Bijlagen A t/m D

MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?

Door: Tom Keim

Bijlage A – Plan van Aanpak
Bijlage B - Onderzoeksrapport
Bijlage C – Technisch Ontwerp oorspronkelijke situatie knowNow
Bijlage D - Requirementsrapport

Titel	Bijlagen A t/m D	**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.

Bijlagen

In dit boek zijn de volgende bijlagen te vinden:

Bijlage A	**Plan van Aanpak**
Bijlage B	**Onderzoeksrapport**
Bijlage C	**Technisch Ontwerp oorspronkelijke situatie knowNow**
Bijlage D	**Requirementsrapport**

DE HAAGSE HOGESCHOOL

Info Support Solid Innovator

Bijlage A – Plan van Aanpak

MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?

Plan van Aanpak

Titel	Bijlage A - Plan van Aanpak
Project/Onderwerp	MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?
Versie	1.0
Status	Definitief
Datum	22-05-2015
Bestand	Plan van Aanpak v1.0
Bedrijf	Info Support
Door	Tom Keim

Info Support B.V.
Kenniscentrum
Kruisboog 42, 3905 TG, Veenendaal,
De Smalle Zijde 39, 3903 LM Veenendaal,
Tel. +31 (0) 318 55 20 20 - Fax +31 (0) 318 55 23 55
Tel. +31 (0) 318 50 11 19 - Fax +31 (0) 318 51 83 59

Inhoudsopgave

1. Opdracht

1.1 Opdrachtgever en opdrachtnemer

De opdrachtnemer voor de in dit plan van aanpak beschreven delen is:

Tom Keim tom.keim@infosupport.com
tomkeim94@gmail.com

De opdrachtgever voor de in dit plan van aanpak beschreven delen is:

Joop Snijder joop.snijder@infosupport.com

1.2 Opdrachtdefinitie

1.2.1 Aanleiding

1.2.1.1 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.

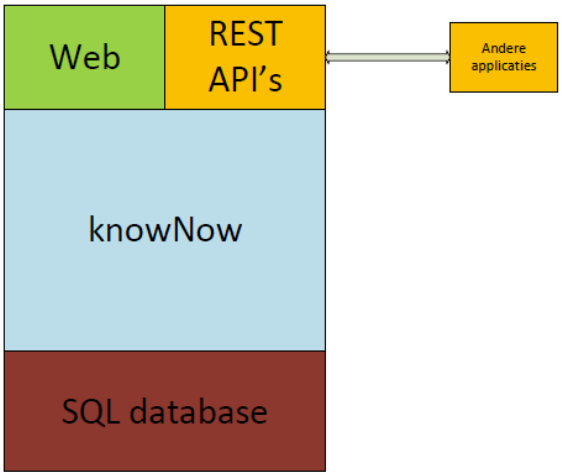
Inmiddels wordt knowNow al bij meerdere organisaties gebruikt, waaronder Info Support en ABN AMRO.

Het team van knowNow is een DevOps ontwikkelteam. DevOps is het uitvoeren van zowel de ontwikkelkant (dev) als de beheerkant (ops)[4]. Hetzelfde team werkt aan het ontwerp, de ontwikkeling en het beheer van de applicatie.

1.2.1.2 knowNow als platform

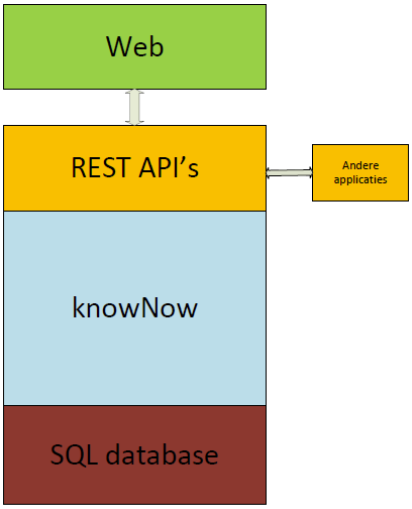
Een jaar geleden is door het DevOps ontwikkelteam van knowNow het idee ontstaan om van de applicatie een platform te maken. Dit houdt in dat de er naast de webinterface, wat nog de enige manier was om het systeem te gebruiken, er REST API's geïntroduceerd worden waar andere applicaties op in kunnen haken. Hierdoor wordt knowNow een platform[2].

De REST API is nu grotendeels compleet en de meeste functionaliteiten zijn ook via deze API's uit te voeren. De API's zijn op dit moment naast de webinterface ontwikkeld.



Figuur 1. De REST API's zijn naast de webinterface ontwikkeld. De web applicatie en REST API's communiceren met dezelfde SQL database.

In de toekomst zal door het ontwikkelteam van knowNow de webinterface worden gescheiden van knowNow en zal deze ook gebruik maken van de REST API's waardoor de API's dé manier zijn om te kunnen communiceren met knowNow.



Figuur 2. Toekomstig knowNow (versimpeld). De webinterface en de andere applicaties maken gebruik van de REST API's om met knowNow te communiceren. De web applicatie en REST API's communiceren met dezelfde SQL database.

1.2.1.3 Verwachte groei

Op het moment van schrijven zijn er 7000 gebruikers van knowNow. De gebruikers worden 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.

1.2.1.4 Scheiding van services

Om de verwachte groei te ondersteunen is er gekozen om een aantal services binnen knowNow van elkaar te scheiden. Onder deze services vallen onder andere:

- Lucene.NET voor de searchengine, een andere afstudeerder is bezig met onderzoek naar Elasticsearch als mogelijke vervanger van Lucene.NET;
- een Recommendation service (op basis van de GraphDB Neo4J).

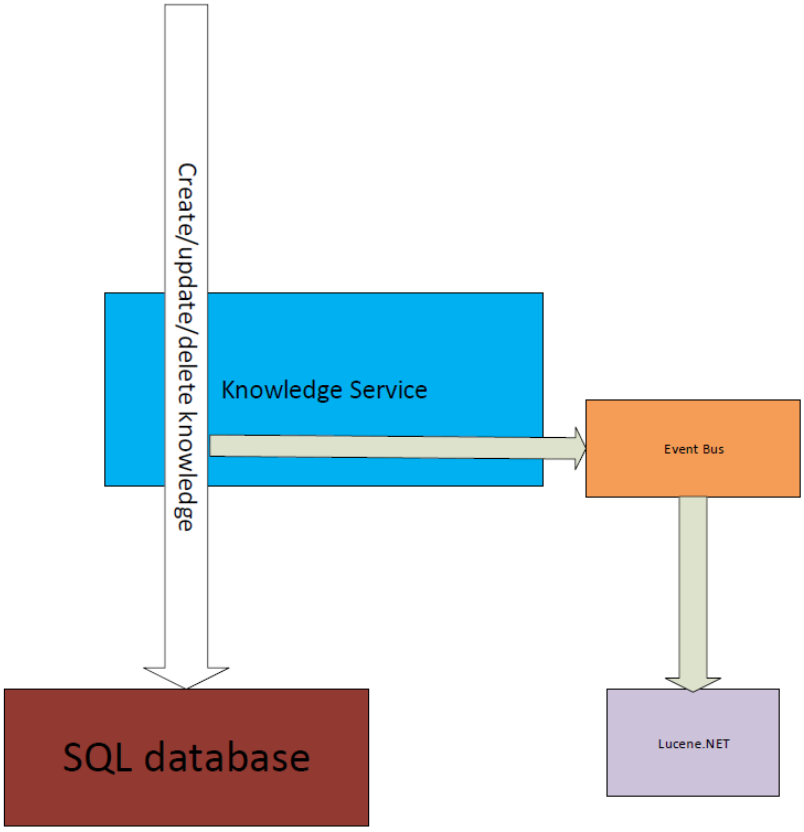
De services zijn van elkaar gescheiden en kunnen daarom ook apart worden uitgerold over de servers die ze nodig hebben. Zo kan er wanneer de searchengine meer rekenkracht nodig heeft voor worden gekozen om alleen die service te dupliceren. Zo hoeft niet de gehele applicatie gedupliceerd te worden maar alleen de services die meer rekenkracht nodig hebben.

Een bijkomend voordeel van gescheiden services is dat voor elke service specifieke keuzes gemaakt kunnen worden. Zo kan bijvoorbeeld, als dat sneller blijkt, gekozen worden om voor de recommendation service Python te gebruiken en voor de andere componenten andere programmeertalen.

1.2.1.5 Event Bus

knowNow maakt gebruikt van een Event Bus. Een Event Bus is een stuk programmatuur waar gebeurtenissen in de applicatie worden geregistreerd. Denk aan de toevoeging of wijziging van data.

In een event staat onder andere wat er veranderd is, wanneer het veranderd is en door wie de transactie is gemaakt. Hierdoor hoeft de gebruiker niet te wachten tot de request door alle delen van de applicatie verwerkt is. Sommige taken, zoals bijvoorbeeld het bijwerken van de index ten behoeve van het zoeken binnen knowNow, kunnen buiten de request gebeuren, waardoor de gebruiker minder lang hoeft te wachten op een response.



Figuur 3. Een schets van de werking van de event bus. Ten tijde van een request wordt de transactie opgeslagen in de Event Bus & de SQL database.

1.2.1.6 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.

1.2.1.7 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.

Bij een NoSQL database is er geen vaste structuur. Elke rij in een tabel/collectie kan zijn eigen structuur hebben (schema-loos). In een SQL oplossing is dit per tabel vastgelegd. Er kan bij NoSQL in een document alle data opgeslagen worden in een rij(document), in plaats van in 50 à 60 tabellen. Hiervoor hoeven er geen 20 verschillende query's uitgevoerd te worden maar staat alles op één plek. Daarnaast zijn NoSQL oplossingen zeer schaalbaar[3].

1.2.2 Probleemstelling

Het knowNow team wil voor de knowledge engine service gebruik gaan maken van de NoSQL document database MongoDB. NoSQL databases vragen een andere manier van data modellering. NoSQL databases kunnen goed om gaan met data-opslag en met verschillen in de data. Maar welke consequenties het gebruik van een NoSQL database zal hebben op het technisch ontwerp en op het ophalen van de data in knowNow is niet bekend en zal moeten worden onderzocht.

Ook is de uitbreidbaarheid van knowNow indien er gebruik wordt gemaakt van MongoDB onbekend. Zo is niet bekend of de datastructuur van de gegevens aangepast dient te worden indien er een nieuwe (REST) interface bijkomt.

Daarnaast zal ook naar de schaalbaarheid van MongoDB gekeken moeten worden. Hoe kan er voor worden gezorgd dat MongoDB mee kan groeien met het aantal gebruikers? Hoe moet er omgegaan worden met Eventual consistency van eventueel meerdere instanties van MongoDB?

1.2.3 Doelstelling

De opdrachtgever wil dat er onderzoek wordt gedaan wanneer voor het opvragen van documenten in de knowledge engine gebruikt gemaakt wordt van MongoDB in plaats van de relationele SQL database.

Er zal een Proof-of-Concept applicatie ontwikkeld moeten waarbij gebruik wordt gemaakt van een MongoDB database. Dit Proof-of-Concept moet functionaliteit die geïmplementeerd wordt via een REST API kunnen afhandelen. Het is belangrijk de requirements van de opdrachtgever in kaart te brengen.

Om dit Proof-of-Concept te kunnen realiseren moet er eerst onderzoek worden gedaan naar een aantal aspecten die nu nog onbekend zijn. Zo moet er onderzoek worden gedaan naar de invloed van MongoDB op het technisch ontwerp van knowNow (in het technisch ontwerp wordt onder andere de architectuur, performance en security besproken).

Er moet onderzoek worden gedaan welke invloed de gebruikte interface (REST endpoints) heeft op de wijze van data opslag in MongoDB. Is het bijvoorbeeld mogelijk om iets op één manier op te slaan wanneer er 3 verschillende endpoints zijn?

Ook moet er onderzoek gedaan worden naar de uitbreidbaarheid bij het gebruik van MongoDB wanneer er later nieuwe (REST endpoints) toegevoegd dienen te worden.

Na de onderzoeken kan het Proof-of-Concept worden gemaakt. Dit Proof-of-Concept bevat een technisch ontwerp waarbij de applicatie gebruik maakt van een MongoDB database en een ontwikkelde knowledge engine waarbij er gebruik wordt gemaakt van een MongoDB database.

Aan de hand van het Proof-of-Concept kunnen de resultaten uit de gedane onderzoeken worden geëvalueerd.

Als er voldoende tijd is zal er ook onderzoek gedaan worden naar de schaalbaarheid wanneer MongoDB gebruikt wordt. Vragen die beantwoord kunnen worden zijn onder andere of MongoDB invloed heeft op het aantal mogelijke concurrent gebruikers of de schaalbaarheid. Ook zal er bekeken kunnen worden hoe er om gegaan moet worden met Eventual Consistency.

Na aanleiding van de hierboven geformuleerde doelstellingen kunnen de volgende vragen worden gesteld:

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

Om deze hoofdvraag te beantwoorden zullen de volgende vragen aan bod komen:

- Welke invloed heeft MongoDB op het technisch ontwerp van knowNow? **(deskresearch)**
- 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? **(deskresearch & experimenten)**
- Wat is de uitbreidbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB, kunnen er later nieuwe REST endpoints worden toegevoegd? **(deskresearch & experimenten)**
- Wat is de schaalbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB, hoe kan MongoDB worden opgeschaald wanneer het aantal gebruikers toeneemt? **(deskresearch & experimenten)**
    - Wat is er nodig 200.000+ gebruikers te bedienen?
    - Hoe moet er worden omgegaan met eventual consistency, hoe lang mag het duren voordat de database consistent is, hoe weet de database dat er een update moet plaatsvinden?
- Wat is de invloed op performance bij gebruik van MongoDB tegenover een relationele database? **(experimenten)**
- Hoe kan de MongoDB database gevuld worden en in sync worden gehouden met een Relationele Database? **(deskresearch & experimenteren)**

1.2.4 Resultaat

Na het uitvoeren van de afstudeeropdracht zullen er conclusies zijn geschreven over de onderzoeken. Ook zal er een werkend Proof-of-Concept waarin MongoDB wordt gebruikt zijn opgeleverd. Dit Proof-of-Concept bevat het technisch ontwerp van de knowledge engine van knowNow en een ontwikkelde knowledge engine van knowNow.

1.2.5 Effect

Na het succesvol afronden van de hierboven geformuleerde doestellingen heeft Info Support een beter beeld van wat de consequenties zijn voor het technisch ontwerp van knowNow indien Microsoft SQL Server vervangen zal worden door MongoDB.

Op basis van het onderzoek en Proof-of-Concept zou Info Support mogelijk kunnen bepalen om MongoDB ook daadwerkelijk in te zetten.

1.3 Afbakening

1.3.1 Binnen scope

De afstudeeropdracht zal de onderzoeksvragen zoals beschreven in 1.2, Doelstellingen beantwoorden en er zal een proof of concept worden ontwikkeld van de Knowledge engine service.

1.3.2 Buiten scope

Code	Afbakening
BUS1	De afstudeeropdracht bevat geen toolselectie voor een NoSQL database; Info Support heeft zelf een toolselectie voor aanvang van de opdracht uitgevoerd. Door Info Support is toen voor MongoDB gekozen.
BUS2	SQL Server zal alleen worden vervangen door MongoDB voor de Knowledge engine service.
BUS3	Het in productie nemen van het gemaakte Proof-of-Concept valt buiten de afstudeeropdracht.

1.4 Afhankelijkheden

Deze opdracht is afhankelijk van het knowNow project. De opdrachtnemer is afhankelijk van het knowNow team op het gebied van kennis voor knowNow.

1.5 Kwaliteitseisen

Code	Kwaliteitseis
KE1	**Beheerbaarheid Proof-of-Concept** De C# code van het Proof-of-Concept zal voorzien worden van commentaar.
KE2	**Kwaliteit code** De code van het Proof-of-Concept zal zich houden aan de C# Coding Convention van Microsoft [1] en eventuele binnen Info Support gemaakte conventies.
KE3	**Documenten gebruiken template Info Support** Alle in te leveren documenten (Microsoft Word & Microsoft Powerpoint) dienen de template van Info Support te gebruiken.
KE4	**De opdracht dient op HBO niveau te zijn** De opdracht dient op HBO niveau uitgevoerd te worden. Er dient in ieder geval voldaan te worden aan de volgende beroepstaken[5]: KE4.1. Ontwerpen, bouwen en bevragen van een database (2.2, Niveau 4) KE4.2. Bouwen applicatie (3.3, Niveau 3) KE4.3. Uitvoeren van en rapporten over het testproces (3.5, Niveau 3) KE4.5. Ontwerpen systeemarchitectuur (3.1, Niveau 3)
KE5	**Documenten in correct Nederlands** Alle documenten dienen in correct Nederlands worden ingeleverd
KE6	**Valide[6] en betrouwbare[7] onderzoeken** De uit te voeren onderzoeken dienen valide(vrij van systematische fouten) te zijn en betrouwbaar (vrij van toevallige fouten).

1.6 Uitgangspunten

Voor uitvoering van de in dit plan van aanpak beschreven delen zijn de onderstaande uitgangspunten van toepassing:

1. Het Proof-of-Concept zal worden ontwikkeld in C# .NET omdat knowNow gebruik maakt van C# .NET en dit dus beter aansluit;
2. De afstudeeropdracht zal bij Info Support in Veenendaal en Zoetermeer worden uitgevoerd;

1.7 Randvoorwaarden

Aan uitvoering van de in dit plan van aanpak beschreven delen zijn de volgende randvoorwaarden gesteld:

1. De uitvoering van de afstudeeropdracht dient plaats te vinden bij Info Support;
2. Het onderzoek dient te voldoen aan de eisen/wensen van de opdrachtgever;
3. Het onderzoek dient te voldoen aan de eisen/wensen van de Hogeschool;
4. De event bus is beschikbaar en kan door de afstudeerder worden gebruikt voor de opdracht;
5. De opdrachtnemer dient een eigen werkplek te hebben;
6. De opdrachtnemer dient toegang te hebben tot de code van knowNow;
7. De opdrachtnemer dient een Windows workstation te hebben waarop Visual Studio, Git, Microsoft Office en andere applicaties noodzakelijk voor de uitvoering van de afstudeeropdracht geïnstalleerd kan worden;
8. De opdrachtnemer kan informatie over knowNow bij het knowNow team vergaren;
9. De opdrachtnemer dient een eigen account voor Microsoft Team Foundation Server te hebben.

2. Risicomanagement

Voor uitvoering van de in dit plan van aanpak beschreven delen zijn de onderstaande risico's onderkend. Bij de risico's zijn de oorzaken en bijbehorende maatregelen ter beheersing opgenomen.

Nr.	Risico omschrijving				
	Oorzaak	Maatregel	P / S	Wie	Wanneer
1	**De opdrachtnemer heeft niet voldoende kennis van de te gebruiken technologieën**				
	Onvoldoende kennis	Het volgen van cursussen in die technieken	P	Opdrachtnemer	29-4-2015 t/m 8-5-2015
	Onvoldoende kennis	Hulp vragen aan technisch begeleider	P	Opdrachtnemer	Voortdurend
2	**De opdrachtnemer is voor langere tijd afwezig**				
	Voor langere tijd ziek	Er wordt een periode gereserveerd voor eventuele uitloop	S	Opdrachtnemer	Indien de opdrachtnemer langer dan een week ziek is
	Werkt voor langere tijd niet op locatie	Er wordt gekeken naar andere communicatie methoden zoals mail, telefoon of Skype.	S	Opdrachtnemer en belanghebbenden	Indien de opdrachtnemer langer dan een week niet op locatie kan werken
3	**Project is niet af te ronden binnen gestelde tijd**				
	Een of meerdere onderdelen zijn niet op tijd opleverbaar	D.m.v. scrum zorgen dat de belangrijkste delen afkomen	P/S	Opdrachtnemer	Wanneer er product backlog items niet af komen
4	**Er kan niet succesvol worden afgestudeerd**				
	Het verslag is van onvoldoende niveau	Het niveau van het verslag verhogen	S	Opdrachtnemer	Indien opdrachtnemer of belanghebbenden dit kenbaar maken
	Een of meerdere beroepstaken zijn van onvoldoende niveau	Opdracht aanpassen in overleg met opdrachtgever & de Haagse Hogeschool zodat alle beroepstaken van voldoende niveau zijn	S	Opdrachtnemer	Indien opdrachtnemer of belanghebbenden dit kenbaar maken
	Het verslag is van onvoldoende niveau	Het niveau van het verslag verhogen	P	Opdrachtnemer	Regelmatig feedback vragen aan belanghebbenden

5	**Onduidelijkheid over de opdracht**				
	Er is niet voldoende gevraagd aan opdrachtgever	Afspraak inplannen met opdrachtgever	S	Opdrachtnemer/Opdrachtgeer	Wanneer (een deel van) de opdracht onduidelijk is
	Er is onduidelijkheid wat er precies gewenst is	Afspraak inplannen met opdrachtgever	S	Opdrachtnemer/Opdrachtgeer	Wanneer (een deel van) de opdracht onduidelijk is
	Er worden verkeerdere dingen gemaakt	Regelmatig feedback vragen van het knowNow team	P	Opdrachtnemer/Opdrachtgever/knowNow team	Twee wekelijks
6	**Werkvertraging door afwezigheid opdrachtgever**				
	Afwezigheid van opdrachtgever	Afspraak maken met opdrachtgever wanneer weer aanwezig	S	Opdrachtnemer/Opdrachtgeer	Indien de opdrachtgever langer dan een week afwezig is
	Werkt voor langere tijd niet op locatie	Er wordt gekeken naar andere communicatie methoden zoals mail, telefoon of Skype.	S	Opdrachtnemer/Opdrachtgeer	Indien de opdrachtgever langer dan een week afwezig is
7	**De opdrachtnemer voert de opdracht onjuist uit**				
	De opdrachtnemer heeft de opdracht verkeerd begrepen	Regelmatig feedback vragen van het knowNow team	P	Opdrachtnemer/Opdrachtgever/knowNow team	Twee wekelijks
	De opdrachtnemer heeft de opdracht verkeerd begrepen	Uitvoering opdracht aanpassen	S	Opdrachtnemer	Indien gewenst door belanghebbenden

P/S = Preventief of Schadebeperkend

3. Aanpak

3.1.1 Scrum

Voor het uitvoeren van de afstudeeropdracht is gekozen voor de ontwikkelmethodiek Scrum. Met Scrum is het, in tegenstelling tot de watervalmethoden, mogelijk om iteratief aan de opdracht te werken. Hierdoor kan er iteratief aan het Proof-of-Concept worden gewerkt en kunnen er later nog wijzigingen worden geïntroduceerd en worden opgepakt. Hierdoor kan sneller worden ingespeeld op bevindingen.

3.1.2 Sprint duur

Elke sprint duurt 2 weken. Deze duur sluit goed aan bij de tweewekelijkse gespreken met de opdrachtgever. Daarnaast zal een korte sprintduur ervoor zorgen dat de hoeveelheid werk per sprint te klein wordt en zal een langere sprint ervoor zorgen dat er te weinig sprints zijn waardoor er minder iteraties zullen ontstaan.

3.1.3 Aanpassingen Scrum

Omdat deze afstudeeropdracht door één persoon zal worden uitgevoerd zullen een paar aspecten van Scrum worden aangepast, waardoor niet alles meer helemaal volgens de Scrum standaard verloopt. Ook wordt het onderzoek uitgevoerd met Scrum.

Daily Standup: De Daily Standups worden op woensdag en vrijdag uitgevoerd met het hele knowNow team. Hier kan de afstudeerder duidelijk maken waar hij mee bezig is en waar de afstudeerder tegen aangelopen is tijdens de uitvoering van de afstudeeropdracht.

De Daily standup zal dus niet dagelijks worden uitgevoerd. Dit komt omdat de opdrachtnemer op maandag, dinsdag en donderdag in Zoetermeer zit terwijl de rest van het team in Veenendaal zit. Daarnaast is de opdrachtgever ook alleen op woensdag en vrijdag aanwezig in Veenendaal. Omdat het niet nuttig is om een daily standup met één persoon te houden, omdat je juist aan andere moet laten weten wat je gedaan hebt en gaat doen. Daarom is besloten de daily standup alleen op woensdag en vrijdag te houden.

Retrospectives: De Scrum's retrospectives zullen aan het eind van elke sprint (2 wekelijks) worden uitgevoerd met de opdrachtnemer en opdrachtgever. Indien gewenst kunnen hier ook de technisch begeleider en procesbegeleider bij aanschuiven.

Scrum master: Er zal voor deze afstudeeropdracht geen Scrum master zijn. De taken van de Scrum master worden uitgevoerd door de opdrachtnemer.

Onderzoek met Scrum: Scrum is een ontwikkelmethode. Toch is er gekozen om Scrum ook te gebruiken voor het onderzoek. De Product Owner 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 Product Owner tijdens het onderzoek ook andere aspecten onderzocht wil hebben. In dit document is een grovere opzet van het onderzoek gemaakt. 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.

3.1.4 Scrum rollen

Er zijn binnen deze afstudeeropdracht 2 Scrum rollen:

Rol	Uitgevoerd door
Product Owner	Joop Snijder (Opdrachtgever)
Team	Tom Keim (Opdrachtnemer)

3.2 Op te leveren producten

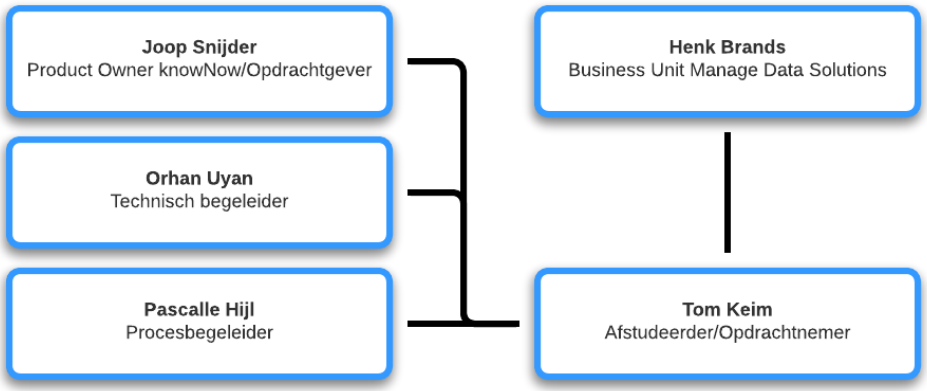
De op te leveren producten zijn als volgt:

- Plan van Aanpak;
- Onderzoekrapportage met daarin:
  - het onderzoek naar de invloed van MongoDB op het technisch ontwerp;
  - het onderzoek naar welke invloed de gebruikte interface (REST endpoints) heeft op de wijze van data opslag in MongoDB;
  - het onderzoek naar de uitbreidbaarheid van knowNow bij het gebruik van MongoDB ;
  - het onderzoek naar de schaalbaarheid van MongoDB;
  - het onderzoek naar de invloed van MongoDB op de performance van knowNow;
- Een technisch ontwerp van (een deel van) knowNow waarbij de traditionele database is vervangen door MongoDB;
- Een Proof-of-Concept ontwerp van (een deel van) knowNow waarbij de traditionele database is vervangen door MongoDB (inclusief Functioneel Ontwerp en Testrapportage);

De op te leveren producten worden onderverdeeld in verschillende sprints. Zie hiervoor Bijlage 1.

4. Beheeraspecten

4.1 Organisatie



Organisatiediagram

Naam	Rol	Omschrijving rol
Tom Keim	Afstudeerder / Opdrachtnemer	Voert de opdracht uit
Joop Snijder	Product Owner knowNow / Opdrachtgever	Is de opdrachtgever voor deze opdracht.
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.
Henk Brands	Business Unit Manager	De Business Unit Manager is de manager van de Business Unit en zal de voortgang in de gaten houden.
Gerard Mijnarends	1e examinator	De 1e examinator zal de afstudeerder adviseren tijdens de afstudeerstage.
Rianne Bechet-Tjoonk	2e examinator	De 2e examinator zal de afstudeerder beoordelen op de tussentijdse momenten en na het afstudeertraject.
Wiljag Denekamp	Architect knowNow	Als architect van knowNow kan de opdrachtnemer bij Wiljag Denekamp terecht bij vragen die de rest van het knowNow team niet kunnen beantwoorden.

4.1.1 Informatie

De afstudeerder zal alle conceptuele documenten opsturen naar de opdrachtgever & technisch begeleider. Na goedkeuring zullen de definitieve documenten worden opgestuurd naar de procesbegeleider, opdrachtgever, business unit manager & technisch begeleider. De technisch begeleider & de opdrachtgever zullen zo snel mogelijk feedback geven op de documenten.

4.2 Tijd

4.2.1 Normstelling

Het project is ingedeeld in meerdere Scrum sprints van elk 2 weken. Zie hiervoor ook hoofdstuk 3. Een planning is opgenomen in Bijlage 1: Planning.

Er zal wekelijks een moment worden ingepland om aan de documentatie te kunnen werken.

4.2.2 Voortgangscontrole

Voortgangscontrole zal plaatsvinden door de tweewekelijkse gespreken met de opdrachtgever, de gesprekken met de procesbegeleider en de gesprekken met de technisch begeleider. Ook zullen er om de 6-8 weken gespreken plaatsvinden met de Business Unit Manager.

4.3 Middelen

4.3.1 Normstelling

Voor het uitvoeren van de afstudeeropdracht zijn een aantal middelen nodig. Er is een werkplek nodig op ten minste een locatie. Voor deze opdracht zal de afstudeerder werken op 2 locaties. In Veenendaal & Zoetermeer. Op beide locaties dient een werkende Windows machine klaar te staan. Vanuit Zoetermeer zal er via een Remote Desktop verbinding worden gemaakt met de machine in Veenendaal zodat er op de zelfde machine gewerkt kan blijven worden.

De Windows machine bevat Windows 8 en moet beschikking hebben tot Visual Studio, SQL Server, MongoDB en meer. De software kan zelf geïnstalleerd worden. De gebruiker is Local Admin op zijn systeem.

4.4 Kwaliteit

4.4.1 Normstelling

De documenten dienen in correct Nederlands te worden ingeleverd. Ook zullen de documenten in de huisstijl Info Support gemaakt moeten worden.

4.4.2 Voortgangscontrole

Alle documenten worden gecontroleerd met een spellingschecker en zullen daarnaast nog meermaals worden doorgelezen door de afstudeerder.

Ook de broncode van het Proof-of-Concept zal vaak opnieuw worden doorgenomen, gecontroleerd en getest.

4.5 Overlegvormen

Overleg	Deelnemers	Frequentie
Mailen van voortgangsverslag	Opdrachtgever, Technisch begeleider, docentbegeleider, proces begeleider en Business Unit Manager	Tweewekelijks
Gesprek met Business Unit Manager	Opdrachtnemer & Business Unit Manager	Om de 4 weken
Sprint retrospective (Voortgangsgesprek)	Opdrachtnemer, opdrachtgever & Procesbegeleider	Tweewekelijks (na einde sprint)
Sprint review	Opdrachtnemer & Opdrachtgever	Tweewekelijks (na einde sprint)
Bezoek 1e examinator	1e examinator en technisch begeleider.	Op 25% van het afstuderen
Daily Standup	Het gehele knowNow team	Elke woensdag & vrijdagochtend

4.6 Communicatie

Naam & Functie	Email
Tom Keim *Afstudeerder /Opdrachtnemer*	tomkeim94@gmail.com
Joop Snijder *Opdrachtgever*	joop.snijder@infosupport.com
Pascalle Hijl *Procesbegeleider*	pascalle.hijl@infosupport.com
Orhan Uyan *Technisch begeleider*	orhan.uyan@infosupport.com
Henk Brands *Business Unit Manager*	henk.brands@infosupport.com
Gerard Mijnarends *1e examinator (school)*	G.A.Mijnarends@hhs.nl
Rianne Bechet-Tjoonk *2e examinator*	H.G.J.Bechet-Tjoonk@hhs.nl
Wiljag Denekamp Architect knowNow	wiljag.denekamp@infosupport.com

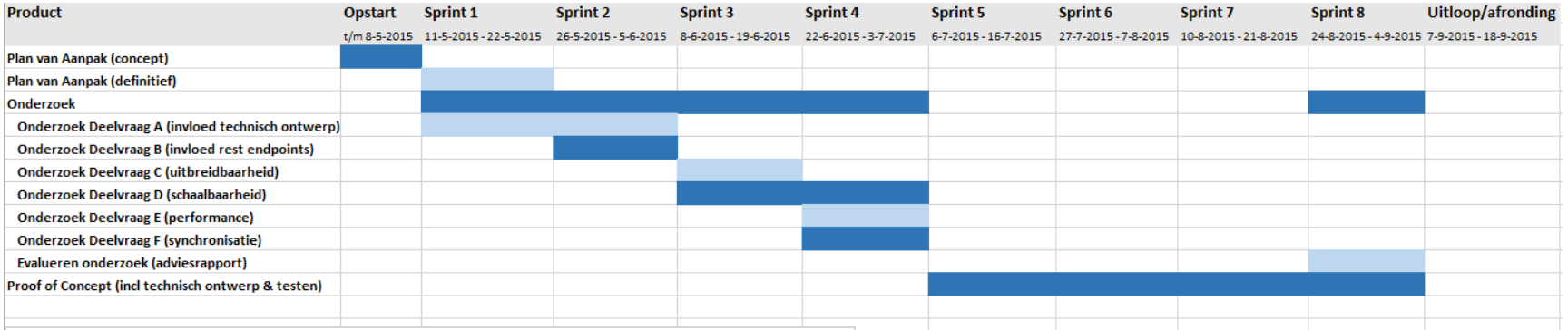
5. Oplevering

Zie paragraaf 3.2 en Bijlage 1 voor de op te leveren producten. Aan het einde van elke sprint zullen alle verrichtte onderzoeken en andere documenten beschikbaar worden gesteld voor alle belanghebbenden.

6. Referenties

[1]	Microsoft, “C# Coding Conventions (C# Programming Guide)” (Online) https://msdn.microsoft.com/en-us/library/ff926074.aspx
[2]	Techopedia, “Platform” (Online) http://www.techopedia.com/definition/3411/platform
[3]	Devbridge, “Benefits of NoSQL” https://www.devbridge.com/articles/benefits-nosql/
[4]	The agile admin, “What is DevOps? http://theagileadmin.com/what-is-devops/
[5]	Beroepstaken Informatica Haagse Hogeschool (Juni 2009) Beroepstaken_Informatica_1.1.pdf
[6]	Als je op zoek bent…, 4. Betrouwbaarheid, validiteit & bruikbaarheid, Validiteit https://infobronnen.wordpress.com/betrouwbaarheid-validiteit-bruikbaarheid/
[7]	Als je op zoek bent…, 4. Betrouwbaarheid, validiteit & bruikbaarheid, Betrouwbaarheid https://infobronnen.wordpress.com/betrouwbaarheid-validiteit-bruikbaarheid/

Bijlage 1: Planning



Onderzoek Deelvraag A: Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?

Onderzoek 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?

Onderzoek Deelvraag C: Wat is de uitbreidbaarheid van knowNow wanneer er gebruik wordt gemaakt van MongoDB, kunnen er later nieuwe REST endpoints worden toegevoegd?

Onderzoek 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?

* Wat is er nodig 200.000+ gebruikers te bedienen?
* Hoe moet er worden omgegaan met eventual consistency, hoe lang mag het duren voordat de database consistent is, hoe weet de database dat er een update moet plaatsvinden?

Onderzoek Deelvraag E: Wat is de invloed op performance bij gebruik van MongoDB tegenover een relationele database?

Onderzoek Deelvraag F: Hoe kan de MongoDB database gevuld worden en in sync worden gehouden met een Relationele Database?

Tussen Sprint 5 en 6 is een week vakantie ingepland (na toestemming). Naast deze week vakantie kunnen er nog 4 vrije dagen worden opgenomen

DE HAAGSE HOGESCHOOL

Info Support Solid Innovator

Bijlage B - Onderzoeksrapport

MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?

DE HAAGSE HOGESCHOOL

Info Support Solid Innovator

Onderzoeksrapport

Titel	Onderzoeksrapport
Project/Onderwerp	MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?
Versie	1.0
Status	Definitief
Datum	01-10-2015
Bestand	Onderzoeksrapport v0.7
Bedrijf	Info Support
Door	Tom Keim

Info Support B.V.
Kenniscentrum
Kruisboog 42, 3905 TG, Veenendaal,
De Smalle Zijde 39, 3903 LM Veenendaal,
Tel. +31 (0) 318 55 20 20 - Fax +31 (0) 318 55 23 55
Tel. +31 (0) 318 50 11 19 - Fax +31 (0) 318 51 83 59

Historie

Versie	Status	Datum	Auteur	Verandering
0.1	Concept	10-06-2015	Tom Keim	Creatie, samengevoegd met onderzoek technisch ontwerp & onderzoek uitbreidbaarheid
0.2	Concept	19-06-2015	Tom Keim	Feedback Orhan verwerkt + hoofdstukken met 'managementsamenvatting' geschreven
0.3	Concept	26-06-2015	Tom Keim	Feedback Joop verwerkt, schrijfstijl verbeterd
0.4	Concept	06-07-2015	Tom Keim	Feedback
0.5	Concept	27-07-2015	Tom Keim	Start onderzoek transactie
0.6	Concept	21-08-2015	Tom Keim	Oplevering Sprint 7
0.7	Concept	24-08-2015	Tom Keim	Huisstijl aangepast
0.8	Concept	08-09-2015	Tom Keim	Feedback verwerkt
1.0	Definitief	01-10-2015	Tom Keim	Definitieve versie

Distributie

Versie	Status	Datum	Aan
0.1	Concept	15-06-2015	Orhan Uyan
0.2	Concept	19-06-2015	Joop Snijder
0.3	Concept	26-06-2015	Joop Snijder, Orhan Uyan
0.4	Concept	02-07-2015	Joop Snijder
0.6	Concept	21-08-2015	Joop Snijder, Orhan Uyan
0.7	Concept	25-08-2015	Oplevering 80% Tussentijdse beoordeling
1.0	Definitief	01-10-2015	Haagse Hogeschool, Joop Snijder

Inhoudsopgave

1. Inleiding

Dit onderzoek is uitgevoerd in opdracht van Info Support voor het afstudeerproject van de afstudeerder.

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.
Bij een NoSQL database is er geen vaste structuur. Elke rij in een tabel kan zijn eigen structuur hebben (schema-loos). In een SQL oplossing is dit per tabel vastgelegd. Er kan bij NoSQL in een document alle data opgeslagen

worden in een rij(document), in plaats van in 40 à 60 tabellen. Hiervoor hoeven er geen 20 verschillende query's uitgevoerd te worden maar staat alles op één plek. Daarnaast zijn NoSQL oplossingen zeer schaalbaar.

Het knowNow team wil voor de Knowledge Items gebruik gaan maken van MongoDB. MongoDB vraagt een andere manier van data modellering. MongoDB kan goed om gaan met data-opslag en met verschillen in de data. Maar of MongoDB ook een daadwerkelijk alternatief is voor SQL Server is nog onbekend.

Zo zijn de consequenties van het gebruik van MongoDB database op het technisch ontwerp en op het ophalen van de data in knowNow niet bekend. Ook is de uitbreidbaarheid van knowNow indien er gebruik wordt gemaakt van MongoDB onbekend en is niet bekend of de datastructuur van de gegevens aangepast dient te worden indien er een nieuwe functionaliteiten bijkomen.

Om deze vragen te kunnen beantwoorden is dit onderzoek opgesteld.

1.1 Doelgroep

Dit onderzoek is bedoeld voor de Product Owner van knowNow, Joop Snijder en overige mensen die aan knowNow werken. Ook is dit onderzoek onderdeel van een afstudeerproject en wordt gebruikt voor de beoordeling van de afstudeerder.

1.2 Hoofd- en deelvragen

Dit onderzoek geeft antwoord op de volgende centrale vraag:

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

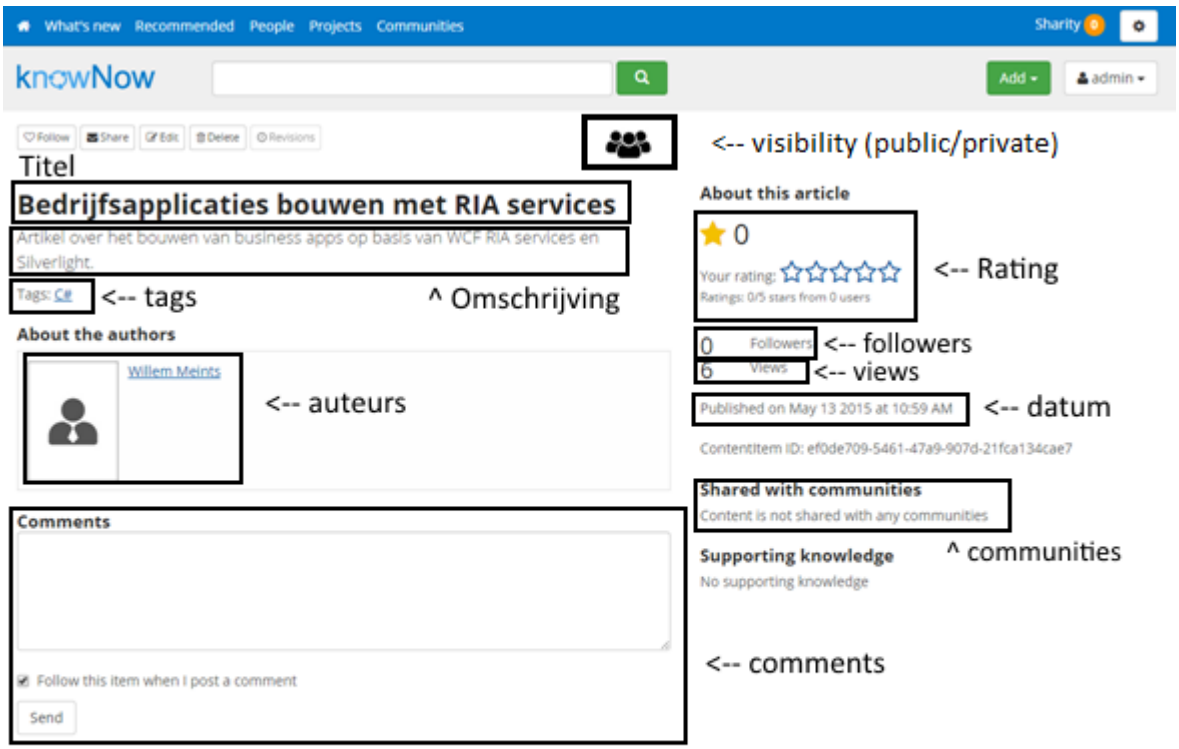
Om deze vraag te kunnen beantwoorden is deze opgesplitst in de volgende deelvragen:

#	Deelvraag	Type onderzoek
1	**Welke invloed heeft MongoDB op het technisch ontwerp van knowNow?**	**Deskresearch**
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**
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?	**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**
4	**Is het voor knowNow beter een hiërarchisch documentschema te hanteren of een relationeel documentschema?**	**Experimenteren**

In de komende hoofdstukken wordt elke deelvraag onderzocht en beschreven. Daarna wordt de hoofdvraag beantwoord.

2. Afbakening

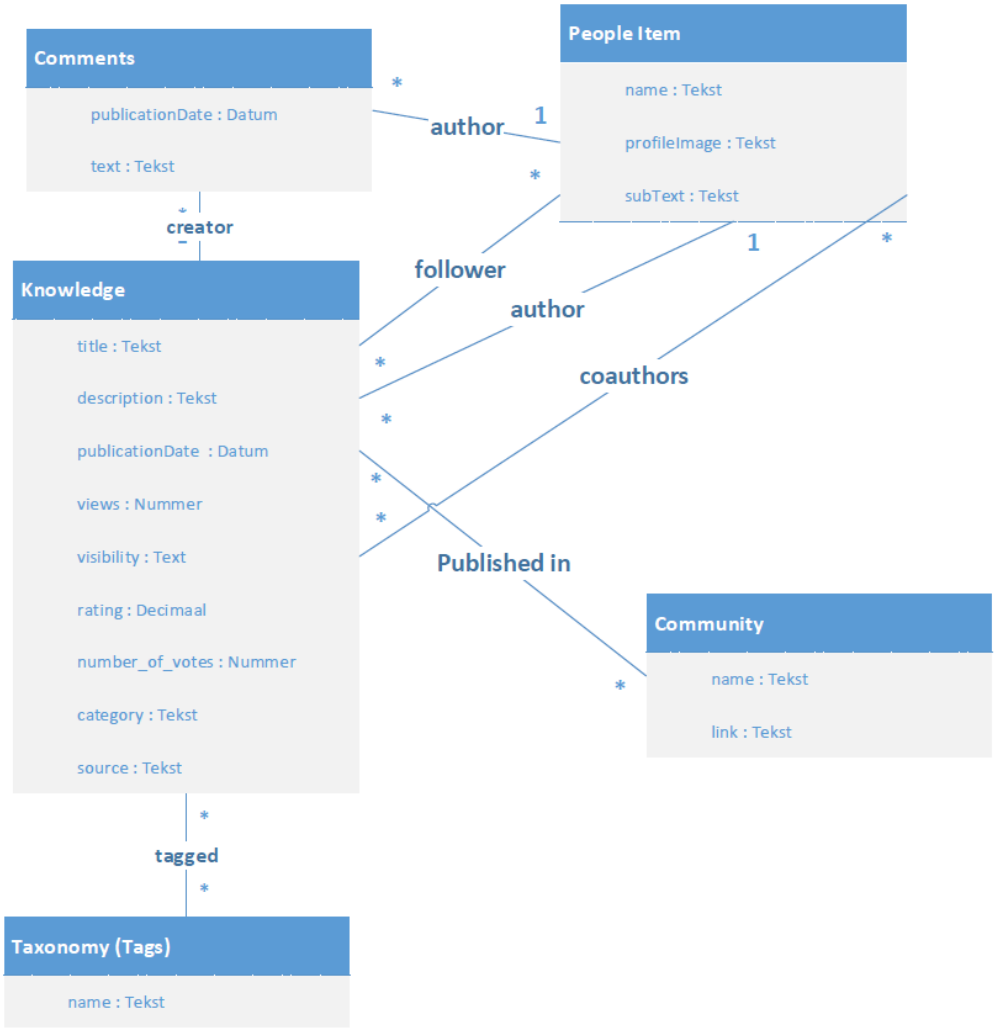
Dit onderzoek wordt uitgevoerd over de Knowledge Items van knowNow. Knowledge Items zijn een samenstelling van allerlei attributen die op een Knowledge Pagina staan. Onder een Knowledge Item valt onder andere een titel van het document, omschrijving en tags.



Figuur 2-1. Een Knowledge Item pagina

Het begrip Knowledge Items omvat een aantal attributen die voorkomen op de Knowledge Item pagina in knowNow (zie figuur 2-1). Het gaat hierbij om:

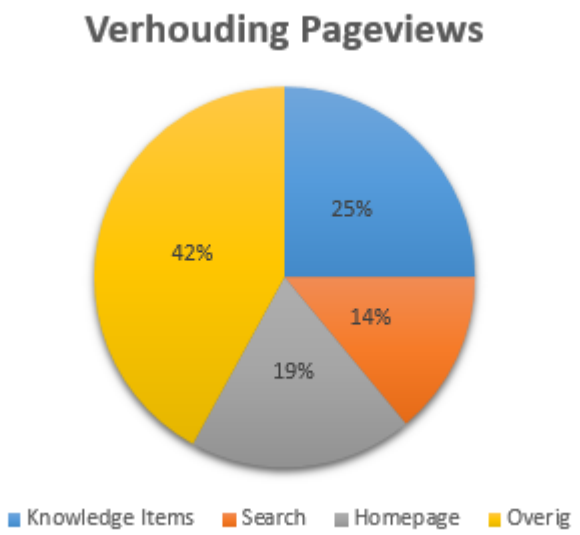
Attribuut	**Omschrijving**
Title	De titel van het knowledge item.
Description	De omschrijving van het knowledge item.
Rating	De beoordeling van het knowledge item.
Comments	Reacties op het knowledge item met daarin de tekst van de reactie, de schrijver van de reactie en de datum van de reactie.
Tags	Tags van het knowledge item. Tags zijn trefwoorden uit het Knowledge Item. Op basis van de tags kan ook worden gezocht naar artikelen die diezelfde tag bevatten.
Authors	Een Knowledge item wordt geschreven door een auteur. De gebruiker kan zelf opgeven wie de Auteurs en Medeauteurs (Co-authors) van het Knowledge Item zijn.
Followers	Het aantal mensen dat een knowledge item volgt. Wanneer een gebruiker een knowledge item volgt, krijgt deze gebruiker updates wanneer het artikel wordt gewijzigd.
Communities	Communities waar het Knowledge Item onder valt. Communities kunnen bijvoorbeeld afdelingen binnen een bedrijf zijn of interessegebieden.
Views	Hoeveel mensen het Knowledge item hebben bekeken.
Date	De publicatiedatum van het Knowledge Item.
Visibility	Public of Private. Bij Public is het item voor iedere gebruiker zichtbaar, bij private is het item alleen zichtbaar voor de gebruikers.
Categorieën	De categorieën waaronder het Knowledge Item valt, de huidige categorieën zijn als volgt: • Article • Concept • Practice • Presentation • Sample • Standard & Guidline • Template • Tool De categorieën zijn configureerbaar in de applicatie.



Figuur 2-2. De functionele database diagram van de Knowledge Items

In Figuur 2-2 staat een functioneel database diagram van de Knowledge Items. In dit diagram zijn de multipliciteiten aangeduid met de cijfers of een '1.*', wat staat voor één of meer.

Naast de Knowledge Items zijn er nog andere soorten pagina's op de knowNow site. In Google Analytics[1] is gekeken naar de verhouding van pageviews. Een pageview vindt plaats wanneer een gebruiker een pagina bezoekt. In Google Analytics is de onderstaande verhouding gevonden.



Figuur 2-3. De verhouding van pageviews tussen de verschillende delen van de site

Het onderzoek wordt afgebakend op de Knowledge Items vanwege het in verhouding hoge aantal items. Wanneer MongoDB voor de Knowledge Item goed blijkt te werken, is het mogelijk MongoDB ook voor de andere soorten items in te zetten zijn. Een andere reden voor de afbakening op Knowledge Items zijn de vaak grotere documenten die opgehaald moeten worden. Een Knowledge Item kan lange teksten bevatten.

Daarnaast is het een combinatie van gestructureerde en ongestructureerde data. Met de ongestructureerde data wordt de omschrijving bedoeld, dit is niet op een vaste manier opgeslagen, met de gestructureerde data worden de velden als datum en aantal followers bedoeld.

Volgens SQL Server Profiler[2] worden er 91 queries uitgevoerd om een Knowledge Item pagina op te vragen

Voor het type 'People' is dit aantal volgens SQL Server Profiler al minder, namelijk 33. 'People' zijn de profielen van de gebruikers van knowNow. Het potentieel te verlagen queries bij het gebruik van MongoDB is daarmee bij Knowledge Items het hoogst.

Het voordeel van het verlagen van het aantal uit te voeren queries is dat alle data dan meer op een plek staat. De data hoeft niet eerst bij elkaar worden gezocht door bijvoorbeeld joins.

[1] Een web applicatie van Google waar statistieken over een website te zien zijn.
[2] Een applicatie van Microsoft waarin alle uitgevoerde queries te zien zijn.

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

3.1 Inleiding

De architectuur van knowNow moet waarschijnlijk aangepast worden om MongoDB te kunnen integreren. In het document “Technisch ontwerp van oorspronkelijke situatie knowNow” (Bijlage C) is beschreven dat knowNow op dit moment gebruik maakt van Orchard. Maar de impact van het gebruik van MongoDB op de huidige applicatie van knowNow is onbekend. Met dit onderzoek wordt getracht deze impact in kaart te brengen.

Dit onderzoek gaat uit dat de architectuur zoals beschreven in het document 'Technisch ontwerp van oorspronkelijke situatie’ bij de lezer bekend is. Dit onderzoek gebruikt dat document als basis, er worden enkele begrippen uit dit document gebruikt.

De deelvraag is opgesplitst in de volgende vragen:

- Wat is de best te implementeren architectuur voor knowNow bij het gebruik van MongoDB?
- Welke invloed heeft MongoDB op de uitbreidbaarheid van knowNow?
- Welke invloed heeft MongoDB op de security van knowNow?
- Welke invloed heeft MongoDB op de schaalbaarheid van knowNow?

3.2 Architectuur

3.2.1 Inleiding

Dit hoofdstuk geeft antwoord op de volgende vraag:

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

Er zijn twee alternatieven die dit hoofdstuk beschrijft.

Door goed te kijken naar de huidige architectuur van knowNow en de wensen van de Product Owner in ogenschouw te nemen zijn er 3 mogelijke oplossingen geïnventariseerd. De 3 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.

In dit hoofdstuk worden de volgende architecturen toegelicht:

- **Het inpassen van MongoDB in Orchard**
  - **Met het behouden van de huidige RDBMS**
    Bij deze architectuur wordt de huidige RDBMS behouden en wordt deze gebruikt als database waar de mutaties op worden uitgevoerd. MongoDB wordt gebruikt als leesdatabase. Deze twee databases worden consistent gehouden.
  - **Met het geheel vervangen van de huidige RDBMS**
    Bij deze architectuur wordt Orchard zo aangepast dat deze voor het Knowledge Item gedeelte volledig gebruik maakt van MongoDB, voor zowel het lezen als de mutaties.
- **De Knowledge Service (Microservice architectuur)**
  De Knowledge Service is een service die totaal losstaat van Orchard en dit totaal vervangt. Het enige doel van de Knowledge Service zijn de Knowledge Items.

3.2.2 Inpassen in Orchard

Een alternatief is om de huidige Orchard applicatie voor de Knowledge Items aan te passen, zodat er gebruik wordt gemaakt van een MongoDB database. Hiervoor moeten er aanpassingen gemaakt worden aan Orchard, dit kan met modules (uitbreidingen op Orchard). Mogelijk is niet alles via deze modules te realiseren.

Mogelijk moeten aanpassingen buiten modules om gedaan worden. Indien voor deze architectuur gekozen wordt, moet eerst onderzoek gedaan worden naar hoe MongoDB in te passen is binnen Orchard.

Er zijn 2 mogelijke varianten besproken voor deze architectuur:

1. Het geheel vervangen van de relationele database.
2. MongoDB implementeren naast de huidige databases.

Beide varianten worden hieronder besproken.

3.2.2.1 Behouden van de RDBMS

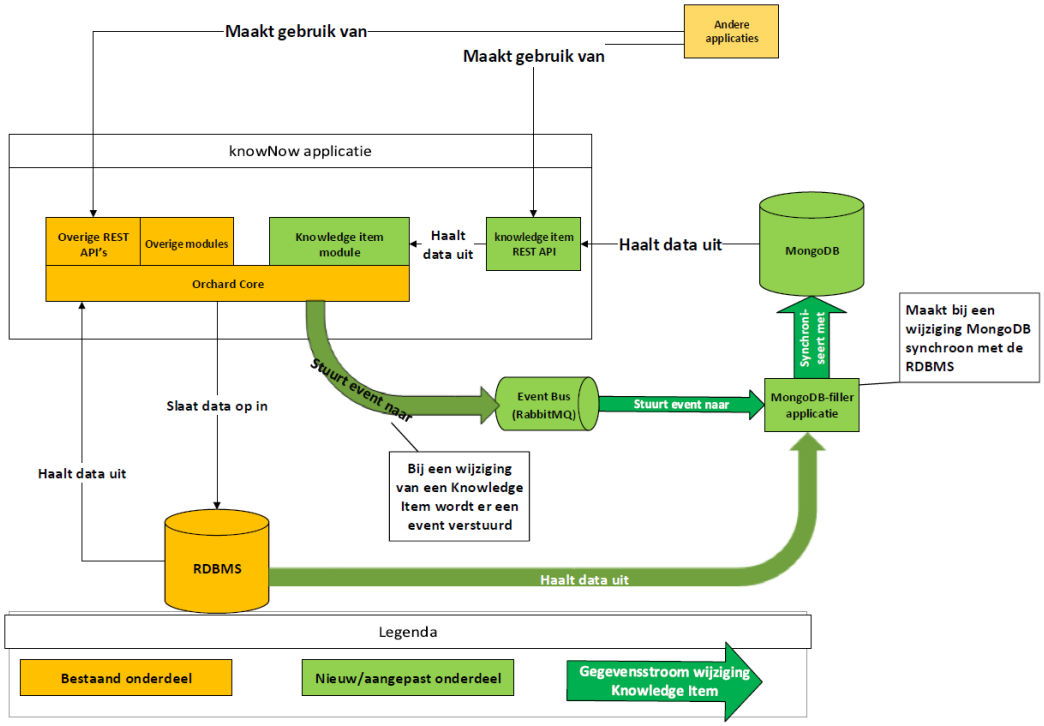
Een variant van het inpassen in Orchard is het behouden van de RDBMS naast MongoDB.

Bij het behouden van de Relationele Database Management Server naast MongoDB moeten beide databases consistent met elkaar blijven. Het mag niet zo zijn dat een record wel in de RDBMS is te vinden, maar niet in de MongoDB database. Wanneer dit voorkomt kan dit tot een inconsistentie leiden.

Een voorbeeld van deze inconsistentie is wanneer een gebruiker een nieuw Knowledge Item toevoegt. Deze wordt eerst in de RDBMS opgeslagen en wordt daarna (door een onbekend probleem) niet opgeslagen in de MongoDB database. Hierdoor zijn deze 2 databases inconsistent geraakt. Wanneer de gebruiker nu opzoek gaat naar zijn zojuist toegevoegde Knowledge Item in de MongoDB database kan hij deze niet terug vinden.

Een mogelijke oplossing is het ontwikkelen van een nieuwe applicatie die aan een Event Bus wordt gekoppeld. Een Event Bus is een middel waarmee applicaties (asynchroon) met elkaar kunnen communiceren. Een applicatie 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 wat er veranderd is, wanneer het veranderd is en door wie de transactie is gemaakt. Door het gebruik van de Event Bus kan ook na de request van de gebruiker de data worden verwerkt en hoeft de gebruiker minder lang te wachten op een response (er wordt niet gewacht op een antwoord van de Event Bus).

Het doel van deze extra applicatie is het consistent houden van de MongoDB database met de RDBMS. Wanneer een item wordt toegevoegd of verwijderd in knowNow, wordt een event afgevuurd. De applicatie pakt dit event op en zorgt dat het nieuwe artikel ook in de MongoDB database terecht komt.



Figuur 3-1. Mogelijke architectuur waarbij RBDMS naast MongoDB blijft bestaan.

Bij het lezen van data haalt de applicatie de data uit MongoDB. Wanneer data toegevoegd of gewijzigd wordt, wordt dit gedaan op de RDBMS. De MongoDB-filler applicatie houdt MongoDB en de RDBMS consistent. De MongoDB-filler handelt nadat deze een ‘event’ heeft ontvangen. Een ‘event’ wordt naar de MongoDB-filler gestuurd wanneer er een Knowledge Item is toegevoegd of gewijzigd.

Naast het consistent maken van MongoDB na het plaatsvinden van een event moet er ook onderzocht moeten worden hoe de database gevuld moet worden wanneer deze leeg is, bijvoorbeeld na een crash van de database.

Wanneer er voor deze mogelijkheid gekozen wordt is het daarom van belang verder te onderzoeken hoe beide databases consistent (in sync) worden gehouden.

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.

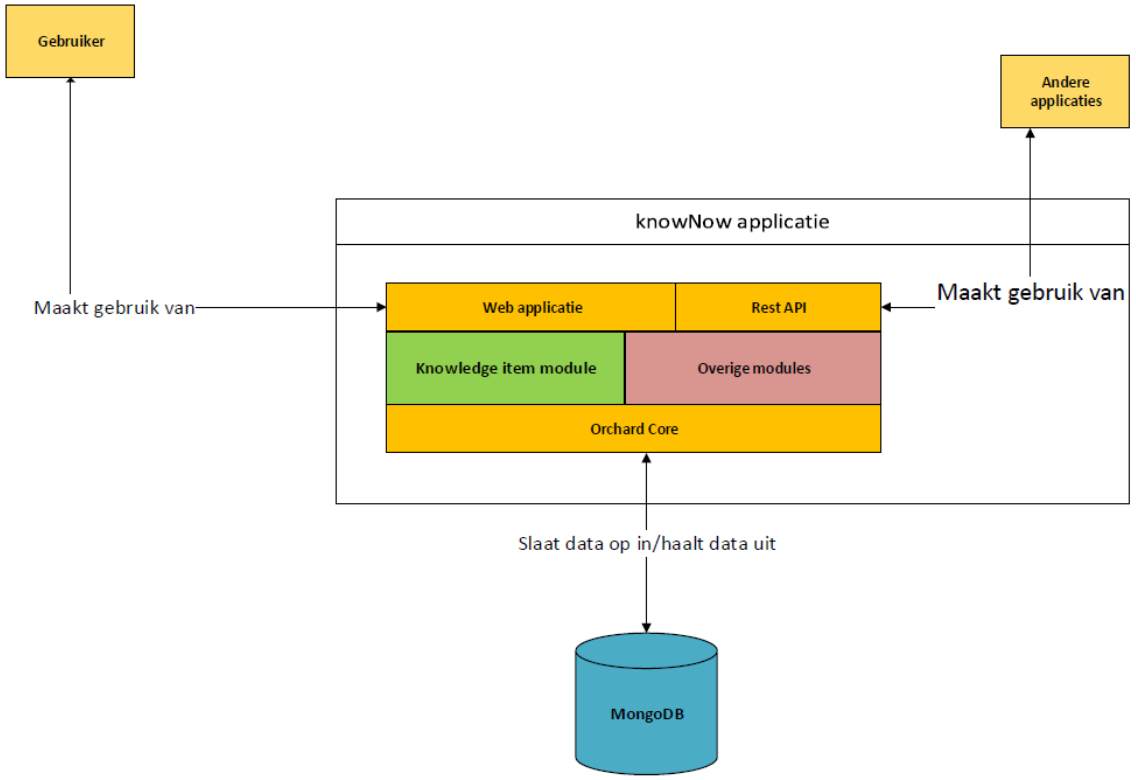
3.2.2.2 Geheel vervangen RDBMS

De andere variant van het inpassen in Orchard is het geheel vervangen van de RDBMS door MongoDB.

Bij deze variant hoeven er geen meerdere databases consistent gehouden te worden omdat MongoDB de enige database is.

Deze oplossing brengt echter ook andere moeilijkheden met zich mee. Naast aanpassingen aan de frontend moet er meer aangepast moeten worden, zoals:

- De verwerking van alle toevoegings- en wijzigingsformulieren voor de Knowledge Items, deze kunnen niet meer schrijven naar de RDBMS.
- De backend van Orchard, in het vorige oplossing kon die blijven schrijven naar de relationele database. In deze oplossing bestaat de RDBMS niet meer waardoor de backend opgebouwd moet worden om met MongoDB om te kunnen gaan.



Figuur 3-2. Voorbeeld waarbij MongoDB de relationele database vervangen heeft

Bij deze oplossing zowel de lees als schrijffunctionaliteit van de huidige applicatie aangepast worden, waardoor mogelijk voor zowel het lezen als het schrijven performancewinst te behalen is. Dit betekend logischerwijs dat er wel meer aanpassingen nodig zijn dan bij het deels vervangen van de RDBMS.

3.2.3 Knowledge Service (Microservice architectuur)

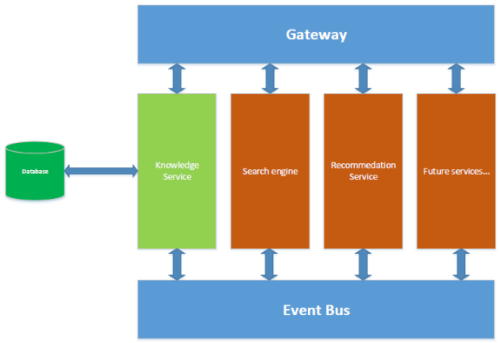
3.2.3.1 Inleiding

De huidige knowNow-applicatie is een monolithisch systeem. Dit betekent dat de functionaliteit van een deel van een applicatie niet kan worden hergebruikt zonder de hele applicatie te gebruiken[7].

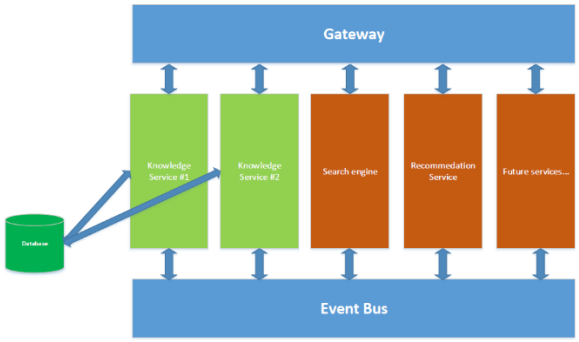
knowNow maakt al deels gebruik van scheiding van services[3] en wil in de toekomst helemaal overstappen op een Microservice architectuur. Bij een Microservice architectuur is een applicatie opgedeeld in meerdere kleine services die elk hun eigen verantwoordelijkheid hebben[A19]. Door het gebruik van Microservices 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. De services weten niet van elkaars bestaan en staan los van elkaar.

Dit is een van de belangrijkste voordelen van de microservice architectuur. Doordat ze niks van elkaar weten, kunnen ze ook gemakkelijk worden vervangen. Ze zijn niet aan elkaar gekoppeld. Communicatie tussen meerdere services gebeurt via een zogenaamde Event Bus.

Naast dat de koppeling tussen verschillende services laag is, is de schaalbaarheid ook een belangrijk voordeel. De verschillende services kunnen los van elkaar worden gedupliceerd. Hierdoor wordt niet de gehele knowNow-applicatie gedupliceerd, maar alleen de services waarvoor dit nodig is. Dit is zichtbaar in het voorbeeld hieronder. In Figuur 3-3 zie je een Knowledge Service (de groene rechthoek), in Figuur 3-4 staan 2 Knowledge Services (de 2 groene rechthoeken). Ze maken nog steeds gebruik van dezelfde database. De services zijn gedupliceerd.



Figuur 3-3. Er is één Knowledge Service



Figuur 3-4. Er zijn 2 Knowledge Services. Ze maken gebruik van dezelfde database

[3] Zie 4.6 van het Technisch ontwerp van oorspronkelijke situatie knowNow

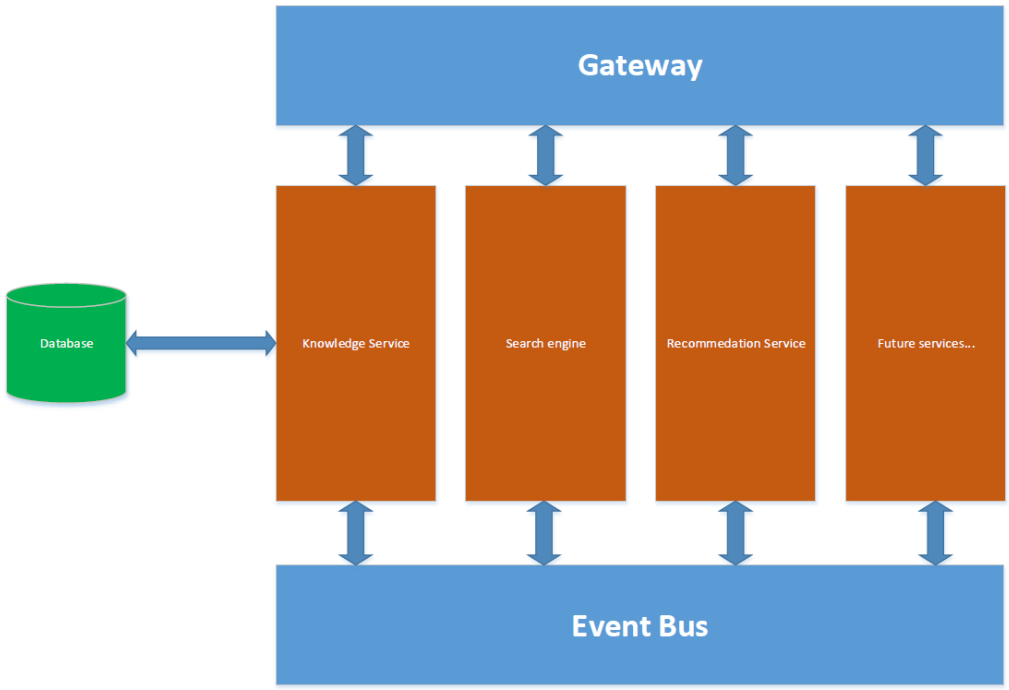
De Knowledge Item service is daarmee voorbereid op de toekomst, de gehele knowNow applicatie wordt beter schaalbaar. En de service past in het toekomstbeeld van de knowNow. De microservice architectuur is het toekomstbeeld van knowNow omdat:

- Het apart kunnen dupliceren van services zorgt voor een hogere schaalbaarheid; alleen de services waarvan nodig kunnen geschaald worden. Dit maakt de hele knowNow applicatie meer schaalbaar.
- Duidelijke verantwoordelijkheden; De services zijn zelf verantwoordelijk voor hun eigen taken. Dit maakt de knowNow applicatie beter onderhoudbaar.
- Losse koppeling; De services weten niet van elkaar bestaan ze communiceren met elkaar door middel van events die niet specifiek zijn gericht aan één service. Dit maakt de knowNow applicatie beter onderhoudbaar.

3.2.3.2 Architectuurontwerp

De Knowledge Service wordt een losstaande service die kan worden bevraagd via een REST API. De Knowledge Service bevat geen webinterface. Een webinterface kan via de REST API communiceren maar valt niet onder de Knowledge Service zelf.

Via de gateway kan er gecommuniceerd worden met de verschillende services. Dit kan via bijvoorbeeld een REST API plaatsvinden.



Figuur 3-5. Voorbeeld van Werking Services met Event Bus

Alle services zijn losstaand van elkaar en weten niet van elkaars bestaan. Door middel van de event bus kunnen de services data uitwisselen. Een interessante gebeurtenis voor andere services wordt door de service doorgegeven aan de Event Bus.

Een service die is ingehaakt op de event bus leest de voor hem interessante event uit en handelen daarop.

Een voorbeeld is het schrijven van een nieuw Knowledge Item naar de Knowledge Service. De Knowledge Service handelt dit af en geeft hiervan een event af aan de event bus. In dit event staat wat voor soort event het is (een nieuw Knowledge Item) en wat er is veranderd (de data van het nieuwe toegevoegde Knowledge Item). De Search Engine weet dat wanneer er een nieuw Knowledge Item is toegevoegd dat hij dit moet indexeren.
Daarom 'luistert' de Search Engine op de Event Bus naar 'nieuw Knowledge Item'-events. De Search Engine neemt een Knowledge Item op wanneer zo'n event ingelezen is.

Omdat deze applicatie nieuw wordt gebouwd kan er een documentschema voor MongoDB ontworpen kunnen worden, specifiek voor de Knowledge Service.

Er wordt dan geen rekening gehouden met het huidige Orchard model en er hoeft geen frontend gebouwd te worden. Dit wordt voor deze oplossing geheel losgelaten en er wordt een nieuwe applicatie ontwikkeld. De Knowledge Service is te gebruiken via REST API's en maakt gebruik van de Event Bus voor het afgeven en luisteren naar events voor communicatie tussen services.

Deze architectuur heeft meerdere voordelen, waaronder:

- Het hoeft niet doorontwikkeld te worden op een bestaande applicatie. Hierdoor is er geen tijd nodig om tot in detail te leren hoe de Orchard applicatie werkt.
- Er is één database nodig: de MongoDB database.
- Toekomstgericht. Voldoet aan toekomstbeeld van het knowNow team (hoge schaalbaarheid, duidelijke verantwoordelijkheden, losse koppeling)

Het nadeel is dat de huidige frontend nog niet met deze te realiseren service om kan gaan.

3.2.4 Conclusie

In de tabel hieronder zijn de bevindingen in dit hoofdstuk nogmaals vermeld. Ze zijn onderverdeeld in voor- en nadelen voor een duidelijke vergelijking.

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. • 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
Microservices	• Documentschema hoeft niet geschikt te zijn voor Orchard en kan daarom speciaal voor de Knowledge Service worden ontwikkeld; • Hoge schaalbaarheid • Duidelijke verantwoordelijkheden • Losse koppeling • Toekomstgerichte oplossing • Zowel lezen en schrijven van items mogelijk sneller door gebruik van MongoDB.	• Geheel nieuw systeem, er is nog geen web applicatie(frontend) beschikbaar die met de Knowledge Service kan omgaan;

Aan de hand van de hierboven genoemde voor- en nadelen is er gekozen voor de Microservice architectuur. Het inpassen in Orchard brengt extra moeilijkheden met zich mee. Het is belangrijk om de twee databases consistent te houden. Daar moet extra onderzoek voor worden gedaan, het moet zeker zijn dat het consistent houden van de meerdere databases goed werkt. Ook is deze architectuur complexer dan die van de Knowledge Service. Het is meer een combinatie van een monolithisch systeem de Knowledge Service wat de architectuur niet eenvoudiger maakt.

Daarnaast moeten er twee databases beheerd worden, de SQL Server database en de MongoDB. Dit zorgt voor extra complexiteit en meer werklast bij het onderhouden van deze databases.

De MongoDB database kan specifiek voor de Knowledge Service ontworpen worden. Er hoeft geen rekening gehouden te worden met de Content Types/Content Parts van Orchard, daardoor bevat het documentschema alleen de puur noodzakelijke velden en tabellen.

Voordat er aan het Proof-Of-Concept wordt begonnen dient de gekozen architectuur verder uitgewerkt worden, dit wordt gedaan aan de hand van een functioneel en technisch ontwerp.

3.3 Security

In dit hoofdstuk wordt de security van MongoDB onderzocht. Er wordt antwoord gegeven op de volgende vraag:

"Welke invloed heeft MongoDB op de security van knowNow?"

MongoDB heeft meerdere mogelijkheden waarmee de security te verbeteren is. Zo kent MongoDB authenticatie, alhoewel dat standaard is uitgeschakeld. Daarnaast is het met autorisatie mogelijk om te bepalen welke gebruiker waar bij kan (tot op collectieniveau).

MongoDB is standaard voor iedereen te bereiken, maar kan zo ingesteld worden dat alleen vertrouwde bronnen MongoDB kunnen benaderen.

Voor authenticatie is SCRAM-SHA1 de beste optie. Dit is na het aanzetten van authenticatie ook de standaard optie. Autorisatie kan beter worden afgehandeld door de applicatie en niet door MongoDB. MongoDB moet alleen bereikbaar zijn voor de applicatie.

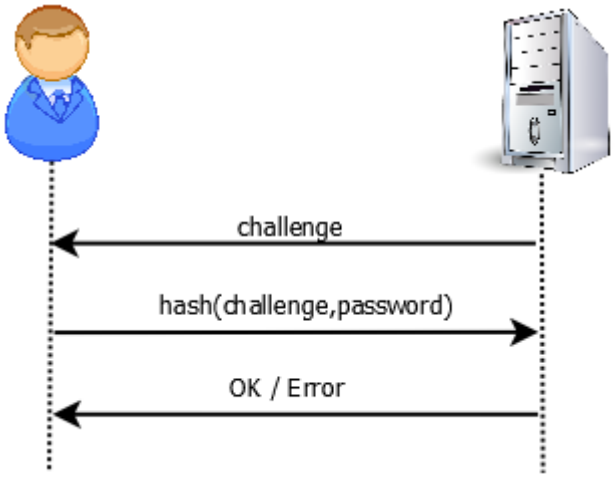
3.3.1 Aanpak

MongoDB zelf heeft meerdere security maatregelen die door de gebruiker ingesteld kunnen worden. Deze maatregelen worden genoemd in de documentatie van MongoDB die zij zelf op hun website beschikbaar hebben gemaakt. Deze maatregelen worden uitgezocht.

3.3.2 Authenticatie

Authenticatie is het identificeren van een gebruiker. Met authenticatie komt het systeem erachter of iemand die toegang tot een systeem wilt hebben ook echt de gebruiker is die hij zegt dat hij is. Dit kan door middel van een gebruikersnaam of wachtwoord maar er zijn ook andere authenticatie methoden.

Authenticatiemethode	**Omschrijving**
SCRAM-SHA1	SCRAM-SHA1*[A1]* is een Challenge-Response authenticatiemethode die vanaf MongoDB versie 3.0 standaard ingeschakeld wordt. Bij SCRAM-SHA1 logt de gebruiker in met een gebruikersnaam en wachtwoord. Die dan via een Challenge-Response wordt verzonden. Bij deze Challenge Response wordt het wachtwoord gehasht doorgegeven. In eerdere versies van MongoDB werd MONGODB-CR gebruikt. Ook dit was een Challenge-Response authenticatiemethode. MONGODB-CR is door MongoDB zelf ontwikkeld en is in versie 3.0 vervangen door de standaard SCRAM-SHA1.
X.509	Hierbij kan een gebruiker op een TSL/SSL verbinding inloggen met een X.509 certificaat. Er is dan geen gebruikersnaam en wachtwoord meer nodig.
Kerberos (alleen bij MongoDB enterprise)	Hierbij kan er gebruik worden gemaakt van een Kerberos authenticatie server. Een Kerberos Key Distribution Server geeft tickets af, deze tickets dienen daarna als authenticatiemiddel voor de gebruikers[A8].
LDAP (alleen bij MongoDB enterprise)	Met MongoDB is het via Lightweight Directory Access Protocol mogelijk om in te loggen met Microsoft Active Directory Services en andere soortgelijke diensten. LDAP voor MongoDB is alleen te gebruiken op Windows[A9].



Figuur 3-6. Een versimpelde weergave van SCRAM-SHA1
(bron: https://www.incibe.es/blogs/post/Security/SecurityBlog/Article_and_comments/password_based_authentication)

3.3.3 Functioneel & Data

MongoDB heeft naast authenticatie ook ondersteuning voor autorisatie. Bij authenticatie maakt de gebruiker bekend wie hij is, autorisatie bepaald wat de gebruiker kan doen.

Standaard staat autorisatie in MongoDB uit. Dit is aan te zetten door –auth toe te voegen aan het opstart commando van MongoDB of neer te zetten in een configuratie bestand.

MongoDB kent standaard een aantal rollen die aan gebruikers toe te wijzen zijn. Beheerder kunnen zelf ook rollen aanmaken, hier rechten aan toe kennen en hier gebruikers aan koppelen. Elke rol kan weer andere rechten hebben op andere collecties (Collection-Level Access Control[A3]) en database niveau. Per rol kan ingesteld worden of collecties ze mogen lezen, of ze mogen schrijven en of ze nieuwe collecties kunnen maken etc.

Autorisatie is niet mogelijk per document (row-level security).

3.3.3.1 Autorisatie data

Naast beveiliging van de server zelf moet er ook gezorgd voor worden dat gebruikers geen Knowledge Items kunnen ophalen die zij niet mogen zien. Hiermee autoriseer je gebruikers voor bepaalde data.

Knowledge Items kunnen als 'Private' worden toegevoegd. Deze Knowledge Items mogen dan alleen opgevraagd worden door een gebruiker die auteur is van het item. Een Knowledge Item kan ook aan een community worden toegevoegd. Dan mag het item door de leden van die community gelezen worden.

Het is in MongoDB alleen mogelijk om gebruikers te autoriseren op collectie niveau [A18]. Het is hierdoor niet mogelijk om door MongoDB te laten bepalen welke gebruiker van knowNow welke documenten binnen een collectie kan opvragen.

Maar zelfs mocht dit mogelijk zijn, is dit niet de gewenste manier. Dat betekend dat elke gebruiker van de knowNow applicatie ook eigen inloggegevens moet hebben voor MongoDB. Daarnaast moet MongoDB weten welke gebruiker welke Knowledge Items mag bekijken.

Wanneer de MongoDB database alleen benaderbaar is voor de applicatie is dat ook niet nodig. De gebruiker kan niet direct bij de MongoDB database. Het is makkelijker en verstandiger om de logica voor het bepalen wie welke items mag opvragen in de applicatie in te bouwen. Deze weet al wie de gebruiker is en welke items de gebruiker mag opvragen en welke niet.

3.3.4 Auditing

Met auditing wordt bekeken wat er plaats heeft gevonden. Auditing maakt het mogelijk om te kijken wie welke instellingen of schema's heeft van MongoDB[A4]. Hierdoor kan er wanneer noodzakelijk gekeken worden wie ergens voor verantwoordelijk is geweest en kan daarop actie worden ondernomen.

Het verschil tussen logging en auditing is dat er bij auditing wordt gekeken wie wat heeft gedaan, terwijl bij logging wordt bekeken wat er is gebeurt.

Om gebruik te maken van auditing dient er een licentie voor MongoDB Enterprise te worden afgenomen. Logging zit wel in de gratis versie van MongoDB.

3.3.5 Conclusie

Het is belangrijk om MongoDB zelf goed te beveiliging. Er moet voor worden gezorgd dat alleen de knowNow applicatie en het knowNow team bij MongoDB kan. Dit kan door middel van authenticatie en door er voor te zorgen dat de server niet van buiten te benaderen.

Als authenticatiemethode wordt SCRAM-SHA1 gebruikt. SCRAM-SHA1 staat aan wanneer er de authenticatie instelling aan staat in MongoDB.

De autorisatie van gebruikers wordt door de applicatie afgehandeld en niet door MongoDB. Dit omdat het in MongoDB niet mogelijk is om een gebruiker te autoriseren op een deel van een collectie, alleen op de collectie in zijn geheel.

Doordat MongoDB zelf te beveiligen is, heeft MongoDB geen invloed op de security van knowNow als het zoals hierboven omschreven is ingesteld. De knowNow applicatie authentiseert en autoriseert de gebruiker en bepaald dus wie welke data mag inzien. De applicatie dient zich alleen te authentiseren bij de MongoDB server, autoriseren is niet nodig. Dat is nu ook niet het geval bij de SQL Server installatie die door knowNow wordt gebruikt, wanneer de gebruiker van de database bij de database kan, kan deze ook bij alle data.

3.4 Uitbreidbaarheid

Managementsamenvatting

Dit hoofdstuk beschrijft de invloed van MongoDB op de uitbreidbaarheid van knowNow.

Uitbreidbaarheid is voor knowNow erg belangrijk. MongoDB is erg flexibel, het kent geen vast schema. Elk document in een collectie kan een ander schema hebben.

De consequenties op de uitbreidbaarheid voor knowNow zijn echter nog onbekend. Hier wordt later onderzoek naar gedaan.

3.4.1 Inleiding

In dit hoofdstuk wordt de uitbreidbaarheid van MongoDB voor knowNow onderzocht. Er wordt antwoord gegeven op de volgende vraag:

"Welke invloed heeft MongoDB op de uitbreidbaarheid van knowNow?"

De uitbreidbaarheid van knowNow is erg belangrijk en is ook zichtbaar in de architectuur van de knowNow applicatie. Zoals in het technisch ontwerp omschreven biedt Orchard ondersteuning voor migraties. Dit zorgt voor een grotere uitbreidbaarheid van de applicatie. Er dient bekeken te worden wat het gebruik van MongoDB voor invloed gaat hebben op de uitbreidbaarheid van knowNow..

3.4.2 Uitbreidbaarheid van MongoDB

Een collectie van MongoDB heeft geen vast schema. Elk document (rij) kan zijn eigen set fields (kolommen) hebben. MongoDB controleert niet of een document voldoet aan een opgegeven schema. Hierdoor zit een database in MongoDB niet vast aan een bepaald schema. Voor MongoDB is het daarom ook niet nodig om de definitie van het schema aan te passen.

Als data een andere structuur krijgt kan dit op meerdere manieren aangepakt worden:

1. De data voor nieuwe records volgens de nieuwe structuur opslaan en de oude documenten hetzelfde laten. De applicatie moet dan wel om kunnen gaan met verschillende structuren.
2. De data voor zowel nieuwe als oude records in de nieuwe structuur opslaan. De applicatie hoeft dan niet om te kunnen gaan met meerdere structuren.

3.4.3 Invloed MongoDB op uitbreidbaarheid knowNow

knowNow maakt op dit moment gebruik van de migraties om de schema's van de relationele database aan te passen. Dit zorgt ervoor dat uitbreidingen maar een keer ontwikkeld worden. Wanneer knowNow opnieuw geïnstalleerd wordt, worden alle migratiestappen weer doorlopen zodat elke installatie altijd hetzelfde documentschema heeft.

De definitie van het schema van MongoDB hoeft bij een migratie niet aangepast te worden, MongoDB heeft immers geen vast schema per document. Maar wanneer er data of de structuur aangepast wordt, moet dit wel consistent gehouden worden. Ik adviseer om het schema van de documenten consistent te houden. Het is niet aan te raden om de helft van de documenten in een collectie volgens een oud formaat op te slaan en een andere helft van de documenten volgens een nieuwer formaat. De applicatie moet dan met elk formaat om kunnen gaan wat extra complexiteit met zich mee brengt.

De Knowledge Service is te gebruiken door middel van REST API's. Deze API's laten andere applicaties gebruik maken van de data en functionaliteiten van de Knowledge Service. Wanneer er nieuwe functionaliteiten worden geïntegreerd in knowNow betekend dit vaak ook dat hier een nieuwe of aangepaste REST API van gemaakt moet worden.

Wanneer deze data uit MongoDB wordt gehaald moet de database bij nieuwe data mogelijk worden aangepast. Of moet er een nieuwe collectie gemaakt worden waar de data op het juiste formaat en mogelijk gedupliceerd wordt opgeslagen.

MongoDB heeft geen ondersteuning voor joins[A6]. Doordat MongoDB geen joins kent kan data niet ten tijde van selecties gecombineerd worden. Wel kan er voor worden gekozen om dit te doen in de applicatie. Hierdoor wordt data niet dubbel opgeslagen in de database, maar neemt de verwerkingstijd toe. De data moet eerst door de applicatie bij elkaar worden gezocht.

Het is onbekend wat de gevolgen zijn van het, indien nodig, dubbel opslaan van de data. Hoeveel extra ruimte dit inneemt en wat de daaraan de verbonden extra kosten zijn.

3.4.4 Conclusie

Doordat er in MongoDB geen schema gedefinieerd hoeft te worden is MongoDB erg flexibel bij aanpassingen aan het schema. Dit neemt echter niet weg dat er wel aanpassingen gedaan moet worden mocht de structuur van data wijzigen.

Het is daarom belangrijk om te kijken hoe dat het best in kan spelen op de huidige migraties van knowNow. Hoe kan er, bij aanpassingen of uitbreiding voor worden gezorgd dat de database op alle installaties gelijk blijft?

Ook moet er gekeken worden wat de beste manier is waarop MongoDB kan inspelen op de REST API's. Moet data dubbel opgeslagen worden of is het toch beter om deze in de applicatie samen te voegen? Dan moeten er wel extra stappen doorlopen worden tijdens het verwerken van requests maar wordt er zuinig omgegaan met de opslagruimte.

Om deze vragen te kunnen beantwoorden moet hier eerst verder onderzoek naar gedaan worden dit wordt verder in dit onderzoek uitgevoerd (Deelvraag 2).

3.5 Schaalbaarheid

Managementsamenvatting

In dit hoofdstuk is gekeken naar de schaalbaarheid van MongoDB. MongoDB biedt ondersteuning voor zowel verticaal schalen(upgraden van hardware) en horizontaal schalen(toevoegen van nieuwe machines). Horizontaal schalen wordt sharding genoemd. Het aantal shards is onbeperkt en hiermee is MongoDB theoretisch onbeperkt schaalbaar.

Door middel van MMS[4] of Ops Manager is het mogelijk de geschaalde omgeving te monitoren.

Met Replica Sets is het daarnaast, ook in een gesharde omgeving, mogelijk om data redundant te maken.

Uit de gevonden informatie kan nog geen conclusie worden getrokken voor de specifieke benodigdheden voor knowNow. Dit kan in een later stadia onderzocht worden.

3.5.1 Inleiding

In dit hoofdstuk wordt de schaalbaarheid van knowNow bij het gebruik van MongoDB onderzocht. Er wordt antwoord gegeven op de volgende vraag:

"Welke invloed heeft MongoDB op de schaalbaarheid van knowNow?"

Vanwege de verwachte groei van het aantal gebruikers van knowNow is het zeer belangrijk dat de knowNow applicatie met de gebruikers mee kan groeien. Het is nog onbekend wat voor invloed MongoDB op de schaalbaarheid van de knowNow applicatie kan gaan hebben. Daarom wordt er met dit onderzoek getracht dit te verhelderen.

Het schalen van databases kan zowel verticaal als horizontaal. Bij verticaal schalen verbeteren de specificaties van de al bestaande server. Zo kan er bijvoorbeeld meer geheugen in gestopt worden of de opslagruimte worden vergroot. Bij horizontaal schalen worden er nieuwe servers toegevoegd naast de al bestaande server.

Verticaal schalen is al snel duurder dan horizontaal schalen. Grote aantallen CPU's en veel RAM in een server stoppen is duurder dan meerdere kleine systemen[A10][A11].

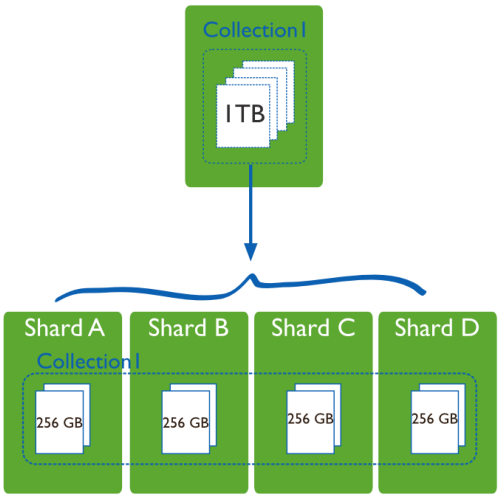
MongoDB heeft ondersteuning voor beide soorten schaling. Door het verhogen van bijvoorbeeld het RAM, de CPU of de opslagruimte kan een MongoDB instantie verticaal schalen. Horizontaal schalen noemt MongoDB sharding.

[4] MMS is een een cloud-monitoring service van Mongo, Inc.

3.5.2 Sharding

Sharding is het verdelen van data over meerdere MongoDB servers, ook wel shards genoemd[A10]. Elke shard heeft zijn eigen MongoDB instantie en zijn eigen data set.

Wanneer er een collectie van bijvoorbeeld 1.024GB over 4 gelijke shards wordt verdeeld, betekent dit dat er op elk van de 4 shards 256GB komt te staan. Dit is in figuur 3-7 weergegeven.



Figuur 3-7. Voorbeeld van sharding. De collectie wordt over meerder shards verdeeld

Door het verdelen van de records over meerdere shards is elke shard verantwoordelijk voor een eigen set records. Bij het bijwerken van één record hoeft dit ook maar op één shard uitgevoerd te worden. De taken die op een collectie uitgevoerd worden, worden hierdoor verdeeld over de shards.

Records worden niet willekeurig verdeeld over meerdere shards. Dit wordt bepaald door een Shard Key. Door het kiezen van een Shard Key bepaald de ontwikkelaar welke records op welke shard terecht komen. De Shard Key kan voor elke gesharde collectie anders zijn. Het kan het ID van een record zijn, maar ook de naam van een persoon of een gehasht attribuut.

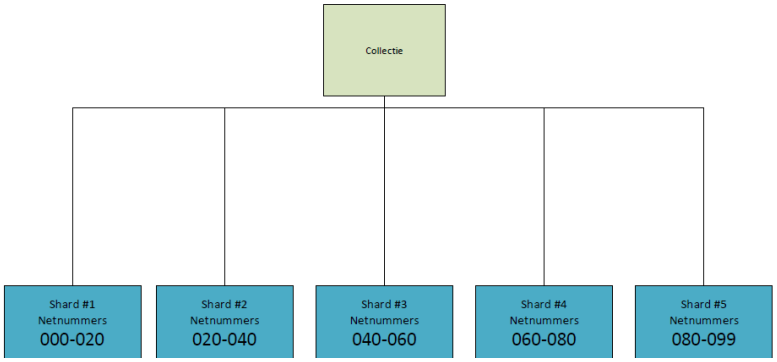
Het kiezen van een goede Shard Key is erg belangrijk. Shard Keys zijn niet aanpasbaar. Indien er een eenmaal gekozen Shard Key veranderd moet worden, moet de gehele database geleegd worden, en daarna weer worden teruggezet[A13].

Een goede shard key hangt af van de situatie en meerdere aspecten. Een goede Shard Key kan voor een situatie goed zijn terwijl het voor de andere situatie niet werkt.

Wanneer bijvoorbeeld het netnummer van de vaste telefoon is gekozen als Shard Key, en de dataset bestaat uit iedereen van Nederland dan verdeeld (voor dit voorbeeld uitgaande dat het aantal telefoonnummers per netnummer evenredig verdeeld is) de data ook goed over de shards.

Wanneer de dataset echter bestaat uit de inwoners van Amsterdam, heeft een groot deel hetzelfde netnummer, die van de regio Amsterdam. Hierdoor is de verdeling niet meer gelijk over de shards en bevatten enkele shards bijna alle data. Sommige shards hebben het erg druk terwijl andere shards bijna geen data bevatten en bijna niks hoeven te doen.

In dat geval kan er voor worden gekozen een andere Shard Key te kiezen, bijvoorbeeld de naam of een gehast ID. Door het hashen van een ID wordt de data alsnog gelijk verdeeld over de shards[A13].



Figuur 3-8. Voorbeeld van verdeling netnummers per Shard

Per gesharde collectie moet er een goede shard key worden gekozen. De shard key kan voor elke collectie anders zijn. Het is daarom altijd belangrijk een shard key te kiezen die goed te verdelen is over de meerdere shards.

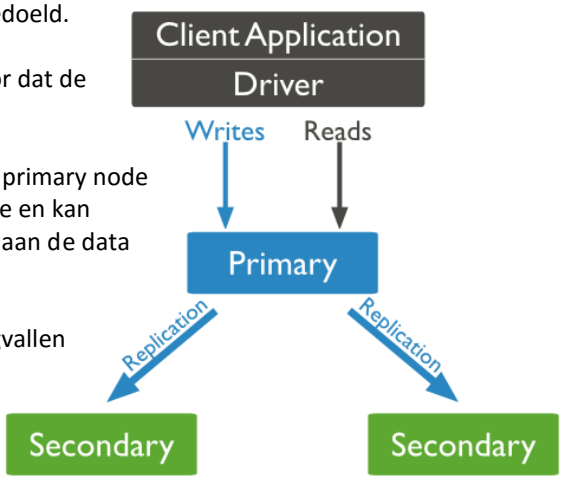
De beste Shard Key voor knowNow zal later worden onderzocht. De keuze wordt gemaakt in het Adviesrapport die aan het einde het afstudeerproject wordt gemaakt.

3.5.3 Replica Sets

MongoDB kent zogenaamde 'Replica Sets'. Met 'replica' wordt een kopie bedoeld. Replica sets is een manier waarmee ervoor gezorgd kan worden dat de data redundant blijft. Wanneer er één server uitvalt zorgen de Replica Sets ervoor dat de MongoDB database toegankelijk blijft en er geen data verloren gaat.

MongoDB raad aan om minimaal een replica set te maken met 3 leden, één primary node en twee secondary nodes. Een applicatie schrijft alleen naar de primary node en kan alleen lezen uit een secondary node. De primary node geeft alle wijzigingen aan de data door aan de secondary nodes zodat deze consistent met elkaar blijven.

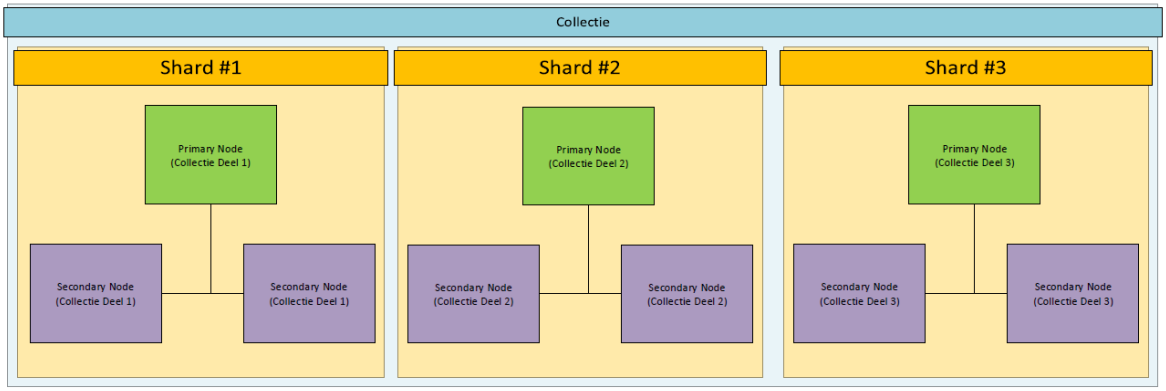
Het aantal nodes dient te allen tijde oneven te zijn. In het geval van het wegvallen van de primary node dient er een nieuwe primary node gekozen te worden. Hiervoor is een meerderheid nodig. Als bij 3 nodes 1 node wegvalt kunnen 2 van de 3 nodes stemmen en is er een meerderheid. Wanneer er maar 2 nodes zijn en er valt er 1 weg dan kunnen 1 van de 2 nodes stemmen, dat is geen meerderheid en de node blijft dan secondary en read-only.



Figuur 3-9. Een Replica Set (Bron: MongoDB)

Door middel van priority te geven aan de nodes kan worden aangeven welke nodes het beste een nieuwe primary node kan worden. Dit is bijvoorbeeld handig wanneer nodes geografisch op andere plekken liggen. Het kan wenselijk zijn om een primary node te hebben die in het eigen land ligt, en niet een van de nodes te kiezen die mogelijk overzees liggen. Hoe hoger de priority is, hoe hoger de kans dat de node een primary wordt in het geval dat een eerdere primary node is weggevallen. In een replica set kunnen maximaal 7 stemmende nodes zitten[A14]. Het aantal nodes in een replica set is maximaal 50. Er moeten dan niet-stemmende nodes toegevoegd worden. Dit kan door een instelling aan te passen per node.

Replica Sets zijn te combineren met Shards. Een shard krijgt dan zijn eigen replica set. Replica sets zijn niet verplicht en ook niet alle shards moeten replica sets.



Figuur 3-10. Voorbeeld van Replica Sets in een Gesharde Collectie

Naast Primary en Secondary Nodes kunnen ook Arbiter Nodes worden toegevoegd. Deze Nodes kunnen niks anders doen dan stemmen bij een verkiezing voor een nieuwe primary node. Arbiter nodes kunnen worden gebruikt om een meerderheid te behalen wanneer er een of meerdere primary nodes wegvallen. Arbiter nodes hebben zelf geen dataset.

3.5.4 Write Concern

Replica Sets zijn 'Eventually consistent'. Dit betekent dat aanpassingen die gedaan worden op de Primary Node niet meteen doorgevoerd worden op de secondary nodes. Hier kan enige tijd overheen gaan.

Wanneer een aanpassing gedaan moet worden aan data kan er een Write Concern meegeven worden. Met een Write Concern wordt aangegeven of de applicatie wilt weten of en wanneer een aanpassing doorgevoerd is. MongoDB kent een aantal verschillende Write Concerns Levels

Write Concern Level	**Bevestiging**	**Antwoord wanneer**
Unacknowledged	Nee	Geen antwoord
Acknowledged	Ja	Wanneer opgenomen in geheugen van Primary Node
Journaled	Ja	Wanneer opgeslagen op de Journal van Primary Node. Hiervoor moet de journalling optie in MongoDB aanstaan. Een Journal is een bestand op de harde schrijf waar alle aanpassingen in worden opgeslagen voordat deze worden verwerkt in het databestand op de harde schrijf.
Replica Acknowledged	Ja	Wanneer data is opgeslagen in het geheugen van de Primary Node en in een aantal Secondary Nodes. De applicatie kan het aantal Secondary Nodes waar de data in het geheugen opgeslagen moet worden voordat de applicatie antwoord krijgt zelf kiezen. Ook kan de applicatie kiezen voor "majority", dan wordt de data eerst geschreven naar de meerderheid van het aantal secondary nodes voordat er een bevestiging wordt gestuurd. Replica Journaled bestaat niet. Echter is het wel mogelijk om gelijktijdig gebruik te maken van Journaled en Replica Acknoledged. Journaled geeft antwoord terug wanneer het op de primary node is verwerkt in de Journal, Relica Acknowledged geeft antworod terug wanneer het op het opgegeven deel van de servers staat.

Voor elke toepassing kan de Write Concern verschillen. Unacknowledged kan zeer geschikt zijn voor een applicatie waar zeer veel data gegenereerd wordt maar het niet uitmaakt als er af en toe een record niet wordt opgeslagen. Bijvoorbeeld bij een sensor die elke paar milliseconden een meting doet. De voordelen van het snel doorgeven van een record (er hoeft niet meer op een antwoord worden gewacht) zijn groter dan het voordeel wanneer er zeker is dat alles verwerkt is.

Acknowledged is de standaard Write Concern bij een nieuwe MongoDB installatie. Bij Acknowledged krijgt de applicatie een bevestiging wanneer de aanpassing is doorgevoerd in het geheugen van de MongoDB database. De applicatie weet dan zeker dat de aanpassing is aangekomen bij de MongoDB database. De MongoDB database voert op bepaalde momenten aanpassingen in het geheugen door op de harde schijf. Mocht in de tussentijds de server uitvallen dan gaan alle aanpassingen die alleen op het geheugen staan verloren. Bij de Acknowledged Write Concern is daarom altijd nog een kleine kans dat de aanpassing verloren gaat.

Journaled lijkt grotendeels op de Acknowledged Write Concern, maar hier krijgt de applicatie pas een bevestiging nadat deze is verwerkt door in de Journal. Bij Journaled Write Concern blijft de data, ook wanneer de server tijdelijk uitvalt, opgeslagen.

Replica Acknowledged betekend dat de applicatie pas een antwoord krijgt wanneer de aanpassing is verwerkt in het geheugen op de Primary Node een 1 of meerdere Secondary Nodes. De applicatie weet hierbij zeker dat de data zich op het aantal nodes bevindt dat is opgegeven. Wanneer er 'majority' is opgegeven weet de applicatie zeker dat het op de meerderheid van de beschikbare Replica nodes aanwezig is.

3.5.5 MMS & Ops Manager

MongoDB Management Service (MMS) is een cloudoplossing van Mongo, Inc.[5] waarin het mogelijk is één of meerdere MongoDB instanties te monitoren en te beheren via een cloudapplicatie[A15].

Naast het monitoren van het gebruik van de servers is het ook mogelijk om instanties te back-uppen en te schalen. Het is via de applicatie mogelijk om te sharden en replica sets in te delen.

MMS geeft de mogelijkheid om zowel op servers in eigen beheer alsmede servers bij Amazons's AMS cloud service toe te voegen aan de applicatie. Een zogenaamde "MMS Agent" zorgt ervoor dat de server kan communiceren met MMS.

Alhoewel servers in eigen beheer toegevoegd kunnen worden, draait MMS zelf wel in de cloud. Gegevens over de server komen in de cloud terecht en daarnaast kunnen de servers ook worden beheerd vanuit de cloud. Hiermee worden de gegevens doorgegeven aan een derde partij, MongoDB, en daarnaast toegang verleend tot servers wat niet voor elke organisatie wenselijk is.

Bij gebruik van MMS wordt metadata over de MongoDB database naar Mongo, Inc. verzonden. Het gaat hierbij volgens de Terms of Service[A17] van Mongo, Inc. in ieder geval om de volgende informatie:

Metadata	**Omschrijving**
Header Information	Informatie wat meegezonden wordt in de headers van de request. Hier is onder andere informatie te vinden over het type request, bijvoorbeeld het type data wat wordt toegevoegt, de collectie waar de data aan wordt toegevoegd etc.
Checksum Quantities	Checksums is een sleutel welke wordt gegenereerd van de data. Aan de hand van een checksum kan worden gekeken of data gelijk aan elkaar is zonder dat de data zelf voor MongoDB beschikbaar is.
File Size	De grootte van een bestand (bijvoorbeeld een backupbestand)
File Type	Het type van een bestand (bijvoorbeeld een backupbestand)
Archival dates	Data's van archieven (o.a. backups)

Uit de Terms of Service is niet af te leiden welke data precies wordt verstuurd naar MongoDB. Zo wordt 'header information' wel genoemd maar wat precies in deze informatie staat is niet bekend.
Naast de bovenstaande data kunnen er ook backups naar MMS worden verstuurd. Deze backups kunnen worden via een veilige SSL verbinding naar MongoDB verstuurd, maar worden bij Mongo niet versleuteld opgeslagen[A16]. Theoretisch kan Mongo, Inc. de bij hun opgeslagen backups uitlezen met alle data daarin.

[5] Mongo, Inc. is het commerciële bedrijf achter MongoDB.

Wanneer er met vertrouwelijke informatie wordt omgegaan is dit niet altijd wenselijk. Mongo, Inc. ziet dit zelf ook in en heeft daarom Ops Manager geïntroduceerd. Ops Manager is een on-premise variant van MMS. Dit houdt in dat een organisatie deze applicatie op eigen servers kan hosten en dit niet meer in de cloud bij Mongo gebeurd.

Voor Ops Manager is een licentie voor MongoDB Enterprise Advanced nodig. Voor MMS is dit niet benodigd, echter dient er voor MMS wel betaald te worden. De prijs hiervan scheelt per situatie en is in de tabel hieronder samengevat.

Aantal servers	t/m 8 servers gratis Daarna $50 per server per maand
Backup (optioneel)	Eerste 1GB replica set gratis Daarna $2.50 per GB per maand

3.5.6 Conclusie

knowNow heeft nu 7000 gebruikers en wil binnen 5 jaar doorgroeien tot 3 miljoen gebruikers. Het knowNow team is nu al bezig met de technische voorbereiding om dit aan te kunnen. Zo is er een begin gemaakt aan het scheiden van services zoals een Search Engine en een Graph engine zoals te lezen valt in het Technisch Ontwerp van de huidige applicatie.

Ook het te ontwikkelen Proof-Of-Concept is een voorbereiding op deze groei. Aan de hand van de onderzoeken en het Proof-Of-Concept kan worden besloten om dit te integreren in de knowNow applicatie.

MongoDB lijkt door middel van Sharding goed schaalbaar te zijn. Het aantal shards is ongelimiteerd[A5] en groeit daarom theoretisch onbeperkt mee met de Knowledge Service. Wel is het erg belangrijk om een goede Sharding Key voor elke collectie te bepalen. Wanneer deze ten tijde van het ontwerpen van het model verkeerd wordt gekozen kan dit vergaande consequenties hebben, de database moet dan geheel opnieuw gevuld worden, wat tot down time kan gaan leiden.

Uit de gevonden informatie kan geen conclusie worden getrokken op de specifieke benodigdheden voor knowNow. Hoeveel shards zijn er nodig? Wat moeten de systeemeisen van de computers zijn en wat zijn de goede sharding keys? Zijn er replica sets nodig en hebben deze invloed op de schaalbaarheid? Om deze vragen te beantwoorden moet er verder onderzoek gedaan worden naar de schaalbaarheid en de performance. Het is beter het onderzoek naar schaalbaarheid uit te voeren op een daadwerkelijke applicatie. Hierdoor kunnen harde cijfers gegeven worden over wat nodig is in plaats van alleen een theoretische benadering. Wanneer de Product Owner van knowNow dit nodig acht wordt dit in een later stadia onderzocht.

Het gebruik van MMS of Ops Manager, is zeker voor het DevOps team van knowNow, erg nuttig. Met MMS of Ops Manager is het mogelijk om te monitoren op het gebied van gebruik, hoeveelheid opgeslagen data, status en eventuele problemen. Het DevOps team kan zo snel, in een oogopslag, de status van de MongoDB database zien en bij eventuele problemen snel ingrijpen. Dit brengt echter wel een prijs met zich mee. De hoogte van deze prijs hangt af van de hoeveelheid benodigde servers wat eerst verder onderzocht moet worden. Wanneer een monitor tool door de Product Owner gewenst is, moet besloten worden of er gekozen wordt voor de cloudoplossing, waarbij metadata naar een derde wordt gestuurd, of de in-premise Ops Manager.

3.6 Conclusie Deelvraag 1

Op het Technisch Ontwerp van knowNow heeft MongoDB een grote invloed. Er wordt een nieuwe applicatie ontwikkeld op basis van de Microservice architectuur, de Knowledge Service. Door het gebruik van microservices dient de gehele architectuur te worden aangepast. Het Technisch Ontwerp is hierdoor totaal anders waardoor de invloed zeer groot is.

Op het security vlak moet MongoDB zo ingericht worden dat deze niet van buiten het netwerk te bereiken is en dat er bij het verbinden met MongoDB door de applicatie moet worden ingelogd met een gebruikersnaam en wachtwoord. MongoDB zal dan niet bepalen wie bij welke data mag. Dit dient te worden gerealiseerd in de applicatie.

Het documentschema van MongoDB is zeer flexibel. Elk document kan een ander schema hebben. Welke invloed de MongoDB precies heeft op de uitbreidbaarheid van knowNow wordt verder onderzocht in Deelvraag 2.

MongoDB biedt door middel van Sharding mogelijkheden om te schalen. Dit is theoretisch onbeperkt. Echter, is er doordat er nog geen applicatie is om mee te testen, niet duidelijk wat er nodig is om te schalen voor knowNow. De gehele architectuur moet aangepast worden. De precieze invloeden zijn nog niet bekend. Dit kan worden getest door eerst een Proof-of-Concept te ontwikkelen waarna eventueel de schaalbaarheid getest kan worden en de bevindingen in dit hoofdstuk kan worden geëvalueerd.

4. Deelvraag 2: Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?

4.1.1 Inleiding

In dit hoofdstuk wordt onderzoek gedaan naar de uitbreidbaarheid van knowNow bij het gebruik van MongoDB. Door middel van dit onderzoek wordt daar getracht antwoord op te geven.

Samen met de opdrachtgever zijn de volgende onderzoeksvragen opgesteld die in dit onderzoek beantwoord moeten worden.

- Wat is de uitbreidbaarheid van knowNow bij het gebruik van MongoDB?
    - Wat zijn de consequenties van het uitbreiden van het MongoDB documentschema op de performance en schaalbaarheid?
    - Wat zijn de consequenties van het toevoegen van een nieuw REST endpoint op het documentschema en de performance?
    - Is het mogelijk een MongoDB database te migreren door middel van gestapelde updates?

4.2 Het MongoDB documentschema

4.2.1 Inleiding

In dit hoofdstuk wordt informatie gegeven over het documentschema van MongoDB en wordt gekeken naar de verschillen met een relationele database.

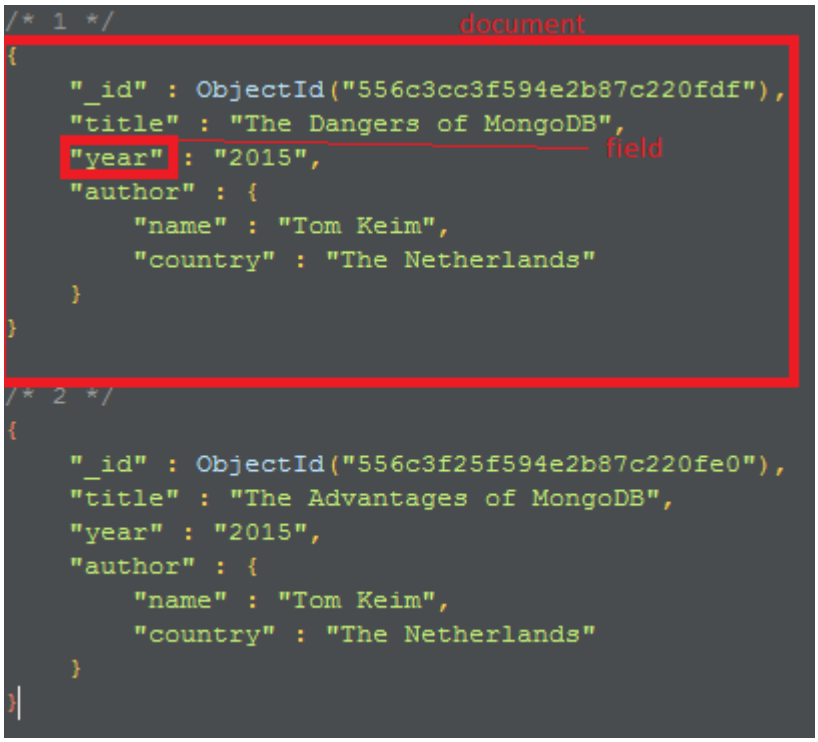
4.2.2 Collecties, velden & documenten

Een MongoDB database werkt totaal anders dan een relationele database. Waarbij een relationele database zoals SQL server tabellen, kolommen en rijen kent, kent MongoDB collecties, velden en documenten.

Data wordt in MongoDB in BSON opgeslagen. BSON staat voor Binary JSON (JavaScript Object Notation) en is een binaire variant van het veelgebruikte JSON. De structuur van BSON toont dan ook veel gelijkenissen met die van JSON.

MongoDB kent geen relaties. Het is natuurlijk wel mogelijk om te verwijzen naar een ander document met bijvoorbeeld een ID maar er kunnen geen voorwaardes (constraints) aangegeven worden zoals in een relationele database gebruikelijk is.

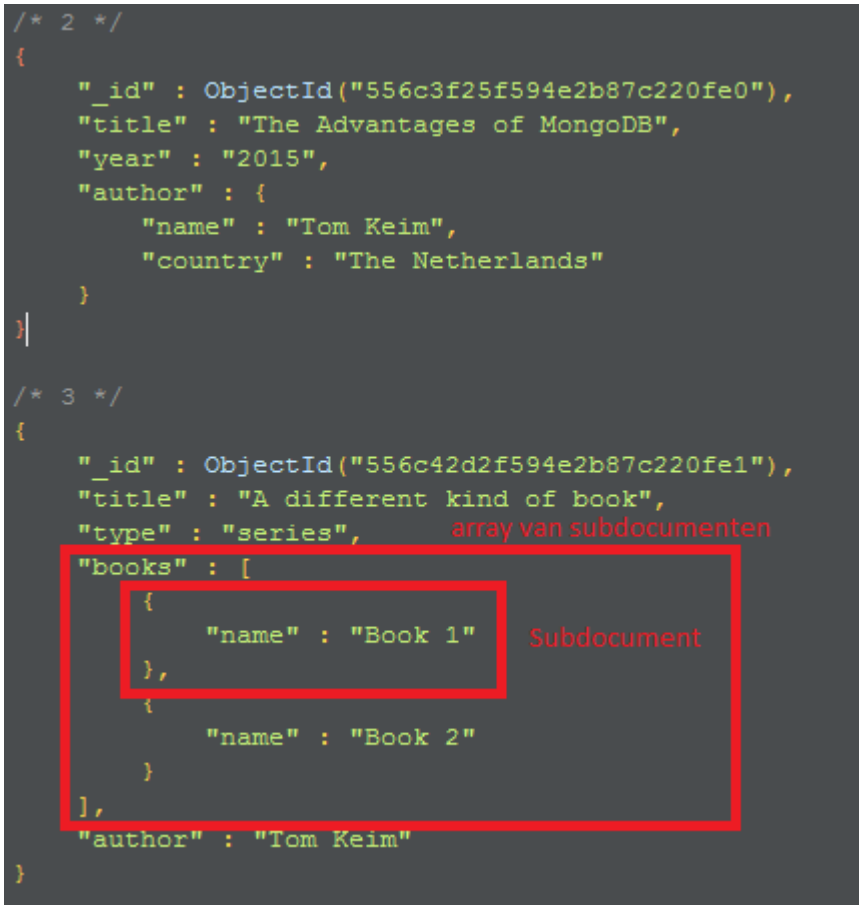
In plaats van relaties aanleggen kan MongoDB documenten 'embedden'. Er worden dan subdocumenten in een document geplaatst. Deze sub documenten bevatten de velden waar dan normaal gesproken heen verwezen wordt.



Figuur 4-1. Voorbeeld van een document in MongoDB

In Figuur 4-1 staat een voorbeeld van een collectie “books”. Deze collectie bevat 2 documenten. Elk van deze documenten hebben 4 velden: een uniek id, een title, een jaar van verschijning en een auteur. Het veld auteur bevat weer een subdocument met daarin gegevens over de auteur. Dit subdocument heeft 2 velden: een naam en een land van herkomst.

Er is geen vaste structuur voor elk document binnen een collectie. Elk document kan andere velden hebben dan de andere documenten binnen die collectie.



Figuur 4-2. Documenten hoeven niet dezelfde structuur te hebben

In Figuur 4-2 is een voorbeeld te vinden waarbij het eerste document een andere structuur heeft dan het 2e document. Hierdoor is MongoDB zeer flexibel. Het zit niet vast aan een vooraf vastgestelde structuur die eerst aangepast moet worden voordat er een nieuwe structuur gehanteerd kan worden.

Elk subdocument kan weer zijn eigen subdocument kan hebben. Dit ‘nesten’ kan tot maximaal honderd niveaus diep[B1].

4.2.3 Datatypes

MongoDB kent verschillende soorten datatypen. Dit zij zogenaamde BSON types. BSON kent de volgende datatypes[B2]:

- String (voor strings)
- Object (voor geneste subdocumenten)
- Double
- Array
- Binaire data
- Object ID
- Boolean
- Date
- Null
- Regular Expression
- JavaScript
- Symbol (hetzelfde als string)
- 32 bits integer
- 64 bits integer

Daarnaast zijn er nog enkele typen voor intern gebruik zoals Timestamp, minkey en maxkey[B2].

4.2.4 Verschillen met een relationele database

In dit hoofdstuk worden een aantal termen die in een relationele database worden gebruikt naast de termen gelegd die worden gebruikt in MongoDB. Hierdoor wordt getracht de lezer bekender te maken met de termen die door MongoDB worden gebruikt. Voor deze vergelijking is uit gegaan van SQL Server als relationele database.

Relationele Database term	**MongoDB database term**[B3]	**Opmerking**
Database	Database	Net zoals een relationele database kent MongoDB databases.
Tabel	Collectie	In MongoDB worden tabellen collecties genoemd. Een collectie valt, net als een tabel, onder een database. In tegenstelling tot een Tabel wordt er in een MongoDB collectie geen datamodel gedefinieerd.
Kolom	Veld (field)	Een kolom van een relationele database is te vergelijken met een veld in MongoDB. In een relationele database staan de kolommen vast per tabel, in MongoDB niet.
Rij	Document	Een rij in een relationele database is vergelijkbaar met een document in MongoDB. Het document bevat een of meerdere velden en valt onder een collectie. Elk document kan een of meerdere velden hebben. Deze velden kunnen per document in de collectie verschillen.
Primaire Sleutel	Primaire Sleutel	Ook MongoDB kent primaire sleutels. In een Relationele database kan een database ontwerper zelf de primaire sleutel kiezen, in MongoDB is dit altijd het veld '_id'.
Join	-	MongoDB kent geen joins. Het is niet mogelijk om aan de hand van een vreemde sleutel documenten te combineren zoals dat met Joins in een relationele database gebeurd. Een mogelijk alternatief is het gebruik van subdocumenten. Er worden dan meerdere documenten in elkaar 'embed'.
Index	Index	Het is net zoals in een relationele database indexen te leggen op velden in MongoDB. MongoDB kent verschillende soorten index typen[B4].

4.2.5 Aanpassen Documentschema

Voor een MongoDB collectie kan geen schema gedefinieerd worden. Elk document in een collectie kan zijn eigen velden hebben. Hierdoor hoeft bij een aanpassing aan het data model, geen schema aangepast te worden. Er kan in een keer nieuwe documenten volgens het nieuwe documentschema toevoegt worden aan de MongoDB collectie.

Het nadeel hiervan is echter dat de applicaties die gebruik moeten maken van de MongoDB collectie dan ook met deze verschillende modellen om moeten kunnen gaan. De applicatie krijgt dan extra constructies, er moet eerst worden gekeken om welke variant van het documentschema het gaat voordat er in de applicatie mee gewerkt kan worden.

Om deze constructies te voorkomen is raad ik aan om het documentschema met terugwerkende kracht aan te passen. De applicatie moet dan aangepast worden maar er hoeft dan geen rekening gehouden te worden met meerdere schema's.

Het aanpassen van een documentschema kan direct op de MongoDB server of via een van de vele drivers, zoals een C# driver waarmee het mogelijk is om vanuit C# .NET met MongoDB te communiceren.

Later in dit document wordt gekeken op welke manieren het documentschema het beste voor knowNow aangepast kan worden.

4.3 Geschetst MongoDB documentschema

4.3.1 Inleiding

In dit hoofdstuk wordt vanwege de afwezigheid van een documentschema van knowNow (deze wordt pas later ontwikkeld) voor MongoDB een mogelijk documentschema voor de Knowledge Items geschetst. Dit geschetste model kan gebruikt worden in dit onderzoek.

4.3.2 Situatie

In hoofdstuk 2 is gesproken over het begrip Knowledge Items. Kort samengevat bevat een Knowledge Item alles wat op de Knowledge pagina staat. De velden zijn als volgt:

Attribuut	**Omschrijving**
Titel	De titel van het Knowledge Item
Omschrijving	De omschrijving van het Knowledge Item
Rating	De beoordeling van het Knowledge Item
Comments	Reacties op het Knowledge Item met daarin de tekst van de reactie, de schrijver van de reactie en de datum van de reactie.
Tags	Tags van het Knowledge Item
Authors	Auteurs en Co-auteurs van het Knowledge Item
Followers	Het aantal mensen dat een Knowledge Item volgt.
Communities	Communities waar het Knowledge Item onder valt. Hiernaast is ook zichtbaar of de community voor elke ingelogde gebruiker te bekijken is of alleen voor gebruikers die lid zijn van deze commuity
Views	Hoeveel mensen het Knowledge Item hebben bekeken
Datum	Datum van een Knowledge Item

Voor dit document wordt er gebruik gemaakt van een documentschema die op basis van de velden op de vorige pagina is opgezet. Dit documentschema is mogelijk niet de versie die uiteindelijk in het Proof-of-Concept gebruikt gaat worden. Dit documentschema is opgezet zodat voor de lezer van dit document duidelijk is wat de consequenties zijn van de verderop beschreven situaties voor het documentschema.

```
{
    "_id" : ObjectId("556d73a668afe0c4d1382a47"),
    "title" : "Dit is een titel van een Knowledge Item",
    "description" : "<p>Dit bevat een omschrijving van een Knowledge Item</p>",
    "tags" : [
        "tag1",
        "tag2",
        "tag3",
        "tag4"
    ],
    "followers" : 15.0000000000000000,
    "views" : 10.0000000000000000,
    "authors" : [
        {
            "name" : "Tom Keim",
            "co_author" : false
        },
        {
            "name" : "Iemand anders",
            "co_author" : true
        }
    ],
    "rating" : {
        "rating" : 4.5000000000000000,
        "votes" : 10.0000000000000000
    },
    "comments" : [
        {
            "author" : "Tom Keim",
            "comment" : "Dit is de reactie",
            "date" : ISODate("2015-06-02T09:13:10.168Z")
        },
        {
            "author" : "Iemand anders",
            "comment" : "Dit is de andereereactie",
            "date" : ISODate("2015-06-02T09:13:10.168Z")
        }
    ],
    "communities" : [
        {
            "name" : "Een Community",
            "public" : true
        }
    ],
    "date" : ISODate("2015-06-02T09:13:10.168Z")
}
```

Figuur 4-3. Een schets van een mogelijk documentschema voor de Knowledge Items. Dit documentschema is mogelijk niet de uiteindelijke situatie die gebruikt gaat worden in het Proof-Of-Concept.

4.4 Uitbreiden van het documentschema

Managementsamenvatting

Dit hoofdstuk beschrijft de uitbreiding van het documentschema met basale attributen. MongoDB biedt meerdere commando's waar het documentschema's eenvoudig mee aan te passen is. Uit gedaan onderzoek blijkt dat de invloed van aanpassingen van het documentschema op de performance niet groot is.

Echter is het belangrijk om onverwachte consequenties te voorkomen door bij elke aanpassing eerst het mogelijke effect te bekijken voordat deze doorgevoerd wordt.

Er kan hiermee gesteld worden dat MongoDB uitbreidbaar is.

4.4.1 Inleiding

In dit hoofdstuk wordt antwoord gegeven op de vraag:

"Wat zijn de consequenties van het uitbreiden van het MongoDB documentschema op de performance en schaalbaarheid"

Met de volgende daarbij horende deelvragen:

- Wat zijn de consequenties op het documentschema?
- Wat zijn de consequenties op de leesperformance bij het uitbreiden van het documentschema?
- Wat zijn de consequenties op de schaalbaarheid bij het uitbreiden van het documentschema?

4.4.2 Consequenties op het documentschema

Zoals in 4.2 al beschreven is, heeft een MongoDB collectie geen vast, vooraf gedefinieerd, schema. Hierdoor hoeven er ook geen aanpassingen gedaan te worden aan een schema wanneer nieuwe attributen toegevoegd worden.

Wanneer de 2 nieuwe attributen toegevoegd worden, de korte omschrijving en een verwijzing naar een (cover)afbeelding, kan dit op de volgende manieren worden gedaan:

1. Alleen voor nieuwe documenten.
2. Met terugwerkende kracht op de al bestaande documenten en voor alle nieuwe documenten.

In 4.2.5 is al beschreven dat het niet verstandig is om meerdere documentschema's te hanteren binnen één collectie. Applicaties die gebruik maken van die MongoDB collectie moeten dan ook met meerdere documentschema om kunnen gaan, wat de applicatie complexer maakt dan nodig is.

Het is daarom ook aan te raden om aanpassingen aan het documentschema's dan ook door te voeren in de al bestaande documenten.

Via een update methode kan MongoDB ook eerdere documenten aanpassen. In Figuur 4-4 staat een voorbeeld van de collectie ‘knowledge’ met daarin één document.

Figuur 4-4. De collectie ‘knowledge’ met één document

In Figuur 4-5 worden doormiddel van een update-statement op de ‘knowledge’-collectie de velden ‘short_description’ en ‘image_url’ toegevoegd. Omdat de waarden voor de oude documenten niet aanwezig waren, blijven deze voor alle documenten leeg. Het is natuurlijk ook mogelijk hier een dummy waarde in te zetten

Figuur 4-5. De instructie om een veld aan alle records toe te voegen aan de collectie van Figuur 4

In Figuur 4-6 is de collectie na het uitvoeren van de update-statement in Figuur 4-5 te zien. Zichtbaar is dat bij het document de velden ‘short_description’ en ‘image_url’ zijn toegevoegd.


```
{
    "_id" : ObjectId("556d8dc00dda61dd8a7b7dfc"),
    "title" : "Dit is een titel van een Knowledge Item",
    "description" : "<p>Dit bevat een omschrijving van een Knowledge Item</p>",
    "tags" : [
        "tag1",
        "tag2",
        "tag3",
        "tag4"
    ],
    "followers" : 15.0000000000000000,
    "views" : 10.0000000000000000,
    "authors" : [
        {
            "name" : "Tom Keim",
            "co_author" : false
        },
        {
            "name" : "Iemand anders",
            "co_author" : true
        }
    ],
    "rating" : {
        "rating" : 4.5000000000000000,
        "votes" : 10.0000000000000000
    },
    "comments" : [
        {
            "author" : "Tom Keim",
            "comment" : "Dit is de reactie",
            "date" : ISODate("2015-06-02T11:04:32.115Z")
        },
        {
            "author" : "Iemand anders",
            "comment" : "Dit is de andereereactie",
            "date" : ISODate("2015-06-02T11:04:32.115Z")
        }
    ],
    "communities" : [
        {
            "name" : "Een Community",
            "public" : true
        }
    ],
    "date" : ISODate("2015-06-02T11:04:32.115Z"),
    "short_description" : "",
    "image_url" : ""
}
```


Figuur 4-6. De collectie van figuur 4-4 na de instructie van figuur 4-5

Wanneer er nu nieuwe documenten toegevoegd worden met alle oude velden en de 2 nieuwe velden blijven de velden voor elk document in de collectie gelijk.

De applicatie of applicaties die gebruik maken van de MongoDB collectie moeten ook bijgewerkt worden zodat er gebruik kan worden gemaakt van de nieuwe attributen. Dit geldt ook voor de te realiseren Knowledge Service. De door de Knowledge Service aangeboden REST API moet afweten van de 2 nieuwe attributen.

4.4.3 Consequenties op de performance

4.4.3.1 Inleiding

De uitbreiding van een documentschema kan mogelijk ook invloed hebben op de performance. Er wordt daarom gekeken of de uitbreiding van een documentschema met enkele 'basale' attributen effect heeft op de performance

Met 'basale' attributen worden in dit onderzoek attributen als een titel, korte omschrijving of afbeelding bedoeld. Dit zijn attributen die vaak enkel tekst bevatten en geen verwijzingen naar andere documenten.

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.

Naast dat er een aanpassing gedaan moet worden aan het documentschema heeft het toevoegen van de 2 attributen mogelijk ook consequenties op de performance van de MongoDB collectie.

Door het toevoegen van de 2 attributen wordt elk document groter. Er worden 2 nieuwe attributen aan toegevoegd en die moeten ergens opgeslagen worden. De grootte van het document neemt daardoor theoretisch toe per toegevoegd attribuut. Omdat dit document ingelezen en verstuurd moet worden naar de applicatie neemt dit theoretisch dan ook meer tijd in beslag.

Of dit in praktijk ook zo is niet bekend. Ook is niet te achterhalen hoe groot de invloed op de performance is. Duurt het ook echt langer om documenten op te vragen, en hoeveel langer duurt dat dan?

Om dat te achterhalen is er een test opgezet. Er is gebruik gemaakt van een zogenaamde performance test. Met de opgezette test wordt gekeken naar hoe snel MongoDB een vast aantal data aanvragen afhandelt.

Deze test is alleen opgezet om het verschil in performance te meten bij een aanpassing van het documentschema. De MongoDB instantie die voor deze test gebruikt wordt is niet geoptimaliseerd. De test is daarom ook niet bruikbaar om de performance te meten van MongoDB over het algemeen.

De test is uitgevoerd met het programma Apache JMeter. JMeter is een applicatie waarmee onder andere opdracht kunt geven om queries meermaals uit te voeren op een database om de performance te testen.

Deze test zijn met 2 collecties uitgevoerd. Elke collectie is apart getest. De eerste collectie is het documentschema zoals geschetst in hoofdstuk 3. Aan de 2^{e} collectie zijn de 2 'basale' attributen, short_description en image_url toegevoegd. Beide collecties bevatten 10.000 documenten.

Testmachine: 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.

4.4.3.2 Documentschema

Voor het testen worden de volgende 2 documentschema's gebruikt.

Standaardmodel	
Attribuut	**Omschrijving**
Titel	De titel van het Knowledge Item.
Omschrijving	De omschrijving van het Knowledge Item.
Rating	De beoordeling van het Knowledge Item.
Comments	Reacties op het Knowledge Item met daarin de tekst van de reactie, de schrijver van de reactie en de datum van de reactie.
Tags	Tags van het Knowledge Item.
Authors	Auteurs en Co-auteurs van het Knowledge Item .
Followers	Het aantal mensen dat een Knowledge Item volgt.
Communities	Communities waar het Knowledge Item onder valt. Hiernaast is ook zichtbaar of de community voor elke ingelogde gebruiker te bekijken is of alleen voor gebruikers die lid zijn van deze community.
Views	Hoeveel mensen het Knowledge Item hebben bekeken.
Datum	Datum van een Knowledge Item.
number	Een nummer oplopend van 1 t/m 10.000.

De collectie met het standaardmodel werd gevuld met het volgende script, die MongoDB kan uitvoeren. Deze genereerd 10.000 documenten volgens het bovenstaande model.

```
var dbs = db.getSiblingDB('loadtest');
for(var n = 1;n <= 10000;n++) {
dbs.loadtestTable.insert({
        title: "Dit is een titel van een Knwoledge Item #" + n,
        description: "<p>Dit is een omschrijving van een Knowledge Item #" + n + "</p>",
        tags: ["tag1","tag2","tag3","tag4"],
        followers: 15,
        views: 10,
        authors: [
            {
                name: "Tom Keim",
                co_author: false
            },
            {
                name: "Iemand anders",
                co_author: true
            }
        ],
        rating: {
            rating: 4.5,
            votes: 10
        },
        comments: [
            {
                author: "Tom Keim",
                comment: "Dit is de reactie",
                date: ISODate('2015-06-02')
            },
            {
                author: "Iemand anders",
                comment: "Dit is andere reactie",
                date: ISODate('2015-06-02')
            }
        ],
        "communities": [
            {
                name: "Een community",
                "public": true
            }
        ],
        "date": ISODate("2015-06-02"),
        number: n
    });
}
```

Uitgebreid model (met de 2 toegevoegde velden)	
Attribuut	**Omschrijving**
Titel	De titel van het Knowledge Item.
Omschrijving	De omschrijving van het Knowledge Item.
Rating	De beoordeling van het Knowledge Item.
Comments	Reacties op het Knowledge Item met daarin de tekst van de reactie, de schrijver van de reactie en de datum van de reactie.
Tags	Tags van het Knowledge Item.
Authors	Auteurs en Co-auteurs van het Knowledge Item.
Followers	Het aantal mensen dat een Knowledge Item volgt.
Communities	Communities waar het Knowledge Item onder valt. Hiernaast is ook zichtbaar of de community voor elke ingelogde gebruiker te bekijken is of alleen voor gebruikers die lid zijn van deze community.
Views	Hoeveel mensen het Knowledge Item hebben bekeken.
Datum	Datum van een Knowledge Item.
number	Een nummer oplopend van 1 t/m 10.000.
short_description	**Een korte omschrijving bestaande uit 'Dit is een omschrijving voor item #' met daarachter het number van document.**
image_url	**Een fictieve url van een image bestaande uit 'http://images.example/' met daarachter het nummer en '.png'.**

De collectie met het uitgebreid model werd gevuld met het volgende script, die MongoDB kan uitvoeren. Deze genereerd 10.000 documenten volgens het bovenstaande model.

```
var dbs = db.getSiblingDB('loadtest');
for(var n = 1;n <= 10000;n++) {
dbs.loadtestTable.insert({
        title: "Dit is een titel van een Knwoledge Item #" + n,
        description: "<p>Dit is een omschrijving van een Knowledge Item #" + n + "</p>",
        tags: ["tag1","tag2","tag3","tag4"],
        followers: 15,
        views: 10,
        authors: [
            {
                name: "Tom Keim",
                co_author: false
            },
            {
                name: "Iemand anders",
                co_author: true
            }
        ],
        rating: {
            rating: 4.5,
            votes: 10
        },
        comments: [
            {
                author: "Tom Keim",
                comment: "Dit is de reactie",
                date: ISODate('2015-06-02')
            },
            {
                author: "Iemand anders",
                comment: "Dit is andere reactie",
                date: ISODate('2015-06-02')
            }
        ],
        "communities": [
            {
                name: "Een community",
                "public": true
            }
        ],
        "date": ISODate("2015-06-02"),
        number: n,
        short_description: "Dit is een omschrijving voor item #" + n,
        image_url: "http://images.example/" + n
    });
}
```

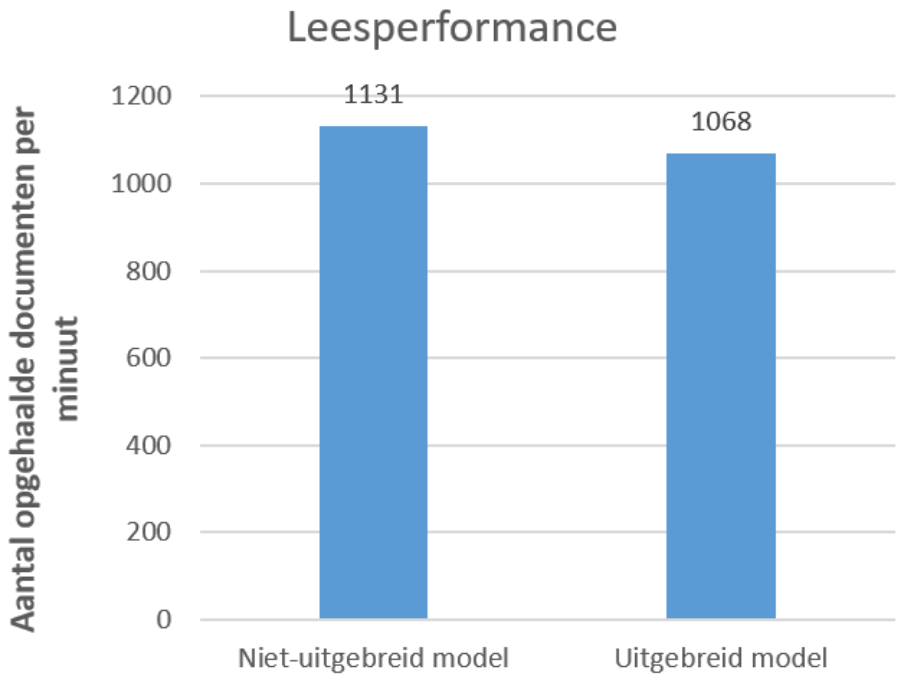
4.4.3.3 Leesperformance

Voor de test worden door JMeter 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.

Eerst is de test uitgevoerd met de collectie voor de nulmeting (standaardmodel). Het resultaat van deze test is een throughput[6] gemiddeld 1.131/minuut is. Dit betekend dat er elke minuut 1.131x een resultaat van de MongoDB database is ontvangen. De average is 258ms. Dit betekend dat JMeter gemiddeld 258 milliseconden moet wachten voordat er een antwoord is ontvangen van MongoDB.

Omdat er 10 gelijktijdige processen bezig zijn met het uitvoeren van de testen (10 threads) kunnen er ook 10 commando's tegelijkertijd uitgevoerd worden. Dit verklaart het verschil tussen de throughput, 1.131/minuut (wat komt op 53ms per commando wanneer ze niet gelijktijdig uitgevoerd wouden) en de gemiddelde uitvoertijd van 258ms.

De tweede test is uitgevoerd op de collectie met de 2 toegevoegde attributen (uitgebreid model). Het resultaat van deze test is een throughput van gemiddeld 1.068/minuut is. Dit betekend dat er elke minuut 1.068x een resultaat van de MongoDB database is ontvangen. De gemiddelde ophaaltijd van een document is 296ms. Dit betekend dat JMeter gemiddeld 296 milliseconden moet wachten voordat er een antwoord is ontvangen van MongoDB.



Figuur 4-7. Resultaat van de leesperformance test (aantal gelezen documenten per minuut)

[6] Throughput is het aantal uitgevoerde commando's per tijdseenheid

Duidelijk is dat de testen aantonen dat het verwachte resultaat, dat het toevoegen van 2 nieuwe attributen ten koste gaat van de performance, klopt. Uit de cijfers blijkt dat de oorspronkelijke collectie gemiddeld 1.131 commando's per minuut uitvoert en dat de collectie waar de 2 basale attributen zijn toegevoegd 1.068 commando's per minuut uitvoert.

In dit voorbeeld kan de collectie met de 2 toegevoegde basale attributen 106 minder commando's uitvoeren per minuut. De vermindering in performance op het gebied van throughput is 5%.

Op het gebied van de uitvoersnelheid is de vermindering 13%. Dit verschil is waarschijnlijk te verklaren doordat het aantal velden per document groeit van 11 naar 13 velden en de benodigde opslagruimte dus toeneemt.

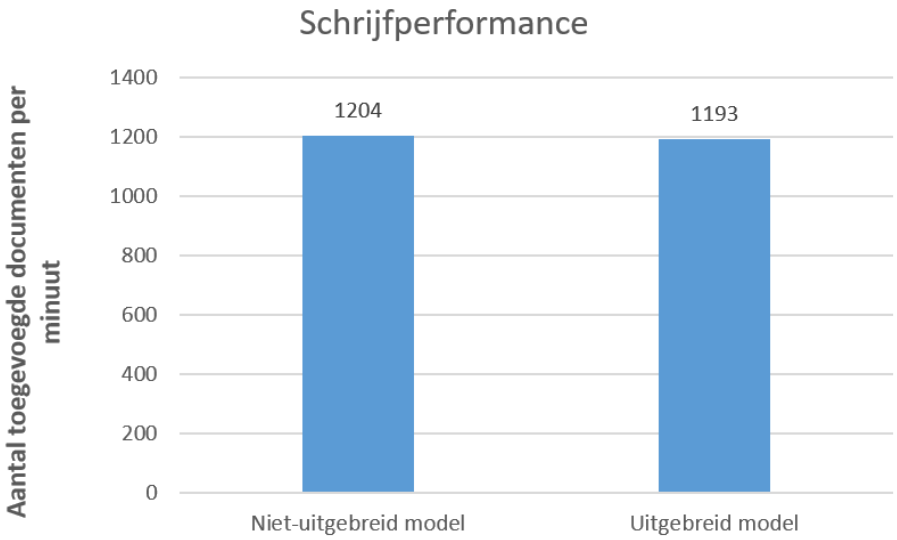
4.4.3.4 Schrijfperformance

Naast de leesperformance is ook de schrijfperformance getest. Deze test is gedaan met de documentschema's uit 4.4.3.2.

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 Standaardmodel, de tweede test voegt ook 5.000 documenten aan de database, maar dan in het formaat van het uitgebreide model. De 5.000 documenten werden eerlijk verdeeld over 10 threads. Elke thread voegt 500 documenten toe.

De test met het standaardmodel resulteerde een gemiddelde verwerkingstijd per document van 297ms, en de throughput, het aantal documenten wat wordt toegevoegd per minuut in de database, 1.204/minuut.

Het verschil is minimaal, de gemiddelde uitvoertijd scheelt maar 3 milliseconden en de doorvoersnelheid maar 10/minuut.



Figuur 4-8. Het resultaat van de schrijftest test (aantal geschreven documenten per minuut)

4.4.3.5 Conclusie

In het geval van de uitbreiding van een documentschema met basale attributen kan MongoDB goed omgaan met deze wijzingen. De aanpassing is met een commando gemakkelijk door te voeren op het documentschema (4.2) en het blijkt dat de impact op de performance niet groot is. Uit de testen in 4.3 blijkt het verschil 13% te zijn. Dit verschil is waarschijnlijk te verklaren doordat het aantal velden per document groeit van 11 naar 13 velden en de benodigde opslagruimte dus toeneemt. Voor de schrijfsnelheid is dit ook minimaal.

Er is voor de testen alleen gekeken naar de toevoeging van 2, simpelere, attributen. Er zijn natuurlijk ook andere uitbreidingen mogelijk, zoals het uitbreiden van een model met subdocumenten. Deze hebben ook effect op de performance. Het effect op de performance kan mogelijk verschillen per situatie. Het is daarom belangrijk om bij elke uitbreiding altijd te kijken of dat mogelijk effect heeft op de performance en of dit effect niet te groot is en de uitbreiding het performanceverlies waard is. Hiermee worden onverwachte consequenties van uitbreidingen voorkomen.

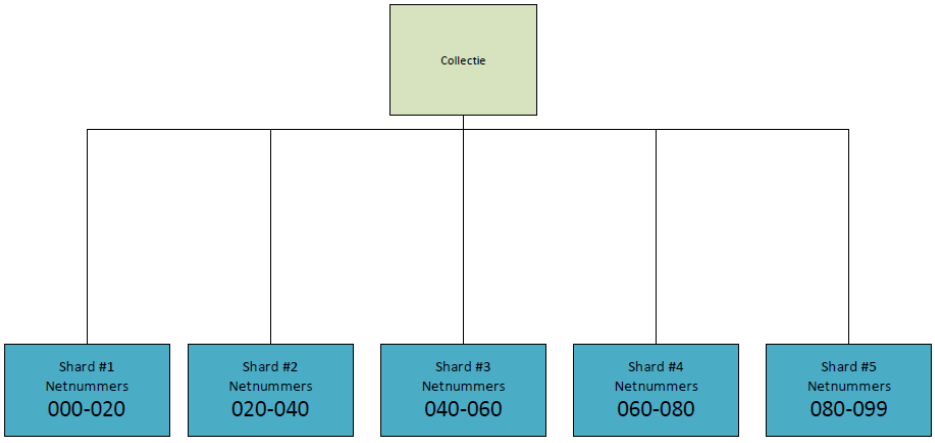
4.4.4 Consequenties op de schaalbaarheid

Schaalbaarheid is het mee kunnen schalen van de applicatie wanneer deze groeit. Zoals eerder beschreven is, kent MongoDB zowel horizontaal als verticaal schalen.

Bij verticaal schalen worden de specificaties van de al bestaande server of servers verbeterd. Bij het horizontaal schalen worden nieuwe servers geplaatst waar de load van de service dan over wordt verdeeld.

Dit horizontaal schalen heet in MongoDB sharding. Bij het sharden wordt data en de verantwoordelijkheden voor deze data over meerdere MongoDB instanties verspreid.

Wanneer een MongoDB collectie uitgebreid wordt met 2 basale attributen en ervoor gekozen wordt dit op alle documenten toe te passen, moet dit logischerwijs ook worden verspreid over de meerdere shards. Elke shard is verantwoordelijk voor een eigen stuk data.



Figuur 4-9. Voorbeeld verdeling data over meerdere shards

De Shard Key zorgt er voor dat data wordt verdeeld over meerdere chunks. Elke chunk komt dan op één shard terecht. Een shard kan meerdere chunks hebben. In Figuur 4-9 staat een voorbeeld van een collectie met een shard key 'netnummers'.

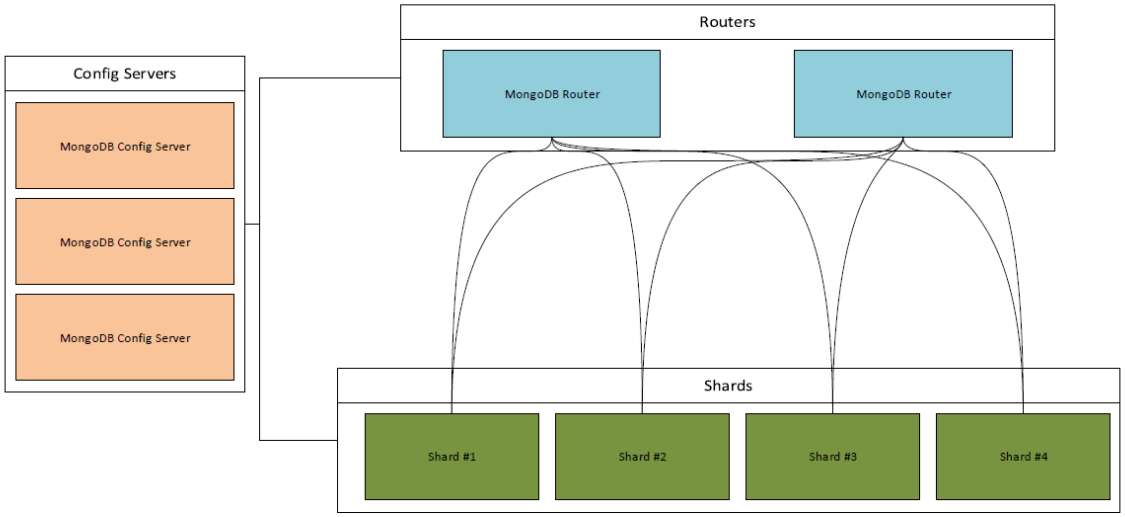
In een Sharded Cluster zitten alle MongoDB instanties die onder die cluster vallen. Er zijn 3 soorten MongoDB instanties in een cluster. Routing servers, Config servers en Shards.

Naam	**Doel**
Routing servers	Een Routing server is de server waar een applicatie mee verbind. Deze zorgt ervoor dat alle data uit de verschillende shards allemaal via deze server op te vragen zijn en dat data toegevoegd wordt op de juiste shards. Deze server bevat zelf geen data.
Config servers	Config servers zijn MongoDB instanties die de metadata bevatten van de shard. De config server bevat de informatie over welke data op welke chunks op welke shard moeten staan. Een productieomgeving heeft exact 3 config servers nodig voor elke sharded database. Wanneer alle config servers uitvallen is alle data is de shard niet te gebruiken[B5].
Shard servers	Een shard server bevat chunks met data. Dit zijn de servers waar de data daadwerkelijk op staat.

Om te kijken wat de invloed is op een Sharded Cluster wanneer er aan een geclusterde collectie 2 basale attributen toegevoegd worden is er een proef opgezet. Deze proef bestaat uit een geclusterde MongoDB database die lokaal op een desktop is geïnstalleerd.

De Sharded Cluster bevat de volgende instanties:

- 3 Config Servers, dit wordt vereist gesteld door MongoDB voor een productieomgeving en om de proef zo dicht mogelijk bij een productiesituatie te houden bevat de proef er ook 3.
- 2 Router servers, met deze 2 servers is van buiten te communiceren
- 4 Shard Servers, de data wordt onderverdeeld over deze 4 servers.

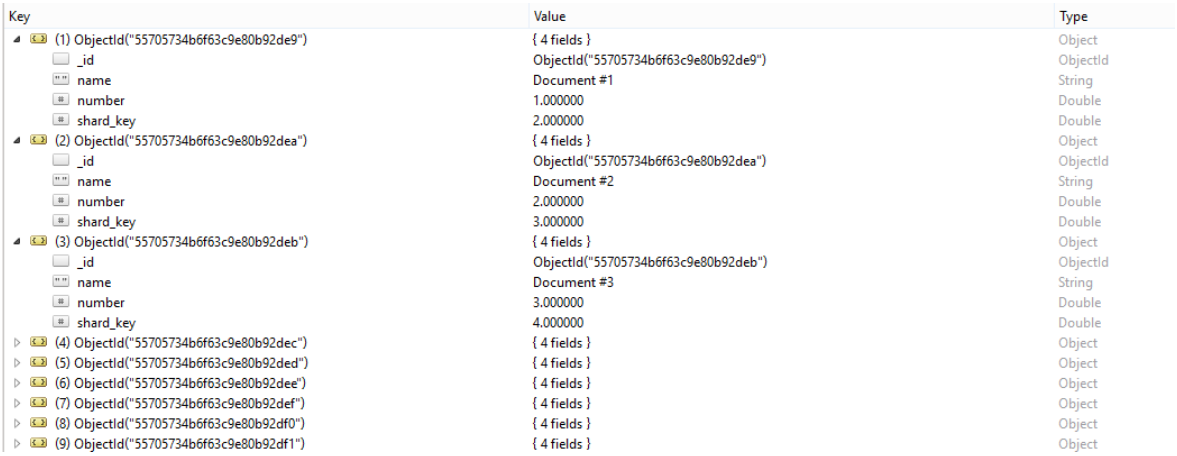


Figuur 4-10. De Shard Cluster

Na het opzetten van al deze instanties kan er een database en collectie worden aangemaakt. Deze database is sDatabase genoemd, de collectie is sCollection genoemd.

De collectie die voor deze proef is aangemaakt bevat 100.000 documenten, elk van deze documenten is opeenvolgend genummerd van 1 t/m 100.000.

Veld	**Doel**
_id	Veld wat automatisch wordt aangemaakt door MongoDB, bevat een unieke waarde.
name	Bevat een korte string ‘Document #’ met daarachter een nummer
shard_key	Een getal tussen de 1 en de 8. Wordt bepaald door het nummer van het record, daarvan de module van 8 en dat plus 1. Er is voor 8 getallen gekozen zodat deze eerlijk te verdelen zijn over de meerdere shards (2 per shard).
number	Elk document krijgt een opvolgend nummer, startend met 1, eindigend met 100.000.



Key	Value	Type
(1) ObjectId("55705734b6f63c9e80b92de9")	{ 4 fields }	Object
_id	ObjectId("55705734b6f63c9e80b92de9")	ObjectId
name	Document #1	String
number	1.000000	Double
shard_key	2.000000	Double
(2) ObjectId("55705734b6f63c9e80b92dea")	{ 4 fields }	Object
_id	ObjectId("55705734b6f63c9e80b92dea")	ObjectId
name	Document #2	String
number	2.000000	Double
shard_key	3.000000	Double
(3) ObjectId("55705734b6f63c9e80b92deb")	{ 4 fields }	Object
_id	ObjectId("55705734b6f63c9e80b92deb")	ObjectId
name	Document #3	String
number	3.000000	Double
shard_key	4.000000	Double
(4) ObjectId("55705734b6f63c9e80b92dec")	{ 4 fields }	Object
(5) ObjectId("55705734b6f63c9e80b92ded")	{ 4 fields }	Object
(6) ObjectId("55705734b6f63c9e80b92dee")	{ 4 fields }	Object
(7) ObjectId("55705734b6f63c9e80b92def")	{ 4 fields }	Object
(8) ObjectId("55705734b6f63c9e80b92df0")	{ 4 fields }	Object
(9) ObjectId("55705734b6f63c9e80b92df1")	{ 4 fields }	Object

Figuur 4-11. De data voor de aanpassing

De collectie is als ‘sharded’ aangemaakt en wordt nu ingedeeld in diverse chunks. MongoDB heeft hierop de volgende indeling gemaakt

Shard instantie	**Chunks**	**Aantal documenten**
Shard 1	Sharded key 2,3,4,5,6 of 7	75.000
Shard 2	Sharded key hoger dan 7	12.500
Shard 3	Sharded key 1	12.500
Shard 4	-	0

Wanneer nu op 1 van de Routing Servers alle documenten worden opgevraagd worden 100.000 documenten teruggegeven. De Routing Server zorgt er voor dat alle documenten verdeeld over de shards weer worden samengevoegd.

Wanneer op één van de shard instanties alle documenten worden opgevraagd, worden alleen de documenten teruggegeven die op die shard zitten.

Bij het controleren gaf Shard 1 75.000 documenten terug, Shard 2 gaf 12.500 documenten terug, Shard 3 gaf 12.500 documenten terug en Shard 3 gaf 0 documenten terug.

Om te kijken of het uitbreiden van het data model goed gaat wordt er op een van de router servers een commando uitgevoerd die 2 nieuwe velden toevoegt: een short_description en een image_url veld. De waarde van beide velden voor alle documenten is ‘dummywaarde’. Dit commando is zichtbaar in Figuur 4-12.

Er wordt daarna gekeken of deze aanpassing correct is doorgevoerd op alle shards. Als dat zo is kan er goed worden omgegaan met de uitbreiding van het documentschema en het schalen.

```
db.sCollection.update({}, { $set: {
    short_description: 'dummywaarde',
    image_url: 'dummywaarde'
}}, { multi: true });
```

Figuur 4-12. Het commando om 2 velden toe te voegen aan alle documenten

Na het uitvoeren van dit commando is er op elk van de 4 shards gekeken of de wijziging hier is doorgekomen. Dat was het geval, de aanpassingen waren direct zichtbaar in alle 4 de shards.

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, doordat de verdeling van de documenten op basis van de sharding key wordt veranderd.

4.4.5 Conclusie uitbreiding van het documentschema

In het geval van de uitbreiding van een documentschema met basale attributen kan MongoDB goed omgaan met deze wijzingen. De aanpassing is met een commando gemakkelijk door te voeren op het documentschema (paragraaf 4.4.2) en het blijkt dat de impact op de performance niet groot is. Uit de testen in 4.4 blijkt dat het verschil voor die situatie maar 13% is. Wel is het belangrijk om het effect van een uitbreiding op de performance altijd te meten voordat dit in productie wordt opgenomen. Hiermee worden onverwachte consequenties voorkomen.

De invloed op het gebied van schaalbaarheid is ook niet groot. De aanpassingen op het documentschema werden doorgevoerd op alle shards en de basale attributen hebben geen invloed op de verdeling van de documenten over de shards. Wel moet er bij elke uitbreiding gekeken worden of de shard key nog volstaat voor een collectie.

Er is daarmee te stellen dat MongoDB uitbreidbaarheid is. Het documentschema is zeer flexibel en is altijd aanpasbaar, het is zelfs mogelijk meerdere documentschema's te hebben per collectie. Uitbreidingen hebben wel effect op de performance, maar dit is niet zeer significant. Wanneer een mogelijke uitbreiding consequenties heeft over de verdeling van data zal er opgelet moeten worden of dit geen negatieve impact heeft op de schaling, doordat de verdeling van de documenten op basis van de sharding key wordt veranderd.

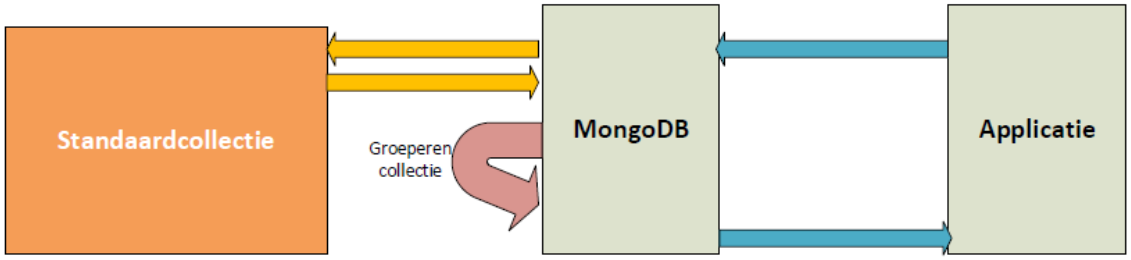
4.5 Toevoeging van een nieuw REST API endpoint

Managementsamenvatting

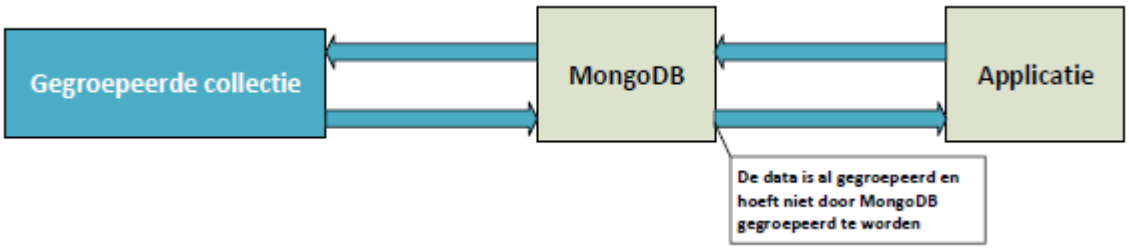
Dit hoofdstuk beschrijft de impact op het documentschema, performance en schaalbaarheid voor het uitbreiden van een REST API. Een ontwikkelde demo laat de impact zien. Er is uitgegaan van een situatie waarbij er geen nieuwe velden benodigd zijn, maar de data op een andere manier gerepresenteerd moet worden.

Er zijn twee mogelijke oplossingen:

1. Het ophalen van Knowledge Items die door MongoDB worden gegroepeerd op communities tijdens de request (Figuur 4-13);
2. Het ophalen van Knowledge Items uit een collectie waar deze van te voren gegroepeerd op communities zijn opgeslagen (Figuur 4-14).



Figuur 4-13 Tijdens het ophalen moet de collectie eerst in het gewenste formaat gegroepeerd worden.



Figuur 4-14 De collectie is al gegroepeerd opgeslagen en eerst aangepast te worden.

Vooraf opslaan van Knowledge Items gegroepeerd op communities heeft geen voordelen ten opzichte van het groeperen van de communities tijdens de request. De winst op het gebied van performance is minimaal. Het bijhouden van meerdere documentschema's brengt moeilijkheden met zich mee. Beide modellen moeten consistent worden gehouden, het normale model met alle Knowledge Items en het model gegroepeerd op communities.

Een nieuwe REST API heeft geen invloed op de uitbreidbaarheid, omdat het documentschema ongewijzigd blijft.

4.5.1 Inleiding

Een mogelijke uitbreiding van de knowNow Knowledge Service kan het toevoegen zijn van een REST API endpoint. Met een REST API kan er gecommuniceerd worden met de applicatie. knowNow maakt hiervoor gebruik van JSON. Het kan voorkomen dat er in de applicatie nieuwe functionaliteit toegevoegd moet worden, waarvoor de bestaande velden al gebruikt kunnen worden, maar alleen op een andere manier gerepresenteerd moet worden.

Wanneer het documentschema niet uitgebreid hoeft te worden heeft dit ook geen invloed op de uitbreidbaarheid. MongoDB kan data bijvoorbeeld zelf groeperen bij een request. Echter is het de vraag of dit in elke situatie gewenst is, misschien is het beter om de data van te voren in het gewenste formaat voor de functionaliteit op te slaan, hiermee wordt data gedupliceerd.

In dit onderzoek wordt antwoord gegeven op de volgende deelvraag:

- Wat zijn de consequenties van het toevoegen van een nieuw REST endpoint op het documentschema en de performance?

Voor het beantwoorden van deze deelvraag is in C# .NET een demo ontwikkeld waar door middel van een ASP.NET MVC Web API data opgehaald kan worden uit MongoDB met REST. Hiervoor is gebruik gemakt van het geschetste documentschema in 4.3. In deze demo zijn de volgende functionaliteiten ingebouwd:

- Het ophalen van Knowledge Items die door MongoDB worden gegroepeerd op communities tijdens de request;
- Het ophalen van Knowledge Items uit een collectie waar deze gegroepeerd op communities zijn opgeslagen.

4.5.2 Consequenties op het documentschema

Groeperen tijdens request
De eerste mogelijke oplossing, het tijdens de request groeperen van de Knowledge Items op community, heeft geen invloed op het documentschema. Deze wordt dan niet aangepast. De data wordt gegroepeerd op communities door middel van aggregatie functies die MongoDB aanbiedt. In de ontwikkelde demo is hiervoor de onderstaande code gebruikt.

```
DatabaseContext dbContext = DatabaseContext.getInstance();
Dictionary<string, List<Knowledge>> knowledgePerCommunityDictionary = new Dictionary<string, List<Knowledge>>();

var unwind = new BsonDocument {
   { "$unwind", "$communities"}
};

var group = new BsonDocument {
    {
        "$group",
        new BsonDocument
        {
            {
                "_id", "$communities.name"
            },
            {
                "result",
                new BsonDocument {
                    {
                        new BsonDocument {
                            {
                                "$push",
                                new BsonDocument {
                                    {new BsonDocument("_id","$_id")},
                                    {new BsonDocument("title", "$title")},
                                    {new BsonDocument("description", "$description")},
                                    {new BsonDocument("tags", "$tags")},
                                    {new BsonDocument("followers", "$followers")},
                                    {new BsonDocument("views", "$views")},
                                    {new BsonDocument("authors", "$authors")},
                                    {new BsonDocument("rating", "$rating")},
                                    {new BsonDocument("comments", "$comments")},
                                    {new BsonDocument("communities", "$communities")},
                                    {new BsonDocument("date", "$date")},
                                    {new BsonDocument("shard_key", "$shard_key")},
                                    {new BsonDocument("number", "$number")}
                                }
                            }
                        }
                    }
                }
            }
        }
    }
};

PipelineDefinition<BsonDocument, BsonDocument> pipeline = new BsonDocument[] { unwind, group };


using (var agg = await dbContext.getAggregateForCollection<BsonDocument>("sCollection", pipeline))
{
```

Figuur 4-15. De code voor het groeperen op community in C# .NET

Er worden eerst 2 BsonDocuments aangemaakt. Het eerste document, unwind, maakt van een Knowledge Item met meerdere communities voor elke community een eigen Knowledge Item aan.

Bijvoorbeeld wanneer Knowledge Item #1 communities A en B heeft wordt er door unwind een Knowledge Item #1 met alleen community A teruggegeven en een Knowledge Item #1 met alleen community B.

Bij het 2e document, group, wordt opgegeven dat de Knowledge Items gegroepeerd moeten worden op de naam van de communities en dat alle velden van de bijhorende Knowledge Items ook geretourneerd worden.

Deze 2 BsonDocuments worden verstuurd naar de MongoDB server die de Knowledge Items per community retourneert. Deze wordt door de C# .NET demo omgezet naar een JSON formaat. Dit resulteert in het resultaat in Figuur 4-16.


```
"Een community": [
    {
        "Id": "557fc595e214ac3a9e90cad7",
        "Title": "Dit is een titel van een Knwoledge Item #2",
        "Description": "<p>Dit is een omschrijving van een Knowledge Item #2</p>",
        "Tags": [
            "tag3-1",
            "tag3-2",
            "tag3-3",
            "tag3-4"
        ],
        "Followers": 15,
        "Views": 10,
        "Authors": [
            {
                "Name": "Tom Keim",
                "CoAuthor": false
            },
            {
                "Name": "Iemand anders",
                "CoAuthor": true
            }
        ],
        "Rating": {
            "Rate": 4.5,
            "Votes": 10
        },
        "Comments": [
            {
                "Author": "Tom Keim",
                "Text": "Dit is de reactie",
                "Date": "2015-06-02T00:00:00Z"
            },
            {
                "Author": "Iemand anders",
                "Text": "Dit is andere reactie",
                "Date": "2015-06-02T00:00:00Z"
            }
        ],
        "Communities": [
            {
                "Name": "Een community",
                "Public": true
            }
        ],
        "Date": "2015-06-02T00:00:00Z",
        "ShardKey": 3,
        "Number": 2
    }
],
"Een tweede com": [
    {
        "Id": "557fc594e214ac3a9e90cad6",
        "Title": "Dit is een titel van een Knwoledge Item #1",
        "Description": "<p>Dit is een omschrijving van een Knowledge Item #1</p>",
        "Tags": [
            "tag2-1",
            "tag2-2",
            "tag2-3",
            "tag2-4"
        ],
        "Followers": 15,
        "Views": 10,
        "Authors": [
            {
                "Name": "Tom Keim",
                "CoAuthor": false
            },
            {
                "Name": "Iemand anders",
                "CoAuthor": true
            }
        ],
        "Rating": {
            "Rate": 4.5,
            "Votes": 10
        },
        "Comments": [
            {
                "Author": "Tom Keim",
                "Text": "Dit is de reactie",
                "Date": "2015-06-02T00:00:00Z"
            },
            {
                "Author": "Iemand anders",
                "Text": "Dit is andere reactie",
                "Date": "2015-06-02T00:00:00Z"
            }
        ],
        "Communities": [
            {
                "Name": "Een tweede com",
                "Public": true
            }
        ],
        "Date": "2015-06-02T00:00:00Z",
        "ShardKey": 2,
        "Number": 1
    }
]
```


Figuur 4-16. Het resultaat van de gegroepeerde Knowledge Items

Het groeperen gebeurt door middel van enkele commando's. Het documentschema wordt niet aangepast voor deze optie.

Collectie met Knowledge Items gegroepeerd op communities
De tweede optie is het opslaan van de data waarbij de data al is gegroepeerd op communities. Deze optie heeft natuurlijk wel gevolgen voor het documentschema. Data wordt immers nogmaals opgeslagen volgens het gewenste formaat, in een andere collectie. Hierdoor ontstaat duplicatie van data. De data staat in de normale vorm in een collectie, en in een andere collectie gegroepeerd op de community. Het voordeel hiervan is dat er door een applicatie niet gegroepeerd hoeft te worden. Het nadeel is dat bij wijzigingen van Knowledge Items dit op meerdere plekken moet gebeuren.

Het resultaat van de vooraf gegroepeerde collectie is hetzelfde als bij Figuur 4-16. In de tabel hieronder zijn beide opties naast elkaar gezet.

Optie	Voordelen	Nadelen
Groeperen bij request	Geen dubbele datasets, de documenten hoeven niet op beide manieren opgeslagen te worden.	Moet bij het ophalen eerst elke keer worden gegroepeerd. Mogelijk performanceproblemen en moet ingebouwd worden in de applicatie.
Maken collectie met gegroepeerde Knowledge Items	Hoeft niet te worden gegroepeerd bij het ophalen van data.	Dubbele datasets. Bij wijzigingen moet er naar meerdere plekken worden geschreven.

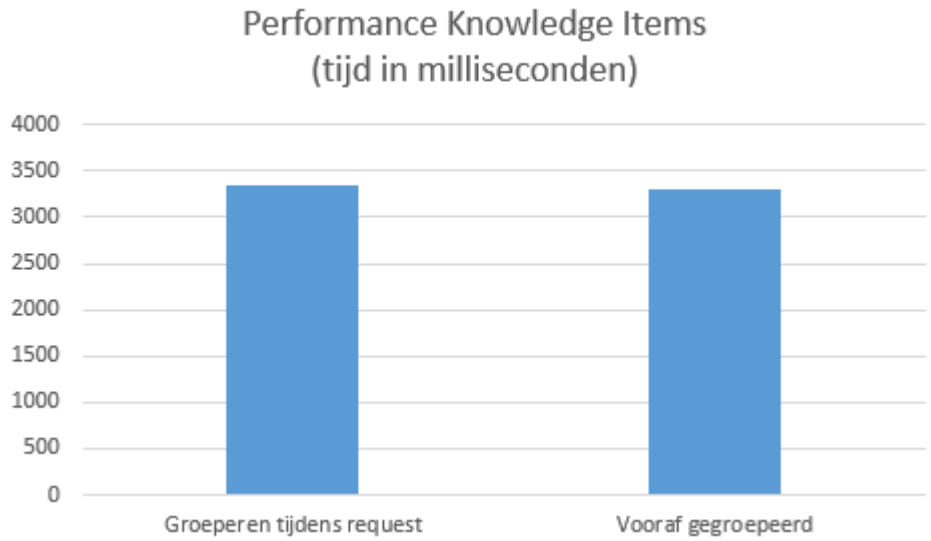
4.5.3 Consequenties op de performance

Om de consequenties van de opties op de performance te achterhalen zijn er, net als eerder in dit hoofdstuk, enkele performance testen uitgevoerd met JMeter.

In plaats van het direct bevragen van de MongoDB database gaat dat nu via de opgezette demo. JMeter doet een request naar de demo en de C# applicatie bevraagd de database en retourneert het resultaat naar JMeter. Één request het ophalen van een gehele collectie, waarin 1000 documenten zitten.

De eerste test is uitgevoerd voor de optie waarbij de gehele collectie wordt gegroepeerd tijdens de request. De throughput is 174,124 requests per minuut. De verwerktijd van één request is gemiddeld 3351ms. Er zijn in totaal 500 testen uitgevoerd, eerlijk verdeeld over 10 threads.

Diezelfde test is ook uitgevoerd voor de vooraf gegroepeerde collectie. De hele collectie wordt opgehaald. De throughput is hier 175,86/minuut. Deze oplossing kan per minuut 1 extra requests uitvoeren vergeleken met het groeperen tijdens de request. Ook de gemiddelde uitvoertijd is iets lager, 3298ms per request.



Figuur 4-17. Resultaat testen

Het verschil tussen het vooraf groeperen van de data in een collectie en het groeperen tijdens de request is minimaal, slechts 53ms, zoals te zien in figuur 4-17. Door dit kleine verschil is het voordeel op het gebied van performance zeer klein, het consistent houden van 2 collecties heeft een veel grotere impact. Data moet dan ook naar beide collecties geschreven worden.

Daarom is het verstandig om geen aanpassingen te doen aan het documentschema's bij deze uitbreiding, de nadelen zijn groter dan de voordelen:

Optie	**Voordelen**	**Nadelen**
Groeperen bij request	Geen extra documentschema nodig	Trager dan maken gegroepeerde collectie, verschil echter minimaal
Maken collectie met gegroepeerde Knowledge Items	Iets sneller dan groeperen bij request, verschil echter minimaal	Extra documentschema nodig, data moet consistent worden gehouden.

Er wordt geen test uitgevoerd voor de schrijfperformance. Het schrijven naar 2 collecties kost meer tijd dan schrijven naar 1 collectie en er dient extra logica geschreven te worden om ervoor te zorgen dat de data consistent blijft. Vanwege de kleine winst op het gebied van leesperformance is het vooraf groeperen in een collectie geen reële optie.

4.5.4 Conclusie

Het uitbreiden van een REST Interface met functionaliteit waarvoor geen nieuwe data nodig is, heeft geen impact op het documentschema. De beste optie blijkt voor de onderzochte situatie om bewerkingen aan de data uit te voeren doormiddel van een request, het volgens het gewenste formaat opslaan van data naast de al bestaande data heeft in de in dit hoofdstuk geteste situatie nauwelijks voordelen.

Wanneer je groepeert tijdens de request wordt het documentschema niet aangepast. Daardoor is waarschijnlijk(er is in dit onderzoek alleen gekeken naar de performance van de uitbreiding) ook er ook geen invloed op de performance of schaalbaarheid van bestaande functionaliteiten. Er wordt niets aan de MongoDB documenten en database aangepast. Het groeperen is enkel een commando wat de applicatie naar de MongoDB database stuurt.

Wanneer er wel data aangepast moet worden, zoals in 4.4 beschreven is, moet er natuurlijk wel rekening gehouden worden met mogelijke gevolgen op het gebied van performance en schaalbaarheid voor bestaande functionaliteiten.

4.6 Migraties

Managementsamenvatting

Dit hoofdstuk beschrijft of het mogelijk is een MongoDB database te migreren door middel van gestapelde updates. Dit gebeurt nu in de huidige applicatie al, maar dan met een SQL Server database.

Een ontwikkelde demo heeft laten zien dat het mogelijk is om MongoDB vanuit een applicatie te updaten en dat dit er aanpassingen gegaan kunnen worden op het schema maar ook aan de data. Ook bleek dat deze migraties ook te stapelen zijn.

4.6.1 Inleiding

De knowNow applicatie maakt gebruik van migraties. Met een migratie in knowNow is het mogelijk om een datamodel 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 plaats gevonden.

Een voorbeeld is wanneer een collectie op versie 1 draait. Versie 4 is inmiddels al beschikbaar. De applicatie zal dan eerst moeten updaten naar versie 2, daarna naar versie 3 en als laatst naar versie 4. Het is niet mogelijk in een keer te updaten naar versie 4.

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.

In dit hoofdstuk wordt bekeken of eenzelfde soort opzet ook mogelijk is met MongoDB als database. Er wordt gegeven op de volgende vraag:

> “Is het mogelijk een MongoDB database te migreren door middel van gestapelde updates?”

4.6.2 Huidige Situatie

De knowNow applicatie maakt gebruik van Orchard CMS migraties. Deze functionaliteit wordt standaard in Orchard aangeboden. In de Orchard migraties zijn de volgende functionaliteiten ingebouwd:

- Het toevoegen van nieuwe attributen in een Orchard datamodel;
- Het weghalen van attributen in een Orchard datamodel.

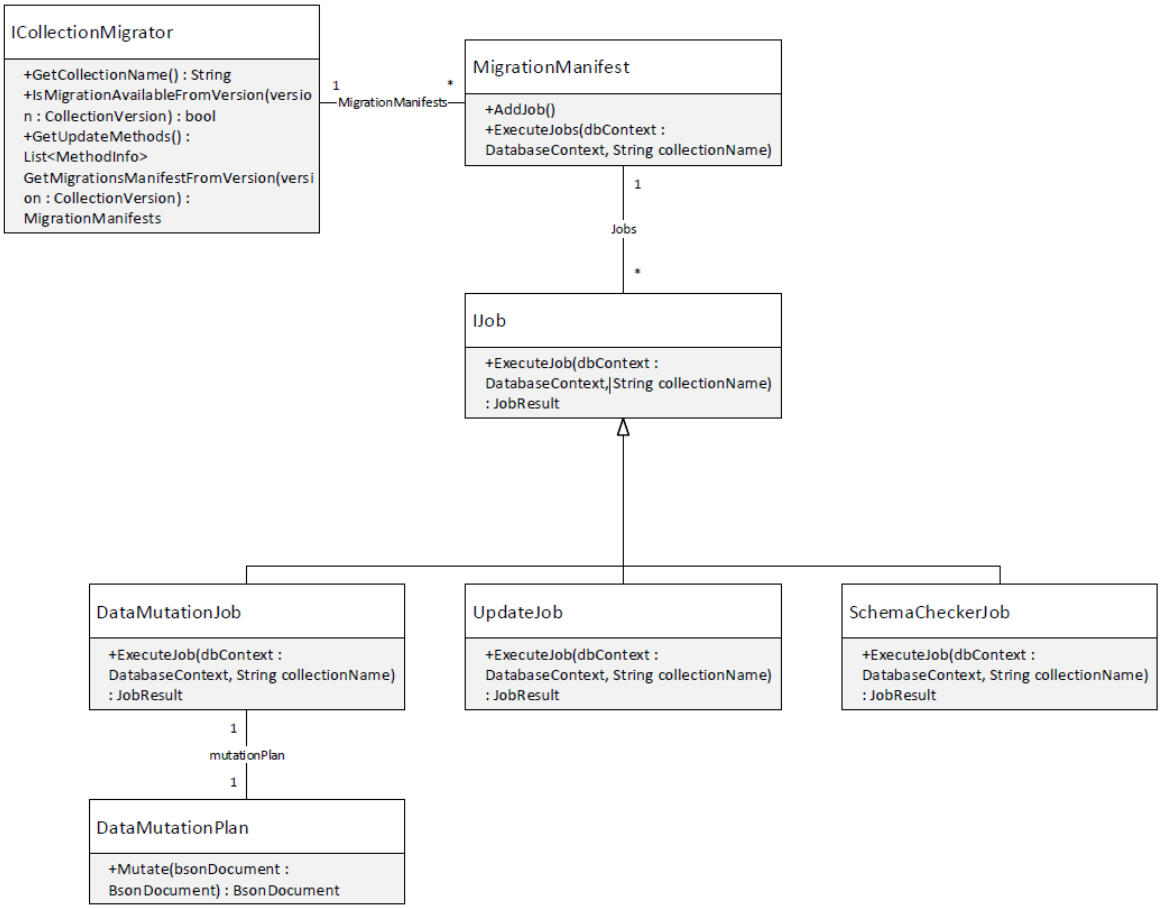
4.6.3 Migraties voor MongoDB

Om de deelvraag te kunnen beantwoorden is besloten een demo te ontwikkelen. Deze demo moet laten zien dat het mogelijk is een applicatie te ontwikkelen die één of meerdere MongoDB collectie(s) kan migreren. Voor het onderzoek is door de afstudeerder een demo ontwikkeld. De demo applicatie kan de volgende taken uitvoeren:

- **Het per collectie kunnen aanpassen van schema's van alle documenten**
  In de Orchard migratie was dit het toevoegen of verwijderen van attributen.
- **Controleren of alle documenten in een collectie voldoen aan een bepaald formaat**
  Deze functionaliteit zit niet in de Orchard migratie, 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.
- **Het muteren van data in documenten, bijvoorbeeld het samenvoegen van een voor- en achternaam.**
  Ook deze functionaliteit zat niet in Orchard, maar is wel meegenomen in de demo zodat duidelijk wordt hoe MongoDB om kan gaan met een grote hoeveelheid datamutaties.

Deze taken worden in de applicatie 'Jobs' genoemd. Één of meerdere 'Jobs' kunnen toegevoegd worden aan een 'MigrationManifest'. In een MigrationManifest staan alle taken die uitgevoerd moeten worden. In een CollectionMigrator staat voor elke MongoDB collectie omschreven welke MigrationManifest uitgevoerd moet worden om van de ene versie naar de andere versie te komen.

Klasse	Functie
Job	Er zijn 3 soorten jobs: • **UpdateSchema** Deze kan commando's uitvoeren via de update functie van MongoDB. Deze update wordt dan server-side uitgevoerd. Mogelijke updates zijn[B9]: ○ 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)[B10]. ○ Het verwijderen van een veld. • **DataMutationJob** Kan velden in een documenten document combineren. De bewerking van de data gebeurt in de C# applicatie. Daarom moeten alle documenten eerst van de server worden opgehaald. • **SchemaCheckerJob** Controleert of alle documenten in een collectie voldoen aan een opgegeven schema.
MigrationManifest	Aan een 'MigrationManifest' kunnen één of meerdere 'Jobs' worden toegevoegd. Bij het uitvoeren van een 'MigrationManifest ' worden alle 'Jobs' uitgevoerd op de volgorde waarop ze zijn toegevoegd.
CollectionMigrator	In een 'CollectionMigrator' wordt opgegeven om welke collectie het gaat, en welke MigrationManifests uitgevoerd moeten worden om de collectie naar de nieuwste versie te updaten.
DataMutationPlan	Het DataMutationPlan ontvangt een document via de Mutate functie. Daarop kan de klasse Mutaties uitvoeren en het aangepaste document retourneren. Het



Figuur 4-18. Klassendiagram van de Migratiedemo

Er is een situatie bedacht waarmee aangetoond kan worden dat de demo werkt. In deze situatie wordt de geschetste collectie in 4.3 uitgebreid met het attribuut 'short_description' (schema-aanpassing). Daarna wordt de rating van een 5-sterren rating naar een 10-sterren rating herrekend (datamutatie). Als laatste wordt gecontroleerd of het schema voldoet aan het nieuwe formaat.

Voor deze situatie is gekozen zodat alle 3 ontwikkelde 'Jobs' gebruikt worden.

Wordt uitgevoerd als	**Klasse**	**Omschrijving**
1e	UpdateJob	Past het schema aan van de documenten in de collectie. Voegt het veld "short_description" toe.
2e	DataMutationJob	Vermenigvuldigt het veld rating met 2. De stappen van een DataMutationJob kunnen worden omschreven in een DataMutationPlan.
3e	SchemaCheckJob	Controleert of het schema van alle documenten voldoet aan opgegeven schema.

In Figuur 4-18 op de volgende pagina is de initialisatie van het Manifest zichtbaar. Dit gebeurt in een 'CollectionMigrator', daarin worden alle Manifests geïnitialiseerd voor die collectie, voor elke versie.

Wanneer de huidige versie van een collectie 1 is en er Migratie Manifests zijn voor versie 2 en versie 3, voert de applicatie deze automatisch uit.

```
[Migration(fromVersion: 0)]
0 references
public MigrationManifest UpdateFrom0()
{
    MigrationManifest migrationManifest = new MigrationManifest();


    migrationManifest.AddJob(
        new SchemaChangeJob(
            new BsonDocument { },
            new BsonDocument {
                { "$set",
                    new BsonDocument {
                     new BsonDocument( "short_description", "" )
                }
            }
        })
       );

    migrationManifest.AddJob(new DataMutationJob(new RatingMutationPlan()));
    var requiredSchema = new BsonDocument {
        new BsonDocument("_id",1),
        new BsonDocument("title", 1),
        new BsonDocument("description",1),
        new BsonDocument("tags",1),
        new BsonDocument("authors", new BsonDocument {
            new BsonDocument("name",1),
            new BsonDocument("co_author",1)
        }),
        new BsonDocument("rating", new BsonDocument {
            new BsonDocument("rating",1),
            new BsonDocument("votes",1)
        }),
        new BsonDocument("followers",1),
        new BsonDocument("comments", new BsonDocument {
            new BsonDocument("author",1),
            new BsonDocument("comment",1),
            new BsonDocument("date",1)
        }),
        new BsonDocument("communities", new BsonDocument {
            new BsonDocument("name",1),
            new BsonDocument("public",1),
        }),
        new BsonDocument("date",1),
        new BsonDocument("short_description",1)
    };

    migrationManifest.AddJob(new SchemaCheckJob(new BsonDocument(requiredSchema)));


    return migrationManifest;
```

Figuur 4-19. De initialisatie van het Migratie manifest

Na het uitvoeren van de migratie is de collectie bekeken. De velden bleken toegevoegd te zijn en de data gemuteerd. De migratie was geslaagd.

De migraties zijn idempotent[7]. Voor de migraties houdt dit in dat een migratie per systeem maar één keer uitgevoerd wordt. Bijvoorbeeld wanneer de migratie van versie 0 naar versie 1 wordt uitgevoerd. De volgende keer dat de applicatie draait is de versie van de applicatie al 1 en hoeft de migratie van versie 0 naar versie 1 niet opnieuw uitgevoerd te worden.

Het toevoegen van een nieuwe migratie is eenvoudig. Daarvoor moet een nieuwe methode worden toegevoegd aan het manifest.

[7] Idempotent is een eigenschap van een object wat zegt dat het object niet meer veranderd als een operatie nogmaals wordt uitgevoerd[B11].

4.6.4 Server-side data mutaties

Via het update commando van MongoDB die op de server wordt uitgevoerd kunnen de volgende opdrachten worden uitgevoerd[B9]:

- 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)[B10].
- Het verwijderen van een veld.

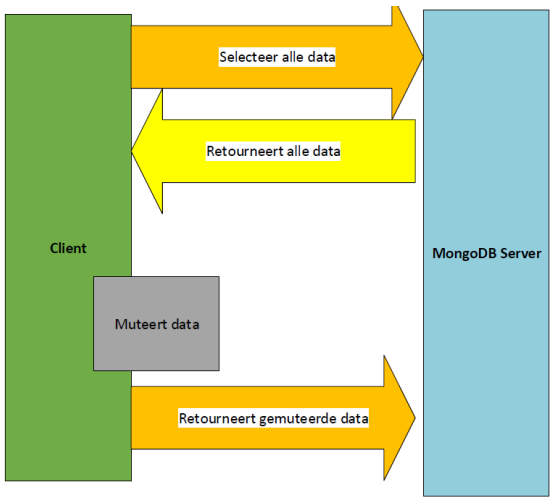
Het is niet mogelijk om een waarde van een veld te muteren als dit niet het optellen of multipliceren van die waarde betreft. Het toevoegen van een tekst aan een bestaande tekst is bijvoorbeeld niet mogelijk via het update commando[B10].

De migratie waarbij data gemanipuleerd kan worden, door middel van de ontwikkelde DataMutationJob haalt alle data op van de server, past de data aan, en stuurt de aangepaste data daarna terug naar de server (Figuur 4-20). Hierdoor is het toch mogelijk data te manipuleren.

Met het manipuleren van data wordt in dit hoofdstuk het aanpassen van velden in een document bedoeld, waaraan iets veranderd moet worden aan het al bestaande veld. De data van het veld is dus voor de mutatie nodig. Het geheel vervangen van data van een veld is wel mogelijk op de server en kan ook worden uitgevoerd via de UpdateJob.

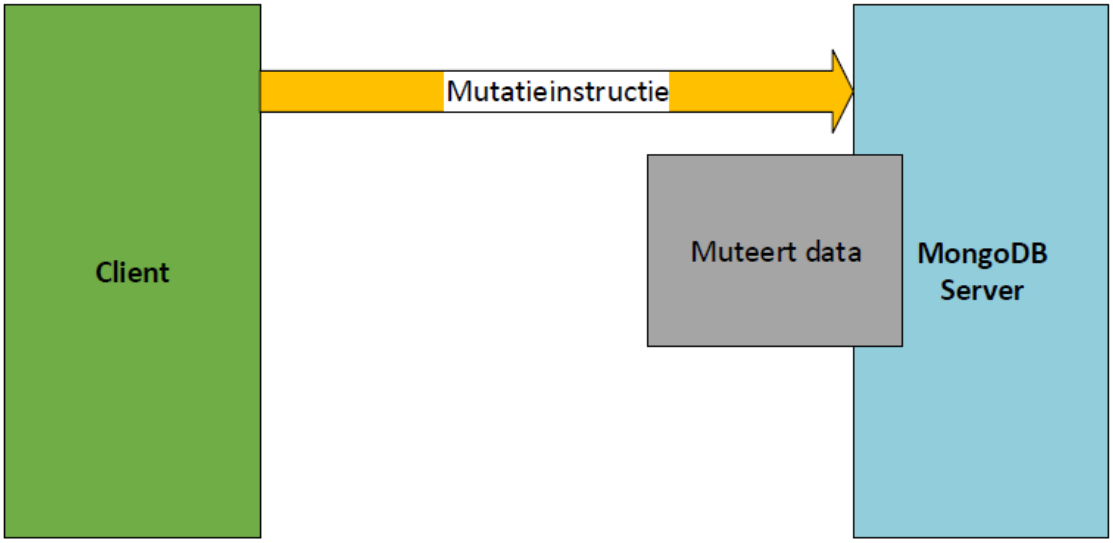
Een voorbeeld hiervan is het toevoegen van een Achternaam aan de Voornaam van een persoon. De Voornaam stond al in de data, en de achternaam moet er dus aan toegevoegd worden.

Het nadeel is dat de data bij deze oplossing heen en weer wordt verstuurd tussen de C# applicatie en de server.



Figuur 4-20. De data moet heen en weer worden gestuurd tussen de Client en de Server om het te kunnen muteren

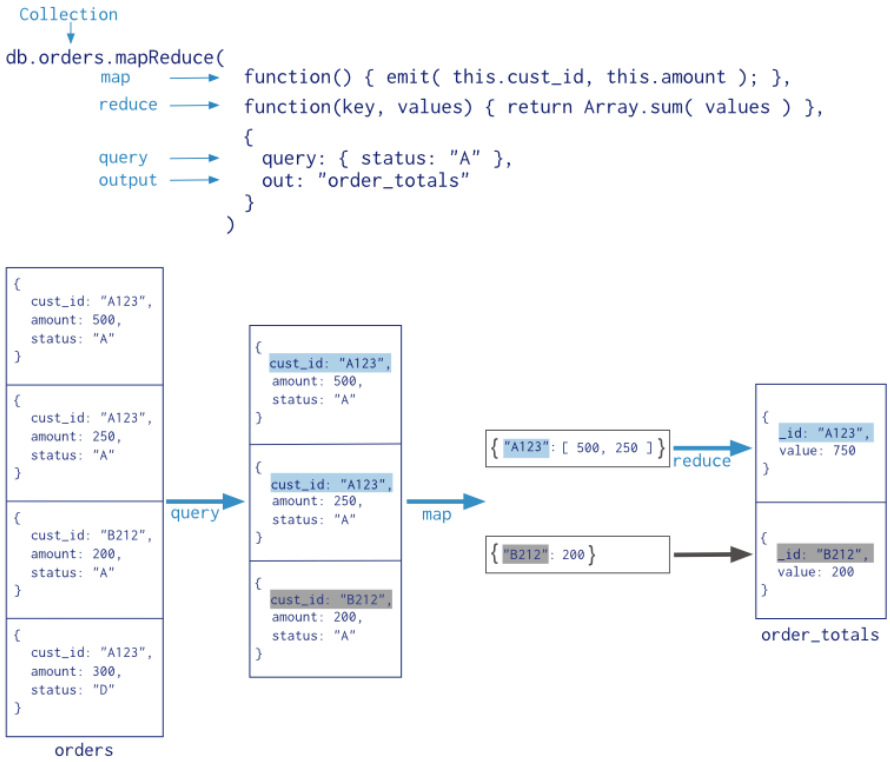
Een mogelijke betere oplossing is om de instructies voor de datamutatie naar de MongoDB server te sturen en deze op deze instructie op de server uit te voeren. (Figuur 4-21) Bij deze oplossing moet de C# applicatie dus alleen de instructie aan de server geven en hoeft er geen data heen en weer verzonden te worden.



Figuur 4-21. De mutatie wordt uitgevoerd op de MongoDB server

Echter blijkt het niet mogelijk om een script te draaien op een MongoDB server zelf welke data kan manipuleren. De enige manieren waarop javascript direct uitgevoerd kan worden op de server is via een mapReduce en $where commando[B5].

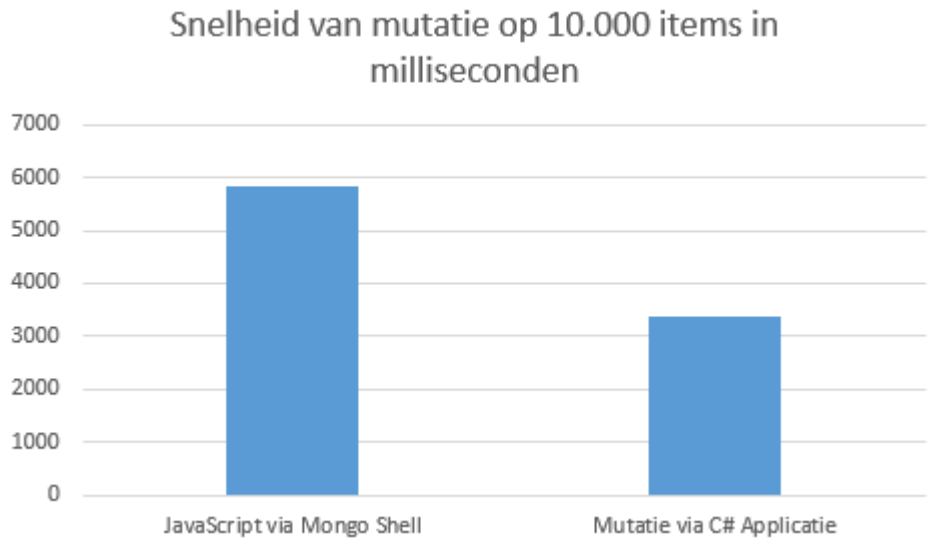
In de tabel op de volgende pagina worden mapReduce en $where toegelicht.

mapReduce	Bij mapReduce wordt een script naar de MongoDB server geschreven. Deze mapped eerst de data en reduced deze daarna. De data wordt eerst getransformeerd naar een geschikt formaat (mappen) en daarna reduceert gewenste formaat (reduce)[B6]. In het voorbeeld van Figuur 4-22 wordt voor elke klant op basis van cust_id de totale omzet berekend.  **Figuur 4-22. Een voorbeeld van mapReduce** *(bron: http://docs.mongodb.org/manual/core/map-reduce/)* Echter is het bij mapReduce alleen mogelijk data te retourneren naar de client of in een nieuwe collectie te plaatsen[B7]. mapReduce kan niet worden gebruikt om data binnen een collectie te wijzigen, omdat die collectie al bestaat.
$where	Met $where is het mogelijk om via JavaScript data te filteren[B8]. Zo kan er bijvoorbeeld in een JavaScript functie een vergelijking worden toegevoegd. Deze functie wordt uitgevoerd op de MongoDB server. Wanneer data aan deze vergelijking voldoet retourneerd MongoDB deze data naar de client. Omdat $where een filter methode is, kun je met where geen data manipuleren.

Bij het uitvoeren van Javascript via de Mongo Shell, wordt de data via de Mongo Shell uitgevoerd op de MongoDB server[B5]. Hierdoor is er alsnog een applicatie (Mongo Shell) die de data eerst moet ophalen van de server, daarna manipuleert en daarna weer retourneert naar de server. Deze oplossing heeft alsnog een tussenlaag.

Uit een test opgezet op een lokale omgeving blijkt zelfs, dat het gebruik van de Mongo Shell op een lokale machine langzamer is dan het gebruik van manipulatie via een C# applicatie op diezelfde lokale machine.

Via de Mongo Shell duurt een migratie waarbij de rating van een 10.000 Knowledge Item wordt vermenigvuldig met 2 na 3 testen gemiddeld 5845 milliseconden, via de C# applicatie duurt het voor diezelfde 10.000 items na 3s testen gemiddeld 3373 milliseconden.



Figuur 4-23. De testresultaten in een grafiek

Daarmee is aangeduid dat er op dit moment geen betere alternatieven zijn dan het manipuleren via een client applicatie. Het direct op de MongoDB server manipuleren van data is niet mogelijk.

4.6.5 Conclusie

Uit de ontwikkelde demo is gebleken dat het mogelijk is om migraties te stappelen voor collecties in MongoDB. Zowel het aanpassen van het schema, het aanpassen van data en het controleren van schema's is in de demo mogelijk. Daarom is te stellen dat gestapelde migraties, zoals nu al in het huidige knowNow gebeurt ook mogelijk is.

Echter, is het op dit moment alleen mogelijk om mutatie aan data client-side uit te voeren. Dit kan tot performance problemen leiden bij migraties van grote hoeveelheden documenten. Echter, geld dit alleen wanneer data zelf gemanipuleerd moet worden. Aanpassingen aan het schema kunnen wel server-side worden uitgevoerd.

4.7 Conclusie Deelvraag 2

Het uitbreiden van een documentschema heeft een kleine invloed op de performance. Door het flexibele documentschema van MongoDB is het uitbreiden van enkele, of alle documenten in een collectie geen probleem. Echter is het wel aan te raden de schema's van documenten in een collectie zo consistent mogelijk te houden. Anders moet extra logica in de applicatie geschreven worden.

Ook komt uit het onderzoek naar voren dat het anders weergeven van data voor een REST endpoint geen gevolgen heeft grote impact heeft op de performance. Bij een andere representatie van data is het niet nodig om deze data ook op een andere manier op te slaan in de database voor deze situatie.

Uit een ontwikkelde demo blijkt het stapelen van migraties voor een MongoDB collectie mogelijk.

Hierdoor is de uitbreidbaarheid van MongoDB voor knowNow groot. Aanpassingen aan het documentschema hebben weinig gevolgen, en zijn eenvoudig door te voeren. Het updaten van een MongoDB documentschema is mogelijk doormiddel 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. Echter is het belangrijk om wel voor elke aanpassing altijd te kijken of er onverwachte consequenties zijn op het gebied van performance en schaalbaarheid. En of de uitbreiding het geringe, maar wel aanwezige, performance verlies waard is.

5. Deelvraag 3: Hoe kan de Knowledge Service uiteindelijk consistent worden gemaakt met andere services?

5.1 Aanpak

In paragraaf 3.2 is gekozen voor een Microservice architectuur, de Knowledge Service. Hierin wordt de knowNow applicatie opgedeeld in verschillende kleinere services die elk hun eigen verantwoordelijkheden hebben.

Wanneer de Knowledge Service alle data van een Knowledge Item bevat is het belangrijk dat deze gegevens consistent met elkaar worden. Wanneer bijvoorbeeld de naam van een auteur wordt geüpdate in de People Service, moet dit ook in alle Knowledge Items worden bijgewerkt.

Zeker bij een grootte hoeveelheid Knowledge Items kan dit enige tijd duren. Doordat MongoDB alleen (atomaire) transacties op document niveau kent en niet op collectie niveau[B14] zal bij het falen van een update het kunnen voorkomen dat een deel van de documenten wel de geüpdate naam bevat, en een deel van de documenten niet. Hierdoor raakt de database inconsistent.

Dit mag niet voorkomen in de productieomgeving. De gebruikers van knowNow zien dan niet altijd de juiste data. Daarom is besloten te onderzoeken of het mogelijk is om ervoor te zorgen dat de data in de Knowledge Service uiteindelijk consistent wordt met de data uit de andere services.

Er wordt antwoord gegeven op de volgende deelvraag:

> *"Hoe kan de Knowledge Service uiteindelijk consistent worden gemaakt met andere services?"*

In dit hoofdstuk wordt gekeken naar 3 verschillende mogelijkheden:

- 2-Phase-Commit
- Transactional Replication
- Job Queue

Deze mogelijkheden zijn gevonden door het raadplegen van diverse bronnen zoals de MongoDB documentatie en diverse zoekopdrachten. Elk van de gevonden mogelijkheden zal worden vergeleken en er wordt een keuze gemaakt aan de hand van de voor- en nadelen per oplossing. Na het kiezen van een oplossing zal deze verder worden uitgewerkt zodat deze kan worden geïmplementeerd in het Proof-of-Concept. Het Proof-of-Concept zal samen met een test bewijzen of de oplossing ook daadwerkelijk werkt.

5.2 Situaties

De oplossing moet om kunnen gaan met de onderstaande situaties. Deze situaties zijn opgesteld om de gevonden mogelijkheden te kunnen toetsen. De 2 situaties zijn tot stand gekomen omdat dit situaties zijn die er in het op collectieniveau transactie loze MongoDB voor zorgen dat de database inconsistent raakt.

- **Situatie 1:** Wanneer MongoDB uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de update opnieuw uitgevoerd worden zodra de server weer bereikbaar is.
- **Situatie 2:** Wanneer de Knowledge Service uitvalt tijdens uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de transactie opnieuw uitgevoerd worden zodra de Knowledge Service weer is opgestart.

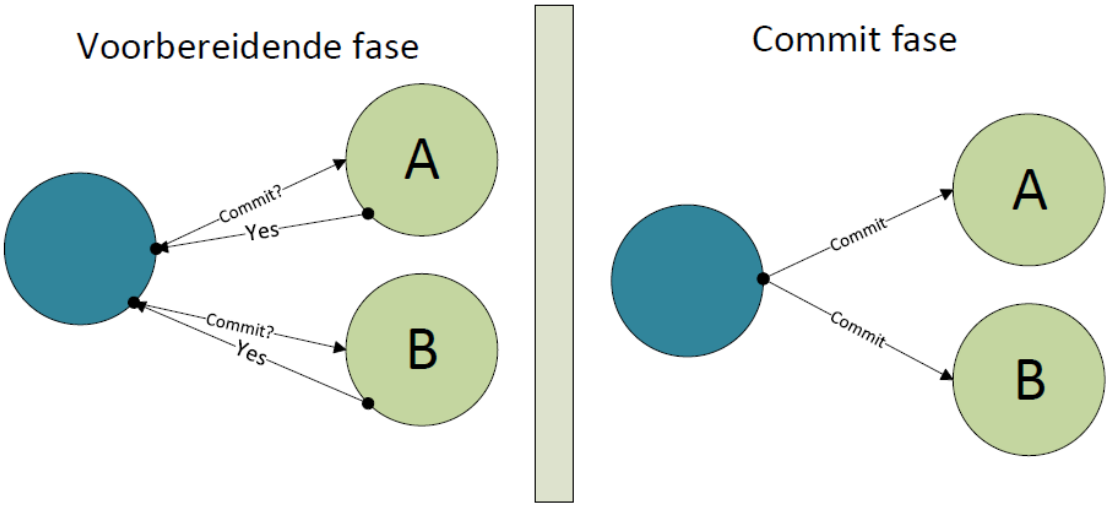
5.3 Mogelijkheden

5.3.1 2-Phase-Commit

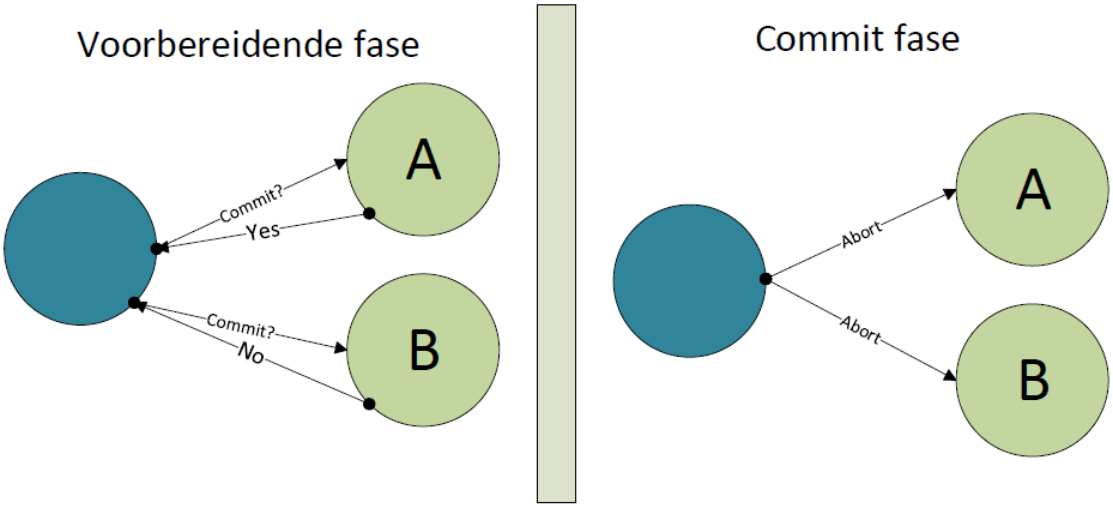
Een 2-Phase-Commit, hierna afgekort naar 2PC, is een algoritme waarmee het mogelijk is om bij het gebruik van meerdere databases of een transactie die meerdere rijen raakt ervoor te zorgen dat een transactie op alle databases wordt uitgevoerd of bij een fout helemaal niet[B16].

Het 2PC algoritme bestaat uit 2 fasen:

1. **Voorbereidende fase.** In deze fase wordt de transactie voorbereid. De applicatie neemt contact op met de databases die voor de transacties gebruikt en maakt duidelijk wat er voor de transactie moet gebeuren. Elke database stemt daarna. Een database stemt “commit” wanneer de database de transactie kan verwerken en stemt “abort” wanneer de transactie niet verwerkt kan worden.
2. **Commit fase.** In deze fase wordt wanneer alle databases “commit” hebben gestemd de transactie op alle databases verwerkt. Wanneer één of meer databases “abort” heeft gestemd wordt de transactie niet uitgevoerd.



Figuur 5-1. Een Two Phase Commit waarbij de transactie slaagt



Figuur 5-2. Een Two Phase Commit waarbij de transactie faalt

De 2PC wordt ook door MongoDB benoemd op hun website[B12]. Hierin is een voorbeeld geschetst waarin 2PC wordt toegepast. MongoDB biedt zelf geen implementatie voor 2PC. De logica van een 2PC zal moeten worden geïmplementeerd in de applicatie.

Het 2PC algoritme kan in MongoDB worden gebruikt voor een transactie die 2 of meer documenten in een collectie raakt. Bijvoorbeeld bij het overschrijven van saldo tussen 2 rekeningen. Hierbij wordt in de voorbereidende fase opgeslagen wat er moet veranderen, wat de oude waarde is en wat de status van de transactie is.

Bij 2PC voor MongoDB wordt in feite de functionaliteiten die een Relationele Database kent met ACID transacties nagebootst. Dit gebeurt echter niet door MongoDB maar in de applicatie die met MongoDB communiceert.

Het voordeel van 2PC voor MongoDB is dat het mogelijk is een transactie terug te draaien wanneer deze faalt. Bij bijvoorbeeld het overmaken van een saldo tussen 2 rekeningen kan er door de 2PC voor worden gezorgd dat bij een fout tijdens het overschrijven de saldi van de rekening kan worden teruggezet waardoor er geen geld verloren gaat.

Echter is het terug draaien van een transactie voor de benodigde oplossing niet nodig, de oplossing dient er alleen voor te zorgen dat de data uiteindelijk consistent is, een rollback is niet nodig omdat de oude data toch niet meer geldig is. De transactie zal juist opnieuw uitgevoerd moeten worden en dat kan met de data die in de andere service staat.

Het grootte nadeel van 2PC voor MongoDB is dat er functionaliteiten die ingebouwd zijn in een Relationele Database door de applicatie worden nagebootst. Er dient een ingewikkeld algoritme in een applicatie te worden geïntegreerd waarvoor veel ervaring nodig is.

Om het 2PC algoritme te laten voldoen aan alle situaties zoals geformuleerd in 5.2 dient er naast het implementeren van het algoritme de volgende logica te worden ingebouwd:

Situatie	**Hoe te voldoen**
Situatie 1: Wanneer MongoDB uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de update opnieuw uitgevoerd worden zodra de server weer bereikbaar is.	Wanneer MongoDB uitvalt zal de Knowledge Service de transactie “aborted” worden. De Knowledge Service zal de “aborted” transactie dan opnieuw kunnen uitvoeren door de transactie opnieuw uit te voeren.
Situatie 2: Wanneer de Knowledge Service uitvalt tijdens uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de transactie opnieuw uitgevoerd worden zodra de Knowledge Service weer is opgestart.	Bij het uitvallen van de Knowledge Service zal de transactie nog niet zijn afgerond en staat dus open. De Knoweldge Service zal bij het opstarten moeten kijken welke transacties er open staan en deze opnieuw starten.

Het 2PC algoritme voor MongoDB heeft de volgende voor- en nadelen:

Voordeel	**Nadeel**
• Voldoet aan beide opgestelde situaties • Een gefaalde transactie kan worden teruggedraaid.	• Applicatie bootst functionaliteit van een RDBMS na. Algoritme is ingewikkeld en er is veel kennis nodig om het goed te kunnen implementeren. • Doet meer dan nodig. Het 2PC algoritme is bedoeld om transacties terug te kunnen draaien, maar voor de situatie moet een transactie juist hervat worden.

5.3.2 Transactional Replication

Transactional Replication is het her afspelen van een logbestand wanneer er een fout op is getreden[B15]. In dit logbestand zijn alle transacties behalve leestransacties die plaatsvinden op de MongoDB database opgeslagen.

Bij Transactional Replication wordt voordat een transactie wordt gestart deze transactie opgeslagen in een log. De transactie zal daarna worden uitgevoerd. Wanneer de transactie voltooid is wordt dit ook opgeslagen in het log.

Wanneer de transactie om welke reden dan ook faalt kan op basis van het logbestand de transactie opnieuw worden gestart. In het logbestand was de transactie namelijk nog niet als klaar aangemerkt.

Op Transactional Replication te laten voldoen aan alle situaties zoals geformuleerd in 5.2 dient er logica in de Knowledge Service te worden gebouwd. Dit is in de tabel hieronder beschreven.

Situatie	Hoe te voldoen
Situatie 1: Wanneer MongoDB uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de update opnieuw uitgevoerd worden zodra de server weer bereikbaar is.	Wanneer MongoDB uitvalt zal de Knowledge Service net zo lang moeten wachten totdat de MongoDB server weer beschikbaar is. Hierna kan deze de transactie in zijn geheel opnieuw uitvoeren.
Situatie 2: Wanneer de Knowledge Service uitvalt tijdens uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de transactie opnieuw uitgevoerd worden zodra de Knowledge Service weer is opgestart.	Bij het uitvallen van de Knowledge Service zal de transactie in het log niet als 'klaar' zijn aangemerkt. Wanneer de Knowledge Service opnieuw opstart zal deze moeten kijken of er nog niet-afgeronde transacties in het log staan. Als dat het geval is zal de Knowledge Service de transactie in zijn geheel opnieuw afspelen.

Het voordeel van Transactional Replication is dat het op basis van het logbestand altijd mogelijk is de volledige MongoDB database opnieuw op te bouwen. Alle transacties zijn hier namelijk in opgeslagen en wanneer alle transacties opnieuw worden uitgevoerd wordt dezelfde staat van de database weer bereikt. Ook kan deze mogelijkheid, na implementatie van enige logica in de Knowledge Service omgaan met beide geschetste situaties als omschreven in 5.2.

Het nadeel is echter dat het logbestand groot kan worden, alle transacties worden namelijk opgeslagen in het logbestand. De vraag is of dit nodig is. Wanneer een transactie maar één document raakt hoeft dit eigenlijk niet te worden opgenomen in het logbestand. Alleen transacties die bedoeld zijn voor het uitvoeren van een update uit een andere service hoeven opgeslagen te worden en her uitgevoerd worden wanneer het mis gaat om te voldoen aan beide situaties. Het is ook niet nodig om door middel van het her afspelen van het log de database te kunnen herstellen. Er worden immers backups gemaakt van de database.

Voordeel	Nadeel
• Voldoet aan beide opgestelde situaties • Gehele log kan opnieuw worden afgespeeld	• Logbestand groeit met de tijd en kan zeer groot worden • Bevat alle transacties, ook transacties die niet nodig zijn.

5.3.3 Job Queue

De laatste mogelijkheid is de Job Queue. De Job Queue lijkt veel op de Transactional Replication mogelijkheid en is hier een afgeslankte versie van. De Job Queue is geschetst op basis van de situaties beschreven in 5.2.

Het idee van een Job Queue is een wachtrij van 'jobs'. Jobs zijn in dit geval alleen de transacties die uitgevoerd worden op de Knowledge Service om die weer consistent te maken na een update vanuit een andere service. Hierdoor bevat de Job Queue alleen de transacties en de functionaliteit die nodig zijn voor het consistent houden van de Knowledge Service.

Wanneer een update binnenkomt bij de Knowledge Service, bijvoorbeeld die van gewijzigde naam van een auteur wordt deze transactie eerst geregistreerd in de Job Queue. In deze 'Job' wordt opgeslagen wat er verandert en waarin het wordt veranderd. Nadat de update volledig is uitgevoerd wordt de transactie uit de Job Queue gehaald, deze is namelijk niet meer nodig, de transactie is afgerond. Echter, wanneer er fouten optreden, blijft de 'job' in de Queue staan, en kan deze wanneer het systeem klaar is om de transactie opnieuw te verwerken, de 'job' opnieuw uitvoeren.

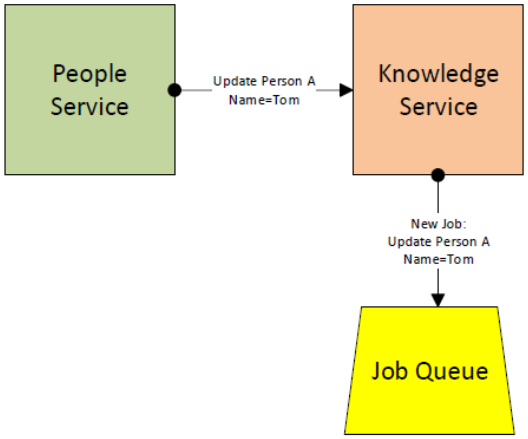
De Job Queue moet idempotent zijn. Dit houdt in dat wanneer de Job Queue opnieuw wordt uitgevoerd, de data in dezelfde staat komt. Om dit te bereiken moeten Jobs die binnenkomen in de Queue altijd op de volgorde worden uitgevoerd van binnenkomst. Het kan niet zo zijn dat wanneer een gebruiker zijn emailadres aanpast, hier een typefout in maakt en deze corrigeert dat eerst de update met het gecorrigeerde emailadres wordt uitgevoerd en daarna de update met het foutieve emailadres. De data bevat dan het foutieve emailadres en het gecorrigeerde emailadres gaat verloren.

In de tabel hieronder is het proces beschreven hoe de situaties zoals omschreven in 5.2 opgelost moeten worden.

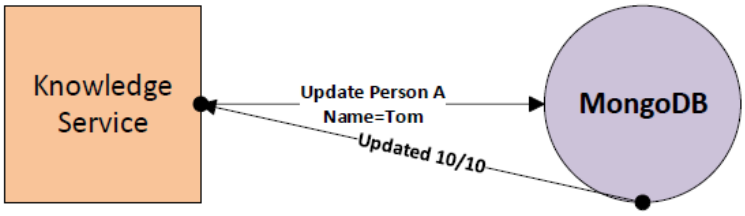
Situatie	Hoe te voldoen
Situatie 1: Wanneer MongoDB uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de update opnieuw uitgevoerd worden zodra de server weer bereikbaar is.	Wanneer MongoDB uitvalt zal de Knowledge Service de transactie in de Job Queue laten zitten. De Knowledge Service dient in de gaten te houden wanneer de MongoDB database weer beschikbaar is. Wanneer dat het geval is kan de Knowledge Service de transactie opnieuw uitvoeren.
Situatie 2: Wanneer de Knowledge Service uitvalt tijdens uitvalt tijdens een update van een persoon uit de People Service die wordt verwerkt in de Knowledge Service moet de transactie opnieuw uitgevoerd worden zodra de Knowledge Service weer is opgestart.	Bij het uitvallen van de Knowledge Service zal de transactie nog in de Job Queue staan. Wanneer de Knowledge Service opnieuw opstart zal deze moeten kijken of er nog niet-afgeronde transacties in de Job Queue staan. Als dat het geval is zal de Knowledge Service de transactie opnieuw uitvoeren.

In de figuren hieronder wordt de werking van de Job Queue verduidelijkt.

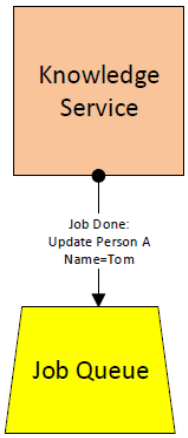
Geslaagde update



Stap 1: Er komt een update binnen vanuit de People Service dat de naam van Persoon A aangepast moet worden.

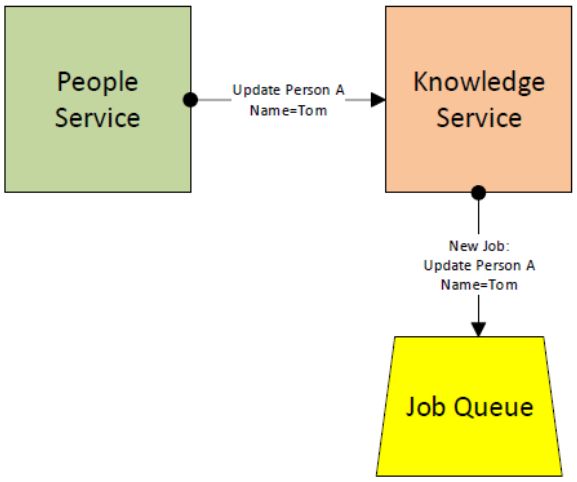


Stap 2: De Knowledge Service stuurt de update naar MongoDB. MongoDB retourneert dat de update is geslaagd.

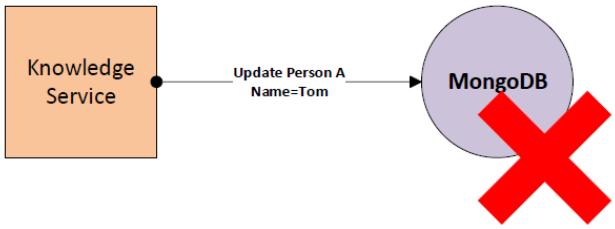


Stap 3: De Knowledge Service haalt de Job uit de Queue

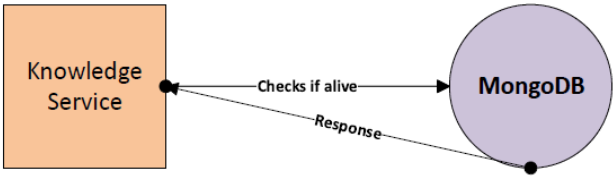
MongoDB valt uit tijdens update (situatie 1)



Stap 1: Er komt een update binnen vanuit de People Service dat de naam van Persoon A aangepast moet worden.

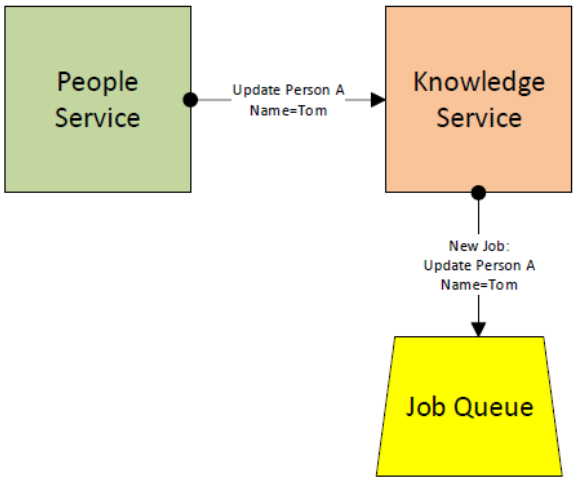


Stap 2: De Knowledge Service stuurt de update naar MongoDB. Echter, MongoDB is niet bereikbaar of valt tijdens de update uit.

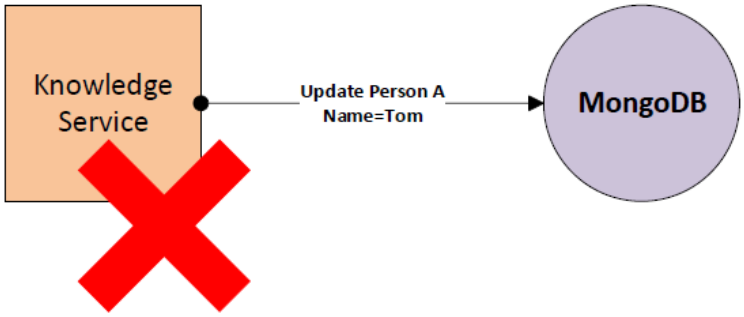


Stap 3: De Knowledge Service blijft controleren of MongoDB weer te gebruiken is. Als dat zo is start de Knowledge Service de Job opnieuw

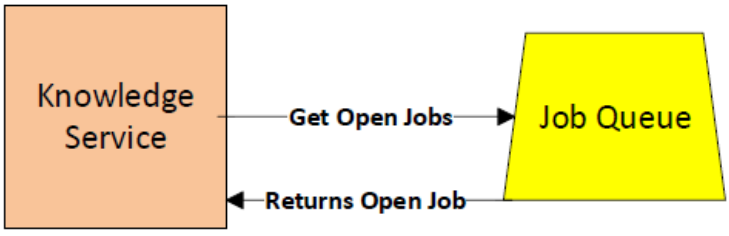
Knowledge Service valt uit tijdens update (Situatie 2)



Stap 1: Er komt een update binnen vanuit de People Service dat de naam van Persoon A aangepast moet worden.



Stap 2: De Knowledge Service stuurt de update naar MongoDB. Echter, de Knowledge Service valt daarna uit. De Job in de Job Queue blijft bestaan.



Stap 3: De Knowledge Service wordt weer opgestart en kijkt in de Job Queue of er nog open Jobs zijn.

Stap 4: Alle open Jobs worden opnieuw uitgevoerd

Het voordeel van de Job Queue is dat dit exact doet wat nodig is: het consistent maken van de Knowledge Service wanneer er tijdens het uitvoeren van zo'n update iets mis gaat. In het Job Queue staat dan een niet afgeronde Job die opnieuw uitgevoerd kan worden.

Voordeel	Nadeel
• Voldoet aan beide opgestelde situaties • Bevat alleen functionaliteit die nodig is om te voldoen aan de opgestelde situaties.	

5.4 Conclusie

Er is gekozen voor de Job Queue. Alhoewel alle mogelijkheden om kunnen gaan met de geschetste situaties is de Job Queue het eenvoudigst te implementeren en doet dit precies wat nodig is. De 2-Phase-Commit en Transactional Replication oplossingen doen meer dan dat en zijn ingewikkelder te begrijpen. Bij Transactional Replication worden alle transacties opgeslagen, terwijl bij de Job Queue alleen de transacties worden opgeslagen die nodig zijn voor het kunnen ondersteunen van de situaties van 5.2. Wat leid tot een duidelijkere te begrijpen implementatie en dus een betere onderhoudbaarheid.

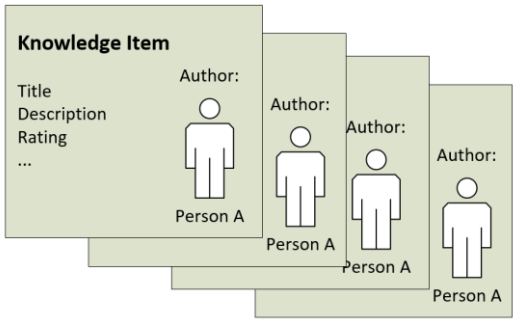
Bij de 2-Phase-Commit wordt er logica nagebouwd in de applicatie die eigenlijk bedoeld is voor het kunnen terugdraaien van transacties die meerdere documenten in een collectie raken, wat voor de situaties niet nodig is.

De Job Queue wordt verder beschreven in het Technisch Ontwerp en wordt gerealiseerd in het Proof-of-Concept. Er wordt door middel van testen gekeken of de gekozen oplossing ook in praktijk werkt.

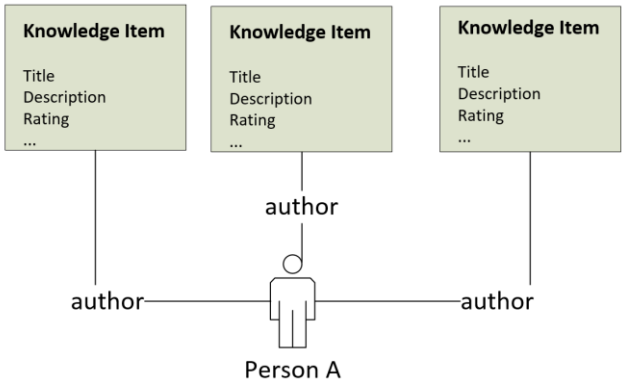
6. Deelvraag *4: Is het voor knowNow beter een hiërarchisch documentschema te hanteren of een relationeel documentschema?*

MongoDB heeft geen ondersteuning voor joins[A6]. MongoDB kan tijdens het ophalen van data geen relaties bij elkaar zoeken zoals vaak in een RDBMS gebruikelijk is. Om toch een relatie te leggen tussen documenten in MongoDB kunnen er 2 varianten worden toegepast:

1. Hiërarchisch documentschema: De gegevens van de relatie opnemen als subdocument[8] (Figuur 6-1).
2. Relationeel documentschema: Verwijzen naar de locatie waar de gegevens van de gegevens van de relatie te vinden zijn. De applicatie zal daarna zelf de relatie op moeten halen (Figuur 6-2).



Figuur 6-1. Een documentschema waarbij data van relaties is opgenomen in een subdocument (Hiërarchisch documentschema)



Figuur 6-2. Een documentschema waarbij er alleen gerefereerd wordt naar de relaties (Relationeel documentschema)

In eerste instantie werd er in het voorgestelde documentschema voor knowNow van uitgegaan om van het Hiërarchische documentschema. Bij dit documentschema kan bij het ophalen van een document in één keer alle data terug worden gegeven en hoeft de applicatie niet eerst data bij elkaar te zoeken. In deze situatie is het van belang dat de data in de subdocumenten consistent wordt gehouden als bijvoorbeeld een auteur zijn naam aanpast. De naam van de auteur moet dan in alle subdocumenten worden aangepast.

In Deelvraag 3 is onderzocht hoe ervoor gezorgd kan worden dat de als subdocument opgenomen relaties uiteindelijk consistent gemaakt kunnen worden. Uit dit onderzoek bleek de Job Queue de betere optie te zijn.

[8] Een subdocument is een document in een document. Zoals te zien in figuur 6.1

Echter, deed de Job Queue een nieuwe vraag ontstaan. Is het opnemen van alle data die op een Knowledge Item pagina staat wel daadwerkelijk de beste oplossing? Deze keuze was berust op een advies van MongoDB zelf en op basis van het onderzoek uitbreidbaarheid waaruit bleek dat het toevoegen van nieuwe attributen geen grote impact had op de performance. Echter, dient er voor bijvoorbeeld het updaten van een als subdocument opgenomen auteur, extra functionaliteit in de applicatie geïmplementeerd te worden die ervoor moet zorgen om de data consistent te houden. Ook brengt dit mogelijk performance issues met zich mee. Voor elke update van een persoon moeten namelijk alle documenten in MongoDB nagelopen worden om daar de gegevens aan te passen.

Er is daarom besloten om te kijken of het hiërarchisch model wel de beste optie is en er niet beter gebruik kan worden gemaakt van het relationeel documentschema. (Figuur 6-2). De onderstaande deelvraag is opgesteld:

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

6.1 Aanpak

Om de 2 mogelijke varianten te kunnen onderzoeken worden er 2 Proof-of-Concepts ontwikkeld. Beide Proof-of-Concepts bestaan uit 2 collecties met data: Knowledge Items en People Items. De People Item collectie is bij beide Proof-of-Concepts hetzelfde. Deze bevat van elke persoon een unieke code, een naam, een ondertekst en een profielafbeelding.

```
{
    "_id" : ObjectId("55c60f9855534926dc700e49"),
    "name" : "Dai Potts",
    "profileImage" : "http://example.ie/image.png",
    "subText" : "Manager HR"
}
```

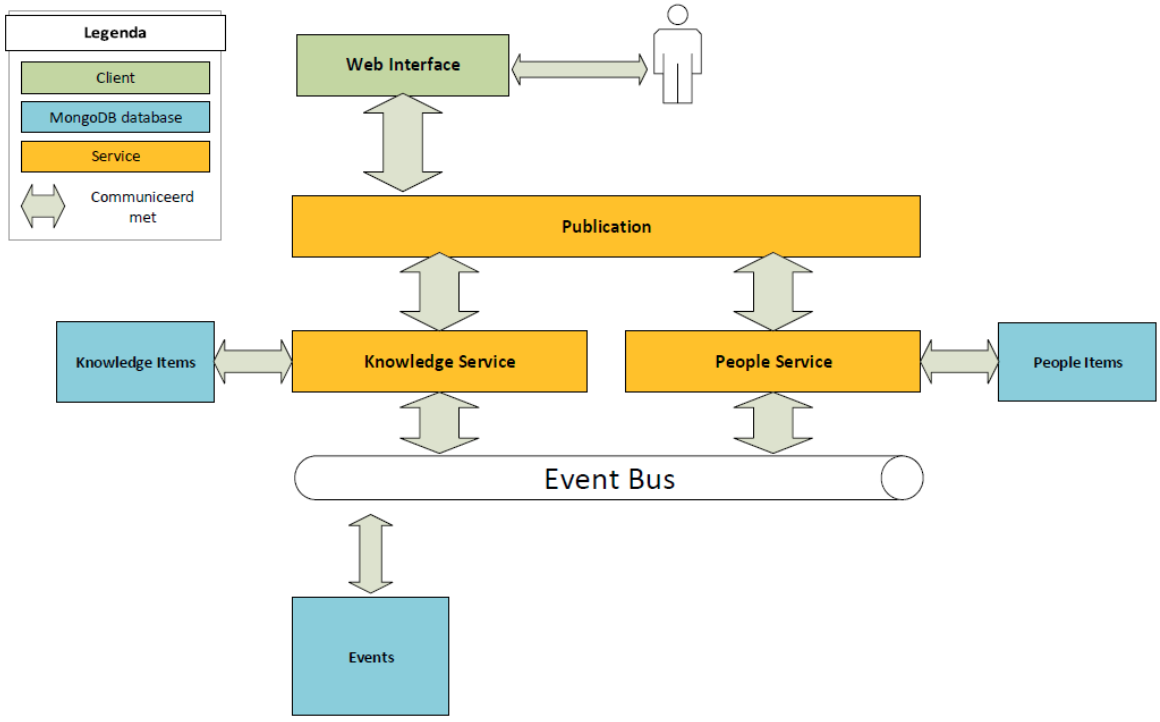
Figuur 6-3. De People Items collectie. Deze collectie is bij beide varianten hetzelfde

De Knowledge Item collectie is verschillend in de 2 varianten:

1. Het eerste Proof-of-Concept bevat voor de Knowledge Items het documentschema van Figuur 6-1, waarbij de relaties van de auteur, zijn opgenomen als subdocument (Hiërarchisch documentschema)
2. Het tweede Proof-of-Concept bevat voor de Knowledge Items het documentschema van Figuur 6-2. Hierbij wordt er alleen gerefereerd aan de hand van een vreemde sleutel. De applicatie dient hierbij zelf de bijhorende auteur van een Knowledge Item op te halen. (Relationeel documentschema)

De documentschema's (zowel de Knowledge en People Items) van beide varianten worden volledig getoond in het document Ontwerp documentschema (Bijlage E).

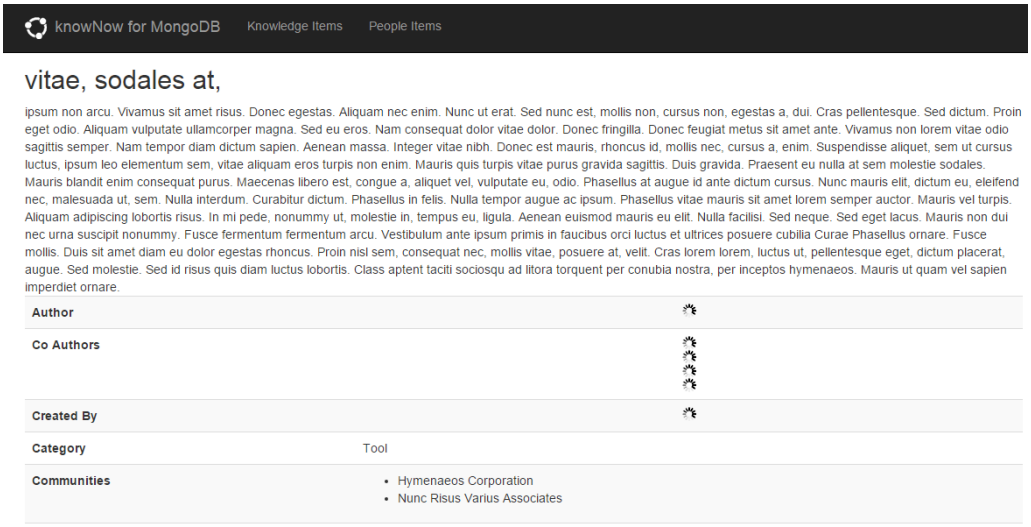
Beide varianten van het Proof-of-Concept zijn opgezet volgens een Microservice architectuur zoals besloten in paragraaf 3.2 van dit document. In het Technisch ontwerp wordt dieper ingegaan op de technische werking van het Proof-of-Concept.



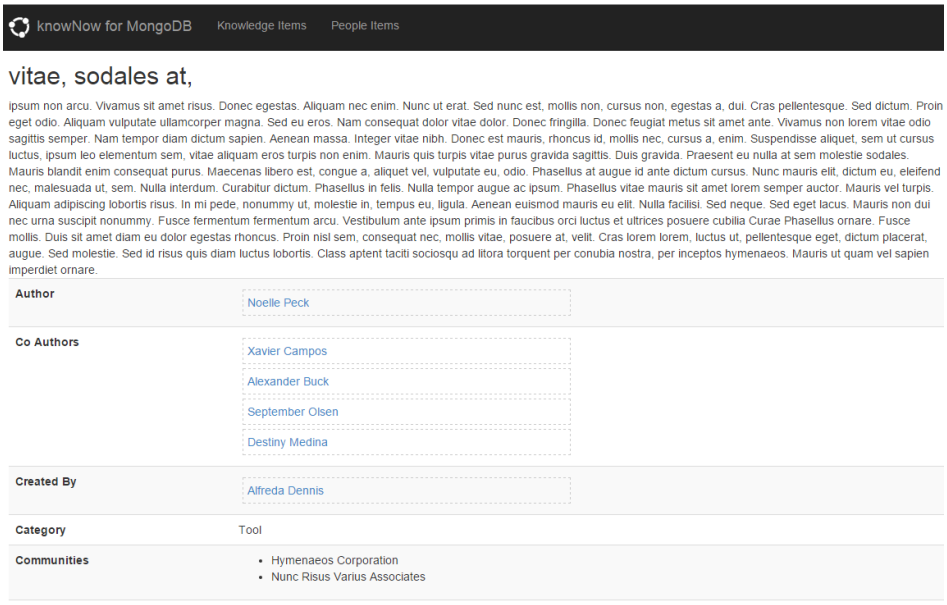
Figuur 6-4. De microservice architectuur van de 2 Proof-of-Concepts

Het ophalen van bijhorende People Items in het relationele documentschema dient door de client gedaan te worden aan de hand van de referentie in het Knowledge Item. Dit wordt asynchroon uitgevoerd. De gebruiker van de webinterface ziet eerst de gegevens die al in het Knowledge Item staan in zijn browser verschijnen. In de achtergrond worden de bijhorende People Items geladen. De People Items worden pas op de pagina getoond wanneer deze zijn geladen.

Het voordeel van het asynchroon laden van relaties is dat de gebruiker al kan beginnen met het lezen van de pagina terwijl additionele informatie nog geladen wordt. Bij het synchroon laden moet de gebruiker wachten totdat alle informatie is opgehaald.



Figuur 6-5. In het relationele documentschema worden de gegevens asynchroon geladen. De gebruiker kan alvast beginnen met lezen terwijl de additionele informatie nog wordt bijgeladen.



Figuur 6-6. In het hiërarchische documentschema wordt het volledige artikel in één keer geladen. De gebruiker moet wachten totdat alle informatie is gelezen.

De 2 verschillende documentschema's worden vergeleken op basis van de volgende opgestelde toetsingspunten die specifiek voor knowNow belangrijk zijn gevonden door de onderzoeker en de opdrachtgever:

- **Snelheid (performance)**
  De snelheid is van belang voor de gebruiker zodat deze niet te lang hoeft te wachten totdat de actie is uitgevoerd. 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 hogere schaalbaarheid is dan een met een lagere schaalbaarheid. Een oplossing met hogere schaalbaarheid kan beter meegroeien wanneer het aantal gebruikers van de applicatie toeneemt. knowNow moet om kunnen gaan met 200.000 gebruikers waardoor een hoge schaalbaarheid belangrijk is.
- **Consistentie**
  Het hebben van direct consistente data heeft de voorkeur tegenover uiteindelijke consistentie.
- **Onderhoudbaarheid/eenvoudigheid 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.

Deze punten worden op de volgende Use Cases, die voor knowNow belangrijk zijn, getoetst. Per Use Case is het verschil tussen de 2 verschillende documentschema's uitgelegd.

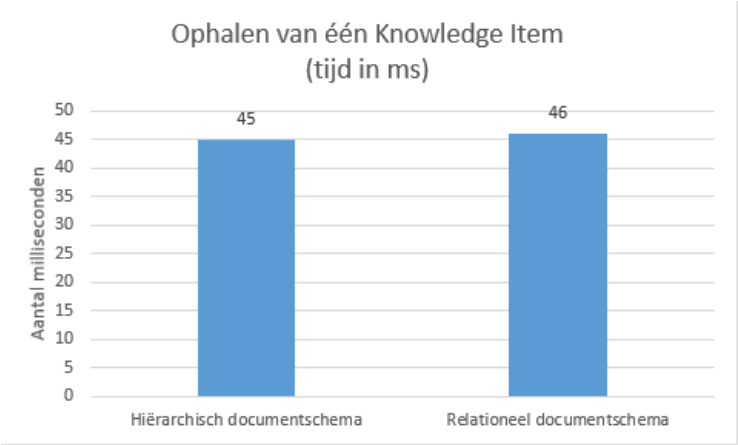
Use Case	Hiërarchisch documentschema	Relationeel documentschema
Ophalen Knowledge Item	Omdat alle gegevens in een document zitten kan het Knowledge Item in een keer volledig en synchroon worden geretourneerd aan de gebruiker.	Het document bevat referenties naar de People Items. Het Knowledge Item wordt eerst naar de gebruiker geretourneerd waarna asynchroon 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.

De andere Use Cases van het Proof-Of-Concept, beschreven in het Functioneel ontwerp, worden niet getoetst omdat deze in beide varianten technisch hetzelfde werken.

De performancetesten worden uitgevoerd op een desktop van de afstudeerder en niet op een (vergelijkbare) productieomgeving. Hierdoor zijn de testresultaten niet gelijk aan een productieomgeving. De performancetesten zijn bedoeld om beide documentschema's te vergelijken. Dezelfde 470 Knowledge Items zitten in beide varianten van het Proof-of-Concept.

6.2 Ophalen Knowledge Item

Snelheid
Met JMeter wordt de laadtijd in beide varianten voor het opvragen van 5 keer alle 470 Knowledge Items, die in het Proof-of-Concept zijn toegevoegd zijn gemeten. Voor deze aantallen zijn gekozen zodat enkele uitschieters geen grote invloed hebben op het gemiddelde. Uit de test kwam het volgende resultaat naar voren:



Figuur 6-7. Testresultaat ophalen Knowledge Item

Het ophalen van een Knowledge Item in het Hiërarchisch documentschema duurt gemiddeld 46 milliseconden. Het ophalen van een Knowledge Item in het Relationele documentschema duurt gemiddeld 45 milliseconden.

Het ophalen van Knowledge Items met het Relationele documentschema is dus 1 milliseconde sneller, echter, moeten hierna de People Items nog asynchroon opgehaald te worden. Dit is niet meegenomen in de test omdat de gebruiker nu kan beginnen met lezen terwijl additionele informatie wordt geladen.

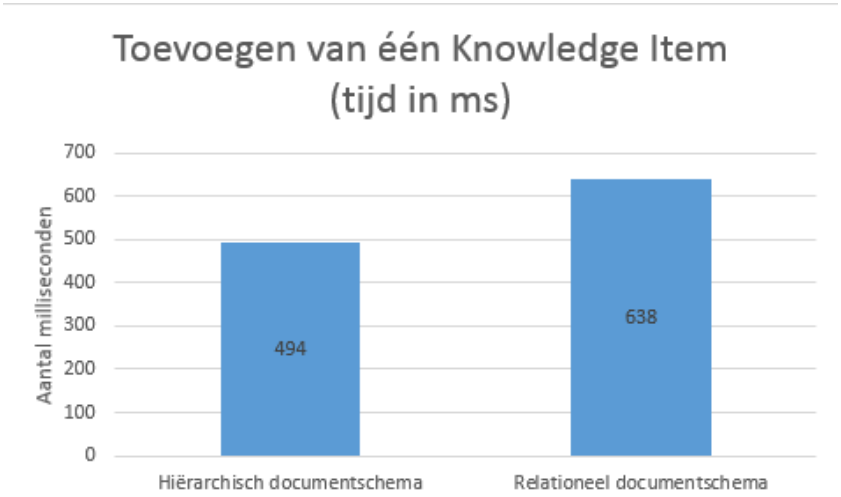
Door dit minimale verschil is te stellen dat het Relationele documentschema geen voordelen heeft tegenover het hiërarchische documentschema wat betreft de snelheid.

Toetsing

Toets punt	Toelichting	Score hiërarchisch documentschema	Score relationeel documentschema
Snelheid	Nauwelijks verschil	+/-	+/-
Schaalbaarheid	Bij het relationeel documentschema wordt er bij het ophalen van een Knowledge Item ook additionele informatie opgehaald via een extra http-requests. Hierdoor is het aantal http-requests bij het relationeel documentschema meer waardoor dit mogelijk in de toekomst eerder geschaald moet worden.	+/-	-
Consistentie	Bij het Hiërarchisch documentschema kan de data tijdelijk inconsistent zijn. Bij het relationeel documentschema is de data altijd consistent.	-	+
Onderhoudbaarheid	Geen verschil	+/-	+/-

6.3 Toevoegen Knowledge Item

Snelheid
In JMeter worden voor beide documentschema's 100 Knowledge Items toegevoegd. De Knowledge Items zijn voor beide Proof-of-Concepts hetzelfde. Voor het aantal 100 is gekozen zodat een enkele uitschieter niet direct consequenties heeft op het resultaat. Uit de test kwam het volgende resultaat naar voren:



Figuur 6-8. Testresultaat toevoegen Knowledge Item

Uit het testresultaat valt op dat het toevoegen van het Knowledge Item in het Proof-of-Concept met een relationeel documentschema 144ms sneller is dan het toevoegen van een Knowledge Item in het hiërarchisch documentschema. Dit is te verklaren doordat bij het hiërarchisch documentschema eerst alle bijhorende People Items opgehaald dienen te worden.

Hieruit kan worden geconcludeerd dat voor een snelle toevoeging van Knowledge Items het relationele documentschema beter is.

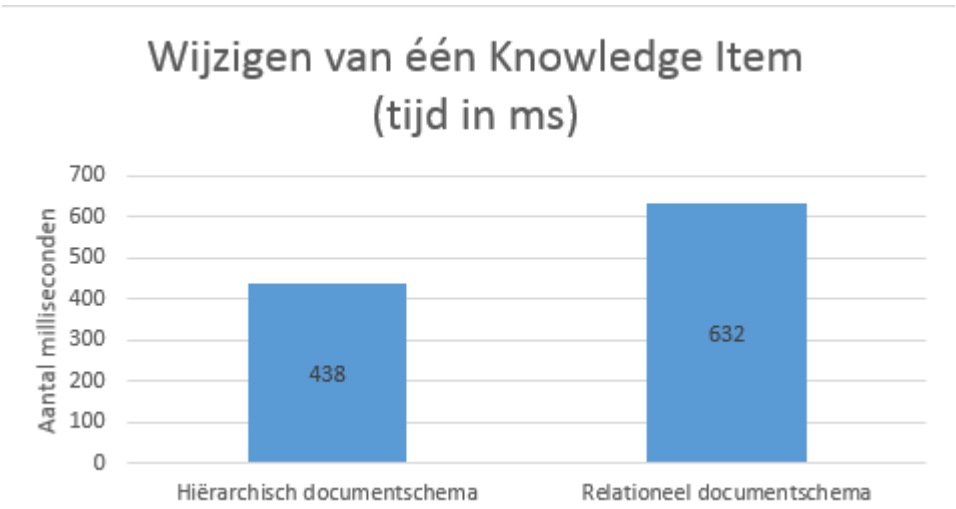
Toetsing

Toets punt	Toelichting	Score hiërarchisch documentschema	Score relationeel documentschema
Snelheid	Relationeel model is sneller	-	+
Schaalbaarheid	Geen invloed	+/-	+/-
Consistentie	Geen invloed	+/-	+/-
Onderhoudbaarheid	Bij het hiërarchische documentschema moet bij het toevoegen de bijhorende People Items worden opgehaald. In het relationeel model is geen extra functionaliteit nodig.	-	+

6.4 Wijzigen Knowledge Item

Snelheid
De test voor het wijzigen van een Knowledge Item is hetzelfde uitgevoerd als de test voor het toevoegen van een Knowledge Item. Het enige verschil is dat er geen 100 artikelen worden toegevoegd, maar 100 artikelen worden gewijzigd.

Uit de test kwam het volgende resultaat naar voren:



Figuur 6-9. Resultaat wijzigen Knowledge Item

Uit het testresultaat valt op dat het wijzigen van het Knowledge Item in het Proof-of-Concept met een relationeel documentschema 194ms sneller is dan het wijzigen van een Knowledge Item in het hiërarchisch documentschema. Dit is te verklaren doordat bij het hiërarchisch documentschema eerst alle bijhorende People Items opgehaald dienen te worden.

Hieruit kan worden geconcludeerd dat voor een snelle wijziging van Knowledge Items het relationele datamodel beter is.

Toetsing

Toets punt	Toelichting	Score hiërarchisch documentschema	Score relationeel documentschema
Snelheid	Relationeel model is sneller	-	+
Schaalbaarheid	Geen verschil	+/-	+/-
Consistentie	Geen verschil	+/-	+/-
Onderhoudbaarheid	Bij het hiërarchische documentschema moet bij het wijzigen de bijhorende People Items worden opgehaald. In het relationeel model is geen extra functionaliteit nodig.	-	+

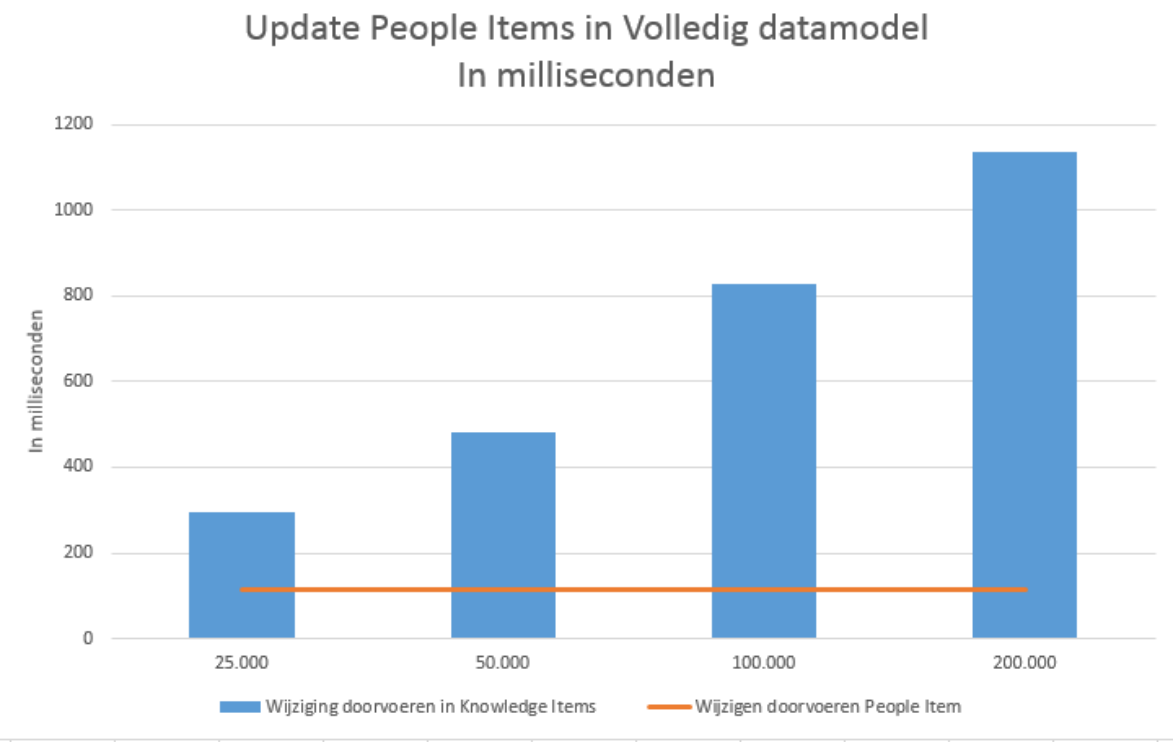
6.5 Wijzigen People Item

Snelheid
Het wijzigen van een People Item in de People Item collectie is voor de gebruiker voor beide varianten hetzelfde. Het wijzigen van één People Item duurt in beide varianten 116ms. De test is op eenzelfde manier uitgevoerd als het de test bij het Wijzigen van een Knowledge Model.

Schaalbaarheid
Bij het hiërarchisch documentschema moeten alle in de Knowledge Items als subdocument opgenomen People Items worden bijgewerkt na een wijziging. Dit wordt gedaan door middel van de Job Queue. Deze Job Queue draait op de achtergrond, en heeft dus geen invloed op de snelheid van de webinterface. Echter, mogelijk wel op de schaalbaarheid omdat dit wel rekenkracht van de applicatie in beslag zal nemen.

De Job Queue is getest door te kijken hoe lang het duurt om een wijziging van één People Item door te voeren in 25.000, 50.000, 100.000 en 200.000 Knowledge Items waarvan 200 People Items eerlijk zijn verdeeld als auteur over deze Knowledge Items. Op de auteur in de Knowledge Item is een index[9] gelegd zodat het vinden van Knowledge Items bij het te wijzigen People Item op het snelst gaat.

Doordat het aantal geteste Knowledge Items lineair toeneemt kan de invloed van de Job Queue op de schaalbaarheid worden getest. In de grafiek hieronder zijn de resultaten van deze test te vinden.



Figuur 6-10. Update People items in hiërarchisch documentschema met verschillend aantal documenten

[9] Een index is een methode om selecties op data te versnellen door het aantal benodigde vergelijkingen te verminderen[B4].

De oranje lijn toont het aantal milliseconden dat het duurt om de wijziging door te voeren in de People Items en dat de gebruiker van de applicatie moet wachten (snelheid). De blauwe balken tonen het aantal milliseconden dat het duurt om de update uit te voeren op alle documenten in de Knowledge Item collectie.

Het valt op dat de verwerkingstijd lineair toeneemt met het aantal documenten. Bij 25.000 documenten duurt het 296ms, bij 200.000 documenten is het al meer dan een seconde. Hierdoor is de Job Queue slecht schaalbaarheid. De uitvoertijd groeit naarmate het documenten groeien. Wanneer People Items veel wijzigen is het mogelijk te zwaar om bij elke wijziging alle documenten in de Knowledge Service te bewerken.

Toetsing

Toets punt	Toelichting	Score hiërarchisch documentschema	Score relationeel documentschema
Snelheid	Voor de gebruiker zijn beide schema's even snel.	+/-	+/-
Schaalbaarheid	Bij het hiërarchisch documentschema moet de Job Queue alle Knowledge Items consistent maken. De uitvoertijd hiervan groeit lineair met het aantal Knowledge Items. Bij het relationele documentschema's	-	+
Consistentie	Bij een hiërarchisch documentschema zij de Knowledge Items niet direct consistent met de People Items na een wijziging	-	+
Onderhoudbaarheid	Bij het hiërarchische documentschema moet voor het wijzigen de bijhorende People Items een Job Queue worden geïmplementeerd.	-	+

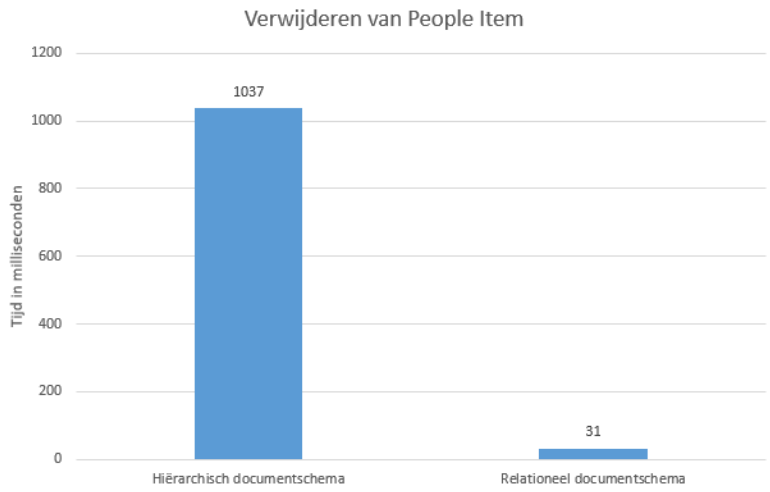
6.6 Verwijderen People Item

Snelheid
Het verwijderen van een People Item in de People Item collectie is voor beide Proof-of-Concepts hetzelfde voor de gebruiker.

Schaalbaarbeid
Het verwijderen van People Items uit de Knowledge Items gebeurt bij het hiërarchisch documentschema door middel van de Job Queue.

Ook de niet meer bestaande referenties naar People Items in het Proof-Of-Concept met relationele documentschema moeten verwijderd worden. Dit wordt pas gedaan op het moment dat er een Knowledge Item wordt opgevraagd en niet op het moment dat een People Item verwijderd wordt.

Er is voor beide implementaties gekeken hoe lang het gemiddeld duurt om één People Item te verwijderen uit een Knowledge Item. De test is uitgevoerd op een database met 100.000 Knowledge Items en 200 verschillende People Items. De People Items zijn eerlijk verdeeld over de Knowledge Items.



Figuur 6-11. Verwijderen van People Items uit Knowledge Items

Zichtbaar is dat het verwijderen van People Items in het relationele documentschema vele male sneller is dan verwijderen van een People Item in het hiërarchisch documentschema. Deze resultaten zijn echter niet met elkaar te vergelijken. Bij het hiërarchisch documentschema wordt een verwijderd People Item direct uit alle Knowledge Items gehaald. Bij de implementatie van het verwijderscenario bij het relationele documentschema wordt alleen de referentie verwijderd wanneer bij het ophalen van een Knowledge Item blijkt dat een People Item niet meer bestaat.

De referentie wordt dan ook alleen bij dat item verwijderd. Door deze oplossing zal een taak veel korter duren, er wordt maar één referentie verwijderd voor één Knowledge Items in plaats van alle Knowledge Items met die referentie. Hierdoor is de oplossing veel beter schaalbaar. Het wordt pas bijgewerkt op het moment dat het nodig is en wordt zo dus beter verdeeld in plaats dat alle People Items in een verwijderd moeten worden. Echter betekend het wel dat de Knowledge Items pas consistent worden gemaakt nadat deze wordt opgevraagd.

Toetsing

Toets punt	Toelichting	Score hiërarchisch documentschema	Score relationeel documentschema
Snelheid	Voor de gebruiker zijn beide schema's even snel.	+/-	+/-
Schaalbaarheid	Bij het hiërarchisch documentschema moet de Job Queue alle Knowledge Items in een keer consistent maken. Bij het relationeel documentschema wordt dit beter verdeeld doordat dit pas gedaan wordt op het moment dat het echt nodig is.	-	+
Consistentie	In beide gevallen is de data niet direct consistent. Bij het hiërarchisch documentschema wordt het echter wel in één keer consistent gemaakt terwijl bij het relationeel document het pas consistent wordt gemaakt op het moment dat een Knowledge Item wordt opgevraagd.	+/-	-
Onderhoudbaarheid	In beide gevallen moet er extra functionaliteit worden geïmplementeerd.	-	-

6.7 Conclusie

In de tabel hieronder zijn alle scores van de diverse Use Cases samengevoegd en is een totaalscore berekend.

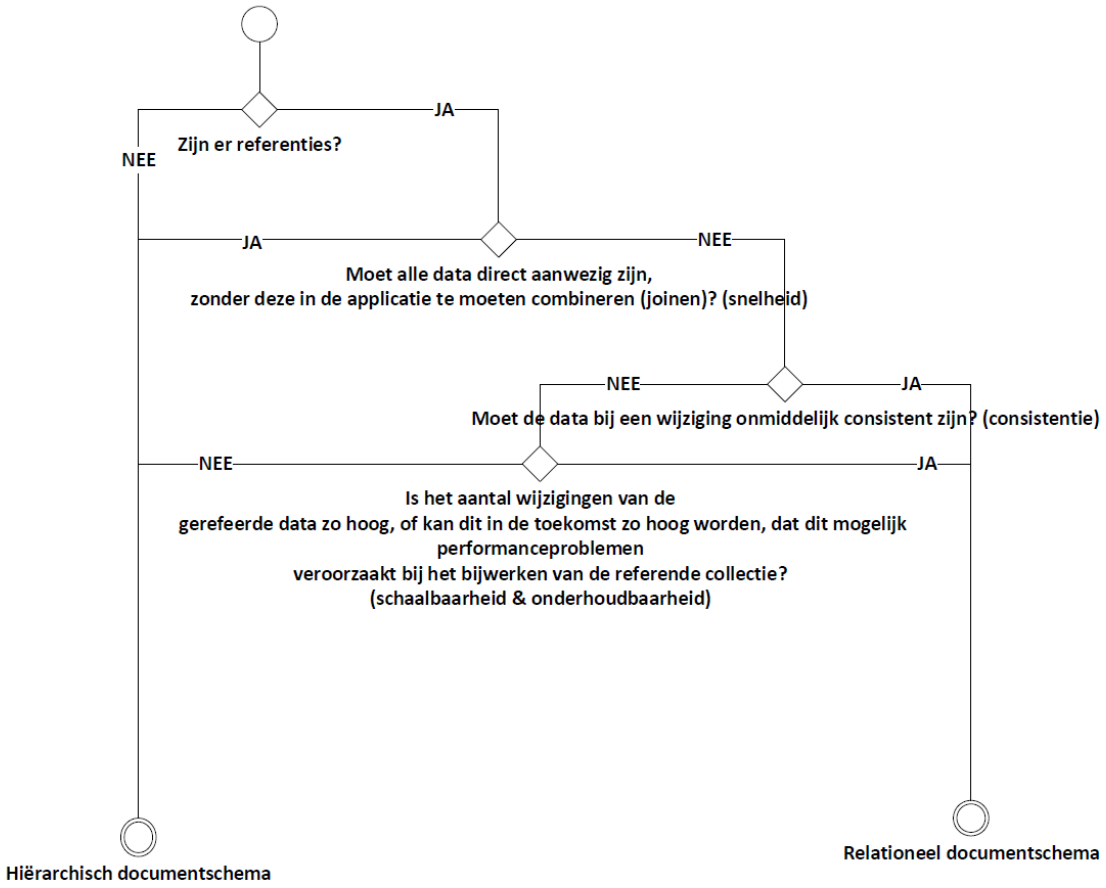
Use Case/Toetsingspunt	Score hiërarchisch documentschema	Score relationeel documentschema
Ophalen Knowledge Item		
Snelheid	+/-	+/-
Schaalbaarheid	+/-	-
Consistentie	-	+
Onderhoudbaarheid	+/-	+/-
Toevoegen Knowledge Item		
Snelheid	-	+
Schaalbaarheid	+/-	+/-
Consistentie	+/-	+/-
Onderhoudbaarheid	-	+
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

Er kan 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 slecht schaalt, de uitvoertijd neemt lineair toe naarmate het aantal Knowledge Items toeneemt.

6.8 Generalisatie

Voor de Use Cases van knowNow is een hoge schaalbaarheid en hoge onderhoudbaarheid van belang. Ook is het voor knowNow geen probleem dat additionele informatie zoals de People Items asynchroon geladen moeten worden. Echter gelden deze punten mogelijk niet voor andere systemen. Daarom is er aan de hand van het hierboven uitgevoerde onderzoek een beslissingsboom gemaakt waarmee ook voor andere systemen de keuze tussen een hiërarchisch documentschema en een relationeel documentschema gemaakt kan worden.



Figuur 6-12. Beslissingsboom voor keuze tussen Hiërarchisch- en relationeel documentschema

7. Conclusie

Aan de hand van de hierboven onderzochte deelvragen kan een conclusie worden getrokken. Deze conclusie is gemaakt vóór de afronding van het Proof-of-Concept.

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.

Doordat MongoDB zelf goed te beveiligen is, heeft MongoDB geen invloed op de security van knowNow. De knowNow applicatie authentiseert en autoriseert de gebruiker en bepaald dus wie welke data mag inzien. De applicatie dient zich alleen te authentiseren bij de MongoDB server, autoriseren is niet nodig. Dat is nu ook niet het geval bij de SQL Server installatie die door knowNow wordt gebruikt. MongoDB zal dus niet bepalen wie bij welke data mag. Dit dient te worden gerealiseerd in de applicatie.

In Deelvraag 2 is geconstateerd dat MongoDB een grote uitbreidbaarheid heeft. Aanpassingen aan het documentschema hebben weinig gevolgen, en zijn eenvoudig door te voeren. Het updaten van het MongoDB documentschema is mogelijk doormiddel 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. Het is wel belangrijk om wel voor elke aanpassing altijd te kijken of er onverwachte consequenties zijn op het gebied van performance en schaalbaarheid. En of de uitbreiding het geringe, maar wel aanwezige, performance verlies waard is.

In Deelvraag 3 is gekeken hoe de Knowledge Service te allen tijde uiteindelijk consistent worden gehouden met andere services. Er is hier een mogelijke oplossing bedacht, de Job Queue. Het is nu nog niet mogelijk te stellen of de gevonden oplossing ook in praktijk werkt. Dit wordt getest door middel van de implementatie in het Proof-of-Concept.

Bij Deelvraag 4 is geconstateerd dat het relationele documentschema het best voor de Use Cases van de Knowledge Item gebruikt kan worden. Het relationele documentschema is beter onderhoudbaar en heeft geen intensief achtergrondproces doordat de Job Queue niet gebruikt hoeft te worden. Daarnaast is het ophalen van Knowledge Items iets sneller, echter moet daarna nog wel additionele informatie, zoals de People Items, worden opgehaald.

Er kan voorzichtig worden gesteld dat MongoDB een *werkbaar* alternatief is voor SQL Server. De uitbreidbaarheid is hoog, zowel migraties van data en uitbreidingen aan het documentschema zijn mogelijk. MongoDB heeft weliswaar een grote impact op het Technisch Ontwerp, maar wanneer het knowNow team kiest om een Microservice architectuur te implementeren is het goed mogelijk bij de ontwikkeling hiervan ook MongoDB te gaan gebruiken. Wel moeten er extra oplossingen ontwikkeld worden, zoals de Job Queue, om van MongoDB een *werkbaar* alternatief te maken. Omdat deze niet nodig zijn in een RDBMS zoals SQL server ontstaat de vraag of MongoDB een *beter* alternatief is dan SQL Server. Dat is onder andere pas vast te stellen door een performance onderzoek uit te voeren zodat kan worden bekeken of MongoDB veel sneller is en beter schaalt dan SQL Server. Daarna kan worden besloten of de te ontwikkelen oplossingen het waard zijn.

Aan de hand van het te realiseren Proof-of-Concept zal worden geëvalueerd of de in dit onderzoek gedane bevindingen ook in praktijk blijken te werken. In het Adviesrapport wordt het advies gegeven of MongoDB in te zetten is voor knowNow en of ik dit zelf aanraad.

Referenties

[A1]	MongoDB, “Authentication” http://docs.mongodb.org/manual/core/authentication/
[A2]	MongoDB, “Security Network” http://docs.mongodb.org/manual/core/security-network/
[A3]	MongoDB, “Collection Level Access Control” http://docs.mongodb.org/manual/core/collection-level-access-control/
[A4]	MongoDB, “Auditing” http://docs.mongodb.org/manual/core/auditing/
[A5]	MongoDB, “MongoDB Limits and Thresholds” http://docs.mongodb.org/manual/reference/limits/
[A6]	MongoDB, “FAQ: MongoDB Fundamentals” http://docs.mongodb.org/manual/faq/fundamentals/
[A7]	C2, Monolithic Design http://c2.com/cgi/wiki?MonolithicDesign
[A8]	Microsoft, “Kerberos” https://msdn.microsoft.com/en-us/library/windows/desktop/aa378747%28v=vs.85%29.aspx
[A9]	MongoDB, “Authenticate Using SASL and LDAP with OpenLDAP” http://docs.mongodb.org/manual/tutorial/configure-ldap-sasl-openldap/
[A10]	MongoDB, “Sharding Introduction” http://docs.mongodb.org/manual/core/sharding-introduction/
[A11]	Vtagion, “Scalability: Scale-up or Scale-out, What it is and Why You Should Care” http://www.vtagion.com/scalability-scale-up-scale-out-care/
[A12]	MongoDB Enterprise Advanced Datasheet http://info-mongodb-com.s3.amazonaws.com/MongoDB_Enterprise_Advanced_Datasheet.pdf
[A13]	MongoDB, “Shard Keys” http://docs.mongodb.org/manual/core/sharding-shard-key/#shard-keys
[A14]	MongoDB, “Replica Set Elections” http://docs.mongodb.org/manual/core/replica-set-elections/
[A15]	MMS: The Easiest Way to Run MongoDB https://mms.mongodb.com/
[A16]	MMS FAQ, “Is My Data Safe?” https://docs.mms.mongodb.com/faq/
[A17]	MongoDB MMS, “Terms of Service” https://mms.mongodb.com/links/terms-of-service
[A18]	MongoDB, “Access Control” http://docs.mongodb.org/v2.4/core/access-control/
[A19]	Martin Fowler, “Microservices” http://martinfowler.com/articles/Microservices.html
[B1]	MongoDB, “MongoDB Limits and Thresholds” http://docs.mongodb.org/manual/reference/limits/

[B2]	MongoDB, “BSON Types” http://docs.mongodb.org/manual/reference/bson-types/
[B3]	MongoDB, “SQL to MongoDB Mapping Chart” http://docs.mongodb.org/manual/reference/sql-comparison/
[B4]	MongoDB, “Index Types” http://docs.mongodb.org/manual/core/index-types/
[B5]	MongoDB, “Server-Side JavaScript” http://docs.mongodb.org/manual/core/server-side-javascript/
[B6]	MongoDB, “Map Reduce” http://docs.mongodb.org/manual/reference/glossary/#term-map-reduce
[B7]	MongoDB, “Map Reduce out Option” http://docs.mongodb.org/manual/reference/method/db.collection.mapReduce/#mapreduce-out-mtd
[B8]	MongoDB, “$where” http://docs.mongodb.org/manual/reference/operator/query/where/#op._S_where
[B9]	MongoDB, “Update Operators” http://docs.mongodb.org/manual/reference/operator/update/
[B10]	Stackoverflow, “Update MongoDB field using another field” http://stackoverflow.com/questions/3974985/update-mongodb-field-using-value-of-another-field
[B11]	Wikipedia, “Idempotentie” https://nl.wikipedia.org/wiki/Idempotentie
[B12]	MongoDB, “Perform Two Phase Commit” http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/
[B13]	Webopedia, “Two Phase Commit” http://www.webopedia.com/TERM/T/two_phase_commit.html
[B14]	MongoDB, “Atomicity and Transactions” http://docs.mongodb.org/manual/core/write-operations-atomicity/
[B15]	MSDN, “Transactional Replication” https://msdn.microsoft.com/en-us/library/ms151176.aspx
[B16]	C2, “Two Phase Commit” http://c2.com/cgi/wiki?TwoPhaseCommit
[B17]	W3Schools, “AJAX” http://www.w3schools.com/Ajax/ajax_intro.asp

DE HAAGSE HOGESCHOOL

InfoSupport Solid Innovator

Bijlage C – Technisch Ontwerp oorspronkelijke situatie knowNow

MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?

Technisch Ontwerp oorspronkelijke situatie knowNow

Titel	Technisch Ontwerp oorspronkelijke situatie knowNow
Project/Onderwerp	MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?
Versie	1.0
Status	Definitief
Datum	24-08-2015
Bestand	Technisch Ontwerp oorspronkelijke situatie knowNow v1.0
Bedrijf	Info Support
Door	Tom Keim

Info Support B.V. Kruisboog 42, 3905 TG, Veenendaal, Tel. +31 (0) 318 55 20 20 - Fax +31 (0) 318 55 23 55
Kenniscentrum De Smalle Zijde 39, 3903 LM Veenendaal, Tel. +31 (0) 318 50 11 19 - Fax +31 (0) 318 51 83 59

Inhoudsopgave

1. Inleiding

In dit document wordt het technisch ontwerp van de knowNow applicatie besproken hoe deze er voor de uit te voeren afstudeeropdracht uit zag. In dit technisch ontwerp worden de volgende punten besproken:

- De structuur van de huidige database;
- De technische werking van knowNow (architectuur, performance, security en data access);
- De third-party applicaties waar knowNow gebruik van maakt.

2. Scope van Technisch Ontwerp

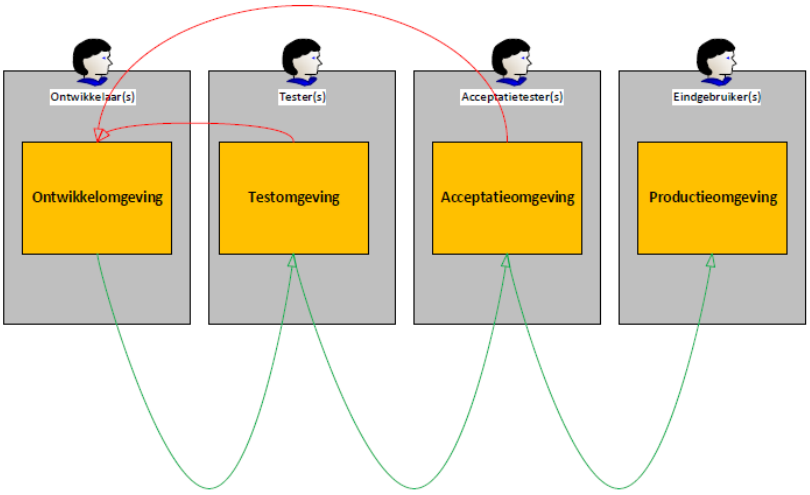
knowNow is een erg grote applicatie waarvan het beschrijven van de gehele architectuur leidt tot een erg groot document. Omdat dit technisch ontwerp alleen nodig is voor het onderzoek naar de invloed van MongoDB op dit Technisch Ontwerp zullen alleen de delen van het Technisch Ontwerp waar MongoDB mogelijk invloed op kan hebben worden beschreven. Onderdelen waarvan duidelijk is dat MongoDB hier geen raakvlakken mee kan hebben zullen maar deels worden toegelicht.

3. Ontwikkelomgeving

knowNow wordt ontwikkeld in C# in Visual Studio 2013. Voor versiebeheer wordt er gebruik gemaakt van Microsoft Team Foundation Server in combinatie met Git.

Het knowNow team maakt gebruik van een OTAP-straat[9]. OTAP staat voor Ontwikkel, Test, Acceptatie & Productieomgeving. In de tabel hieronder zijn de diverse doelen van de omgevingen uitgelegd.

Omgeving	Doel	Gebruiker
Ontwikkelomgeving	In deze omgeving wordt de applicatie ontwikkeld. Daarna wordt de applicatie doorgezet naar de testomgeving.	Ontwikkelaar(s)
Testomgeving	In deze omgeving wordt de applicatie getest. Als niet alles naar behoren functioneert, gaat het terug naar de ontwikkelomgeving. Als alles door de testen komt wordt het doorgezet naar de acceptatieomgeving.	Tester(s)
Acceptatieomgeving	De acceptatieomgeving is vaak identiek aan de productieomgeving. In de acceptatieomgeving wordt de applicatie door de acceptanten(o.a. de klanten) goedgekeurd. Als de applicatie is goedgekeurd gaat de applicatie door naar de productieomgeving. Als de applicatie wordt afgekeurd gaat deze weer terug naar de ontwikkelomgeving en wordt het proces opnieuw doorlopen.	Acceptatietester(s)
Productieomgeving	De productieomgeving is de omgeving waar de applicatie door de eindgebruikers gebruikt kan worden.	Eindgebruiker(s)



Figuur 1. Het OTAP-straat proces

Het knowNow team kan met een druk op de knop de applicatie doorzetten naar de volgende omgeving. Alle geautomatiseerde testen (zie 4.9 Testen) worden dan automatisch uitgevoerd. Indien de testen niet slagen wordt het doorzetten van de applicatie stopgezet.

4. Softwarearchitectuur

4.1 Inleiding

knowNow is een op Orchard ontwikkelde applicatie. Orchard is een gratis, open-source Content Management Systeem (CMS).

Orchard is in C# .NET ontwikkeld. En biedt ondersteuning voor een aantal Relationele Database Management Systemen (RDBMS) zoals Microsoft SQL Server en MySQL.

De Orchard installatie maakt gebruik van Microsoft SQL Server als RDBMS. Voor knowNow zijn specifieke uitbreidingen op Orchard gemaakt. Dit zijn uitbreidingen gemaakt om extra functionaliteiten aan Orchard toe te voegen, functionaliteiten die er standaard niet in zaten. Uitbreidingen worden niet in de core van Orchard toegevoegd maar hiervoor worden zogeheten modules aangemaakt. Er zijn modules voor:

- API ondersteuning voor knowNow (zie 4.5. knowNow als platform)
- Ondersteuning voor de Event Bus (zie 4.8. Event Bus)
- MailChimp (een service waarmee mailings te maken en te versturen zijn)[1]
- Neo4j (Recommendation service op basis van Graph Database)[2]

In dit document wordt alleen de basis van Orchard besproken. Voor verdere informatie van Orchard kunt u de documentatie op de website van Orchard raadplegen.

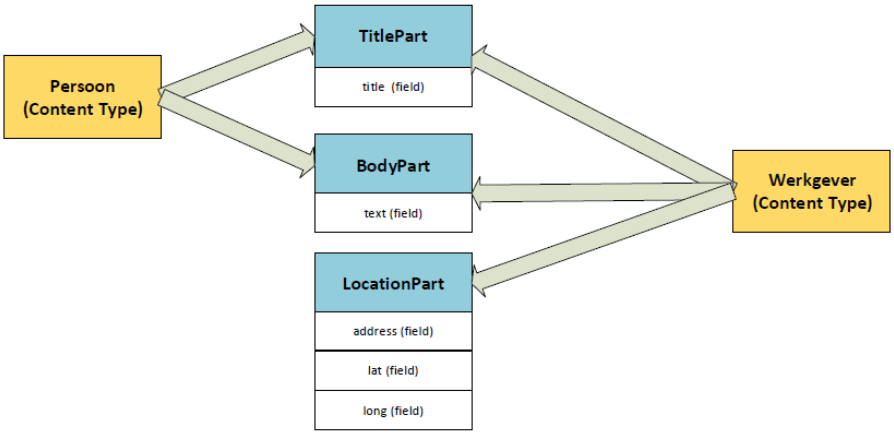
4.2 Content in Orchard

In veel applicaties is het gebruikelijk om voor elke entiteit een eigen tabel aan te maken. Zo worden bijvoorbeeld personen in de tabel personen opgeslagen, groepen in de tabel groepen en werkgevers in de tabel werkgevers.

Orchard pakt dit anders aan. Bij Orchard worden entiteiten in de code gedefinieerd en krijgt een entiteit niet een eigen tabel. Orchard kent Content Items, Content Type's, Content Parts en Fields.

Type	Omschrijving
Content Item	Een Content Item is een stuk content wat bewerkt kan worden.
Content Type	Content Type's is het type van een Content Item, bijvoorbeeld een persoon, een groep of een werkgever.
Content Parts	Content Parts zijn onderdelen die wat zeggen over het type. Zoals een TitlePart (titels) bij groepen of een AgePart (leeftijd) bij personen. Content Parts kunnen hergebruikt worden voor elk type. Zo kunnen bijvoorbeeld zowel de groepen als de werkgevers gebruik maken van een TitlePart. Hierdoor hoeft een part maar een keer gedefinieerd te worden en kan dit worden gedeeld over verschillende typen.
Fields	Fields zijn velden in een part. Voor een TitlePart zou er bijvoorbeeld een title veld kunnen zijn. Meerdere velden per part zijn mogelijk.

In het onderstaande figuur is geprobeerd dit principe aan de hand van een voorbeeld te verduidelijken. De Persoon & Werkgever zijn hier Content Types. Beiden hebben een title- en bodypart. Daarnaast heeft de werkgever ook een location part waar het adres en de coördinaten in opgeslagen zijn.



Het voordeel van dit systeem is de uitbreidbaarheid. Wanneer er een nieuwe entiteit gemaakt moet worden, hoeven hier alleen de parts aan toegevoegd worden. De frontend (website voor de gebruikers) en de backend (administratie) worden door Orchard afgehandeld.

Wanneer er een nieuwe functionaliteit aan een entiteit toegevoegd moet worden, bijvoorbeeld een lijst met familieleden, kan er een nieuwe part ontwikkeld worden. De nieuwe part kan dan natuurlijk weer (als het nodig is) hergebruikt worden in andere content typen.

4.3 NHibernate

Orchard maakt gebruik van NHibernate. NHibernate is een Object Relational Mapper (ORM). NHibernate maakt het makkelijk om Objecten in C# in te lezen en op te halen uit een relationele database.

4.4 Modules voor Orchard

Modules zijn uitbreidingen op Orchard. Op de Orchard Gallery[4] kan men vele modules vinden. Deze zijn dan te installeren in Orchard.

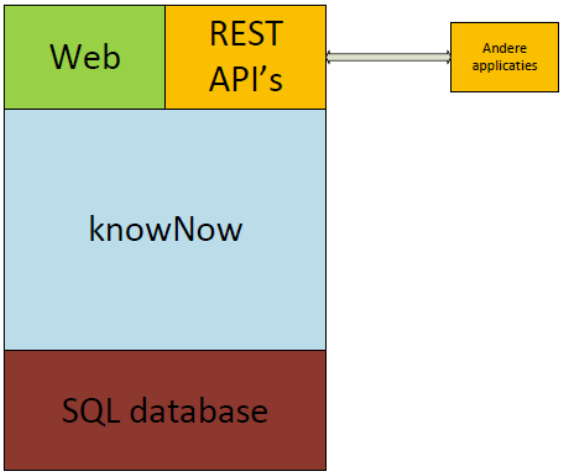
Naast het downloaden van Modules is het ook mogelijk zelf modules te schrijven. Info Support heeft meerdere modules ontwikkeld op Orchard:

Naam Module	Omschrijving
InfoSupport.knowNow	Dit is de algemene, grootste, maatwerk module voor knowNow. Deze bevatten onder andere de announcement (aankondigingen) en groups (communities).
InfoSupport.knowNow.Api.*	Er zijn door Info Support diverse API modules gemaakt. Hiermee is het mogelijk om diverse functionaliteiten van knowNow aan te spreken via een Application Programming Interface en niet meer alleen via de webinterface. (Zie ook 4.5. knowNow als een platform)
InfoSupport.knowNow.Events	Deze module zet events door aan de Event Bus (zie 4.8. Event Bus)
InfoSupport.knowNow.Security	Deze module bevat diverse uitbreidingen op het gebied van security. Zoals Windows authentication en OAuth authentication
InfoSupport.knowNow.Neo4j	Neo4j[2] is een Graph database. Deze module maakt de Recommendation service van knowNow mogelijk. Met de knowledge service worden gebruikers van knowNow aanbevelingen gedaan van mogelijk interessante artikelen.

4.5 knowNow als platform

Een jaar geleden is door het DevOps ontwikkelteam van knowNow het idee ontstaan om van de applicatie een platform te maken. Dit houdt in dat de er naast de webinterface, wat nog de enige manier was om het systeem te gebruiken, er REST API's geïntroduceerd worden waar andere applicaties op in kunnen haken. Hierdoor wordt knowNow een platform[10].

De REST API is nu grotendeels compleet en de meeste functionaliteiten zijn ook via deze API's uit te voeren. De API's zijn op dit moment naast de webinterface ontwikkeld.



Figuur 2. De REST API's zijn naast de webinterface ontwikkeld. De web applicatie en REST API's communiceren met dezelfde SQL database.

Een REST API (Representational State Transfer Application Programming Interface) is een manier om te communiceren met een applicatie[11].

knowNow maakt voor het uitwisselen van data binnen de REST API gebruik van JSON (JavaScript Object Notation)[12].

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

Figuur 3. Voorbeeld van een JSON object.

De volgende functionaliteiten worden via de REST API aangeboden:

- Het opvragen, aanpassen en toevoegen van knowledge items;
- Het opvragen, aanpassen en toevoegen van gebruikersprofielen;
- Het opvragen, aanpassen en toevoegen van projecten;
- Zoeken in o.a. projecten, knowledge items en gebruikersprofielen;
- Het opvragen, aanpassen en toevoegen van taxonomy.

4.6 Scheiding van services

Om de verwachte groei te ondersteunen zijn diverse services binnen knowNow gescheiden. knowNow maakt onder andere gebruik van de onderstaande services:

- Lucene.NET voor de searchengine
- een Recommendation service (op basis van de GraphDB Neo4J).

De services zijn van elkaar gescheiden en kunnen daarom ook apart worden gedupliceerd. Zo kan er wanneer de searchengine meer rekenkracht nodig heeft voor worden gekozen om alleen die service te dupliceren. Zo hoeft niet de gehele applicatie gedupliceerd te worden maar alleen de services die meer rekenkracht nodig hebben.

Een bijkomend voordeel van gescheiden services is dat voor elke service specifieke keuzes gemaakt kunnen worden. Zo kan bijvoorbeeld, als dat sneller blijkt, gekozen worden om voor de recommendation service Python te gebruiken en voor de andere componenten andere programmeertalen.

4.7 Performance

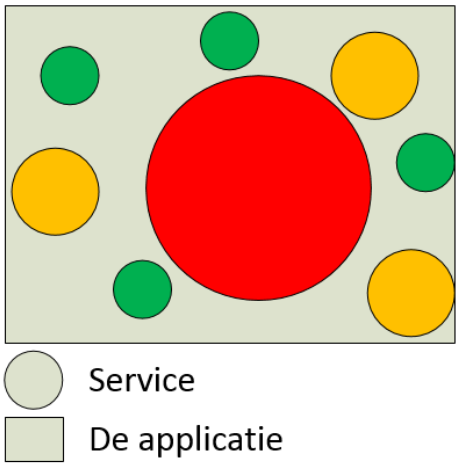
Op het moment van schrijven zijn er 7000 gebruikers van knowNow waarvan een aantal concurrent (gelijktijdig) zullen werken. Deze zijn verdeeld over 2 frontends. Frontends zijn aparte installaties van knowNow die wel dezelfde databases delen. De load wordt door middel van een load balancer verdeeld over de 2 frontends.

Een load balancer[8] is een systeem waarbij het werk verdeeld kan worden over verschillende systemen (de frontends). Het bijkomend voordeel is ook dat wanneer een frontend uitvalt, de load balancer zorgt ervoor dat het verkeer wordt doorgestuurd naar de nog wel werkende frontend waardoor de applicatie blijft werken.

knowNow is op dit moment bezig met een klant met potentieel 200.000 gebruikers, waar een groot deel concurrent van werkt. Als er voor 7000 gebruikers 2 frontends nodig zijn zullen er voor 200.000 gebruikers ongeveer 57 frontends (2/7000*200.000) nodig zijn. Dit wordt naast dat het waarschijnlijk lastig wordt om dit te beheren met een klein DevOps team waarschijnlijk ook nog zwaar voor de gedeelde database.

Binnen 5 jaar wil knowNow doorgroeien naar 3 miljoen gebruikers. Waardoor het beheren van deze frontends waarschijnlijk niet meer vol te houden is.

Daarnaast is het zonde van de rekenkracht om de gehele applicatie volledig te dupliceren. Niet alle onderdelen van de applicatie zullen namelijk even hard groeien in het gebruik van rekenkracht als het aantal gebruikers toeneemt.



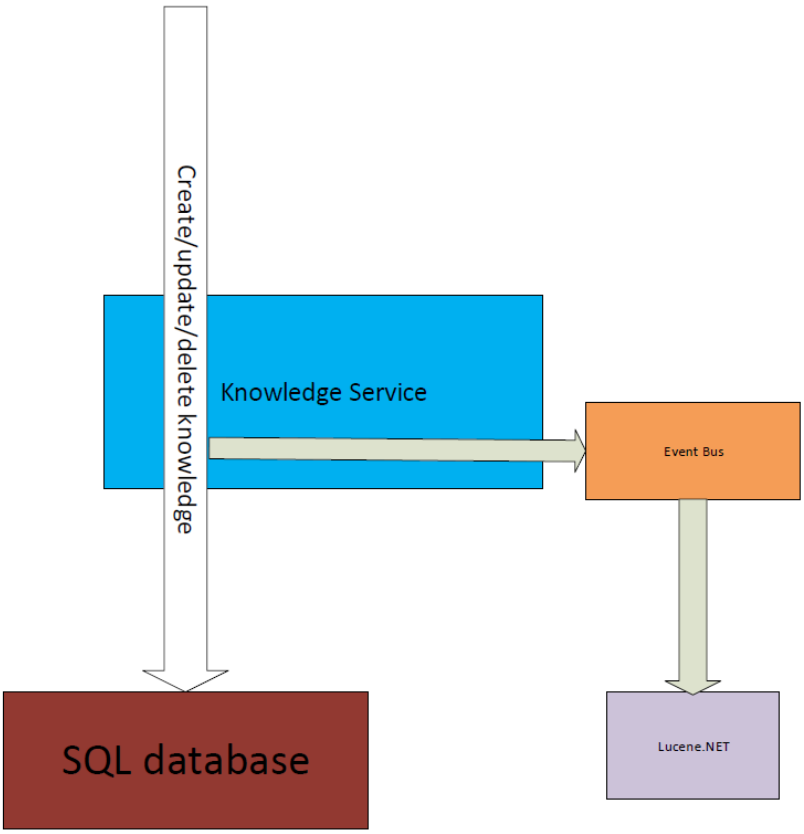
Figuur 4. De grootte van de cirkels en de kleur tonen de groei van benodigde rekenkracht bij toename concurrent gebruikers. Groen voor weinig groei, rood voor veel groei.

In het figuur hierboven staat een versimpelde weergave van de verschillende services van knowNow. De cirkels zijn de services, de grootte van de cirkels en de kleur(rood voor veel groei, groen voor weinig groei) is de groei van de rekenkracht die ze nodig hebben wanneer het aantal concurrent gebruikers toeneemt. Dit laat zien dat sommige services nauwelijks groeien onder druk en dus eigenlijk niet gedupliceerd hoeven te worden omdat ze nog niet hun volledige beschikbare rekenkracht benutten. Het is dan zonde om ze meer rekenkracht toe te kennen als dat helemaal niet nodig is.

4.8 Event Bus

4.8.1 Inleiding

De knowNow applicatie maakt gebruikt van een Event Bus. Een Event Bus is een stuk programmatuur waar na een request waar een Insert, Update of Delete transactie wordt gedaan een event in wordt geregistreerd. Hierin staat onder andere wat er veranderd is, wanneer het veranderd is en door wie de transactie is gemaakt. Hierdoor kan buiten de request (asynchroon) die data worden verwerkt waardoor de gebruiker minder lang hoeft te wachten op een response. Zo wordt bijvoorbeeld via de Event Bus de Lucene.NET search engine bijgewerkt na een aanpassing in de data.



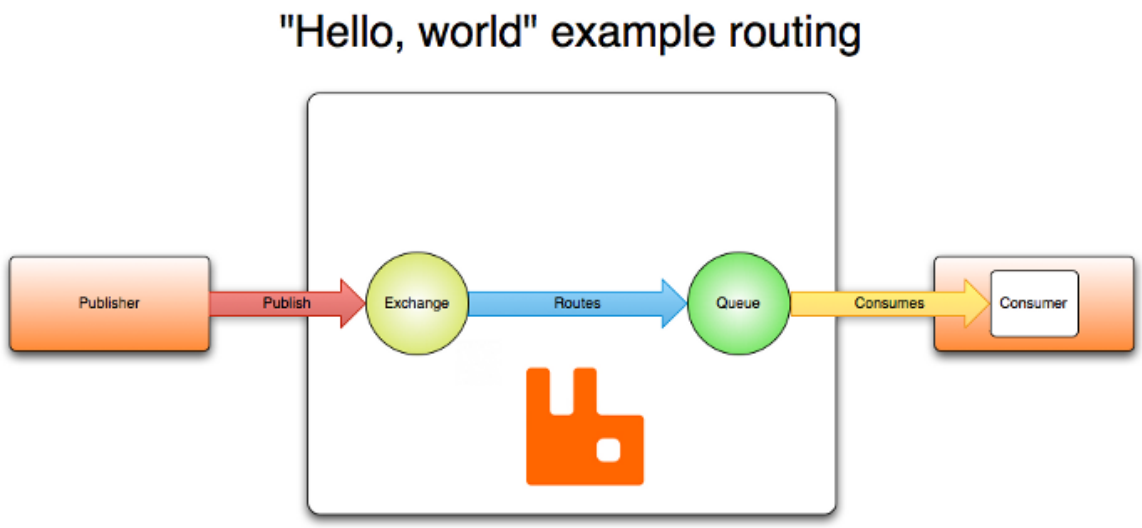
Figuur 5. Een schets van de werking van de event bus. Ten tijde van een request wordt de transactie opgeslagen in de Event Bus & de SQL database.

Lucene.NET is hierdoor niet meteen consistent met de SQL database maar wordt dat pas wanneer een proces de transacties die opgeslagen staan in de Event Bus heeft verwerkt. Dit heet Eventual consistency. Eventual consistency betekend dat in dit geval de search index van Lucene.NET niet direct gelijk is met de SQL database maar dat dit pas gebeurt wanneer de transacties in de Event Bus worden verwerkt.

4.8.2 RabbitMQ

De Event Bus maakt gebruik van RabbitMQ. RabbitMQ is een applicatie wat 'robuuste communicatie voor applicaties' aanbiedt[6]. Applicaties kunnen berichten aanbieden aan RabbitMQ. RabbitMQ zorgt er daarna voor dat deze berichten bij andere applicaties terechtkomen. Deze berichten kunnen bijvoorbeeld de events zijn van een applicatie. Zoals de aanpassing van een artikel. De consumers (de ontvangende applicaties) krijgen dan te horen wat er is aangepast zodat ze dit kunnen verwerken.

Wanneer er een gebeurtenis (event) in de applicatie plaatsvindt wordt dit verstuurd naar RabbitMQ. Dit wordt gedaan met het Advanced Message Queuing Protocol (AMQP)[7].



Figuur 6. Werking van RabbitMQ
(bron: https://www.rabbitmq.com/tutorials/amqp-concepts.html)

In het bovenstaande figuur vind u de werking van RabbitMQ. De publisher, in dit geval knowNow, stuurt een bericht (published) naar RabbitMQ. RabbitMQ zorgt ervoor dat het bij een of meerdere Consumers terecht komt. Voor knowNow is dit onder andere Lucene.NET.

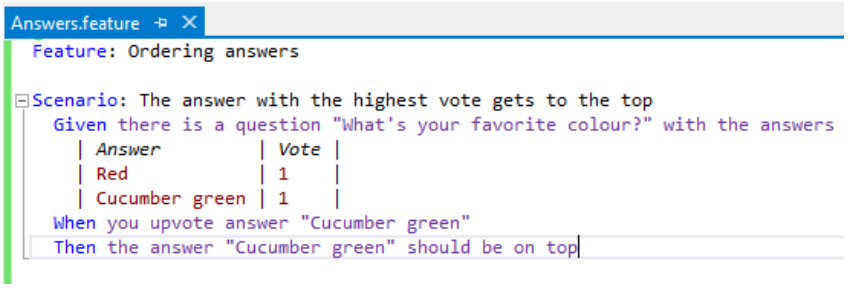
Binnen RabbitMQ zitten een aantal onderdelen die zorgen voor de afhandeling. De eerste is een Exchange. Een exchange is het onderdeel waar het bericht naar toe wordt verzonden door de publisher. Er zijn verschillende soorten Exchanges. knowNow maakt gebruik van een Topic Exchange[6]. Een Topic Exchange routeert berichten op basis van een 'routing key'. Een routing key is sleutel waarmee aangegeven wordt waar het bericht over gaat. Een topic exchange kan aan de hand van deze sleutel het bericht doorzetten naar de correcte queues. Queues zijn wachtrijen waar consumers de berichten van kunnen inlezen en verwerken.

4.8.3 Soorten events

knowNow kent verschillende events. Deze events worden verstuurd aan RabbitMQ bij diverse gebeurtenissen. Deze gebeurtenissen zijn bijvoorbeeld nieuwe, aangepaste of verwijderde artikelen(knowledge), profielen, communities, reacties en innovation pitches. Ook wanneer een persoon een profiel, community of innovation pitch volgt of ontvolgt wordt hier een event van verstuurd.

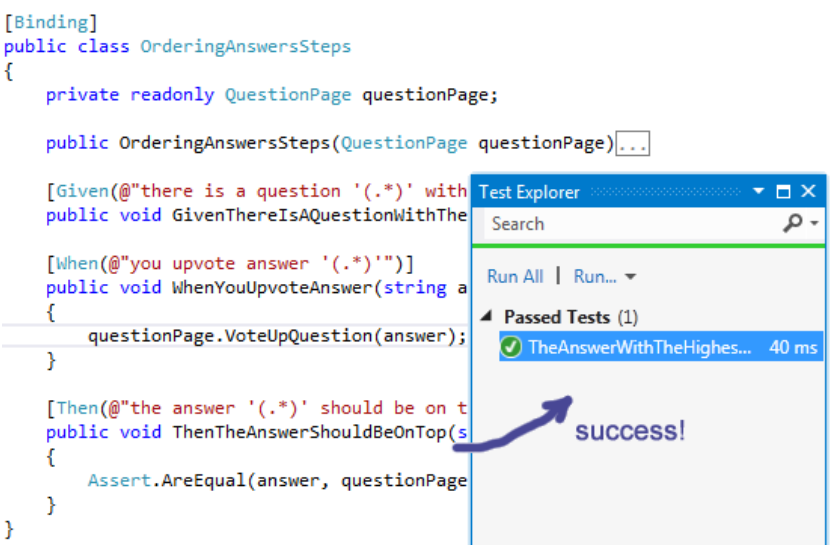
4.9 Testen

Het testen is voor het knowNow DevOps team erg belangrijk. Er zijn daarom een geautomatiseerde testen ontwikkeld. Een deel hiervan zijn zogeheten geautomatiseerde ui testen. Deze testen worden uitgevoerd door Specflow & Selenium. Specflow is een manier waarmee de business leesbare unittestscenario's kan uitschrijven. Dit staat in 'feature' bestanden.



Figuur 7. Voorbeeld scenariobeschrijving SpecFlow (Bron: http://specflow.org)

In een C# klasse kan er daarna een 'binding' aangemaakt worden. Hier wordt de beschrijving naar een uitvoerbare unit test vertaald.



Figuur 8. Voorbeeld van een binding (Bron: http://specflow.org)

Met Selenium is het mogelijk geautomatiseerde UI testen uit te voeren voor web applicaties. Selenium voert de testen uit in een webbrowser en vergelijkt het resultaat met het verwachte resultaat van de test

Er zijn nu meer dan 500 testen die de knowNow applicatie testen. Dit zijn zowel normale Unit Tests alsmede Selenium testen.

4.10 Security

De SQL Server database van knowNow is afgeschermd en niet voor iedereen bereikbaar. De gebruiker moet op de server zijn ingelogd voordat de gebruiker deze kan benaderen.

Gebruikers van de knowNow applicatie dienen eerst in te loggen. Dit kan via inloggegevens die alleen te gebruiken zijn voor knowNow, maar het is, wanneer de installatie daar op voorbereid is, ook mogelijk om via een Active Directory Account in te loggen. Dit kan door middel van een Microsoft Active Directory Federation Services (ADFS).

Eenmaal ingelogd in de applicatie heeft een gebruiker standaard toegang tot alle publieke toegankelijke knowledge items, Communities, Projects, Innovation Pitches en de profielen van alle gebruikers van knowNow.

Naast de publiekelijke toegankelijke Knowledge Items, Communities, Projects, en Innovation Pitches kan een gebruiker ook toegang hebben tot niet-publiekelijk toegankelijke items. Dit gebeurt wanneer de auteur van het item opgeeft dat het een 'Private' item is. Dan kunnen alleen de auteur en co-auteurs de items bekijken en mensen die in een Community zitten waar een item onder valt.

Item	Public/Private	Toegankelijk voor
Knowledge Item	Public	Iedere ingelogde gebruiker
	Private	Iedere ingelogde gebruiker die lid is van de community waar het item onder valt en de auteur en co-auteurs
Communities	Public	Iedere ingelogde gebruiker, een gebruiker kan alleen lid worden na toestemming auteur of beheerders van community.
Projects	Public	Iedere ingelogde gebruiker
	Private	Alleen de auteur
Innovation Pitches	Public	Iedere ingelogde gebruiker
	Private	Alleen de auteur en co-auteurs

4.11 Migraties

Orchard maakt gebruik van zogeheten migraties. In migraties worden instructies geprogrammeerd die zorgen dat het datamodel van de applicatie wordt bijgewerkt wanneer hier aanpassingen aan zijn gedaan.

Dit wordt bereikt door middel van het stappelen van de migraties. In de Orchard knowNow applicatie staat voor elke versie wat de veranderingen zijn t.o.v. van de versie daarvoor.

Zo kan, ongeachte de versie van de knowNow installatie, elke stap doorlopen worden om tot de nieuwste versie te komen. Om in het voorbeeld hierboven van v0.1 naar v0.4 te komen worden Migratie 1,2 en 3 uitgevoerd. Als de versie van de installatie v0.3 is, hoeft er maar 1 migratie te worden uitgevoerd, namelijk Migratie 3

Versie	Migratie ten opzichte van vorige versie
V0.1	-
V0.2	Migratie 1
V0.3	Migratie 2
V0.4	Migratie 3

Door het gebruik van migraties is knowNow eenvoudig consistent te houden over meerdere omgevingen. Er hoeft niet meer handmatig het datamodel op elke omgeving aangepast te worden. De beheerders van knowNow hoeven alleen nog maar de migratie te starten, de rest wordt automatisch gedaan.

5. Database architectuur

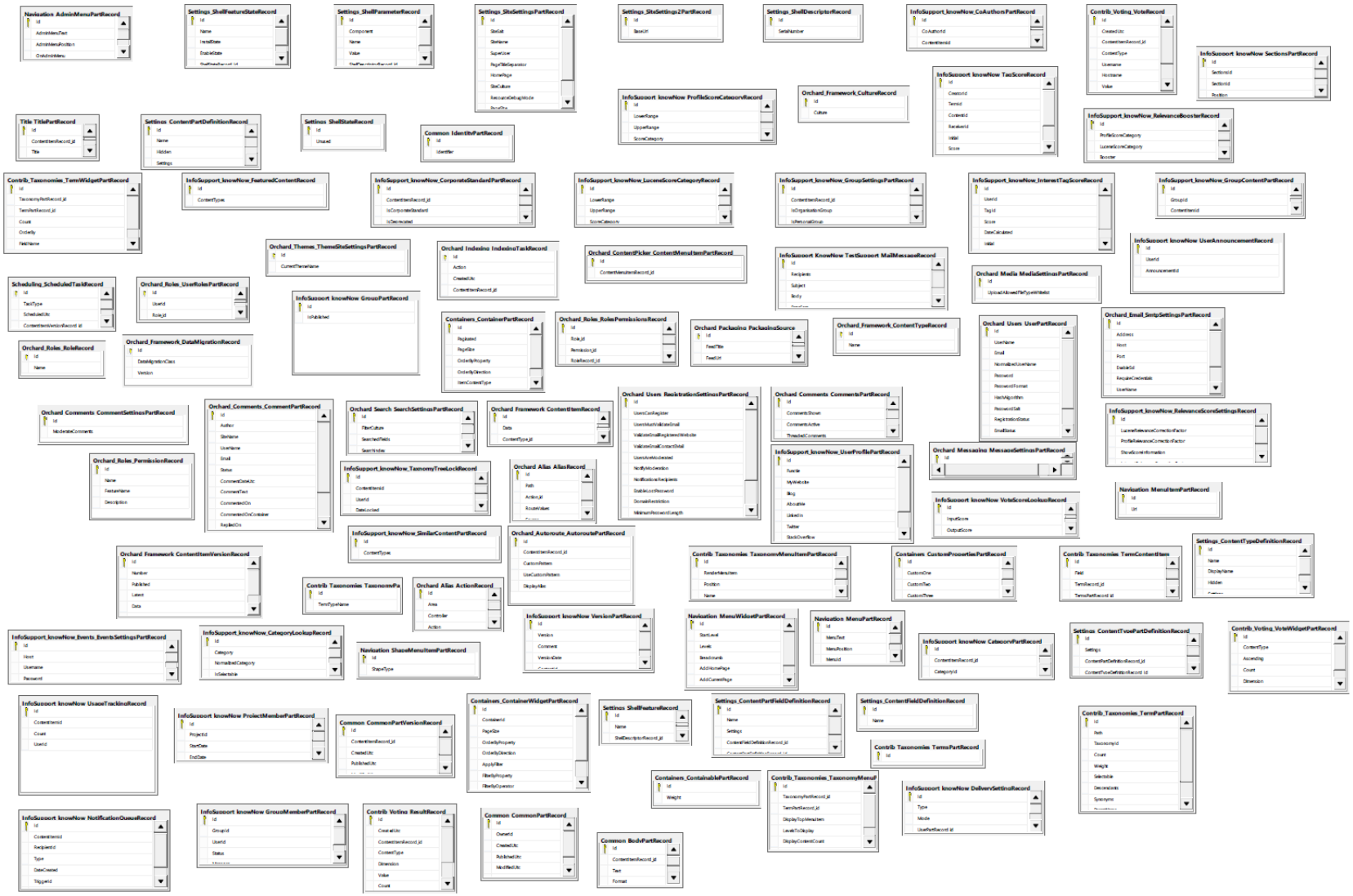
5.1 Inleiding

In dit hoofdstuk wordt de architectuur van de database van knowNow beschreven. knowNow maakt gebruik van het Relationele Database Management Systeem (RDBMS) Microsoft SQL Server. De gebruikte versie is 2014. In 5.2 is de opbouw van content besproken. De database architectuur is hier dan ook op ontworpen. Er zijn geen tabellen voor elk soort entiteit (Community, Knowledge item etc). Maar deze zijn onderverdeeld in de Content Types, Content Parts, en Content Fields.

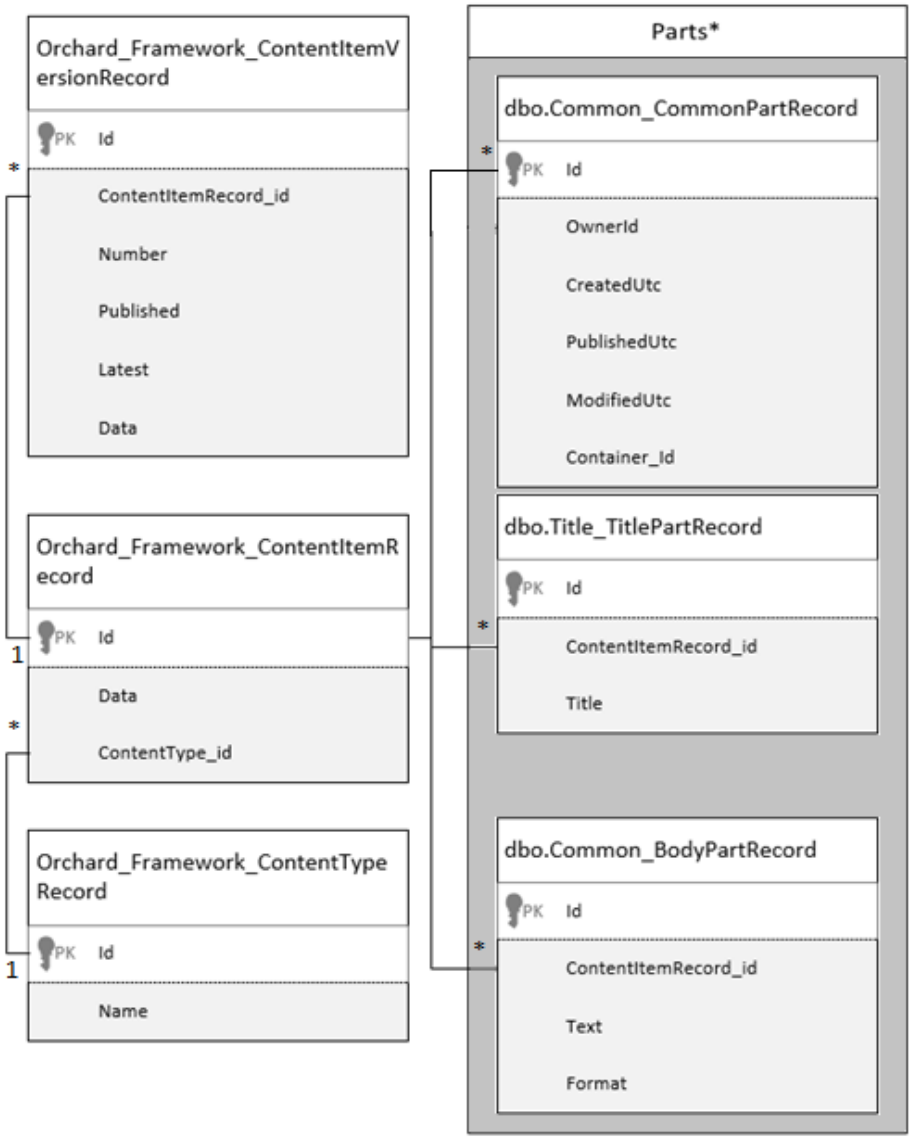
5.2 Diagram

5.2.1 Gehele database

In het diagram op de volgende pagina staat een diagram van de gehele database van knowNow. Orchard maakt geen Foreign Keys aan voor de relaties. Daardoor is het diagram erg onduidelijk. De tabellen lijken allemaal los van elkaar te zijn, terwijl er wel degelijk relaties zijn, deze zijn dus alleen niet aangegeven door middel van Foreign Keys en worden door de database niet gehandhaafd. Deze relaties zijn er niet omdat dit het eenvoudiger maakt om bij relaties records en tabellen aan te passen.



5.2.2 Gescopte diagram



** Parts kunnen per Content Type verschillen. De relaties tussen een ContentItem en PartRecord kunnen per ContentType verschillen.*

In het bovenstaande diagram staat een deel van de database van knowNow. Alleen het deel wat nodig is voor het onderzoek (zie het Plan van Aanpak), is in het diagram verwerkt. De relaties zijn hier zelf gelegd om het diagram te verduidelijken. De database bevat zelf geen Foreign Keys.

5.3 Werking

Aan de hand van dit voorbeeld situatie wordt de werking getoond van de database architectuur. Hierdoor wordt getracht de werking die in 3.2., 4.1. en 4.2. beschreven is te verduidelijken.

Er zijn voor dit voorbeeld 3 entiteiten:

Entiteit	Kolommen
Persoon	• Naam • Geboortedatum
Plaats	• Naam • Gemeente • Land
Auto	• Kenteken • Brandstof • Merk

Deze entiteiten hebben geen onderlinge relaties.

Elke soort entiteit (Persoon, Plaats & Auto) is een eigen Content Type. De verschillende Content Types worden in de tabel Orchard_Framework_ContentTypeRecord opgeslagen.

ID	Naam
1	Persoon
2	Plaats
3	Auto

De rijen van de entiteiten zijn Content Items. Alle Content Items horen bij een Content Part. Wanneer elke soort entiteit 2 rijen heeft zijn er in totaal dus 6 Content Items. De Content Items worden in de tabel Orchard_Framework_ContentItemsRecord opgeslagen.

Id	Data*	ContentType_Id**
1	NULL	1
2	NULL	1
3	NULL	2
4	NULL	2
5	NULL	3
6	NULL	3

** Data bevat voor de ContentItemsRecords altijd NULL-waarden.*
*** ContentType_Id verwijst naar het bijhorende ContentTypeRecord.*

De data wordt daarna onder verdeeld in Parts. In het Orchard systeem zijn er voor dit voorbeeld 4 parts: Naam, Persoon, Plaats & Auto. De Naam-part wordt zowel gebruikt door persoon als door plaats.

Entiteit	Parts
Persoon	Naam-part Persoon-part
Plaats	Naam-part Plaats-part
Auto	Auto-part

Elk part heeft weer zijn eigen veld.

Part	Fields
Naam-part	• Naam
Persoon-part	• Geboortedatum
Plaats-part	• Gemeente • Land
Auto-part	• Kenteken • Brandstof • Merk

Elk part heeft zijn eigen tabel. Bijvoorbeeld Naam_NaampartRecord of Persoon_PersoonpartRecord.

Deze tabellen hebben voor elke field een kolom. Naast de kolommen hebben ze ook een unieke sleutel en een verwijzing naar het bijhorende ContentItem.

Een voorbeeld van de fictieve tabel Naam_naampartRecord staat hieronder.

Id	ContentItemRecord_id	Title
1	1	Naam van persoon #1
2	1	Naam van persoon #2
3	2	Naam van plaats #1
4	2	Naam van plaats #2

6. Referenties

[1]	Zendesk, What is MailChimp and why should I use it? https://artfully.zendesk.com/entries/23201362-What-is-MailChimp-and-why-should-I-use-it-
[2]	Neo4j, What is Neo4j? http://neo4j.com/developer/graph-database/
[3]	Orchard, Understanding Data Access http://docs.orchardproject.net/Documentation/Understanding-data-access
[4]	Orchard Gallery https://gallery.orchardproject.net/
[5]	NHibernate http://nhibernate.info/
[6]	RabbitMQ https://www.rabbitmq.com/
[7]	AMQP http://www.amqp.org/about/what
[8]	Nginx, What is Load Balancing? http://nginx.com/resources/glossary/load-balancing/
[9]	ACC ICT, De OTAP-ontwikkelstraat: vier fases https://www.acc-ict.com/blog/de-otap-ontwikkelstraat-vier-fases
[10]	Techopedia, “Platform” (Online) http://www.techopedia.com/definition/3411/platform
[11]	Learn REST: A Tutorial http://rest.elkstein.org/
[12]	W3Resource, “JSON Introduction” http://www.w3resource.com/JSON/introduction.php

DE HAAGSE HOGESCHOOL

Info Support Solid Innovator

Bijlage D – Requirementsrapport

MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?

Requirementsrapport

Titel	Requirementsrapport
Project/Onderwerp	MongoDB: Een geschikte database voor het kennismanagementsysteem knowNow?
Versie	1.0
Status	Definitief
Datum	28-07-2015
Bestand	Requirementsrapport 1.0
Bedrijf	Info Support
Door	Tom Keim

Info Support B.V. Kruisboog 42, 3905 TG, Veenendaal, Tel. +31 (0) 318 55 20 20 - Fax +31 (0) 318 55 23 55
Kenniscentrum De Smalle Zijde 39, 3903 LM Veenendaal, Tel. +31 (0) 318 50 11 19 - Fax +31 (0) 318 51 83 59

Inhoudsopgave

1. Inleiding

In dit rapport zijn de requirements beschreven van de Product Owner voor het Proof-of-Concept van knowNow. Eerst worden de stakeholders beschreven, waarna de requirements worden besproken. De requirements zijn opgesteld na interviews met de Product Owner.

Het Proof-of-Concept heeft als doel aan te tonen dat MongoDB voor knowNow een werkend alternatief is in een Microservice architectuur. De architectuur van het Proof-of-Concept is een Microservice architectuur omdat dit als beste mogelijke architectuur naar voren kwam uit het onderzoek (Onderzoeksrapport, paragraaf 3.2).

Er kon nog niet worden besloten welk documentschema het beste gehanteerd kan worden voor knowNow. Er zijn 2 mogelijke varianten. De eerste variant heeft een hiërarchisch documentschema, waarbij alle data van relaties is opgenomen als subdocument. De andere variant heeft een relationeel documentschema waarbij alleen wordt verwezen naar de relaties.

De requirements in dit rapport gelden voor beide varianten(hiërarchisch documentschema & relationeel documentschema) van het Proof-of-Concept wanneer niet anders vermeld.

In het Functioneel Ontwerp zijn de uitgewerkte Use Cases te vinden.

2. Stakeholders

Dit rapport omvat de volgende stakeholders:

Naam	Rol
Joop Snijder	De Product Owner van knowNow.
Wiljag Denakamp	De Lead Architect van knowNow.
Tom Keim	Afstudeerder bij Info Support.
Gebruiker	Een gebruiker van het Proof-of-Concept

3. Prioritering van Requirements

De requirements zijn geprioriteerd volgens de MoSCoW-prioritering (Stapleton, 2002).

Mo**SC**o**W** staat voor **M**ust have, **S**hould have, **C**ould have en **W**on't have this time.

Prioriteit	**Omschrijving**	**Afkorting***
Must have	Deze requirement moet het system hebben, zonder dit requirement werkt het systeem niet.	M
Should have	Hoewel niet wenselijk, kan besloten worden wanneer het niet anders kan dit requirement niet te realiseren.	S
Could have	Het requirement heeft een toegevoegde waarde maar het systeem kan ook zonder dit requirement.	C
Won't have at this time	Deze requirements worden niet in de release meegenomen.	W

** De afkorting wordt verder in dit document gebruikt om de prioriteit van een requirement aan te geven.*

4. Knowledge Item

In dit document wordt gesproken over Knowledge Items. Knowledge Items zijn de kennisitems van knowNow. In het Proof-of-Concept wordt uitgegaan van de Knowledge Items zoals ze op dit moment al in het huidige knowNow bestaan. Voor het Proof-of-Concept vinden geen aanpassingen plaats aan de attributen van een Knowledge Item.

De attributen van een Knowledge Item zijn als volgt:

Attribuut	**Omschrijving**
Title	De titel van het knowledge item.
Description	De omschrijving van het knowledge item.
Rating	De beoordeling van het knowledge item.
Comments	Reacties op het knowledge item met daarin de tekst van de reactie, de schrijver van de reactie en de datum van de reactie.
Tags	Tags van het knowledge item. Tags zijn trefwoorden uit het Knowledge Item. Op basis van de tags kan ook worden gezocht naar artikelen die diezelfde tag bevatten.
Authors	Een Knowledge item wordt geschreven door een auteur. De gebruiker kan zelf opgeven wie de Auteurs en Medeauteurs (Co-authors) van het Knowledge Item zijn.
Followers	Het aantal mensen dat een knowledge item volgt. Wanneer een gebruiker een knowledge item volgt, krijgt deze gebruiker updates wanneer het artikel wordt gewijzigd.
Communities	Communities waar het Knowledge Item onder valt. Communities kunnen bijvoorbeeld afdelingen binnen een bedrijf zijn of interessegebieden.
Views	Hoeveel mensen het Knowledge item hebben bekeken.
Date	De publicatiedatum van het Knowledge Item.
Visibility	Public of Private. Bij Public is het item voor iedere gebruiker zichtbaar, bij private is het item alleen zichtbaar voor de gebruikers.
Categorieën	De categorieën waaronder het Knowledge Item valt, de huidige categorieën zijn als volgt: • Article • Concept • Practice • Presentation • Sample • Standard & Guidline • Template • Tool De categorieën zijn configureerbaar in de applicatie.

5. User Requirements

De tabel hieronder bevat de User Requirements voor het Proof-of-Concept. Van de User Requirements worden Use Cases ontworpen, deze zijn te vinden in het Functioneel Ontwerp (Bijlage F).

ID	Requirement	Owner	Prioriteit
R.U4	De gebruiker moet Knowledge Items kunnen toevoegen	Opdrachtgever	M
R.U5	De gebruiker moet Knowledge Items kunnen wijzigen	Opdrachtgever	M
R.U6	De gebruiker moet Knowledge Items kunnen verwijderen	Opdrachtgever	M
R.U12	De gebruiker moet Knowledge Items aan de hand van een ID kunnen opvragen	Opdrachtgever	M
R.U14	De gebruiker moet kunnen inloggen	Opdrachtgever	W
R.U15	Een gebruiker mag alle Knowledge Items zien die public zijn zien	Opdrachtgever	C
R.U25	De gebruiker kan een lijst opvragen met alle Knowledge Items	Opdrachtgever	M
R.U28	De gebruiker moet People Items kunnen toevoegen	Opdrachtgever	S
R.U29	De gebruiker moet People Items kunnen wijzigen	Opdrachtgever	S
R.U30	De gebruiker moet People items kunnen verwijderen	Opdrachtgever	S
R.U31	De gebruiker moet alle People Items kunnen opvragen	Opdrachtgever	S
R.U32	De gebruiker moet één People item kunnen opvragen aan de hand van een ID	Opdrachtgever	S

6. Functional Software Requirements

De tabel hieronder bevat de Functionele Software Requirement voor het Proof-of-Concept.

ID	Requirement	Owner	Prioriteit	Verwijzing
R.S1	Het systeem moet alle velden van Knowledge Items* bevatten	Opdrachtgever	M	
R.S4	De gebruiker moet een Knowledge Item kunnen toevoegen	Opdrachtgever	M	R.U4
R.S5	De gebruiker moet een Knowledge Item kunnen wijzigen	Opdrachtgever	M	R.U5
R.S6	De Knowledge Service moet Knowledge Items kunnen verwijderen	Opdrachtgever	M	R.U6
R.S10	Het systeem kan het datamodel updaten met stapelbare migraties	Opdrachtgever	W	
R.S11	Het systeem moet revisies bijhouden van aanpassingen aan een Knowledge Item	Opdrachtgever	C	
R.S12	De gebruiker kan een Knowledge Items opvragen aan de hand van een ID	Opdrachtgever	M	R.U12
R.S14	Het systeem moet een gebruiker kunnen authentiseren	Opdrachtgever	W	R.U14
R.S15	Het systeem moet alle public items aan geauthentiseerde gebruikers kunnen laten zien	Opdrachtgever	C	R.U15
R.S16	Het systeem moet private items aan de geauthentiseerde auteurs en de leden van de community waar het item onder valt kunnen laten zien	Opdrachtgever	C	
R.S25	De gebruiker kan een lijst opvragen met alle Knowledge Items	Opdrachtgever	M	R.U25
R.S28	De gebruiker moet People Items kunnen toevoegen	Opdrachtgever	S	
R.S29	De gebruiker moet People Items kunnen wijzigen	Opdrachtgever	S	
R.S30	De gebruiker moet People items kunnen verwijderen	Opdrachtgever	S	
R.S31	De gebruiker moet alle People Items kunnen opvragen	Opdrachtgever	S	
R.S32	De gebruiker moet één People item kunnen opvragen aan de hand van een ID	Opdrachtgever	S	

7. Non-Functional Software Requirements

De tabel hieronder bevat de Niet-Functionele Software Requirement voor het Proof-of-Concept.

ID	Requirement	Owner	Prioriteit
R.S7	Het systeem moet om kunnen gaan met 200.000 gebruikers waarvan 15% concurrent aan het lezen is en 5% concurrent aan het schrijven is	Opdrachtgever	M
R.S8	Het systeem moet high-performant, binnen 100ms uit de database, Knowledge Items kunnen laten zien	Opdrachtgever	M
R.S18	Het systeem is uiteindelijk consistent (Eventual Consistent) met data van de andere services (R.S26)	Opdrachtgever	M
R.S17	Het systeem moet onderhoudbaar zijn, de architect (Wiljag) bepaald de mate van de onderhoudbaarheid dmv code review.	Opdrachtgever	C
R.S33	Het systeem moet in een keer alle velden van een Knowledge Item pagina kunnen teruggeven. De gebruiker moet zelf geen data meer combineren (joinen). (alleen voor Proof-of-Concept met hiërarchisch documentschema)	Opdrachtgever	M

8. Technische Eisen

De tabel hieronder bevat de Technische Eisen voor het Proof-of-Concept.

ID	Requirement	Owner	Prioriteit
E.T1	Het systeem bevat productiedata van knowNow	Opdrachtgever	C
E.T2	Het systeem heeft een Microservice architectuur	Opdrachtgever	M
E.T3	Het systeem wordt geschreven in C#	Afstudeerder	M
E.T4	Het systeem maakt gebruik van MongoDB	Opdrachtgever	M
E.T5	Het systeem moet een REST API hebben	Opdrachtgever	M
E.T6	Het systeem moet een Web interface hebben	Opdrachtgever	S
E.T7	De Knowledge Service moet configureerbare categorieën hebben	Opdrachtgever	S
E.T8	Het systeem bevat een People Service stub** om een People service na te bootsen	Afstudeerder	M
E.T9	Het systeem bevat een Taxonomy Service stub** om een Taxonomy service na te bootsen	Afstudeerder	W
E.T10	Het systeem bevat een Rating Service stub** om een Rating service na te bootsen	Afstudeerder	W
E.T11	Het systeem bevat een Community Service stub** om een Community service na te bootsen	Afstudeerder	W
E.T12	Het systeem bevat een Authentication Service stub** om een Authentication service na te bootsen	Afstudeerder	W
E.T13	Het systeem bevat een Comments Service stub** om een comments service na te bootsen	Afstudeerder	W

** Een stub is stuk programmatuur van een deel van een applicatie wat nog niet bestaat, maar wel getest moet worden. Er wordt dan software geschreven welke de later te ontwikkelen software tijdelijk vervangt.