DE HAAGSE HOGESCHOOL

infoSupport Solid Innovator

Afstudeerverslag

MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?

Door: Tom Keim

Titel	Afstudeerverslag	**Tweede Examinator**	Rianne Bechet-Tjoonk
Project	MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?	**Opdrachtgever**	Joop Snijder
Auteur	Tom Keim	**Procesbeleider**	Pascalle Hijl
Studentnummer	11031425	**Technisch begeleider**	Orhan Uyan
School	Haagse Hogeschool	**Datum**	02-10-2015
Eerste Examinator	Gerard Mijnarends	**Bedrijf**	Info Support B.V.

Historie				
Versie	Status	Datum	Auteur	Verandering
0.1	Concept	29-06-2015	Tom Keim	Creatie
0.2	Concept	06-07-2015	Tom Keim	Verwerking feedback Joop Snijder (opdrachtgever). Huisstijl aangepast
0.3	Concept	01-08-2015	Tom Keim	Verwerking feedback Gerard Mijnarends (Haagse Hogeschool), Joop Snijder & Orhan Uyan, verkort door aanpassen schrijfstijl, uitbreiden met nieuwe sprints, evaluatie toegevoegd
0.4	Concept	24-08-2015	Tom Keim	Feedback verwerkt.
0.5	Concept	25-08-2015	Tom Keim	Feedback school deels verwerkt
0.6	Concept	02-09-2015	Tom Keim	Feedback verwerkt, Sprint 8 toegevoegd
0.7	Concept	07-09-2015	Tom Keim	Verwerken feedback Willem Meints
0.8	Concept	20-09-2015	Tom Keim	Verwerken feedback Joop Snijder.
0.9	Concept	25-09-2015	Tom Keim	Verklarende woordenlijst toegevoegd.
1.0	Definitief	01-10-2015	Tom Keim	Definitieve versie

Distributie			
Versie	Status	Datum	Aan
0.1	Concept (60% versie)	05-07-2015	Joop Snijder, Gerard Mijnarends (Haagse Hogeschool)
0.2	Concept	07-07-2015	Joop Snijder, Orhan Uyan
0.3	Concept	21-08-2015	Joop Snijder, Orhan Uyan
0.4	Concept (80% Tussentijdse beoordeling)	25-08-2015	De Haagse Hogeschool (Gerard Mijnarends & Rianne Bechet-Tjoonk)
0.5	Concept	01-09-2015	Orhan Uyan
0.6	Concept	04-09-2015	Willem Meints, Orhan Uyan, Joop Snijder
0.7	Concept	15-09-2015	Joop Snijder, Orhan Uyan, Pascalle Hijl
0.8	Concept	25-09-2015	Joop Snijder
1.0	Definitief	01-10-2015	De Haagse Hogeschool (Gerard Mijnarends & Rianne Bechet-Tjoonk), Joop Snijder

Referaat

Keim T.M., Afstudeerverslag – MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?, Veenendaal, Info Support BV, 02-10-2015.

Dit afstudeerverslag is geschreven in het kader van de afstudeerstage voor de opleiding Informatica aan de Haagse Hogeschool te Den Haag. De auteur schrijft in dit verslag over zijn werkzaamheden tijdens het afstuderen. Hierbij wordt op beschrijvende en evaluerende wijze ingegaan op de werkzaamheden die in de periode 28 april tot en met 2 oktober 2015 zijn uitgevoerd voor Info Support BV te Veenendaal.

Descriptoren

- MongoDB
- C#
- knowNow
- Onderzoek
- Proof-of-Concept
- Microservice-architectuur

Voorwoord

Mijn naam is Tom Keim, student HBO Informatica aan de Haagse Hogeschool te Den Haag. Voor u ligt het verslag dat ik in het kader van mijn afstudeerstage bij Info Support voor de Haagse Hogeschool heb geschreven. Dit verslag is bedoeld om Gerard Mijnarends, Rianne Bechet-Tjoonk en een nog onbekende externe gecommitteerde inzicht te geven in de activiteiten die tijdens mijn afstuderen hebben plaatsgevonden.

In dit voorwoord wil ik graag Joop Snijder bedanken voor de rol als opdrachtgever en de uitstekende begeleiding die hij tijdens het afstuderen heeft gegeven. Pascalle Hijl wil ik graag bedanken voor de procesbegeleiding en Orhan Uyan voor de technische begeleiding. Ook wil ik Willem Meints bedanken voor het overnemen van de rol als opdrachtgever en begeleider tijdens de vakantie van Joop Snijder.

Vanuit de opleiding wil ik Gerard Mijnarends en Rianne Bechet-Tjoonk bedanken voor tussentijdse feedback op mijn afstudeerverslag.

Tot slot wil ik Bart Bijl, Hanjo de Wit, Allard Soeters en Maikel Hofman bedanken voor de prettige werksfeer voor de dagen dat ik in Zoetermeer aanwezig was en het knowNow team voor de prettige werksfeer en de hulp op de dagen dat ik in Veenendaal aanwezig was.

Tom Keim
Pijnacker, 02-10-2015

Inhoudsopgave

Inleiding

Voor u ligt het afstudeerverslag van de tussen 28 april 2015 en 2 oktober 2015 uitgevoerde afstudeeropdracht “MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?”. Deze opdracht is in het kader van het afstuderen van de opleiding Informatica aan de Haagse Hogeschool te Den Haag uitgevoerd bij Info Support B.V.

Het doel van dit verslag is om Gerard Mijnarends, Rianne Bechet-Tjoonk en een nog onbekende externe gecommitteerde inzicht te geven in de activiteiten die tijdens mijn afstuderen hebben plaatsgevonden, om op deze manier de afstudeeropdracht te beoordelen.

In Hoofdstuk 1 ‘Organisatie’ wordt beschreven hoe Info Support er als organisatie uit ziet. In Hoofdstuk 2 ‘Opdrachtomschrijving’ wordt de uit te voeren opdracht naar voren gebracht. In dit hoofdstuk wordt de aanleiding, de probleemstelling, doelstelling en het beoogde resultaat van de opdracht beschreven. Hoofdstuk 3 ‘Methoden en Technieken’ beschrijft de keuze voor Scrum en beschrijft enkele termen van MongoDB. In Hoofdstuk 4 ‘Huidige Situatie’ wordt de huidige situatie van knowNow beschreven. In Hoofdstuk 5 ‘Producten & Planning’ worden de op te leveren producten en de planning beschreven.

Hoofdstuk 6 ‘Sprints’ beschrijft de uitvoering van het project. De uitvoering van het project is opgedeeld in een opstartfase en 8 sprints. In elk van deze sprints worden de keuzes en de producten uit die sprint beschreven. Elke sprint heeft een titel waarmee duidelijk wordt waar er in die sprints aan is gewerkt. In Hoofdstuk 7 ‘Conclusie & aanbevelingen’ wordt de conclusie van het onderzoek, de aanbeveling die de afstudeerder doet na de uitvoering van het project en het ontwikkelen van het Proof-of-Concept beschreven. In hoofdstuk 8 ‘Evaluatie’ wordt het afstudeerproject geëvalueerd. Er wordt gekeken naar het proces, de opgeleverde producten en de te behalen beroepstaken.

In ‘Referenties’ zijn diverse referenties opgenomen. Wanneer in het verslag bij een tekst ‘[…]’ met op de plek van de puntjes een nummer staat, wordt hier verwezen naar een referentie in dit hoofdstuk met het bijhorende nummer.

1. Organisatie

Info Support is een succesvol en winstgevend IT bedrijf met meer dan 400 medewerkers in dienst waarvan het hoofdkantoor gevestigd is in Veenendaal. Het ontwikkelt, beheert en host software op maat voor een groot aantal klanten. Het bedrijf, opgericht in 1986, kiest vooral voor Microsoft (.NET, C#, SQL Server, Azure), Oracle en Java technologie. Info Support heeft een eigen kenniscentrum waar meerdere docenten de medewerkers van het bedrijf trainingen geven over diverse branche- en IT gerelateerde onderwerpen. Hierdoor blijft het bedrijf innoveren en past het, waar mogelijk, de nieuwste technologieën toe.

In het logo van Info Support is het motto “Solid Innovator” te vinden. Dit motto geeft weer waar het bedrijf voor staat. Soliditeit door het streven naar kwalitatief hoogwaardige oplossingen en Innovatie door het toepassen van nieuwe technologieën in de snel veranderde wereld van de ICT.

1.1 Klantprofiel

Info Support kent vele opdrachtgevers in diverse sectoren. Veel van deze opdrachtgevers behoren tot de grotere bedrijven in Nederland. Er wordt onder andere gewerkt in de sectoren overheid, handel & industrie en zorg & verzekeringen.

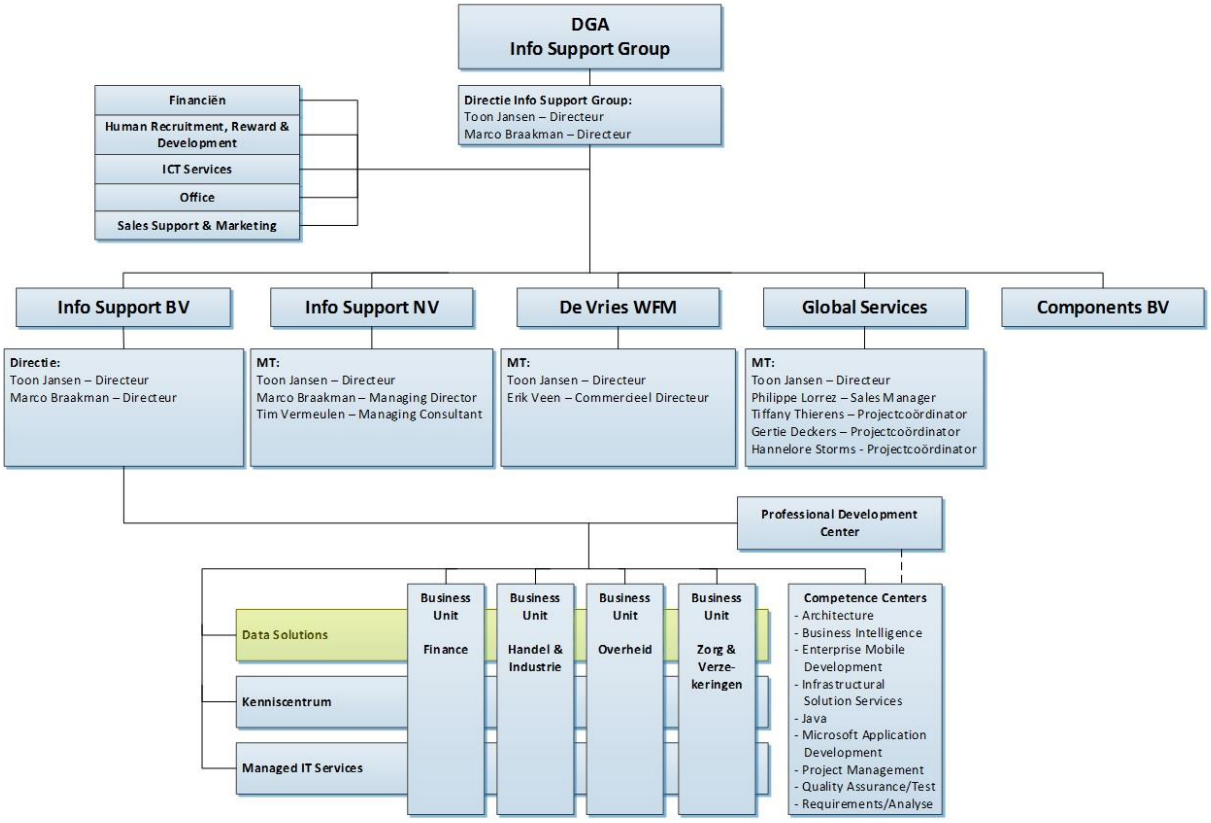
In Figuur 1-1 is een overzicht te zien van enkele van de klanten van Info Support.

Figuur 1-1. Enkele klanten van Info Support[17]

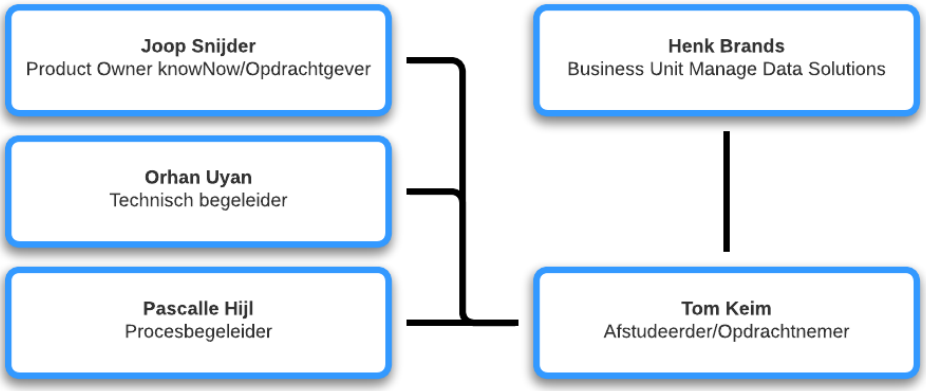
1.2 Organogram

Info Support is onder verdeeld in verschillende units zoals Finance, Handel & Industrie, Overheid en Zorg & Verzekeringen die elk hun eigen klanten bedienen. Elke Unit heeft een business unit manager.

Ik maak onderdeel uit van de Data Solutions Unit. De business unit manager van deze unit is Henk Brands. In het Figuur 1-3 zijn de opdrachtgever, Business Unit Manager en de begeleiders van Info Support weergeven. Figuur 1-2 is een organogram van de gehele organisatie.



Figuur 1-2. Organogram van Info Support. Ik val tijdens mijn afstuderen onder de unit Data Solutions (geel gemarkeerd). Bron: Info Support



Figuur 1-3. De relaties tussen de afstudeerder en de verschillende begeleiders Info Support

Naam	Rol	Omschrijving rol
Tom Keim	Afstudeerder / Opdrachtnemer	Voert de opdracht uit
Joop Snijder	Product Owner knowNow / Opdrachtgever	Is de opdrachtgever voor deze opdracht en is tevens de Product Owner van knowNow.
Orhan Uyan	Technisch begeleider	De technisch begeleider zorgt voor hulp bij het technische en inhoudelijke deel van de opdracht
Pascalle Hijl	Procesbegeleider	De procesbegeleider ondersteund de afstudeerder tijdens zijn afstudeerstage en houdt de voortgang in de gaten.
Henk Brands	Business Unit Manager	De business unit manager is de manager van de business unit en houdt de voortgang in de gaten.
Willem Meints	Opdrachtgever tijdens afwezigheid Joop Snijder	Willem Meints is de opdrachtgever van deze opdracht geweest tijdens de afwezigheid van Joop Snijder wegens vakantie.

2. Opdrachtomschrijving

2.1 Aanleiding

Inleiding
Info Support had tot 3 jaar geleden een applicatie waarmee kennis binnen het bedrijf gedeeld kon worden. Deze applicatie, genaamd Digital Coach, was in feite niets anders dan een webpagina onderhouden door het Professional Development Center (PDC) van Info Support.

Digital Coach was slecht doorzoekbaar waardoor gebruikers moesten weten waar informatie in de applicatie was opgeslagen om deze te vinden. Projectteams konden zelf geen informatie aan Digital Coach toevoegen, dit moesten zij doen via het PDC.

Deze redenen zorgden ervoor dat Digital Coach intern nauwelijks werd gebruikt, waardoor kennis binnen projectteams verloren ging. Het wiel moest telkens opnieuw worden uitgevonden terwijl iemand binnen de organisatie misschien het antwoord op de kennisvraag al had uitgezocht.

Om die redenen heeft Info Support 3 jaar geleden besloten, te starten met het ontwikkelen van het kennismanagementsysteem knowNow.

knowNow stemt het kennisaanbod binnen een organisatie af op het zoekgedrag van de gebruiker. De organisatie kan daardoor de productiviteit en kwaliteit van de informatie in het kennismanagement-systeem verhogen. In knowNow is het onder andere mogelijk om kennis te delen (in de vorm van artikelen) en op te geven welke ervaringen/skills een persoon heeft.

Verwachte groei
Op het moment van schrijven zijn er 7000 gebruikers van knowNow. De gebruikers wordt verdeeld over 2 frontend servers, die dezelfde databases delen.

Het knowNow team is op dit moment bezig met een klant met potentieel 200.000 gebruikers. Als er voor 7000 gebruikers al 2 frontend servers nodig zijn, zullen er voor 200.000 gebruikers ongeveer 57 frontend servers nodig zijn. Dit vormt naast het beheren van alle frontend servers voor het kleine DevOps team, waarschijnlijk een zware belasting voor de gedeelde database.

Binnen 5 jaar wil knowNow doorgroeien naar 3 miljoen gebruikers. Hierdoor wordt de beheerlast te groot, omdat het knowNow team dan tientallen servers moet gaan beheren.

Huidige database
knowNow werkt op dit moment met een relationele database waarbij voor het opvragen van één Knowledge Item 50 à 60 tabellen worden aangesproken. Dit gebeurt door middel van tientallen verschillende query's. Dit aantal is hoog door de manier waarop het opgeslagen is in de relationele SQL database. Informatie over de personen staat in de tabel personen, reacties staan in de tabel reacties, tags staan bij de tags etc. Maar ook die tabellen zijn weer apart gesplitst, zo is er een aparte tabel voor de titel en tekst. Hierdoor kunnen alle onderdelen van een item apart worden in- en uitgeschakeld. Het nadeel is echter dat alle data voor één document niet in een tabel staat maar verspreid over vele tabellen die eerst gecombineerd moeten worden. Het knowNow team verwacht dat dit een bottleneck gaat vormen bij een groei van het aantal gebruikers.

NoSQL
Het knowNow team heeft gekeken naar een mogelijke vervanging voor de relationele database. Hierin is onderzoek gedaan naar Cassandra en MongoDB. Uit dit onderzoek is gebleken dat de NoSQL document-storage database MongoDB de beste optie voor knowNow lijkt. De keuze voor MongoDB staat vast. Om deze reden wordt geen onderzoek gedaan naar andere NoSQL databases.

2.2 Probleemstelling

Het knowNow team wil voor de Knowledge Items gebruik gaan maken van MongoDB. Maar of MongoDB ook echt een alternatief is voor SQL server in de opslag van Knowledge Items is onbekend. MongoDB vraagt een andere manier van data modellering. Of MongoDB voor knowNow inzetbaar is, welke impact het gebruik van MongoDB heeft op het technisch ontwerp, het ophalen van de data, de uitbreidbaarheid van het datamodel en schaalbaarheid van knowNow heeft, is niet bekend en moet worden onderzocht.

2.3 Doelstelling

De opdrachtgever wil dat onderzocht wordt wat de consequenties zijn van het gebruik van MongoDB voor de Knowledge Items in plaats van een RDBMS. De opdrachtgever wil weten wat de impact is van MongoDB op het technisch ontwerp, de uitbreidbaarheid van de datamodellen en (indien tijd over) de schaalbaarheid. De opdrachtgever wil kortom weten of MongoDB voor knowNow een alternatief is voor SQL Server.

2.4 Resultaat

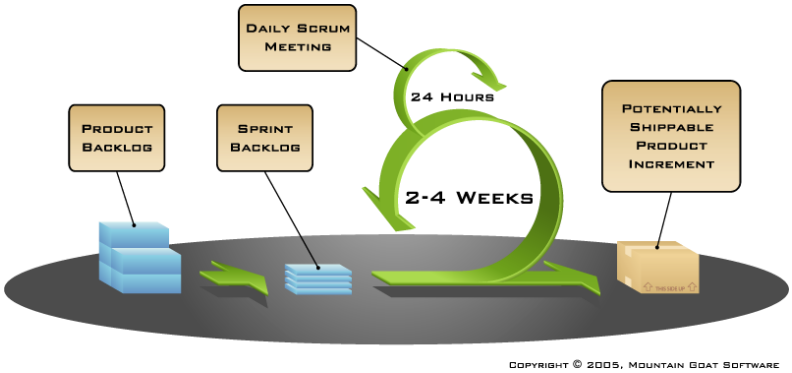
Na het uitvoeren van de afstudeeropdracht zijn er conclusies geschreven over de onderzoeken waarmee kan worden bepaald of MongoDB voor knowNow een alternatief is voor een RDBMS. Een werkend Proof-of-Concept is opgeleverd waarin MongoDB wordt gebruikt. Dit Proof-of-Concept bevat het technisch ontwerp en een ontwikkelde applicatie waarbij de Knowledge Items worden opgeslagen in MongoDB. Op basis van het Proof-of-Concept en het onderzoek wordt een advies gegeven of MongoDB een alternatief is voor knowNow en ook of het een beter alternatief is voor SQL Server.

3. Methoden en Technieken

3.1 Ontwikkelmethode

3.1.1 Inleiding

Gedurende het afstuderen is gebruik gemaakt van de methode Scrum. Scrum is een ontwikkelmethode waarmee het mogelijk is goed om te gaan met veranderingen waarbij de focus ligt op de meerwaarde voor de klant. Er wordt iteratief aan het project gewerkt, aan het eind van elke sprint kan feedback ontvangen worden en kan het project worden bijgestuurd[4].



Figuur 3-1. Een voorbeeld van een Scrum Project
(bron: http://epf.eclipse.org/wikis/scrum/)

Scrum is een ontwikkelmethode die kan worden gebruikt in ontwikkelteams, echter wordt deze afstudeeropdracht uitgevoerd door één persoon. Daarom worden enkele onderdelen van Scrum aangepast zodat het beter aansluit bij het project.

3.1.2 Gebruikte Scrum variant

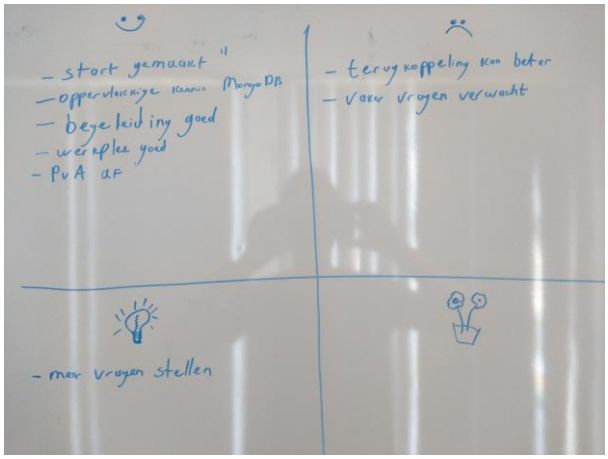
Sprintlengte
Een Scrum-project is opgedeeld in verschillende sprints. Het afstudeerproject is opgedeeld in 8 sprints van elk 2 weken. Voor deze lengte is gekozen zodat er voldoende feedbackmomenten zijn.

Daily Standup
De daily stand-ups vinden alleen plaats op dinsdag, woensdag en vrijdag met alle aanwezigen van het knowNow team. De daily standup vindt niet dagelijks plaats omdat ik op maandag en donderdag vanuit Zoetermeer werkt en niet aanwezig ben in Veenendaal, waar het knowNow team werkt. Gedurende de daily stand-up kan ik duidelijk maken waar ik mee bezig ben, tegen aangelopen ben en kan ik hulp vragen wanneer ik vast zit. Op de dagen in Zoetermeer zijn er geen belanghebbenden van het project aanwezig waardoor het houden van een stand-up geen nut heeft.

Sprint Review + Sprint Retrospective + Sprint Planning
De sprint review, sprint retrospective en sprint planning worden gecombineerd aan het eind van de Sprint gehouden.

Tijdens een sprint review worden de opgeleverde producten besproken met de Product Owner en geeft deze zijn bevindingen aan.

Tijdens de sprint retrospective geven de Product Owner en ik de goede punten, de slechte punten en de ideeën aan voor de volgende Sprint. Er kunnen complimenten worden gegeven aan personen die te maken hebben met de afstudeeropdracht. Dit wordt gedaan op een whiteboard waar beide partijen kunnen aangeven of ze het eens zijn met de opgeschreven punten.



Figuur 3-2. Een voorbeeld van de bevindingen tijdens een Sprint Retrospective. **Linksboven: de positieve punten, rechtsboven: de minder goede punten. Linksonder: de ideeën voor volgende sprint(s). Rechtsonder: Complimenten voor belanghebbenden afstudeeropdracht**

Tijdens de Sprint Planning wordt de planning voor de volgende sprint besproken. De planning wordt voorbereid door mij. De planning kan daarna worden bijgewerkt op basis van de bevindingen uit de Sprint Review of Sprint Retrospective en invloeden van de Product Owner.

Definition of Done: Het Definition of Done wordt door mij gemaakt na de Sprint Planning. Het Definition of Done bevat alle producten die aan het eind van de sprint opgeleverd moeten worden. Alle producten in een Definition of Done moeten Shippable zijn. Dit houdt in dat ze aan de opdrachtgever opgeleverd moeten kunnen worden en de opdrachtgever tevreden moet zijn met het resultaat.

Scrum master: Voor de afstudeeropdracht is er geen Scrum master. De taken van de Scrum master worden uitgevoerd door de opdrachtnemer. De scrum master organiseert onder andere de review, retrospective en planning.

Onderzoek: De Scrum aanpak van het onderzoek is in paragraaf 3.1.3 beschreven.

Product Backlog: In het Product Backlog worden alle Backlog items bijgehouden die nog gedaan moeten worden. Backlog items zijn de opleverbare producten.

Sprint Backlog: Elke sprint heeft een eigen Sprint Backlog. In een Sprint Backlog staan alle items waar in die sprint aan gewerkt wordt.

Product Owner: De klant waar het product voor wordt gemaakt. In project is dit de opdrachtgever, Joop Snijder.

3.1.3 Onderzoeken met Scrum

Scrum is een ontwikkelmethode. Toch is er gekozen om Scrum ook te gebruiken voor het onderzoek. De opdrachtgever kan hierdoor regelmatig feedback geven op het onderzoeksdocument. Daarnaast kan het onderzoek per deelvraag worden opgesplitst en zo toch in verschillende sprints worden ingedeeld. De mogelijkheid bestaat dat de opdrachtgever tijdens het onderzoek ook andere aspecten onderzocht wil hebben.

Een groot gedeelte van het afstuderen is besteed aan het onderzoek naar MongoDB voor knowNow (Bijlage B). Vooral de eerste drie Sprints staan in het teken van het onderzoek. In het Plan van Aanpak (Bijlage A) zijn de hoofdvraag en de deelvragen voor het onderzoek met de opdrachtgever opgesteld.

Aan het begin van het project is een grovere opzet van het onderzoek gemaakt, te vinden in het Plan van Aanpak (Bijlage A). In elke Sprint wordt dit onderzoek wanneer nodig verder uitgewerkt. Hierdoor is het mogelijk in te spelen op nieuwe inzichten die opgedaan worden tijdens het uitvoeren van het project.

Tijdens de Sprint Planning wordt met de opdrachtgever besproken welke deelvragen in de Sprint onderzocht gaan worden en hoe dit bereikt wordt. Hierdoor is het ook mogelijk de deelvragen anders te formuleren dan in het oorspronkelijke Plan van Aanpak beschreven is. In het Plan van Aanpak is een globale planning opgezet, waarin is aangeven welke vragen onderzocht worden in een sprint. Hierdoor is er toch duidelijk wat er in het onderzoek moet gebeuren.

Door de incrementele en iteratieve aanpak van de korte Sprints kan snel worden ingespeeld op bevindingen in het onderzoek, nieuwe inzichten en eventuele wensen en feedback van de opdrachtgever.

Aan het einde van elke sprint dienen er volledige producten opgeleverd te worden. Hier valt het onderzoek ook onder. Er wordt getracht om deelvragen zoveel mogelijk binnen één Sprint in te passen, echter is dat soms niet haalbaar. In dat geval wordt er van te voren afgesproken om een logboek op te leveren waarin de bevindingen tijdens het beantwoorden van de deelvraag te vinden zijn.

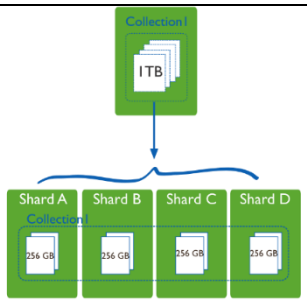
3.1.4 Andere methodes

Kanban
Het knowNow team maakt gebruik van de methode Kanban. Kanban is een methode die Just-In-Time delivery ondersteunt. Bij Just-In-Time worden producten pas gemaakt wanneer ze ook echt nodig zijn[15]. Met Kanban wordt het Work in Progress gelimiteerd. Hiermee wordt voorkomen dat het team aan te veel producten tegelijkertijd werkt. Wanneer er meer taken bij komen moeten eerst de oude taken worden afgerond voordat er aan de nieuwe taak mag worden begonnen. Bij Kanban kunnen er continue nieuwe taken worden toegevoegd, terwijl dit bij Scrum in de Sprint Planning per Sprint wordt vastgezet.

Kanban is vooral handig is bij continuous delivery, het continue kunnen uitbrengen van nieuwe versies[14]. Kanban heeft namelijk een continue stroom van werk.
Er is toch voor Scrum gekozen omdat bij Scrum het werk per Sprint wordt vastgesteld. Hierdoor is het voor mij duidelijker wat er nog moet gebeuren. Daarnaast heb ik al ervaring met Scrum en niet met Kanban, waardoor er sneller kan worden gestart met het uitvoeren van het project
RUP
RUP is net zoals Scrum een softwareontwikkelmethode. Voor Scrum is gekozen omdat bij Scrum beter ingespeeld kan worden op bevindingen tijdens het uitvoeren van het project door na een Sprint wanneer nodig het Product Backlog aan te passen. RUP heeft een End-to-end planning waardoor hier nieuwe inzichten van de opdrachtgever tijdens het uitvoeren van het project minder goed opgepakt kunnen worden[19].

3.2 MongoDB

MongoDB is een Open-Source NoSQL document storage database. MongoDB is in 2009 voor het eerst uitgebracht en wordt inmiddels door diverse grote organisaties gebruikt[1]. MongoDB hanteert geen schema zoals in een RDBMS gebruikelijk is en documenten worden in BSON (Binair JSON[2] opgeslagen). In de tabel worden enkele begrippen toegelicht.

Collectie	Een collectie in MongoDB is vergelijkbaar met een tabel in een RDBMS[3]. Een collectie bevat *documenten*.
Document	Een document is vergelijkbaar met een rij in een RDBMS en valt onder een *collectie*. MongoDB kent geen vast schema per *collectie*. Elk document in een *collectie* kan een ander schema hebben. Een document heeft *velden*. Een document is opgeslagen in *BSON* formaat.
Veld	Een veld is vergelijkbaar met een kolom in een RDBMS. Velden zitten in een *document*. Elk document kan andere velden hebben omdat MongoDB geen vast schema hanteert.
Subdocument	Een document kan bestaan uit subdocumenten. Deze documenten kunnen ook weer hun eigen velden hebben. Elk subdocument kan weer een subdocument hebben. Dit kan tot maximaal 100 niveaus diep[11].
BSON	MongoDB slaat documenten op in BSON formaat. Dit is een binaire vorm van JSON.
Sharding	Sharding is het horizontaal schalen van een MongoDB database. Bij Sharding worden nieuwe MongoDB instanties toegevoegd en wordt de data verdeeld over meerdere instanties. Dit verdelen gaat met behulp van *Sharding Keys*.
Sharding Key	Een Sharding Key is een *veld* die elk *document* in een *gesharde collectie* bevat. Aan de hand van een sharding key wordt de data over de verschillende *shards* verdeeld.
Shard	Een shard is een MongoDB instantie in een *Sharded Cluster*. Elke shard is verantwoordelijk voor een deel van de data van een collectie.
Sharded Cluster	Een Sharded Cluster is een gesharde MongoDB database. Een Sharded Cluster bevat meerdere *shards*. 
Joins	MongoDB kent geen Joins zoals in een RDBMS vaak gebruikelijk is. Het is niet mogelijk om MongoDB data te laten combineren bij het ophalen.
ObjectId	Het ObjectId is de unieke code van een document in MongoDB. Elk *document* moet een _id *veld* hebben met een ObjectId. Een ObjectId kan er als volgt uit zien: **55ffc9128b608cd5211043a6**

1 https://www.mongodb.com/who-uses-mongodb

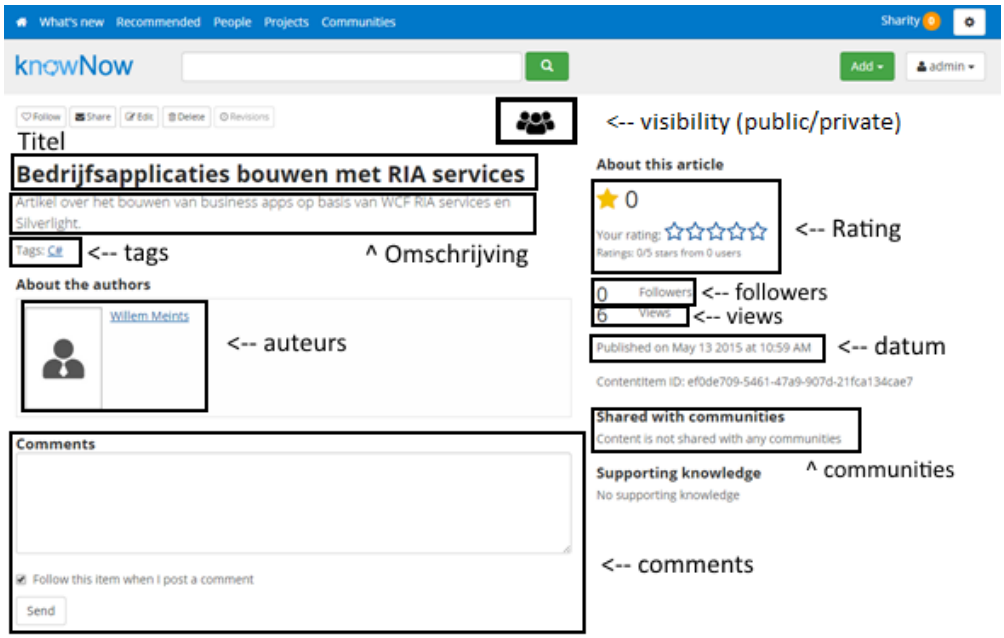
2 JSON, JavaScript Object Notation is een gegevensformaat waarbij data in een leesbaar formaat wordt opgeslagen. http://json.org/

3 RDBMS staat voor Relationeel Database Management Systeem. Een bekend RDBMS is Microsoft SQL Server[12].

4. Huidige Situatie

knowNow is een kennismanagementsysteem dat het kennisaanbod binnen een organisatie afstemt op het leesgedrag van de gebruiker. De organisatie kan daardoor de productiviteit en kwaliteit van de informatie in het kennismanagementsysteem verhogen. In knowNow is het onder andere mogelijk om kennis te delen (in de vorm van artikelen) en op te geven welke ervaringen/skills een persoon heeft.

Een Knowledge Item is een onderdeel van de knowNow applicatie. Het is een samenstelling van allerlei attributen die op een Knowledge Pagina staan. Onder een Knowledge Item vallen onder andere een titel van het document, omschrijving en tags.

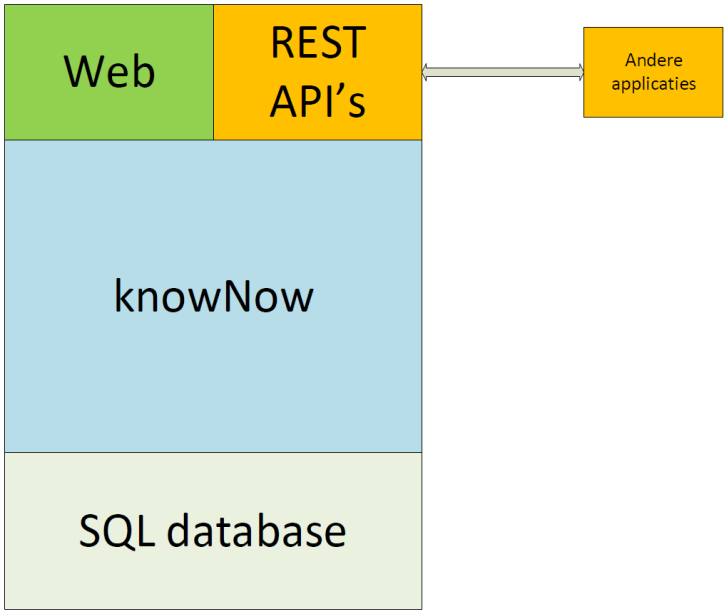


Figuur 4-1. Een Knowledge Item pagina

knowNow is een op Orchard ontwikkelde applicatie. Orchard is een open-source Content Management Systeem (CMS)[5].

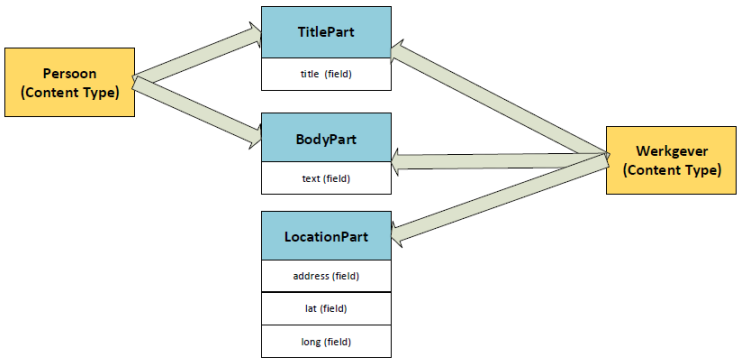
De Orchard installatie maakt gebruik van Microsoft SQL Server als RDBMS. Voor knowNow zijn specifieke uitbreidingen (modules) op Orchard gemaakt. Dit zijn uitbreidingen om extra functionaliteiten aan Orchard toe te voegen. Zo is er ook een module voor een JSON REST API.

In de bijlage “Technisch Ontwerp oorspronkelijke situatie knowNow” (bijlage C)” wordt verder ingegaan op de huidige situatie en wordt onder andere Orchard en de inhoud van een Knowledge Item toegelicht.



Figuur 4-2. De huidige knowNow applicatie

Voor het opvragen van één Knowledge Item worden 50 à 60 tabellen aangesproken. Dit gebeurt door middel van 90 verschillende query's. Dit aantal is hoog door de manier waarop de data is opgeslagen in de relationele SQL database. Informatie over de personen staat in de tabel personen, reacties staan in de tabel reacties, tags staan bij de tags etc. Maar ook die tabellen zijn weer apart gesplitst, zo is er een aparte tabel voor de titel en tekst. Hierdoor kunnen alle onderdelen van een item in Ochard apart worden in- en uitgeschakeld. Het nadeel is echter dat alle data voor één document niet in een tabel staat maar verspreid over vele tabellen die eerst gecombineerd moeten worden. Het knowNow team verwacht dat dit een bottleneck gaat vormen bij een groei van het aantal gebruikers en zoekt manieren om het aantal query's terug te brengen.



Figuur 4-3. Een voorbeeld van hoe de data wordt opgeslagen in Orchard. Een entiteit (bijvoorbeeld persoon of werkgever) bestaat weer uit diverse onderdelen (titel, body). Elk van deze onderdelen (parts) is een eigen tabel.

In Bijlage C, Technisch ontwerp van oorspronkelijke situatie knowNow, paragraaf 5.3 wordt dieper ingegaan op de werking van het database ontwerp van knowNow.

5. Producten & Planning

Dit hoofdstuk beschrijft de onderdelen en de planning van de afstudeeropdracht zoals deze aan het begin van de afstudeeropdracht is opgezet. De afstudeeropdracht bestaat uit 3 onderdelen:

- Het Onderzoek
- Het Proof of Concept
- Het Adviesrapport

5.1 Onderzoek

Het onderzoek geeft antwoord op de hoofdvraag:

"Is MongoDB een alternatief voor SQL Server voor de Knowledge Items van knowNow?"

De hoofdvragen zijn opgedeeld in verschillende deelvragen. Deze deelvragen zijn afgeleid van vragen van de opdrachtgever. De oorspronkelijke deelvragen, zoals in week 1 opgesteld, waren als volgt:

Deelvraag A: Welke invloed heeft MongoDB op het technisch ontwerp van knowNow? **(deskresearch)**
Deelvraag B: Welke invloed heeft MongoDB op de gebruikte interface (REST endpoints[4]), is het mogelijk iets op één manier op te slaan wanneer er meerdere REST endpoints zijn? **(deskresearch & experimenteren)**
Deelvraag C: Wat is de uitbreidbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB, kunnen er nieuwe REST endpoints worden toegevoegd? **(deskresearch & experimenteren)**
Deelvraag D: Wat is de schaalbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB, hoe kan MongoDB worden opgeschaald wanneer het aantal gebruikers toeneemt? **(deskresearch & experimenteren)**
Deelvraag E: Wat is de invloed op performance bij gebruik van MongoDB tegenover een relationele database? **(experimenteren)**
Deelvraag F: Hoe kan de MongoDB database gevuld worden en in sync worden gehouden met een Relationele Database? **(deskresearch & experimenteren)**

[4] Een REST API is een communicatiemiddel waarbij gebruik wordt gemaakt het HTTP protocol[7]. De data wordt in JSON (JavaScript Object Notation)[8] uitgewisseld.

5.2 Proof-of-Concept

Het Proof-of-Concept heeft als doel om aan te tonen dat MongoDB voor knowNow een werkend alternatief is voor een RDBMS.

Samen met de opdrachtgever zijn requirements opgesteld waaraan moet worden voldaan wil MongoDB inzetbaar zijn voor knowNow. De functionaliteiten zijn gebaseerd op de huidige functionaliteit van knowNow. Er is gekozen voor lees, schrijf, wijzig en verwijder functionaliteit omdat MongoDB mogelijk direct invloed heeft op de werking van deze functionaliteiten. Wanneer blijkt dat deze functionaliteiten niet werken met MongoDB is het niet inzetbaar voor knowNow.

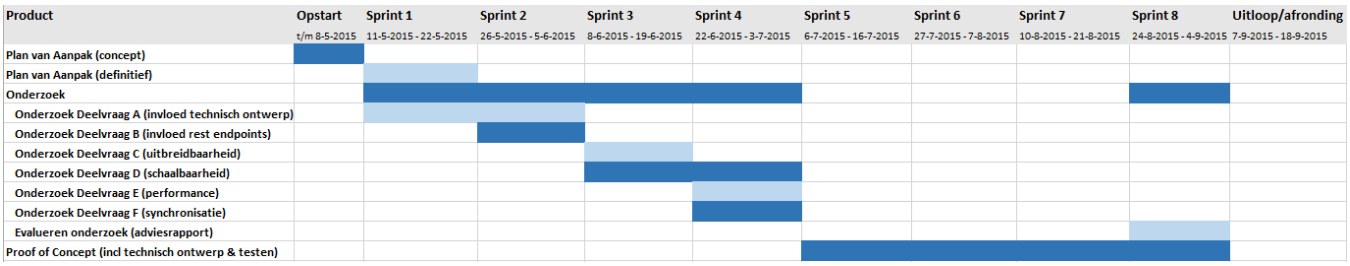
In hoofdstuk 6 wordt het proces van de totstandkoming van het Proof-of-Concept besproken.

5.3 Adviesrapport

In het adviesrapport doe ik een aanbeveling aan het knowNow team of ik zelf MongoDB zal inzetten en waar het team op moet letten bij het gebruik van MongoDB.

5.4 Originele planning

De originele planning, opgesteld in week 1 van de afstudeeropdracht was als volgt:



De diverse deelvragen zijn al in paragraaf 5.1 benoemd.

6. Sprints

6.1 Inleiding

Dit hoofdstuk beschrijft de producten waaraan per sprint is gewerkt en welke keuzes gedurende de sprint zijn gemaakt. Het project is onderverdeeld in een opstartfase, 8 sprints, en een uitloop fase.

6.2 Opstart

Duur	2 weken 28-04-2015 t/m 08-05-2015
Producten	• Plan van Aanpak

In de opstartfase van 2 weken is kennis gemaakt met het bedrijf Info Support en alle personen die te maken hebben met mijn afstuderen. Er zijn overleggen geweest met de opdrachtgever waarna het Plan van Aanpak opgezet kon worden. In dit Plan van Aanpak is de Aanleiding, Doelstelling en Aanpak van de afstudeeropdracht besproken.

De planning die in het oorspronkelijke Plan van Aanpak is opgesteld is beschreven in hoofdstuk 6.3.

De verschillende deelvragen van het onderzoek zijn verdeeld over de verschillende Sprints.

Ook al wordt er met Scrum gewerkt, het hebben van een valide planning is belangrijk. Wanneer er een te ambitieuze planning wordt opgezet is het mogelijk dat niet alle onderdelen afkomen. Om toch tot een zo valide mogelijke planning te komen is het onderzoek in de planning gesplitst per deelvraag. Per deelvraag is er bekeken hoeveel tijd er in kan gaan zitten. Hierdoor is het onderzoek verdeeld in kleinere delen waardoor de totaal benodigde tijd beter in te schatten is.

Wanneer er alsnog te weinig tijd voor een deelvraag is ingepland en deze dus niet afkomt aan het einde van een Sprint kan besloten worden hier verder aan te werken in de Sprint daarna. Wanneer hierdoor niet aan andere onderdelen in het project gewerkt kan worden, kan de ingeplande uitloop hiervoor worden gebruikt.

6.3 Sprint 1: Onderzoek invloed Technisch Ontwerp

Duur	2 weken 11-05-2015 t/m 22-05-2015
Producten	• Onderzoek naar de invloed van MongoDB op het technisch ontwerp (Deelvraag A)

6.3.1 Proces

In Sprint 1 is gewerkt aan Deelvraag A: “Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?”.

Ik heb een Technisch Ontwerp van het oorspronkelijke knowNow opgezet, zodat voor mijzelf duidelijk wordt hoe het huidige systeem werkt. Hierdoor kan antwoord gegeven worden op de vraag ‘Wat is de invloed op een Technisch Ontwerp?’. Dit Technisch ontwerp is te vinden in Bijlage C en beschrijft de softwarearchitectuur en de database van de bestaande knowNow applicatie.

Door het schrijven van het Technisch Ontwerp heb ik diepgaande kennis opgedaan van de werking van de huidige knowNow applicatie, wat nuttig is geweest voor de verdere uitvoering van het afstudeerproject.

Voor het beantwoorden van Deelvraag A: “Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?” zijn 2 Sprints ingepland. Hierdoor moest de deelvraag worden opgesplitst. In Sprint 1 worden de bevindingen in een logboek verwerkt die ingeleverd kan worden aan de opdrachtgever. In Sprint 2 wordt van dit logboek een onderzoeksdocument gemaakt. Voor deze opzet is gekozen, zodat de opdrachtgever een duidelijk beeld van de voortgang krijgt, zonder dat er al direct een volledig onderzoeksdocument moet zijn.

Deelvraag A is opgesplitst in 4 subvragen. Alle beantwoorde subvragen zijn te vinden in het Onderzoeksrapport, Bijlage B, hoofdstuk 3.

1. Wat is de best te implementeren architectuur voor knowNow bij het gebruik van MongoDB?
2. Welke invloed heeft MongoDB op de uitbreidbaarheid van knowNow?
3. Welke invloed heeft MongoDB op de security van knowNow?
4. Welke invloed heeft MongoDB op de schaalbaarheid van knowNow?

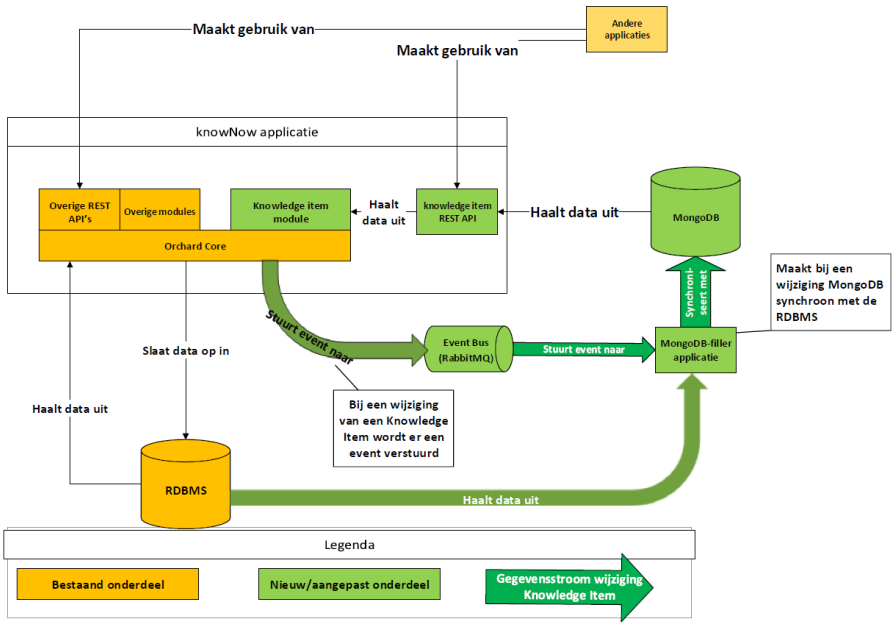
6.3.1.1 Wat is de best te implementeren architectuur voor knowNow bij het gebruik van MongoDB?

Bij deze subvraag van Deelvraag A wordt een keuze gemaakt voor een architectuur zodat de rest van het onderzoek hier op gebaseerd kon worden.

Eén van de wensen van de opdrachtgever is dat er verder ontwikkeld wordt op de bestaande applicatie die gebruik maakt van Orchard.

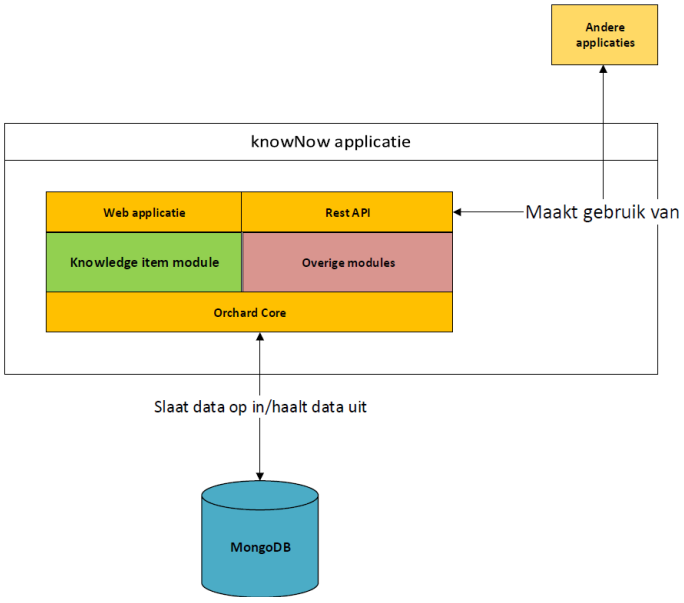
Door goed te kijken naar de huidige architectuur van knowNow en de wensen van de opdrachtgever in ogenschouw te nemen zijn er 2 mogelijke oplossingen geïnventariseerd. De 2 oplossingen zijn vergeleken op onder andere de hoeveelheid werk, onderhoudbaarheid, haalbaarheid en toekomstgerichtheid waardoor de best te implementeren architectuur bepaald kon worden. Op deze architectuur kan de rest van het onderzoek gebaseerd worden.

De eerste oplossing is het aanpassen van de knowNow Orchard applicatie om bij het lezen van items gebruik te maken van een MongoDB database, en bij het schrijven van items gebruik te maken van de RDBMS. De RDBMS wordt hierbij dus behouden. Bij deze oplossing worden alleen de delen aangepast waar data in MongoDB opgeslagen wordt en er worden enkele nieuwe onderdelen toegevoegd (groene blokken). De RDBMS en MongoDB database worden consistent gehouden door middel van een applicatie die aanpassingen op de RDBMS doorvoert in MongoDB. Deze oplossing is te zien in Figuur 6-1. De groene pijlen tonen de gegevensstroom die doorlopen wordt om MongoDB consistent te maken met de RDBMS. Na een wijziging van een Knowledge Item wordt een bericht verstuurd naar het onderdeel dat MongoDB consistent moet maken. Dit onderdeel haalt de aangepaste data uit het RDBMS en zorgt dat deze data ook wordt aangepast in MongoDB. Bij deze oplossing wordt alleen gelezen uit MongoDB waardoor er alleen mogelijke performancewinst kan worden gehaald bij het lezen. Door niet het hele systeem aan te hoeven passen is dit mogelijk een snelle manier om MongoDB in knowNow te implementeren.



Figuur 6-1. Mogelijke architectuur waarbij RBDMS naast MongoDB blijft bestaan. De groene pijlen tonen de gegevensstroom aan die wordt doorlopen om MongoDB consistent te maken met het RDBMS.

De andere oplossing bij het door ontwikkelen op de huidige applicatie is het geheel vervangen van de database door MongoDB. Bij deze oplossing moet zowel het lezen als het schrijven van de huidige applicatie aangepast worden. Dit betekent logischerwijs dat er veel extra aanpassingen nodig zijn. Deze oplossing is te zien in Figuur 6-2. Bij deze oplossing wordt er mogelijk winst behaald op zowel de lees- als schrijfperformance. Echter, vergt deze oplossing wel meer werk. Alle onderdelen moeten waarschijnlijk worden aangepast.



Figuur 6-2. Mogelijke architectuur waarbij de RDBMS geheel wordt vervangen door MongoDB en Orchard.NET vervangen wordt. Hierbij moet elk onderdeel waarschijnlijk worden aangepast om MongoDB te kunnen gebruiken.

In het onderzoek zijn de verschillende oplossingen bekeken en is er onder andere gekeken naar de hoeveelheid werk, onderhoudbaarheid, haalbaarheid en toekomstgerichtheid van de oplossing. Uit het onderzoek kwamen de opties en de bijhorende voor- en nadelen naar voren:

Architectuur	Voordelen	Nadelen
Inpassen in Orchard met het behouden van de RDBMS	• Alleen aanpassingen in Orchard voor het lezen. • Lezen van items mogelijk sneller door gebruik van MongoDB	• Orchard moet aangepast worden; • De 2 databases moeten consistent gehouden worden; • Het MongoDB documentschema moet geschikt zijn voor Orchard; • Orchard is mogelijk niet meer te updaten. Geen performancewinst voor schrijven van items
Inpassen in Orchard waarbij RDBMS geheel is vervangen	• Geen meerdere databases, waardoor beter onderhoudbaar. • Zowel lezen en schrijven van items mogelijk sneller door gebruik van MongoDB.	• Orchard moet aangepast worden; Mogelijk ook buiten modules om; Hierdoor is Orchard mogelijk niet meer te updaten, • Het MongoDB documentschema moet geschikt zijn voor Orchard

In Sprint 1 is door mij geadviseerd te kiezen voor het inpassen in Orchard, waarbij alle aanpassingen worden geschreven naar het al bestaande RDBMS en er wordt gelezen uit de nieuwe MongoDB database. Hiervoor is gekozen omdat dit minder aanpassingen vergt aan het systeem, het huidige Orchard systeem blijft behouden. Hierdoor kan het systeem sneller worden opgeleverd en kan er sneller feedback worden ontvangen van de klanten. Bij deze optie moet wel de RDBMS waar alle aanpassingen in plaatsvinden uiteindelijk consistent worden gemaakt met MongoDB.

6.3.1.2 Welke invloed heeft MongoDB op de uitbreidbaarheid van knowNow?

Er is in het onderzoek uitgevoerd in deze sprint geconcludeerd dat MongoDB theoretisch zelf erg flexibel is. Echter, was het binnen de tijd die voor het beantwoorden van de vraag stond niet mogelijk was om de uitbreidbaarheid van knowNow bij het gebruik van MongoDB aan te tonen. Omdat er andere deelvragen gepland waren omtrent de uitbreidbaarheid van knowNow met MongoDB is besloten het beantwoorden van deze deelvragen te combineren.

De al in de komende sprints ingeplande deelvragen zijn:

1. Deelvraag B: Welke invloed heeft MongoDB op de gebruikte interface (REST endpoints); is het mogelijk iets op één manier op te slaan wanneer er meerdere REST endpoints zijn?
2. Deelvraag C: Wat is de uitbreidbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB; kunnen er later nieuwe REST endpoints worden toegevoegd?

Deze deelvragen zijn gecombineerd tot een nieuwe deelvraag:

> "**Deelvraag 2:** Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?"

Het proces van deze deelvraag wordt in een van de komende sprints omschreven.

6.3.1.3 Welke invloed heeft MongoDB op de security van knowNow?

Om deze vraag te beantwoorden is deskresearch gedaan naar de security van MongoDB. Er is gekeken welke security aspecten MongoDB kent en of deze in te zetten zijn voor knowNow. Er kon worden geconcludeerd dat MongoDB eenvoudig te beveiligingen is. Specifiek voor knowNow moet er voor worden gezorgd dat alleen de knowNow applicatie en het knowNow team bij de applicatie kan door ervoor te zorgen dat de MongoDB server(s) niet van buiten zijn te benaderen en de gebruikers zich eerst moeten authentiseren voordat ze toegang krijgen tot MongoDB. Het is mogelijk om in MongoDB een gebruiker alleen toegang te geven over een of meerdere collecties (tabellen) in een database, maar daar wordt in de SQL Server configuratie die door knowNow wordt gebruikt, ook geen gebruik van gemaakt. Wanneer de gebruiker van de database bij de database kan, kan deze ook bij alle data. Geconcludeerd is dat MongoDB geen invloed heeft op de security van de knowNow applicatie, MongoDB is op de zelfde manier te beveiligen als nu door het knowNow team wordt gedaan voor SQL Server.

6.3.1.4 Welke invloed heeft MongoDB op de schaalbaarheid van knowNow?

Om deze vraag te beantwoorden is er een deskresearch gedaan naar de schaalbaarheid van MongoDB. Hieruit kwam naar voren dat MongoDB goede ondersteuning heeft voor schalen door 'Sharding[5]'. Het schalen van MongoDB is theoretisch ongelimiteerd[6]. Tijdens het onderzoek kwam ik erachter dat het antwoord op de vraag nog niet gegeven kon worden en niet op een bestaande applicatie worden getest. Doordat knowNow nog geen gebruik maakt van MongoDB kan deze vraag alleen theoretisch benaderd worden. Besloten is, indien er tijd over is, te kijken wat er aan servers nodig is om MongoDB te gebruiken voor knowNow. Dit wordt dan gedaan met het nog het te realiseren Proof-of-Concept. Hierdoor kunnen harde cijfers, voor bijvoorbeeld het aantal benodigde servers, worden gegeven in plaats van alleen een theoretische benadering.

6.3.2 Sprint Retrospective & Sprint Review

Aan het einde van elke sprint vindt een Sprint Retrospective plaats. Tijdens de Retrospective wordt de Sprint doorgenomen samen met de opdrachtgever. Tijdens de Sprint Retrospective worden ook, omdat deze is gecombineerd met een Sprint Review, de producten die zijn opgeleverd besproken.

Vlak voor de Sprint Retrospective, werd de gekozen architectuur besproken met de opdrachtgever en de Lead Architect van knowNow, Wiljag Denekamp. Gedurende dit gesprek werd duidelijk dat de wens van het doorontwikkelen op de huidige knowNow applicatie op een verkeerde aanname berustte. In de Sprint Retrospective is dit bijgesteld en zijn afspraken gemaakt ten einde herhaling te voorkomen. Samen met de opdrachtgever is een toekomstbeeld van knowNow omschreven waaraan de te kiezen architectuur zoveel mogelijk moet voldoen:

- Hoge schaalbaarheid: diverse onderdelen van de applicatie moeten apart geschaald kunnen worden.
- Duidelijke verantwoordelijkheden: elk onderdeel van de applicatie moet zijn eigen duidelijke verantwoordelijkheden hebben. Dit maakt de applicatie beter onderhoudbaar.
- Losse koppeling: diverse onderdelen moeten apart van elkaar te vervangen zijn, zonder dat hier aanpassingen nodig zijn aan andere onderdelen. Dit maakt de applicatie beter onderhoudbaar.

Omdat de wens van het doorontwikkelen op de al bestaande applicatie is komen te vervallen, vervalt die afbakening bij de keuze van de architectuur. Daarom is het onderzoek in Sprint 1 niet volledig. Het onderzoek naar een geschikte architectuur wordt in Sprint 2 opnieuw uitgevoerd, zonder de afbakening om zo tot een volledig onderzoek te komen.

[5] Sharding is het horizontaal schalen van een MongoDB database. Bij Sharding worden nieuwe MongoDB instanties (servers) toegevoegd en wordt data verdeeld over deze meerdere instanties. Dit verdelen gaat door middel van een *Sharding Key*.

6.4 Sprint 2: Onderzoek invloed Technisch Ontwerp & Uitbreidbaarheid

Duur	2 weken 26-05-2015 t/m 05-06-2015
Producten	Deelvraag 1: “Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?” Deelvraag 2: “Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?”

Naast het beantwoorden van Deelvraag 1 wordt er deze sprint aan Deelvraag 2 gewerkt. Deelvraag 2 is opgesplitst omdat deze te groot is voor 1 sprint. De deelvraag is in de volgende vragen opgesplitst:

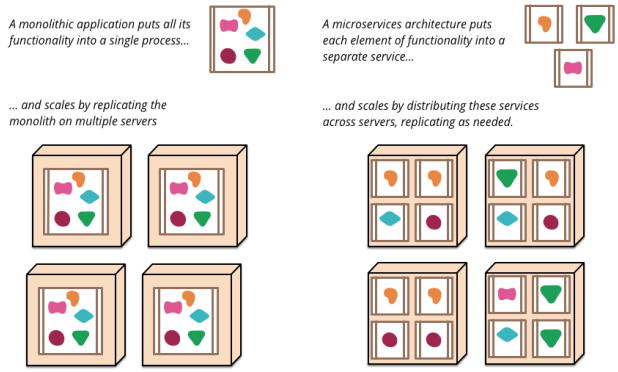
1. Wat zijn de consequenties van het uitbreiden van het MongoDB documentschema op de performance en schaalbaarheid?
2. Wat zijn de consequenties van het toevoegen van een nieuw REST endpoint op het documentschema en de performance? *(niet behandeld in dit document, maar is wel te vinden in het Onderzoeksrapport, Bijlage B, paragraaf 4.5)*
3. Is het mogelijk een MongoDB database te migreren door middel van gestapelde updates? *(behandeld in sprint 3)*

6.4.1.1 Deelvraag 1: Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?

De twee in Sprint 1 geïdentificeerde mogelijke architecturen zijn nog steeds mogelijk:

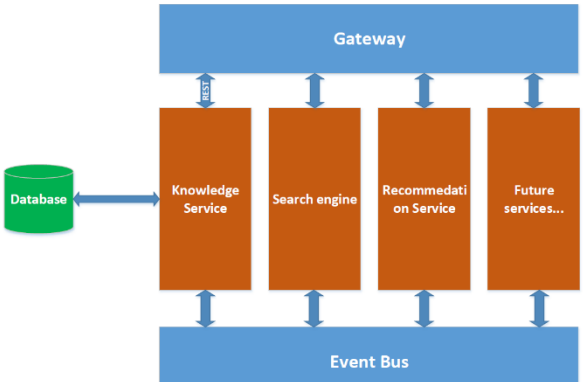
- het inpassen in Orchard met het behouden van de RDBMS;
- het inpassen in Orchard waarbij RDBMS geheel is vervangen, voor de architectuur.

Daarom is besloten ze wel in het onderzoek te laten staan. In het onderzoek zijn de verschillende oplossingen uitgewerkt en is er onder andere gekeken naar de hoeveelheid werk, onderhoudbaarheid, haalbaarheid en toekomstgerichtheid van de oplossing. Gekeken is naar een nieuwe toekomstgerichte architectuur. Dit bleek een Microservice-architectuur. Bij deze architectuur is een applicatie opgedeeld in meerdere kleine services die elk hun eigen duidelijke verantwoordelijkheid hebben[10]. Door het gebruik van een Microservice-architectuur kan elke service apart worden vervangen en worden geschaald, terwijl bij een monolithisch systeem het gehele systeem gedupliceerd wordt wanneer er geschaald moet worden. Hierdoor is een Microservice-architectuur erg schaalbaar.



Figuur 6-3. Links een Monolithisch systeem, zoals het huidige knowNow. Rechts een Microservice architectuur
(bron: http://martinfowler.com/articles/Microservices.html)

Onder andere vanwege de hoge schaalbaarheid is er door mij gekozen voor de Microservice-architectuur. Hierbij wordt de gehele applicatie vervangen. De andere oplossingen (die doorbouwen op de huidige applicatie) zijn minder toekomstgericht. Ze passen niet meer binnen het toekomstbeeld van het knowNow team en zijn na de aanpassingen mogelijk niet meer up te daten naar nieuwe versies van het Orchard CMS systeem. De opdrachtgever gaf aan in de toekomst naar de Microservice-architectuur over te willen stappen.



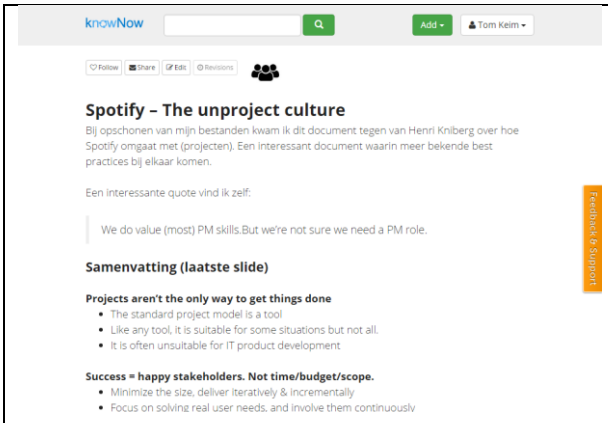
Figuur 6-3b. Een voorbeeld van een mogelijke Microservice architectuur voor knowNow. Elke service heeft zijn eigen verantwoordelijkheden en is apart schaalbaar.

Door de keuze voor de Microservice-architectuur, vervalt Deelvraag F:
“Hoe kan de MongoDB database gevuld worden en in sync worden gehouden?” van het onderzoek. Het is niet meer nodig om 2 databases synchroon te houden aangezien in die architectuur MongoDB de enige database is.

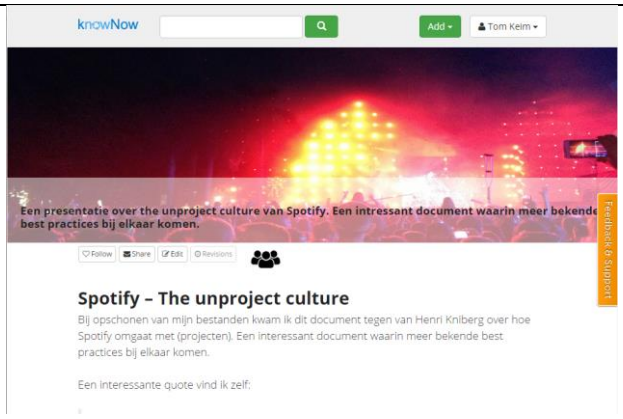
6.4.1.2 Deelvraag 2: “Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?” (uitbreidbaarheid documentschema)

In Sprint 2 is onderzocht wat de consequenties zijn op de performance en schaalbaarheid wanneer er nieuwe velden aan een Knowledge Item toegevoegd moeten worden (een uitbreiding van het documentschema). Omdat er nog geen MongoDB documentschema voor knowNow aanwezig is, het Proof-of-Concept wordt pas later ontwikkeld, is een tijdelijk documentschema geschetst waarop het onderzoek wordt uitgevoerd.

Aan de opdrachtgever is gevraagd wat een mogelijke toekomstige uitbreiding is voor het knowNow model. Hierdoor kon onderzoek worden gedaan naar een realistische uitbreiding van het documentschema. De opdrachtgever gaf aan dat in de toekomst mogelijk een **korte omschrijving** en **een afbeelding** aan een Knowledge Item toegevoegd kunnen worden. In de afbeeldingen hieronder is het effect van deze uitbreiding te vinden: links een huidig Knowledge Item, rechts een Knowledge Item na de toevoeging van de 2 nieuwe velden.



Figuur 6-4. Een oorspronkelijk Knowledge Item (niet-uitgebreid model)



Figuur 6-5. Een Knowledge Item na de uitbreiding: een afbeelding en korte omschrijving zijn toegevoegd (uitgebreid documentschema)

Performance
Om de invloed van een uitbreiding van het documentschema op de performance te onderzoeken wordt er een performance test op MongoDB uitgevoerd. Eerst zonder en daarna met de toegevoegde velden. Hierdoor wordt de invloed op de performance zichtbaar.

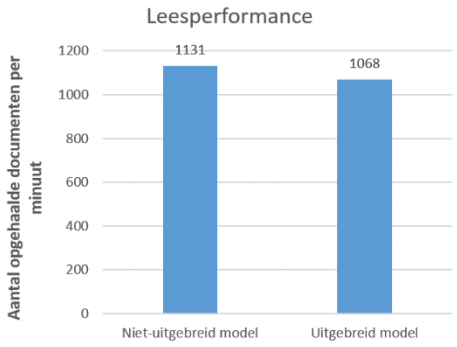
Zowel de lees- en schrijfperformance wordt onderzocht omdat de uitbreiding van het documentschema mogelijk op beiden invloed heeft. Voor de testen zijn 2 MongoDB collecties gemaakt met elk dezelfde 10.000 gegenereerde documenten. Een collectie met documentschema zonder uitbreiding, een collectie met documentschema mét uitbreiding.

Alle testen zijn uitgevoerd op een computer[6] met een standaard installatie van MongoDB. Hierdoor zijn de resultaten van de testen met elkaar te vergelijken. De testen worden met standaardinstellingen uitgevoerd en daarom is er nog ruimte voor tuning, de resultaten zijn niet bedoeld om aan te tonen hoe snel MongoDB performt in vergelijking met een andere database.

[6] De testen zijn uitgevoerd op een 64-bits Windows 8 machine. Deze machine bevat 8GB RAM en een Intel® Core™2 Quad CPU Q9400 geklokt op 2.66Ghz.

Leesperformance

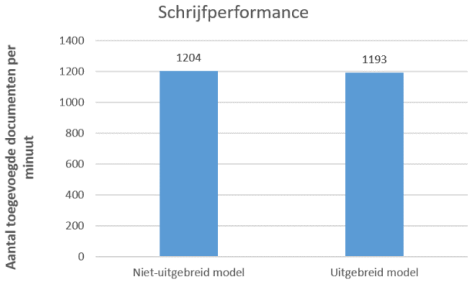
Voor de test worden door JMeter[7] 10 threads aangemaakt. Dit zorgt ervoor dat MongoDB 10 connecties tegelijkertijd moet afhandelen wat meer lijkt op een realistische situatie waarbij meerdere personen tegelijkertijd de applicatie gebruiken. Elk van deze threads haalt 500 willekeurige documenten op uit de collectie, dus 5.000 documenten totaal. Er is gekozen voor deze aantallen zodat wanneer er eenmalig uitschieters zijn dit niet leidt tot een vertekend testresultaat. De test wordt uitgevoerd op de collectie met het oorspronkelijke documentschema en de collectie met het uitgebreide documentschema. Na de tests worden de gemiddelden gebruikt. De resultaten zijn zichtbaar in figuur 6-6. Het aantal opgehaalde documenten is bij het niet-uitgebreide model met 1.131 per minuut, 13% hoger dan bij het met de nieuwe velden uitgebreide model met 1.068 documenten per minuut. Dit verschil is waarschijnlijk te verklaren doordat het aantal velden per document groeit van 11 naar 13 velden en de benodigde opslagruimte dus toeneemt.



Figuur 6-6. Resultaat test leesperformance

Schrijfperformance

Een zelfde soort test is uitgevoerd om de invloed van het uitbreiden van het documentschema op de schrijfperformance te meten. De testen zijn uitgevoerd op een nieuwe, lege MongoDB instantie. De eerste test voegt 5.000 documenten toe aan de database in het formaat van het niet-uitgebreide model, de tweede test voegt ook 5.000 documenten aan de database, maar dan in het formaat van het uitgebreide model. De 5.000 documenten worden eerlijk verdeeld over 10 threads. Elke thread voegt daardoor 500 documenten toe. De resultaten zijn te vinden in Figuur 6-7. Het verschil bij de schrijfsnelheid is minimaal. De test op het uitgebreide model kon maar 10 toevoegingen per minuut minder uitvoeren dan die van het niet-uitgebreide model. Hiermee kon worden geconcludeerd dat de uitbreiding van het documentschema weliswaar geen grote invloed op de performance heeft, maar er wel verlies optreedt.



Figuur 6-7. Resultaat test schrijfperformance

Schaalbaarheid

Bij sharding worden documenten door MongoDB verdeeld over meerdere servers. Dit verdelen gebeurt door middel van een sharding key[8]. Het toevoegen van de basale attributen zorgt er meestal niet voor dat de sharding key aangepast moet worden. De attributen short_description en image_url zijn enkel toevoegingen en hebben geen invloed op de verdeling van de data. Dit is ook uit de test duidelijk geworden. De verdeling van het aantal documenten per shard is na de aanpassing hetzelfde gebleven. Wanneer een mogelijke uitbreiding wel consequenties heeft over de verdeling van data zal echter wel opgelet moeten worden of dit geen negatieve impact heeft op de schaling. In het Onderzoeksrapport, Bijlage B, paragraaf 4.4.4 wordt verder ingegaan op schaalbaarheid.

Conclusie uitbreiding documentschema

Geconcludeerd kon worden dat de uitbreiding van een documentschema invloed heeft op de performance, maar dat dit niet zeer groot is. Er moet per situatie bekeken worden of dit het performanceverlies waard is, het toevoegen van nieuwe servers om dit performanceverlies te verkleinen is mogelijk, maar waarschijnlijk ook niet oneindig. Op de schaalbaarheid was door de uitbreiding geen invloed geconstateerd. De schaling van MongoDB werd niet beïnvloed door de uitbreiding in het documentschema.

---

[7] Apache JMeter is een applicatie waarmee verschillende soorten performance tests zijn uit te voeren[1].

[8] Een sharding key is een sleutel, vaak een bestaand veld in een document, waarmee wordt bepaald welk document op elke shard terecht komt[3].

6.5 Sprint 3: Demo migraties

Duur	2 weken 08-06-2015 t/m 19-06-2015
Producten	• Deelvraag 2: “Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?”

Is het mogelijk een MongoDB database te migreren door middel van gestapelde updates?
De knowNow-applicatie maakt gebruik van migraties. Met een migratie in knowNow is het mogelijk om een documentschema in de SQL te migreren naar een nieuwere versie. Deze migraties zijn ‘gestapeld’. Dit houdt in dat elke migratie verder gaat op de migratie die daarvoor heeft plaatsgevonden. Door het gebruik van migraties is knowNow eenvoudig consistent te houden over meerdere omgevingen. Het datamodel hoeft niet meer op elke omgeving handmatig aangepast te worden. De beheerders van knowNow hoeven alleen nog maar de migratie te starten, de rest wordt automatisch gedaan. De opdrachtgever had de vraag of dit mogelijk was bij het gebruik van MongoDB omdat stapelbare migraties in het huidige knowNow veel gebruik worden. Er is daarom besloten in deze sprint een demo te ontwikkelen die laat zien of stapelbare migraties wel of niet mogelijk zijn.

De demo is door mij ontwikkeld in C# .NET. Deze demo wordt diepgaand uitgelegd in Bijlage B, Onderzoeksrapport, paragraaf 4.6. De demo heeft de volgende, van de huidige applicatie afgeleide, functionaliteiten:

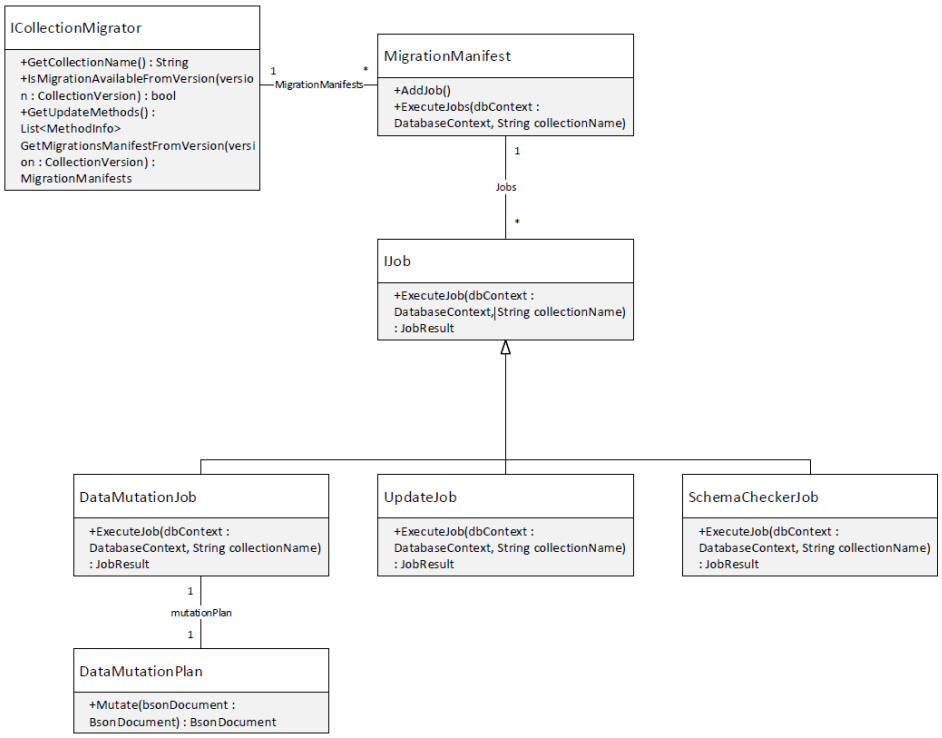
- Het per versie kunnen aangeven wat er moet gebeuren om tot de volgende versie te komen; De volgende taken zijn daarin uit te voeren:
    - Het aanpassen van het schema per document in een MongoDB collectie;
    - Het aanpassen van data per document in een MongoDB collectie; Dit gebeurt door data op te halen naar de applicatie en deze aan te passen en de aangepaste data terug te sturen naar de MongoDB server.
    - Het controleren of elk document in een MongoDB collectie overeenkomt met een schema (deze functionaliteit zit niet in de huidige applicatie, maar is toegevoegd omdat MongoDB geen schemacontrole hanteert waardoor er mogelijk inconsistenties kunnen optreden in de schema’s van documenten. Zo kan je met deze functionaliteit controleren of alle vereiste velden in elk document in een collectie aanwezig zijn).

Aan de hand van een versienummer per collectie bepaalt de applicatie welke versie een collectie heeft. Als de applicatie merkt dat er een nieuwe versie beschikbaar is, voert de applicatie de migratie naar die versie uit. Een ontwikkelaar kan een migratie maken door een zogenaamd ‘Manifest’ (figuur 6-9) aan de applicatie toe te voegen. De werking is getest aan de hand van een bedachte situatie die langs alle aspecten loopt die in de demo mogelijk zijn. Er is een migratie in de demo gebouwd die de volgende taken uitvoert, en de collectie hiermee migreert van versie 0 naar versie 1.

- Het toevoegen van een veld aan de collectie;
- Het aanpassen van een waarde in elk document van een collectie;
- Controleren of het schema van elk document voldoet aan het verwachte schema.

Na deze migratie uitgevoerd te hebben bleek de migratie geslaagd. Hiermee is geconcludeerd dat MongoDB te migreren is met stapelbare updates.

Later bleek dat het bijhouden van het versienummer per collectie misschien niet de handigste oplossing is omdat MongoDB geen schema per collectie kent, maar per document. Daarom raad ik aan om in plaats van het bijhouden van de versie per collectie de versie per document bij te houden. Hierdoor kan, mocht MongoDB tijdens een migratie uitvallen, bepaald worden welke documenten nog geüpdatet moeten worden.



Figuur 6-8. Klassendiagram van de door mij ontwikkelde migratie demo

```
[Collection(name: "knowledge")] //De naam van de te migreren MongoDB collectie
class KnowledgeMigrator : CollectionMigratorImpl
{
    //Het per versie kunnen aangeven wat er moet gebeuren om tot de volgende versie te komen
    [Migration(fromVersion: 0)]
    // Deze methode update de collectie van versie 0 naar versie 1
    public MigrationManifest UpdateFrom0()
    {
        MigrationManifest migrationManifest = new MigrationManifest();
        migrationManifest.AddJob(
            new SchemaChangeJob(
                new BsonDocument { },
                new BsonDocument {
                    { "$set",
                        new BsonDocument {

                            new BsonDocument ( "short_description", "" )
                        }
                    }
                })
            );

        return migrationManifest;
    }
}
```

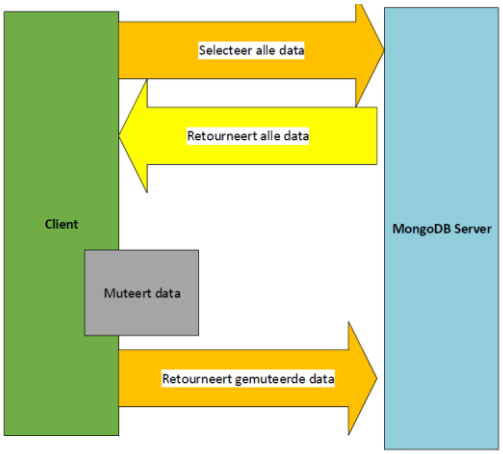
Figuur 6-9. Voorbeeld van een deel van een Manifest. In een Manifest wordt opgegeven wat er moet veranderen om een collectie te migreren. In dit voorbeeld wordt er een nieuw veld toegevoegd (“short_description”) aan alle documenten in de collectie “knowledge” en wordt de versie van de collectie verhoogd van 0 naar 1.

6.6 Sprint 4: Migraties & Opstart Proof-of-Concept

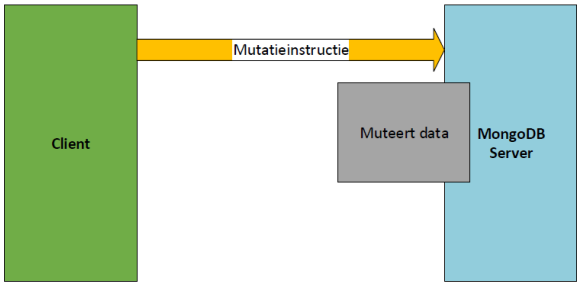
Duur	1,5 weken* 24-06-2015 t/m 04-07-2015 * De reden van de verkorte sprint is een cursus C# die in de eerste 2 dagen van deze Sprint is gevolgd. Op deze dagen kon geen werk worden verricht. Samen met de opdrachtgever is afgesproken de sprint daarom van te voren in te korten.
Producten	• Onderzoeksdocument • Deelvraag 2: "Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?" • Requirementsrapport Proof of Concept • Ontwerp documentschema

6.6.1 Onderzoek

De oplossing voor het migreren van data in MongoDB die in Sprint 3 is voorgesteld bleek niet volledig de wens van de opdrachtgever. De oplossing haalt bij het muteren van data eerst alle data op, past de data in de applicatie aan en stuurt de data daarna terug naar de database. Het nadeel hiervan is dat het ophalen van data naar de cliënt en het terug versturen van data naar de server meer tijd kost dan de mutatie op de databaseserver uit te laten voeren. Daarom wordt in deze sprint gekeken of het mogelijk is data server-side te manipuleren.



Client-side mutatie, de data wordt eerst opgehaald door de applicatie alvorens deze wordt gemuteerd.



Server-Side mutatie, de data wordt op de server gemanipuleerd.

Na het uitvoeren van het onderzoek is geconcludeerd is dat het niet mogelijk is om data server-side te manipuleren. Alleen de volgende opdrachten zijn server-side uit te voeren:

- Het optellen van een waarde bij een waarde.
- Het multipliceren van een waarde.
- Het hernoemen van een veld.
- Het aanpassen van de waarde van een veld (maar er kan geen gebruik gemaakt worden van de al bestaande waarde, waarden zijn alleen te vervangen).
- Het verwijderen van een veld.

Het is niet mogelijk om een waarde van een veld te muteren als dit niet het optellen of multipliceren van die waarde betreft. Het toevoegen van een tekst aan een bestaande tekst is bijvoorbeeld niet mogelijk via het update commando. De precieze bevinding staat beschreven in Bijlage B, Onderzoeksrapport, paragraaf 4.6.4.

6.6.2 Proof-of-Concept

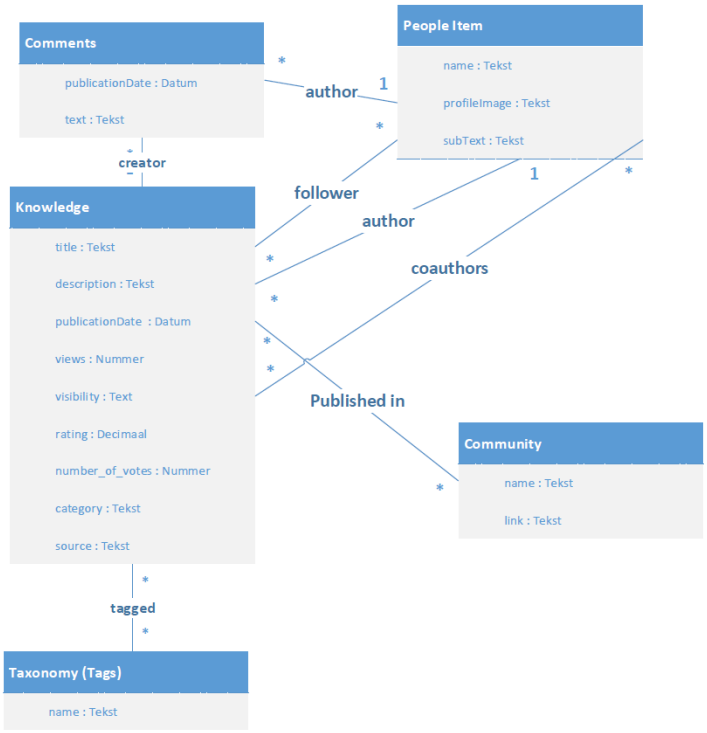
In Sprint 4 is gestart met de voorbereiding van de ontwikkeling van het Proof-of-Concept. Het Proof-of-Concept heeft als doel aan te tonen dat MongoDB voor knowNow een werkend alternatief is in een Microservice-architectuur. Door de keuze voor de Microservice-architectuur wordt er een gehele nieuwe applicatie gebouwd. Er wordt niet doorontwikkeld op het huidige Orchard.

Een requirementsanalyse is uitgevoerd door onder andere interviews te houden met de opdrachtgever. De requirements zijn gebaseerd op functionaliteiten die knowNow zeker moet bevatten wil MongoDB een mogelijkheid zijn. De opdrachtgever en ik waren het eens dat de functionaliteit voor het toevoegen, wijzigen, verwijderen en tonen van Knowledge Items in het Proof-of-Concept erg belangrijk is om aan te tonen of MongoDB een waardig alternatief is voor een RDBMS. Zonder deze functionaliteiten is knowNow niet functioneel. Daarnaast heeft een database direct invloed op deze functionaliteiten. De applicatie moet bij deze functionaliteiten schrijven, lezen of documenten wijzigen in MongoDB. Samen met de opdrachtgever zijn de requirements daarna geprioriteerd door middel van MosCoW[9]. Alle requirements zijn te vinden in het requirementsrapport, Bijlage D.

Op basis van de requirements is door mij een documentschema voor MongoDB ontworpen. Dit documentschema bevat alle velden voor gegevens van een Knowledge Item zoals deze al in knowNow aanwezig waren.

Dit documentschema en de keuzes die gemaakt zijn tijdens het ontwerpen van dit documentschema zijn te vinden in Bijlage E. De belangrijkste keuze in dit ontwerp is om alle data van een Knowledge Item op te nemen in het documentschema. Dit betekent dus ook de namen van bijvoorbeeld de auteurs (People Items). Hierdoor is het een hiërarchisch documentschema. Deze keuze is gemaakt omdat het door MongoDB zelf wordt aangeraden en uit het onderzoek uitbreidbaarheid bleek dat het toevoegen van nieuwe attributen geen grote performance impact heeft. Wanneer alle data van een Knowledge Item al direct op één plek staat betekent dit dat de data in een keer naar de gebruiker kan worden teruggeven, wat mogelijk sneller is.

In Figuur 6-10 is een functioneel domeinmodel te vinden van de Knowledge en People Items. People Items zijn gebruikers van knowNow, en kunnen zowel de auteur, coauteur, maker of volger (follower) zijn van een Knowledge Item. Dit is zo ook het geval in het huidige knowNow.



Figuur 6-10. Het functioneel domeinmodel van het Proof-of-Concept

[9] De requirements zijn geprioriteerd volgens de MoSCoW-prioritering (Stapleton, 2002). MoSCoW staat voor Must have, Should have, Could have en Won't have this time.

Het documentschema in BSON[10] formaat is zichtbaar in Figuur 6-11. De documenten worden volgens dit formaat opgeslagen in MongoDB. Rood omkaderd zijn diverse mogelijke elementen van een MongoDB BSON document. Het '_id' veld is toegevoegd omdat MongoDB verplicht dat elk veld een unieke code moet bevatten. Wanneer er tijdens het toevoegen van een document geen _id wordt meegegeven voegt MongoDB deze alsnog zelf toe[22].

De creator en author velden bevatten een zogeheten subdocument[11] met daarin de gegevens van de auteur. Het veld communities kan meerdere subdocumenten van een community bevatten. Dit is een array[12] van subdocumenten.

```
{
    peopleServiceItemId: ...,
    name: '',
    profileImage: 'http://',
    subText: ''
}
```

Figuur 6-11. Het documentschema van de collectie 'people', de collectie voor People Items. Ontworpen voor het Proof-of-Concept.

Figuur 6-12. Het documentschema gebruikt in het Proof-of-Concept, gebaseerd op het functioneel domeinmodel van figuur 6-10. Diverse termen van MongoDB zijn in het figuur omkaderd. Zichtbaar is het schema in de collectie 'knowledge', de collectie voor de Knowledge Items.

Migraties

Mijn advies aan de opdrachtgever is de ontwikkelde demo van stapelbare migraties van Sprint 4 en 5 niet te implementeren in het Proof-of-Concept. De demo heeft al laten zien de stapelbare migraties voor MongoDB werken. Dit advies is door de opdrachtgever overgenomen.

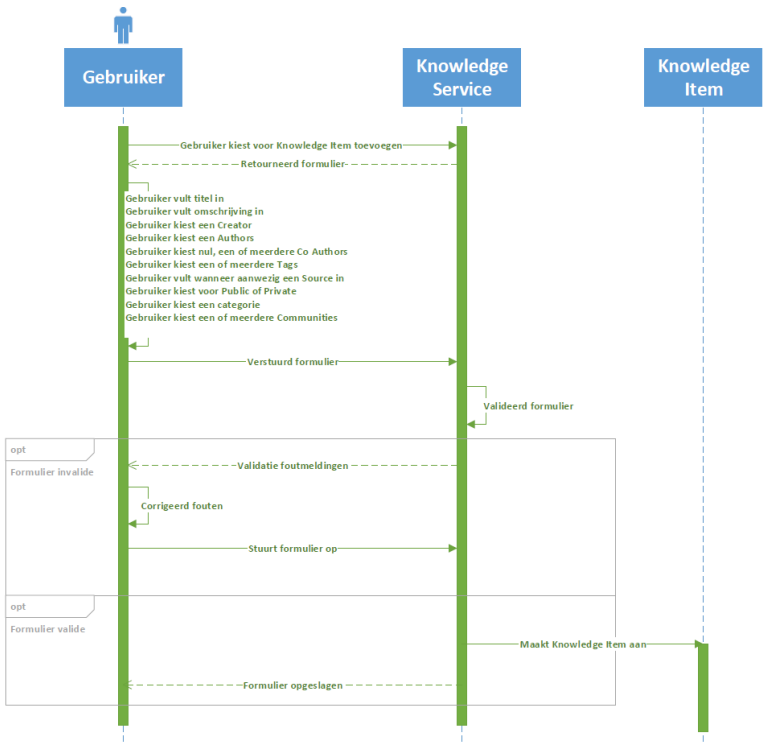
[10] MongoDB slaat documenten op in BSON formaat. Dit is een binaire vorm van JSON.
[11] Een document kan bestaan uit subdocumenten. Deze documenten kunnen ook weer hun eigen velden hebben. Elk subdocument kan ook weer een subdocument hebben. Dit kan tot maximaal 100 niveaus diep[11].
[12] Een array is een rij van nul of meerdere objecten

6.7 Sprint 5: Ontwerp & ontwikkeling PoC in een Microservice-architectuur

Duur	2 weken 06-07-2015 t/m 16-07-2015
Producten	Proof-of-Concept Functioneel Ontwerp Proof-of-Concept Technisch Ontwerp Proof-of-Concept

In Sprint 5 is gestart met het ontwerp en de ontwikkeling van het Proof-of-Concept. Voor het Proof-of-Concept is er gebruik gemaakt van de C# en .NET vanwege de aanwezige Web API in het .NET framework. Met de Web API is het mogelijk om snel een REST interface op te zetten waardoor daar minder tijd in gestoken kan worden. Daarnaast maakt het knowNow team vooral gebruik van C# wat de onderhoudbaarheid voor dat team verhoogt. Ook heb ik al enige ervaring met C#. Voor versiebeheer wordt er gebruik gemaakt van Microsoft Team Foundation Server. Op de ontwikkelmachine is MongoDB geïnstalleerd die kan worden gebruikt door het Proof-of-Concept door de door MongoDB ontwikkelde C# driver.

In het functioneel ontwerp zijn diverse Use Cases opgezet. Dit zijn functionaliteiten die de gebruiker in het Proof-of-Concept moet kunnen uitvoeren en zijn opgesteld naar aanleiding van de requirements. Alle Use Cases zijn CRUD[13]-acties voor Knowledge en People Items. In figuur 6-13 staat een functionele sequence diagram van de Use Case 'Toevoegen van Knowledge Item'. De overige Use Cases zijn te vinden in Bijlage F.



Figuur 6-13. Sequencediagram van Use Case 'Toevoegen van Knowledge Item'

[13] CRUD staat voor Create, Read, Update en Delete oftewel het toevoegen, ophalen, wijzigen of verwijderen van items.

Het Proof-of-Concept is een door mij geheel nieuw ontworpen en ontwikkelde applicatie die geen gebruik maakt van de huidige implementaties van knowNow. Van het huidige Orchard is totaal afgestapt omdat dit geen gebruik maakt van een Microservice-architectuur. Het Proof-of-Concept maakt gebruik van een Microservice-architectuur omdat dit als beste optie naar voren kwam in Deelvraag 1 van het onderzoek. Bij een Microservice worden diverse services klein gehouden door elke service zijn eigen specifieke verantwoordelijkheid te geven. Er is besloten een Knowledge Service[14] en een People Service[15] te ontwikkelen. Beide services maken gebruik van een andere collectie van MongoDB. Voor de Knowledge Service is gekozen omdat het Proof-of-Concept de Knowledge Items moet bevatten. De Knowledge Service is alleen verantwoordelijk voor de CRUD-acties van Knowledge Items waardoor de verantwoordelijkheden van de service klein zijn gehouden.

De People Service wordt ontwikkeld zodat er ook met de People Items gewerkt kan worden. People Items worden opgenomen als subdocument in de Knowledge Items (zie figuur 6-10 en figuur 6-12). Bij een wijziging van een People Item in de People Service moet de wijziging daarom ook worden doorgevoerd in de Knowledge Items (in de Knowledge Service). Hoe dat gedaan wordt moet nog worden onderzocht en wordt later toegelicht. De People Service is alleen verantwoordelijk voor de CRUD-acties van People Items waardoor de verantwoordelijkheden van de service klein zijn gehouden.

In de architectuur communiceert de gebruiker niet direct met de services. Dit zou erg onhandig zijn. Wanneer er meerdere services zijn, moet de gebruiker weten op welk adres elk van deze services (apart) te bereiken is. Ook kunnen bij de vervanging van services mogelijk de adressen veranderen. Het is veel handiger om één adres te hebben waar de gebruiker alle services mee kan bereiken. Hiervoor wordt er gebruik gemaakt van een zogenaamde Publication Layer. Deze laag weet waar alle services staan, de gebruiker hoeft dit niet te weten. Wanneer een service van http-adres verandert, hoeft dit alleen in de Publication Layer aangepast te worden.

In een eigenlijke Microservice-architectuur zit vaak op de plek van de Publication Layer een Gateway. De Gateway dient er alleen voor om data door te geven. Echter in het Proof-of-Concept wordt naast het doorgeven van data ook data gecombineerd. Zo haalt deze laag bijvoorbeeld bij het toevoegen van een Knowledge Item eerst de bijhorende People Item (zie domeinmodel figuur 6-10) uit de People Service op voordat deze het Knowledge Item doorzet naar de Knowledge Service. Voor deze opzet is gekozen omdat dit sneller is te implementeren dan op de correcte manier in een Microservice-architectuur. Hierbij voert de Gateway deze taken niet uit en moet dit door de services zelf geregeld worden.

Dit is gedaan omdat de belangrijkste reden van het ontwikkelen van het Proof of Concept is om te testen of MongoDB een optie is voor knowNow. De gekozen architectuur is een Microservice-architectuur en deze wordt deels toegepast. Echter, is dit niet het hoofdzakelijk doel van het Proof-of-Concept: er hoeft niet bewezen te worden dat de Microservice-architectuur werkt voor knowNow.

Naast het voordeel dat de gebruiker niet het http-adres van elke aparte service moet weten heeft een Gateway of Publication Layer nog andere voordelen. Zo is het de ideale plek om de gebruiker te authentiseren.

[14] De Knowledge Service is verantwoordelijk voor het toevoegen, wijzigen, ophalen en verwijderen van Knowledge Items

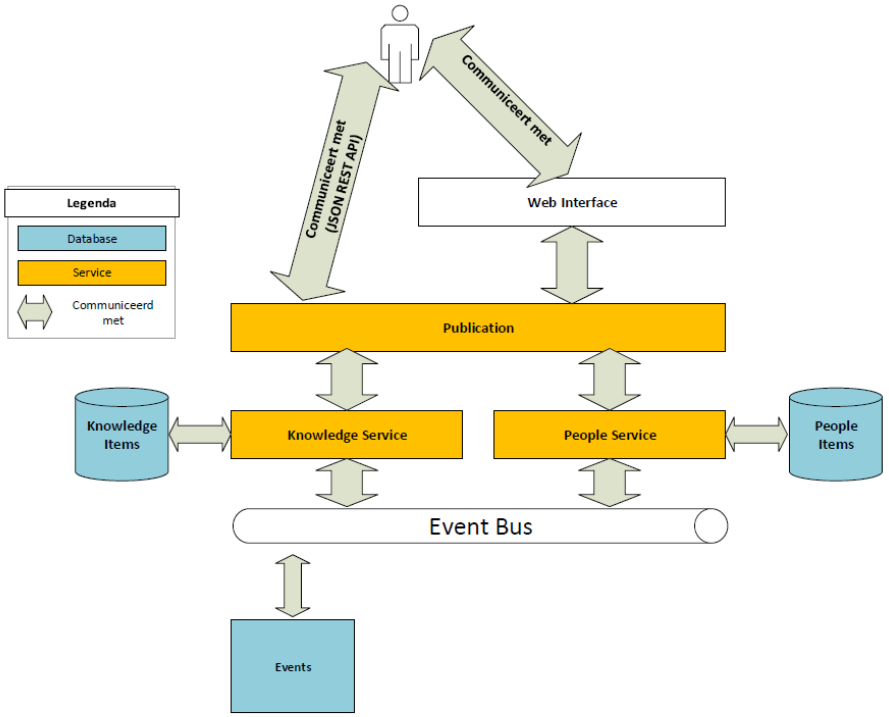
[15] De People Service is verantwoordelijk voor het toevoegen, wijzigen, ophalen en verwijderen van People Items (personen)

In Figuur 6-15 is de ontworpen Microservice architectuur voor het Proof-of-Concept te vinden, met daarin de volgende aspecten:

- De Knowledge Service en People Service hebben hun eigen collectie in MongoDB;
- Beide services communiceren alleen met hun eigen collectie. De People Service doet zelf geen aanpassingen aan de Knowledge Items omdat dit de verantwoordelijkheid is van de Knowledge Service;
- De gebruiker kan via de Publication Layer met de applicatie communiceren via een REST API en via een Web interface. Er is gekozen om naast de REST API ook een Web Interface te maken zodat het Proof-of-Concept ook visueel te tonen is aan de belanghebbenden van deze opdracht;
- De Web Interface maakt net als een eventuele gebruiker gebruik van de Publication Layer. Hiervoor is gekozen zodat eventuele logica die toekomstig in de Publication Layer wordt geïmplementeerd, zoals mogelijk authenticatie niet zowel in de Publication Layer als Web Interface geïmplementeerd hoeft te worden, maar alleen in de Publication Layer.

In de architectuur is ook gebruik gemaakt van een Event Bus. De Event Bus is een middel waarmee services asynchroon met elkaar kunnen communiceren. Een service kan een event versturen naar de Event Bus. De Event Bus zal dit event vervolgens verspreiden naar de applicaties die gebruik maken van de Event Bus. Een event kan bijvoorbeeld na een Insert, Update of Delete transactie worden verstuurd. In een Event staat onder andere wie, wat, wanneer veranderd heeft. Door het gebruik van de Event Bus kan ook na de request van de gebruiker de data asynchroon[16] worden verwerkt en hoeft de gebruiker minder lang te wachten op een response (er wordt niet gewacht op een antwoord van de Event Bus).

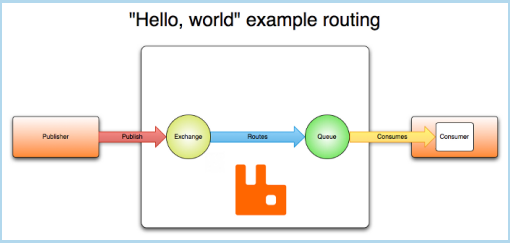
De Event Bus wordt in het Proof-of-Concept gebruikt om bij een wijziging van een People Item deze door te geven aan de Knowledge Service zodat deze de wijziging kan verwerken. Hoe dit er precies uit gaat zien wordt in komende sprints besproken.

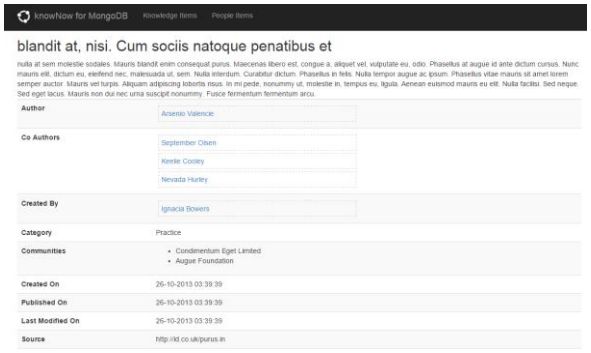


Figuur 6-15. Overzicht van de Microservice architectuur zoals gebruikt in het Proof-of-Concept. De gebruiker kan gebruik maken van de Web Interface of de Publication Layer.

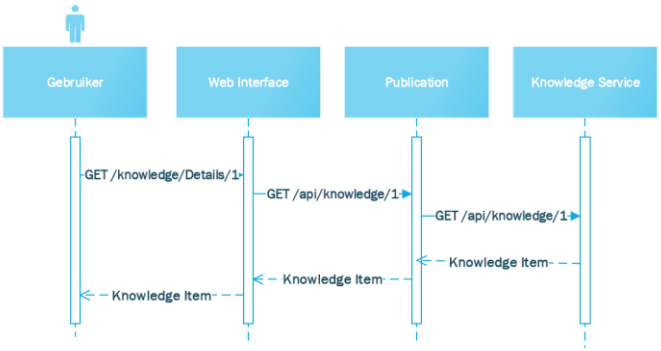
[16] Bij asynchroon laden wordt additionele informatie later geladen. De gebruiker kan alvast beginnen met lezen terwijl de additionele informatie nog wordt bijgeladen

In de tabel hieronder worden van de verschillende onderdelen van de architectuur de gebruikte technieken toegelicht. Het ontwerp van elk deel wordt uitgebreid toegelicht in het Technisch Ontwerp door middel van UML klassendiagrammen. Deze klassendiagrammen tonen de verschillende klassen van elk van de door mij ontwikkelde delen van de architectuur.

Deel	Omschrijving
Knowledge Service	De Knowledge service is een service die Knowledge Items beheerd. Met de Knowledge Service kan gecommuniceerd worden met een JSON REST API. De gebruiker communiceert niet direct met de Knowledge Service maar via de Publication Layer.
People Service	De People Service bevat de gegevens van alle personen die knowNow gebruiken (**People Items**). Met de Knowledge Service kan gecommuniceerd worden met een JSON REST API. De gebruiker communiceert niet direct met de People Service maar via de Publication Layer.
Publication Layer	Met de Publication Layer kan de eindgebruiker communiceren door middel van een JSON REST API. De Publication Layer communiceert weer met de Knowledge Service en People Service om de eindgebruiker te kunnen beantwoorden.
Web Interface	De Web Interface communiceert met de Publication Layer met een JSON REST API. De Web Interface is een MVC project. Dit houdt in dat de modellen, views en controllers van elkaar gescheiden zijn en de verantwoordelijkheden in de applicatie duidelijk worden[21]. Voor het grafische ontwerp wordt gebruik gemaakt van Bootstrap[17] zodat er snel een ontwerp op te zetten is zonder dat hier veel tijd in gaat zitten.
Event Bus	Het Proof-of-Concept maakt voor de Event Bus gebruik van RabbitMQ[9]. Met de Event Bus kunnen verschillende services in het Proof-of-Concept met elkaar communiceren door middel van Events. (bron afbeelding: https://www.rabbitmq.com/tutorials/amqp-concepts.html) 



Figuur 6-16. Een detailpagina van een Knowledge Item. Het item wordt geladen uit MongoDB via een Microservice-architectuur

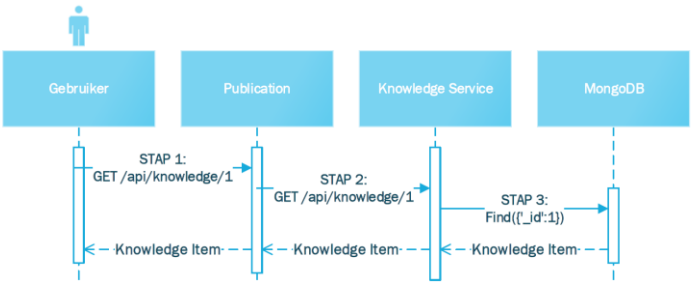


Figuur 6-17. Het ophalen van een Knowledge Item via de Microservice-architectuur getoond op serviceniveau.

17 Bootstrap is een HTML, CSS & JS framework waarmee het onder andere mogelijk is snel een grafisch ontwerp van een website op te zetten http://getbootstrap.com/

6.7.1 Voorbeeld: Ophalen van een Knowledge Item via de REST API

Het Proof-of-Concept maakt gebruik van een REST API voor de communicatie tussen de Publication Layer en de services. Een REST API is een communicatiemiddel waarbij gebruik wordt gemaakt van het HTTP protocol[7]. De data wordt in JSON (JavaScript Object Notation)[8] uitgewisseld. In het onderstaande voorbeeld wordt getoond hoe een Knowledge Item via de REST API opgehaald kan worden.



Figuur 6-18. Het ophalen van een Knowledge Item door de gebruiker

Stap 1: De gebruiker vraagt het Knowledge Item op bij de Publication Layer
De gebruiker kan niet direct gebruikmaken van alle services maar doet dit via de Publication Layer. De gebruiker kan ervoor kiezen om direct via REST te communiceren met de Publication Layer of gebruik te maken van de Webinterface. In dit voorbeeld wordt er direct met de Publication Layer gecommuniceerd.

De gebruiker wil in dit voorbeeld het Knowledge Item met de unieke code '55ffc9128b608cd5211043a6' ophalen. Het ophalen van een item via REST gebeurt met een HTTP GET-request waarbij de unieke code van het Knowledge Item in de URL wordt opgenomen: "*GET http://localhost:36359/api/knowledge/55ffc9128b608cd5211043a6".*

Het ASP.NET MVC framework bepaalt welke controller-klasse en methode in het Publication Layer project de request moet afhandelen. In deze methode wordt via REST contact opgenomen met de Knowledge Service (stap 2).

```
private string SendRequest(string url, RequestMethod method = RequestMethod.GET, string postData = "")
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
    request.ContentType = "application/json"; //Hier wordt opgegeven dat er JSON wordt terugverwacht.
    request.Method = method;
    // Als het een POST of PUT request betreft waarbij data naar de service moet worden verzonden
    // (bijvoorbeeld een gewijzigd Knowledge Item) wordt dit hieronder in de request meegegeven
    // dit wordt niet gebruikt in dit voorbeeld omdat het een GET-request betreft.
    if (method == RequestMethod.POST || method == RequestMethod.PUT)
    {
        using (var streamWriter = new StreamWriter(request.GetRequestStream()))
        {
            streamWriter.Write(postData);
            streamWriter.Flush();
            streamWriter.Close();
        }
    }
    WebResponse response = request.GetResponse();
    using (Stream responseStream = response.GetResponseStream())
    {
        System.IO.StreamReader reader = new StreamReader(responseStream, Encoding.UTF8);
        var returnStream = reader.ReadToEnd();
        // Hieronder wordt het resultaat teruggeven in een JSON-string
        return returnStream;
    }
}
```

Figuur 6-19. Het versturen van een (GET) request naar een andere service

Stap 2: De Publication Layer vraagt het Knowledge Item op bij de Knowledge Service
Nadat de Publication Layer de aanvraag voor het ophalen van een Knowledge Item heeft ontvangen, moet de Publication Layer deze data ophalen bij de Knowledge Service. Ook de communicatie tussen de Publication Layer en de services gebeurt via REST. In het figuur hiernaast (Figuur 6-19) is een deel van de code zichtbaar die wordt gebruikt om een request te sturen naar een service.

Stap 3: De Knowledge Service haalt het Knowledge Item uit de MongoDB database
Nadat de Knowledge Service de aanvraag van het ophalen van een Knowledge Item heeft ontvangen, maakt de Knowledge Service verbinding met de MongoDB database om het Knowledge Item op te halen. Dit wordt gedaan door gebruik te maken van de door mij geschreven methodes zichtbaar in Figuur 6-20.

```
public async Task<T> FindOneByIdInCollection<T>(string collection, ObjectId id)
{ //Deze methode kan één document ophalen uit een collectie aan de hand een unieke code.
    //Er wordt slim gebruik gemaakt van de methode "FindOneInCollection" op dubbele code te voorkomen.
    return await FindOneInCollection<T>(collection, new BsonDocument("_id", id));
}
public async Task<T> FindOneInCollection<T>(string collection, BsonDocument filter)
{  //Deze methode kan één document ophalen uit een collectie aan de hand van een filter
    try
    {
        // Door een timeout aan te maken van 500ms zal wanneer MongoDB niet binnen 500ms reageert de applicatie de poging stoppen.
        // MongoDB is dan waarschijnlijk onbereikbaar
        using (var timeout = new CancellationTokenSource(TimeSpan.FromMilliseconds(500)))
        {
            //Hier wordt het document uit MongoDB opgehaald aan de hand van de opgegeven filter.
            return await _database.GetCollection<T>(collection).Find(filter).FirstOrDefaultAsync(timeout.Token);

        }
    }
    catch (OperationCanceledException e)
    { //Er kon geen verbinding worden gemaakt met MongoDB binnen 500ms
        Logging.Log("MongoDB timeout exception: " + e.Message);
        return default(T);
    }
    catch (Exception e)
    { //Er is een andere fout opgetreden bij het verbinden met MongoDB. De foutmelding wordt gelogd
        Logging.Log("Exception occurred: " + e.Message);
        return default(T);
    }
}
```

Figuur 6-20. Het ophalen van het Knowledge Item uit MongoDB

De Knowledge Service haalt een Knowledge Item op door gebruik te maken van de volgende methode:

FindOneByIdInCollection<KnowledgeModel>(“knowledge”, “55ffc9128b608cd5211043a6”)

Deze methode retourneert een KnowledgeModel van het gewenste Knowledge Item (aan de hand van de unieke code). In MongoDB wordt data opgeslagen in BSON formaat (Figuur 6-21). Door middel van Class Maps[27] (figuur 6-22) is het mogelijk de MongoDB driver bij het ophalen automatisch het BSON document om te zetten in een C# object van een bepaalde klasse, zoals het KnowledgeModel (Figuur 6-23). Het voordeel van het gebruik van een Class Map is dat MongoDB het BSON object direct omzet in een in C# eenvoudig bruikbaar object. Een Class Map hoeft maar één keer in de applicatie geregistreerd te worden, en niet elke keer dat een Knowledge Item wordt opgevraagd.

Figuur 6-21. Een (deel van het) BSON document van het opgevraagde document. In figuur 6-12 is een document in groter formaat te vinden.

```
BsonClassMap.RegisterClassMap<KnowledgeModel>(k =>
{
    k.MapIdMember(km => km.Id)
        .SetIdGenerator(StringObjectIdGenerator.Instance)
        .SetSerializer(new StringSerializer(BsonType.ObjectId))
        .SetElementName("_id");
    k.MapMember(km => km.Title).SetElementName("title");
    k.MapMember(km => km.Description).SetElementName("description");
    k.MapMember(km => km.Rating).SetElementName("rating");
    //... (de rest van de class map)
});
```

Figuur 6-22. De Class Map. Hiermee kan MongoDB bij het ophalen van het document automatisch het BSON object omzetten naar een C# object van klasse 6-23

```
namespace SharedModels
{
    public class KnowledgeModel : IModel {
        public string Id { get; set; }
        public String Title { get; set; }
        public String Description { get; set; }
        public RatingModel Rating { get; set; }
        public List<TaxonomyModel> Tags { get; set; }
        public PeopleModel Creator { get; set; }
        public PeopleModel Author { get; set; }
        public List<PeopleModel> CoAuthors { get; set; }
        public List<PeopleModel> Followers { get; set; }
        public List<CommunityModel> Communities { get; set; }
        public List<CommentModel> Comments { get; set; }
        public int Views { get; set; }
        public DateTime ModificationDate { get; set; }
        public DateTime PublicationDate { get; set; }
        public DateTime CreationDate { get; set; }
        [Required]
        [Newtonsoft.Json.JsonConverter(
        typeof(Newtonsoft.Json.Converters.StringEnumConverter))]
        public Visibility Visibility { get; set; }
        public bool Deleted { get; set; }
        public String Source { get; set; }
        [Required]
        [IsValidCategory]
        public String Category {get;set;}
    }
}
```

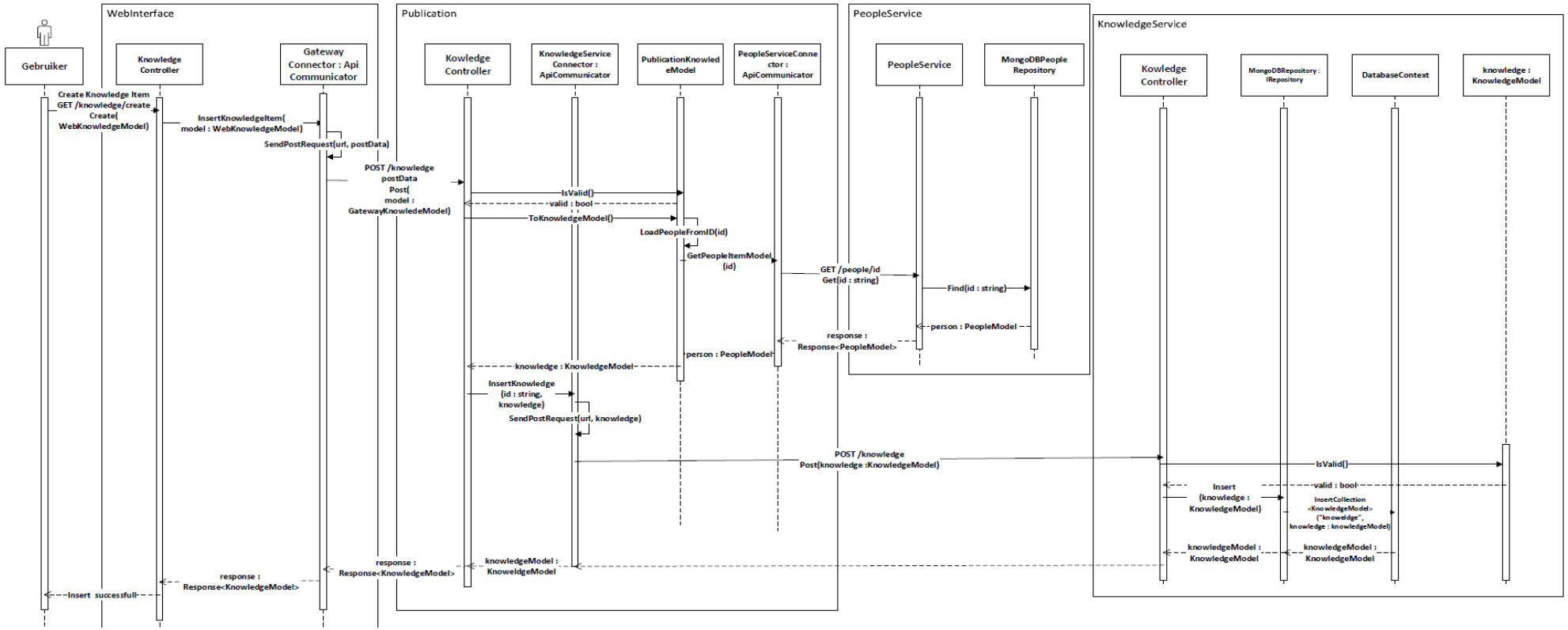
Figuur 6-23. De KnowledgeModel klasse.

Het KnowledgeModel wordt uiteindelijk omgezet in JSON-formaat zodat het geretourneerd kan worden aan de Publication Layer en uiteindelijk de gebruiker. ASP.NET MVC zorgt zelf voor het omzetten van een model naar JSON-formaat wanneer een C# klasse zoals het KnowledgeModel wordt geretourneerd. In Figuur 6-24 is het aan de gebruiker geretourneerde Knowledge Item in JSON-formaat te vinden.

Figuur 6-24. Een deel van het in JSON-formaat geretourneerde Knowledge Item

6.7.2 Werking: Toevoegen van een Knowledge Item

Op de volgende pagina’s wordt de werking toegelicht van het toevoegen van een Knowledge Item (Use case van diagram 6-12).



Figuur 6-25. De technische werking van het toevoegen van een Knowledge Item in het Proof-of-Concept

In het bovenstaande figuur is de werking van het toevoegen van een Knowledge Item door een gebruiker via de Web Interface zichtbaar. Zichtbaar zijn de diverse services die doorlopen moeten worden. De Publication Layer verzamelt tijdens het toevoegen van een Knowledge Item eerst de door de gebruiker gekozen People Items uit de People Service. Daarna kan het volledige Knowledge Item naar de Knowledge Service worden verstuurd alwaar deze opgeslagen kan worden.

```
public async Task<Response<KnowledgeModel>> Post([FromBody]KnowledgeModel value)
{
    if (!ModelState.IsValid) //Controleerd of model valide is
    {  //Model is niet valide
        var errors = new List<string>();
        foreach (var state in ModelState)
        {
            foreach (var error in state.Value.Errors)
            {
                errors.Add(error.ErrorMessage);
            }
        }

        var errorString = String.Join("; ", errors);
        //In een response object staat de status van de response(geslaagd of fout),
        //de eventuele foutmelding, en het resultaat (bijv. toegevoegd knowledge object)
        var response = new Response<KnowledgeModel>
        {
            ResponseState = ResponseState.DATA_VALIDATION_ERRORS,
            Result = null,
            Message = errorString
        };
        return response; //Stuur response errors terug
    }
    else
    { //Midel is valide, voeg het Knowledge Item toe
        // Het knowledge item wordt toegevoegd aan de repository
        KnowledgeModel model = await _repository.Insert(value);
        if (model != null)
        { //Het model is opgeslagen in MongoDB
            return new Response<KnowledgeModel>
            { //Geef een response met succes-status en het zojuist toegevoegde model terug
                ResponseState = ResponseState.SUCCESS,
                Result = model,
                Message = ""
            };
        }
        else
        { //Het model is niet opgeslagen in MongoDB
            return new Response<KnowledgeModel>
            { //Geef een response met foutmelding en fout-status terug.
                ResponseState = ResponseState.ERROR,
                Result = null,
                Message = "The Knowledge Item could not be inserted"
            };
        }
    }
}
```

Figuur 6-26. Een controller (KnowledgeController) methode van het toevoegen van een Knowledge Item in de Knowledge Service.

In het bovenstaande figuur is de code van een methode van de Knowledge Controller in de Knowledge Service te vinden. Deze methode controleert of het model valide is en voegt het Knowledge Item daarna toe aan MongoDB. Het antwoord wordt in een Response-object gezet welke door C# MVC wordt omgezet naar JSON-formaat en naar de aanvrager (de Publication Layer) wordt geantwoord. De Response klasse is door mij gemaakt en wordt in zowel de People Service en Knowledge Service gebruikt om data te retourneren waardoor de Publication Layer weet in wat voor formaat hij de data kan verwachten.

Omdat een generic[18] wordt gebruikt kunnen verschillende soorten objecten als resultaat worden opgegeven in een Response klasse, zoals een KnowledgeModel (het model van Knowledge Items) of een PeopleModel(het model van People Items).

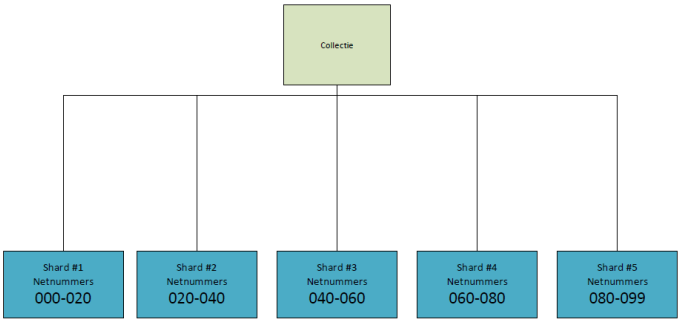
[18] Generics maken het onder andere mogelijk om het type van een variabele te bepalen bij het aanmaken van een object. https://msdn.microsoft.com/en-us/library/512aeb7t.aspx

6.8 Sprint 6: Uiteindelijke Consistentie & Advies Backups & Sharding

Duur	2 weken 27-07-2015 t/m 07-08-2015
Producten	• Advies Backups van MongoDB • Advies Sharding Key Knowledge Items • Onderzoek Deelvraag 3 • Implementatie Deelvraag 3 in onderzoek • Testen van implementatie Deelvraag 3

6.8.1 Sharding Key

Ik heb advies gegeven over de te kiezen Sharding Key voor de Knowledge Items. De Sharding Key bepaalt hoe data verdeeld moet worden wanneer er gebruik gemaakt wordt van meerdere MongoDB servers. Het kiezen van een goede Shard Key is erg belangrijk. Shard Keys zijn niet aanpasbaar. Indien er een eenmaal gekozen Shard Key veranderd moet worden, moet de gehele database geleegd worden, en daarna weer worden teruggezet. Er is voor alle mogelijke velden van een Knowledge Item getoetst op:



Figuur 6-27. Door middel van de Sharding Key wordt data verdeeld over meerdere shards (MongoDB servers). In dit voorbeeld is de Sharding Key het netnummer van een telefoonnummer.

- **Veel splitsingsmogelijkheden**; bij een hoog aantal splitsingsmogelijkheden kunnen documenten beter worden verdeeld over meerdere servers.
- **Hoge Write Scaling**; bij een hoge willekeurigheid worden schrijfopdrachten beter verdeeld over de meerdere servers.
- **Query Isolation;** bij een hoge Query Isolation wordt de Sharding Key vaak gebruikt voor het ophalen van documenten.

Er is gekozen voor een gehashte unieke code (het _id) van een document als Sharding Key. Hierdoor wordt data gelijk verdeeld over alle verschillende servers (door een hoog aantal splitsingsmogelijkheden, elke code is anders) waardoor de lees- en schrijfopdrachten (write scaling) ook gelijk worden verdeeld. De Query Isolation is hoog omdat de code wordt gebruikt om documenten op te halen. De totstandkoming van dit advies is te vinden in het Adviesrapport, Bijlage I. In de tabel hieronder is de toetsing van alle opties te vinden:

Veld	Veel splitsings-mogelijkheden	Write Scaling	Query Isolation	Score
_id (gehasht)	+	+	+	+++
creationDate (gehasht)	+	+	-	+
title (oorspronkelijke)	+	+	-	+
title (oorspronkelijke, gehasht)	+	+	-	+
description (oorspronkelijke)	+	+	-	+
description (oorspronkelijke, gehasht)	+	+	-	+
_id	+	-	+	+
creator	+	-	-	-
creator (gehasht)	+	-	-	-
creationDate	+	-	-	-
category	+/-	-	-	--
category (gehasht)	+/-	-	-	--
visibility	-	-	-	- - -
visibility (gehasht)	-	-	-	- - -

6.8.2 Job Queue

In Sprint 4 is er gekozen om in het documentschema alle gegevens van één Knowledge Item pagina op te slaan in één MongoDB document. Dit houdt in dat elk document in de MongoDB database alle benodigde gegevens van bijvoorbeeld de auteur bevat, en niet alleen een vreemde sleutel wat gebruikelijk is in een RDBMS. Dit is een hiërarchisch documentschema.

Het updaten van alle Knowledge Items wanneer een persoon zijn gegevens aanpast in knowNow is niet zomaar mogelijk in MongoDB. MongoDB kent namelijk alleen (atomaire[19]) transacties op document niveau en niet op collectie niveau[15]. Hierdoor kan het voorkomen dat bij het falen van een update een deel van de documenten wel de geüpdate naam bevat, en een deel van de documenten niet. Hierdoor raakt de data inconsistent.

Dit mag natuurlijk niet voorkomen in de productieomgeving. De gebruikers van knowNow zien dan niet de juiste data. Besloten is te onderzoeken of het mogelijk is om ervoor te zorgen dat de data in de Knowledge Service maximaal uiteindelijk consistent[20] te krijgen en te houden is met de data uit de andere services. Hierdoor is "Deelvraag 3: Hoe kan de Knowledge Service uiteindelijk consistent worden gemaakt met andere services?" in deze sprint toegevoegd aan het onderzoek.

Een inconsistente database is zo onwenselijk (omdat gebruikers dan onjuiste data blijven zien), dat deze Sprint voorrang heeft verkregen op de nog op de planning staande deelvragen D & E. Wanneer er geen oplossing kan worden gevonden voor dit inconsistentie-probleem is MongoDB geen alternatief voor SQL Server, waardoor het nut van Deelvraag D & E sowieso vervalt.

Tijdens het onderzoek zijn drie verschillende oplossingen beschreven en bedacht. Deze mogelijkheden zijn gevonden door het raadplegen van diverse bronnen zoals de MongoDB documentatie en diverse zoekopdrachten. Hierbij is gekeken naar 2-Phase-Commit, Log Reconciliation en de Job Queue. Er is gekozen voor de Job Queue. Het proces voor de keuze van de Job Queue en een uitgebreidere uitleg van de andere mogelijkheden is beschreven in Hoofdstuk 5 'Deelvraag 3' in het Onderzoeksrapport (Bijlage B).

Bij de Job Queue worden updates vanuit de People Service in een wachtrij opgeslagen. De applicatie voert de taken in de wachtrij daarna één voor één uit. Dit gaat volgens het First In, First Out principe. Een taak die als eerst in de wachtrij terecht komt, wordt ook als eerst uitgevoerd. Hierdoor is de Job Queue een idempotent[21] proces. Als de taken op dezelfde volgorde meerdere keren worden uitgevoerd, wordt de data altijd hetzelfde[18].

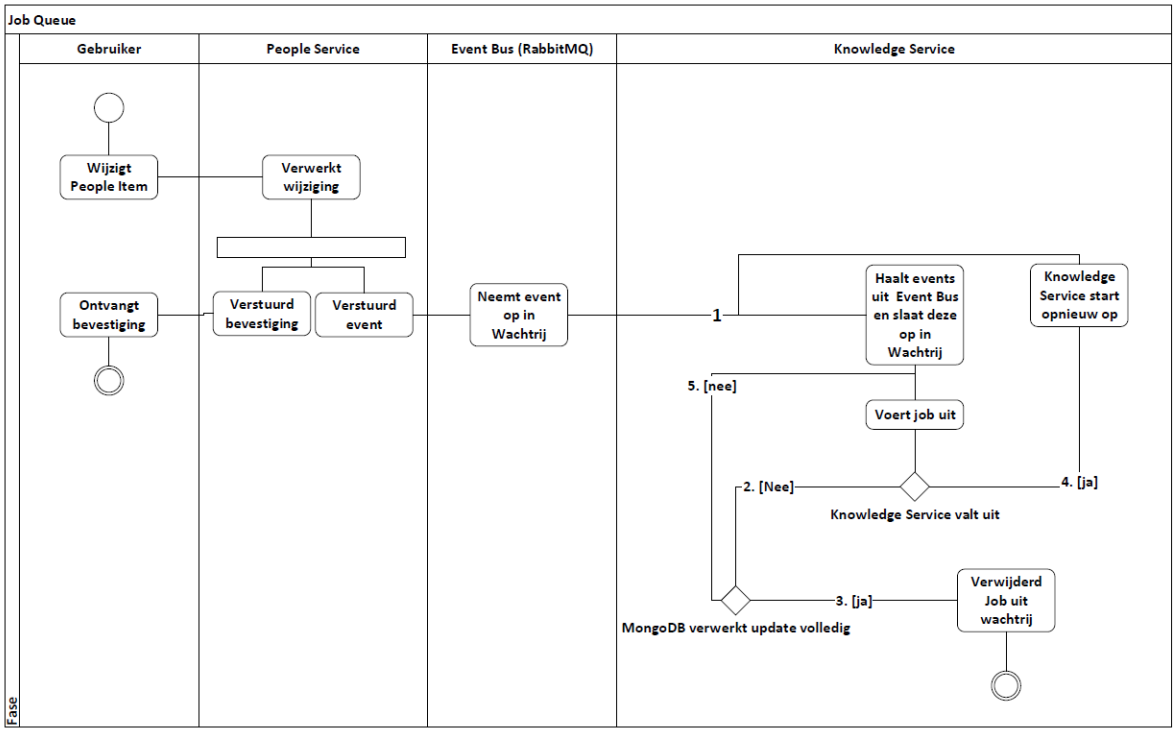
In het Proof-of-Concept is de Job Queue (Figuur 6-28) door mij ontworpen voor een Microservice-architectuur, ontwikkeld en geïmplementeerd door gebruik te maken van de Event Bus, RabbitMQ, en een lijst met taken (wachtrij) in de applicatie. Wanneer een persoon zijn gegevens wijzigt, stuurt de People Service een event naar de RabbitMQ Event Bus. De Knowledge Service luistert naar deze Event Bus en registreert alle update events in een lijst en werkt deze één voor één af. Wanneer een taak is uitgevoerd bevestigd de applicatie de update aan RabbitMQ. RabbitMQ verwijderd hierna het event.

---

[19] Bij een atomaire transactie slaagt een transactie volledig (ook al bestaat de transactie uit meerdere onderdelen), of helemaal niet.

[20] Bij uiteindelijk consistent (Eventual consisitency) is data niet direct consistent en kan het even duren voordat de gebruiker de aanpassing ziet[20].

[21] Idempotentie houdt in dat wanneer een taak opnieuw wordt uitgevoerd, het resultaat van deze taak hetzelfde is.



Figuur 6-28. De geïmplementeerde Job Queue

Mocht tijdens het verwerken van een update MongoDB uitvallen, dan bewaart de applicatie de update taak en wordt de gehele wachtrij gepauzeerd. Wanneer MongoDB weer beschikbaar komt voert deze de taak opnieuw uit en gaat verder met het verwerken van de taken in de wachtrij.

Mocht de gehele Knowledge Service uitvallen dan ontvangt de Knowledge Service alle onbevestigde events opnieuw vanuit RabbitMQ. Hierdoor raken er geen onuitgevoerde events verloren. De Knowledge Service bevestigt namelijk alleen events wanneer deze volledig zijn uitgevoerd. De Job Queue is door mij ontwikkeld en geïmplementeerd in het Proof-of-Concept.

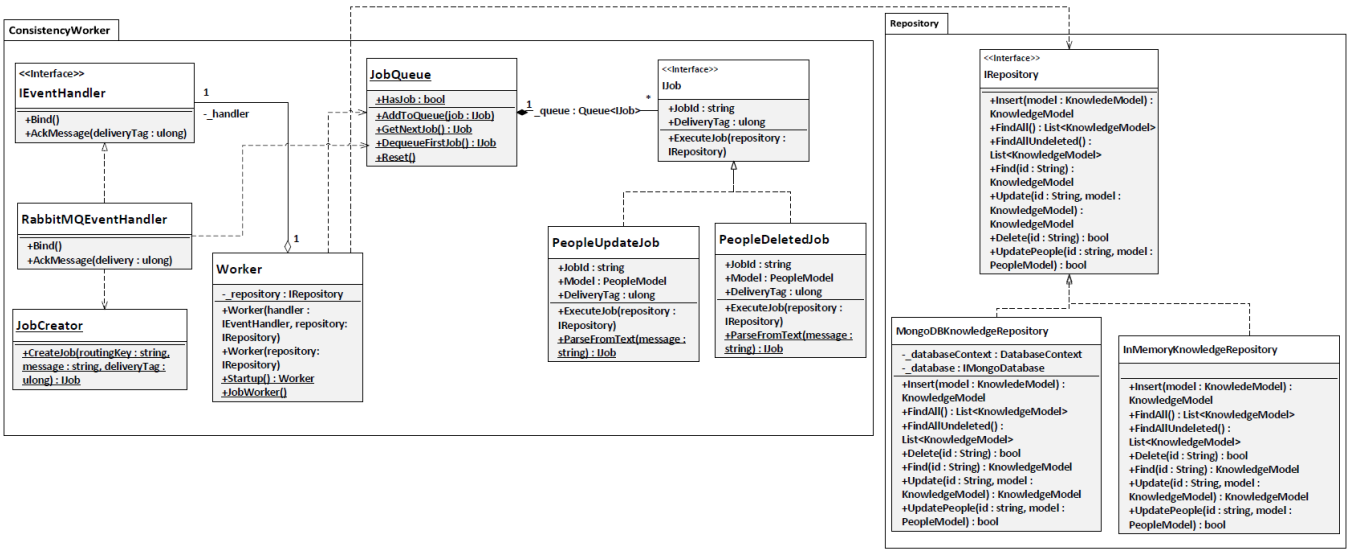
Er is gezocht naar een geschikte testvorm van TMap Next om de Job Queue te testen. Dit bleek de Proces Cyclus Test[22]. Voor de Proces Cyclus Test is gekozen omdat deze test duidelijk de verschillende paden test die de Job Queue kan belopen. Hierdoor worden alle mogelijke situaties van de Job Queue getest. De test is uitgevoerd op basis van het activity diagram van figuur 6-28. Uit de test bleek dat de Job Queue naar behoren functioneerde. Met de Job Queue is het dus mogelijk data uiteindelijk consistent te maken met data uit andere services, zoals de gegevens van de auteur. Een van de fysiek doorlopen paden (1-2-5-2-3) van het activity diagram is zichtbaar op de volgende pagina. Door de gekozen testmaat (testmaat 2) wordt elk pad in het activity diagram minstens één keer doorlopen. De op de volgende pagina getoonde testcase, waarbij MongoDB tijdens een update van een People item uitvalt, laat goed zien hoe de Job Queue zijn werk doet. De andere test cases van de Procescyclustest zijn te vinden in de Testrapportage, Bijlage H.

[22] TMap, Procescyclustest
http://www.tmap.net/wiki/process-cycle-test-pct

Stap	Verwacht resultaat	Klopt?
De gebruiker wijzigt de naam van het People Item 1 van ‘Dai Potts’ naar ‘Dai Pott’.	-	Ja
De People Service wijzigt in zijn database People Item 1 van ‘Dai Potts’ naar ‘Dia Pott’	In de People Service is de naam aangepast van ‘Dia Potts’ naar ‘Dia Pott’	Ja
De People Service maakt een Event aan waarin staat dat People Item 1 is aangepast van ‘Dia Potts’ naar ‘Dia Pott’ en stuurt deze naar de RabbitMQ Event Bus	In de RabbitMQ Event bus staat een event waarin staat dat People Item 1 is aangepast van ‘Dia Potts’ naar ‘Dia Pott’	Ja
De Knowledge Service ontvangt de Job die People Item 1 aanpast van ‘Dia Potts’ naar ‘Dia Pott’ van de RabbitMQ event Bus en slaat deze op in de wachtrij.	De Knowledge Service heeft het event/job opgenomen die People Item 1 aanpast van ‘Dia Potts’ naar ‘Dia Pott’ in zijn wachtrij	Ja
De Knowledge Service start de uitvoering van de update om People Item 1 van ‘Dia Potts’ naar ‘Dia Pott’ aan te passen.	-	Ja
MongoDB past in alle documenten de auteur met People item 1 de naam aan van ‘Dia Potts’ naar ‘Dia Pott’.	Alle auteurs in de documenten in de Knowledge Service MongoDB database zijn aangepast van ‘Dia Potts’ naar ‘Dia Pott’	Ja
MongoDB wordt uitgeschakeld tijdens het uitvoeren van de job.	De Job faalt en de Knowledge Service blijft het proberen. MongoDB is mogelijk in een inconsistente staat waarbij een deel van de documenten wel is aangepast en een deel van de documenten nog niet.	Ja
MongoDB wordt weer ingeschakeld.	MongoDB zal opstarten.	Ja
De Knowledge Service start de uitvoering van de update om People Item 1 van ‘Dia Potts’ naar ‘Dia Pott’ aan te passen opnieuw.	-	Ja
MongoDB past in alle documenten de auteur met People item 1 de naam aan van ‘Dia Potts’ naar ‘Dia Pott’.	Alle auteurs in de documenten in de Knowledge Service MongoDB database zijn aangepast van ‘Dia Potts’ naar ‘Dia Pott’	Ja
De Knowledge Service verwijdert de Job die People item 1 aanpast uit de wachtrij.	De job is uit de wachtrij verwijderd. Alle documenten in de MongoDB database die een auteur met People item 1 hebben ,hebben de nieuwe naam ‘Dia Pott’.	Ja

De Job Queue is door mij geïmplementeerd in de Knowledge Service. Echter is het misschien wenselijk om van de Job Queue een eigen service te maken. De Job Queue en de Knowledge Service worden dan apart schaalbaar. Voor het Proof-of-Concept is de Job Queue in de Knowledge Service gehouden omdat hierdoor geen nieuwe service opgezet wordt waardoor het Proof-of-Concept nog complexer wordt en omdat de connectie met MongoDB al aanwezig was in de Knowledge Service.

De Job Queue draait wel in een aparte thread, hierdoor is het proces van de Job Queue gescheiden van de rest van de Knowledge Service. Op de volgende pagina is een klassendiagram en een deel van de code te zien die in deze Sprint (Sprint 6) in het Proof-of-Concept zijn geïmplementeerd voor de Job Queue.



Figuur 6-29. De verschillende klassen van de Job Queue. De Worker klasse communiceert met MongoDB door middel van een Repository klasse. Door gebruik te maken van deze Repository pattern[26] kan bij bijvoorbeeld het testen gebruik worden gemaakt van een andere data.

```
private async Task<UpdateResult> UpdateKnowledge(FilterDefinition<BsonDocument> filter, UpdateDefinition<BsonDocument> update)
{
    //De writeconcern geeft aan dat de update op minimaal op meer dan de
    //helft van de MongoDB servers geupdate moet zijn voordat deze wordt bevestigd.
    var writeConcern = WriteConcern.WMajority;
    return await _database.GetCollection<BsonDocument>("knowledge").WithWriteConcern(writeConcern).UpdateManyAsync(filter, update);
}
public async Task<bool> UpdatePerson(string idString, PeopleModel model)
{
    try
    {
        var id = ObjectId.Parse(idString); //Het id wordt vertaald naar een MongoDB ObjectId

        var authorUpdate = await UpdateKnowledge(
          Builders<BsonDocument>.Filter.Eq("author.peopleServiceItemId", id),
          Builders<BsonDocument>.Update.Set<PeopleModel>("author", model)
        ); //Updaten van een aueur
        Logging.Log("Updated author Matched " + authorUpdate.MatchedCount + ", Modified " + authorUpdate.ModifiedCount + " documents");

        var builder = Builders<BsonDocument>.Update.Set<PeopleModel>("coauthors.$", model);

        if (model == null) {
            //Model is null, remove subdocument from array
            builder = Builders<BsonDocument>.Update.Pull<BsonDocument>("coauthors",new BsonDocument("peopleServiceItemId", id));
        }

        var coauthorsUpdate = await UpdateKnowledge(
            Builders<BsonDocument>.Filter.Eq("coauthors.peopleServiceItemId", id),
            builder
        );
        Logging.Log("Updated co-authors. Matched " + coauthorsUpdate.MatchedCount + ", Modified " + coauthorsUpdate.ModifiedCount + " documents");


        // Hier worden ook nog de creator, followers en auteurs van comments aangepast, maar omdat dit hetzelfde werkt is dit weggelaten uit dit voorbeeld.

        return (authorUpdate.IsAcknowledged && coauthorsUpdate.IsAcknowledged);
    }
    catch (Exception e)
    {
        Logging.Log("An error occured while updating person " + idString + ":" + e.Message);
        return false;
    }
}
```

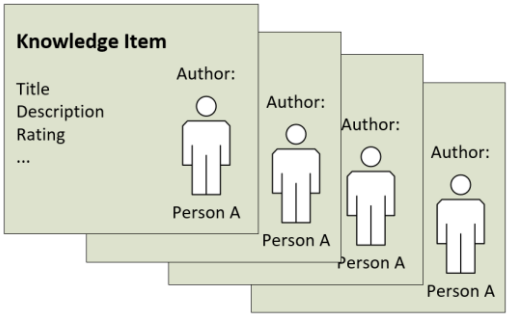
Figuur 6-30. Een deel van de code die gebruikt wordt in de Job Queue. Deze methode stuurt de update van een People Item in alle Knowledge Items door naar MongoDB en controleert of deze slaagt.

In de bovenstaande code is zichtbaar hoe de Job Queue een update uit laat voeren in MongoDB. Door middel van een Write Concern wordt aangegeven wanneer een transactie als geslaagd aangemerkt mag worden[23]. Er wordt gebruik gemaakt van de officiële MongoDB C# driver om verbinding te maken met MongoDB en hier transacties op uit te voeren.

6.8.3 Herziening documentschema

De implementatie van de Job Queue deed een nieuwe vraag ontstaan. Is het opnemen van alle data die op een Knowledge Item pagina staat wel daadwerkelijk de beste oplossing (Figuur 6-31)? Deze keuze berustte op een advies van MongoDB zelf en op het resultaat van het onderzoek uitbreidbaarheid waaruit bleek dat het toevoegen van nieuwe attributen geen grote impact had op de performance.

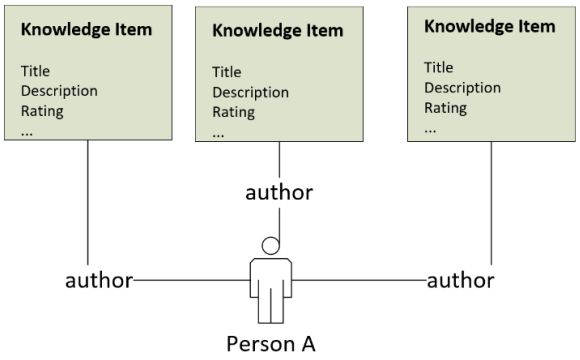
Echter dient er voor het updaten van alle data, zoals bijvoorbeeld de als subdocument opgenomen auteur, extra functionaliteit in de applicatie geïmplementeerd te worden die ervoor moet zorgen de data consistent te houden. Dit brengt mogelijk performance issues met zich mee. Dit is op te lossen met het opschalen van servers, maar dat is ook niet onbeperkt mogelijk omdat het aantal servers wel beheersbaar moet blijven. Voor elke update van een persoon moeten namelijk alle documenten in de Knowledge collectie van MongoDB nagelopen worden om daar de naam aan te passen. Er is, samen met de opdrachtgever, besloten om te kijken of het opnemen van alle data wel echt de beste optie is en of er niet beter alsnog gerefereerd kan worden naar de bijhorende relaties (Figuur 6-33). De data van de auteur moet dan in de applicatie apart opgehaald worden.



Figuur 6-31. Een documentschema waarbij data van relaties is opgenomen in een subdocument (Hiërarchisch documentschema)

```
{
    "_id" : ObjectId("55ddbc665dec6a6d0942ac54"),
    "title" : "description",
    "rating" : {
        "currentRating" : 4.0000000000000000,
        "numberOfVotes" : 935.0000000000000000
    },
    "tags" : [
        {
            "name" : "Porta Elit A Company",
            "link" : ""
        },
    ],
    "creator" : {
        "peopleServiceItemId" : ObjectId("55c60f9955534926dc700ea1"),
        "name" : "Brett Hendrix",
        "profileImage" : "",
        "subText" : "Senior Software Engineer"
    },
    "coauthors" : [
        {
            "peopleServiceItemId" : ObjectId("55c60f9855534926dc700e77"),
            "name" : "Piper Summers",
            "profileImage" : "",
            "subText" : "Manager IT Solutions"
        }
    ],
    "author" : {
        "peopleServiceItemId" : ObjectId("55c60f9855534926dc700e50"),
        "name" : "Joy Johns",
        "profileImage" : "",
        "subText" : ""
    }
}
```

Figuur 6-32. Een (deel van een) BSON document van een Knowledge Item met hiërarchisch documentschema

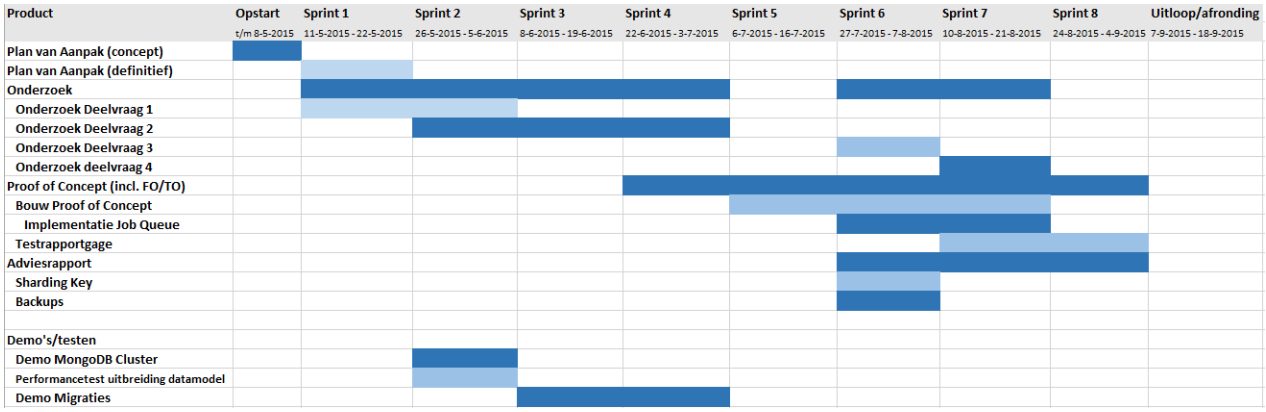


Figuur 6-33. Een documentschema waarbij er alleen gerefereerd wordt naar de relaties (Relationeel documentschema)

```
{
    "_id" : ObjectId("55ddbc665dec6a6d0942ac54"),
    "title" : "description",
    "rating" : {
        "currentRating" : 4.0000000000000000,
        "numberOfVotes" : 935.0000000000000000
    },
    "tags" : [
        {
            "name" : "Porta Elit A Company",
            "link" : ""
        },
    ],
    "creator" : ObjectId("55c60f9955534926dc700ea1"),
    "coauthors" : [
        ObjectId("55c60f9855534926dc700e77")
    ],
    "author" : ObjectId("55c60f9855534926dc700e50")
}
```

Figuur 6-34. Een (deel van een) BSON document van een Knowledge Item met relationeel documentschema

Door het ontstaan van deze nieuwe onderzoeksvraag is er naar alle waarschijnlijkheid geen tijd meer voor Deelvraag D & E. Het is belangrijker gevonden eerst deze nieuwe onderzoeksvraag te onderzoeken. Het performance onderzoek (deelvraag E) waarbij wordt gekeken of MongoDB sneller is dan SQL Server kan namelijk niet gedaan worden op een situatie die mogelijk verschilt door de uitkomst van de nieuw ontstaande deelvragen. Door de nieuwe inzichten, ook in de eerdere sprints, is de planning bijgesteld ten opzichte van week 1. Aan het einde van Sprint 6 is de planning als volgt:



Deelvraag D, E en F zijn afgevallen. Deelvraag 3 & 4 zijn toegevoegd. De redenen hiervan zijn in de voorgaande sprints besproken. De deelvragen die in Sprint 6 bekend waren zijn als volgt:

#	Deelvraag	Type onderzoek	Oorspronkelijke deelvraag
1	**Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?**	**Deskresearch**	**Deelvraag A**
1.1	Wat is de best te implementeren architectuur voor knowNow bij het gebruik van MongoDB?	Deskresearch	
1.2	Welke invloed heeft MongoDB op de uitbreidbaarheid van knowNow?	Deskresearch	
1.3	Welke invloed heeft MongoDB op de security van knowNow?	Deskresearch	
1.4	Welke invloed heeft MongoDB op de schaalbaarheid van knowNow?	Deskresearch	
2	**Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?**	**Deskresearch & Experimenteren**	**Deelvraag B & C**
2.1	Wat zijn de consequenties van het uitbreiden van het MongoDB documentschema op de performance en schaalbaarheid?	Deskresearch & Experimenteren	
2.2	Wat zijn de consequenties van het toevoegen van een nieuw REST endpoint op het documentschema en de performance?[23]	Deskresearch & Experimenteren	
2.3	Is het mogelijk een MongoDB database te migreren door middel van gestapelde updates?	Experimenteren	
3	**Hoe kan de Knowledge Service uiteindelijk consistent worden gemaakt met andere services?**	**Deskresearch & Experimenteren**	**Nieuwe deelvraag (ontstaan in sprint 5)**
4	**Is het voor knowNow beter een hiërarchisch documentschema te hanteren of een relationeel documentschema?**	**Experimenteren**	**Nieuwe deelvraag (ontstaan in sprint 6)**
D	**(afgevallen, was al optioneel en is geen tijd voor gevonden door ontstaan van deelvraag 3 & 4)** **Wat is de schaalbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB, hoe kan MongoDB worden opgeschaald wanneer het aantal gebruikers toeneemt?**		
E	**(afgevallen, geen tijd meer voor gevonden door nieuwe deelvragen 3 & 4. Deelvraag 3&4 hebben mogelijk invloed op de performance waardoor die voorrang krijgen):** **Wat is de invloed op performance bij gebruik van MongoDB tegenover een relationele database?**		
F	**(afgevallen, niet meer nodig omdat er wordt volledig wordt afgestapt van een RDBMS en er is gekozen voor een Microservice-architectuur)** **Hoe kan de MongoDB database gevuld worden en in sync worden gehouden met een Relationele Database?**		

[23] *niet behandeld in dit document, maar is wel te vinden in het Onderzoeksrapport, Bijlage B, paragraaf 4.5*

6.9 Sprint 7: Proof-of-Concept (Hiërarchisch documentschema & Relationeel documentschema)

Duur	2 weken 10-08-2015 t/m 21-08-2015
Producten	• Proof of Concept met Relationeel documentschema • Onderzoek deelvraag 4

Na aanleiding van de bevindingen van Sprint 6 is besloten om te kijken of het hiërarchisch documentschema wel de beste optie is en er niet beter gebruik kan worden gemaakt van het relationeel documentschema (Figuur 6-33). De onderstaande deelvraag is opgesteld:

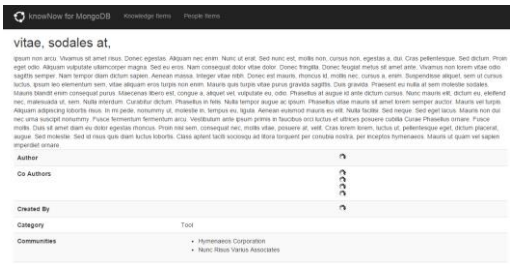
"Is het voor knowNow beter een hiërarchisch documentschema te hanteren of een relationeel documentschema?"

Om een keuze te kunnen maken heb ik besloten om naast het Proof-of-Concept, wat gebruik maakt van het hiërarchisch documentschema een 2e variant van het Proof-of-Concept te ontwikkelen. De 2e variant maakt gebruik van een relationeel documentschema. Beide Proof-of-Concepts hebben dezelfde functionaliteit, te vinden in het Functioneel Ontwerp (Bijlage F). In de tabel hieronder staan de verschillen in de werking tussen beide varianten.

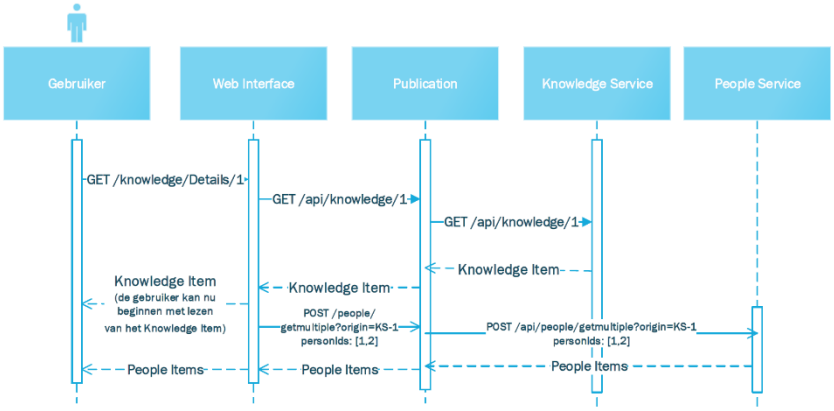
Use Case	Hiërarchisch documentschema	Relationeel documentschema
Ophalen Knowledge Item	Omdat alle gegevens in een document zitten kan het Knowledge Item in een keer volledig en synchroon[24] worden geretourneerd aan de gebruiker.	Het document bevat referenties naar de People Items. Het Knowledge Item wordt eerst naar de gebruiker geretourneerd waarna asynchroon[25] de People Items worden geladen.
Toevoegen Knowledge Item	De applicatie moet bij het toevoegen van een Knowledge Item eerst de bijhorende People Items ophalen om op te slaan in het document.	De applicatie hoeft alleen de referentie op te slaan en hoeft niet de bijhorende People Items op te halen.
Wijzigen Knowledge Item	De applicatie moet bij het toevoegen van een Knowledge Item eerst de bijhorende People Items ophalen om op te slaan in het document.	De applicatie hoeft alleen de referentie op te slaan en hoeft niet de bijhorende People Items op te halen.
Wijzigen People Item	Alle Knowledge Items met het gewijzigde People Item moeten worden bijgewerkt. Dit gebeurt door middel van de Job Queue. De Knowledge Items raken dus uiteindelijk consistent met de People Items.	Er worden geen Knowledge Items bijgewerkt. Er wordt in de Knowledge Items alleen gerefereerd naar de bijhorende People Items.
Verwijderen People Item	Uit alle Knowledge Items moet het verwijderde People Item weggehaald worden. Dit gebeurt door middel van de Job Queue.	Wanneer een gebruiker een Knowledge Item opvraagt met een verwijderd People Item worden eventuele niet meer bestaande referenties verwijderd.

[24] Bij synchroon laden wordt alle informatie gezamenlijk geladen. Hierdoor moet de gebruiker wachten totdat alle informatie is geladen.
[25] Bij asynchroon laden wordt additionele informatie later geladen. De gebruiker kan alvast beginnen met lezen terwijl de additionele informatie nog wordt bijgeladen.

Voor het nieuwe Proof-of-Concept met relationeel documentschema kon veel code van het eerder ontwikkelde Proof-of-Concept worden hergebruikt. De People Service is hetzelfde gebleven. Er zijn vooral aanpassingen gedaan aan de Knowledge Service zodat die om kan gaan met het andere documentschema. Ook de Webinterface is voor het nieuwe Proof-of-Concept aangepast. Deze moet namelijk bij het ophalen van een Knowledge Item ook de bijhorende People Items ophalen aan de hand van de referentie. Dit gebeurt asynchroon, de gebruiker kan al beginnen met het lezen van het Knowledge Item wanneer de bijhorende People Items nog worden geladen (Figuur 6-35 en Figuur 6-36).



Figuur 6-35. Het laden van bijhorende People Items (auteur, co auteurs) gebeurt in het Proof-of-Concept met relationeel documentschema asynchroon. De gebruiker kan alvast beginnen met lezen terwijl de additionele informatie nog wordt bijgeladen.



Figuur 6-36. Het ophalen van een Knowledge Item via de Microservice-architectuur getoond op serviceniveau met het relationeel documentschema. De gebruiker kan na het eerste antwoord beginnen met het lezen van het Knowledge Item.

Het asynchroon ophalen van People Items (auteurs, co auteurs) wordt gedaan door middel van jQuery en AJAX[26]. Hiermee is het mogelijk een webbrowser een asynchroon request uit te laten voeren na het laden van een pagina (Figuur 6-37). Hierdoor kan de gebruiker alvast beginnen met het lezen van het Knowledge Item terwijl de additionele informatie zoals de People Items nog wordt bijgeladen door de webbrowser.

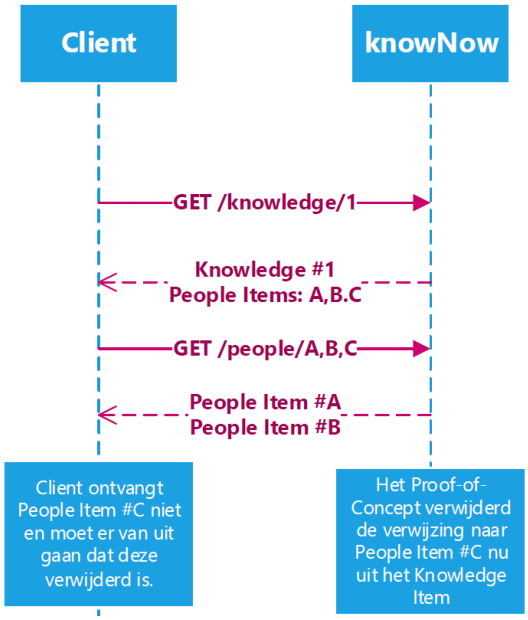
```
ProcessResponse: function (json) {
    //Rebuild list
    var persons = {};
    $.each(json, function (index, person) {
        //Het JSON object wordt omgezet naar JavaScript
        persons[person.Id] = AjaxService.getHTMLForPerson(person, true);
    });
    $(".serviceObject-person").each(function (index, element) {
        var id = jQuery(element).attr("data-id");
        if (typeof persons[id] != 'undefined') {
           jQuery(element).html(persons[id]);
        } else { //De persoon is niet gevonden.
            jQuery(element).html(AjaxService.getDeletedPersonHTML());
        }
        jQuery(element).removeClass('ajaxSpinner');
    })
},

_loadPeopleObjectsFor: function(IDs) {
    var currentUrl = window.location.href;
    var currentId = currentUrl.replace(/^(.*?)etails\/(.*?)\/?$/, '$2';
    $.ajax({
        url: AjaxService.serverAddress + AjaxService.peopleEndpoint + "?origin=KS-" +  currentId,
        method: 'post',
        dataType: "json",
        data: { personIDs: IDs}
    }).done(function (message) {
        AjaxService.ProcessResponse(message);
    });
},
```

Figuur 6-37. Het asynchroon ophalen van additionele informatie gebeurt d.m.v. Javascript, AJAX en jQuery

[26] jQuery is een JavaScript framework http://jquery.com. AJAX is een manier om de browser asynchroon te laten communiceren met een webservice

Een ander belangrijk verschil naast het ophalen van een Knowledge Item tussen beide varianten is het wijzigen en verwijderen van een People Item. Voor het wijzigen van een People Item is bij het relationeel documentschema geen Job Queue benodigd omdat er door de Knowledge Items alleen wordt gerefereerd naar de People Items.

Bij het verwijderen moet wel wat gebeuren, de referentie van het Knowledge Item naar de bijhorende People Items moet verwijderd worden zodat de data uiteindelijk consistent raakt. In SQL Server is dit eenvoudig te implementeren door referentiële integriteit[27] waarbij de database ook alle relaties automatisch verwijderd. MongoDB kent net zoals vele andere NoSQL databases geen referentiële integriteit. Hierdoor moet een verwijdering van een People Item door de applicatie gedaan worden om er voor te zorgen dat de data niet inconsistent blijft.



Figuur 6-38. Het ophalen van een Knowledge Item met verwijzing naar verwijderd People item

In het hiërarchisch documentschema is dit opgelost door de Job Queue die de referenties naar een People Item weghaalt uit alle Knowledge Items wanneer dit People Item verwijderd wordt. Het in één keer verwijderen van alle relaties bij het verwijderen van People Items zal ook voor het relationeel documentschema kunnen werken. Echter is er besloten te kijken naar een andere oplossing omdat de nadelen van het hiërarchisch documentschema dan ook in de variant met het relationeel documentschema zouden gelden. Er is gekozen voor een oplossing waarbij een relatie pas wordt verwijderd op het moment dat er tijdens het ophalen van een Knowledge Item blijkt dat een referentie naar een People Item niet meer bestaat. Ook dit gebeurt via de Job Queue, maar de relatie naar het verwijderde People Item wordt dan alleen verwijderd in het opgevraagde Knowledge Item. Een Knowledge Item wordt dus pas consistent wanneer een Knowledge Item wordt opgevraagd, niet op het moment dat een People Item wordt verwijderd zoals bij het hiërarchisch documentschema het geval is. Voor deze oplossing is gekozen zodat de data uiteindelijk wel consistent is, maar dat niet in één keer alle Knowledge Items bijgewerkt moeten worden door de Job Queue, wat mogelijk tot performanceproblemen leidt.

Na het ontwerp en de ontwikkeling van het nieuwe Proof-of-Concept is begonnen met het beantwoorden van de onderzoeksvraag. De 2 verschillende documentschema's worden vergeleken op basis van de volgende opgestelde toetsingspunten die voor knowNow belangrijk zijn gevonden:

Snelheid (performance)	De snelheid is van belang voor de gebruiker zodat deze niet te lang hoeft te wachten voordat de actie is uitgevoerd. De snelheid wordt onderzocht door middel van een performancetest op de Use Cases in de tabel van de vorige pagina. Voor de snelheid is alleen de initiële query van belang omdat deze maatgevend is voor het lezen, de gebruiker kan op dat moment starten met het lezen van het item terwijl additionele informatie, zoals bijhorende auteurs, nog wordt bijgeladen.
Schaalbaarheid	Een oplossing met een hoge schaalbaarheid is nadrukkelijk gewenst. Een oplossing met hogere schaalbaarheid kan beter meegroeien wanneer het aantal gebruikers van de applicatie toeneemt. knowNow moet om kunnen gaan met in ieder geval 200.000 en mogelijk zelfs 3 miljoen gebruikers waardoor een hoge schaalbaarheid belangrijk is.
Consistentie	Het hebben van direct consistente data heeft de voorkeur tegenover uiteindelijke consistentie.
Onderhoudbaarheid / eenvoud van de oplossing	Een beter onderhoudbare, eenvoudige oplossing is beter omdat dit het voor andere ontwikkelaars makkelijker maakt aanpassingen te doen aan de applicatie en eventuele problemen zo sneller opgelost kunnen worden.

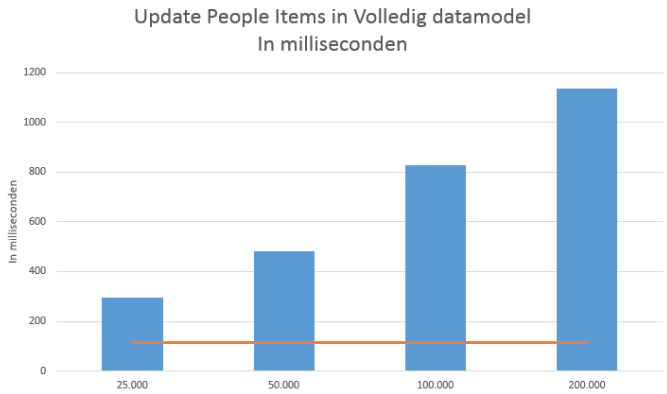
[27] Bij Referentiele integriteit garandeert de database dat er geen inconsistentie ontstaat bij het aanbrengen, aanpassen of verwijderen van een relatie. Dit wordt vaak bereikt door een transactie te blokkeren of de relatie aan te passen[24].

Alle 4 punten zijn op alle Use Cases in de tabel op pagina 49 getoetst. Dit zijn Use Cases van het Proof-of-Concept waarbij de werking per documentschema anders is. In de tabel op de volgende pagina zijn de resultaten te vinden.

Use Case/Toetsingspunt	Score hiërarchisch documentschema	Score relationeel documentschema
Ophalen Knowledge Item		
Snelheid	+/-	+/-
Schaalbaarheid	+/-	-
Consistentie	-	+
Onderhoudbaarheid	+/-	+/-
Toevoegen Knowledge Item		
Snelheid	-	+
Schaalbaarheid	+/-	+/-
Consistentie	+/-	+/-
Onderhoudbaarheid	-	+
Wijzigen Knowledge Item		
Snelheid	-	+
Schaalbaarheid	+/-	+/-
Consistentie	+/-	+/-
Onderhoudbaarheid	-	+
Wijzigen People Item		
Snelheid	+/-	+/-
Schaalbaarheid	-	+
Consistentie	-	+
Onderhoudbaarheid	-	+
Verwijderen People Item		
Snelheid	+/-	+/-
Schaalbaarheid	-	+
Consistentie	+/-	-
Onderhoudbaarheid	-	-
Totaal	**-10**	**+6**
Schaalbaarheid	-2	+1
Snelheid	-2	+2
Consistentie	-2	+1
Onderhoudbaarheid	-4	+2

Aan de hand van de bovenstaande resultaten kon worden geconcludeerd dat het relationeel documentschema voor knowNow de beste optie is. Het relationeel documentschema is meer schaalbaar, is sneller, hoeft bij wijzigingen van People Items niet consistent gemaakt te worden door de Job Queue en is daardoor ook beter onderhoudbaar omdat er minder verschillende processen zijn.

De Job Queue bij het hiërarchisch documentschema is een extra proces welke in de achtergrond de wijzingen doorvoert. Alhoewel de gebruiker hier niks van merkt, heeft het proces wel rekenkracht nodig. Uit het onderzoek is gebleken dat de Job Queue in het hiërarchisch documentschema slecht schaalt, de uitvoertijd neemt lineair toe naarmate het aantal Knowledge Items toeneemt. In Bijlage B, Deelvraag 4, wordt dieper ingegaan op het onderzoek naar het beste documentschema.



Figuur 6-39. Het wijzigen van People Items in de Knowledge Item collectie bij het hiërarchisch documentschema. De uitvoertijd neemt toe naarmate het aantal Knowledge Items toeneemt.

6.10 Sprint 8: Use Case Testing & Afronding

Duur	2 weken 24-08-2015 t/m 04-09-2015
Producten	• Adviesrapport • Testrapportage

In deze sprint is de functionaliteit van het Proof-of-Concept getest om te kijken of deze ook daadwerkelijk functioneert met MongoDB. Er is gezocht naar een geschikte testvorm van TMap Next. In het document 'Overzicht toegepaste testvormen'[25] is te vinden welke kwaliteitsattributen gelden voor welk testvormen. Er is gekozen voor een Use Case test. Bij een Use Case test worden de kwaliteitsattributen functionaliteit en inpasbaarheid getest.

Als Testbasis worden de Use Cases uit het Functioneel ontwerp (Bijlage F) en beide varianten van het Proof-of-Concept gebruikt. Door de functionaliteit van het Proof-of-Concept te testen kan worden bewezen of de Use Cases die belangrijk zijn voor de werking van knowNow ook werken in combinatie met MongoDB.

De bedoeling van de Use Case testen is niet om te testen of het Proof-of-Concept volledig functioneert maar of MongoDB kan werken voor knowNow (hierbij blijf ik dus nadrukkelijk bij de onderzoeksopdracht). Niet alle Use Cases worden getest met een testmethode. Alleen de Use Cases van de Knowledge Service worden getest (U.1 t/m U.5). De Use Cases van de People Service (U.7 t/m U.11) worden niet getest met een testmethode. Dit is gedaan omdat de functionaliteiten erg veel op elkaar lijken. Wanneer alle Use Cases van de Knowledge Service met MongoDB werken, zullen ook alle Use Cases van de People Service met MongoDB kunnen werken. Het testen of een wijziging van een People Item wordt doorgevoerd in de Knowledge Items is al gedaan met de Proces Cyclus Test. Wanneer de niet-geteste Use Cases toch niet blijken te werken ligt dat niet aan MongoDB maar aan fouten in het Proof-of-Concept zelf. Door niet alle Use Cases te testen met een testmethode wordt er tijd bespaard maar kan toch worden bewezen of MongoDB een alternatief is voor SQL server voor knowNow.

Na het uitvoeren van de fysieke testen kunnen eventueel gevonden fouten waar mogelijk worden hersteld en kan worden gesteld of de Use Cases volledig aan de verwachtingen voldoen. Wanneer dat het geval is kan worden gesteld dat MongoDB een mogelijke database is voor knowNow.

Op de volgende pagina's staat één van de uitgevoerde Use Case testen. Van deze test is eerst een Use Case (U.1: Toevoegen van een Knowledge Item) te vinden uit het Functioneel Ontwerp (Bijlage F)). Op basis van deze Use case wordt voor de Main flow en elke mogelijke alternatieve flow een logische test opgezet. Van elke logische test wordt daarna weer een fysieke test gemaakt die wordt uitgevoerd op beide varianten van het Proof-of-Concept. Wanneer het verwachte resultaat hetzelfde is als het resultaat van de test is de test geslaagd.

De andere uitgevoerde testen zijn te vinden in de Testrapportage, Bijlage G.

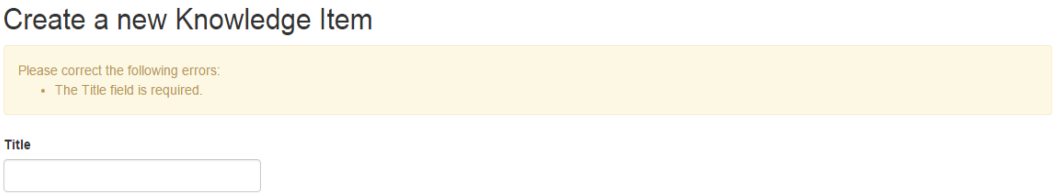
Naam	Toevoegen van Knowledge Item
ID	U.1
Omschrijving	Een gebruiker voegt een nieuw Knowledge Item toe
Primaire actors	Gebruiker
Secondaire actors	Geen
Pre-condities	Geen
Main flow	1. Gebruiker klikt op knop om Knowledge Item toe te voegen 2. Knowledge Service retourneert formulier 3. Gebruiker vult titel in 4. Gebruiker vult omschrijving in 5. Gebruiker kiest een Creator uit een lijst met Persons uit de People Service 6. Gebruiker kiest een Author uit een lijst met Persons uit de People Service 7. Gebruiker kiest nul, een of meerdere Co Authors uit een lijst met Persons uit de People Service 8. Gebruiker kiest een of meerdere Tags 9. Gebruiker vult wanneer aanwezig een Source in 10. Gebruiker kiest voor Public of Private 11. Gebruiker kiest een categorie 12. Gebruiker kiest een of meerdere Communities 13. Gebruiker verstuurd formulier 14. De Knowledge Service controleert of alle velden zijn ingevuld [1] 15. De Knowledge Service maakt nieuw Knowledge Item aan.
Post-condities	Er is een nieuwe Knowledge Item aangemaakt.
Alternatieve flow	[1] Formulier is invalide, Knowledge Service geeft foutmelding terug. Gebruiker kan formulier wijzigen en opnieuw verzenden.

In de tabel hierboven is één van de geteste Use Cases te vinden. Hiervan zijn 2 logische testcases opgezet: één voor de main flow, en één voor de alternatieve flow. De logische testcase van de alternatieve flow is hieronder te vinden:

Logische testcase U.1.2 **Main flow**	Uitzondering	Gewenst resultaat	Werkelijk resultaat	Opmerkingen
1. Gebruiker klikt op kop om Knowledge Item toe te voegen		Knowledge Service retourneert formulier	Knowledge Service retourneert formulier	
3-12. Gebruiker vult niet alle velden in				
13. Gebruiker verstuurd formulier		De Knowledge Service controleert of alle velden zijn ingevuld	De Knowledge Service controleert of alle velden zijn ingevuld	
	[1]	Formulier is invalide, Knowledge Service geeft foutmelding terug. Gebruiker kan formulier wijzigen en opnieuw verzenden.	Formulier is invalide, Knowledge Service geeft foutmelding terug. Gebruiker kan formulier wijzigen en opnieuw verzenden.	

Op basis van de logische testcase wordt een fysieke testcase uitgewerkt. In de testcase hieronder vergeet de gebruiker de titel in te vullen waardoor het formulier invalide is.

Fysieke testcase U.1.2 **Main flow**	**Uitgevoerd door:** Tom Keim **Gewenst resultaat**	**Test geslaagd:** PoC hiërarchisch documentschema: JA PoC relationeel documentschema: JA **Werkelijk resultaat**
1. Gebruiker klikt op kop om Knowledge Item toe te voegen	Knowledge Service retourneert formulier	Knowledge Service retourneert formulier
3-12. Gebruiker vult velden in: **Titel:** **Omschrijving: Test Omschrijving** **Creator: Dia Potts** **Author: Alden Bernhard** **Co Authors: Xavier Campos, Joy Johns** **Tags: Condimentum Eget Limited, Nec Ligula Ltd** **Source: http://nu.nl** **Visibility: Public** **Category: Article** **Communities: Inceptos Hymenaeos** **Industries**		
13. Gebruiker verstuurd formulier	De Knowledge Service controleert of alle velden zijn ingevuld	De Knowledge Service controleert of alle velden zijn ingevuld
	Formulier is invalide (titel is leeg), Knowledge Service geeft foutmelding terug. Gebruiker kan formulier wijzigen en opnieuw verzenden.	Formulier is invalide (titel is leeg), Knowledge geeft foutmelding terug. Gebruiker kan formulier wijzigen en opnieuw verzenden. (Figuur 6-40)



Figuur 6-40. De titel was leeggelaten. Systeem geeft foutmelding

De fysieke test is door mij uitgevoerd op beide Proof-of-Concepts. Op beide Proof-of-Concepts was de test geslaagd.

Ook alle andere Use Case tests zijn geslaagd op beide varianten van het Proof-of-Concept. Hiermee kan worden gesteld dat MongoDB voor de belangrijkste Use Cases van knowNow functioneert en het een alternatief is voor de SQL Server database van knowNow.

In deze sprint is een advies over de mogelijke inzet van MongoDB aan de opdrachtgever gegeven naar aanleiding van de constateringen die ik gedaan heb tijdens het uitvoeren van het afstudeerproject. Dit advies is te vinden in Hoofdstuk 7 van dit document.

In de weken na Sprint 8 (t/m 1 oktober) is gewerkt aan de afronding van het afstudeerverslag en het bijschaven van de bijlagen.

7. Conclusie & aanbevelingen

MongoDB heeft een zeer grote invloed op het Technisch Ontwerp van knowNow. Dit is mogelijk echter geen al te groot probleem omdat men toch al een overstap voorziet naar een Microservice-architectuur (onder ander in verband met de hoge schaalbaarheid) waarbij het Technisch Ontwerp al drastisch moet worden herzien, onafhankelijk van het type database. De overstap naar een Microservice-architectuur is daarmee bij uitstek het geschikte moment om MongoDB in te zetten.

In Deelvraag 2 is geconstateerd dat MongoDB een hoge uitbreidbaarheid heeft. Aanpassingen aan het documentschema hebben weinig gevolgen, en zijn eenvoudig door te voeren. Het updaten van het MongoDB documentschema is mogelijk door middel van migraties, wat eventueel in een applicatie in te bouwen is. Hierdoor hoeft alleen een nieuwe versie van de applicatie op een omgeving te worden gezet waarna deze zelf de collecties migreert naar de nieuwste versie.

Door de uitkomst van het onderzoek naar het documentschema raad ik het knowNow team aan gebruik te maken van een relationeel documentschema. Het relationele documentschema scoort op de geteste Use Cases en toetsingspunten het beste. Ik raad wel aan om eerst door middel van een gebruikersacceptatie test te kijken of het asynchroon ophalen van additionele informatie niet als hinderlijk wordt ervaringen.

Alhoewel MongoDB geen vast documentschema hanteert en het daarmee dus mogelijk is verschillende schema's te hanteren binnen een collectie, raad ik aan om ervoor te zorgen dat het documentschema binnen een collectie hetzelfde is. Dit zorgt ervoor dat een applicatie die communiceert met MongoDB weet welk schema hij moet verwachten en niet eerst moet kijken volgens welk schema het document is opgebouwd. Hierdoor blijft de applicatie overzichtelijker en ook beter onderhoudbaar.

In het relationele documentschema wordt de referentiele integriteit, wat MongoDB niet kent, bij het verwijderen van een relatie opgelost door een relatie te verwijderen op het moment dat wordt gemerkt dat deze niet meer bestaat. In het Proof-of-Concept is bewezen dat deze methode werkt. Echter betekent dit wel dat de documenten in een inconsistente staat blijven tot ze allemaal één keer zijn opgevraagd. De gebruiker merkt dit zelf niet en het is hierdoor voor de gebruiker van de applicatie een goede oplossing. Echter kan ik mij voorstellen dat in sommige gevallen, bijvoorbeeld wanneer er ook rapporten gemaakt moeten worden, inconsistente data wel voor problemen kunnen zorgen. Wanneer dat het geval is, raad ik aan te kijken of de Job Queue in het relationele documentschema inzetbaar is voor het direct verwijderen van alle relaties net zoals bij het hiërarchisch documentschema. Hierdoor wordt de data wel zo snel mogelijk consistent gemaakt, maar vervalt het voordeel op schaalbaarheid bij het relationeel documentschema. Bij het verwijderen van een People Item worden dan ook alle relaties in één keer verwijderd in plaats van alleen op het moment dat wordt gemerkt dat de relatie niet meer bestaat.

Op basis van het Proof-of-Concept en het onderzoek kan geconcludeerd worden dat MongoDB een werkbare database is voor de Knowledge Items van knowNow. Er moet echter wel heel veel gedaan worden om dit te bereiken. Zo moeten er diverse oplossingen worden ontwikkeld die bij een RDBMS helemaal niet nodig zijn. Daarom ontstaat de vraag of MongoDB wel daadwerkelijk een beter alternatief is. Er zijn door mij diverse zaken uitgezocht en ontwikkeld die functionaliteit nabootsen die standaard al aanwezig zijn in een RDBMS, zoals de Job Queue die ervoor zorgt dat er toch gegarandeerd kan worden dat een wijziging volledig wordt doorgevoerd. In een RDBMS is dit standaard mogelijk door middel van atomaire transacties waarbij de transactie of volledig slaagt, of helemaal niet.

Ook het missen van referentiële integriteit in MongoDB is een nadeel. De applicatie die van de database gebruik maakt zal zelf oplossingen in moeten bouwen om de data uiteindelijk consistent te maken. SQL Server heeft wel referentiële integriteit waardoor verwijzingen door de database zelf verwijderd worden.

Het knowNow team moet zich afvragen of de voordelen van MongoDB opwegen tegen de diverse concessies die gedaan moeten worden op het gebied van atomaire transacties en referentiële integriteit. De voordelen die MongoDB kan bieden op het gebied van schaalbaarheid en performance zijn binnen deze opdracht niet uitputtend onderzocht omdat er vanuit de Sprints nieuwe onderzoeksvragen zijn ontstaan waardoor hier geen tijd meer voor is gevonden. De schaalbaarheid van MongoDB lijkt erg hoog, onder andere door het toepassen van sharding maar wat exact nodig is voor knowNow om aan de verwachte groei te voldoen is onbekend. Mocht de opdrachtgever MongoDB toch willen inzetten, ondanks deze zwaarwegende concessies, raad ik aan eerst een performance onderzoek te doen waarbij wordt gekeken naar wat benodigd is om de gebruikers een high performant applicatie te kunnen aanbieden. Wanneer de groep gebruikers in die test steeds wordt vergroot en de MongoDB database indien nodig bij wordt geschaald, kan worden gekeken wat er benodigd is aan servers.

De resultaten van dit onderzoek kunnen dan vergeleken worden met een vergelijkbaar Proof-of-Concept (ook met Microservice-architectuur) wat gebruik maakt van SQL Server of een alternatief RDBMS als database. Maak geen gebruik van de huidige datastructuur van het op Orchard ontwikkelde knowNow, maar een voor een Microservice-architectuur geoptimaliseerde datastructuur waardoor de resultaten beter te vergelijken zijn.

Wanneer blijkt dat de performance van MongoDB zo veel beter is dan die van een RDBMS waarbij dit voordeel opweegt tegen de concessies van MongoDB, kan besloten worden om MongoDB daadwerkelijk in te zetten. Iedereen die aan het project werkt moet dan wel zeker weten waar ze aan beginnen en hoe de diverse processen, zoals een Job Queue, werken om ongewenste problemen in de toekomst te voorkomen. Voor het team moet duidelijk zijn wat wel met MongoDB kan en vooral wat MongoDB niet kan. Met name de missende atomaire transacties en het niet bieden van referentiële integriteit zijn sterk complicerende factoren voor het slagen van MongoDB als vervanger van de huidige database.

8. Evaluatie

In dit hoofdstuk wordt de evaluatie van het project behandeld. Het proces en de producten worden geëvalueerd en de behaalde beroepstaken worden omschreven. Ook wordt beschreven hoe het afstuderen binnen Info Support door mij is ervaren.

8.1 Procesevaluatie

Over het algemeen ben ik tevreden over het proces. Het project is goed uitgevoerd en er is invulling gegeven aan de wensen van de opdrachtgever. Gedurende het project zijn er een aantal kritische punten en leermomenten geweest die hieronder worden toegelicht

8.1.1 Onderzoeken met Scrum

Als ontwikkelmethode is gebruik gemaakt van Scrum. Door middel van 8 korte sprints van elk 2 weken kon er vaak feedback worden verwerkt van de opdrachtgever en de technisch begeleider van Info Support in de Sprint Retrospective. Door het gebruik van Scrum voor zowel het onderzoek als het Proof-of-Concept kon snel worden ingespeeld op nieuwe inzichten en wensen van de opdrachtgever, zoals het consistent moeten houden van de data of de ontstane vraag of een eerder ontworpen documentschema wel de beste optie is.

De nieuwe inzichten hebben er toe geleid dat er aanpassingen zijn gedaan in het project en de bijhorende planning. Zo zijn er nieuwe deelvragen ontstaan, maar heb ik ook in samenspraak met de opdrachtgever enkele onderdelen moeten laten vallen. Door het missende performance onderzoek, waarbij SQL Server vergeleken zou worden met MongoDB, kon niet worden bepaald of MongoDB een sneller alternatief als database is voor knowNow. Ondanks dat er door het gebruik van de Scrum methodiek een aantal belangrijke onderzoeksvragen zijn komen te vervallen, ben ik van mening dat de keuze voor Scrum, ook tijdens het onderzoek, de juiste is gebleken. Wanneer niet was ingespeeld op de nieuwe inzichten waren er producten opgeleverd waar de opdrachtgever weinig voordeel aan zou hebben. Zo zou het Proof-of-Concept alleen een hiërarchisch documentschema gehad hebben, wat helemaal niet de beste optie bleek voor knowNow en was een performanceonderzoek uitgevoerd op een Proof-of-Concept met minder geschikt documentschema. Dit had een vertekend beeld kunnen geven.

De vele Sprint Retrospectives hebben gezorgd voor veel nuttige feedback op de verschillende documenten. Deze feedback ging over zowel taalgebruik als inhoud. De opdrachtgever adviseerde mij in de eerste weken van het afstuderen te letten op mijn schrijfstijl en heeft mij ook gedurende het verdere verloop van het afstudeerproject regelmatig van feedback en voorbeelden voorzien. Deze feedback heeft geleid tot verbeterde kwaliteit van dit afstudeerverslag en bijbehorende documenten.

Gedurende Sprint 1 werd duidelijk dat de wens van het doorontwikkelen op de huidige knowNow applicatie op een verkeerde aanname berustte. In de bijhorende Sprint Retrospective is dit bijgesteld en zijn afspraken gemaakt ten einde herhaling te voorkomen. Herhaling heeft ook niet plaatsgevonden waardoor ik vind dat dit goed is opgelost, waarmee het nut van Scrum nogmaals werd benadrukt.

De volgende keer zal ik het onderzoek op een zelfde manier doen. Alhoewel de deelvragen van het onderzoek veranderd zijn is er nog steeds antwoord gegeven op de hoofdvraag en heeft de Scrum methodiek er voor gezorgd dat de belangrijkste vragen van de opdrachtgever naar voren zijn gekomen en beantwoord konden worden.

8.1.2 Proof-of-Concept

Over het proces van het ontwikkelen van het Proof-of-Concept ben ik zeer tevreden. Voordat er is begonnen aan het Proof-of-Concept is er eerst een C# cursus gevolgd waardoor mijn kennis van C# is opgehaald en uitgebreid. Vervolgens is een requirements analyse uitgevoerd waardoor de requirements duidelijk konden wordenopgesteld. Er is een Functioneel en Technisch ontwerp opgezet waarna gestart is met de ontwikkeling van het Proof-of-Concept. De ontwikkeling van het Proof-of-Concept ging goed. Door middel van Scrum werd vooraf duidelijk welke functionaliteiten in welke Sprint geïmplementeerd zouden worden. Hierdoor is een Proof-of-Concept gerealiseerd waarmee zowel ik als de opdrachtgever tevreden zijn en die bewijst dat MongoDB een alternatief is voor SQL server.

In Sprint 7 is besloten om een 2e variant van het Proof-of-Concept op te zetten waarbij een ander documentschema werd gehanteerd. Dit was nodig om te kunnen onderzoeken welk documentschema beter is voor knowNow. De opzet van de 2e variant van het Proof-of-Concept vergde dusdanig veel tijd dat besloten is het performanceonderzoek te schrappen, waarmee dus helaas 2 niet-functionele requirements (die eisten dat het Proof-of-Concept high-performant moet zijn) niet getest konden worden. Ondanks dat ik de enige was die aan het Proof-of-Concept heeft gewerkt, heb ik toch gemeend gebruik te moeten maken van versiebeheer om te allen tijde terug te kunnen vallen op een voorgaande versie.

8.1.3 Testen (TMap)

Voor het testen van de Job Queue is gebruik gemaakt van de Proces Cyclus Test van TMap Next. In dit document is beschreven welke kwaliteitsattributen zijn getest, voor welke testmaat is gekozen en hoe de test is uitgevoerd. Voor de test van de Job Queue zijn eerst logische testgevallen opgezet waarna fysieke testgevallen zijn uitgevoerd. Na het uitvoeren van de fysieke testgevallen bleek de Job Queue volledig te werken.

Ook is er een Use Case Test uitgevoerd op beide varianten van het Proof-of-Concept zodat bekeken kon worden of het Proof-of-Concept ook daadwerkelijk functioneert met MongoDB. Dit is gedaan in de laatste sprint, toen het gehele Proof-of-Concept af was. Uit de test bleek dat de geteste Use Cases functioneerde waardoor de conclusie getrokken kon worden dat MongoDB een werkende database kan zijn voor MongoDB. Ik zal een testrapportage de volgende keer op dezelfde manier opzetten.

8.1.4 Info Support

Binnen Info Support zijn er meerdere personen die mij hebben begeleidt:

- de procesbegeleider, die het proces in de gaten houdt;
- de technische begeleider waar technische vragen aan te stellen waren, en
- de opdrachtgever die de producten controleert.

Ik ben zeer tevreden over de begeleiding. Bij vragen werd er spoedig geantwoord en de opdrachtgever gaf altijd zeer behulpzame en constructieve feedback. Binnen Info Support heerst een zeer prettige en professionele sfeer. Niet alleen de belanghebbenden van de afstudeeropdracht zijn behulpzaam maar ook andere mensen binnen de organisatie bleken bereid hun kennis te delen en stonden altijd snel paraat. Ook bood Info Support door middel van cursussen de mogelijkheid om kennis waar nodig bij te schalen. Zo heb ik een cursus C# gevolgd om mijn kennis daarvan te vergroten en toe te passen.

8.2 Productevaluatie

8.2.1 Plan van Aanpak

Over het Plan van Aanpak ben ik redelijk tevreden. Het Plan van Aanpak bevat de eerste planning van het project en beschrijft wat er moet worden uitgevoerd en hoe dit moet worden aangepakt. De planning uit het Plan van Aanpak is in het begin nog grotendeels gevolgd. Door het aanpassen van de deelvragen (die naar voren zijn gekomen door Scurm) heeft het onderzoek langer geduurd en is het niet gelukt het onderzoek volledig af te ronden voordat er werd begonnen aan het Proof-of-Concept en is er afgeweken van de oorspronkelijke planning in het Plan van Aanpak . Ik ben van mening dat de keuze om van de planning af te wijken goed was om zo de nieuwe, belangrijke inzichten te kunnen onderzoeken. In het Plan van Aanpak worden naast de planning ook de opdrachtomschrijving, de op te leveren producten en de risico's besproken zodat de opdrachtgever en ik weten wat er gedaan gaat worden.

8.2.2 Onderzoeksrapport

Het onderzoeksrapport heeft samen met het Proof-of-Concept de meeste tijd gevergd.. Ik ben over het algemeen tevreden over het onderzoeksrapport. De belangrijkste vragen van de opdrachtgever zijn onderzocht en de hoofdvraag is beantwoord. Toch had ik van te voren misschien beter vast kunnen zetten wat er precies onderzocht moest worden, immers zijn er 2 nieuwe deelvragen bijgekomen en 2 deelvragen vervallen waardoor de oorspronkelijke planning niet meer klopte. Ik twijfel of het van te voren opzetten van een meer uitgebreid onderzoeksontwerp hierbij had geholpen. Hierbij had ik alsnog dezelfde, oorspronkelijke deelvragen, gebruikt en waren de deelvragen waarschijnlijk alsnog aangepast door nieuwe inzichten, deze werden immers belangrijker gevonden door zowel de opdrachtgever als ik.

Ik ben zeer tevreden over de conclusie van het documentschema. Alhoewel ik de oorspronkelijke keuze voor het hiërarchisch documentschema te snel gemaakt heb, vind ik dat ik het onderzoek naar het beste documentschema daarna goed heb uitgevoerd. Ook de opdrachtgever ging in eerste instantie uit van het hiërarchisch documentschema. De nieuwe inzichten en de uitkomst van het daarop gerichte onderzoek heeft er voor gezorgd dat de opdrachtgever nu ook van mening is dat het relationeel documentschema beter is.

Op basis van het onderzoek en advies is door de opdrachtgever op basis van de zwaarwegende concessies besloten om MongoDB niet in te zetten voor knowNow. De opdrachtgever heeft aangegeven door het door mij uitgevoerde onderzoek en bijbehorende advies veel tijd, geld en energie te besparen doordat anders tijdens de ontwikkeling van de applicatie duidelijk was geworden dat MongoDB niet de oplossing is voor knowNow

8.2.3 Proof-of-Concept

Ik ben zeer tevreden over het gerealiseerde Proof-of-Concept. In het Functioneel en Technisch ontwerp werd duidelijk wat er moest gebeuren voor het Proof-of-Concept. Het Functioneel Ontwerp bevat alle Use Cases die het Proof-of-Concept moet bevatten. Het Proof-of-Concept gebruikt diverse technieken waar ik nog nooit mee gewerkt had zoals de Microservice-architectuur en de Event Bus RabbitMQ. Beide technieken voldoen aan alle hoge geprioriteerde (met een *Must have* of *Should have* prioritering) requirements en het documentschema bevat alle velden van een Knowledge Item. De werkende Use Cases in het functioneel ontwerp hebben aangetoond dat MongoDB ook een alternatief is. Voor het Proof-of-Concept is gebruik gemaakt van Object Georiënteerd Programmeren (OOP) en van versiebeheer door middel van TFS. OOP heeft ervoor gezorgd dat de verantwoordelijkheden binnen de verschillende services duidelijk zijn. Het ontwerpen en ontwikkelen van een Proof-of-Concept zal ik de volgende keer op dezelfde manier doen.

8.2.4 Requirementsrapport

Over het requirementsrapport ben ik zeer tevreden. Samen met de opdrachtgever zijn in een interview de requirements opgesteld en daarna door middel van MoSCoW geprioriteerd. Dit heeft ervoor gezorgd dat het voor de opdrachtgever en ik duidelijk was wat er moest gebeuren. Het requirementsrapport zal ik de volgende keer op dezelfde manier opstellen.

8.2.5 Testrapportage

Over de testrapportage ben ik tevreden. Door de Proces Cyclus test is de Job Queue goed getest en kan gesteld worden dat deze werkt. Door middel van de Use Case test zijn Use Cases positief getest waarmee gesteld kan worden dat het Proof-of-Concept functioneert met MongoDB waardoor het nut van het testen zeer hoog was. Ik zal de testrapportage de volgende keer op dezelfde manier uitvoeren.

8.2.6 Adviesrapport

Over het Adviesrapport ben ik zeer tevreden. Ik vind dat ik de keuze voor de Sharding Key goed heb opgepakt en door te kijken naar diverse toetsingpunten de beste keuze heb kunnen maken waar het knowNow team ook op kan vertrouwen. Ook over de rest van het rapport ben ik zeer tevreden. Ik heb ondanks het werkende Proof-of-Concept kritisch naar MongoDB als database voor knowNow gekeken en aan de opdrachtgever duidelijk gemaakt dat MongoDB misschien niet de oplossing is voor knowNow.

8.3 Beroepstaken

8.3.1 Ontwerpen, bouwen en bevragen van een database

Tijdens het afstuderen is gebruik gemaakt van MongoDB, een nieuwere NoSQL document storage database die afwijkt van een traditionele relationele database. Tijdens het afstuderen heb ik kennis gemaakt met MongoDB en geleerd hoe de database werkt. Er zijn diverse experimenten uitgevoerd met MongoDB waarmee de kennis van het product is verhoogd.

Er zijn 2 varianten van het documentschema voor de Knowledge Items van knowNow ontworpen: een hiërarchisch documentschema en een relationeel documentschema. Door het uitvoeren van een onderzoek kon een advies worden gegeven voor het beste ontwerp voor het documentschema ten behoevevan knowNow.

Beide varianten zijn gebruikt in een lokale MongoDB installatie. Deze database werd bevraagd door beide Proof-of-Concepts. Door de door mij ontworpen en ontwikkelde Job Queue is het ondanks het missen van transacties op collectieniveau te garanderen dat de Knowledge Items uiteindelijk consistent worden.

Door middel van de keuze van de Sharding Key voor de Knowledge Items heb ik knowNow al voorbereid op een eventuele opschaling in de toekomst.

Ik ben van mening dat ik deze beroepstaak goed heb uitgevoerd en heb gehaald. Er is gebruik gemaakt van een nieuwe, voor mij onbekende, database die een andere manier van modeleren vraagt. Op basis van de 2 ontworpen en geteste varianten kon een goed onderbouwde keuze voor het ontwerp van het documentschema worden gemaakt. De MongoDB database is gebouwd en vervolgens bevraagd door beide Proof-of-Concepts. Door middel van de Job Queue heb ik aangetoond dat ondanks het missen van atomaire transacties op collectie niveau, het toch mogelijk is te garanderen dat de MongoDB database uiteindelijk consistent raakt.

8.3.2 Bouwen applicatie

Deze beroepstaak is gehaald door het bouwen van 2 varianten van het Proof-of-Concept in een Microservice-architectuur en het daarbij om kunnen gaan met MongoDB binnen deze applicaties.

Elk deel van de Microservice-architectuur, de Presentation Layer, de Knowledge Service, de People Service en de client (de webinterface), is een opzichzelfstaande applicatie ontwikkeld in C# .NET. Er is gebruik gemaakt van de C# MongoDB .NET driver en de Web API van .NET. Er is Object georiënteerd geprogrammeerd en gebruik gemaakt van enkele design patterns zoals de Repository pattern. Ook is er gebruik gemaakt van het MVC-model waardoor de gebruikersinterface is gescheiden van de logica van de applicaties waarbij de verantwoordelijkheden duidelijk zijn verdeeld. Voor versiebeheer is gebruik gemaakt van Team Foundation Server. De door mij ontwikkelde Job Queue maakt gebruik van de Event Bus RabbitMQ driver en zal bij een update-event van bijvoorbeeld de People Service automatisch, in de achtergrond, de wijzigingen asynchroon verwerken in de Knowledge Items. Dit gebeurt in een aparte thread.

Ik ben van mening dat ik deze beroepstaak goed heb uitgevoerd en behaald vanwege de meerdere losstaande applicaties in de Microservice-architectuur die door mij zijn ontwikkeld. Hierbij is gebruik gemaakt van frameworks als MVC .NET en de Web API. Ook is er gebruik gemaakt van voor mij nieuwe technologieën zoals een REST API voor communicatie tussen de services en de Event Bus RabbitMQ.

8.3.3 Uitvoeren van en rapporten over het testproces

Deze beroepstaak is op twee manieren uitgevoerd en behaald. Er zijn tijdens het onderzoek diverse experimenten, zoals bijvoorbeeld de performancetest tussen de 2 documentschema's, uitgevoerd die in het onderzoeksrapport (Bijlage B) uitgebreid beschreven zijn. Hierbij wordt beschreven wat en hoe er getest wordt en wat het resultaat van de test is.

Daarnaast is voor de Job Queue, die door mij is ontwikkeld voor het Proof-of-Concept, een Proces Cyclus Test uitgevoerd die beschreven is in de testrapportage (Bijlage H). In dit document heb ik beschreven welke kwaliteitsattributen ik heb getest, hoe ik de test heb opgezet, voor welke testmaat ik heb gekozen, hoe ik de logische testen heb opgezet en hoe ik de fysieke testen heb uitgevoerd. Ook is er een Use Case test uitgevoerd waarmee is aangetoond dat MongoDB voor knowNow een werkende database is. Ik ben van mening dat ik deze beroepstaak heb behaald. Dit komt mede door de diverse experimenten die zijn uitgevoerd en de testrapportage die is opgesteld voor de Job Queue en de Use Cases.

8.3.4 Ontwerpen systeemarchitectuur

De beroepstaak is uitgevoerd en behaal door het ontwerpen van het Proof-of-Concept. In Deelvraag 1 (Onderzoeksrapport, Bijlage B) is gekozen voor de Microservice-architectuur. Dit is een architectuur met meerdere losstaande applicaties die elk hun eigen verantwoordelijkheden hebben. Door het gebruik van een Microservice-architectuur kan elke service apart worden vervangen en worden geschaald (terwijl bij een monolithisch systeem het gehele systeem gedupliceerd wordt wanneer er geschaald moet worden). Hierdoor is een Microservice-architectuur erg schaalbaar. In het Technisch Ontwerp zijn beide varianten van het Proof-of-Concept, met de verschillende documentschema's, ontworpen. Door middel van UML klassen- en sequentiediagrammen wordt de ontworpen architectuur verduidelijkt.

Ik ben van mening dat ik deze beroepstaak heb behaald. Er zijn diverse mogelijke architecturen vergeleken waarbij rekening werd gehouden met onderhoudbaarheid en toekomstgerichtheid. In het Technisch Ontwerp wordt de architectuur, die bestaat uit meerdere losstaande applicaties, uitgebreid beschreven door middel van UML diagrammen.

Bijlagen

Externe bijlagen

Bijlage	Documentnaam	Meegeleverd in
Bijlage A	Plan van Aanpak	Bijlagenboek 1 (Bijlagen A t/m D)
Bijlage B	Onderzoeksrapport	Bijlagenboek 1 (Bijlagen A t/m D)
Bijlage C	Technisch Ontwerp oorspronkelijke situatie knowNow	Bijlagenboek 1 (Bijlagen A t/m D)
Bijlage D	Requirementsrapport	Bijlagenboek 1 (Bijlagen A t/m D)
Bijlage E	Ontwerp documentschema	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage F	Functioneel ontwerp	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage G	Technisch ontwerp	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage H	Testrapportage	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage I	Adviesrapport	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage J	Afstudeerplan	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage K	Evaluatieformulier afstuderen	Bijlagenboek 2 (Bijlagen E t/m L)
Bijlage L	Beoordeling tussentijds assessment	Bijlagenboek 2 (Bijlagen E t/m L)

Verklarende woordenlijst

Asynchroon laden	Bij asynchroon laden wordt additionele informatie later geladen. De gebruiker kan alvast beginnen met lezen terwijl de additionele informatie nog wordt bijgeladen.
Atomaire transactie	Een atomaire transactie slaagt volledig, of helemaal niet. Een atomaire transactie kan niet deels slagen.
BSON	MongoDB slaat documenten op in BSON formaat. Dit is een binaire vorm van JSON.
Consistentie	Bij consistentie is dezelfde data binnen een of meerdere datasets gelijk
Document	Een document is vergelijkbaar met een rij in een RDBMS en valt onder een collectie. MongoDB kent geen vast schema per collectie. Elk document in een collectie kan een ander schema hebben. Een document heeft velden. Een document is opgeslagen in BSON formaat.
Event Bus	De Event Bus is een middel waarmee services asynchroon met elkaar kunnen communiceren
Eventual consisitency	Bij uiteindelijk consistent (Eventual consisitency) is data niet direct consistent en kan het even duren voordat de gebruiker de aanpassing ziet
HTTP	Hypertext Transfer Protocol (HTTP) is het protocol voor de communicatie tussen een client en een server
Idempotentie	Wanneer een taak opnieuw wordt uitgevoerd is het resultaat van deze taak hetzelfde.
JSON	JSON, JavaScript Object Notation is een gegevensformaat waarbij data in een leesbaar formaat wordt opgeslagen. http://json.org/
Knowledge Item	Een Knowledge Item is een onderdeel van de knowNow applicatie. Het is een samenstelling van allerlei attributen die op een Knowledge Pagina staan. Onder een Knowledge Item vallen onder andere een titel van het document, omschrijving en tags.
MoSCoW	Prioriteren van requirements d.m.v. MoSCoW- (Stapleton, 2002). Staat voor Must have, Should have, Could have en Won't have this time.
People Item	Een onderdeel van knowNow. Een People Item bevat alle informatie van een persoon in knowNow, zoals bijvoorbeeld een auteur.
RabbitMQ	RabbitMQ is een Event Bus applicatie. https://www.rabbitmq.com/
RDBMS	RDBMS staat voor Relationeel Database Management Systeem. Een bekend RDBMS is Microsoft SQL Server
Referentiele integriteit	Bij Referentiele integriteit garandeert de database dat er geen inconsistentie ontstaat bij het aanbrengen, aanpassen of verwijderen van een relatie. Dit wordt vaak bereikt door een transactie te blokkeren of de relatie aan te passen
REST API	Een REST API is een communicatiemiddel waarbij gebruik wordt gemaakt het HTTP protocol. De data wordt in JSON uitgewisseld.
REST endpoint	Een REST endpoint is een deel van een REST API, bijvoorbeeld het ophalen van bepaalde items.
Sharding	Sharding is het horizontaal schalen van een MongoDB database. Bij Sharding worden nieuwe MongoDB instanties toegevoegd en wordt de data verdeeld over meerdere instanties. Dit verdelen gaat met behulp van Sharding Keys.
Sharding Key	Een Sharding Key is een veld die elk document in een gesharde collectie bevat. Aan de hand van een sharding key wordt de data over de verschillende shards (een MongoDB instantie) verdeeld.
Sycnhroon laden	Bij synchroon laden wordt alle informatie gezamenlijk geladen. Hierdoor moet de gebruiker wachten totdat alle informatie is geladen.
Thread	In een applicatie kunnen meerdere threads worden aangemaakt die allemaal in een ander proces draaien en parallel kunnen worden uitgevoerd.
TMap NEXT	TMAP Next is een testaanpak ontwikkeld door Sogeti.
Use Case	Een Use Case is een beschrijving van het gedrag van het systeem, veroorzaakt door een verzoek van buiten het system zoals de gebruiker.

Referenties

[1]	Apache Jmeter http://jmeter.apache.org/
[2]	MongoDB, “Shard” http://docs.mongodb.org/manual/sharding/
[3]	MongoDB, cSharding Key” http://docs.mongodb.org/manual/core/sharding-shard-key/
[4]	Scrum Guide http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf
[5]	Orchard, “What is Orchard?” http://docs.orchardproject.net/Documentation/frequently-asked-questions#WhatisOrchard?
[6]	MongoDB, “MongoDB Limits and Thresholds” http://docs.mongodb.org/manual/reference/limits/
[7]	Elkstein, “Learn REST: A Tutorial” http://rest.elkstein.org/
[8]	JSON (Javascript Object Notation) http://json.org/
[9]	RabbitMQ, “Features” https://www.rabbitmq.com/features.html
[10]	Martin Fowler, “Microservices” http://martinfowler.com/articles/Microservices.html
[11]	MongoDB, “MongoDB Limits and Thresholds” http://docs.mongodb.org/manual/reference/limits/
[12]	Techopedia, “What is SQL Server?” http://www.techopedia.com/definition/1243/sql-server
[13]	Devbridge, “Benefits of NoSQL” https://www.devbridge.com/articles/benefits-nosql/
[14]	How Kanban Enables Continuous Delivery: LeanKit’s Perspective http://leankit.com/blog/2014/06/kanban-and-continuous-delivery/
[15]	Shingijutsu-global, “Just in Time” http://www.shingijutsu-global.com/en/justintime.html
[16]	MongoDB, “Write Operation Atomicity” http://docs.mongodb.org/manual/core/write-operations-atomicity/
[17]	Info Support, “Klantenreferenties” https://www.infosupport.com/klantreferenties/
[18]	Foldoc, “Idempotentie” http://foldoc.org/idempotent
[19]	Chiron Solutions, “What is the difference between RUP and SCRUM methodologies?” http://www.chiron-solutions.com/chiron-professional-journal/2010/12/20/what-is-the-difference-between-rup-and-scrum-methodologies/
[20]	Oracle, “Eventual Consistency Explained” http://www.oracle.com/technetwork/products/nosqldb/documentation/consistency-explained-1659908.pdf
[21]	MSDN, “Model-View-Controller” https://msdn.microsoft.com/en-us/library/ff649643.aspx
[22]	MongoDB, “Glossary” http://docs.mongodb.org/manual/reference/glossary/
[23]	MongoDB, “Write Concern” http://docs.mongodb.org/master/core/write-concern/
[24]	MSDN, “Implementing Referential Integrity and Cascading Actions” https://technet.microsoft.com/en-us/library/aa902684(v=sql.80).aspx
[25]	TMap, “Overzicht Toegepaste testvormen” http://www.mentorum.nl/docs/Traindocs/Overzicht_Toegepaste_testvormen.pdf
[26]	Microsoft Developer Network, “The Repository Pattern” https://msdn.microsoft.com/en-us/library/ff649690.aspx
[27]	MongoDB, “Mapping Classes” http://mongodb.github.io/mongo-csharp-driver/2.0/reference/bson/mapping/