

Scriptie – Jonker BV.

De centralisering van Jonker’s softwarepakket

Versie 1.0

Jochem Pouwels (1613727)

Software Engineering VT

Docentbegeleider: Rik Boss

Eerste examinator: Rory Sie

Afstudeerperiode: 26 maart t/m 13 oktober

Scriptie

Samenvatting:

Dit document doet verslag van mijn afstudeertraject bij Jonker.

**Auteur Instelling**

Jochem Pouwels Hogeschool Utrecht

Lammerenhof 76 Faculteit Natuur en Techniek

6902BA te Zevenaar Nijenoord 1

[jochem.pouwels@student.hu.nl](mailto:jochem.pouwels@student.hu.nl) 3552AS te Utrecht

06 519 130 51

**Bedrijfsbegeleider Docentbegeleider Opdrachtgever**

Marco Zieverink Rik Boss Jonker BV

[m.zieverink@jonker.nl](mailto:m.zieverink@jonker.nl) [rik.boss@hu.nl](mailto:rik.boss@hu.nl) De mors 19

06 247 996 40 088 481 82 83 7151MX te Eibergen

Voorwoord

Voor u ligt de scriptie betreffend de afstudeeropdracht: ‘De centralisering van Jonker’s softwarepakket’. Een onderzoek naar hoe en welke bestaande technieken het best gebruikt kunnen worden om de administratiesoftware van Jonker te centraliseren. Deze scriptie is geschreven in het kader van mijn afstuderen aan de opleiding Software Engineering aan de Hogeschool van Utrecht. Van 26 maart 2015 tot en met 13 oktober 2015 ben ik bezig geweest met de opdracht en het schrijven van deze scriptie.

De vraag naar dit onderzoek is tot stand gekomen in lijn met de toekomstplannen van Jonker, samen met de hulp van mijn bedrijfsbegeleider Marco Zieverink heb ik hiervoor een geschikte afstudeeropdracht weten te specificeren.

Het onderzoek wat ik heb uitgevoerd was groot, maar na uitvoerig kwalitatief onderzoek heb ik de onderzoeksvragen weten te beantwoorden.

Graag wil ik Jonker bedanken voor de kans die ze mij hebben geboden om deze uitdagende afstudeeropdracht bij hen uit te voeren en ook voor de hulp waarvan ze mij hebben voorzien. Deze hulp kwam in de vorm van tijd, moeite en het beschikbaar stellen van materialen. Daarnaast wil ik alle collega’s bedanken voor alle feedback en niet te vergeten, de fijne werksfeer.

Ten slotte wil ik de volgende personen nog in het bijzonder bedanken:

* Jurrian Vlijm – Voor het aanbieden van deze opdracht
* Marco Zieverink en Kjeld Kiezenbrink – Voor het vele sparren over gemaakte en gewenste resultaten, hierdoor kon ik doelgericht werken en is uiteindelijk het eindresultaat van hogere kwaliteit geworden.
* Rik Boss – Voor het altijd spoedig beantwoorden van mijn vragen en voor zijn erg fijne en doelmatige manier van feedback geven.

Ik wens iedere lezer veel leesplezier toe.

*Jochem Pouwels*

*13 oktober 2015*

Management samenvatting

Dit afstudeerproject is uitgevoerd om Jonker te helpen bij het opzetten van de nieuwe versie van hun administratieve softwarepakket. De huidige versie kwam voor het eerst uit in 2004. Internetvoorzieningen bij de klanten van Jonker waren toen nog niet stabiel genoeg om het softwarepakket hierop te gaan bouwen. Daarom is toen besloten lokaal een server en een database te plaatsen bij de klanten, waarmee hun softwarepakket kan communiceren.

Nu in 2015 is Jonker van mening dat het tijd is om het softwarepakket te centraliseren en wil alle business logica wel bereikbaar maken via het internet. Het probleem is echter dat er kennis ontbreekt in het software development team, waardoor ze niet zeker weten welke softwaretechnieken ze hiervoor moeten gaan gebruiken. Naar aanleiding van dit probleem is de volgende doelstelling opgesteld: “Jonker inzicht geven wat technieken die tegenwoordig op de markt zijn kunnen betekenen bij het vernieuwen van hun administratieve softwarepakket naar een gecentraliseerde omgeving.

Er is kwalitatief onderzoek gedaan om de juiste kennis te vergaren. De vraag die centraal stond in dit onderzoek luidde: Hoe kan de huidig gedecentraliseerde back-end van Jonker’s administratiesoftware omgezet worden naar een gecentraliseerde omgeving die ontsloten wordt naar verschillende clients door middel van een generieke service laag? Om deze vraag te beantwoorden zijn er verschillende deelvragen opgesteld die tezamen de centrale vraag kunnen beantwoorden. De volgende deelvragen zijn onderzocht:

* **Deelvraag 1** – Is de huidige database nog het meest geschikt voor een gecentraliseerde opzet van de administratiesoftware?
* **Deelvraag 2** – Op welke manieren kunnen bestaande persistence frameworks voordelen opleveren voor de nieuwe back-end?
* **Deelvraag 3** – Welke web services zijn er geschikt voor de nieuwe back-end en wat zijn de voor en nadelen van elk?
* **Deelvraag 4** – Welke business logica uit het softwarepakket wat omgezet gaat worden naar de nieuwe back-end kan worden verplaatst in de service laag.
* **Deelvraag 5** – Hoe kan er aangetoond worden dat het omgezette softwarepakket hetzelfde functioneert als voor de omzetting.

Door bronnenstudie, kennis al in het bezit, analyseren van het huidige pakket en het opstellen van een test plan is er antwoord kunnen geven op de onderzoeksvragen. Hieronder staan de belangrijkste resultaten op een rijtje.

* De huidige Firebird database is niet meer de juiste database voor de nieuwe versie van het software pakket, de PostgreSQL database wel.
* NoSQL databases hebben zo hun voordelen, maar zijn op dit moment niet geschikt voor Jonker.
* De structuur van de relationele database moet ook aangepakt worden.
* Het MyBatis persistence framework biedt de meeste voordelen voor Jonker.
* Voor de generieke service laag kan het best een REST web service gebruikt worden.

De conclusie is dat de huidig gedecentraliseerde back-end van Jonker’s administratiesoftware het best omgezet kan worden door eerst de database te vervangen met een nieuwe PostgreSQL database waarbij ook de structuur van de database wordt verbeterd met id’s, foreign keys en nieuwe datatypes. Er moet een REST service worden gebouwd waarbij methodes idempotent moeten worden ingericht. De REST service kan het best met de database verbinden door gebruik te maken van het MyBatis persistence framework, omdat dit het persistence framework is met de meeste voordelen voor Jonker. Per module van het software pakket moet bekeken worden welke business logica geschikt is om te plaatsen in de service laag. Dit kan worden gedaan door het pakket te analyseren en de usecases eruit af te leiden.

Voor de productie module is een proof of concept gemaakt. Hiervoor is een service laag opgezet die de gekozen technieken uit het onderzoek gebruikt. Daarnaast is een nieuwe web client gemaakt met JSF2 en PrimeFaces. De nieuwe situatie is getest door het schrijven en laten uitvoeren van een acceptatietest. Deze test is met succes doorlopen.

Inhoudsopgave

1 Lijst met figuren 9

2 Lijst met begrippen 11

3 Inleiding 12

4 Achtergrond 13

4.1 Organisatiebeschrijving 13

4.2 Plaats van de afstudeerder 14

4.2.1 Communicatie 14

4.3 Technisch context 15

5 Opdrachtomschrijving 16

5.1 Probleemstelling 16

5.2 Opdrachtformulering 16

5.3 Doelstelling 18

5.3.1 Doelstelling van het afstudeerproject 18

5.3.2 Doelstelling van Jonker 18

5.4 Eindproducten 19

6 Onderzoeksopzet 20

6.1 Structuur/werkwijze van het onderzoek 20

6.1.1 Verkenningsfase 20

6.1.2 Verzamelfase 21

6.1.3 Selectiefase 21

6.1.4 Verwerkfase 21

6.2 Onderzoeksvragen 21

6.3 Methode van onderzoek 22

6.3.1 Deelvraag 1 22

6.3.1.1 Deelvraag 1a: Wat zijn NoSQL databases en welke voordelen en/of nadelen hebben ze ten opzichte van relationele databases? 22

6.3.1.2 Deelvraag 1b: Wat zijn de voor en nadelen van de relationele databases die tegenwoordig op de markt zijn? 22

6.3.2 Deelvraag 2 23

6.3.3 Deelvraag 3 23

6.3.4 Deelvraag 4 23

6.3.5 Deelvraag 5 24

7 Oriëntatie huidige situatie 25

7.1 De applicatie 25

7.1.1 Modules 26

7.2 De database 27

7.2.1 Eerste blik 27

7.2.2 Foreign keys 27

7.2.3 Verbinding met client applicatie 27

8 Onderzoeksresultaten 28

8.1 Deelvraag 1a: NoSQL databases 28

8.1.1 Algemene voor- en nadelen 28

8.1.2 Graph databases 29

8.1.3 Key-value stores 30

8.1.4 Document stores 30

8.1.5 Conclusie 30

8.2 Deelvraag 1b: Relationele databases 31

8.2.1 Conclusie 32

8.3 Deelvraag 2: Persistence framework 32

8.3.1 Voordelen van JOOQ 33

8.3.2 Nadelen van JOOQ 33

8.3.3 Voordelen van MyBatis 33

8.3.4 Nadelen van MyBatis 34

8.3.5 Conclusie 34

8.4 Deelvraag 3: Web service communicatie 35

8.4.1 Wat is SOAP? 35

8.4.2 Wat is REST? 35

8.4.3 Verschillen SOAP en REST 36

8.4.4 Wanneer SOAP niet gebruiken? 36

8.4.5 Wanneer SOAP wel gebruiken? 36

8.4.6 Wanneer REST niet gebruiken? 37

8.4.7 Wanneer REST wel gebruiken? 37

8.4.8 Conclusie 37

8.5 Deelvraag 4: Business logica 37

8.6 Deelvraag 5: Test 38

9 Proof of Concept 39

9.1 Voorbereiding 39

9.2 Functionele specificaties 39

9.3 Technische specificaties 45

9.3.1 Code conventies 45

9.3.2 Maven 45

9.3.3 Applicatieserver 45

9.3.4 REST 45

9.3.5 PostgreSQL 46

9.3.6 MyBatis 46

9.3.7 JSF + Primefaces 47

9.3.8 Beveiliging 47

9.4 Het eindresultaat 48

9.4.1 Inloggen 48

9.4.2 Het startmenu 49

9.4.3 De productiemodule 49

9.4.4 Conclusie 52

10 Afwijking plan van aanpak 53

10.1 Authenticatie 53

10.2 Proof of concept (client) 53

11 Terugblik eindresultaten 54

11.1 Onderzoeksresultaten 54

11.2 Proof of concept 54

11.3 Testrapport 54

11.4 Scriptie 54

12 Evaluatie 55

12.1 Proces **Error! Bookmark not defined.**

12.2 Product **Error! Bookmark not defined.**

13 Literatuurlijst 56

14 Bijlagen 57

14.1 Bijlage A – Plan van aanpak 57

14.2 Bijlage B – NoSQL onderzoeksresultaten 74

14.3 Bijlage C – Relationele databases onderzoeksresultaten 89

14.4 Bijlage D – Persistence framework onderzoeksresultaten 108

14.4 Bijlage E – Web service onderzoeksresultaten 121

14.4 Bijlage F – Test plan: proof of concept 132

# Lijst met figuren

Figuur 1 – Personeelsindeling 14

Figuur 2 – Softwarestructuur nieuwe basis 17

Figuur 3 – Mindmap met het onderwerp: “database” 20

Figuur 4 - Voorbeeld graph database 29

Figuur 5 – Het productieoverzicht 43

Figuur 6 – De productie informatie popup 43

Figuur 7 – De productie filter popup 44

Figuur 8 – De kolomconfiguratie popup 44

Figuur 9 - PoC: inlogscherm 48

Figuur 10 - PoC: startmenu 49

Figuur 11 - PoC: Productieoverzicht 50

Figuur 12 - PoC: filter criteria 50

Figuur 13 - PoC: uitgebreide informatie 51

Figuur 14 - PoC: kolomconfiguratie 51

Figuur 16 - Globale planning 69

Figuur 17 – Risicocijfers kans/schade 70

Figuur 18 - Risico matrix 71

Figuur 19 – Risico niveaus 71

Figuur 22 – Usecase: film database - Relationele database 83

Figuur 23 – Usecase: film database - Graph database 83

Figuur 24 - Query graph database 84

Figuur 25 - Query relationele database 85

Figuur 27 - Datastructuur van een document 87

Figuur 29 - Populariteitsscores juli 2015 94

Figuur 30 - Structuur RDF store 99

Figuur 31 - Ontwerp van de door ModelMapper gegenereerde Java objecten 115

Figuur 32 - normale situatie 126

Figuur 33 - request komt niet aan 127

Figuur 34 - response wordt niet ontvangen 127

Figuur 35 - Idempotentie oplossing 128

Figuur 36 - Werking WS-ReliableMessaging 129

\*Enkele figuren zijn niet opgenomen in deze lijst, de reden hiervoor is een bug in Microsoft Office Word.

# Lijst met begrippen

|  |  |
| --- | --- |
| Begrip | Definitie/verklaring |
| ADM | Het grootste onderdeel van de tClick software dat zich richt op de administratieve taken in de betonmortelindustrie. |
| Apache Subversion | Een opensource versiebeheersysteem |
| Betoncentrale | Fabriek waar betonmortel wordt geproduceerd |
| Betonmortel | een nog plastisch mengsel van cement, grof en fijn toeslagmateriaal en water waaraan vul- en hulpstoffen kunnen zijn toegevoegd. |
| Business logica | Het gedeelte van een applicatie dat taken encodeert specifiek aan de gerelateerde business. Het beschrijft hoe en welke data wordt gemaakt, weergegeven, opgeslagen en veranderd. |
| Databasemanagementsysteem | Een systeem dat als database opgeslagen gegevens ontsluit, bewaakt en beheert. |
| Dependency | Een Java klasse of library welke geïmporteerd moet worden in een project wil deze compileren. |
| Idempotent | De eigenschap van een programmaoperatie dat aangeeft dat er niks zal veranderen aan de programmaomgeving als deze meerdere keren wordt uitgevoerd. Alleen de eerste aanroep mag iets veranderen. |
| NoSQL | Een groepering van databasemanagementsystemen die op aanmerkelijke wijze verschillen van klassieke relationele databasemanagementsystemen. |
| Mindmap | Een diagram in de vorm van een boomstructuur. Opgebouwd uit begrippen, teksen, relaties en-of plaatjes. De mindmap wordt gebruikt om creatieve en analytische processen te ondersteunen. |
| Persistence framework | Software dat helpt bij het overbruggen van de conceptuele verschillen van data in een database en data in programmacode. |
| Replicatie | Een computertechniek waarmee bestanden en/of databases van het ene naar een ander computer- of opslagsysteem wordt gekopieerd. |
| Service laag | Software dat bedoelt is om de communicatie te faciliteren tussen verschillende applicaties, bijvoorbeeld tussen een Java/web client en een database. |
| tClick | tClick Software is een verzameling van softwareproducten van Jonker die naadloos aansluit op de processen zoals deze binnen de betonmortelindustrie worden uitgevoerd.  tClick draait op een Linux, Windows of OSX computer. |

# Inleiding

“Er zijn weer conflicten bij de database replicatie van bedrijf A” of “Voer jij de update uit bij bedrijf X? Dan doe ik dat bij bedrijf Y vanavond”. Dit zijn uitspraken die vaak zijn voorgekomen bij Jonker op de werkvloer. Ze hebben te maken met het feit dat het softwarepakket van Jonker samen met benodigde hardware (server/database) in de betoncentrales van de klant moet worden geplaatst. Een dergelijke gedecentraliseerde opzet is voor Jonker lastig te onderhouden en houdt de innovatie en de groei van het bedrijf tegen.

Jonker is van plan om een geheel nieuwe versie van hun softwarepakket te ontwikkelen, waarbij alle gebruikersdata en business logica beschikbaar wordt via het internet. Aan de afstudeerder is gevraagd om te gaan onderzoeken welke bestaande technieken gebruikt zouden kunnen worden in de volgende versie van het software pakket en welke voor en nadelen dit oplevert. Naast het onderzoek bevat de opdracht ook het maken van een proof of concept.

Door middel van bronnenstudie, tezamen met kennis van de afstudeerder zullen de volgende onderdelen worden onderzocht:

* Database – Welke past het best bij het ADM softwarepakket
* Persistence – Welke voordelen bieden bestaande persistence frameworks bij het ontwikkelen van Java code i.c.m. een database
* Service laag – Welke soort service laag kan het best geïmplementeerd worden

In dit document wordt eerst achtergrondinformatie gegeven over het bedrijf Jonker en hun activiteiten. Vervolgens wordt het project omschreven middels de probleemstelling, doelstelling, opdracht formulering en een beschrijving van de eindproducten. Daarna wordt het onderzoeksopzet beschreven gevolgd door een ondersteunende beschrijving van het huidige softwarepakket van Jonker. Het hoofdstuk hierna bevat een samenvatting van de resultaten die uit het onderzoek zijn gekomen, samen met de antwoorden op de deelvragen. In het daarop volgende onderdeel wordt het gemaakte proof of concept beschreven. Vanaf hier volgt nog een beschrijving en verantwoording van de afwijkingen die gemaakt zijn ten opzichte van het plan van aanpak. Tot slot wordt er nog een terugblik geworpen naar de eindresultaten en wordt het proces en het eindproduct geëvalueerd.

Bijgevoegd zijn nog de gebruikte bronnen in de literatuurlijst, en de bijlages.

In de bijlages kan het volgende gevonden worden:

* Plan van aanpak
* NoSQL databases onderzoeksresultaten
* Relationele databases onderzoeksresultaten
* Persistence framework onderzoeksresultaten
* Web service onderzoeksresultaten
* Acceptatietest van het proof of concept

# Achtergrond

In dit hoofdstuk wordt eerst de organisatiebeschrijving gegeven die de missie en kernactiviteiten van Jonker duidelijk moet maken. Ook wordt de plaats van de afstudeerder binnen deze organisatie duidelijk gemaakt en worden de communicatieafspraken van de afstudeerder benoemd. Tot slot wordt een uitgebreidere omschrijving gegeven van de ICT omgeving binnen het bedrijf om zo een duidelijker beeld te geven over de context van het afstudeerproject.

## Organisatiebeschrijving

Jonker is specialist op het gebied van besturingen en administratieve software voor processen waar wegen en doseren een belangrijk aspect is. Jonker is onder andere actief in de marktsegmenten betonmortel, betonwaren, droge mortel, asfalt, diervoeding, chemie en food.

Voor het nauwkeurig wegen en doseren heeft Jonker voor hun besturingen lange tijd eigen hardware in de vorm van printplaten en sensoren moeten ontwikkelen. Hardware van 25 jaar en ouder zijn hierbij geen uitzondering. Standaard componenten die heden ten dage op de markt zijn voldoen wel aan de eisen en daarom is Jonker overgestapt van eigen hardware naar standaard componenten. Jonker levert nog steeds besturingen aan klanten en heeft hier ook nog een aparte afdeling voor, echter door de overstap naar standaard componenten kon Jonker zich meer gaan focussen op hun software.

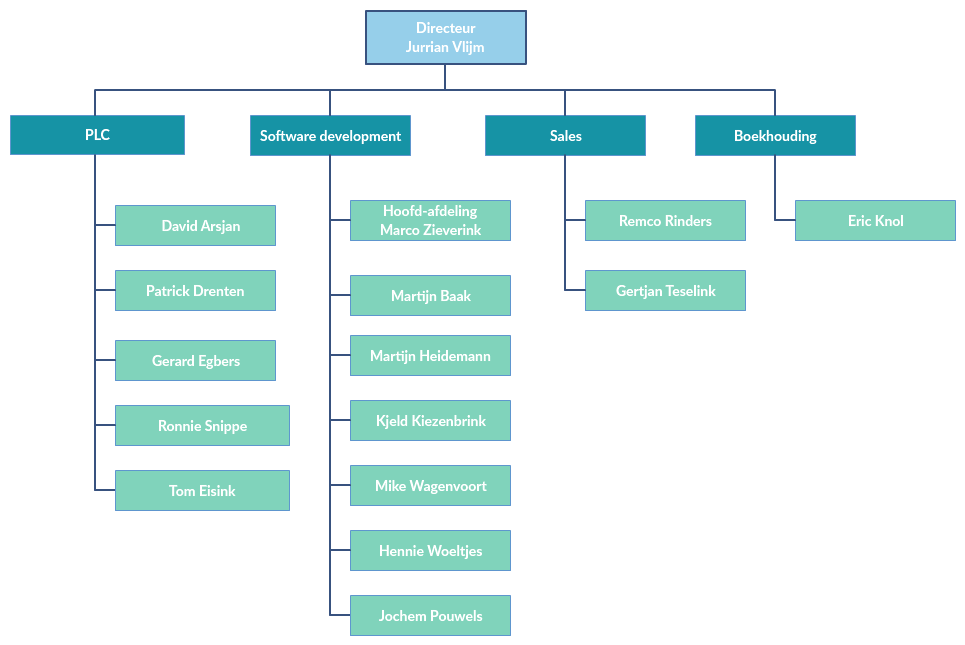
Een belangrijke stap voor Jonker is de ontwikkeling van de administratieve software. Bij veel bedrijven staat de administratieve software los van de besturingssoftware. Integratie van beiden levert vele voordelen op voor de klant.

## Plaats van de afstudeerder

Binnen Jonker bestaan er tegenwoordig vier afdelingen; PLC, software development, sales en boekhouding. Onderverdeeld onder deze afdelingen staan vijftien vaste werknemers. De afstudeerder is de zestiende werknemer.

De afstudeerder bevindt zich in het software development team en zit hier samen met de zes anderen op één kantoor. De indeling van het kantoor is zo gemaakt dat degene die het meest met elkaar moeten bespreken bij elkaar komen te zitten. De afstudeerder zit dichtbij zijn bedrijfsbegeleider en een andere collega waarmee hij over het ADM pakket kan sparren, Kjeld Kiezenbrink.

De organisatie van Jonker is hieronder weergegeven in een diagram.



Figuur 1 – Personeelsindeling

### Communicatie

De volgende interne communicatie afspraken zijn gemaakt.

* Elke vrijdag wordt er een voortgangsbespreking gehouden tussen de afstudeerder en bedrijfsbegeleider en minimaal één extra collega.
* Indien de bedrijfsbegeleider op dat moment niet beschikbaar is, wordt er eerst gekeken of het de maandag erna kan, zo niet dan wordt een plaatsvervangende collega ingezet.
* Indien de afstudeerder tijdens het onderzoek ontdekkingen doet die besproken moeten worden, kan dit besproken worden met bij voorkeur de bedrijfsbegeleider of collega Kjeld Kiezenbrink

## Technisch context

Jonker’s administratieve software voegt allerlei taken samen in één pakket, tClick ADM. tClick ADM dient om betonmortelcentrales te ondersteunen in een efficiënte bedrijfsvoering. Het pakket bestaat uit meerdere modules waarmee de gebruikers verschillende bedrijfsprocessen kunnen automatiseren. Naast handige modules voor het maken van offertes en het factureren van klanten zijn er ook modules die specifiek voor betonproductie zijn ontwikkeld.

Zo kunnen gebruikers zelf nieuwe betonrecepten doorberekenen met de laboratorium module. Aan de hand van specifieke eisen zoals sterkte, soort, kleur, chlorideklasse, weerstand en consistentie worden ingewikkelde algoritmen gebruikt om de goedkoopste samenstelling te maken voor het beton.

Bij het maken van een order zorgt tClick ADM ervoor dat de machines in de betoncentrales meteen beginnen met het mengen van het bestelde beton.

Na het maken van het beton moet deze ook nog geleverd worden, het leveren van beton is een ingewikkeld proces, vooral als er meerdere klanten tegelijk beton bestellen. Er moet rekening worden gehouden met beschikbaarheid van de truckmixers, loscapaciteit van elke klant, en droogsnelheid van het beton.

Door middel van complexe algoritmen en een partnerrelatie met TomTom om de GPS locatie en status van elke truckmixer bij te houden, kan tClick ADM de gehele planning op de minuut precies volledig automatisch regelen zonder dat hier input van de gebruiker nodig is.

De tClick ADM software is ontwikkeld met Java als programmeertaal en gebruikt een GUI gemaakt met Java‘s Swing API. De client verbindt direct met de database door gebruik te maken van een JDBC driver waarmee zelfgemaakte query’s worden uitgevoerd.

De back-end van tClick ADM is gedecentraliseerd, wat betekent dat de data die gebruikt wordt in de software niet op één plek verwerkt wordt. Elke vestiging van de klant heeft zijn eigen server en database.

De klant verbindt met hun tClick ADM client met de server in de vestiging waar hij zich bevindt, waarna deze de data verwerkt op de lokale Firebird database. Deze database maakt gebruik van eigen geschreven replicatiesoftware, dit is software dat er voor zorgt dat de data gesynchroniseerd wordt naar de andere gekoppelde databases die op andere vestigingen van de klant staan. Alle databases kunnen zowel data schrijven als lezen, deze opstelling heet multi-master. De replicatiesoftware houdt rekening met kleine internetstoringen en gebruikt daarom per vestiging een kleine marge in de records van de database, zo worden bijvoorbeeld bepaalde factuurnummers gereserveerd om conflicten te voorkomen met het synchroniseren van de databases.

# Opdrachtomschrijving

## Probleemstelling

tClick ADM is vooral geschikt voor een bepaald marktsegment, de betonwereld. De software wordt al door vele klanten gebruikt. Er is in 2004 voor een gedecentraliseerde back-end constructie gekozen omdat toentertijd de internet voorzieningen nog niet overal goed op orde was. Dat ergens de verbinding er een uur uit lag werd niet als uitzondering gezien.

Door de gedecentraliseerde structuur is het noodzakelijk voor Jonker om multi-master replicatiesoftware te gebruiken om de data bij elke vestiging van de klant gelijk te houden, dit heeft echter nadelen voor Jonker en hun klanten. Voor de klant is het vervelend omdat hierdoor gaten ontstaan in de record nummering en kan er niet gegarandeerd worden dat de records op chronologische volgorde staan. Dit is niet handig omdat ze voor hun boekhouding bepaalde nummeringen zoals die van facturen op chronologische volgorde moeten houden.

Voor Jonker zorgt de replicatiesoftware voor onnodig handmatig werk, dit komt omdat als bepaalde vestigingen van de klant een langere tijd offline werken er conflicten kunnen voorkomen bij het synchroniseren als ze daarna weer online komen. Deze conflicten moeten telkens handmatig door Jonker opgelost worden.

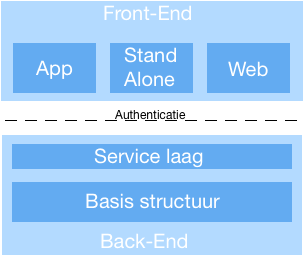
Ook loopt Jonker er tegenwoordig tegenaan dat er geen service laag bestaat in hun software. Er is namelijk veel vraag naar meerdere soorten clients, de klant zou graag bepaalde taken willen uitvoeren via web of app toepassingen en willen dit ook kunnen doen via een mobiel netwerk. Echter is het zonder service laag en een centraal toegangspunt niet rendabel om te kijken naar meerdere clients die eventueel zelfs op het mobiele netwerk kunnen functioneren.

Tegenwoordig is internet een stuk stabieler en met ondersteuning van bijvoorbeeld een 4G dongel is Jonker van mening dat er geen rekening meer hoeft gehouden worden met internetstoringen. Jonker vindt het nu een goed moment om te kijken naar een nieuwe gecentraliseerde opzet van hun administratieve software, waarmee ook bovengenoemde nadelen opgelost kunnen worden.

## Opdrachtformulering

De opdracht van Jonker bestaat uit het adviseren in het leggen van een nieuwe gecentraliseerde basis voor hun administratieve software. Jonker heeft op dit moment een gedecentraliseerd systeem waar de databases en servers op de verschillende vestigingen van de klant staan. Het onderhouden van de software en database gaat te omslachtig door een gedecentraliseerde database en door het ontbreken van een service laag wordt Jonker gelimiteerd met het uitbreiden naar meerdere clients. De nieuwe basis zal ervoor moeten zorgen dat deze problemen worden opgelost wordt en daardoor het gemak voor Jonker en hun klanten verbeterd zal gaan worden.

Aangezien de centralisering ervoor zorgt dat alle verschillende klanten van Jonker’s softwarepakket gaan verbinden met maar één systeem, komt er veel meer verkeer over de server en is daarom de snelheid van communicatie en het ophalen van data een belangrijk aspect bij het ontwerpen van de nieuwe basis. Het bedrijf wil graag geadviseerd worden wat de beste manier is om de basis te leggen. Dit moet ondersteund worden met onderzoeksresultaten en een proof of concept. De opdracht bestaat uit een aantal verschillende onderdelen die hieronder worden genoemd.



Figuur 2 geeft globaal weer hoe de softwarestructuur van de nieuwe basis er uiteindelijk uit moet gaan zien.

Voor de basisstructuur wil Jonker geadviseerd worden welke types databases geschikt zijn voor de data die ze verwerken. Hierbij dient er op de features, mogelijke serverrestricties, licentiekosten en replicatie/back-up functionaliteiten van de database worden gelet.

Figuur 2 – Softwarestructuur nieuwe basis

Verder is het gewenst voor Jonker geadviseerd te worden over de mogelijkheden van een persistence laag voor de gekozen database. Jonker wil graag weten hoe een aantal complexe query’s die ze nu gebruiken uitgevoerd zouden kunnen worden in de persistence frameworks die er op de markt zijn, ook willen ze graag weten wat de andere voor en nadelen zijn van het gebruik van deze frameworks.

Voor de service laag wil Jonker weten welke web service geschikt is in combinatie met hun software. De service laag moet te koppelen zijn met meerdere soorten clients, waaronder mobiele apps, standalone en web clients. Er zal onderzocht moeten worden welke communicatieprotocollen gebruikt kunnen worden en wat de voor en nadelen van elk zijn. Ook zal bekeken moeten worden hoe de business logic die nu nog in de clients zelf staat gescheiden kan worden in deze service laag en de clients. Uit elk van de bovenstaande adviezen wordt samen met Jonker een keuze gemaakt voor de nieuw te realiseren back-end.

De opdracht omvat naast het geven van advies ook het maken van een proof of concept waar één van de softwarepakketten van Jonker wordt omgezet zodat deze gebruik maakt van de nieuwe basis, hiervoor moet dus ook de back-end worden gerealiseerd. Er zal een test moeten worden opgesteld en uitgevoerd om aan te tonen dat het aangepaste pakket hetzelfde functioneert als het in de huidige omgeving doet. Als deze test goed doorlopen is, dan zal dat aantonen dat de nieuwe basis op andere pakketten van Jonker kan worden toegepast aangezien deze op dezelfde manier zijn opgezet.

## Doelstelling

In dit hoofdstuk wordt de doelstelling van het afstudeerproject en die van Jonker apart beschreven aangezien deze van elkaar verschillen.

### Doelstelling van het afstudeerproject

Jonker heeft een aantal limitaties door de verouderde structuur van hun administratieve software. Het doel van dit project is om Jonker inzicht te geven wat technieken die tegenwoordig op de markt zijn kunnen betekenen bij het vernieuwen van hun tClick software naar een gecentraliseerde omgeving.

Hiervoor moet worden onderzocht wat de beste keuze is in de te gebruiken databasetechnieken, welke persistence laag past bij de software, hoe een service laag kan worden opgezet en welke clients hiermee zouden kunnen verbinden.

De adviezen worden samen met Jonker besproken en terwijl het project vordert wordt er gekozen welke technieken gebruikt gaan worden in de nieuwe omgeving.

Er zal een proof of concept worden gemaakt door één van Jonker’s softwarepakketten om te zetten zodat deze werkt met de nieuwe technieken. Dit proof of concept zal dan kunnen worden getest op prestaties en functionaliteit.

### Doelstelling van Jonker

Jonker wil graag hun administratiesoftware gecentraliseerd opzetten om het zo meer toekomstgericht te maken, naar een hoger niveau te tillen en het mogelijk te maken de back-end te bereiken met een mobiel netwerk.

Ook kan Jonker met gecentraliseerde software een nieuw verdienmodel introduceren. Op dit moment wordt er geld verdient door het pakket te verkopen en de benodigde hardware bij de klant te plaatsen. Met een gecentraliseerde back-end kan de data en de service laag intern bij Jonker geplaatst worden en kunnen ze hun pakket aanbieden middels een leaseconstructie. Op deze manier is er meer controle over de veiligheid van data omdat Jonker zelf een goede back-up constructie kan maken.

## Eindproducten

Na het uitvoeren van het project zullen de volgende eindproducten opgeleverd worden. Hieronder worden deze producten benoemd.

|  |  |  |
| --- | --- | --- |
| Product | Oplevering | Criteria |
| Onderzoeksresultaten | In bijlage scriptie | Geeft antwoord op de vooraf opgestelde deelvragen |
| Proof of concept | Interne Apache Subversion repository locatie van Jonker | Maakt gebruik van de technieken gekozen door Jonker.  Heeft gelijke functionaliteiten als één van de modules in het tClick ADM pakket van Jonker. |
| Testrapport | Intern document Jonker | Geeft de resultaten weer van de functionaliteiten in de nieuwe en oude situatie van één van Jonker’s modules. |
| Scriptie | Hogeschool Utrecht Nijenoord 1 (3x hard copy, 1x DVD) | Voldoet aan de eisen gesteld in het afstudeerleidraad. |
| Presentatie | Wordt gegeven tijdens de afstudeerzitting in november | Voldoet aan de eisen gesteld in het afstudeerleidraad. |

# Onderzoeksopzet

In dit hoofdstuk worden de onderzoeksvragen opgenoemd die gelden bij dit onderzoek. Daarnaast wordt ook de opzet van het uitgevoerde onderzoek beschreven. Tot slot wordt per onderzoeksvraag een verslag gegeven hoe te werk is gegaan om de vraag te beantwoorden.

Voorafgaand aan dit onderzoek werd eerst een oriëntatiefase van het tClick ADM pakket ingepland, om zo beter zicht te krijgen over de problemen die Jonker ondervindt en een beter idee te krijgen wat precies de wensen van Jonker zijn

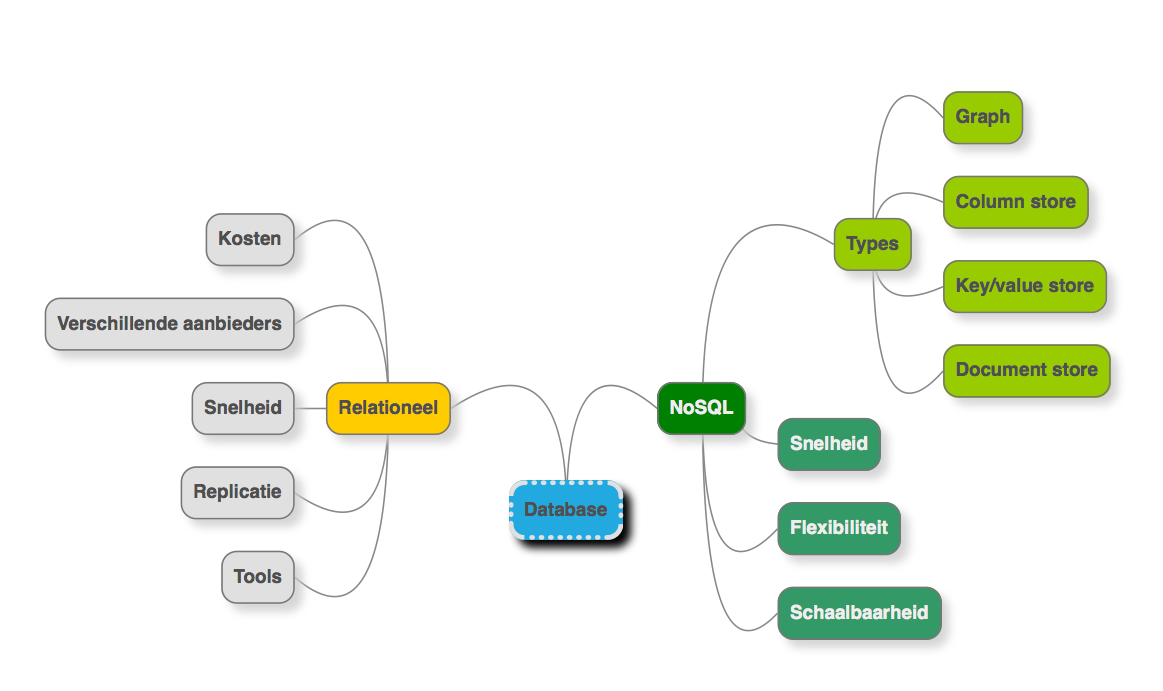
## Structuur/werkwijze van het onderzoek

Om de hoofdvraag en deelvragen te beantwoorden is veelal gebruik gemaakt van kwalitatief onderzoek. De keuze voor kwalitatief onderzoek was voor de hand liggend aangezien elke vraag in feite een ‘open’ vraag betreft. Volgens een artikel van (Shiny, 2013) is kwalitatief onderzoek hier de beste keuze.

Bij het doen van dit kwalitatief onderzoek is er gekozen voor een systematische aanpak om zo doelgericht informatie te vergaren over de onderwerpen en dus niet uit te wijken van het doel. Deze aanpak kan worden opgedeeld in verschillende fases, ze worden hieronder benoemd.

### Verkenningsfase

In de verkenningsfase is het de bedoeling een beter beeld te krijgen van het te onderzoeken onderwerp. Deze fase begint met het inlezen op internet, dit gebeurd door te zoeken naar relevante artikelen en deze te bestuderen. De focus ligt hier vooral op artikelen over aspecten die nog niet zijn aangeleerd binnen de studie.

Na het inlezen in het onderwerp wordt er een mindmap gemaakt met het onderwerp van de onderzoeksvraag. Volgens (ITASC, 2009) zorgt een mindmap namelijk voor het stimuleren van zowel de creatieve als de analytische helft van de hersenen. De mindmap begint met het onderwerp van de vraag in het midden en heeft vertakkingen naar de onderdelen die relevant zijn om te gaan onderzoeken. Nadat de mindmap is gemaakt zal deze gebruikt worden om een lijst van kernwoorden op te bouwen. Een voorbeeld van zo’n mindmap staat hieronder.

Figuur 3 – Mindmap met het onderwerp: “database”

### Verzamelfase

Na de verkenningsfase is er nu een lijst met kernwoorden. Deze worden gebruikt om zoektermen mee te creëren die dan weer worden gebruikt in online databanken zoals: <http://www.gartner.com/>, <http://dl.acm.org> en <https://scholar.google.nl/>, maar ook in reguliere zoekengines.

### Selectiefase

In de verzamelfase is grofweg zoveel mogelijk relevante bronnen verzameld. Om te voorkomen dat er te veel tijd wordt besteed aan het lezen hiervan zal er een kleinere selectie worden gemaakt. De bronnen in de bronnenlijst worden eerst ‘gescand’ door eerst de inleiding of abstract en de conclusie te lezen, als er bij meerdere artikelen dezelfde informatie wordt gegeven, zal de meest betrouwbare bron blijven bestaan in de bronnenlijst en de rest wordt niet meegenomen.

### Verwerkfase

Nadat alle relevante bronnen zijn verzameld wordt als eerst een inhoudsopgave gemaakt voor het onderzoeksrapport, de mindmap kan hierbij helpen om de hoofdstukken te identificeren.

De inhoud van de bronnen wordt bestudeerd, tevens wordt de benodigde informatie om de onderzoeksvraag te beantwoorden verwerkt in het onderzoeksrapport. Bij het verwerken van de informatie wordt er bekeken of er gerefereerd kan worden naar de activiteiten/problemen binnen Jonker die naar boven zijn gekomen in de oriëntatiefase om zo hetgeen extra te verduidelijken voor Jonker’s werknemers.

## Onderzoeksvragen

In het onderzoek bestaat er één hoofdvraag die is opgedeeld in kleinere deelvragen die erbij helpen de hoofdvraag te beantwoorden. De vraag die centraal staat in het onderzoek luidt: “Hoe kan de huidig gedecentraliseerde back-end van Jonker’s administratiesoftware omgezet worden naar een gecentraliseerde omgeving die ontsloten wordt naar verschillende clients door middel van een generieke service laag?”

Er zijn een aantal deelvragen opgesteld, de antwoorden op deze vragen zullen helpen bij het beantwoorden van de hoofdvraag. De deelvragen luiden als volgt:

* **Deelvraag 1** – Is de huidige database nog het meest geschikt voor een gecentraliseerde opzet van de administratiesoftware?
* **Deelvraag 2** – Op welke manieren kunnen bestaande persistence frameworks voordelen opleveren voor de nieuwe back-end?
* **Deelvraag 3** – Welke web services zijn er geschikt voor de nieuwe back-end en wat zijn de voor en nadelen van elk?
* **Deelvraag 4** – Welke business logica uit het softwarepakket wat omgezet gaat worden naar de nieuwe back-end kan worden verplaatst in de service laag.
* **Deelvraag 5** – Hoe kan er aangetoond worden dat het omgezette softwarepakket hetzelfde functioneert als voor de omzetting.

## Methode van onderzoek

Het beantwoorden van de deelvragen vergt voor elke vraag een andere aanpak. In dit hoofdstuk wordt per deelvraag uitgelegd welke aanpak gebruikt is.

### Deelvraag 1

(Is de huidige database nog het meest geschikt voor een gecentraliseerde opzet van de administratiesoftware?)

Om deelvraag 1 te beantwoorden is gebruik gemaakt van de werkwijze benoemd in [hoofdstuk 6.1](#_Structuur/werkwijze_van_het). Vooraf aan het onderzoek is eerst een oriëntatie geweest waar het huidige ADM pakket onder de loep werd genomen. Hieruit bleek dat de database van Jonker erg groot is (De database heeft namelijk 200+ tabellen). Ook is de data structureel niet makkelijk te begrijpen door het gebrek aan documentatie en foreign keys. Dit feit is besproken met de bedrijfsbegeleider en er is een voorstel gedaan om het onderzoek van deelvraag 1 abstract te houden van de huidige werkwijze van de database en juist te focussen op de algemene voor- en nadelen van de onderzochte databases. Dit voorstel is gedaan omdat het niet rendabel leek binnen de scope van het afstudeerproject alle ins en outs van de huidige database te bestuderen. De bedrijfsbegeleider deelde deze mening.

Na het inlezen in de verkenningsfase zijn de bevindingen besproken met de bedrijfsbegeleider en een collega. In dit gesprek is de gevonden term NoSQL veel besproken. Om een zo volledig mogelijk antwoord te kunnen geven op deelvraag 1 is besloten de vraag op te delen in twee sub vragen.

Deze luiden nu als volgt:

#### Deelvraag 1a: Wat zijn NoSQL databases en welke voordelen en/of nadelen hebben ze ten opzichte van relationele databases?

NoSQL is een verzamelnaam voor meerdere type databases die zich onderscheiden van de relationele database. Voor elk van deze type databases die Jonker interessant vond is bronnenstudie verricht om een goed beeld te vormen van de functionaliteiten en de voor en nadelen deze bieden. Dit heeft een duidelijk onderzoeksrapport opgeleverd welke Jonker meeneemt in de keuze voor de database. De onderzoeksresultaten zijn te vinden in [bijlage B](#BijlageB).

#### Deelvraag 1b: Wat zijn de voor en nadelen van de relationele databases die tegenwoordig op de markt zijn?

Voor deelvraag 1b is ervoor gekozen om een vergelijking te maken van de beschikbare aanbieders van relationele databases. Er is besproken met de bedrijfsbegeleider wat de problemen waren waar ze momenteel tegenaanlopen, en of er nog andere aspecten zijn die belangrijk zijn in de database voor de volgende versie van tClick ADM. Deze criteria aan de databases zijn meegenomen bij het onderzoek, om zo het aantal mogelijke databases te verkleinen en de duur van het onderzoek te beperken.

De resultaten van het onderzoek hebben een duidelijke vergelijking opgeleverd, en Jonker heeft hieruit een keuze kunnen maken. Het rapport is te vinden in [bijlage C](#BijlageC).

### Deelvraag 2

(Op welke manieren kunnen bestaande persistence frameworks voordelen opleveren voor de nieuwe back-end?)

Om deelvraag 2 te beantwoorden is bronnenonderzoek verricht om te achterhalen welke persistence frameworks er tegenwoordig op de markt zijn. Voor de persistence frameworks geldt de eis dat deze moet kunnen werken met een Java applicatieserver en de PostgreSQL database. De mindmap die gemaakt is in de verkenningsfase in het onderzoek bracht twee verschillende categorieën persistence frameworks ten sprake. Samen met de bedrijfsbegeleider en een andere collega is besproken het onderzoek verder te focussen op één categorie, de categorie welke het best paste bij de requirements.

Na het bronnenonderzoek te hebben voltooid is er tijd gemaakt om naast de theorie, ook de praktijk te testen. De bevindingen die tijdens het testen zijn gevonden zijn verwerkt in de onderzoeksresultaten. Uiteindelijk is er een duidelijk beeld ontstaan wat de verschillende persistence frameworks doen en wat de voor- en nadelen zijn wanneer deze gebruikt worden.

De resultaten zijn besproken met alle collega’s van de development afdeling, hiervoor is een meeting gepland zodat iedereen zijn mening kon delen. Hierbij is ook de eigen ervaring met de frameworks gedeeld. Unaniem is daarna een keuze gemaakt. De onderzoeksresultaten zijn te vinden in [bijlage D](#BijlageD).

### Deelvraag 3

(Welke web services zijn er geschikt voor de nieuwe back-end en wat zijn de voor en nadelen van elk?)

Voor deelvraag 3 is er wederom gebruik gemaakt van bronnenonderzoek om de verschillende mogelijkheden uit te zoeken. Na de verkenningsfase was er al duidelijk welke twee mogelijke web services gebruikt konden worden. Met de bedrijfsbegeleider is een gesprek ingepland waar van tevoren gevraagd is om specifieke criteria samen te stellen voor de nieuwe situatie. Na dit gesprek konden beide web services beoordeeld worden op de gestelde criteria, deze gegevens zijn verwerkt in het onderzoeksresultaat te vinden in [bijlage E](#BijlageE).

### Deelvraag 4

(Welke business logica uit het softwarepakket wat omgezet gaat worden naar de nieuwe back-end kan worden verplaatst in de service laag.)

Om deelvraag 4 te beantwoorden moest eerst de definitie van business logica duidelijk worden. Een korte verkenning op het internet heeft een omschrijving hiervan opgeleverd.

Hierna is met de bedrijfsbegeleider besproken welke module van het ADM pakket wordt omgezet voor het proof of concept. Deze module is geanalyseerd en met kennis al in het bezit, is de business logica gescheiden en geplaatst in de service laag.

### Deelvraag 5

(Hoe kan er aangetoond worden dat het omgezette softwarepakket hetzelfde functioneert als voor de omzetting.)

Van tevoren is gevraagd of de nieuwe situatie dezelfde functionaliteiten en belevenis zal hebben als in de huidige situatie. Om dit te testen is een functioneel acceptatietest opgezet, welke door collega’s kon worden uitgevoerd. Het test plan is te vinden in [bijlage F](#BijlageF).

# Oriëntatie huidige situatie

Voordat er met het onderzoek begonnen is, werd eerst de huidige omgeving van het ADM pakket geanalyseerd om zo de werking van de software in kaart te brengen. Dit hoofdstuk bevat de resultaten van deze analyse.

## De applicatie

De client bestaat uit een Java applicatie die samen met andere benodigde bestanden wordt geïnstalleerd bij de klanten. Bij het opstarten van de client wordt pas toegang gegeven aan het systeem na het invullen van een inlognaam en wachtwoord. De authenticatie hiervoor loopt via de database. Hier staan gebruikers gespecificeerd samen met het wachtwoord.

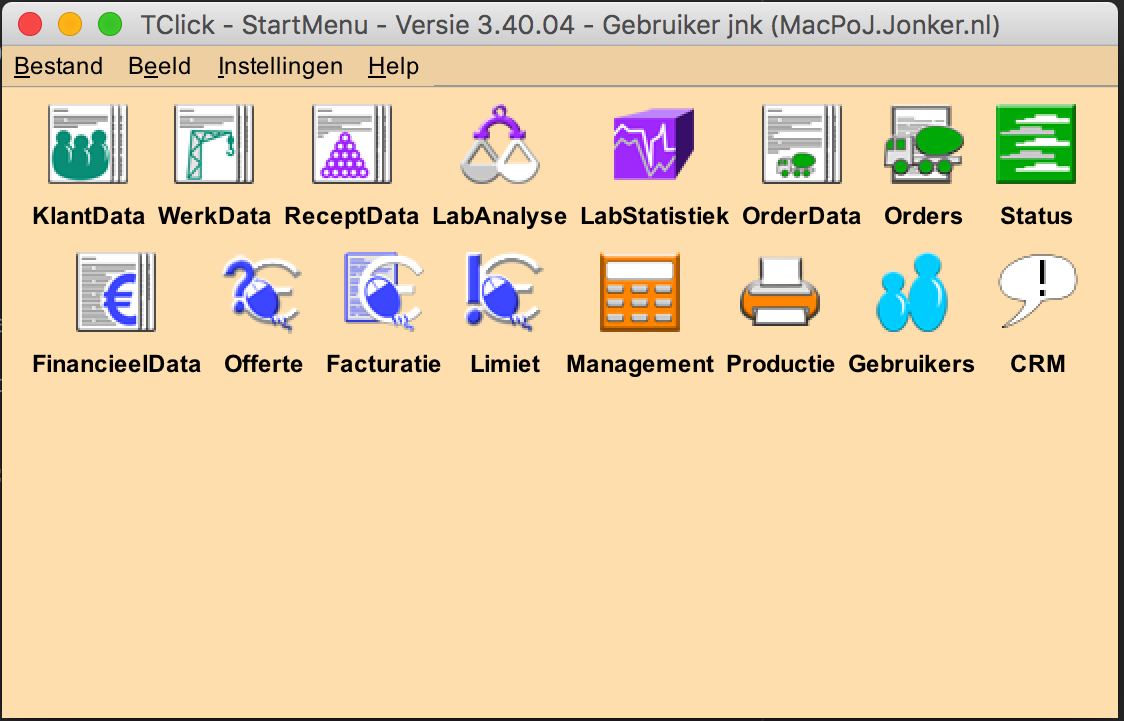
De grafische interface is gemaakt met de Swing API. Bij het ontwerpen van de schermen wordt alles op de pixel precies neergezet. De reden hiervoor is omdat de klanten vaak met kleine schermen werken op de betoncentrale, Jonker gebruikt graag zo veel mogelijk van de beschikbare ruimte.

Hieronder staat het inlogscherm.



Figuur 2 – Scherm inloggen

Nadat de gebruiker heeft ingelogd komt deze op het startmenu, een menu dat alle modules van het ADM pakket met elkaar verbindt. Vanuit hier kan de gebruiker elk gewenst pakket opstarten, deze komen dan in een nieuw venster. Een screenshot van het startmenu is hieronder te vinden.



Figuur 3 – Scherm startmenu

### Modules

Het ADM pakket van Jonker bestaat uit zestien verschillende modules. De modules bevatten elk een bepaalde taak in het bedrijfsproces van de klanten van Jonker. Zo is er onder andere een module voor het monitoren van trucks via live data van TomTom, een module om orders mee te beheren en een module om de productie in te zien.

Binnen de tijd van dit onderzoek is het niet rendabel om naar het hele software pakket te kijken, er is samen met Jonker gekozen om de scope van het onderzoek te beperken naar één module. De productie module, de keuze is gebaseerd op dat de betreffende module het meest opzichzelfstaand is. Deze module werkt zonder dat er data nodig is vanuit andere modules.

## De database

Binnen de applicatie wordt data persistent opgeslagen in een database, hiervoor wordt gebruik gemaakt van de Firebird database. In dit hoofdstuk worden specifieke onderdelen van de database verder beschreven.

### Eerste blik

Wat als eerst opvalt als de database bekeken wordt met een database manager, is dat er een flink aantal tabellen zijn. Om te kijken hoeveel dit er nou precies zijn is de volgende query gebruikt:

select count(rdb$relation\_name), avg(rdb$field\_id)

from rdb$relations

where rdb$view\_blr is null

and (rdb$system\_flag is null or rdb$system\_flag = 0);

Het resultaat van deze query geeft een count(optelling) van alle tabellen en een average(gemiddelde) van het aantal kolommen. Hieruit blijkt dat de database 244 verschillende tabellen heeft met gemiddeld 21 kolommen per tabel.

### Foreign keys

Wat ook opvalt is dat er veelal gewerkt wordt met relationele data, terwijl tabellen niet gekoppeld zijn d.m.v. foreign keys. Het gebrek aan foreign keys komt volgens de bedrijfsbegeleider door de replicatiesoftware. Deze zorgt er voor dat bij klanten met meerdere betoncentrales, alle databases met elkaar worden gesynchroniseerd. Dit gebeurt echter op een manier dat deze de regels van een foreign key kan overschrijden.

Door het gebrek aan foreign keys kan de database alleen in kaart worden gebracht door de relaties tussen tabellen op te zoeken in de sourcecode van de client applicatie.

### Verbinding met client applicatie

Binnen de applicatie wordt er gebruik gemaakt van de Jaybird JDBC driver om SQL query’s uit te voeren op de database. De query’s zijn gescheiden van de rest door gebruik te maken van Java klassen die enkel database functionaliteit bevatten.

# Onderzoeksresultaten

Het doel van het onderzoek was uiteindelijk het beantwoorden van de deelvragen, dit is gelukt. Chronologisch wordt in dit hoofdstuk wordt per onderzoeksvraag de onderzoeksresultaten kort samengevat (de uitgebreide versies zijn te vinden in de bijlages), ook wordt het antwoord op de vraag geformuleerd.

## Deelvraag 1a: NoSQL databases

Zoals verantwoordt in de onderzoeksopzet is deelvraag 1 opgedeeld in twee sub vragen. Voor de eerste sub vraag zijn verschillende NoSQL databases onderzocht. Er is per NoSQL database type onderzoek gedaan naar de werking en naar en welke voor- en nadelen dit biedt ten opzichte van relationele databases. Een samenvatting van de onderzoeksresultaten voor deze deelvraag is hieronder te vinden.

### Algemene voor- en nadelen

NoSQL databases blijken elk op hun eigen manier te verschillen van relationele databases. Toch zijn er een aantal gezamenlijke verschillen voor alle types NoSQL databases. De reden dat er een overkoepelend verschil is lijkt te komen door de reden waarom NoSQL databases in de wereld zijn gebracht, namelijk door de gebreken van een relationele database.

De algemene voordelen zijn hieronder opgenoemd.

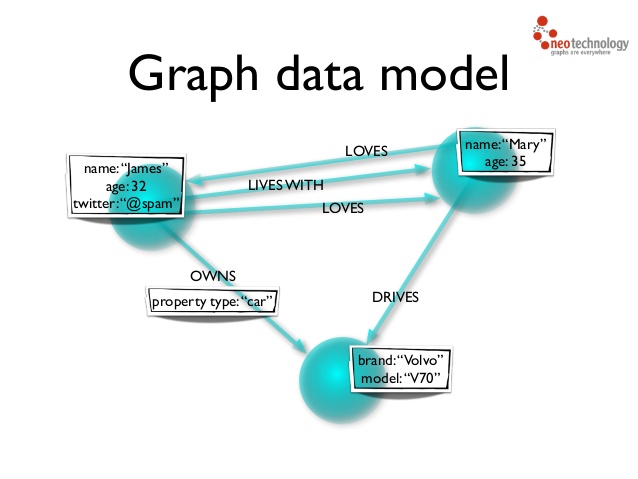
* Omdat de structuur van NoSQL databases niet relationeel is, zijn deze daardoor vaak beter te schalen.
* Een verbeterde snelheid in de juiste usecases.
* Flexibel: de structuur van de data hoeft meestal niet van tevoren vast te worden gelegd.
* Doordat NoSQL databases zich focussen op één bepaald opslagprobleem, zijn ze vaak simpeler in gebruik.

Ook zijn er een aantal verschillen die nadelig zijn t.o.v. relationele databases. Deze zijn hieronder opgenoemd

* Juist omdat NoSQL databases zich focussen op één bepaald opslagprobleem, zorgt dit er ook voor dat er meerdere databasen door elkaar gebruikt moet worden in een complexe omgeving.
* Er zijn verschillende type NoSQL database, en ook binnen elk type worden er geen standaarden geïmplementeerd. Het overstappen naar een andere database is daarom niet zo gemakkelijk als nu wordt gedaan met relationele databases

### Graph databases

De graph database is een NoSQL database opgebouwd uit nodes, relaties en properties. Een node is vergelijkbaar met een entiteit, de nodes worden met elkaar verbonden door relaties. Beide de nodes en relaties kunnen properties (eigenschappen) hebben. Dit alles gecombineerd zorgt voor een groot web van data. Een voorbeeld van een zeer kleine graph database is hieronder te zien.



Figuur 4 - Voorbeeld graph database

Een query in een graph database zoekt altijd eerst naar een node, of meerdere nodes om de query vanaf te starten: de start-node(s). Vanaf hier kunnen de relaties van deze node(s) gevolgd worden, waarna ook de relaties van de gerelateerde nodes gevolgd kunnen worden. Uiteindelijk komt de query uit bij een end-node, of meerdere end-nodes.

Er kan bijvoorbeeld een query worden uitgevoerd welke alle Film nodes ophaalt waarin een Acteur node heeft gespeeld. Dit gebeurd in één database actie, terwijl een relationele database hier drie acties voor nodig heeft omdat deze eerst de gerelateerde tabellen moet joinen. Een complete beschrijving staat in de uitgewerkte onderzoeksresultaten, te vinden in de bijlages.

### Key-value stores

De key-value store is de meest simpele NoSQL database, ook een van de snelste. De database bestaat enkel uit een verzameling van unieke keys met bijhorende values, te vergelijken met een Java HashMap.

Het ophalen van waardes uit de key-value store duurt altijd even lang, of eigenlijk even kort. De database doet dit namelijk erg snel.

De waardes die de key-value store kan opslaan verschilt per aanbieder. Zo kan de ene alleen strings opslaan en heeft de ander meerdere formaten zoals arrays of lijsten. Voor het updaten van waardes worden vaak passende methoden aangeboden in de API. Zo hebben lijsten functies zoals pop en push, en hebben arrays de mogelijkheden om velden in de array apart te wijzigen.

### Document stores

Een document store slaat data op in documents, een datastructuur dat vergelijkbaar is met het populaire JSON formaat. Een document bestaat uit een collectie van key-value paren waar de value van de key mogelijk weer kan bestaan uit een ander document. Dit betekend dat documents hiërarchisch in elkaar kunnen worden geplaatst. De root (dit is het document dat niet in een ander document is geplaatst) moet wel altijd uniek identificeerbaar zijn door middel van een key. In het figuur hieronder staat een visuele weergave van de datastructuur van een document.

In grote delen lijkt de opslag van een document store op dat van een relationele database.

* Documents wordt opgeslagen in een collection, wat in feite hetzelfde is als een tabel in een relationele database.
* Als een collection een tabel is, dan is de document dus eigenlijk een “rij”
* Een document heeft meerdere velden, deze lijken dan weer erg op de kolommen

Het grote verschil is dat documents hiërarchisch in elkaar kunnen worden geplaatst. Rijen in een tabel kunnen dat niet. Het relationele model heeft echter wel een functie die hier erg op lijkt: de join. Een hiërarchisch document kan gezien worden als een vooraf gejoinde tabel.

### Conclusie

In het onderzoek zijn de verschillende types NoSQL databases in kaart gebracht. Het relativeren van deze resultaten aan de huidige database valt niet binnen de scope van het project, de database was namelijk veel te groot daarvoor. Daarom heeft Jonker de onderzoeksresultaten zelf meegenomen in de uiteindelijk keuze. Deze keuze wordt in deelvraag 1b genoemd.

## Deelvraag 1b: Relationele databases

Een belangrijke database requirement voor Jonker is dat er genoeg tools beschikbaar zijn. De mate van populariteit werd voor het onderzoek gebruikt om vooraf in te schatten bij welke databases dit het geval is. Hierbij is ervanuit gegaan dat een grote community ook betekend dat er in ieder geval voldoende tools zijn.

De populairste databaseaanbieder zijn bepaald door gebruik te maken van de website <http://db-engines.com/>, dit is een website die elke databaseaanbieder een score geeft op basis van populariteit. De scores worden opgebouwd door middel van een algoritme dat gebruik maakt van informatie die publiekelijk te vinden is in social-media, zoekengines, vacaturesites en bekende vraag & antwoord sites zoals Stack Overflow en DBA Stack Exchange.

De databaseaanbieders bieden soms meerdere varianten aan, indien meerdere varianten interessant leken werden deze meegenomen in het onderzoek. Daarnaast werd voor Jonker ook de huidige database meegenomen. De volgende databases zijn onderzocht:

* Oracle Express Edition / Standard Edition One
* MySQL Community Edition / Standard Edition
* PostgreSQL
* DB2 Express-C / Express
* Firebird

De volgende aspecten van de databases zijn opgenomen in het onderzoek:

* Licentie kosten
* Server restricties
* Replicatie mogelijkheden
* Uitgelichte features

De aspecten zijn in het onderzoek zelf uitgebreid uitgewerkt. Hieronder staan de resultaten verwerkt in een vergelijkingstabel. Een rode kleur betekent een slechte beoordeling, oranje betekent gemiddeld en groen betekent dat een goede beoordeling.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Database | Kosten | Restricties | Replicatie | Featureset |
| Oracle XE |  |  |  |  |
| Oracle SE1 |  |  |  |  |
| MySQL CE |  |  |  |  |
| MySQL SE |  |  |  |  |
| PostgreSQL |  |  |  |  |
| DB2 Ex-C |  |  |  |  |
| DB2 Express |  |  |  |  |
| Firebird |  |  |  |  |

Tabel 1 - Relationele databases vergelijkingstabel

### Conclusie

Nadat de onderzoeksresultaten van deelvraag 1a en 1b zijn besproken, heeft Jonker een antwoord kunnen geven op deelvraag 1. De vraag luidde: Is de huidige database nog het meest geschikt voor een gecentraliseerde opzet van de administratiesoftware?

Het antwoord op deze vraag is dat de huidige database niet meer de meeste geschikte is voor de nieuwe opzet van de administratiesoftware. De PostgreSQL database kwam namelijk als beste uit de test bij de onderzochte databases.

Ook een keuze voor een NoSQL database leek Jonker niet de beste, het huidige software pakket is namelijk te complex. Er zou meer als één database gebruikt moeten worden.

## Deelvraag 2: Persistence framework

In de nieuwe situatie voor Jonker gaat een Java applicatieserver verbinden en communiceren met een PostgreSQL database.

Java biedt de mogelijkheid de JDBC API te gebruiken om te werken met de database. Het gebruiken van enkel JDBC komt gepaard met een aantal nadelen. Zo kunnen complexe query’s snel onoverzichtelijk doordat Java Strings conditioneel aan elkaar geplakt moeten worden. Een voorbeeld van zo’n onoverzichtelijke query is hieronder te zien.

String query = "select count (\*) from OfferteDetail B " +

"LEFT OUTER JOIN OfferteCentrale OC ON " +

"OC.OfferteCentralenr = B.Centralenr " +

"LEFT OUTER JOIN Centrale C ON " +

"C.Centralenr = OC.HoofdCentralenr " +

"WHERE C.KlantGroepnr=" + beheer.getCentrale().getKlantGroepnr() +

" AND B.KoppelKlantnr=" + klantnr +

" AND B.KoppelReceptnr=" + itemnr;

Daarnaast is het werken met de JDBC ResultSet eindeloos code kloppen. Onderstaand voorbeeld geeft een tabel aan met 3 kolommen, maar stel eens een tabel voor met 100 kolommen.

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery("SELECT \* FROM AUTHOR");

while (rs.next())

{

    Integer id = rs.getInt(“id”);

String firstName = rs.getString(“firstname”);

String lastName = r.getString(“lastname”);

}

rs.close();

st.close();

Er zijn een aantal software oplossingen beschikbaar met als doel ervoor te willen zorgen dat het koppelen en gebruiken van een database binnen Java software makkelijker wordt. Jonker wenste veel controle te blijven houden over de uitgevoerde SQL. Frameworks zoals Hibernate waren hierdoor geen optie.

De twee meest transparante frameworks zijn onderzocht: JOOQ en MyBatis.

Beide frameworks zorgen er op hun eigen manier voor dat het maken van SQL query’s niet meer vooral bestaat uit het aan elkaar plakken van strings, ook zorgen ze ervoor dat de resultaten die terug komen van de database handig verwerkt wordt. Hieronder worden de voor en nadelen op een rijtje gezet.

### Voordelen van JOOQ

JOOQ helpt je met het bouwen van SQL query’s door de tekst gebaseerde SQL query’s om te zetten naar query’s opgebouwd in een Java syntax. Samen met de code generator zorgt dit voor een aantal handige features. Fouten in SQL worden getoond voordat de code zelfs wordt uitgevoerd, en door auto-complete op tabelnamen, kolomnamen en SQL functies kunnen query’s sneller worden gemaakt.

Het verwerken van resultaten gaat simpel, de API biedt hiervoor verschillende methoden die de resultaten direct om kan zetten naar Java objecten of web service resultaten zoals JSON.

Daarnaast zorgt JOOQ ervoor dat er niet meer op verschillende SQL dialecten van databases hoeft worden gelet, de vertaalstap wordt namelijk geautomatiseerd in JOOQ. Het overstappen naar een andere database in de toekomst zal daardoor een kleine moeite worden.

### Nadelen van JOOQ

Alhoewel de DSL van JOOQ erg lijkt op SQL, moet er toch een nieuwe syntax geleerd worden om SQL query’s te schrijven.

JOOQ biedt de mogelijkheid tot vele geavanceerde SQL functionaliteiten zoals nested selects, aliasing, unions, self-joins. Voor het gros van de developers is dit ruim genoeg. Het is echter wel mogelijk dat een database nieuwe functionaliteiten introduceert, deze zijn dan niet meteen beschikbaar in JOOQ. Hiervoor zou wel zelf een uitbreiding in JOOQ kunnen worden gemaakt.

### Voordelen van MyBatis

MyBatis zorgt voor een nieuwe manier van SQL query’s maken in Java. In een speciale XML syntax kunnen tekst gebaseerde query’s worden gemaakt en met enkele toevoegingen kunnen deze aangepast worden op de context met bijvoorbeeld parameters, IF statements en FOR loops.

Het plaatsen van query’s in aparte XML bestanden is ook bevorderlijk voor de herbruikbaarheid van de query’s.

Daarnaast is het relatief simpel om resultaten om te zetten naar Java objecten, dit kunnen standaard Java types zijn, maar er kunnen ook geavanceerde Java objecten gemaakt worden die je zelf kan specificeren. Deze kunnen gebruikt worden in de Java code, maar ook bijvoorbeeld geparsed worden door een JSON library om er web service resultaten van te maken.

### Nadelen van MyBatis

Er zijn een aantal problemen die MyBatis nog niet oplost, zo kunnen er nog syntax fouten gemaakt worden in SQL die pas tijdens het uitvoeren van code naar boven komt, alhoewel dit wel minder vaak zal gebeuren door de betere overzicht die de XML notatie biedt ten opzichte van query Strings in Java.

Ook is de SQL nog steeds dialect gevoelig, wil je wisselen van database zal je zelf de verschillen in SQL moeten herkennen en deze moeten aanpassen in je code.

### Conclusie

Jonker wilt graag het gemak van een persistence framework gebruiken, dit bevordert namelijk de snelheid waarmee ontwikkeld kan worden in de nieuwe versie van het software pakket. In het huidige pakket wordt veel gebruik gemaakt van query’s die conditioneel wijzigen, dit willen ze blijven doen. Daarnaast wenst Jonker het uiterste te kunnen halen uit de database die ze gebruiken.

De voordelen van MyBatis passen beter bij de wensen die Jonker heeft. Met de dynamische XML query’s kunnen query’s makkelijk conditioneel en toch overzichtelijk aangepast worden. Daarnaast genereert MyBatis de SQL niet, maar zorgt alleen voor gemak, hierdoor heeft Jonker zelf meer controle over wat er met database gedaan wordt.

Deelvraag 2 luidde: Op welke manieren kunnen bestaande persistence frameworks voordelen opleveren voor de nieuwe back-end?

De onderzoeksresultaten geven een duidelijk beeld welke voor- en nadelen bij de beschikbare persistence frameworks horen. De uitgewerkte onderzoeksresultaten in de bijlage zorgen voor extra verduidelijking hoe de voor- en nadelen zijn gevonden.

## Deelvraag 3: Web service communicatie

De web service die voor Jonker moet worden opgezet zal door meerdere clients bereikt moeten kunnen worden. Daarom wil Jonker graag gebruik maken van een communicatiemethode dat door veel platformen gebruikt kan worden.

SOAP en REST zijn hiervoor de meest gebruikelijke opties. Hieronder wordt een korte uitleg gegeven over SOAP en REST.

### Wat is SOAP?

SOAP is een open standaard communicatieprotocol voor web services en wordt mede daardoor door veel programmeertalen geïmplementeerd. SOAP werkt door berichten te versturen en ontvangen in een template op basis van XML. De template is hieronder te zien.

<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<MethodeNaam>

<ParameterNaam> data </ParameterNaam>

</MethodeNaam>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

SOAP berichten kunnen op elk internetprotocol worden verstuurd. Bijvoorbeeld via het HTTP, HTTPS, SMTP, JMS of FTP protocol.

### Wat is REST?

REST is geen op zichzelf staand protocol, maar een vastgelegde manier voor communiceren met een web service door middel van het HTTP protocol. De verschillende functionaliteiten van het HTTP protocol worden gebruikt om data te versturen, zo worden de methodes POST, GET, PUT en DELETE gebruikt om de soort methode aan te geven en worden parameters doorgegeven via de URI of via de post parameters van HTTP. Hieronder is een voorbeeld te zien van een REST methode aanroep.

http://www.host.nl/service/methodeNaam?param=data

### Verschillen SOAP en REST

SOAP en REST zijn op zichzelf staande manieren van communiceren en doen dit ook allebei anders. De volgende aspecten zijn onderzocht voor beide web services.

**Snelheid**

Hoe snel een web service reageert is vooral afhankelijk van de tijd dat het kost om de data te versturen. Per communicatiemethode wordt overzichtelijk gemaakt hoe de data verpakt wordt voordat het over het netwerk gestuurd wordt.

**API**

De API is het aanspreekpunt van de web service. De verschillende communicatiemethoden hebben hun eigen manier van het aanbieden van deze API. Hoe dit gebeurd wordt uitgelegd.

**Betrouwbaarheid**

Een web service is te bereiken via het internet, de communicatie van en naar de web service via een netwerk is niet altijd even stabiel. Er wordt bekeken welke mogelijkheden de communicatiemethoden bieden in verband met het oplossen van onstabiliteit van internet.

### Wanneer SOAP niet gebruiken?

SOAP gebruikt relatief veel netwerk bandbreedte, in het algemeen is dit te verwaarlozen als er gebruik gemaakt wordt van redelijke WIFI verbindingen. Gebruik je een mobiele of een trage WIFI verbindingen? dan is het vaak beter een REST web service te gebruiken.

Daarnaast is het verbinden met een SOAP web service niet mogelijk in alle programmeertalen, hier zijn namelijk extra libraries voor nodig. Mede hierdoor is SOAP geen populaire keuze als er web en/of mobile clients worden gebruikt.

### Wanneer SOAP wel gebruiken?

SOAP een opzichzelfstaand protocol, hierdoor is communicatie niet gelimiteerd via HTTP en kunnen er ook andere manieren worden gebruikt. REST kan dit niet, wil je geen HTTP gebruiken? Dan is SOAP je keuze.

Een andere reden kan zijn omdat SOAP een contract heeft tussen client en server. Alle methoden, objecten en datatypes zijn voor het gebruiken al bekend bij beide partijen; de client en de server. Dit opent mogelijkheid tot handigheden zoals auto-complete en het omzetten van complexe datetypen.

Daarnaast kan een SOAP service stateful worden gemaakt, wat voor sommige usecases essentieel is. In een stateful service kunnen requests die gedaan worden afhankelijk zijn van requests die eerder zijn uitgevoerd door dezelfde gebruiker, denk hierbij aan bijvoorbeeld een winkelwagen in een web shop waar in meerdere stappen producten worden toegevoegd.

### Wanneer REST niet gebruiken?

REST zou niet gebruikt moeten worden voor een stateful web service, hiervoor moet de server namelijk een sessie bijhouden van elke gebruiker. De sessie van een gebruiker kan echter wel bijgehouden worden op de client zelf. Waarna deze sessie naar de web service wordt gecommuniceerd waar nodig.

### Wanneer REST wel gebruiken?

REST is een lichtgewicht web service. Het is simpel te koppelen omdat enkel een URI en standaard HTTP functionaliteiten nodig zijn. Daardoor is REST geschikt voor zowel web, mobile als desktop clients.

Ook de flexibiliteit van REST kan soms voordelen opleveren, zo kan elke client (indien de developer van de service dit toelaat) zelf aangeven in welk mediatype ze de resultaten van een service willen ontvangen.

### Conclusie

Beide onderdelen kunnen gebruikt worden voor de gecentraliseerde omgeving van Jonker. REST heeft het voordeel dat er minder data wordt verstuurd over het netwerk, en dus sneller is. SOAP heeft het voordeel dat de API makkelijker kan worden gecommuniceerd en er een contract bestaat tussen server en client.

Deelvraag 3 luidde: Welke web services zijn er geschikt voor de nieuwe back-end en wat zijn de voor en nadelen van elk?

De onderzoeksresultaten leveren hier een goed inzicht waardoor de vraag kan worden beantwoord. De voor- en nadelen van elke methode is bekend. Jonker heeft de keuze gemaakt voor een REST service, omdat ze de snelheid een belangrijk aspect vinden.

## Deelvraag 4: Business logica

Deelvraag 4 luidde: Welke business logica uit het softwarepakket wat omgezet gaat worden naar de nieuwe back-end kan worden verplaatst in de service laag.

Met kennis van de afstudeerder is het huidige pakket geanalyseerd en zijn er usecases beschreven. Deze zijn samen met de bedrijfsbegeleider besproken en na toevoeging van de laatste usecase ook goedgekeurd.

De usecases zijn te vinden in de [functionele specificaties](#_Functionele_specificaties_1) van het proof of concept.

## Deelvraag 5: Test

Deelvraag 5 luidde: Hoe kan er aangetoond worden dat het omgezette softwarepakket hetzelfde functioneert als voor de omzetting.

Om de werking van het omgezette softwarepakket te testen moest er gekeken worden of dezelfde functionaliteiten aanwezig zijn als in het huidige pakket. Aan de hand van de al gemaakte usecases is een acceptatietest opgesteld. De test controleert of het omgezette softwarepakket de usecases naar verwachting afhandelt.

De acceptatietest is uitgevoerd door meerdere collega’s en is telkens met succes voltooid.

# Proof of Concept

In dit hoofdstuk komt de aanpak en de implementatie van het proof of concept aan bod. Het proof of concept maakt gebruik van alle technieken die Jonker heeft gekozen aan de hand van de onderzoeksresultaten.

## Voorbereiding

Om een duidelijk beeld te krijgen wat de functionaliteiten zijn van de productie module is er eerst kort geëxperimenteerd met de mogelijkheden van de module. Tegelijk is er zo snel mogelijk een afspraak gemaakt met de bedrijfsbegeleider om de bevindingen te kunnen bespreken en verifiëren.

## Functionele specificaties

Nadat de werking en het nut van de productie module helder was, is er vastgelegd welke functionaliteiten zullen worden geïmplementeerd in het proof of concept. Dit werd gedaan in de vorm van usecases, deze staan hieronder. De inhoud van de usecases zijn functionaliteiten overgenomen van en/of verbeterd ten opzichte van de productie module.

Tabel 2 - Usecase 1: Een overzicht maken van producties ingevoerd op huidige dag

|  |  |
| --- | --- |
| Nr: | 1 |
| Versie nr: | 1.0 |
| Auteur: | Jochem Pouwels |
| Usecase: | Een overzicht maken van producties ingevoerd op huidige dag |
| Actoren: | Gebruiker |
| Samenvatting: | Er wordt een productieoverzicht (figuur 5) getoond van alle producties die vandaag ingevoerd zijn |
| Prioriteit: | Must |
| Pre conditie |  |
| Hoofdscenario | * 1. Actor start de productiemodule   2.1 Systeem haalt producties op van huidige dag  2.2 Systeem toont producties van huidige dag |
| Post conditie | Producties van huidige dag worden op het scherm getoond |

Tabel 3 - Usecase 2: Een overzicht maken van producties ingevoerd op een geselecteerde dag

|  |  |
| --- | --- |
| Nr: | 2 |
| Versie nr: | 1.0 |
| Auteur: | Jochem Pouwels |
| Usecase: | Een overzicht maken van producties ingevoerd op een geselecteerde dag |
| Actoren: | Gebruiker |
| Samenvatting: | Er wordt een productieoverzicht getoond van alle producties op een door de gebruiker geselecteerde dag |
| Prioriteit: | Must |
| Pre conditie | Actor heeft productiemodule geopend. |
| Hoofdscenario | * 1. Actor klikt op de “kalender” knop   2.1 Systeem toont kalender invoerscherm  3.1 Actor selecteert dag, maand en jaar  4.1 Systeem haalt producties op van geselecteerde dag  4.2 Systeem toont producties van geselecteerde dag |
| Post conditie | Producties van geselecteerde dag worden op het scherm getoond |
| Alternatieve scenario 1: | [ Volgende dag knop ]   * 1. Actor klikt op de “volgende dag” knop   2.1 Systeem selecteert de dag na de al geselecteerde dag  2.2 Systeem gaat verder vanaf HS 4.1 |
| Post conditie: AS1 | Producties van de dag na de al geselecteerde dag worden op het scherm getoond |
| Alternatieve scenario 2: | [ Vorige dag knop ]   * 1. Actor klikt op de “vorige dag” knop   2.1 Systeem selecteert de dag voor de al geselecteerde dag  2.2 Systeem gaat verder vanaf HS 4.1 |
| Post conditie: AS2 | Producties van de dag voor de al geselecteerde dag worden op het scherm getoond |

Tabel 4 - Usecase 3: Een overzicht maken van producties die overeenkomen met de criteria van een filter

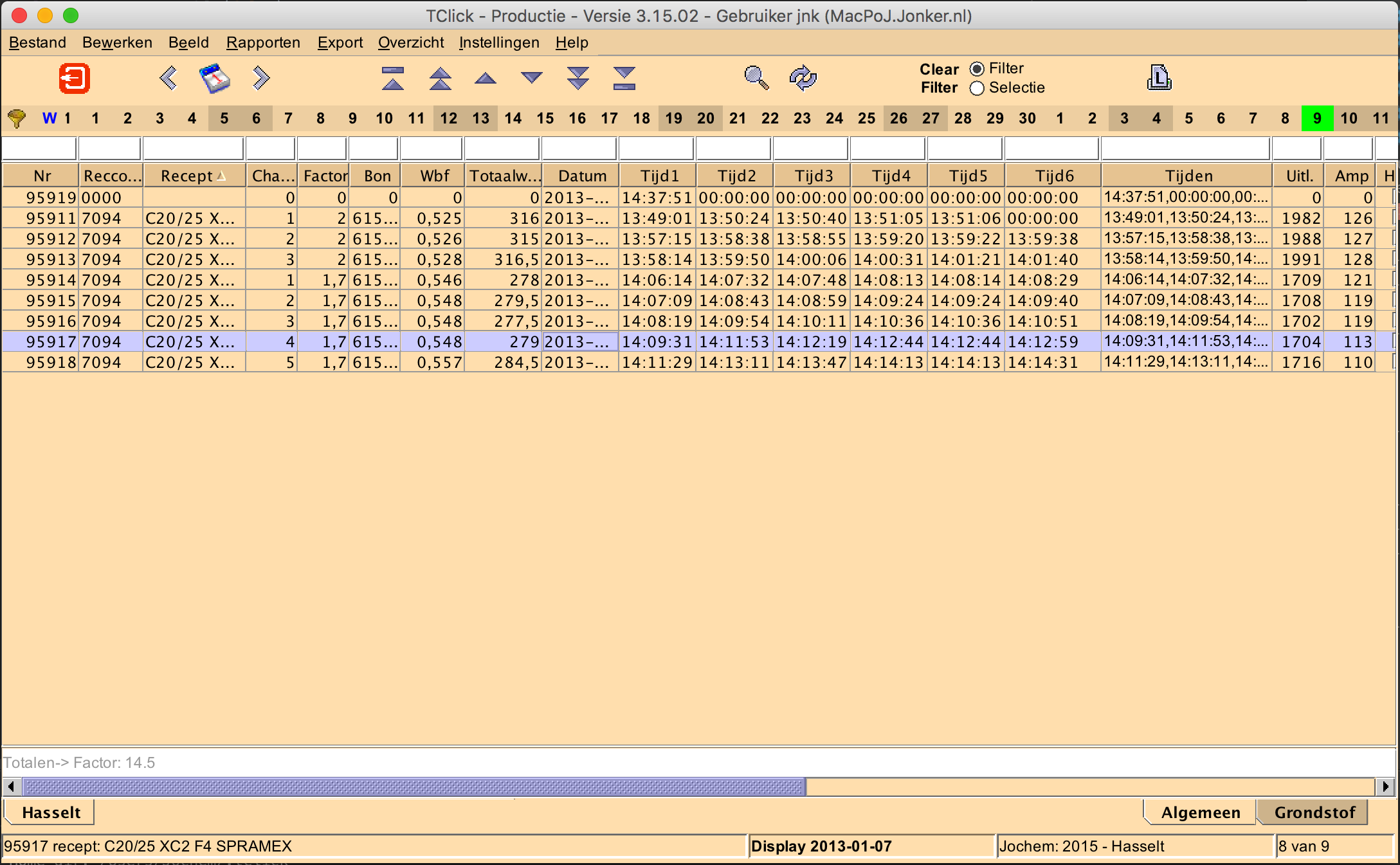
|  |  |
| --- | --- |
| Nr: | 3 |
| Versie nr: | 1.0 |
| Auteur: | Jochem Pouwels |
| Usecase: | Een overzicht maken van producties die overeenkomen met de criteria van een filter |
| Actoren: | Gebruiker |
| Samenvatting: | Er wordt een productieoverzicht getoond van alle producties die voldoen aan de criteria van het aangegeven filter |
| Prioriteit: | Should |
| Pre conditie | Actor heeft productiemodule geopend |
| Hoofdscenario | * 1. Actor opent de filter popup   2.1 Systeem toont de filter popup (figuur 7)  2.2 Systeem haalt alle klanten en recepten op  2.3 Systeem toont klanten in een selectielijst bij de criteria  2.4 Systeem toont recepten in een picklist bij de criteria  3.1 Actor vult criteria in voor periode van het overzicht  3.2 Actor vult criteria in voor het matchen van een receptcode  3.3 Actor vult criteria in voor het matchen van een receptnaam  3.4 Actor vult criteria in voor het matchen van bestemming   * 1. Actor vult criteria in voor het matchen op klantnaam   2. Actor vult criteria in voor het matchen op recepten   3. Actor klikt op de “aan” knop   4.1 Systeem slaat filtercriteria op  4.2 Systeem sluit filter popup  4.3 Systeem haalt producties op basis van de filtercriteria  4.4 Systeem toont producties op basis van de filtercriteria |
| Post conditie | Filtercriteria opgeslagen en producties die overeenkomen met de criteria van het filter worden op het scherm getoond |
| Alternatief scenario 1: | [ Actor zet filter uit ]   * 1. Actor klikt op de “uit” knop   4.1 Systeem slaat filtercriteria op  4.2 Systeem sluit filter popup  4.3 Systeem haalt producties op van huidige of geselecteerde dag indien deze nog niet opgehaald zijn  4.4 Systeem toont producties op van huidige of geselecteerde dag |
| Post conditie AS1 | Filtercriteria opgeslagen en producties van geselecteerde dag worden getoond |
| Alternatief scenario 2: | [ Actor annuleert de filtercriteria ]   * 1. Actor klikt op de “annuleer” knop   4.1 Systeem sluit filter popup |
| Post conditie AS2 | Geen |

Tabel 5 - Usecase 4: Extra informatie tonen van een productie

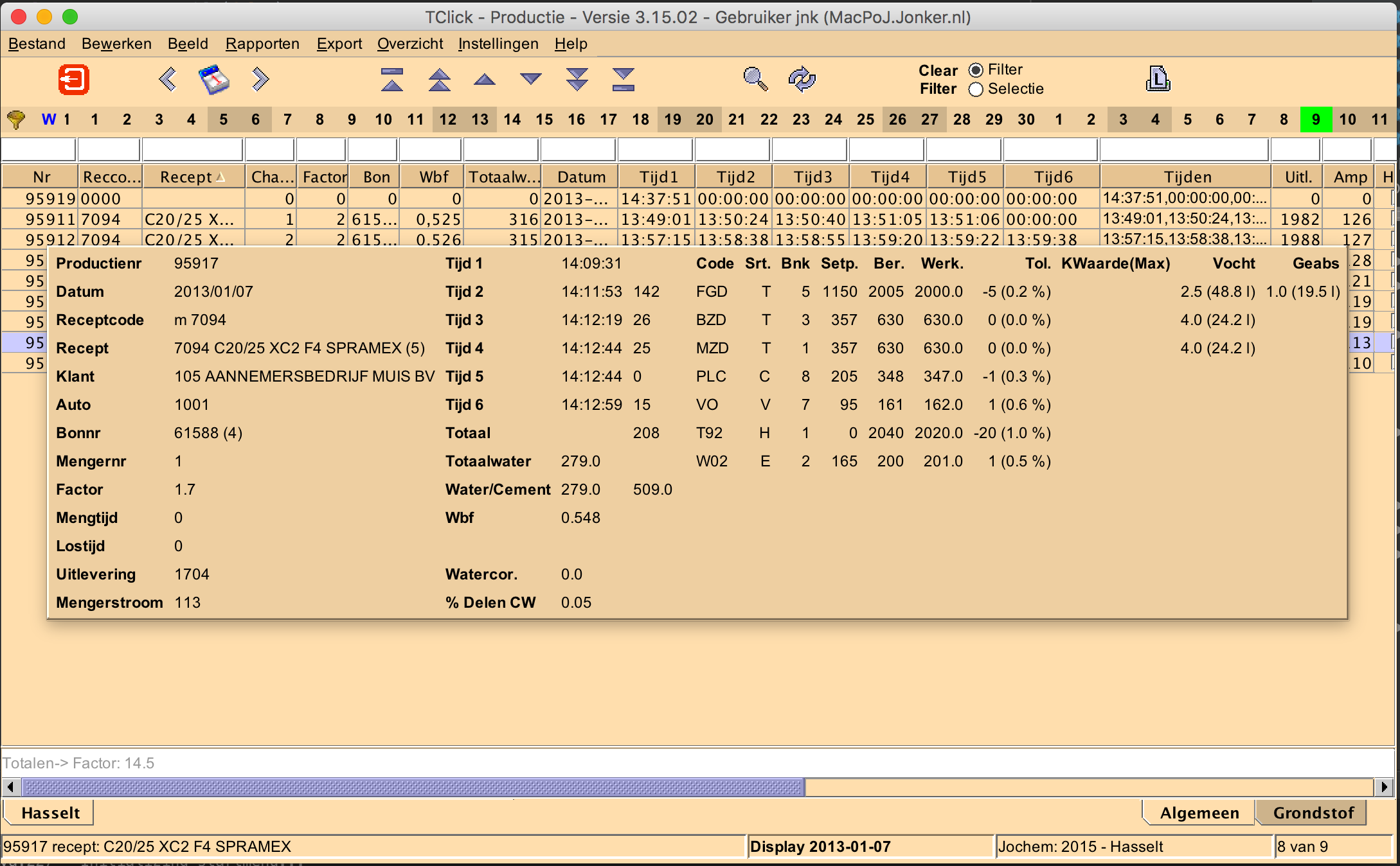
|  |  |
| --- | --- |
| Nr: | 4 |
| Versie nr: | 1.0 |
| Auteur: | Jochem Pouwels |
| Usecase: | Uitgebreidere informatie tonen van een productie |
| Actoren: | Gebruiker |
| Samenvatting: | Er wordt een productie informatie popup (figuur 6) getoond die meer informatie over de productie toont dan het productieoverzicht |
| Prioriteit: | Must |
| Pre conditie | Actor heeft productiemodule geopend |
| Hoofdscenario | * 1. Actor dubbelklikt op een rij in het productieoverzicht   2. Systeem toont productie informatie popup |
| Post conditie | Uitgebreide informatie wordt getoond van geselecteerde productie |
| Alternatief scenario 1: | [productie informatie popup wordt al getoond]   * 1. Actor klikt op een andere rij in het productieoverzicht   2.1 Systeem update productie informatie popup met de nieuw geselecteerde productie |
| Post conditie AS1 | Uitgebreide informatie wordt getoond van geselecteerde productie |

Tabel 6 - Usecase5: Kolomconfiguratie wijzigen

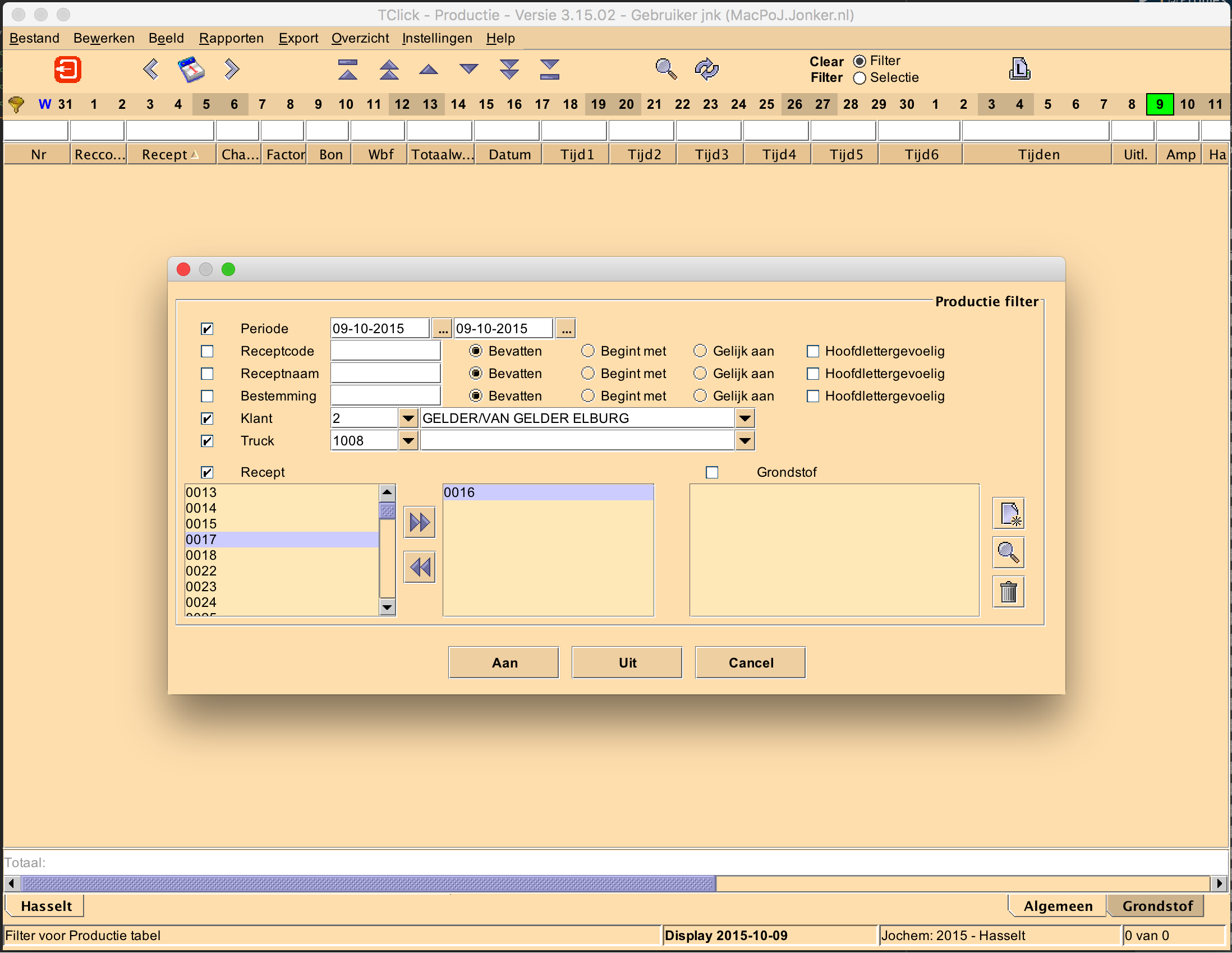
|  |  |
| --- | --- |
| Nr: | 4 |
| Versie nr: | 1.0 |
| Auteur: | Jochem Pouwels |
| Usecase: | Kolomconfiguratie wijzigen |
| Actoren: | Gebruiker |
| Samenvatting: | De breedte en zichtbaarheid van kolommen kunnen worden ingesteld door de gebruiker |
| Prioriteit: | Would |
| Pre conditie | Actor heeft productiemodule geopend |
| Hoofdscenario | * 1. Actor klikt op de kolomconfiguratie knop   2. Systeem toont kolomconfiguratie popup (figuur 8)   3.1 Actor wijzigt breedte en/of zichtbaarheid van kolommen  3.2 Actor klikt op de “opslaan” knop  4.1 Systeem wijzigt de kolomconfiguratie instellingen in de database  4.2 Systeem sluit kolomconfiguratie popup  4.3 Systeem vernieuwt de productietabel |
| Post conditie | Uitgebreide informatie wordt getoond van geselecteerde productie |
| Alternatief scenario 1: | [Actor annuleert de wijzigingen]   * 1. Actor klikt op de “annuleer” knop   4.1 Systeem sluit de kolomconfiguratie popup |
| Post conditie AS1 | Geen |

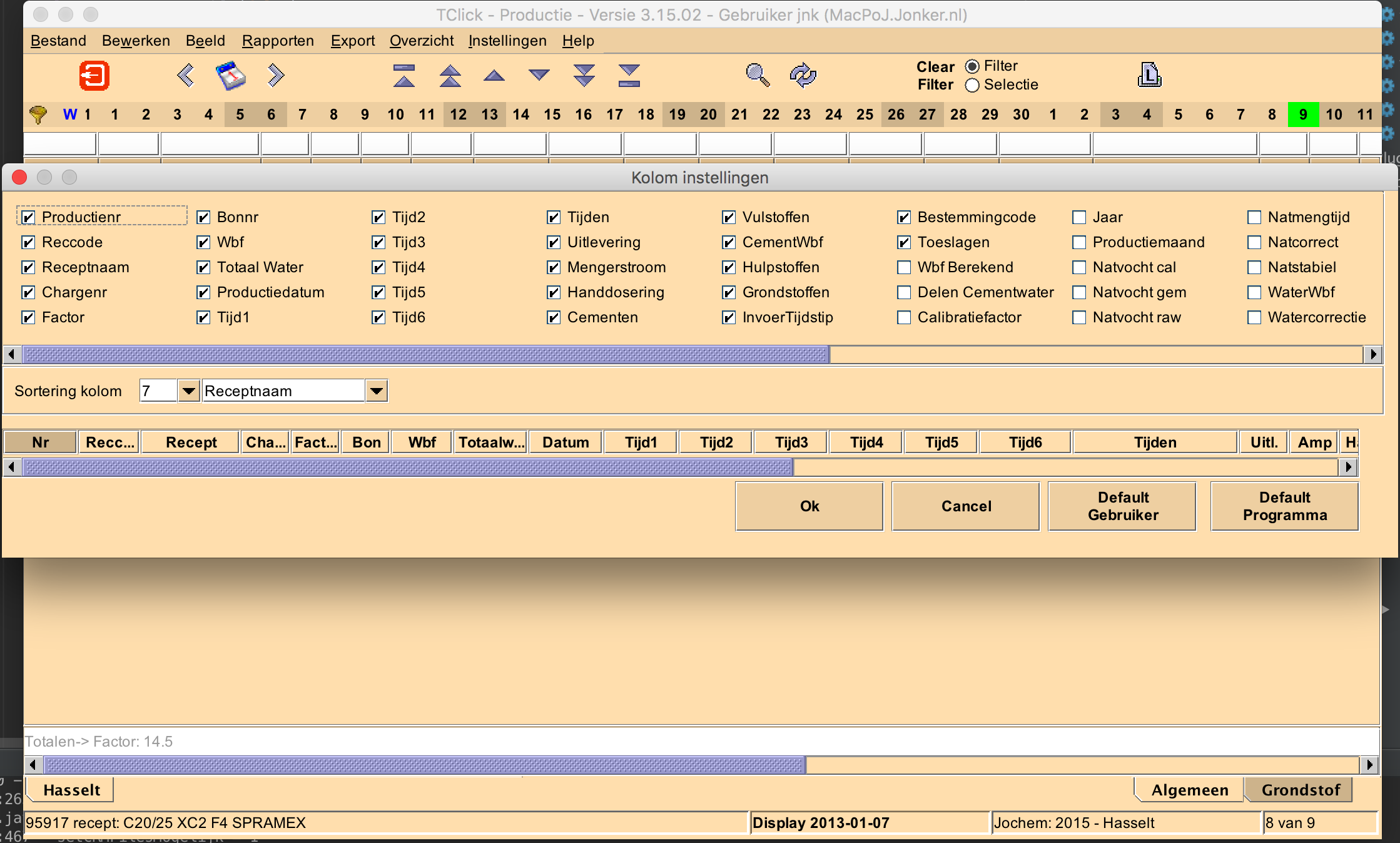


Figuur 5 – Het productieoverzicht



Figuur 6 – De productie informatie popup

Figuur 7 – De productie filter popup

Figuur 8 – De kolomconfiguratie popup

## Technische specificaties

Dit hoofdstuk beschrijft de technische specificaties van het proof of concept. Hierbij wordt onderscheidt gemaakt in twee verschillende onderdelen, de web service en de web client. In dit hoofdstuk wordt altijd aangegeven voor welk onderdeel de specificatie geldt.

### Code conventies

Beide onderdelen worden geschreven met Java als hoofdprogrammeertaal. Om ervoor te zorgen dat de code in het algemeen goed te onderhouden blijft door andere programmeurs wordt er gebruik gemaakt van: Code Conventions for the Java Programma Language te vinden op: <http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>

### Maven

Apache Maven wordt in zowel de web service en de web client gebruikt. Dit was een van Jonker’s wensen, aangezien elk project binnen Jonker tegenwoordig in Maven wordt gemaakt.

Er wordt vooral gebruik gemaakt van het dependency mechanisme van Maven. Deze zorgt ervoor dat alle dependencies van een project beheert kunnen worden op een centrale locatie, indien een dependency niet aanwezig is in het project zal Maven deze zoeken en automatisch downloaden van het internet.

### Applicatieserver

Voor het starten van de web service is er gebruik gemaakt van een WildFly applicatie server. De reden hiervoor is omdat de applicatieserver al bekend en gebruikt is binnen het software development team van Jonker. Tevens komt JBoss(alias WildFly) volgens (Maple, 2013). als beste uit de test in een vergelijking tussen de populairste Java applicatie servers.

De configuratie van de applicatieserver is aangepast om alleen te werken met een SSL verbinding, dit heeft te maken met de vorm van authenticatie die wordt gebruikt. De SSL verbinding zorgt er onder andere voor dat een man-in-the-middle attack praktisch onmogelijk wordt.

### REST

Op de applicatieserver is een REST web service gebouwd. De methodes op de REST web services zijn zo ingedeeld zodat de usecases gespecificeerd in de functionele specificaties van [hoofdstuk 9.2](#_Functionele_specificaties) mogelijk worden gemaakt.

De REST web service biedt de data in meerdere vormen aan, namelijk in XML of JSON. De gebouwde web client maakt enkel gebruik van JSON aangezien er dan minder data over het netwerk wordt gestuurd. XML zou gebruikt kunnen worden als een andere client dat wenst.

### PostgreSQL

In dit proof of concept is de Firebird database van Jonker overgezet naar een PostgreSQL database. Om dit te realiseren moesten een aantal taken gebeuren.

* Het databasegebruik van de productiemodule moest worden geanalyseerd om erachter te komen welke tabellen er moesten bestaan om de nieuwe applicatie te laten werken, in totaal zijn dit er 18.
* Voor de gebruikte tabellen moest een nieuw DDL script worden gegenereerd in de PostgreSQL syntax.
* De structuur van de tabellen moest worden herzien en aangepast zodat deze gebruik maken van id’s, foreign keys en functionaliteiten die wel in PostgreSQL aanwezig zijn, maar niet in Firebird.
* De data van de Firebird database moest worden geëxporteerd naar de nieuwe PostgreSQL database. Hiervoor is vanuit de oude database een insert script gegenereerd, dit is gedaan met een tool genaamd RazorSQL. Echter omdat de structuur van de tabellen ook is veranderd moesten deze aanpassingen ook worden doorgevoerd in de insert scripts. Voor sommige tabellen waren er meer dan 40.000 inserts nodig, handmatig veranderen was geen optie. Er is gebruik gemaakt van RegExp om dit semiautomatisch aan te passen.

De PostgreSQL staat in verbinding met de web service i.c.m. een connection pool.

### MyBatis

In het proof of concept zijn de SQL query’s gemaakt met Java Strings vervangen door de SQL query’s gemaakt in MyBatis XML.

Om alle query’s uit de huidige software te halen is er een aanpassing gemaakt in de source code van de productie module. Hierdoor werden alle uitgevoerde query’s in een logbestand geplaatst. Deze methode bleek handiger dan het doorzoeken van de source code.

Daarnaast zijn ook een aantal query’s geoptimaliseerd. Dit is gedaan door sommige query’s anders op te zetten en te vergelijken in een “explain” query. De executietijd kon daarna worden vergeleken met pgAdmin, een tool geleverd met de PostgreSQL database.

### JSF + Primefaces

De web client is gemaakt met JavaServer Faces 2.0 (JSF2) + PrimeFaces. Deze combinatie is ook gebruikt bij een ander web project van Jonker, het was wenselijk dezelfde technieken te gebruiken bij dit project.

JSF is een user interface framework voor Java web applicaties. JSF zorgt ervoor dat het ontwikkelen van een web applicatie wordt versimpeld door het makkelijk te maken om de interface te koppelen met Java controllers (aka beans). Hieronder worden een aantal basisfunctionaliteiten van JSF benoemd:

* Maakt gebruik van UI componenten met parameters zodat deze makkelijk herbruikbaar zijn.
* Biedt een simpele API dat helpt bij het koppelen van client-side html pagina met server-side controllers.
* Helpt met het bijhouden gebruikerssessies.

PrimeFaces is een componenten bibliotheek dat gebruikt kan worden om JSF2 uit te breiden.

### Beveiliging

De REST web service maakt gebruik van Apache Shiro om methodes af te schermen die niet bereikt mogen worden zonder in te loggen. Shiro is een Java security framework waarmee web applicaties snel en makkelijk kunnen worden beveiligd. Authenticatiemethodes zijn pluggable in Shiro, dit betekend dat er gemakkelijk gewisseld kan worden.

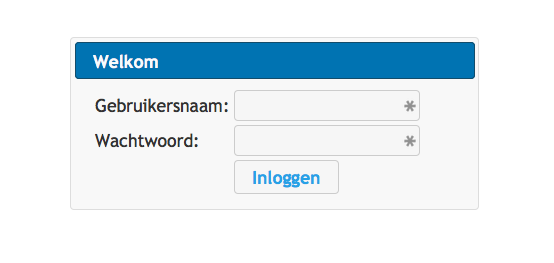
Er wordt in het geval van het proof of concept gebruik gemaakt van HTTP Basic Auth in combinatie met een SSL verbinding om gebruikers in te laten loggen.

## Het eindresultaat

In dit hoofdstuk wordt het eindresultaat en de werking ervan getoond. Dit wordt gedaan door screenshots en toelichtingen.

### Inloggen

Net zoals in de huidige applicatie, kan er pas gewerkt worden met het systeem na het invullen van een inlognaam en een wachtwoord. De authenticatie loopt in de nieuwe situatie alleen niet meer direct via de database. Zoals in de technische specificaties al is genoemd, gebruiker we hiervoor HTTP Basic Auth in combinatie met een SSL verbinding. Hieronder, in figuur 9, wordt het inlogscherm getoond



Figuur 9 - PoC: inlogscherm

Nadat de gebruiker het formulier heeft ingevuld en op de knop “inloggen” klikt worden de inloggegevens verstuurt in het base64 formaat.

De rest service geeft hierna terug of het inloggen is gelukt. Als de gebruiker eenmaal succesvol is ingelogd wordt hij doorgestuurd naar het startmenu. Daarnaast wordt in de sessie van de gebruiker opgeslagen dat hij is ingelogd. De gebruiker heeft hierna toegang tot de rest van de client, totdat hij uitlogt.

Het afschermen van het grootste deel van de client werd gedaan door het implementeren van een AuthenticationFilter en deze aan te geven in het web.xml bestand.

<filter>

<filter-name>AuthenticationFilter</filter-name>

<filterclass>nl.jonker.admwebclient.controllers.AuthenticationFilter</filter- class>

</filter>

<filter-mapping>

<filter-name>AuthenticationFilter</filter-name>

<url-pattern>/index.xhtml</url-pattern>

<url-pattern>/productie/\*</url-pattern>

</filter-mapping>

### Het startmenu

Na het inloggen wordt de gebruiker doorgestuurd naar het startmenu. Dit is een feite niks anders als een doorstuurpaneel. Vanaf hier kan de gebruiker elk gewenst pakket opstarten. Voor het proof of concept is alleen de productie module geactiveerd. Hieronder bevindt zich een screenshot van het startmenu.



Figuur 10 - PoC: startmenu

### De productiemodule

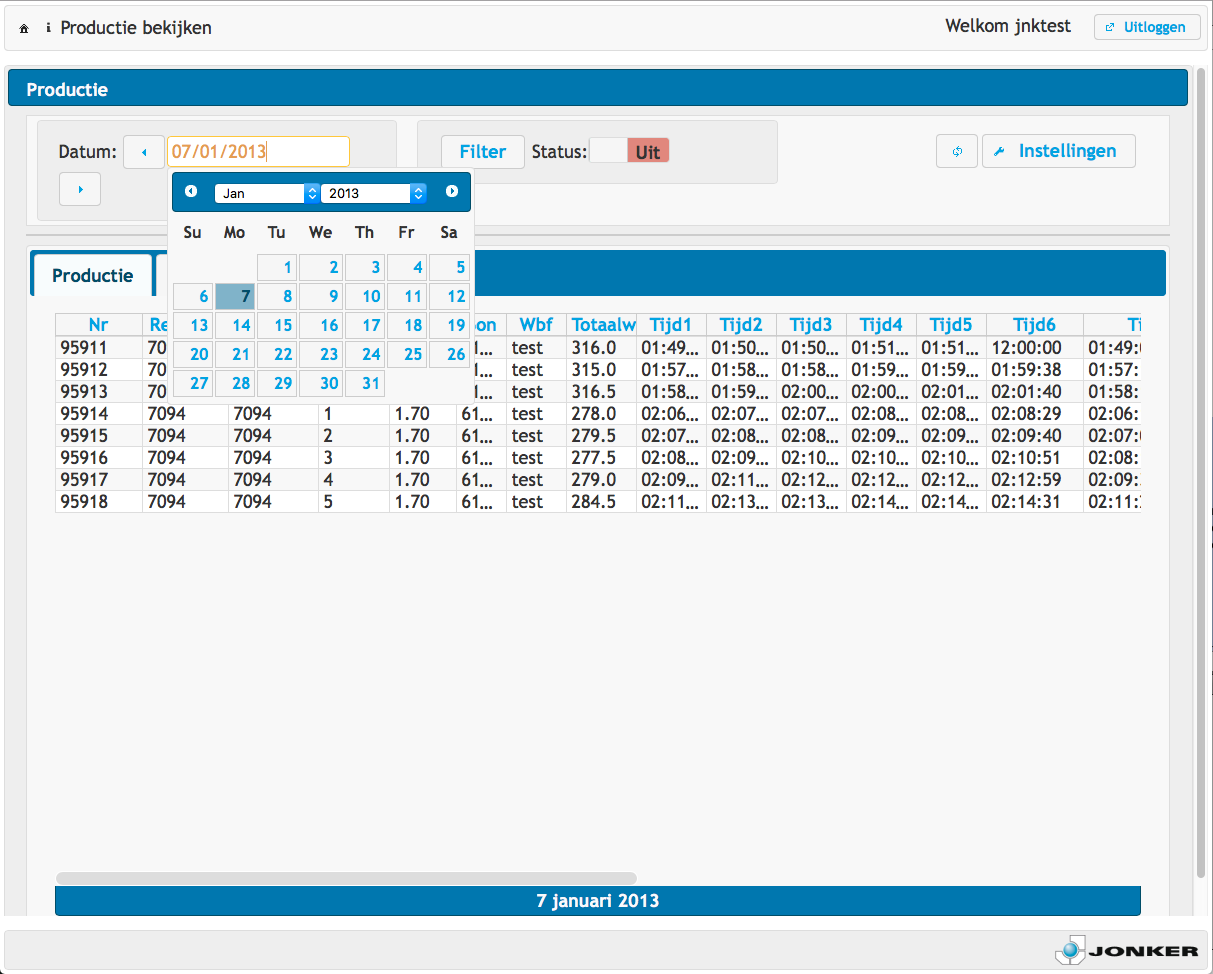
De productiemodule in de nieuwe web client bevat alle functionaliteiten die omschreven zijn in de usecases van de functionele specificaties. Zo wordt bij het opstarten van de module meteen een overzicht getoond van de huidige dag en is het hierna mogelijk om:

* Een overzicht te maken van producties op een dag, door deze te selecteren in een kalenderinvoerscherm of van dag te wisselen met de “volgende”/”vorige” knop (figuur 11)
* Een overzicht te maken van productie op basis van filtercriteria (figuur 12)
* Uitgebreide informatie te tonen over een productie door er twee keer op te klikken (figuur 13)
* De kolom breedte en zichtbaarheid te wijzigen (figuur 14)

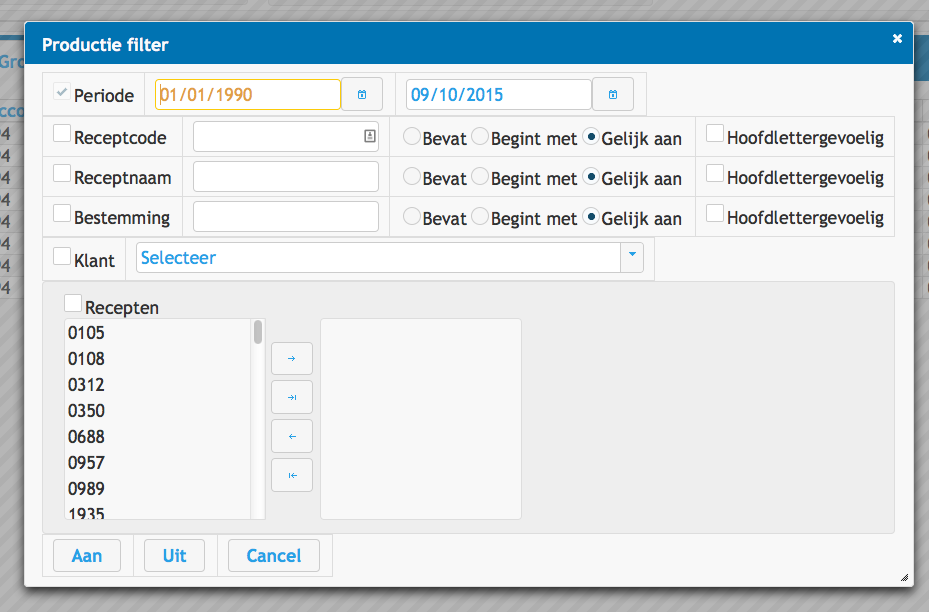
Naast deze functionaliteiten is er een uitbreiding gemaakt op het productieoverzicht, deze maakt nu namelijk gebruik van lazy-loading. Dit is een techniek die ervoor zorgt dat niet alle data in één keer wordt opgehaald, maar dat dit wordt gedaan in kleinere gedeeltes. Als de gebruiker bijvoorbeeld filtercriteria invult waarmee 100.000 producties overeen komen, wordt niet al deze data meteen opgehaald. De lijst wordt hier eerst gevuld met de eerste X aantal producties, laten we als voorbeeld het aantal 30 nemen.

Als de gebruiker naar beneden scrollt en op een gegeven moment de 30e rij in het scherm komt. Dan zorgt de web client ervoor dat een volgende query wordt uitgevoerd, die alleen 31e t/m de 60e productie ophaalt. De gebruiker kan hierna weer scrollen tot de 60e rij en daarna worden weer de volgende producties opgehaald. En dit kan zo door gaan tot de 100.000 is bereikt.

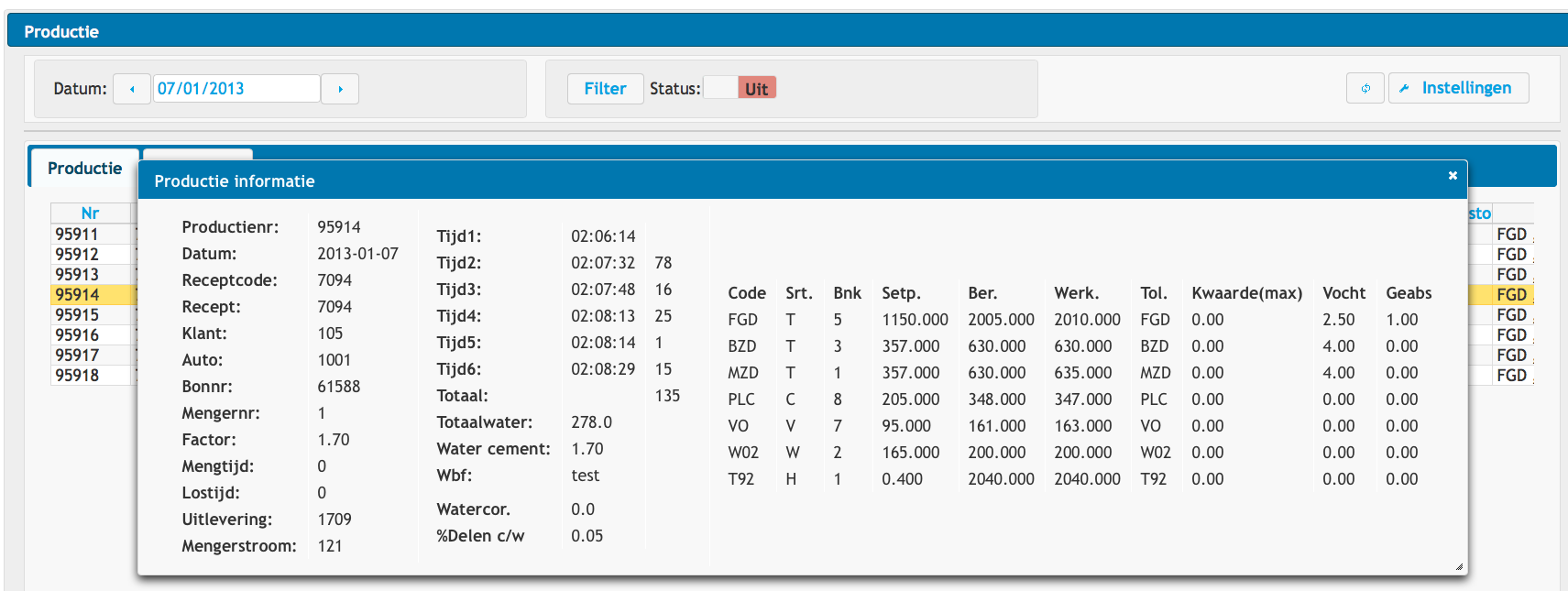
Het implementeren van lazy-loading zorgt er in de productieomgeving voor dat de laadtijden bij het ophalen van producties vele malen kunnen worden verkort.



Figuur 11 - PoC: Productieoverzicht



Figuur 12 - PoC: filter criteria



Figuur 13 - PoC: uitgebreide informatie

### Macintosh HD:Users:Jochem:Desktop:Screen Shot 2015-10-09 at 17.05.08.png

Figuur 14 - PoC: kolomconfiguratie

### Conclusie

De wensen van Jonker om dezelfde look en feel te behouden ten opzichte van het huidige pakket is zeker gelukt. Alle componenten die gebruikt werden in de Java Swing applicatie zijn op bijna gelijke wijze geïmplementeerd in de web client gemaakt in PrimeFaces.

De business logica gebeurt nu niet meer direct vanaf de client op de database, maar er wordt gebruik gemaakt van een REST service die query’s uitvoert met MyBatis.

De source van de web client is te bekijken op: <https://www.dropbox.com/sh/8pwp5xoskmnsw18/AADb1Jgslwbx73Xz4F8g_UcYa?dl=0>

En de source van de REST web service is te bekijken op: <https://www.dropbox.com/sh/vbv7ceule2sxhe4/AAC0x0pKFPrOL5PBh21FosnTa?dl=0>

# Afwijking plan van aanpak

## Authenticatie

In het plan van aanpak was de volgende deelvraag aanwezig: Welke authenticatiemethode kan gebruikt worden om de klant remote te laten inloggen in het systeem, waarmee de klant ook kan wisselen tussen de software pakketten zonder opnieuw in te moeten loggen?

Tijdens de stage periode is deze vraag niet kunnen onderzoeken. De wensen/eisen van Jonker voor authenticatie in de gecentraliseerde omgeving staan nog niet vast. Er zijn meerdere mogelijkheden, bijvoorbeeld het werken met API keys of het gebruiken van OAuth2 providers. Jonker wilt zelf nog onderzoeken op welke manier de web service wordt aangeboden aan derde partijen.

Wel is bij het maken van de proof of concept rekening gehouden met een mogelijke wijziging in authenticatie door gebruik te maken van Shiro.

## Proof of concept (client)

Bij het proof of concept heeft ook een wijziging plaatsgevonden. Het originele idee was om de huidige Java code aan te passen zodat deze kan werken met een web service i.p.v. direct met de database. Er is nu echter gekozen om een hele nieuwe client te maken. De reden hiervoor komt vanuit een recente wens van Jonker om de Betondag 2015 bij te wonen. De Betondag is een beurs voor de betonindustrie. Hier wil Jonker nieuwe ontwikkelingen laten zien van o.a. het ADM pakket. Aan de afstudeerder is gevraagd hier iets moois voor te maken.

Het wijzigen van het proof of concept heeft er ook tot geleid dat een ander soort test moet worden gedaan. In het plan van aanpak werd gesproken over een unit test om de werking van de software te testen, deze wordt nu vervangen door een acceptatietest.

# Terugblik eindresultaten

In [hoofdstuk 5.4](#_Eindproducten) van deze scriptie staat een lijst met op te leveren producten. Per product worden de criteria beschreven waaraan het product moet voldoen. In dit hoofdstuk wordt de huidige stand van zaken aangegeven van deze eindproducten.

## Onderzoeksresultaten

Alle onderzoeksresultaten zijn compleet en werden gebruikt om de onderzoeksvragen te beantwoorden.

## Proof of concept

Er is een proof of concept waarin alle gekozen technieken goed samen werken. De functionaliteiten van het proof of concept komen overeen met vastgestelde usecases uit het huidige ADM pakket.

## Testrapport

Er is een test plan opgesteld welke is gebaseerd op de functionaliteiten van het ADM softwarepakket. Het proof of concept heeft deze succesvol doorlopen. De resultaten van de verschillende testen zijn ingevuld door meerdere collega’s, dit vormt het testrapport.

## Scriptie

De scriptie is inmiddels afgerond en voldoet aan de eisen zoals beschreven in het afstudeerleidraad.

# Evaluatie van de procesgang

In dit hoofdstuk wordt een terugblik geworpen op de het verloop van het afstuderen.

Voordat het afstudeerproject begon ben ik eerst op zoek gegaan naar een geschikt bedrijf. Ik heb niet specifiek op één soort bedrijf gezocht, maar ik ben gewoon vacatures gaan rondkijken naar opdrachten die mij erg interessant leken. Ik had inmiddels meerdere sollicitaties gehad: een mobile app ontwikkelaar, een bedrijf gespecialiseerd in e-learning software en een softwarebedrijf in de betonindustrie.

Ikzelf kon het eerst niet geloven, maar ik koos voor het bedrijf in de betonindustrie. Waarbij ik voordat ik naar de sollicitatie ging denkend: betonindustrie, is dat wel wat voor mij? Ach we kijken wel even. Dacht ik nadat ik klaar was met de sollicitatie en weer naar huis reed: “Wauw, dit is wel echt een bedrijf voor mij!”.

Eenmaal thuis had ik nog even de tijd om na te denken, want alle drie de bedrijven hadden tijdens het eerste gesprek al de toezegging gedaan dat ik zou kunnen beginnen, als ik dat wou. Na kort nadenken was de keus gemaakt, ik ga afstuderen bij Jonker.

Eenmaal begonnen bij Jonker kreeg ik eerst een rondleiding en uitleg over de organisatie, daarna kon ik meteen beginnen. Ik begon als eerst met het installeren van alle software, waar mijn collega’s mij bij hielpen. Eenmaal de ochtend besteed had ik de meeste software wel geïnstalleerd en de Subversion repositories uitgecheckt. De rest van de dag mocht ik de tijd nemen om de werking van hun software wat te bekijken, vragen stellen kon gemakkelijk, want iedereen zat op hetzelfde kantoor.

Na die eerste dag stonden de twee weken daarop vooral in het thema “Plan van aanpak maken”. Een lastige, maar wel een verhelderende klus. Ik kreeg steeds meer kennis van de bestaande software, en door de feedback die ik kreeg van de bedrijfsbegeleider en mijn collega’s kreeg ik een steeds beter beeld van de uiteindelijke opdracht.

Na uiteindelijk nog twee weken half aan het plan van aanpak was begonnen en deze inmiddels was goedgekeurd, en half aan het database onderzoek was begonnen. Dacht ik: “Dit wordt een makkie, zo’n database voor een nieuwe omgeving heb ik zo opgezet”. Dit idee veranderde echter toen ik met een database tool in de database kon kijken. Hiervoor had ik alleen nog maar de sourcecode van de applicatie bekeken. De database bleek groot, heel groot! Ik probeerde zelf te achterhalen wat elke tabel deed, maar al snel raakte ik de weg kwijt. Ik heb dit besproken met de bedrijfsbegeleider, en die snapte mij wel. De huidige database is zo groot, heeft geen documentatie. Deze is niet zomaar te begrijpen, we besloten het database onderzoek niet specifiek op deze database te richten, maar gewoon te kijken naar de algemene databasezaken. Het database onderzoek liep hierna nog een aantal weken en was daarna toch uiteindelijk helemaal klaar. Het bespreken liep wat vertraging op, dus in de tussentijd kon ik mooi werken aan een deel van de scriptie.

De andere onderzoeken die zijn verricht liepen eigenlijk soepeltjes, het documenteren en het zoeken van de juiste informatie kostte een hoop tijd, maar echte problemen kwam ik niet tegen. Ook de sparsessies met collega’s brachten veel op qua kennis.

Eenmaal klaar met deze onderzoeken, was ik ook wel een beetje klaar met onderzoeken. Na al dat type en onderzoek werk al die weken was ik blij dat ik kon beginnen aan het proof of concept. Een volgende keer zou ik wel het onderzoek wat meer verdelen over de gehele periode, dit was nu namelijk allemaal aaneengesloten.

Het maken van het proof of concept kon eigenlijk niet beter. Door al het onderzoek wat ik gedaan had, had ik ook al heel veel voorkennis. Hierdoor heb ik erg snel de gekozen technieken werkend kunnen maken. De usecases heb ik besproken met de bedrijfsbegeleider en ik kon zo beginnen met het implementeren ervan. Bij het implementeren kon ik mijn programmeerkennis in Java goed gebruiken, want Jonker had ook de wens om de client te maken in een Java framework. Eenmaal hiermee klaar, kon ik weer beginnen aan het afronden van mij scriptie. Dit is naar mijn idee ook soepeltjes verlopen, op een aantal stoeisessies met Microsoft Word OSX na dan.

Al met al ben ik zeker tevreden met het resultaat, het onderzoek heeft ervoor gezorgd dat Jonker gegronde keuzes kon maken voor de nieuwe software. En daarnaast was iedereen zeker te spreken over het resultaat van het proof of concept en hoe goed dit eigenlijk ging met de gekozen technieken.

# Literatuurlijst

Shiny. (2013, 07 01). *Kenmerken: Kwalitatief & Kwantitatief onderzoek*. Opgehaald van wetenschap.infonu.nl: http://wetenschap.infonu.nl/onderzoek/106079-kenmerken-kwalitatief-kwantitatief-onderzoek.html

ITASC. (2009, 03 03). *De mindmap en ons denkproces*. Opgehaald van www.competentietest.com: http://www.competentietest.com/trainotheek/ob/Mindmappen.pdf

Maple, S. (2013, 3 21). *The Great Java Application Server Debate*. Opgehaald van zeroturnaround.com: http://zeroturnaround.com/rebellabs/the-great-java-application-server-debate-with-tomcat-jboss-glassfish-jetty-and-liberty-profile/

# Bijlagen

## Bijlage A – Plan van aanpak



Plan van aanpak – Jonker BV.

Versie 1.2

Jochem Pouwels - 1613727  
Docentbegeleider: Rik Boss

Inhoudsopgave

1 Inleiding 59

2 Aanleiding en context 59

2.1 Organisatiebeschrijving 59

2.2 Technisch context 60

2.3 Aanleiding 61

3 Doelstelling 62

3.1 Doelstelling van het afstudeerproject 62

3.2 Doelstelling van Jonker 62

4 Opdrachtformulering 62

4.1 Beschrijving van de opdracht 63

4.2 Hoofdvraag 64

4.3 Deelvragen 64

4.4 Kwaliteitscriteria 64

4.4.1 Documenten 64

4.4.2 Proof of concept 64

4.4.3 Onderzoek 64

4.4.4 Risico’s 65

4.5 Op te leveren producten 65

4.5.1 Onderzoeksrapport 65

4.5.2 Proof of concept 65

4.5.3 Test rapport 66

4.6 Projectgrenzen en randvoorwaarden 66

5 Aanpak 66

5.1 Concept inhoudsopgave scriptie 67

5.2 Onderzoeksmethoden 68

5.3 Globale planning 69

6 Risicoanalyse 70

6.1 Risico’s 70

6.2 Risicomatrix 71

6.3 Tegenmaatregelen 72

6.3.1 A - Tijdelijke uitval van uitvoerend project lid 72

6.3.2 B - Complete uitval van uitvoerend project lid 72

6.3.3 C - Conflicten binnen het bedrijf 72

6.3.4 D - Keuzes nieuwe situatie ingrijpender dan verwacht 72

7 Gegevens betrokkenen 73

7.1 Algemene informatie 73

7.2 Contactgegevens 73

# Inleiding

Dit document bevat het plan van aanpak voor de afstudeeropdracht van Jochem Pouwels. Het onderwerp van de opdracht is het onderzoeken van de mogelijkheden voor een nieuwe structuur voor de softwarepakketen van opdrachtgever Jonker BV.

Als eerst zal in hoofdstuk 2 een korte beschrijving worden gegeven van de opdrachtgever en de aanleiding van deze opdracht. In het hoofdstuk hierna wordt in hoofdstuk 3 uitgelegd wat de doelstelling is van dit afstudeerproject. Hoofdstuk 4 bevat de opdrachtformulering met hierbij ook de hoofd en deelvragen, een lijst van producten die op het eind van dit project worden opgeleverd en een uitleg hoe de kwaliteit van deze producten gegarandeerd worden. In hoofdstuk 5 staat de opzet en aanpak van dit project en in hoofdstuk 6 volgt een risicoanalyse. Tot slot staan nog de gegevens van alle betrokkenen genoteerd.

# Aanleiding en context

In dit hoofdstuk zullen de missie en de kernactiviteiten van de opdrachtgever worden beschreven samen met een omschrijving van de ICT omgeving binnen het bedrijf, om zo een duidelijker beeld te geven over de context van het afstudeerproject. Ook zal de aanleiding aan bod komen hoe deze opdracht is ontstaan.

## Organisatiebeschrijving

Jonker is specialist op het gebied van besturingen en administratieve software voor processen waar wegen en doseren een belangrijk aspect is. Jonker is onder andere actief in de marktsegmenten betonmortel, betonwaren, droge mortel, asfalt, diervoeding, chemie en food.

Voor het nauwkeurig wegen en doseren heeft Jonker voor hun besturingen lange tijd eigen hardware in de vorm van printplaten en sensoren moeten ontwikkelen. Hardware van 25 jaar en ouder zijn hierbij geen uitzondering. Standaard componenten die heden ten dage op de markt zijn voldoen wel aan de eisen en daarom is Jonker overgestapt van eigen hardware naar standaard componenten. Jonker levert nog steeds besturingen aan klanten en heeft hier ook nog een aparte afdeling voor, echter door de overstap naar standaard componenten kon Jonker zich meer gaan focussen op hun software.

Een belangrijke stap voor Jonker is de ontwikkeling van de administratieve software. Bij veel bedrijven staat de administratieve software los van de besturingssoftware. Integratie van beiden levert vele voordelen op voor de klant.

## Technisch context

Jonker’s administratieve software voegt allerlei taken samen in één pakket, tClick.

tClick dient om betonmortelcentrales te ondersteunen in een efficiënte bedrijfsvoering. Het pakket bestaat uit meerdere modules waarmee de gebruikers verschillende bedrijfsprocessen kunnen automatiseren. Naast handige modules voor het maken van offertes en het factureren van klanten zijn er ook modules die specifiek voor betonproductie zijn ontwikkeld.

Zo kunnen gebruikers zelf nieuwe betonrecepten doorberekenen met de laboratorium module. Aan de hand van specifieke eisen zoals sterkte, soort, kleur, chlorideklasse, weerstand en consistentie worden ingewikkelde algoritmen gebruikt om de goedkoopste samenstelling te maken voor het beton.

Bij het maken van een order zorgt tClick ervoor dat de machines in de betoncentrales meteen beginnen met het mengen van het bestelde beton.

Na het maken van het beton moet deze ook nog geleverd worden, het leveren van beton is een ingewikkeld proces, vooral als er meerdere klanten tegelijk beton bestellen. Er moet rekening worden gehouden met beschikbaarheid van de truckmixers, loscapaciteit van elke klant, en droogsnelheid van het beton.

Door middel van complexe algoritmen en een partnerrelatie met TomTom om de GPS locatie en status van elke truckmixer bij te houden, kan tClick de gehele planning op de minuut precies volledig automatisch regelen zonder dat hier input van de gebruiker nodig is.

De tClick software is ontwikkeld met Java als programmeertaal en gebruikt een GUI gemaakt met Java‘s Swing API. De client verbindt direct met de database door gebruik te maken van een JDBC driver waarmee zelfgemaakte query’s worden uitgevoerd.

De back-end van tClick is gedecentraliseerd, wat betekent dat de data die gebruikt wordt in de software niet op één plek verwerkt wordt. Elke vestiging van de klant heeft zijn eigen server en database.

De klant verbindt met hun tClick client met de server in de vestiging waar hij zich bevindt, waarna deze de data verwerkt op de lokale Firebird database. Deze database maakt gebruik van eigen geschreven replicatiesoftware, dit is software dat er voor zorgt dat de data gesynchroniseerd wordt naar de andere gekoppelde databases die op andere vestigingen van de klant staan. Alle databases kunnen zowel data schrijven als lezen, deze opstelling heet multi-master. De replicatiesoftware houdt rekening met kleine internetstoringen en gebruikt daarom per vestiging een kleine marge in de records van de database, zo worden bijvoorbeeld bepaalde factuurnummers gereserveerd om conflicten te voorkomen met het synchroniseren van de databases.

## Aanleiding

Jonker’s administratiesoftware wordt al door vele klanten gebruikt. Er is in 2004 voor een gedecentraliseerde back-end constructie gekozen, omdat toentertijd de internet voorzieningen nog niet overal goed op orde was. Dat ergens de verbinding er een uur uit lag werd niet als uitzondering gezien. De gedecentraliseerde back-end zorgde er toen voor dat als het internet uit viel, men de software gewoon nog kon gebruiken.

Door de gedecentraliseerde structuur is het noodzakelijk voor Jonker om multi-master replicatiesoftware te gebruiken om de data bij elke vestiging van de klant gelijk te houden, dit heeft echter nadelen voor Jonker en hun klanten.

Voor de klanten is het een nadeel dat er gaten kunnen ontstaan in de nummeringen van de records. Ook kan er niet gegarandeerd kan worden dat records op chronologische volgorde staan. Voor taken zoals de boekhouding kan dit problemen opleveren.

Voor Jonker zorgt de replicatiesoftware voor onnodig handmatig werk als bepaalde vestigingen van de klant een langere tijd offline is geweest. Er kunnen dan conflicten ontstaan bij het synchroniseren als ze daarna weer online werken. Deze conflicten moeten telkens handmatig door Jonker opgelost worden.

Ook heeft Jonker last van een ontbrekende service laag in hun software. Er is namelijk veel vraag naar meerdere soorten clients. De klant zou graag bepaalde taken willen uitvoeren via web of app toepassingen en wil dit dan ook kunnen doen via een mobiel netwerk. Echter is het zonder service laag en een centraal toegangspunt niet rendabel om te kijken naar meerdere clients die eventueel zelfs op het mobiele netwerk kunnen functioneren.

Tegenwoordig is internet een stuk stabieler en met ondersteuning van bijvoorbeeld een 4G dongel is Jonker van mening dat er geen rekening meer hoeft gehouden worden met internetstoringen. Jonker vindt het nu een goed moment om te kijken naar een nieuwe gecentraliseerde opzet van hun administratieve software, waarmee ook bovengenoemde nadelen opgelost kunnen worden.

# Doelstelling

In dit hoofdstuk wordt de doelstelling van het afstudeerproject en die van Jonker apart beschreven aangezien deze van elkaar verschillen.

## Doelstelling van het afstudeerproject

Jonker heeft een aantal limitaties door de verouderde structuur van hun administratieve software. Het doel van dit project is om Jonker inzicht te geven wat technieken die tegenwoordig op de markt zijn kunnen betekenen bij het vernieuwen van hun tClick software naar een gecentraliseerde omgeving.

Hiervoor moet worden onderzocht wat de beste keuze is in de te gebruiken databasetechnieken, welke persistence laag past bij de software, hoe een service laag kan worden opgezet en welke clients hiermee zouden kunnen verbinden.

De adviezen worden samen met Jonker besproken en terwijl het project vordert wordt er gekozen welke technieken gebruikt gaan worden in de nieuwe omgeving.

Er zal een proof of concept worden gemaakt door één van Jonker’s softwarepakketten om te zetten zodat deze werkt met de nieuwe technieken. Dit proof of concept zal dan kunnen worden getest op prestaties en functionaliteit.

## Doelstelling van Jonker

Jonker wil graag hun administratiesoftware gecentraliseerd opzetten om het zo meer toekomstgericht te maken, naar een hoger niveau te tillen en het mogelijk te maken de back-end te bereiken met een mobiel netwerk.

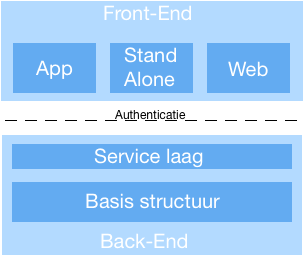
Ook kan Jonker met gecentraliseerde software een nieuw verdienmodel introduceren. Op dit moment wordt er geld verdient door het pakket te verkopen en de benodigde hardware bij de klant te plaatsen. Met een gecentraliseerde back-end kan de data en de service laag intern bij Jonker geplaatst worden en kunnen ze hun pakket aanbieden middels een leaseconstructie. Op deze manier is er meer controle over de veiligheid van data omdat Jonker zelf een goede back-up constructie kan maken.

# Opdrachtformulering

## Beschrijving van de opdracht

Jonker heeft op dit moment een gedecentraliseerde back-end voor hun tClick software waarbij de databases en servers op de verschillende vestigingen van de klant staan. Het onderhouden van de software en database gaat op dit moment te omslachtig volgens Jonker. Ook wordt Jonker gelimiteerd met het uitbreiden naar meerdere clients door het ontbreken van een service laag.

De opdracht bestaat uit het adviseren van Jonker welke technieken ze kunnen gebruiken bij het maken van een nieuwe gecentraliseerde basis voor hun administratieve software.



Het advies bestaat uit een aantal verschillende onderdelen, in figuur 15 staan deze onderdelen. Hierbij ligt de focus op de back-end en is in de scope van het afstuderen de front-end een proof of concept.

Figuur 15 - Software structuur

Voor de basisstructuur wil Jonker geadviseerd worden welke types databases er tegenwoordig worden aangeboden en welke het best geschikt zijn voor de data die ze verwerken.

De nog op te zetten service laag moet uiteindelijk te koppelen zijn met meerdere soorten clients, waaronder mobiele apps, standalone en web clients. Er zal onderzocht moeten worden welke communicatiemethoden er gebruikt kunnen worden en wat de voor en nadelen van elk zijn. Daarnaast zal bekeken moeten worden hoe de business logic die nu nog in de clients zelf staat gescheiden kan worden in de service laag en de clients.

Ook is het gewenst voor Jonker geadviseerd te worden over de mogelijkheden van een persistence laag die goed samenwerkt met de service laag en de gekozen database. Jonker wil graag weten hoe een aantal complexe query’s die ze nu gebruiken uitgevoerd zouden kunnen worden in de persistence frameworks die er op de markt zijn, ook willen ze graag weten wat de andere voor en nadelen zijn van het gebruik van deze persistence frameworks.

Voorheen werd alleen door de klant op eigen lokale server ingelogd. De huidige manier van inloggen moet worden herzien om dit via het internet bereikbaar te maken. Daarnaast moet worden onderzocht of het mogelijk is dat de gebruiker kan wisselen tussen de software pakketten van Jonker zonder dat de gebruiker opnieuw moet inloggen.

Uit elk van de bovenstaande adviezen wordt samen met Jonker een keuze gemaakt voor de nieuw te realiseren back-end.

Naast het geven van advies omvat de opdracht ook het maken van een proof of concept ondersteund met technische documentatie. In het proof of concept zal één van de softwarepakketten van Jonker worden omgezet, zodat deze gebruik maakt van de nieuwe basis.

## Hoofdvraag

In het onderzoek bestaat er één hoofdvraag die is opgedeeld in kleinere deelvragen die erbij helpen de hoofdvraag te beantwoorden. De vraag die centraal staat in het onderzoek luidt: “Hoe kan de huidig gedecentraliseerde back-end van Jonker’s administratiesoftware omgezet worden naar een gecentraliseerde omgeving die ontsloten wordt naar verschillende clients door middel van een generieke service laag?”

## Deelvragen

Er zijn een aantal deelvragen opgesteld, de antwoorden op deze vragen zullen helpen bij het beantwoorden van de hoofdvraag. De deelvragen luiden als volgt:

* Is de huidige database nog het meest geschikt voor een gecentraliseerde opzet van de administratiesoftware?
* Kunnen bestaande persistence frameworks voordelen opleveren voor de nieuwe back-end? Zo ja, op welke manier?
* Welke web services zijn er geschikt voor de nieuwe back-end en wat zijn de voor en nadelen van elk?
* Welke authenticatiemethode kan gebruikt worden om de klant remote te laten inloggen in het systeem, waarmee de klant ook kan wisselen tussen de software pakketten zonder opnieuw in te moeten loggen?
* Welke business logica uit het softwarepakket wat omgezet gaat worden naar de nieuwe back-end kan worden verplaatst in de service laag.
* Hoe kan er aangetoond worden dat het omgezette softwarepakket hetzelfde functioneert als voor de omzetting.

## Kwaliteitscriteria

In dit hoofdstuk worden een aantal maatregelen benoemd die betrekking hebben op de kwaliteitswaarborging van de producten en diensten die worden opgeleverd.

### Documenten

Documenten zullen pas definitief worden opgeleverd nadat deze grondig gecontroleerd zijn op correct Nederlands taalgebruik. spel- en taalfouten.

### Proof of concept

Om te kunnen waarborgen dat de functionaliteit van het softwarepakket wat wordt omgezet naar de nieuwe omgeving hetzelfde blijft, wordt er een unittest uitgevoerd op de nieuwe en de huidige versie van het pakket. Als deze test succesvol verlopen is, dan kan het proof of concept als af benoemd worden.

### Onderzoek

Een groot gedeelte van dit project zal bestaan uit onderzoek. Er worden daarom aantal afspraken gemaakt voor het doen van dit onderzoek.

Zo zal bij het zoeken naar informatie als eerst gezocht worden in de databanken die aangeboden worden door de Hogeschool Utrecht, dit zijn bijvoorbeeld Gartner (<http://www.gartner.com/>) en ACM (<http://dl.acm.org/>). De informatie uit deze databanken zijn namelijk een betrouwbare bron.

Als de benodigde informatie niet te vinden is via de databanken wordt er pas gezocht op de rest van het internet of in het beschikbare literatuur op de werklocatie. Als binnen het onderzoek informatie uit een externe bron wordt gebruikt, zal er worden gerefereerd naar de bron waar de betreffende informatie vandaan komt. Voor de bronvermelding wordt enkel gebruik gemaakt van APA richtlijnen (<http://itswww.uvt.nl/lis/es/apa/apa-handleiding.pdf>)

Bij het onderzoek wordt vooraf samen met Jonker bepaald welke onderdelen belangrijk zijn in de nieuwe omgeving, zodat een goede afbakening gemaakt kan worden voor wat in het onderzoek moet worden onderzocht en wat niet.

Verder wordt voor het onderzoek alleen vragen onderzocht die staan beschreven in hoofdstuk 4.2 en 4.3 en wordt er gebruik gemaakt van vooraf bepaalde onderzoeksmethoden zoals beschreven in hoofdstuk 5.2.

### Risico’s

In het plan van aanpak staan de mogelijke risico’s van dit project, om de risico’s te beperken zijn maatregelen opgesteld. Deze zijn te vinden in hoofdstuk 6.3.

## Op te leveren producten

Tijdens dit project dienen een aantal producten te worden opgeleverd aan de opdrachtgever. Hieronder worden deze producten benoemd.

### Onderzoeksrapport

Er zullen meerdere onderzoeken verricht worden die tezamen het advies voor Jonker moeten ondersteunen. Zo zal er onderzoek worden gedaan naar databases, persistence frameworks, service lagen en authenticatiemethodes. Elk van deze onderzoek zal worden opgeleverd in een apart document. De aanpak van deze onderzoeken worden bepaald in hoofdstuk 5.2.

### Proof of concept

Aan de hand van de adviezen zal Jonker een aantal technieken kiezen voor de nieuwe service laag. Er wordt binnen de tClick software één module gekozen die moet worden omgezet zodat deze gebruik maakt van de nieuwe service laag.

Bij voorkeur wordt de orders module omgezet, echter als de bedrijfsbegeleider inschat dat er te weinig tijd voor is kan er voor een kleinere module worden gekozen. Er moet een technische implementatie van de service laag worden gemaakt en opgeleverd. In de service laag moet de business logica van de gekozen softwaremodule aan te roepen zijn. De code van de service laag moet worden opgeleverd aan Jonker.

Daarnaast zal ook de standalone client code van de gekozen module moeten worden aangepast en opgeleverd. Technische documentatie

Om Jonker te ondersteunen bij het verdere gebruik van de nieuwe technieken wordt er een handleiding geschreven die de meeste voorkomende handelingen zal beschrijven. Voor verdere ondersteuning wordt er verwezen naar handleidingen van de technieken.

### Test rapport

Er wordt een unittest geschreven die de functionaliteit van de oude en nieuwe situatie zal gaan testen. Zowel het test plan, implementatie en de testresultaten worden aan Jonker geleverd.

## Projectgrenzen en randvoorwaarden

Om te voorkomen dat er onduidelijkheid ontstaat wat er niet of wel binnen dit project valt wordt in dit hoofdstuk een aantal projectgrenzen en randvoorwaarden benoemd die gelden binnen dit project.

* Het project zal zich alleen focussen op de administratieve software van Jonker. De andere software die ze aanbieden worden buitenwege gelaten.
* Het uitvoerend project lid zal enkel de in hoofdstuk 4.4 genoemde producten hoeven opleveren.
* Als er hardware (bijvoorbeeld een server) is vereist om het proof of concept te ontwikkelen, wordt dit door de opdrachtgever ter beschikking gesteld.
* Tijdens het project zal het uitvoerend project lid voldoende tijd krijgen om te werken aan een scriptie

# Aanpak

## Concept inhoudsopgave scriptie

Voorblad

Voorwoord

Samenvatting

Inhoudsopgave

1. **Inleiding**
2. **Achtergrond**
   1. Jonker
   2. Afstudeerder
3. **Opdrachtomschrijving**
   1. Probleem
   2. Opdrachtformulering
   3. Doelstelling
   4. Onderzoeksvragen
4. **Aanpak**
   1. Projectactiviteiten
   2. Methode van onderzoek
5. **Huidige situatiebeschrijving**
   1. Scopebepaling
   2. Database
   3. Client software
   4. Uitgevoerde tests huidige omgeving
6. **Onderzoeksresultaten**
   1. Database
   2. Persistence
   3. Service laag en authenticatie
   4. Client software
7. **Nieuwe situatiebeschrijving**
   1. Database
   2. Persistence
   3. Service laag en authenticatie
   4. Client software
8. **Proof of Concept**
   1. Technisch ontwerp
   2. Uitgevoerde tests
9. **Conclusie**
   1. Deelvragen
   2. Hoofdvraag
10. **Evaluatie**
11. **Literatuurlijst**
12. **Bijlagen**

## Onderzoeksmethoden

Om de doelstelling van dit project te halen zal er kwalitatief onderzoek uitgevoerd worden. Door een combinatie te gebruiken van een casestudie en regulier literatuurstudie zal de hoofdvraag en de deelvragen beantwoord kunnen worden.

Als eerst zal er een casestudie gedaan worden naar de huidige database, dit onderzoek moet achterhalen welke data er op dit moment in de tClick software wordt verwerkt. Hiervoor wordt eerst de opbouw van de database bekeken om zo te kunnen zien welke data er wordt opgeslagen, daarna wordt de source code van tClick gedownload en bekeken op welke manier met de database gecommuniceerd wordt.

Na het onderzoeken van de huidige database wordt er een gesprek met de begeleider gepland waarin samen de precieze requirements van de nieuwe omgeving zullen worden opgesteld. Door middel van literatuurstudie wordt bekeken of er tegenwoordig databases op de markt zijn die beter passen bij de requirements van Jonker dan de huidige database. Door te zoeken via google en databanken wordt eerst onderzoek gedaan naar de mogelijkheden van NoSQL databases. De resultaten worden getoond aan Jonker waarna een keuze wordt gemaakt of er een NoSQL of een relationele database wordt gebruikt. Aan de hand van deze keuze worden databases gezocht die aan de eisen van Jonker voldoen. Deze databases worden vergeleken met elkaar om zo van elke database de voordelen en nadelen voor Jonker op een rijtje te zetten.

Na het kiezen van een database kan er literatuurstudie gedaan worden om er achter te komen op welke manieren de service laag kan worden toegepast in de nieuwe omgeving. Eerst zal er gezocht worden op internet naar de verschillende mogelijkheden om een service laag te implementeren en de verschillende manieren van communiceren met de service laag. Deze informatie wordt gedocumenteerd, zodat een samen met Jonker bepaald kan worden hoe de service laag zal worden opgezet.

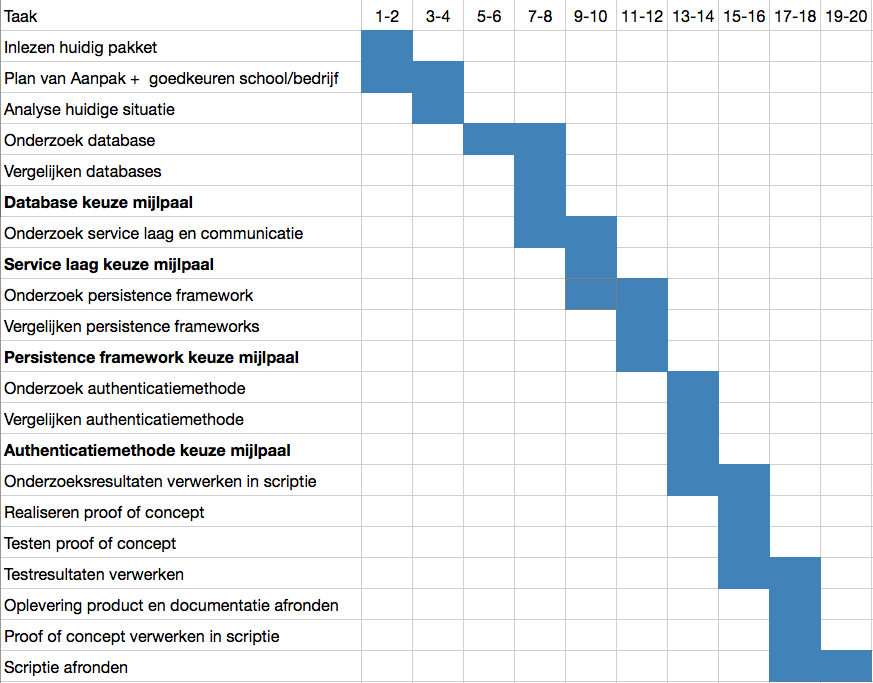
Als er is bepaald op welke manier de service laag wordt opgezet kan er gekeken worden naar een persistence laag passend bij de gekozen database en de service laag. Er wordt literatuuronderzoek gedaan naar welke verschillende persistence frameworks er bestaan die de nieuwe omgeving van Jonker kunnen ondersteunen. Door op internet te zoeken naar de voor en nadelen van de persistence frameworks kan er samen met Jonker een framework worden uitgekozen.

Een literatuurstudie naar mogelijke authenticatiemethodes moet daarna aantonen welke authenticatie kan worden gebruikt om de klanten van Jonker ingelogd te laten blijven tijdens het gebruik van tClick. Er wordt op internet gezocht naar authenticatiemethodes waarbij gelet wordt op de mate van veiligheid en de snelheid.

Nadat alle te gebruiken technieken zijn bepaald wordt er een proof of concept gemaakt waarbij de nieuwe technieken in een van Jonker’s software modules worden ingezet. Als eerst wordt de service laag geïmplementeerd op een lokale server, waarna business logica uit de software module gehaald en geplaatst in de service laag. De Java softwaremodule wordt aangepast en hierna wordt door een unittest test uit te voeren bekeken of het proof of concept nog dezelfde functionaliteit heeft als voor de omzetting.

## Globale planning

Hieronder in figuur 16 staat een globale planning van de taken binnen het project. De nummers bovenin geven de weeknummers aan waarin deze taak uitgevoerd zal worden. Als het vakje blauwgekleurd is betekent dat, dat er in deze weken aan de taak gewerkt wordt.



Figuur 15 - Globale planning

# Risicoanalyse

Bij de uitvoering van dit project zijn er factoren die een gevaar kunnen vormen voor het succesvol afronden van dit project. Om zo goed mogelijk in te kunnen spelen op deze mogelijke gevaren worden in dit hoofdstuk deze risico’s geanalyseerd en worden mogelijke tegenmaatregelen beschreven.

In het figuur 17 hieronder staan de mogelijke risico’s. Per risico worden twee cijfers gegeven op een schaal van 1 tot 5. Het eerste cijfer (kolom ‘kans’) geeft de kans aan dat het gevaar zich voordoet tijdens dit project. Het tweede cijfer (kolom ‘schade) geeft aan hoeveel schade het gevaar kan hebben op het succesvol afronden van dit project.

Door deze cijfers met elkaar te vermenigvuldigen zal zo per onderdeel een risicocijfer ontstaan. Dit risicocijfer geeft op een schaal van 1 tot 25 het totale risico aan. Alle risicocijfers worden daarna opgenomen in een risicomatrix, waar uit afgelezen kan worden of het een laag, middel of hoog risico is.

## Risico’s

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Gevaar** | **Kans** | **Schade** |
| A | Tijdelijke (meer als 2 weken) uitval van uitvoerend project lid. | 2 | 2 |
| B | Complete uitval van uitvoerend project lid | 1 | 5 |
| C | Conflicten binnen het bedrijf | 1 | 2 |
| D | Keuzes nieuwe situatie ingrijpender dan verwacht. | 3 | 3 |

Figuur 16 – Risicocijfers kans/schade

## Risicomatrix

In de risicomatrix die te zien is in figuur 18 staan de ID’s van de risico’s uit hoofdstuk 6.1. Door te kijken in welk vak een risico staat kan het risico niveau benoemd worden. Een groen vak betekend een laag risico niveau, een geel vak een middelmatig risico niveau en een rood vak een hoog risico niveau.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **SCHADE** | | | | |
|  | | **1** | **2** | **3** | **4** | **5** |
| **KANS** | **1** |  | **C** |  |  | **B** |
| **2** |  | **A** |  |  |  |
| **3** |  |  | **D** |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

Figuur 17 - Risico matrix

Hieronder in figuur 19 is staat een overzicht van alle risico’s samen met het risico niveau.

|  |  |  |
| --- | --- | --- |
| **ID** | **Gevaar** | **Risiconiveau** |
| A | Tijdelijke (meer als 2 weken) uitval van uitvoerend project lid. | Laag |
| B | Complete uitval van uitvoerend project lid | Laag |
| C | Conflicten binnen het bedrijf | Laag |
| D | Keuzes nieuwe situatie ingrijpender dan verwacht. | Middel |

Figuur 18 – Risico niveaus

## Tegenmaatregelen

In dit hoofdstuk wordt per risico beschreven welke maatregelen getroffen kunnen worden om de kans en/of schade van het risico te beperken.

### A - Tijdelijke uitval van uitvoerend project lid

Dit risico heeft een laag risico niveau, maar om toch nog de schade te beperken wordt er afgesproken dat uitvoerend project lid tijdig zijn afwezigheid doorgeeft aan zijn begeleider binnen het bedrijf. Zo mogelijk wordt ook een termijn doorgegeven voor terugkomst. Door dit tijdig te communiceren kan de begeleider passende acties ondernemen om de schade te beperken.

### B - Complete uitval van uitvoerend project lid

De kans is erg laag op een complete uitval, maar de schade is hoog. Er kunnen geen maatregelen worden genomen om of de kans, of de schade te beperken.

### C - Conflicten binnen het bedrijf

In het geval dat er zich een conflict voordoet tussen uitvoerend project lid en collega’s binnen het bedrijf, zal als eerst geprobeerd worden dit op te lossen door te praten met elkaar.

Als dit niet helpt, dan wordt er hulp ingeschakeld van de begeleider binnen het bedrijf die van beide partijen het conflict aanhoort, zodat hij het conflict kan analyseren en een passende oplossing kan bieden.

Als het echt niet lukt om het conflict op te lossen wordt als laatste middel contact opgenomen met de docentbegeleider op school.

### D - Keuzes nieuwe situatie ingrijpender dan verwacht

Tijdens het project worden een aantal technieken gekozen die worden gebruikt in het proof of concept, het is nog onbekend hoe moeilijk deze zijn te implementeren en hoe lang dat duurt. Er is een schatting gemaakt in de planning hoe lang het proof of concept ongeveer zal duren. Als uitvoerend project lid merkt dat het langer gaat duren zal er bespaard worden op het onderdeel wat voor Jonker het minst belangrijk is bij het proof of concept. Er wordt dan een kleinere module gekozen dat wordt omgezet voor het proof of concept.

# Gegevens betrokkenen

In dit hoofdstuk word een overzicht gegeven van de hoofdbetrokkenen in dit project en hun contactgegevens.

## Algemene informatie

Opdrachtgever: Jurrian Vlijm

Bedrijfsbegeleider: Marco Zieverink

Afstudeerder: Jochem Pouwels

Docentbegeleider: Rik Boss

## Contactgegevens

Naam: Jurrian Vlijm

Tel-nummer: 06 247 995 53

Email: [j.vlijm@jonker.nl](mailto:j.vlijm@jonker.nl)

Naam: Marco Zieverink

Tel-nummer: 06 247 996 40

Email: [m.zieverink@jonker.nl](mailto:j.vlijm@jonker.nl)

Naam: Jochem Pouwels

Tel-nummer: 06 519 130 51

Email: [jochem.pouwels@student.hu.nl](mailto:jochem.pouwels@student.hu.nl)

Naam: Rik Boss

Tel-nummer: 088 481 82 83

Email: [rik.boss@hu.nl](mailto:rik.boss@hu.nl)

## 14.2 Bijlage B – NoSQL onderzoeksresultaten



NoSQL Databases – Jonker BV.

Versie 1.0

Jochem Pouwels - 1613727  
Docentbegeleider: Rik Boss

Inhoudsopgave

1 Inleiding 76

2 Verschillende soorten databases 77

2.1 Scope 77

3 NoSQL databases 77

4 Algemene voordelen NoSQL database 78

4.1 Schaalbaarheid 78

4.1.1 Horizontaal schalen in NoSQL databases 79

4.1.2 Horizontaal schalen in relationele databases 79

4.2 Snelheid 80

4.3 Flexibiliteit 80

4.4 Simpliciteit 80

5 Algemene nadelen NoSQL databases 81

5.1 Beperkte query mogelijkheden 81

5.2 Geen tot weinig standaarden 81

6 Graph databases 82

6.1 Wat is een graph database? 82

6.2 82

6.2 Werking van een graph database 83

6.2.1 Query in een graph database 84

6.2.2 Query in een relationele database 85

6.3 Wanneer kan een graph database handig zijn 85

7 Key-value stores 86

7.1 Wat is een key-value store? 86

7.2 Werking key-value stores 86

7.3 Wanneer kan een key-value store handig zijn 86

8 Document stores 87

8.1 Wat is een document store? 87

8.2 De werking van een document store 87

8.3 Wanneer kan een document store handig zijn? 88

9 Literatuurlijst 88

# Inleiding

Dit document bevat het onderzoek naar NoSQL databases voor de opdrachtgever Jonker BV. Als eerst wordt in hoofdstuk 2 een introductie gegeven over verschillende soorten databases, daarna volgt in hoofdstuk 3 in beschrijving van het begrip NoSQL. De twee hoofdstukken hierna benoemen de algemene voor- en nadelen van deze type databases, waarna in hoofdstuk 6,7 en 8 verder op de types wordt ingegaan.

Voor het lezen van dit document gaat men ervanuit dat op z’n minst basiskennis over relationele databases aanwezig is.

# Verschillende soorten databases

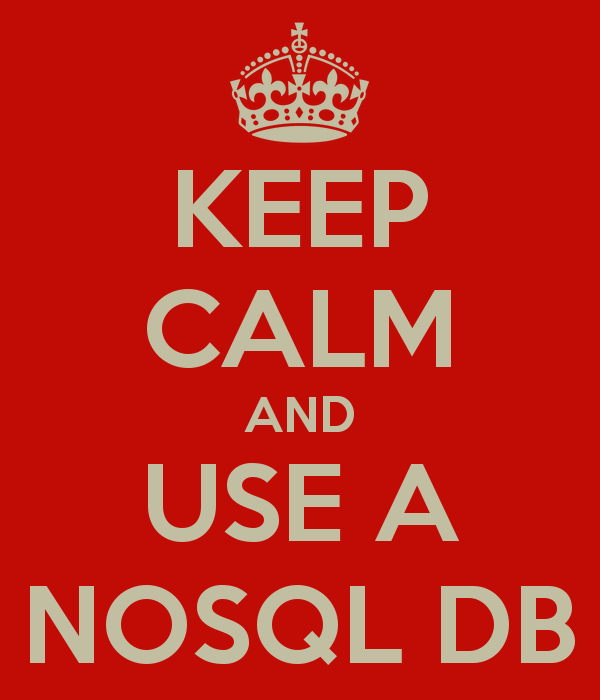
In de wereld van dataopslag bestaan er vele verschillende soorten databases. Elke soort heeft zijn eigen voor en nadelen en wordt voor verschillende doeleinden gebruikt. Zo heb je relationele databases, die al jaren lang voor bijna elk opslagprobleem gebruikt werden, maar heb je tegenwoordig ook NoSQL databases die sterk in populariteit stijgen. Dit onderzoek duikt dieper in op de definitie van een NoSQL database. Ook wordt bekeken voor welke dataopslagproblemen NoSQL databases beter geschikt zijn dan relationele databases.

## Scope

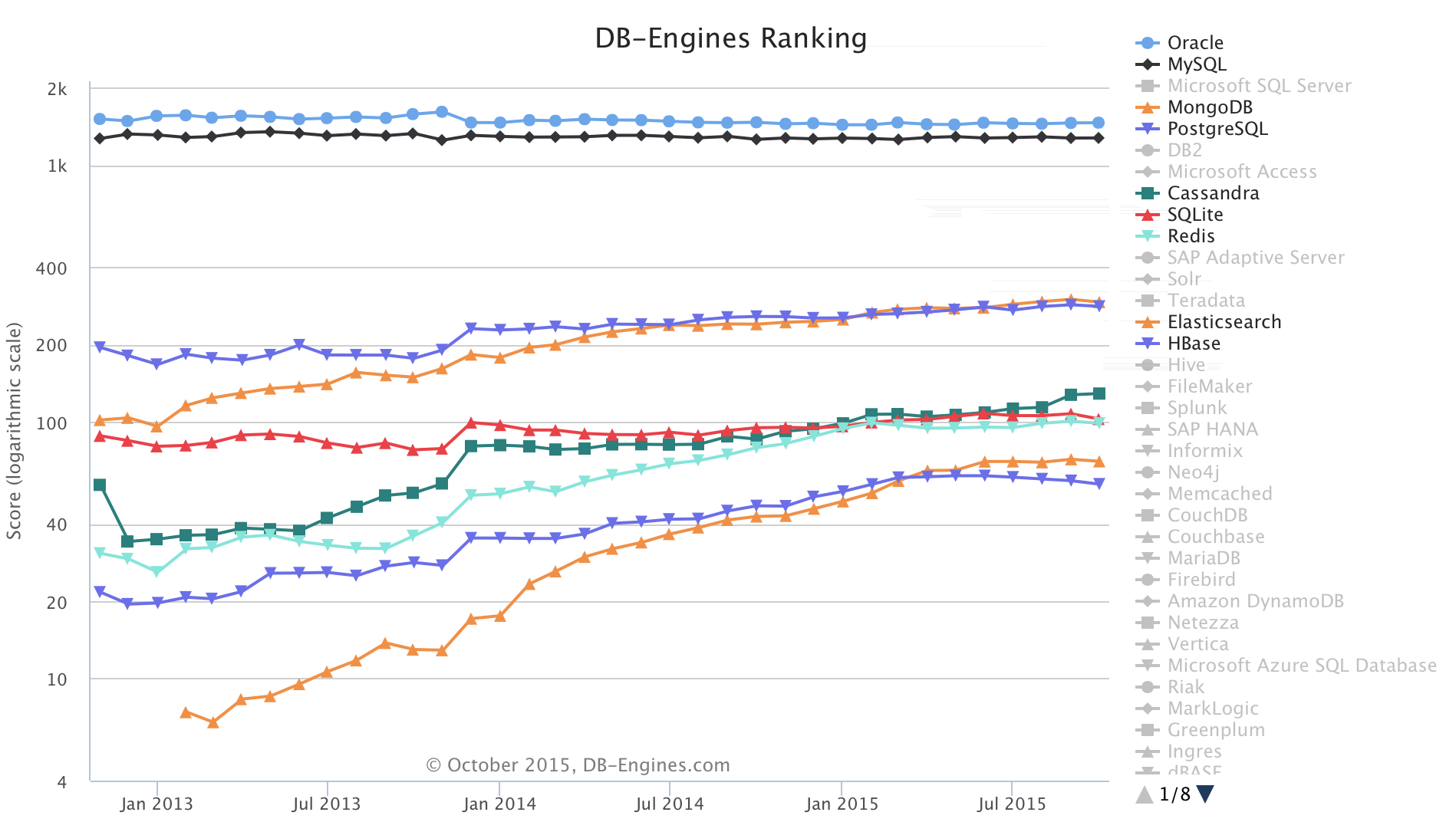
Het onderzoek beperkt zich op de drie types NoSQL databases waar Jonker meer over wou weten. De keuze voor deze drie is gemaakt nadat ze zelf een korte analyse op het internet hebben verricht over NoSQL.

De andere types NoSQL databases worden buitenwege gelaten.

# NoSQL databases

NoSQL staat voor “Not Only SQL” en is een groep van database-oplossingen die zich onderscheidt van de relationele databases. Volgens onderzoek van (christof-strauch.de, 2011) zijn NoSQL databases zijn ontstaan doordat velen gingen twijfelen aan SQL databases als magische oplossing voor elk dataopslagprobleem. Zo zijn er meerdere type oplossingen tot stand gekomen die specifieke opslagproblemen beter proberen op te lossen dan dat relationele databases dat doen.

De opkomende populariteit is hieronder goed te zien in een overzicht van (Ranking Trend, 2015)

****

Figuur 20 – Database ranking: NoSQL/Relationeel

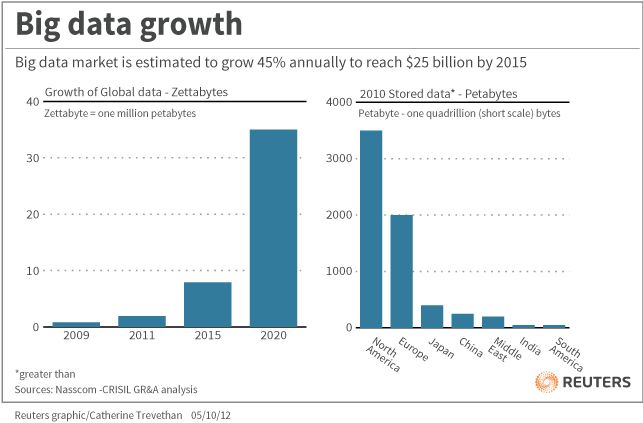
Iedere oplossing heeft zo zijn eigen kenmerken en eigen manier van data opslaan. Een aantal voor en nadelen hebben de meeste NoSQL databases gemeen.

In het hoofdstuk hierna volgen de algemene voor en nadelen van NoSQL databases en daarna wordt per type NoSQL database een nadere uitleg gegeven.

# Algemene voordelen NoSQL database

## Schaalbaarheid

Een veel voorkomend obstakel bij databases is de schaalbaarheid. Het datavolume en het dataverkeer op hedendaags internet blijft groeien en snel ook. Dit is te zien in onderstaand diagram uit onderzoek van (Koutrik, 2014)



Figuur 21 – Verwachting groei big data

Er wordt steeds meer data bijgehouden van steeds meer gebruikers, waardoor ook het gemiddelde aantal schrijf/lees transacties per seconde voor een database sterk zijn gestegen. Om deze groei van database gebruik bij te kunnen houden kan het bij databases nodig zijn om goed te kunnen schalen. NoSQL databases kunnen dit vaak erg goed, omdat ze ontworpen zijn met horizontaal schalen als een van de prioriteiten.

Het schalen van een database kan men op twee manieren:

**Verticaal schalen:** Het vergroten van de capaciteit van de database door betere hardware te plaatsen in een bestaande server. Zo kan er bijvoorbeeld een snellere CPU of een grotere harde schijf geplaatst worden.

**Horizontaal schalen**: Het vergroten van de capaciteit van de database door het koppelen van één of meerdere servers. De dataopslag en/of het dataverkeer kan dan worden verdeeld over de meerdere servers. Een database dat bestaat uit meerdere servers wordt een database cluster genoemd.

Verticaal schalen is in de meeste gevallen mogelijk, omdat het in principe niks meer is als het upgraden van de hardware. Wel moet de database efficiënt kunnen omgaan met bijvoorbeeld meerdere cores of meer RAM geheugen. Horizontaal schalen is vaak wat complexer omdat de structuur of opzet van de database ook moet worden aangepast om te werken met meerdere servers. Dit is nou net waar de meeste NoSQL databases het je wat makkelijker maken.

Hieronder worden de mogelijkheden tot horizontaal schalen uitgelegd voor relationele en NoSQL databases, om zo het verschil tussen deze twee te verklaren.

### Horizontaal schalen in NoSQL databases

Een nieuw gekoppelde server kan in een NoSQL database cluster op twee manieren worden opgezet. Hierbij horen twee termen, replicatie en sharding.

Als een nieuwe server d.m.v. replicatie wordt gekoppeld. Wordt alle data op de hoofdserver automatisch naar de replica server gesynchroniseerd. De nieuwe server kan daarna worden gebruikt als back-up en/of om het aantal read query’s wat de rest van het cluster moet verwerken te reduceren.

Een nieuwe kan ook d.m.v. sharding gekoppeld worden. Met sharding is het mogelijk om de records van één tabel verdeeld op te slaan over meerdere servers. Dit heeft twee voordelen, de eerste voordeel is dat write query’s verdeeld kunnen worden over de servers binnen het cluster om de snelheid te verbeteren. Het tweede voordeel is dat de dataopslag capaciteit van tabellen wordt vergroot, wat belangrijk kan zijn als data van een tabel niet meer op één enkele server past.

Door de non-relationele datastructuur in NoSQL databases hebben veel van deze databases de mogelijkheid sharding heel makkelijk toe te passen. Dit hoeft enkel op de database ingesteld te worden waarna de applicatie hier geen rekening mee hoeft te houden. Doordat data in een NoSQL data vaak gedenormaliseerd is, hoeft er helemaal niks in de structuur van de database te veranderen om van de voordelen van sharding gebruik te maken.

### Horizontaal schalen in relationele databases

Het principe van horizontaal schalen is bij relationele databases hetzelfde als bij NoSQL databases. Zo wordt bij replicatie read query’s geschaald en bij sharding de write query’s en de capaciteit. Echter zijn relationele databases niet gebouwd met horizontaal schalen als prioriteit, daardoor schalen ze vaak minder makkelijk als een NoSQL database, dit is vooral te merken bij het sharden.

Het is meestal een complex proces om sharding te implementeren in relationele databases, vooral in grote databases met veel relaties. Het tabelschema moet namelijk speciaal ontworpen worden om gebruik te kunnen maken van shards. Hierbij van tevoren bepaald worden welke range van records in welke shard komen te staan waarbij ook nog eens nagedacht moet worden dat data gerelateerd aan deze records ook op dezelfde shard komen te staan. Naast dat shard logica in het ontwerp van de database komt te staan, moet deze logica ook bekend zijn in de applicatie die gebruik maakt van de database.

Een voorbeeld van sharding in een relationele database is het scheiden van klanten door op de ene shard een tabel met alle Amerikaanse klanten te zetten en op de andere shard een met alle Europese klanten. De applicatie zal moeten weten welke data het verwerkt, en dus connectie maken met de juiste shard.

## Snelheid

De verschillende opslagmethodes van NoSQL databases kunnen in bepaalde situaties ervoor zorgen dat data sneller en efficiënter opgeslagen en/of opgehaald kan worden.

NoSQL databases gebruiken verschillende opslagmethodes, de een is geoptimaliseerd om snel complexe relaties tussen data te zoeken, en de ander kan bijvoorbeeld data gerelateerd aan elkaar hiërarchisch opslaan om zo efficiënter te kunnen query’en ten opzichte van een relationele database.

De diverse opslagmethoden van NoSQL databases bieden verschillende oplossingen die soms beter passen bij het opslagprobleem. Om erachter te komen of een opslagprobleem beter past bij een NoSQL database zal elk type beter moeten worden geanalyseerd, deze analyses zijn te vinden in hoofdstuk 6 t/m 9.

## Flexibiliteit

In relationele databases kan vaak alleen data geplaatst worden met een vooraf bepaalde structuur welke wordt aangegeven door een tabelschema. In hoofdstuk 4.1.2 is al verteld welk nadeel dit geeft voor database replicatie. Dit is echter niet het enige nadeel. In een database waar de datastructuur nog wel eens wil veranderen moeten steeds migraties worden uitgevoerd door de beheerder, om zo de al bestaande data aan te passen aan de nieuwe structuur. Dit kan uiteindelijk een flink aantal manuren gaan kosten.

NoSQL databases hebben in tegenstelling tot relationele databases vaak een flexibele datastructuur. Wil je bijvoorbeeld van de nieuw ingevoerde klanten ook het telefoonnummer bijhouden? Dan kan dat zonder dat de oude klant gegevens in de database geüpdatet moeten worden. Zo zorg je er ook voor dat de database niet vol staat met “NULL” kolommen, welke ook ruimte innemen op het opslagsysteem van de database.

Het gebruiken van een NoSQL databases kan bij de juiste usecases zorgen voor een snellere ontwikkeling en dus ook lagere ontwikkelingskosten.

## Simpliciteit

NoSQL databases zijn niet altijd ontworpen voor een breed scala aan opslagproblemen, maar focussen zich vaak op een specifiek probleem. Hierdoor kunnen ze zich wel optimaliseren voor dit probleem.

Dit is vaak terug te zien in de API. Deze zijn gelimiteerd, maar daarom vaak wel simpeler te gebruiken. Ook worden veel van de API’s aangeboden in één of meerdere native talen (bijvoorbeeld Java, Ruby of C#) wat de leercurve nog verder verkleind voor developers die object georiënteerd programmeren gewend zijn.

Het kiezen van de juiste database voor een opslagprobleem kan ervoor zorgen dat de ontwikkelaar minder tijd nodig heeft om dezelfde data op te halen of te schrijven.

# Algemene nadelen NoSQL databases

## Beperkte query mogelijkheden

De simpliciteit van de NoSQL databases heeft zo ook z’n nadelen. Het betekend namelijk ook dat de query functionaliteit gelimiteerd is ten opzichte van vele relationele databases. Waar in een relationele database de mogelijkheden eindeloos lijken, is dat in een NoSQL database niet zo.

Voor een complexe omgeving is het dus bijna altijd nodig om meerdere NoSQL databases door elkaar te gebruiken of een combinatie te gebruiken van een NoSQL en een relationele database. Dit maakt het moeilijker om het geheel te beheren.

## Geen tot weinig standaarden

Relationele databases implementeren allemaal tot op een zekere hoogte de SQL standaard. Hierdoor is het relatief gemakkelijk om op een later moment over te stappen naar een andere relationele database met kleine aanpassingen.

NoSQL databases daarentegen hebben elk hun eigen unieke methodes om hun data te beheren. Het niet implementeren van een standaard voor het query’en van de database maakt het lastig om over te stappen naar een nieuwe aanbieder.

# Graph databases

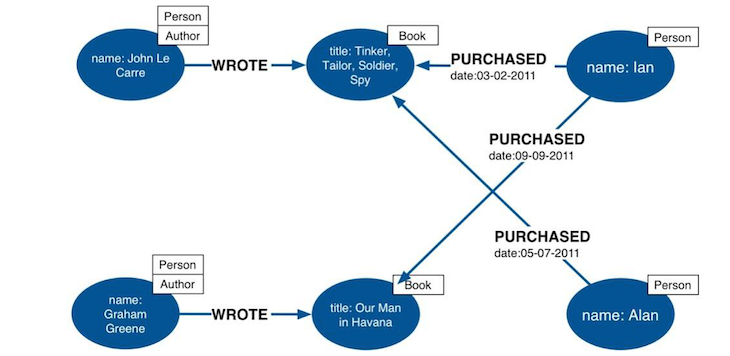
In dit hoofdstuk wordt de werking van de graph database behandeld. Ook wordt uitgelegd in welke situaties zo’n graph database handig kan zijn.

## Wat is een graph database?

Een graph database wordt opgebouwd uit **nodes, relaties** en **properties**.

Nodes zijn te vergelijken met de objecten zoals deze bekend staan bij de object georiënteerde programmeurs, het zijn entiteiten van een specifiek type, met elk zijn eigen eigenschappen (properties). Een voorbeeld is de node van het type: “Acteur” met de properties “naam” en “geboortejaar”.

Dan heb je nog de relaties, dit zijn de verwijzingen tussen de nodes. Zo kan de “Acteur” node bijvoorbeeld een relatie hebben naar een andere “Acteur” node met de relatie genaamd: “Collega”. Of een relatie naar een “Film” node met de relatie genaamd: “Speelt in”. Elke relatie heeft ook de mogelijkheid om properties aan zichzelf toe te wijzen, zo kan er bij de “Speelt in” relatie bijvoorbeeld een property “jaar” worden opgeslagen. Hieronder staat een figuur waarop de datastructuur van een graph database te zien is. Dit figuur is afkomstig van (Neo4j)



## Werking van een graph database

Figuur 22 - De structuur van een graph database

Een query in een graph database begint altijd met het zoeken van een start-node. Waarbij een node vaak te vergelijken is met een rij in een tabel van de relationele database. Zo is bijvoorbeeld elke rij van tabel “Acteurs” een Acteur node.

Nadat een start-node of meerdere start-nodes zijn gevonden, kan er in de query worden aangegeven welk relaties gevolgd moeten worden.

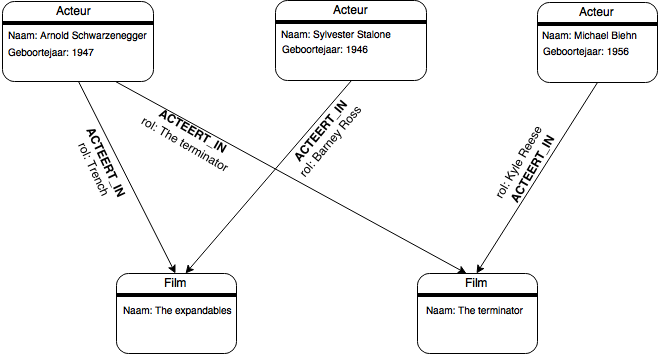
Om een beter beeld te geven hoe dit te werk gaat, staat in dit hoofdstuk beschreven hoe een bepaalde usecase wordt geïmplementeerd in een graph database in vergelijking met een relationele database. De usecase is in dit geval een kleine film database, waar we de volgende data uit willen halen.

“De namen van films waarin een acteur speelt die in 1947 geboren is”.

Figuur 22 toont de benodigde data voor de usecase in een relationele database. Figuur 23 toont dezelfde data maar dan in een graph database.



Figuur 19 – Usecase: film database - Relationele database



Figuur 20 – Usecase: film database - Graph database

In de figuren is het verschil in datastructuur duidelijk te zien. In de relationele database wordt alle data opgeslagen in tabellen en in de graph database staat alle data in de vorm van nodes die verbonden zijn met relaties. In de subhoofdstukken hierna wordt toegelicht hoe beide databases te werk gaan.

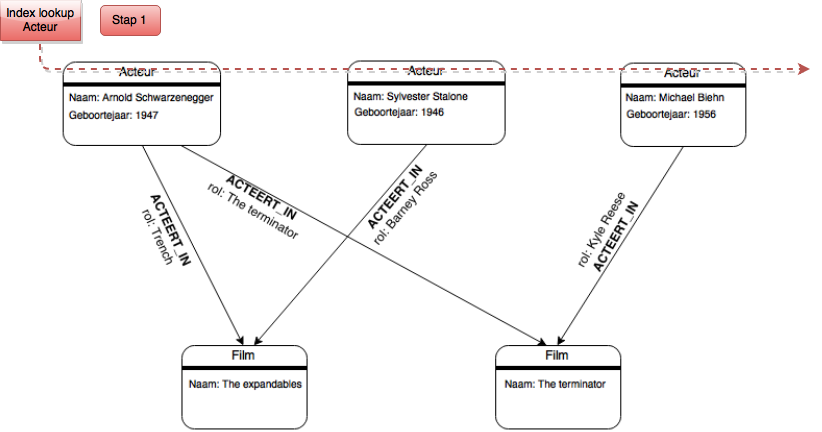
### Query in een graph database

Hieronder wordt eerst de query getoond die gebruikt kan worden om de data van de usecase op te halen, in figuur 24 daaronder wordt visueel weergeven welke stappen de graph database onderneemt.

MATCH (actor:Acteur)-[:ACTS\_IN]->(film)

WHERE actor.geboortejaar = 1947

RETURN film.naam



Figuur 21 - Query graph database

Een de query hierboven wordt eerst gezocht naar een startnode of meerdere startnodes, dit is iets waar elke query mee begint in een graph database. In dit geval is dat acteurs met geboortejaar 1947. Vanaf hier kunnen relaties gevolgd worden om data gerelateerd aan deze node op te zoeken. In de huidige usecase heeft een graph database al toegang tot de data die nodig is na één stap. In stap 1 wordt er gezocht naar een acteur uit 1947. Als deze gevonden is, is er meteen een directe verwijzing naar de Film node. Relaties zijn namelijk directe verwijzingen naar de opslaglocatie van de end-node.

### Query in een relationele database

Hieronder wordt eerst de query getoond die gebruikt kan worden om de data van de usecase op te halen, in figuur 25 daaronder wordt visueel weergeven welke stappen de relationele database onderneemt.

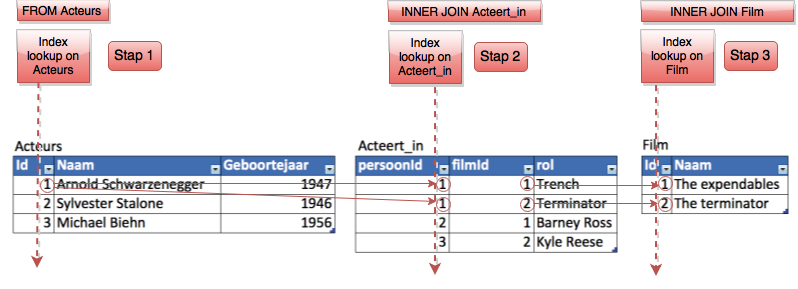
SELECT f.naam

FROM Acteurs a

INNER JOIN Acteert\_in act ON act.persoonId = a.id

INNER JOIN Film f ON act.filmId = f.id

WHERE a.geboortejaar = 1947



Figuur 22 - Query relationele database

Voor een optimale query in de relationele database gebruiken we een index op de “geboortejaar“ kolom in de Acteurs tabel, een index op de “persoonId” kolom in de Acteert\_in tabel en een index op de “id” kolom in de Film tabel.

Een relationele database heeft drie stappen nodig om de data op te halen. In stap 1 zal eerst de acteurs tabel moeten doorzocht worden op acteurs met een geboortejaar gelijk aan 1947. Als deze gevonden zijn, wordt in stap 2 de Acteert\_in tabel doorzocht om te zoeken naar records waarbij de kolom persoonId gelijk is aan de id van de acteur(s) gevonden in stap 1. Daarna wordt in stap 3 elke rij van de films tabel bekeken om te zoeken naar films die horen bij de id’s die net in stap 2 gevonden zijn.

## Wanneer kan een graph database handig zijn

Het zoeken van een start-node is vaak wat minder snel als met indexes die gebruikt worden in relationele database. Maar door de directe relaties tussen nodes is het vinden van gerelateerde data wel een stuk sneller dan de joins in een relationele database. Dit voordeel wordt groter naarmate er steeds meer verschillende relaties worden toegevoegd. In een relationele database zou hiervoor steeds een nieuwe koppeltabel voor moeten worden gemaakt.

Daarnaast is het een gemak dat alle relaties en nodes flexibel kunnen worden toegevoegd.

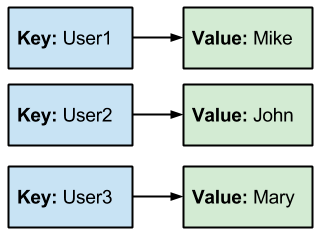
Als de data in de database veelal bestaat uit koppelingen en wil je deze data flexibel toevoegen, dan is het handig om te kijken naar een graph database.

# Key-value stores

In dit hoofdstuk wordt de key-value store behandeld. Op het eind wordt uitgelegd in welke situaties de key-value store handig is in te zetten.

## Wat is een key-value store?

Figuur 26 - Structuur van een key-value store

Een key-value store is een van de meest simpele manier om iets op te slaan, ook een van de snelste. De database bestaat enkel uit een verzameling van unieke keys met bijhorende values. te vergelijken met de HashMap van Java. (<https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>)

Hiernaast staat een figuur dat de datastructuur van een key-value store laat zien.

## Werking key-value stores

Het verschil met relationele databases is dat bij key-value stores het niet uit maakt hoeveel data er in de database staat, elke schrijf en lees query duurt dezelfde (korte) tijd. De data in een key-value store kan alleen opgehaald of aangepast worden door direct te zoeken op “key”, deze moet dus wel bekend zijn bij de gebruiker van de database.

De waardes die de key-value store kan opslaan verschilt per aanbieder. Zo kan de ene alleen strings opslaan en heeft de ander meerdere formaten zoals arrays of lijsten. Voor het updaten van waardes worden vaak passende methoden aangeboden in de API. Zo hebben lijsten functies zoals pop en push, en hebben arrays de mogelijkheden om velden in de array apart te wijzigen.

## Wanneer kan een key-value store handig zijn

Door de eenvoud van key-value stores zijn ze enorm schaalbaar, zeer snel en ook erg flexibel. Echter vereist dit wel dat de data die je wil opslaan altijd geïdentificeerd kan worden met een key. Voorbeelden van keys zijn usernames of sessionid’s.

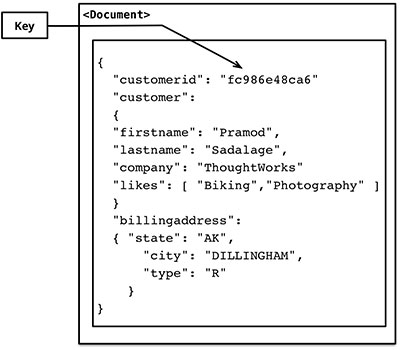
Bij het kiezen van een key-value store moet er wel gelet worden op dat er verschillende vormen bestaan. Zo zijn er key-value stores die enkel in het RAM geheugen van de server werken, maar zijn er ook key-value stores die data opslaan op bijvoorbeeld een SSD. Het verschil tussen deze twee is dat werken in het RAM geheugen nog sneller gaat als op een SSD, maar de data wel verloren gaat bij bijvoorbeeld een herstart van de server.

# Document stores

In dit hoofdstuk wordt een document store behandeld. Op het eind wordt uitgelegd wanneer dit soort database goed zijn in te zetten.

## Wat is een document store?

Een document store slaat data op in een datastructuur dat vergelijkbaar is met het populaire JSON formaat, de document. Een document bestaat uit een collectie van key-value paren waar de value van de key mogelijk weer bestaat uit een document. Dit betekend dat documents hiërarchisch in elkaar kunnen worden geplaatst. De root (dit is het document dat niet in een ander document is geplaatst) moet wel uniek identificeerbaar zijn door middel van een key. In het figuur hieronder staat een visuele weergave van de datastructuur van een document.



Figuur 23 - Datastructuur van een document

## De werking van een document store

In grote delen lijkt de opslag van een document store op dat van een relationele database.

* Documents wordt opgeslagen in een collection, wat in feite hetzelfde is als een tabel in een relationele database.
* Als een collection een tabel is, dan is de document dus eigenlijk een “rij”
* Een document heeft meerdere velden, deze lijken dan weer erg op de kolommen

Het grote verschil is dat documents hiërarchisch in elkaar kunnen worden geplaatst. Rijen in een tabel kunnen dat niet. Het relationele model heeft echter wel een functie die hier erg op lijkt: de join.

Een hiërarchisch document kan gezien worden als een vooraf gejoinde tabel.

## Wanneer kan een document store handig zijn?

Een document store kan handig zijn in situaties waar data dat gerelateerd aan elkaar is, ook altijd samen moet worden opgehaald. Dit scheelt namelijk het steeds moeten joinen van tabellen.

Een nadeel is echter wel dat de data binnen een document niet meer los te gebruiken is, bijvoorbeeld in de situatie waar “reactie” documents binnen een “forumBericht” document is geplaatst. Het ophalen van een forum bericht gaat nu erg makkelijk en snel, maar wil je statistieken bijhouden van hoeveel reacties een bepaalde gebruiker heeft geplaatst? Dan kan dat niet meer.

Of een document store handig is, wordt dus bepaald door de usecase.

# Literatuurlijst

Koutrik, A. v. (2014, 08 5). *Wat is big data?* Opgehaald van intrapreneur.nl: https://www.intrapreneur.nl/2014/08/05/wat-is-big-data/

Neo4j. (sd). *What is a graph database?* Opgehaald van Neo4j.com: http://neo4j.com/developer/graph-database/

Oliver, A. C. (2013, 08 01). *What are graph databases good for?* Opgehaald van infoworld.com: http://www.infoworld.com/article/2611503/application-development/what-are-graph-databases-good-for--here-s-a-killer-app.html

Bryden, J. (sd). *Graph Databases Explained*. Opgehaald van nosqlguide.com: http://nosqlguide.com/graph-database/nosql-databases-explained-graph-databases/

olemiss. (2010, 04 15). Opgehaald van olemiss.edu: http://cs.olemiss.edu/~ychen/publications/conference/vicknair\_acmse10.pdf

Flux7 Labs. (sd). *NoSQL database solutions*. Opgehaald van flux7.com: http://blog.flux7.com/blogs/nosql/nosql-database-solutions-types-and-examples

MongoDB. (2015, 04 15). *Thinking in Documents*. Opgehaald van mongodb.com: https://www.mongodb.com/blog/post/thinking-documents-part-1

MongoDB. (2014, 02 26). *Documents - Introduction*. Opgehaald van mongodb.com: https://docs.mongodb.org/manual/core/introduction/

Rebelic. (2011, 05 28). *The four categories of NoSQL databases*. Opgehaald van rebelic.nl: http://rebelic.nl/2011/05/28/the-four-categories-of-nosql-databases/

Lerman, J. (2011, 11 01). *What the Heck Are Doucment Databases*. Opgehaald van microsoft.com: https://msdn.microsoft.com/en-us/magazine/hh547103.aspx

## 14.3 Bijlage C – Relationele databases onderzoeksresultaten



Relationele databases – Jonker BV.

Versie 1.0

Jochem Pouwels - 1613727  
Docentbegeleider: Rik Boss

Inhoudsopgave

1 Inleiding 92

2 Introductie 93

3 Populariteit 93

4 Oracle 95

9.1 Oracle Database 11g Express Edition (XE) 95

4.1.1 Licenties & kosten 95

4.1.2 Server restricties 95

4.1.3 Replicatie mogelijkheden 95

4.1.4 Uitgelichte features 95

9.2 Oracle Database 12c Standard Edition One (SE1) 96

4.1.5 Licenties & kosten 96

4.1.6 Server restricties 96

4.1.7 Replicatie mogelijkheden 96

4.1.8 Uitgelichte features 96

5 MySQL 97

9.3 MySQL Community Edition (CE) 97

5.1.1 Licenties & kosten 97

5.1.2 Replicatiemogelijkheden 97

5.1.3 Uitgelichte features 97

9.4 MySQL Standard Edition (SE) 97

5.1.4 Licenties & kosten 97

5.1.5 Replicatiemogelijkheden 97

5.1.6 Uitgelichte features 98

6 PostgreSQL 98

9.5 Licenties & kosten 98

9.6 Replicatiemogelijkheden 98

9.7 Uitgelichte features 98

7 DB2 99

9.8 DB2 Express-C (EX-C) 99

7.1.1 Licenties & kosten 99

7.1.2 Server restricties 99

7.1.3 Replicatiemogelijkheden 99

7.1.4 Uitgelichte features 99

9.9 DB2 Express (EX) 100

7.1.5 Licenties & kosten 100

7.1.6 Server restricties 100

7.1.7 Replicatiemogelijkheden 100

8 Firebird 101

9.10 Licenties & kosten 101

9.11 Replicatie 101

9.12 Uitgelichte features 101

9 Voor & nadelen per database 102

9.13 Oracle Express Edition 102

9.14 Oracle Standard Edition 102

9.15 MySQL Community Edition 103

9.16 MySQL Standard Edition 103

9.17 PostgreSQL 103

9.18 DB2 Express–C 104

9.19 DB2 Express 104

9.20 Firebird 104

10 Literatuurlijst 105

11 Bijlages 106

9.21 Bijlage A – Oracle functionaliteiten per versie 106

# Inleiding

De originele relationele database ontworpen in 1970 (IMB100) wordt hedendaags aangeboden in verschillende varianten. Elk met hun eigen functionaliteiten en eigenaardigheden. Al vanaf het begin zijn meerdere bedrijven aan de gang gegaan om een eigen versie van de relationele database te ontwikkelen en te verkopen. Tegenwoordig zie je soms door de bomen het bos niet meer. Volgens de site <http://db-engines.com/> zijn er tegenwoordig in ieder geval 107 aanbieders.

Om de keuze makkelijker te maken is er onderzoek gedaan naar de verschillende databases. Hierbij is niet elke variant in behandeling genomen, maar alleen de databases van de vier populairste aanbieders. Dit onderzoek wordt uitgevoerd in opdracht van Jonker, daarom is ook de huidige database meegenomen in het onderzoek.

Als eerst wordt in hoofdstuk 2 een korte introductie gegeven over welke aspecten van databases onderzocht zullen gaan worden. Daarna wordt in hoofdstuk 3 bekeken wat de populairste databases zijn, deze zullen in de hoofdstukken hierna verder worden onderzocht. Het laatste hoofdstuk zet alle voor en nadelen nog eens op een rijtje.

# Introductie

In het onderzoek wordt voor elke databases de volgende aspecten onderzocht.

**Licentie kosten**

Sommigen kosten geld, de ander is gratis te gebruiken. Per database zal het kostenmodel van de database verklaard worden, zodat een duidelijk overzicht ontstaat wat het gebruik van de database je gaat kosten.

**Server restrictie**

Vooral als je kijkt naar de goedkope of gratis versies van een database zijn er vaak restricties bij het gebruik van een database. Er is dan bijvoorbeeld een maximale opslagcapaciteit of de specificaties van de server mogen niet te hoog zijn. Deze restricties worden overzichtelijk gemaakt in dit onderzoek.

**Replicatie mogelijkheden**

Elke database heeft zo zijn eigen manier om data te repliceren. Om de mogelijkheden inzichtelijk te maken is van elke database een beschrijving toegevoegd van de replicatiemogelijkheden.

**Uitgelichte features**

Binnen de scope van het onderzoek is het niet rendabel om alle functionaliteiten van de database in kaart te brengen. Toch worden er per database een aantal functionaliteiten beschreven, welke functionaliteiten dit zijn wordt bepaald op verwachte relevantie voor Jonker.

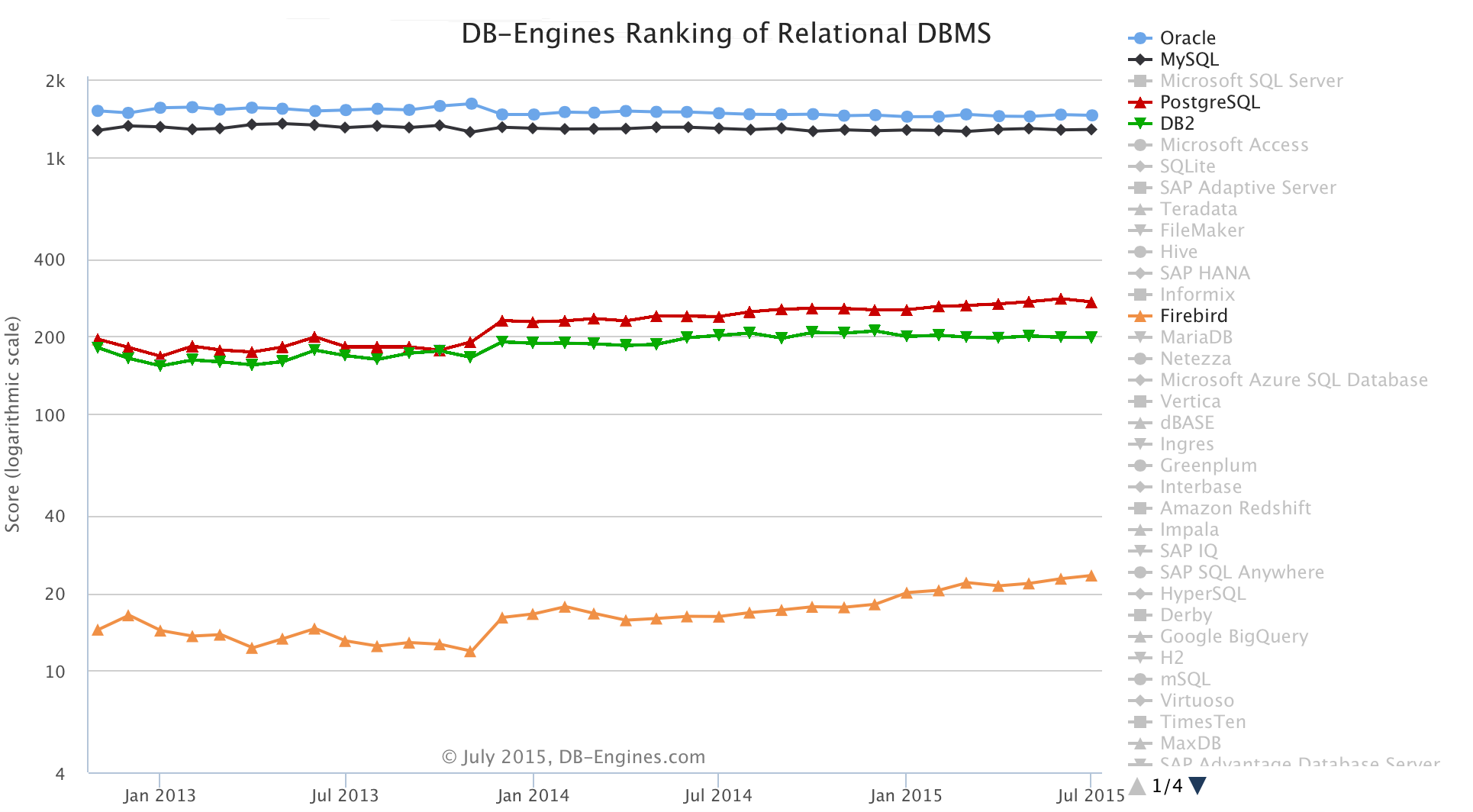
# Populariteit

Een belangrijk requirement die Jonker stelt aan de database is dat er genoeg tools beschikbaar zijn. De mate van populariteit wordt voor dit onderzoek gebruikt om vooraf in te schatten bij welke databases dit het geval is. Ervanuit gaande dat een grote community ook betekend dat er in ieder geval genoeg tools zijn.

In dit onderzoek worden de vier populairste op Linux draaiende databases en ook de huidige Firebird database van Jonker onderling vergeleken.

De populairste databases zijn bepaald door gebruik te maken van de website <http://db-engines.com/>, dit is een website die elke database een score geeft op basis van populariteit. De scores worden opgebouwd door middel van een algoritme dat gebruik maakt van informatie die publiekelijk te vinden is in social-media, zoekengines, vacaturesites en bekende vraag & antwoord sites zoals Stack Overflow en DBA Stack Exchange.

In figuur 28 is een grafiek te zien van db-engines.com waarin de populariteitsscores van de vijf gekozen databases sinds november 2012 worden getoond. De scores van juli 2015 worden getoond in figuur 29.



Figuur 28 - Database ranking

|  |  |
| --- | --- |
| Database | Populariteits score |
| Oracle | 1456.72 |
| MySQL | 1283.33 |
| PostgreSQL | 272.83 |
| DB2 | 198.12 |
| Firebird | 23.42 |

Figuur 24 - Populariteitsscores juli 2015

De databases bieden soms meerdere varianten aan, indien meerdere varianten interessant lijken worden deze meegenomen in het onderzoek.

# Oracle

Oracle heeft volgens de populariteitsscore van db-engines.com de meest gebruikte database. De Oracle Database wordt aangeboden in verschillende versies.

Zo heb je Oracle Express Edition, Standard Edition One, Standard Edition en Enterprise Edition. Versies verschillen in het aanbod van functionaliteiten, restricties en gebruikskosten. In dit hoofdstuk wordt de gratis versie (Express Edition) en de goedkoopste versie (Standard Edition One) behandeld.

## Oracle Database 11g Express Edition (XE)

De Oracle Database 11g XE is de gratis database van Oracle. Het wordt gezien als de instap database van Oracle, maar dat betekend niet dat het niet noemenswaardig is voor een volwassen omgeving. Oracle Database 11g XE is namelijk gebaseerd op dezelfde basis als de een na nieuwste versie van de dure Enterprise Edition, echter beschikt de database niet over alle functionaliteiten en worden er een aantal beperkingen opgelegd aan de server.

### Licenties & kosten

De Oracle Database 11g Express Edition wordt standaard aangeboden onder een OTN licentie. Hier zijn geen kosten aan verbonden. De voorwaarden zijn te vinden op <http://www.oracle.com/technetwork/licenses/database-11g-express-license-459621.html>

### Server restricties

* Één database instantie per server
* Mag gebruik maken van maximaal één core van één processor\*
* Maximale opslagcapaciteit van 11GB per server
* Mag maximaal 1GB RAM gebruiken per server\*

\*Indien de server wel meer tot beschikking heeft, limiteert de database zelf op het ingestelde limiet.

### Replicatie mogelijkheden

De Oracle Database biedt meerdere manieren om te repliceren. De Express Edition kan echter maar een van die manieren gebruiken: read-only materialized views.

Een read-only materialized view maakt in principe een nieuwe tabel die wordt opgebouwd uit de resultaten van een SQL query. De data wordt daarna geüpdatet vanaf de master database, er kan worden gekozen om te updaten per tijdsinterval of bij elke commit\* op de gerepliceerde data. Replicatie met materialized views werkt asynchroon en kan als een master-slave worden ingesteld.

\*Updaten bij elke commit is niet mogelijk met alle datatypes en werkt niet samen met transacties.

### Uitgelichte features

**Flashback Query’s**

De mogelijkheid om query’s uit te voeren op data uit het verleden.

**Oracle SQL Developer**

Een ontwikkeltool voor het modelleren, beheren en migreren van de database, beschikbaar op de meeste besturingssystemen.

**Oracle Application Express (APEX)**

Een web-only ontwikkeltool waarmee makkelijk web applicaties ontwikkeld kunnen worden rondom de database.

## Oracle Database 12c Standard Edition One (SE1)

De Oracle Database 12c SE1 is een iets nieuwere versie als de XE, de nieuwe update had als hoofdthema: “cloud”. De functionaliteiten die erbij zijn gekomen hebben ook vooral met dit thema te maken. De Oracle Database SE1 heeft meerdere licentiemodellen en meer functionaliteiten dan de Express Edition van Oracle Database.

### Licenties & kosten

Bij het gebruiken van de Oracle Database 12c SE1 kan worden gekozen voor twee soorten licenties: Een Named User Plus Licentie en een Processor Licentie.

**Named User Plus licentie**

Bij een Named User Plus licentie betaal je per individu dat geautoriseerd is data te bewerken of ophalen uit de database. De kosten liggen hier op $180 per gebruiker met een minimum van vijf gebruikers.

**Processor licentie**

Een processor licentie kan worden gebruikt in een omgeving waar het aantal gebruikers niet van tevoren kan worden bepaald. De kosten worden bepaald aan de hand van de processor(en) van de server. Er wordt echter geen limiet gelegd op hoeveel database instanties er op deze server worden geïnstalleerd.

De kosten van de licentie zijn $5800 voor altijd of $1160 per jaar. Als in een server meerdere CPU sockets worden gebruikt zal voor elke socket een licentie moeten worden aangeschaft.

### Server restricties

* Mag gebruik maken van maximaal twee CPU sockets op een server\*

\*Indien de server wel meer tot beschikking heeft, limiteert de database zelf op het ingestelde limiet.

### Replicatie mogelijkheden

De read-only materialized views van de Express Edition van Oracle worden in deze versie van de database uitgebreid met updateable materialized views waarmee data in de materialized view ook gewijzigd kan worden met update, delete en insert query’s. Het voordeel dat hierbij ontstaat is dat de views ook kunnen worden aangepast wanneer de verbinding met de master database er niet is. De aangepaste data in de materialized view wordt bij de volgende synchronisatie van de master samengevoegd. Bij het samenvoegen kunnen wel conflicten ontstaan, Oracle Database biedt verschillende mogelijkheden om deze automatisch op te laten lossen.

### Uitgelichte features

Naast de uitgelichte features van de Oracle Database 11g XE zijn ook de volgende features uitgelicht.

**SQL Pattern Matching**

Uitgebreide mogelijkheden om patronen te herkennen in de gebruikersdata.

**Java**

Naast PL/SQL (Oracle’s taal voor stored procedures) kunnen ook native Java functies worden aangeroepen.

# MySQL

MySQL is de populairste open-source database ter wereld (Oracle).

MySQL database focust op een hoge performance en een uiterst schaalbare omgeving. In dit hoofdstuk wordt er gekeken naar twee versies: de MySQL Community Edition en de Standard Edition. Functioneel is er nauwelijks verschil, maar het verschil ligt bij het licentiemodel.

## MySQL Community Edition (CE)

MySQL Community Edition is de gratis versie van de MySQL. De Community Edition is gelimiteerd in commercieel gebruik.

### Licenties & kosten

De MySQL CE valt onder de GPL versie 2.0 licentie. Hier zijn geen kosten aan verbonden. De voorwaarden van GPL zijn te vinden op: <https://www.gnu.org/licenses/gpl-2.0.html>.

Als de voorwaarden van de GPL licentie niet passen binnen de bedrijfsvoering kan er gekozen worden voor MySQL Standard Edition.

### Replicatiemogelijkheden

MySQL heeft master-master en master-slave replicatie ingebouwd. Er kan worden gekozen om asynchroon te repliceren of om te repliceren op een manier dat MySQL semi-synchroon noemt.

Met semi-synchroon repliceren wacht de master database bij een commit alleen op reactie van één enkele slave database of de commit is ontvangen. Bij volledig synchroon repliceren wacht de master totdat zeker is dat de commit niet alleen ontvangen is, maar ook is uitgevoerd op alle slaves.

### Uitgelichte features

**MySQL WorkBench**

Een ontwikkeltool voor het modelleren, beheren en migreren van de database, beschikbaar op de meeste besturingssystemen.

## MySQL Standard Edition (SE)

MySQL SE biedt dezelfde functionaliteiten als de Community Edition. Er zijn jaarlijkse kosten verbonden aan een licentie voor de database, maar dan zijn wel de restricties in commercieel gebruik opgeheven.

### Licenties & kosten

Voor de MySQL Standard Edition moet een commerciële licentie worden aangeschaft, waarna de GPL restricties niet meer gelden. De kosten van een commerciële licentie zijn $2000 per server per jaar, de licentie komt samen met 24/7 support.

### Replicatiemogelijkheden

De MySQL Standard Edition biedt geen extra functionaliteiten betreft replicatie ten opzichte van de Community Edition.

### Uitgelichte features

De MySQL Standard Edition biedt geen extra uitgelichte features ten opzichte van de Community Edition.

# PostgreSQL

PostgreSQL is een krachtige en een van de meest geavanceerde open-source database. PostgreSQL wordt gratis aangeboden en wordt ontwikkeld door een team van vrijwilligers verspreidt over de wereld. In tegenstelling tot de meeste databases heeft PostgreSQL maar één versie van hun database, waarin meteen alle functionaliteiten beschikbaar zijn.

## Licenties & kosten

PostgreSQL is uitgebracht onder de PostgreSQL licentie, dit is een licentie vergelijkbaar met BSD or MIT license en zorgt ervoor dat PostgreSQL kosteloos gebruikt kan worden. De voorwaarden zijn te vinden op: <http://opensource.org/licenses/postgresql>

## Replicatiemogelijkheden

In PostgreSQL worden twee verschillende replicatiemethoden aangeboden: hot standby replicatie en streaming replicatie. Beiden zijn geschikt voor asynchrone en synchrone master-slave replicatie.

Hot standby replicatie werkt met logbestanden waarin elke update van de master worden opgeslagen. De logbestanden worden naar alle slaves gestuurd als het logbestand een formaat van 16MB heeft. Met hot standby lopen de slaves altijd net iets achter op de master database.

Streaming replicatie genereert een socket tussen de master en de slaves waardoor updates op de master bijna direct doorgestuurd kunnen worden. Met streaming replicatie bevatten de slaves bijna zonder vertraging meteen de data die ook op de master staat.

Voor de beste replicatie kan een combinatie van beide replicatiemethoden ingezet.

Waarbij streaming replicatie zorgt voor een snelle synchronisatie naar de slave databases en hot standby replicatie ervoor zorgt dat data opnieuw gesynchroniseerd wordt als de slave een tijdje offline is geweest.

## Uitgelichte features

**JSON**

PostgreSQL kan JSON data opslaan en query’en, wat zorgt voor een extra niveau van flexibiliteit.

**HSTORE**

Een Key-Value store binnen de relationele database

# DB2

IBM DB2 database software is snel schaalbaar en betrouwbaar. Er bestaan meerdere versies waarvan er één gratis is. In dit hoofdstuk wordt de gratis versie en de goedkoopste versie behandeld.

## DB2 Express-C (EX-C)

DB2 Express-C is de gratis versie van de DB2 database. Er zijn restricties op de server en functionaliteiten zijn geschrapt ten opzichte van de andere versies van de database.

### Licenties & kosten

Aan de DB2 Express-C versie zijn geen kosten verbonden.

### Server restricties

DB2 Express-C mag gebruik maken van twee cores en 16GB RAM.

Ook mag er een maximum van 15TB data worden opgeslagen.

### Replicatiemogelijkheden

Geen

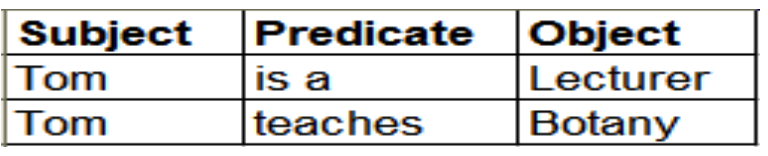
### Uitgelichte features

**JSON**

DB2 database kan JSON data opslaan en query’en, wat zorgt voor een extra niveau van flexibiliteit.

**RDF Store (Graph database)**

Optimalisatie voor het opslaan en ophalen van relaties tussen objecten, vergelijkbaar met een graph database. Het figuur hieronder laat zien hoe de structuur eruit ziet.



Figuur 25 - Structuur RDF store

**Time Travel Query’s**

De mogelijkheid om query’s uit te voeren op data uit het verleden.

## DB2 Express (EX)

DB2 Express is de goedkoopste versie van de DB2 database. De server restricties zijn versoepeld ten opzichte van de Express-C versie, er zijn minder functionaliteiten geschrapt en je hebt beschikking tot support vanuit IBM.

### Licenties & kosten

**Processor Value Unit (PVU) licentie**

Een PVU licentie model is gebaseerd op de PVU score van processoren. Elke processor heeft bij DB2 een PVU score. Scores lopen van 30 t/m 120 PVU per core.

De PVU scores zijn te vinden op: <http://www-01.ibm.com/software/passportadvantage/pvu_licensing_for_customers.html>

De officiële kosten per PVU zijn niet online te verkrijgen, echter is wel op de volgende site een indicatie van €63,68 per PVU te vinden: <http://www.cursor-distribution.de/en/portfolio-and-sales/sales-db2/db2-price-lists>

**Authorized User Single Install (AUSI) licentie**

Bij een AUSI licentie betaal je per apparaat dat data bewerkt of ophaalt van de database. De kosten liggen hier op €217 per gebruiker met een minimum van vijf gebruikers.

**Fixed Term licentie (FTL)**

Met een FTL licentie betaal je voor het gebruik per jaar, per server. Na dit jaar is de licentie verlopen. De kosten van een FTL licentie zijn niet bekend en alleen beschikbaar via een offerte.

**Limited Use Virtual Server (LUVS) licentie**

Een LUVS licentie werkt hetzelfde als een FT licentie, echter bezit je het product in plaats van het te leasen voor één jaar. De kosten van een LUVS licentie zijn €7043 per server.

### Server restricties

DB2 Express mag gebruik maken van acht cores en 64GB RAM.

Ook mag er een maximum van 15TB data opgeslagen worden.

### Replicatiemogelijkheden

DB2 Express heeft mogelijkheid om data synchroon te repliceren in een master-slave of een master-master opstelling. DB2 Express geeft hierbij de optie om in te stellen wanneer de replicatie plaatsvind: Per transactie, in een tijdsinterval, of handmatig.

# Firebird

Firebird staat bekend als een snelle database met ondersteuning voor veel van de standaard SQL functionaliteiten en wordt gratis aangeboden. Firebird heeft vier verschillende versies die vooral verschillen qua configuratie en optimalisatie.

## Licenties & kosten

Firebird is uitgebracht onder het Mozilla Public Licence 1.1 (MPL). Hier zijn geen kosten aan verbonden.

## Replicatie

Firebird heeft geen ingebouwde replicatie. Firebird raad zelf aan third-party software te gebruiken om replicatie te realiseren. Een aantal opties worden aangeraden op: <http://www.firebirdfaq.org/faq249/>

## Uitgelichte features

Geen

# Voor & nadelen per database

In dit hoofdstuk wordt per database de belangrijkste voor- en nadelen benoemd.

## Oracle Express Edition

**Voordelen**

Oracle Database 11g Express Edition wordt gratis aangeboden door Oracle. Je krijgt volgens Oracle een goede starter database dat eenvoudig te beheren is.

Niet alle functionaliteiten van de Oracle Database zijn beschikbaar in deze versie, maar toch bezit je met deze database over een flink aantal functionaliteiten en handigheidjes die de ontwikkelsnelheid ten goede kan komen.

De database komt met een handige tool genaamd SQL Developer, deze biedt een user interface waarmee je makkelijk je database kan modeleren of beheren.

**Nadelen**

De Oracle Database Express Edition maakt gebruik van een iets oudere versie van de Oracle Database en heeft daarom niet alle verbeteringen, optimalisaties en functionaliteiten die aanwezig zijn in de nieuwste update. Daarnaast is de functionaliteit nog verder beperkt omdat Oracle een aantal functionaliteiten achterwege laat in de Express Edition. Een vergelijking van functionaliteiten die wel en niet beschikbaar zijn in de verschillende versies van Oracle Database is te vinden in [bijlage A](#_Bijlage_A_–_1).

Naast de beperkte functionaliteit kan de database door restricties ook maar gebruik maken van een beperkt deel van de servercapaciteit.

## Oracle Standard Edition

**Voordelen**

Oracle Database 12c Starter Edition One is de meest betaalbare database van Oracle en komt net als de Express Edition samen met de SQL Developer tool. De database heeft meer functionaliteiten en verbeteringen dan de Express Edition. Ook zijn de serverrestricties voor het grootste deel opgeheven. Welke functionaliteiten er bij zijn gekomen ten opzichte van de Express Edition is te vinden in [Bijlage A](#_Bijlage_A_–).

**Nadelen**

Alhoewel er meer functionaliteiten beschikbaar zijn dan in de Express Edition, zijn nog lang niet alle functionaliteiten aanwezig. Ook kunnen de licentiekosten ervoor zorgen dat het minder rendabel wordt om te kiezen voor de Standard Edition.

## MySQL Community Edition

**Voordelen**

De MySQL Community Edition is onder de GPL licentie gratis te gebruiken. De database staat vooral bekend erg snel te presteren in simpele omgevingen.

MySQL bundelt de database met de MySQL WorkBench tool, waarmee gemakkelijk allerlei taken rondom de database verricht kunnen worden.

**Nadelen**

De MySQL Community Edition wordt afgenomen met een GPL licentie, er zitten daardoor een aantal restricties aan het commercieel gebruiken van de MySQL database.

## MySQL Standard Edition

**Voordelen**

De Standard Edition van MySQL heeft grotendeels dezelfde voordelen als de Community Edition. Het enige verschil is dat de Standard Edition vrij is in commercieel gebruik.

**Nadelen**

Er zitten jaarlijkse kosten verbonden aan de database, de software is in principe nooit helemaal in eigen beheer.

## PostgreSQL

**Voordelen**

PostgreSQL is een zeer uitgebreide database die geheel gratis te gebruiken is.

Vooral complexe omgevingen hebben baat bij een PostgreSQL database. De vele datatypes en de geavanceerde SQL functionaliteiten helpen bij het snel ontwikkelen van de applicatie rondom de database. Daarnaast is PostgreSQL geoptimaliseerd voor query’s met allerlei soorten joins. (Momjian, 2015)

**Nadelen**

De vele PostgreSQL functionaliteiten zijn soms overkill voor simpele applicaties waar vooral index look ups worden gedaan. Daarnaast wordt er geen standaard tool meegeleverd om de database te beheren, maar er wordt wel gerefereerd naar allerlei software van derde partijen dat hiervoor wel geschikt is. <http://www.postgresql.org/download/products/1-administrationdevelopment-tools/>

## DB2 Express–C

**Voordelen**

De IBM DB2 Express-C database is gratis, de database heeft veel handige features die helpen bij ontwikkeling. De database staat bekend snel te werken met allerlei soorten data.

**Nadelen**

Er is geen enkele vorm van replicatie mogelijk.

## DB2 Express

**Voordelen**

De DB2 Express versie heeft veelal dezelfde voordelen als de Express-C, echter zijn er meer functionaliteiten beschikbaar waaronder replicatie. Een overzicht van de verschillen tussen Express en Express-C is te vinden op: <http://www.ibm.com/developerworks/data/library/techarticle/dm-1311db2compare/>

Verder is er ook onbeperkt support beschikbaar vanuit IBM.

**Nadelen**

De kosten voor de DB2 Express database zijn redelijk hoog en nog niet volledig bekend, bijvoorbeeld voor de FTL licentie moet eerst een offerte worden aangevraagd.

## Firebird

**Voordelen**

Firebird is een snelle en stabiele database. Daarnaast stelt het de minste eisen aan de serverspecificaties.

**Nadelen**

Firebird is een relatief onbekende database, waardoor er weinig support beschikbaar is van ervaringsdeskundigen. Mede door de populariteit is er ook een gebrek aan goede tools om de database mee te beheren.

Er is geen ingebouwde replicatie en de documentatie blijkt niet volledig volgens velen op het internet.

# Literatuurlijst

IMB100. (sd). *Relational Database*. Opgehaald van imb.com: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/reldb/

Oracle. (sd). *MySQL*. Opgehaald van mysql.com: http://www.oracle.com/nl/products/mysql/overview/index.html

Momjian, B. (2015, 09 1). *PostgreSQL query optimizer explained*. Opgehaald van momjian.us: https://momjian.us/main/writings/pgsql/optimizer.pdf

Intuit. (sd). *Timeline of database history*. Opgehaald van intuit.com: http://quickbase.intuit.com/articles/timeline-of-database-history

Oracle. (sd). *Oracle Express Documentation*. Opgehaald van oracle.com: http://docs.oracle.com/cd/E17781\_01/index.htm

Oracle. (2015, 08 01). *Oracle 12c whitepaper*. Opgehaald van oracle.com: http://www.oracle.com/technetwork/database/oracle-database-editions-wp-12c-1896124.pdf

Oracle. (sd). *Oracle Database Comparison*. Opgehaald van oracle.com: http://www.oracle.com/us/products/database/enterprise-edition/comparisons/index.html

Bolton, D. (2015, 03 19). *PostgreSQL over MySQL*. Opgehaald van dice.com: http://insights.dice.com/2015/03/19/why-i-choose-postgresql-over-mysqlmariadb/

Tezer, O. (2014, 2 14). *Comparison of Relational Databases*. Opgehaald van digitalocean.com: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

IBM. (sd). *DB2 Express c* . Opgehaald van ibm.com: https://www.ibm.com/developerworks/vn/library/contest/dw-freebooks/Nhap\_Mon\_DB2\_ExpressC/Getting\_Started\_with\_DB2\_Express\_v9.7.pdf

IBM. (2013, 06 14). *DB2 Comparison*. Opgehaald van ibm.com: DB2 comparison

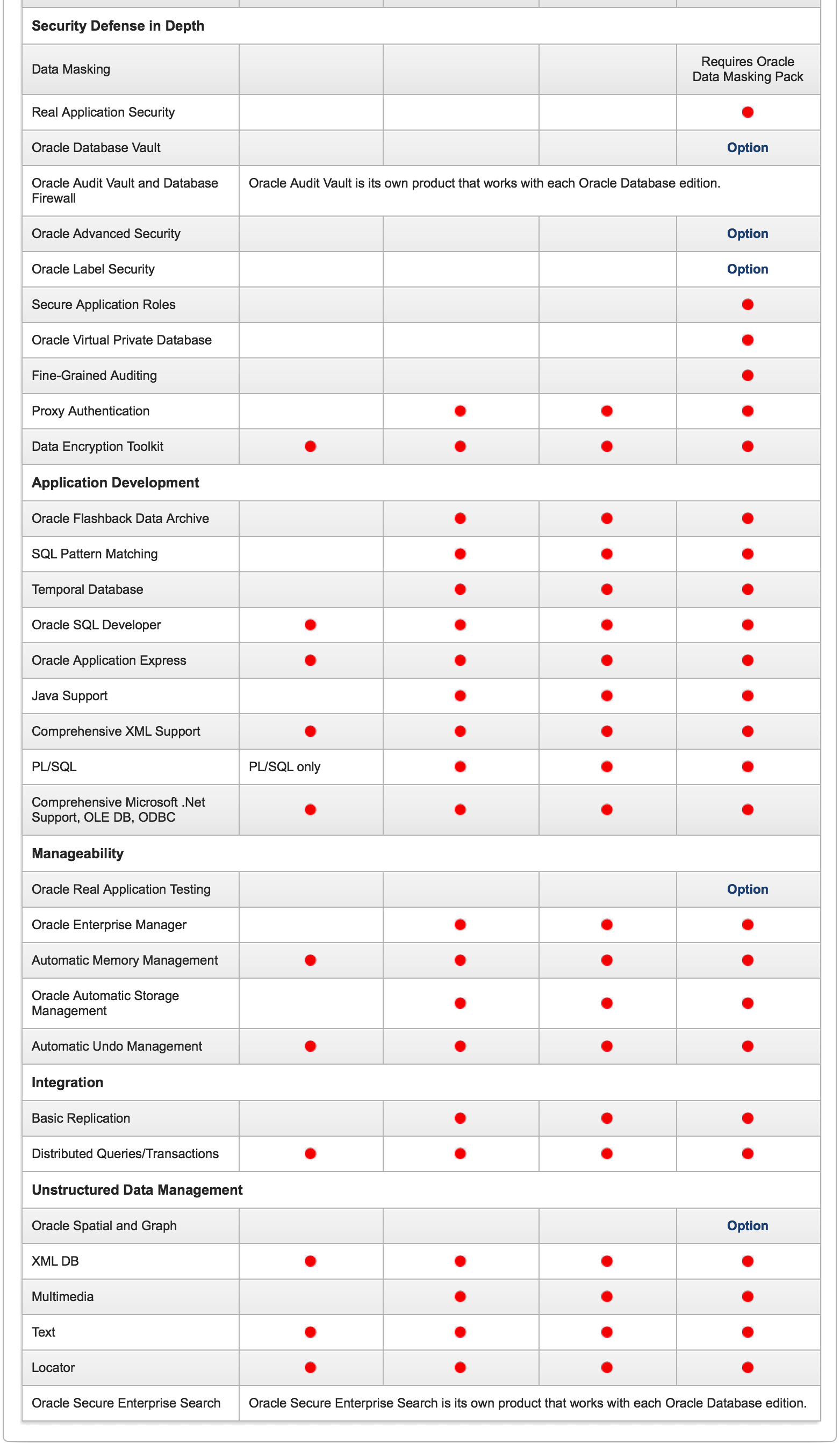
PostgreSQL. (sd). *About PostgreSQL*. Opgehaald van postgresql.com: http://www.postgresql.org/about

IBM. (2009, 10 29). *DB2 Licenses*. Opgehaald van ibm.com: http://www.ibm.com/developerworks/data/library/techarticle/dm-0611zikopoulos2/

Oracle. (2015, 09 01). *Oracle license cost*. Opgehaald van oracle.com: http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf

# Bijlages

## Bijlage A – Oracle functionaliteiten per versieMacintosh HD:Users:Jochem:Documents:oracle1.png



## 14.4 Bijlage D – Persistence framework onderzoeksresultaten



Persistence – Jonker BV.

Versie 1.0

Jochem Pouwels - 1613727  
Docentbegeleider: Rik Boss

Inhoudsopgave

1 Inleiding 110

2 Introductie 110

3 JOOQ 111

3.1 SQL builder 111

3.1.1 Dialect onafhankelijk 111

3.1.2 Syntax safe en auto-complete 111

3.2 Het genereren van Java code 112

3.3 SQL uitvoeren 113

3.3.1 JOOQ Result versus JDBC ResultSet 113

3.3.2 Omzetten naar standaard Java objecten. 114

3.3.3 Web resultaten 114

3.3.4 Omzetten naar POJO’s 114

4 MyBatis 115

4.1 SQL building 115

4.1.1 XML met Dynamic SQL 115

4.1.2 Annotaties 116

4.1.3 SQL Class 116

4.2 MyBatis Generator 117

4.3 SQL uitvoeren 117

4.3.1 Omzetten naar Java types 117

4.3.2 Omzetten naar Java objecten 118

5 Conclusie 119

5.1 Voordelen van JOOQ 119

5.2 Nadelen van JOOQ 119

5.3 Voordelen van MyBatis 119

5.4 Nadelen van MyBatis 120

6 Literatuurlijst 120

# Inleiding

In dit document wordt twee software libraries bekeken die dienen als hulpmiddel in het gebruiken van een database binnen Java. Als eerst wordt in hoofdstuk 2 een korte introductie over het onderzoek gegeven, daarna in hoofdstuk 3 en 4 wordt er dieper ingegaan op de onderzochte software. Tot slot wordt in hoofdstuk 5 de voor en nadelen op een rijtje gezet.

# Introductie

In de nieuwe situatie voor Jonker gaat een Java applicatieserver verbinden en communiceren met een PostgreSQL database.

Standaard Java biedt de mogelijkheid de JDBC API te gebruiken om te werken met de database. Het gebruiken van enkel JDBC komt gepaard met een aantal nadelen. Zo kunnen complexe query’s snel onoverzichtelijk doordat Java Strings conditioneel aan elkaar geplakt moeten worden. Een voorbeeld van zo’n onoverzichtelijke query is hieronder te zien.

String query = "select count (\*) from OfferteDetail B " +

"LEFT OUTER JOIN OfferteCentrale OC ON " +

"OC.OfferteCentralenr = B.Centralenr " +

"LEFT OUTER JOIN Centrale C ON " +

"C.Centralenr = OC.HoofdCentralenr " +

"WHERE C.KlantGroepnr=" + beheer.getCentrale().getKlantGroepnr() +

" AND B.KoppelKlantnr=" + klantnr +

" AND B.KoppelReceptnr=" + itemnr;

Daarnaast is het werken met de JDBC ResultSet eindeloos code kloppen. Onderstaand voorbeeld geeft een tabel aan met 3 kolommen, maar stel eens een tabel voor met 100 kolommen.

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery("SELECT \* FROM AUTHOR");

while (rs.next())

{

    Integer id = rs.getInt(“id”);

String firstName = rs.getString(“firstname”);

String lastName = r.getString(“lastname”);

}

rs.close();

st.close();

Er zijn een aantal software oplossingen beschikbaar met als doel ervoor te willen zorgen dat het koppelen en gebruiken van een database binnen Java software makkelijker wordt. In dit onderzoek worden een twee van deze software oplossingen beschreven en wordt er een advies gegeven wanneer welke te gebruiken is.

# JOOQ

De makers van JOOQ noemen het op hun eigen site (JOOQ): “The easiest way to write SQL in Java”. JOOQ genereert Java code vanuit je database en laat je veilig SQL query’s bouwen en/of uitvoeren met behulp van hun native Java API.

JOOQ zorgt ervoor dat je minder tijd bezig bent met het controleren en maken van de communicatie met je database en meer met de business-logica zelf.

Omdat JOOQ alle SQL standaarden heeft geïmplementeerd en daarnaast ook nog veel unieke functionaliteiten per database ondersteund is zo goed als elke SQL statement mogelijk.

JOOQ wordt geleverd door distributie van drie verschillende .jar bestanden.

**jooq.jar**

Bevat de basis library welke nodig is om JOOQ methodes uit te kunnen voeren.

**jooq-codegen.jar**

Bevat een tool dat Java code genereert op basis van metadata van de database. De Java code dient als een indexering van alle tabellen, views, kollommen, procedures en enums in de database.

**jooq-meta.jar**

Bevat een tool dat verbinding maakt met een database en door het schema heen kan crawlen om zo metadata te genereren. Deze metadata wordt ook gebruikt door de jooq-codegen.jar, maar zou ook los gebruikt kunnen worden.;

## SQL builder

JOOQ komt met een API die helpt bij het maken van veilige SQL query’s door ze op te bouwen met een Java API. Deze API noemen ze de SQL Builder.

Door gebruik te maken van de SQL Builder heeft JOOQ een aantal voordelen ten opzichte van JDBC bij het bouwen van query’s.

### Dialect onafhankelijk

Verschillende databases hebben vaak eigen unieke functionaliteiten en ook een eigen SQL dialect, wat betekend dat een SQL query met dezelfde uitkomst net iets anders wordt geschreven. JOOQ zorgt ervoor dat de verschillen tussen databases geabstraheerd wordt. SQL query’s worden op één manier opgebouwd, waarna het vertalen naar database specifieke SQL query’s door JOOQ geregeld wordt.

### Syntax safe en auto-complete

De query’s in de SQL Builder API worden direct gecontroleerd op een correcte SQL-syntax. Hiervoor maakt JOOQ slim gebruik van interfaces in combinatie met de syntax detectie die in vele IDE’s aanwezig zijn. Daarnaast zorgen de interfaces ook nog eens ervoor dat auto-complete activeert in de IDE’s die dit ondersteunen. Om dit wat beter uit te leggen wordt hierna een voorbeeld gegeven van een SQL query opgebouwd in JOOQ.

create.select()

.from(AUTHOR.as("a"))

.join(BOOK.as("b")).on(a.ID.equal(b.AUTHOR\_ID))

.where(a.YEAR\_OF\_BIRTH.greaterThan(1920)

.and(a.FIRST\_NAME.equal("Paulo")))

.orderBy(b.TITLE)

.fetch();

Bovenstaande JOOQ SQL query genereert de volgende SQL

SELECT \*

FROM author a

JOIN book b ON a.id = b.author\_id

WHERE a.year\_of\_birth > 1920

AND a.first\_name = 'Paulo'

ORDER BY b.title

Als we kijken naar de JOOQ query worden er op veel momenten auto-complete voorgesteld om je op weg te helpen bij het schrijven van de query, een aantal van deze momenten zijn hieronder in de tabel weergegeven.

|  |  |
| --- | --- |
| Gebruiker typt : | Auto-complete stelt voor: |
| “create.select().” | .from(Table tableConstant)  .from(String tableName) |
| “.from(AUTHOR). | .join(Table tableConstant)  .join(String tableName)  .where(Condition condition) |
| “.join(BOOK) | .on(Condition condition) |

Tabel 7 - Auto-complete momenten

## Het genereren van Java code

Een optionele, maar wel een van de handige functionaliteiten is het generen van Java code op basis van de database. De code generator van JOOQ zet meerdere aspecten van de database om naar Java constanten.

**Keys.java:** Bevat referenties naar primary, unique en foreign key objecten

**Routines.java:** Bevat uitvoerbare procedures in de vorm van Java methodes

**Sequences.java:** Bevat sequence objecten

**Tables.java:** Bevat alle referenties naar table objecten

**UDTs.java:** Bevat alle referenties naar UDT objecten

Functionaliteiten zoals auto-complete en detecteren van fouten vóór het compileren worden hierdoor ook beschikbaar voor de tabel structuur van de database waardoor tabelnamen niet meer verkeerd gespeld kunnen worden. (Een voorbeeld is de AUTHOR of BOOK in de JOOQ query van [hoofdstuk 3.1.2](#_Syntax_safe_en)) Zelfs als het schema van de database in de toekomst veranderd, kan na het opnieuw gebruiken van de code generator nieuwe fouten meteen gedetecteerd worden. Bijvoorbeeld als er een kolom is verwijderd uit een tabel.

Naast het genereren van constanten is er ook de mogelijkheid om DAO’s te laten genereren, dit zijn Java klassen met een aantal verschillende CRUD methoden, zoals fetchByColumn, exists, insert, update en dergelijke. Deze DAO’s kunnen zelf worden uitgebreid. Hieronder wordt een voorbeeld gegeven van het gebruik van zo’n DAO.

BookDao bookDao = new BookDao();

Book book = bookDao.findById(5);

book.setTitle("1984");

book.setPublishedIn(1948);

bookDao.update(book);

## SQL uitvoeren

In de vorige hoofdstukken werd uitgelegd hoe JOOQ een eigen manier aanbiedt om SQL query’s te maken. Deze SQL query’s kunnen gebruikt worden in een API naar keuze, ook in JOOQ zelf. JOOQ maakt verbinding met een database via een JDBC driver, net zoals dit vaak in native Java wordt gedaan. JOOQ probeert het gebruik van JDBC te verbeteren door de resultaten in een meer object-georiënteerde manier aan te bieden. In dit hoofdstuk worden een aantal verschillen benoemd tussen JOOQ en JDBC resultaten.

### JOOQ Result versus JDBC ResultSet

Bij het uitvoeren van een SELECT query worden de resultaten opgeslagen in een Result object die enigszins lijkt op het ResultSet object waar JDBC de resultaten in staan, echter zijn er wel degelijk een aantal verschillen.

* Je bent niet langer zelf verantwoordelijk voor het sluiten van database connecties
* De objecten staan meteen in een in-memory Java object i.p.v. een cursor.

Hieronder wordt een voorbeeld gegeven van zo’n JOOQ Result.

Result<Record> result = create.select().from(AUTHOR).fetch();

for (Record r : result) {

Integer id = r.getValue(AUTHOR.ID);

String firstName = r.getValue(AUTHOR.FIRST\_NAME);

String lastName = r.getValue(AUTHOR.LAST\_NAME);  
 }

De records van een Result object in JOOQ kunnen handmatig verwerkt worden, maar ook kan worden gekozen om de objecten direct door JOOQ om te laten zetten naar Java types, web resultaten of zelf gedefinieerde objecten. Dit wordt in de volgende hoofdstukken behandeld.

### Omzetten naar standaard Java objecten.

Waar in een JDBC ResultSet de data altijd handmatig moet worden omgezet naar bruikbare data, zijn in een JOOQ Result een aantal methodes te gebruiken die je de meeste usecases netter en sneller kan laten oplossen. Zo wordt het bijvoorbeeld makkelijk gemaakt om direct een resultaat naar een Java objecten om te laten zetten. Hieronder staan twee voorbeelden.

Map<Integer, String> map = create.selectFrom(BOOK).fetch().intoMap(BOOK.ID, BOOK.TITLE);

String[] titles = create.select().from(BOOK).fetch().intoArray(BOOK.TITLE);

### Web resultaten

String json = create.selectFrom(BOOK).fetch().intoJson();

Met het Result object van JOOQ kunnen resultaten ook direct gecodeerd worden naar populaire formaten gebruikt in web services, bijvoorbeeld XML, JSON of CSV.

### Omzetten naar POJO’s

Ook kan er gekozen worden om de resultaten om te zetten naar POJO’s, dit zijn Java classes die zonder dependencies moeten kunnen bestaan.

JOOQ vereist ook dat de POJO benoemd is naar de tabelnaam, en variabelen heeft die gelijknamig zijn aan de kolomnamen van de tabel. Als aan deze eisen is voldaan kan aan JOOQ worden doorgegeven de resultaten van de database direct om te zetten. Handig is dat JOOQ ook zelf deze POJO’s kan genereren vanuit de database.

BookRecord book = create.selectFrom(BOOK).where(BOOK.ID.equal(1)).fetchOne();

System.out.println("Title : " + book.getTitle());

System.out.println("Published in: " + book.getPublishedIn());

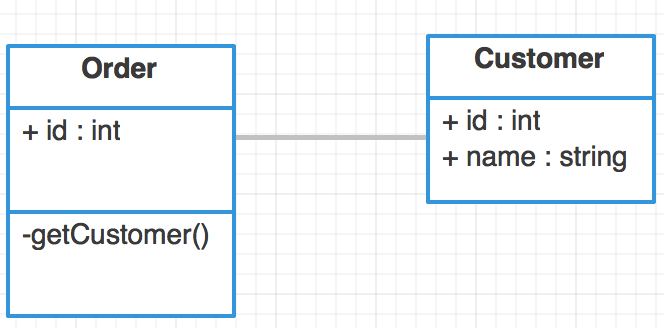
De klasse die verantwoordelijk is voor het omzetten van de Result kan ook uitgebreid worden met nieuwe regels. ModelMapper is een software library dat hier goed op inspeelt.

ModelMapper heeft een uitbreiding gemaakt ondersteunt daardoor hiërarchische objecten (deze ontstaan vaak bij one-to-many of many to many join query’s).

Een voorbeeld is hieronder te zien. In tabel 2 staan de database resultaten die JOOQ ontvangt, en in figuur 1 het ontwerp van de objecten die ModelMapper maakt.

|  |  |  |
| --- | --- | --- |
| ORDER\_ID | ORDER\_CUSTOMER\_ID | ORDER\_CUSTOMER\_NAME |
| 123 | 987 | John |

Tabel 8 - Voorbeeld database resultaten



Figuur 26 - Ontwerp van de door ModelMapper gegenereerde Java objecten

# MyBatis

MyBatis is een SQL framework voor Java, het zorgt naar eigen zeggen voor minder JDBC code en automatiseert een hoop veelgebruikte functionaliteiten. Query’s worden meestal met XML bestanden geconfigureerd, maar er kan ook voor een andere manier worden gekozen.

De MyBatis library kan gebruikt worden door het importeren van mybatis-x.x.x.jar in een Java project.

## SQL building

MyBatis biedt verschillende methoden voor het maken van SQL query’s. De krachtigste en meest gebruikte is het XML formaat, deze ondersteund namelijk Dynamic SQL wat in het volgende hoofdstuk wordt uitgelegd.

Naast XML is er ook een SQL API of kunnen query’s worden gemaakt met annotaties.

### XML met Dynamic SQL

Het maken van complexe query’s gaat in software vaak samen met het conditioneel wijzigen van de query gebaseerd op de context waarin de gebruiker zich bevindt. In Standaard Java eindig je vaak met het aan elkaar plakken van meerdere Java Strings.

MyBatis laat je SQL query’s maken in XML en probeert je hierdoor meer met de SQL bezig te laten zijn dan met Java code. Het eerste voordeel wat opvalt bij het maken van een SQL query’s in XML is dat het er allemaal een stuk overzichtelijker van wordt, de query kan bijvoorbeeld gewoon op meerdere lijnen geplaatst worden zonder quotes (“”) of plussen (+) te gebruiken.

MyBatis biedt verschillende manieren om de SQL dynamisch te maken, hiermee wordt bedoeld dat bepaalde elementen in de SQL veranderen naarmate de context waarin deze wordt uitgevoerd veranderd.

* Er kunnen parameters in de query worden geplaats
* Er kunnen if/else statements gemaakt worden met XML tags
* Er kunnen for loops gedaan worden met XML tags
* Tekst of parameters kunnen worden gewijzigd met patterns

De parameters kunnen simpelweg geplaats worden door “#{parameternaam}” in de SQL te plaatsen. Hieronder staat een voorbeeld van zo’n query met een parameter.

<select id="selectUser" resultType="User">

SELECT id, username, password

FROM users

WHERE id = #{id}

</select>

Parameters hoeven niet per se letterlijk in de SQL te worden geplaatst, ze kunnen ook gebruikt worden om een conditionele situatie aan te duiden. Met de “IF” XML tag kan bijvoorbeeld de waarde van een parameter gecontroleerd worden zoals hieronder wordt gedaan.

<select id="selectUser" resultType="User">

SELECT id, username, password

FROM users

WHERE id = #{id}

<if test="city != null">

AND city = #{city}

</if>

</select>

Naast de IF tag kan ook de rest van de in dit hoofdstuk genoemde functionaliteiten met tags worden verricht.

### Annotaties

Naast XML biedt MyBatis ook een mogelijkheid om annotaties te gebruiken zoals hieronder.

@Select("select id, username, password from users where id = #{id}")

User selectUser(int id);

Deze functionaliteit zorgt voor minder overhead in code dan het XML formaat, maar is wel wat beperkt. Dynamic SQL is hier bijvoorbeeld niet mogelijk.

### SQL Class

Query’s in het XML formaat en met annotaties zijn bedoeld om direct uit te laten voeren door MyBatis. Wil je echter een SQL string genereren en deze ergens anders voor gebruiken dan kan er gebruik gemaakt worden van de SQL class van MyBatis.

De SQL class biedt een aantal handigheidjes ten opzichte van query’s bouwen met Java strings. Er hoeft bijvoorbeeld geen rekening meer worden gehouden met het toevoegen van spaties tussen verschillende SQL componenten. Daarnaast hoeft er ook niet meer op de volgorde van WHERE of AND gelet te worden. De AND is namelijk niet nodig. Een voorbeeld staat hieronder waar eerst een SQL query wordt gemaakt met Java SQL en daarna dezelfde query met de SQL class van MyBatis.

String sql = "SELECT U.id, U.username, U.password "

"FROM users U " +

"WHERE U.id = ? + "

“AND U.city = ?”;

String sql = new SQL()

.SELECT("U.id, U.username, U.password")

.FROM("users U")

.WHERE(“U.id = ?")

.WHERE("U.city = ?");

## MyBatis Generator

Een zij-project van MyBatis is de MyBatis Generator. De generator kan de tabellen van je database inspecteren en genereert daarvoor project bestanden voor MyBatis.

Na het genereren zijn voor elke tabel de standaard CRUD operaties beschikbaar via Java door gebruik te maken van de gegenereerde DAO’s. Hierna hoeven enkel join query’s, stored procedures en unieke query’s die niet onder de standaard CRUD operaties vallen nog zelf gemaakt worden.

## SQL uitvoeren

MyBatis zorgt er ook voor dat het werken met resultaten makkelijker wordt ten opzichte van JDBC. De database resultaten worden altijd omgezet naar een Java object. Dit kan een van de standaard Java types zijn zoals een String, int, Map of een ArrayList zijn, maar ook een zelf gespecificeerde Java object.

### Omzetten naar Java types

Standaard zet MyBatis alle resultaten om naar een Java HashMap waarin elke kolom een key is in de HashMap. Ook kan er aangegeven worden een andere Java type te gebruiken. Hieronder staan twee voorbeelden van SQL query’s die een Java type maken, met daarbij de aanroepbare methode.

<select id="getUsernames" resultType="string">

SELECT username

FROM users

</select>

<select id="countUsers" resultType="int">

SELECT SUM(\*)

FROM users

</select>

public static ArrayList<String> getUsernames()

public static int countUsers()

### Omzetten naar Java objecten

Naast Java types, kan MyBatis ook de data in JavaBeans laten zetten. Voor alle tabellen genereerd de MyBatis Generator al zo’n JavaBean. Deze beans hebben dezelfde namen als de database tabellen, en de kolomnamen van die tabellen zijn beschikbaar als variabelen onder dezelfde naam, bereikbaar via getters en setters.

Wil je buiten de simpele JavaBeans om en wil je bijvoorbeeld objecten hiërarchisch in elkaar verwerken, dan kan dit door een resultMap te gebruiken. ResultMaps zijn XML configuraties waarmee de structuur van een JavaBean kan worden aangegeven. Er worden kolommen gekoppeld aan variabelen, maar ook aan associaties en collecties. In het volgende voorbeeld wordt eerst een blogPost object gedefinieerd, waarna deze in een blog object wordt geassocieerd als een lijst met blogPosts.

<resultMap id="blogPostResult" type="Post">

<id property="id" column="id"/>

<result property="subject" column="subject"/>

<result property="body" column="body"/>

</resultMap>

<resultMap id="blogResult" type="Blog">

<id property="id" column="blog\_id" />

<result property="title" column="blog\_title"/>

<collection property="posts" ofType="Post" resultMap="blogPostResult" columnPrefix="post\_"/>

</resultMap>

Het Blog object (met daarin een lijst van Post objecten) kan opgehaald worden met de volgende select query, dit wordt gedaan door de naam van de resultMap aan te geven in een XML property.

<select id="selectBlog" resultMap="blogResult">

select

B.id as blog\_id,

B.title as blog\_title,

B.author\_id as blog\_author\_id,

P.id as post\_id,

P.subject as post\_subject,

P.body as post\_body,

from Blog B

left outer join Post P on B.id = P.blog\_id

where B.id = #{id}

</select>

# 

# Conclusie

Beide frameworks zorgen er op hun eigen manier voor dat het maken van SQL query’s niet meer vooral bestaat uit het aan elkaar plakken van strings, ook zorgen ze ervoor dat de resultaten die terug komen van de database handig verwerkt wordt. Dit hoofdstuk zet de voor en nadelen van elk framework nog eens op een rijtje.

## Voordelen van JOOQ

JOOQ helpt je met het bouwen van SQL query’s door de tekst gebaseerde SQL query’s om te zetten naar query’s opgebouwd in een Java syntax. Samen met de code generator zorgt dit voor een aantal handige features. Fouten in SQL worden getoond voordat de code zelfs wordt uitgevoerd, en door auto-complete op tabelnamen, kolomnamen en SQL functies kunnen query’s sneller worden gemaakt.

Het verwerken van resultaten gaat redelijk simpel, de API biedt hiervoor verschillende methoden die de resultaten direct om kan zetten naar Java objecten, maar kan het ook omzetten naar een resultaat voor de web service in bijvoorbeeld JSON.

Daarnaast zorgt JOOQ er ook voor dat er niet meer op verschillende SQL dialecten van databases hoeft worden gelet, de vertaalstap wordt geautomatiseerd door JOOQ. Het overstappen naar een andere database in de toekomst zal daardoor zo goed als moeiteloos kunnen.

## Nadelen van JOOQ

Alhoewel de DSL van JOOQ erg lijkt op SQL, moet er toch een nieuwe syntax geleerd worden om SQL query’s te schrijven.

JOOQ biedt de mogelijkheid tot vele geavanceerde SQL functionaliteiten zoals nested selects, aliasing, unions, self-joins. Voor het gros van de developers is dit ruim genoeg, maar het is mogelijk dat als een database nieuwe functionaliteiten introduceert dat deze niet meteen beschikbaar zijn in JOOQ. Hiervoor zou wel zelf een uitbreiding in JOOQ kunnen worden gemaakt.

## Voordelen van MyBatis

MyBatis zorgt voor een nieuwe manier van SQL query’s maken in Java. In een speciale XML syntax kunnen tekst gebaseerde query’s worden gemaakt en met enkele toevoegingen kunnen deze aangepast worden op de context met bijvoorbeeld parameters, IF statements en FOR loops.

Doordat query’s in aparte XML bestanden worden geplaatst is dit ook meteen bevorderlijk voor de herbruikbaarheid van de query’s.

Daarnaast is het relatief simpel om resultaten om te zetten naar Java objecten, dit kunnen standaard Java types zijn, maar er kunnen ook geavanceerde Java objecten gemaakt worden die je zelf specificeert. Deze kunnen gebruikt worden in de Java code, maar ook geparsed worden door bijvoorbeeld een JSON library om er zo direct web service resultaten van te maken.

## Nadelen van MyBatis

Er zijn een aantal problemen die MyBatis nog niet oplost, zo kunnen er nog syntax fouten gemaakt worden in SQL die pas tijdens het uitvoeren van code naar boven komt, alhoewel dit wel minder vaak zal gebeuren door de betere overzicht die de XML notatie biedt ten opzichte van query Strings in Java.

Ook is de SQL nog steeds dialect gevoelig, wil je wisselen van database zal je zelf de verschillen in SQL moeten herkennen en deze moeten aanpassen in je code.

# 6 Literatuurlijst

JOOQ. (sd). *JOOQ*. Opgehaald van jooq.org: http://www.jooq.org

Wikipedia. (2015, 09 01). *Plain Old Java Object*. Opgehaald van wikipedia.org: https://en.wikipedia.org/wiki/Plain\_Old\_Java\_Object

MyBatis. (2015, 09 24). *Dynamic SQL*. Opgehaald van mybatis.github.io: https://mybatis.github.io/mybatis-3/dynamic-sql.html

MyBatis. (2015, 09 24). *Java API*. Opgehaald van mybatis.github.io: https://mybatis.github.io/mybatis-3/java-api.html

JOOQ. (sd). *JOOQ Manual*. Opgehaald van jooq.org: http://www.jooq.org/doc/3.7/manual-single-page/

BEA Systems, IBM, Microsoft en TIBCO. (2005, 02 1). *WS-ReliableMessaging*. Opgehaald van xmlsoap.org: http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf

## 14.4 Bijlage E – Web service onderzoeksresultaten



Web service - Jonker BV.

Versie 1.0

Jochem Pouwels - 1613727  
Docentbegeleider: Rik Boss

Inhoudsopgave

1 Inleiding 123

2 Introductie 123

3 SOAP en REST 124

3.1 Wat is SOAP? 124

3.2 Wat is REST? 124

4 Verschillen SOAP en REST 125

4.1 Snelheid 125

4.2 API 125

4.3 Betrouwbaarheid 126

4.3.1 Idempotentie (SOAP en REST) 128

4.3.2 WS-ReliableMessaging (SOAP) 129

5 Conclusie 130

5.1 Wanneer SOAP niet gebruiken? 130

5.2 Wanneer SOAP wel gebruiken? 130

5.3 Wanneer REST niet gebruiken? 130

5.4 Wanneer REST wel gebruiken? 130

# Inleiding

In dit document worden populaire communicatiemethodes voor web services onderzocht. Als eerst wordt in hoofdstuk 2 een korte introductie over het onderzoek gegeven, daarna in hoofdstuk 3 worden de communicatiemethoden met elkaar vergeleken. Tot slot wordt in hoofdstuk 4 de voor en nadelen van elke methode nog op een rijtje gezet.

# Introductie

In het onderzoek worden verschillende communicatiemethoden voor web services met elkaar vergeleken.

**Snelheid**

Hoe snel een web service reageert is vooral afhankelijk van de tijd dat het kost om de data te versturen. Per communicatiemethode wordt overzichtelijk gemaakt hoe de data verpakt wordt voordat het over het netwerk gestuurd wordt.

**API**

De API is het aanspreekpunt van de web service. De verschillende communicatiemethoden hebben hun eigen manier van het aanbieden van deze API. Hoe dit gebeurd wordt uitgelegd.

**Betrouwbaarheid**

Een web service is te bereiken via het internet, de communicatie van en naar de web service via een netwerk is niet altijd even stabiel. Er wordt bekeken welke mogelijkheden de communicatiemethoden bieden in verband met het oplossen van onstabiliteit van internet.

# SOAP en REST

De web service die voor Jonker moet worden opgezet zal door meerdere clients bereikt moeten kunnen worden. Daarom wil Jonker graag gebruik maken van een communicatiemethode dat door veel platformen gebruikt kan worden.

SOAP en REST worden het meest gebruikt. Hieronder wordt een korte uitleg gegeven over SOAP en REST. In het hoofdstuk hierna worden de methodes met elkaar vergeleken.

## Wat is SOAP?

SOAP is een open standaard communicatieprotocol voor web services en wordt mede daardoor door veel programmeertalen geïmplementeerd. SOAP werkt door berichten te versturen en ontvangen in een template op basis van XML. De template is hieronder te zien.

<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<MethodeNaam>

<ParameterNaam> data </ParameterNaam>

</MethodeNaam>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

SOAP berichten kunnen op elk internetprotocol worden verstuurd. Bijvoorbeeld via het HTTP, HTTPS, SMTP, JMS of FTP protocol.

## Wat is REST?

REST is geen op zichzelf staand protocol, maar een vastgelegde manier voor communiceren met een web service door middel van het HTTP protocol. De verschillende functionaliteiten van het HTTP protocol worden gebruikt om data te versturen, zo worden de methodes POST, GET, PUT en DELETE gebruikt om de soort methode aan te geven en worden parameters doorgegeven via de URI of via de post parameters van HTTP. Hieronder is een voorbeeld te zien van een REST methode aanroep.

http://www.host.nl/service/methodeNaam?param=data

# Verschillen SOAP en REST

SOAP en REST zijn op zichzelf staande manieren van communiceren en doen dit ook allebei anders. In dit hoofdstuk worden de grootste verschillen op een rij gezet.

## Snelheid

De snelheid van web service communicatie wordt grotendeels bepaald door de tijd dat het kost om data te versturen over het netwerk. Simpel gezegd: hoe meer data je over het netwerk stuurt, hoe meer tijd het kost.

SOAP encodeert data in berichten met XML en stuurt daardoor data over het netwerk met de standaard XML overhead. Hoe groot deze overhead is, is niet van tevoren te zeggen. Dit wordt namelijk vooral bepaald door de XML tag-naam. In het voorbeeld hierna wordt in het blauw de daadwerkelijke data getoond en in het rood wordt de XML overhead aangegeven. <tagnaam> data <tagnaam>.

Daarnaast moet samen met het bericht, ook het standaard SOAP envelop meegestuurd worden. Dit zorgt voor een extra overhead per bericht.

REST is in het geval van netwerkverkeer altijd zuiniger. REST stuurt namelijk data in elk gewenst formaat. Er kan dan gekozen worden voor de meest efficiënte manier voor elke situatie. Dit kan bijvoorbeeld in de vorm van enkel een String voor simpele data of in het JSON (minder overhead dan XML) formaat voor de complexere data.

Daarnaast hoeft er geen extra envelop meegestuurd te worden.

## API

Beide communicatiemethoden bieden hun API aan op een andere manier, met elk zijn voor en nadelen. Hieronder volgt een uitleg voor beide communicatiemethodes.

Het aanbieden van een SOAP API gaat middels een gestandaardiseerd .wsdl bestand waarin allerlei specificaties van de web service staan beschreven. Er staat bijvoorbeeld aangegeven welke methodes beschikbaar zijn, welke parameters hierbij horen en van welke datatypes deze parameters zijn. Het uitlezen van een .wsdl bestand vergt kennis over de structuur, daarom wordt dit vaak automatisch gedaan door de programmeertaal van de client of via externe libraries. Het .wsdl bestand is tevens een contract tussen de client en web service, alleen data aangegeven in het .wsdl bestand wordt geaccepteerd.

Het gebruiken van een .wsdl voor de API heeft zo zijn voordelen en nadelen. Een voordeel is dat de hele API na het laten uitlezen van het .wsdl bestand bekend is bij de gebruiker. Indien dit ondersteund wordt door de IDE van de developer zijn functionaliteiten zoals auto-complete beschikbaar. Daarnaast zijn gebruikers door het .wsdl contract geforceerd om data van het juiste structuur mee te sturen waardoor dit niet gecontroleerd hoeft te worden op de server. Het nadeel is echter wel dat een client vaak libraries nodig heeft om de .wsdl uit te kunnen lezen, waardoor het koppelen van programmatuur aan de web service niet altijd even makkelijk gaat.

De API van SOAP heeft één enkel aanspreekpunt, deze wordt geconfigureerd in het contract. Methodes zijn hierna bereikbaar door de methodenaam aan te geven in het SOAP envelop.

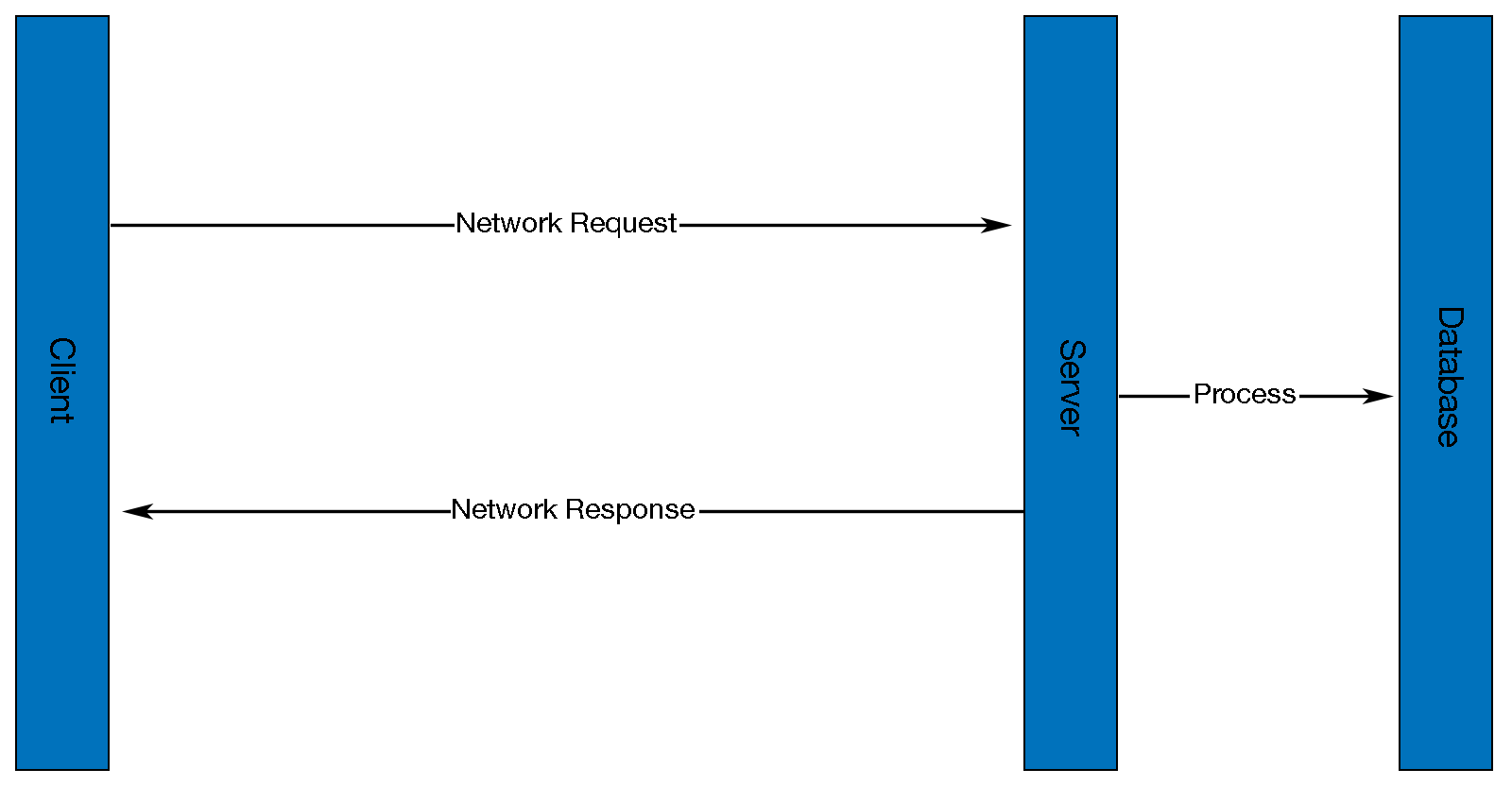
Een REST web service kan bereikt worden door enkel een URI aan te roepen, net als een website. Je moet er zelf ervoor zorgen dat de functionaliteiten bekend zijn bij de gebruikers door bijvoorbeeld de API te hosten op een website of door het verspreiden van een specificatiedocument. Een voorbeeld van een REST API is te vinden op: <https://dev.twitter.com/rest/public>. De API van Twitter.

Ook een REST API heeft zijn voor en nadelen. Het voordeel is dat het relatief simpel is om clients te laten communiceren met een REST web service, elke client dat HTTP requests kan maken kan ook een REST web service bereiken. Daarnaast kan de client zelf aangeven welk mediatype wordt verwacht. dit zou bijvoorbeeld XML, JSON of gewoon tekst kunnen zijn. Het gebrek aan een standaard manier om de API bekend te maken is echter wel een nadeel.

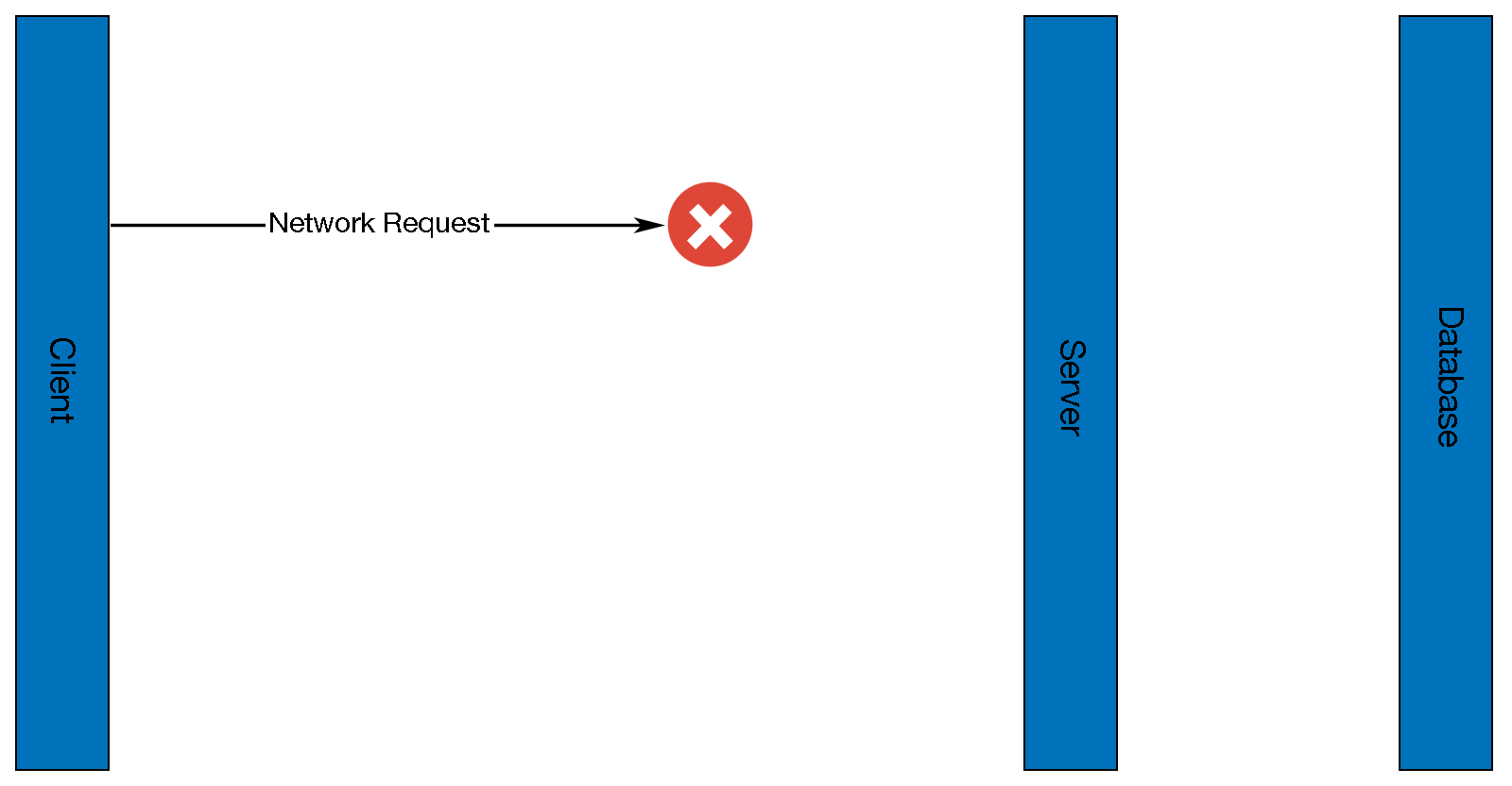
## Betrouwbaarheid

Web services communiceren altijd over het netwerk. Dit terwijl die verbindingen met het netwerk niet altijd betrouwbaar zijn. Het is bekend dat, vooral bij mobiele verbindingen, er berichten verloren kunnen gaan. Dit kan gaan over het bericht naar de server toe (de request) of het bericht terug naar de client (de response).

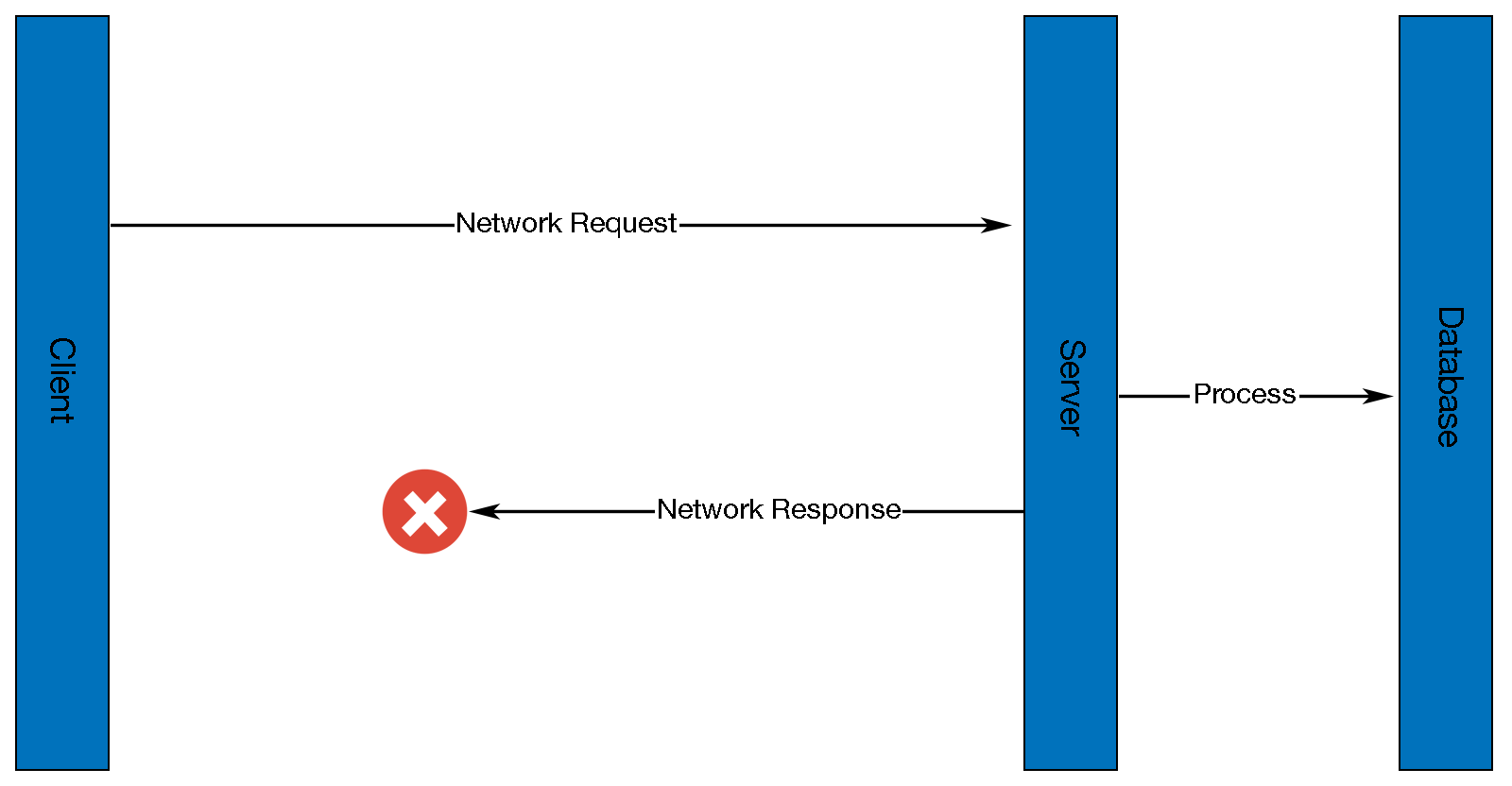
Hieronder worden drie mogelijke situaties getoond met betrekking tot berichten verstuurd via het netwerk. In figuur 32 staat de gebruikelijke situatie waarin de berichten tussen client en server correct worden verstuurd en aankomen. In figuur 33 staat de situatie waarin een request niet aankomt bij de server en in figuur 34 staat de situatie waarin de response de client niet bereikt.



Figuur 27 - normale situatie



Figuur 28 - request komt niet aan



Figuur 29 - response wordt niet ontvangen

Het is voor Jonker belangrijk dat bepaalde berichten, zoals bijvoorbeeld facturaties maar één keer worden verwerkt. Niet vaker, en ook niet minder. Echter kan dit met het standaard HTTP niet worden gegarandeerd. Er moeten maatregelen worden genomen om dit wel te kunnen garanderen. In dit hoofdstuk worden twee oplossingen vergeleken, als voorbeeld wordt het facturatieprobleem toegepast.

### Idempotentie (SOAP en REST)

De eerste oplossing is “idempotentie”, oftewel methodes zo inrichten dat er niks veranderd als de methode nogmaals wordt uitgevoerd. Deze oplossing kan gebruikt worden op zowel SOAP en REST. Hieronder volgt een nadere toelichting.

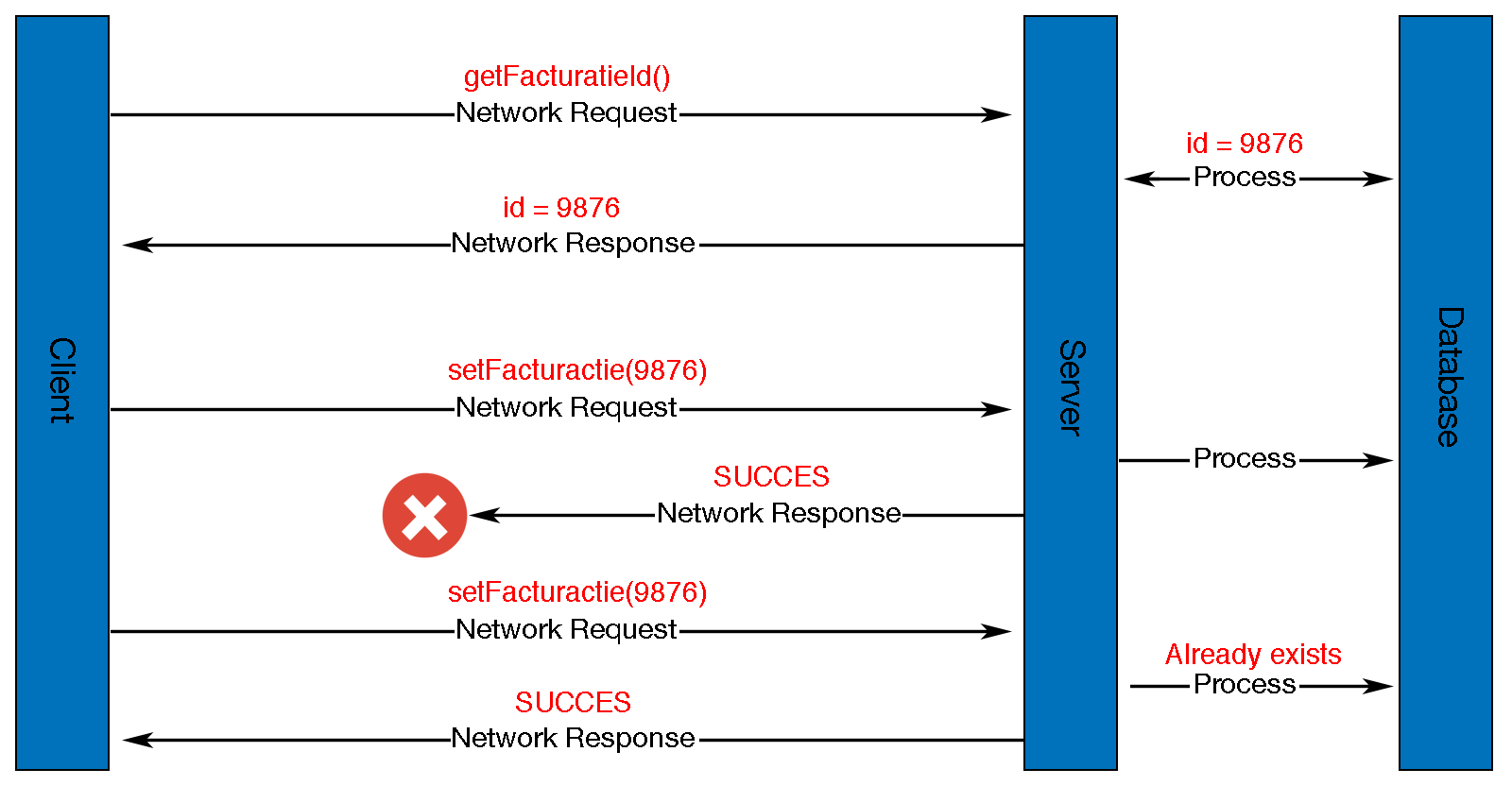
Als een client een request doet op een web service is het gebruikelijk dat de server een response terug stuurt nadat het request succesvol verwerkt is. Zonder een response weet de client natuurlijk niet of er daadwerkelijk wel iets gebeurd is.

Voor een client is het daarom gebruikelijk om een bericht opnieuw te versturen als na een bepaalde tijd nog geen response is ontvangen.

Als de request naar de server niet aankomt zoals in figuur 2 is het niet erg dat berichten opnieuw wordt verstuurd, het bericht is dan namelijk nog niet eerder verwerkt en kan gerust opnieuw verzonden worden. Echter in het geval in figuur 3 waar de response niet bij de client aankomt kan het wel voor problematiek zorgen. In het geval van het facturatieprobleem wordt nu dus twee keer dezelfde factuur opgeslagen. Een client kan niet herkennen waar het bericht is verloren gegaan, daarom moet de oplossing liggen bij de server zelf.

Door methodes in de web service altijd zo in te richten dat ze idempotent zijn, wordt bovenstaand probleem voorkomen. Idempotente methoden kunnen namelijk twee keer worden uitgevoerd met eenzelfde resultaat.

In het geval van facturatie kan bijvoorbeeld van te voren eerst een lege facturatie worden ingevoerd in de database via een extra request, het unieke ID dat bij deze facturatie hoort wordt daarna teruggegeven aan de client. De client kan hierna een methode aanroepen waarmee alleen de data wordt ingevuld voor het unieke ID. Deze methode kan namelijk meerdere keren worden uitgevoerd met hetzelfde resultaat. In figuur 35 staat idempotentie oplossing.



Figuur 30 - Idempotentie oplossing

### WS-ReliableMessaging (SOAP)

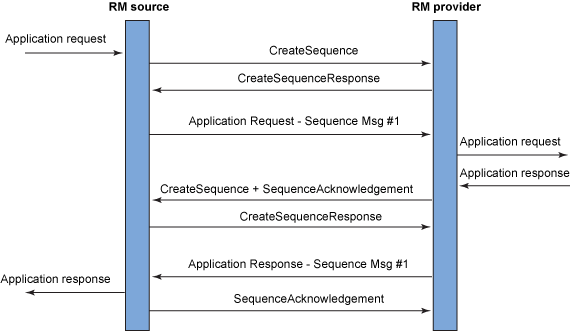
De tweede oplossing is alleen mogelijk voor het SOAP protocol, deze maakt namelijk gebruik van een functie die alleen in SOAP beschikbaar is. Namelijk WS-ReliableMessaging.

WS-ReliableMessaging beschrijft een manier van communiceren met een web service waarbij rekening wordt gehouden dat er geen fouten ontstaan als software, hardware of het netwerk onstabiel is. Volgens (BEA Systems, IBM, Microsoft en TIBCO, 2005) kan WS-ReliableMessaging garanderen dat berichten Op z’n minst één keer, maximaal één keer of precies één keer geleverd kunnen worden.

ReliableMessaging werkt door middleware software bij de client en de server te activeren. Een bericht van de client verloopt dan via de client middleware, die daarna het bericht communiceert in verschillende fases met de middleware van de server. Als een response of request verloren gaat, wordt dit vanzelf opgelost.

Al met al komt het erop neer dat een bericht met zekerheid is verwerkt als de client een response terug krijgt, de client hoeft eigenlijk nooit opnieuw te proberen een bericht te sturen.

Beide de client en de server moeten WS-ReliableMessaging functionaliteiten ingebouwd hebben via interne of externe libraries. Als een client deze library niet tot beschikking heeft, is het niet mogelijk te communiceren met de web service.



Figuur 31 - Werking WS-ReliableMessaging

# Conclusie

Terwijl beide communicatiemethoden hetzelfde doel hebben, namelijk data of services beschikbaar maken via het web, hebben ze elk hun specifieke eigenschappen. In dit hoofdstuk wordt opgesomd in welke situaties het wel of niet handig is om voor een bepaalde communicatiemethode te kiezen.

## Wanneer SOAP niet gebruiken?

SOAP gebruikt relatief veel netwerk bandbreedte, in het algemeen is dit te verwaarlozen als er gebruik gemaakt wordt van redelijke WIFI verbindingen. Gebruik je een mobiele of een trage WIFI verbindingen? dan is het vaak beter een REST web service te gebruiken.

Daarnaast is het verbinden met een SOAP web service niet mogelijk in alle programmeertalen, hier zijn namelijk extra libraries voor nodig. Mede hierdoor is SOAP geen populaire keuze als er web en/of mobile clients worden gebruikt.

## Wanneer SOAP wel gebruiken?

SOAP een opzichzelfstaand protocol, hierdoor is communicatie niet gelimiteerd via HTTP en kunnen er ook andere manieren worden gebruikt. REST kan dit niet, wil je geen HTTP gebruiken? Dan is SOAP je keuze.

Een andere reden kan zijn omdat SOAP een contract heeft tussen client en server. Alle methoden, objecten en datatypes zijn voor het gebruiken al bekend bij beide partijen; de client en de server. Dit opent mogelijkheid tot handigheden zoals auto-complete en het omzetten van complexe datetypen.

Daarnaast kan een SOAP service stateful worden gemaakt, wat voor sommige usecases essentieel is. In een stateful service kunnen requests die gedaan worden afhankelijk zijn van requests die eerder zijn uitgevoerd door dezelfde gebruiker, denk hierbij aan bijvoorbeeld een winkelwagen in een web shop waar in meerdere stappen producten worden toegevoegd.

## Wanneer REST niet gebruiken?

REST zou niet gebruikt moeten worden voor een stateful web service, hiervoor moet de server namelijk een sessie bijhouden van elke gebruiker. De sessie van een gebruiker kan echter wel bijgehouden worden op de client zelf. Waarna deze sessie naar de web service wordt gecommuniceerd waar nodig.

## Wanneer REST wel gebruiken?

REST is een lichtgewicht web service. Het is simpel te koppelen omdat enkel een URI en standaard HTTP functionaliteiten nodig zijn. Daardoor is REST geschikt voor zowel web, mobile als desktop clients.

Ook de flexibiliteit van REST kan soms voordelen opleveren, zo kan elke client (indien de developer van de service dit toelaat) zelf aangeven in welk mediatype ze de resultaten van een service willen ontvangen.

# Literatuurlijst

BEA Systems, IBM, Microsoft en TIBCO. (2005, 02 1). *WS-ReliableMessaging*. Opgehaald van xmlsoap.org: http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf

Mueller, J. (2013, 01 08). *Understanding SOAP and REST*. Opgehaald van smartbear.com: http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/

Rozlog, M. (2010, 04 01). *Rest and SOAP: when should I use each*. Opgehaald van InfoQ: http://www.infoq.com/articles/rest-soap-when-to-use-each

Motamarri, J. (2013, 05 08). *Compare RESTful vs SOAP*. Opgehaald van dzone.com: https://dzone.com/articles/j2ee-compare-restful-vs-soap

Subramaniam, P. (2014, 08 15). *REST API Design*. Opgehaald van thoughtworks.com: https://www.thoughtworks.com/insights/blog/rest-api-design-resource-modeling

Kulandai, J. (2012, 11 13). *REST vs SOAP*. Opgehaald van javapapers: http://javapapers.com/web-service/rest-vs-soap/

## 14.4 Bijlage F – Test plan: proof of concept



Test plan– Jonker BV.

Versie 1.0

Jochem Pouwels - 1613727  
Docentbegeleider: Rik Boss

Inhoudsopgave

1 Inleiding 134

2 Test 1 – Productiegegevens van vandaag 134

2.1 Pre 134

2.2 Acties 134

2.3 Vervolg 134

3 Test 2 – Productiegegevens van geselecteerde dag 134

3.1 Acties 134

3.2 Vervolg 134

4 Test 3 – Productiegegevens van vorige dag 135

4.1 Acties 135

4.2 Vervolg 135

5 Test 4 – Productiegegevens van volgende dag 135

5.1 Acties 135

5.2 Vervolg 135

6 Test 5 – Productiegegevens met filtercriteria 135

6.1 Acties 135

6.2 Vervolg 135

7 Test 6 – Extra informatie tonen van een productie 136

7.1 Acties 136

7.2 Vervolg 136

8 Test 7 – Kolomconfiguratie wijzigen 136

8.1 Acties 136

8.2 Vervolg 136

# Inleiding

Dit document bevat het test plan van het proof of concept gemaakt door Jochem Pouwels. Elk hoofdstuk bevat een acceptatietest, het test plan is geslaagd als bij elk controle veld van de acceptatietesten “klopt” kan worden ingevuld.

Elk hoofdstuk bevat de volgende subhoofdstukken:

**Pre:** De voorwaarden voordat de test mag worden uitgevoerd.

**Acties:** Een tabel met gebruikersacties en verwachte resultaten.

**Vervolg:** Te doen nadat alle acties volgens verwachting zijn verlopen.

Indien een controle niet klopt, stop dan met de test en geef de resultaten door aan Jochem Pouwels.

# Test 1 – Productiegegevens van vandaag

## Pre

* Het startmenu is al geopend.

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Start de productiemodule | Een overzicht van alle producties van vandaag verschijnt |  |

## Vervolg

Voer “Test 2” uit.

# Test 2 – Productiegegevens van geselecteerde dag

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Klik op de “kalender” knop | Een kalender invoerscherm verschijnt |  |
| A02 | Selecteer 7 Januari 2013 | Kalender invoerscherm verdwijnt.  producties met nr 91000, 91002 en 91003 verschijnen in het overzicht. |  |

## Vervolg

Voer “Test 3” uit.

# Test 3 – Productiegegevens van vorige dag

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Klik op de “vorige dag” knop | Producties met nr 90001 en 90002 verschijnen in het overzicht. |  |

## Vervolg

Voer “Test 4” uit.

# Test 4 – Productiegegevens van volgende dag

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Klik op de “volgende dag” knop | Producties met nr 91000, 91002 en 91003 verschijnen in het overzicht. |  |

## Vervolg

Voer “Test 5” uit.

# Test 5 – Productiegegevens met filtercriteria

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Klik op de “filter” knop | De filter popup verschijnt |  |
| A02 | Zet de periode op 7 Januari tot en met 8 Januari, klik daarna op de “aan” knop | De filter popup verdwijnt.  Producties met nr 90001, 90002, 91000, 91002 en 91003 verschijnen in het overzicht. |  |
| A03 | Klik op de “filter” knop | De filter popup verschijnt |  |
| A04 | Klik op de “uit” knop | De filter popup verdwijnt.  Producties met nr 91000, 91002 en 91003 verschijnen in het overzicht. |  |
| A05 | Klik op de “filter” knop | De filter popup verschijnt |  |
| A06 | Klik op de “annuleer” knop | De filter popup verdwijnt |  |

## Vervolg

Voer “Test 6” uit.

# Test 6 – Extra informatie tonen van een productie

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Klik twee keer op productierij met nr 91000 | Een informatie popup verschijnt  Het veld “productienr” heeft het getal 91000 |  |
| A02 | Klik een keer op productie rij met nr 91002 | De informatie vertoont andere data  Het veld “productienr” heeft het getal 91002 |  |

## Vervolg

Voer “Test 7” uit.

# Test 7 – Kolomconfiguratie wijzigen

## Acties

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Actie | Verwacht resultaat | Controle |
| A01 | Klik op de instellingen knop | De instellingen popup verschijnt |  |
| A02 | Klik op de checkbox naast “productienr” | De checkbox wisselt van waarde |  |
| A03 | Klik op de “opslaan” knop | De instellingen popup verdwijnt  De kolom genaamd “nr” wordt niet meer getoond. |  |
| A04 | Klik op de instellingen knop | De instellingen popup verschijnt |  |
| A05 | Wijzig de breedte van de “reccode” kolom en klik op de “opslaan” knop | De instelingen popup verdwijnt  De kolom genaamd “reccode” is van breedte gewijzigd. |  |
| A06 | Klik op de instellingen knop | De instelling popup verschijnt |  |

## Vervolg

Lever de test in bij Jochem Pouwels.