit is het laatste figuur van het hoofdstuk, dat beloof ik) toont een standaard dialoogvenster dat verschijnt wanneer men een keuze moet maken. Zoals op de afbeelding te zien is zijn 'KCC planners', 'Planning', en 'Staf projectbureau' geselecteerd. Het tonen van deze dialoog en het verwerken van de input van de gebruiker is de verantwoordelijkheid van OVInfo. De manier waarop OVInfo deze dialoog toont vormde de voornaamste aanleiding voor dit project, waar ik in de komende hoofdstukken dieper op in zal gaan naarmate de sprints vorderen.

6. Voorbereidend werk: Leren kennen van de codebase

In dit hoofdstuk worden de voorbereidende werkzaamheden voor het project besproken. Omdat deze periode plaatsvond tijdens een normale sprint hou ik in dit hoofdstuk de opbouw aan die ik ook voor de rest van de sprints zal gebruiken ondanks het feit dat de werkzaamheden een stuk korter zijn.

6.1 Doel van de sprint

Het doel van deze sprint was het kennismaken met de code van OVInfo en leren hoe deze code in de applicatie wordt gebruikt.

6.2 Werkzaamheden

Meestal is het zo dat een afstudeerder helemaal nieuw is bij het beginnen van een afstudeertraject. In mijn geval was dat anders, want ik werkte op het moment dat mijn opdracht begon al bijna een half jaar bij Aenova. Daarom hoefde ik de werkwijze van het bedrijf en mijn collega's niet meer te leren kennen.

Eén van de eerste dingen die in dit project moesten gebeuren was vastleggen wat precies de scope van de opdracht zou zijn omdat OVInfo op veel plekken in Tim werd gebruikt. Om dit te kunnen doen zou ik een architectuuroverleg bijwonen. Normaal gesproken vindt dit overleg plaats tussen de directie, het hoofd Productmanagement Edwin Delsman, en senior developer Patrick Roosendaal. Dit keer zou ik het overleg ook bijwonen.

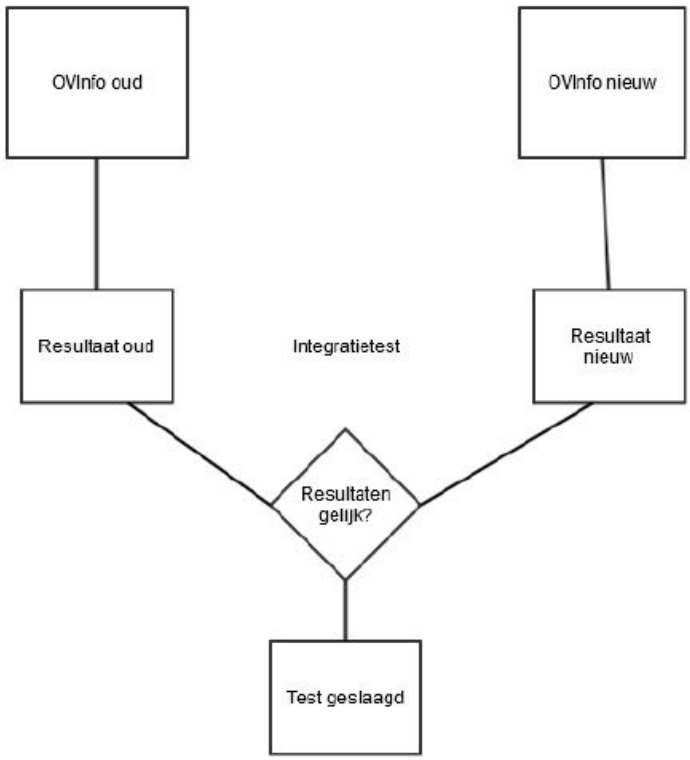
Tijdens het overleg werd mij uitgelegd wat specifiek het probleem was van OVInfo. Omdat ik dat al in Hoofdstuk 3 heb beschreven zal ik dat hier niet nogmaals doen. Wel belangrijk voor deze sprint is de manier waarop we de situatie zouden kunnen aanpakken.

Omdat OVInfo in zijn huidige vorm niet unittestbaar was en de integratietest te klein was om als goede basis te dienen zou het lastig zijn om vanaf de grond een nieuwe implementatie voor OVInfo te bouwen. In de oude code zaten genoeg *quick fixes* en *hacks* om met zekerheid te kunnen zeggen dat er *side effects* zouden zijn en omdat daar geen documentatie van bestond zou een goede werking van een eventuele compleet nieuwe OVInfo niet gegarandeerd kunnen worden. Aangezien het hier gaat om core business logic was dit geen acceptabele oplossing.

Een tweede mogelijkheid was het bouwen van een zogenaamde parallelle implementatie. Hierbij zou de oude OVInfo blijven werken terwijl deze stap voor stap werd vervangen door een nieuwe versie. Dan moest er natuurlijk wel gegarandeerd kunnen worden dat de nieuwe versie dezelfde resultaten zou opleveren als de oude versie. Om dit te kunnen doen zou ik een integratietest schrijven die zowel de oude als de nieuwe OVInfo zou draaien. Figuur 9 op volgende pagina geeft dit schematisch weer.

Na het architectuuroverleg heb ik samen met Edwin en Patrick de code van OVInfo doorgenomen. Hierbij legde Edwin uit wat elke method ongeveer deed en wezen hij en Patrick plekken in de code aan waar er veel fixes waren ingebouwd. Mijn taak voor de eerste echte sprint van het project zou het vastleggen van de dependencies en verantwoordelijkheden van OVInfo worden, om op die manier een beeld te kunnen schetsen van hoe de nieuwe situatie eruit zou gaan zien.

Voordat het zover was moest ik echter eerst de bevindingen van het architectuuroverleg vastleggen. Dit document bevindt zich in de bijlagen. Ook het PVA, dat ik in deze sprint heb geschreven, kunt u vinden in de bijlagen.



Figuur 9: Schematische weergave parallelle implementatie

6.3 Evaluatie

Hoewel er nog niet veel werkzaamheden waren uitgevoerd in deze sprint, ben ik toch tevreden met de manier waarop deze is verlopen. Tijdens het architectuuroverleg had ik het idee dat ik goed begreep wat de problemen met OVInfo waren en wat de wenselijke situatie voor de toekomst zou zijn. De reacties die ik van Edwin en Patrick kreeg op het document dat ik opstelde na het overleg droegen bij aan mijn zelfvertrouwen, waardoor ik met een positieve instelling de rest van het project inging.

7. Sprint 1: Vastleggen dependencies en verantwoordelijkheden

In dit hoofdstuk beschrijf ik de werkzaamheden die ik heb uitgevoerd tijdens de eerste echte sprint van het project.

7.1 Doel van de sprint

Het doel van deze sprint was het vastleggen van de verantwoordelijkheden die OVInfo op dat moment had, en de dependencies die de class bezat. Aan de hand van deze informatie zou ik daarna een ontwerp moeten opstellen voor de nieuwe situatie.

7.2 Werkzaamheden

In het vorige hoofdstuk beschreef ik mijn eerste kennismaking met OVInfo. Maar waar die kennismaking nog plaatsvond in het bijzijn van Edwin, was het nu tijd om zelf met de code aan de slag te gaan. Hoewel ik op dit punt niet meer geheel onbekend was met Tim, was dit een onderdeel waar ik nog nooit mee had gewerkt. Dit was core business logic die over een tijdsspanne van vijftien jaar was gebouwd. Eén van de methods, buildMultipleRoots (die in detail zal terugkomen in het volgende hoofdstuk) telde meer dan 300 regels code. De eerste keer dat ik zelf de class opende en de code bekeek, vroeg ik me even af waar ik aan was begonnen. Hoe moest ik ooit deze brij, die vaak onduidelijke namen zoals ‘fmd’ en ‘pfld’ bevatte, herstructureren tot iets dat coherent genoemd kon worden?

Omdat alles binnen OVInfo met elkaar in verbinding leek te staan, besloot ik te beginnen met het uitzoeken van de huidige verantwoordelijkheden van OVInfo, naast de hoofdverantwoordelijkheid. Op dat moment ging ik er nog vanuit dat dit het opbouwen van queries voor overzichten op basis van gebruikersinput was. Later zou blijken dat dit niet de correcte interpretatie was. Meer daarover later in dit hoofdstuk.

In eerste instantie wilde ik de verantwoordelijkheden bepalen door te kijken wat er binnen elke method gebeurde. Ik had dit idee omdat er zoveel *monster methods* waren dat ik niet kon geloven dat elke method een enkele functionaliteit had. Het doornemen van alle code nam veel tijd in beslag, maar ik slaagde er uiteindelijk in de volgende verantwoordelijkheden vast te stellen:

- Ophalen van competenties.
- Ophalen van de selectielijst. (Het bouwen hiervan is de werkelijke hoofdfunctionaliteit van OVInfo)
- Bepalen of een bepaald type object (OIType) geselecteerd is.
- Bepalen welke vragen er gesteld worden in schermen die geen overzicht genereren.

Ik noteerde deze bevindingen in een document (dat in de bijlagen gevonden kan worden) en plaatste dit op Confluence. Nu ik de verantwoordelijkheden van OVInfo kende (of in elk geval dacht dat ik dat deed) kon ik beginnen met het uitzoeken van dependencies. Ik begon met kijken welke attributen OVInfo had.

```
private static int COMPACTNIV  = 1;
private static int COMPACTCOMP = 2;
private static int COMPACTALL  = COMPACTCOMP|COMPACTNIV;

private boolean isnew, weekbound, personal, useStartDate, useEndDate, useAutoStartDate, weekMonthBoundFixed,
        useSpecialRoosterUurQuery, doMultipleCompetentie, periodeAxis, timeAxis;
private int bestPeriodLevel, compactMultipleCompetentie;
private PersoonObject pers;
private ASDate periodBase;
private ASDatePeriod per, forcePer;
private TimJPerspFlags rightsPerspective;
private ASPrefetchedList<OverzichtSelectieObject> ovzSelectionList;
private final Map<OIType, OVTableInfo> tableInfo;
private DialogableState rightsDlg, perspectiveDlg;  // add all DialogableStates to isOtherBusy method!!!
private final String helpItemKey;
private ASPeriodType bestPeriodType;
private List<ASPeriodType> periodeIndex;
private List<ASPeriodType> timeIndex;
private HashMap competentieCache;
private TimJPerspFlags rightsPerspectiveAnswer;
private boolean disableQuestions;

private final String resourcebundle = ServerExtStrings.class.getName();
private ResourceBundle rb = ASLocalization.getBundle( resourcebundle );
private ASPrimaryKeySet wtypePers;
```

Figuur 10: De attributen van OVInfo.

Figuur 10 geeft de attributen van OVInfo weer aan het begin van het project. Het is een grote hoeveelheid, maar hoe slecht zijn deze dependencies?

De dependency die me het eerst opviel gezien de context van het project was de DialogableState. Dit waren de vragen die tijdens het proces gesteld werden, en dit was dan ook de voornaamste dependency die verwijderd zou moeten worden.

De tweede dependency die uit de lijst sprong was het PersoonObject. Het PersoonObject binnen Tim bevat een zeer groot aantal velden, en het was zeer onwaarschijnlijk dat al deze velden noodzakelijk zouden zijn binnen OVInfo. Een usage search op dit attribuut leverde op dat slechts vier velden van het PersoonObject werden gebruikt: het id, het werknemertype, de naam van de persoon, en een boolean die bepaalt of de persoon exportbestanden mag maken. Mijn conclusie was dan ook dat de afhankelijkheid van het PersoonObject geëlimineerd zou moeten worden met het oog op toekomstige serialisatie.

De derde en laatste dependency in deze lijst die me meteen opviel was de TimJPerspFlags class. Dit is een class die wordt gebruikt om het perspectief van de budgetmatrix of het overzicht te bepalen. Dit is vooral van toepassing op applicatiebeheerders, die beschikken over globale rechten naast hun eigen afdelingsrechten. Deze class zal in het volgende hoofdstuk nog aan bod komen.

Voor de lezer zijn de classes die het prefix ‘AS’ waarschijnlijk onbekend. Dit zijn in-house gemaakte utility classes die worden gebruikt om gemakkelijk en consistent met data te kunnen omgaan. Bij een applicatie als Tim, die draait om het correct verwerken van tijden en data, zijn dergelijke classes onmisbaar. Ze worden gebruikt door heel Tim en veranderen in de regel niet. Dependencies op deze classes zijn dan ook niet slecht en hoeven niet te worden verwijderd.

Dat waren de dependencies in de attributen, maar daarmee had ik nog niet alle bestaande dependencies vastgelegd. Binnen de methods werd er nog rijkelijk gebruikgemaakt van andere objecten.

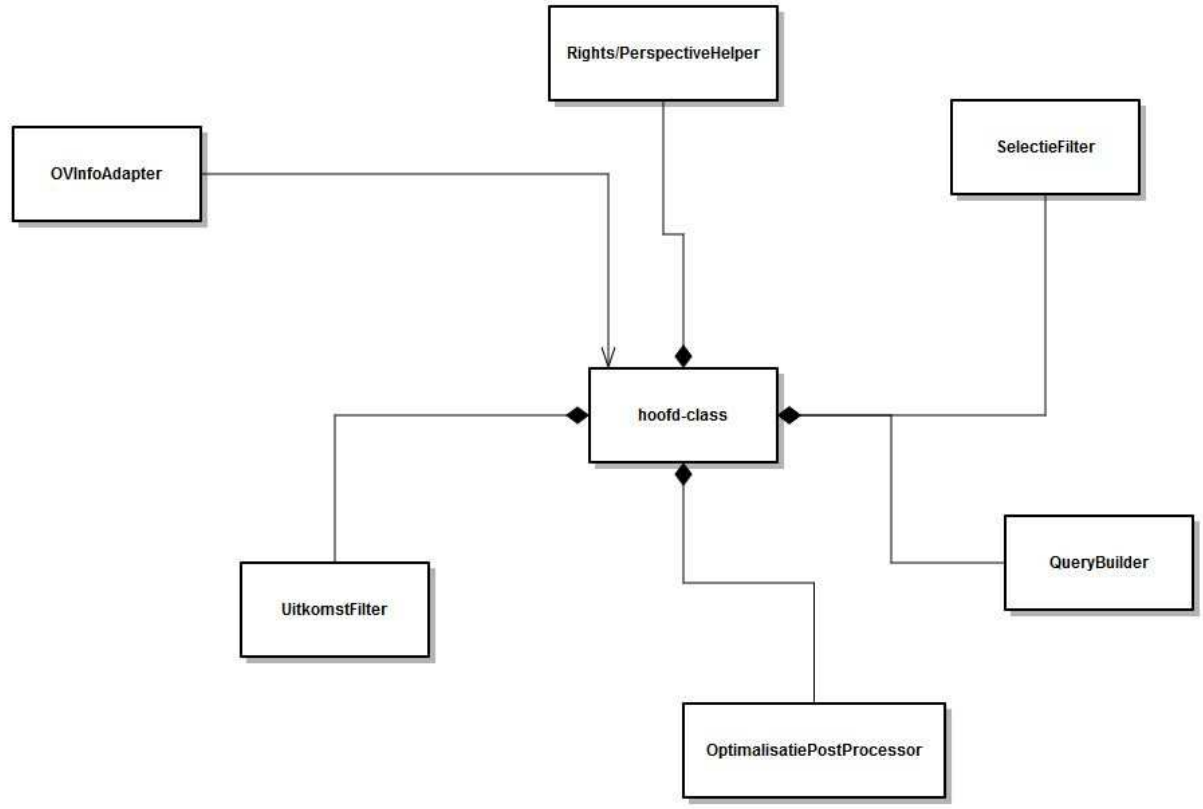
Eén van de meest voorkomende dependencies is die op het OverzichtObject. In alle 'build' methods die OVInfo rijk is moet zo'n object worden meegegeven. Voor het OverzichtObject geldt bijna hetzelfde als voor het PersoonObject, namelijk dat lang niet alle velden van dit object worden gebruikt. Het probleem met het verwijderden van deze dependency zit hem in de plek waar methods van het object worden aangeroepen. In OVInfo zelf wordt het object alleen maar doorgegeven, terwijl de werkelijke informatie eruit pas in een andere class, OVTableInfo, wordt opgehaald. In deze class wordt onder andere een method aangeroepen die niet eenvoudig te vervangen is zoals bij PersoonObject wel het geval was. Deze method haalt de waarde van een specifiek veld op. Wat dit veld is hangt af van de OVTableInfo waarin de method wordt aangeroepen. In dit stadium hoefde ik echter alleen nog maar vast te leggen dat de dependency bestond, dus hield ik me nog niet bezig met het oplossen van deze situatie. In Sprint 5 (Hoofdstuk 11) zou ik me hier weer mee bezig houden.

Op dit punt is het wellicht handig als ik wat vertel over de naamgeving van de classes, omdat de classes die ik aanmaak een Engelse naam hebben, terwijl andere classes zoals PersoonObject een Nederlandse naam hebben. De standaard die bij Aenova wordt gehanteerd is dat de zogenaamde FieldsObject classes, oftewel de classes die gepersisteerd worden naar de database, een Nederlandse naam krijgen. Alle andere classes, waaronder de classes die ik heb aangemaakt, krijgen een Engelse naam. De reden daarvoor is dat het merendeel van de klanten van Aenova Nederlands zijn. Het is in de communicatie daarom handiger om over Nederlandse namen te praten dan over Engelse.

De dependencies die ik hier heb genoemd zijn niet alle dependencies van OVInfo. Het zijn echter wel de belangrijkste dependencies voor dit project. Ik legde mijn bevindingen vast in hetzelfde document waar ik eerder de gevonden verantwoordelijkheden had opgenomen. Zoals eerder gezegd bevindt dit document zich in de bijlagen.

Nadat ik het document op Confluence had geüpload werd het nagekeken door Edwin en Patrick, om te zorgen dat de informatie erin correct was. Hoewel ze het in grote lijnen een goed document vonden waren er een aantal zaken die niet helemaal overeenkwamen met de werkelijkheid. Eén van deze dingen was de hoofdverantwoordelijkheid van OVInfo. In tegenstelling tot wat ik eerst dacht was de hoofdverantwoordelijkheid van OVInfo niet het bouwen van queries, maar het bouwen van de gebruikersselectie. In deze fase van het project had dit nog geen grote gevolgen omdat er nog geen code was geschreven, maar als dit later in het traject pas was ontdekt was het mogelijk een serieuzer probleem geweest.

Nu in grote lijnen de dependencies en verantwoordelijkheden van OVInfo bekend waren kon ik beginnen met nadenken over een mogelijke nieuwe samenstelling van classes, die de werkzaamheden van wat nu OVInfo was zouden overnemen. Figuur 11 geeft mijn eerste voorstel voor een nieuwe samenstelling weer.



Figuur 11: Eerste voorstel nieuw OVInfo.

Centraal in het diagram staat de hoofd-class. Op dit punt in het traject wist ik nog niet goed wat hier een goede naam voor zou zijn. Tijdens het uitvoeren van de werkzaamheden refereerde ik er dan ook nog steeds naar als OVInfo.

De gedachte achter de hoofd-class zoals deze hier is weergegeven is dat dit object de zeer belangrijke ‘main loop’ van OVInfo en de state van het huidige bouwproces zou bijhouden. De ‘main loop’ staat centraal in het volgende hoofdstuk.

De Rights/PerspectiveHelper heeft, zoals in het diagram ook te zien is, nog geen eenduidige naamgeving. Op dit punt in het traject wist ik namelijk nog niet zeker hoeveel van de rechtencode in OVInfo daadwerkelijk te maken had met de overzichtsrechten, en hoeveel alleen maar met het rechtenperspectief. Naast het rechtenperspectief bestaat er ook nog een persoonsperspectief, dat gebruikt kan worden door bijvoorbeeld een afdelingsleider om op die manier een overzicht te generen alsof dit werd aangemaakt door één van de medewerkers van diens afdeling. Mocht er echter code zijn die de rechten op een overzicht zelf bepaalde, dan zou dit een aparte class moeten zijn om een goede separation of concerns te kunnen waarborgen.

De SelectieFilter class heeft als verantwoordelijkheid het bepalen waar een gebruiker uit kan kiezen, gebaseerd op reeds gegeven antwoorden. Als men bijvoorbeeld bij de afdelingsvraag 'Projecten' heeft gekozen, is het niet wenselijk als bij de persoonsvraag personen van andere afdelingen kiesbaar zijn.

De QueryBuilder is de class die verantwoordelijk is voor het bouwen van de queries die de gevraagde data zal ophalen, zoals de naam al aangeeft.

De class OptimalisatiePostProcessor was een class die de opgebouwde OVTableInfo's, van waaruit uiteindelijk het overzicht opgebouwd wordt, te optimaliseren. De class heeft nooit een implementatie gehad, omdat het SelectionFilter hier al rekening mee houdt.

De laatste voorgestelde class voor OVInfo was UitkomstFilter. Deze class zou de verantwoordelijkheid hebben om de hiërarchie die in Figuur 4 (Hoofdstuk 5) te zien is op te bouwen. Het bleek echter dat hier al rekening mee was gehouden in het opbouwen van de selectie, waardoor ook deze class nooit een implementatie heeft gehad.

De laatste class in het diagram is OVInfoAdapter. Deze adapter zou zorgen dat de bestaande code zou kunnen werken met de nieuwe implementatie van de class. Hoofdstuk 9 gaat hier verder op in.

7.4 Evaluatie

Vooral aan het begin van de sprint had ik moeite met het doorgronden van de code. Omdat dit echter een puur theoretische sprint was leverde dit geen echte problemen op. Om me te helpen de code te kunnen ontcijferen raadde Edwin me aan de code in dit stadium nog te beschouwen als een *black box*, in plaats van elke regel individueel te onderzoeken. Omdat het inderdaad nu nog ging om een zeer globaal beeld was het niet zo heel erg dat de methodnamen niet altijd een eenduidig beeld van hun activiteiten gaven. Dit was het enige probleem dat ik in deze sprint ben tegengekomen.

Aan het begin van de sprint was ik gefrustreerd dat ik de code moeilijk te begrijpen vond. Dat was, zeker zo vroeg in het project en met code die zo nieuw voor me was, niet de juiste houding om te hebben. Na het advies van Edwin ging de sprint een stuk soepeler, maar ook in de rest van het project zou ik me soms nog teveel vastbijten op een klein deel van de code en daardoor de rest over het hoofd zien. Meer daarover in de evaluatie van verdere sprints en de uiteindelijke evaluatie van het traject.

Het opstellen van het diagram verliep naar mijn mening ook soepel. Tijdens het uitzoeken van classes overlegde ik met Edwin, die sturing gaf aan dit proces om te zorgen dat de juiste classes zouden worden toegevoegd.

Ondanks de naar mijn mening stroeve start vind ik dat sprint goed is verlopen. Toen ik eenmaal de black box manier gebruikte om naar de code te kijken was ik in staat om redelijk snel de verantwoordelijkheden en dependencies op te stellen. Ook het maken van het diagram verliep soepel.

8. Sprint 2: Main Loop

In dit hoofdstuk bespreek ik de werkzaamheden die ik tijdens de tweede sprint van het project heb uitgevoerd.

8.1 Doel van de sprint

Het doel van deze sprint was het schrijven van een *integratietest* en het uit elkaar halen van de belangrijkste monster method van OVInfo, buildMultipleRoots. Deze method bevat de zogenaamde 'main loop', waarin vragen aan de gebruiker worden gesteld als dat nodig is.

8.2 Werkzaamheden

Na de werkzaamheden die ik in de vorige sprint had uitgevoerd had ik een redelijk beeld van de verantwoordelijkheden en dependencies van OVInfo. Ik kon echter nog niet beginnen met het aanpassen van de code, want zoals in Hoofdstuk 6 werd besproken zou de nieuwe implementatie van OVInfo in eerste instantie parallel zou draaien met de oude versie, zodat Tim goed zou blijven werken tijdens de werkzaamheden.

Vanwege de parallelle implementatie besloot ik op dit punt het *DRY* principe met voeten te treden door de gehele package waarin OVInfo en diens hulpclasses zoals OVTableInfo zich bevonden te dupliceren. De vraag rijst nu misschien waarom ik meteen overging tot zo'n drastische duplicatie. Had ik niet gewoon OVInfo zelf kunnen dupliceren en de rest van de classes ongemoeid kunnen laten? De reden dat ik deze keuze heb gemaakt is dat ik niet zeker wist of OVInfo de enige class zou blijven waar ik wijzigingen in zou aanbrengen. OVTableInfo, bijvoorbeeld, is sterk verbonden met OVInfo en daardoor was het zeer aannemelijk dat ik ook aan deze class werk zou moeten verrichten. Om zeker te zijn dat ik de correcte werking van Tim op geen enkele manier in gevaar zou brengen dupliceerde ik dus de gehele package.

Ik had op dit punt het plan om alvast een integratietest te bouwen, die de oude en nieuwe implementaties naast elkaar zou draaien zodat ik tijdens mijn werkzaamheden eenvoudig kon zien of de nieuwe OVInfo nog het gewenste gedrag vertoonde. De bestaande testclass voor OVInfo gebruikte echter niet de no-args constructor die in de rest van de applicatie veel werd gebruikt. In plaats daarvan was er een constructor gemaakt die vrijwel geen van de vele attributen initialiseerde. Kijkend naar de no-args constructor van OVInfo kwam ik erachter dat deze gebruikmaakte van een environmentvariabele die alleen geïnitialiseerd was als er een actieve sessie was, en daarvoor moest de server draaien en verbinding hebben met een database. Ik zou er dus voor moeten zorgen dat de test een databaseverbinding kon opzetten.

Een story waaraan ik had gewerkt voor ik begon met OVInfo had een test die een databaseverbinding gebruikte. Ik besloot om OVInfo op dezelfde manier verbinding te laten maken met de database zodat ik mijn tests kon schrijven. Ik stelde vast dat de test in kwestie erfde van DBTestBase en zorgde dat mijn testclass ook DBTestBase extendde. Omdat ik eerst wilde weten of deze methode überhaupt zou werken gebruikte ik dezelfde database die de andere test ook gebruikte. Als testcase gebruikte ik enkel de no-args constructor van OVInfo. Als de verbinding werkte zou de globale environment van Tim geïnitialiseerd zijn en zou de test slagen. Zo niet, dan zou er een *exception* optreden en zou de test falen.

In de console-output was duidelijk te zien dat een verbinding met de database werd opgebouwd, maar de test faalde op exact dezelfde manier als eerder: de globale environment gaf een NullPointer. Ik bekeek nogmaals de code van de test die ik als voorbeeld had gebruikt. Nu zag ik dat er een aantal *mocks* werden gebruikt, waaronder één van de globale environment.

Ik wilde de mock gaan overnemen in de test, maar voor ik dat kon doen wees Jeroen me erop dat er in de nieuwe branch van Tim een manier was om geen omgevingen te hoeven mocken, omdat een echte server gestart zou worden. Ik besloot daarom te wisselen van branch en daarop verder te werken aan de test.

Nog voor ik daar goed en wel aan kon beginnen had ik een gesprek met Edwin, die zei dat hij me kon helpen aan een goede database om mee te testen. Hij kon die echter niet op korte termijn beschikbaar hebben, waardoor het schrijven van de tests zou worden uitgesteld naar een later moment. Dit hield in dat ik bij het refactoren extra voorzichtig zou moeten zijn, omdat ik alleen zou kunnen testen in de applicatie zelf. Dit zou de kans op fouten vergroten.

Nu het testen was uitgesteld richtte ik me op de tweede belangrijke taak die ik in deze sprint zou moeten uitvoeren: het uit elkaar halen van de monster method buildMultipleRoots. OVInfo heeft verschillende build methods voor verschillende situaties, maar binnen die methods wordt altijd buildMultipleRoots aangeroepen om het echte werk te doen. Deze 241 regels tellende monster method initialiseerde de OVTableInfo's, bepaalde de persoon voor wie het overzicht bedoeld was, vroeg om de periode, vroeg om het perspectief, bepaalde rechten, en itereerde over de OVTableInfo's om indien nodig daar nog vragen in te stellen.

```
if (!isBusy()) {
    //if (!noQuestions  &&  forceAsk == null)
    //GlobalTimJEnv.getUserEnv().flushRightsCache();
    boolean force = forcePer != null;
    reset(noQuestions, forceAsk);
    if (!noQuestions)
    {
        personal = ispersonal;
    }
    if (forceAsk != null)
    {
        getTableInfo(forceAsk).force = true;
    }
    preparePeriod(ovz, noQuestions, forceAsk == OIType.PER || force);
    ovztype.apply(new OverzichtTypeSelector());
}
final boolean isBoekBr = ovztype == OverzichtTypeEnum.BOEKINGBRIEFJE;
final boolean isResPlan = ovz.isTypeResPlan();
final boolean isFiatBr = ovz.isTypeFiat();
if (noQuestions)
{
    persRef = getPersRef();
}
else if (!isBoekBr && !isFiatBr && !hasGlobalPersonRight(true)) // cannot switch perspective!
{
    persRef = null;
}
else if (persRef == null && GlobalTimJEnv.getUserEnv().isPickPerspective(ovztype))
{
    persRef = askPerspective(ovz, dlg);
}
```

Figuur 12: Deel van de oude buildMultipleRoots method

Om een beeld te kunnen geven van de omvang van buildMultipleRoots heb ik Figuur 12 bijgevoegd. Dit is slechts het begin van de method, maar laat al een deel van de control flow constructie zien die door de hele method te vinden is. Door de grote hoeveelheid ifs in de code vond ik het in eerste instantie lastig om te zien welk deel van de code een verantwoordelijkheid vervulde die in een aparte method gezet kon worden. Om hier een oplossing voor te vinden besloot ik te kijken naar lokaal aangemaakte variabelen en uit te zoeken op welke plekken deze in de method werden gebruikt. Als een bepaalde lokaal aangemaakte variabele maar op een klein aantal plekken werd gebruikt en die plekken lagen dicht bij elkaar, dan was de kans immers groot dat ik er een aparte method van zou kunnen maken.

Eén van de eerste plekken in de code die me opviel is te zien bovenin Figuur 12. Op eerste gezicht lijkt de if-statement dingen te doen die weinig met elkaar te maken hebben. Een tweede blik laat echter zien dat deze statements wel een gezamenlijk doel hebben: er worden voorbereidingen getroffen voor acties verderop in de workflow. Dit was een deel van de code dat verplaatst kon worden naar een eigen method, en dat deed ik. IntelliJ, de IDE die ik bij de project heb gebruikt, heeft een functie waarmee een deel van een method kan worden gerefactord naar een aparte method. Hierdoor wordt de kans op het maken van kopieerfouten kleiner en wordt gelijk inzichtelijk welke parameters een gerefactorde method nodig zal hebben. Tijdens dit deel van het project heb ik hier veelvuldig gebruik van gemaakt. Hoe nuttig de tool echter ook is bij het uitvoeren van de refactoring zelf, het blijft de verantwoordelijk van de programmeur om ervoor te zorgen dat de uitgevoerde handeling zinnig is.

Het was nog maar een begin, maar zelfs met alleen dit kleine stukje gerefactord begon ik meer vertrouwen te krijgen in het uit elkaar kunnen halen van de rest van de method. Figuur 13 toont nog een deel van de code met een enkele verantwoordelijkheid. Dit was het deel van de method waar de OVTableInfo's, die de selectie van de gebruiker zouden gaan bevatten werden geïnitialiseerd.

```
getTableInfo(OIType.WTYPE).prefList = new ASPrimaryKeySet(true, true);
getTableInfo( OIType.WTYPE ).prefList.add( pers.getRefOfReference(PersoonObject.WERKNEMER_TYPE_ID));
getTableInfo(OIType.UURS).prefList = new ASPrimaryKeySet(true, true);
getTableInfo( OIType.UURS ).prefList.add(UursoortEnum.NORMAAL);
getTableInfo(OIType.CMPNIV).prefList = new ASPrimaryKeySet(true, true);
getTableInfo( OIType.CMPNIV ).prefList.add(CompetentieDeelnameNiveau.PRIMAIR);
getTableInfo(OIType.AANW).prefList = new ASPrimaryKeySet(true, true);
getTableInfo( OIType.AANW ).prefList.add(AanwezigheidstypeEnum.AANWEZIG);
getTableInfo(OIType.ACCEPT).prefList = new ASPrimaryKeySet(true, true);
getTableInfo( OIType.ACCEPT ).prefList.add(RegistratieAcceptatie.WACHT);
TimJUserEnv env = GlobalTimJEnv.getUserEnv();
afdti.lidList = env.getCachedRightsList(getPersRef(),
                                        TimJFieldsObjectIDEnum.AfdelingObjectID,
                                        per,
                                        FieldConstants.INVALID_FIELD_ID,
                                        FieldConstants.INVALID_FIELD_ID,
                                        false,
                                        AfdelingslidObject.RECHT_GEEN_TIJD_SCHRIJVEN_ID);
afdti.prefList = env.getCachedRightsList(getPersRef(),
                                         TimJFieldsObjectIDEnum.AfdelingObjectID,
                                         now, AfdelingslidObject.RECHT_VOORKEURAFDELING_ID,
                                         FieldConstants.INVALID_FIELD_ID,
                                         false, AfdelingslidObject.RECHT_GEEN_TIJD_SCHRIJVEN_ID);
```

Figuur 13: Initialisatie van de OVTableInfo's

Op deze manier heb ik de methode uit elkaar weten te halen tot 11 methodes. Een belangrijke method die daaronder valt is buildEndSelection. Deze method bevat in de nieuwe implementatie de for-loop waarin de selectievragen worden gesteld. Omdat de loop nu een eigen method heeft zal het later makkelijker worden hem aan te passen zodat er geen vragen gesteld worden. De method bevat nu al een parameter die 'noQuestions' heet. Wellicht zou dat meteen al een antwoord op het probleem zijn.

Voordat ik zomaar deze parameter zou kunnen gebruiken zou ik echter eerst moeten uitzoeken hoe de method aan de juiste selectie komt als er geen vragen gesteld mogen worden. Om hierachter te komen zocht ik naar plekken in de code waar noQuestions true zou zijn. Ik kwam erachter dat de methods die de periode waarover een overzicht zou worden opgebouwd veranderen, buildNextPeriod en buildForPeriod noQuestions altijd op true hadden staan. Toen ik keek wanneer deze methods werden aangeroepen werd me al snel duidelijk dat ik niet zou kunnen vertrouwen op enkel deze parameter. In alle gevallen waar noQuestions true was, was de gebruikersinput al bekend. Dit zou niet werken voor de nieuwe implementatie, en daarom zou ik een nieuwe manier moeten vinden om te voorkomen dat er vragen gesteld zouden worden tijdens het bouwproces. In Hoofdstuk 10 leest u hier meer over.

8.3 Evaluatie

De meeste problemen die ik tijdens deze sprint ben tegengekomen hadden te maken met het testen. OVInfo bleek een stuk lastiger in een *test harness* te krijgen dan ik had gehoopt. Het grootste probleem werd daarbij gevormd door de afhankelijkheid van globale variabelen die alleen beschikbaar waren als

de applicatie echt draaide. Het overschakelen naar de nieuwe branch loste dat probleem in elk geval op, maar dat liet nog steeds het probleem over dat ik niet wist hoe ik zou kunnen controleren of de tests op de juiste manier werden uitgevoerd. Veel methods binnen OVInfo zijn void methods, en daardoor zou ik geen vergelijkingen kunnen maken tussen de oude en nieuwe implementatie. Het uitstellen van het testen betekende dat ik dit later allemaal zou moeten uitzoeken. Gelukkig zou ik er in de volgende sprint al achter komen hoe ik te werk zou kunnen gaan.

Eerder in het hoofdstuk schreef ik dat ik de tests heb uitgesteld omdat de testdatabase niet op korte termijn beschikbaar was. Zonder een goede testdatabase hebben de tests geen betekenis en daarom zou het niet nuttig zijn geweest om verder te werken aan de tests op dit punt in het traject.

Een ander probleem dat ik tegenkwam was het goed opdelen van buildMultipleRoots. In meerdere gevallen tijdens het proces moest ik grote stukken code in één keer refactoren omdat een method maar één returnwaarde mag hebben en deze code in veel gevallen zou leiden tot meerdere returnwaarden. Dit heeft ertoe geleid dat sommige nieuwe methods nog steeds niet klein genoeg waren om een eenduidige verantwoordelijkheid te hebben. Omdat het belangrijkste deel van de werkzaamheden het isoleren van de loop was, werd het verder opdelen van de nieuwe methods uitgesteld.

Voor mijn gevoel verliep de sprint rommelig. Dit kwam doordat ik naar mijn mening niet goed genoeg heb ingepland en voorbereid wat er tijdens de sprint moest gebeuren. Hierdoor raakte ik met testen in de problemen en duurde het langer voor ik kon beginnen met het opbreken van de buildMultipleRoots method. Dit betekende weer dat ik het reorganiseren van de methods naar andere classes moest uitstellen tot de volgende sprint. Wel vind ik dat het proces van opknippen goed is verlopen en dat het resultaat hiervan goed verdeeld zou kunnen worden over de nieuwe classes.

9. Sprint 3: Adapter

In dit hoofdstuk beschrijf ik de werkzaamheden die ik heb uitgevoerd in de derde sprint van het project.

9.1 Doel van de sprint

Het doel van deze sprint was het opdelen van verantwoordelijkheden van OVInfo over de nieuwe classes en het bouwen van een adapter voor de swing client, waarmee de nieuwe OVInfo alsnog vragen zou kunnen stellen om input van de gebruiker te krijgen.

9.2 Werkzaamheden

In de vorige sprint kwam ik niet toe aan het verdelen van de methods die waren ontstaan bij het uit elkaar halen van buildMultipleRoots over de nieuwe classes. Voordat ik werk kon maken van de parallelle implementatie waar in Hoofdstuk 6 over werd gesproken zou ik dit alsnog moeten doen.

Ik begon met het aanmaken van een nieuwe class, genaamd SelectionData. In deze class plaatste ik een aantal OVTableInfo's, dezelfde die in Figuur 13 (Hoofdstuk 8) al te zien waren. Ik maakte een getter en setter voor elk van deze OVTableInfo's en paste de geëxtraheerde method, waarin deze werden geïnitialiseerd, aan zodat deze een SelectionData-object zou teruggeven.

```
selectionData = initializeSelectionData(selectionData);
selectionData.getAfdelingInfo().lidList = env.getCachedRightsList(selectionData.getPersRef(),
        TimJFieldsObjectIDEnum.AfdelingObjectID,
        selectionData.getPeriod(),
        FieldConstants.INVALID_FIELD_ID,
        FieldConstants.INVALID_FIELD_ID,
        false,
        AfdelingslidObject.RECHT_GEEN_TIJD_SCHRIJVEN_ID);
selectionData.getAfdelingInfo().prefList = env.getCachedRightsList(selectionData.getPersRef(),
        TimJFieldsObjectIDEnum.AfdelingObjectID,
        now, AfdelingslidObject.RECHT_VOORKEURAFDELING_ID,
        FieldConstants.INVALID_FIELD_ID,
        false, AfdelingslidObject.RECHT_GEEN_TIJD_SCHRIJVEN_ID);
```

Figuur 14: De gerefactorde initialisatie van de SelectionData

Figuur 14 toont de situatie na deze handeling. Te zien is dat de leesbaarheid ten opzichte van de situatie in Figuur 13 sterk is verbeterd. Ook op andere plekken in de code waar deze OVTableInfo's werden gebruikt verving ik de aanroepen door het SelectionData-object. Terwijl ik hiermee bezig was realiseerde ik mij echter dat de manier waarop ik de situatie nu aanpakte niet zou kunnen werken. De SelectionData moest immers hetzelfde zijn in de hele class en ik maakte nu meerdere instances aan. Ik loste dit probleem op door het SelectionData-object als attribuut op te slaan en dit op de relevante plekken in de class te gebruiken zodat de data consistent zou blijven. Later in het traject komt een soortgelijke situatie nogmaals voor.

De volgende twee methods die werden aangeroepen in buildMultipleRoots hadden beiden betrekking op de rechten van de gebruiker. Hiervoor had ik reeds een class aangemaakt, RightsPerspectiveHelper. Aan de naam is te zien dat deze class zich zowel met rechten als met perspectief bezighoudt. Eigenlijk is de separation of concerns in deze class dus niet optimaal. Dit zou later aangepast moeten worden. Het uitstellen van deze handeling zou het risico met zich meebrengen dat de nodige veranderingen uiteindelijk niet zouden gebeuren, dus is het raadzaam zulke werkzaamheden direct uit te voeren. Op

het moment van uitvoeren dacht ik hier echter niet aan, en daarom wordt er vooralsnog van deze 'combinatie-class' gebruikgemaakt.

Ik wilde nu de rechtenmethods extraheren, maar op het moment dat ik probeerde deze te verplaatsen kwam ik erachter dat ze gebruikmaakten van private methods binnen OVInfo, die uiteraard niet aangeroepen zouden kunnen worden vanuit een andere class. Ik besloot daarom eerst uit te zoeken waar in de class die methods nog meer werden gebruikt. Als ze slechts op één plek werden aangeroepen zou ik ze zonder problemen kunnen verplaatsen.

In dit geval had ik het geluk dat de gebruikte methods alleen maar in de methods die ik wilde verplaatsen werden aangeroepen. Ik begon met het kopiëren van de private methods naar de nieuwe class. Daarna voerde ik refactoring uit. De nu ongebruikte private methods in OVInfo haalde ik weg om zo de code weer DRY te krijgen.

Helaas kon ik niet bij elke verplaatsing de private methods meenemen naar de nieuwe class. Ik loste dat probleem op door de method toegankelijk te maken binnen de package. Hiermee zou de nieuwe class de method kunnen aanroepen, maar zou er ook een dependency ontstaan op OVInfo, iets wat ik eigenlijk wilde voorkomen omdat dat niet aansloot bij het ontwerp dat ik had gemaakt. De oorspronkelijke gedachte, zoals in Figuur 11 (Hoofdstuk 7) te zien is, was dat er een compositie zou ontstaan. Bij een compositie heeft het 'onderdeel' echter geen kennis van het 'geheel'. Op dat moment was de enige oplossing die ik voor dat probleem zag een duplicatie, waarin zowel OVInfo als de nieuwe class dezelfde method bezat. Dit zou geen wenselijke situatie zijn, omdat duplicatie ervoor zorgt dat code minder goed onderhoudbaar wordt. Een wijziging aan één van de gedupliceerde methods zou immers ook in de andere method moeten worden doorgevoerd en duplicatie is meestal niet goed zichtbaar. Ik dacht ook nog aan *inheritance*, maar dat was in dit geval niet logisch omdat de nieuwe classes geen uitbreidingen van OVInfo waren, maar onderdelen. Ik koos ervoor om toch de dependency op OVInfo te houden omdat de dependency voor minder onderhoudbaarheidsproblemen zou zorgen dan duplicatie en verkeerd gebruik van inheritance de code onnodig complex zou maken.

Nu ik de rechtenmethods had verplaatst besloot ik me te richten op het verplaatsen van de queries uit OVInfo naar hun nieuwe class, de SelectionQueryBuilder. Hoewel de queries allemaal betrekking hadden tot een bepaald aspect van het opbouwen van selectiedata, zoals afdelingen of competentieniveau, besloot ik toch een enkele class voor alle queries aan te houden, omdat de leesbaarheid van de code achteruit zou gaan op het moment dat er voor elk type query een aparte querybuilder zou worden gebruikt. Omdat het verplaatsen van de queries op dezelfde manier verliep als bij de RightsPerspectiveHelper zal ik dit niet verder in detail beschrijven.

De volgende class die ik wilde inrichten was het SelectionFilter. Deze class zou, zoals beschreven in Hoofdstuk 7, zich bezighouden met het verwerken van OVTableInfo's om op die manier te zien welke selecties er gemaakt kunnen worden. Ik verwachtte dat de methods die deze class nodig zou hebben ook op andere plekken gebruikt zouden worden, maar de het verplaatsen van de code verliep zonder problemen.

De laatste functionaliteit van OVInfo die ik op dit moment verplaatste was de code die gericht was op het uitzoeken van competenties. Omdat deze code zo doelgericht was maakte ik een nieuwe class,

CompetentieHelper, aan om de code in onder te brengen. Deze class bevond zich niet op het oorspronkelijke diagram, maar is wel zichtbaar op het diagram dat in de bijlage is opgenomen.

Nu ik een groot deel van de code van OVInfo had verplaatst werd het tijd me te richten op het maken van de parallelle implementatie die in Hoofdstuk 6 werd besproken. Omdat er sowieso een adapter gebouwd zou moeten worden, besloot ik op die plek de twee versies van OVInfo naast elkaar te draaien.

Ik begon door de class OVInfoAdapter twee attributen te geven, de oude OVInfo en de nieuwe. Ik nam de constructors van de oude OVInfo over voor de adapter: een no-args constructor en eentje met een String als parameter. Omdat deze class een adapter was, was het van belang dat de juiste methods beschikbaar waren. Om dat te verzekeren nam ik alle public methods van de oude OVInfo over in de adapter.

Toen ik de methods wilde implementeren door de methods uit de OVInfo classes erin aan te roepen bedacht ik me dat het niet mogelijk zou zijn een enkele method twee waardes te laten returnen. Ik kon dus in sommige gevallen niet meteen de implementaties naast elkaar draaien; in elk geval kon ik niet beide waardes gebruiken in het genereren van het scherm dat de gebruiker uiteindelijk te zien zou krijgen.

Om dit probleem te verhelpen introduceerde ik een boolean genaamd ‘old’, die bepaalde welke van de twee implementaties uitgevoerd zou worden. Vooralsnog werd deze *hard-coded* neergezet, maar later zou dat worden veranderd. Maar wacht, als slechts één van de twee OVInfo’s werd gebruikt kon er toch niet echt gesproken worden van een parallelle implementatie? Dat klopt, en daarom bedacht ik, in overleg met Edwin, een manier om dit toch te kunnen doen.

Door de gehele applicatie heen zitten er controles in de code op ongewone waardes of acties in zogenaamde asserts. Bij standaard gebruik van de applicatie worden deze asserts niet meegenomen en business logica dient dan ook niet afhankelijk te zijn van deze asserts, maar voor development-doeleinden zijn ze zeer nuttig. Ik besloot deze asserts ook te gebruiken om de twee OVInfo’s naast elkaar te kunnen draaien.

Ik introduceerde een tweede boolean in de adapter, genaamd ‘both’. Ook deze was op dit moment nog hard-coded. Als de boolean op true stond werden beide implementaties van OVInfo aangeroepen, waarna middels een assert werd gecontroleerd of deze waardes overeenkwamen.

```
public boolean isSelected(OIType type, Object ref)
{
    if(both)
    {
        assert oldOvInfo.isSelected(type, ref) == ovInfo.isSelected(type, ref);
    }
    if(old)
    {
        return oldOvInfo.isSelected(type, ref);
    }
    return ovInfo.isSelected(type, ref);
}
```

Figuur 15: Een method met dubbele implementatie

Figuur 15 laat zien hoe dit er bij een eenvoudige method uitziet. De code in de eerste if heeft geen invloed op wat er wordt gereturnd door isSelected, en als de applicatie wordt gestart zonder de VM optie –ea (enable asserts) zal er zelfs als both true is niks gebeuren. Als asserts echter aanstaan en de waardes verschillen van elkaar zal er een exception optreden. Omdat ik niet de beschikking had over een echte test, zou dit tijdens de rest van mijn werkzaamheden een vervanging vormen. Om deze methode ook bij void methods te kunnen gebruiken riep ik andere methods aan om te controleren of de methods het juiste antwoord hadden opgleverd.

```
if( oldOvInfo.getSelectionList() != null && ovInfo.selectionData.getSelectionList() != null)
{
    assert oldOvInfo.getSelectionList().equals(ovInfo.selectionData.getSelectionList());
    assert oldOvInfo.getSelection().containsAll(ovInfo.getSelection());
}
```

Figuur 16: Assert op een void method

In Figuur 16 is te zien hoe ik na een build vaststel of de uitkomst gelijk is. Als de selectielijst en de selectie overeenkomen betekent dat dat de build methods hetzelfde resultaat hebben opgeleverd en dat de nieuwe implementatie dus correct werkt.

Ik was echter nog niet klaar met mijn werkzaamheden in de adapter. In de nieuwe OVInfo zaten nog steeds methods die vragen konden stellen aan de gebruiker. Dit zou niet meer nodig zijn in de nieuwe, web-based versie van OVInfo, maar moest wel ondersteund blijven in de bestaande swing client. Ik besloot daarom de relevante code uit OVInfo over te zetten naar de adapter. Mijn redenering op dat moment was dat OVInfo de code niet meer nodig had, en dat deze enkel in de adapter aangeroepen zou worden. Achteraf bezien was dit natuurlijk geen goede redenatie. De verantwoordelijkheid van de adapter is ervoor zorgen dat de oude code met de nieuwe kan praten, maar niet het stellen van vragen aan de gebruiker. Dit realiseerde ik me echter pas later. In de volgende sprint komt dit meer aan bod. Mijn directe probleem was nu dat de OVInfo implementatie een dependency had gekregen op zijn adapter. Dat was al een grote indicatie dat mijn gedachtegang bij het nemen van deze beslissing niet goed was geweest. In de volgende sprint zou ik dit rechtzetten. Ook zonder dit stonden er echter nog grote veranderingen op het programma. De oplettende lezer heeft dit wellicht al opgemerkt in Figuur 15, maar ook als dit niet het geval is zal het in het volgende hoofdstuk duidelijk worden.

9.3 Evaluatie

Ik heb bij het verplaatsen van de code een aantal verkeerde beslissingen genomen. Het belangrijkste voorbeeld hiervan noemde ik aan het einde van de vorige paragraaf: OVInfo had een dependency gekregen op de adapter. In geen enkel geval hoort een class afhankelijk te zijn van een adapter. In dat geval is namelijk de verantwoordelijkheid op een verkeerde plek geplaatst, en dat had ik kunnen en moeten weten op het moment dat ik de beslissing nam. Het leidde nog niet tot problemen met het draaien van de code zelf, maar het was niet clean en ik wilde het er snel uithalen.

Ondanks de verkeerde beslissingen ben ik redelijk tevreden over deze sprint. De werkzaamheden zelf verliepen goed en ik bereikte het doel van de sprint zonder al te grote problemen. Wel vind ik dat ik zorgvuldiger had moeten nadenken om fouten zoals die met de adapter te voorkomen. Hoewel het geen

grote problemen heeft opgeleverd kostte het toch tijd om het goed te zetten, en dat was niet nodig geweest als ik meteen de juiste beslissing zou hebben genomen.

10. Sprint 4: Serialisatie van OVInfo

In dit hoofdstuk beschrijf ik de werkzaamheden die ik in de 4e sprint van het project heb uitgevoerd.

10.1 Doel van de sprint

Het doel van deze sprint was het serialiseerbaar maken van OVInfo. Dit was nodig om de nieuwe OVInfo te kunnen aansluiten op de webclient.

10.2 Werkzaamheden

In het verlengde van de vorige sprint was mijn eerste actie het rechtzetten van de fout die ik had gemaakt bij het weghalen van de Dialogable code uit OVInfo. Ik maakte een aparte class aan die ik QuestionHelper noemde. Hierin bracht ik de logica onder die gebruikt zou worden bij het stellen van vragen. Op die manier loste ik de dependency van OVInfo op de adapter op. Tegelijkertijd bood het me een mogelijkheid om de QuestionHelper helemaal niet als attribuut op te slaan bij OVInfo, omdat er alleen vragen werden gesteld in de build methods. In de build methods van de nieuwe OVInfo voegde ik een parameter toe: questionHelper. Omdat dit het aantal parameters, dat eigenlijk al te hoog was, verhoogde ging hiermee ook de technical debt omhoog. Ik gaf echter geen prioriteit aan het verlagen van het aantal parameters omdat ik in eerste instantie alles goed werkend wilde hebben.

In OVInfo zelf paste ik de buildMultipleRoots method verder aan om te zorgen dat de code kon omgaan met een QuestionHelper die null was. De webclient zou immers geen vragen stellen.

```
if(questionHelper != null)
{
    if ((!isBusy() || selectionData.getTableInfo(OIType.PER).isBusy()) &&
            (selectionData.getPeriod() == null || forceAsk == OIType.PER || ovz.isVraagPeriode() && !noQuestions))
    {
        selectionData.setPeriod(questionHelper.askPeriod(dlg, ovz, selectionData));
    }
}
```

Figuur 17: Gebruik van de QuestionHelper

Figuur 17 laat zien hoe de periode-vraag nu alleen gesteld wordt als de QuestionHelper aanwezig is. Als dit niet het geval is wordt de periode gebruikt die reeds aanwezig is in de SelectionData. Wacht, de periode in de SelectionData? Was SelectionData niet alleen een class met OVTableInfo's erin?

Oorspronkelijk was dat inderdaad het geval. In deze sprint zou ik echter moeten gaan nadenken over de manier waarop gebruikersinput van en naar de server gestuurd kon worden, en hoe deze gebruikersinput verkregen zou moeten worden zonder vragen. Ook was het de bedoeling dat OVInfo zoveel mogelijk *stateless* zou worden.

Ik besloot de state van OVInfo zoveel mogelijk over te zetten naar het SelectionData-object. Dit object zou vervolgens heen en weer gestuurd kunnen worden. SelectionData was al een attribuut van OVInfo sinds de vorige sprint, dus zou het verplaatsen van attributen van OVInfo naar SelectionData niet al teveel moeite moeten kosten.

In het begin van dit proces verliep het werk snel. Ik haalde de attributen weg uit OVInfo, plaatste deze in SelectionData, en riep ze aan via de instance hiervan die OVInfo als attribuut had. Op dat moment realiseerde ik me echter dat ik er ook voor moest zorgen dat de attributen zelf geserialiseerd konden

worden. In het geval van *primitives* zoals de booleans was dat geen probleem, maar er waren ook complexere objecten aanwezig. Eén van die objecten was het PersoonObject, dat ik al besprak in Hoofdstuk 7. Omdat er maar zeer weinig attributen van dit object daadwerkelijk nodig waren voor de correcte werking van OVInfo was mijn oplossing voor dit probleem eenvoudig: ik maakte een nieuwe class aan, OVPersoonWrapper, waarin ik de gebruikte velden neerzette. Omdat dit object niets anders was dan een *DTO* maakte ik alle attributen public. Ook gaf ik de class een static functie waarmee de class geïnstantieerd kon worden via een volledig PersoonObject. Na het aanmaken van de class verving ik alle aanroepen van het PersoonObject door de nieuwe wrapper. Het omzetten van PersoonObject naar OVPersoonWrapper vond plaats in buildMultipleRoots, omdat alle verdere aanroepen van het object daarop volgden.

Een andere complexe class die ik moest voorbereiden op serialisatie was OVTableInfo. Ik hoopte dat de serialisatie hiervan net zo goed zou gaan als bij het PersoonObject. Helaas bevatte OVTableInfo een DialogableState, en die mocht niet meegenomen worden in de serialisatie. Verder bevatte OVTableInfo nog een aantal Strings die alleen nodig waren voor de DialogableState.

Mijn eerste gedachte was het refactoren van de DialogableState naar de QuestionHelper. Dit was helaas geen goede oplossing, omdat er zoveel types OVTableInfo waren dat het verwijderen van de vraag te complex zou worden om nog nuttig te zijn. Wat nu werd gedaan door private methods zou bij een extractie door externe calls geregeld moeten worden, waardoor de verantwoordelijkheid van het opbouwen van de vraag niet meer bij de QuestionHelper zou liggen, maar meer bij OVInfo.

De volgende optie die ik overwoog was het markeren van de ‘ongewenste’ velden als transient. Hiermee zou ik voorkomen dat de virtual machine deze attributen zou meenemen bij de serialisatie. Dit leek mij op dat moment een werkende oplossing, dus markeerde ik de velden die ik niet wilde meesturen en ging verder met het nalopen van andere attributen in SelectionData.

Tijdens mijn controle van de andere attributen in SelectionData merkte ik dat mijn gedachten steeds teruggingen naar OVTableInfo. Ik had het gevoel dat ik iets miste en dat mijn oplossing voor het serialisatieprobleem daar een belangrijke rol in speelde. Ik bekeek nogmaals de class, maar kon niet direct vinden wat er fout was. Alle velden die enkel voor de vraag gebruikt werden waren gemarkeerd als transient, dus dat kon het probleem niet zijn. Ik besloot om een paar tests te doen in de applicatie zelf.

Ook hier leek alles goed te werken. Ik draaide de nieuwe implementatie met de QuestionHelper, dus alle vragen werden gesteld. Alle vragen. Dat was wat ik over het hoofd had gezien. De vragen mochten dan niet worden geserialiseerd, maar bij het deserialiseren zouden ze alsnog worden aangemaakt en gesteld. Deze vragen waren niet afhankelijk van de QuestionHelper, dus als deze null was zouden er geen exceptions optreden.

Ik dacht eraan om alsnog de vraag uit OVTableInfo te verplaatsen naar QuestionHelper en besloot te kijken naar de beste manier om dit te doen, maar nog voordat ik iets kon bedenken kreeg ik een beter idee. De vraag van OVTableInfo mocht best blijven staan in de class; het enige dat uitmaakte was dat de vraag niet gesteld zou worden.

```
if (t != OIType.PER && !isOtherBusy(t) && (selectionData.getTableInfo(t).ask || t == forceAsk) && questionHelper != null)
{
    askSelection(ovz, t, dlg, noQuestions && t != forceAsk);
}
else
{
    noQuestions = (noQuestions && t != forceAsk) || questionHelper == null;
    ti.checkSelection(ovz, noQuestions);
}
```

Figuur 18: QuestionHelper bij vragen OVTableInfo

Figuur 18 toont de manier waarop ik voorkom dat OVInfo vragen zou stellen als dit niet gewenst is. Hoewel de DialogableState in OVTableInfo geen deel uitmaakt van de QuestionHelper is diens aan- of afwezigheid wel de factor die bepaalt of er al dan niet een vraag gesteld moet worden. Door dit als extra conditie op te nemen in de if kon ik forceren dat er geen vragen gesteld zouden worden, maar dat was nog niet genoeg.

In de else statement in Figuur 17 wordt de boolean noQuestions gezet. Dat gebeurde oorspronkelijk alleen met de statement links van de or. Dit zou tot gevolg kunnen hebben dat noQuestions false zou zijn, waardoor de code in checkSelection niet uitgevoerd zou worden en daarmee de uitkomst van de method zou kunnen veranderen. Om die reden voegde ik de extra conditie toe.

Nu dit was gedaan dacht ik klaar te zijn met de serialisatie van OVInfo. In het volgende hoofdstuk wordt duidelijk dat dit nog niet het geval was.

10.3 Evaluatie

Het verwijderen van de vraag uit OVTableInfo heeft me in deze sprint veel tijd gekost. Omdat het zo'n kritiek punt was in het serialisatieproces heb ik me voornamelijk daarop gericht. Het gevolg daarvan is dat ik het serialisatieproces nog niet volledig had afgerond aan het einde van de sprint, terwijl dit wel zo had moeten zijn. In de volgende sprint zou ik daarom nog tijd moeten besteden aan de losse eindjes van deze sprint.

Afgezien van mijn tunnelvisie op de OVTableInfo denk ik dat de rest van de sprint goed is verlopen. Het introduceren van de QuestionHelper, die ik eigenlijk alleen had gebouwd voor de vragen in OVInfo zelf, bleek een goede manier te zijn om ook OVTableInfo tegen te houden met het stellen van vragen. Omdat ik al in de eerste sprint had vastgesteld welke velden van het PersoonObject nodig waren kon ik in deze sprint snel een wrapper ervoor maken en ook het implementeren van deze nieuwe wrapper verliep zonder problemen.

11. Sprint 5: REST service en repareren serialisatie

In dit hoofdstuk beschrijf ik mijn werkzaamheden tijdens de vijfde sprint van het project.

11.1 Doel van de sprint

In deze sprint was mijn doel het maken van een REST-service voor de nieuwe OVInfo, waarmee de SelectionData van de server naar de webclient kon worden gestuurd.

11.2 Werkzaamheden

In deze sprint wordt er gebruikgemaakt van het Spring Framework. Relevante zaken van dit framework worden uitgelegd in het verslag, maar omdat de configuratie ervan buiten de scope van dit project valt zal ik dat deel achterwege laten.

Ik begon de sprint door een interface aan te maken. Omdat OVInfo belangrijk is voor de rapportages in Tim noemde ik deze service ReportService. Ik annoteerde de service met de Spring annotatie @Service zodat ik de *dependency injection* van Spring zou kunnen gebruiken. Ik bouwde deze service als interface zodat ik later bij het testen van de controller makkelijker kon werken met een mock. De implementatie van de service noemde ik TimReportService. Ik koos deze naam omdat soortgelijke namen ook worden gebruikt voor andere service implementaties binnen Tim en ik mijn code consistent wilde houden met wat er al aanwezig was.

De service kreeg als attribuut de nieuwe OVInfo. De eerste method die ik maakte retourneerde de SelectionData van OVInfo en had zelf geen parameters. De andere methods waren de verschillende build methods en de pushQueries method, waarmee de nodige database calls uitgevoerd zouden worden om de data die de gebruiker geselecteerd heeft daadwerkelijk op te halen. Deze methods hadden als parameter naast de SelectionData ook een OverzichtObject.

Het OverzichtObject is net als het PersoonObject een FieldsObject, wat betekent dat het een groot aantal velden bevat die voor de werkzaamheid van OVInfo niet van belang zijn. Zoals ik in Hoofdstuk 7 al aangaf was het echter niet mogelijk om het OverzichtObject op een soortgelijke manier als het PersoonObject te voorzien van een wrapper. In OVTableInfo wordt er namelijk een veld opgehaald van het OverzichtObject dat afhankelijk is van het type OVTableInfo. Omdat een eventuele wrapper class herbruikbaar moet zijn is het niet clean om al deze types op te nemen in de wrapper. Mocht er in de toekomst een nieuw type bijkomen zou specifieke code problemen opleveren.

De eenvoudigste oplossing die ik kon bedenken was het opnemen van het id van het OverzichtObject in de SelectionData. Aan de hand van het id zou ik dan in de service de OverzichtDAO kunnen aanroepen om het juiste object op te halen. Ik gaf de service een tweede attribuut, een Map waarin ik id's als sleutel gebruikte en OverzichtObjecten als waarde. Vervolgens maakte ik een private method aan die als parameter een id meekreeg.

Figuur 19 op de volgende pagina toont de method in kwestie. Eerst controleer ik of de Map leeg is. Als dit het geval is wordt de database aangeroepen om deze te vullen, waarna het gevraagde OverzichtObject wordt teruggegeven. Maar waarom op deze manier? Waarom niet gewoon met het id direct het object uit de database ophalen? Dat had inderdaad gekund, maar om de *disc I/O* beperkt te houden heb ik ervoor gekozen een lokaal attribuut te vullen met de data. Omdat inlezen uit RAM sneller is dan inlezen van de schijf komt dit de performance van de applicatie ten goede. Deze aanpak brengt

echter wel risico's met zich mee. Als een OverzichtObject wordt aangepast terwijl de applicatie draait is het object in de Map niet consistent met het echte overzicht. De kans dat dit gebeurt is niet groot, en om die reden heb ik besloten toch deze oplossing te gebruiken.

```
private OverzichtObject getOverzicht(Object ref)
{
    if(overzichten.isEmpty())
    {
        List<OverzichtObject> overzichtObjectList = new OverzichtDAO().findAll();
        for(OverzichtObject ovz : overzichtObjectList)
        {
            overzichten.put(ovz.getRef(), ovz);
        }
    }
    return overzichten.get(ref);
}
```

Figuur 19: Het ophalen van de overzichten via hun id

Nu ik deze method had voegde ik aan de SelectionData het id van het OverzichtObject toe. Daarna verwijderde ik de parameter uit de method-declaratie, daar het door OVInfo benodigde object nu uit de SelectionData opgehaald kon worden.

Op dit punt liet ik de service even voor wat hij was om me op de controller te kunnen richten. Om de naamgeving met de service consistent te houden noemde ik deze controller ReportController. De controller kreeg twee annotaties. @RestController wordt gebruikt om Spring te vertellen dat de class een controller is en dat deze een *JSON*-antwoord teruggeeft in de body van de response. @RequestMapping("/report") vertelt Spring dat de URL van deze controller '/report' is.

Ik gaf de controller een ReportService als attribuut. Op de constructor plaatste ik de annotatie @Autowired, zodat Spring de instantie zelf zou injecteren. Er bestaat nog een annotatie om ditzelfde te doen, @Inject. Deze heeft het voordeel dat hij niet Spring-specifiek is en ook door andere frameworks gebruikt zou kunnen worden. De reden dat ik toch @Autowired gebruik is consistentie. In alle andere controllers wordt @Autowired gebruikt voor dependency injection en om de code goed leesbaar te houden vond ik het beter om ook die conventie te volgen.

De methods in de controller waren dezelfde als in de service. Waar in de service de build methods echter void methods waren koos ik er hier voor om elke method de SelectionData te laten teruggeven, zodat de client altijd de meest up to date SelectionData zou hebben. Ik annoteerde deze methods net als de class met @RequestMapping, met elk een andere waarde. Deze mappings zouden worden toegevoegd aan de mapping op classniveau om op die manier de correcte URL te vormen. De standaard build method had bijvoorbeeld '/build' als mapping. Dit betekent dat de URL '/report/build' deze method zou aanroepen.

Voordat ik overging tot het maken van tests voor de controller wilde ik controleren of de applicatie werkte met beide implementaties aan. Toen ik in het vorige hoofdstuk testte deed ik dit met enkel de nieuwe aan, en dat leek goed te gaan, maar ik zou pas zeker weten of alles klopte als ik beide implementaties naast elkaar draaide. Als u nu denkt, 'goh, misschien had je dat eerder moeten doen'

heeft u volkomen gelijk. Zodra ik beide implementaties naast elkaar draaide kreeg ik assertion errors. De waardes van de oude en nieuwe OVInfo's kwamen niet overeen, en dus was er ergens iets fout gegaan.

Ik gebruikte de debug mode van IntelliJ om te kijken op welke manier de waarden van elkaar verschilden. Ik stelde vast dat het leek of de SelectionData leeg was in de nieuwe situatie. Ik vroeg me af waarom dat zo zou zijn, want ik gaf de SelectionData gewoon mee aan OVInfo, die het vervolgens weer doorgaf bij het instantiëren van diens hulpclasses. Terwijl ik de code van OVInfo naliep bedacht ik me wat het probleem was. Ik gaf de SelectionData wel door aan de nieuwe classes, maar vervolgens werd deze in OVInfo zelf niet meer geüpdatet, waardoor de teruggegeven data niet correct was. Omdat er hierbij geen exceptions optraden had ik dit niet opgemerkt bij mijn controle in de vorige sprint.

Ik wilde dit eerst oplossen door een updateSelectionData method te maken in OVInfo, die in het bouwproces aangeroepen zou worden om de SelectionData up to date te krijgen. Ik bedacht me echter dat dit niet zou helpen, omdat ook de andere classes de juiste SelectionData moesten hebben. Om die heb ik de oplossing uit Figuur 20 bedacht.

```
selectionData = selectionFilter.processSelectionData(ovz, ovzSelectionList, now, ovztype, isBoekBr, isResPlan, isFiatBr, selectionData);
selectionData = rightsPerspectiveHelper.determineRights(ovz, noQuestions, now, ovztype, perspflg, selectionData);
selectionData = rightsPerspectiveHelper.determineRightsPerspective(ovz, dlg, disableQuestions, helpItemKey, selectionData, questionHelper);
selectionData = rightsPerspectiveHelper.generateRightsLists(selectionData);
```

Figuur 20: Up to date houden van SelectionData

Zoals in de figuur te zien is heb ik de methods van de andere classes aangepast zodat ze allemaal het SelectionData object teruggeven. Deze waarde assign ik vervolgens aan de instance die OVInfo bezit, waarna ik hem in de volgende method call weer doorgeef aan de volgende class (of dezelfde, zoals hier met de RightsPerspectiveHelper). Door deze verandering hoeven de nieuwe classes de SelectionData niet meer op te slaan als attribuut, omdat ze nu in elke method de huidige versie ervan binnenkrijgen. Alleen OVInfo slaat nu de SelectionData nog op. Eigenlijk was het de intentie om OVInfo geheel stateless te maken, maar door het grote aantal private methods binnen OVInfo waarin de SelectionData nodig is heb ik besloten deze toch als attribuut te behouden. Nu ik deze veranderingen had doorgevoerd werkte de parallelle implementatie. Vanaf dit moment zou ik ervoor zorgen dat ik altijd draaide met beide implementaties, om verdere problemen zoals dit te voorkomen.

Ik ging verder met de eigenlijke werkzaamheden van de sprint door unittests voor de controller te maken. Deze tests maakten gebruik van een gemockte webclient waarin een URL wordt doorgegeven, en eventuele parameters die de controller method die wordt aangeroepen nodig heeft. De eerste test die ik maakte was het ophalen van de huidige SelectionData, omdat deze method geen parameters had. Toen de test in één keer slaagde was ik erg blij, want dat betekende dat ik de serialisatie goed had aangepakt. Tenminste, dat dacht ik.

Ik nam de JSON die ik had verkregen van de eerste test over om de testbody van de tweede test, een build test, op te stellen. Voor deze test moest ik ook een mock maken van de service. Dit was gelukkig eenvoudig omdat de service een interface was. Ik noemde de mock ReportServiceMock en zorgde dat er een nieuw SelectionData object werd teruggegeven in de getSelectionData method.

Voor ik echter kon gaan testen moest ik eerst zorgen dat de formattering van de enorme JSON String klopte, dus was ik een tijd bezig met het escapen van alle quotes. Al tijdens dit proces begon ik me af te

vragen of deze serialisatie juist was, want ik zag bij de OVTableInfo's alleen de selectionList terug, terwijl er meerdere belangrijke lists waren. Dit was echter slechts het topje van de ijsberg.

Op het moment dat ik klaar was met het noteren van de JSON voor de requestbody van de test en de build test voor het eerst draaide trad er een exception op. Er ging iets fout bij het deserialiseren van een attribuut van SelectionData, de ovzSelectionList. Dit was een lijst van het type ASPrefetchedList, een abstracte class. Hierdoor wist de deserializer niet welke implementatie van deze class gebruikt moest worden.

Gelukkig was dit een probleem dat snel verholpen kon worden met een annotatie. Door de getter van het attribuut te annoteren met @JsonDeserialize(as = ASFieldsObjectList.class) was het voor de deserializer duidelijk welke implementatie gewenst was bij het deserialiseren. Ik koos deze specifieke class omdat ik in de oude code op zoek was gegaan naar het moment dat de lijst geïnstantieerd werd en zag dat ASFieldsObjectList de concrete class was die daarvoor werd gebruikt.

Ik hoopte dat hiermee mijn problemen waren verholpen, maar helaas was dit niet het geval. Er trad weer een exception op, ditmaal op het deserialiseren van de tableInfo Map. Deze Map had als sleutel een OIType en als waarde een OVTableInfo. Dit waren al concrete classes, dus kon ik niet dezelfde annotatie gebruiken als de vorige keer. Ik zocht op of @JsonDeserialize nog andere waarden had naast 'as' die ik in dit geval zou kunnen gebruiken. Ik kwam erachter dat dit inderdaad zo was. Voor Maps waren de opties keyUsing en valueUsing beschikbaar. Hiermee kon ik een zelfgemaakte deserializer specificeren.

Ik richtte me eerst op het maken van een deserializer van OIType. Dit leek me de gemakkelijke deserializer, omdat OIType een static fromString method bevat, en ik in de JSON al had gezien dat ik beschikte over een geserialiseerde String waarvan ik aannam dat het de juiste zou zijn. Er bestaat in dit vak een gezegde 'assumption is the mother of all fuck-ups'. Ik zou er snel achterkomen dat dit inderdaad het geval was.

Voordat ik mijn openbaring had moest ik echter nog een deserializer bouwen, dit keer voor OVTableInfo. Deze was een stuk complexer dan die van OIType omdat ik hierbij alle waarden die ik wilde hebben expliciet moest opgeven. Dit betekende dat ik in OVTableInfo zelf een aantal getters en setters zou moeten maken. Op dit moment had namelijk alleen een getter en setter voor de selectionList. Dat verklaarde gelijk waarom er bij de serialisatie geen andere lijsten aanwezig waren: ze hadden geen getters en setters en werden dus overgeslagen door de serializer. Dit hield ook in dat mijn transient markeringen geheel nutteloos waren, omdat deze attributen toch geen getters en setters hadden. Figuur 21 op de volgende pagina toont een deel van de deserializer. Zoals te zien is in de figuur wordt elke waarde apart ingelezen uit de JsonNode. Nadat alle waardes zijn ingelezen wordt er een OVTableInfo aangemaakt waar alle ingelezen waardes in geset worden.

Toen ik mijn test opnieuw draaide kreeg ik een nieuwe error. Ditmaal was het een HTTP status error, een 400: Bad Request. Er klopte dus iets niet aan mijn syntax, en ik wist al snel wat er fout was. De enorme JSON string die ik had bevatte nog de verkeerde serialisatie van OVTableInfo, waarin alleen de selectionList aanwezig was. Ik had weinig zin om de gigantische lap tekst aan te passen, dus besloot ik te kijken naar een eenvoudigere methode om aan de JSON van SelectionData te komen. Ik ontdekte dat ik een JsonParser kon instantiëren waarmee ik SelectionData kon serialiseren. Deze String kon ik

vervolgens gebruiken als requestbody in de test. Hiermee was mijn handgetypte String niet meer nodig, wat de onderhoudbaarheid van de test ten goede zou komen omdat een wijziging in de code nu niet expliciet in deze test aangegeven hoefde te worden.

Helaas voor mij was ook dit niet genoeg om de test te laten slagen. OIType kon niet goed gedeserialiseerd worden. Ik wist niet waarom, want ik had hiervoor al een deserializer geschreven. Ik bekeek het resultaat van de serialisatie en zag niks verkeerds. Afdeling werd geserialiseerd als Department, maar dat was nog steeds wat ik als 'correct' beschouwde.

```
JsonNode node = jsonParser.getCodec().readTree(jsonParser);

OIType type = OIType.fromString(node.get("type").asText());
int requiredRight1 = (Integer) node.get("requiredRight1").numberValue();
int requiredRight2 = (Integer) node.get("requiredRight2").numberValue();
int requiredRight3 = (Integer) node.get("requiredRight3").numberValue();
int forbiddenRight = (Integer) node.get("forbiddenRight").numberValue();
ASPrimaryKeySet lidList = getList("lidList", node);
ASPrimaryKeySet askList = getList("askList", node);
ASPrimaryKeySet selectionList = getList("selectionList", node);
ASPrimaryKeySet revSelectionList = getList("revSelectionList", node);
ASPrimaryKeySet badList = getList("badList", node);
ASPrimaryKeySet rootList = getList("rootList", node);
ASPrimaryKeySet rightsList = getList("rightsList", node);
ASPrimaryKeySet prefList = getList("prefList", node);
boolean ask = node.get("ask").asBoolean();
boolean force = node.get("force").asBoolean();
boolean reverse = node.get("reverse").asBoolean();
boolean single = node.get("single").asBoolean();
boolean forbidAsk = node.get("forbidAsk").asBoolean();
boolean useLidPer = node.get("useLidPer").asBoolean();
```

Figuur 21: OVTableInfo deserializer

Een tweede blik op OIType toonde me echter wat er foutging. Het name attribuut van OIType was een String-weergave van een enum. In het geval van een afdeling was de naam dus 'OIType.AFD' en niet 'Department'.

Ach, als het serialiseren foutging was de oplossing duidelijk, leek me. Ik moest gewoon een serializer bouwen om te zorgen dat het goedging. Ik zocht op hoe de JsonSerializer werkte en was aangenaam verrast om te zien dat het slechts een paar eenvoudige methods waren. Het JSON-object werd geopend met de writeStart method, gevuld door methods als writeString en writeNumber, en gesloten door writeEnd. Ik bouwde een serializer waarin ik enkel de naam van het OIType serialiseerde en draaide de test.

Nog steeds kreeg ik de error dat OIType niet goed gedeserialiseerd kon worden, maar ditmaal was het omdat de JSON niet leesbaar was. Dat vond ik vreemd, want ik maakte de String niet meer zelf aan omdat nu de beschikking had over een parse method. Ik bekeek het resultaat van het serialiseren en zag niet meteen wat er mis was. De naam van het OIType klopte nu, dus wat was er mis? Toen zag ik dat er

voor de naam nog een ':' stond die ik niet had verwacht. Ik controleerde de documentatie van de JsonSerializer om te kijken wat ik fout had gedaan, maar ik kon niks vinden. Ik experimenteerde met het al dan niet gebruiken van de writeStart en writeEnd methodes omdat ik dacht dat er nu misschien een apart object werd gemaakt van OIType en dat dat voor problemen zorgde, maar dit bleek het probleem niet op te lossen. Ik probeerde een overloaded versie van de writeString method, waarmee het gemaakte object een String als 'sleutel' kreeg, maar ook dit hielp niet.

Uiteindelijk heb ik ervoor gekozen om in plaats van het serialiseren van OIType een andere oplossing te gebruiken. Ik veranderde het type van de sleutel van de Map van OIType naar String, en koos hier als waarde de naam van het OIType voor. Omdat een OIType eenvoudig te maken is uit niks anders dan de naam levert dit in de rest van de applicatie geen problemen op.

Na lang proberen, werkten mijn controllertests nu eindelijk. Helaas was de sprint voorbij, waardoor de rest van het testen in de volgende sprint zou moeten gebeuren.

11.3 Evaluatie

Deze sprint was lastig om de verkeerde redenen. De werkzaamheden die ik moest uitvoeren hadden bijna allemaal betrekking op de serialisatie, terwijl dat het onderwerp van de vorige sprint was. Het feit dat ik hier zoveel tijd aan moest besteden geeft aan dat ik me in de vorige sprint teveel heb gericht op het omzeilen van de eventuele vragen die OVInfo zou stellen. Daardoor heb ik niet nauwkeurig genoeg gekeken naar de rest van de serialisatie en nam ik te snel aan dat het in orde was. Gelukkig had ik al enige ervaring met Spring, dus hoefde ik daar geen onderzoek naar te verrichten. Dat is een onderdeel van de sprint waar ik wel tevreden over ben.

12. Sprint 6: Integratietests schrijven

In dit hoofdstuk beschrijf ik de werkzaamheden die ik heb uitgevoerd tijdens de zesde sprint van het project.

12.1 Doel van de sprint

Het doel van deze sprint was het schrijven van integratietests, gebaseerd op veel voorkomende en zeldzame testcases.

12.2 Werkzaamheden

Zodra de sprint begon wilde ik beginnen aan het testen van de OVInfo-service. Ik besefte echter al snel dat de methods in deze class doorgeefluiken waren naar OVInfo. Om OVInfo goed te kunnen testen was het noodzakelijk om integratietests te schrijven, want er was een databaseconnectie nodig om de juiste informatie te kunnen ophalen.

In Hoofdstuk 8 besprak ik al het begin dat ik maakte met het schrijven van deze tests. Toen had ik het werk uitgesteld omdat de database met testcases nog niet beschikbaar was. Ook aan het begin van deze sprint was dat nog het geval, maar ik kon in elk geval beginnen met het opstellen van een test. Deze zou ik draaien zodra ik de beschikking had over de juiste data.

Omdat ik in de derde sprint van het project een adapter had gebouwd waarin beide implementaties van OVInfo naast elkaar draaiden wist ik hoe ik kon controleren of de build methods een goed resultaat hadden opgeleverd. Ik gaf de integratietest twee attributen: de oude OVInfo en de nieuwe. Ik gebruikte de @BeforeClass annotatie om de databaseconnectie op te zetten terwijl ik de testcode zelf binnen een method genaamd runInUserContext draaide. Deze method accepteerde een username als parameter, waarna de test zou draaien als deze gebruiker.

Binnen de test instantieerde ik de twee OVInfo's. Dit moest binnen de test gebeuren en niet binnen bijvoorbeeld een @Before geannoteerde method, omdat het voor de correcte werking belangrijk is dat de instances worden aangemaakt als de juiste gebruiker.

Op dit punt ontving ik de database met testdata van Edwin. Het eerste dat ik deed was de database inlezen in mijn gewone applicatie. De database bevatte zes gebruikers die voor mij van belang zouden zijn. In volgorde van weinig naar veel rechten zijn dit:

- Een gewone werknemer
- Een afdelingshoofd
- Een projectleider
- Een afdelingshoofd die projectleider is
- Een applicatiebeheerder
- Een applicatiebeheerder die afdelingshoofd en projectleider is

Voordat ik in de test kon inloggen onder deze gebruikers moest ik eerst zorgen dat deze ongeverifieerd mochten inloggen. Dit was nodig omdat de runInUserContext alleen een username accepteert en geen wachtwoord. Ik opende de serverconsole van Tim, een webpagina (draaiend op localhost, niet op een externe server) van waar een applicatiebeheerder de serverinstellingen en databaseverbinding kon beheren. Ik navigeerde naar de gebruikerspagina, waar ik de namen van de testgebruikers in de lijst

opzocht om een ongeverifieerde login te laten toestaan. Het viel me op dat ik de gebruikers allemaal twee keer in de lijst zag staan, onder verschillende netwerknamen. Eén van die namen was de volledige naam, de andere slechts een paar letters die kort aangaven wat de functie van de werknemer was. Ik besloot enkel de korte gebruikersnamen ongeverifieerde logins te geven, omdat deze makkelijker in gebruik zouden zijn in de test dan de langere namen.

Nu de gebruikers de juiste rechten hadden kon ik gaan kijken naar de overzichten die er in de database stonden. Per overzichtdefinitie zou ik een test moeten schrijven voor alle gebruikers. Er was een groot aantal categorieën die elk een aantal overzichten bevatte. De overzichten bevatten alle permutaties van enkelvoudige selecties, meervoudige selecties, negatieve selecties en alles er tussenin. Ook was er een overzicht waarin alles zou worden gevraagd. Dit overzicht zou in een echte applicatie geen nut hebben, maar in de test zou het onmisbaar zijn om te controleren of alle vragen goed worden gesteld in de nieuwe implementatie.

Ik plaatste het conversiebestand van de database in de resources folder van de test en configureerde de setup van de test om er verbinding mee te maken. In de test zelf schreef ik een Assert om te controleren of de build method die ik aanriep in beide gevallen hetzelfde resultaat teruggaf. De reden dat ik niet zelf definieerde wat in elke situatie mijn verwachte uitkomst was, was dat ik niet precies wist wat die uitkomst in de oude situatie zou moeten zijn. De oude class was zo vaak uitgebreid met quick fixes en hacks dat er side effects zouden zijn die niet eenvoudig te vinden zouden zijn. Het was echter zo dat de oude situatie de gewenste uitkomsten opleverde, en daarom zag ik een test als geslaagd op het moment dat de nieuwe implementatie dezelfde uitkomst had als de oude.

Nu ik de basis van een build test had opgezet besloot ik te controleren of de test werkte in een eenvoudige situatie waarin een enkele vraag over het competentieniveau gesteld zou worden. Hiervoor moest ik rekening houden met een class die ik tot nu toe nog niet nodig had gehad, de Dialogable. De DialogableState mocht dan verantwoordelijk zijn voor de vraag die gesteld werd en de keuzes die gemaakt konden worden, de Dialogable zelf was de class die de vraag toonde en waarin de gebruiker input gaf. Ik zocht in de code van de applicatie naar de plek waar de Dialogables werden geïnstantieerd voordat de build method werd aangeroepen. Ik kwam uit op een class die DisabledDialogable heette. Ik maakte me zorgen om de naam, maar besloot eerst te kijken of ik de test kon draaien met een instantie van deze class.

Ik bedacht me op dat moment dat ik nog niet had doorgegeven welk overzicht ik nodig had. Dat was een probleem, want ik wist niet wat de id's van de overzichten waren. Ik wist echter wel hun namen. Ik bouwde een soortgelijke method als in de vorige sprint, waarin ik alle OverzichtObjecten ophaalde en in een Map plaatste. In dit geval koos ik echter niet het id als sleutel, maar de naam. Als de Map gevuld was zou ik met enkel de naam het correcte overzicht kunnen opvragen.

Terwijl ik hiermee bezig was kwam ik tot de conclusie dat er van case tot case weinig zou veranderen in de uitvoering van de test. Het enige dat per situatie veranderde was de gebruiker en het overzicht. Om duplicatie te voorkomen veranderde ik de build test in een private method met twee parameters: username en overzichtsnaam. Daarboven maakte ik nog een private method die ik testCompetentieVraag noemde, en als parameter enkel een username meegaf. Binnen deze method riep ik test zelf aan met de doorgegeven username en "Competentieniveau vraag" als overzichtsnaam. Op deze manier zou ik zonder duplicatie de echte test kunnen schrijven. Ik maakte nu de daadwerkelijke

test method, testCompetentieVraagAsProjectleiderAfdelingshoofdAndApplicatiebeheerder, waarin ik de eerder gemaakte testCompetentieVraag method aanriep met username “ahplab”.

Ik beschrijf dit proces nu in detail, omdat het verder schrijven van testcases neerkomt op deze handelingen. Omdat er meer dan 500 testcases zijn zal ik niet voor elke testcase apart mijn code beschrijven.

Ik draaide de test, die goed werkte. De uitkomsten van beide OVInfo’s waren dus gelijk, maar was de vraag die gesteld moest worden echt gesteld? Dat leek me nogal sterk omdat de Dialogable die ik had meegegeven een DisabledDialogable was. Om er zeker van te zijn dat er een keuze gemaakt zou worden moest ik dus een eigen Dialogable schrijven.

Ik maakte de class TestDialogable aan en implementeerde de Dialogable interface, maar ik realiseerde me dat ik niet precies wist welke method er gebruikt werd om informatie te tonen en te verwerken. Ik ging hiernaar op zoek in de bestaande code en kwam erachter dat ik de showDialogAndWait method nodig had. Binnen deze method zou ik gebruikersinput moeten simuleren, maar ik wist niet wat in elke situatie mijn keuzes zouden zijn. Ik zag ook dat de method getInput de ingevoerde input zou ophalen. Ik voegde daarom een input attribuut van het type Object toe. Dit was wat de getInput methode teruggaf.

Ik richtte me nogmaals op de bestaande code. De showDialogAndWait method kreeg een Message object door, maar dit was een abstracte class. Ik vond in de bestaande code een subclass van Message genaamd MessageList. Deze MessageList bevatte items die de keuzes voorstelden.

In mijn TestDialogable liet ik de method controleren of de binnengekomen Message een instantie was van MessageList. Als dat zo was haalde ik de items op en maakte ik afhankelijk van het aantal items een keuze. Ik wilde een gespreide selectie en koos er daarom voor om alle oneven keuzes te selecteren in een lijst. Als er slechts één keuze was zette ik de input op die keuze, maar als er meerdere selecties waren maakte ik van de input een ASPrimaryKeySet. Ik wist dat dit de juiste class was omdat ik in OVInfo al meerdere malen had gezien wat voor input er afgehandeld kon worden.

Als de MessageList meer items had dan 10 zette ik de input op null. Binnen OVInfo betekent een null input dat de gebruiker heeft gekozen voor de optie ‘Alles’. Een lege lijst betekent een keuze voor ‘Geen’.

Nu ik mijn eigen Dialogable had kon ik deze gebruiken in de test. Ik gaf in de build methods een instance van TestDialogable mee en draaide nogmaals mijn test. De test slaagde, en ik was tevreden dat mijn Dialogable goed werkte.

Ik schreef een nog een aantal testcases voordat ik bedacht dat ik eigenlijk beter had kunnen testen met de ‘vraag alles’ testcase in het begin, in plaats van met een case die slechts één vraag stelde. Ik dacht niet dat het een verschil zou maken omdat eerdere tests met vragen al goed waren gegaan, maar ik besloot toch om de ‘vraag alles’ test prioriteit te geven. Ik draaide de test en hij faalde, maar niet op de Assertion zoals ik zou hebben verwacht. In plaats daarvan faalde de test al in de build method van de oude OVInfo. Er werd een query opgebouwd waarin werd vergeleken met een parameter van een verkeerd type.

Mijn eerste gedachte was dat er iets niet klopte met mijn TestDialogable, maar ik kon geen vreemde zaken vinden in die code. Daarom zette ik breakpoints in de code op het punt dat het misging en logde

in op de swing client om de testcase handmatig uit te voeren. Het breakpoint werd niet getriggerd en er traden geen exceptions op, dus alles leek te werken. Ik draaide nogmaals de test met het breakpoint nog aan. Nu triggerde het breakpoint wel en trad dezelfde exception op als eerst. Er zat dus toch iets fout in mijn TestDialogable, maar wat? Om dat uit te vinden besloot ik een breakpoint in de Dialogable te zetten om te kijken wat ik doorkreeg. Ik kwam erachter dat het grootste deel van de vragen die gesteld werden een ander type hadden MessageList. Het grootste deel was in plaats daarvan een instance van MessageServerQueryList.

De test faalde omdat ik geen code had geschreven die kon omgaan met andere types dan MessageList. Hierdoor werd de input nooit gereset en kreeg Tim het antwoord op een verkeerde vraag door.

Ik schreef een nieuwe conditie waarin ik zou kunnen omgaan met de MessageServerQueryList. Voordat ik daarmee aan de slag ging besloot ik eerst om de input op null te zetten aan het begin van de method, nog voordat ik naar de inhoud van de Message ging kijken. Hierdoor zou de fout die ik had veroorzaakt niet meer optreden.

Ik moest nu een manier vinden om in de MessageServerQueryList een keuze te maken. Dit was lastig, omdat deze class geen handige method had om aan de keuzes te komen. Ik zette een breakpoint in de method en draaide de test om op die manier inzicht te krijgen in het object. Het bevatte een label dat enkel beschreef om welke soort objecten de keuze ging. Ik dacht dat ik misschien hiermee een query zou kunnen samenstellen om zelf objecten in te lezen uit te database en op die manier de lijst zou simuleren, maar dan zou ik geen rekening kunnen houden met de filters en rechten die een gebruiker had.

Ik zag ook dat de class een ListProducer object bevatte. Ik hoopte dat dit object een List zou kunnen leveren met daarin de keuzes, maar op het eerste gezicht leek dit niet het geval te zijn. Omdat ik aan deze kant van de code niks kon vinden besloot ik te kijken naar het moment dat de Message objecten worden aangemaakt. Dit gebeurde in de ask methods van OVTableInfo.

De ListProducer in de ask methods kreeg een query mee op het moment dat de MessageServerQueryList werd aangemaakt. Dit betekende dat er een manier moest zijn om die query te draaien binnen de Dialogable, waardoor ik een lijst met keuzes zou krijgen. Ik kwam erachter dat het vrij eenvoudig was om in de ListProducer de query aan te roepen. Gelukkig voor mij gaf de query een gewone List terug, waardoor ik op dat moment dezelfde selectiecriteria kon toepassen die ik eerder al had gebruikt bij de MessageList.

Ik besloot ook de ‘meer dan 10 opties betekent alles’ conditie af te schaffen. OVInfo draait om het verwerken van selecties, dus is het een beter idee om altijd een selectie te hebben. Als er meer dan tien keuzes waren koos ik nu gewoon alle oneven keuzes.

Nu ik deze reparatie had doorgevoerd draaide ik de test opnieuw, en dit keer slaagde deze.

12.3 Evaluatie

Net als in de vorige sprint kwam ik in deze sprint door een aanname in de problemen. Ik nam aan dat de Message altijd van het type MessageList zou zijn, terwijl er niks was dat daarop wees. Ik heb daarna veel tijd besteed aan het vinden waar het probleem zat en zelfs toen ik dat wist moest ik nog een manier vinden om het op te lossen. Afgezien van die situatie vind ik echter dat de sprint goed is verlopen. Het

schrijven van de tests zelf nam veel tijd in beslag, maar dat kwam door de hoeveelheid tests en niet door hun complexiteit. De methods die ik schreef om duplicatie te voorkomen zorgde dat het invoeren van een testcase een pijnloos proces was.

13. Evaluatie

In dit hoofdstuk bespreek ik mijn ervaringen met het project als geheel. Ook zal ik hier doornemen wat ik heb geleerd en hoe ik dit in de toekomst in de praktijk zal brengen.

13.1 Terugblik

Tijdens de sprints heb ik al aangegeven hoe ik vond dat de werkzaamheden gingen. Kijkend naar het hele project denk ik dat er nog genoeg ruimte over is voor verbetering, omdat ik vaak tijdens het uitvoeren van de werkzaamheden fouten maakte die ik wellicht niet had gemaakt als ik beter naar de code had gekeken. Aan de andere kant is achteraf kijken altijd makkelijk, en de codebase waarin ik heb gewerkt was zowel oud als complex. Daarom vind ik het niet vreemd dat ik in een aantal gevallen niet meteen de juiste beslissing heb genomen.

Dat gezegd hebbende zijn er ook zeker momenten geweest waarop ik aannames maakte die eigenlijk op niets gebaseerd waren. Zulke aannames zijn zelfs in een eenvoudigere codebase gevaarlijk en ik vind dat ik beter had moeten weten dan er maar op te vertrouwen dat mijn aanname klopte.

Afgezien van de soms aparte beslissingen die ik heb genomen vind ik wel dat het traject goed is verlopen. De feedback die ik gedurende het proces heb gekregen van Edwin heeft me erg geholpen om mijn gedachten te focussen ik denk dat ik het project zelf tot een goed einde heb weten te brengen.

13.2 Geleerde lessen

De belangrijkste les die ik heb geleerd is dat ik goed het overzicht moete bewaren van waar ik mee bezig ben. Soms begin ik te diep te graven in code die me op dat moment niet verder zou kunnen brengen, terwijl een hoger abstractieniveau dat misschien wel zou kunnen.

Verder heb ik opnieuw geleerd dat aannames leiden tot fouten en dat ik voorzichtig moet zijn als ik iets aanneem. Ik moet er altijd rekening meer houden dat ik wellicht niet goed bezig ben, en ik moet als ik twijfel aan mijn oplossing sneller hulp vragen om te voorkomen dat ik later een grote reparatieslag moet uitvoeren.

14. Beroepstaken

In dit hoofdstuk bespreek ik de beroepstaken waaraan ik in dit project heb gewerkt. Hierbij zal ik refereren aan de sprints en de bijlagen.

14.1 Ontwerpen systeemdeel

Hiermee heb ik me vooral in het begin van het traject veelvuldig bijgehouden. Verder uitgewerkte diagrammen dan degene in Hoofdstuk 7 vindt u in de bijlagen. Hier vindt u ook een sequence diagram van buildMultipleRoots.

14.2 Bouwen systeemdeel

Nadat de ontwerpen waren gebouwd, moest ik me bezighouden met het bouwen van die ontwerpen. Hierbij moest ik ook rekening houden met eventuele extra functionaliteit die gemaakt zou moeten worden.

14.3 Initiëren en plannen van het testproces

Het initiëren van het testproces heeft net als het ontwerpen veelal in de vroege fase van het project plaatsgevonden. Ik heb hieraan gewerkt door vast te stellen welke tests er uitgevoerd zouden moeten worden om er zeker van te zijn dat OVInfo in de nieuwe situatie dezelfde werking zou behouden als in de oude situatie. De tests die hiervoor zijn gemaakt worden beschreven in Hoofdstuk 12.

14.4 Uitvoeren van en rapporteren over het testproces

Het uitvoeren van het testproces heb ik gedaan door de integratietests te schrijven in de laatste sprint van het proces. Het handmatige testen heb ik niet zelf uitgevoerd omdat dit de taak is van de consultants.

15. Glossary

In dit hoofdstuk vindt u een woordenlijst van de woorden die in het verslag cursief gedrukt waren.

God class	Een class die een groot aantal verantwoordelijkheden heeft en daardoor te belangrijk is binnen een applicatie.
Legacy code	Code die gebouwd is voordat de huidige standaarden en technieken bestonden. Deze code is vaak niet getest omdat er door de jaren heen toevoegingen aan zijn gemaakt.
Core business logic	Bedrijfsprocessen die kritisch zijn voor het goed functioneren van een product. Dit is de belangrijkste logica in een programma omdat het direct invloed op de business heeft.
Unittests	Tests die elke methode afzonderlijk testen. Het doel ervan is zeker zijn dat een method doet wat hij moet doen.
Integratietests	Deze tests testen de code dieper dan een unittest. Bij een integratietest kan er bijvoorbeeld gebruikgemaakt worden van een database, of code uit andere classes.
Separation of concerns	De mate waarin een class verantwoordelijk is voor zaken waar hij verantwoordelijk voor moet zijn. Bij een slechte separation of concerns heeft een class meer verantwoordelijkheden dan hij hoort te hebben.
REST	Staat voor REpresentational State Transfer. Dit is een architectuurstijl in de softwareontwikkeling die zich kenmerkt door een focus op performance, schaalbaarheid, onderhoudbaarheid en uitbreidbaarheid.
Quick fix	Een quick fix is een stukje code dat aan een method wordt toegevoegd om een specifieke situatie te verhelpen. Een quick fix hoeft geen hack te zijn, maar een groot aantal quick fixes vermindert de leesbaarheid en onderhoudbaarheid van de code.
Hack	Een hack is een stuk code dat aan een class wordt toegevoegd om een functionaliteit in te bouwen of een probleem op te lossen. Hacks lijken op quick fixes, maar een hack gebruikt altijd code die niet clean is.
Side effect	Dit houdt in dat een method een variabele of ebstand aanpast buiten zijn eigen scope. Side effects kunnen wenselijk zijn, maar vaak zijn ze dat niet.

Monster method	Een monster method is een method die een groot aantal regels code bevat en meer dan één taak uitvoert. Deze methods zijn vaak niet unittestbaar omdat niet duidelijk is wat ze horen te doen, of omdat ze veel parameters hebben.
Black box	Dit houdt in dat de innerlijke werking van een class of method niet bekend is, maar dat er wel werkzaamheden mee of op worden verricht.
DRY	Don't Repeat Yourself. Een principe dat bedoeld is om onderhoudbaarheid hoog te houden door code niet meerdere malen voluit te schrijven. Als dit toch gebeurt, zou het eigenlijk een method moeten zijn.
Exception	De naam van een foutmelding in Java. Omdat een fout een uitzondering vormt op de normale werking van de code wordt dit een exception genoemd.
Mock	Een object dat gebruikt wordt in unittests. Een mock geeft bij de aanroep van een bepaalde method een vaste waarde terug, zodat een method die daar gebruik van maakt in isolatie kan worden getest.
Test harness	Een benaming voor het testbaar maken van een method. Het houdt in dat de method goed aangeroepen kan worden in een geïsoleerde omgeving.
Inheritance	Als een class een meer gespecialiseerde class onder zich heeft die de functionaliteit van de eerste class ook bezit, dan erft de tweede class van de eerste. Een class waarvan wordt geërfd kan zelf ook bruikbaar zijn, maar hij kan ook abstract zijn, waarbij alleen subclasses ervan geïnstantieerd kunnen worden.
Hard-coded	Dit wil zeggen dat een variabele die normaal extern wordt aangeleverd direct in de code wordt gezet om te forceren dat deze een bepaalde waarde heeft. Dit hoort in productiecode niet voor te komen.
Stateless	Dit houdt in dat een class geen interne opslag heeft van de variabelen waarmee deze werkt. Een class die alleen maar werk op aangeleverde data verricht is vaak stateless.
Primitive	Primitive types in Java zijn variabelen die geen objecten zijn. Types zoals int, double, en boolean zijn primitive types. Een kenmerk van primitives in Java is ook dat ze niet null kunnen zijn.

DTO	Data Transfer Object. Deze objecten bevatten data, maar voeren hier geen werkzaamheden op uit. Ze worden meestal gebruikt om data van en naar de server te sturen.
Dependency injection	Dit is een term die aangeeft dat een framework ervoor zorgt dat bepaalde variabelen geïnstantieerd worden en dat het niet expliciet door de programmeur wordt gedaan.
Disc I/O	Het aantal keer dat de harde schijf moet worden gelezen voor een operatie. Een groot aantal databaseoperaties leidt tot een hoog disc I/O. Omdat lezen van de harde schijf langzaam is, is een laag disc I/O gewenst met het oog op performance.
JSON	JavaScript Object Notation. Een veelgebruikte manier om objecten van de server naar de client te sturen. Het object wordt hierbij geserialiseerd tot human-readable tekst.

Tabel 2: Verklarende woordenlijst

Bijlagen

In de pagina's hieronder vindt u de bijlagen. De tabel hieronder geeft de paginanummers van ieder document:

PVA	**54**
Beschrijving OVInfo	60
Voorstel nieuwe OVInfo	67
Verantwoordelijkheden en Dependencies OVInfo	70
Voorstel nieuwe classes OVInfo	73
Sequence diagram	74
Class diagram	75

Opdrachtomschrijving

In dit hoofdstuk beschrijf ik het bedrijf, de probleemstelling en de opdracht.

Bedrijf

Aenova is een bedrijf dat zich richt op het samenbrengen van de manager en de werknemer. De laatste jaren is flexwerken steeds meer in opkomst, waardoor medewerkers niet altijd even gemakkelijk hun uren kunnen verantwoorden.

Het antwoord dat Aenova hierop biedt is TimEnterprise. Dit softwarepakket stelt werknemers in staat uren te verantwoorden vanaf elke locatie en geeft zo managers inzicht in de voortgang van het project zonder dat er extensief micro-management nodig is. Naast het schrijven van uren is TimEnterprise ook geschikt om te koppelen met veelgebruikte backoffice systemen zodat verlofregistraties uit bijvoorbeeld ADP eenvoudig kunnen worden overgenomen in Tim, zodat alle informatie altijd bij de hand is.

Probleemstelling

in TimEnterprise bevinden zich oude classes die niet binnen een moderne softwareomgeving passen. Met het oog op de toekomst richt Aenova zich steeds meer op het aanbieden van TimEnterprise als een webservice. Op dit moment wordt dat tegengehouden door een class, genaamd OVInfo, die zich bezighoudt met het uitzoeken van rechten en selecties voor onder andere overzichten, maar die ook intern wordt gebruikt om bijvoorbeeld weekbriefjes en dagplanningen op te zetten.

Om de gewenste informatie te kunnen tonen worden er door deze class soms vragen gesteld aan de gebruiker door middel van zogenaamde dialogables. Deze dialogables blokkeren de server totdat de gebruiker input heeft gegeven. Dit is een model dat in een webomgeving niet bruikbaar is, en daarom moet OVInfo herschreven worden. Het is daarbij de bedoeling dat er een geheel nieuwe implementatie komt, die ook de oude manier van werken nog ondersteunt voor de desktop client.

Doelstelling opdracht

De opdracht is het vastleggen van de verantwoordelijkheden en dependencies van OVInfo, en gebaseerd daarop met een nieuw ontwerp te komen waarin de separation of concerns goed is geregeld en dit ontwerp vervolgens te bouwen. Daarnaast moet er voor de nieuwe webclient ook een REST service gebouwd worden. Hierbij is het van belang dat de afhankelijkheid van dialogables verdwijnt, maar dat er geen bestaande functionaliteit verloren gaat. Daarnaast is het zeer belangrijk dat het nieuwe ontwerp goed testbaar is, zodat toekomstige uitbreidingen veilig ingebouwd kunnen worden.

Scope van het project

In dit hoofdstuk wordt de scope van het project besproken.

Binnen de scope

- Vastleggen van dependencies en verantwoordelijkheden
- Ontwerp maken voor nieuwe situatie
- Bouwen nieuwe situatie
- Schrijven tests voor nieuwe situatie
- Bouwen serverside REST service

Buiten de scope

- Volledig handmatig testen
- Front-end webclient bouwen

Risicofactoren

In dit hoofdstuk worden de risico's van dit project besproken

Verlies van oorspronkelijke functionaliteit

Risico: Bij het aanpassen van de bestaande code bestaat de kans dat huidige functionaliteit verloren gaat. Als dit gebeurt, werkt TimEnterprise niet meer naar behoren. Dit zou voor het bedrijf een grote impact hebben.

Oplossing: Huidige code behouden en gecontroleerd uitfaseren van de bestaande code. Hiermee is er ruimte om eerst met integratietests te controleren of de nieuwe code correct werkt.

Projectorganisatie

In dit hoofdstuk wordt projectorganisatie beschreven. Eerst wordt de programmeertaal besproken, daarna de ontwikkelmethode.

Programmeertaal

De taal waarin dit project wordt uitgevoerd is Java. In 1999 werd Tim hiernaar geport vanuit C en sindsdien is de programmeertaal niet meer veranderd. De applicatie wordt gebouwd met Java 7 in plaats van 8, omdat er oude onderdelen van Tim zijn die methods gebruiken die in Java 8 niet meer bestaan.

Ontwikkelmethode

Bij het volbrengen van deze opdracht heb ik gebruik gemaakt van de ontwikkelmethode Scrum. Dit lijkt wellicht op het eerste gezicht vreemd voor een opdracht waarbij voor het grootste deel geen nieuwe functionaliteit wordt gemaakt, maar omdat ik zou meedraaien in reguliere sprints tijdens dit project en de taken die ik zou uitvoeren goed in stories verdeeld konden worden zou ik toch deze methode gebruiken.

De sprintduur was twee weken. Twee weken wordt door het bedrijf gezien als een goede lengte omdat het voldoende tijd biedt functionaliteit af te krijgen, maar ook nog kort genoeg is dat er tijdig feedback gegeven kan worden door de Product Owner.

Voorgaande aan de sprint vindt er een sprintplanning plaats waarin wordt bepaald welke stories op de sprint zouden komen. Het maximaal aantal te behalen punten wordt vastgesteld door te kijken naar vorige sprint en aanwezigheid van de developers.

Tijdens de sprint is er elke dag een daily standup, waarbij elke developer vertelt wat hij die dag heeft gedaan. Hierbij is Productmanagement ook aanwezig om eventuele vragen te kunnen stellen en op de hoogte te blijven van de voortgang. Tijdens de daily standup wordt ook gekeken naar de burndown chart.

De sprint wordt afgesloten met een retrospective, waarin wordt besproken waarom bepaalde stories niet af zijn, wat er fout ging, en wat er goed ging. Ook wordt hier al een schatting gemaakt voor het aantal punten voor de volgende sprint.

De Product Owner en Scrum Master zijn dezelfde persoon in dit geval. Beide rollen worden vervuld door Eva van der Last, directeur van Aenova.

Mijlpaalproducten

In dit hoofdstuk beschrijf ik de mijlpaalproducten die tijdens dit project worden opgeleverd.

- PVA
- Beschrijving dependencies en verantwoordelijkheden OVInfo
- Voorstel nieuw ontwerp OVInfo
- Nieuwe code
- Integratietests

OVInfo

OVInfo - rights and selections for Budget Grids

The OVInfo class handles the rights and selections for Abstract Matrix and the Budget Grid that is built upon that. Although Abstract Matrix has it's own mechanism for building queries based on the selections from the statics, the current implementation makes no use of this feature and instead uses the OVInfo class to do most of the handling. (Tim32 used to be slightly different in this respect but this had to change because there is no way to quickly asses if a Java object complies with a database query).

Currently, OVInfo handles all selections and rights management **except** the query part for the date period; the queries for the period are built by the BBPeriodElement. (For the time of day support, which is new, no queries are built whatsoever since that is hard to impossible and not that useful to begin with).

Main responsibilities

The OVinfo class has the following responsibilities:

- For all dimensions possible: determine which values are allowed and which are not by:
  - Determine the rights of the user for the current situation
  - Determine the required selection (within the boundaries of the user's rights)
  - If needed: Ask questions to establish the above mentioned selection
  - Filter out any values that should never make it to any report (such as Tim's model persons) If any dimension is hierarchical, find out if the selected items share a single common root
- Build a query stack for all the data sources that are needed to fill the matrix (with the exception of the date period test)
- Do all preparations for the date period selection (asking questions, validating the results with any axis)

The OVInfo class retains the information it gathers in order for the budgetmatrix to check if data added to the matrix is actually allowed, to determine the values for the statics (which are the single roots for the determined selections). Also, if a user wishes to drill down by using the buttons in the budget grid, OVInfo will handle the question based on the previous selection.

Basic approach

The basic structure of any of any report or budget grid is stored in the database using an **OverzichtObject** with corresponding **OverzichtSelectieObject**s. Indeed, also other screens, such as the Time Sheet use an internal OverzichtObject as a description. OVInfo uses this as input, although it is only interested in the fields related to the selection.

The OVInfo class uses an enumeration called **OIType** to describe all the dimensions that OVInfo has data for. The basic behaviour is to loop over all OIType values and perform the same process for each value. Each OIType value has many properties to help generalize this process. The order of the OIType values determines the order in which the data is processed. This is relevant, since for instance the selection for the Departments might limit the options for the selection of Persons.

For each value of the OIType enumeration an **OVTableInfo** object is built, that holds information for that type. This information consists mainly of a series of Lists with ID's. The most important list in it is the SelectionList, which is the end result of the selection process. This list might either be a list of items to include in the report, or a list of items to exclude (depending on the isReverse() boolean. A null list implies all, an empty list implies none.

The process is complete if all the OVTableInfo's are built. From that information the all the queries are constructed and placed on a QueryStack.

Processing steps

1. First of all the date period is determined (outside of the OIType loop), since all the other work (including other preparations) depend on it. For instance, if you report on the past you want the department members for that period, not for today. This step might involve asking a question to the user.
2. Perspective is determined. Usually a report is created based on your own rights. However, an application administrator may want to impersonate someone else in order to see what that person would see if he or she uses the same report. A question may be asked who the administrator wants to impersonate.
3. Some simple and basic lists are set for specific tables, such as the departments the report owner is member of, his preferred department, and items that should never make it in any report (model persons, archived products)
4. If this report is actually a report definition from the database, the list of selection specifications is read from the database. These selection records might hold specific values, the wish for a question or a 'current' value (me, my department, ...). These requests are not processed yet but simply recorded in OVTableInfo.
5. For each table that has corresponding rights, a rights list is determined. For instance, a department head may generate reports for his (sub)departments, but not any others. Therefore a list of allowed departments is build. Similarly, if you are a project leader you may report on all things related that project, so a list of (sub)products is built.
6. If you have more than one set of rights (above: both department and product related rights) a question is asked which rights to use at this time, because using both would result in a weird mix of products and departments. All but the chosen rights list are cleared, so only one set of rights remains.
7. For each dimension, the end selection is built:
    a. If the selection is based on a question, a dialog is produced to ask the question.
    b. If the selection is hierarchical, children are added to the selection
    c. Rights lists and selection lists are combined to produce a single result.
    d. The tree is examined to see how many roots are in the selected items. Are all items part of the same tree or not?
8. Sometimes, if multiple roots for one axis are not allowed (you cannot have a time sheet for two persons at the same time) the selection is stripped down to only the first root (a single parent).

OVTableInfo

The following lists are built and stored in an OVTableInfo, in order to come to the end result:

List	Purpose
RightsList	ID's of all the items you have rights for
BadList	ID's of all items that must not be included in the end result (e.g. model persons)

AskList	The ID's of all the items that were answered in response to a question
LidList	The ID's of items that you are a member of (e.g. departments you are a member of)
PrefList	The ID's of any items that might be used as a default, such as yourself, the preferred department, etc.
SelectionList	The list of all the items that are included or excluded for the the selection (depends on the boolean isReverse())
RevSelectionList	If present, the reverse of the Selection list - its complement. All the ID's that are not on the selection list.

RootList	The smallest list ID's that would describe the selection if one would include all their children

The OVTableInfo data is still pretty raw data. In fact, the end result can either be a SelectionList, a RevSelectionList or a BadList, and the isReverse() boolean should be respected as well with regard to the first two:

- In general, the SelectionList should represent the end result, and the isReverse() boolean signals if the selection describes items to be included or items to be excluded.
- OVInfo may store the inverse of the SelectionList into a RevSelectionList (usually if the number of items in the list is expected to be significantly smaller). This is done to reduce the size of IN() or NOT IN() queries, while keeping the original list intact for future reference.
- If no SelectionList or RevSelectionList is present there still may be a BadList to consider (If there is a SelectionList or RevSelectionList, the BadList is already incorporated in that result and can safely be ignored). The reason for this is that the BadList should be completely transparent to the user. The user shouldn't be aware that something is left out and must think that everything is included, even though this is really not the case.

To conclude; only one list should ever be used as the end result of the selection. If a SelectionList or a RevSelection list is present you may use either one. The isReverse() boolean will then describe how these lists are to be interpreted. If neither a SelectionList or a RevSelection list is present, the BadList is the list of items that should be excluded.

Rights

There are 3 rightsfields in each table info, which are fed into the rightscache to determine valid refs for each dimension (rightslist). The third rightsfield is only used and implemented for the for persons dimension. Specifically to combine the queries for request accept and attendance screen in case of the Servoy webclient (see: Source).

Special behaviour

Questions

In general, separate dimensions are treated separate. That means that for instance a department selection does NOT imply a person selection with the department members. Instead, the registrations are examined whether they hold the right department reference, regardless if the person was supposed to be a department member. Using a person selection is dangerous, since the department memberships might no longer be accurate.

There are however several reasons to convert a selection for one dimension into another. One is when a selection is based on a question to the user. Lets say the report has two selections: one selection on the department, which should be the current department (the department the person generation the report is a member of). The second question is a person selection by means of a question. In order to make sure the user is not presented with a list of all persons but only the persons for the current department, an extra person query is pushed for the purpose of the question.

selection conversions

There are other reasons to convert one selection type into another. For instance, when reporting on presence there is no department data available in the presence records. However, it is obvious that the user expects the presence to be filtered for members of the department. For presence records the department selection is therefore converted to a person selection (for that table only).

Other examples are registration and fiat date reports. Both registration dates and fiat dates are not department specific. However, for the same reason as above both department and employee type selections are converted to

person selections to suit the user's expectations.

Architectuur OVInfo (SHOULD-BE)

Huidige situatie

Op dit moment maakt OVInfo gebruik van Dialogables om aan de gebruiker informatie te vragen. Deze Dialogables zorgen ervoor dat de code wordt 'geblokkeerd' terwijl de gebruiker zijn input levert. Het gevolg hiervan is dat OVInf in zijn huidige vorm niet bruikbaar is voor web, omdat de vragen die door de code gesteld worden daar niet wenselijk zijn.

Verder is het zo dat OVInfo op dit moment slecht onderhoudbaar is vanwege het grote aantal harde dependencies en 'quick fixes' voor oude problemen. Dit geeft problemen als men de code wil unit testen, omdat er geen goede separation of concerns is, en alle fixes voor side effects kunnen zorgen. Omdat de code niet testbaar is, is het ook niet mogelijk de klasse in delen te refactoren.

Aanpak

De volgende zaken moeten hoe dan ook gebeuren om OVInfo te refactoren:

- Vastleggen welke dependencies OVInfo op dit moment bevat
- Vastleggen welke verantwoordelijkheden OVInfo op dit moment heeft
- De grote loop in OVInfo moet worden opgeknipt

Het opknippen van de loop klinkt wellicht al te gedetailleerd, maar dit is niet het geval. De Dialogables die op dit moment de kern van het probleem vormen zitten namelijk in deze loop. Om deze er op een goede manier uit te krijgen is het hoe dan ook nodig de bestaande loop op te knippen.

Mogelijke oplossingen

Een mogelijke oplossing voor het probleem is het gebruik van een parallelle implementatie. Hierbij wordt de vervanger van OVInfo gebouwd terwijl de oude klasse nog steeds werkzaam is. Stap voor stap worden dan de verantwoordelijkheden van OVInfo overgenomen door de nieuwe code. Een belangrijk hulpmiddel hierbij is het schrijven van een integratietest. Hierdoor wordt het mogelijk om overzichtelijk alle mogelijke cases die in OVInfo aan bod kunnen komen vast te leggen, en het zorgt ervoor dat de tester kan kiezen welke implementatie van de functionaliteit wordt gebruikt, oud of nieuw.

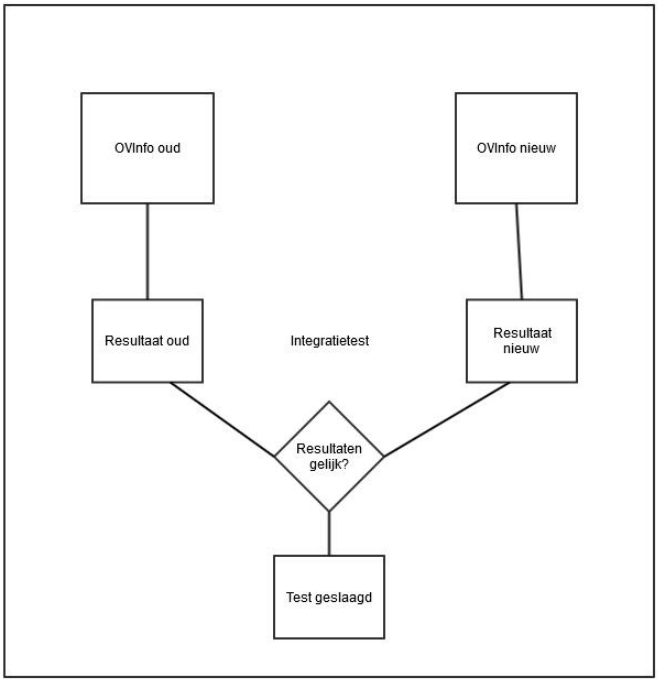
In dit model is het noodzakelijk dat de test aansluit op de oude situatie. Wanneer de nieuwe situatie vervolgens op de de test wordt 'aangesloten' moeten alle testcases blijven slagen. Hiermee wordt gegarandeerd dat de nieuwe code exact dezelfde functionaliteit biedt als de oude, en eventuele menselijke fouten bij het testen worden voorkomen. Verder is deze oplossing minder arbeidsintensief, omdat niet na elke wijziging consultants opnieuw alle scenario's af hoeven te lopen. In plaats daarvan kan men gewoon de integratietest draaien.

Risico van niet parallel implementeren

Als ervoor gekozen wordt OVInfo in zijn geheel om te bouwen zonder parallelle implementatie is er een grote kans dat Tim daardoor niet meer zou werken. Een aantal schermen die niet direct met OVInfo werken maken er namelijk wel gebruik van. De volgende schermen zouden waarschijnlijk niet meer werken als OVInfo in zijn geheel wordt aangepast:

- Weekbriefje
- Onkostenbriefje
- Kengetallen
- Dagplanning.

Hieronder de schematische weergave van het parallel testen van OVInfo.



Verantwoordelijkheden en dependencies OVInfo

Verantwoordelijkheden

Om een goede separation of concerns te kunnen waarborgen hoort een class in principe maar een enkele verantwoordelijkheid te hebben. In het geval van OVInfo hoort dat de volgende te zijn:

- Opbouwen queries voor overzichten gebaseerd op gebruikersinput

Op dit moment gebruiken andere classes OVInfo echter ook voor andere doeleinden. Dit zijn verantwoordelijkheden die OVInfo nu ook op zich neemt.

- Bepalen welke vragen er gesteld moeten worden in schermen die geen overzicht genereren.
- Competenties ophalen
- Bepalen of een bepaald OIType geselecteerd
- is Selectielijst ophalen

De wens is dat in de toekomst de verantwoordelijkheid voor het bouwen van queries nog wel bij OVInfo ligt, maar dat deze ook los van overzichten gebruikt kunnen worden.

Dependencies

OVInfo gebruikt veel andere classes om te kunnen functioneren. Veel daarvan zijn zogenaamde utility classes die door heel Tim veel worden gebruikt. Er zijn echter ook andere classes die worden gebruikt.

De volgende classes worden door OVInfo als attribuut opgeslagen:

- PersoonObject
- TimJPerspFlags
- DialogableState

Naast attributen heeft OVInfo ook nog dependencies op andere classes binnen zijn eigen package, die binnen methodes worden aangemaakt en gebruikt.

De belangrijkste classes van binnen de package die OVInfo gebruikt:

- OIType
- OVTableInfo
- TimJRightsHelper

Van de classes OvInfoPeriodeSelector en RegFiatComparator worden geen variabelen aangemaakt, ze worden enkel meegegeven als parameter aan andere methods, door in de aanroep de constructor te gebruiken.

Naast dependencies op objecten binnen de package, worden er ook veel classes uit andere packages aangeroepen.

Omdat een lijst met alle losse classes 70 items zou tellen, noem ik van packages waar meerdere classes uit gebruikt worden enkel de package.

- QBCompleteQuery
- QueryBuilder
- QueryBuilderStack
- FieldInfoPersistentOnlyFilter
- as.clientserver.objects
- as.timj.clientserver.enums

- as.timj.clientserver.objects
- GlobalTimJEnv
- TimJUserEnv
- ProductrechtenHome
- TimJHomeOfHomes

Dependencies in de nieuwe situatie

Niet alle dependencies die in de paragraaf hierboven genoemd worden zijn onwenselijk. Met name de utility classes die door de hele applicatie heen gebruikt worden mogen ook in de nieuwe situatie behouden blijven. Ook de QueryBuilder en gerelateerde dependencies zullen in de nieuwe situatie aanwezig blijven, omdat ze direct te maken hebben met de verantwoordelijkheid van OVInfo.

Dependencies die zullen verdwijnen zijn degenen die ervoor zorgen dat OVInfo minder generaal inzetbaar is. De volgende dependencies zullen dus in elk geval verdwijnen:

- OverzichtObject
- OverzichtSelectieObject
- Dialogable

Hoewel de input van de gebruiker nog steeds gehonoreerd dient te worden, moet dit op een andere wijze worden opgepakt. Er moet daarbij wel worden gezorgd dat de data intern zinnig blijft. Als bijvoorbeeld slechts een enkele afdeling wordt opgegeven, moeten niet ook personen die daar niet bijhoren worden meegenomen. Het Dialogable object zal echter niet meer worden gebruikt.

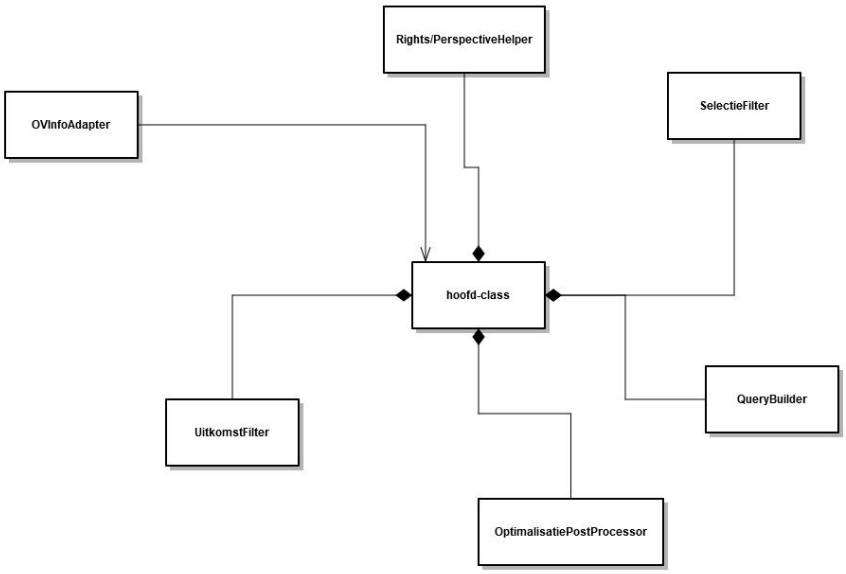
Mogelijk moeten ook de dependencies op TimJHomeOfHomes en GlobalTimJEnv worden verwijderd. Dat is op dit punt echter nog niet volledig duidelijk.

Nieuwe classes OVInfo

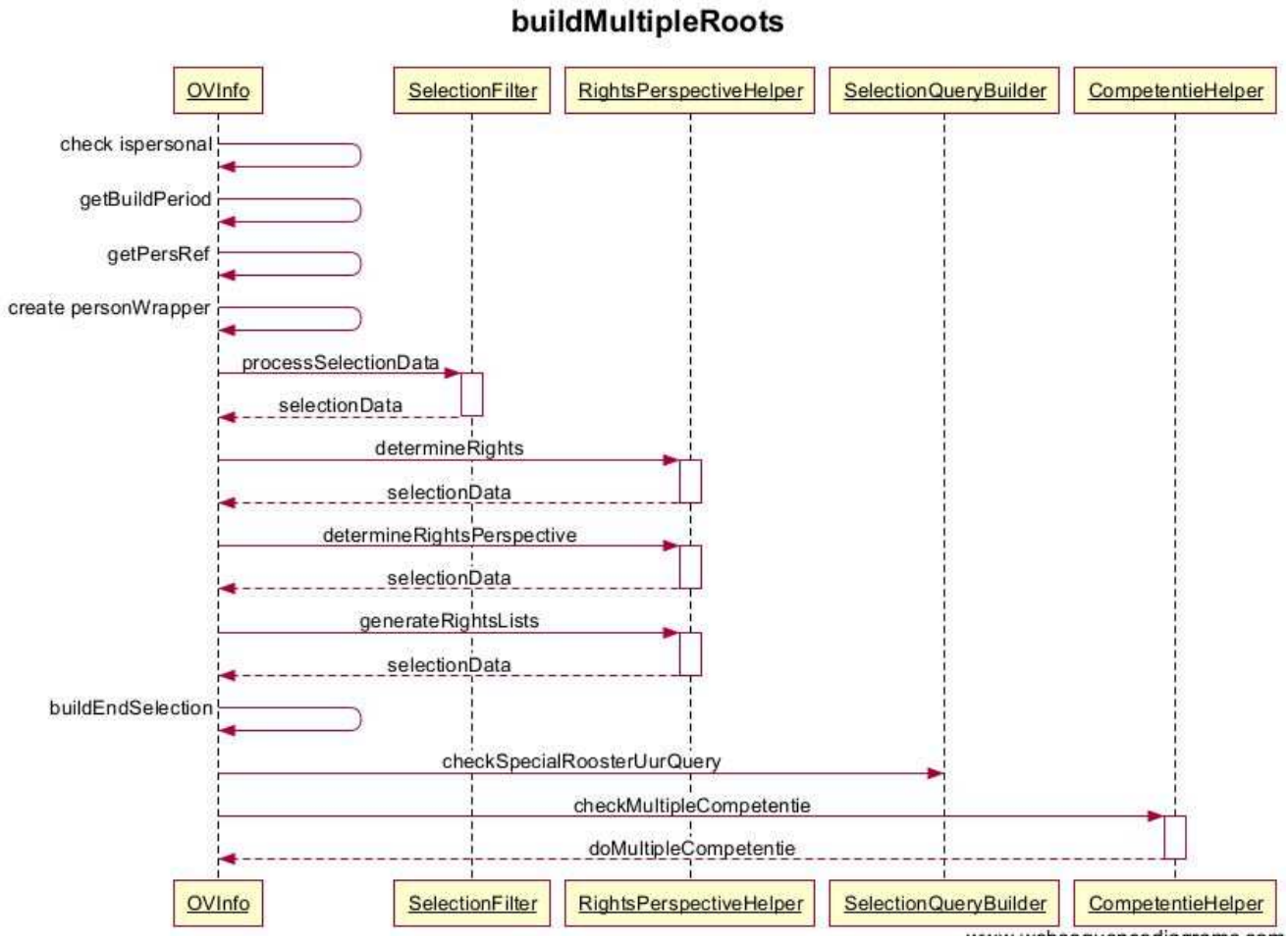
In dit document wordt een overzicht gegeven van de nieuwe classes van OVInfo, en een diagram getoond waarin de samenhang tussen deze classes wordt weergegeven.

Tot zover zijn de volgende classes vastgesteld:

- 'hoofd'-class, naam nog te bepalen. Deze class bevat de iteratie en houdt state bij, waar de andere classes gebruik van maken
- Rights/PerspectiveHelper (mogelijk twee classes) bepaalt met welke rechten en perspectief het overzicht gebouwd wordt
- SelectieFilter, deze class zorgt dat, wanneer bijvoorbeeld een afdeling is gekozen, alleen de bijbehorden personen kunnen worden geselecteerd. Wat al bekend is wordt bijgehouden in de hoofd-class
- QueryBuilder, bouwt de daadwerkelijke queries
- OptimalisatiePostProcessor, optimaliseert de opgebouwde OVTableInfo
- UitkomstFilter, werkt met rootlists om hierarchie te kunnen opbouwen
- OVInfoAdapter, zorgt ervoor dat Swing kan werken met de nieuwe implementatie en biedt de public methods van de huidige OVInfo aan
- ContextInformation, nodig om de afhankelijkheid van overzichtobjecten te verwijderen (hoeft mogelijk geen class te zijn, kan wellicht in losse parameters)



Sequence diagram buildMultipleRoots



Class diagram

Vanwege de omvang van dit diagram is dit niet opgenomen op papier. Het is echter wel te vinden op de bijgeleverde CD.