Professionele Bachelor Toegepaste Informatica

Softwaremanagement

# Functioneel en automatisch testing

Babette Zengers

Promotoren:

Bram Thys — Refleqt
Wim Vervust — Refleqt
Dries Swinnen — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2017-2018

Professionele Bachelor Toegepaste Informatica

# Functioneel en automatisch testing

Babette Zengers

Promotoren:

Bram Thys — Refleqt
Wim Vervust — Refleqt
Dries Swinnen — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2017-2018

# Dankwoord

Ik zou graag de hogeschool PXL willen bedanken, met name Francis Vos, Marijke Willems, Marijke Sporen, Nathalie Fuchs, Louis Van Gaal en Dries Swinnen. Dankzij jullie heb ik de mogelijkheid gehad om deze stage te mogen uitvoeren.

Ook bedank ik graag alle lectoren over mijn PXL-carrière. Door jullie lessen sta ik nu waar ik moet staan, met de kennis die ik heb kunnen toepassen gedurende mijn stageperiode.

Verder wil ik ook zeker Refleqt bedanken. Jullie bedrijf heeft me laten zien wat testen inhoud. Specifiek hier wil ik graag Wim Vervust en Bram Thys bedanken. Dankzij jullie opportuniteit heb ik een zeer leerrijke stageperiode achter de rug. Het komt ook door jullie vlotte medewerking en feedback dat ik een goed eindproduct heb kunnen afleveren.

Mijn collega's van SmartSpoken wil ik ook graag bedanken, Stijn Coppens, Jan De Belder, Michaël Cuypers en Glenn Van Schil. Jullie vingen me op in de groep alsof ik er al van dag één bij was, en dit maakte mijn stage nog aangenamer.

Mr Dries Swinnen, u wil ik nog eens bedanken, zonder al uw ondersteuning en feedback zouden mijn documenten, en ikzelf niet tot het uiterste gegaan zijn.

Aan familiale steun ben ik ook niets tekortgekomen. Meerdere keren zorgden mijn ouders, zussen, vriend en schoonfamilie voor de steun en opkikkers die ik nodig had. Ze gaven me niet enkel steun maar ook een kritische blik op mijn werkwijze en aanpak.

Allemaal heel erg bedankt!

Babette Zengers

# Abstract

Binnen Xplore Group is twee jaar geleden een project opgestart met als doel het zoekproces naar gedetailleerde producten en diensten te vereenvoudigen op basis van natural language om op die manier klanten met meerdere zoektermen en/of natuurlijke taal een veel correcter zoekresultaat terug te geven.

Het stageproject is opgedeeld in twee deeltaken. Enerzijds is er het reviewen van de 'getting started'-guide van een 'natural language' zoekalgoritme dat geïmplementeerd kan worden op e-commercewebsites. Dit gebeurt aan de hand van de swaggerdocumentatie en implementatie in ReadyAPI. Anderzijds moeten er functionele en automatische testen worden gebouwd en geautomatiseerd met behulp van ReadyAPI. Deze testen worden op regelmatige tijdstippen uitgevoerd met behulp van Jenkins.

De focus van de onderzoeksopdracht sluit hierbij aan en ligt op de impact van 'model based testing' op testautomatisatie. Een literatuurstudie beschrijft de voor- en nadelen van 'model based testing' en gaat na hoe deze methode toepasbaar is op testautomatisatie.

Het einddoel is een zo kwalitatief mogelijk eindproduct te verwezenlijken en regressie tegen te gaan.

# Inhoudsopgave

## Inhoud

# Lijst van gebruikte figuren

# Lijst van gebruikte tabellen

# Lijst van gebruikte afkortingen en begrippen

| | |
|---|---|
| BDD | Behaviour Driven Development: is een manier van programmeren waarbij eerst het gedrag beschreven wordt alvorens men daadwerkelijk gaat programmeren. In de praktijk bouwt dit op de principes van Domain Driven Design en Test Driven Development. |
| DDD | Domain Driven Design: Dit is een ontwikkelmethodiek die zeer sterk de focus heeft op domein waarvoor software wordt ontwikkeld. |
| TDD | Test Driven Development: Dit is een ontwikkelmethode voor software waarbij eerst tests worden geschreven en daarna pas de code. |
| SPOC | Single Point Of Contact: De mogelijkheid klanten één enkel contactpunt aan te bieden voor alles wat betrekking heeft op de geleverde diensten vormt voor dit soort diensten een concurrentievoordeel. |
| MBT | Model Based Testing: is het automatisch genereren van testgevallen, met behulp van abstracte modellen gebaseerd op de requirements en het gedrag. |
| LUM | Language Understanding Model: tekst input voor een meer gestructureerd representatie van een model. |
| IWT | Agentschap voor Innovatie door Wetenschap en Technologie |
| O&O | Een individueel project van onderzoek of ontwikkeling. |
| GUI | Graphical User Interface |
| Jenkins | Is een open source automation server geschreven in Java. Jenkins helpt om het niet menselijke deel van een software development process te automatiseren. Dit doet hij met behulp van continuous integration en technische aspecten van continuous delivery. |
| PoC | Proof of Concept. Een methode om aan te tonen dat de oplossing voldoet of kan voldoen aan de specificaties die de klant vraagt. |
| NLP | Natural language processing. Een wetenschap-techniek die zich bezighoudt met verwerking van mensentaal door computer. |
| SUT | System under test. Deze verwijzen naar een systeem dat gevalideerd is door de testers. Het komt ook overeen met de software die klaar gemaakt is en door de unit en integratie testen is geweest. |
| MDD | Model-driven-development: Dit is een aanpak die modellen gebruikt om een product te creëren. |
| IBM | International Business Machines Corporation |
| UML | Undefined Modeling Language is een modelmatige taal om object georiënteerde analyses en ontwerpen voor een informatiesysteem te kunnen maken. |
| SysML | Systems Modeling Language: is een algemene modelleertaal voor technische toepassingen-systemen. Het ondersteunt de specificatie, analyse, ontwerp, verificatie en validatie van een breed scala van systemen en systemen-van-systemen. |
| ATG | Automatic Test Generation: een belangrijk onderdeel van software testen. ATG is het creëren van een set gegevens voor het testen van de geschiktheid van nieuwe of herziene softwaretoepassingen. |
| UTP | UML testing profiel |

| | |
|---|---|
| VB | Visual basic |
| CI | Continuous Integration is een term die vaak wordt gebruikt voor softwareontwikkeling. CI is eigenlijk niets meer, of minder, dan een aanpak die ervoor zorgt dat tijdens de gehele ontwikkeling van een product de code continu wordt gecontroleerd. |
| CD | Continuous Delivery is een softwareontwikkelmethode die gericht is op ideeën zo snel en efficiënt mogelijk in productie te krijgen. |
| IDE | Integrated development environment: een software-ontwikkelomgeving voor programmeurs. |
| MID | Model-Implementation-Description |
| MIM | Model-Implementation-Mapping |
| IOCO | Input-Output Conformance |
| B2B | Business to business |
| PoC | Proof of concept |

# Inleiding

In de e-commerce sector vragen steeds meer klanten om nieuwe features sneller en kwalitatiever op te leveren. Het spreekt voor zich dat deze features allemaal vlekkeloos werken op alle mogelijke browsers en devices. Om aan deze vraag te voldoen moeten we meer en meer inzetten op automatisatie van het test process.

Hierdoor zijn we begonnen met het schrijven van automatische functionele testen. Het testen van software of applicaties is een kritisch punt dat sterk afhangt van de kwaliteit van de testen. Hoe beter je testen, hoe beter de kwaliteit van je software of applicatie. Als er niet getest wordt is de kans groter dat het eindproduct met veel fouten naar de klanten gaat. Als er wel getest wordt is deze kans veel kleiner. Dit brengt een zeer groot nadeel met zich mee, namelijk het drijft je klanten weg.

Bij het testen van software of applicaties komen er veel vragen kijken. Enkele hiervan zijn: welke testen gaat men schrijven? Gaat men gebruik maken van model based testing? Welke tool gaat men gebruiken? Dit zijn vragen waarvan ik mijn best ga doen om deze zo correct mogelijk te beantwoorden.

Het schrijven van goede tests is niet eenvoudig. Je moet enige kennis hebben van programeer en scripting talen. Wanneer weet je nu als je een goede test geschreven hebt? Dit is een vraag die elke tester zich stelt. Je kan als tester beter zo veel mogelijk bugs vinden in de software of de applicatie, zo kunnen de developers dit oplossen voor deze naar de klant gaat. Minder bugs bij de klant zal dan ook uitdraaien op een blije klant.

Het is verder ook zeer belangrijk dat je eindproduct niet enkel getest wordt door automatische functionele testen, maar ook door andere manieren van testen, zoals performance testing, security testing, usability testing, … Hierdoor kan je nog meer bugs vinden en optimaliseer je je eindproduct steeds meer.

Testen blijft van groot belang voor het succes van een bedrijf. Zonder tests heeft het eindproduct vaak een grote kans op mislukking.

# I. Stageverslag

# 1 Bedrijfsvoorstelling

## 1.1 Refleqt


*Figuur 1 Logo van Refleqt*

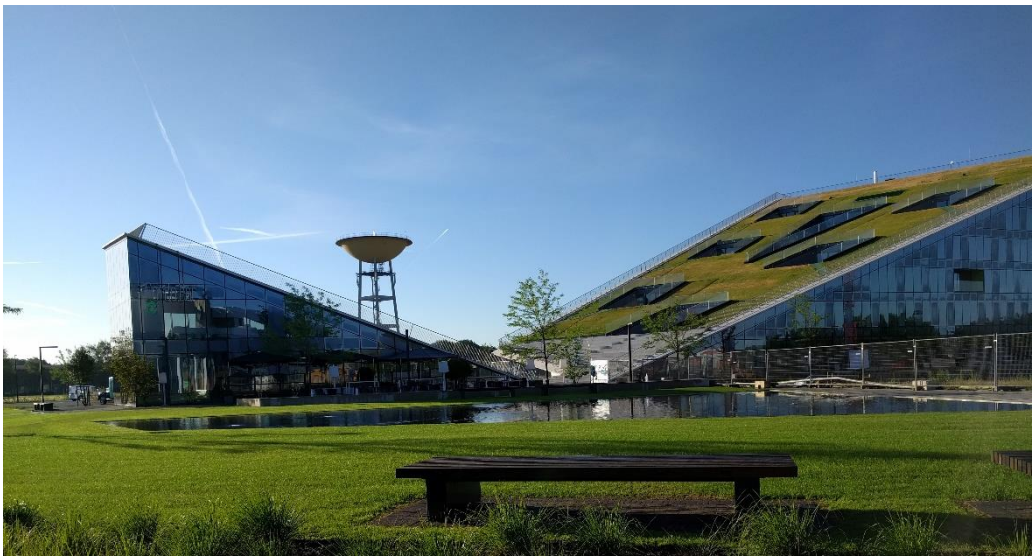
Refleqt is een bedrijf opgericht in juni 2017 door en voor quality engineers binnen Xplore Group (Cronos). Ze bestaan momenteel uit 15 medewerkers met hun hoofdvestiging te Kontich.
Verder heeft Refleqt ook een tweede vestiging in de kantoren van Xplore Group in Hasselt.

Refleqt is zeer sterk in testautomatisatie. Ze leggen vooral de focus op API testing, UI testing, Non-functioneel testing en continuous delivery. Hierbuiten staan ze ook zeer sterk in het aanleveren van functionele automatische testen en performantietesten. Dit met als hoofddoel regressie tegen te gaan wat bijgevolg ook hun Unique Selling Point is. Ze doen dit met behulp van volgende tools:

- Swagger & Swagger-Codegen
- Apium
- Selenium
- CI-Servers: Jenkins, Bamboo
- IDE-Omgeving: Intellij, Eclipse, PHPStorm
- Behaviour Driven Frameworks: Cucumber, Jbehave, SpecFlow, Behat
- Codeception
- ReadyApi
- Jmeter, Gatling
- Artifactory
- Atlassian stack, Jira, Confluence, Bitbucket

Binnen het Refleqt team krijgen stagiairs de positie test automation engineer. Hiermee kunnen ze dan aan de slag op een project waarop ze dan kunnen beginnen met het schrijven en automatiseren van testen. Ook zal er voor elke stagiair de nodige software en licenties worden voorzien. Zo kunnen ze hun stage goed uitwerken zonder zelf voor de licenties moeten betalen.

Hiernaast volgt Refleqt het hele development en testing proces mee. Verder staan ze ook in voor het analyse gedeelte met technieken zoals behaviour driven development (BDD). Tot slot zijn ze de single point of contact (SPOC) binnen het development team voor functionele vragen



*Figuur 2 Vestiging Refleqt in Hasselt. Corda 1, verdieping 5*


*Figuur 3 Vestiging Refleqt in Kontich, verdieping 2*

## 1.2 Achtergrondinformatie SmartSpoken

Het verhaal van SmartSpoken begon in 2014 aan de KU Leuven, met het MIX-ICON-project Sunshine. Eén van de doelen van dit project was een virtuele assistent te maken, die journalisten toelaat om met natuurlijke taal spreadsheets te bevragen en te visualiseren. Bijvoorbeeld, met een Excel document met daarin de statistieken van het aantal inbraken per gemeente doorheen de jaren, konden de journalisten vragen stellen zoals "in welke 10 streken is het aantal inbraken het meeste toegenomen". De partners in het project waren, naast de KU Leuven, onder andere Mediahuis en VRT.

Begin 2015 is SmartSpoken officieel opgericht, met als doel de technologie uit het Sunshine project te commercialiseren. Snel na de oprichting werd SmartSpoken geselecteerd voor Start-it @KBC. Hierdoor kregen ze kantoorruimte ter beschikking in Leuven. Omdat de markt voor een business intelligence toepassing van de technologie moeilijker was dan verwacht, begon SmartSpoken andere markten te verkennen, zoals e-commerce.

In de zomer van 2015 zijn ze op de radar van Cronos terecht gekomen dat toen net bezig was met het opstarten van The CoFoundry, een ecosysteem dat investeringen doet in startups. Einde zomer 2015 had SmartSpoken hun eerste klant, OmniaTravel een reisbureau uit Leuven. Dankzij The CoFoundry werden er personeelsleden van Xplore Group beschikbaar gemaakt voor het SmartSpoken team. Dit zijn Stijn Coppens, Michaël Cuypers en Glenn Van Schil. Een half jaar lang hebben ze gewerkt aan de OmniaTravel implementatie, waarbij klanten kunnen zoeken naar vakanties m.b.v. natuurlijke taal, zoals bijvoorbeeld "1 week in de paasvakantie naar Madrid voor 2 personen”.

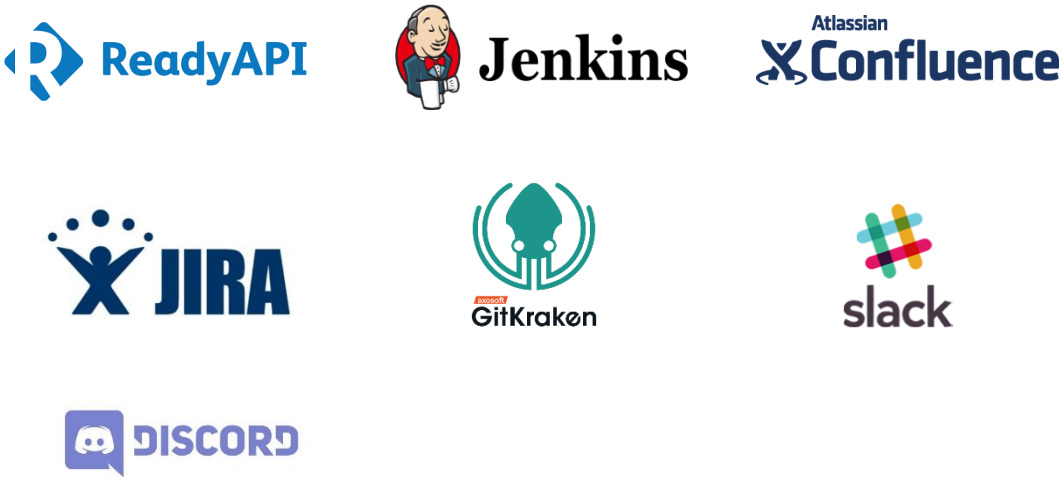
Na de succesvolle oplevering van het OmniaTravel project werkten ze verder aan de e-commerce toepassing, met financiële steun van het Agentschap voor Innovatie door Wetenschap en Technologie (IWT) in de vorm van een O&O subsidie (februari 2016). Na een half jaar werken aan de e-commerce toepassing bleek dat ze enkele heel geavanceerde features hadden, maar een gebrek aan de basis features die nodig zijn voor een e-commerce zoekoplossing. De ontwikkeling van de e-commerce zoekoplossing werd on-hold gezet en de aandacht ging weer terug naar het verder verfijnen de basis natural language processing (NLP) technologie. Momenteel, november 2017, ligt de focus op het uitwerken van Proof of Concept (PoC), de grafische interface (GUI) en automated testing.

# 2 Stage

## 2.1 Voorstelling stageopdracht

Deze stageopdracht is tot stand gekomen doordat SmartSpoken een functionele tester nodig had. Het primaire doel van deze stageopdracht is functionele automatische testen schrijven zodat SmartSpoken het product kan optimaliseren.

Onderstaande afbeeldingen geven een overzicht van de verschillende tools waarmee ik gewerkt heb:




*Tabel 1 Logo's gebruikte tools*

Knowledge based tools zoals Confluence, issue tracking tools zoals Jira, ReadyAPI voor testautomatisatie, Git als technologie voor de code op te slaan met de tool GitKraken. Jenkins als platform bedoelt voor het herhaaldelijk uitvoeren en monitoren van build-taken. Slack voor communicatie met de bedrijfspromotor en Discord voor communicatie met het SmartSpoken team.

Op Confluence kwamen de sprint retrospectives terecht. Aan het einde van elke sprint zit iedereen samen om zo van mekaar te horen wat er beter kon en wat er goed gegaan was. Verder was er ook een kalender aanwezig op Confluence. Op deze kalender stond waar het team zit, Hasselt of Kontich, en wanneer er iemand verlof had. Zo konden we beter op mekaar afspelen en samen als team op één locatie zitten. Vervolgens werd er dagelijks een scrum meeting gehouden om zo op de hoogte te zijn van wie er met welke taak bezig was.

Om te kunnen werken met een Kanban bord, maakten we onder andere gebruikt van Jira. Hier kwamen alle developers taken en bugs op terecht. Ook konden we hier zien wie de bug gevonden had, het ticketje had aangemaakt, wie met welke taak bezig was en wie welke taak op ‘done’ gezet had.

ReadyAPI was de API waarin de functionele automatische testen geschreven werden aan de hand van Groovy. Jenkins werd gebruikt om de testen automatisch te laten lopen op hun server.

## 2.2 Uitwerking stageopdracht

Door vrijwel elke dag een stand-up meeting en elke sprint een sprintplanning en sprint retrospective te houden, was de transparantie binnen het team heel hoog. Iedereen wist waar de ander mee bezig was. Als er bugs tevoorschijn kwamen moesten er tickets aangemaakt worden op Jira. Zo waren de betrokken ontwikkelaars op de hoogte van de bugs. Elke bug werd nagekeken, gemaakt en hertest.

Een eerste taak bij het begin van de stage was het kritisch reviewen van de documentatie van SmartSpoken. Dit had als bijkomend voordeel dat ik de structuur en de mogelijkheden van de SmartSpoken API leerde kennen. Hierna kon er begonnen worden aan het creëren van de functionele testen. Tot slot was de laatste stap deze testen automatisch te maken.

### 2.2.1 Planning

Het is zeer belangrijk om een representatieve planning te maken. Dit is niet gemakkelijk, voornamelijk omdat er taken komen die men overschat of onderschat.

Onderstaande afbeelding is de eerste planning die er opgesteld werd.

| | | | | |
|---|---|---|---|---|
| | Stage semester 1 | 75 dagen | maa 2/10/17 | vri 12/01/18 |
| | **Sprint 1** | **5 dagen** | **maa 2/10/17** | **vri 6/10/17** |
| | Introductie | 0,5 dagen | maa 2/10/17 | maa 2/10/17 |
| | Orienterende studie API | 2 dagen | maa 2/10/17 | din 3/10/17 |
| | Planning opstellen | 0,5 dagen | din 3/10/17 | din 3/10/17 |
| | Api documentatie lezen | 2 dagen | woe 4/10/17 | don 5/10/17 |
| | Project opstellen | 1 dag | vri 6/10/17 | vri 6/10/17 |
| | **Sprint 2: Project fase 1 getting started** | **25 dagen** | **maa 9/10/17** | **vri 10/11/17** |
| | Api documentatie lezen, volgen en feedback geven | 12 dagen | maa 9/10/17 | din 24/10/17 |
| | Testdata ontwikkelen | 5,5 dagen | woe 25/10/17 | woe 1/11/17 |
| | Positieve testcases opstellen | 7,5 dagen | woe 1/11/17 | vri 10/11/17 |
| | **Sprint 3: Project fase 2: uitbreiding** | **30 dagen** | **maa 13/11/17** | **vri 22/12/17** |
| | Vertraagd werk inhalen waar nodig | 2 dagen | maa 13/11/17 | din 14/11/17 |
| | Opzetten van Jenkins | 3 dagen | woe 15/11/17 | vri 17/11/17 |
| | Uitbreiden testdata en core testen van de zoekfunctionaliteit | 12 dagen | maa 20/11/17 | din 5/12/17 |
| | Uitbreiden testcases zowel positieve als negatieve testen | 13 dagen | woe 6/12/17 | vri 22/12/17 |
| | Vakantie | 11 dagen | maa 25/12/17 | zon 7/01/18 |
| | **Sprint 4: Eindfase** | **5 dagen** | **maa 8/01/18** | **vri 12/01/18** |
| | Project reviewen | 1 dag | maa 8/01/18 | maa 8/01/18 |
| | Voorbereiding eindpresentatie + demo | 2 dagen | din 9/01/18 | woe 10/01/18 |
| | Demo geven | 1 dag | don 11/01/18 | don 11/01/18 |
| | Eindpresentatie oefenen | 1 dag | vri 12/01/18 | vri 12/01/18 |
| | Dag juryexamen | 1 dag | maa 29/01/18 | maa 29/01/18 |

*Figuur 4 Eerste planning*

Zoals men kan waarnemen, komen de twee planningen, in figuren 'figuur 3 Eerste planning' en 'figuur 4 Effectieve planning', grotendeels overeen. Er zijn aanpassingen gebeurd aan de effectieve planning omdat er sommige taken onder of overschat werden. Verder neemt men waar dat de planning opgedeeld is in vier sprints. Sprint twee was de eerste effectieve fase van het project. Hierin werd er bij het aanmaken van projecten, enkel gebaseerd op documentatie. In fase twee werd er dan weer geen rekening gehouden met documentatie maar moest er zelf een LUM opgesteld worden. Sprint één was het voorbereiden van testen en ReadyAPI leren kennen en sprint vier is voornamelijk voor presentaties te oefenen en meer te documenteren.

| | | | | |
|---|---|---|---|---|
| | Stage semester 1 | 75 dagen | maa 2/10/17 | vri 12/01/18 |
| | **Sprint 1** | **5 dagen** | **maa 2/10/17** | **vri 6/10/17** |
| | Introductie | 0,5 dagen | maa 2/10/17 | maa 2/10/17 |
| | Orienterende studie API | 2 dagen | maa 2/10/17 | din 3/10/17 |
| | Planning opstellen | 0,5 dagen | din 3/10/17 | din 3/10/17 |
| | Api documentatie lezen | 2 dagen | woe 4/10/17 | don 5/10/17 |
| | Project opstellen | 1 dag | vri 6/10/17 | vri 6/10/17 |
| | **Sprint 2: Project fase 1 getting started** | **25 dagen** | **maa 9/10/17** | **vri 10/11/17** |
| | Api documentatie lezen, volgen en feedback geven | 11 dagen | maa 9/10/17 | maa 23/10/17 |
| | ZIEK | 5 dagen | don 12/10/17 | woe 18/10/17 |
| | Test project automatisch maken | 11 dagen | din 24/10/17 | din 7/11/17 |
| | Testdata ontwikkelen | 3 dagen | woe 8/11/17 | vri 10/11/17 |
| | **Sprint 3: Project fase 2: uitbreiding** | **30 dagen** | **maa 13/11/17** | **vri 22/12/17** |
| | Vertraagd werk inhalen waar nodig (Positieve testcases opstellen) | 3 dagen | maa 13/11/17 | woe 15/11/17 |
| | Opzetten van Jenkins | 3 dagen | don 16/11/17 | maa 20/11/17 |
| | Uitbreiden testdata en core testen van de zoekfunctionaliteit | 13 dagen | din 21/11/17 | don 7/12/17 |
| | Feedback verwerken | 11 dagen | vri 8/12/17 | vri 22/12/17 |
| | **Sprint 4: Project fase 3: eindfase** | **5 dagen** | **maa 8/01/18** | **vri 12/01/18** |
| | Demo geven | 1 dag | maa 8/01/18 | maa 8/01/18 |
| | Project reviewen | 1 dag | din 9/01/18 | din 9/01/18 |
| | Voorbereiding definitieve eindpresentatie + demo | 2 dagen | woe 10/01/18 | don 11/01/18 |
| | Eindpresentatie oefenen | 1 dag | vri 12/01/18 | vri 12/01/18 |
| | Dag juryexamen | 1 dag | don 1/02/18 | don 1/02/18 |

*Figuur 5 Effectieve planning*

De eerste sprint verliep zonder moeite, alle deadlines werden gehaald. De tweede sprint daarentegen verliep moeizamer. De voornaamste reden hiervoor is afwezigheid door ziekte. Hierdoor zijn er deadlines niet gehaald. Vervolgens is er een inhaalmanoeuvre moeten gebeuren om terug op schema te geraken. Bij het begin van sprint drie werd er terug volgens schema gewerkt.

### 2.2.2 Stageverloop

Voor het mogelijk was om deze stageopdracht uit te voeren moest er eerst kennis gemaakt worden met ReadyAPI. Hierin moesten er functionele testen geschreven worden voor het project SmartSpoken. De bedoeling is dat als er een vraag stel wordt, er ook de juiste response voor terugkomt. De stageopdracht werd opgedeeld in drie deeltaken:

#### 2.2.2.1 Projectfase één

De eerste deeltaak (fase 1) van het stageproject bevat het nabootsen van de gedachtegang een klant. Hier werd er enkel rekening gehouden met de documentatie van SmartSpoken die er gegeven was. In deze documentatie staat er onder andere genoteerd hoe hun Swagger werkt, welke velden er bestaan en aangemaakt kunnen worden, etc. Dus met andere woorden er werden nieuwe projecten aangemaakt met behulp van de documentatie. Indien er iets niet duidelijk was, werd dit doorgegeven aan de ontwikkelaars. Deze verbeterden dit in hun documentatie zodat een klant ook effectief een project kan aanmaken aan de hand van hun documentatie zonder enige hulp.

Door de documentatie volledig te overlopen is er testcode geschreven. Dit is een ReadyAPI project waarin er testen geschreven werden. Deze testen werden opgesteld door Groovy scripts en een Excel bestand. Verder werd er ook gezorgd dat de testen dynamisch opgesteld waren, met andere woorden wanneer er een ander project gebruikt moest worden kon dit in het Excel bestand gestoken worden die gebruikt wordt in het ReadyAPI project. Dankzij deze testen zijn er een aantal bugs naar voren gekomen. Sommige met meer prioriteit dan andere. De bugs met de hoogste prioriteit waren de ‘500 Internal Server Error’. Deze bugs mogen onder geen enkele voorwaarde optreden. '500 Internal server error' bugs zijn fouten die optreden in de broncode van een applicatie of server. In ons geval traden deze bugs op omwille van kritische fouten in de SmartSpoken applicatie. Deze bugs dienen z.s.m. opgelost te worden omdat ze het hele systeem kunnen schaden. In totaal zijn er 24 bugs in totaal gevonden. Wanneer er een bug gevonden werd, deed het team een root cause analysis, waarbij het nog kan zijn dat de fout aan de onjuistheid van de test ligt i.p.v. aan de applicatie. Wanneer de fout bij de software lag werd deze zo snel mogelijk opgelost, maar wanneer het aan de test zelf lag, moest men het testscript gaan updaten. Verder moesten ze ook altijd gemeld worden in Jira, zo wisten de ontwikkelaars als er een bug gevonden was en konden ze deze zo snel mogelijk oplossen. Onderstaand een screenshot hoe de tickets er uitzien in Jira.

| | |
|---|---|
| ELISMASK-1357 | Smartsynonyms |
| ELISMASK-1351 | Onderzoek wat te doen met spaties voor of na een string |
| ELISMASK-1294 | Getting started: Starting the project |
| ELISMASK-1296 | Getting started: Dates & Expressions & Subtypes / custom operators |
| ELISMASK-1298 | Getting started: Subtypes / custom operators |
| ELISMASK-1300 | Smartspoken API Documentatie: Quality report |
| ELISMASK-1302 | Smartspoken API Documentatie: create fieldfamily |
| ELISMASK-1301 | Smartspoken API Documentatie: create fieldfamily |
| ELISMASK-1338 | Make sure the pricing overview is documented on confluence, incl. AWS cost + ElasticCloud cost + other |
| ELISMASK-1329 | Content-Type/Accept |
| ELISMASK-1326 | WordsFrequencies: id met een % |
| ELISMASK-1327 | Fieldhierarchy: 500 |
| ELISMASK-1299 | Update API documentation to latest version to incl. status |
| ELISMASK-1319 | Smartspoken API Documentatie: get dictionary |
| ELISMASK-1322 | Smartspoken API Documentatie: updateDateValue |
| ELISMASK-1323 | Importing LUM with same ID as an analytics project throws 500 |
| ELISMASK-1324 | 500 when autocompleting |
| ELISMASK-1328 | Asking a MergedQI gives no result but asking a QI does give result |
| ELISMASK-1314 | Smartspoken Documentatie API: Create FieldHierarchies |
| ELISMASK-1320 | Smartspoken API Documentatie |
| ELISMASK-1337 | Smartspoken API Documentatie: Create new text values |
| ELISMASK-1358 | SmartSpoken Terms |
| ELISMASK-1380 | [Analyse] Smartspoken: 1m50 of 150cm |
| ELISMASK-1381 | Smartspoken API: Foute waarde terugkrijgen |

*Figuur 6 Bug tickets Jira*

#### 2.2.2.2 Projectfase twee

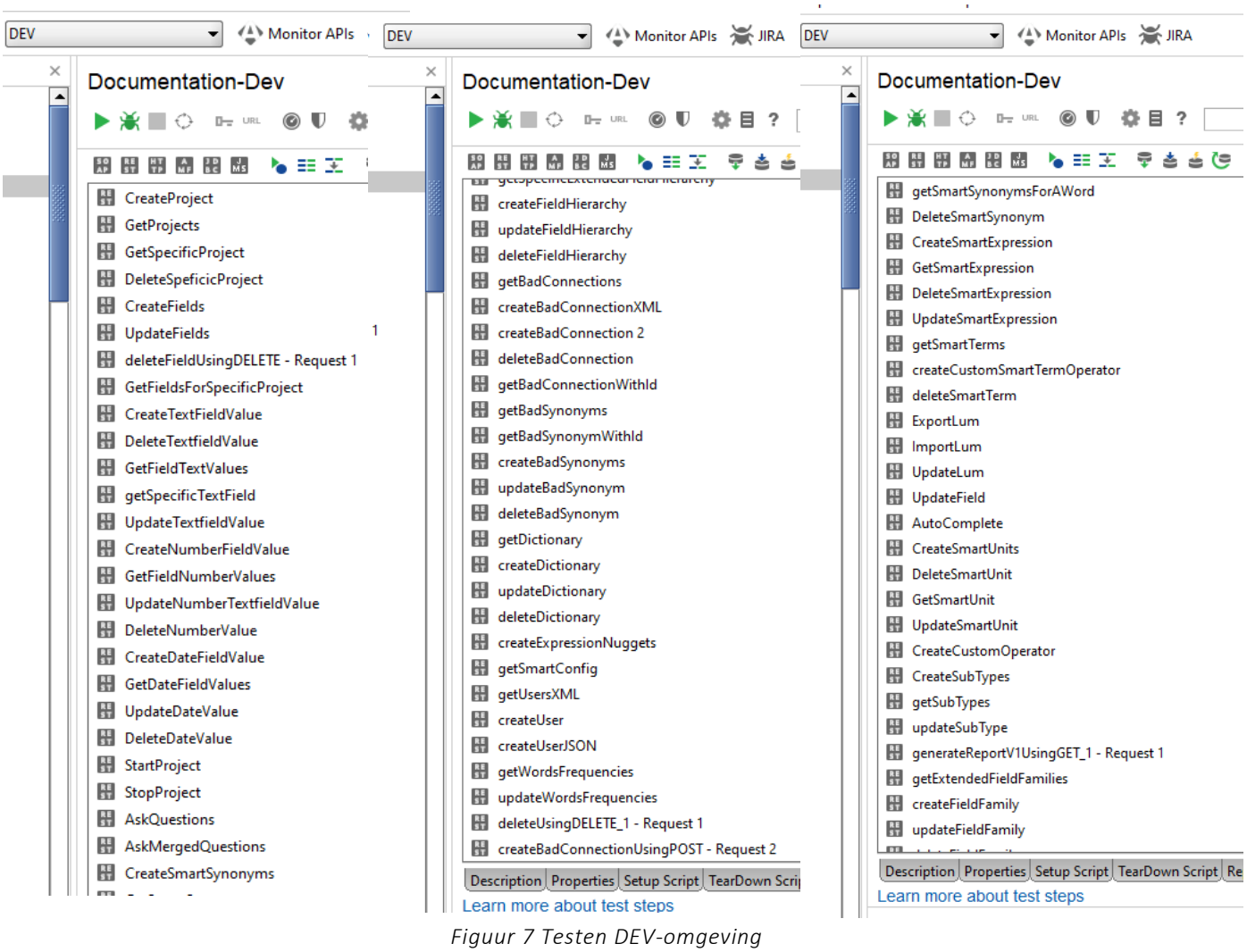
De tweede deeltaak (fase 2) was het runnen van de testen op beide werk omgevingen. Dit zijn de DEV en de TEST omgeving. De TEST omgeving is de omgeving waarop de developers code opslaan waarvan ze weten dat deze werkt en meestal bug vrij is. De DEV-omgeving is dan de omgeving waar alle code eerst geïnstalleerd en getest wordt. Indien er hieruit bugs tevoorschijn komen worden deze opgelost en opnieuw geïnstalleerd. Wanneer er dan nog eens getest wordt op deze omgeving en alle testen slagen, zal de code op de TEST omgeving terecht komen. Dit wil dus zeggen dat de documentatie van SmartSpoken op regelmatige tijdstippen opnieuw getest moet worden op beide omgevingen. Doordat men in ReadyAPI gemakkelijk verschillende omgevingen kan toevoegen, vergemakkelijkt het dan ook om op verschillende omgevingen te testen.

Het grote voordeel van testen op verschillende omgevingen is wanneer er op de een omgeving een bug naar voren komt, dit daarom niet op de andere omgeving gebeurt. Ook kan het gebeuren dat er bij het opslaan van de DEV-omgeving naar de TEST omgeving conflicten tevoorschijn komen. Hierdoor kunnen er nieuwe bugs tevoorschijn komen die er op de TEST omgeving niet waren. Vervolgens is het zeer belangrijk dat er altijd op beide omgevingen getest en op regelmatige tijdstippen opnieuw getest wordt. Bij het opnieuw testen kan men waarnemen als het oplossen van de ene bug niet resulteert in het veroorzaken van een nieuwe bug.

Zodra de documentatie volledig getest en gereviewd was, zowel op de TEST omgeving als op de DEV-omgeving, moest erop nog een andere manier getest worden. Deze manier was het aanmaken van een eigen language understanding model (LUM). De bedoeling hier was dat SmartSpoken kan zien welke soort gegevens een klant voornamelijk gaat gebruiken indien hij zelf de LUM moet aanmaken. Bijvoorbeeld: een klant zou eerder naar synoniemen grijpen dan naar smartexpressions. Hier is er een persoonlijk project ontworpen waardoor ze mij als klant gingen bekijken. Dus met andere woorden werd er gevraagd om een persoonlijk project op te stellen met zoveel mogelijk gegevens. Dit met als grootste doel om verwarring proberen te creëren. Doordat er een grote hoeveelheid aan gegevens gecreëerd werd, kwamen er nog gegevens naar boven die verkeerd geïnterpreteerd werden. Met de resultaten van dit project kan het team van SmartSpoken meer verduidelijking toevoegen in hun documentatie over wanneer men welk veld kan gebruiken. (Bijvoorbeeld: synoniem of expressies)

### 2.2.2.3 Functionele testen ReadyAPI

Het schrijven van de functionele testen werd gedaan in een tool genaamd ReadyAPI. Hier werd de Swagger van SmartSpoken geïmporteerd om zo de nodige requests al in ReadyAPI te krijgen. Wanneer er een bepaalde test geschreven werd moest deze request toevoegen aan de testcase hiervoor. Hier moest er dan nog zorgen voor de juiste request body en variabelen. Volgende afbeeldingen zijn een voorbeeld van enkele requests die er gebruikt zijn om de documentatie van SmartSpoken op de DEV-omgeving te testen. Dezelfde testen werden ook gebruikt om op de TEST omgeving te testen.




*Figuur 7 Testen DEV-omgeving*

De reden waarom er in veel woorden ‘smart’ voorkomt is omdat dit een referentie is naar SmartSpoken. Bepaalde stappen komen elk project terug, namelijk:

- CreateProject
- CreateFields
- AddFieldValues
- StartProject
- AskQuestions
- StopProject

De reden waarom deze stappen bij elk project terug komen is omdat dit de belangrijkste stappen zijn. Het creëren van een project (createProjects) is het effectieve aanmaken van een project. In deze teststap wordt er een naam en een taal toegewezen aan het project. Dit kan de klant zelf doen door middel van de Excel in te vullen die later in dit document wordt uitgelegd.

Het creëren van velden (createFields) is ook zoals de naam zelf zegt, het aanmaken van velden. Hier moet de klant in de Excel een naam, type en waardes meegeven per veld. De types van de velden kunnen variëren van type tekst, getal, boolean of datum. De naam en de waardes die de klant wil meegeven is volledig vrij te kiezen voor de klant.

Het effectieve toewijzen van de naam en de waardes aan de bepaalde velden wordt gedaan in de volgende teststap, namelijk in het toevoegen van de veldwaardes (AddFieldValues). Tussen het aanmaken van deze waardes en het starten van het project zijn er uiteraard nog een aantal teststappen, maar deze variëren van project en komen daarom niet elk project terug.

Het starten van het project (StartProject) is ook effectief zoals de naam zegt, het starten van het project. Indien dit niet gebeurt kan men niet naar de volgende teststap gaan en vragen stellen aan de API. De reden waarom het project eerst gestart moet worden is omdat deze stap ervoor zorgt dat alle data die er aangemaakt is ook effectief in het project geladen wordt. Wanneer er tijdens of na dat het project opgestart is, data veranderd moet het project herstart worden om de nieuwe data ingeladen te hebben.

Na het starten van het project kunnen we vragen beginnen te stellen (AskQuestions). Ook voor deze stap moet er een Excel file ingevuld worden. Hier wordt er de vraag en het te verwachte resultaat ingevuld. Op deze manier worden de testen opgesteld en wordt het te verwachte resultaat vergeleken met het effectieve resultaat. Indien beide overeenkomen zal de test slagen, zo niet stopt het project op de bepaalde vraag die faalt en moet er gekeken worden waarom dit niet overeenkomt. Dit kan variëren van een komma of spelfout tot een verkeerde set. (Een set is een tabel met gegevens. Sommige vragen zullen meerdere sets bevatten omdat er een én of een óf in de vraag staat).

Wanneer alle testen geslaagd zijn wordt het project gestopt (stopProject).

Voor de teststappen die projectafhankelijk zijn hebben we in dit voorbeeld volgende teststappen:

- SmartSynonym
- SmartExpression
- BadSynonym
- BadConnection
- SmartTerms
- FieldHierarchy
- Import/export LUM

Bovenstaande teststappen zijn voornamelijk voor het project te optimaliseren. SmartSynonym is het toevoegen van synoniemen. Het kan voorkomen dat er synoniemen gebruikt worden voor een bepaalde term. Wanneer de synoniemen niet geconfigureerd zijn in het tabblad synonym in Excel, gebeurt het dat de bijhorende vraag zal falen. Hier gaat hij vermelden dat hij de bepaalde term 'synoniem' niet kent. Belangrijk is wel dat bij het configureren van deze synoniemen het laatste woord de effectieve term is waarop dit synoniem zal gelden. Een voorbeeld hiervan is: eten, voeding. Met andere woorden zal er een synoniem aangemaakt worden genaamd eten voor de waarde voeding. Dus als er een vraag gesteld wordt met "eten is vlees" of "vlees eten", zal de API gaan kijken bestaat er hier een synoniem voor? Ja, voeding. Voeding is de naam van het veld dus wordt dit gekoppeld aan het veld Voeding. Dus komt er als uitkomst hier voeding = vlees.

SmartExpressions daarentegen zijn geen synoniemen maar expressies. Het verschil tussen beide kan verwarrend zijn voor de klant. Een expressie in dit voorbeeld is carnivoor voor het veld voeding met de waarde vlees. Het verschil tussen beide is dat bij expressions dit niet enkel aan een veld wordt toegewezen, maar ook aan een bepaalde content. Dus als er nu een vraag komt met "carnivoor" zal het antwoord ook zijn: voeding = vlees.

BadSynonym en BadConnection zijn testen voor het toevoegen van foute waardes. Een voorbeeld hier is pad. Dit kan bezien worden als het dier maar het kan ook bezien worden als een pad waarop men wandelt. Wanneer er hier een BadConnection of BadSynonym voor wordt aangemaakt zal de API weten wat hij moet nemen. Deze stappen moeten heel uitzonderlijk worden toegepast. Dit omdat SmartSpoken zelf al een lijst heeft met BadSynonyms en BadConnections.

SmartTerms is ook een voorbeeld van een stap die uitzonderlijk moet worden toegepast. Het kan voorkomen dat er vragen gesteld worden en een foutief resultaat terugkomt. Bijvoorbeeld meer dan, hier wil men >= terugkrijgen in de response. Maar doordat de API dit niet kent zal er een SmartTerm voor moeten aangemaakt worden. Dit is een moeilijke teststap, want niet alleen de term die men wil gebruiken zal moeten worden toegevoegd, maar indien er al een term bestaat waar men >= gebruikt, moet men alle keywoorden die al aangemaakt zijn, worden toegevoegd bij het updaten van deze stap. Wordt dit niet gedaan dan gaan al deze keywoorden verloren en zal er enkel het keywoord toegevoegd worden dat het Excel bestand zegt. Er is nog een tweede mogelijkheid dat is het aanmaken van een volledige nieuwe term. Wanneer er nog geen bestaande term bestaat voor >= zal de gebruiker of SmartSpoken deze moeten aanmaken. Belangrijk hier is dat de naam die aan de term wordt gegeven automatisch overschreven wordt. De reden waarom SmartSpoken dit zo geconfigureerd heeft is omdat ze zo een onderscheid kunnen maken tussen de terms die zij of de klant hebben aangemaakt.

Bij het maken van een FieldHierarchy wordt er een link gelegd tussen twee velden. Deze stap is ook project afhankelijk en zal dus niet voor elk project moeten gebeuren. Wanneer het voorkomt dat er een connectie gemaakt moet worden tussen twee velden, moet dit ook enkel ingevuld worden in de Excel. Deze kan ingevuld worden of leeggelaten worden. Het project in ReadyAPI zal altijd een check doen om te kijken indien deze ingevuld is of niet. Wanneer deze ingevuld is, zal er ook daar een link aangemaakt worden tussen de twee velden. Een voorbeeld hiervan is er wordt een FieldHierarchy aangemaakt met de naam diersoorten voor het veld diersoorten. Deze wordt gelinkt aan het veld types waardoor de diersoort Vogels gelinkt kan worden aan een arend.

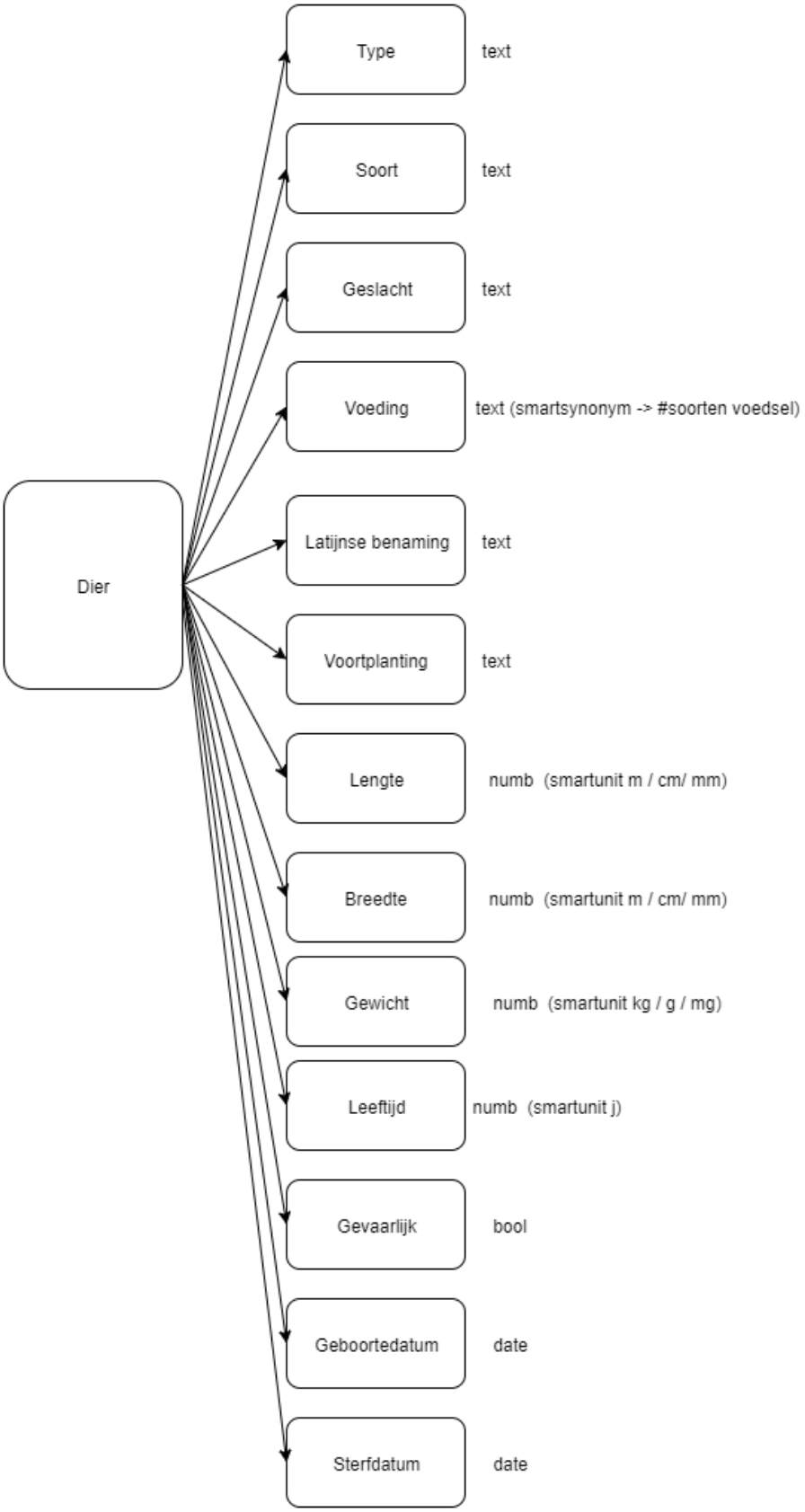
Tot slot zijn er de stappen import en export van de LUM. Dit is ook zoals de stap zelf zegt, het importeren en exporteren van een LUM. Wanneer er een project wordt aangemaakt waarvan een LUM al bestaat, kan dit met de teststap import LUM snel en eenvoudig aangemaakt worden. Wanneer het project gecreëerd is, en men het project wil delen kan men dit project exporteren en krijgt men de LUM van dit project.

Achter elke teststap die zojuist uitgelegd zijn, is er een kleine basis. Dankzij de Swagger van SmartSpoken worden de responsen aangemaakt, enkel wanneer de swagger geïmporteerd is in ReadyAPI, en kan men zo de basis van de test aanmaken. Het effectieve invullen van de apikey, bijhorende content-type en de body moet nog gebeuren door de tester zelf. Na het schrijven en implementeren van de testen leest ReadyAPI de juiste data op het juiste moment uit van de Excel in de juiste teststap. Dit is zodanig geautomatiseerd dat wanneer er nieuwe gegevens ingevuld moeten worden, enkel de Excel sheet aangepast moet worden.

#### 2.2.2.4 Projectfase drie

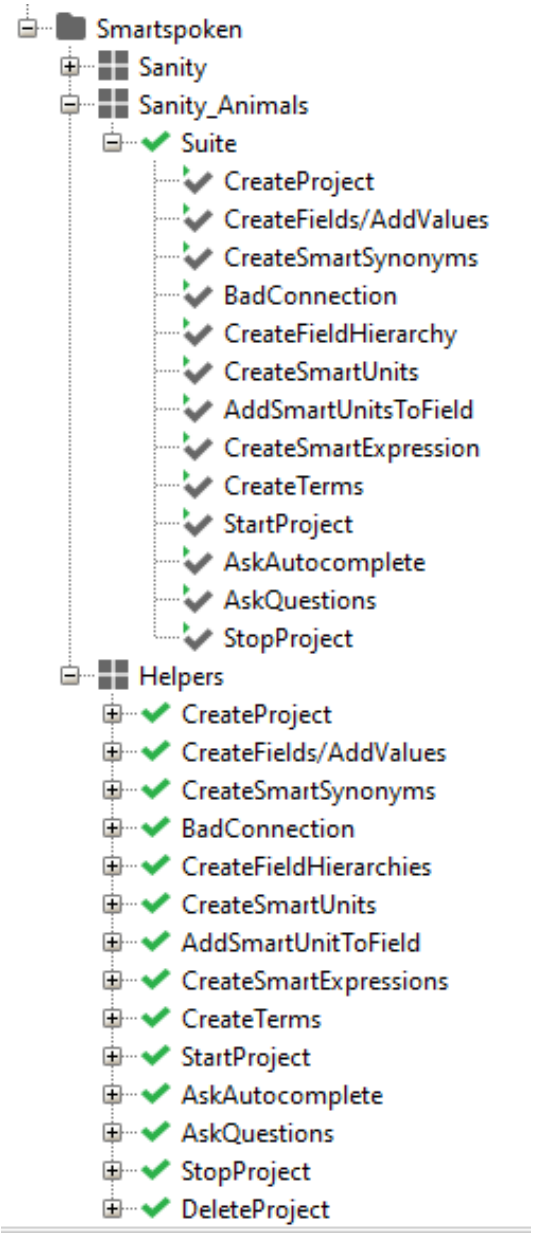
Als derde deeltaak van deze stage moest er een persoonlijk project opgesteld worden. Het ontwikkelen van dit project hoort nog steeds bij projectfase twee. De bedoeling van dit persoonlijk project is dat er door de stagiair wordt gedacht als een klant. Dus met andere woorden, de stagiair, (ikzelf dus) moet een zelf een LUM creëren om zo een project te creëren en specifieke vragen te kunnen stellen. Zo wil SmartSpoken te weten komen welke gedachtegang de klant heeft, als het ook effectief zo gaat zoals zij gedacht hadden en als er geen verwarring komt bij het creëren van een project met veel data. Hierdoor kunnen ze hun code en documentatie nog beter optimaliseren voor het effectief naar de uiteindelijke klant verstuurd wordt. De manier waarop de klant de Excel file zal moeten invullen wordt besproken in het volgende hoofdstuk.

Vooraleer er voor het persoonlijk project in ReadyAPI testen geschreven kunnen worden, moest er eerst een onderwerp bedacht worden en hiervan een LUM gecreëerd worden. Voor het onderwerp van dit persoonlijk project werd er besloten om geen B2B project op te stellen. Na lang nadenken werd er besloten om een project over dieren op te stellen Dit met de gedachtegang er zijn heel veel verschillende soorten dieren dus er kan veel data in het project gestoken worden. Ook kan dit project worden uitgebreid naar een zoo. Op de volgende pagina is er een afbeelding bijgevoegd van de LUM die aangemaakt is voor het persoonlijk project “Dieren”.




Figuur 8 LUM Dieren

Na het aanmaken van de LUM van het project “Dieren” worden de testen opgezet. De testen bestaan uit twee verschillende testsuites, Sanity_Animals en Helpers. In Helpers staan de effectieve testen. Hierin wordt er gewerkt met DataSources, deze gaan gegevens uit de Excel sheet halen. Ook wordt er gescript met Groovy. Deze testcases zijn de effectieve testen die uitgevoerd moeten worden. In Sanity_Animals roepen we deze testcases op. Dit vergemakkelijkt het proces wanneer er gewerkt gaat worden met negatieve testcases. Dan moet men enkel de nodige testcase oproepen in plek van het hele project. Zoals op onderstaande afbeelding te zien is, komen de testen CreateProject, CreateFields, AddValues, StartProject, AskQuestions en StopProject ook hier terug. Dankzij het feit dat de testen automatisch zijn, moet er enkel in de Excel sheet gegevens ingevuld/ aangepast worden. De calls die deze teststappen moeten doen komen uit de Swagger van SmartSpoken. Het effectief genereren van gegevens en testen van deze data is ontwikkeld door middel van scripten in Groovy.




*Figuur 9 ReadyAPI Sanity_Animals*

Daaropvolgend komen de testen terecht op Jenkins. Wanneer er met GIT een push gedaan wordt naar de code repository, zal er op Jenkins automatisch een test job starten die gaat valideren of de software op de DEV en/of TEST omgeving ook werkt Het kan voorkomen dat testen locaal werken maar niet op een server. Dit kan onder andere een simpele instelling zijn, of een bestandsmap die niet dynamisch is ingesteld. Wanneer alles zowel locaal als op de server werkt, kunnen de developers elke keer als ze een push doen de testen runnen. Zo kunnen ze kijken als er een nieuwe bug is bijgekomen of niet. Want zoals hierboven vermeld kan het zijn doordat men een bepaalde bug oplost, er een andere bug ontstaat.

#### 2.2.2.5 Het gebruik van de Excel file

Bij de start van een nieuw project zal de klant een LUM moeten creëren. Aan de hand van deze LUM zal de Excel file ook ingevuld worden. Hier moet er wel strikt aan het voorbeeld gevolgd worden. Indien dit niet zo is zullen er testen zijn die falen. Dit komt doordat de testen de gegevens uit de Excel file halen op een bepaalde manier.

##### 2.2.2.5.1 Aanmaak project

Wanneer een nieuw project start zal er altijd een naam en een taal ingevuld moeten worden. Belangrijk is dat wanneer er een naam ingevuld wordt, er in ReadyAPI automatisch een timestamp wordt achter gezets. Zo heeft men altijd een unieke project naam en is de kans veel kleiner dat er conflicten ontstaan doordat er een project aangemaakt waarvan de naam al bestaat.

| Name | Language |
|---|---|
| SmartSpokenTEST | NL |
| | |

*Figuur 10 project aanmaken*

##### 2.2.2.5.2 Aanmaak en invullen van velden

Wanneer er velden aangemaakt worden moet er altijd een naam en een type aan worden meegegeven. Let op, deze types zullen altijd in het Engels gedefinieerd worden, ook al staat het project dat U aanmaakt in het Nederlands. Bij het invullen van de waardes van de velden (de kolom value) is het ook belangrijk dat wanneer er meerdere waardes worden ingevuld, deze gescheiden worden door een komma. Indien dit door een ander teken of een ander teken plus een spatie gescheiden wordt, zal er in ReadyAPI bij de custom properties van het project SmartSpoken de delimiter veranderd moeten worden. Zo moet er niet in elke code waar deze delimiter gebruikt wordt het teken veranderd worden, maar moet men dit enkel op een plek veranderen.

Verder is het ook belangrijk dat men voor nummer types een minimum en een maximum invult. Wanneer men een van de twee of beide niet invult zal hij deze niet kunnen aanmaken. Bij datumvelden is dit een ander verhaal, indien er hier geen minimum of maximum datum ingevuld wordt, zal hij altijd de correcte datum teruggeven. Als er dus gevraagd wordt voor een dier met geboortedatum 7 maart 1995 gaat de API ons antwoorden met “geboortedatum = 07-05-1995”. Wanneer de minimum en maximum datum wel worden ingevuld zal de API altijd de laatst beschikbare datum nemen. Op de volgende pagina ziet u een voorbeeld van het persoonlijk project “Dieren”.

| fieldType | fieldName | value | min | max | mindat | maxdat |
|---|---|---|---|---|---|---|
| TEXT | Type | koi,karper,guppy,goudvis,snoek,zalm,kabeljauw,goudmakreel,zwaardvis,baars,haai,leguaan,kameleon,g | | | | |
| TEXT | Soort | vissen,reptielen,vogels,zoogdieren,amfibieen | | | | |
| TEXT | Geslacht | mannelijk,vrouwelijk | | | | |
| TEXT | Voeding | vlees,vis,planten,eieren,alles | | | | |
| TEXT | Latijnse benaming | cyprinus carpio carpio,poecilia reticulata,carassius auratus auratus,esox lucius,salmo salar,gadus morhua | | | | |
| TEXT | Voortplanting | eieren,barend | | | | |
| NUMBER | Lengte | | 0,05 | 1500 | | |
| NUMBER | Breedte | | 0,05 | 1500 | | |
| NUMBER | Gewicht | | 0,15 | 1500 | | |
| NUMBER | Leeftijd | | 0 | 1500 | | |
| BOOLEAN | Gevaarlijk | | | | | |
| DATE | Geboortedatum | | | | | |
| DATE | Sterfdatum | | | | | |
| DATE | TestDatum | | | | 2000-01-01 | 2030-12-31 |

Figuur 11 velden aanmaken

#### 2.2.2.5.3 Aanmaak synoniemen

Het kan altijd voorkomen dat er een term gebruikt wordt waarvoor er synoniemen bestaan. Indien dit het geval is en men de synoniemen ook in de vraag wilt gebruiken, kan men synoniemen aanmaken in het Excel bestand. Let wel op, de manier waarop men dit moet doen moet strikt volgen op het voorbeeld. Indien dit niet zo is gaan er testen falen. Dit is hetzelfde verhaal met de delimiter bij de waarden instellen van een veld. Deze delimiter geldt ook voor de synoniemen. Indien men dus een synoniem wil aanmaken moet dit als volgt: synoniem1, synoniem2, synoniemx, veld. Een concreet voorbeeld hiervan kan men zien in volgende afbeelding. De blauw onderstreepte woorden zijn de synoniemen voor de waardes van de velden, de rood onderstreepte woorden. Dus mannelijk is de waarde voor het veld geslacht, maar voor mannelijk maken we de synoniemen mannetje, mannetjes en man aan. Voor men synoniemen aanmaakt, moet er wel nagedacht worden als dit onder synoniemen behoort, of onder smartexpressions. Op dit laatste komen we later nog terug.

| Synonyms |
|---|
| mannetje,mannetjes,man,mannelijk |
| vrouwtje,vrouwtjes,vrouw,vrouwelijk |
| leggen,voortplanting |
| luipaard,panter |
| eten,voeding |
| lang,lengte |
| breed,breedte |
| baart,barend |
| geboren,geboortedatum |
| gestorven,sterfdatum |

*Figuur 12 synoniemen aanmaken*

#### 2.2.2.5.4 Aanmaak bad connections

Wat ook projectafhankelijk is, zijn bad connections. Dit zijn connecties die we aanmaken waar we tegen de API zeggen dat deze link niet gelegd mag worden. Concreet betekent dit aan de hand van een voorbeeld wanneer ik in een vraag het woord pad zeg, wil ik het dier pad hebben, niet wegen. Het dier pad omdat ik in dit project met dieren werk. Als de connectie niet wordt gelegd gaat de API als volgt redeneren: pad -> paden -> wegen -> weg. Daarom maken we een bad connection aan voor het type pad met als token wegen.

| | A | B | C |
|---|---|---|---|
| 1 | fieldName | valueContent | tokenContent |
| 2 | Type | pad | wegen |

*Figuur 13 BadConnections aanmaken*

#### 2.2.2.5.5 Aanmaak fieldhierarchy

Bij fieldhierarchy kunnen er twee tabellen aan elkaar gelinkt worden. Hier moet men eerst de fieldhierarchy een unieke naam geven. Dan moeten de veldnamen ingevuld worden die men aan elkaar willen linken. In onderstaand voorbeeld linken we soort aan type. Dus wanneer er een vraag gesteld wordt zal er sneller een connectie gemaakt worden bij de response tussen deze twee velden dan wanneer men dit niet doet.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Name | firstLevel | firstFieldName | secondLevel | secondFieldName |
| 2 | Diersoorten | 1 | Type | 2 | Soort |

*Figuur 14 aanmaken fieldhierarchy*

#### 2.2.2.5.6 Aanmaak smartunits

SmartUnits worden gebruikt voor het korter schrijven van bepaalde termen in een vraag. Doordat men units aanmaakt voor bepaalde termen kan er tijdens de vragen de verkorte versie voor gebruikt worden zonder dat de API niet weet waarover het gaat. In dit voorbeeld wordt er gewerkt met jaar, kilogram en meter. Wanneer het om een simpele vraag gaat maakt dit niet veel uit, maar wanneer het over een EN/OF vraag gaat kan het al veel schelen in lengte van de zin. Een voorbeeld hiervan is: “een olifant van 3 jaar”/ “olifant 3j of panter 1j”. Een unit heeft ook verschillende conversions. Dit is waaruit de unit bestaat. Hierdoor kan men ook vragen “een olifant van 365d” en hetzelfde resultaat terugkrijgen dan dat men de vraag stelt “een olifant van 1j”.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | naam | symbool | conversionnaam | conversionsymbool | ratio |
| 2 | jaar | j | maanden | mnd | 12 |
| 3 | | | weken | w | 52 |
| 4 | | | dagen | d | 365,25 |
| 5 | kilogram | kg | gram | g | 100 |
| 6 | | | decigram | dg | 10000 |
| 7 | | | centigram | cg | 100000 |
| 8 | meter | m | decimeter | dm | 10 |
| 9 | | | centimerer | cm | 100 |
| 10 | | | millimeter | mm | 1000 |

Figuur 15 aanmaken van units

#### 2.2.2.5.7 Units toevoegen aan het bijhorende veld

Wanneer de units aangemaakt zijn, moeten ze enkel nog gelinkt worden aan het bijhorende veld. In dit project worden er drie units aangemaakt en worden ze gelinkt aan vier velden. De hoofdreden waarom er twee dezelfde units aan een ander veld gelinkt worden in dit project is omdat er zo getest kan worden als de API elke keer hetzelfde antwoord teruggeeft. Dit is belangrijk omdat de API consistent moet zijn in zijn antwoorden.

| | A | B | C |
|---|---|---|---|
| 1 | fieldname | fieldtype | symb |
| 2 | Leeftijd | NUMBER | j |
| 3 | Gewicht | NUMBER | kg |
| 4 | Lengte | NUMBER | m |
| 5 | Breedte | NUMBER | m |

*Figuur 16 unit toevoegen aan veld*

#### 2.2.2.5.8 Aanmaken expressions

Bij synoniemen werd er verweren naar smartexpressions. Dit komt nu aan bod. Bij smartexpressions worden er ingevuld welke expression er wordt aangemaakt. Hier is het belangrijk dat er correct ingevuld wordt voor welk veld en op welke waarde dit geldt. Het verschil met synoniemen is dat er bij synoniemen enkel de waarde waarop dit geld achter de synoniemen komt. Terwijl bij smartexpressions er ook effectief wordt vermeld op welk veld dit specifiek geldt.

| | A | B | C |
|---|---|---|---|
| 1 | SmartExpressions | fieldname | fieldcontent |
| 2 | carnivoor | Voeding | vlees |
| 3 | herbivoor | Voeding | planten |
| 4 | piscivoor | Voeding | vis |
| 5 | ovivoor | Voeding | eieren |
| 6 | omnivoor | Voeding | alles |

*Figuur 17 aanmaken expressions*

#### 2.2.2.5.9 Aanmaken smartterms

Het kan ook voorkomen dat er waarden gebruikt worden die gelijk zijn aan een bepaald teken/term. Hiervoor worden er terms aangemaakt of geüpdatet. Indien er al terms bestaan voor dat bepaald teken, maar deze waarde komt er nog niet in voor moet men de term gaan updaten. Belangrijk hier is dat men bij het updaten van een term, alle voorafgaande keywoorden ook mee overneemt. Wanneer dit niet gedaan wordt gaan deze keywoorden verloren. Wanneer er nog geen terms bestaan voor dat bepaald teken moeten we smartterms gaan aanmaken. Indien dit het geval is maakt het niet uit wat de klant invult als naam, deze wordt namelijk altijd overschreven door de API. De reden waarom elke nieuwe term dezelfde naam krijgt, is zodat SmartSpoken een overzicht heeft van de terms die zij aangemaakt hebben en de terms die de klanten aangemaakt hebben.

| | A | B | C |
|---|---|---|---|
| 1 | name | subname | keywords |
| 2 | lt | filterout | <,kleiner dan,lager dan,minder dan,hoogstens,onder,maximum,max,tot,minder,jonger dan,geleden |
| 3 | gt | filterout | <,groter dan,hoger dan,meer dan,minstens,min,vanaf,meer,ouder dan,boven,binnen |

*Figuur 18 aanmaken/updaten terms*

#### 2.2.2.5.10 Autocomplete

Na het definiëren van de velden moeten de auto complete en vragen nog ingevuld worden. In de auto complete tab wordt er gewerkt met tabellen. De namen van deze tabellen zijn de namen die er in het tabblad fields aan de velden gegeven zijn. Hier moet er worden ingevuld op welke auto complete er getest wil worden, en welk resultaat men hiervoor verwacht. Let wel op, het kan zijn dat niet alle resultaten voorkomen, de reden hiervoor is omdat er maar een x aantal resultaten getoond worden bij auto completes.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **autocomplete** | **Type** | **Soort** | **Geslacht** | **Voeding** |
| 2 | vr | | | vrouwelijk | |
| 3 | ma | marmot,marter | | mannelijk | |
| 4 | ja | jaguar | | | |
| 5 | oli | olifant | | | |
| 6 | vi | | vissen | | vis |
| 7 | vl | vlinder,vleermuis | | | vlees |
| 8 | rep | | reptielen | | |
| 9 | vo | | vogels | | |
| 10 | zoo | | zoogdieren | | |
| 11 | ei | | | | eieren |

*Figuur 19 autocomplete*

#### 2.2.2.5.11 Vragen

Tot slot moet er nagedacht worden welke vragen men wil stellen aan de API. Het is vanzelfsprekend dat dit project gebaseerd moet zijn. Het kan voorkomen dat er vragen gesteld worden waar er twee sets van antwoorden teruggestuurd worden. In dit geval moeten de juiste waarden ook in de juiste set ingevuld worden. Men kan gaan tot vijf sets, maar dit is in het verleden nog niet voorgekomen. Hetgeen dat het meest voorkomt zijn één of twee sets. Ook hier is de Excel file het te verwachte resultaat, en wordt het vergeleken met de JSON die er uit de response opgehaald wordt. Belangrijk hier is ook dat de values met een komma én een spatie van elkaar gescheiden worden. Bijvoorbeeld: Type=panda, Voeding=planten. Indien er bij het definiëren van de velden, de namen een hoofdletter gekregen hebben gaat men dit ook exact zo moeten overnemen in dit tabblad. De API is namelijk hoofdletter gevoelig.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Vraag | Set 1 | Set 2 | Set 3 | Set 4 | Set 5 |
| 2 | reptielen 5 j | Soort=reptielen, Leeftijd=5 | | | | |
| 3 | vogels die eieren leggen | Soort=vogels, Voortplanting=eieren | | | | |
| 4 | jaguar | Type=jaguar | | | | |
| 5 | kat | Type=kat | | | | |
| 6 | schaap die 500g weegt | Type=schaap, Gewicht=5 | | | | |
| 7 | cyprinus carpio carpio 2j en 25 dm | Latijnse benaming=cyprinus carpio carpio, Leeftijd=2, Breedte=2.5 | | | | |
| 8 | vrouwelijke boa 8500g weegt en 320 weken is | Type=boa, Geslacht=vrouwelijk, Leeftijd=6.153846153846154, Gewicht=85 | | | | |
| 9 | chamaeleonidae 1500 mm, piscivoor | Voeding=vis, Latijnse benaming=chamaeleonidae, Breedte=1.5 | | | | |
| 10 | guppy of goudvis, 5cm lang | Type=guppy, Lengte=0.05 | Type=goudvis,Lengte=0.05 | | | |
| 11 | zalm 2kg 1m lang 20cm breed | Type=zalm, Gewicht=2, Lengte=1, Breedte=0.2 | | | | |
| 12 | koi mannelijk 6 maanden | Type=koi, Geslacht=mannelijk, Leeftijd=0.5 | | | | |
| 13 | gorilla die eieren eet en 8kg weegt | Latijnse benaming=gorilla gorilla, Voeding=eieren, Gewicht=8 | | | | |
| 14 | mannelijke hond 18 maanden | Type=hond, Geslacht=mannelijk, Leeftijd=1.5 | | | | |
| 15 | dieren die eieren leggen | Voortplanting=eieren | | | | |
| 16 | dieren die alles eten en barend | Voeding=alles, Voortplanting=barend | | | | |
| 17 | catopuma temminckii 58w 650g lengte 105cm | Latijnse benaming=catopuma temminckii, Leeftijd=1.1153846153846154, Gewicht=6.5, Lengte=1.05 | | | | |
| 18 | krokodillen van 25 jaar | Type=krokodil, Leeftijd=25 | | | | |
| 19 | ursus maritimus | Latijnse benaming=ursus maritimus | | | | |
| 20 | panda 25 maanden, eet planten | Type=panda, Leeftijd=2.0833333333333335, Voeding=planten | | | | |
| 21 | carnivoor schreckensteiniidae 26 maanden vrouwelijk | Leeftijd=2.1666666666666665, Latijnse benaming=schreckensteiniidae, Geslacht=vrouwelijk, Voeding=vlees | | | | |

*Figuur 20 voorbeeldvragen*

#### 2.2.2.6 ReadyAPI implementeren in Jenkins

Tijdens het implementeren van ReadyAPI in Jenkins zijn er problemen opgedoken. Deze waren dat er geen connectie gemaakt kon worden op de server met Jenkins. Na enig opzoek werk merkten we dat we niet de juiste license key gebruikten voor ReadyAPI. We configureerden Jenkins om gebruik te maken van een fixed license key, terwijl in onze geval we een floating license key hadden. Op de meeste websites werd er ook gesproken over een fixed key, terwijl wij naar een oplossing met een floating key zochten. Het tweede probleem dat opdook was dat er niet naar de juiste locatie gelinkt was. Het is wel belangrijk dat zowel de server als de client waarop de ReadyAPI testen geschreven worden beide een licentie hebben.

Beide problemen zijn opgelost en voor het key probleem is er documentatie geschreven op Jira. Op deze manier kunnen collega's die hetzelfde probleem tegen komen, deze documentatie gebruiken en minder tijd verliezen dan dat er op dit project verloren is. De documentatie hierrond vindt u in de bijlage terug.

#### 2.2.2.7 Projectfase 3

De vierde deeltaak (fase 3) valt voornamelijk onder documentatie. Tijdens de derde deeltaak wordt er meer focus gelegd op het verbeteren van de documentatie in verband met de stage zelf. Hiermee wordt er bedoelt dat er tijdens deze laatste week, nog feedback verwerkt kan worden op de stageplaats zelf. Zo kan er bij twijfel nog hulp gevraagd worden. Ook wordt er nog meer feedback gegeven. Het geven en oefenen van de eindpresentatie en demo valt hier ook onder. Tot slot wordt er in deze fase het project nog een maal doorgelopen en bekeken op nieuwebugs. De manier waarop dit gedaan wordt is door de testen die geschreven worden in ReadyAPI aan de hand van Swagger opnieuw te runnen. Doordat de code opgeslagen is in Jenkins kan dit via de Jenkins zelf uitgevoerd worden. Wanneer de testen rood of oranje zien, betekend dat er iets fout gegaan is en dus kan het zijn dat er een nieuwe bug ontwikkeld is.

## 2.3 Reflectie

### 2.3.1 Persoonlijke reflectie Stageopdracht

Bij de start van deze stage was er een hoge leercurve. Omdat het project getest moest worden in ReadyAPI, werd er van mij verwacht dat ik zelf onderzocht hoe ReadyAPI werkt. Dit vond ik geen enkel probleem, enerzijds omdat ik graag met nieuwe tools leer werken en anderzijds omdat ik geïnteresseerd ben in het schrijven van testen. Ook werd er gewerkt met Confluence, Git en Jira. Deze tools had ik al eens gebruikt maar nog niet dagelijks. De snelheid waarbij je bijleert tijdens een stage lijkt ook exponentieel te verlopen. Je kan maar tot een bepaald niveau iets bijleren op school, pas wanneer je leerstof moet toepassen merk je wanneer je de kennis goed beheert of niet.

Tijdens de stage heb ik in het algemeen een beter beeld van mezelf gekregen op professioneel vlak. Naar het einde van mijn project toe voelde ik me er veel comfortabeler bij om naar developers toe te stappen, te vragen hoever ze met mijn bug zaten of om hulp vragen wanneer ik vast zat. Wel zorgde ik ervoor dat wanneer ik naar de developers moest gaan, ik er zeker van was dat de bug bij hun lag en niet bij mij. Hiervoor maakte ik gebruik van advanced rest client.

Dankzij deze stage heb ik ook veel leerstof, die ze mij op school geleerd hebben, kunnen toepassen in de praktijk.

### 2.3.2 Persoonlijke reflectie eigen functioneren

Bij het begin van de stage had ik geen ervaring met ReadyAPI. Hierdoor verliepen de eerste twee weken moeizaam. Dit gepaard met het werken met Git, Jira en Confluence, waar ik zelden gebruik van gemaakt had. Zij hielpen me steeds als ze het antwoord op de vraag wisten. Het testen zelf vond ik redelijk goed gaan, enkel in het begin verliep dit moeizamer. Maar er is geen enkel probleem geweest die ik niet met mijn hogeschoolpromotor, bedrijfspromotor of mijn collega's kon oplossen.

Verder kwam het door ReadyAPI en Groovy dat ik op sommige testcases vast zat. Hier weer met de reden dat ik nog nooit gescript had in Groovy of testen geschreven had in ReadyAPI.

Mijn Java en Agile skills zijn ook toegenomen. Doordat ik in Java en Groovy heb moeten scripten, ben ik hier nu veel vlotter in geworden. Het Agile gedeelte is vooral toegepast geweest bij de sprint planningen, sprint retrospectives en de daily stand ups. Hier moest ik zelf ook een bijdragen leveren als stagiair. Wel vond ik het positief dat wanneer ik vragen stelde aan iemand, ze nooit meteen het antwoord prijsgaven. Ze gaven me echter tips om me in de juiste richting te sturen.

Dankzij mijn lessen van softwaremanagement kon ik ook zeer goed volgen tijdens de sprint planning, sprint retrospectives en de scrum meetings. Binnen de opleiding hebben we gezien welke soorten testen er zijn en hoe je agile moet werken, maar we hebben dit amper toegepast in de praktijk. Verder heb ik mijn lessen Java wel kunnen toepassen, Groovy is namelijk een scripting language gebaseerd op Java. Tot slot heb ik mijn lessen van Communcation Skills 1 en 2 ook kunnen toepassen namelijk bij het schrijven van e-mails, het schrijven van documentatie en bij het netwerken op de werkvloer.

Het volgen van de deadlines bracht veel stress met zich mee, dit kwam ook gepaard met het feit dat ik mijn lat altijd te hoog legde. Ik probeerde me in die zin altijd te pushen om beter te presteren, maar er kwam een punt tijdens mijn stage dat ik voelde dat ik het rustiger aan moest doen. Dit gebeurde in de tweede week van de kerstvakantie, omdat ik even het bos door de bomen nier meer zag, ben ik dan ook samen gaan zitten met Bram en Wim.

Ik heb zeer veel bijgeleerd door mijn stage. Niet alleen over testing, maar ook over de tools die ik gebruikt heb en teamwork. Het is zeer belangrijk om goede testen te schrijven, maar het aller belangrijkste is een goed team. Hiermee bedoel ik niet enkel het testing team, maar het projectteam in het algemeen. Als je in een team terecht komt dat je met open armen ontvangt en helpt ga je sowieso een beter eindresultaat kunnen afleveren dan wanneer dit niet het geval is. Ik heb gelukkig een team en een omgeving gehad dat me met open armen ontving.

### 2.3.3 Eindbesluit

Doordat ik met een nieuwe tool heb leren werken verliepen sommige aspecten van mijn stage minder vlot dan dat ik gehoopt had. Doorheen het verloop van de stage heb ik mijn planning moeten aanpassen en zijn er noodgedwongen taken geschrapt moeten worden. Hierdoor is het project in een andere richting gestuurd dan dat ik zelf gehoopt had. Toch heb ik hier het beste van proberen maken. Maar door mijn passie voor testen heb ik mijn project volledig af kunnen werken met al een voorbereiding op het schrijven van toekomstige negatieve testcases. Al bij al vind ik het uiteindelijke resultaat toch noemenswaardig.

Als ik deze stage opnieuw zou kunnen doen dan zou ik mijn planning anders ingedeeld hebben. Ik zou sommige taken meer prioriteit geven hebben en andere minder. Ook zou ik mijn dagen beter hebben ingedeeld. Op dit moment was de planning die ik in week één heb opgesteld geen probleem, maar deze heb ik ook een aantal keren ingeschroefd wanneer ik merkte dat de deadline niet haalbaar was. Het zou alleen gemakkelijker geweest zijn als ik op voorhand wist hoe ReadyAPI en Groovy werkte.

Het feit dat ik mij hierdoor op nogal onbekend terrein begaf, zorgde ervoor dat ik mijn plan leerde te trekken in het leren van een nieuwe tool. Hierdoor heb ik niet enkel bijgeleerd hoe met ReadyAPI te werken, maar ook hoe ik moet scripten in Groovy en hoe ik goed moet googelen. Uiteindelijk ben ik zeer tevreden over het eindresultaat dat ik afgeleverd heb tijdens mijn stage. Het gaf mij de kans om fulltime aan een project te werken waar ik gepassioneerd in ben.

# II. Onderzoekstopic

# 1 Onderzoeksrapport

## 1.1 Probleemstelling

Om het testproces van Refleqt te kunnen verbeteren gaat er onderzoek gedaan worden naar de impact van model based testing op test automatisatie. Wat zijn de voor en nadelen hiervan en wat is er allemaal realiseerbaar door model based testing toe te passen.

## 1.2 Link met stagebedrijf

Sinds dit onderzoek geen directe link heeft met mijn stageopdracht is dit onderzoek voor mijn stagebedrijf. Het resultaat van dit onderzoeksrapport zal een positief effect hebben op de marktligging van het bedrijf.

## 1.3 Onderzoeksmethode

In dit onderzoek ga ik bekijken hoe er getest kan worden met model based testing. Om dit op een correcte manier te behandelen zal er een studie gedaan worden gevolgd door een vergelijkingsmatrix. De studie zal gaan over wat model based testing is, welke de voor en nadelen hiervan zijn en hoe dit toepasbaar is op test automatisatie. De vergelijkingsmatrix daarentegen wordt een matrix waarin er tools vergeleken gaan worden die Refleqt in de toekomst zou gaan kunnen gebruiken. In de tool die het beste uit deze vergelijking komt zal er een kleine proof of concept in uitgevoerd worden.

Het doel van mijn onderzoek is om een conclusie te verkrijgen die een duidelijk beeld schept over de impact van model based testing op test automatisatie.

# 2 Onderzoeksrapport uitwerken

## 2.1 Uitwerking literatuurstudie

Vooraleer we het testproces kunnen verbeteren met behulp van model based testing dienen we eerst twee belangrijke dingen te weten namelijk: “Wat is testen” en ”Wat is MBT”.

### 2.1.1 Testen [23][24]

Testen is een kwaliteitscontrole die niet enkel dient om fouten te vinden, maar ook om te evalueren of de gebouwde functionaliteiten voldoen aan de verwachtingen van de klanten. Het belang van testen kan gemakkelijk aangetoond worden met enkele voorbeelden. Indien functionaliteit niet getest is en bij klanten geïnstalleerd wordt, en er blijken defecten aanwezig te zijn, loopt men als leverancier de kans om zijn geloofwaardigheid als partner te kunnen verliezen. Erger, de klant zou kunnen beslissen om over te stappen naar een andere IT-partner. Om dit tegen te gaan en de klant van hoge kwaliteit te kunnen verzekeren, is het belangrijk om op een juiste manier te testen.

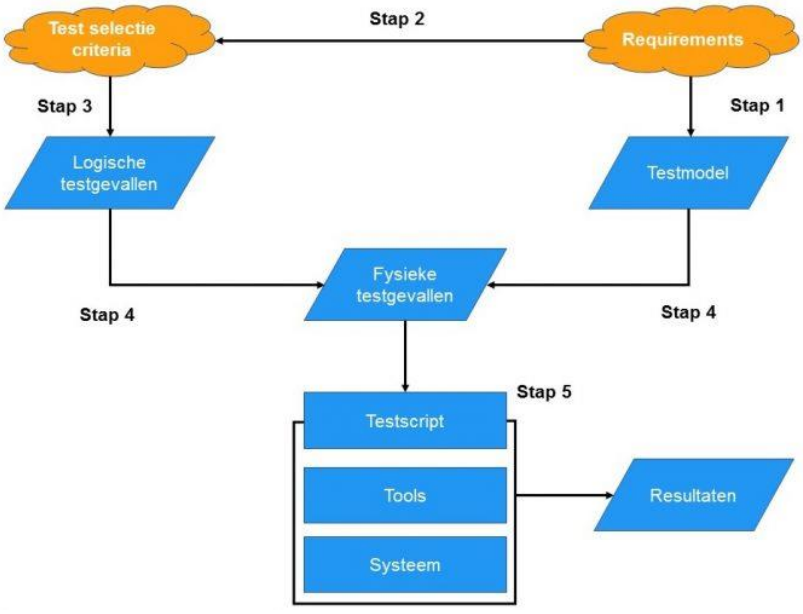
Testen is vaak een cruciaal onderdeel van een project. Hiervoor moeten er tijd en middelen vrij gemaakt worden. De meest optimale vorm van testing is wanneer men bij het begin van het project van start kan gaan met testen. De grootste reden waarom men bij het begin van een project ook al moet testen is omdat er zo sneller bugs gevonden en opgelost kunnen worden. Hierdoor groeit de kwaliteit van het eindproduct. Vandaag de dag zijn er veel te weinig middelen om alle applicaties te kunnen testen die er op de markt zijn. Dit is daarom ook de hoofdreden waarom het een tijd duurt voor goede testen te schrijven.

### 2.1.2 Model based testing [22][23][24]

Voor we ons kunnen verdiepen in MBT, gaan we eerst in op wat een model is. Een model beschrijft het gedrag van een systeem. Het gedrag kan worden beschreven in termen van invoer sequenties, acties, voorwaarden, uitvoer en de dataflow van invoer tot uitvoer. Er zijn talrijke modellen beschikbaar en deze beschrijven uiteenlopende aspecten van het systeemgedrag. Voorbeelden zijn

- DataFlow
- ProcesFlow
- Graaf
- State transition diagrammen
- UML

Nu we weten wat een model is, kunnen we ons verdiepen op MBT. MBT is het automatisch genereren van testgevallen, met behulp van abstracte modellen gebaseerd op de requirements en het gedrag. Waarna deze testgevallen handmatig of automatisch kunnen worden uitgevoerd. De mogelijke stappen binnen MBT-proces kan men zien in onderstaande figuur. Afhankelijk van de tool die men als tester gebruikt, kunnen sommige stappen in dit MBT-proces in meer of mindere mate aanwezig zijn.




*Figuur 21 Stappenplan Model Based Testing*

Als eerste begint men met het testmodel op te stellen. Dit wordt gedaan aan de hand van de requirements die door de klant opgesteld zijn. Het testmodel dat men dan verkrijgt is meestal een samenvatting van de functionaliteiten van het te testen systeem met de focus op de test doelstellingen. Belangrijk is ook het valideren van het testmodel. Dit zorgt ervoor dat de requirements op consistentie en compleetheid worden gecontroleerd.

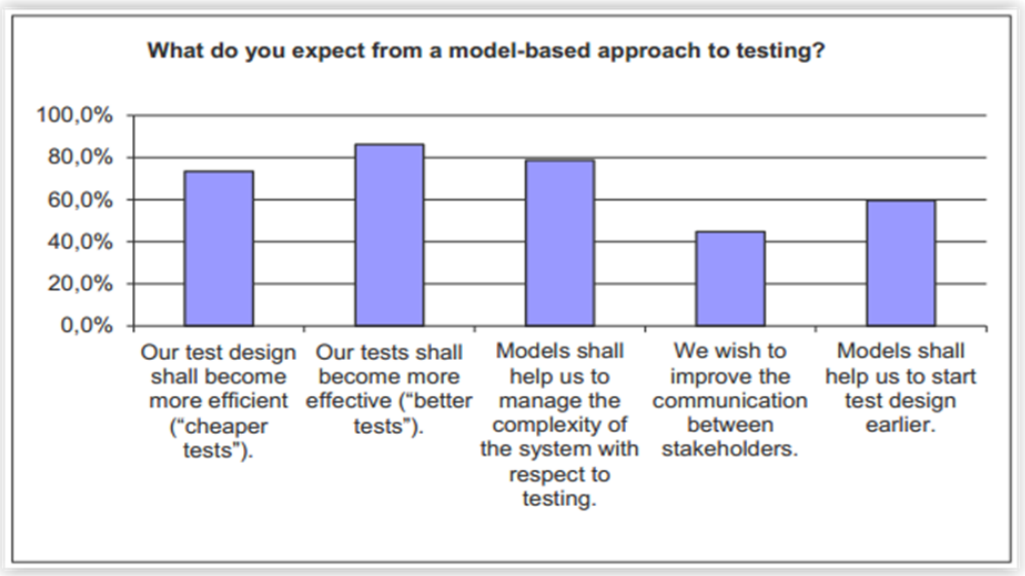
Ten tweede zullen de selectiecriteria gekozen worden aan de hand van de requirements en testdoelen. Die criteria zullen worden gebruikt bij het automatisch genereren van de logische testgevallen.

Dan worden deze logische testgevallen opgesteld of automatisch gegenereerd. Weer op basis van deze selectiecriteria die gekozen worden in de tweede stap.

Wanneer de logische testgevallen aangemaakt zijn, worden ze herwerkt naar fysieke testgevallen (uitvoerbare testgevallen). Het verschil tussen logische en fysieke testgevallen zijn de volgende. Bij logische testgevallen baseert men zich op de “wat vraag”. Concreet betekend dit “Wat zou er getest moeten worden”? Dit wordt ook beschreven op een algemeen niveau, dus men weet nog niet specifiek wat men moet testen. Fysieke testgevallen daarentegen baseren zich op de “hoe vraag”. Dit betekend concreet “Hoe wordt er getest?” Hier wordt wel specifiek aangegeven welke zaken getest moeten worden dus kunnen we hierdoor de testen ook uitvoeren.

Het uitvoeren van de testen kan op twee manieren gebeuren, online of offline. Bij het online uitvoeren van testen worden de testgevallen gegenereerd en worden deze ook onmiddellijk uitgevoerd. Bij het offline testen worden de testgevallen vooraf gedefinieerd en pas op een later tijdstip uitgevoerd. Tijdens het genereren van de testgevallen kunnen er sommige applicaties controleren op consistentie. Wanneer de consistentie van een test case niet bereikt is, zal er een error optreden. Doordat MBT test cases genereerd op basis van een model van de vereisten, is het gemakkelijker om dit model aan te passen wanneer de vereisten veranderen.

Bijkomend is er nog een onderzoek gedaan over MBT zelf. Hieruit blijkt dat de meerderheid van de MBT-gebruikers verwacht dat hun testen meer efficiënt en effectief worden. Dit kan worden afgeleid uit onderstaande afbeelding.




*Figuur 22 onderzoek MBT resultaat*

### 2.1.3 Voor- en nadelen van model based testing

De voordelen van MBT zijn:

- MBT kan altijd modellen toepassen, ongeacht hoe de system under test (SUT) hiervoor opgesteld is.
- MBT bied een mogelijkheid aan tot het automatiseren van de test cases.
- MBT creëert versnelling in de specificaties van test scripts
- MBT maakt de tijd en middelen consumerende taken van testspecificatie minder afhankelijk van het aantal en expertise van de testers.
- Met behulp van modellen zijn nieuwe features vertaald in nieuwe test scripts in enkele seconden.
- MBT laat oprichting van standaard testgevallen toe. Dit kan een meerwaarde zijn voor de mogelijkheid tot het uitbesteden van de uitvoering van een test.
- Doordat MBT test cases genereert aan de hand van een model van de vereisten, is het gemakkelijker om dit model aan te passen wanneer de vereisten veranderen.
- MBT genereert een minimale test waarbij alle toestanden en alle transities ten minste één keer worden geraakt. De dekking is dus maximaal bij een minimale testtijd. Wel is de variatie beperkt tussen de verschillende testruns.
- Modellering bevordert de communicatie tussen de betrokken partijen.
- Grafische modellen zorgen ervoor dat stakeholders makkelijker betrokken kunnen worden.
- Het opstellen van modellen zorgt voor een continue verbetering van de competentie van testers in een bepaald domein.

Het grootste nadeel dat MBT met zich meebrengt is dat MBT niet kan gebruikt worden om een volledig test object te testen. Het heeft alleen betrekking op de functionele attributen. Andere kwaliteit attributen zullen dus getest moeten worden op een andere manier. MBT kijkt enkel als de functionele vereisten gedekt zijn. Wanneer er ook niet functionele vereisten gedekt moeten worden, zullen er andere testmethoden gebruikt moeten worden.

De grootste uitdaging bij het gebruik van MBT is hoe moet men een goed model aanmaken. Een goed model hangt af van de persoonlijke eindcriteria van het te testen project. Want op basis van dit model worden de testcases gegenereerd. Dit is voor vele testers nog een zeer onbekend vlak. Doordat dit vaak voorkomt vind ik dat er een aparte specificatie moet zijn bij testers, namelijk het maken van goede models. Pas wanneer het lukt om een goed model aan te maken wordt het automatiseren van testcases pas een realistische mogelijkheid.

### 2.1.4 Vergelijking met klassieke testing methodologieën

#### 2.1.4.1 Wat zijn test methodologieën

Test methodologieën zijn strategieën en benaderingen die gebruikt zijn voor het testen van een bepaald product zodat het geschikt is voor gebruik. Het bevat meestal het zodanig testen dat het product werkt met al zijn functionaliteiten en specificaties. Ook moeten er geen ongewenste bijwerkingen meer optreden.

Software test methodologieën zijn verschillende benaderingen en manieren om zeker te zijn dat de softwareapplicatie met name volledig getest is. Software test methodologieën omvatten alles van unit testen van individuele modules, integratie testen van het volledig systeem met gespecialiseerde soorten testen zoals security en performance testen.

#### 2.1.4.2 Voorbeelden van test methodologieën

Test methodologieën kan men onderverdelen in twee grote categorieën, namelijk functioneel testen en non-functioneel testing. Beide bevatten op zich een onderverdeling van test manieren.

| Functioneel testing | Non-functioneel testing |
|---|---|
| Unit testing | Performance testing |
| Integration testing | Security testing |
| System testing | Usability testing |
| Acceptance testing | Compatibility testing |

*Tabel 2 Functionele en non functionele methodologieën*

Unit testen (unit testing) is een methode om softwaremodulen of stukjes code afzonderlijk te testen. Bij unittesten zal er voor iedere unit een of meerdere tests ontwikkeld worden. Hierbij worden dan verschillende testcases doorlopen. In het ideale geval zijn alle testcases onafhankelijk van andere tests.

Integratie testen (integration testing) zijn softwaretesten waarbij individuele softwaremodules verbonden worden en als een geheel getest worden. Deze fase komt na de unit testen en voor de systeem testen. Voor de integratie testen te kunnen uitvoeren, hebben we modules nodig die door de unit tests gekomen zijn. Deze goedgekeurde modules worden aan elkaar gekoppeld en hier worden tests op losgelaten die in het integratie-testplan beschreven zijn.

Een systeem test (system testing) is een test van het volledige systeem, waarbij gekeken wordt of het voldoet aan de requirements. Een systeemtest is een zogenaamde blackboxtest, er is dus geen kennis vereist van de interne structuur van het te testen systeem. De input van een systeem test zijn alle componenten die door de integratie testen gekomen zijn. Het doel van een systeem test is om de onderdelen gezamenlijk te testen die daarvoor al in een unittest getest zijn.

Acceptatie tests (acceptance testing) zijn testen om iets wel of niet te accepteren. Hier wordt nagegaan of er naast het feit dat de testen nu al voldoen aan de requirements van voorgaande testen, problemen zijn te verwachten in het gebruik die eerder nog niet gevonden.

Performance testing geeft een overzicht over het gedrag van de systemen wanneer deze gebruikt worden door een gespecificeerde groep gebruikers. De nadruk van de performance testen ligt op het verkrijgen van inzicht in de reactietijden die de gebruiker ervaart en de mogelijke belastbaarheid van het systeem. Om deze reactietijden vast te stellen kunnen diverse performance testsoorten ingezet worden.

Usability testing is het beoordelen van websites en applicaties door een testpanel een aantal opdrachten uit te laten voeren en aan de hand hiervan knelpunten en andere problemen op deze websites en applicaties te achterhalen. Aan de hand van de resultaten kunnen voorstellen gemaakt worden om de gebruikerservaring van de geteste websites en applicaties te verbeteren.

Compatibiliteitstesten zijn testen om te kijken indien de software/applicatie compatibel is met verschillende browsers en verschillende besturingssystemen. Zeker in de e-commerce sector en mobiele applicaties is dit een heel belangrijk onderdeel

Een security test is een test waarin de analyse en beoordeling van beveiligingsrisico's voor uw organisatie wordt gedaan. Het resultaat is een risicoanalyse.

#### 2.1.4.3 Vergelijk test methodologieën met model-based testing

Het hangt van tool tot tool af welke functionele test methodologen een MBT-tool implementeert. Wel is het belangrijk om te weten dat MBT enkel kan gebruikt worden bij de functionele testen. Dus het grote verschil tussen MBT en de test methodologieën die er al bestaan is dus dat er ook niet functionele test methodologieën bestaan. Deze kunnen niet geïmplementeerd worden bij eender welke MBT-tool, terwijl deze testen wel gebruikt worden. Doordat MBT de test cases automatisch genereert gebaseerd op het model dat doorgegeven is, kan er wel meer tijd gespendeerd worden aan de eventuele non functionele testen, indien dit nodig is voor dit bepaald project.

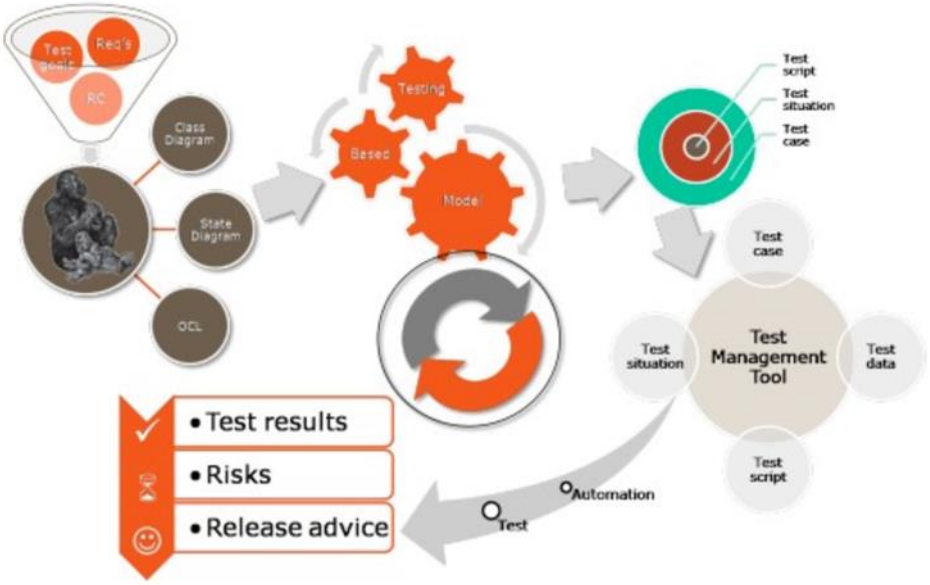
### 2.1.5 Model based testing binnen testautomatisatie

MBT is een toepassing op model based design en bied een zeer hoge graad van testautomatisatie. Hier worden modellen gebruikt om het gedrag van systeem onder test (SUT) te representeren of om testing strategieën en testomgevingen te representeren. Een SUT-model is meestal een abstract model, dit is omdat het model niet altijd een representatie is van de echte wereld. Het gebruik van modellen wordt model-driven-development (MDD) genoemd. Door MDD te gebruiken in de development lifestyle moet er geen extra model aangemaakt worden voor testing.

Het kan wel gebeuren dat er gebruik gemaakt wordt van een geïsoleerd model. Wanneer dit het geval is bevat het testmodel andere onderwerpen dan het development model. Bijvoorbeeld het gebruik van testdata en het gebruik van booleaanse. Ook heeft men door een geïsoleerd model het voordeel dat de errors die in het development model niet opgenomen worden in het test model. Hierdoor vinden testers nog sneller bugs tijdens het maken van het design.

Wanneer men het test model maakt, komen er een aantal vragen naar boven over de kwaliteit van de functionele vereisten. Deze vragen zijn eenvoudige ontdekkingen van bugs. Om deze te kunnen oplossen moeten ze voorgelegd worden bij de designer of analist die het model heeft opgesteld. In een normale omstandigheid, zonder MBT, komen deze bugs enkel naar boven tijdens het manuele specificatie van de test case. Zeer laat in het project dus. Hierdoor kan het zijn dat deze bugs al toegepast zijn op het prototype. In een situatie met MBT gebeuren deze ontdekkingen veel eerder in het proces. Hierdoor is de voorbereidingsfase van het ontwikkelingsproces nog belangrijker en heeft deze een hogere prioriteit.

Bij het gebruik van model-based testing is het duidelijk te zien dat automatisch gegenereerde testgevallen heel anders door de applicatie heen lopen dan een tester ooit kan bedenken en uitvoeren (handmatig dan wel geautomatiseerd). Hierdoor worden de testgevallen langer en zijn ze interessant. Hierdoor ontstaan vaak nieuwe inzichten doordat dezelfde functionaliteit vaker wordt geraakt.




*Figuur 23 MBT diagram*

## 2.2 Uitwerking testing tools en testautomatisatiemogelijkheden

Bij het begin van de stage werd er een onderzoeksopdracht opgesteld, deze ging over de impact van model-based testing op testautomatisatie. Doordat MBT uitgevoerd wordt aan de hand van tools is er gekozen om opzoekwerk te verrichten naar welke tools voor Refleqt toepasbaar zijn. Na enige opzoekwerk zijn er een tiental tools naar boven gekomen die vergeleken werden met mekaar.

- GraphWalker
- SpecExplorer
- Mista
- TestOptimal
- TorXakis
- NModel
- 4Test
- TEMA
- JUMBL
- Cow_Suite
- NModel

Door meer onderzoek naar deze tools te doen en ze met mekaar te vergelijken zijn er vijf tools uitgesprongen. De reden waarom er voor deze vier tools gekozen is, kan men uit onderstaande opsomming afleiden:

- Deze vier tools kwamen het meest terug op de verschillende websites.
- Deze vier tools ondersteunen verschillende soorten testen.
- Deze vier tools kunnen op verschillende soorten test frameworks gebruikt worden.
- Ze ondersteunen Java, dit is de codeertaal van Refleqt.
- Deze vier tools maken allen gebruik van MBT.

De tools die niet gekozen werden, waren niet interessant. Deze waren oude tools, werden niet meer ondersteund, werden niet meer geüpdatet of ondersteunde geen Java. De tools waar men wel voor gekozen heeft, werden verder onderzocht en hier werd uiteindelijk een vergelijkingsmatrix voor opgesteld. Hierdoor kan er een gedetailleerde keuze gemaakt worden voor welke tool het best past voor de toekomst voor Refleqt.

- SpecExplorer
- GraphWalker
- MISTA
- TorXakis

### 2.2.1 Wat is SpecExplorer

Spec Explorer is een Model-based testing hulpmiddel dat zich uitstrekt van Visual Studio. Hiermee kan men het model software-gedrag analyseren door grafische visualisatie, het model controleren en standalone test code genereren van de modellen. Het gedrag hiervan is gemodelleerd op twee verschillende manieren: door het schrijven van regels in c# en door het definiëren van Cord script configuraties, parameterwaarden en scenario's als actie patronen in een reguliere-expressie stijl.

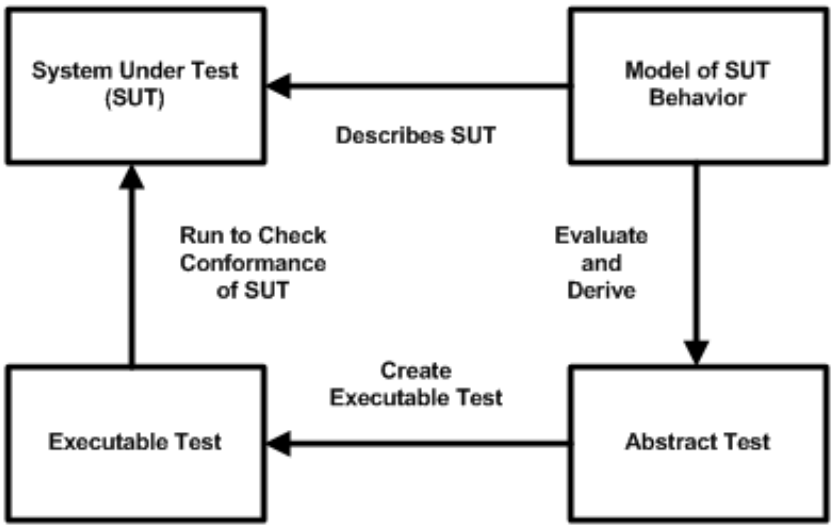
De kenmerken van Spec Explorer zijn:

- Ondersteunt behaviour modellen met C# en Cord en meerdere modellering stijlen
- Visualiseert behaviour gedrag als een exploratie-grafiek
- Genereert automatisch zelfstandige test code
- Modelleert synchrone, asynchrone en niet deterministische systemen
- De SUT wordt automatisch vergeleken met het model. Als je test faalt, dan is er een verschil tussen de SUT en het model. Als je test slaagt, dan zijn deze consistent met elkaar.

De functionaliteiten van Spec Explorer kunnen worden bediend via een van de volgende handelingen.

- Een gebruikersinterface die gelijk is aan de gebruikersinterface van Visual Studio.
- De command-line gebruiken voor het inroepen van een zelfstandig uitvoerband bestand namelijk, Spec Explorer.exe

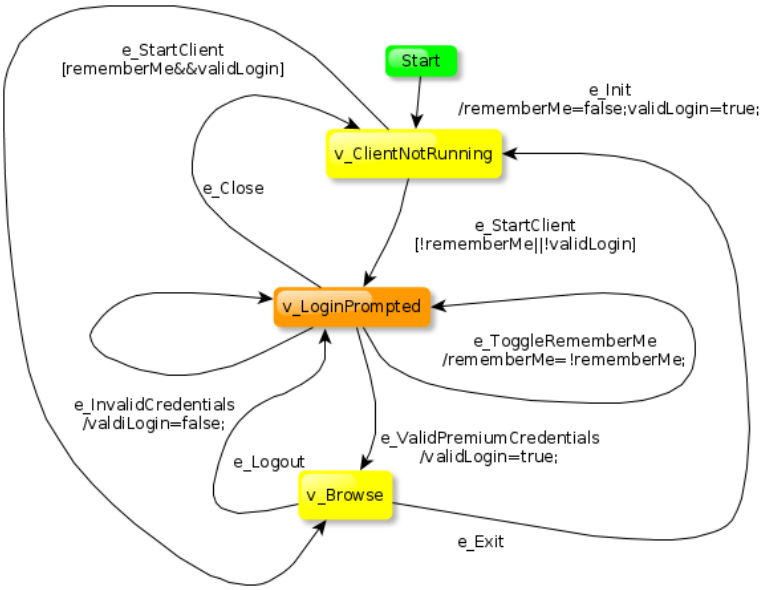
De volgende afbeelding geeft een overzicht van de belangrijkste onderdelen en activiteiten van Spec Explorer van het MBT-proces. Het veronderstelt een bestaande SUT en kan dienen als basis voor het vergelijken van de twee belangrijkste wijzen van testen in Spec Explorer.




Figuur 24 Spec Explorer MBT-proces

### 2.2.2 Wat is GraphWalker

GraphWalker is een open source Model-based testing hulpmiddel voor test automatisatie. Het is ontworpen voor het makkelijk ontwerpen van je tests met behulp van grafieken. Het model is een verzameling van pijlen en knooppunten die samen een grafiek maken. Een pijl betekend een actie en een knooppunt vertegenwoordigt een verificatie. Onderstaand ziet u een voorbeeld hiervan.




*Figuur 25 GraphWalker*

Er zijn twee kenmerken die GraphWalker niet doet, namelijk:

- GraphWalker kan geen testen uitvoeren. GraphWalker maakt geen interactie met je SUT. Indien je wilt dat er interactie met je SUT gebeurt, moet je nog een ander hulpmiddel gebruiken. Als je bijvoorbeeld een webapplicatie wil testen, kan je gebruik maken van bijvoorbeeld Selenium. Als je een mobiele app wilt maken, zou je je eerder richten op Apium.
- In GraphWalker kan men geen grafieken bewerken. Tot op vandaag wordt er gebruik gemaakt van een externe tool genaamd yEd van yWorks. Men verwacht wel dat dit zal worden opgenomen in de 4.0 release van GraphWalker.

De kenmerken die GraphWalker wel heeft zijn:

- Uit een bepaalde set van grafieken, generatoren en stop voorwaarden zal GraphWalker een pad door de grafieken genereren. Het pad vertegenwoordigt je test of je testcase en het pad is een reeks van elementen.
- Offline-mode. Het genereren van het pad wordt een keer gedaan. Het is niet rechtstreeks verbonden op elke test automatiseringscode. Het pad moet worden opgeslagen in een bepaald formaat. Meestal wordt het pad gegenereerd vanaf de command line en wordt de output opgeslagen in het bestand. De inhoud van het bestand wordt vervolgens gebruikt om door uw automatische testen getest te worden.
- Online-mode. Het genereren van het pad wordt gedaan tijdens de uitvoering van de testen (tijdens run-time). Dit betekend dat GraphWalker moet worden geïmplementeerd in je test automatisatie code. Dit vergt wat complexiteit, maar de voordelen hiervan zijn:
- Er is geen behoefte om de tussenliggende paden in bestanden op te slaan.
- Je automatische testen hebben direct toegang tot de uitvoeringscontext van de grafieken.

### 2.2.3 Wat is MISTA

MISTA is een tool voor het genereren van uitvoerbare test code van een Model-Implementation-Description (MID), die uit een testmodel, model-implementation mapping (MIM) en helper code bestaat. De kenmerken van MISTA zijn:

- Het gebruikt visuele notaties voor het bouwen van test modellen zoals de functie nette en eindige statuscomputers. Functie netten die lichtgewicht op hoog niveau Petri nets[1] zijn, kunnen beide controle-georiënteerde en gegevens-georiënteerde test modellen specificeren.
- Het biedt test generatoren aan voor uitgebreide dekking van test modellen, inbegrepen onderzoek dekking, diepte dekking, doel dekking, toeval beweging, tegenvoorbeelden van model controleren, impasse/beëindigen van state coverage en sequenties. Paarsgewijs en partiele orde technieken zijn opties voor het verkleinen van test suites.
- Het ondersteunt een aantal programmeertalen (Java, C, C++, C#, PHP, Python, HTML en VB) en test frameworks (bijvoorbeeld xUnit, winde van het Selenium en Robot Framework) voor offline test uitvoering.
- Het is geschikt voor OTF testen en online uitvoering van gegenereerde tests via Selenium WebDriver of een Remote Produce Call(RPC)-protocol.
- MISTA kan gebruikt worden voor functionele testen, acceptatie testen, GUI testen, beveiligingstesten en programmeur testen.

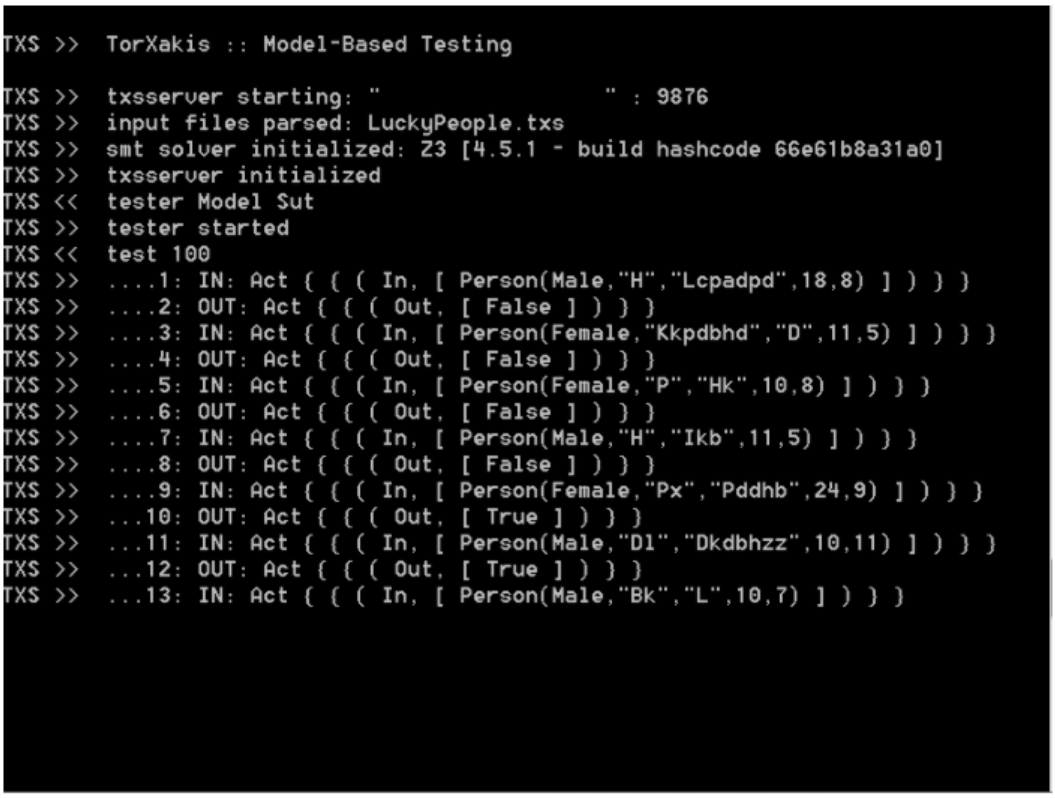
De gegevens die er gevonden zijn over MISTA zijn enkel terug te vinden in onderzoek documenten. Er is geen officiële webpagina van te vinden.

### 2.2.4 Wat is TorXakis

TorXakis maakt gebruikt van een specificatie om SUT te testen of om een systeem te simuleren. Een specificatie van TorXakis bestaat uit drie delen:

- Model: samenstelling van exemplaren van processen die het gedrag van SUT modelleren.
- Adapter: specificaties van SUT abstract in kaart brengen.
- Verbinding: beschrijft externe interfaces van SUT of van gesimuleerde systemen.

TorXakis maakt gebruik van de theorie van de Input-Output Conformance (IOCO) om het exacte gedrag te vergelijken met het waarneembare gedrag. Verder is TorXakis een volledig command-line hulpmiddel waarmee geen specificatie bestanden nodig zijn als invoer.




*Figuur 26 TorXakis*

## 2.3 Resultaten onderzoeksrapport

### 2.3.1 Tools verkennen

| | Spec Explorer | GraphWalker | MISTA |
|---|---|---|---|
| Open source | Commercieel | Open source | Geen informatie gevonden |
| Licentie | Geen informatie gevonden | MIT | Commercieel, gratis voor academische instituties |
| Release | 2002 | 2010 | 2013 |
| Prijs | Geen informatie gevonden | Geen | Geen informatie gevonden |
| Soort Testing | System testing | System under test, Function testing, Stability testing, Smoke testing | Function testing, acceptance testing, GUI testing, security testing, programmer testing |
| Test Frameworks | Visual Studio, NUnit | Selenium, Appium, Maven(tool), Junit | JUnit, NUnit, Selenium IDE, Robot Framework |
| Documentatie | Officiële pagina: Microsoft | Officiële pagina: Github | Enkel research, geen officiële pagina |
| Cross platform | Geen informatie gevonden | yEd Graph Editor | Geen informatie gevonden |
| Maakt gebruik van MBT | Tool voor MBT | Tool voor MBT | Tool voor MBT |
| Codeer talen | C++,C#,Java,.Net,Perl | Java | Java,C,C++,C#,HTML,VB |

| | TorXakis |
|---|---|
| Open source | Commercieel |
| Licentie | BSD3 licentie |
| Release | Geen informatie gevonden |
| Prijs | Geen informatie gevonden |
| Soort Testing | System under test |
| Test Frameworks | Command line |
| Documentatie | Officiële pagina: Github |
| Cross platform | Ja |
| Maakt gebruik van MBT | Tool voor MBT |
| Codeer talen | Java |

| | | |
|---|---|---|
| Slecht | Gemiddeld | Goed |

*Figuur 27 Vergelijkingsmatrix*

De vergelijkingsmatrix opgesteld voor de uitwerking van de tools kan gezien worden in figuur 24. De resultaten hiervan zullen verder worden uitgelegd in dit hoofdstuk per tool. De vergelijking is gemaakt met oog op Refleqt. Zij coderen bijvoorbeeld in Java, dus wanneer er Java kan gebruikt worden is dit goed, indien er geen Java gebruikt kan worden maar wel andere codeertalen wordt dit gemiddeld.

Met open source gaan we kijken als de tool open source is. Bij licentie wordt er gekeken indien er een licentie nodig is voor deze tool, zo ja welke. Release en prijs spreken voor zich, wanneer is de tool uitgekomen en indien er een prijs betaald moet worden, hoeveel zal dit zijn. Bij soort testing verwijzen we naar welke testen er uitgevoerd kunnen worden per tool. Dit is vooral belangrijk voor Refleqt. Zo kunnen zij op voorhand zien indien zij gebruik kunnen maken van een tool of niet. Met test framework bedoelt men het framework waarop er getest kan worden. Documentatie verwijst naar hoeveel informatie er over de tool te vinden is online. Is dit de officiële website, of word er gebruik gemaakt van een andere website, bv GitHub. Het is voor Refleqt ook zeker van belang dat de tools op cross platform kunnen werken, hiermee wordt er het besturingssysteem bedoelt. Ook belangrijk is als er gebruik gemaakt kan worden van model-based testing, maar omdat deze tools specifieke tools hiervoor zijn is dit overal groen.

Tot slot codeert Refleqt voornamelijk in Java, vandaar het onderdeel codeer talen. Wanneer de tool aan alle vereisten voldoet behalve dat deze geen Java ondersteunt, zal de tool niet gebruikt worden binnen het bedrijf.

#### 2.3.1.1 Spec Explorer

Spec Explorer is een tool voor model-based testen die enkel commercieel gebruikt kan worden. Hij is in 2002 op de markt gekomen, dus het is nogal een redelijke oude tool. Hierdoor wordt deze versie van 2002 niet meer ondersteunt en zijn ze aan het werken aan een nieuwe versie van Spec Explorer. Het kan enkel systeem testen maken en kan enkel gebruikt worden via Visual Studio met NUnit. De officiële documentatie kan teruggevonden worden op pagina van Microsoft. Wel vinden we niets terug over als de tool open source is of niet. Doordat men gebruik maakt van Visual Studio kan er in verschillende programmeertalen geprogrammeerd worden namelijk: C++, C#, Java, .Net en Perl.

#### 2.3.1.2 GraphWalker

GraphWalker is de enige tool uit de selectie die positief is voor Refleqt in alle opzichten. Het is een open source tool met de MIT-licentie en op de markt gekomen in 2010. Het ondersteunt verschillende soorten testen, namelijk: System under test, Function testing, Stability testing en Smoke testing. Verder kan er gewerkt worden met de frameworks Selenium, Appium, JUnit en kan er ook gebruik gemaakt worden van Maven. Verder heeft GraphWalker zijn eigen editor waardoor de tool open source gebruikt kan worden, deze editor is yEd Graph Editor. Tot slot wordt er geprogrammeerd in Java.

#### 2.3.1.3 Mista

Voor Mista is er geen informatie gevonden als deze tool open source is of niet. Ook een officiële documentatie pagina werd er niet teruggevonden, enkel research pagina's. In 2013 is Mista op de markt gebracht met een commerciële licentie. Ook Mista ondersteunt verschillende soorten testen namelijk: Function testing, acceptance testing, GUI testing, security testing en programmer testing. De frameworks waarmee Mista kan werken zijn JUnit; NUnit, Selenium IDE en Robot Framework. Het ondersteunt ook verschillende programeer talen zoals Java, C, C++, C#, Html en Vb. Verdere informatie over Mista is nergens teruggevonden, enkel bovenstaande informatie uit de research pagina's online.

#### 2.3.1.4 TorXakis

Tot slot komen we bij TorXakis. Deze tool is commercieel met een BSD3 licentie. Over de prijs en wanneer deze op de markt gekomen is werd er niets over terug gevonden. Het kan enkel SUT testen maken en werkt volledig via command line. De documentatie over TorXakis is te vinden op hun github pagina. Hier werd er gevonden dat ze ook cross platform werken en het enkel in Java geprogrammeerd kan worden.

### 2.3.2 Aanbevelingen

Als ik de tool mocht kiezen voor Refleqt zou ik voor GraphWalker kiezen. Het enige nadeel aan GraphWalker is dat de documentatie enkel op hun GitHub te vinden is. Dus ze hebben geen website van GraphWalker zelf. De reden waarom ik voor GraphWalker kies is omdat deze tool zeer veel potentieel aanbied voor Refleqt. Het ondersteunt de frameworks en programmeertaal waar Refleqt voornamelijk mee werkt. Verder ondersteunt GraphWalker verschillende soorten testen. Dit kan van pas komen wanneer er een applicatie/ software getest moet worden met deze soorten testen.

Wanneer er niet enkel in Java geprogrammeerd wordt en men als test tool in Visual Studio werken, zou ik de tool Spec Explorer aanraden. Deze zal wel aangekocht moeten worden, maar doordat dit door Microsoft ondersteunt wordt, is er ook een goed support wanneer er problemen tevoorschijn komen. Het enige onwetende over deze tool is het cross platform gedeelte.

### 2.3.3 Wat lijkt een volgende logische onderzoeksvraag

Een logische volgende onderzoeksvraag die hierop kan volgen is hoe kan dit gemakkelijk opgenomen worden in een Continuous Integration pipeline.

### 2.3.4 Relevante resultaten voor het stagebedrijf

De resultaten vanuit de vergelijkingsmatrix zijn zeer relevant voor Refleqt. Zo kunnen zij ook al snel een selectie maken van tools die zij in de toekomst willen gebruiken. Ook kunnen ze dan kijken welke testen een model based testing tool het beste aanmaakt. Want zoals voorgaand in dit onderzoeksitem besproken, maakt een model based testing tool de testen automatisch aan op basis van een model dat opgesteld werd aan de hand van de vereisten van de software en/of de app Hierdoor kan Refleqt een ander inzicht krijgen op die bepaalde testen en kunnen ze ook meer focussen op de test cases die deze tool niet automatisch genereert.

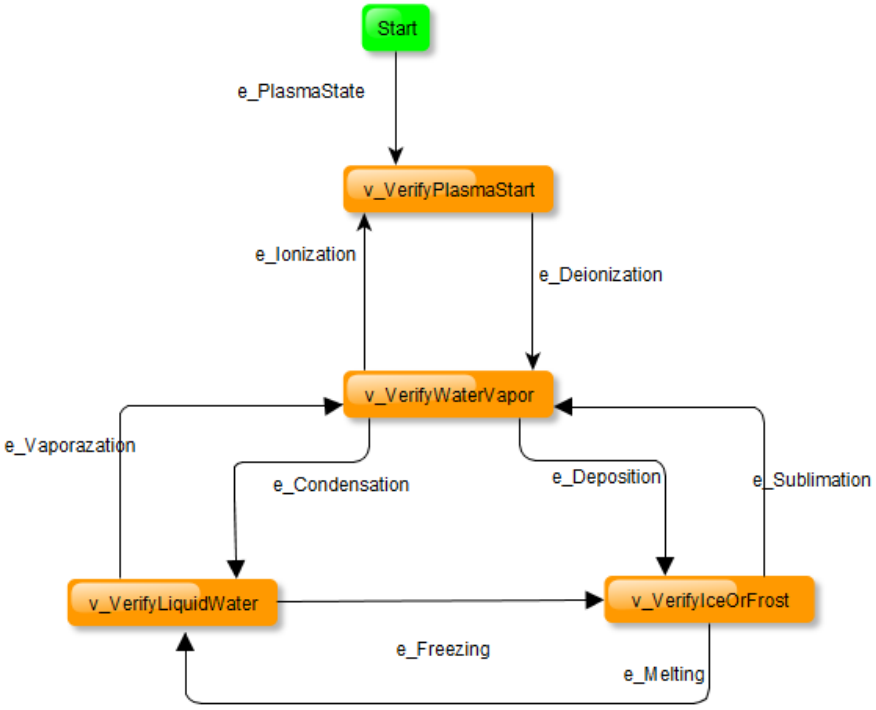
## 2.4 Proof of Concept

In deze proof of concept is de tool die het best representerend is voor Refleqt uit de vergelijkingsmatrix genomen en zullen er twee verschillende testcases worden gemaakt.

Om te beginnen is er gekozen voor de tool GraphWalker. Dit doordat deze de beste kwaliteiten biedt voor Refleqt. GraphWalker kan geïmplementeerd worden in IntelliJ samenwerkend met Maven, hier zijn ook de twee testcases in uitgewerkt.

### 2.4.1 Testcase zonder parameters

Om kennis te maken met de tool te maken zijn we begonnen met een simpele testcase. Een case zonder parameters.




*Figuur 28 GraphWalker model zonder parameters*

Vooraleer men van start kan gaan met het aanmaken van het model, moet er een project worden aangemaakt. Dit gebeurt via de command line en een speciaal commando dat terug te vinden is op de website van GraphWalker. Hierna moet er dus een model worden aangemaakt. In dit model definieert men welke acties en functies er uitgevoerd moeten worden. Op bovenstaand model kan men zien dat er met e_ of v_ gewerkt wordt, dit zijn edges en vertex. Een vertex is eigenlijk de functie die opgeroepen wordt en een edge is de actie die uitgevoerd moet worden. Dit was het enige wat men moet invullen als user. De rest wordt door GraphWalker automatisch gegenereerd op basis van dit model. Wel is het belangrijk om te weten dat men eerst een commando uit moet voeren voordat GraphWalker dit ook effectief uitvoert. Wat GraphWalker dan genereerd is een interface waar de functies en acties in gedeclareerd staan en een test.java bestand dat gebaseerd is op de interface.

De testen lopen in de volgorde van het model, dus met andere woorden: hoe het model opgesteld is. De functies worden aangemaakt door GraphWalker, maar de achterliggende code niet. Dit zal de gebruiker ook zelf moeten doen. Wanneer er verschillende mogelijkheden zijn en er ingesteld is dat men 100% coverage wilt, zal GraphWalker alle verschillende soorten testcases afgaan die er gegenereerd kunnen worden op basis van het model. Wel zal er indien het model bijgewerkt wordt, nagekeken moeten worden of alle code ook bijgewerkt is. Concreet voorbeeld hiervan is wanneer er 'freezing' staat, maar dit veranderd wordt naar 'e_Freezing', dat de namen van de functies ook mee veranderen.

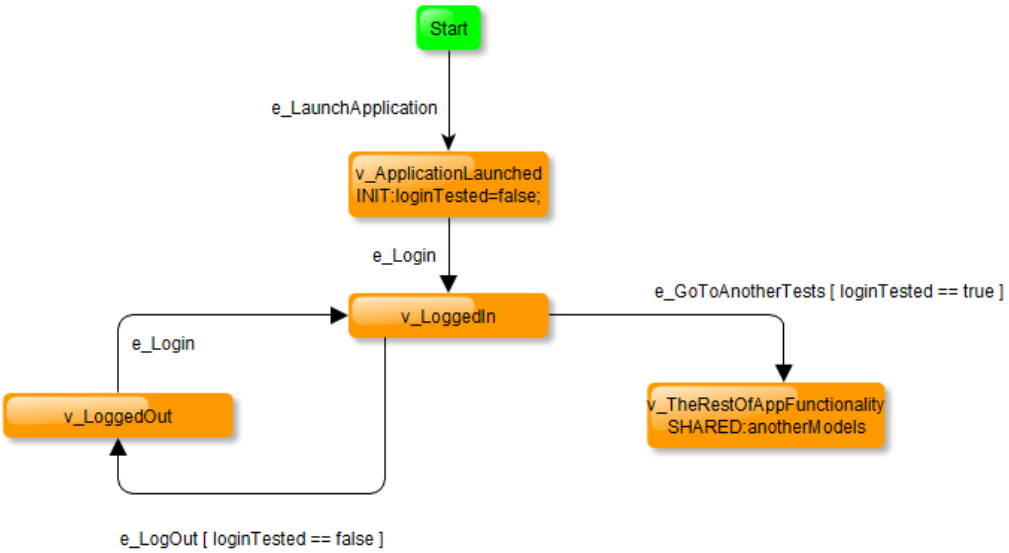
```
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] Result :
[INFO]
[INFO] {
  "totalFailedNumberOfModels": 0,
  "totalNotExecutedNumberOfModels": 0,
  "totalNumberOfUnvisitedVertices": 0,
  "verticesNotVisited": [],
  "totalNumberOfModels": 1,
  "totalCompletedNumberOfModels": 1,
  "totalNumberOfVisitedEdges": 9,
  "totalIncompleteNumberOfModels": 0,
  "edgesNotVisited": [],
  "vertexCoverage": 100,
  "totalNumberOfEdges": 9,
  "totalNumberOfVisitedVertices": 4,
  "edgeCoverage": 100,
  "totalNumberOfVertices": 4,
  "totalNumberOfUnvisitedEdges": 0
}
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

*Figuur 29 GraphWalker resultaat zonder parameters*

Zoals men kan zien is bovenstaande afbeelding een klein snapshot van de logfile van de test. Wat niet te zien is op de afbeelding is in welke volgorde GraphWalker de testen uitgevoerd heeft. Wel kan men zien dat er een volledige coverage werd bereikt, dit ziet men omdat zowel bij 'vertexcoverage' als 'edgecoverage' 100 staat. Ook laat GraphWalker weten hoeveel edges en hoeveel vertices er overlopen zijn. Hier kan men als kleine check via het model natellen wanneer er niets vergeten is.

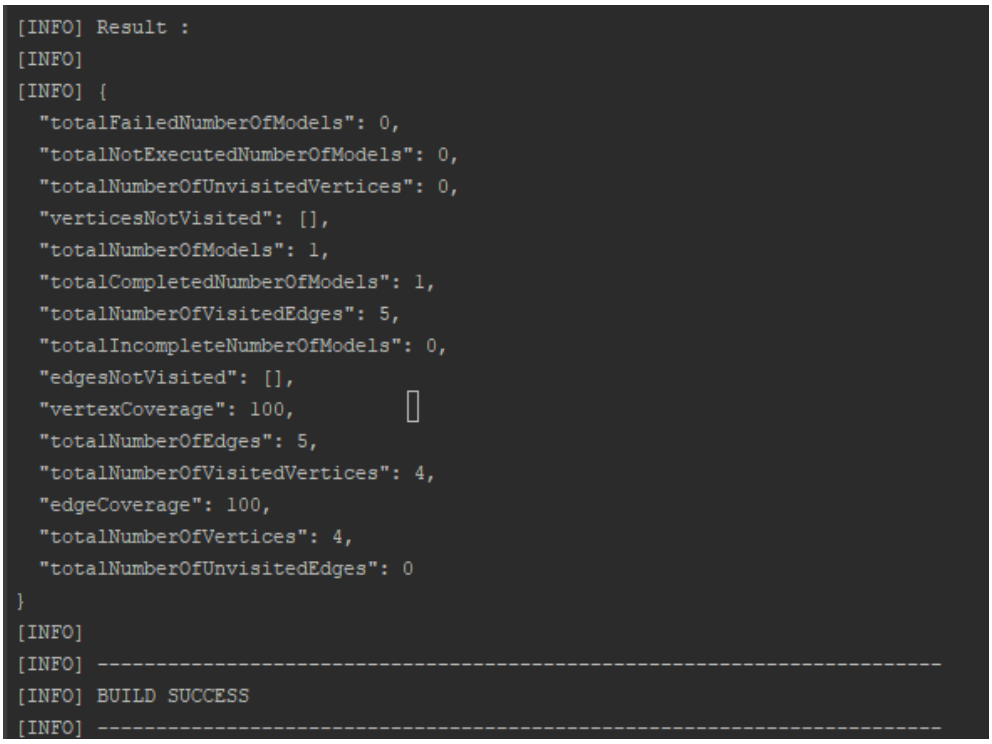
### 2.4.2 Testcase met parameters

Na kennismaking met de tool probeerden we het iets moeilijker te maken: Een model met gebruik van parameters. Ook hier is de basis hetzelfde. Het verschil vergeleken met de vorige testcase is dat men hier gebruik maakt van guards. Guards zijn rollen zoals in een if statement en hierdoor komt een edge in aanmerking om uitgevoerd te worden of niet.




*Figuur 30 GraphWalker model met parameters*

Als dit model gegenereerd is, zal de gebruiker zelf nog test data moeten aanmaken om door de lus te gaan. In dit voorbeeld is er gewerkt met een login, dus in de achterliggende code is er gewerkt met lijsten en iterators. Wanneer dit niet gegenereerd wordt zal er een oneindige lus tevoorschijn komen. Ook is het belangrijk dat er niet vergeten wordt om de variabele op true te zetten. Anders kan GraphWalker niet naar de 'v_TheRestOfAppFuntionality' vertex gaan.




*Figuur 31 GraphWalker resultaat met parameters*

Zoals bij de vorige testcase is ook hier een klein snapshot genomen van de logfiles. Hier is er te zien dat er in totaal vijf edges en vier vertices zijn en er een coverage is van 100%. In de bijlagen treft u de volledige logfiles aan in een tabel.

### 2.4.3 Persoonlijke reflectie

Bij het begin van de stage vond ik deze onderzoeksopdracht niet duidelijk. Ik wist niet 100% wat er van mij verwacht werd of hoe ik een goed onderzoeksopdracht moest maken. Ook wist ik voor deze opdracht zelf niet wat model-based testing is en wat het allemaal inhoud.  Na verloop van tijd werd het allemaal veel duidelijker. Dankzij deze opdracht weet ik zelf ook wat model-based testing is en hoe men dit kan toepassen. Welke tools hiervoor goed zijn en welke niet. Doordat ik bij het begin zelf niet wist wat model-based testing inhield, ben ik opzoek gegaan naar goede artikelen en tutorials om dit te snappen. Ik heb mijn uiterste best hier gedaan en alles gegeven om deze onderzoeksopdracht tot een goed einde te brengen. Uiteindelijk naargelang mijn gevoel denk ik wel dat ik goede conclusies getrokken heb uit de informatie die ik hierover heb kunnen vinden.

Nu hoop ik dat de door mij voorgestelde tools en dit onderzoekstopic zelf genoeg informatie bevatten om Refleqt een aanzet te geven naar het gebruik van model-based testen.

# Bibliografie


[1] „Youtube,” [Online]. Available: https://www.youtube.com/watch?v=uJY50JuJ8N8&t=0s. [Geopend 4 oktober 2017].

[2] „Community SmartBear,” SmartBear, [Online]. Available: https://community.smartbear.com/t5/SoapUI-Pro/Property-Transfer-Between-Test-Cases/td-p/22732. [Geopend 11 oktober 2017].

[3] „SoapUI,” SoapUI, [Online]. Available: https://www.soapui.org/. [Geopend 4 oktober 2017].

[4] „Functional testing,” SoapUI, [Online]. Available: https://www.soapui.org/functional-testing/teststep-reference/property-transfers.html. [Geopend 11 oktober 2017].

[5] „SoapUI introduction,” Tutorialspoint, [Online]. Available: https://www.tutorialspoint.com/soapui/soapui_introduction.htm. [Geopend 4 oktober 2017].

[6] „Script testing,” SoapUI, [Online]. Available: https://www.soapui.org/articles/script-teststep-branching.html. [Geopend 19 oktober 20117].

[7] „Support SmartBear Groovy,” SmartBear, [Online]. Available: https://support.smartbear.com/readyapi/docs/soapui/steps/groovy.html.

[8] „Support SmartBear Properties,” SmartBear, [Online]. Available: https://support.smartbear.com/readyapi/docs/soapui/steps/property-transfer.html. [Geopend 11 oktober 2017].

[9] „FyiCenter loop a test step,” FyiCenter, [Online]. Available: http://sqa.fyicenter.com/FAQ/SoapUI/How_to_loop_a_sequence_of_TestSteps_X_times.html. [Geopend 25 oktober 2017].

[10] „SoapUI Groovy scripting,” Softwaretestinghelp, [Online]. Available: http://www.softwaretestinghelp.com/soapui-tutorial-9-groovy-scripting-concepts/. [Geopend 26 oktober 2017].

[11] „Groovy script tips,” learnsoapui, [Online]. Available: https://learnsoapui.wordpress.com/2011/07/17/10-groovy-scripts-on-your-finger-tips-soapui/. [Geopend 26 oktober 2017].

[12] „Header value as property,” stackoverflow, [Online]. Available: https://stackoverflow.com/questions/15929941/passing-a-header-value-as-a-property . [Geopend 2 november 2017].

[13] „For loop in a test case,” Smartbear, [Online]. Available: https://community.smartbear.com/t5/SoapUI-Open-Source/For-Loop-in-a-Test-Case/td-p/16507. [Geopend 2 november 2017].

[14] „Core testing guide,” Groovy, [Online]. Available: http://docs.groovy-lang.org/next/html/documentation/core-testing-guide.html. [Geopend 3 november 2017].

[15] „Groovy assert examples," Grails, [Online]. Available: http://grails.asia/groovy-assert-examples. [Geopend 3 november 2017].

[16] „Groovy map tutorial," Grails, [Online]. Available: http://grails.asia/groovy-map-tutorial. [Geopend 6 november 2017].

[17] „Loop trough map in groovy," Stackoverflow, [Online]. Available: https://stackoverflow.com/questions/10037374/loop-through-map-in-groovy. [Geopend 6 oktober 2017].

[18] „Jsonpath," jsonpath, [Online]. Available: https://jsonpath.curiousconcept.com/. [Geopend 7 november 2017].

[19] „Jsonlint," jsonlint, [Online]. Available: https://jsonlint.com. [Geopend 7 november 2017].

[20] „Goessner," goessner, [Online]. Available: http://goessner.net/articles/JsonPath/. [Geopend 7 november 2017].

[21] „Testing support," soapui, [Online]. Available: https://www.soapui.org/soapui-projects/team-testing-support.html. [Geopend 20 november 2017].

[22] „Model-based testing," wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Model-based_testing. [Geopend 17 november 2017].

[23] „Ibm developerworks," ibm, [Online]. Available: https://www.ibm.com/developerworks/community/blogs/35dfcb99-111b-423a-aaa4-50f3fddae141/entry/Model_Based_Testing_what_it_is_and_what_does_it_entail?lang=en. [Geopend 17 november 2017].

[24] „Methods and tools," methodsandtools, [Online]. Available: http://www.methodsandtools.com/archive/archive.php?id=102. [Geopend 17 november 2017].

[25] „Microsoft SpecExplorer," Microsoft, [Online]. Available: https://blogs.msdn.microsoft.com/somasegar/2009/10/26/spec-explorer-a-model-based-testing-tool/. [Geopend 1 12 2017].

[26] „GitHub TorXakis," TorXakis, [Online]. Available: https://github.com/TorXakis/TorXakis. [Geopend 1 12 2017].

[27] „Graphwalker," Github, [Online]. Available: Available: http://graphwalker.github.io/. [Geopend 1 12 2017].

[28] „Mista," Mista, [Online]. Available: http://model-based-testing.info/2012/02/12/mista-a-flexible-mbt-tool-and-it-is-free/. [Geopend 1 12 2017].

[29] „TestOptimal," mbt, [Online]. Available: http://mbt.testoptimal.com/index.html. [Geopend 1 12 2017].

[30] „On-The-Fly testing," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ff976543.aspx?f=255&MSPPError=-2147217396. [Geopend 17 12 2017].

[31] „Spec Explorer Overview," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ee620518.aspx. [Geopend 17 12 2017].

[32] „Spec Explorer," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ee620411.aspx. [Geopend 17 12 2017].

[33] „MISTA," Onderzoek, [Online]. Available: https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db/mista.html. [Geopend 17 12 2017].

[34] „MISTA: Model-based Integration and System Test Automation," Onderzoek, [Online]. Available: http://cs.boisestate.edu/~dxu/research/MBT.html. [Geopend 17 12 2017].

[35] „TestOptimal tutorial list," Testoptimal, [Online]. Available: http://testoptimal.com/tutorials/TutorialList.html. [Geopend 17 12 2017].

[36] „TestOptimal," TestOptimal, [Online]. Available: http://mbt.testoptimal.com/index.html. [Geopend 17 12 2017].

[37] „PDF TestOptimal," Testnet, MBT, [Online]. Available: https://www.testnet.org/testnet/download/preview-voorjaar-2015/presentatie-testnet-voorjaarsevent-20150430-testoptimal.pdf. [Geopend 17 12 2017].

[38] „TorXakis wiki," github, [Online]. Available: https://github.com/TorXakis/TorXakis/wiki. [Geopend 17 12 2017].

[39] „Readme TorXakis," github, [Online]. Available: https://github.com/TorXakis/TorXakis/blob/develop/README.md. [Geopend 17 12 2017].

# Bijlagen

A. **ReadyAPI implementeren in Jenkins**

B. **Excel bestand**

C. **GraphWalker proof of concept logfiles**

# A. Omschrijving Bijlage A, ReadyAPI implementeren in Jenkins

Created by Zengers Babette, last modified on Nov 17, 2017

### 1. Create and configure Jenkins nodes

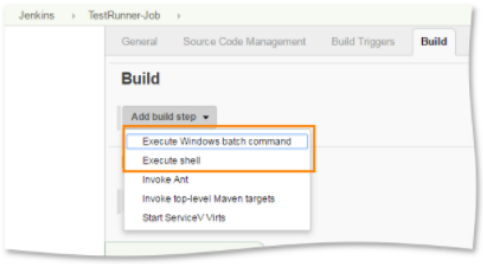
Jenkins documentation includes a detailed guide on how to configure Jenkins nodes. Follow the instructions on the following page:

https://wiki.jenkins.io/display/JENKINS/Step+by+step+guide+to+set+up+master+and+slave+machines+on+Windows

### 2. Install ReadyAPI on your Jenkins nodes

If you want to install ReadyAPI automatically you need to do the following:

1. Use the 'Execute batch' command build step. The way you add it depens on the OS of your Jenkins node.




2. Configure the step to run the ReadyAPI installer. Add the -q argument to the installer command line to run the installation in silent mode. This means that it will run withouth any user interaction and use the default values for all options.
    1. Here is an example command line that installs ReadyAPI in linux:
       "sh ReadyApi-x64-2.2.0.sh"

### 3. Get ReadyAPI License

Before you go further, you first need to find out what type of licence key you have. If you have a floating key you have to follow 3.1, if you have a fixed key you have to go to 3.2.

**3.1 Floating key:**

#### 3.1.1 Jenkins

On your Jenkins node you have to create a text file that will cointain the code of the ReadpAPI application, whose license you want to consume:

| | |
|---|---|
| 1 | SoapUI |
| 2 | LoadUI |
| 3 | Secure |
| 4 | ServiceV |
| 5 | VirtServer |
| 6 | TestServer |

#### 3.1.2 ReadyAPI license

To get the ReadyAPI license, run the License Manager .jar file (ready-api-license-manager-1.1.jar) from the command line. To do this, add another *Execute batch command* step to your Jenkins job. Use the `-s license-server-IP:portfile` argument to specify the license server from which the .jar file will consume a Floating license:java -jar c:\path\ready-api-license-manager-1.1.jar -s 10.0.81.128:1099 < c:\readyapi-folder\MyCode.txt

Here, MyCode.txt is the file that contains the code of the licensed ReadyAPI application (see above).

#### 3.1.3 Copy Test Projects and Data Files to Nodes

To run ReadyAPI tests from a Jenkins node, you need to copy your ReadyAPI project to this node. You also need to copy the data files your tests use. To copy the files, use the Jenkins *Copy file* build step. A possible alternative is to place the project and data file to some shared network folder to which Jenkins nodes have access.

**3.2 Fixed key:**

You can activate licenses by using a command-line licence manager.

- Download the license manager package application from the SmartBear website.
- Run the license .jar file with -f <licence file> option
  Example: "java -jar ready-api-license-manager-1.1.jar -f SoapUILicense.key"
- Choose your activation mode (Online/Offline)

#### 3.2.1 Online activation procedure

If you select the online activation, the licence manager will ask you to provide next credentials:

- Title
- First name
- Last name
- Company name
- Address

*Figuur 32 ReadyAPI documentatie zelfgeschreven*

If you are done specifying your credentials, the license manager will now activate your licence.

**4. Run Tests**

If you want to run your tests, you have to use one of the ReadyAPI command-line test runners:

- **Test Runner** – Runs SoapUI functional tests.
- **LoadUITestRunner** – Runs load tests created with LoadUI.
- **SecurityTestRunner** – Runs security tests created in ReadyAPISecure.
- **VirtRunner** – Starts or stops a virtual API created with ServiceV.

To run any of these, you need to add an execute shell command step to your Jenkins job. This depens on what OS your Jenkins node has. Here is an example command line that starts a functional ReadyAPI test on a Ubuntu Jenkins node. This will run all the tests in ReadyAPI.

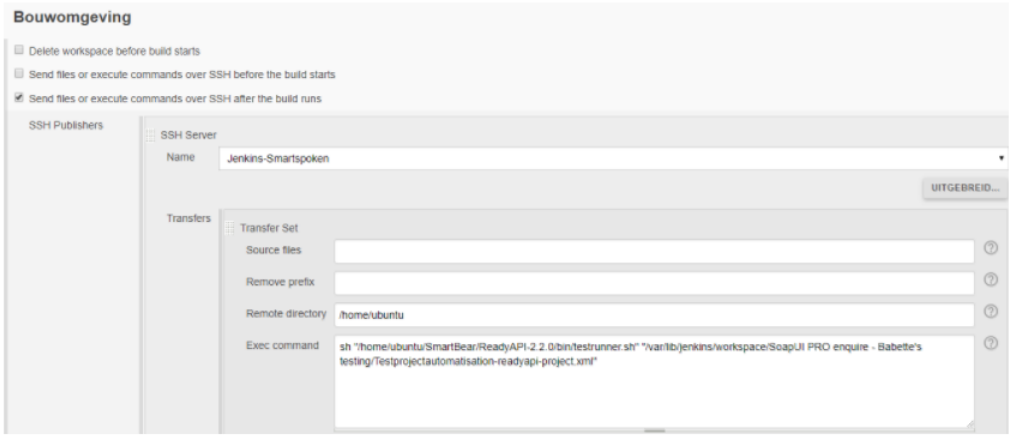
sh "/home/ubuntu/SmartBear/ReadyAPI-2.2.0/bin/testrunner.sh" "/var/lib/jenkins/workspace/SoapUI PRO enquire - Babette's testing/Testprojectautomatisation-readyapi-project.xml"

**5. Jenkins licence not found**

When you run your tests through the test runner from Jenkins, you may encounter the "lincence not found" error.

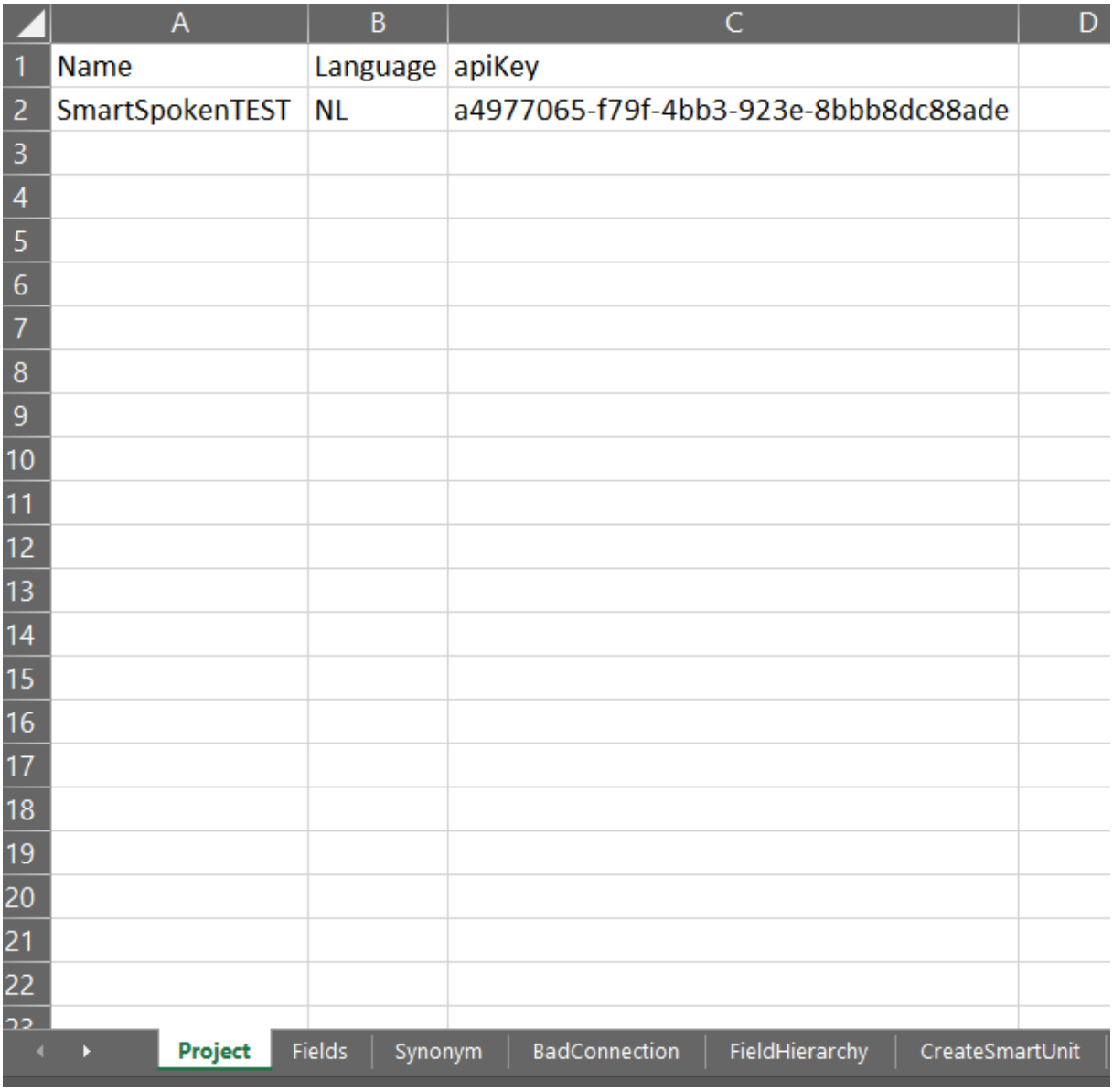
If this is your issue, you can visit the following website and follow their steps: https://support.smartbear.com/readyapi/docs/general-info/licensing/troubleshooting/jenkins.html

After reading the documentation we came up with the following solution. This replaced the build step you made previously.




*Figuur 33 ReadyAPI implementeren in Jenkins*

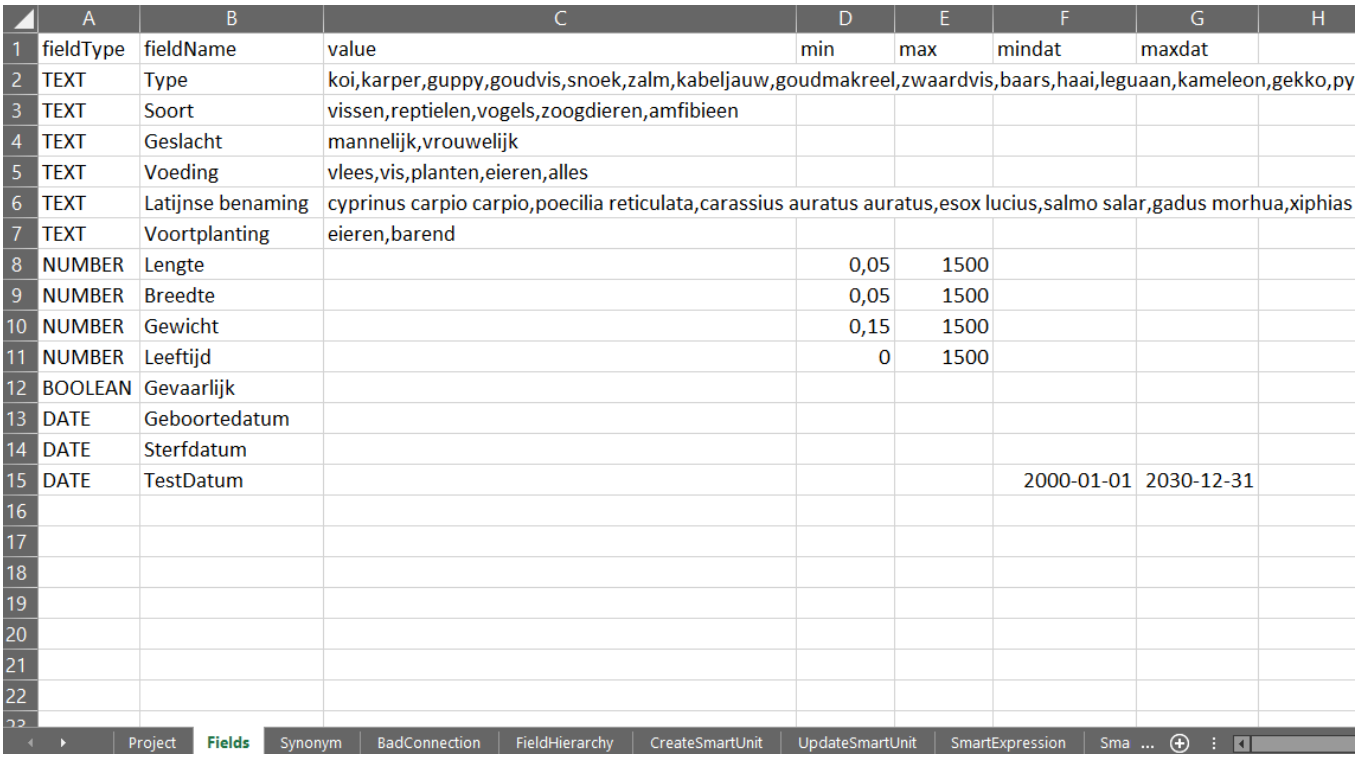
## B. Omschrijving bijlage B, Excel bestand



| | A | B | C | D |
|---|---|---|---|---|
| 1 | Name | Language | apiKey | |
| 2 | SmartSpokenTEST | NL | a4977065-f79f-4bb3-923e-8bbb8dc88ade | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |
| 16 | | | | |
| 17 | | | | |
| 18 | | | | |
| 19 | | | | |
| 20 | | | | |
| 21 | | | | |
| 22 | | | | |

Project | Fields | Synonym | BadConnection | FieldHierarchy | CreateSmartUnit

*Figuur 34 Excel bestand: project*



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | fieldType | fieldName | value | min | max | mindat | maxdat | |
| 2 | TEXT | Type | koi,karper,guppy,goudvis,snoek,zalm,kabeljauw,goudmakreel,zwaardvis,baars,haai,leguaan,kameleon,gekko,py | | | | | |
| 3 | TEXT | Soort | vissen,reptielen,vogels,zoogdieren,amfibieen | | | | | |
| 4 | TEXT | Geslacht | mannelijk,vrouwelijk | | | | | |
| 5 | TEXT | Voeding | vlees,vis,planten,eieren,alles | | | | | |
| 6 | TEXT | Latijnse benaming | cyprinus carpio carpio,poecilia reticulata,carassius auratus auratus,esox lucius,salmo salar,gadus morhua,xiphias | | | | | |
| 7 | TEXT | Voortplanting | eieren,barend | | | | | |
| 8 | NUMBER | Lengte | | 0,05 | 1500 | | | |
| 9 | NUMBER | Breedte | | 0,05 | 1500 | | | |
| 10 | NUMBER | Gewicht | | 0,15 | 1500 | | | |
| 11 | NUMBER | Leeftijd | | 0 | 1500 | | | |
| 12 | BOOLEAN | Gevaarlijk | | | | | | |
| 13 | DATE | Geboortedatum | | | | | | |
| 14 | DATE | Sterfdatum | | | | | | |
| 15 | DATE | TestDatum | | | | 2000-01-01 | 2030-12-31 | |

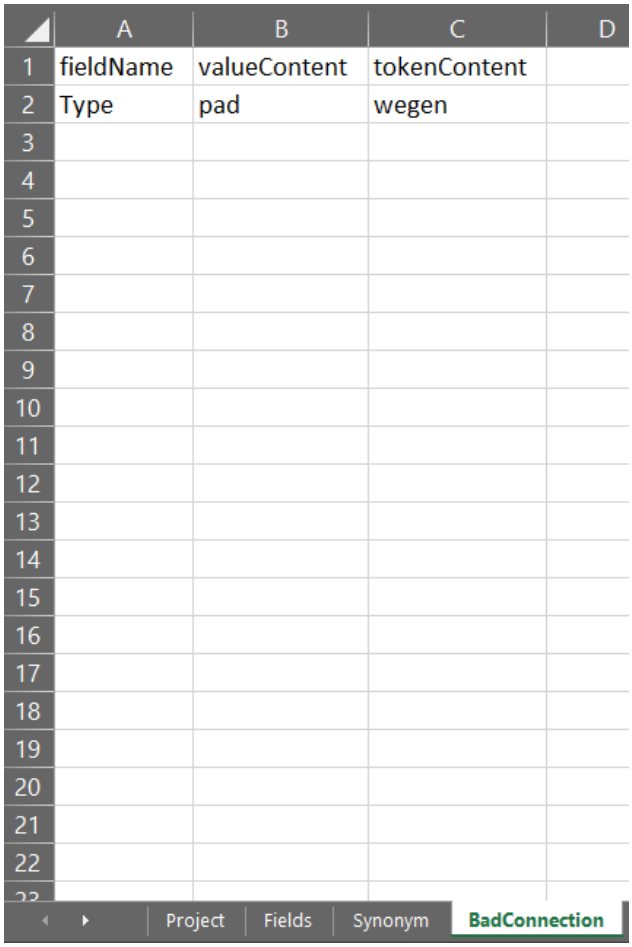


*Figuur 37 Excel bestand: fields*

| | A |
|---|---|
| 1 | Synonyms |
| 2 | mannetje,mannetjes,man,mannelijk |
| 3 | vrouwtje,vrouwtjes,vrouw,vrouwelijk |
| 4 | leggen,voortplanting |
| 5 | luipaard,panter |
| 6 | eten,voeding |
| 7 | lang,lengte |
| 8 | breed,breedte |
| 9 | baart,barend |
| 10 | geboren,geboortedatum |
| 11 | gestorven,sterfdatum |



*Figuur 36 Excel bestand: synonyms*



| | A | B | C | D |
|---|---|---|---|---|
| 1 | fieldName | valueContent | tokenContent | |
| 2 | Type | pad | wegen | |



*Figuur 35 Excel bestand: badconnection*

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Name | firstLevel | firstFieldName | secondLevel | secondFieldName |
| 2 | Diersoorten | 1 | Type | 2 | Soort |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |

Project | Fields | Synonym | BadConnection | **FieldHierarchy** | CreateSmart

*Figuur 38 Excel bestand: fieldhierarchy*

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | naam | symbool | conversionnaam | conversionsymbool | ratio |
| 2 | jaar | j | maanden | mnd | 12 |
| 3 | | | weken | w | 52 |
| 4 | | | dagen | d | 365,25 |
| 5 | kilogram | kg | gram | g | 100 |
| 6 | | | decigram | dg | 10000 |
| 7 | | | centigram | cg | 100000 |
| 8 | meter | m | decimeter | dm | 10 |
| 9 | | | centimerer | cm | 100 |
| 10 | | | millimeter | mm | 1000 |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |

... | FieldHierarchy | **CreateSmartUnit** | UpdateSmartUnit | SmartExpression | Sm

*Figuur 39 Excel bestand: createsmartunit*

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | fieldname | fieldtype | symb | | | |
| 2 | Leeftijd | NUMBER | j | | | |
| 3 | Gewicht | NUMBER | kg | | | |
| 4 | Lengte | NUMBER | m | | | |
| 5 | Breedte | NUMBER | m | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |
| 18 | | | | | | |
| 19 | | | | | | |
| 20 | | | | | | |
| 21 | | | | | | |
| 22 | | | | | | |

... FieldHierarchy CreateSmartUnit **UpdateSmartUnit**

*Figuur 40 Excel bestand: updatesmartunit*

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | SmartExpressions | fieldname | fieldcontent | | |
| 2 | carnivoor | Voeding | vlees | | |
| 3 | herbivoor | Voeding | planten | | |
| 4 | piscivoor | Voeding | vis | | |
| 5 | ovivoor | Voeding | eieren | | |
| 6 | omnivoor | Voeding | alles | | |
| 7 | vleesetend | Voeding | vlees | | |
| 8 | plantetend | Voeding | planten | | |
| 9 | visetend | Voeding | vis | | |
| 10 | eierenetend | Voeding | eieren | | |
| 11 | alleseter | Voeding | alles | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |

... FieldHierarchy CreateSmartUnit UpdateSmartUnit **SmartExpression**

*Figuur 41 Excel bestand: smartexpression*

| | A | B | C |
|---|---|---|---|
| 1 | name | subname | keywords |
| 2 | lt | filterout | <,kleiner dan,lager dan,minder dan,hoogstens,onder,maximum,max,tot,minder,jonger,geleden |
| 3 | gt | filterout | >,groter dan,hoger dan,meer dan,minstens,min,vanaf,of meer,ouder,boven,binnen |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 12 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | | | |
| 17 | | | |
| 18 | | | |
| 19 | | | |
| 20 | | | |
| 21 | | | |
| 22 | | | |

... | FieldHierarchy | CreateSmartUnit | UpdateSmartUnit | SmartExpression | **SmartTerm** | Autocomplete | Vragen

*Figuur 42 Excel bestand: smartterm*

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **autocomplete** | **Type** | **Soort** | **Geslacht** | **Voeding** | **Latijnse benaming** | **Voortplanting** |
| 2 | vr | | | vrouwelijk | | | |
| 3 | ma | marmot,marter | | mannelijk | | martes,marmota,ursus maritimus | |
| 4 | ja | jaguar | | | | | |
| 5 | oli | olifant | | | | | |
| 6 | vi | | vissen | | vis | | |
| 7 | vl | vlinder,vleermuis | | | vlees | | |
| 8 | rep | | reptielen | | | | |
| 9 | vo | | vogels | | | | |
| 10 | zoo | | zoogdieren | | | | |
| 11 | ei | | | | eieren | | eieren |
| 12 | amf | | amfibieen | | | | |
| 13 | am | | amfibieen | | | hippopotamus amphibius | |
| 14 | j | jaguar | | | | | |
| 15 | gou | goudkat,goudvis,goudmakreel | | | | | |
| 16 | do | dolfijn | | | | | |
| 17 | zw | zwaardvis | | | | | |
| 18 | ki | kikker | | | | | |
| 19 | pa | pad,paard,panda,panter | | | | pan,panthera leo,panthera onca,panthera pardus | |
| 20 | ro | | | | | | |
| 21 | cy | | | | | cyprinus carpio carpio | |
| 22 | ee | eekhoorn | | | | | |
| 23 | cha | | | | | chamaeleonidae | |
| 24 | ig | | | | | iguana iguana | |
| 25 | gr | | | | | | |
| 26 | le | leeuw,leguaan | | | | lepidoptera,panthera leo | |
| 27 | mo | mol,mot | | | | gadus morhua | |
| 28 | sch | schaap,schildpad | | | | schreckensteiniidae | |
| 29 | ze | zeeleeuw | | | | | |

*Figuur 43 Excel bestand: autocomplete deel 1*

| | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|
| 1 | **SmartSynonym** | **Day** | **SmartUnit** | **SmartTerm** | **Month** | **Field** | **SmartExpression** |
| 2 | vrouw,vrouwtje,vrouwtjes | vrijdag | | | | | |
| 3 | man,mannetje,mannetjes | maandag | maanden | | maart | | |
| 4 | | | jaar | | januari | | |
| 5 | | | | | | | |
| 6 | | | | | | | visetend |
| 7 | | | | | | | vleesetend |
| 8 | | | | | | | |
| 9 | voortplanting | | | | | Voeding,Voortplanting | |
| 10 | | | | | | | |
| 11 | | | | | | | eierenetend |
| 12 | | | | | | | |
| 13 | | | | | | | |
| 14 | | | j,jaar | | juli,juni,januari | | |
| 15 | | | | | | | |
| 16 | | donderdag | | | | | |
| 17 | | | | | | | |
| 18 | | | kilogram | | | | |
| 19 | | | | | | | |
| 20 | | | | rond | | | |
| 21 | | | | | | | |
| 22 | | | | | | | |
| 23 | | | | | | | |
| 24 | | | | | | | |
| 25 | | | gram | groter dan of gelijk aan | | | |
| 26 | leggen | | | | | | |
| 27 | | | | | | | |

*Figuur 45 Excel bestand: vragen*

*Figuur 44 Excel bestand: autocomplete deel 2*

## C. Omschrijving bijlage C, GraphWalker proof of concept logfiles

| Logfiles testcase één, zonder parameters. |
| --- |
| [INFO] ------------------------------------------------------------------------<br>16:12:53.896 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:53.899 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_PlasmaState<br>Running: e_PlasmaState<br>16:12:53.987 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:53.992 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_VerifyPlasmaStart<br>Running: v_VerifyPlasmaStart<br>16:12:54.007 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.008 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Deionization<br>Running: e_Deionization<br>16:12:54.015 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.016 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_VerifyWaterVapor<br>Running: v_VerifyWaterVapor<br>16:12:54.024 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.025 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Ionization<br>Running: e_Ionization<br>16:12:54.037 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.038 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_VerifyPlasmaStart<br>Running: v_VerifyPlasmaStart<br>16:12:54.040 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.041 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Deionization<br>Running: e_Deionization<br>16:12:54.041 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.042 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_VerifyWaterVapor<br>Running: v_VerifyWaterVapor<br>16:12:54.042 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.042 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Deposition<br>Running: e_Deposition<br>16:12:54.050 [main] DEBUG o.g.core.machine.SimpleMachine - Context: com.company.SomeSmallTest@5700053f<br>16:12:54.051 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_VerifyIceOrFrost<br>Running: v_VerifyIceOrFrost |

```
16:12:54.059 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.060 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Melting
Running: e_Melting
16:12:54.072 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.073 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyLiquidWater
Running: v_VerifyLiquidWater
16:12:54.083 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.084 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Freezing
Running: e_Freezing
16:12:54.100 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.100 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyIceOrFrost
Running: v_VerifyIceOrFrost
16:12:54.101 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.102 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Melting
Running: e_Melting
16:12:54.102 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.103 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyLiquidWater
Running: v_VerifyLiquidWater
16:12:54.103 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.104 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Vaporazation
Running: e_Vaporazation
16:12:54.110 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.111 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
16:12:54.111 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.112 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Ionization
Running: e_Ionization
16:12:54.112 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.112 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyPlasmaStart
Running: v_VerifyPlasmaStart
16:12:54.113 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.113 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Deionization
Running: e_Deionization
```

```
16:12:54.114 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.114 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
16:12:54.115 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.116 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Ionization
Running: e_Ionization
16:12:54.116 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.118 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyPlasmaStart
Running: v_VerifyPlasmaStart
16:12:54.121 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.124 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Deionization
Running: e_Deionization
16:12:54.125 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.125 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
16:12:54.126 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.126 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Deposition
Running: e_Deposition
16:12:54.131 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.132 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyIceOrFrost
Running: v_VerifyIceOrFrost
16:12:54.134 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.135 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Melting
Running: e_Melting
16:12:54.136 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.137 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyLiquidWater
Running: v_VerifyLiquidWater
16:12:54.138 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.139 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Vaporazation
Running: e_Vaporazation
16:12:54.140 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.140 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
```

```
16:12:54.141 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.141 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Condensation
Running: e_Condensation
16:12:54.148 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.148 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyLiquidWater
Running: v_VerifyLiquidWater
16:12:54.150 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.153 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Vaporazation
Running: e_Vaporazation
16:12:54.154 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.155 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
16:12:54.156 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.157 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Ionization
Running: e_Ionization
16:12:54.158 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.159 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyPlasmaStart
Running: v_VerifyPlasmaStart
16:12:54.160 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.161 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Deionization
Running: e_Deionization
16:12:54.161 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.162 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
16:12:54.163 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.163 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Deposition
Running: e_Deposition
16:12:54.164 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.165 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyIceOrFrost
Running: v_VerifyIceOrFrost
16:12:54.166 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.169 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Melting
Running: e_Melting
```

```
16:12:54.171 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.171 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyLiquidWater
Running: v_VerifyLiquidWater
16:12:54.172 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.173 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Freezing
Running: e_Freezing
16:12:54.174 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.174 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyIceOrFrost
Running: v_VerifyIceOrFrost
16:12:54.175 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.175 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_Sublimation
Running: e_Sublimation
16:12:54.188 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@5700053f
16:12:54.190 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_VerifyWaterVapor
Running: v_VerifyWaterVapor
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] Result :
[INFO]
[INFO] {
  "totalFailedNumberOfModels": 0,
  "totalNotExecutedNumberOfModels": 0,
  "totalNumberOfUnvisitedVertices": 0,
  "verticesNotVisited": [],
  "totalNumberOfModels": 1,
  "totalCompletedNumberOfModels": 1,
  "totalNumberOfVisitedEdges": 9,
  "totalIncompleteNumberOfModels": 0,
  "edgesNotVisited": [],
  "vertexCoverage": 100,
  "totalNumberOfEdges": 9,
  "totalNumberOfVisitedVertices": 4,
  "edgeCoverage": 100,
  "totalNumberOfVertices": 4,
  "totalNumberOfUnvisitedEdges": 0
}
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

*Tabel 3 logfiles proof of concept testcase één*

Logfiles testcase twee, met parameters

```
[INFO] ------------------------------------------------------------------------
18:49:35.853 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.853 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_LaunchApplication
18:49:35.914 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.914 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_ApplicationLaunched
18:49:35.914 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.914 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Login
Login with Login1
18:49:35.930 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.930 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_LoggedIn
18:49:35.945 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.945 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
org.graphwalker.core.model.Guard@19ad99fa  loginTested == false
18:49:35.945 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
org.graphwalker.core.model.Guard@5331be15  loginTested == true
18:49:35.945 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_LogOut
18:49:35.961 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.961 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_LoggedOut
18:49:35.961 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.961 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Login
Login with Login2
18:49:35.977 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_LoggedIn
18:49:35.977 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
org.graphwalker.core.model.Guard@19ad99fa  loginTested == false
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
org.graphwalker.core.model.Guard@5331be15  loginTested == true
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_LogOut
18:49:35.977 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_LoggedOut
18:49:35.977 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute e_Login
Login with Login3
18:49:35.977 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute v_LoggedIn
```

```
18:49:35.977 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
org.graphwalker.core.model.Guard@19ad99fa  loginTested == false
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
org.graphwalker.core.model.Guard@5331be15  loginTested == true
18:49:35.977 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
e_GoToAnotherTests
18:49:35.997 [main] DEBUG o.g.core.machine.SimpleMachine - Context:
com.company.SomeSmallTest@32a8ce3
18:49:35.998 [main] DEBUG o.g.core.machine.ExecutionContext - Execute
v_TheRestOfAppFunctionality
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] Result :
[INFO]
[INFO] {
  "totalFailedNumberOfModels": 0,
  "totalNotExecutedNumberOfModels": 0,
  "totalNumberOfUnvisitedVertices": 0,
  "verticesNotVisited": [],
  "totalNumberOfModels": 1,
  "totalCompletedNumberOfModels": 1,
  "totalNumberOfVisitedEdges": 5,
  "totalIncompleteNumberOfModels": 0,
  "edgesNotVisited": [],
  "vertexCoverage": 100,
  "totalNumberOfEdges": 5,
  "totalNumberOfVisitedVertices": 4,
  "edgeCoverage": 100,
  "totalNumberOfVertices": 4,
  "totalNumberOfUnvisitedEdges": 0
}
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

*Tabel 4 logfiles proof of concept testcase twee*