MOBO Requests

Afstudeerverslag

REDHOTMINUTE
sites | stores | apps

Graaf Reinaldweg 22
NL-4176 LX Waardenburg (Tuil)

t: +31 (0)418 51 00 68
f: +31 (0)418 51 00 74
i: www.redhotminute.com

Student
Ferry Gruiters
1581942
Information Engineering
Voltijd

Bedrijfsbegeleider
Nico Lubbers

Docentbegeleider
Marco Dumont

Tuil – 6/2/2014

Voorwoord

Voor u ligt mijn afstudeerverslag ‘MOBO Requests’, dat na ongeveer vier maanden stage lopen tot stand is gekomen. Dit verslag is geschreven om de vernieuwingen binnen het systeem MOBO Requests toe te lichten en om de resultaten van de onderzoeken weer te geven.

In deze, soms lastige maar zeker leerzame, periode heb ik veel (technische) kennis opgedaan dat ik in mijn verdere loopbaan zeker kan gebruiken. Hopelijk heeft mijn bijdrage ervoor gezorgd dat MOBO Requests op korte termijn daadwerkelijk kan worden geïmplementeerd.

Het resultaat van deze mooie opdracht had er niet geweest zonder de medewerking van de collega’s en de afstudeerbegeleider. Bij dezen wil ik ten eerste Marco Dumont bedanken voor de bijdrage die hij geleverd heeft aan de totstandkoming van dit verslag. Al mijn vragen zijn altijd snel en met nuttige informatie beantwoord. Dit heeft het eindverslag zeker ten goede gedaan.

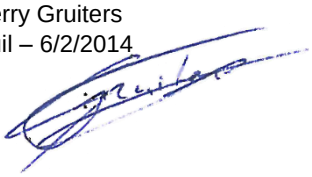
Ten tweede wil ik mijn begeleiders vanuit Redhotminute, Nico Lubbers en Stefan Sluijter, bedanken voor de ondersteuning die ik het gehele project van hen gekregen heb. Voor zowel technische vragen, vragen over de planning als vragen over algemeen werk kon ik bij hen terecht.

Naast de begeleiders vanuit Redhotminute wil ik ook de product owner van het project bedanken; Renso Bek. Dankzij de tweewekelijkse meeting is zijn visie voor het eindproduct steeds duidelijker geworden, waardoor ik altijd doelgericht heb kunnen werken. Ik ben blij met het vertrouwen dat me gegeven is om aan het product te mogen werken.

Ten slotte wil ik Redhotminute als bedrijf in geheel bedanken. Vanaf dag één ben ik met open armen ontvangen, waardoor ik elke dag met een goed gevoel aan mijn afstudeerproject heb kunnen werken.

Ik wens u als lezer veel plezier met het lezen van dit verslag.

Ferry Gruiters
Tuil – 6/2/2014



Managementsamenvatting

Opdracht en doelstelling

Redhotminute heeft in 2013 een systeem laten ontwikkelen waarmee data van verschillende kanalen ontvangen kan worden, bijvoorbeeld van een webformulier. De ontvangen data kan vervolgens door het systeem verwerkt worden om het daarna naar verschillende kanalen door te kunnen sturen, zoals bijvoorbeeld een CRM. Dit systeem, MOBO Requests genaamd, moet op diverse klanten van Redhotminute toegepast kunnen worden.

Tot op heden was het systeem niet uitgebreid genoeg waardoor het niet op iedere klant toegepast kan worden. Redhotminute heeft daarom de wens om het systeem dusdanig ver uit te breiden, dat het op iedere klant toepasbaar is. Hiervoor zullen de behoeften geïnventariseerd moeten worden om ze vervolgens te implementeren.

Naast het uitbreiden van het systeem zal er onderzocht moeten worden welke verbeterpunten er voor MOBO Requests te behalen zijn.

Aanpak

Om deze wens van Redhotminute te kunnen realiseren, is er gedurende het project volgens een bepaalde aanpak gewerkt. Allereerst is er onderzocht welke functionaliteiten er tot nu toe ontbraken en welke er van belang zijn om het systeem op klanten toe te kunnen passen. Dit onderzoek is gedaan door met verschillende medewerkers van Redhotminute interviews te houden met de vraag waar het systeem volgens hen aan moet voldoen. De bevindingen hiervan zijn vervolgens in kaart gebracht.

Nadat de functionele wensen in kaart gebracht waren, is er begonnen met het ontwerpen en realiseren van de functionaliteiten. Vanaf dit moment is er volgens scrum gewerkt. Elke twee weken is er met de product owner en de bedrijfsbegeleider overleg geweest om de resultaten van de afgelopen sprint te bespreken en om de komende sprint in te delen. Afhankelijk van de prioriteit en de bevindingen van de afgelopen sprint, is er bepaald wat er in de komende sprint gedaan moest worden.

Voor het ontwerpen van de nieuwe functionaliteiten is er getracht gebruik te maken van vergelijkbaar materiaal op het internet. Om de nieuwe functionaliteiten zo goed mogelijk, passend binnen de huidige programmatuur, te implementeren, is er geprobeerd om de nieuwe functionaliteiten zo veel mogelijk binnen de huidige oplossingen toe te passen. Om de werking van de bestaande en nieuwe functionaliteiten te waarborgen zijn er Unit Tests gebruikt en gemaakt, waardoor eventuele problemen snel gedetecteerd konden worden. Daarnaast is er intensief met verschillende scenario's getest, om het systeem zo generiek mogelijk te maken. Door de testscenario's steeds complexer te maken is elke functionaliteit steeds generieker opgezet.

Voor het overige onderzoek is er van literatuur gebruik gemaakt.

Resultaat

Uiteindelijk is MOBO Requests een stuk generieker opgeleverd dan dat het bij aanvang van de stageperiode was. Dankzij deze uitbreiding ondersteunt het systeem nu diverse lijst-scenario's, waardoor het systeem op veel meer scenario's en klanten toegepast kan worden. Uit het onderzoek naar de verbeterpunten is gebleken dat er nog veel aan MOBO Requests aangepast moet worden om het systeem te optimaliseren. Zo blijkt dat er nog een aantal tests uitgevoerd moeten worden, moet MOBO Requests de ontvangen data sneller kunnen verwerken en zullen er nog bepaalde functionaliteiten toegevoegd moeten worden om het systeem nog generieker op te zetten. Doordat het verwerken van de data nog niet snel genoeg gaat, kunnen er lange wachtrijen ontstaan.

Dankzij dit onderzoek is het duidelijk geworden dat het nog niet verstandig is om MOBO op klanten toe te passen, aangezien de prestaties niet gegarandeerd kunnen worden. Het is ten slotte zeer onwenselijk om door een onstabiel systeem klanten te verliezen.

Het is jammer om te moeten concluderen dat het systeem, ondanks dat het uitgebreider geworden is, op dit moment nog niet op klanten toegepast kan worden. Het had mooi geweest als het systeem na de stageperiode ingezet had kunnen worden. Desondanks is het een belangrijke vinding, omdat het nu voor Redhotminute duidelijk is waar nog met zekerheid aan gewerkt moet worden.

Inhoudsopgave

1. Inleiding

1.1. Aanleiding

MOBO Requests, ook wel MOBO genoemd, is een applicatie dat voor en door Redhotminute ontwikkeld is, wat als vervanging voor bestaande ESB-systemen gebruikt zal gaan worden. In het schooljaar 2012-2013 hebben twee studenten de basis van de applicatie in elkaar gezet. Omdat de applicatie nog niet uitgebreid genoeg is, heeft Redhotminute tot op heden nog niet de mogelijkheid gehad om de applicatie voor klanten in te zetten. Om dit in de toekomst wel te kunnen doen, heeft Redhotminute een opdracht aangeboden om de applicatie uit te breiden en om te onderzoeken wat er precies nog aan het systeem gedaan dient te worden.

1.2. Doelstelling

Aangezien het project bijna een jaar stilgelegen heeft, is bepaalde informatie omtrent het uitbreiden van de applicatie verloren gegaan. Om de applicatie uit te kunnen breiden is er onderzoek nodig om te achterhalen waaraan het systeem uiteindelijk moet voldoen. Tevens is er de vraag gesteld of het systeem, eventueel na uitbreiding, op dit moment wel op klanten toegepast kan worden. De applicatie wordt tenslotte steeds complexer en is tot op heden nog nauwelijks getest.

Het doel is uiteindelijk om MOBO dusdanig uit te breiden, dat het volwassen genoeg wordt en op klanten van Redhotminute toegepast kan worden. Het volwassen maken van de applicatie zal in eerste instantie gerealiseerd worden door de functionaliteiten binnen het programma uit te breiden. Daarnaast zal er uit onderzoek moeten blijken of er meer dan alleen het uitbreiden van de functionaliteiten in het systeem moet gebeuren. Een goede applicatie heeft tenslotte niet alleen veel functionaliteiten, maar dient ook zo optimaal mogelijk gebruikt te kunnen worden.

1.3. Leeswijzer

In hoofdstuk 2, Over Redhotminute, wordt een beschrijving gegeven van de organisatie waarbij de afstudeeropdracht uitgevoerd is. Hierbij wordt tevens verteld op welke wijze het bedrijf te werk gaat.

In hoofdstuk 3, MOBO Requests, is de benodigde informatie omtrent het uit te breiden systeem te vinden. Zo worden de huidige functionaliteiten en de technieken die gebruikt worden toegelicht.

In hoofdstuk 4, De opdracht, wordt de opdracht, die tijdens het afstudeerproject is uitgevoerd, toegelicht en worden de onderzoeksvragen geformuleerd.

In hoofdstuk 5, CoffeeScript, staan de eerste onderzoeksresultaten vermeld. Hierbij wordt er naar de inhoud van de programmeertaal CoffeeScript gekeken en worden er alternatieven mee vergeleken.

In hoofdstuk 6, Onderzoek authenticatie MOBO, wordt de huidige authenticatie van het systeem aan de tand gevoeld. Er wordt onderzocht of deze authenticatiemethode wel de meest geschikte methode is.

In hoofdstuk 7, Realisatie functionaliteit ‘Repeating items’, wordt de grootst toegevoegde functionaliteit uitgebreid toegelicht. Hierbij wordt het duidelijk welke keuzes er gemaakt zijn en waar de nieuwe functionaliteit precies aan voldoet.

In hoofdstuk 8, Overzicht gerealiseerde functionaliteiten, worden alle aanpassingen in het systeem in een kort overzicht weergegeven.

In hoofdstuk 9, Onderzoek: van ontwikkeling naar livegang, zijn alle overige onderzoeksresultaten terug te vinden. Hierbij wordt er toegelicht welke tests er nog uitgevoerd moeten worden, hoe de prestaties en veiligheid gewaarborgd kunnen worden, wat er nog aan MOBO aangepast moet worden en hoe men de werking van MOBO kan controleren zodra het systeem live is.

In hoofdstuk 10, Conclusies, worden de hoofd- en deelvragen beantwoord.

In hoofdstuk 11, Aanbevelingen, worden naar aanleiding van de onderzoeksresultaten aanbevelingen gedaan.

In hoofdstuk 12, Evaluatie procesgang, wordt er op het project teruggekeken.

2. Over Redhotminute

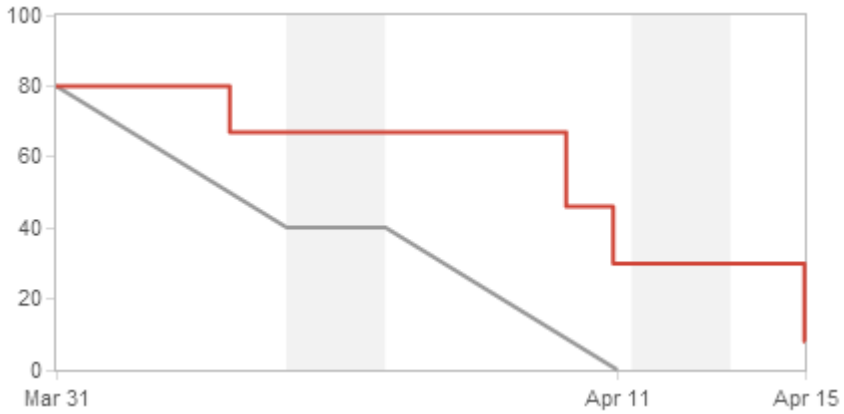
Mijn afstudeerstage heb ik met plezier bij Redhotminute, gevestigd in Tuil, gelopen. Redhotminute is een full-service internetbureau dat voor bedrijven websites, webwinkels, webapplicaties, online campagnes, mobiele sites en apps maakt. Het bedrijf is in 2001 opgericht en heeft ondertussen ongeveer 90 werknemers. De functies van de werknemers variëren van Designers tot aan Testers, van Functioneel Ontwerpers tot Front-end ontwikkelaars en van Software Engineers tot aan CMS specialisten. Kortom, Redhotminute bestaat uit een hecht en ervaren team van professionals met uitgebreide kennis van belangrijke aspecten van online marketing. Met het grote team aan specialisten werken ze onder andere voor bedrijven als KIA, Van der Valk, KLM en Landal Greenparks. (Redhotminute, Over ons, 2014)

2.1. Agile

Redhotminute werkt sinds 2013 volledig volgens de Agile ontwikkelmethodiek. De ervaring binnen en buiten Redhotminute is dat een Agile methodiek sneller leidt tot werkende producten, minder overbodige functionaliteiten geeft en naast veel andere voordelen ook meer commitment van medewerkers brengt. Ontwikkeling en oplevering van een platform verlopen volgens een continu proces. In het tweeweekse ritme van een Agile proces krijgt de klant steeds de zaken geleverd die de hoogste prioriteit hebben. Een onderdeel dat heel belangrijk is, kan in principe altijd direct in de volgende sprint worden opgepakt. Een onderdeel dat gereed is, uiteraard mits goed getest en door de opdrachtgever geaccepteerd, is dan direct live beschikbaar voor de klant en haar gebruikers. (Bek, 2014)

Binnen de organisatie werkt Redhotminute met een aantal teams, waarbij elk team met een project bezig is, bijvoorbeeld voor Van der Valk. Teams kunnen gedurende de tijd uitbreiden of inklinken, afhankelijk van de drukte. Tijdens mijn stage heb ik niet in één van de teams meegedraaid, omdat de opdracht een op zich zelf staand project is waar geen vaste medewerkers mee bezig zijn.

Omdat er bij Redhotminute volgens scrum (de Agile methodiek) gewerkt word, heb ik ook volgens deze methode gewerkt. Door op deze manier te werken heb ik zoveel mogelijk aansluiting bij het bedrijf gekregen en heb ik mijn project qua planning goed kunnen bewaken. Voor deze projectplanning en –bewaking maakt Redhotminute gebruik van de tool JIRA. In deze tool kunnen de sprints, die twee weken duren, met userstories en taken ingedeeld worden, waarbij teamleden taken toegekend kunnen krijgen. Door middel van grafieken kan de vordering van de sprint bewaakt worden. Om mijn project in te plannen heb ik ook van deze tool gebruik gemaakt.



Figuur 1 : Voorbeeld Burndown Chart

Naast het gebruik van dezelfde tools als Redhotminute, heb ik ook gebruik gemaakt van dezelfde termen. Hierdoor is het tijdens overleg met de medewerkers direct duidelijk waar over gesproken wordt. De termen waar vooral over gesproken is, zijn ‘Sprints’, ‘Epics’, ‘User stories’ en ‘Tasks’. Hieronder volgt een kleine uitleg van deze begrippen:

Elke sprint is een periode van twee weken, waarin een aantal user stories opgepakt worden. Een epic is een te realiseren functionaliteit, bestaande uit verschillende user stories. Om een user story af te kunnen ronden, moeten er diverse taken gedaan worden.

Aangezien er tijdens de gehele stageperiode gebruik gemaakt is van deze begrippen, zullen deze begrippen ook in dit verslag terugkomen.

3. MOBO Requests

Redhotminute heeft in het verleden een Enterprise Service Bus (ESB)[1] ontwikkeld die voor de website van een specifieke klant werd ingezet. Dit systeem is geschreven om de complexiteit in formulieren op de website van de klant tussen de verschillende landen te verkleinen. (Ze hebben voor verschillende landen verschillende websites). Het systeem is echter specifiek voor die ene klant ontworpen en kan dus niet voor elke website worden ingezet. Redhotminute heeft daarom onderzoek gedaan naar bestaande ESB systemen, maar deze bevatten veel functionaliteiten die voor Redhotminute overbodig zijn waardoor het programma te onhandig wordt. Daarnaast zou Redhotminute teveel voor deze systemen betalen, aangezien ze niet alle functionaliteiten zullen gebruiken.

Daarom heeft Redhotminute zelf een systeem laten bouwen dat specifiek aan hun eigen wensen voldoet. Dit systeem wordt MOBO Requests genoemd, wat staat voor 'Mid-Office / Back-Office aanvragen'. Het systeem bevatte bij de aanvang van de stage ongeveer de volgende functionaliteiten:

Het systeem kan data ontvangen van een aantal vastgestelde kanalen (bijvoorbeeld Facebook of een websiteformulier). Deze informatie wordt verwerkt, bijvoorbeeld getransformeerd en vervolgens verzonden naar één of meerdere vastgestelde kanalen (bijvoorbeeld een mailserver of een CRM). Nadat de data naar MOBO verzonden is, mag deze data niet verloren gaan, ook niet indien het systeem onverwachts afgesloten wordt. Gebruikers kunnen via Windows Azure inloggen om het controlpanel van MOBO te bereiken en hebben vervolgens via een eenvoudige user interface de mogelijkheid om input- en outputkanalen aan elkaar te verbinden. Ook kan de gebruiker aangeven welke transformaties de data moet ondergaan en hoe vaak de data opnieuw verzonden moet worden indien een outputkanaal niet beschikbaar is. De gebruiker kan ook een zogenaamd 'Fallback scenario' instellen voor wanneer een outputkanaal (na aantal keer proberen) niet reageert. Dit kan bijvoorbeeld het versturen van een e-mail zijn.

Alle acties die het systeem uitvoert worden bijgehouden en opgeslagen. De gebruiker kan door middel van een logboek nagaan of bepaalde transacties geslaagd of mislukt zijn. Verdere specifieke informatie, bijvoorbeeld de duur van transacties of overzichtelijke grafieken, wordt niet opgeslagen.

Om een duidelijk beeld te geven van de situatie van MOBO Requests bij aanvang van de stageperiode, is er een aantal schermafbeeldingen in 'Bijlage 3 : Impressie oude situatie MOBO' toegevoegd.

3.1. Technieken

MOBO Requests is in C# / ASP.NET geschreven en maakt gebruikt van Microsoft SQL Server voor het opslaan van de data. Het programma is in deze taal geschreven aangezien dit een vereiste vanuit het bedrijf is. Microsoft SQL is gebruikt omdat het goed met de C# / ASP.NET applicatie overweg kan en omdat deze omgeving vanuit Redhotminute is ingericht. Voor de front-end van MOBO wordt er HTML, CSS, JavaScript en CoffeeScript gebruikt. Omdat CoffeeScript bij de aanvang van mijn stage nog geheel onbekend voor me was, is er eerst een kort onderzoek naar deze taal gedaan. De resultaten hiervan staan in 'Hoofdstuk 5 : CoffeeScript'.

3.2. Authenticatie

Voor de authenticatie wordt er gebruikt gemaakt van Windows Azure Active Directory. Tijdens het realiseren van de eerste versie van MOBO is er onderzocht of men de authenticatie zelf zou gaan programmeren, of dat men van Windows Azure gebruik zou gaan maken. Destijds is er besloten om voor de laatste optie te gaan omdat dit veel (programmeer)tijd bespaart en omdat het veiliger is.

Omdat er destijds maar twee oplossingen onderzocht zijn, is het de vraag of Windows Azure wel de beste oplossing is. Om hier achter te komen is er onderzoek naar Windows Azure en andere mogelijke oplossingen gedaan. De resultaten van dit onderzoek staan in 'Hoofdstuk 6 : Onderzoek authenticatie MOBO' uitgewerkt.

[1] Een Enterprise Service bus (ESB) verbindt ERP-systemen en alle soorten applicaties op een eenvoudige manier. Hierdoor ontstaat een stabiele software-architectuur die simpel te onderhouden is. De verbindingen zijn niet langer 'point-to-point', maar verlopen centraal via de ESB. De werking van een ESB is te vergelijken met een tolk: het neemt informatie in een bepaalde taal en een specifiek format tot zich vanuit een applicatie, waarna het deze informatie aanbiedt aan andere applicaties in de juiste talen en formats. (Condor, 2014)

4. De opdracht

Tijdens het project is er aan twee aspecten gewerkt, namelijk het product en de onderzoeksvragen.

4.1. Het product

Een tweetal studenten is een aantal maanden geleden begonnen met het realiseren van MOBO Requests en heeft uiteindelijk de basis van het product opgeleverd. Deze basis bevat echter niet genoeg functionaliteiten om het product daadwerkelijk in productie te kunnen nemen. Om het product uiteindelijk in productie te krijgen, moeten er een meerdere functionaliteiten aan toegevoegd worden. Tevens zal er een aantal verbeterpunten aangepakt moeten worden om het systeem te optimaliseren.

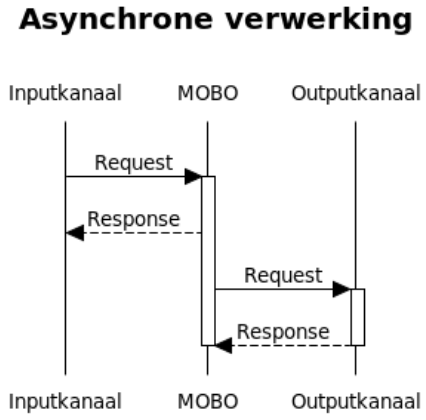
Voordat de stageperiode was ingegaan, waren er al twee functionaliteiten bekend waarvan met zekerheid gezegd kon worden dat ze aan het systeem toegevoegd moesten gaan worden. Om een compleet overzicht van de overige ontbrekende functionaliteiten te krijgen, zijn er interviews met een sommige medewerkers van Redhotminute gehouden die in het verleden iets met MOBO te maken hebben gehad. Uit de interviews is een lijst van functionaliteiten en verbeterpunten gekomen waarvan met zekerheid gezegd kan worden dat er wat mee gedaan moet worden. De lijst die hieruit naar voorgekomen is, ziet er als volgt uit:

1. Repeating items
Deze functionaliteit houdt in dat het systeem ook formulieren met lijsten kan verwerken. Op dit moment kan de applicatie enkel formulieren afhandelen waarin eendimensionale velden verwerkt zitten (bijvoorbeeld een formulier met velden voor NAW-gegevens). Het is de bedoeling dat het systeem formulieren kan verwerken waarin multidimensionale velden verwerkt zitten. Dit kan een formulier zijn met velden voor NAW-gegevens en een veld met een lijst van bijvoorbeeld contactpersonen. Maar het kunnen ook velden zijn met lijsten in lijsten (bijvoorbeeld een lijst met automerken met daarin lijsten met autotypes).

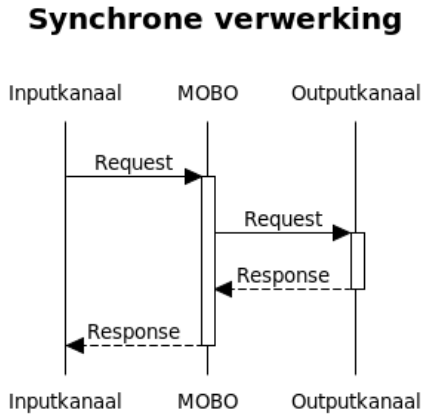
2. BAM (Business Activity Monitoring)
Op dit moment is er nog geen Business Activity Monitoring waarin bepaalde informatie van MOBO Requests gemonitord kan worden. Denk hierbij bijvoorbeeld aan het slagingspercentage en de aanvraagtijd van bepaalde requests. Door deze informatie te monitoren krijgt men een betere indruk over de prestaties van het systeem.

3. Synchrone verwerking
Op dit moment werkt het systeem asynchroon. Dat wil het volgende zeggen: een inputkanaal (bijvoorbeeld een webformulier) verstuurt informatie naar MOBO waarna het inputkanaal een response krijgt dat de data in MOBO ontvangen is. Vervolgens verstuurt MOBO de data door naar het outputkanaal (bijvoorbeeld een CRM), waarvan enkel MOBO een response krijgt, het inputkanaal krijgt hier geen response. Het inputkanaal heeft dus alleen een synchrone werking tot aan MOBO, maar niet met het outputkanaal. Zie ‘Figuur 2 : Asynchrone verwerking’.

De gewenste situatie is dat er een synchrone werking voor het gehele systeem is. Hierdoor kan de response van het outputkanaal op het inputkanaal weergegeven worden (bijvoorbeeld voor het opvragen van kentekengegevens uit een database). Zie ‘Figuur 3 : Synchrone verwerking’.



Figuur 2 : Asynchrone verwerking



Figuur 3 : Synchrone verwerking

4. Data verwerken van Input Channels
Op dit moment is er nog geen mogelijkheid om data op te halen van een externe webservice, om deze vervolgens te verwerken in een service. Deze functionaliteit kan in de toekomst wellicht noodzakelijk zijn om volledig van systeem gebruik te kunnen maken, aangezien sommige processen dit zullen gebruiken.

5. Nieuwe Input Channels
Er is een aantal Input Channels die aan het systeem toegevoegd zouden kunnen worden, waaronder:

- Twitter Input Channel: Een Input Channel dat tweets ophaalt aan de hand van een zoekterm (bijvoorbeeld de zoekterm '#Redhotminute')
- Facebook Input Channel: Een Input Channel dat berichten/ foto's ophaalt van een Facebook pagina.

Voor beide kanalen moet het mogelijk zijn om een schema in te stellen, zodat bijvoorbeeld elk uur op nieuwe berichten gecontroleerd kan worden. Om deze kanalen te kunnen implementeren is het volgens medewerkers noodzakelijk om eerst de functie 'Data verwerken van Input Channels' te realiseren. Ook zou de software Copernica, wat gebruikt wordt voor o.a. e-mailmarketing en geautomatiseerde campagnes, als nieuwe Input Channel gebruikt kunnen worden.

6. Verbeterpunten
Buiten de geheel ontbrekende functionaliteiten om is er volgens de geïnterviewde medewerkers nog een aantal punten waarop MOBO verbeterd zou kunnen worden.

- Vormgeving/usability: de vormgeving zou op bepaalde plekken binnen MOBO verbeterd kunnen worden. Dit geldt ook voor de usability. Hier is tijdens het maken van de eerste versie van MOBO geen rekening mee gehouden.
- Validatie: bepaalde velden worden binnen MOBO gevalideerd (controle op e-mail, leeftijd, int, etc.). Op sommige plekken ontbreken validaties of werken validaties niet naar behoren. Hier kan dus het één en ander in verbeterd worden.

Er zal nader onderzocht moeten worden op welke plekken deze verbeterpunten van toepassing zijn.

Naast deze ontbrekende functionaliteiten en verbeterpunten zijn er tijdens het project nog diverse andere verbeteringen aan het licht gekomen. Daarom is er een lijst met aanbevelingen opgesteld waaraan in de toekomst nog gewerkt kan worden. Zie hiervoor 'Hoofdstuk 11 : Aanbevelingen'.

Omdat er uit de interviews meerdere functionaliteiten en verbeterpunten gekomen zijn, zijn deze punten met de product owner (Renso Bek) besproken om ze vervolgens te prioriteren. Vanaf nu worden de functionaliteiten epics genoemd. Uit de prioritering zijn uiteindelijk de volgende epics gekomen:

Prioriteit 1:
De product owner wil eerst de epic 'Repeating items' gerealiseerd hebben. Volgens hem is dit het belangrijkste onderdeel om MOBO een stuk volwassener te maken, aangezien deze functionaliteit ook in de overige epics nodig kan zijn.

Prioriteit 2:
De epic die vervolgens gerealiseerd zou moeten gaan worden is de synchrone verwerking.

Vervolg:
Aangezien er maar met één persoon aan het project gewerkt wordt, is er van te voren ingeschat dat we waarschijnlijk niet verder dan de twee epics – die hierboven genoemd zijn – zouden komen. Aangezien er volgens Agile gewerkt wordt, kan een epic tijdens het project ook ineens vele malen groter worden omdat de product owner de epic van te voren anders ingeschat had. Daarom is er gestart met het realiseren van de eerste epic, om vervolgens in de loop van het project te kijken hoe ver er uiteindelijk gekomen wordt. Wat er uiteindelijk allemaal gerealiseerd is, is verderop in het verslag te lezen.

4.2. Onderzoeksvragen

Naast het uitbreiden van MOBO Requests is er tijdens het project ook naar een aantal aspecten onderzoek gedaan. De hoofdvraag die hierbij centraal stond is als volgt: Hoe kan MOBO, zodra het volwassen genoeg is, foutloos op een bestaand systeem (van een klant) ingezet worden, zonder dat er gegevens verloren gaan of juist dubbel opgeslagen worden en wat dient er vóór de implementatie nog te gebeuren?

De deelvragen die hierbij behandeld zijn, zijn als volgt geformuleerd:

- Wat is CoffeeScript en zijn hier alternatieven voor? (Later toegevoegd omdat kennis noodzakelijk bleek.)
- Moet Microsoft Azure als authenticatie gebruikt blijven worden, of zijn hier betere/goedkopere alternatieven voor? Welke alternatieven zijn er?
- Hoe ziet elke toe te voegen functionaliteit er geheel uit, wat doet het, wat zijn de eisen aan de vormgeving, wat zijn de specifieke kwaliteitseisen, etc.?
- Welke tests moeten er nog uitgevoerd worden voordat MOBO op klanten toegepast kan gaan worden?
- Hoe kunnen we er zo zeker mogelijk van zijn dat de prestaties (snelheid e.d.) van het systeem goed zijn/blijven na de implementatie van MOBO?
- Wat moet er, naast de ontbrekende functionaliteiten, aan het huidige systeem aangepast worden om MOBO op klanten toe te kunnen passen?
- Hoe moet er gecontroleerd gaan worden of MOBO alle data op het toegepaste systeem correct verwerkt, zonder dat er data verloren gaat of onbeheerst dubbel opgeslagen wordt?

5. CoffeeScript

De technieken van MOBO zijn eerder in dit document al even toegelicht, waarbij de front-end naast HTML en CSS voornamelijk uit CoffeeScript is bestaat. Deze taal was volledig onbekend voor me, waardoor het niet eenvoudig was om hier direct in te duiken. Om eerst een goede basiskennis op te doen, is er kort onderzoek gedaan naar CoffeeScript. Hierbij is er gekeken naar wat de taal precies inhoudt en of er alternatieven zijn die misschien wel beter zijn. Voorafgaand aan het project is er geen rekening met dit onderzoek gehouden, maar aangezien het misschien nuttige informatie op zou leveren is er besloten om dit als aanvullend onderzoek aan het project toe te voegen.

5.1. Wat is CoffeeScript

CoffeeScript is een taal die compileert naar JavaScript. De taal probeert hiermee dezelfde functionaliteiten als JavaScript te bieden, maar doet dit door minder regels code te gebruiken. Om dit te bereiken is de syntax zo kort mogelijk gehouden. Hiermee wordt direct geprobeerd om één van de grootste nadelen van JavaScript op te lossen, namelijk 'bloated code', waarbij men het produceren van code als onhandig en onnodig lang ervaart. Uiteindelijk levert dit 33% minder regels code op waarvan de prestatie niet minder is dan de standaard JavaScript-code. (Finley, Interview: Jeremy Ashkenas Talks About CoffeeScript, 2011)

De CoffeeScript code wordt één op één naar JavaScript gecompileerd. Hierdoor kan er vanuit CoffeeScript probleemloos naar externe JavaScript bestanden of –library's verwezen worden. De compiler maakt gebruik van JavaScript Lint waarmee de meest gemaakte JavaScript fouten gecorrigeerd worden. Tevens maakt de compiler in de gegenereerde code gebruik van whitespaces en inspringen. Hierdoor blijft de gecompileerde code goed leesbaar. (Embregts, 2012)

Hieronder wordt een functie, geprogrammeerd in CoffeeScript, weergegeven met daaronder de functie gecompileerd naar JavaScript. Hierin is duidelijk te zien dat CoffeeScript aanzienlijk minder code nodig heeft. Waar CoffeeScript vijf regels gebruikt, gebruikt JavaScript er veertien. Tevens heeft CoffeeScript maar 193 karakters nodig, waarbij JavaScript er 398 nodig heeft.

CoffeeScript:

```
LoadNodeTypes: =>
  $.post("/Services/GetNodeTypes/", (data) =>
    for nodeType in data
      Configurator.ConfiguratorModel.ServiceModel.NodeTypes.push(new Configurator.NodeType(nodeType));
  ).fail (e) => @HandleError e
```

JavaScript:

```
ConfiguratorModel.prototype.LoadNodeTypes = function() {
  var _this = this;

  return $.post("/Services/GetNodeTypes/", function(data) {
    var nodeType, _i, _len, _results;

    _results = [];
    for (_i = 0, _len = data.length; _i < _len; _i++) {
      nodeType = data[_i];
      _results.push(Configurator.ConfiguratorModel.ServiceModel.NodeTypes.push(new
      Configurator.NodeType(nodeType)));
    }
    return _results;
  }).fail(function(e) {
    return _this.HandleError(e);
  });
};
```

5.2. Alternatieven

De studenten die de eerste versie van MOBO hebben gemaakt, hebben destijds voor CoffeeScript gekozen. Het blijkt dat het nu lastiger is om het project over te nemen wanneer er geen kennis van CoffeeScript is. Het blijkt dat CoffeeScript zeker voordelen heeft ten opzichte van JavaScript, maar doordat het een andere syntax dan JavaScript heeft, wordt de instapdrempel groter wat weer een nadeel is.

Uit literatuuronderzoek blijkt dat er meer talen zijn die als alternatief voor JavaScript gebruikt kunnen worden. De meest genoemde alternatieve talen voor JavaScript zijn CoffeeScript, Dart en TypeScript.

Op het internet lopen de meningen omtrent de keuze van één van deze talen uiteen, omdat de keuze ook afhankelijk is van de omgeving waarop de applicatie gemaakt wordt of waarop de applicatie uiteindelijk komt te draaien. Het blijkt dat alle alternatieven naar JavaScript compileren, waardoor er van externe JavaScript bestanden of –library's gebruik gemaakt kan worden. Bij de één gaat dit echter makkelijker dan bij de ander. (CodeforHire, 2013)

De website Something Somewhere heeft de voor- en nadelen van de alternatieven in kaart gebracht. (SomethingSomewhere, 2013). Hieruit komt het volgende overzicht voort:

	CoffeeScript	JavaScript	TypeScript	Dart
Better Structure	✓	✗	✓	✓
Editor Friendly	✗	✗	✓	✓
Better Speed	✗	✗	✗	✓
Large Application	✗	✗	✓	✓
Brevity of Code	✓	✗	✗	✗
Ease of Learning	✗	✓	✓	✗
Easy Debugging	✗	✓	✓	✓

Tabel 1 : Overzicht talen

Het blijkt dat TypeScript het beste op Microsoft applicaties toegepast kan worden. (CodeforHire, 2013) Aangezien MOBO ook in een Microsoft omgeving is opgebouwd, had deze keuze misschien logischer geweest dan CoffeeScript. Tevens heeft TypeScript betere referenties dan CoffeeScript. Desondanks werkt CoffeeScript goed in de huidige applicatie. Het veranderen van deze code is dus niet noodzakelijk, maar voor het maken van applicaties in de toekomst is het goed om te weten dat er verschillende alternatieven zijn voor JavaScript die qua performance en structuur zeker voordelen hebben ten opzichte van de meest gebruikte taal; JavaScript. (Finley, JavaScript Tops Latest Programming Language Popularity Ranking From RedMonk, 2012) Aangezien TypeScript het meest voor Microsoft applicaties wordt aangeraden, is het aan te bevelen om in de toekomst TypeScript te gebruiken indien er een alternatief voor JavaScript in een Microsoft applicatie gebruikt gaat worden.

6. Onderzoek authenticatie MOBO

In de oude situatie wordt er voor de authenticatie van MOBO gebruik gemaakt van Windows Azure Active Directory. Omdat er ongetwijfeld andere mogelijkheden voor authenticatie zijn en omdat er tijdens het realiseren van de eerste versie van MOBO weinig onderzoek naar gedaan is, is er besloten om er tijdens dit project opnieuw naar te kijken. Er is nu ook ervaring met de huidige oplossing, waardoor de voor- en nadelen in het onderzoek meegenomen kunnen worden

6.1. Windows Azure Active Directory

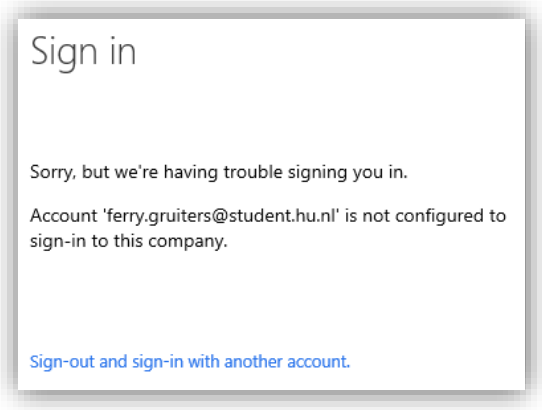
De eerste oplossing waar naar gekeken is, is de huidige oplossing: Windows Azure Active Directory (WAAD). WAAD is een onderdeel van Windows Azure dat een cloud computing-platform van Microsoft is wat sinds 2008 gebruikt kan worden om softwarediensten uit de cloud te leveren. Via dit platform worden diverse internet services aangeboden die voor elke klant breed aanpasbaar en uit te breiden zijn. Er kunnen bijvoorbeeld applicaties ontwikkeld worden met een SQL-database, maar het is ook te gebruiken als een storagelocatie voor grote hoeveelheden data. (Kieft, 2013)

Windows Azure biedt dus een tal van oplossingen als het gaat om het bouwen en hosten van (web)applicaties. Voor het hosten heeft MOBO Windows Azure echter niet nodig, aangezien MOBO op één van de servers van Redhotminute draait. Het is dus niet nodig om de hosting mogelijkheden van Windows Azure hiervoor te gebruiken. Naast de opslagmogelijkheden biedt Windows Azure ook een oplossing voor identiteits- en toegangsbeheer aan, WAAD. Deze oplossing wordt op dit moment voor MOBO gebruikt.

Aangezien Windows Azure al geheel in MOBO geïntegreerd is, is er weinig onderzoek gedaan naar de manier waarop het geïmplementeerd moet worden. Volgens de studenten die MOBO ontwikkeld hebben, heeft Microsoft een API beschikbaar gesteld die het heel eenvoudig maakt om de koppeling met WAAD te maken. (Sluijter & De Kam, 2013) Er is vooral gekeken wat op dit moment de nadelen van WAAD binnen MOBO zijn, om deze vervolgens tegen andere oplossing af te wegen.

Het eerste nadeel waar we met WAAD mee te maken hebben, is dat er voor het inloggen gebruik wordt gemaakt van een externe inlogpagina, namelijk die van Microsoft. Het kan gebeuren dat een gebruiker nog op zijn of haar Microsoftaccount is ingelogd (bijvoorbeeld Webmail), waarmee de gebruiker geen toegang tot MOBO heeft. In dit geval kan de gebruiker niet direct bij MOBO inloggen, maar moet men zich eerst op ingelogde Microsoftaccount uitloggen. Webmail (of een andere Microsoft service) en MOBO tegelijkertijd gebruiken is dus geen optie.

Het volgende nadeel is dat gebruikers altijd met een lastige inlognaam in moeten loggen. Dit komt omdat er voor de geconfigureerde applicatie in Windows Azure een specifiek e-mailadres gebruikt moet worden. Microsoft gebruikt hiervoor een vastgestelde structuur, namelijk *gebruikersnaam@applicatienaam.*onmicrosoft.com



Voor een gebruiker die niet dagelijks van het systeem gebruik maakt, kan dit erg lastig zijn om te onthouden. Het zou in dit geval een betere oplossing zijn om persoonlijke inloggegevens (een eigen gebruikersnaam of e-mailadres) te gebruiken.

In het huidige systeem is het ook niet mogelijk om je wachtwoord opnieuw in te stellen of om je gebruikersnaam op te vragen. Het opnieuw instellen van het wachtwoord moet via de beheerder (die het vervolgens weer via Windows Azure moet doen). Het opvragen van de gebruikersnaam (het e-mailadres) is simpelweg niet mogelijk.

Aangezien er in de huidige versie van MOBO gebruik wordt gemaakt van de gratis versie van WAAD, ontbreken er een aantal functionaliteiten. Sinds 2 april 2014 is er een betaalde versie van WAAD beschikbaar. (Deuby, 2014) Deze nieuwe versie biedt bijvoorbeeld een ‘Self-service password reset’ functionaliteit om gebruikers de mogelijkheid te geven om zelf hun wachtwoord te laten resetten, zonder de administrator hiervoor te moeten benaderen. (Microsoft, Azure Active Directory Premium, 2013) Om deze functionaliteit te kunnen gebruiken dient MOBO echter wel aangepast te worden.

Een ander klein voordeel van de betaalde versie is dat de huisstijl van het bedrijf, of in dit geval van MOBO, in de aanmeldingspagina van Microsoft opgenomen kan worden. Hierdoor lijkt het inloggen direct persoonlijker.

Verder zijn er weinig nadelen te ontdekken. Het enige nadeel dat nog gevonden zou kunnen worden, is dat er een maximaal aantal gebruikers in de Active Directory opgeslagen kan worden. Maar aangezien dit aantal op 500.000 vastgesteld is (standaard op 150.000, maar na aanvraag te verhogen naar 500.000), zal er niet zomaar een tekort aan ruimte ontstaan. (Microsoft, Azure Active Directory Premium, 2013)

Al met al zijn er dus geen enorme nadelen te vinden waar per direct een oplossing voor gevonden zou moeten worden. De authenticatie zelf werkt zonder problemen, de knelpunten liggen vooral bij de extra functionaliteiten die bij de authenticatie van pas kunnen komen.

Ondanks dat er bij Windows Azure Active Directory weinig zwaarwegende nadelen gevonden zijn, is er toch besloten om naar andere mogelijkheden te kijken. Misschien dat hier voordelen aan het licht komen die uiteindelijk goed van pas kunnen komen. Omdat de authenticatie van MOBO geen kernfunctionaliteit is, moeten de andere mogelijkheden eenvoudig geïmplementeerd kunnen worden, zonder hier al te veel tijd in te moeten steken. Het moeten dus standaardoplossingen waarbij weinig (extra) code geschreven hoeft te worden. Voorafgaand aan het onderzoek was er verwacht dat er redelijk veel over authenticatie-oplossingen van (C#) applicaties te vinden zou zijn. Tijdens het onderzoek bleek het tegendeel waar; er zijn weinig standaardoplossingen voor een goede authenticatie. Naast authenticatie moet de oplossing namelijk ook een oplossing voor gebruikersbeheer bieden. Uiteindelijk is er ook maar één goed alternatief voor WAAD gevonden, namelijk Thinktecture IdentityServer.

6.2. Thinktecture IdentityServer

Thinktecture IdentityServer is een lichtgewicht security-token service, gemaakt met .NET 4.5, MVC 4, Web API en WCF. (Thinktecture, 2012) Identityserver is een erg populaire identity provider met uitstekende ondersteuning voor WS-Federation en WS-Trust. Tevens worden OAuth 2.0 en OpenID Connect sinds de laatste versie ondersteund. Standaard wordt ASP.NET Membership gebruikt, maar dit kan eenvoudig worden uitgebreid met andere datastores. (Leastprivilege, 2014) Daarnaast is deze oplossing al eens eerder bij een klant van Redhotminute ingezet. Er zijn dus al medewerkers die ervaring met deze oplossing hebben.

Om gebruik van Thinktecture IdentityServer te kunnen maken, dient er van .NET Framework 4.5, IIS 7 of hoger en van een SSL verbinding gebruik gemaakt te worden. Aangezien MOBO al aan deze eisen voldoet, vormt dit geen probleem, mocht er overwogen worden om voor deze oplossing te kiezen.

Het is bij dit onderzoek vooral belangrijk of de functionaliteiten die door de oplossing aangeboden worden voldoende zijn, of de authenticatie veilig verloopt en of het zonder ingrijpende veranderingen van MOBO te implementeren is.

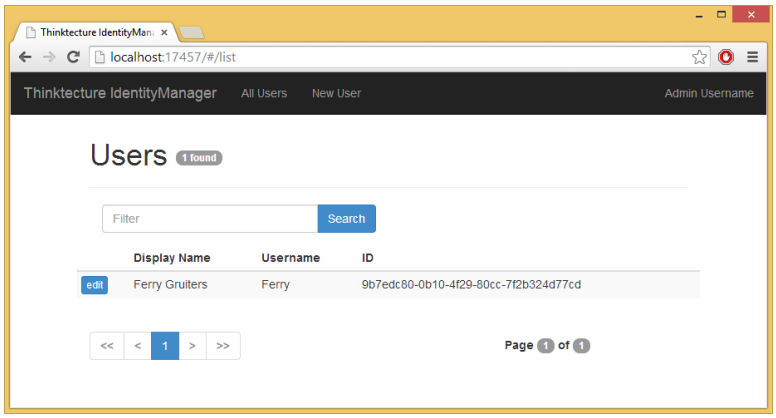
Thinktecture IdentityServer biedt als basis natuurlijk de mogelijk om in te loggen op een (web)applicatie. Aangezien het een OpenSource project is en er op internet sourcecodes en een aantal wiki's te vinden zijn (onder andere op https://github.com/thinktecture/Thinktecture.IdentityServer.v3 en http://thinktecture.github.io/), is de instapdrempel laag. Er hoeft namelijk niet voor geïnvesteerd te worden om het te mogen gebruiken en dankzij de vele code die al beschikbaar is, is het niet nodig om alles volledig zelf op te zetten.

Omdat Thinktecture volledig in een eigen applicatie te integreren is, hoeft er niet van een externe inlogpagina gebruik gemaakt te worden, wat bij WAAD wel het geval is. De inlogpagina kan dus volledig aan de lay-out van MOBO aangepast worden.

Het volgende voordeel ten opzichte van WAAD is dat de gebruikers met herkenbaardere gegevens in kunnen loggen. Bij Thinktecture IdentityServer wordt er namelijk niet van een onhandig lang e-mailadres gebruik gemaakt, maar van een normale gebruikersnaam. Binnen de standaardoplossing die door Thinktecture wordt aangeboden, is het echter niet mogelijk om je gebruikersnaam/wachtwoord te resetten of op te vragen. Wel is het mogelijk om dit door middel van MembershipReboot uit te breiden. (Brockallen, 2014) Naast het inloggen met een gebruikersnaam en wachtwoord, is het ook mogelijk om in te loggen via een externe service. Denk hierbij aan Facebook, Twitter of Google. Dit kan een bijdrage leveren aan het vereenvoudigen van het inloggen, aangezien men tegenwoordig al veel verschillende wachtwoorden moet onthouden. Uit onderzoek van Citrix Systems blijkt dat dit regelmatig voor problemen zorgt. (Trosradar, 2006) Door deze oplossing te implementeren, is de gebruiker niet genoodzaakt het wachtwoord te onthouden. Het kan namelijk voorkomen dat MOBO niet wekelijks gebruikt wordt, wat het onthouden van het wachtwoord niet makkelijker maakt.

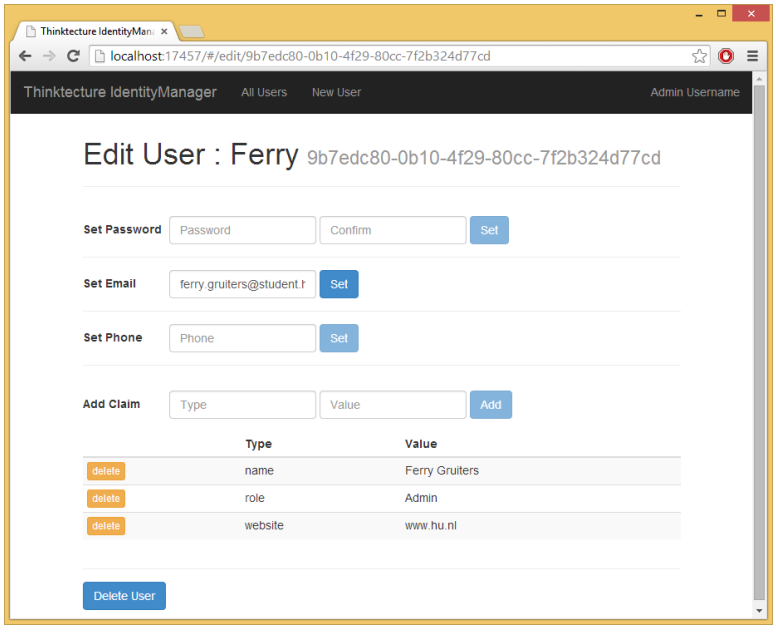
In de huidige oplossing van MOBO heeft de administrator de mogelijkheid om via het controlpanel gebruikers aan te maken of te verwijderen. Het is hierbij niet mogelijk om de inloggegevens van de gebruikers aan te passen. Thinktecture heeft hier wel een oplossing voor, namelijk IdentityManager. Deze oplossing wordt naast de IdentityServer aangeboden. Met de IdentityManager is het mogelijk om via een controlpanel eenvoudig gebruikers toe te voegen, aan te passen of te verwijderen. Deze oplossing wordt als een moderne vervanging gezien voor de ASP.NET WebSite Administration tool dat tot voorheen in Visual Studio aangemaakt kon worden. (Videos, 2014) In verband met problemen met Microsoft's Membership is de WebSite Administration tool verwijderd. (Galloway,

2012) De IdentityManager past perfect binnen de oplossing die voor MOBO nodig is, aangezien het voor de administrator van MOBO mogelijk moet zijn om eenvoudig gebruikers te kunnen beheren, zonder dat men steeds op een externe website (bijvoorbeeld Azure) in moet loggen. Om de functionaliteiten van de IdentityManager te testen, is er een aantal handleidingen doorlopen om vervolgens de IdentityManager op te bouwen. Het opzetten van de IdentityManager blijkt zeer eenvoudig te zijn.



Figuur 4 : IdentityManager Users

Aangezien er binnen MOBO gebruik gemaakt wordt van rollen (gebruikersniveaus), moet de oplossing voor de authenticatie hier ook een oplossing voor bieden. De IdentityServer/Manager biedt hier een perfecte oplossing voor, aangezien aan elke gebruiker zogenaamde claims gekoppeld kunnen worden. In een claim kan bijvoorbeeld extra informatie van een gebruiker opgeslagen worden (denk aan een website), kunnen rollen ingesteld worden, maar kan ook de informatie van een externe inlog (bijvoorbeeld Twitter) ingesteld worden, zodat de gebruiker de mogelijkheid heeft om zich via een externe inlog aan te melden.



Figuur 5 : IdentityManager Edit User

Het is mogelijk om de IdentityManager geheel in MOBO in te bouwen, waardoor het gebruikersbeheer via het normale controlpanel van MOBO bereikbaar is en tevens dezelfde lay-out heeft.

Qua functionaliteiten voldoet Thinktecture IdentityServer, inclusief de IdentityManager en de uitbreiding van MembershipReboot, geheel aan de functionaliteiten die MOBO nodig heeft. Aangezien er van een beveiligde verbinding gebruik gemaakt moet worden en omdat er bekende protocollen (OpenID Connect, OAuth 2.0, WS-Federation, etc.) gebruikt kunnen worden, zal de veiligheid van de verbinding geen probleem opleveren. Omdat MOBO precies aan de vereisten van Thinktecture voldoet (ASP.NET 4.5, IIS7+ en een SSL verbinding), hoeft er wat dat betreft niets aan MOBO aangepast te worden. Natuurlijk dienen er wel wat codes aangepast te worden om de oude authenticatie volledig te verwijderen en de nieuwe geheel te implementeren.

6.3. Conclusie

Windows Azure Active Directory biedt op dit moment een oplossing die correct werkt. Ongeautoriseerde personen kunnen de applicatie niet bereiken en personen die wel over inloggegevens beschikken kunnen zonder problemen inloggen. Natuurlijk zijn er wel verbeterpunten, maar dit hoeft niet per direct aangepakt te worden, omdat de autorisatie op dit moment gewoon werkt.

De verbeterpunten blijken met Thinktecture allemaal aangepakt te kunnen worden. Er hoeft namelijk niet meer op een externe pagina ingelogd te worden, gebruikers hebben eenvoudigere inloggegevens (en kunnen zelfs met andere services inloggen) en gebruikers worden in staat gesteld hun eigen wachtwoord te resetten.

Op dit moment is de prioriteit er echter niet om deze verbeterpunten aan te pakken. De authenticatie biedt een bijdrage aan het product, maar heeft verder niets met de kernfunctionaliteit te maken. Er is daarom besloten om niets aan de authenticatie aan te passen. Zodra MOBO af is, dan kan er in de toekomst eventueel gekeken worden of er wat aan de authenticatie gedaan moet worden. En aangezien Windows Azure Active Directory pas sinds april 2013 bestaat (ScottGu, 2013), is de kans groot dat hier nog verbeteringen in komen, zeker gezien Microsoft op 2 april 2014 nog een betaalde versie van WAAD geïntroduceerd heeft, waarmee het resetten van wachtwoorden al vereenvoudigd is. (Deuby, 2014)

7. Realisatie functionaliteit ‘Repeating items’

Om een eerste stap te zetten in het volwassen maken van MOBO is er op basis van prioritering besloten om te beginnen met het realiseren van de ondersteuning van repeating items, oftewel lijsten. Om deze functionaliteit te kunnen realiseren, is er een aantal zaken aangepast om lijsten te kunnen ontvangen en is er vervolgens onderzocht in welke situaties er lijsten verwerkt moeten kunnen worden.

Tijdens dit onderzoek is er geprobeerd om een zo goed mogelijke schets te maken voor de implementatie van de functionaliteit ‘Repeating items’. Door hier vooraf goed over te na te denken, gaat er tijdens het realiseren van deze functionaliteit minder tijd verloren door het realiseren van userstories die achteraf foutief of overbodig blijken.

7.1. Wat houdt de functionaliteit ‘Repeating items’ precies in?

In de oude situatie kon MOBO alleen inputdata ontvangen dat bestaat uit eendimensionale velden (bijvoorbeeld een webformulier met velden bestaande uit NAW-gegevens). Het systeem moet echter ook data kunnen verwerken dat multidimensionale velden bevat, bijvoorbeeld: een webservice die data verstuurt bestaand uit NAW-gegevens en een lijst van contactpersonen.

Om uiteindelijk aan de behoefte van de klant te kunnen voldoen, moest er met de klant afgestemd worden hoe de functionaliteit er uit moest komen te zien. Er is daarom in eerste instantie met Renso Bek overlegd. Renso Bek speelt in dit project de rol als klant (Product Owner) en geeft daarom weinig prijs over de technische mogelijkheden van de implementatie, maar geeft alleen informatie vanuit zijn rol als klant. Het enige wat Renso vooraf bekend heeft gemaakt is dat het systeem alle mogelijke scenario’s met lijsten moet gaan ondersteunen, omdat het systeem simpelweg zo generiek mogelijk moet zijn. Er is daarom onderzocht welke scenario’s voor kunnen komen, daarnaast zijn deze scenario’s uitgewerkt en ontworpen om ze uiteindelijk te kunnen implementeren. De scenario’s die hieruit naar voren gekomen zijn, staan vanaf ‘Hoofdstuk 7.4. : Mapping’ beschreven.

7.2. Structuur lijsten

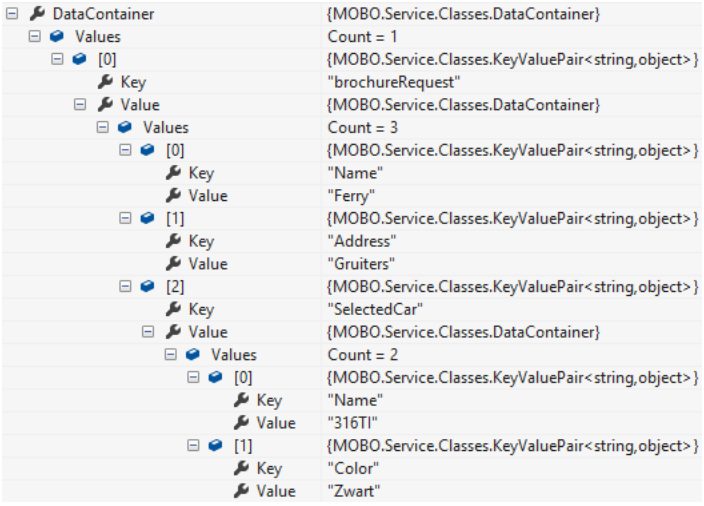
Aangezien het oude systeem nog geen lijsten kon verwerken, moest er eerst een oplossing gevonden worden voor de structuur waarop lijsten binnen MOBO verwerkt moeten worden. Zonder deze oplossing zou het anders niet mogelijk zijn om de scenario’s te testen, omdat het niet mogelijk is om data met lijsten te ontvangen.

Alle data die bij MOBO binnenkomt, wordt in een zogenaamde DataContainer opgeslagen. Een DataContainer bestaat uit KeyValuePairs, waarbij elke pair een key en een value heeft. Bij een formulier met de velden ‘adres’ en ‘woonplaats’ worden de velden als twee KeyValuePairs in de DataContainer opgeslagen, waarbij de key de naam van het veld is en de value de waarde van het veld.

Het is binnen MOBO echter ook mogelijk om definities aan te maken. Definities zijn als het ware objecten die eigenschappen bevatten. Denk hierbij bijvoorbeeld aan het object ‘Persoon’ met de eigenschappen ‘naam’ en ‘leeftijd’. De definities worden een stuk complexer dan simpele velden opgeslagen. Omdat een definitie meerdere velden kan bevatten, wordt een definitie weer als een aparte DataContainer opgeslagen. In deze DataContainer worden vervolgens de velden van een definitie als KeyValuePair opslagen. Op deze manier ontstaat er een boomstructuur die door MOBO goed uit te lezen is.

Om een beter beeld van deze structuur weer te geven, is er in de afbeelding hiernaast een definitie ‘BrochureRequest’ gedefinieerd, die vervolgens weer een definitie ‘SelectedCar’ bevat.

Om in deze structuur lijsten op te kunnen nemen, moeten KeyValuePairs de value “List<item>” of “item[]” kunnen krijgen. Deze situatie is uitgeprobeerd, waarbij er bijvoorbeeld een lijst van Cars naar MOBO gestuurd werd. In dit geval krijgt de KeyValuePair de waarde “List<Car>” of “Car[]”. Het werken met lijsten of arrays is namelijk allebei mogelijk.

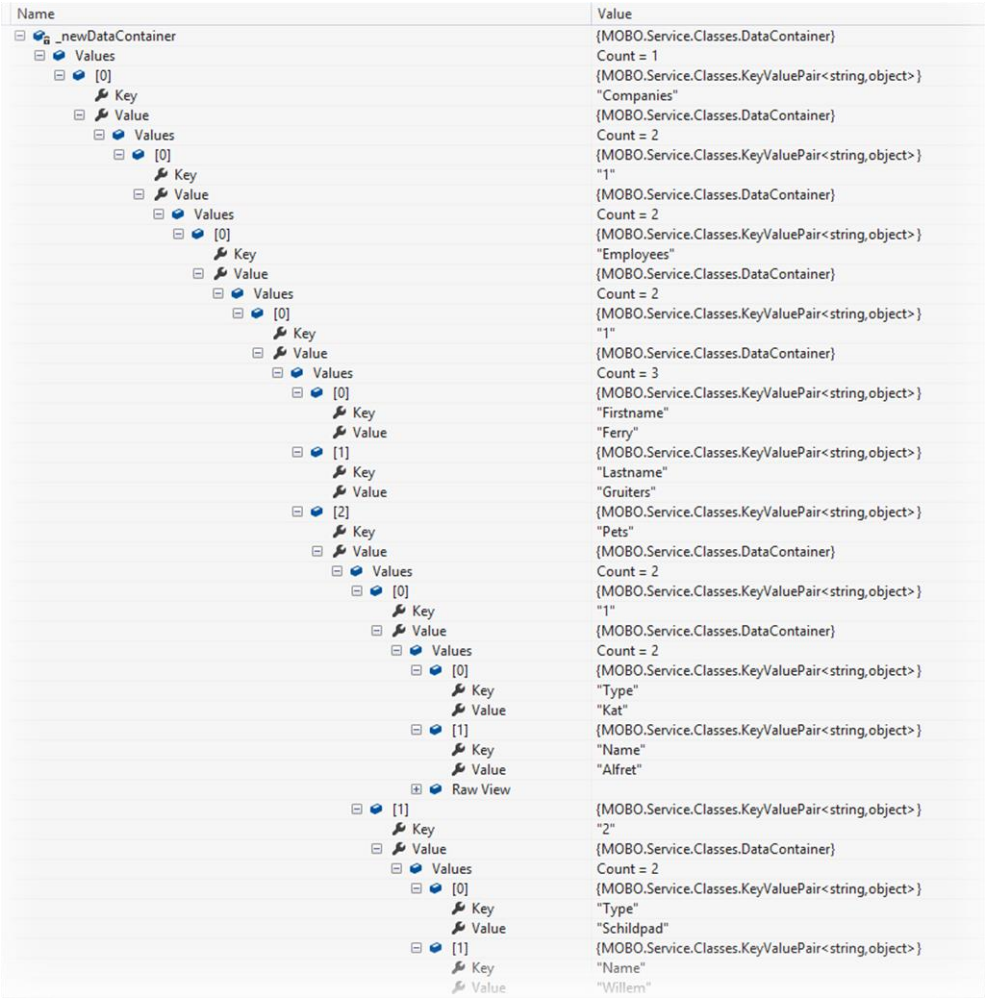


Figuur 6 : Structuur DataContainer

Uiteindelijk bleek deze situatie niet eenvoudig te implementeren, aangezien MOBO het object ‘Car’ niet herkent. Definities worden namelijk in het ControlPanel van MOBO gedefinieerd, maar de definities zijn in de Host van MOBO niet meer te herleiden, waardoor ze verder niet serializable zijn. Het is wel mogelijk om op deze manier lijsten van eenvoudige types (bijvoorbeeld strings, ints, bools, etc.) te versturen, aangezien deze types standaard classes zijn. Hierdoor worden deze types wel in de Host van MOBO herkend.

Om lijsten van definities te ondersteunen, moest er dus een andere oplossing bedacht worden. Aangezien MOBO de huidige structuur goed ondersteunt, is er gekozen om voor lijsten dezelfde structuur aan te houden. Elke lijst van definities zal daarom uit een extra DataContainer bestaan. In plaats van een KeyValuePair met een List<> of array als value, krijgt de pair nu een nieuwe DataContainer als value. Deze structuur zal uiteindelijk heel complex worden, maar wijkt verder niet af van de huidige structuur waardoor het systeem generiek blijft.

Een DataContainer waarin een lijst in een lijst in een lijst is opgenomen, kan er als volgt uit komen te zien:

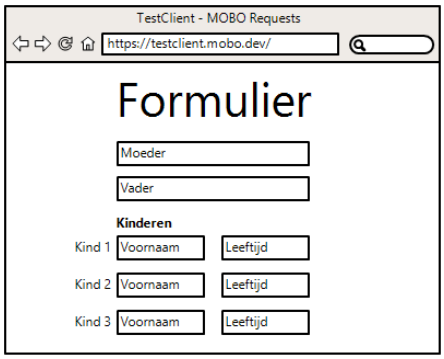


Figuur 7 : Structuur DataContainer met lijst

In dit voorbeeld worden er meerdere genestelde lijsten gebruikt. In dit geval is er een lijst ‘Companies’ aanwezig. Elk item in deze lijst bevat een lijst ‘Employees’. De items in de lijst ‘Employees’ bevatten vervolgens weer ieder een lijst ‘Pets’. Om de complexiteit aan te geven: het voorbeeld bevat één Company, één Employee en twee Pets. Een lijst met een aantal companies die vervolgens allemaal een lijst van employees bevatten zal dus een structuur opleveren die vele malen groter is dan in het voorbeeld.

7.3. Kanalen configureren

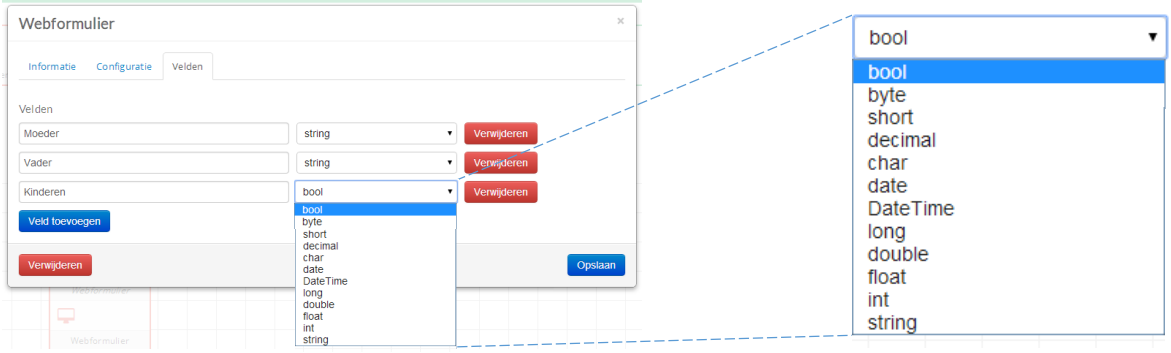
In MOBO wordt er gebruikt gemaakt van kanalen; input- en outputkanalen. Een inputkanaal zal de data naar MOBO versturen waarna MOBO de data naar een outputkanaal stuurt. Om te zorgen dat MOBO dit proces kan doorlopen, moeten de kanalen geconfigureerd worden. Om een duidelijk beeld te geven van de oude situatie van de configuratie van de kanalen, wordt er hieronder kort door de configuratie heen gelopen. Hierdoor wordt het duidelijk dat er in de oude situatie geen lijsten in de kanalen geconfigureerd konden worden.



Figuur 8 : Voorbeeld webformulier met lijst

Het is mogelijk om in MOBO een inputkanaal aan te maken, bijvoorbeeld een webformulier, om deze vervolgens naar een outputkanaal, bijvoorbeeld een CRM, door te sturen. In MOBO moet worden aangegeven welke velden met data er door het webformulier verstuurd worden en uiteindelijk bij MOBO binnenkomen. De data van het webformulier wordt door middel van een POST-methode naar MOBO gestuurd. In dit voorbeeld gaan we uit van de volgende situatie: een formulier met daarin de naam van een moeder, de naam van een vader en een lijst van kinderen. Elk kind heeft de eigenschappen 'Voornaam' en 'Leeftijd'.

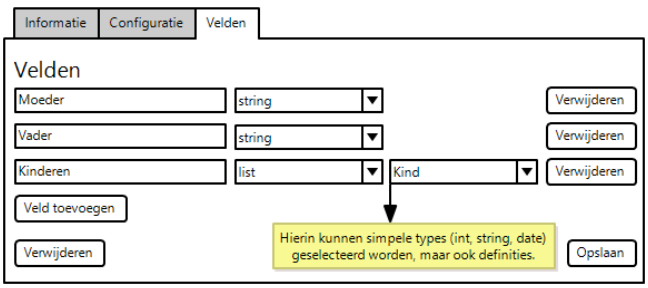
Deze velden kunnen binnen MOBO in het inputkanaal als volgt geconfigureerd worden:



Figuur 9 : Configuratie velden inputkanaal

In deze configuratie kunnen de namen van de velden opgegeven worden, inclusief het type van het veld (bijvoorbeeld een int, string of double). Zoals in de afbeelding te zien is, is er geen mogelijkheid om een lijst te selecteren. Hierdoor is het niet mogelijk om aan te geven dat een veld van het type lijst is en wat voor soort items de lijst bevat. In het geval van het voorbeeld zou er aangegeven moeten kunnen worden dat er veld 'Kinderen' is, van het type 'list', bestaande uit een lijst van het type 'Kind'.

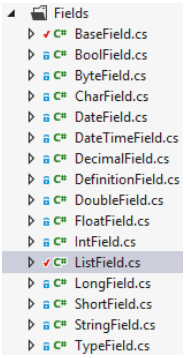
Om deze configuratie mogelijk te kunnen maken, moest er een oplossing bedacht worden. Hiervoor zijn er diverse ontwerpen gemaakt die aansluiten op het huidige uiterlijk van de applicatie. De verschillende ontwerpen staan in 'Bijlage 4 : Schermontwerpen veldconfiguratie' beschreven, waarbij uiteindelijk het volgende ontwerp als definitief ontwerp gekozen is:



Figuur 10 : Ontwerp configuratie lijst

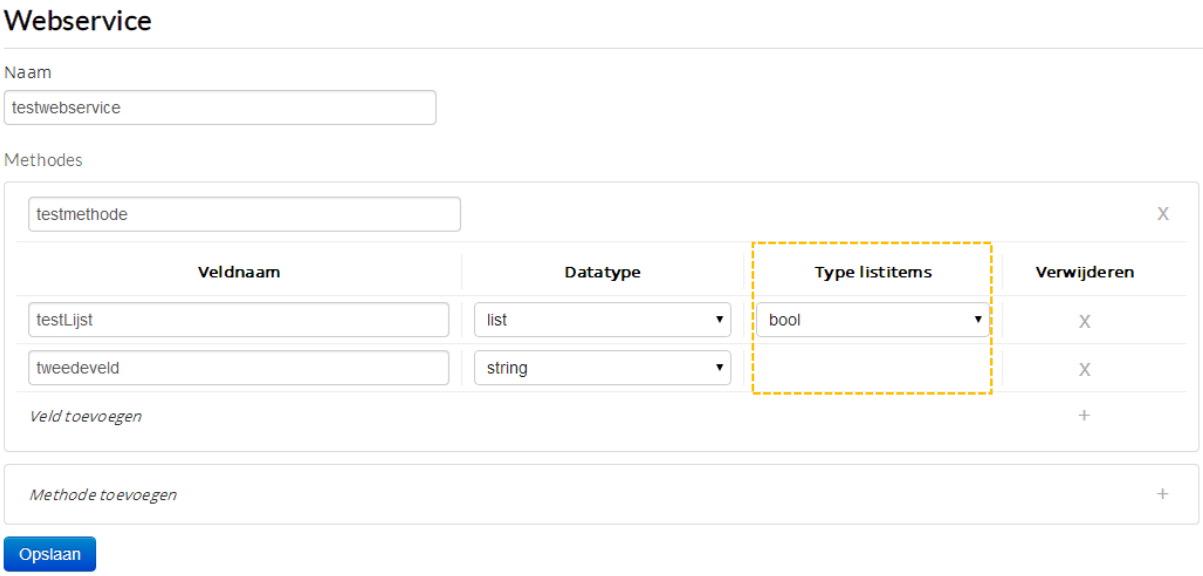
Door bij een veld de optie 'list' te selecteren, krijgt de gebruiker de mogelijkheid om aan te geven wat voor items er in een lijst zitten. Dit kan een lijst van 'strings' zijn, een lijst van 'ints', maar ook van alle andere overige types die beschikbaar zijn. Ook is het mogelijk om een definitie te selecteren. Een definitie kan meerdere velden bevatten, bijvoorbeeld een Voornaam en Leeftijd. Om een lijst van kinderen te selecteren die ieder een Voornaam en Leeftijd bevat, kan er een definitie 'Kind' met daarin deze velden aangemaakt worden. Vervolgens kan de definitie aan de lijst gekoppeld worden.

Om het ontwerp te kunnen realiseren, moest er een nieuw veldtype ‘list’ aangemaakt worden, waarbij dit type een extra eigenschap heeft waarin opgeslagen kan worden wat voor types er in de lijst zitten. Elk veld dat in MOBO gedefinieerd is, is opgebouwd uit een klasse BaseField waarin opgeslagen staat hoe het veld heet en van welk type het is. Indien een type extra eigenschappen bevat, bijvoorbeeld het type ‘list’ die een eigenschap TypeOfItems heeft, dan staat dit in de klasse van het specifieke type gedefinieerd. Voor elk type veld is er dus een specifieke klasse aangemaakt waarbij er voor de nieuwe situatie een nieuwe klasse voor het ListField toegevoegd moest worden.



Voor de front-end zijn dus alle locaties aangepast waarbij er velden gedefinieerd kunnen worden. Hiervoor zijn alle afhankelijke .html, .css en .coffee bestanden uitgebreid en zijn waar nodig extra bestanden toegevoegd.

Het is ook mogelijk om een Webservice-inputkanaal aan te maken. Hierbij kan de gebruiker een methode aanmaken om daar vervolgens velden in te plaatsen.



Figuur 11 : Webservice configuratie – nieuw ontwerp

Opmerking: In de oude situatie was de kolom ‘Type listitems’ niet aanwezig.

Na het aanmaken van de webservice zal er een WSDL (Web Services Description Language) bestand aangemaakt worden waarmee de webservice vervolgens aangeroepen kan worden. Omdat de gebruiker de methodes en velden zelf aan kan maken, dient het genereren van het WSDL bestand generiek opgezet te worden. Om de WSDL bestanden te generen, wordt er gebruik gemaakt van T4 (Text Template Transformation Toolkit) die op basis van de ingevoerde gegevens de code in de WSDL bestanden genereert.

Aangezien de structuur waarop de lijsten verwerkt moeten worden bekend is en omdat de lijst-velden gedefinieerd konden worden, kon de template die de code voor het WSDL bestand genereert aangepast worden.

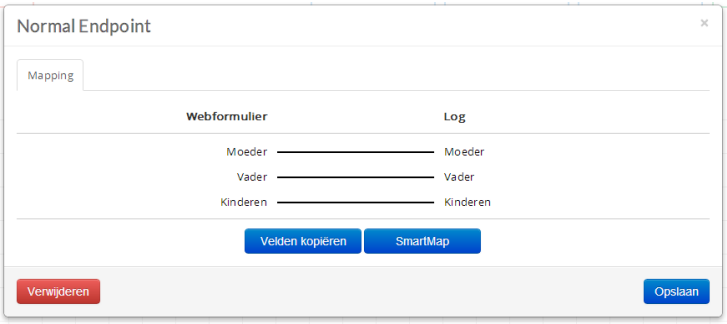
Om een idee te geven hoe de WSDL-bestanden nu gegenereerd worden, is de oude en nieuwe code van de T4-template op de CD bijlage opgenomen. Zie hiervoor het bestand ‘CodeBijlage 1 – WSDL.cs’.

Na het realiseren van de juiste template, waardoor MOBO via een webservice eenvoudig lijsten kan ontvangen, is er begonnen met het onderzoeken naar de verschillende scenario’s waarop de lijsten toegepast kunnen worden.

7.4. Mapping

Nadat een inputkanaal (met lijsten) geconfigureerd is en het voor MOBO duidelijk is welke velden het systeem moet verwerken, moet de volgende stap geconfigureerd worden: de mapping. Bij de mapping kunnen de velden van het inputkanaal aan de velden van het outputkanaal gekoppeld worden.

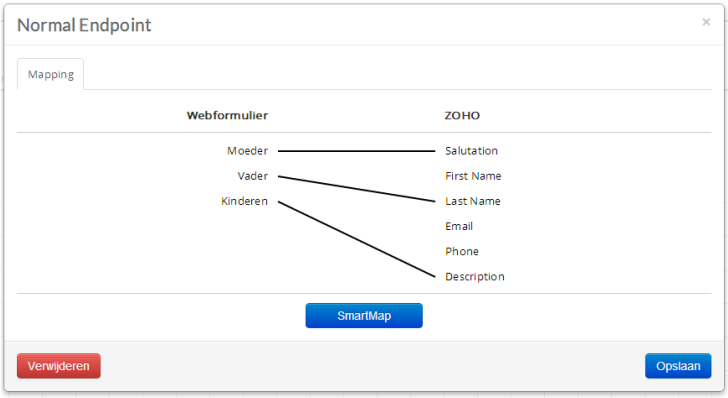
In de afbeelding hieronder wordt aangegeven hoe de velden van een inputkanaal (in dit geval een webformulier) verbonden kunnen worden met de velden van het outputkanaal (in dit geval een log).



Figuur 12 : Mapping van Webformulier naar Log

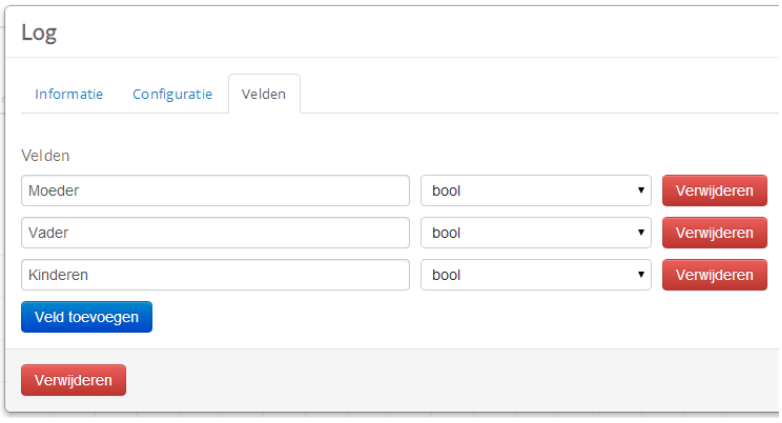
Bij de mapping tussen het inputkanaal en het outputkanaal kunnen lijnen getrokken worden tussen een veld aan de inputzijde en een veld aan de outputzijde. Hierdoor kan de waarde van bijvoorbeeld het veld 'Moeder' van het inputkanaal naar veld 'Moeder' op het outputkanaal gestuurd worden. Aangezien een log geen vaste outputvelden heeft, maar deze door de gebruiker bepaald kunnen worden, kunnen de velden in deze situatie gewoon van het inputkanaal naar het outputkanaal gekopieerd worden.

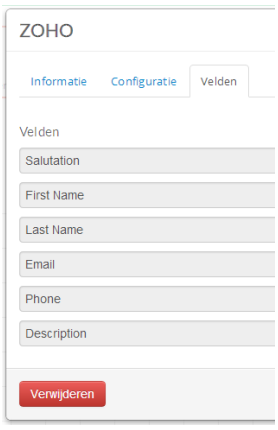
Er kan echter ook een mapping plaatsvinden tussen een input- en outputkanaal waarbij de velden vooraf gedefinieerd zijn. Zo'n situatie doet zich voor bij ZOHO CRM en Webservices. Hierbij worden de velden uit een WSDL bestand geladen. In dit geval kan het voorkomen dat de velden aan de inputzijde anders zijn dan aan de outputzijde. Hieronder is een extreme situatie geschetst waarbij de namen ongelijk zijn. Ondanks dat de velden niet aan beide kanten gelijk zijn, is het door lijnen te trekken toch mogelijk om de velden aan elkaar te verbinden.



Figuur 13 : Mapping van Webformulier naar ZOHO

In de afbeeldingen hieronder is het verschil te zien tussen een outputkanaal waarbij de velden geconfigureerd kunnen worden en een outputkanaal waarbij de velden vooraf gedefinieerd zijn.



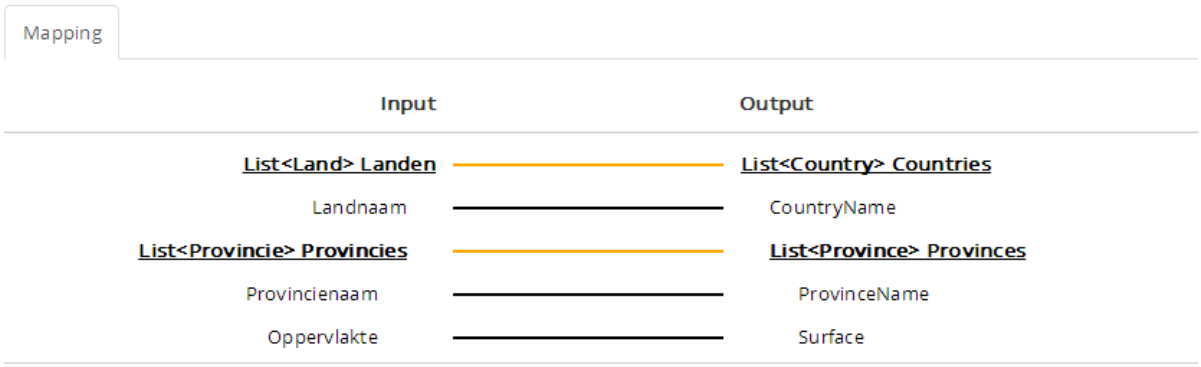


Figuur 14 : Verschil in configuratie velden tussen categorie 1 en categorie 2

In de oude situatie was het nog niet mogelijk om velden van het type lijst te mappen. Om lijsten van een input- naar een outputkanaal te mappen, moest de mapping dus aangepast worden. Zoals in de inleiding van dit hoofdstuk vermeld is, kunnen er zich verschillende scenario's voordoen. Er is daarom een aantal scenario's onderzocht. Hierbij zijn de ontwerpen gebouwd aan de hand van de huidige lay-out van MOBO en de wensen van de klant. Tevens is er getracht om literatuur of voorbeelden uit andere applicaties te gebruiken. Helaas is hier weinig bruikbare informatie uit voortgekomen.

7.4.1. Scenario 1: Één op één mapping

Het eerste scenario dat zich voor kan doen is het mappen van een lijst op lijst, waarbij beide lijsten dezelfde structuur hebben. Om tot een goed ontwerp voor de oplossing te komen, zijn er verschillende wireframes ontworpen, welke in 'Bijlage 5 : Schermontwerpen Scenario 1' uitgewerkt staan. Het laatste ontwerp is uiteindelijk gerealiseerd:

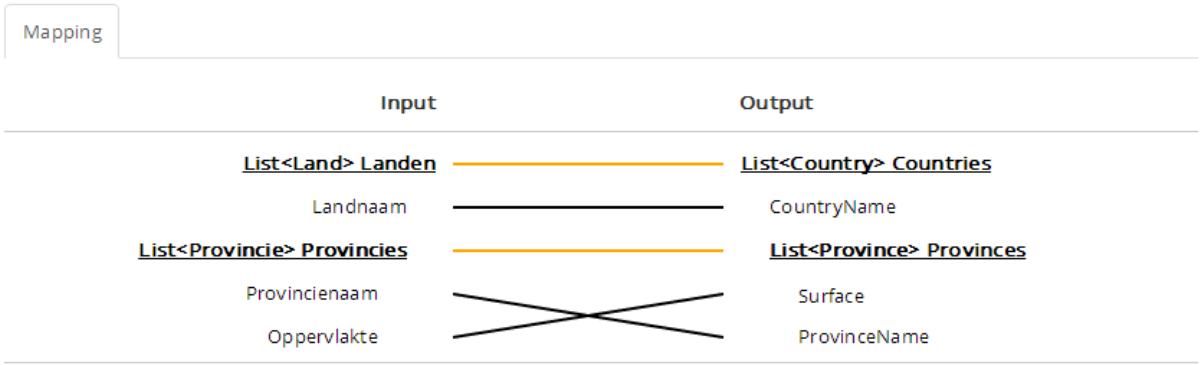


Figuur 15 : Mapping scenario 1

Opmerking: De velden 'Landnaam' en 'Provincies' komen voort uit de definitie 'Land'. De velden 'Provincienaam en 'Oppervlakte' komen voort uit de definitie 'Provincie'. Deze definities zijn elders binnen MOBO gedefinieerd.

Om duidelijk aan te geven welke velden in de mapping van het type 'list' zijn, is er gekozen om de stijl hetzelfde te houden zoals lijsten in de code verwerkt worden. Een lijst wordt dus als volgt weergegeven: "List<item> name". Ook is de naam dikgedrukt en onderstreept, waardoor het duidelijk is dat het om een speciaal soort item gaat. Door de velden van de items in een lijst een left/right marge te geven, wordt het duidelijk dat deze velden bij de lijst horen. De oranje lijn maakt het duidelijk dat er een mapping tussen twee lijsten is.

Voor de realisatie van dit scenario leek er in eerste instantie weinig aan de mapping aangepast te moeten worden. De lijsten worden in dit scenario namelijk één op één gekopieerd, wat in principe met de oude situatie overeenkomt. Desondanks kan er een verandering in de lijst optreden, bijvoorbeeld zoals in de volgende situatie is geschetst:



Hierbij kan de lijst niet meer één op één gekopieerd worden, aangezien de velden in een lijst-item op een andere plek staan. Om beide situaties te kunnen ondersteunen, is eerst de front-end gerealiseerd. Hierbij zijn er diverse checks ingebouwd zodat lijsten bijvoorbeeld alleen op lijsten gemapt kunnen worden.

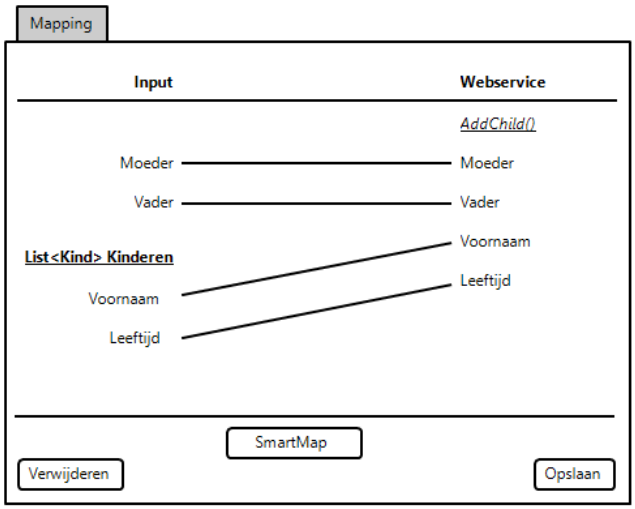
In de oude situatie werden de waardes aan de inputzijde eenvoudig gekopieerd naar de outputzijde. Aangezien er in de oude situatie geen lijsten voorkwamen, waren de hoeveelheid input- en outputvelden altijd bekend. Deze velden zijn tenslotte in het controlpanel gedefinieerd.

Op de CD bijlage staat in 'CodeBijlage 2 – Mapping.cs', onder het kopje 'OUDE CODE', de oude code verwerkt.

Omdat er in de nieuwe situatie lijsten verwerkt moeten worden, moet een bepaalde mapping voor meerdere lijst-items herhaald worden. Indien er een 'Voornaam' uit een lijst op een 'Achternaam' uit een lijst gemapt wordt, moet deze mapping voor ieder item in de lijst plaatsvinden. Om deze situatie te kunnen verwerken, is er een recursieve methode opgesteld die elk item in iedere lijst doorloopt. Vervolgens wordt er gecontroleerd of er een mapping aanwezig is waarna ieder (lijst-)item op de outputzijde gemapt wordt. De nieuwe code die hieruit is voortgekomen is in dezelfde bijlage onder het kopje 'NIEUWE CODE' opgenomen.

7.4.2. Scenario 2: Mapping van een lijst op een outputkanaal zonder lijst

Het kan ook voorkomen dat er bij het inputkanaal een lijst aanwezig is, terwijl er op het outputkanaal geen lijst aanwezig is. In dit geval kan de mapping er als volgt uit komen te zien:

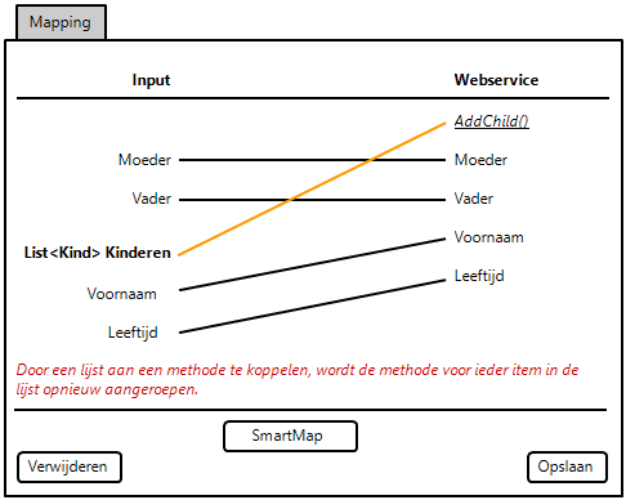


Figuur 16 : Mapping lijst op een outputkanaal zonder lijst

In deze situatie worden lijst-items (aan de linkerkant) op enkele items (aan de rechterkant) gemapt. De data zal hierdoor niet correct opgeslagen worden. Indien er namelijk vijf kinderen in de lijst zitten, zullen er vijf voornamen en vijf leeftijden uitkomen. Het is niet mogelijk om vijf voornamen of leeftijden op één veld op te slaan.

Om de service toch goed uit te kunnen voeren, moest er een optie bedacht worden om de lijst op een manier te mappen waardoor de lijst alsnog op de juiste manier opgeslagen kan worden. De oplossing die hiervoor in eerste instantie gevonden werd is het "mappen van een lijst op een methode". Door een lijst aan een methode te verbinden (zie afbeeldingen hieronder), wordt voor elk item in de lijst (bijvoorbeeld voor elk Kind in de lijst Kinderen) de methode (in het geval van het voorbeeld AddChild) uitgevoerd. Indien een lijst vijf items (vijf Kinderen) bevat, zal de methode AddChild dus vijf keer uitgevoerd worden.

Dit scenario zou zowel op een Webservices als op ZOHO toepasbaar moeten zijn, aangezien deze outputkanalen methodes bevatten. De andere kanalen, zoals Mail en Log, bevatten geen methodes.



Figuur 17 : Mappen van een lijst op een methode

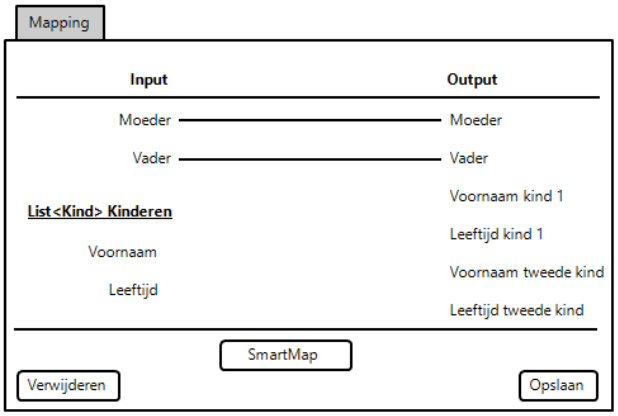
Het scenario is vervolgens met Renso Bek besproken. Hieruit bleek dat hij deze oplossing niet duidelijk genoeg vond. Lijnen komen namelijk allemaal door elkaar te staan, wat bij een complexe mapping erg onduidelijk kan worden. Tevens bleek scenario 1 uitgebreider dan vooraf gedacht was. Want wat als een outputkanaal zonder methode, bijvoorbeeld het Mail-outputkanaal, per lijst-item uitgevoerd moet worden?

Aangezien de mapping van een lijst op een methode niet aan de wensen van de klant voldoet, is dit ontwerp geschrapt. Hier moest dus een andere oplossing voor bedacht worden. Aangezien er binnen de mapping weinig andere mogelijkheden meer waren, moest deze oplossing op een ander niveau gevonden worden. Om deze situatie te kunnen realiseren is uiteindelijk een Foreach-component ontworpen, zie hiervoor 'Hoofdstuk 7.5. : Foreach-component'.

7.4.3. Scenario 3: Mapping van lijst naar platte structuur

Dit scenario komt redelijk overeen met scenario 2, waarbij er aan de outputzijde geen lijst aanwezig is. In dit scenario is er ook geen lijst aanwezig, maar wel een soort lijststructuur, waarbij de lijst als het ware platgeslagen is. In dit scenario moet het mogelijk zijn om een lijst van het inputkanaal door te sturen naar een outputkanaal zonder lijst, maar waarbij er voor een aantal items in de lijst wel een veld op het outputkanaal beschikbaar is.

Een voorbeeld hiervan zou er als volgt uit kunnen zien:



Dit scenario lijkt in eerste instantie onwaarschijnlijk, aangezien de hoeveelheid items in een lijst vooraf niet bekend is en de hoeveelheid velden aan de outputzijde wel. Desondanks kan dit scenario volgens de product owner voorkomen. Indien een lijst aan de inputzijde meer items bevat dan dat er aan de outputzijde aanwezig zijn, zou er een manier bedacht kunnen worden om dit probleem af te vangen. Dit valt voor dit project echter buiten de scope.

Aangezien het niet mogelijk is om lijnen te trekken tussen een lijst-item en een eenvoudig veld, moest hier eerst een oplossing voor bedacht worden.

Net als het vorige scenario is dit niet binnen de mapping mogelijk. Het blijkt dat de mapping, het tekenen van de lijnen tussen de velden, veel te basis is om een complexe mapping mee op te bouwen. Van alle scenario's die tot nu toe onderzocht zijn, blijkt er maar één toepasbaar in de mapping. Dit scenario is ook het simpelste scenario, waarbij lijsten op lijsten gemapt worden die (ongeveer) dezelfde structuur hebben. Zodra het complexer wordt, is de mapping geen goede oplossing meer. De overige scenario's zullen dus op een ander niveau binnen MOBO geconfigureerd moeten worden.

7.5. Foreach-component

Aangezien er in verschillende scenario's lijsten ondersteund moeten gaan worden, lag het uiteindelijk voor de hand om een component te maken waarmee lijsten doorlopen kunnen worden. Wat dit component precies allemaal moest gaan ondersteunen was vooraf niet precies bekend. Desondanks was het wel duidelijk dat er een oplossing moest komen om meer met de lijsten te kunnen doen.

In de oude situatie was het mogelijk om één of meerdere componenten tussen een input- en outputkanaal toe te voegen. Er waren binnen MOBO vier componenten beschikbaar:

Figuur 18 : Componenten

- **Fallback:** In het Fallback-component kan een Fallback-scenario ingesteld worden. Hierbij kan een gebruiker aangeven wat MOBO moet doen als een bepaald outputkanaal niet bereikbaar is. In dit geval kan de gebruiker bijvoorbeeld instellen dat er dan een mail naar de administratie verstuurd moet worden.
- **IF:** Het IF-component heeft twee soorten uitgangen: IF en ELSE. Bij de IF-uitgang kan de gebruiker een controle instellen (if naam is gelijk aan “HU” || if naam is niet null) en aangeven welk outputkanaal vervolgens uitgevoerd moet worden. Een ELSE-uitgang kan niet ingesteld worden. Het outputkanaal dat hieraan gekoppeld zit, wordt uitgevoerd indien geen van alle IF-uitgangen de ingestelde controle doorstaan.
- **Transformatie:** In het Transformatie-component kan de gebruiker een transformatie instellen. De gebruiker kan bijvoorbeeld instellen dat de velden ‘Voornaam’ en ‘Achternaam’ aan elkaar geplakt moeten worden.
- **Validatie:** In het Validatie-component kan de gebruiker een controle instellen om te garanderen dat de data die binnenkomt aan alle eisen voldoet. Indien de controles niet slagen, dan zal de transactie geannuleerd worden.

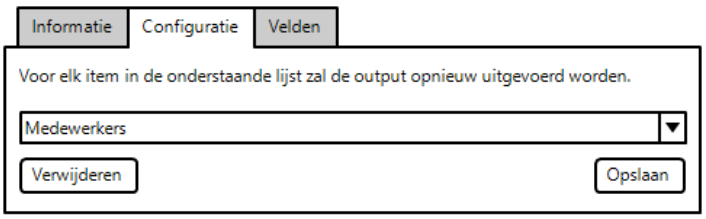
Om in de loop van het project steeds meer lijst-scenario’s te ondersteunen, is er een nieuw component toegevoegd, namelijk het **Foreach**-component. Dit component is tot stand gekomen dankzij overleg met collega’s en door naar een aantal voorbeelden in het systeem OutSystems te kijken. (OutSystems, 2014) Wat het Foreach-component na oplevering allemaal moet kunnen, wordt in de volgende subhoofdstukken toegelicht.

7.5.1. Uitvoeren van een outputkanaal per lijst-item

Omdat scenario 2 (Mapping van een lijst op een outputkanaal zonder lijst) niet in de mapping opgelost kon worden, moest hier een andere oplossing voor gevonden worden. Na het introduceren van het Foreach component, lag het voor de hand om dit component voor dit scenario te gebruiken.

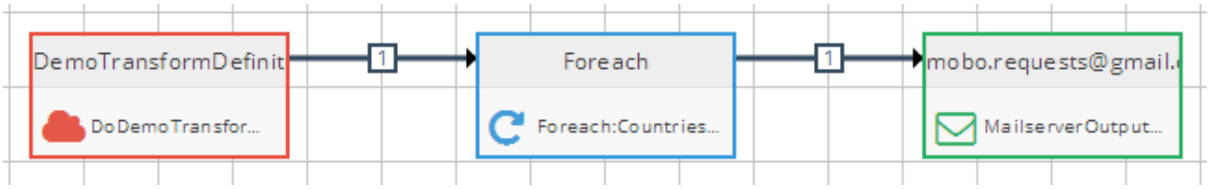
Aangezien een inputkanaal meerdere lijsten kan bevatten, bijvoorbeeld een veld ‘Bedrijfsnaam’, een lijst ‘Medewerkers’ en een lijst ‘Vestigingen’, moet er in het component aangegeven kunnen worden op welke lijst de ‘foreach’ van toepassing moet zijn.

Hieruit is het volgende ontwerp naar voren gekomen:



Figuur 19 : Foreach-component

Door in de configuratie van het Foreach-component een lijst te selecteren, is het voor MOBO duidelijk welke lijst steeds opnieuw doorlopen moet worden. Het component zal vervolgens het outputkanaal meerdere malen uitvoeren. Een service met een Foreach-component zal er als volgt uitzien:



Figuur 20 : Mapping Foreach-component

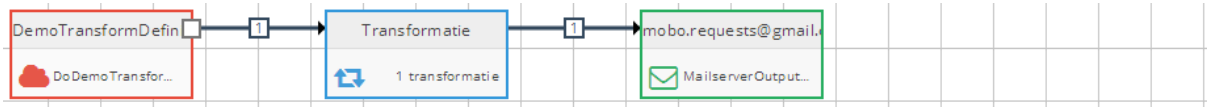
De lijst die in de configuratie van het Foreach-component geselecteerd is en via de testwebservice binnenkomt, zal steeds opnieuw doorlopen worden. Per lijst-item zal er vervolgens een email verstuurd worden.

7.5.2. Acties uitvoeren op lijst-items

Het is mogelijk om binnen MOBO een validatie, transformatie of een if/else scenario op velden toe te passen. In de oude situatie controleert MOBO de opgegeven validatie/transformatie voor een specifiek veld. Een validatie of transformatie voor items in lijsten wordt echter veel complexer. In dat geval moet de validatie of transformatie niet eenmaal uitgevoerd worden, maar voor ieder lijst-item.

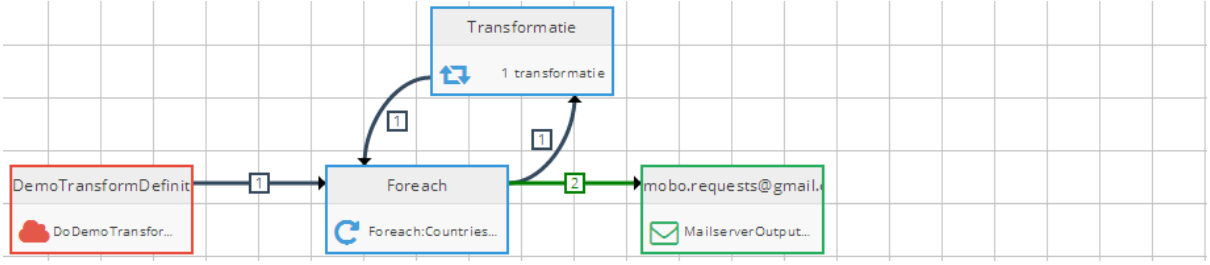
Vanwege de introductie van het Foreach-component is de vraag gesteld welke oplossing hiervoor bedacht moest worden. Er zijn namelijk twee mogelijke oplossingen:

Oplossing 1:
De eerste oplossing is om de huidige componenten aan te passen om ervoor te zorgen dat de componenten lijsten ondersteunen. Indien er een transformatie voor een lijst-item opgegeven wordt, zal het component de lijst meerdere malen moeten doorlopen om de transformatie op ieder lijst-item toe te kunnen passen. Visueel is het niet heel duidelijk, maar functioneel komt het met de tweede oplossing overeen.



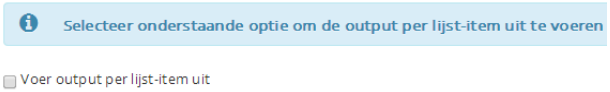
Figuur 21 : Oplossing 1 - lijsten in componenten

Oplossing 2:
Onderstaande afbeelding laat zien dat het ook een optie is om het Foreach-component uit te breiden waardoor er een 'loop' gemaakt kan worden. In dit geval wordt het Foreach-component niet gebruikt om de output per lijst-item aan te roepen, maar om voor ieder lijst-item een transformatie te starten.

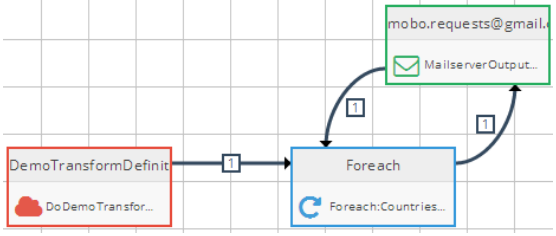


Figuur 22 : Oplossing 2 - lijsten in componenten

Indien er voor deze oplossing gekozen wordt, zijn er vervolgens twee mogelijkheden om de output per lijst-item aan te roepen. Er kan gekozen worden om het tweede endpoint (de groene lijn) van een optie te voorzien waarmee de gebruiker aan kan geven dat de output per lijst-item aangeroepen moet worden. Een andere oplossing is om het outputkanaal in de loop te plaatsen, waardoor er in plaats van de transformatie het outputkanaal per lijst-item aangeroepen wordt.

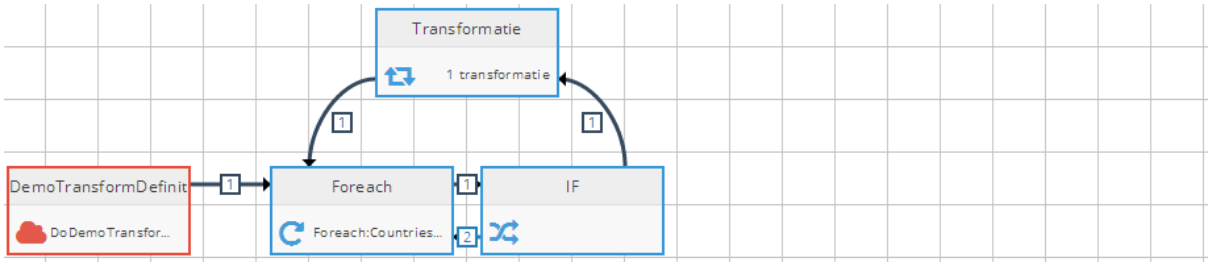


Figuur 23 : Mogelijkheid 1 - Output per lijst-item



Figuur 24 : Mogelijkheid 2 - Output per lijst-item

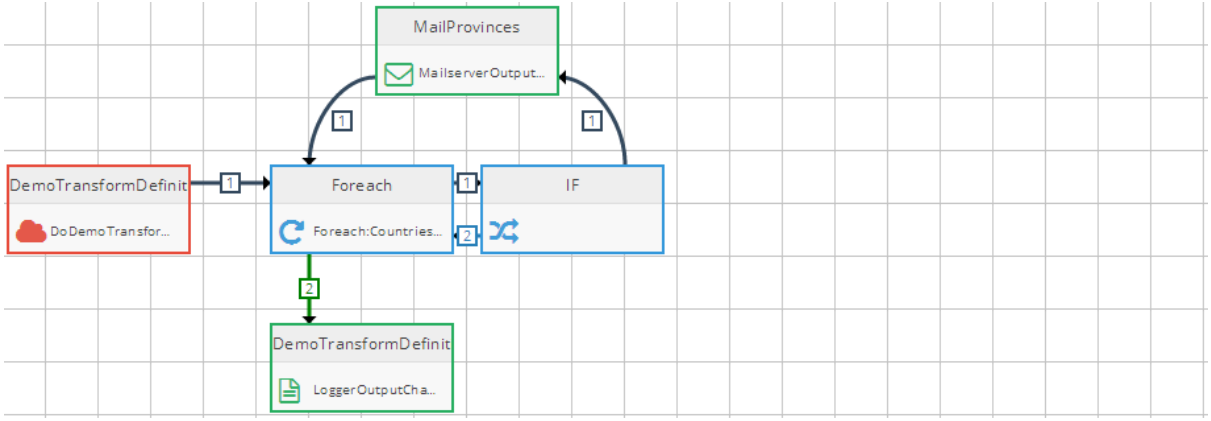
Keuze tussen Oplossing 1 en Oplossing 2:
Er is besloten om voor de tweede oplossing te kiezen. Door standaard het Foreach-component te gebruiken op het moment er een bepaalde actie op de lijst uitgevoerd moet worden, is het visueel direct duidelijk wat er gebeurt. Aangezien MOBO door bijvoorbeeld secretaresses zonder programmeerkennis geconfigureerd moet kunnen worden, moet elke functionaliteit zo duidelijk mogelijk zijn. De tweede oplossing is hierin het duidelijkst, zeker indien er bijvoorbeeld een validatie- en transformatie-component na elkaar geplaatst worden, zoals in de situatie op volgende bladzijde.



Figuur 25 : Foreach met Transformatie- & IF-component

In deze situatie wordt de transformatie enkel uitgevoerd indien de data aan de opgegeven controle voldoet. Indien dit niet zo is, zal het volgende item in de lijst doorlopen worden.

Na overleg met de product owner is er besloten om de output per lijst-item op dezelfde manier te laten configureren, waardoor de volgende situatie ontstaat:



Figuur 26 : Mogelijkheid 2 - Output per lijst-item

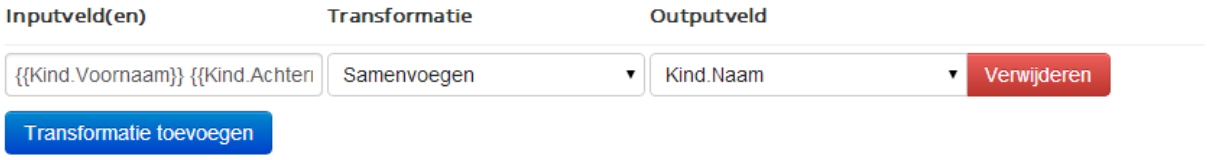
Hierbij kan er na het Foreach-component een loop gemaakt worden om het outputkanaal per lijst-item uit te voeren en kan er een endpoint toegevoegd worden die uitgevoerd wordt zodra alle items doorlopen zijn.

Door het maken van deze oplossing is er direct een oplossing gevonden voor een nieuw scenario: het mappen van een lijst op een aangepaste lijst.

7.5.3. Mappen van een lijst op een aangepaste lijst

Nu er de mogelijkheid is om door middel van het Foreach-component lijsten te doorlopen, kan MOBO lijsten qua structuur aanpassen door een transformatie-component na het Foreach-component te plaatsen om de lijst vervolgens te mappen op een lijst met een andere structuur. Dit wil zeggen: er kan een lijst bij het inputkanaal binnenkomen met lijst-items die ieder 3 eigenschappen hebben, bijvoorbeeld een lijst van kinderen met de eigenschappen ‘voornaam’, ‘achternaam’ en ‘leeftijd’. Vervolgens moet deze lijst gemapt kunnen worden op een lijst van kinderen met de eigenschappen ‘naam’ en ‘leeftijd’. In dit geval moeten de eigenschappen ‘voornaam’ en ‘achternaam’ per lijst-item samengevoegd worden.

In het transformatie-component kunnen de inputvelden op de volgende manier opgenomen worden: *{{Kind.Voornaam}} {{Kind.Achternaam}}*. Hierbij worden de velden tussen *{{ }}* opgenomen. De transformatie van deze velden kan vervolgens op het veld Kind.Naam opgeslagen worden. Dit ziet er in het Transformatie-component als volgt uit:



Figuur 27 : Transformatie lijst-items

Nadat de transformatie per lijst-item is uitgevoerd, zal de nieuwe lijst naar het volgende component doorgestuurd worden, bijvoorbeeld naar een Webservice-outputchannel.

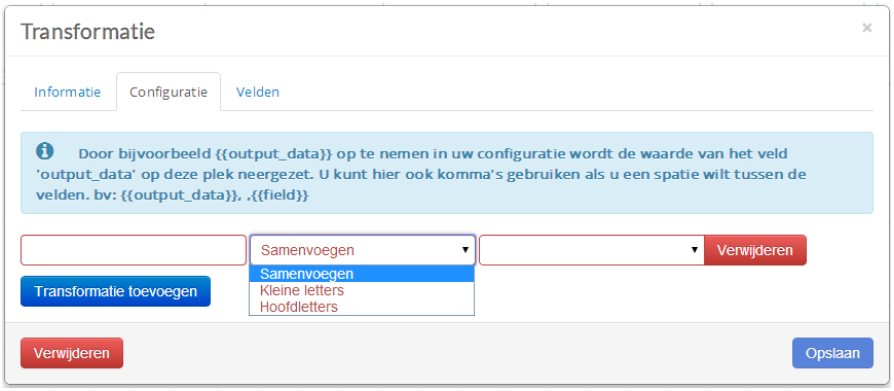
7.6. Lijst-component

Voor alle mogelijke situaties is er nu een oplossing gevonden, behalve voor de situatie waarbij er een lijst op een platte structuur gemapt moet kunnen worden, zoals in ‘Hoofdstuk 7.4.3. : Scenario 3: Mapping van lijst naar platte structuur’ beschreven staat.

Omdat het niet mogelijk is om deze situatie in de mapping te configureren, moest er net als voor de andere scenario’s een geschikte oplossing gevonden worden. Er is in eerste instantie getracht de situatie in een huidig component te verwerken, om de hoeveelheid componenten te beperken.

Mogelijke oplossing in het Transformatie-component
Allereerst is er geprobeerd de situatie via het Transformatie-component te ondersteunen. De gebruiker moet aan kunnen geven welke velden uit welk lijst-item (het eerste, het tweede, het derde, etc.) op welk outputveld gemapt moet worden. Aangezien deze handeling een soort van transformatie is, lijkt het voor de hand te liggen om iets met het Transformatie-component te doen.

Het oude Transformatie-component ziet er als volgt uit:



Figuur 28 : Oude Transformatie-component

De gebruiker krijgt in het component drie velden te zien. In het eerste veld kunnen de velden opgenomen worden die getransformeerd moeten worden, in het tweede veld kan een transformatie gekozen worden en in het derde veld moet een veld geselecteerd worden waar de transformatie op opgeslagen moet worden.

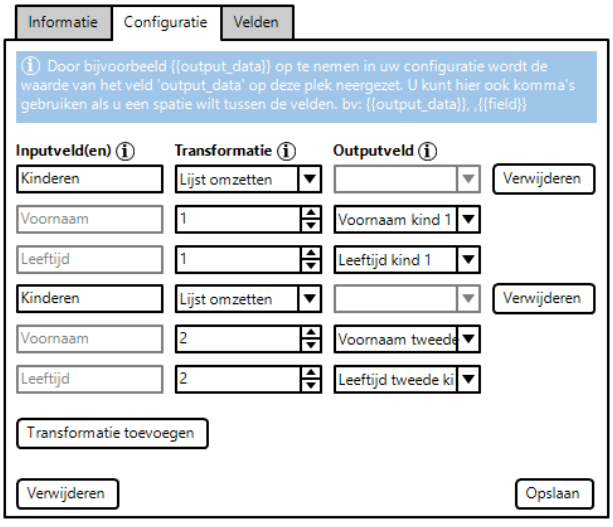
Door in het eerste veld bijvoorbeeld de velden {{voornaam}}.{{achternaam}} op te nemen en de transformatie ‘Samenvoegen’ te selecteren, zullen de velden ‘voornaam’ en ‘achternaam’ samengevoegd worden met een punt ertussen.

Figuur 29 : Voorbeeld transformatie

Om een lijst naar een platte structuur te kunnen transformeren, is in eerste instantie de volgende oplossing bedacht:

Indien een gebruiker bij het inputveld een lijst selecteert en als transformatie ‘Lijst omzetten’ kiest, zullen de velden van ieder item in de lijst getoond worden. Per veld kan de gebruiker vervolgens een nummer invullen. Dit nummer is de index van het item in de lijst. In de afbeelding hiernaast is te zien dat de Voornaam van het eerste lijst-item omgezet moet worden naar het veld ‘Voornaam kind 1’ en de Leeftijd van het eerste lijst-item naar het veld ‘Leeftijd kind 1’.

Op deze manier zou de gebruiker een onbeperkt aantal transformaties voor alle lijst-items toe kunnen voegen.



Figuur 30 : Nieuw Transformatie-component

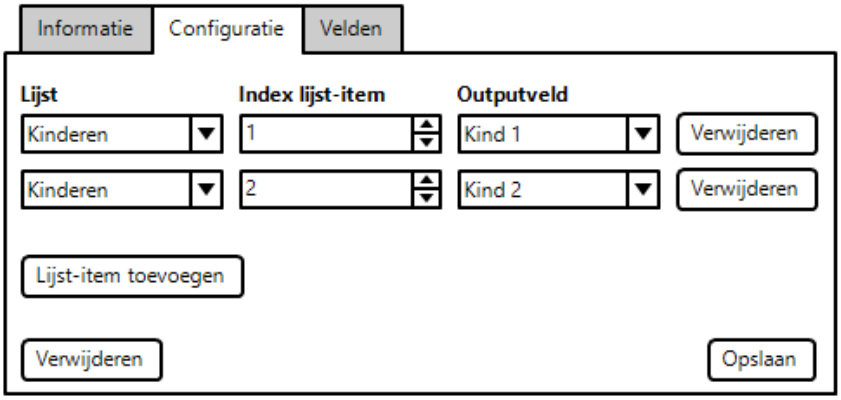
Bij nader inzien bleek deze oplossing echter niet ideaal. Om dit scenario in het Transformatie-component te kunnen ondersteunen, dient het hele component opnieuw opgebouwd te worden. Tevens is de weergave niet duidelijk en is de bediening niet eenvoudig, aangezien er bij grote lijsten enorm veel velden ingevuld moeten worden.

Aangezien de overige componenten geen transformaties ondersteunen, maar enkel lijsten kunnen doorlopen of waardes kunnen valideren, is er binnen deze bestaande componenten geen oplossing gevonden. Literatuuronderzoek heeft hierbij ook geen oplossing kunnen bieden. Er is daarom besloten om een nieuw component te maken, namelijk het 'Lijst-component'.

Oplossing in het Lijst-component
Om de visuele weergave en de bediening zo gebruiksvriendelijk mogelijk te houden, is er geprobeerd om zo weinig mogelijk velden in het component op te nemen. Er is daarom een lijst opgesteld met wat de gebruiker op moet kunnen geven. Hieruit is de volgende lijst gekomen:

- Aan kunnen geven welke item (de index) uit welke lijst gebruikt moet worden
- Aan kunnen geven op welk veld het item opgeslagen moet worden

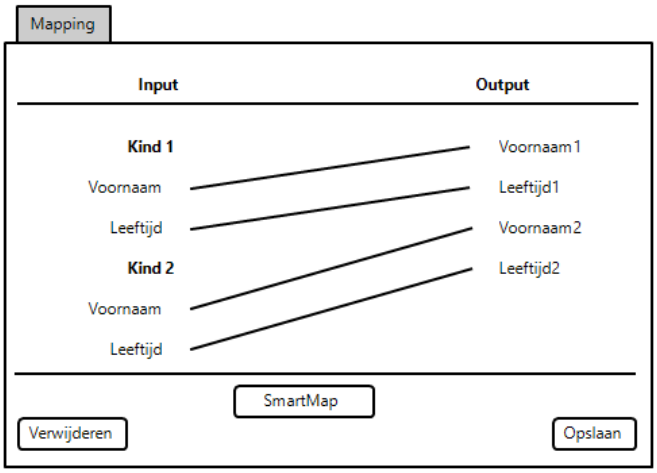
Hieruit blijkt dat er drie instellingen nodig zijn. Er moet kunnen worden aangeven welke lijst, welk item in de lijst (index) en welk outputveld er gebruikt moet worden. Hieruit is het volgende ontwerp naar voren gekomen:



Figuur 31 : Lijst-component

Door de lijst 'Kinderen' te selecteren, de index '1' op te geven en bij het outputveld 'Kind 1' in te stellen, zal het eerste item in de lijst 'Kinderen' naar het veld 'Kind 1' gekopieerd worden. Op deze manier hoeven de eigenschappen van het Kind (Voornaam en Leeftijd) niet handmatig ingesteld te worden, wat in het Transformatie-component wel nodig was.

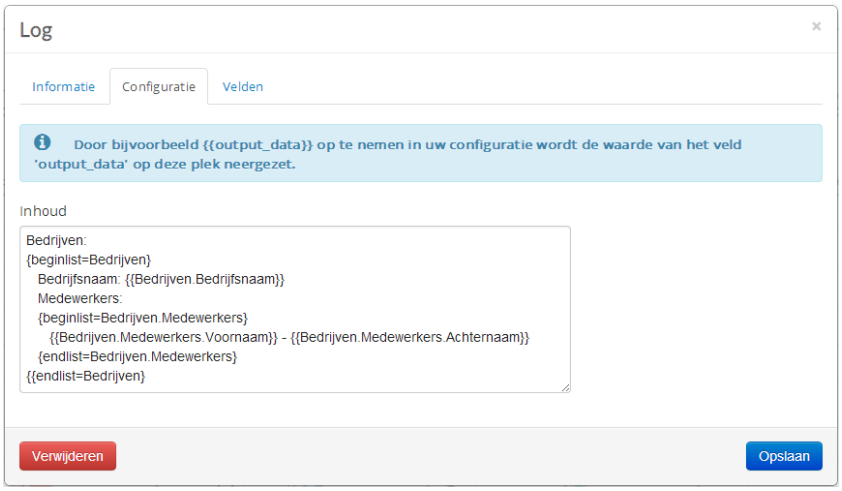
Indien er in de eigenschappen van het kind ook veranderingen plaats moeten vinden, dan is dit uiteindelijk mogelijk om in de mapping te doen, aangezien in de mapping de nieuwe outputvelden op het outputkanaal gemapt moeten worden:



Figuur 32 : Mapping na Lijst-component

7.7. Lijsten in outputconfiguratie

Nadat het inputkanaal geconfigureerd is en de mapping tussen de verschillende componenten is aangebracht, dient het outputkanaal (of -kanalen) geconfigureerd te worden. In een aantal outputkanalen is het mogelijk om een configuratie op te geven. Dit zijn de kanalen waarbij er een tekstuele output gegenereerd wordt. Het gaat hierbij om het Log-, Mail-, Facebook- en Twitter-outputkanaal. Hierbij heeft de gebruiker de mogelijkheid om aan te geven welke velden geëxporteerd moeten worden. Om een veld in het outputkanaal op te nemen, kan het veld in de configuratie eenvoudig tussen *{{* en *}}* opgenomen worden. In de oude situatie konden lijsten echter nog niet opgenomen worden. Er is daarom een nieuwe mogelijkheid gecreëerd om lijsten in de outputconfiguratie op te kunnen nemen.



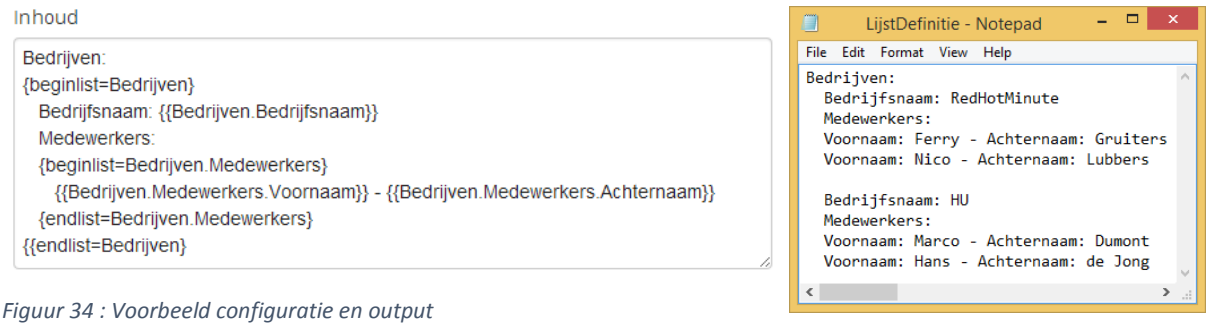
Figuur 33 : Nieuwe situatie outputconfiguratie categorie 1

Door lijsten tussen de tags {beginlist=name} *en* {endlist=name} op te nemen, met daarin de naam van de lijst (dus bijvoorbeeld {beginlist=Kinderen}), is het mogelijk om de lijsten volgens de juiste opmaak te exporteren.

Om de lijsten van definities uiteindelijk in een log op te nemen, moeten de velden in de configuratie opgenomen worden (zie Figuur 34 : Voorbeeld configuratie en output). In de oude situatie worden de opgenomen velden (gedefinieerd als {{veldnaam}}) door middel van een string.Replace methode eenvoudig vervangen door de waarde van een veld met dezelfde naam. Aangezien het doorlopen van lijsten complexer in elkaar zit, moest hier een andere oplossing voor gevonden worden.

Omdat de huidige manier waarop velden in de configuratie opgenomen kunnen worden goed werkt, is er besloten om hier niets aan te veranderen. De optie die dus bijgebouwd moest worden is het opnemen van lijsten. Door naast de string.Replace methodes ook verschillende reguliere expressies toe te passen, kan het systeem de lijst-tags in de configuratie herleiden om vervolgens de velden binnen de tags te vervangen door de waardes van een lijst-item. Deze actie wordt per lijst-item herhaald, waardoor elk lijst-item op de juiste manier in het log opgenomen wordt.

De oude en nieuwe code is op de CD bijlage opgenomen. Zie hiervoor het bestand ‘CodeBijlage 3 – Outputconfiguratie.cs’.



Figuur 34 : Voorbeeld configuratie en output

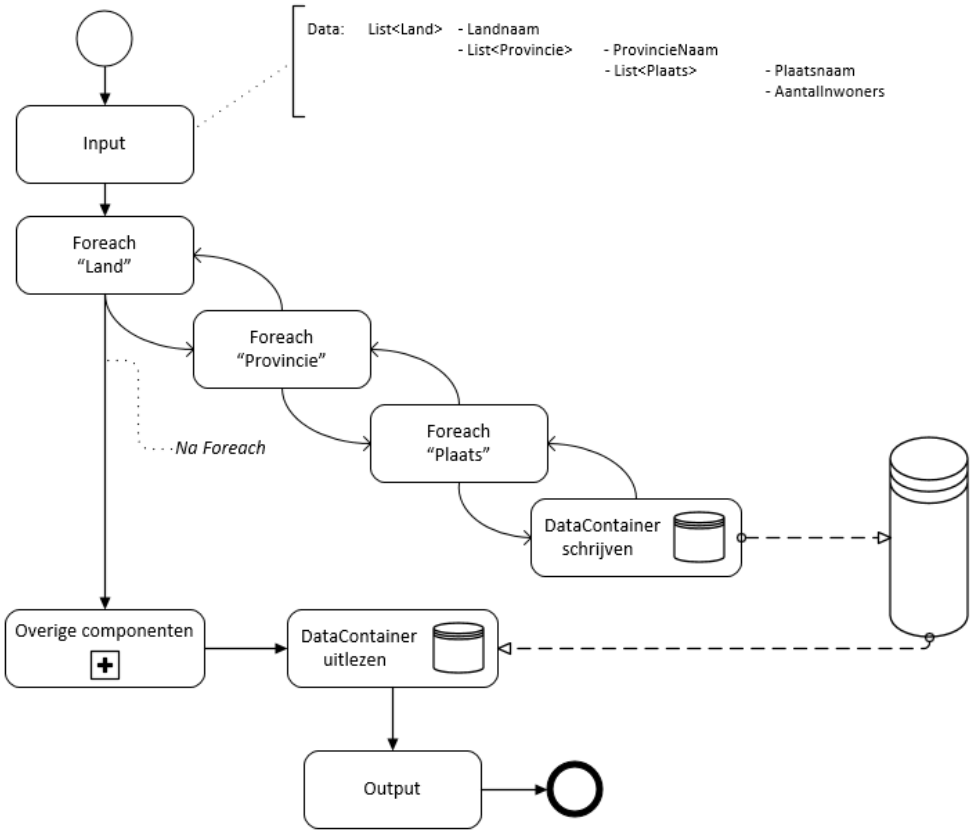
7.8. Realisatie overige componenten

Dankzij de realisatie van het Foreach- en het Lijst-component, in combinatie met de overige aanpassingen, is het nu mogelijk om lijsten in diverse scenario's binnen MOBO toe te passen. Om MOBO uiteindelijk zo flexibel mogelijk te kunnen gebruiken, had de product owner een extra component als wens om MOBO uiteindelijk nog beter te kunnen gebruiken.

7.8.1. DataContainer-component

Het component dat gerealiseerd moest gaan worden, moest data op kunnen slaan om de data op een ander moment in de service vervolgens weer uit te kunnen lezen. Aangezien MOBO gebruik maakt van DataContainers, zal dit component het DataContainer-component worden genoemd. Het moet bijvoorbeeld mogelijk worden om alle plaatsen in de lijst 'Provincies' in de lijst 'Landen' in het component op te slaan. De gecreëerde lijst van plaatsen kan vervolgens op een andere locatie in de service opnieuw gebruikt worden.

Om dit proces te verduidelijken is het volgende schema opgesteld:

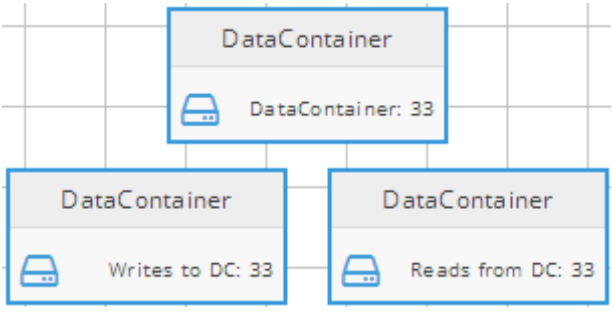


Figuur 35: Schema DataContainer-component

Het moet uiteindelijk mogelijk zijn om op elke willekeurige locatie in de service data in de DataContainer bij te schrijven of uit te lezen. In de configuratie van de service moet de gebruiker uiteindelijk drie mogelijkheden hebben:

- Een nieuwe DataContainer definiëren
- Een DataContainer bijschrijven
- Een DataContainer uitlezen

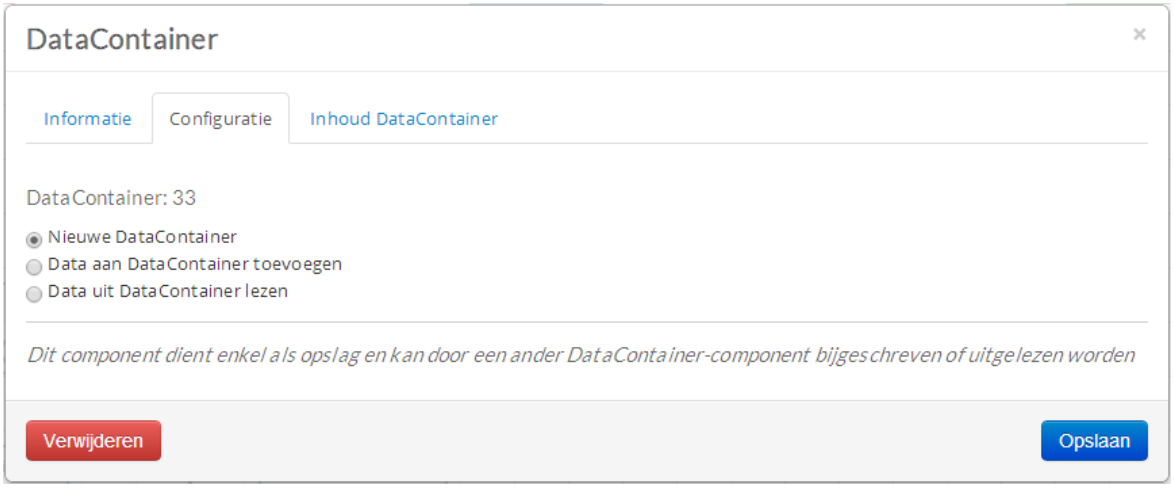
Een DataContainer-component kan dus voor drie oplossingen gebruikt worden: als opslag-component, als schrijf-component voor een bestaand opslag-component en als uitlees-component voor een bestaand opslag-component.



Figuur 36 : Verschillende DataContainer-componenten

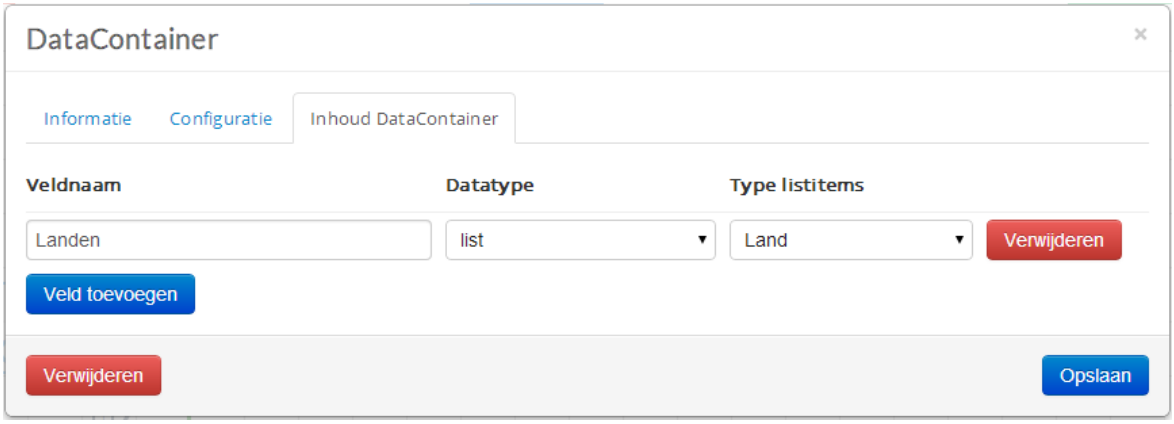
Indien een DataContainer-component gebruikt wordt om data weg te schrijven of uit te lezen, zal er van een DataContainer-component gebruik gemaakt moeten worden die als opslag dient. Er dient dus eerst een DataContainer-component als opslag ingesteld te worden, voordat een ander DataContainer-component als schrijver/lezer ingesteld kan worden.

DataContainer-component als opslag-component
In de configuratie van het component kan ingesteld worden dat het component een nieuwe DataContainer is.



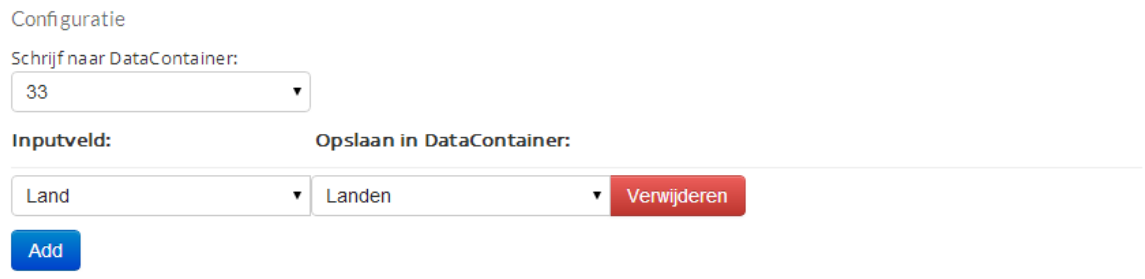
Figuur 37 : DataContainer configuratie

In het tabblad 'Inhoud DataContainer' kunnen vervolgens de velden ingesteld worden waaruit de DataContainer bestaat, bijvoorbeeld uit een lijst van Landen. Deze velden kunnen vervolgens tijdens het uitlezen/bijschrijven gebruikt worden.



Figuur 38 : Inhoud DataContainer

DataContainer-component als schrijf-component
Indien een DataContainer-component gebruikt wordt om een andere DataContainer bij te schrijven, dient de optie 'Data aan DataContainer toevoegen' geselecteerd te worden. De gebruiker dient vervolgens een DataContainer te selecteren waar de data naartoe geschreven moet worden. Vervolgens krijgt men de mogelijkheid om een inputveld aan een veld in de DataContainer te koppelen.

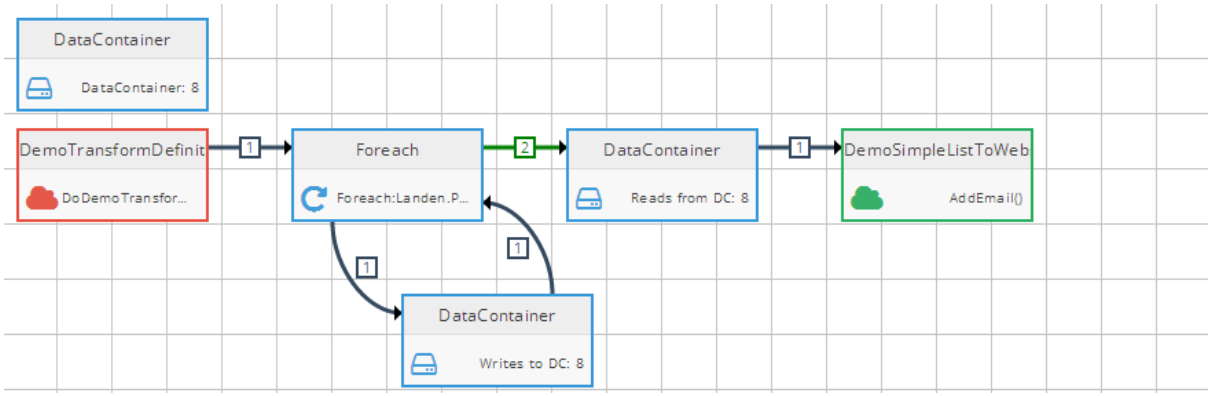


Figuur 39 : Configuratie schrijf-DataContainer-component

De velden waar data naartoe geschreven kan worden, worden automatisch met het geselecteerde DataContainer-component gesynchroniseerd. Indien de inhoud van de opslag-DataContainer aangepast wordt, zullen de beschikbare 'Opslagvelden' automatisch bijgewerkt worden. Deze synchronisatie, wat op de front-end gebeurt, wordt mogelijk gemaakt door de JavaScript library 'KnockoutJS'.

DataContainer-component als lees-component
Indien een DataContainer-component gebruikt wordt om een andere DataContainer bij te schrijven, dient de optie 'Data uit DataContainer lezen' geselecteerd te worden. De gebruiker dient vervolgens een DataContainer te selecteren waar de data vandaan gehaald moet worden. Vervolgens zullen de outputvelden automatisch ingevuld worden door deze met de inhoud van opslag-DataContainer te synchroniseren.

Een eenvoudig geconfigureerde service, waarbij er van een DataContainer gebruik gemaakt wordt, zou er als volgt uit kunnen zien:

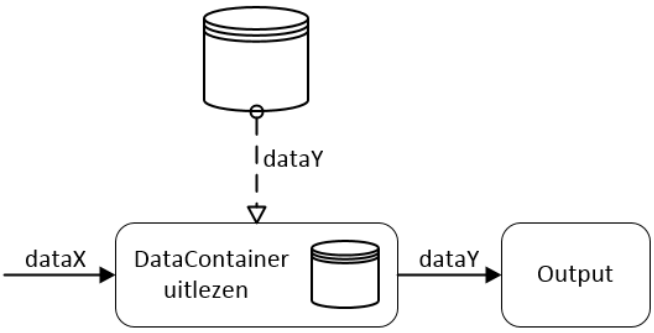


Figuur 40 : Service met DataContainer

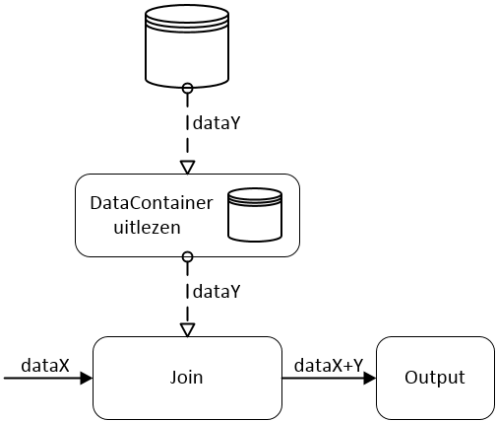
7.8.2. Join-component

Door de introductie van het DataContainer-component was het noodzakelijk om nog een extra component te realiseren. Doordat een DataContainer-component, die data ophaalt, de inputdata vervangt, gaat de input-data verloren. Door een component te maken dat data van meerdere componenten bij elkaar kan voegen, blijft alle data bestaan en gaat er niets verloren. Dit component is tevens te gebruiken voor een functionaliteit die nog gerealiseerd moet worden, namelijk de functionaliteit 'Data verwerken van Input Channels'. Bij deze functionaliteit wordt er tijdens de service een ander inputkanaal aangeroepen, bijvoorbeeld een Webservice of Database, om deze data vervolgens samen te voegen met data uit het oorspronkelijke inputkanaal.

Om het verschil tussen een service met en zonder Join-component weer te geven, zijn onderstaande schema's gemaakt.



Figuur 41 : Service zonder Join-component



Figuur 42 : Service met Join-component

Er is voor de naam 'Join-component' gekozen, omdat het begrip 'Join' in de ICT vaker wordt gebruikt, waarbij er over het algemeen twee of meerdere dingen samengevoegd worden.

8. Overzicht gerealiseerde functionaliteiten

De doelstelling van het project was om de eerste twee epics, de repeating items en de synchrone verwerking, zo ver mogelijk te realiseren. Tijdens het realiseren van de repeating items bleek dat deze functionaliteit groter en complexer was dan vooraf begroot. Omdat het systeem zo generiek mogelijk moet zijn, moeten er enorm veel scenario's ondersteund worden. Het realiseren, testen en uitzoeken van al deze scenario's kostte meer tijd dan verwacht, waardoor er door tijdgebrek helaas geen andere functionaliteiten gerealiseerd konden worden.

Om een kort overzicht te geven van de aanpassingen die binnen MOBO verricht zijn, is er een tabel opgesteld waarin de meeste aanpassingen staan opgenomen. Om de aanpassingen visueel te ondersteunen is er in 'Bijlage 6 : Impressie nieuwe situatie MOBO' een overzicht toegevoegd met daarin screenshots van de oude en nieuwe situaties.

Aanpassing	Beschrijving
Veld-configuratie	Er kunnen lijsten in de veld-configuratie ingesteld worden
Webservice lijsten	Er kunnen webservices met lijsten aangemaakt worden (d.m.v. gegenereerde WSDL files)
Lijsten	Lijsten kunnen in de DataContainer verwerkt worden
Lijst-mapping	Lijsten kunnen op de juiste wijze gemapt worden
Foreach-component	Componenten kunnen door middel van dit component per lijst-item uitgevoerd worden
Datacontainer-component	Component voor het (tussentijds) opslaan, bijschrijven en uitlezen van data in een service
Join-component	Component voor het samenvoegen van data uit twee nodes[2].
Output-configuratie	Lijsten kunnen in tekstuele output-configuraties opgenomen worden
Mailing	Tijd van mailen is verkort van +/- 2 seconden naar +/- 0.5 seconde
Lay-out	Op diverse plekken is de visuele weergave verbeterd
Endpoint-volgorde	De volgorde van het uitvoeren van endpoints kan nu bepaald worden
IF-Endpoint	Er kunnen meerdere validaties toegepast worden, in plaats van één validatie
Diverse bug fixes	Er zijn o.a. inlog-errors, validatiefouten, en diverse configuratiefouten opgelost
Unit tests	Er zijn unit tests gemaakt die handmatig gestart kunnen worden om de code te testen
Testproject	Het testproject test automatisch alle mogelijk scenario's om te controleren of MOBO naar behoren werkt.
Lijst-component	Component voor het mappen van lijst-items op een platte structuur
Comment-component	Doordat services complex kunnen worden, kan de eindgebruiker het overzicht kwijtraken. Er is daarom besloten om een Comment-component toe te voegen, waarmee opmerkingen in de service geplaatst kunnen worden

Tabel 2 : Realisaties

[2] Een service is opgebouwd uit nodes. Nodes zijn de bouwstenen van een service. Er zijn drie type nodes: inputkanalen, outputkanalen en componenten.

9. Onderzoek: van ontwikkeling naar livegang

Om MOBO uiteindelijk in productie te kunnen nemen, zal er naast het implementeren van nieuwe functionaliteiten het een en ander gedaan en getest moeten worden om dit zonder problemen te laten verlopen. Zodra MOBO live is, mogen er namelijk geen (kritische) fouten meer voorkomen. Om de fouten zoveel mogelijk te voorkomen, is er onderzocht wat er allemaal gedaan moet worden om hier goede voorzorgsmaatregelen voor te nemen.

9.1. Welke tests moeten er nog worden uitgevoerd voordat MOBO op klanten toegepast kan worden?

MOBO is tot op heden alleen getest door de makers van het product zelf. Hierbij is er vooral getest aan de hand van de verwachtingen van de developer. Deze tests zijn tevens vooral gericht op de gerealiseerde functionaliteit(en). Het kan zo zijn dat de eindgebruikers ergens andere verwachtingen hebben dan dat de developers hadden. Om hier achter te komen zijn gebruikerstesten (of usabilitytesting en crowdtesting) een geschikte oplossing. (Kerpel, 2012) Bij het ontwerpen van MOBO is er ook geen ondersteuning van designers geweest. Het is dus aan te raden om er, voor livegang, nog een user experience designer naar te laten kijken.

Naast de gebruiksvriendelijkheid is de applicatie tot op heden nog bijna niet op prestaties getest. Tijdens de productie van de eerste versie van MOBO is er eenmalig gecontroleerd of het systeem een grote hoeveelheid dataverkeer aankon. Volgens de makers verwerkte het systeem destijds 10.000 aanvragen zonder problemen (Sluijter & De Kam, 2013), maar aangezien het systeem ondertussen veranderd is, kunnen de prestaties ondertussen veranderd zijn. Omdat dit als een andere onderzoeksvraag is opgenomen, wordt dit in ‘Hoofdstuk 9.2. : Hoe kan men, na implementatie, zo zeker mogelijk van goede prestaties zijn?’ verder besproken.

9.2. Hoe kan men, na implementatie, zo zeker mogelijk van goede prestaties zijn?

Zodra het systeem live gaat en de eerste klanten er gebruik van gaan maken, moet het systeem de best mogelijke prestaties blijven leveren. Om deze prestaties voor de livegang te controleren, zal hiervoor getest moeten worden. Microsoft heeft speciaal voor Web Applications een boek geschreven (Performance Testing Guidance for Web Applications) waarin alle tips en tricks voor het testen van de prestaties van een webapplicatie worden uitgelegd. Het testen van de prestatie van een systeem wordt over het algemeen performance testen genoemd. Performance testen wordt ook wel als volgt beschreven: “Feitelijk is performance testen een verzamelnaam van verschillende testtypen die als doel hebben het verzekeren van het verkrijgen van een functioneel kloppende response van een softwaresysteem binnen een acceptabele tijd.” (DeTestmanagerBV, 2014)

In het boek dat Microsoft geschreven heeft, wordt het performance testen opgedeeld in vier verschillende soorten tests: (Microsoft, et al., 2007)

Testsoort	Doel
Performance test	Het bepalen of valideren van de snelheid, schaalbaarheid en/of stabiliteit.
Load test	Het gedrag van de applicatie controleren onder normale en hoge belasting.
Stress test	Het bepalen of valideren van het gedrag van de applicatie wanneer het aan piekbelastingen wordt blootgesteld.
Capacity test	Het bepalen van de hoeveelheid gebruikers en/of transacties dat het systeem zal ondersteunen terwijl het aan de prestatiedoeleinden blijft voldoen.

Tabel 3 : Testsoorten

Om de tests uit te kunnen voeren, zal MOBO eerst op de omgeving geplaatst moeten worden waar MOBO uiteindelijk live op gaat draaien. Aangezien de performance van een applicatie ook van het systeem waarop het draait afhankelijk is, heeft een performancetest alleen nut indien het op de juiste apparatuur wordt getest. (Microsoft, et al., 2007)

Om uiteindelijk te kunnen bepalen of MOBO de juiste prestaties levert, zal er onderzocht moeten worden hoeveel data het systeem (ongeveer) te verwerken zal krijgen. Indien dit geheel onbekend is, is het lastig om te concluderen of MOBO aan de juiste prestatie-eisen voldoet.

Om te controleren of MOBO op dit moment een grote hoeveelheid aantal aanvragen tegelijkertijd aankan, is er getest of MOBO 10.000 aanvragen op één moment kan verwerken. Hieruit blijkt dat MOBO deze hoeveelheid aanvragen zonder problemen kan ontvangen. MOBO blijkt echter wel problemen te ondervinden met het verwerken van een grote hoeveelheid aanvragen op één moment. Indien er een paar duizend aanvragen worden verstuurd,

naar een service die complex geconfigureerd is, kan het soms een aantal seconden duren (+/- 3 seconden) voordat één aanvraag verwerkt is. Aangezien MOBO de aanvragen op dit moment synchroon verwerkt, worden de aanvragen één voor één opgepakt en wordt de volgende aanvraag pas uitgevoerd indien de vorige voltooid is. Hierdoor kan er een wachtrij ontstaan, waardoor bepaalde aanvragen pas na enige tijd uitgevoerd kunnen worden. 10.000 aanvragen kunnen in totaal dus 30.000 seconden (≈ 8.3 uur) duren.

In de huidige situatie lijkt dit niet per direct een probleem. Degene die een aanvraag naar MOBO gestuurd heeft, krijgt geen response wanneer MOBO de aanvraag afgehandeld heeft. De persoon hoeft dus nooit op een bevestiging te wachten. Desondanks mag het niet zo zijn dat bepaalde data op een piekmoment pas na een aantal uur verwerkt wordt. Ook kan de geschetste situatie een ander probleem veroorzaken; indien een gebruiker binnen MOBO een service op een incorrecte wijze geconfigureerd heeft, waardoor er een loop ontstaan is, zal de service nooit eindigen. In dit geval zal de wachtrij alleen maar toenemen en zullen de nieuwe aanvragen nooit afgehandeld worden.

Ook zal er in de toekomst waarschijnlijk een synchrone verwerking in het systeem ingebouwd worden, waardoor de gebruiker een reactie ontvangt zodra MOBO de aanvraag afgehandeld heeft, of om data van een ander kanaal op te vragen. In dit geval zal de gebruiker soms (oneindig) lang op een antwoord moeten wachten omdat de wachtrij gevuld is.

Om dit probleem op te lossen, zal MOBO de aanvragen asynchroon moeten gaan verwerken. Hierdoor hoeven de aanvragen niet meer één voor één verwerkt te worden, maar kunnen ze naast elkaar worden verwerkt. Er is daarom onderzocht of het mogelijk is om de verwerking van de aanvragen asynchroon te laten verlopen.

9.2.1. Kan MOBO de ontvangen aanvragen asynchroon verwerken?

Om te achterhalen of het mogelijk is om de aanvragen asynchroon te laten verwerken, is eerst de huidige situatie van de verwerking onderzocht. Alle data die MOBO ontvangt wordt in een zogenaamde queue geplaatst. Een queue is als het ware een wachtrij waarin alle data geplaatst wordt. De data die als eerst geplaatst is, zal ook als eerst verwerkt worden. (Webopedia, 2014) Er wordt namelijk geen prioriteit meegegeven.

MOBO bevat een QueueListener die constant controleert of er data in de queue aanwezig is. Indien dit het geval is, wordt de eerste data uit de queue gelezen en verwerkt. Zodra de data verwerkt en uit de queue verwijderd is, zal de QueueListener de volgende data uit de queue halen. Dit proces herhaalt zich constant.

De code voor dit proces ziet er als volgt uit:

```
messageQueueTransaction.Begin();
Message queueMessage = queue.Receive(messageQueueTransaction);

ServiceMessage request = (ServiceMessage)queueMessage.Body;
logger.RequestId = request.RequestId;
ServiceQueueMessageHandler serviceQueueMessageServiceHandler = new
  ServiceQueueMessageHandler(request.Service, request);

messageQueueTransaction.Commit();
```

Tussen Begin() en Commit() wordt de data via de service verwerkt.

Om het proces asynchroon te laten verlopen, zou de QueuListener niet moeten wachten totdat de verwerking van bepaalde data voltooid is, maar moet de data gewoon constant uit de Queue gehaald worden. Dit lijkt een simpele oplossing, maar dit blijkt op dit moment echter niet mogelijk.

De data wordt op dit moment bewust één voor één uit de queue gehaald, zodat de data persistent blijft. Indien MOBO (onverwachts) afgesloten wordt, mag de data niet verloren gaan. Indien er besloten wordt om de queue constant te legen, zonder dit via de messageQueueTransaction te doen, zal de data niet meer persistent zijn. (Mackenzie, 2002) Er is door middel van literatuuronderzoek en overleg met medewerkers onderzocht of er in de bestaande oplossing een manier voor multithreading[3] beschikbaar is. Helaas is hier geen bruikbare oplossing voor gevonden.

Het is dus niet mogelijk om de aanvragen in de bestaande situatie asynchroon te laten verwerken. Er is daarom onderzocht welke andere oplossing er is om de verwerking van de aanvragen sneller te laten verlopen.

[3] Bij multithreading worden er meerdere taken (threads) naast elkaar uitgevoerd.

9.2.2. MOBO schalen

Om de snelheid van de te verwerken aanvragen te verhogen, moet er een oplossing gevonden worden waardoor er meerdere aanvragen tegelijkertijd verwerkt kunnen worden. Aangezien het in de huidige situatie niet mogelijk is om dit asynchroon te doen, is er een andere oplossing nodig: MOBO schaalbaar maken. Door een systeem schaalbaar te maken, kan de capaciteit van het systeem (eenvoudig) vergroot worden. Bij schaalbaarheid zijn er twee varianten: scale-up en scale-out. Bij scale-up worden er meestal zwaardere of nieuwere servers ingezet, bij scale-out wordt de verwerkingscapaciteit vergroot door de werklast te verdelen over meerdere systemen. (Leeuwesteijn, 2013)

Aangezien MOBO telkens maar één aanvraag verwerkt, zal de CPU van het systeem niet overbelast raken. Een complexe service blijkt op Intel Core i7-3520M maximaal 2% van de CPU in gebruik te nemen. Er is dus ontzettend veel speling, mocht een service uiteindelijk nog vele malen complexer worden. De scale-up variant is dus niet nodig.

De scale-out variant is wel een serieuze oplossing. Deze variant maakt het mogelijk om meerdere systemen in te zetten, zodat er verschillende MOBO-instanties tegelijkertijd gedraaid kunnen worden. Hiermee kan de duur van de verwerking van de aanvragen eenvoudig verlaagd worden. Om deze oplossing te gebruiken, dient er wel een load balancer[4] ingesteld te worden, waardoor de data over de verschillende MOBO-instanties verdeeld wordt.

Windows Azure biedt een oplossing voor zowel het schaalbaar maken als voor de load balancer.

Schalen en Load balancing in Azure
Binnen Azure zijn er twee mogelijkheden om het schalen van een applicatie te verwerken, namelijk handmatig en automatisch. Indien er voor handmatig gekozen wordt, zal een beheerder constant het verkeer in de gaten moeten houden om te controleren of er een nieuwe virtuele machine (met daarop een instantie van MOBO) ingeschakeld moet worden. Indien er voor automatisch gekozen wordt, zal Azure automatisch een extra virtuele machine inschakelen zodra het CPU gebruik boven een bepaalde grens komt, of zodra het aantal aanvragen dat binnenkomt boven een bepaalde grens komt. (Microsoft, How to Scale an Application, 2012)

Aangezien MOBO niet 24 uur per dag bewaakt moet worden, is de automatische optie een betere oplossing dan de handmatige optie. Daarnaast zal het schalen afhankelijk moeten zijn van de hoeveelheid aanvragen die binnenkomen, aangezien het CPU gebruik geen goede indicator is.

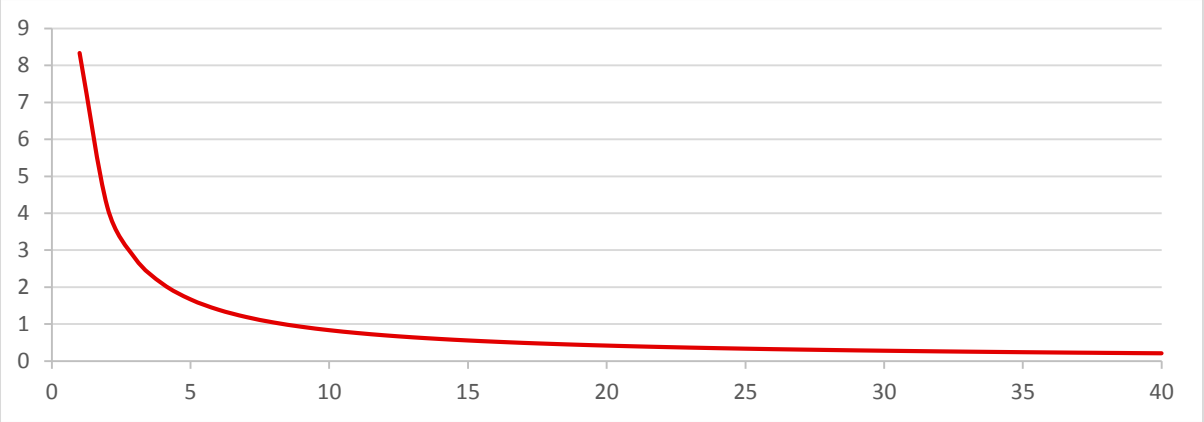
Het is verder zeer eenvoudig om het automatische schalen in Azure in te stellen. Er kan ingesteld worden bij hoeveel aanvragen er een nieuwe virtuele machine ingeschakeld moet worden, hoeveel machines er dan ingeschakeld moeten worden en hoe lang Azure moet wachten met inschakelen op het moment dat het limiet bereikt is. Hetzelfde geldt voor het naar beneden schalen. (Microsoft, How to Scale an Application, 2012)

Door een wachttijd in te stellen, zal Azure niet per direct een nieuwe virtuele machine inschakelen wanneer het limiet maar net overschreven wordt. Dit kan kosten besparen, aangezien het gebruik van virtuele machines geld kost.

Is het schalen voldoende?
Het schalen van MOBO zal de prestaties zeker ten goede komen, aangezien niet alle aanvragen door één systeem verwerkt hoeven te worden. De vraag is echter of het schalen van MOBO uiteindelijk het probleem van de te lange verwerkingstijd op kan lossen.

Om dit in de kaart te brengen, is er een grafiek opgebouwd waarin nog steeds rekening wordt gehouden met een grote hoeveelheid (10.000) aanvragen, die allemaal door een service verwerkt moeten worden die ongeveer drie seconden duurt.

[4] “Loadbalancing betekent heel letterlijk het verdelen van werk over meerdere uitvoerenden.” (Meijden, 2002)



Grafiek 1 : Verwerkingsduur aanvragen per aantal MOBO-instanties

In de grafiek wordt op de x-as de hoeveelheid MOBO-instanties weergegeven. Op de y-as is te zien hoeveel uur het bij de hoeveelheid instanties duurt, voordat er 10.000 aanvragen verwerkt zijn. Hierin is te zien dat de verwerkingstijd exponentieel afneemt, waardoor er steeds meer extra instanties nodig zijn om de verwerkingstijd voldoende te verkleinen. Zo is de verwerkingstijd bij 20 instanties 25 minuten, bij 40 instanties 12,5 minuut en bij 80 instanties nog ruim 6 minuten. Bij deze verhouding zal de verwerkingstijd, indien er 320 instanties gebruikt worden, nog altijd meer dan één minuut bedragen.

Naast de hoge kosten die dit met zich mee zal brengen, is het binnen Azure niet eens mogelijk om meer dan 20 virtuele machines in te laten schakelen. (Microsoft, How to Scale an Application, 2012) Hierdoor zal de verwerkingstijd, bij de geschetste situatie, nooit minder dan 25 minuten zijn.

De kans dat er 10.000 aanvragen op één moment binnenkomen is behoorlijk klein. Er is daarom geïnventariseerd hoeveel aanvragen er bij een bepaalde klant per uur binnen kunnen komen. Hieruit blijkt dat het op piekmomenten voor kan komen dat er zeker 1000, misschien een paar duizend, aanvragen per uur binnen kunnen komen. Indien er 1000 aanvragen over één uur verspreid binnen komen, hoeft dit geen probleem op te leveren. Indien een service drie seconden bezig is met het verwerken van de aanvraag, zullen de aanvragen totaal 3000 seconden duren. Met 3600 beschikbare seconden in een uur is dit mogelijk. In de praktijk zullen de aanvragen echter nooit geheel verspreid over een uur binnenkomen. Indien de aanvragen binnen een half uur binnen komen, zal de laatst verstuurde aanvraag al minstens 20 minuten moeten wachten. Indien de volledige schaling wordt gebruikt, zal de aanvraag nog minstens 1 minuut duren.

Door MOBO alleen te schalen, kan de prestatie van MOBO eigenlijk niet gewaarborgd worden. Indien de aanvragen verspreid binnenkomen, dan zullen de prestaties er niet onder leiden, maar zodra er meerdere aanvragen tegelijkertijd binnenkomen, dan zal de verwerkingstijd alleen maar toenemen. Tevens is vandaag de dag de security van ieder systeem een belangrijk knelpunt. Een onveilig systeem kan de verwerkingstijd van de aanvragen in gevaar brengen. In ‘Hoofdstuk 9.3. : Security’ wordt hier verder op ingegaan.

Om na de implementatie van MOBO zo zeker mogelijk van goede prestaties te zijn, zal er nog het één en ander aangepast moeten worden. Wat er op zijn minst allemaal aangepast moet worden, wordt in ‘Hoofdstuk 9.4. : Wat moet er, naast het toevoegen van de ontbrekende functionaliteiten, aan het systeem aangepast worden voordat het live gezet kan worden?’ verder toegelicht.

9.3. Security

Tegenwoordig worden ontzettend veel systemen door criminelen aangevallen. Op 11 februari 2014 werd bijvoorbeeld nog de grootste DDoS-aanval[5] (Distributed Denial of Service) ooit in Europa uitgevoerd. (Hoek, 2014) Aangezien Redhotminute over grote klanten beschikt, moet MOBO Requests zo goed mogelijk tegen dergelijke aanvallen bestand zijn.

Om data naar MOBO te sturen, dient de verzender een MOBO Service Key mee te sturen. Deze key is gekoppeld aan een service, waardoor MOBO weet naar welke service de data gestuurd moet worden. De key wordt gegenereerd en is uniek. Indien iemand zonder toestemming over een key beschikt, dan is het mogelijk om een nieuwe key te laten genereren.

[5] Bij een DDoS-aanval wordt een systeem beladen met een enorme hoeveelheid data, waardoor het systeem crasht.

Men moet dus zorgen dat de key uit de handen van kwaadwillenden blijft. Zonder geldige key zal MOBO de data namelijk niet verwerken, waardoor het niet mogelijk is om zonder key data naar MOBO te sturen. De MOBO Service Key mag dus nooit op de front-end van een website geplaatst worden, maar zal altijd in de back-end opgenomen moeten worden.

Mocht een key toch in de handen van een onbevoegde komen, omdat een klant de key bijvoorbeeld verkeerd geplaatst heeft of omdat iemand zijn of haar computer niet goed beveiligd heeft waardoor een kwaadwillende op afstand mee kan kijken, dan kan MOBO per direct platgelegd worden. Er kan op dat moment namelijk direct een enorme hoeveelheid data naar MOBO gestuurd worden, waardoor het systeem overbelast raakt en zal crashen. Het is dus zaak dat er zeer zorgvuldig met iedere MOBO Service Key omgegaan wordt.

Er zijn diverse mogelijkheden om de veiligheid van de MOBO Service Key te verbeteren.

- Beperkte data per gebruiker: door de herkomst van de data te achterhalen, kan er bepaald worden dat een gebruiker maar een bepaalde hoeveelheid data per uur mag sturen. Dit wordt bijvoorbeeld ook bij Twitter toegepast om spam te voorkomen. (Twitter, 2014)
- Data per key beperken: door een maximaal aantal aanvragen (per uur) op een MOBO Service Key in te stellen, kan een enorme hoeveelheid aanvragen beperkt worden. Het nadeel is dat er dan een limiet ingesteld moet worden die vaak lastig te bepalen is. Dit zal dan in overleg met de klant moeten gebeuren.

Er zullen ongetwijfeld nog andere mogelijkheden zijn om de beveiliging te verbeteren. Aangezien verder onderzoek naar deze mogelijkheden buiten de scope van dit project valt, zal hier niet verder op worden ingegaan. Desondanks kan er wel geconcludeerd worden dat de beveiliging van MOBO te verbeteren valt.

9.4. Wat moet er, naast het toevoegen van de ontbrekende functionaliteiten, aan het systeem aangepast worden voordat het live gezet kan worden?

Er is aangetoond dat de huidige staat van MOBO niet geschikt is om grote hoeveelheden aanvragen te verwerken. Om meer aanvragen aan te kunnen, zal de verwerking van de aanvragen asynchroon moeten gaan verlopen. Hierdoor kan één MOBO-instantie meerdere aanvragen tegelijkertijd afhandelen en hoeft er niet per direct geschaald te worden. Natuurlijk blijft het schalen een goede oplossing om de prestaties van MOBO goed te houden, zodat niet al het verkeer over één MOBO-instantie hoeft te gaan.

Aangezien het geen oplossing is om met queues te blijven werken, omdat de persistentie anders verloren gaat, zal deze manier van verwerken vervangen moeten worden door een andere oplossing. Literatuuronderzoek levert een mogelijke oplossing, namelijk het gebruik maken van een (SQL) database in combinatie met ThreadPool. (Faktorovich, 2009)

De ThreadPool maakt het mogelijk om meerdere taken naast elkaar uit te voeren (asynchroon), bijvoorbeeld nadat er nieuwe data aan de database is toegevoegd. (Saccone, 2011) Door de data in de database op te slaan en pas te verwijderen na het uitvoeren van een service, zal de data persistent blijven. Standaard kan een ThreadPool maximaal 500 threads tegelijkertijd aan. (Windows, 2011) In de praktijk zou er getest moeten worden of dat dit haalbaar is. Indien dit mogelijk is, dan zullen er 500 keer zoveel aanvragen per minuut verwerkt kunnen worden.

In het onderstaande schema is duidelijk het verschil te zien tussen een synchrone en asynchrone verwerking, waarbij er van een service wordt uitgegaan die ongeveer drie seconden duurt:

Aantal MOBO-instanties	Synchroon (verwerkingen p/min.)	Asynchroon (verwerkingen p/min.)
1	20	10000
5	100	50000
10	200	100000
15	300	150000
20	400	200000

Tabel 4 : Overzicht verwerkingen per minuut

Om de verwerking asynchroon te laten verlopen, dient de hele code van MOBO gerefactord te worden. De huidige opbouw is namelijk niet gebaseerd op een asynchrone verwerking. Hierdoor kunnen er conflicten ontstaan indien een service twee of meerdere keren tegelijkertijd uitgevoerd wordt.

Om MOBO uiteindelijk op klanten toe te kunnen passen, dient er nog best wat aangepast te worden. Naast deze grote aanpassingen hoeft er verder weinig aan de code veranderd te worden. Het enige wat opnieuw ingesteld moet worden zijn de database instellingen van de database van het systeem waarop MOBO gaat draaien. Tevens dienen de Azure instellingen aangepast te worden, aangezien de instellingen nu op een proef-account afgesteld staan. Welke acties verder ondernomen dienen te worden om MOBO op de productieomgeving in te stellen, staan opgesteld in een release-document. Dit document is al eerder door de makers van MOBO opgesteld.

9.5. Hoe moet er gecontroleerd worden of MOBO alle data op het toegepaste systeem correct verwerkt, zonder dat er data verloren gaat of onbeheerst dubbel opgeslagen wordt?

Nadat MOBO geheel getest is en klaar is voor livegang, zal het testen nog niet helemaal klaar zijn. Het systeem is dan wel compleet getest, toch kan het zijn dat men er ook zeker van wil zijn dat het systeem ook op een specifieke situatie werkt. Denk hierbij aan de volgende situatie:

Een klant heeft een webformulier op haar website staan. Na het versturen van het webformulier wordt de data gemaild en in een CRM plaatst. Helaas gaat bepaalde data soms verloren omdat het CRM systeem wel eens onbereikbaar is. De klant zou dit proces graag via MOBO willen laten lopen, zodat MOBO het onbereikbare CRM systeem af kan vangen en zodat een secretaresse eenvoudig aanpassingen in het proces aan kan brengen. De klant wil er echter zeker van zijn dat MOBO de data daadwerkelijk op de juiste wijze verstuurt. Om de klant hiervan te kunnen overtuigen, moet het mogelijk zijn om hier bewijs voor aan te leveren. Tevens willen de medewerkers van de klant misschien eerst veilig met MOBO kunnen oefenen, zonder dat hierdoor data verloren gaat.

Om een nieuw systeem in gebruik te nemen, zijn er twee uiterst verschillende manieren:

- Big Bang: hierbij wordt het oude systeem in zeer korte tijd vervangen door het nieuwe systeem. Denk hierbij aan een dag of een week.
- Schaduwdraaien: hierbij draaien het oude en het nieuwe systeem enige tijd naast elkaar. Men laat deze test doorgaans niet langer dan één of twee weken duren. (Hoogenraad, 2011)

De voor- en nadelen van beide methodes staan hieronder schematisch weergegeven (Wikibooks, 2013):

	Voordelen	Nadelen
Big Bang	Goedkoper	Gevolgen van verstoringen door kinderziekten ernstig
	Eén manier van werken	
Schaduwdraaien	Gevolgen van verstoringen door kinderziekten beperkt	Hogere kosten door dubbel werk
		Risico van niet-synchroon lopen van oude en nieuwe systeem

Tabel 5 : Voor- en nadelen Big Bang & Schaduwdraaien

Om MOBO volgens de geschetste situatie te kunnen testen, zal er van de methode 'schaduwdraaien' gebruik gemaakt moeten worden. Deze testperiode wordt ook wel Pilot, Proeftuin of Pre-productie run genoemd. De doelstellingen van deze periode worden als volgt beschreven:

- Controleren of het systeem past in de bestaande en ontworpen procedures.
- Bepalen of de opgeleverde functionaliteit de werkzaamheden voldoende ondersteunt.
- Het laten wennen van de gebruikers aan het systeem.
- Controleren of de output van het systeem aan de verwachtingen voldoet.
- Vaststellen of aan de tijdens de informatie analyse gestelde eisen (performance) is voldaan.

(Hoogenraad, 2011)

Als de beschreven situatie met de doelstellingen vergeleken wordt, kan er geconcludeerd worden dat het schaduwdraaien de juiste methode is. Men kan echter niet zomaar beslissen om dat te gaan doen. Schaduwdraaien levert namelijk dubbele data op; van het oude en van het nieuwe systeem. Indien dit proces niet bewaakt wordt, kan de hoeveelheid data onbeheersbaar worden, waardoor de data bijna alsnog als verloren beschouwd kan worden. Tevens is er capaciteit (personen en budget) nodig om het schaduwdraaien te kunnen monitoren. De data moet namelijk vergeleken worden om te controleren of het systeem aan de verwachtingen voldoet.

Het vergelijken van de oude en nieuwe data kan in sommige gevallen automatisch gedaan worden en zal in sommige gevallen handmatig gedaan moeten worden. Indien er een hulpprogramma voor beschikbaar is, dan is het automatisch controleren van de data in sommige gevallen mogelijk. De handmatige controle zal in alle gevallen mogelijk zijn, aangezien men weet hoe de oude data eruit ziet en hoe de nieuwe data er uit zou moeten zien. Door deze data naast elkaar te leggen, kan er een conclusie getrokken worden. Het verklaren van verschillen tussen de data kan echter veel werk en tijd kosten, ook als deze niet het gevolg van fouten in het nieuwgebouwde systeem zijn. (Buwalda, 1997)

Door te schaduwdraaien zal er geen data verloren gaan, aangezien de oude situatie pas opgeheven wordt als de nieuwe daadwerkelijk blijkt te werken. Het nadeel van het schaduwdraaien is dat er (tijdelijk) dubbele data aanwezig is. In sommige gevallen hoeft dit geen probleem te zijn, bijvoorbeeld als de data van een webformulier gemaild wordt. Het nieuwe systeem kan de data bijvoorbeeld naar een ander mailadres sturen, of kan de mails van extra informatie voorzien, waardoor het eenvoudig is om de nieuwe (of oude) data te verwijderen zodat er geen duplicaten meer zijn. In andere gevallen zal dit niet mogelijk zijn, waardoor de data handmatig verwerkt moet worden. Afhankelijk van het systeem waar de data uiteindelijk in terecht komt, is het wel of niet mogelijk om de oude van de nieuwe data te onderscheiden.

Het is dus niet mogelijk om direct te bepalen of het schaduwdraaien voor een specifieke klant mogelijk is. Dit is namelijk van een aantal factoren afhankelijk, zoals beschikbaarheid, budget en de gebruikte kanalen. Indien het wel mogelijk is, dan kan de hoeveelheid data beheerst worden door de data door een aantal mensen (afhankelijk van de hoeveelheid data) te laten monitoren, door de data naar een (tijdelijk) ander, maar vergelijkbaar, kanaal te sturen of door bepaalde informatie aan het kanaal toe te voegen waardoor de nieuwe data van de oude data te onderscheiden is om de dubbele informatie uiteindelijk eenvoudig uit het systeem te kunnen verwijderen. Deze informatie kan zowel aan de data zelf, als aan metadata[6] toegevoegd worden.

[6] "Metadata zijn gegevens die de karakteristieken van bepaalde gegevens beschrijven. Het zijn dus eigenlijk data over data. Door het toevoegen van metadata snappen databanken en applicaties wat de functie is van bepaalde data. Deze toegevoegde informatie, of metadata zorgt ervoor dat de basisinformatie te vinden, te controleren en te organiseren is." (Meta4books, 2013)

10. Conclusies

Het hoofddoel van het project was het uitbreiden van MOBO Requests zodat het systeem volwassener wordt om het vervolgens op klanten toe te kunnen passen. Om het systeem uiteindelijk op klanten toe te passen, moet er naast het uitbreiden van het systeem nog meer gebeuren. Om te achterhalen wat er allemaal nog gedaan en gemaakt moeten worden en hoe dat dit gedaan moet worden, is de volgende vraag gesteld: Hoe kan MOBO, zodra het volwassen genoeg is, foutloos op een bestaand systeem (van een klant) ingezet worden, zonder dat er gegevens verloren gaan of juist dubbel opgeslagen worden en wat dient er vóór de implementatie nog te gebeuren?

Om deze vraag te kunnen beantwoorden en om MOBO uit te kunnen breiden, zijn er deelvragen opgesteld. De eerste vraag die hierbij gesteld is; Wat is CoffeeScript en zijn hier alternatieven voor? Dit onderzoek is gedaan omdat de kennis hiervan nodig bleek te zijn om de nieuwe functionaliteiten te kunnen realiseren. CoffeeScript blijkt een alternatief voor JavaScript, dat voornamelijk fouten en onnodige regels corrigeert. Er zijn alternatieven voor, maar die zijn voor dit project niet van belang.

Om de nieuwe functionaliteiten te kunnen realiseren, is vervolgens de volgende vraag gesteld: hoe ziet elke toe te voegen functionaliteit er geheel uit, wat doet het, wat zijn de eisen aan de vormgeving, wat zijn de specifieke kwaliteitseisen, etc.? Hierbij is er onderzocht hoe de toe te voegen functionaliteit er in het geheel uit zou moeten komen te zien. Aangezien niet alle ontbrekende functionaliteiten geïmplementeerd konden worden – dit in verband met tijdgebrek – zijn alleen de functionaliteiten ontworpen die ook daadwerkelijk gerealiseerd zijn.

In dit project is er helaas maar één functionaliteit gerealiseerd omdat deze functionaliteit behoorlijk complex bleek te zijn. Er kan achteraf niet geconcludeerd worden hoe de functionaliteit er precies uit is komen te zien. Er kan wel geconcludeerd worden dat de functionaliteit aan de eisen voldoet die gesteld zijn. Tijdens het realiseren zijn er twee duidelijke eisen naar voren gekomen: de nieuwe functionaliteit ‘repeating items’ moet alle mogelijke lijst-scenario’s gaan ondersteunen en het systeem moet voor een persoon zonder programmeerkennis te bedienen zijn. De benodigde scenario’s zijn allemaal gerealiseerd en de visuele weergave is niet veranderd ten opzichte van de oude versie van MOBO, waardoor het systeem voor de doelgroep bedienbaar blijft.

Naast het realiseren van functionaliteiten is onderzoek verricht naar eventuele aanpassingen die gedaan moeten worden. Omdat er twijfels waren bij de huidige authenticatie, Microsoft Azure, is er onderzocht of hier iets aan gedaan moet worden. Door te vragen of Microsoft Azure als authenticatie gebruikt moet blijven worden, of dat hier betere/goedkopere alternatieven voor zijn en welke alternatieven er dan zijn, is er bepaald of Microsoft Azure vervangen moet worden. Het blijkt dat Windows Azure Active Directory, het onderdeel dat uit Microsoft Azure gebruikt wordt, als authenticatie gebruikt kan blijven worden. De beperkte voordelen van het alternatief wegen namelijk niet op tegen de tijd dat het kost om het alternatief te implementeren.

Vervolgens is er onderzocht welke tests er nog uitgevoerd moeten worden voordat MOBO op klanten toegepast kan gaan worden. Uit dit onderzoek is gebleken dat er nog diverse tests uitgevoerd moeten worden om het systeem aan klanten over te kunnen dragen. Deze tests zijn zowel op functioneel gebied gebaseerd als op het gebied van prestaties. Om de prestaties te kunnen waarborgen blijkt dat MOBO Requests nog zeker niet op klanten toegepast kan worden. Door de huidige architectuur kunnen grote hoeveelheden data niet snel genoeg verwerkt worden, waardoor de klanten geen goede prestaties kunnen verwachten. Dit probleem heeft direct een relatie met de vraag: hoe kunnen we er zo zeker mogelijk van zijn dat de prestaties (snelheid e.d.) van het systeem goed zijn/blijven na de implementatie van MOBO? Hieruit is gebleken dat de architectuur van MOBO aangepast moet worden om de prestaties te verbeteren. In de huidige situatie worden de aanvragen namelijk synchroon verwerkt, wat uiteindelijk asynchroon dient te gebeuren. Het is ook geen overbodige luxe om het systeem uiteindelijk te schalen. Dit geeft het systeem nóg betere prestaties.

Deze aanpassingen geven ook direct een gedeelte van het antwoord op de vraag wat er – naast de ontbrekende functionaliteiten – aan het huidige systeem aangepast moet worden om het op klanten te kunnen toepassen. Het verbeteren van de prestaties is namelijk een vereiste en zal dus zeker moeten gebeuren. Daarnaast zullen er een aantal database- en authenticatie-instellingen aangepast moeten worden. Verder is MOBO, op de ontbrekende functionaliteiten na, klaar voor gebruik.

Indien MOBO aan klanten overgedragen wordt, rijst de volgende vraag: hoe moet er gecontroleerd gaan worden of MOBO alle data op het toegepaste systeem correct verwerkt, zonder dat er data verloren gaat of onbeheerst dubbel opgeslagen wordt? Hieruit blijkt dat men het systeem in eerste instantie zal moeten gaan schaduwdraaien, door eventueel metadata aan de data toe te voegen. Het is aan de klant hoeveel tijd en geld men hieraan wilt besteden.

Al met al kan worden geconcludeerd dat het systeem nog uitgebreid getest, aangepast en verbeterd moet worden, voordat MOBO daadwerkelijk operationeel wordt.

11. Aanbevelingen

MOBO Requests is door de realisatie van de functionaliteit ‘Repeating items’ een stuk uitgebreider geworden. Desondanks kunnen er nog diverse andere zaken aangepast worden om het systeem verder te optimaliseren.

Functioneel:
Omdat er tijdens het project maar één nieuwe functionaliteit gerealiseerd is, staan er nog diverse ongerealiseerde functionaliteiten open. Om de bruikbaarheid en volledigheid van MOBO te vergroten is het aan te raden om de ontbrekende functionaliteiten aan het systeem toe te voegen. Het gaat hierbij om de volgende functionaliteiten:

- Synchrone verwerking: Een synchrone verwerking realiseren tussen het in- en & outputkanaal zodat de response van het outputkanaal bij het inputkanaal afgeleverd kan worden.
- Business Activity Monitoring: Hiermee kan men een betere indruk over de prestaties van het systeem krijgen, zoals slagingspercentage, aanvraagtijden, verwerkingstijden, etc.
- Data verwerken van Input Channels: Deze functionaliteit maakt het mogelijk om data, gedurende het uitvoeren van een service, van externe webserivce op te vragen om hierin vervolgens te verwerken. Het is aan te raden om hiervoor het reeds gerealiseerde Join-component te gebruiken.
- Nieuwe Input Channels: Er kunnen meerdere nieuwe Input Channels toegevoegd worden om de bruikbaarheid van het systeem te vergroten. Denk hierbij aan Facebook, Twitter, Copernica en JIRA.

Naast deze functionaliteiten is het aan te bevelen om alle validaties binnen MOBO na te lopen. Sommige validaties zijn incompleet en valideren dus niet helemaal zoals het zou moeten, terwijl op andere locaties de validatie soms volledig ontbreekt. Door de validaties te optimaliseren, zullen er minder fouten in het systeem plaatsvinden.

Visueel:
Het huidige ontwerp van het controlpanel van MOBO is voornamelijk gebaseerd op een 24 inch monitor, met een resolutie van 1920 x 1200. Indien klanten het systeem gaan gebruiken, kan het voorkomen dat er kleinere schermen en/of lagere resoluties gebruikt worden. Op dit moment is het controlpanel hier niet op gebouwd. Hierdoor moeten gebruikers veel scrollen en is het lastig om een complexe service in één oogopslag te overzien.

Het is daarom aan te raden om het controlpanel visueel onder de loep te nemen en vervolgens aan te passen, zodat het ook op kleinere monitoren eenvoudig te gebruiken is.

Daarnaast heeft er nog nooit een user experience designer naar het ontwerp gekeken. Door dat wel te laten doen, zullen er andere verbeteringen aan het licht komen waardoor het systeem beter op klanten toegepast kan worden.

Prestaties:
Doordat de huidige architectuur van MOBO het niet toelaat om meerdere aanvragen op één moment te verwerken, zal de prestatie van MOBO bij groot gebruik afnemen. Om de prestaties te optimaliseren is het dus aan te raden om de architectuur aan te passen zodat de aanvragen asynchroon verwerkt kunnen gaan worden. Om de maximale belasting extra te vergroten, kan MOBO schaalbaar gemaakt worden om op deze manier de werkdruk over meerdere MOBO-instanties te verdelen.

Tevens kunnen gebruikers op dit moment een loop in de configuratie veroorzaken die nooit zal eindigen. Hierdoor kunnen andere aanvragen niet verwerkt worden. Om de prestaties in orde te houden, kan er een controle ingebouwd worden die ervoor zorgt dat een service niet langer dan een bepaalde tijd bezig mag zijn.

Daarnaast zijn de prestaties van het controlpanel achteruitgegaan. Doordat er meerdere componenten beschikbaar zijn, grotere services gemaakt kunnen worden en er grote logs gegeneerd worden, duurt het laden van de pagina’s langer. Het is daarom aan te raden om de services, logs en channels te cachen, waardoor de pagina’s sneller geladen kunnen worden.

Veiligheid:
Er is gebleken dat de beveiliging door middel van de MOBO Service Key beperkingen met zich meebrengt. Daarom wordt er sterk aangeraden om deze beveiliging opnieuw te bekijken om de veiligheid ervan aan te scherpen. Indien één enkele Service Key op straat komt te liggen, kan het hele systeem platgelegd worden. Het is een te groot risico om dit probleem te bagatelliseren.

12. Evaluatie procesgang

Na ongeveer vier maanden kan er tevreden teruggekeken worden op een periode waarin veel complexe situaties uitgewerkt zijn en waarin onderzoek is gedaan om MOBO naar een nog hoger niveau te brengen.

Van begin af aan is er altijd een goede samenwerking geweest tussen de student, begeleiders en product owner, waardoor er altijd doelgericht gewerkt is. Vanaf het opstellen van het plan van aanpak tot aan het schrijven van de scriptie is er altijd input van de benodigde personen gekomen. Dankzij het tweewekelijkse overleg zijn alle betrokkenen altijd op de hoogte gebleven van de vorderingen in het project. Hierbij zijn altijd de resultaten van de sprint besproken om vervolgens de komende sprint in te delen. Deze manier van werken is het resultaat zeker ten goede gekomen. Door een drukke periode bij de product owner is een enkel overleg verplaatst of geannuleerd, maar door goed overleg met de bedrijfsbegeleider heeft het project hier niet onder geleden.

Om de af te ronden stories en taken te bewaken, is er het gehele project van JIRA gebruik gemaakt. Hierdoor konden zowel de student als de begeleiders op de hoogte blijven van de vorderingen in het project, waardoor er per sprint doelgericht gewerkt kon worden. Zonder deze duidelijke planning hadden bepaalde zaken mogelijk langer geduurd, aangezien er nu strakke en duidelijke deadlines waren gesteld.

Vooraf is afgesproken om volgens de genoemde wijze te werken. Achteraf kan gezegd worden deze werkwijze en de procesgang goed zijn verlopen.

13. Bronnenlijst

Bek, R. (2014, Januari 22). Werkwijze Redhotminute. (F. Gruiters, Interviewer)

Brockallen. (2014, April 11). *BrockAllen.MembershipReboot*. Opgehaald van GitHub: https://github.com/brockallen/BrockAllen.MembershipReboot

Buwalda, H. (1997). *Efficiënt en flexibel testen*. Opgehaald van Computable: http://www.computable.nl/artikel/achtergrond/bankverzekeringswezen/1368602/1277528/efficieumlnt-en-flexibel-testen.html

CodeforHire. (2013, Juni 18). *CoffeeScript vs. TypeScript vs. Dart*. Opgehaald van Code for Hire: http://codeforhire.com/2013/06/18/coffeescript-vs-typescript-vs-dart/

Condor. (2014). *Enterprise Service Bus*. Opgehaald van Condor: http://www.condorsolutions.nl/technologie/enterprise-service-bus/

DeTestmanagerBV. (2014). *Wat is performance testen eigenlijk precies?* Opgehaald van DeTestmanagerBV: http://www.testmanager.nl/performance-testen/

Deuby, S. (2014, Maart 28). *An Overview of Microsoft Azure Active Directory Premium*. Opgehaald van WindowsITPro: http://windowsitpro.com/identity-management/overview-microsoft-azure-active-directory-premium

Embregts, H. (2012, Maart 9). *CoffeeScript: JavaScript done right*. Opgehaald van Software Innovators: http://www.software-innovators.nl/2012/03/09/coffeescript-javascript-done-right/

Faktorovich, Y. (2009, November 30). *C# - Queue and multithreading*. Opgehaald van Stack Overflow: http://stackoverflow.com/questions/1817799/c-sharp-queue-and-multithreading?answertab=active#tab-top

Finley, K. (2011, Januari 7). *Interview: Jeremy Ashkenas Talks About CoffeeScript*. Opgehaald van readwrite: http://readwrite.com/2011/01/07/interview-coffeescript-jeremy-ashkenas

Finley, K. (2012, September 12). *JavaScript Tops Latest Programming Language Popularity Ranking From RedMonk*. Opgehaald van Techcrunch: http://techcrunch.com/2012/09/12/javascript-tops-latest-programming-language-popularity-ranking-from-redmonk/

Galloway, J. (2012, Augustus 29). *SimpleMembership, Membership Providers, Universal Providers and the new ASP.NET 4.5 Web Forms and ASP.NET MVC 4 templates*. Opgehaald van Jon Galloway: http://weblogs.asp.net/jgalloway/archive/2012/08/29/simplemembership-membership-providers-universal-providers-and-the-new-asp-net-4-5-web-forms-and-asp-net-mvc-4-templates.aspx

Hoek, C. v. (2014, Februai 12). *Grootste DDoS-aanval ooit uitgevoerd in Europa*. Opgehaald van NU.nl: http://www.nu.nl/tech/3699853/grootste-ddos-aanval-ooit-uitgevoerd-in-europa.html

Hoogenraad, W. (2011, Januari 28). *Invoeren van een nieuwe applicatie, haken en ogen van schaduwdraaien.* Opgehaald van ITpedia: http://www.itpedia.nl/tag/schaduwdraaien/

Kerpel, H. (2012, Juli 20). *Gebruikerstests - Hoe en waarom*. Opgehaald van Netvlies: http://www.netvlies.nl/blog/design-interactie/gebruikerstests-hoe-waarom

Kieft, T. (2013, Maart 26). *Windows Azure voor MKB een serieuze optie!* Opgehaald van Mygrade: http://www.mygrade.nl/windows-azure-voor-mkb/

Leastprivilege. (2014, Februari 27). *OpenID Connect and the IdentityServer Roadmap*. Opgehaald van Leastprivilege: http://leastprivilege.com/2014/02/27/openid-connect-and-the-identityserver-roadmap/

Leeuwesteijn, A. (2013, 27 Mei). *Bouwen op de groei – schaalbaarheid in het ontwerp van websites en cloud-applicaties*. Opgehaald van Macaw: http://www.macaw.nl/weblog/2013/5/bouwen-op-de-groei-schaalbaarheid-in-het-ontwerp-van-websites-en-cloud-applicaties

Mackenzie, D. (2002, Februari). *Reliable Messaging with MSMQ and .NET*. Opgehaald van Developer Network: http://msdn.microsoft.com/en-us/library/ms978430.aspx

Meijden, A. v. (2002, Mei 22). *Loadbalancing bij Tweakers.net*. Opgehaald van Tweakers: http://tweakers.net/reviews/301/loadbalancing-bij-tweakers-punt-net.html

Meta4books. (2013). *Wat is metadata?* Opgehaald van Meta4books: http://www.meta4books.be/nieuws/wat-metadata

Microsoft. (2012). *How to Scale an Application*. Opgehaald van Microsoft Azure: http://azure.microsoft.com/en-us/documentation/articles/cloud-services-how-to-scale/

Microsoft. (2013, November 21). *Azure Active Directory Premium*. Opgehaald van Microsoft Azure: http://msdn.microsoft.com/library/azure/dn532272.aspx

Microsoft, Meier, J., Farre, C., Bansode, P., Barber, S., & Rea, D. (2007, Augustus 27). *Performance Testing Guidance for Web Applications.* Opgehaald van CodePlex: http://perftestingguide.codeplex.com/downloads/get/17955

OutSystems. (2014). *Get Started with OutSystems® Platform Today*. Opgehaald van OutSystems: http://www.outsystems.com/get-started/

Redhotminute. (2014, Februari). *Diensten*. Opgehaald van Redhotminute: http://www.redhotminute.com/full-service-internetbureau/

Redhotminute. (2014). *Over ons*. Opgehaald van Redhotminute: http://www.redhotminute.com/aboutus/

Saccone, R. (2011, Maart 3). *Thread Pools*. Opgehaald van Microsoft: http://msdn.microsoft.com/en-us/magazine/cc163327.aspx

ScottGu. (2013, April 8). *Windows Azure: Active Directory Release, New Backup Service + Web Site Monitoring and Log Improvements*. Opgehaald van ScottGu's Blog: http://weblogs.asp.net/scottgu/archive/2013/04/08/windows-azure-active-directory-general-availability-new-backup-service-web-site-monitoring-and-diagnostic-improvements.aspx

Sluijter, S., & De Kam, S. (2013). *Afstudeerverslag v1.0 Steven de Kam & Stefan Sluijter.* Tuil.

SomethingSomewhere. (2013, Februari 25). *JavaScript and Friends: CoffeeScript, Dart and TypeScript*. Opgehaald van Something Somewhere: http://smthngsmwhr.wordpress.com/2013/02/25/javascript-and-friends-coffeescript-dart-and-typescript/#conclusions

Thinktecture. (2012). *Thinktecture identityserver 2*. Opgehaald van Thinktecture: http://thinktecture.github.io/Thinktecture.IdentityServer.v2/

Trosradar. (2006, Juni 28). *Teveel wachtwoorden problematisch*. Opgehaald van Trosradar: http://www.trosradar.nl/nieuws/archief/detail/article/teveel-wachtwoorden-problematisch/

Twitter. (2014). *Twitter limits (API, updates, and following)*. Opgehaald van Twitter: https://support.twitter.com/entries/15364-about-twitter-limits-update-api-dm-and-following

Videos, t. (2014, April 8). *Thinktecture IdentityManager - Preview 1*. Opgehaald van Vimeo: http://vimeo.com/91470580

Webopedia. (2014). *Queue*. Opgehaald van Webopedia: http://www.webopedia.com/TERM/Q/queue.html

Wikibooks. (2013, April 18). *Implementeren van informatiesystemen*. Opgehaald van Wikibooks: http://nl.wikibooks.org/wiki/Implementeren_van_informatiesystemen

Windows. (2011). *Thread Pools*. Opgehaald van Windows: http://msdn.microsoft.com/en-us/library/windows/desktop/ms686760(v=vs.85).aspx

14. Bijlagen

14.1. Bijlage 1 : Plan van aanpak

AFSTUDEEROPDRACHT MOBO REQUESTS

PLAN VAN AANPAK

FERRY GRUITERS
REDHOTMINUTE B.V.
1581942

Versiebeheer

Datum oplevering	Auteur	Versie	Beschrijving
14-02-2014	Ferry Gruiters	0.1	Conceptversie
17-02-2014	Ferry Gruiters	0.2	Feedback RHM verwerkt
19-02-2014	Ferry Gruiters	1.0	Sequence Diagrams toegevoegd
27-02-2014	Ferry Gruiters	2.0	Onderzoeksvragen toegevoegd

Tabel 6 : Versiebeheer

Distributielijst

Naam	Instantie	Functie	Datum
Ferry Gruiters	Hogeschool Utrecht	Student	27-02-2014
Nico Lubbers	Redhotminute	Bedrijfsbegeleider	27-02-2014
Marco Dumont	Hogeschool Utrecht	Docentbegeleider	27-02-2014

Tabel 7 : Distributielijst

Begrippenlijst

Begrip	Betekenis
Epic	Een functionaliteit die gerealiseerd gaat worden.
User story	Een functionaliteit binnen een epic. Een epic bestaat meestal uit verschillende user stories.
Taak	Een taak (duurt meestal max. 4 uur) die uitgevoerd moet worden om een user story te kunnen realiseren. Een user story bestaat meestal uit verschillende taken.

Tabel 8 : Begrippenlijst

Management samenvatting

In dit document wordt het plan van aanpak beschreven voor het afstudeerproject ‘MOBO Requests’, waarbij er een bestaande applicatie uitgebreid wordt zodat het product in productie genomen kan worden.

Uit interviews en overleg met de product owner is gebleken dat er in ieder geval twee functionaliteiten, ook wel epics genoemd, aan het systeem toegevoegd moeten gaan worden. Dit zijn de epics ‘lijsten’ en ‘synchrone verwerking’.

De epics zullen door middel van de scrum werkwijze ontwikkeld worden. Dat wil zeggen dat elke epic in user stories opgebouwd wordt, gefaseerd in sprints die twee weken duren. Om de twee weken zullen de resultaten met de product owner besproken worden om vervolgens de volgende sprint te bepalen.

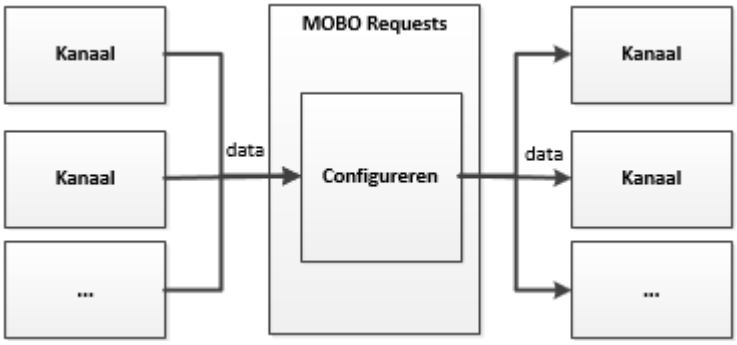
Het project zal uiterlijk 27 juni opgeleverd worden. De totale hoeveelheid studiebelastinguren zal hiermee uiteindelijk ongeveer op 840 uur uitkomen.

Inhoudsopgave

1. Aanleiding opdracht

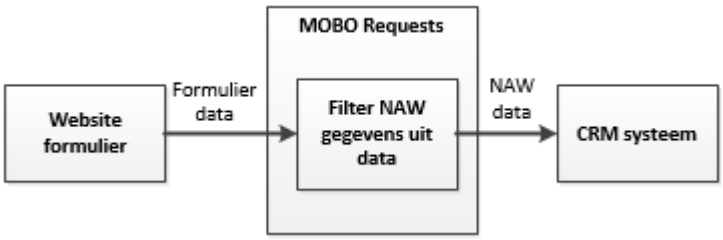
In 2013 is er door een tweetal studenten een systeem voor Redhotminute gerealiseerd dat data van verschillende kanalen kan ontvangen, bijvoorbeeld van een websiteformulier of Facebook/Twitter API, waarna de data in het systeem eventueel bewerkt kan worden om het vervolgens naar een ander kanaal door te sturen. Denk bij het bewerken van de data bijvoorbeeld aan het filteren van de hoeveelheid data dat binnenkomt, waarbij uiteindelijk alleen specifieke gegevens doorgestuurd worden. (Zie Figuur 44 : Praktijkvoorbeeld MOBO Requests)
Indien het systeem onverwachts uitvalt, mag deze data niet verloren gaan. Het systeem is dus eigenlijk een soort tussenstation voor verschillende kanalen zodat data met elkaar uitgewisseld kan worden.
Het systeem wordt ook wel MOBO Requests genoemd, wat staat voor 'Mid-Office / Back-Office aanvragen'.

De High-level architectuur van MOBO Requests ziet er als volgt uit:



Figuur 43 : High-level architectuur MOBO Requests

Een praktijkvoorbeeld:



Figuur 44 : Praktijkvoorbeeld MOBO Requests

Het platform, zoals het hierboven beschreven staat, is op dit moment klaar. Desondanks is het systeem nog niet in productie. Dit komt doordat het systeem nog niet volwassen genoeg is om geheel in gebruik genomen te worden. Dit komt omdat de vorige studenten simpelweg niet genoeg tijd hadden om het systeem verder uit te breiden. Redhotminute wil het platform wel graag in productie krijgen en ziet dit dus als een mooi project voor afstudeerders, omdat dit een mooi leerproject is om het gehele ontwikkelproces te laten leren kennen.

Er wordt meer over de achtergrond van MOBO Requests verteld in Hoofdstuk 2.3 : MOBO Requests. Waarom het systeem nog niet volwassen genoeg is en welke functionaliteiten er ontbreken wordt in Hoofdstuk 3 : Het op te lossen probleem en de te vervullen behoeften uitgelegd.

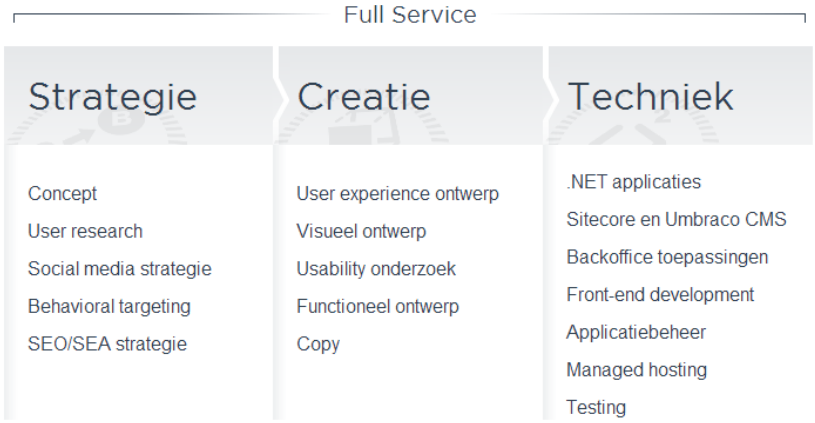
2. De organisatie

In dit hoofdstuk zal meer over het bedrijf Redhotminute verteld worden waar het afstudeerproject plaats zal vinden. Ook wordt er meer informatie gegeven over de achtergrond van MOBO Requests.

2.1. Redhotminute

De stageopdracht vindt plaats bij Redhotminute, gevestigd in Tuil. Redhotminute is een full-service bedrijf dat websites, webwinkels, webapplicaties, online campagnes, mobiele sites en apps maakt. Het bedrijf is opgericht in 2001 en heeft ondertussen 90 werknemers. De functies van de werknemers variëren van Designers tot aan Testers, van Functioneel Ontwerpers tot Front-end ontwikkelaars en van Software Engineers tot aan CMS specialisten. Kortom, Redhotminute bestaat uit een hecht en ervaren team van online professionals met uitgebreide kennis van belangrijke aspecten van online marketing. Met het grote team aan specialisten werken ze onder andere voor bedrijven als KIA, Van der Valk, KLM en Landal Greenparks. (Redhotminute, Over ons, 2014)

Met full-service willen ze het volgende zeggen:



Figuur 45 : Full Service (Redhotminute, Diensten, 2014)

2.2. Agile

Redhotminute is een overtuigd aanhanger van de Agile ontwikkelmethodieken. De ervaring binnen en buiten Redhotminute is dat een Agile methodiek sneller leidt tot werkende producten, minder overbodige functionaliteiten geeft en naast veel andere voordelen ook meer commitment van medewerkers brengt.
Ontwikkeling en oplevering van een platform verlopen volgens een continu proces. In het 2-weekse ritme van een Agile proces krijgt de klant steeds de zaken geleverd die de hoogste prioriteit hebben. Een onderdeel dat heel belangrijk is kan in principe altijd direct in de volgende sprint worden opgepakt. Een onderdeel dat gereed is, uiteraard mits goed getest en door de opdrachtgever geaccepteerd, is dan direct live beschikbaar voor de klant en haar gebruikers. (Bek, Werkwijze Redhotminute, 2014)

Binnen de organisatie werkt Redhotminute met een aantal teams, waarbij elk team met een project bezig is, bijvoorbeeld voor Van der Valk. Ik zal niet in een team meedraaien, maar zal alleen aan MOBO Requests verder werken. Natuurlijk word ik in het project wel begeleid door de bedrijfsbegeleider en zal één van de studenten die aan MOBO Requests begonnen is, en nu bij Redhotminute in dienst is, mij ondersteunen op de momenten dat dit nodig is.
Aangezien er bij Redhotminute volgens scrum (de Agile methodiek) gewerkt wordt, zal ik ook via deze methode werken. Hierdoor krijg ik zoveel mogelijk aansluiting bij het bedrijf. In dit verslag zal ik daarom direct gebruik maken van begrippen die bij de Agile methodiek gebruikt worden, namelijk: ‘epics’, ‘user stories’ en ‘taken’.

2.3. MOBO Requests

Redhotminute heeft in het verleden een Enterprise Service Bus (ESB)7 ontwikkeld die voor de website van een specifieke klant werd ingezet. Dit systeem is geschreven om de complexiteit in formulieren op de website van de klant tussen de verschillende landen te verkleinen. (Ze hebben voor verschillende landen verschillende websites).
Het systeem is echter specifiek voor die ene klant ontworpen en kan dus niet voor elke website worden ingezet. Redhotminute heeft daarom onderzoek gedaan naar bestaande ESB systemen, maar deze bevatten veel functionaliteiten die voor Redhotminute overbodig zijn waardoor het programma te onhandig wordt. Daarnaast zou Redhotminute teveel voor deze systemen betalen, aangezien ze niet alle functionaliteiten zullen gebruiken.

Daarom heeft Redhotminute zelf een systeem laten bouwen dat specifiek aan de wensen van Redhotminute voldoet. Dit systeem ziet er op dit moment als volgt uit:

Het systeem kan data ontvangen van een aantal vastgestelde kanalen (bijvoorbeeld Facebook of een websiteformulier). Deze informatie wordt verwerkt, getransformeerd en verzonden naar één of meerdere vastgestelde kanalen (bijvoorbeeld een mailserver of een CRM).
Nadat de data verzonden is, mag deze data niet verloren gaan, ook niet indien het systeem onverwachts afgesloten wordt.

Gebruikers kunnen via Windows Azure inloggen en hebben vervolgens via een eenvoudige user interface de mogelijkheid om input- en outputkanalen aan elkaar te verbinden. Ook kan de gebruiker aangeven welke transformaties de data ondergaat en hoe vaak de data opnieuw verzonden moet worden indien een outputkanaal niet beschikbaar is. De gebruiker kan ook een zogenaamde ‘Fallback scenario’ opgeven wanneer een outputkanaal na aantal keren proberen nog niet reageert. Dit kan bijvoorbeeld het versturen van een e-mail zijn.
Alle acties die het systeem uitvoert worden bijgehouden en opgeslagen. De gebruiker moet inzicht krijgen in de acties die het systeem heeft uitgevoerd. Welke data transformaties hebben plaatsgevonden? Welke data heeft het systeem ontvangen en verzonden? Zijn de acties allemaal succesvol of misschien gefaald?

3. Het op te lossen probleem en de te vervullen behoeften

Zoals in de aanleiding van de opdracht al uitgelegd is, ontbreken er op dit moment functionaliteiten aan het systeem MOBO Requests. Doordat deze functionaliteiten ontbreken, kan het systeem niet volledig ingezet worden, aangezien bepaalde toepassingen deze functionaliteiten nodig hebben. Om het systeem in productie te krijgen, moet er een aantal functionaliteiten toegevoegd worden en heeft het systeem een aantal verbeterpunten. Voordat er aan de stageopdracht begonnen is, waren er al twee functionaliteiten bekend die in ieder geval ontbreken, namelijk ‘Repeating items’ en een ‘BAM’ functionaliteit. Om een compleet overzicht te krijgen van de overige verbeterpunten zijn er interviews met medewerkers van Redhotminute gehouden die als stakeholders aangewezen zijn. De uitleg van de verbeterpunten en de resultaten van de interviews worden in Hoofdstuk 3.1 : Ontbrekende functionaliteiten weergegeven. Naast de ontbrekende functionaliteiten werkt de authenticatie van Windows Azure op dit moment niet meer. Dit probleem moet eerst door, of in samenwerking met, een medewerker van Redhotminute verholpen worden.

[7] Een Enterprise Service bus (ESB) verbindt ERP-systemen en alle soorten applicaties op een eenvoudige manier. Hierdoor ontstaat een stabiele software-architectuur die simpel te onderhouden is. De verbindingen zijn niet langer ‘point-to-point’, maar verlopen centraal via de ESB. De werking van een ESB is te vergelijken met een tolk: het neemt informatie in een bepaalde taal en een specifiek format tot zich vanuit een applicatie, waarna het deze informatie aanbiedt aan andere applicaties in de juiste talen en formats. (Condor, 2014)

3.1. Ontbrekende functionaliteiten

De functionaliteiten ‘Repeating items’ en ‘BAM’ waren bij aanvang van de stage al bekend en onderbouwd. Desondanks zijn deze functionaliteiten in de interviews opgenomen om er een beter beeld van te krijgen. De overige verbeterpunten zijn allemaal uit de interviews naar voren gekomen.

1. Repeating items

Deze functionaliteit houdt in dat het systeem ook formulieren met lijsten kan verwerken. Op dit moment kan de applicatie enkel formulieren afhandelen waarin 1-dimensonale velden verwerkt zitten (bijvoorbeeld een formulier met velden voor NAW-gegevens). Het is de bedoeling dat het systeem formulieren kan verwerken waarin multidimensionale velden verwerkt zitten. Dit kan een formulier zijn met velden voor NAW-gegevens en een veld met een lijst (array) van bijvoorbeeld contactpersonen. Maar het kunnen ook velden zijn met lijsten in lijsten (bijvoorbeeld een lijst met automerken met daarin lijsten met autotypes).

2. BAM (Business Activity Monitoring)

Op dit moment is er nog geen Business Activity Monitoring waarin bepaalde informatie van MOBO Requests gemonitord kan worden. Denk hierbij bijvoorbeeld aan het slagingspercentage en de aanvraagtijd van bepaalde requests. Door deze informatie te monitoren krijgt men een betere indruk over de prestaties van het systeem.

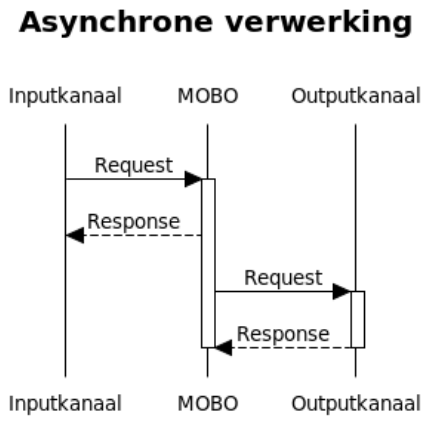
Overige ontbrekende functionaliteiten

Aan het begin van de stageopdracht was het nog niet bekend welke overige functionaliteiten er nog meer ontbreken. Er is daarom met medewerkers van Redhotminute gesproken om te achterhalen welke functionaliteiten er allemaal nog meer gerealiseerd kunnen worden.
De overige functionaliteiten die ontbreken zijn als volgt:

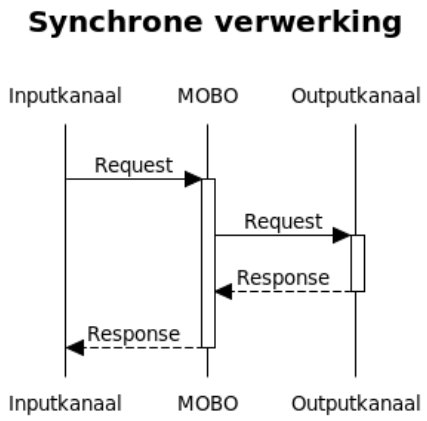
3. Synchrone verwerking

Op dit moment werkt het systeem asynchroon. Dat wil het volgende zeggen: een inputkanaal (bijvoorbeeld een webformulier) verstuurt informatie naar MOBO waarna het inputkanaal een response krijgt dat de data in MOBO ontvangen is. Vervolgens verstuurt MOBO de data door naar het outputkanaal (bijvoorbeeld een CRM), waarvan enkel MOBO een response krijgt, het inputkanaal krijgt er geen. Het inputkanaal heeft dus alleen een synchrone werking tot aan MOBO, maar niet met het outputkanaal. Zie Figuur 46 : Asynchrone verwerking.

De gewenste situatie is dat er een synchrone werking voor het gehele systeem is. Hierdoor kan de response van het outputkanaal op het inputkanaal weergegeven worden (bijvoorbeeld voor het opvragen van kentekengegevens uit een database). Zie Figuur 47 : Synchrone verwerking.



Figuur 46 : Asynchrone verwerking



Figuur 47 : Synchrone verwerking

4. Data verwerken van Input Channels
Op dit moment is er nog geen mogelijkheid om data op te halen van een externe webservice, om deze vervolgens te verwerken in een service. Deze functionaliteit kan in de toekomst wellicht noodzakelijk zijn om volledig van systeem gebruik te kunnen maken, aangezien sommige processen dit zullen gebruiken.

5. Nieuwe Input Channels
Er is een aantal Input Channels die aan het systeem toegevoegd zouden kunnen worden, waaronder:

- Twitter Input Channel: Een Input Channel dat tweets ophaalt aan de hand van een zoekterm (bijvoorbeeld de zoekterm '#Redhotminute')
- Facebook Input Channel: Een Input Channel dat berichten/ foto's ophaalt van een Facebook pagina.

Voor beide kanalen moet het mogelijk zijn om een schema in te stellen, zodat bijvoorbeeld elk uur op nieuwe berichten gecontroleerd kan worden.
Om deze kanalen te kunnen implementeren is het volgens medewerkers noodzakelijk om eerst de functie 'Data verwerken van Input Channels' te realiseren.
Ook zou de software Copernica, wat gebruikt wordt voor o.a. e-mailmarketing en geautomatiseerde campagnes, als nieuwe Input Channel gebruikt kunnen worden.

6. Verbeterpunten
Buiten de geheel ontbrekende functionaliteiten om is er nog een aantal punten waarop MOBO verbeterd zou kunnen worden.

- Type velden uitbreiden: er worden binnen MOBO op dit moment al veel type velden ondersteund (zoals string, int, boolean, etc.). Dit kan echter nog uitgebreid worden, met bijvoorbeeld Key Value Pairs.
- Vormgeving/usability: de vormgeving zou op bepaalde plekken binnen MOBO verbeterd kunnen worden. Dit geldt ook voor de usability.
- Validatie: bepaalde velden worden binnen MOBO gevalideerd (controle op e-mail, leeftijd, int, etc.). Waarschijnlijk kan dit op diverse plekken verbeterd worden.

3.2. Prioritering
Aangezien er meerdere functionaliteiten ontbreken, moet er bepaald worden welke functies de hoogste prioriteit hebben om als eerste gerealiseerd te worden. De prioritering is in overleg met de product owner, aan de hand van de resultaten van de interviews, tot stand gekomen.
Functionaliteiten die gerealiseerd worden, worden in agile-termen ook wel 'epics' genoemd. Daarom zal gedurende het project deze term aangehouden worden.

Prioriteit 1:
De product owner wil eerst de epic 'Repeating items' gerealiseerd zien. Dit is een belangrijk onderdeel om MOBO op meerdere systemen toe te kunnen passen.

Prioriteit 2:
De tweede epic die gerealiseerd moeten worden is de synchrone verwerking.

Vervolg:
Aangezien er alleen aan het project gewerkt wordt, verwachten we dat er tijdens de hele periode aan deze twee epics gewerkt zal worden. Indien er nog tijd beschikbaar komt, dan zal de verdere prioritering nader bepaald worden.

4. Doelstelling

Het doel is om aan het eind van het afstudeerproject MOBO Requests zo ver uitgebreid te hebben dat het volwassen genoeg is om ingezet te kunnen worden. Hiervoor zullen de epics gerealiseerd worden in de volgorde van prioritering, zie Hoofdstuk 3.2 : Prioritering.

Het zou mooi zijn als alle epics en verbeterpunten die in Hoofdstuk 3.1 : Ontbrekende functionaliteiten besproken zijn gerealiseerd kunnen worden, maar gezien de hoeveelheid beschikbare tijd is dit niet realistisch. Het doel is daarom om de epics die geprioriteerd zijn op te leveren, die voldoen aan de volgende kwaliteitscriteria:

Repeating items:
- MOBO kan lijsten van een inputkanaal ontvangen.
- Binnen MOBO moet er door de lijst heengelopen kunnen worden.
- De lijst kan naar een outputkanaal gestuurd worden.
- De epic moet niet op een specifieke lijststructuur gebaseerd zijn, maar moet elke lijst kunnen verwerken. De inhoud van de lijst is namelijk vooraf onbekend.

Synchrone verwerking:
- Het gehele systeem werkt synchroon.
- De responsetijd van MOBO mag niet te lang zijn aangezien de cliënt op een response van MOBO moet wachten. Hierbij gaat het om de netto tijd die besteedt wordt door MOBO, en niet door het outputkanaal (bijvoorbeeld een CRM systeem). Dit is te meten door op zowel input als outputkanaal metingen te doen.

Algemeen:
Om het project een grotere kans van slagen te geven, is er van te voren een specifiek voorbeeld bedacht waar de te realiseren epics op toegepast kunnen worden. Het doel is om de epics in eerste instantie op dit voorbeeld toe te kunnen passen, waarna het vervolgens universeel ingezet kan worden. Tijdens dit project zullen de epics op het CRM systeem ZOHO gebaseerd worden. Dit CRM systeem wordt door Redhotminute in de praktijk gebruikt en bevat dus ook data waarmee in de praktijk gewerkt wordt. Elk veld dat door Redhotminute binnen ZOHO gebruikt wordt, moet door MOBO ondersteund worden.

Opmerking: De kwaliteitscriteria die hierboven beschreven staan kunnen gedurende het project bijgeschaafd worden.

5. De opdracht en onderzoeksvragen

Tijdens het project zullen alle aspecten rondom software development aan bod komen. Er zal geanalyseerd, ontworpen, geprogrammeerd en getest worden. De opdracht zal dus voornamelijk uit het programmeren van functies bestaan, maar inclusief bijbehorende werkzaamheden. Naast het realiseren van software zal er tijdens het project ook onderzoek gedaan worden. De vragen die gesteld worden hebben met name betrekking op de functionaliteiten die nog gerealiseerd moeten worden en hoe het product uiteindelijk in productie genomen kan gaan worden.

De hoofdvraag die hierbij centraal staat is als volgt: Hoe kan MOBO, zodra het volwassen genoeg is, foutloos op een bestaand systeem (van een klant) ingezet worden, zonder dat er gegevens verloren gaan of juist dubbel opgeslagen worden en wat dient er vóór de implementatie nog te gebeuren?

Om de hoofdvraag uiteindelijk te kunnen beantwoorden, zullen de volgende deelvragen beantwoord moeten worden:

- Hoe ziet elke toe te voegen functionaliteit er geheel uit, wat doet het, wat zijn de eisen aan de vormgeving, wat zijn de specifieke kwaliteitseisen, etc.?
- Welke tests moeten er nog uitgevoerd worden voordat MOBO op klanten toegepast kan gaan worden?
- Wat moet er, naast de ontbrekende functionaliteiten, aan het huidige systeem aangepast worden om MOBO op klanten toe te kunnen passen?
- Hoe moet er gecontroleerd gaan worden of MOBO alle data op het toegepaste systeem correct verwerkt, zonder dat er data verloren gaat of onbeheerst dubbel opgeslagen wordt?
- Hoe kunnen we er zo zeker mogelijk van zijn dat de prestaties (snelheid e.d.) van het systeem goed zijn/blijven na de implementatie van MOBO?
- Moet Microsoft Azure als authenticatie gebruikt blijven worden, of zijn hier betere/goedkopere alternatieven voor? Welke alternatieven zijn er?

6. Benodigde documentatie

Om aan het huidige systeem verder te kunnen werken, zal er ingelezen moeten worden hoe het systeem in elkaar zit. Tevens zal er tijdens het programmeren regelmatig wat opgezocht moeten worden over het huidige systeem. Hiervoor is de documentatie nodig die de makers van MOBO Request opgeleverd hebben. Tevens zal er nieuwe kennis opgedaan moeten worden, bijvoorbeeld over C#, ASP.NET of Microsoft SQL Server. Hiervoor kan documentatie op internet gebruikt worden.

7. Technieken en methoden

Tijdens het project zal er van een aantal technieken en methoden gebruik gemaakt worden.

7.1. Technieken

MOBO Requests is in C# / ASP.NET geschreven en maakt gebruikt van Microsoft SQL Server voor het opslaan van de data. Het programma is in deze taal geschreven aangezien dit een vereiste vanuit het bedrijf is. Microsoft SQL is gebruikt omdat het goed met de C# / ASP.NET applicatie overweg kan en omdat deze omgeving vanuit Redhotminute is ingericht.
De code is door de vorige studenten door middel van Unit Tests gecontroleerd.
Om het platform op de juiste wijze verder te ontwikkelen zal ik van dezelfde technieken gebruik maken. De software is ontwikkeld in Microsoft Visual Studie 2012. Dit programma wordt door alle ontwikkelaars binnen het bedrijf gebruikt en zal daarom ook door mij gebruikt worden.

7.2. Methoden

Om de voortgang van het project te bewaken, is er afgesproken om wekelijks een voortgangsrapportage naar de bedrijfsbegeleider te sturen en eventueel te overleggen. Door deze manier van werken blijft de begeleider precies op de hoogte van mijn vorderingen en de eventuele problemen waar ik tegenaan loop.

Zelf zal ik via scrum te werk gaan. Dit betekent dat er vooraf geen gedetailleerde planning voor de gehele stageperiode beschikbaar is. Elke twee weken zal ik een sprint review en een sprint planning meeting hebben. Er zal dan aan de bedrijfsbegeleider, de product owner en mogelijk aan één van de studenten die MOBO Requests gerealiseerd heeft getoond worden wat er de afgelopen twee weken gedaan is. Vervolgens zal de product owner, mogelijk in overleg met de bedrijfsbegeleider, bepalen wat er in de volgende sprint de hoogste prioriteit heeft. Tevens zal er van het programma Jira[8] gebruik gemaakt worden om de voortgang van de sprints te bewaken. In Jira zullen de epics, user stories en taken per sprint uitgewerkt staan waardoor er een duidelijk overzicht gecreëerd wordt.

[8] JIRA is the tracker for teams planning and building great products. Thousands of teams choose JIRA to capture and organize issues, assign work, and follow team activity. At your desk or on the go with the new mobile interface, JIRA helps your team get the job done. (Atlassian, 2014)

Voor het verschaffen van informatie, bijvoorbeeld over de toe te voegen functionaliteiten, zullen voornamelijk medewerkers ondervraagd worden. Dit zal via interviews gebeuren, dan wel via korte gesprekken. Om informatie over de technieken op te doen, zal er gebruik van bronnenonderzoek/literatuuronderzoek gemaakt worden en zal ik de medewerkers persoonlijk benaderen om me bij bepaalde dingen persoonlijk te helpen.

8. Project planning

Er is voor het project een globale planning gemaakt om van te voren een richtlijn te hebben. Door de scrumwerkwijze zal de planning per functie en per sprint later gedefinieerd worden.

8.1. Globale planning

Taak	Begin	Eind
Afstudeerstage	10 februari	27 juni
Plan van aanpak	10 februari	21 maart
Eerste concept	10 februari	14 februari
Functionaliteiten bepalen	13 februari	14 februari
Concept beoordelen	14 februari	14 februari
Functionaliteiten prioriteren	14 februari	14 februari
Eerste versie	17 februari	21 februari
Definitieve versie	24 februari	21 maart
Contract tekenen	N.t.b.	
Inlezen huidige platform	12 februari	26 februari
Bezoek docentbegeleider	26 februari	26 februari
Sprint 0	14 februari	28 februari
Backlog maken	14 februari	20 februari
Architectuurplaat maken	20 februari	28 februari
Realiseren functies (± 8x)	D.m.v. sprints van 2 weken	20 juni
Analyseren		
Ontwerpen		
Bouwen		
Testen		
Presenteren product owner	Eind van iedere sprint	
Scriptie	3 maart	3 juni
Tussentijdse beoordelingen	N.t.b.	
Presentatie afstudeer-voorzitting voorbereiden	3 juni	16 juni – 27 juni
Afstudeerzitting	16 juni – 27 juni	16 juni – 27 juni

Tabel 9 : Globale planning

De planning is uitgebreider uitgewerkt in een Gantt Chart die als bijlage is toegevoegd. Zie Bijlage 1 : Gantt Chart.

8.2. Mijlpalen

De volgende datums zijn mijlpalen in het project waar niet van afgeweken kan worden.

Mijlpaal	Deadline
Vrijdag 21 februari 2014:	Deadline eerste versie plan van aanpak.
Woensdag 26 februari 2014:	Afspraak bedrijfsbegeleider.
Vrijdag 21 maart 2014:	Deadline definitieve versie plan van aanpak + contract.
Dinsdag 3 juni 2014:	Deadline scriptie
16 juni – 27 juni 2014:	Afstudeerzitting

Tabel 10 : Mijlpalen

8.3. Randvoorwaarden

Tijdens het project moet er met een aantal randvoorwaarden rekening gehouden worden;

- Het afstudeerproject (het volgen van de afstudeeropdracht) begint op 10 februari 2014 en eindigt op 20 juni.
- Het gehele project, van voorbereiding tot afronding, zal ongeveer 840 studiebelastinguren in beslag nemen
- De werktijden zijn tussen 08:00 en 18:00, wat ongeveer 8 á 8.5 uur werktijd per dag oplevert.
- De docent- en bedrijfsbegeleider zijn het gehele project betrokken en bieden ondersteuning waar nodig.
- Het project voldoet zoveel mogelijk aan de kwaliteitseisen die in Hoofdstuk 4 : Doelstelling vermeld staan.

9. Eventuele risico's

Omdat er tijdens dit project met scrum gewerkt zal worden, is er een redelijke flexibele planning. Om de week zal er opnieuw bekeken worden hoe de voortgang is en hoe de volgende sprint eruit komt te zien. Indien een deadline niet gehaald is, kan er de volgende sprint mee verder gewerkt worden.

Om eventuele risico's in kaart te brengen, is er een tabel opgesteld waarin de bedreigingen en de tegenmaatregelen zijn opgenomen. Om de maat van de bedreigingen te bepalen, wordt de kans vermenigvuldigd met de impact, wat het risico oplevert. Kans en impact lopen van 1 tot 5, waarbij 1 een kleine kans/impact is en 5 groot. (CreatiefDenken, 2014)

Bedreiging	Tegenmaatregel	Kans	Impact	Risico
Tijd te kort	Planning aanhouden	3	4	12
Realiseren functie lukt niet	Hulp van expert inschakelen	3	4	12
Bronnen onbereikbaar	Informatie via begeleider verschaffen	2	4	8
Verlies data	Back-ups maken / cloud storage	1	5	5
Performance niet goed genoeg		3	1	3

Tabel 11 : Eventuele risico's

10. Bedrijfs- en persoonsgegevens

De volgende personen hebben een relatie met dit project:

Naam	E-mailadres	Telefoon	Functie
Ferry Gruiters	ferry.gruiters@student.hu.nl	06 1111 2629	Student
Nico Lubbers	nico.lubbers@redhotminute.com	0418 510 068	Bedrijfsbegeleider
Renso Bek	renso.bek@redhotminute.com	0418 510 068	CTO / Product owner
Stefan Sluijter	stefan.sluijter@redhotminute.com	0418 510 068	Software Engineer
Marco Dumont	marco.dumont@hu.nl	088 481 8283	Docentbegeleider

Tabel 12 : Bedrijfs- en persoonsgegevens

11. Bronnenlijst

Atlassian. (2014). *Jira*. Opgehaald van Atlassian: https://www.atlassian.com/software/jira

Bek, R. (2014, Januari 22). Werkwijze Redhotminute. (F. Gruiters, Interviewer)

Condor. (2014). *Enterprise Service Bus*. Opgehaald van Condor: http://www.condorsolutions.nl/technologie/enterprise-service-bus/

CreatiefDenken. (2014). *Risicoanalyse*. Opgehaald van CreatiefDenken: http://www.creatiefdenken.com/risicoanalyse-methode.php

Redhotminute. (2014, Februari). *Diensten*. Opgehaald van Redhotminute: http://www.redhotminute.com/full-service-internetbureau/

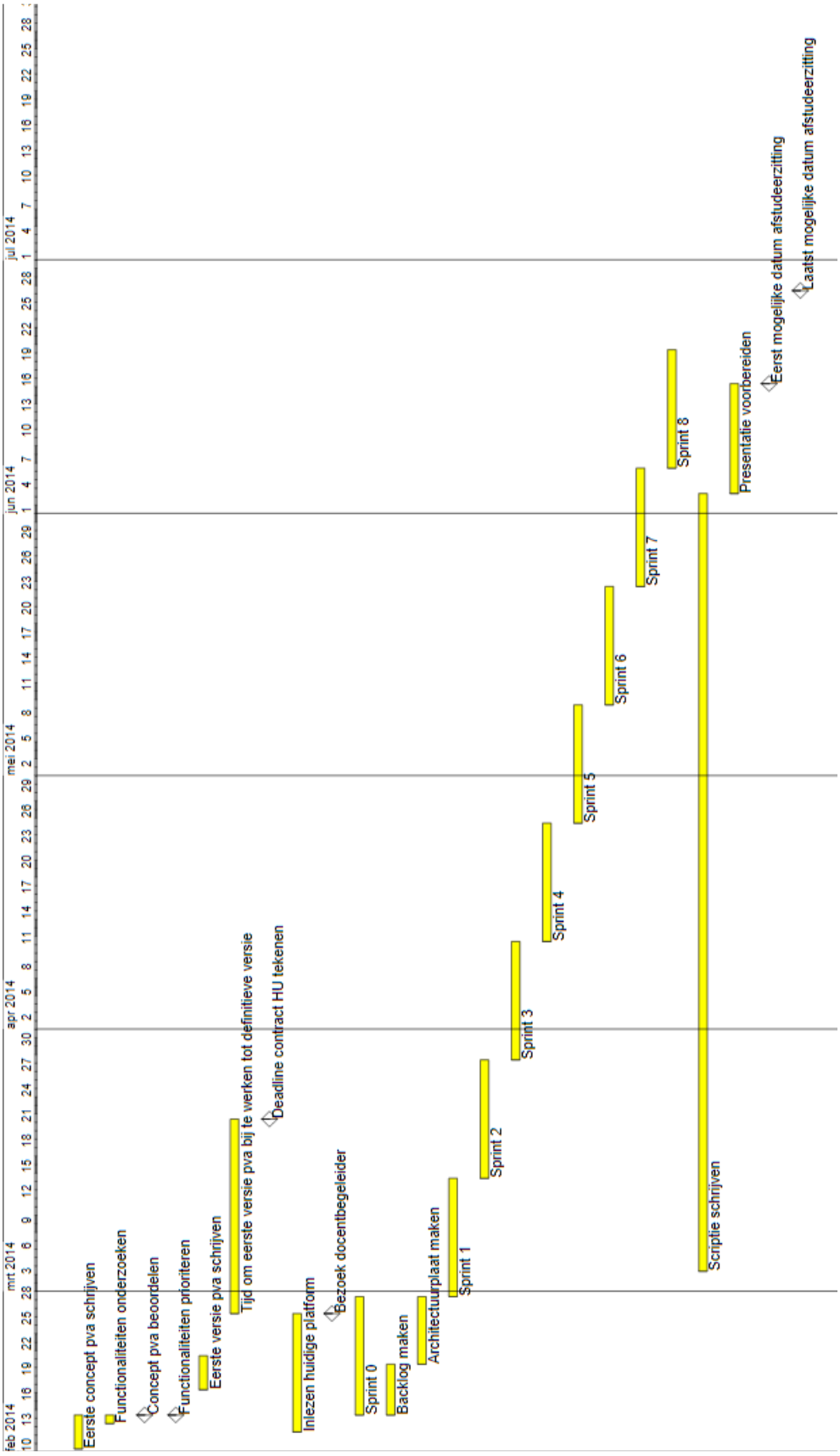
Redhotminute. (2014, Februari). *Over ons*. Opgehaald van Redhotminute: http://www.redhotminute.com/aboutus

Sluijter, S., & De Kam, S. (2013). *Afstudeerverslag v1.0 Steven de Kam & Stefan Sluijter 29-05-2013.* Tuil.

Wensink, M. (2014). *Afstudeerleidraad Instituut voor ICT cursus 2013-2014.* Utrecht: Hogeschool Utrecht.

12. Bijlagen

12.1. Bijlage 1 : Gantt Chart



14.2. Bijlage 2 : Evaluatie eigen functioneren

Na het schrijven van dit verslag kan ik terugkijken op een geslaagde periode, waarin veel gerealiseerd en onderzocht is.

Voorafgaand het project hoopte ik zoveel mogelijk functionaliteiten aan het systeem toe te kunnen voegen, waarbij ik in eerste instantie van ongeveer twee nieuwe functionaliteiten uitging. Achteraf is er 'maar' één functionaliteit toegevoegd, waar ik desondanks zeer tevreden mee ben. Het blijkt namelijk zeer complex om lijsten in het systeem te kunnen ondersteunen, aangezien elke functionaliteit geheel generiek opgezet moet worden. Doordat alles generiek moet zijn, ontstaan er snel problemen. Gelukkig is het gelukt om alle problemen op te lossen en kunnen lijsten nu probleemloos in het systeem verwerkt worden.

Dankzij de werkwijze waarop Redhotminute projecten indeelt, heb ik geleerd om mijn taken per periode beter in te delen. Door elke sprint opnieuw taken aan te maken en hier uren aan te koppelen, heb ik in de loop van de periode deze urenschatting steeds beter kunnen maken.

Wat de technische kant betreft, heb ik behoorlijk wat kennis opgedaan. Bij de aanvang van mijn stage had ik nog nooit aan een project gewerkt dat zo complex was als dit project. Dankzij de structuur waarmee het project is opgezet, heb ik geleerd om volgens dezelfde structuur te werken. Ik heb namelijk ervaren dat het ontzettend fijn is wanneer een project logisch en gestructureerd opgezet is, zeker indien je een project over moet nemen waar je nog geen ervaring mee hebt. Daarnaast had ik nog niet vaak in C# geprogrammeerd, kende ik CoffeeScript niet en had ik nog nooit met een WindowsService-applicatie gewerkt. Dankzij dit project is mijn kennis op deze gebieden verbreed.

Voorafgaand had ik besloten om diverse taken parallel te laten lopen, waaronder het onderzoek, het programmeren en het schrijven van de scriptie. Doordat ik al snel met mijn scriptie begonnen ben, heb ik nooit het gevoel gehad dat ik er met haast aan moest werken. Achteraf blijkt dit ook nadelig uit te kunnen pakken. Doordat ik bepaalde informatie soms te snel in het verslag opgenomen had, bleek deze informatie later soms weer overbodig. Hierdoor kan het voorkomen dat er tijd en (overbodige) informatie verloren gaat. Achteraf gezien had ik deze informatie beter minder uitgebreid op kunnen nemen, waardoor het minder tijd gekost had. Desondanks ben ik blij dat ik direct met het verslag begonnen ben, waardoor het schrijven ervan rustig is verlopen.

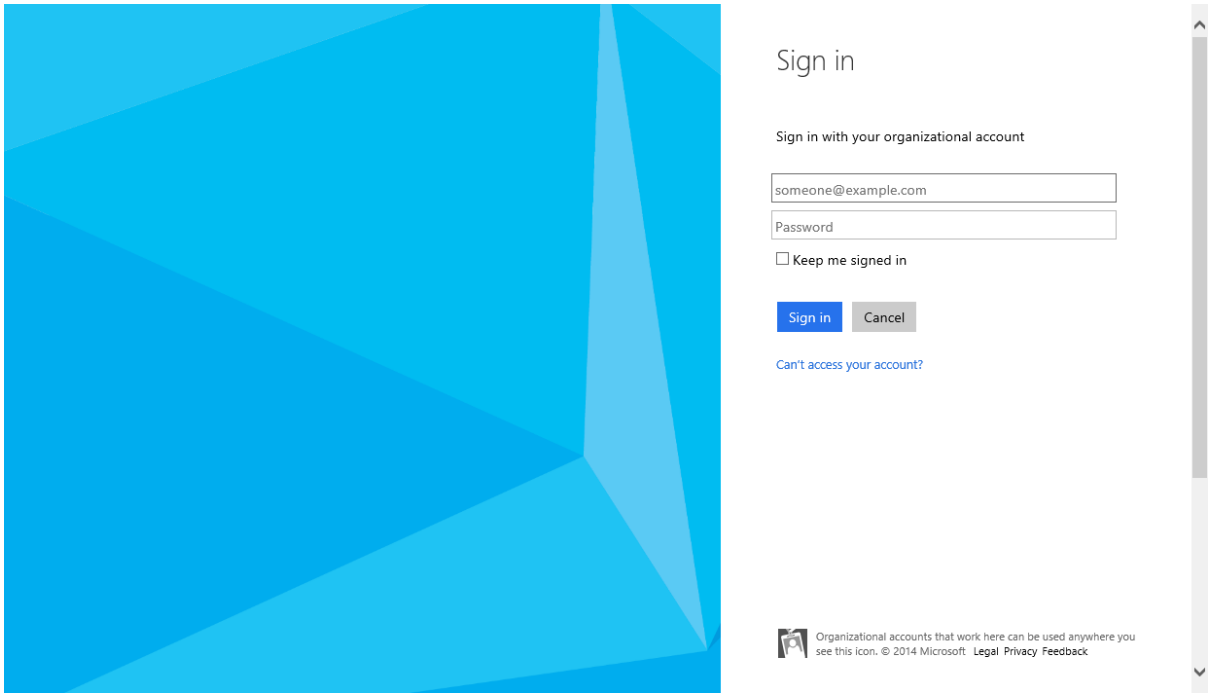
Indien ik ondersteuning kon gebruiken om bepaalde keuzes te maken, voornamelijk voor het realiseren van de functionaliteiten, heb ik ongeveer met vier verschillende medewerkers overlegd. Al de medewerkers waren back-enders, terwijl de keuzes soms ook op de front-end gebaseerd waren. Achteraf gezien had ik misschien beter met een front-ender of user experience designer kunnen overleggen, aangezien ze hier een andere blik op hebben.

Gezien mijn beperkte ervaring met programmeren, ben ik zeer tevreden met het resultaat dat ik behaald heb. De complexiteit van het project maakte het soms lastig om het overzicht te houden, zeker omdat ik helemaal alleen met het project bezig was. Niemand in de omgeving zat er net zo diep in als ik, waardoor het niet altijd mogelijk was om even snel hulp te krijgen. Door veel testscenario's te gebruiken en regelmatig Unit Tests te draaien is het uiteindelijk gelukt om deze lastige functionaliteit, zonder fouten, te implementeren waardoor MOBO een stap dichter bij livegang is. Dankzij de code-review, die af en toe gehouden is, sluit mijn code zoveel mogelijk aan op de bestaande code. Het overnemen van het project moet dus geen probleem zijn.

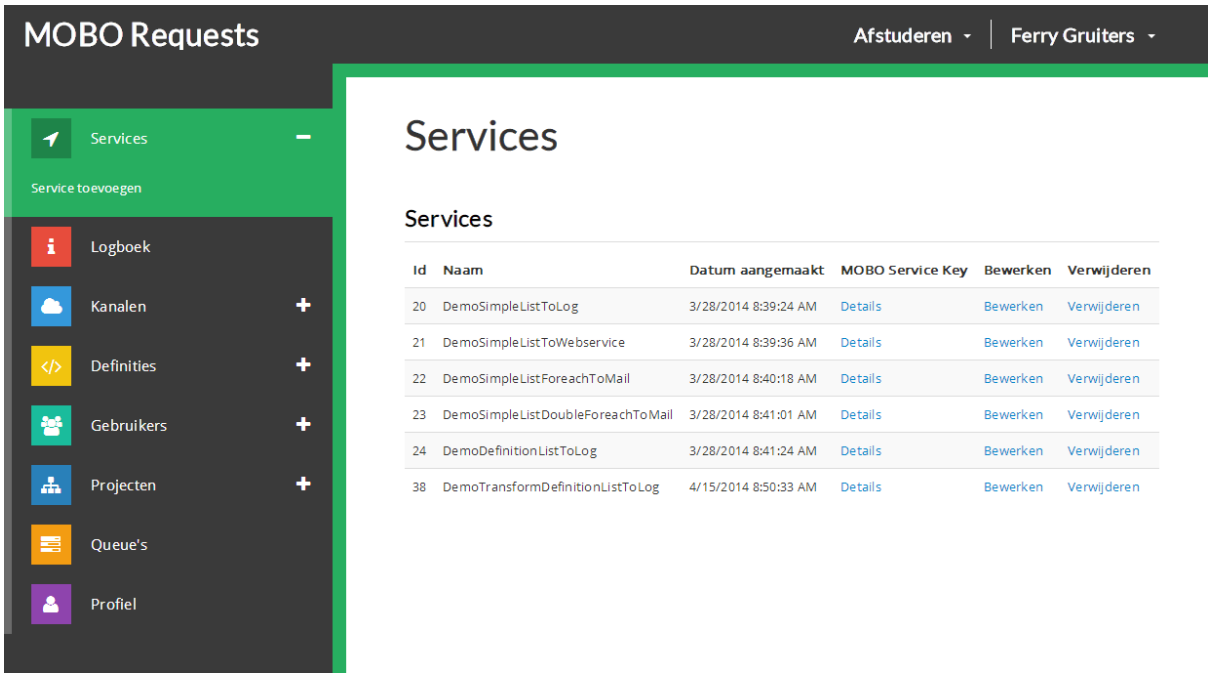
Kortom, het is een leuke en leerzame periode geweest waarin ik planmatig aan een mooi product heb kunnen werken. Hopelijk kan het product dankzij mijn bijdrage op korte termijn gebruikt worden.

14.3. Bijlage 3 : Impressie oude situatie MOBO

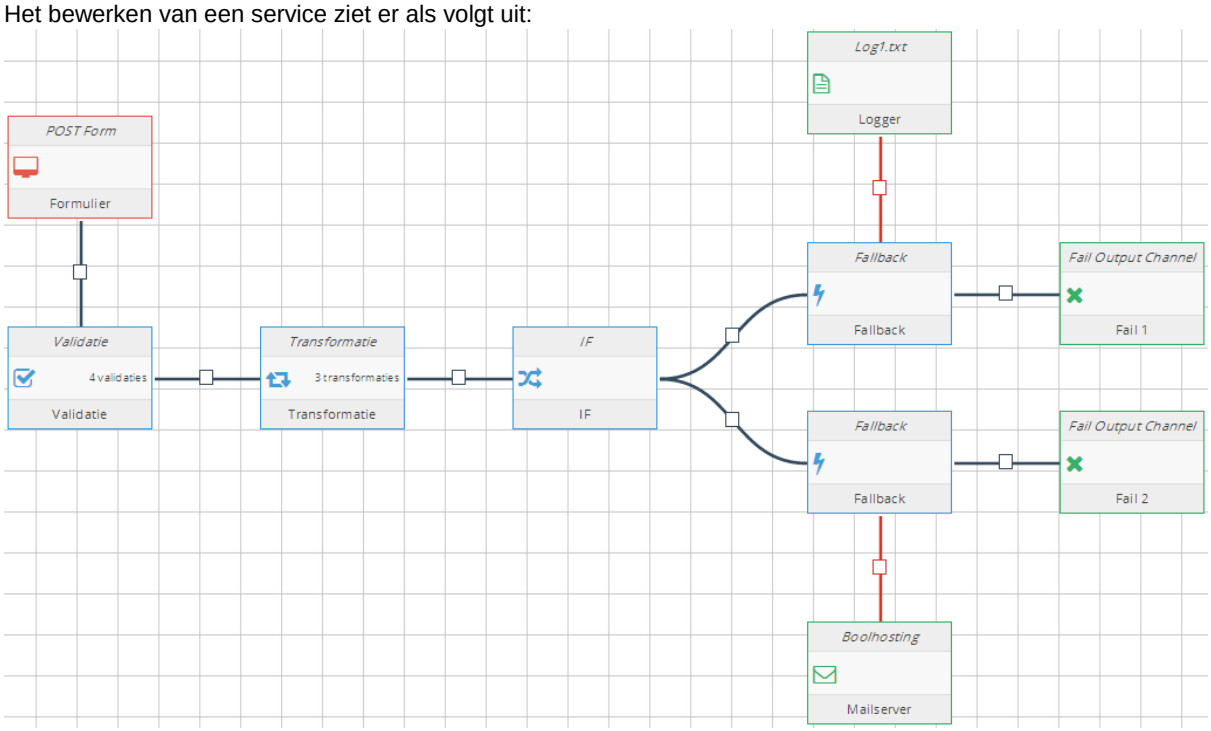
Aangezien er voor de authenticatie van Windows Azure Active Directory gebruik gemaakt wordt, wordt de gebruiker naar een Microsoft Login omgeving gebracht om in te loggen. Na het inloggen zal de gebruiker naar het controlpanel van MOBO doorverwezen worden.



Na het inloggen krijgt de gebruiker een menu te zien waar uit een aantal opties gekozen kan worden. De eerste optie is Services. De services zijn het belangrijkste onderdeel van het Controlpanel. Hierin kan de gebruiker services configureren waarin input- en outputkanalen (eventueel via componenten) met elkaar verbonden kunnen worden.



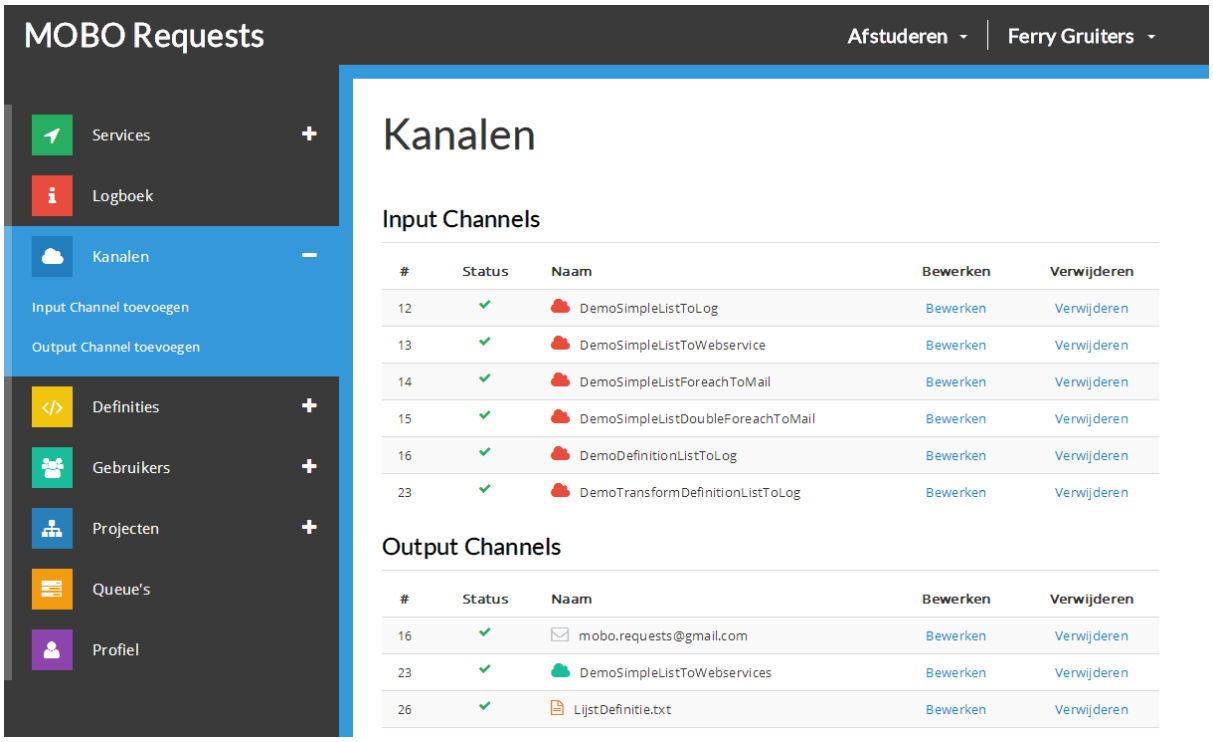
Het bewerken van een service ziet er als volgt uit:



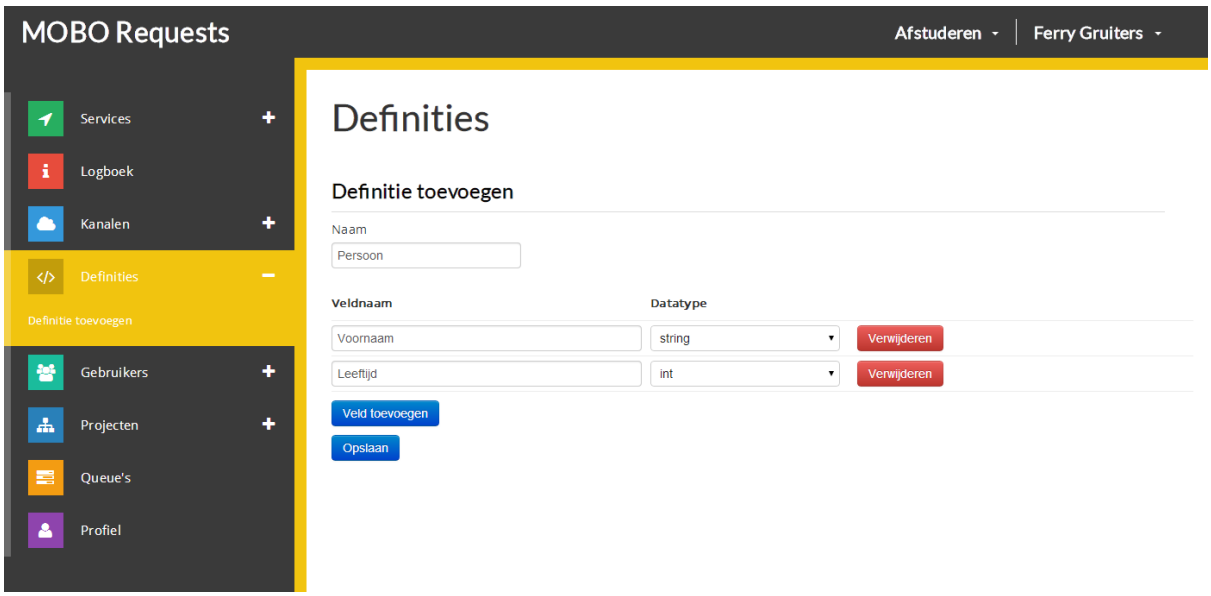
Op dit stuk canvas heeft de gebruiker de mogelijkheid om verschillende nodes aan elkaar te verbinden. De nodes bestaan uit inputkanalen, componenten en outputkanalen. Dit gedeelte is de kernfunctionaliteit van MOBO, aangezien de gebruiker hierin op een eenvoudige manier een service in elkaar kan zetten, zonder enige programmeerkennis nodig te hebben. De nieuwe functionaliteiten zijn, indien ze invloed hebben op de front-end, voornamelijk terug te zien zijn in dit gedeelte van MOBO.

De opties Logboek, Gebruikers, Projecten, Queue's en Profiel hebben weinig tot niets met het project te maken. Daarom zullen deze opties buiten beschouwing gelaten worden.

Bij de optie Kanalen heeft de gebruiker de mogelijkheid om input- en outputkanalen toe te voegen en te bewerken. Deze kanalen kunnen vervolgens in de services gebruikt worden.



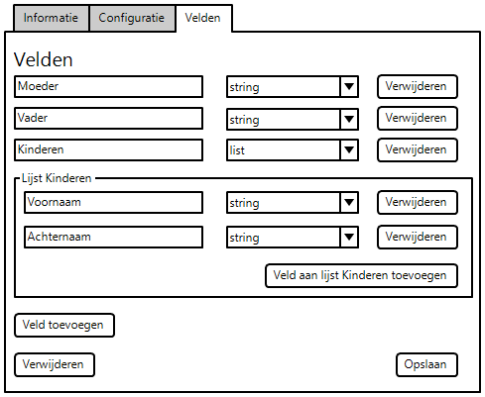
Bij de optie Definities kan de gebruiker definities definiëren. Definities zijn als het ware objecten met eigenschappen. Bijvoorbeeld een Persoon met de eigenschappen ‘Voornaam’ en ‘Leeftijd’.



14.4. Bijlage 4 : Schermontwerpen veldconfiguratie

Er is een aantal schermontwerpen gemaakt om tot een goed ontwerp te komen voor de veldconfiguratie waarin lijsten gedefinieerd kunnen worden. Om lijsten te kunnen selecteren, moet de lijst met datatypes uitgebreid worden. De optie 'list' zal hier aan toegevoegd worden.

Er is begonnen met een ontwerp waarin aangegeven kan worden waaruit een lijst bestaat.
Het eerste ontwerp zag er als volgt uit:



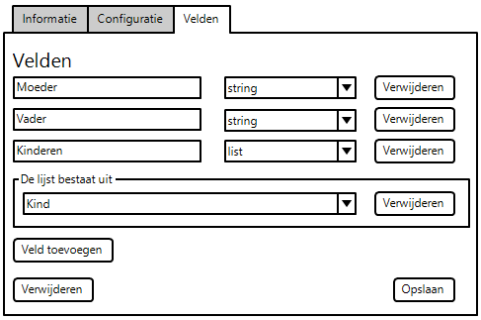
Door bij een veld het type 'list' te selecteren, krijgt de gebruiker de mogelijkheid om aan te geven welke velden er in een lijst-item zitten. Indien een lijst uit Kinderen bestaat, met ieder een voor- en achternaam, dan kan de gebruiker aangeven dat elk lijst-item een voor- en achternaam bevat.

Deze oplossing bleek na nader onderzoek niet de juiste. Een lijst van Kinderen bestaat namelijk niet uit items met een voor- en achternaam, maar uit een lijst van objecten. Bijvoorbeeld het object Kind. Het object Kind kan vervolgens de velden voor- en achternaam bevatten.

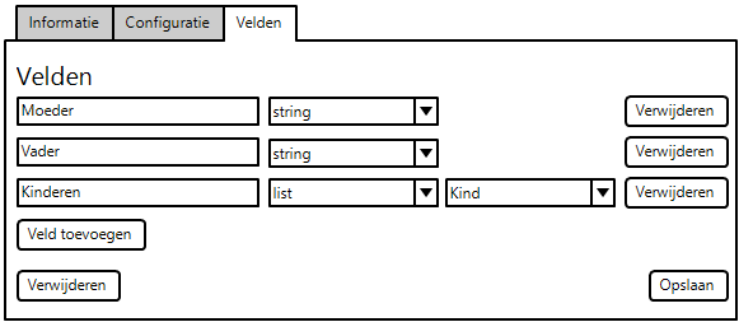
Tevens kan een lijst ook uit alleen simpele items bestaan, bijvoorbeeld een lijst van strings. In dit geval weet je niet of je een string met de naam 'voornaam' of met de naam 'achternaam' krijgt. Het opgeven van een veldnaam is in dat geval dus niet van toepassing.

Om deze problemen op te lossen, is er een andere oplossing bedacht:

Gebruikers kunnen nu aangeven waaruit een lijst bestaat. Dit kunnen simpele items zijn (string, int, bool, etc.), maar ook complexe items (Kind, Persoon, etc.). Aangezien complexe items (in MOBO definities genoemd) binnen MOBO al aangemaakt kunnen worden, kunnen deze gewoon voor de lijst-configuratie hergebruikt worden. De velden 'Voornaam' en 'Achternaam' die een kind bevat, zijn dus elders in het systeem gedefinieerd.



Uiteindelijk bleek dit ontwerp niet helemaal naar wens van de product owner. Doordat de configuratie van een lijst-item onder het bestaande veld wordt weergegeven, verspringt de lay-out van de pagina bij het veranderen van een veldtype. Dit kan tot onoverzichtelijke situaties leiden. Om hier een oplossing voor te bieden is het volgende ontwerp geschetst, waarbij de velden niet onder elkaar, maar naast elkaar getoond worden:

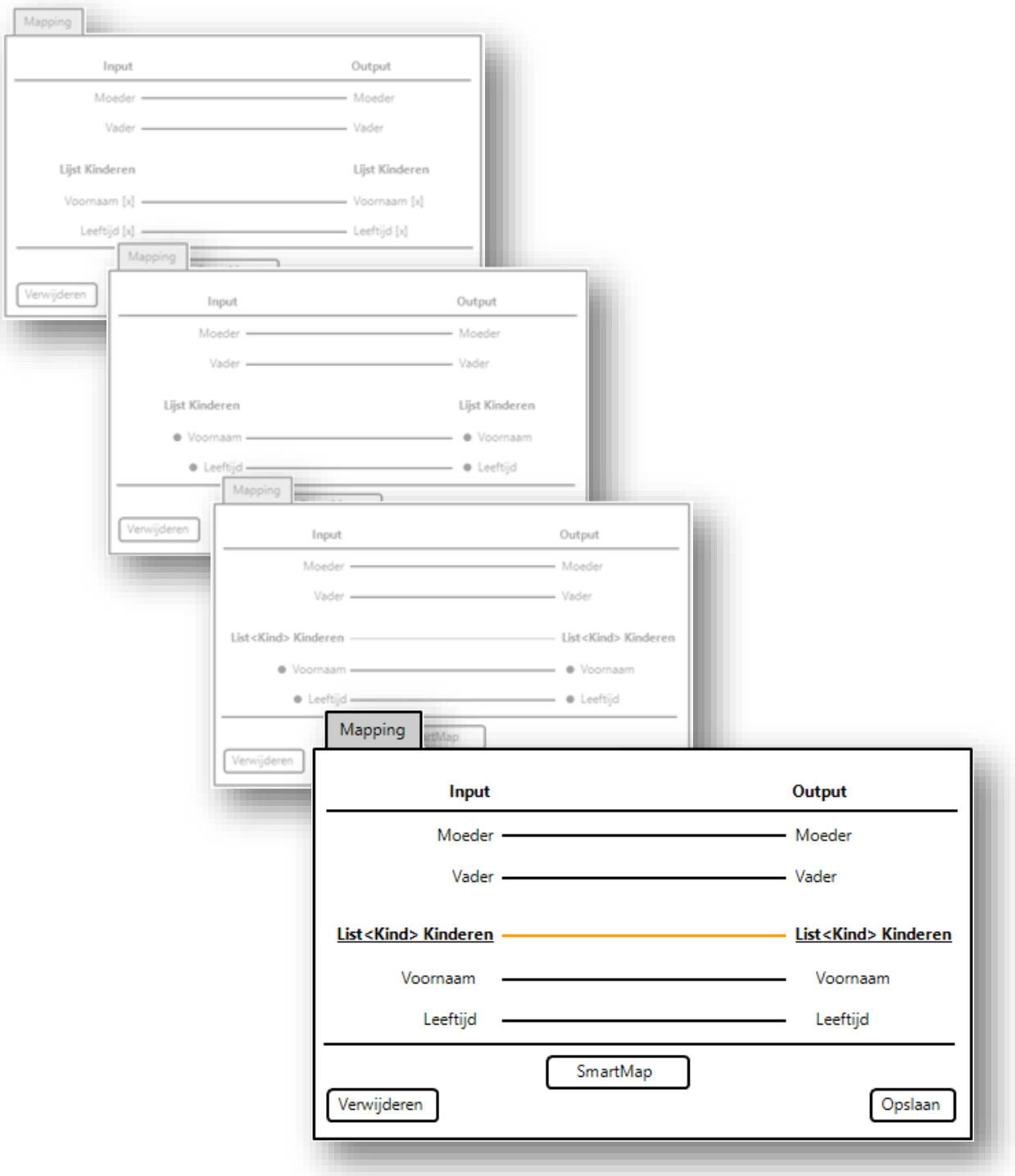


Dit ontwerp is uiteindelijk door de product owner goedgekeurd en is uiteindelijk gerealiseerd.

14.5. Bijlage 5 : Schermontwerpen Scenario 1

Net als bij het ontwerpen van de nieuwe veldconfiguratie, is er voor het ontwerpen van de verschillende scenario's gebruik gemaakt van schermontwerpen. In het eerste scenario moest er een ontwerp gemaakt worden voor het één op één mappen van een lijst.

Hiervoor is er een aantal ontwerpen gemaakt, waarbij er steeds wat aanpassingen zijn geweest om het visueel duidelijker te maken. Het laatste ontwerp is uiteindelijk door de product owner uitgekozen.

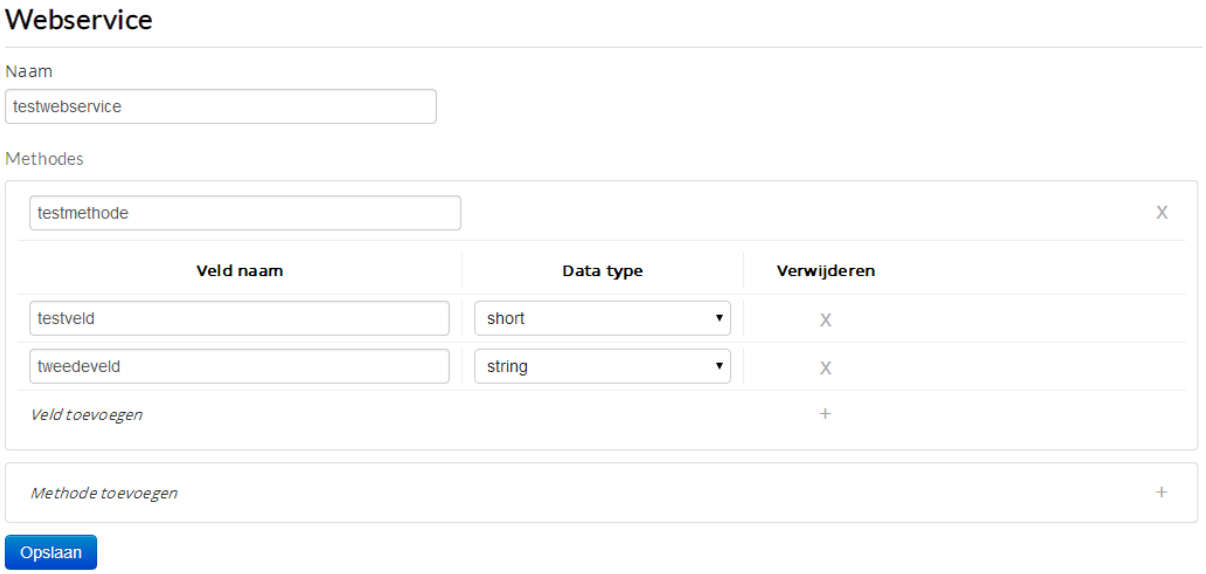


14.6. Bijlage 6 : Impressie nieuwe situatie MOBO

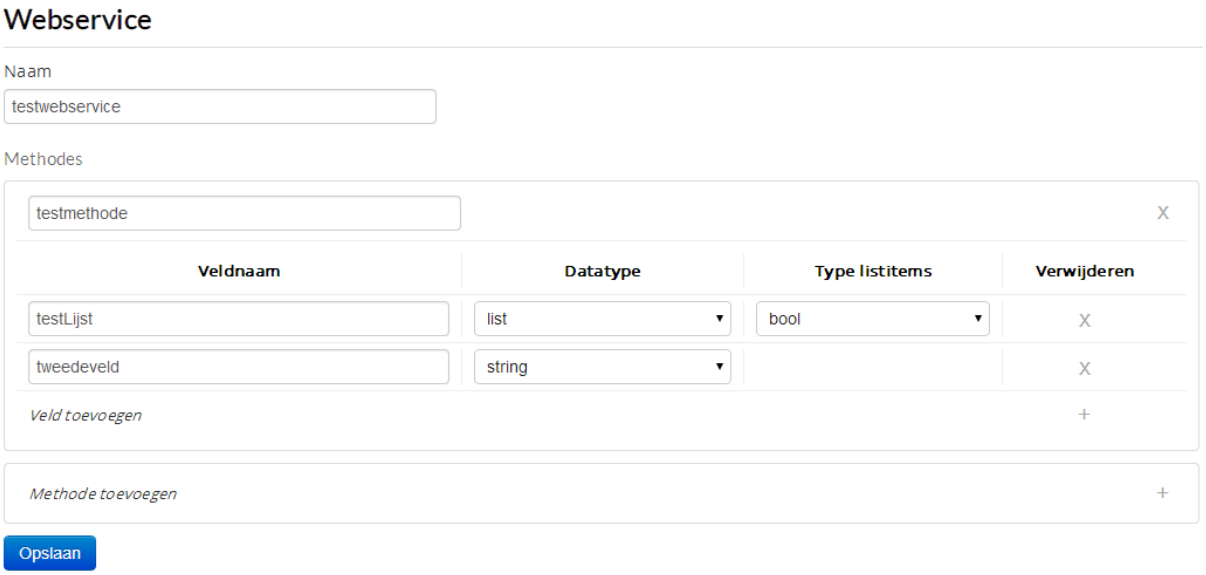
In deze bijlage zijn pagina's opgenomen uit de oude en nieuw versie van MOBO.

WEBSERVICE INPUT CHANNEL BEWERKEN:

Oude situatie:

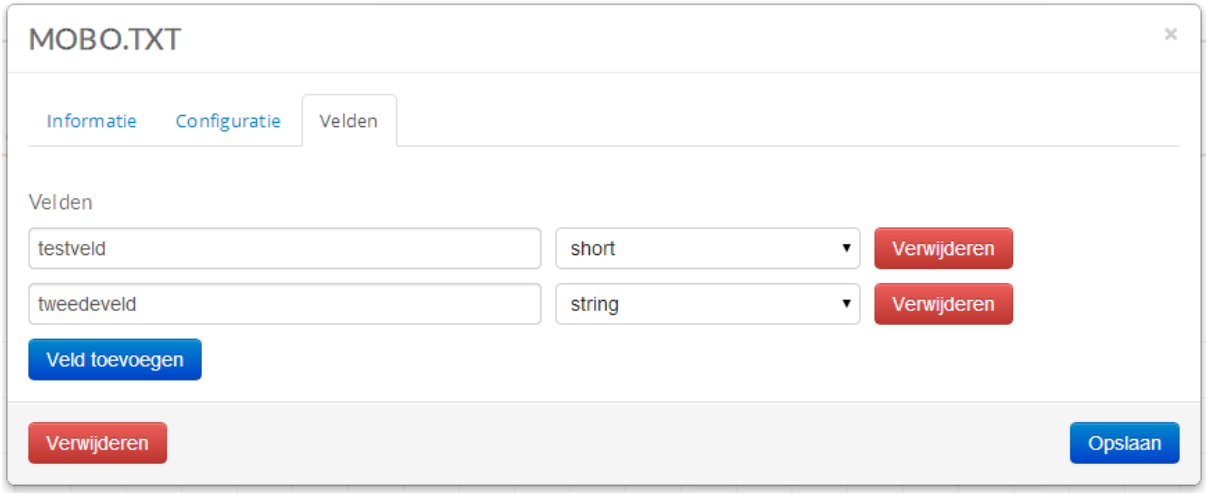


Nieuwe situatie:

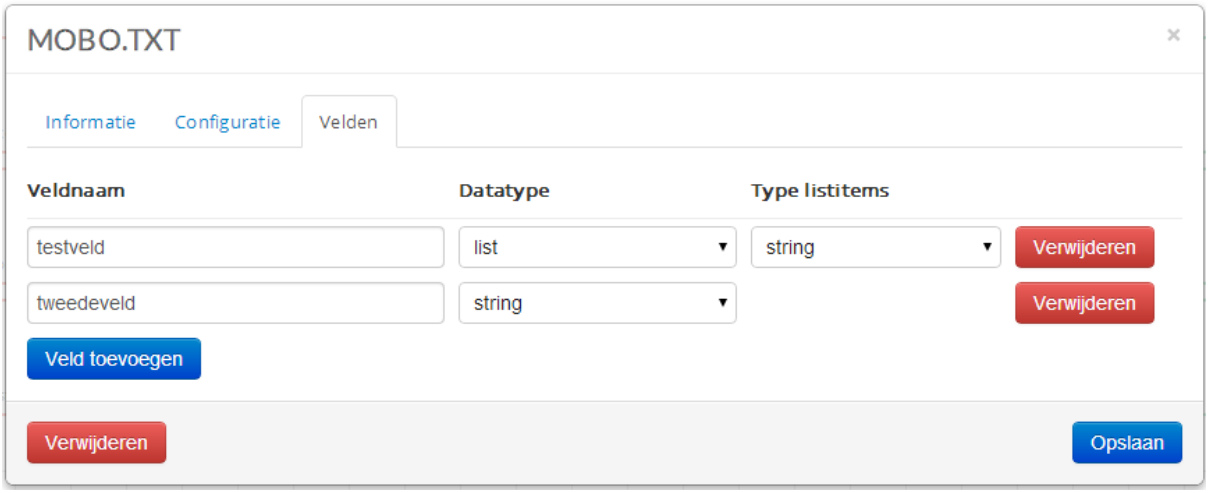


NODE VELDEN BEWERKEN:

Oude situatie:

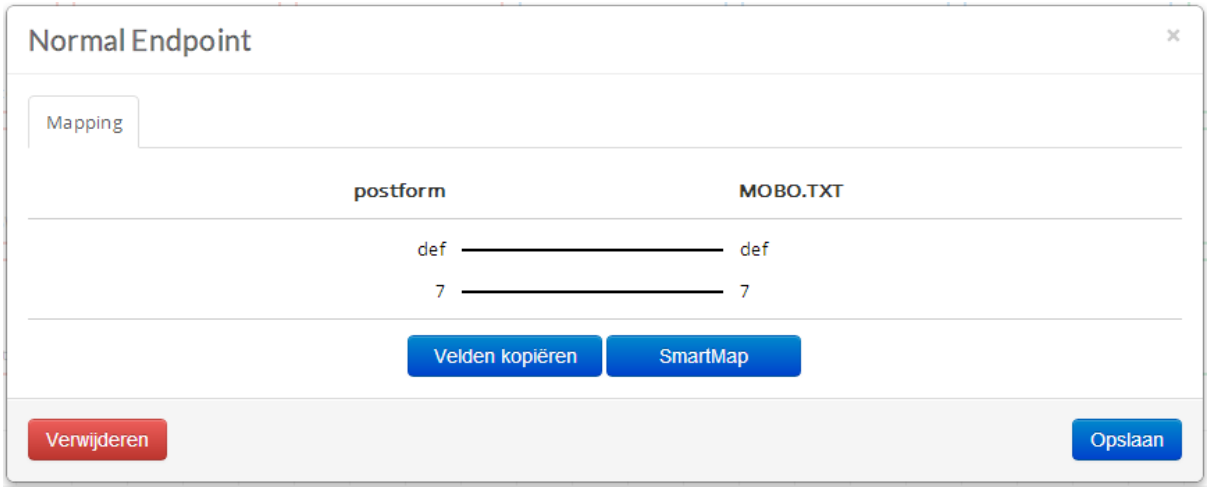


Nieuwe situatie:

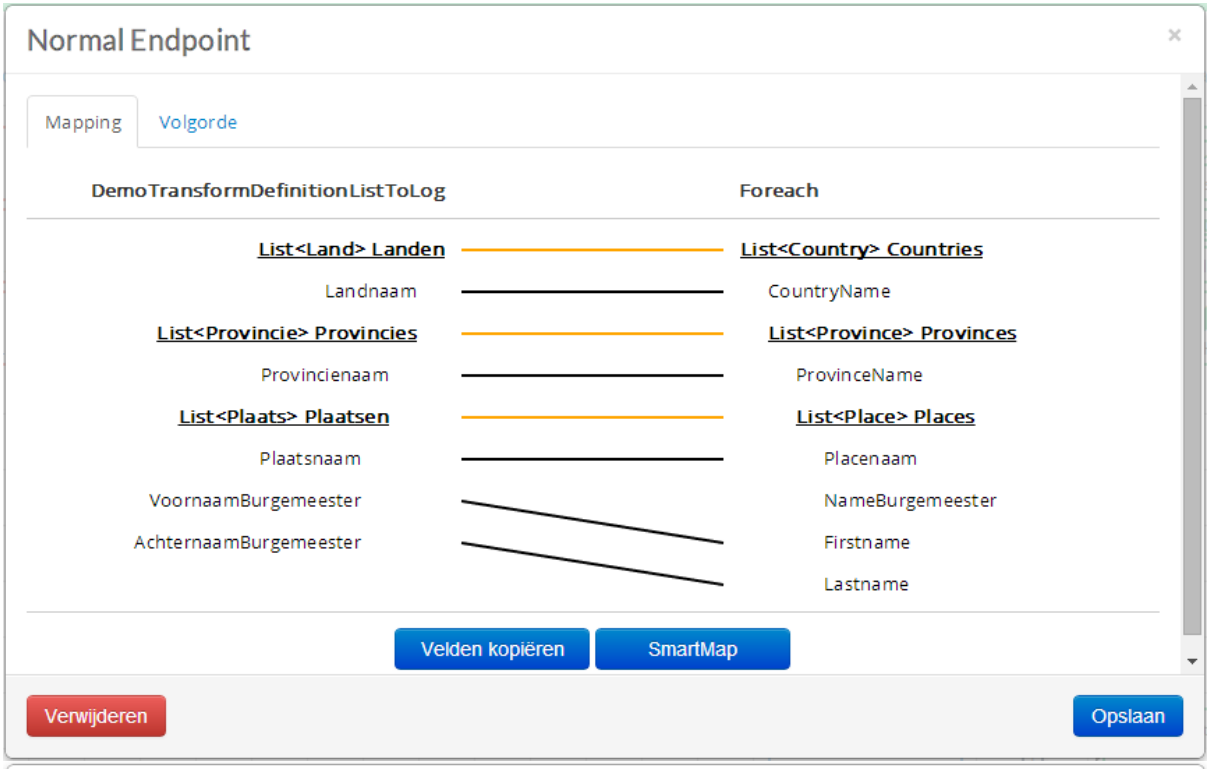


MAPPING:

Oude situatie:



Nieuwe situatie:

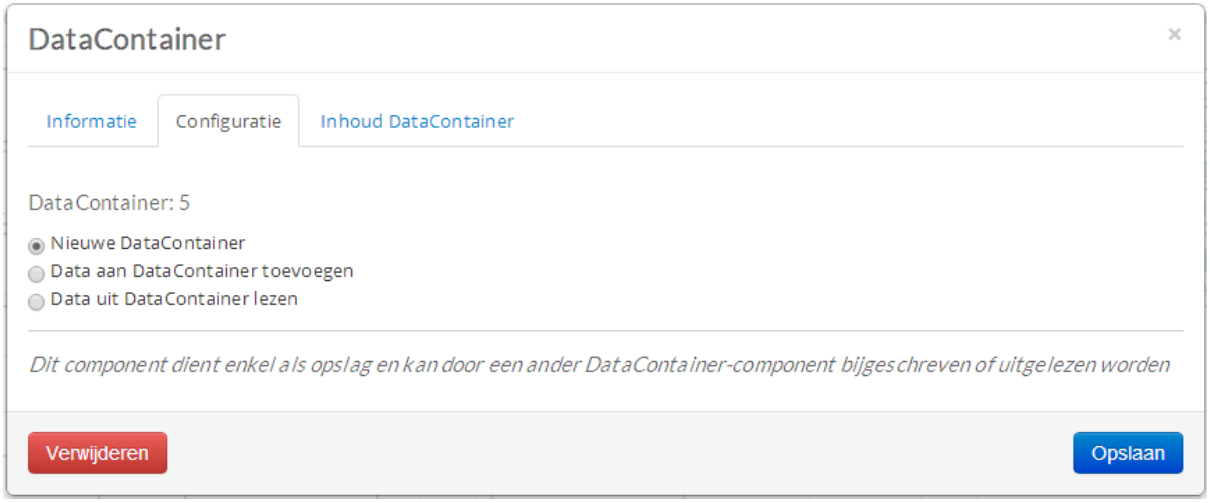


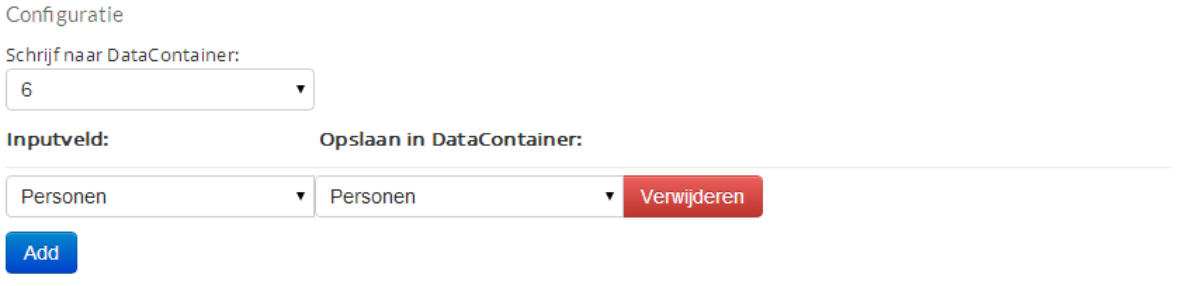
FOREACH COMPONENT:

Nieuwe situatie:

DATACONTAINER COMPONENT:

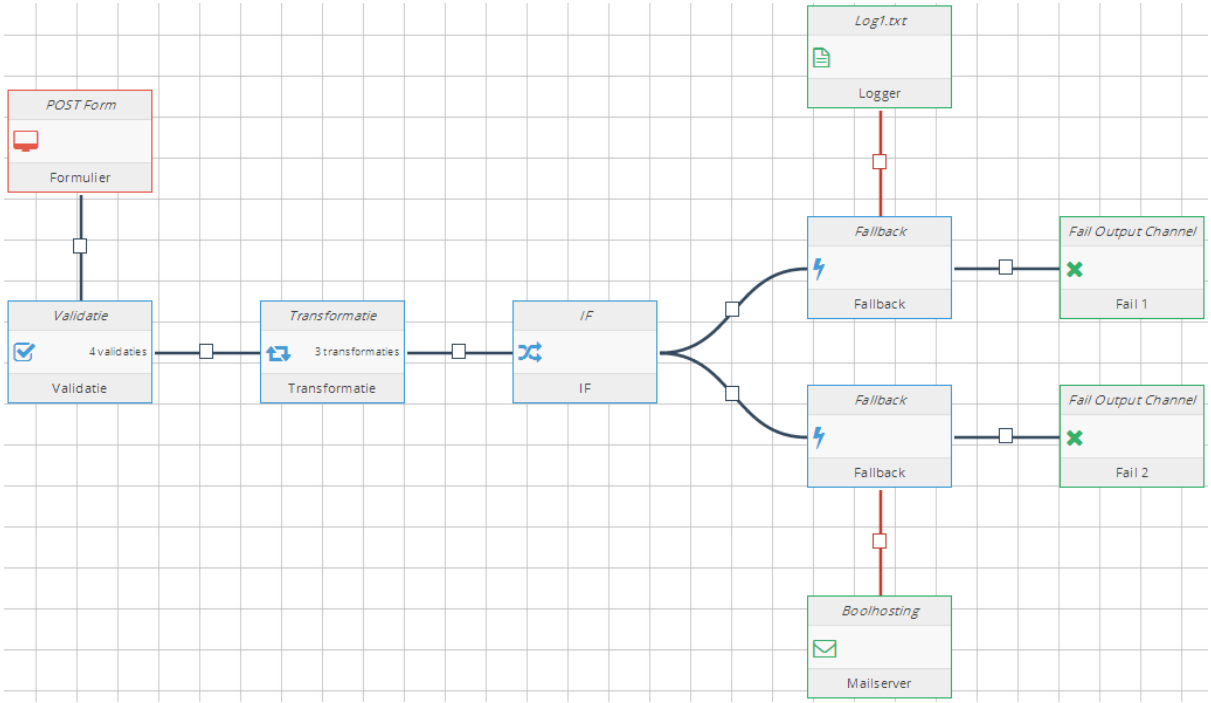
Nieuwe situatie:



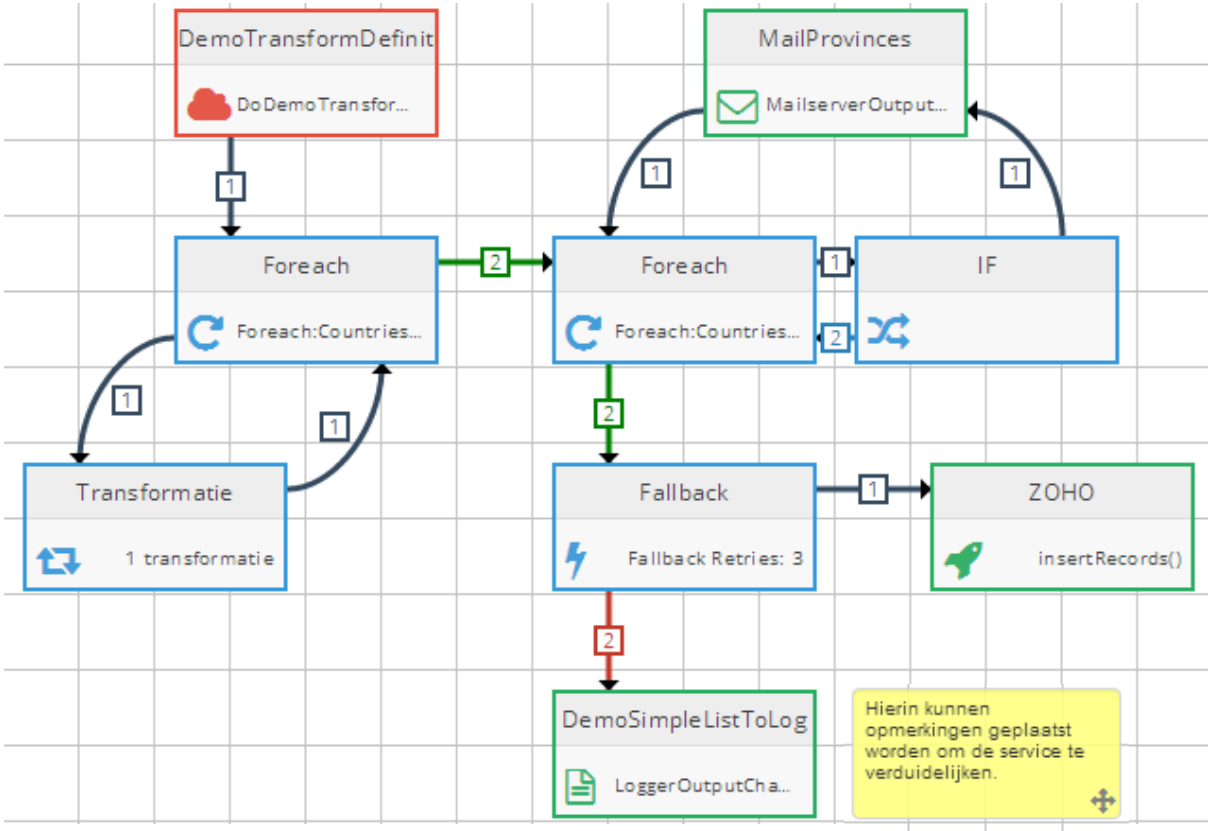


CONFIGURATIE SERVICE:

Oude situatie:



Nieuwe situatie:

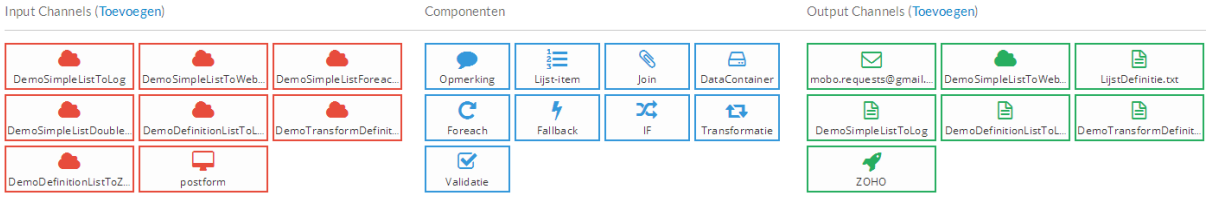


Opmerking: bovenstaande situaties zijn niet één op één te vergelijken, maar laten de verschillen in de mogelijkheden en de lay-out zien. In het gele component kunnen opmerkingen geplaatst worden.

OVERZICHT NODES:

Oude situatie:

Nieuwe situatie:



14.7. Bijlage 7 : CD

Bij dit document is een vertrouwelijke CD als bijlage toegevoegd. Op deze CD zijn de volgende bijlagen te vinden:

- CodeBijlage 1 – WSDL.cs
- CodeBijlage 2 – Mapping.cs
- CodeBijlage 3 – Outputconfiguratie.cs