Professionele Bachelor Toegepaste Informatica

Automatisch testen van de Lifeplanner

Jannick Smeets

Promotoren:

Vik Nelissen	Argeüs
Lowie Vangaal	Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Automatisch testen van de Lifeplanner

Jannick Smeets

Promotoren:

Vik Nelissen — Argeüs
Lowie Vangaal — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

In deze dankbetuiging wil ik stilstaan bij de mensen die mij geholpen en gesteund hebben om dit eindwerk te realiseren.

Om te beginnen wil ik volledig Argeüs bedanken voor de kans die ik gekregen heb om in dit mooie bedrijf mijn stage te mogen doen. Binnen Argeüs wil ik dan een paar mensen specifiek bedanken voor hun steun. Als eerste wil ik Reinaut Krekels bedanken voor alle hulp, zowel zijn technische hulp bij het schrijven van de automatische testen als zijn niet technische hulp bij het uitwerken van mijn eindwerk. Daarnaast wil ik ook alle andere stagestudent binnen Argeüs bedanken. Dit zijn Jonas Gerits, Daan Vankerkom en Natasja Vanherck. Zij hebben mij enorm geholpen bij alle vragen die ik had over mijn eindwerk. En als laatste wil ik Ann Vandeput bedanken voor het nalezen van mijn eindwerk en de eventuele fouten die ze eruit heeft kunnen halen.

Buiten Argeüs wil ik mijn PXL promotor Lowie Vangaal bedanken voor alle feedback die hij gegeven heeft op mijn eindwerk.

Om te eindigen wil ik mijn ouders bedanken voor hun steun en het geduld dat zij hebben gehad voor mij de afgelopen 12 weken.

Abstract

Argeüs is een softwarebedrijf binnen de financiële sector met als klanten vooral banken, notarissen, financiële planners, vermogensbeheerders, accountants, juristen en verzekeringsmaatschappijen. De grote sterkte van Argeüs is dat ze hun producten verwezenlijken door middel van cocreatie. Dit wil zeggen dat de klanten van Argeüs worden gezien als creatieve partners in het realiseren van hun producten.

De stageopdracht bestaat erin om de testen op de Lifeplanner te automatiseren. De Lifeplanner is een cocreatie met een belangrijke speler uit de financiële sector en is gebaseerd op de Financial Planning Software van Argeüs met een *Graphical User Interface* (GUI) gemaakt door een externe partij. De Lifeplanner zorgt voor financieel advies en begeleidt klanten door de financiële wereld van onze partner.

Het volledige testproces van Argeüs gebeurt op dit moment manueel. Er zijn al enkele verbeteringen aangebracht in het testproces door middel van een test-GUI en door gebruik te maken van Excel-bestanden met een intern geschreven programma om deze Excel-bestanden uit te lezen en om te zetten naar JSON-bestanden, deze JSON-bestanden kunnen we dan in onze test-GUI importeren om zo gemakkelijk te testen, maar alles moet nog manueel getest worden.

Automatische testen zorgen ervoor dat de regressiefouten onmiddellijk in kaart kunnen worden gebracht. Deze fouten kunnen dan sneller opgelost worden en dit zal zorgen voor een veel hogere kwaliteit van de Lifeplanner. Om de fouten in kaart te brengen wordt gebruikgemaakt van de *test case management software* Testrail. Alle testen die uitgevoerd worden, rapporteren naar Testrail.

De automatische testen worden uitgevoerd in Postman. Deze tool werd al gebruikt binnen Argeüs en hierop wordt verder gewerkt. Er vindt echter wel een onderzoek plaats om te zien of Postman wel de juiste tool is voor Argeüs. Als er uit dit onderzoek een andere testautomatiseringstool komt, dan stapt Argeüs ook over naar deze tool.

Inhoudsopgave

Dankwoord ii
Abstract iii
Inhoudsopgave iv
Lijst van gebruikte figuren vi
Lijst van gebruikte tabellen viii
Lijst van gebruikte afkortingen ix
Inleiding 1
I. Stageverslag 2
1 Bedrijfsvoorstelling 2
1.1 Profiel 2
1.2 Locatie 4
1.3 Organigram 5
2 Voorstelling opdracht 6
2.1 Probleemstelling 6
2.1.1 Situering van het probleem 6
2.1.2 De testpiramide 6
2.2 Doelstelling 8
2.3 Technologieën 8
3 Uitwerking opdracht 9
3.1 De Lifeplanner/ YuMe 9
3.2 Verleden vs toekomst 10
3.2.1 Verleden 10
3.2.2 Toekomst 11
3.2.3 Opbouw Testrail 13
3.3 Uitwerking van de testen 14
3.3.1 Voorbereiding 14
3.3.2 Aanpak 15
3.3.3 Runnen van de testen 17
3.3.4 Resultaat 23
4 Reflectie Stageopdracht 24
II. Onderzoekstopic 25
1 Onderzoeksvraag 25
1.1 Methode van onderzoek 25

2 Onderzoek naar mogelijke tools ... 25
2.1 Literatuurstudie ... 25
2.2 Verwachtingen Argeüs ... 26
2.3 Schema mogelijke tools ... 27
2.3.1 Postman ... 29
2.3.2 Insomnia Rest-Client ... 30
2.3.3 Soap UI ... 31
2.3.4 Katalon ... 32
2.3.5 Tricentis ... 33
2.3.6 JMeter ... 34
2.3.7 Rest-Assured ... 34
2.3.8 Assertible ... 35
2.3.9 Karate DSL ... 36
2.3.10 CitrusFramework ... 37
3 Prototypes ... 38
3.1 Aanpak ... 38
3.2 De uitwerking ... 38
3.2.1 Postman ... 38
3.2.2 Rest-Assured ... 41
3.2.3 CitrusFramework ... 45
3.2.4 Katalon ... 48
3.3 Vergelijking na prototypes ... 52
Conclusie ... 53
Bibliografie ... 54

Lijst van gebruikte figuren

Figuur 1: Successiecalculator 2
Figuur 2: Financial Planning Software 3
Figuur 3: Ligging Argeüs 4
Figuur 4: Google streetview Argeüs 4
Figuur 5: Organigram Argeüs 5
Figuur 6 de testpiramide 6
Figuur 7: YuMe logo 9
Figuur 8: Flow verleden 10
Figuur 9 Flow automatische testen 11
Figuur 10 Opbouw Testrail 13
Figuur 11: Voorbeeld testcase 14
Figuur 12: Opbouw lijst testcases Confluence 15
Figuur 13: Functies binnen Postman 16
Figuur 14: Variabele van functies 17
Figuur 15: Resultaat runner Postman 18
Figuur 16: Resultaat enkel request Postman 19
Figuur 17: Commando voor Newman 19
Figuur 18: Newman rapportering CLI 20
Figuur 19:Newman-Testrail-Reporter commando 20
Figuur 20: APIKey Testrail 21
Figuur 21: Testrail project overzicht 21
Figuur 22: Rapportering test run in Testrail 22
Figuur 23 Postman logo 29
Figuur 24: Insomnia Rest-Client logo 30
Figuur 25: SoapUI logo 31
Figuur 26: Katalon logo 32
Figuur 27: Tricentis logo 33
Figuur 28: JMeter logo 34
Figuur 29: Rest-Assured logo 34
Figuur 30: Assertible logo 35
Figuur 31: Karate logo 36
Figuur 32: CitrusFramework logo 37
Figuur 33: Postman request body 38
Figuur 34: Response body in Postman 39
Figuur 35: Prototype Postman 40
Figuur 36: Rest-Assured pom.xml dependencies 41
Figuur 37: Prototype Rest-Assured 42
Figuur 38: Rest-Assured functie file 43
Figuur 39: Rest-Assured variabelen file 44
Figuur 40: Resultaat Rest-Assured 44
Figuur 41: CitrusFramework pom.xml 45
Figuur 42: Prototype CitrusFramework 46
Figuur 43: CitrusFramework functiefile 47
Figuur 44: CitrusFramework variabelenfile 47
Figuur 45: CitrusFramework EndpointConfig.Java 47
Figuur 46: Prototype Katalon Scripting mode 48

Figuur 47: Prototype Katalon manual mode 49
Figuur 48: Katalon object repository.. 49
Figuur 49: Katalon response body... 50
Figuur 50: Katalon resultaat 51

Lijst van gebruikte tabellen

Tabel 1: Overzicht mogelijke tools 27
Tabel 2: Voor- en nadelen Postman 30
Tabel 3: Voor-en nadelen Insomnia Rest-Client 31
Tabel 4: Voor- en nadelen SoapUI 32
Tabel 5: Voor- en nadelen Katalon 33
Tabel 6: Voor- en nadelen Tricentis 33
Tabel 7: Voor- en nadelen JMeter 34
Tabel 8: Voor- en nadelen Rest-Assured 35
Tabel 9: Voor- en nadelen Assertible 36
Tabel 10: Voor- en nadelen Karate 37
Tabel 11: Voor- en nadelen CitrusFramework 37
Tabel 12: Vergelijking na prototypes 52

Lijst van gebruikte afkortingen

Begrip	Betekenis
Agile	Een softwareontwikkeling in korte, overzichtelijke periodes.
API	Een API is een aanspreekpunt van een computerprogramma.
BDD	BDD is een manier om testen te schrijven. Met BDD kunnen testen geschreven worden op een normaal leesbare manier.
Collection	Een collection is een verzameling van subcollections en testen in Postman.
Continuous Delivery	Continuous Delivery focust zich op het geautomatiseerd overbrengen van software naar de juiste omgevingen.
Continuous Integration	Continuous integration geeft de ontwikkelaars de mogelijkheid om tegelijkertijd samen te werken aan dezelfde code. De veranderingen worden door middel van CI klaargezet om in zijn geheel getest te worden.
End-to-end testen	End-to-end testen zijn testen die de volledige flow van het te testen programma doorlopen. Ze gaan van begin tot einde van het programma om te zien als elk onderdeel wel werkt zoals het moet werken.
GCFT-tool	De GCFT-tool is een interne testtool die Argeüs gebouwd heeft om de response JSON van de Lifeplanner beter visueel te maken.
Git repository	Git repository is een versiebeheersysteem waar de broncode van de programma’s van Argeüs op bijgehouden wordt en waar veranderingen van deze code zichtbaar gemaakt worden.
Maven	Apache maven is een softwaregereedschap voor Java-projectmanagement.
Persona’s	Persona’s zijn test cases. In een persona beschrijf ik een persoon die een deel van de mogelijkheden van de lifeplanner gebruikt wat getest moet worden. Deze persona’s worden opgebouwd aan de hand van een template Excel.
Postman runner	De Postman runner zorgt ervoor dat alle testen gerund kunnen worden in postman.
Pullen	Als je code van de Git repository wil halen moet dit gedaan worden door te pullen van de repository
Pushen	Als je code op de Git repository wil zetten moet dit gedaan worden door te pushen naar de repository.

Quality assurance	Quality assurence zijn de personen die instaan voor de kwaliteit van het de programma's. Dit zijn meestal testers en analisten.
Regressiefout	Een regressiefout is een fout die zich voorgedaan heeft na het herschrijven een ander onderdeel van het programma. Hierdoor kan het voorkomen dat een regressiefout niet gevonden is door de tester omdat deze fout zich niet voordeed in het onderdeel dat veranderd werd.
REST	Rest is een manier om webservices te creeëren.
SonarQube	Een open-source platform voor continue inspectie van de code.
Subcollection	Een subcollection is een collection die onderdeel is van een andere collection.
Wijs	Wijs is een Belgisch bedrijf dat de frontend bouwt voor YuMe.

Afkorting	**Betekenis**
API	Application Programming Interface
AWS	Amazon Web Services
BDD	Behaviour-Driven development
CD	Continuous Delivery
CI	Continuous Integration
CLI	command-line-interface
CSV	comma-separeted values
GUI	Graphical User Interface
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
NPM	Node Package Manager
QA	Quality Assurance
REST	Representation state stransfer
SOAP	Simple Object Access Protocol
UAT	User Acceptance Testing
XML	Extensible Markup Language

Inleiding

Testuitvoering is in veel projecten nog steeds een tijdrovende handmatige bezigheid. Door testautomatisering kan er veel nauwkeuriger en sneller getest worden. Dit zorgt voor zowel een kortere doorlooptijd als een betere kwaliteit.

Handmatig testen kost veel tijd waardoor het bij grotere projecten onmogelijk is om alles te testen. Als er enkel manueel getest wordt is de kans groot dat er regressiefouten over het hoofd gezien worden door de testers. De kans dat regressiefouten in productie komen, wil Argeüs verkleinen door automatische testen toe te voegen aan het testproces.

In dit eindwerk is te lezen hoe Argeüs de stap naar automatisering binnen het testproces wil zetten.

Door te kijken naar de werking van Argeüs in het verleden wordt bekeken hoe het testproces verbeterd kan worden in de toekomst.

Daarnaast is nog een onderzoek gedaan naar welke API testtool mogelijks de beste tool is voor Argeüs. Doordat elke API anders is, heeft elke API ook een andere testtool die het beste is voor die specifieke API. Met dit onderzoek wil Argeüs weten als de keuze voor Postman de beste keuze was of als er nog een betere tool is voor de API's van Argeüs.

I. Stageverslag

1 Bedrijfsvoorstelling

1.1 Profiel

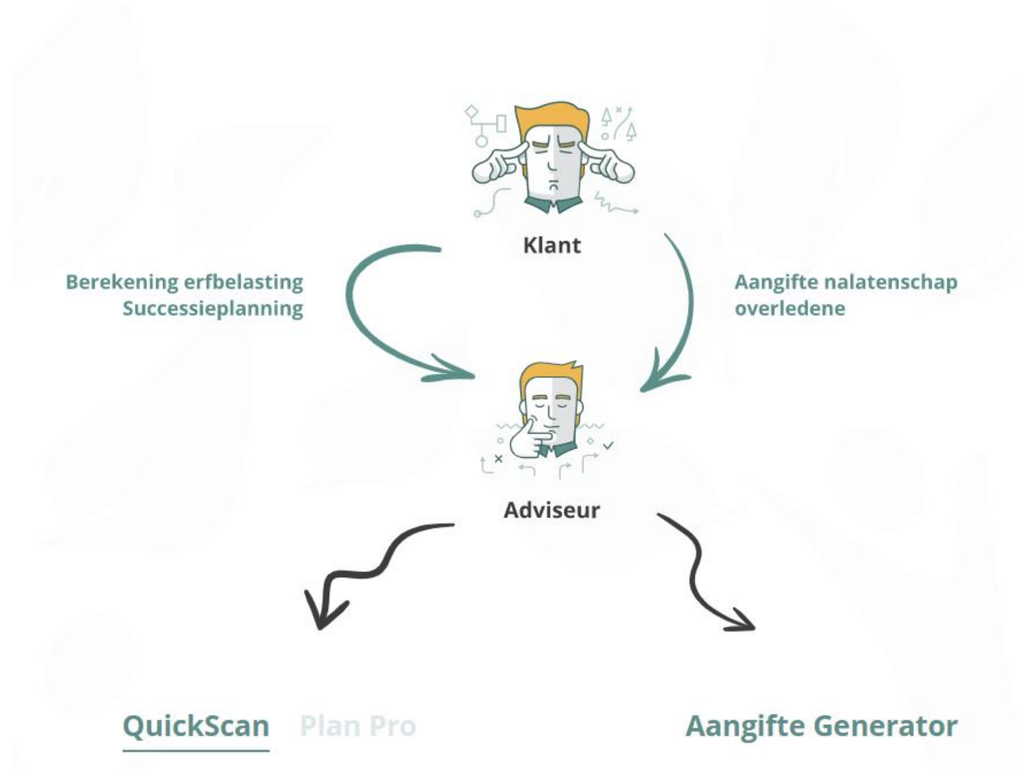
Argeüs werd opgericht in 2011 als een financial planning consultingbureau en heeft doorheen de jaren haar activiteit uitgebreid naar het domein van softwareontwikkeling voor adviseurs over successierechten, financial planning en advies.

De klanten van Argeüs zijn banken, notarissen, financiële planners, vermogensbeheerders, accountants, juristen en verzekeringsmaatschappijen.

De gedrevenheid van de onderneming in combinatie met dit "kritisch oog" en commerciële bruikbaarheid in de praktijk zorgt voor unieke oplossingen die eenvoud, gebruiksgemak, flexibiliteit, efficiëntie en accuraatheid nastreven, kortom de argeüsdefinitie van kwaliteit.

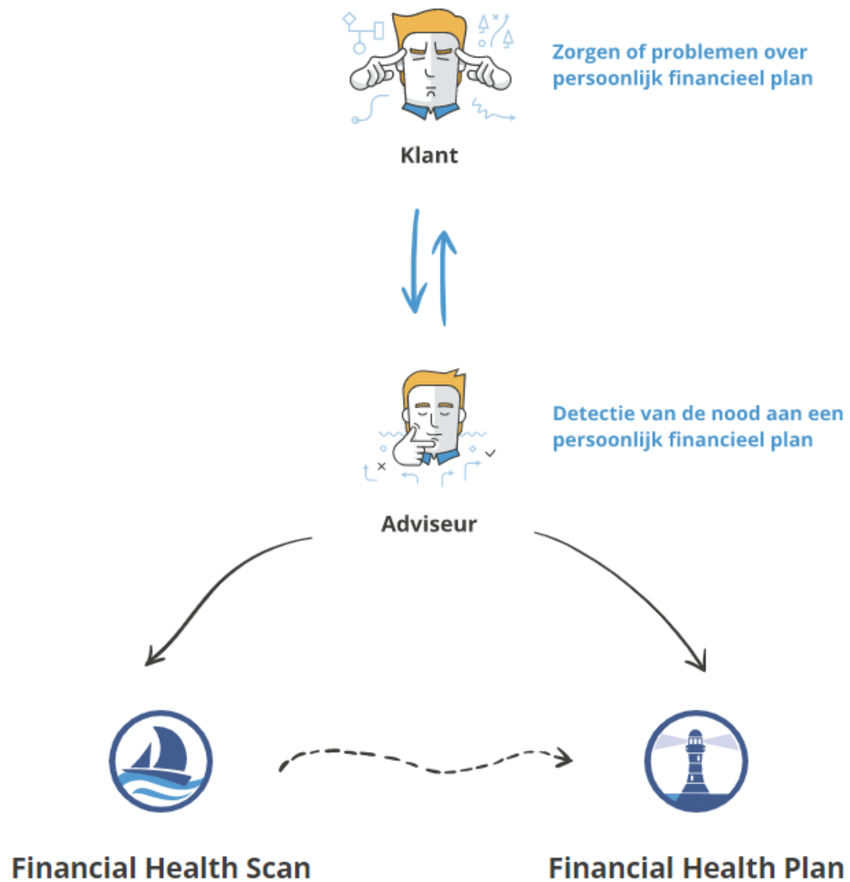
Argeüs brengt op dit moment twee softwarepakketten zelf op de markt. De successiecalculator en de financiële planner.

De successiecalculator zorgt ervoor dat eenvoudig maar toch tot in detail de successierechten en erfbelasting gegenereerd wordt, dat er aan successieplanning gedaan kan worden en dat de aangifte voor de nalatenschap berekend en gerapporteerd wordt naar de normen opgelegd door de overheid.



Figuur 1: Successiecalculator

Daarnaast is de Financial Planning Software. Dankzij de Financial Planning Software met zijn zeer uitgebreide rekenmotor en integratie met de Successiecalculator kan er een zeer diepgaande en onderbouwde analyse gemaakt worden over de persoonlijke financiële status van de klant.



Figuur 2: Financial Planning Software

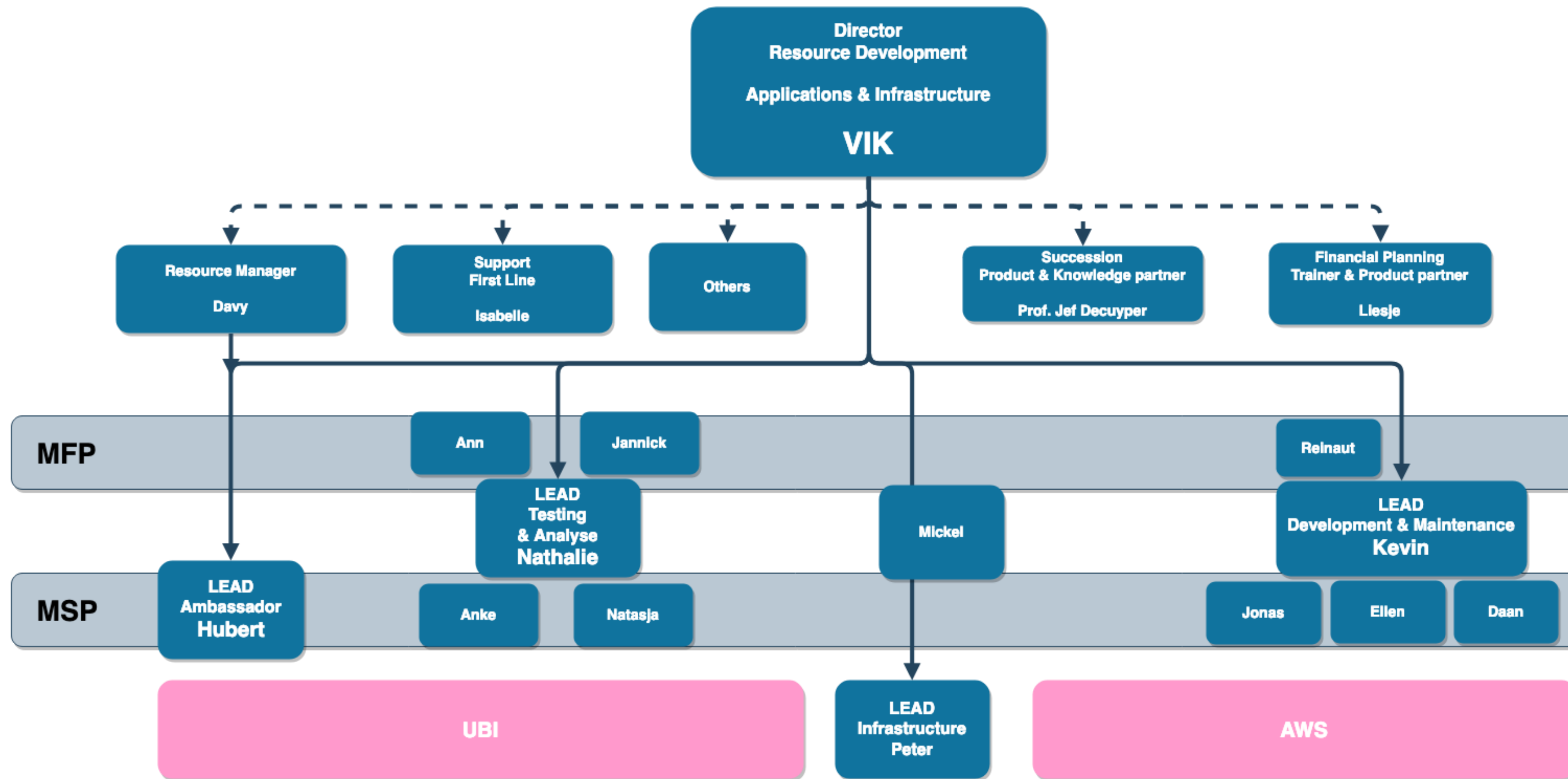
1.2 Locatie

Argeüs is gelegen in Gestel. Gestel is een gehucht van de gemeente Lummen. In juni 2016 heeft Argeüs in Lummen een kerk gekocht. Deze gaan ze verbouwen tot kantoor om er in januari 2020 in te trekken. Het kantoor is strategisch goed gelegen doordat het vlakbij het klaverblad van Lummen waar de E313 en de E314 elkaar kruisen. Hierdoor is het goed bereikbaar voor klant en personeel.

Figuur 3: Ligging Argeüs

Figuur 4: Google streetview Argeüs

1.3 Organigram



Status document: First draft

Opmaak document: Vik & Dominique

Datum opmaak: 25 februari 2019

Figuur 5: Organigram Argeüs

2 Voorstelling opdracht

2.1 Probleemstelling

2.1.1 Situering van het probleem

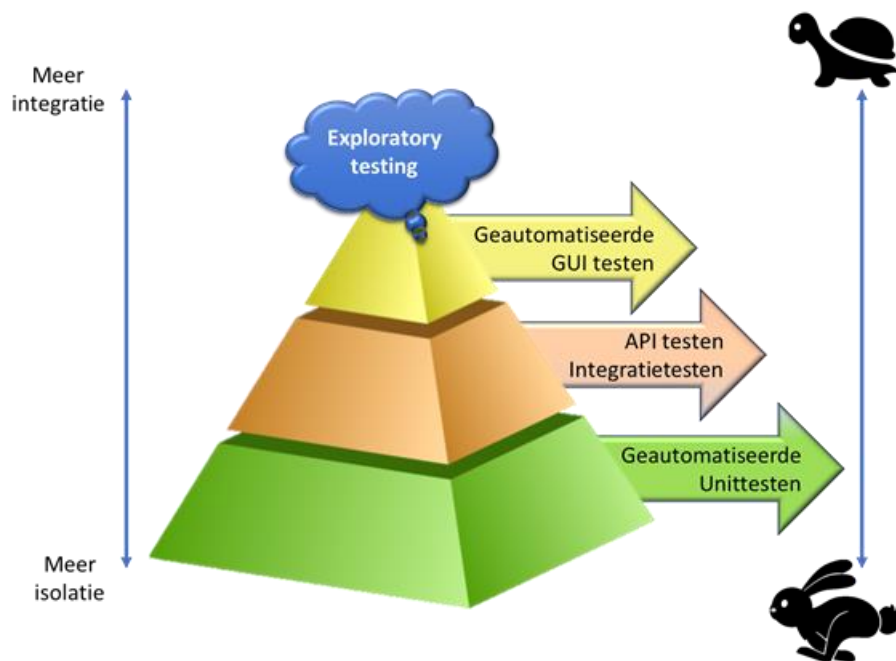
Binnen Argeüs wordt er enkel gebruikgemaakt van manuele testen en dus geen automatische testen. Hierdoor is de dekking van de testen die gedaan worden relatief klein, aangezien het onmogelijk is om alles manueel te testen op ieder moment dat er een update gedaan wordt. Dit heeft als gevolg dat het wel eens gebeurt dat er een bug langs het vangnet van de testers doorglipt en in productie komt.

Dit probleem wil Argeüs oplossen door het testproces te verbeteren aan de hand van de testpiramide.

2.1.2 De testpiramide

De testpiramide, te zien in figuur 6, is een schematische voorstelling van een agile en geautomatiseerde aanpak om bugs te voorkomen en vroegtijdig op te sporen.

De testpiramide bestaat uit vier onderdelen. Deze onderdelen zijn: Unit testen, API- en integratietesten, geautomatiseerde GUI testen en exploratieve testen.



Figuur 6 de testpiramide

2.1.2.1 Unit testen

De unittesten vormen de basis van de testpiramide en nemen dus het grootste deel in beslag. Een unittest neemt een kleine hoeveelheid onafhankelijke code op zich. Er mogen dus geen afhankelijkheden getest worden met unittesten. Dit heeft als reden dat indien er een test zou falen hierbij snel de foutieve code gevonden kan worden.

Het nadeel is echter dat unittesten niet alle fouten kunnen vinden. Omdat veel fouten vaak combinaties van verschillen onderdelen (units) zijn. Deze fouten zijn dan ook integratie- en UI testen. Deze blijven dus zeer belangrijk bij het testen van het volledige product.

De Unittesten hebben als doel om een zo groot mogelijke dekking te krijgen van de code. Om dit te controleren maakt Argeüs gebruik van SonarQube. SonarQube kan vertellen hoe groot de dekking is van de unittesten.

De unittesten worden geschreven door de het developmentteam.

2.1.2.2 API- en integratietesten

Bij de API- en integratietesten wordt zowel de connectiviteit tussen verschillende units alsook de functionaliteit getest. De API- en integratietesten maken geen gebruik van de frontend. Dit heeft als voordeel dat API- en integratietesten sneller en minder onderhoudsgevoelig zijn dan UI testen. De communicatie vindt plaats via de beschikbare API's.

Bij deze testen wordt de backend rechtstreeks aangesproken om te zien of datgene wat de backend teruggeeft wel juist is.

De API- en integratietesten worden geschreven door het *Quality Assurance* (*QA*) team.

2.1.2.3 Geautomatiseerde GUI testen

Als punt van de piramide zijn er de geautomatiseerde *Graphical User Interface* (*GUI*) testen. Deze GUI testen zijn vooral gericht op het *end-to-end* (E2E) testen van een applicatie. Geautomatiseerde GUI testen zijn vaak langzaam. Deze testen worden vooral gebruikt om te zien of de volledige flow van de GUI in orde is en alles in deze flow werkt. Daarnaast worden deze testen ook gebruikt om te controleren of de informatie die doorgestuurd wordt door de backend ook juist weergegeven wordt in de frontend.

De geautomatiseerde GUI testen worden ook uitgevoerd door het QA-team.

2.1.2.4 Exploratieve testen

Boven de piramide hangt nog een zeer belangrijk wolkje. Dit wolkje zijn de exploratieve testen. Dit zijn handmatige testen die meestal gedaan worden om aan te tonen dat de software goed werkt en om de processen en gebruiksvriendelijkheid te kunnen beoordelen.

Exploratief testen is gestructureerd. Het is gepland met testideeën. Voordat een test uitgevoerd wordt moet de tester goed nadenken over het verwachte resultaat. Deze verwachte resultaten kunnen zeer concreet over de juiste waarde gaan maar ook in het groter geheel over de juiste werkwijze.

Ook deze testen worden gedaan door het QA-team.

2.1.2.5 Argeüs binnen testing piramide

Op dit moment maakt Argeüs enkel gebruik van het wolkje boven de piramide en van een gedeelte van de basis, nl. de exploratieve testen en de unittesten. Deze stageopdracht zal automatische API- en integratietesten toevoegen aan het testproces van Argeüs. Geautomatiseerde GUI-testen vallen buiten deze stage aangezien de GUI van dit project ontwikkeld wordt door een externe partij en dus niet binnen de scope van Argeüs valt. De scope van Argeüs gaat enkel over de API van de Lifeplanner.

2.2 Doelstelling

Via deze stage wil Argeüs een verbeterd testproces met automatische testen. Dit zal voor Argeüs zowel intern als naar klanten toe een mooie verbetering zijn. Intern kunnen de fouten sneller gevonden en opgelost worden en de klanten zullen minder bugs en fouten vinden in de software wat dan ook een professionelere indruk geeft.

2.3 Technologieën

Deze opdracht zal uitgevoerd worden vanuit het Argeüs kantoor in Lummen. Hier is al het nodige materiaal beschikbaar om deze opdracht uit te voeren.

Voor testtool wordt gebruikgemaakt van de gratis versie van Postman. Dit is een tool die al binnen Argeüs gebruikt wordt om de API manueel te testen. Verder zal voor de automatisering gebruikgemaakt worden van de combinatie Postman, Newman en Jenkins en een git *repository* op Bitbucket om een volledig automatisch proces te creëren. Dit automatisch proces zal dan rapporteren naar Testrail. Testrail is de testmanagementtool die Argeüs gebruikt.

Er wordt ook een onderzoek gedaan naar andere API-testtools om te controleren of er geen betere tool is voor Argeüs (zie hoofdstuk II, Onderzoekstopic). Als uit dit onderzoek een betere tool komt, zal deze tool in de toekomst ook gebruikt worden voor Argeüs.

Voor documentatie en projectmanagement maakt Argeüs gebruik van Confluence en Jira. Deze tools worden gebruikt om de stage te documenteren voor intern gebruik en om timelogs bij te houden tijdens de stage.

3 Uitwerking opdracht

3.1 De Lifeplanner/ YuMe

De Lifeplanner, voor het publiek bekend als YuMe, is een cocreatie tussen Belfius, Argeüs en Wijs.

Figuur 7: YuMe logo

Belfius is de klant die het product aangeeft. Argeüs zorgt voor de backend, een rekenmotor die het financieel plan opstelt, en Wijs zorgt voor de frontend.

YuMe geeft zijn klanten een heldere blik op nu en later.

> "YuMe kijkt verder dan uw pensioen, uw spaarplan, uw beleggingen en uw inkomen. Ook uw gezin, uw huis en uw passies krijgen een belangrijke plaats. Dankzij een interactief overzicht krijgt u bij alle belangrijke momenten in uw leven uitleg en voorstellen van oplossingen. Zo kan u steeds de juiste beslissingen nemen." [1]

Het doel van YuMe is om de klanten van Belfius een overzicht te geven over hun financiële toekomst. Hiermee kan Belfius door middel van simulaties een overzicht geven over hoe deze simulaties hun financiële toekomst veranderen. Zo kan Belfius zijn producten aanbieden en laten zien welke impact deze producten kunnen geven. Dit gaat bijvoorbeeld over producten als beleggingen en pensioensparen.

YuMe geeft de klanten de mogelijkheid om een grote hoeveelheid van "events" en "life events" in te geven en te bekijken wat de impact van deze events zijn op de financiële staat van de klant. Dit kan gaan van een sabbatical nemen tot kinderen die op kot gaan.

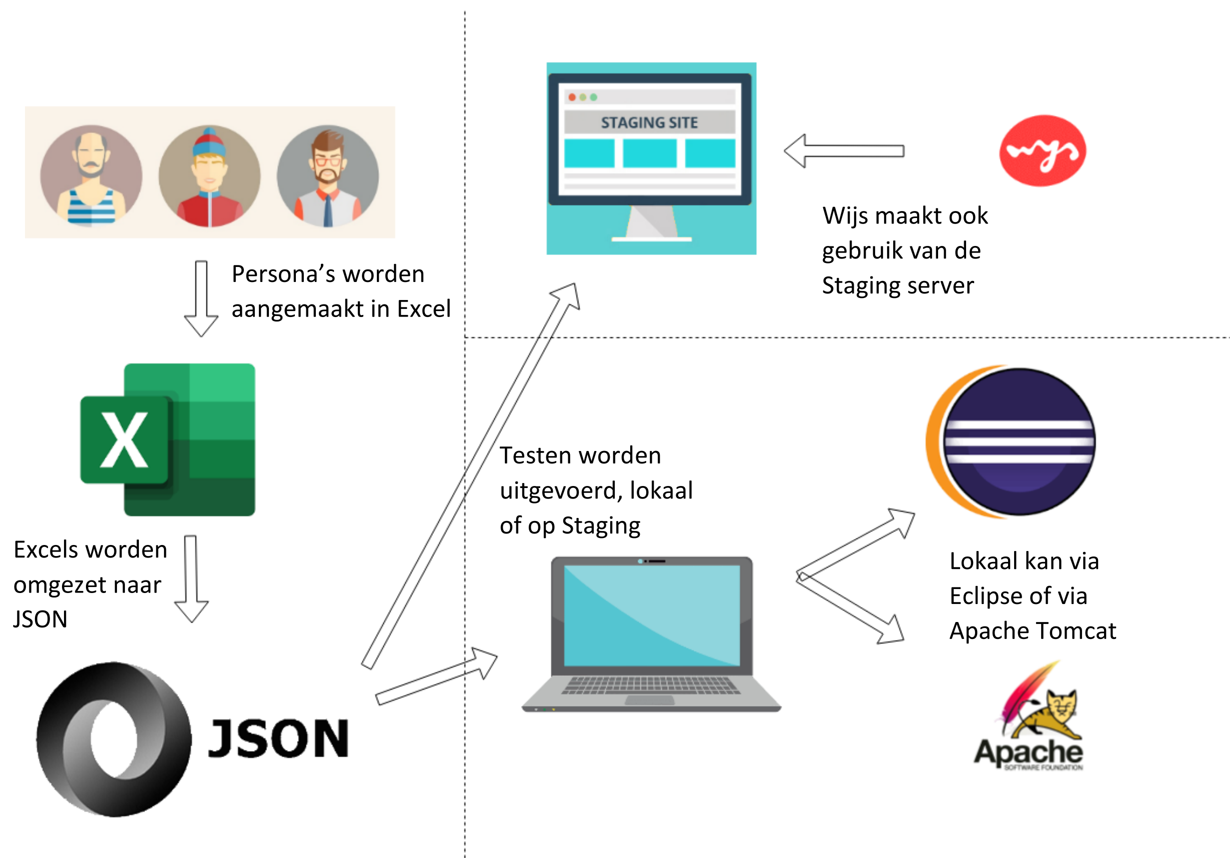
3.2 Verleden vs toekomst

3.2.1 Verleden

In het verleden werd er enkel manueel getest. Om deze manuele testen te versnellen maakt Argeüs gebruik van Excels om persona's en testcases op te bouwen. Deze persona's kunnen dan door middel van een programma geschreven in Java omgezet worden naar JSON-bestanden die geïmporteerd kunnen worden in de *Graphical CashFlowTable* (GCFT). Deze GCFT is een test GUI die intern gebruikt wordt om een visueel overzicht te maken van alle gegevens die de rekenmotor, API, terugstuurt.

Deze manuele testen gebeurden op twee omgevingen. De eerste omgeving is de Stagingserver. Deze server wordt echter niet enkel gebruikt door Argeüs. Wijs, het bedrijf dat de frontend van de Lifeplanner bouwt voor Belfius, gebruikt voor hun testserver onze Stagingserver als backend. Deze testserver is de User Acceptance Testing (UAT) server waar Belfius ook op test. Hierdoor wordt deze server niet bij elke update die intern gemaakt wordt, geüpdatet. Dit zorgt ervoor dat de testen lokaal uitgevoerd worden. Dit gaat op twee manieren. De eerste manier is om de code te pullen van de Bitbucket server en deze code dan via Eclipse te runnen. De tweede manier is door de WAR-file die de Jenkins bouwt te downloaden en deze via een Apache Tomcat te draaien.

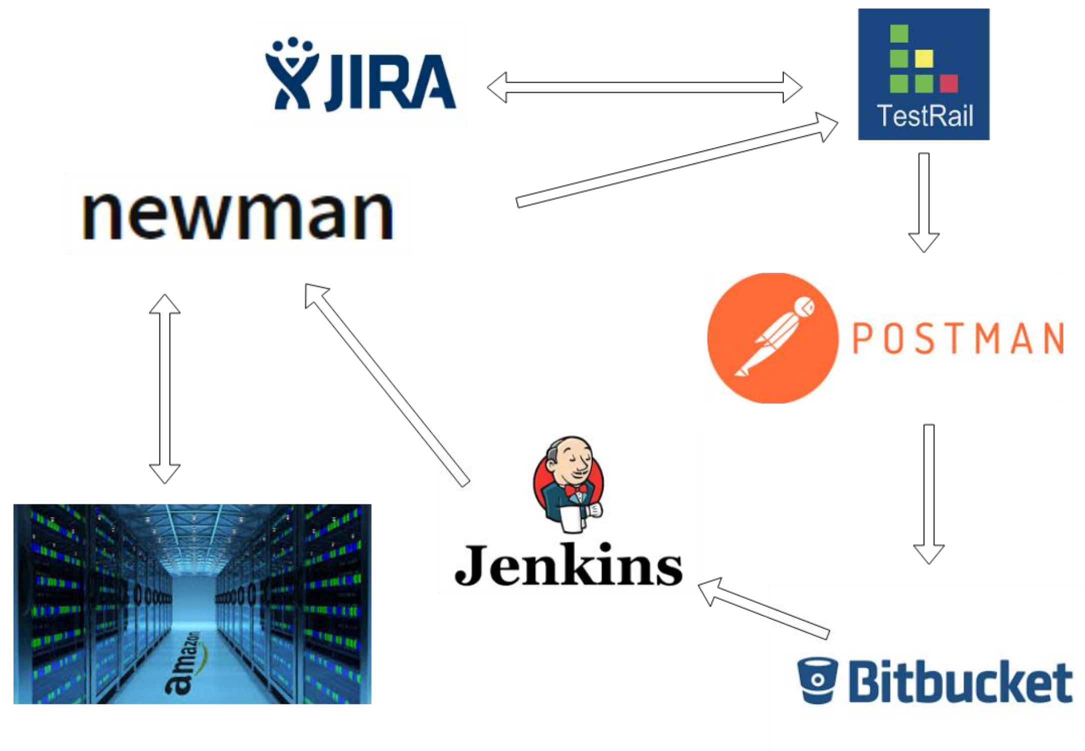
Als er een bugfix of een nieuwe feature geïmplementeerd werd, dan werd deze eerst lokaal getest om te zien of de manuele testen slagen. Als de manuele testen geslaagd zijn, werd na overleg met Wijs, de Stagingserver geüpdatet. Zo kan Wijs altijd verder met een stabiele versie en indien Wijs een bug opmerkt, kan deze getest worden op onze Staging zodat er altijd in dezelfde versie gewerkt wordt. De volledige flow van het maken van persona's tot het testen is te zien in figuur 8.



Figuur 8: Flow verleden

3.2.2 Toekomst

Deze stage heeft als doel de werking naar de toekomst toe te verbeteren. Er gaan nog steeds manuele testen nodig zijn. Zowel om bugs te controleren die gevonden worden door Belfius of Wijs, als interne testen. Bij de uitwerking van de stageopdracht zal er nog een extra server moeten bijkomen. Dit zal dan de testserver worden. Dan heeft Argeüs een eigen testserver die onafhankelijk is van andere partijen, waar constant de laatste nieuwe updates op komen. Dit zal ervoor zorgen dat deze server eigenlijk een acceptatie zal worden naar de Stagingserver toe. Als alle automatische testen slagen op de testserver dan kan na overleg met Wijs de Stagingserver ook de laatste geslaagde versie van de testserver krijgen. Hieronder in figuur 9 is te zien hoe de flow van de automatische testen verloopt.



Figuur 9 Flow automatische testen

De flow begint met de koppeling tussen Jira en Testrail. Elke test die in Testrail staat zal gekoppeld zijn aan een issue, feature of story in Jira. Zo kan er makkelijk teruggekeken worden bij een fout aan welke issue, story of feature dit hangt.

Als volgend is de koppeling tussen Testrail en Postman. Deze koppeling gebeurt doordat in Testrail krijgt iedere test een ID mee. Dit ID moet ook meegegeven worden aan de test in Postman. Zo weet Newman naar welke test in Testrail het resultaat gestuurd moet worden.

Vervolgens worden in Postman alle testen geprogrammeerd, Postman is de Integrated Development Environment (IDE). In Postman is de mogelijkheid om deze testen en de variabelen dan te exporteren naar JSON-bestanden.

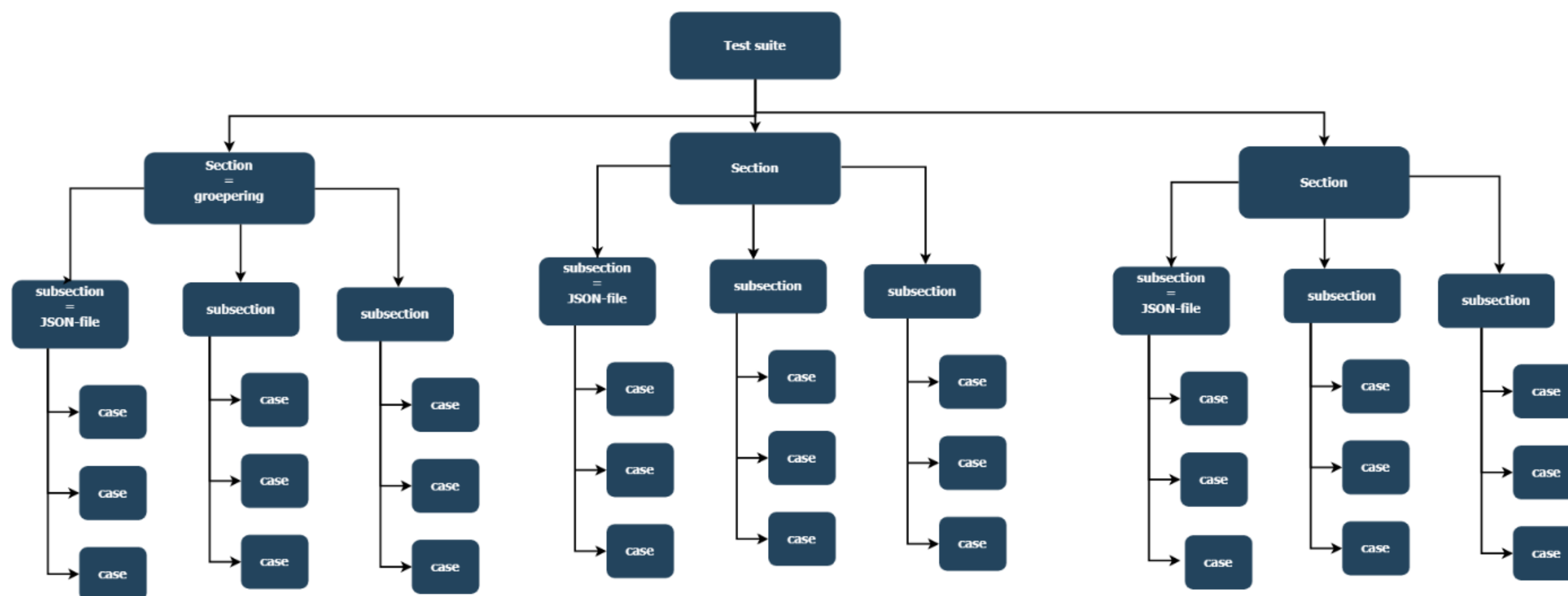
Deze bestanden worden dan *gepushed* naar de BitBucket van Argeüs. Dit geeft enkele voordelen. Als eerste kan Jenkins de testen en variabelen van Bitbucket *pullen* zodat Jenkins deze aan Newman kan meegeven. Een ander voordeel is dat de programmeurs deze testen ook kunnen pullen om lokaal te gebruiken. Zo kunnen ze als ze tijdens de aanpassingen direct zien als ze nergens een regressiefout gemaakt hebben zonder het eerst naar de testserver te moeten sturen.

Zoals al eerder vermeld, maakt Jenkins ook gebruik van de testen op Bitbucket. Jenkins zal zowel bij elke nieuwe *build* van de testserver die op Amazon Web Services (AWS) staat als bij elke update van de testen op Bitbucket alle testen uitvoeren op deze testserver. Hij zal iedere keer alle testen *pullen* van Bitbucket en deze via het Newman command runnen. Ook kunnen er in Postman meerdere *environments* gemaakt worden die elk voor een andere server werken. Zo kan er een *environment* zijn voor de testserver maar ook een voor de Stagingserver en een voor productie.

Als laatste wordt binnen Argeüs samen met Newman nog een plug-in van Newman gebruikt, genaamd Newman-Testrail-Reporter. Deze plug-in zorgt voor de uiteindelijke connectie terug naar Testrail.

3.2.3 Opbouw Testrail

De opbouw van testrail, uitgewerkt in figuur 10, bestaat uit vier niveaus. Het bovenste niveau is de totale collectie, dit komt ook overeen met een collection binnen Postman. Hierin zitten alle testen voor de Lifeplanner. Onder dit niveau zitten de secties. Deze secties zijn in grote lijnen de verschillende groeperingen die gemaakt zijn rond de testen. Dit gaat bijvoorbeeld over de events testen, life events, beleggingen, enzovoort…. Elk van deze secties komt overeen met een map in Postman. Zo kan in Postman ook een onderverdeling maken tussen de verschillende groeperingen. Onder deze secties zitten dan subsecties. Deze subsecties komen overeen met de requests in Postman. Elke subsectie krijgt de body mee van de *request*. Zo kan ook in Testrail direct gezien worden wat opgestuurd wordt bij die *request*. Onder deze subsecties komen dan de testen zelf. Per request in Postman kunnen er verschillende testen geschreven worden. Deze testen gaan dan kijken naar het antwoord van de *request* en zal hierin controleren dat een bepaald onderdeel wel klopt.



Figuur 10 Opbouw Testrail

3.3 Uitwerking van de testen

3.3.1 Voorbereiding

Als voorbereiding is er een lijst opgemaakt met de mogelijke testcases die er getest gaan worden. Zo wordt er eerst opgesomd welke hoofdstukken allemaal gaan komen. Hiervoor zijn *stories* in Jira waar de testen naar gelinkt kunnen worden. Dit gaat over de hoofdstukken Events, Life events, Beleggingen, Kinderkosten, Consumptie, Errors en Regressietesten.

Vanuit deze hoofdstukken werden dan testcases bedacht. Elke testcases, een voorbeeld te zien in figuur 11, is een *request* naar de Lifeplanner waar dan testen op uitgevoerd worden. Een testcase is een persona die een bepaald onderdeel van de Lifeplanner aanhaald.

In het voorbeeld te zien in figuur 11 is een persona die alleenstaand is, geboren in 1990, mannelijk, met een netto inkomen van €2000 en die een pensioeninkomen gaat krijgen van €1500. Dit is een van de meest eenvoudige testcases die gemaakt kan worden voor de lifeplanner.

```
{
    "dossier_number": "E01 - enkel client",
    "partner_1": {
        "year_of_birth": 1990,
        "gender": "MALE",
        "net_income": 2000,
        "pension_income": 1500
    }
}
```

Figuur 11: Voorbeeld testcase

Aan de hand van de documentatie op confluence worden de testcases opgesteld. Dit gebeurt per onderdeel. Zo komt per story, hoofdstuk, een tabel in confluence met alle testcases voor dat hoofdstuk. Hierbij wordt een duidelijk naam gekozen die direct weergeeft wat de bedoeling is van de testcase. De tweede kolom van de tabel zorgt voor extra informatie. Zo kan er meegegeven worden als het bijvoorbeeld een dubbele testcase is en deze dus niet gemaakt gaat worden binnen dit hoofdstuk. Verder kan er ook meegegeven worden als er een probleem is bij de testcase of als er iets speciaals gedaan wordt met de testcase. Als laatste is er nog het JSON-bestand van de testcase. Elke testcase is één request dus één JSON-bestand.

Om het overzichtelijker te maken wordt er gewerkt met kleurcodes binnen Confluence. Een groene test is voor zover mogelijk afgewerkt. Een test die rood is wordt, ofwel nooit ofwel op dit moment, niet gemaakt. Hierbij staat ook uitleg waarom deze test niet gedaan wordt. Dit komt doordat er bijvoorbeeld een verandering is van specificatie rond de gebeurtenissen in de testcase of dat de testcase al eens uitgewerkt is in een ander hoofdstuk. Een test die oranje is moet nog bekeken worden als deze nu hier afgewerkt moet worden of bij een ander hoofdstuk. De opbouw van deze confluence pagina is te zien in figuur 12.

Life events

TC	Wat	Opmerking	input JSONFile
1	client - early retirement		LE01 - client - early retirement.json
2	client + partner - early retirement		LE02 - client + partner - early retirement.json
3	client - early retirement + partner - early retirement		LE03 - client - early retirement + partner - early retirement.json
4	client - work part time		LE04 - client - work part time.json
5	client + partner - work part time		LE05 - client + partner - work part time.json
6	client - work part time + partner work part time		LE06 - client - work part time + partner - work part time.json
7	client - work after retirement		LE07 - client - work after retirement.json
8	client + partner - work after retirement		LE08 - client + partner - work after retirement.json
9	client - work after retirement + partner - work after retirement		LE09 - client - work after retirement + partner - work after retirement.json
10	client - promotion		LE10 - client - promotion.json
11	client + partner promotion		LE11 - client + partner - promotion.json
12	client - promotion + partner - promotion		LE12 - client - promotion + partner - promotion.json
13	client - sabbatical		LE13 - client - sabbatical.json
14	client + partner - sabbatical		LE14 - client + partner - sabbatical.json
15	client - sabbatical + partner - sabbatical		LE15 - client - sabbatical + partner - sabbatical.json
16	client - student housing		LE16 - client - student housing.json
17	client + kind - student housing	Student housing staat niet gelinked aan een kind	

Figuur 12: Opbouw lijst testcases Confluence

3.3.2 Aanpak

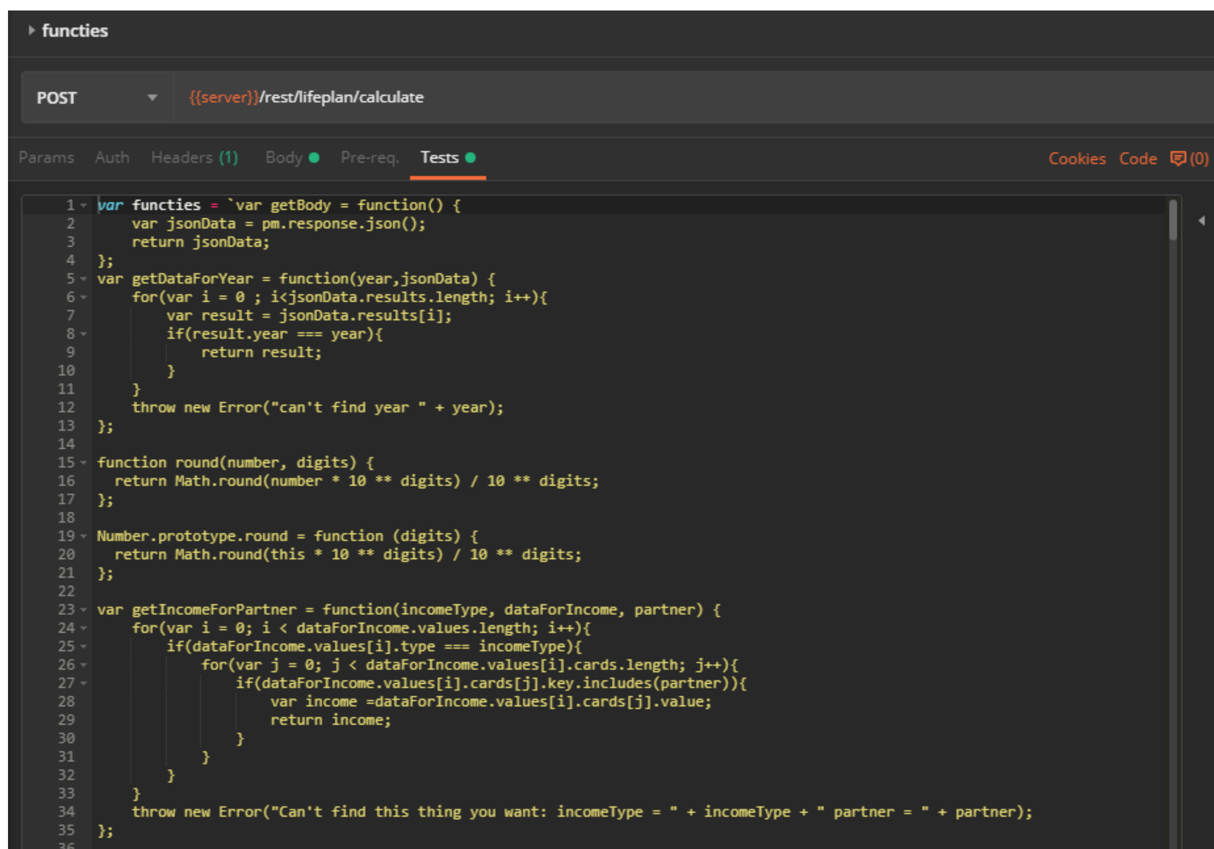
Het schrijven van de testen bestaat uit drie onderdelen. Er wordt begonnen met het opbouwen van de testen vanuit de testcase. Daarna worden deze testen uitschrijven in Postman. Als laatste wordt Testrail bijgewerkt zodat de cirkel afgewerkt kan worden.

3.3.2.1 Testen opbouwen vanuit testcase

Eerst wordt er gekeken wat er verwacht wordt van de testcase. Wat gebeurt er op welk moment? Waar zijn er veranderingen in de cashflow of waar komt het event voor? Dit wordt meestal gedaan door de test door de GCFT-tool te draaien en te kijken naar de tabel waar dat veranderingen komen. Ook wordt er gekeken naar wat moet het doen. Er wordt gekeken op Confluence van wat zijn de specificaties van dit onderdeel. Er moet goed gekeken worden als hier geen afwijkingen zijn met de documentatie en datgeen dat effectief gebeurt in de motor. Als hier afwijkingen zijn dan moet er gekeken worden naar wat het juist moet zijn. Is de documentatie juist of is de code juist? Hiervoor wordt afgestemd met de programmeurs en de klant, in ons geval, Belfius. Als de documentatie fout is dan wordt deze ook aangepast.

3.3.2.2 Uitschrijven testen

Als we weten welke onderdelen veranderen door de input dan kunnen we hierop testen. Deze testen worden dan geschreven in Postman. Omdat we flexibele testen willen, zullen alle testen geschreven worden aan de hand van berekeningen en niet aan de hand van vaste verwachte resultaten. Zo kunnen de berekeningen, deze worden gemaakt door functies te schrijven in Postman in Javascript, hergebruikt worden voor andere testen. Ook kunnen, als er een wijziging is in de berekening, alle testen snel aangepast worden door de functie aan te passen. Als we gebruik maken van vaste waarde moeten alle waarde van de testen die te maken hebben met de nieuwe berekening nog steeds manueel geteld worden en dan aangepast worden. Dit zou ervoor zorgen dat er in de toekomst veel meer tijd moet vrijgemaakt worden voor het onderhouden van de testen. Echter heeft Postman een nadeel, binnen Postman bestaan geen globale functies. Voor dit probleem is dan een *workaround* geschreven. Door een testcase aan te maken speciaal voor alle functies in postman en zo alle functies in een variabele te zetten en die dan door te sturen, kunnen in elke testcase alle functies opgeroepen worden met de functie “eval” van Javascript. In figuur 13 is te zien hoe deze workaround gemaakt is in Postman. De volledige variabele “functies”, waar alle functies in gezet worden als string, wordt dan opgeslagen in de environment variabelen. Dit is te zien in figuur 14.



Figuur 13: Functies binnen Postman

Testing_LP

Edit

VARIABLE	INITIAL VALUE	CURRENT VALUE
server	http://localhost:8082	http://localhost:8082
functies	var getBody = function() { var jsonData = pm.response.json(); return jsonData; }; var getDataForYear = function(year,jsonData) { ...	var getBody = function() { var jsonData = pm.response.json(); return jsonData; }; var getDataForYear = function(year,jsonData) { ...
birthyearMale	1990	1990

Figuur 14: Variabele van functies

3.3.2.3 Testrail

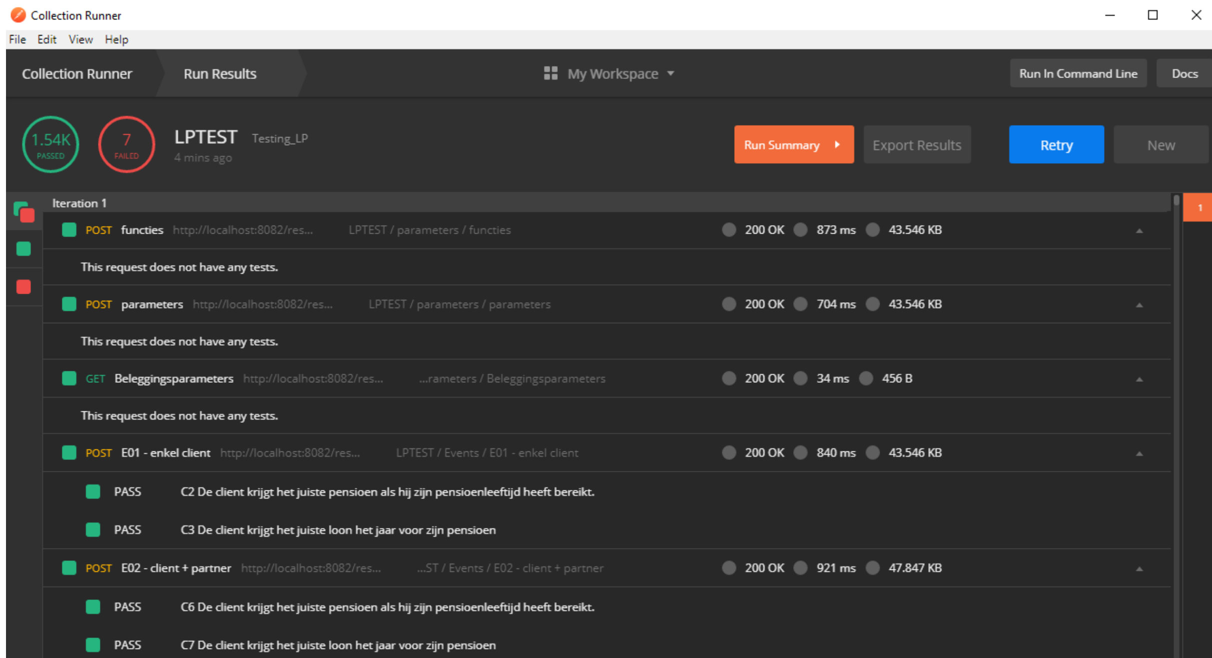
Terwijl de testen geschreven worden in Postman zal er in Testrail ook de test aangemaakt worden volgens het principe uitgelegd in hoofdstuk 3.2.3. Als de test toegevoegd is aan testrail zal deze test een ID krijgen. Dit ID wordt dan toegevoegd aan het begin van de naam van de test in Postman. Zo kan Newman-Testrail-Reporter het resultaat terugsturen naar Testrail.

3.3.3 Runnen van de testen

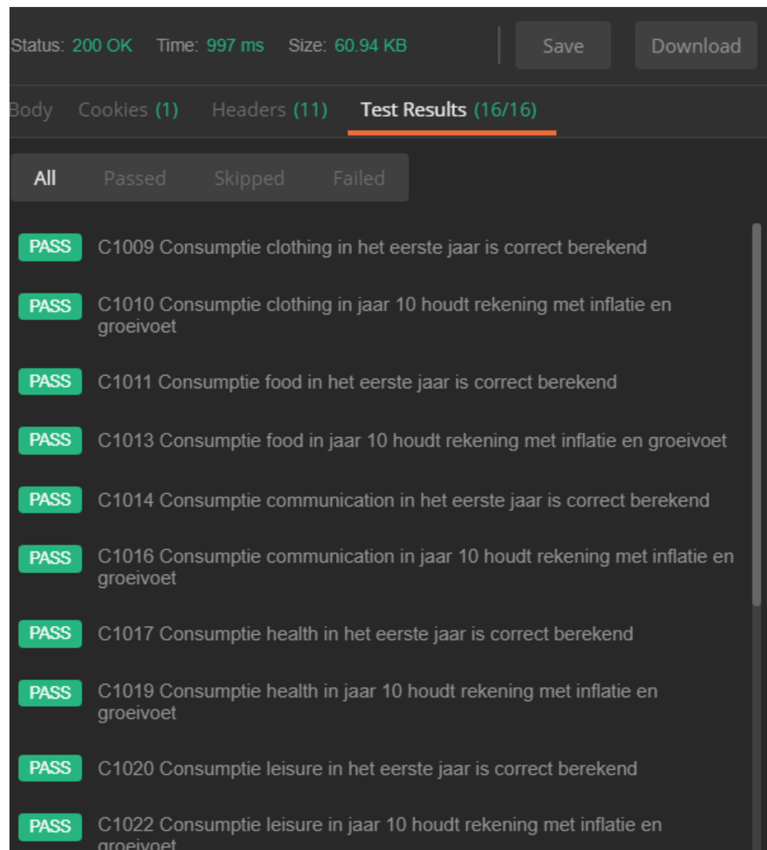
De testen kunnen op verschillende manieren gerund worden. Als eerste kunnen ze in Postman gerund worden. Daarnaast kunnen ze manueel met Newman gerund worden via command line en als laatste is er nog Jenkins die het volledig automatisch laat runnen met Newman.

3.3.3.1 Postman

Binnen postman kan je elke test apart runnen, dit is handig voor tijdens het schrijven van de testen. Daarnaast kunnen deze testen ook nog gerund worden per *collection* of per *subcollection* in de Runner van postman. In figuur 15 is het resultaat te zien van de volledige collection en in figuur 16 is het resultaat te zien van een enkele test die ook gerund is vanuit Postman.



Figuur 15: Resultaat runner Postman

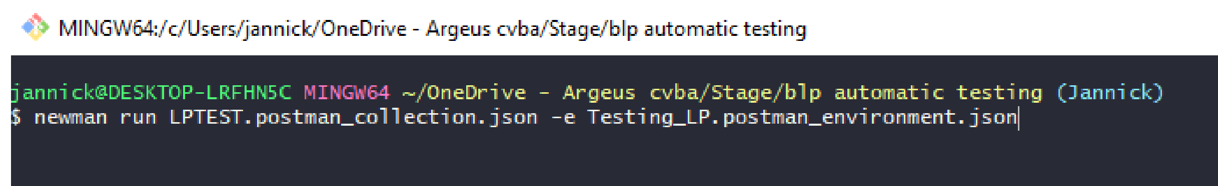


Figuur 16: Resultaat enkel request Postman

Zoals te zien is, in zowel figuur 15 als 16, worden er nog enkele waardes mee gegeven met het resultaat. Zo krijg je bij elke request de status, de tijd dat het request geduurd heeft en de grootte van het request mee. Dit is handig om te zien hoe snel of traag de server reageert.

3.3.3.2 Newman

Newman is een commandline runner voor Postman.



Figuur 17: Commando voor Newman

Newman heeft twee onderdelen nodig voor de testen te runnen. Als eerste komt de collectie. In het geval van figuur 17 is dit dus “LPTEST.postman_collection.json”. Dit is de geëxporteerde versie van de testen die op Bitbucket staan. Daarnaast heeft Newman de variabelen nodig. Alle variabelen worden opgeslagen in een environment. Hier staan dus ook al de functies in die gebruikt worden in de testen. In figuur 17 is dit “Testing_LP.postman_environment.json”. Deze environment is ook geëxporteerd uit postman en geüpload naar Bitbucket. Dit is dus het commando wat de programmeurs gebruiken als ze lokaal al de testen willen uitvoeren.

Verder is in figuur 18 te zien hoe de rapportering van Newman in command-line-interface (CLI) eruitziet. Hier krijgen we dan te zien hoe lang de totale *test run* geduurd heeft. Ook zien we hoe lang elke *request* gemiddeld duurt en hoeveel data er ongeveer gestuurd is. In onderstaand rapport zijn 220 *requests* gestuurd. Deze 220 *request* bevatten 1569 testen waarvan er 13 testen gefaald zijn. Deze *requests* hadden een gemiddelde tijd van 450 ms nodig om een antwoord te krijgen van de Lifeplanner.

	executed	failed
iterations	1	0
requests	220	0
test-scripts	617	0
prerequest-scripts	398	0
assertions	1569	13
total run duration: 3m 13.5s		
total data received: 11.05MB (approx)		
average response time: 450ms		

Figuur 18: Newman rapportering CLI

3.3.3.3 Testrail

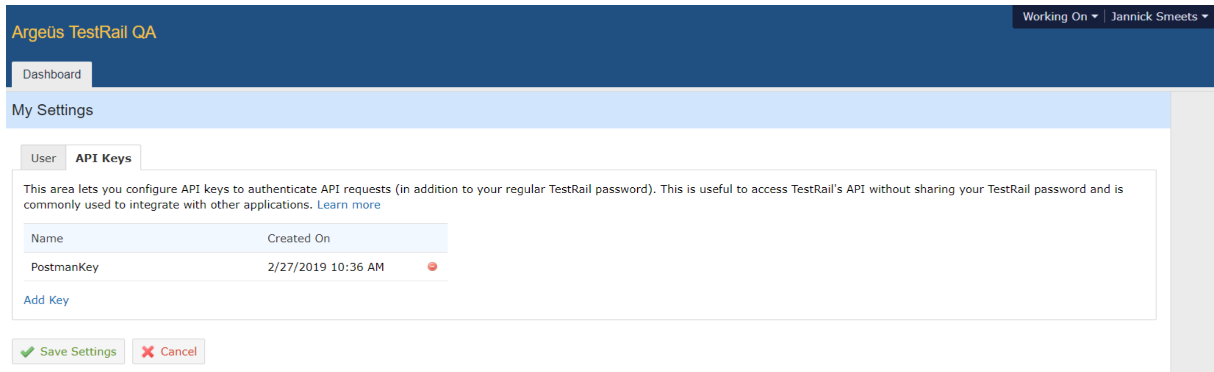
Voor de rapportering naar Testrail wordt ook gebruik gemaakt van Newman. Newman heeft een plug-in genaamd Newman-Reporter-Testrail. Deze plug-in maakt connectie met Testrail en gebruikt de ID die meegegeven wordt aan de testen in Postman in het begin van de naam aan de juiste test in Testrail. Deze plug-in kan geïnstalleerd worden via de Node Package Manager (NPM). Deze plug-in geeft iets extra aan het Newman commando dat te zien is in figuur 17.

In figuur 19 is het Newman-Testrail-Reporter commando te zien.

```
jannick@DESKTOP-LRFHN5C MINGW64 ~/OneDrive - Argeus cvba/Stage/blp automatic testing (Jannick)
$ TESTRAIL_DOMAIN="testrail.argeus.be" TESTRAIL_USERNAME="                    " TESTRAIL_APIKEY='
                           " TESTRAIL_PROJECTID="2" TESTRAIL_SUITEID="3" TESTRAIL_TITLE="tryrun18" newman
 run LPTEST.postman_collection.json -e Testing_LP.postman_environment.json -r testrail,cli
```

Figuur 19:Newman-Testrail-Reporter commando

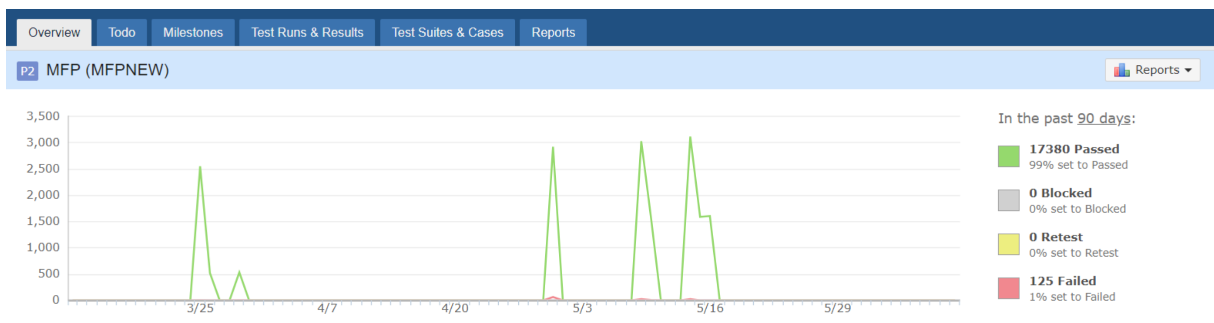
Zo moet er meegegeven worden in welk domein testrail staat. Met welke gebruiker Newman-Testrail-Reporter kan aanmelden. Ook de *APIKey* moet meegeven worden. Dit is een alternatief voor het wachtwoord van de gebruiker. Deze *APIKey* kan aangemaakt worden in Testrail, zie figuur 20. De sleutel die gebruikt wordt voor onze testen is aangemaakt op 27 februari 2019.



Figuur 20: APIKey Testrail

Met deze sleutel kan je aanmelden in Testrail zonder het paswoord te weten. Maar deze sleutel werkt enkel om de API van Testrail te benaderen. Inloggen in de GUI van Testrail is niet mogelijk met deze sleutel, hier is het wachtwoord voor nodig.

Verder heeft Newman-Testrail-Reporter ook nog het project id nodig. Dit is te vinden in het overzicht van een project. Het project MFP (MFPNEW) heeft als project id 2. Dit is te zien op Figuur 21, Verder staat er in dit overzicht hoeveel testen er al gerund zijn afgelopen 90 dagen. Zo is er te zien dat er 17380 testen geslaagd zijn en 125 testen gefaald. Dit wil zeggen dat maar 1% van alle testen gefaald zijn. Dit zijn niet allemaal verschillende testen. Dit kunnen ook dezelfde testen zijn die meerder keren gerund zijn.

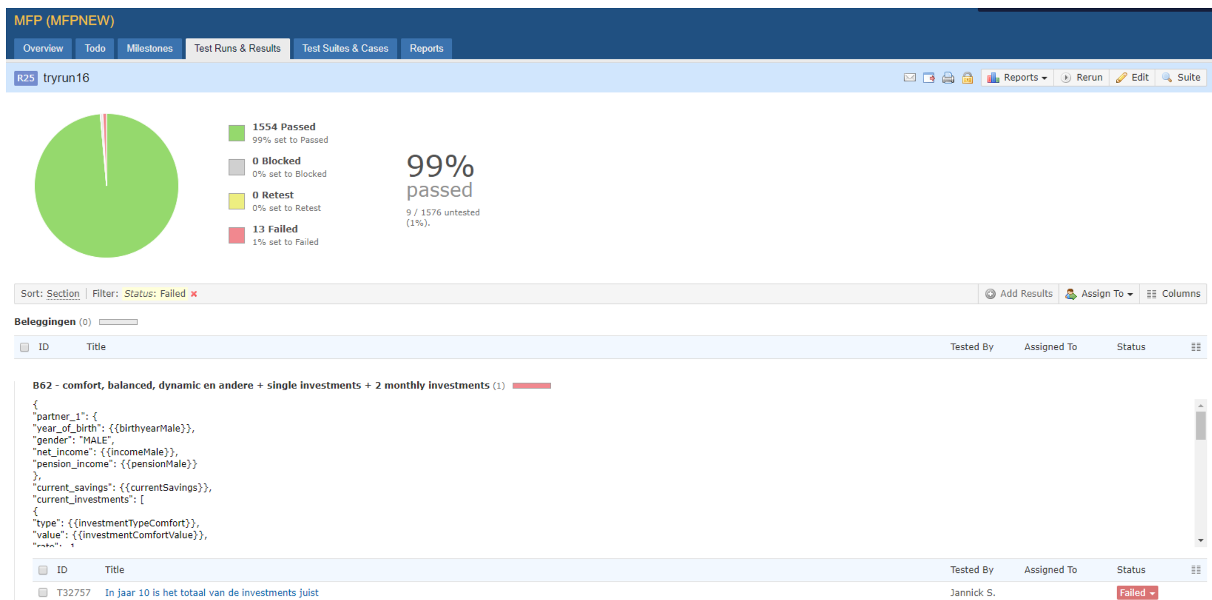


Figuur 21: Testrail project overzicht

Naast het project id heeft Testrail ook nog suite id nodig. In een project kunnen meerdere test suites zitten en daardoor moet Newman-Testrail-Reporter ook dit id hebben.

Als laatste moet de *test run* die gemaakt gaat worden ook nog een naam krijgen. Als al deze onderdelen meegegeven zijn aan het Newman commando en alle testcases die in de collection file zitten een het juiste id hebben meegekregen van de overeengekomen test in Testrail dan zal Newman-Testrail-Reporter een nieuwe *test run* maken waar alle resultaten in komen.

In figuur 22 is te zien hoe de rapportering te zien is in Testrail.



Figuur 22: Rapportering test run in Testrail

3.3.4 Resultaat

3.3.4.1 De automatische testen

Aan het einde van de stage zijn er in totaal 225 testcases uitgewerkt met daarin meer dan 1600 testen. Deze 1600 testen worden in ongeveer vier minuten automatisch uitgevoerd. Moest dit handmatig getest worden dan zou dit bijna twee weken in beslag nemen. In die twee weken manueel testen wordt ook niet zo grondig gegaan als met deze automatische testen. Afrondingsfouten en relatief kleine afwijkingen kunnen bij manueel testen gemakkelijk door de vingers gezien worden. Daarentegen worden bij de automatische testen alle getallen exact berekend en de kleine marges worden al als fout gezien.

3.3.4.2 Intern gebruik

De automatische testen hebben intern al onmiddellijk resultaat geboekt. Door de automatische testen is ontdekt dat er bijvoorbeeld een afrondingsfout gemaakt is binnen de Lifeplanner. Dit is een lichte afwijking die niet gevonden zou worden met manuele testen.

Daarnaast kan, doordat al de testen op Bitbucket staan, het development team deze al binnenhalen en gebruiken. Zo gebruiken de *developers* de automatische testen tijdens het coderen zodat ze onmiddellijk feedback krijgen over de onderdelen waar al automatische testen voor geschreven zijn. Telkens wanneer er een update gedaan wordt van de testen worden de *developers* ook op de hoogte gebracht, mondeling of via mail zodat ze deze al kunnen binnenhalen.

3.3.4.3 Verbeterpunten naar de toekomst

Er zijn nog enkele punten die kunnen verbeteren naar de toekomst. Als eerste is er de automatisatie. Zoals beschreven in figuur 9 in hoofdstuk 3.2.2 zouden alle testen uitgevoerd moeten worden bij elke update van de testen en build van de testserver. Op dit moment is de testserver er nog niet en hierdoor wordt nog niet het volledig potentieel van de automatische testen gebruikt.

Verder is er het verbeterpunt naar Jira. Doordat de requirements van de Lifeplanner niet duidelijk zijn uitgewerkt, is er geen mooie connectie tussen Testrail en Jira. Als deze requirements wel uitgewerkt zouden zijn dan zou er veel sneller en gemakkelijker teruggekeken worden. Zo zou bij een gefaalde test gekeken kunnen worden in de requirements wat het zou moeten doen.

4 Reflectie Stageopdracht

Tijdens mijn stage heb ik ontzettend veel ervaring kunnen opdoen. Deze ervaring is zowel op technisch vlak als op sociaal vlak.

Door alle vergaderingen, sprintretrospectieve en sprintmeetings, de algemene zaken die bij werken in de IT horen, heb ik wel geleerd hoe het eraan toe gaat buiten de schoolbanken. Ook het leren om vragen te stellen. In het begin probeerde ik veel alleen te doen en als ik vast zat dan kon dit wel een tijdje duren vooraleer ik een oplossing gevonden had. Door de tijd ben ik meer en meer vragen gaan stellen aan mijn collega's en zo werden problemen waar ik zelf uren mee bezig was vaak opgelost in enkele minuten. Door deze vragen heb ik ook zeer veel bijgeleerd op technisch vlak. De ervaringen van mijn collega's zijn pas zijn waarde beginnen hebben voor mij vanaf het moment dat ik hun vroeg om hun ervaring met mij te delen. Dit was ook te zien aan mijn testen. De testen die ik in het begin schreef waren klein, simpel en moeilijk aanpasbaar. Deze heb ik dan ook later in mijn stage herschreven naar duidelijkere en beter onderhoudbare testen en dit allemaal door de hulp van mijn collega's.

Tijdens mijn stage heb ik geleerd hoe ik de basis die we op school geleerd hebben kan gebruiken en hoe ik met deze basis mezelf verder kan verdiepen in de onderdelen die ik nodig had voor mijn stage te kunnen uitwerken. Zonder de basis die we op school krijgen was dit nooit gelukt. Maar met enkel die basis was ik er ook nooit gekomen.

Het uitwerken van mijn stage is met vallen en opstaan gebeurt. De testen die ik in het begin geschreven heb werkte, maar zijn toch herschreven omdat ze niet waren gelijk ze moesten zijn. Ze waren niet onderhoudsvriendelijk en niet compleet. Door de tijd heb ik enkele keren een deel van de sprint gereserveerd om mijn al geschreven testen te bekijken en te verbeteren waar nodig. Dit heeft soms wel veel tijd gekost, maar als ik nu terugkijk naar het resultaat ben ik blij dat ik het gedaan heb. Zo heeft Argeüs in de toekomst ook niet veel werk moesten er veranderingen komen aan de API die ik getest heb. Alle veranderingen kunnen snel en gemakkelijk toegepast worden op de testen zonder al te veel te moeten berekenen of aanpassen.

Daarnaast ben ik zeer blij met het uiteindelijk resultaat en de manier waarmee naar dit resultaat toe gewerkt is. De keuze van Postman was een keuze in combinatie van mij en het bedrijf en de uitwerking is uiteindelijk goed gelukt. Er zijn meer dan 1600 testen geschreven tijdens mijn stage. Deze 1600 testen kunnen nu in vier minuten uitgevoerd worden terwijl als deze testen allemaal manueel uitgevoerd moeten worden dan zou dat bijna twee weken in beslag nemen. Daarbij komt ook nog dat als deze testen manueel uitgevoerd worden er niet zo specifiek gekeken wordt en kleine afwijkingen die toch grote gevolgen kunnen hebben, zullen vaak bij deze manuele testen niet gevonden worden. Dit geeft mij toch een grote voldoening. Ook omdat de automatische testen al gebruikt worden door de programmeurs zelf die ze runnen terwijl ze een bug aan het verbeteren zijn of een feature aan het uitwerken zijn om te zien als ze geen regressiefout gemaakt hebben.

Ik ben zeer tevreden over mijn stage en over mijn groeipad wat ik in deze twaalf weken heb gelopen. Deze stage heeft mij gebracht van een student naar een junior collega die klaar is om te beginnen met werken als IT'er.

II. Onderzoekstopic

1 Onderzoeksvraag

Er zijn zeer veel tools op de markt voor het testen van Application Programming Interfaces (API's). Deze tools zijn echter verschillend, net als elke API. Hierdoor heeft elke tool zijn plus- en minpunten.

De onderzoeksopdracht is om verschillende tools te vergelijken en te kijken welke tool het best past in het testproces van Argeüs. Dit wil zeggen dat de tool aan enkele voorwaarden moet voldoen.

Enkele voorwaarden zijn dat de testen automatisch moeten kunnen draaien (dit kan bijvoorbeeld door middel van Jenkins) en dat er communicatie mogelijk moet zijn met Testrail. Argeüs maakt gebruik van Testrail als testdesign- en testmanagementtool.

1.1 Methode van onderzoek

Dit onderzoek gebeurt door eerst te kijken welke testtools er op de markt zijn. Deze lijst van tools zal gemaakt worden aan de hand van een literatuurstudie die verschillende bronnen zal vergelijken om een zo goed mogelijke lijst van API-testtools op te bouwen.

Vanuit deze bronnen wordt dan een opsomming gemaakt met de kenmerken, prijzen en de conclusie van hoe elke tool binnen Argeüs zou passen of waarom hij juist niet binnen Argeüs zou passen. Daaruit worden dan enkele tools gekozen die goed passen binnen het testproces van Argeüs om verder te onderzoeken, te vergelijken en te testen met als doel om met de beste tool voor Argeüs te eindigen. [2] [3] [4] [5]

2 Onderzoek naar mogelijke tools

2.1 Literatuurstudie

Voor de literatuurstudie wordt er gestart vanuit een blog geschreven door Alice Aldaine. Alice Aldaine is een senior QA-engineer die een grote passie heeft voor testen. Zij heeft een top tien lijst opgemaakt van de beste API-testtools van 2018 en hier dan een kleine uitleg bij geschreven. [2]

Deze blog van Alice is ook gedeeld in Smartspate. Smartspate is een informatietechnologieblog ontwikkeld door twee informatici experten. Smartspate is vooral gericht naar het delen en aanleren van technologieën in de IT. [3]

Daarnaast wordt er gekeken naar een blog geschreven door Joe Collantonio. Joe Colantonio is een Test Automation Architect met meer dan twintig jaar ervaring in automatische testen. Hij heeft zijn eigen bedrijf genaamd TestTalks waar hij vanuit zijn ervaring in automatisch testen reviews en tutorials maakt over verschillende testtools en testmethodes. Zijn testtalks zijn altijd zeer specifiek. Vaak gaat hij een gesprek aan met de ontwikkelaar van de desbetreffende tool om te begrijpen uit welke noden deze tool gebouwd is. Zo kan hij zich plaatsen in het idee waarrond de tool gebouwd is. [6]

Verder wordt er gebruik gemaakt van een blog op DynoMapper. Deze blog over top 30 API-testtools is geschreven door de oprichter van DynoMapper, Garenne Bigby. Garenne Bigby is een ervaren web-ontwikkelaar. [7]

Uit deze blogs komen al enkele bekende tools overeen die dan toegevoegd zijn aan de lijst van tools waar verder onderzoek naar gedaan is. Deze tools zijn SoapUI, Postman, Katalon, Trisentis, Rest-Assured, Jmeter en Assertible. Dit zijn op Apigee na ook alle tools die Alice bespreekt in haar blog. Zij heeft Apigee nog toegevoegd, maar deze tool wordt niet meer ondersteund voor testing, enkel nog voor API-management.

Uit de tools die nog beschreven staan in de twee andere blogs, de blog van Joe en van Garenne, zijn dan aan de hand van de informatie die ze hierover beschreven nog enkele andere tools toegevoegd aan de lijst. Dit zijn tools waarbij de uitleg van beide personen positief was en waarbij de community achter de tools actief is. Zo zijn de tools KarateDSL en Citrus Framework nog toegevoegd.

2.2 Verwachtingen Argeüs

Voor de testtool die binnen Argeüs gebruikt zal worden zijn er enkele verwachtingen. Om te beginnen hebben ze liefst een tool die opensource of gratis is, echter als de kosten niet te hoog oplopen en de voordelen groot genoeg zijn is Argeüs bereid om te betalen voor deze tool. Ten tweede maken ze binnen Argeüs gebruik van zowel Windows 10, MacOS en Linux (Ubuntu). Het spreekt dus voor zich dat deze tool best op al de besturingssystemen draait.

Zoals hierboven vermeld zijn er zeer veel verschillende API's. Daarom is het zeer belangrijk dat de juiste tool gevonden wordt voor de API van Argeüs. De API van Argeüs geeft een antwoord op verschillende berekeningen die de motor van de Lifeplanner doet. De automatische testen zullen dus moeten controleren of de berekeningen wel correct zijn. De testtool moet daarom de mogelijkheid hebben om berekeningen te doen zodat het verwacht resultaat niet vastgezet wordt, maar variabel gehouden. Zo kunnen de testen snel en effectief omgaan met veranderingen in de motor van de Lifeplanner.

Verder moeten de testen natuurlijk automatisch kunnen draaien. Dit moet mogelijk zijn door middel van Jenkins. Binnen Argeüs maken ze gebruik van Jenkins als *Continuous Integration* (CI) en *Continuous Delivery* (CD) tool. Jenkins zal het dus mogelijk maken dat bij elke update van de Lifeplanner en elke update van de testen automatisch alle testen gerund kunnen worden op een testserver die Jenkins zelf opzet.

Als laatste moet er de mogelijkheid zijn om te rapporteren in Testrail. Testrail is de testmanagementtool die Argeüs gebruikt. Testrail zal gebruikt worden om bij elke *test run* een overzicht te creëren van wat er gelukt is en wat er mislukt is. Zo krijgt Argeüs een mooi overzicht van alle *test runs* die al ooit gelopen zijn en kan men perfect zien waar er problemen zijn. Aan de andere kant is dit veel overzichtelijker dan terug te moeten kijken in Jenkins welke testen gefaald zijn en welke gelukt zijn.

2.3 Schema mogelijke tools

Tabel 1: Overzicht mogelijke tools

Tool	Prijs	Programmeertaal	Ondersteunde besturingssystemen	Automatisatie mogelijk	Mogelijkheid berekeningen	Integratie Jenkins	Integratie testrail	Mogelijk passend binnen Argeüs
Postman Postman Pro Postman Enterprise	Gratis $8 per gebruiker per maand $18 per gebruiker per maand	Javascript	Windows 10 Ubuntu MacOs	✓	✓	✓	✓	✓
Insomnia Rest-Client Plus Teams	Gratis $5 per maand of $50 per jaar $8 per gebruiker per maand of $80 per gebruiker per jaar	/	Windows 10 Ubuntu MacOs	✗	✗	✗	✗	✗
SoapUI	Gratis	Groovy Javascript	Windows 10 Ubuntu MacOs	✓	✓	✗	✗	✗
SoapUI Pro Fixed license Floating license ReadyAPI	 €595 per jaar €4100 per jaar Bespreken met SoapUI			✓	✓	✓	Zelf programmeren	✗

Tool	Prijs	Programmeertaal	Ondersteunde besturingssystemen	Automatisatie mogelijk	Mogelijkheid berekeningen	Integratie Jenkins	Integratie testrail	Mogelijk passend binnen Argeüs
Katalon	Gratis	Groovy	Windows 10 Ubuntu MacOs	✔	✔	✔	✔	✔
Tricentis	Bespreken met Tricentis	/	Windows 10	✔	✔	✔	Zelf programmeren	✘
JMeter	Gratis	/	Windows 10 Ubuntu MacOs	✔	✘	✔	Zelf programmeren	✘
Rest-assured	Opensource	Java DSL BDD	Windows 10 Ubuntu MacOs	✔	✔	✔	Zelf programmeren	✔
Assertible Personal Standard Startup Business	Gratis $25 per maand $50 per maand $100 per maand	Programmeren niet mogelijk	Windows 10 Ubuntu MacOs	✔	✘	✔	✘	✘
KarateDSL	Opensource	BDD (Cucumber)	Windows 10 Ubuntu MacOs	✔	✘	✔	Zelf programmeren	✘
Citrus Framework	Opensource	BDD (Cucumber) Java XML	Windows 10 Ubuntu MacOs	✔	✔	✔	Zelf programmeren	✔

2.3.1 Postman

Figuur 23 Postman logo

Postman is een *Representation State Transfer* (REST) *client* die gestart is als een Chromebrowser plug-in. Deze is later ontwikkeld tot een applicatie voor Windows, Mac en Linux. Enkele grote klanten die gebruikmaken van de Postman testtool zijn Microsoft, Cisco, Imgur, Movember en Shopify. [8]

Postman is een gemakkelijk te gebruiken REST client met een rijkelijk gevulde interface waardoor Postman een simpele, maar gebruiksvriendelijke tool is.

Postman wordt zowel voor handmatige als voor automatische testen gebruikt. Voor de automatische testen kan gebruikgemaakt worden van Newman als *collection runner* zodat alle testen automatisch kunnen gedraaid worden door Jenkins. Jenkins is een CI-platform om herhaaldelijk taken uit te voeren.

Postman heeft de mogelijkheid om testen te delen op verschillende manieren. De eerste manier binnen Postman Pro is de mogelijkheid om volledige *collection* te delen binnen een bepaald team. De andere manier is om de testen en mogelijk ook de *environment variable* te exporteren naar een *JavaScript Object Notation* (JSON) bestand. Deze bestanden kunnen dan later geïmporteerd worden bij de andere teamleden om de *collections* van de andere teamleden te kunnen bekijken. Verder worden deze JSON-bestanden ook gebruikt voor de bovenstaande vermelde Newman. Newman zal deze bestanden gebruiken om de testen te kunnen draaien. Het commando om Newman op te roepen is te zien in figuur 17, te zien in hoofdstuk 3.3.3.2 Newman.

Het schrijven van de testen gebeurt in JavaScript.

Standaard is Postman een gratis tool. Maar Postman heeft wel twee betalende versies: Postman Pro en Postman Enterprise, deze kosten respectievelijk $8 en $18 per gebruiker per maand. [8]

Postman Pro en Postman Enterprise hebben enkele extra toepassingen ten opzichte van de standaard Postman. Dit is bijvoorbeeld de mogelijkheid om in teams te werken. Deze werking met teams zal ervoor zorgen dat de *requests* en variabelen gedeeld kunnen worden tussen de teamleden via Postman. Voor Postman Pro zijn dit teams van maximaal 50 gebruikers; Postman Enterprise heeft daarentegen geen limiet. [8] Deze extra functionaliteit binnen Argeüs een "*nice to have*", maar geen "must have". Er is nog steeds de mogelijkheid om te delen via een git *repository*.

2.3.1.1 Waarom past Postman wel of niet binnen Argeüs?

Postman wordt op dit moment al gebruikt binnen Argeüs. Hierdoor heeft Postman al een groot voordeel ten opzichte van de andere testtools.

Een ander groot voordeel van Postman is de mogelijkheid om de testen automatisch te draaien via Jenkins. Bij Argeüs wordt al gebruik gemaakt van Jenkins dus deze integratie zal zeer vlot verlopen.

Een volgend voordeel is de koppeling met Testrail. Elke automatische *test run* zal zijn resultaat sturen naar Testrail en daar een nieuwe *test run* maken. Hierdoor zal er een mooi overzicht ontstaan van wanneer welke testen geslaagd en gefaald zijn. Ook de mogelijkheid om de testen te exporteren naar het JSON-formaat zal hier een groot voordeel zijn. Alle testen kunnen dan in dit JSON-formaat op de BitBucket van Argeüs komen waar vervolgens Jenkins deze kan afhalen. Zo kunnen gemakkelijke nieuwe testen geschreven worden en rechtstreeks toegevoegd worden aan de collectie van automatische testen.

Een nadeel van Postman is dat er geen mogelijkheid is om functies te hergebruiken zonder een omweg. Een functie die voor een bepaalde test geschreven wordt, moet dan toegevoegd worden aan een variabele om deze later nog eens te kunnen gebruiken.

Tabel 2: Voor- en nadelen Postman

Voordelen	Nadelen
Al in gebruik binnen Argeüs	Functies hergebruiken via omweg
Automatisering mogelijk via Jenkins	
Koppeling testrail beschikbaar	

2.3.2 Insomnia Rest-Client

Figuur 24: Insomnia Rest-Client logo

Insomnia Rest-Client is een REST client zoals Postman, maar met zijn eigen voor- en nadelen zoals iedere tool. Enkele grote klanten van de Insomnia Rest-Client zijn Netflix, Kayak, Box, Cisco, Logdna en 1800contacts. [9]

Insomnia Rest Client heeft net als Postman een rijkelijk gevulde interface en is daardoor een simpele, maar gebruiksvriendelijke tool.

De Insomnia Rest Client geeft de mogelijkheid om *requests* te sturen naar een API en daarna op het antwoord van de API te filteren. Dit maakt Insomnia Rest Client een goede tool voor de *exploratory* testing. Daarentegen is de Insomnia Rest Client geen tool om automatische testen te schrijven.

Insomnia Rest-Client heeft drie versies: Free, Plus en Teams. De Free versie is zoals het woord het al zegt de gratis versie van Insomnia Rest-Client. [9]

Naast de Free versie is er de Plus versie. Deze versie heeft als pluspunt dat *End-To-End Encryption* mogelijk is. Dit wil zeggen dat de data die gebruikt wordt voor de testen veilig en geëncrypteerd op je account bijgehouden worden voor al de toestellen van die gebruiker. De Plus-versie kost $5 per maand of $50 per jaar. [9]

Als laatste heeft Insomnia Rest-Client nog een Teams-versie. Deze versie heeft alle voordelen van de Plus-versie met nog de mogelijkheid om in teams te werken. De data wordt dan veilig en geëncrypteerd gesynchroniseerd tussen alle gebruikers van dat team. Er is de mogelijkheid om de gebruikers van je team te beheren. De Team-versie heeft ook voorrang op hulp als er problemen optreden. Het Team plan kost $8 per gebruiker per maand of $80 per gebruiker per jaar. [9]

2.3.2.1 Waarom past Insomnia Rest-Client wel of niet binnen Argeüs?

De Insomnia Rest Client is geen tool die gebruikt kan worden voor de automatische testen van de Lifeplanner. Deze tool is een zeer mooie en eenvoudige tool om te controleren als de API een antwoord geeft op een *request*.

Tabel 3: Voor-en nadelen Insomnia Rest-Client

Voordelen	Nadelen
Mooie overzichtelijke GUI	Geen automatisatie mogelijk
Eenvoudig	Geen koppeling naar Testrail
	Niet mogelijk om testen te programmeren

2.3.3 Soap UI

Figuur 25: SoapUI logo

SoapUI Pro is de wereldleider op vlak van functioneel testen voor Simple Object Access Protocol (SOAP) en REST API testing. [10]

SoapUI Pro is de betalende versie van SoapUI. SoapUI Pro vergemakkelijkt automatische testen en het onderhoud van de testen. SoapUI Pro is een onderdeel van het ReadyAPI platform dat ervoor zorgt dat gebruikers gemakkelijker complexe functionele, load en security testen kunnen schrijven. [10]

Verder heeft SoapUI Pro een *native* Jenkins plug-in en een *command line* runner. Deze onderdelen zijn zeer belangrijk binnen een automatisch testproces. [10]

Ook het importeren van data is een functie die aanwezig is in SoapUI Pro en niet de gratis versie. Dit gaat om Tekstbestanden, *comma-separeted values* (CSV) bestanden, Excel-bestanden en nog enkele andere. [10]

Als laatste is er de support. SoapUI Pro heeft telefonische ondersteuning en 24/7 e-mail ondersteuning. [10]

2.3.3.1 Waarom past SoapUI wel of niet binnen Argeüs?

SoapUI zal geen plaats vinden binnen Argeüs. De gratis versie van SoapUi heeft niet genoeg mogelijkheden. De Jenkins integratie is een zeer belangrijk onderdeel voor Argeüs en deze is niet aanwezig binnen de gratis versie. Daar tegenover heeft SoapUI Pro deze integratie wel en nog andere positieve onderdelen, maar de prijs valt buiten budget van Argeüs.

Tabel 4: Voor- en nadelen SoapUI

Voordelen	Nadelen
SoapUI Pro is zeer uitgebreid	Betalende versie is duur
Jenkins plugin in soapUI pro	Geen koppeling naar Testrail aanwezig
	Gratis versie is niet uitgebreid genoeg

2.3.4 Katalon

Figuur 26: Katalon logo

Katalon API testing is een tool die gemaakt is voor het automatiseren van API testen. Katalon wordt gebruikt in meer dan 140 landen voor meer dan 28000 bedrijven over heel de wereld. [11]

Katalon maakt gebruik van de programmeertaal Groovy. Groovy is een scriptingtaal gebaseerd op Java. Verder heeft Katalon de mogelijkheid om *Behavior driven development* (BDD) te programmeren. Dit wordt mogelijk gemaakt door gebruik te maken van Gerkin. Dit wil zeggen dat Katalon Cucumber-*compliant* is. [11]

Katalon biedt ook zeer veel ingebouwde integraties zoals met Jira, Jenkins en Testrail.

De IDE van katalon geeft ook zeer mooie voordelen. Een van deze voordelen is dat er een debugger aanwezig is binnen de IDE. Dit zorgt ervoor dat er gemakkelijk fouten gevonden kunnen worden binnen de testen. [11]

Katalon is volledig gratis te gebruiken.

2.3.4.1 Waarom past Katalon wel of niet binnen Argeüs?

Katalon is een zeer mooie tool voor Argeüs. Er is een integratie met zowel Jenkins als testrail en dit is wat Argeüs zeker nodig heeft. Ook de debugger zal een groot pluspunt zijn om betere testen sneller te kunnen opleveren. De opbouw van de tool is wat ingewikkeld, maar dit kan door trainingen opgelost worden.

Tabel 5: Voor- en nadelen Katalon

Voordelen	Nadelen
Debugger aanwezig	Niet zo vanzelfsprekend als Postman.
Automatisering mogelijk via Jenkins	
Koppeling testrail beschikbaar	

2.3.5 Tricentis

Figuur 27: Tricentis logo

Tricentis is de nummer één van de continue testplatformen in de cloud.

Enkele grote namen die gebruik maken van Tricentis: Siemens, NSW goverment, ASB, Vantiv, Varian.

Tricentis is een tool die zeer veel functies bevat. Tricentis kan gebruikt worden voor zowel *API testing*, Cross browser testing, mobile testing, Sap testing, End-to-end testing en nog veel meer. Tricentis is dus een totaalpakket met vele integraties, veel mogelijkheid tot volledige automatisatie en met een hoge gebruiksvriendlijkheid voor onderhoud. Tricentis is dus een all-in-one tool die zeer veel mogelijkheden te bieden heeft. [12]

Tricentis is een zeer dure tool. De prijzen van Tricentis moeten besproken worden met Tricentis. De prijs kan variëren doordat Tricentis de mogelijkheid heeft om customiseren naar de wensen van de klant.

2.3.5.1 Waarom past Tricentis wel of niet binnen Argeüs?

Deze tool zal zeer veel mogelijkheden bieden voor Argeüs, maar door de prijs zal Tricentis geen plaats krijgen binnen Argeüs. Als de prijs geen probleem zou zijn dan zou Tricentis zeker een tool zijn die verder onderzocht moet worden.

Tabel 6: Voor- en nadelen Tricentis

Voordelen	Nadelen
Tricentis is zeer uitgebreid	Zeer duur
Gebruiksvriendelijk	

2.3.6 JMeter

Figuur 28: JMeter logo

JMeter is een tool die gebouwd is als performance testing tool, maar heeft de mogelijkheid om de response van een API ook te testen. Echter is deze mogelijkheid beperkt. Er is niet de mogelijkheid om het verwacht resultaat te berekenen. Het functioneel testen binnen JMeter is vooral de bedoeling om simpel een bepaald JSONPath op te zoeken en te controleren als die een bepaalde waarde heeft. Dit is niet gebouwd voor uitgebreide response body's waarin gezocht en berekend moet worden. [13]

JMeter is een opensource tool die zowel op Windows als Linux en Mac werkt.

2.3.6.1 Waarom past JMeter wel of niet binnen Argeüs?

JMeter heeft als functioneel API-testtool geen plaats binnen Argeüs. Het kan wel gebruikt worden als performance testing tool. Het niet kunnen berekenen van het verwacht resultaat zorgt ervoor dat er geen plaats is binnen Argeüs voor JMeter.

Tabel 7: Voor- en nadelen JMeter

Voordelen	Nadelen
Opensource	Geen berekeningen mogelijk
Functioneel en niet functioneel testen	Geen integratie met Testrail

2.3.7 Rest-Assured

Figuur 29: Rest-Assured logo

Rest-Assured is een opensource testtool. De testen worden geschreven in Java *Domain-Specific Language* (DSL). Dit wil zeggen dat Rest-Assured speciaal ontwikkeld is om Rest API's gemakkelijker te kunnen testen. Door gebruik te maken van BDD, de "*given*", "*when*", "*then*" structuur in java, worden alle testen geschreven binnen een Java IDE. Rest-Assured werkt in alle Java IDE's die Maven ondersteunen. Hierdoor is Rest-Assured beschikbaar in de drie grote besturingsystemen: Windows, Linux en Mac. [14]

Er is wel een nadeel aan rest-Assured. Het meegeven van de *JSON-Body* moet gebeuren in Java en niet in JSON. Hierdoor moet in Java een omzetting gebeuren die de JSON omzet naar een string in Java. Dit hoeft bijvoorbeeld bij Postman niet.

2.3.7.1 Waarom past Rest-Assured wel of niet binnen Argeüs?

Rest-Assured is een zeer mooie tool voor Argeüs. Het heeft veel mogelijkheden doordat het geschreven wordt in Java. Verder is er de mogelijkheid voor integratie met Jenkins en de integratie met Testrail kan zelfgeschreven worden. Doordat Rest-Assured in Java wordt geschreven, heeft het dus de mogelijkheid om het expected result te programmeren wat voor Argeüs zeer belangrijk is. Daarnaast is er wel het probleem van de *JSON-body* dat meegegeven wordt, moet nog omgevormd worden. Binnen Argeüs is het JSON-bestand aanwezig, maar niet de Java String variant.

Tabel 8: Voor- en nadelen Rest-Assured

Voordelen	Nadelen
Opensource	Integratie Testrail zelf schrijven
BDD is mogelijk	
Java	
Integratie Jenkins	
Berekenen mogelijk	

2.3.8 Assertible

Figuur 30: Assertible logo

Assertible is een API test en management tool. Assertible is een webapplicatie dat dus in de browser werkt. Dit zorgt ervoor dat deze tool werkt op de drie belangrijke besturingssystemen: Windows, Linux en Mac. Verder is Assertible een "no code" tool. Er zijn geen programmeerskills nodig om met Assertible te kunnen werken. Alles wordt gedaan vanuit een overzichtelijke GUI. [15]

Binnen Assertible zijn er vier plannen. Om te beginnen is er het Personal plan. Dit is de gratis versie van Assertible en geeft de mogelijkheid om met één tot twee personen te testen op twee webservices waarbij er per webservice tien testen gemaakt kunnen worden. Ook kunnen er bij deze versie, maar 1000 resultaten van uitgevoerde testen opgeslagen worden.
Daarnaast is er het Standard plan. Dit plan kost $25 per maand en vergroot het aantal webservices naar 50, de testen per service naar 1000 en het aantal opgeslagen resultaten naar 10 000. Echter kan bij deze versie maar één persoon werken.
Na het Standard plan komt het Startup plan. Dit plan kost $50 per maand. Dit plan geeft maar de helft aan webservices, testen per service en resultaten die bijgehouden kunnen worden dan het Standard plan, maar hiertegenover staat wel dat je met een team van tien personen kan werken aan deze testen.
Als laatste is er het Business plan. Dit is het duurste plan van Assertible en kost $100 per maand. Met dit plan krijg je dezelfde resources als het Standard plan, maar kan je met een team van twintig personen werken.

Het "no code" verhaal zorgt er wel voor dat de mogelijkheden van Assertible beperkter zijn dan een tool gelijk Postman waar in Javascript alle testen geprogrammeerd moeten worden.

2.3.8.1 Waarom past Assertible of niet binnen Argeüs?

Deze tool past niet in het kader van Argeüs. Argeüs heeft een ingewikkelde rekenmotor en hierdoor moet de tool de mogelijkheid hebben om het antwoord van de API te controlleren door middel van berekeningen te doen in de testen. Deze mogelijkheid heeft Assertible niet.
Verder zou het ook zeer handig zijn moesten de programmeurs binnen Argeüs de mogelijkheid hebben om de testen lokaal uit te voeren terwijl ze een nieuwe feauture of een *bugfix* aan het maken zijn. Zo kunnen ze halverwege, wanneer ze een blok afhebben, controleren als alles nog werkt voor ze verder werken aan het volgende deel van de *bugfix* of feature.
Ook de prijs speelt een rol. Argeüs verwacht een opensource of gratis tool. Enkel als de meerwaarde zeer groot is zal Argeüs overwegen om toch een betalende tool te kiezen. Daar komt Assertible niet voor in aanmerking.

Tabel 9: Voor- en nadelen Assertible

Voordelen	Nadelen
Opensource	Integratie Testrail zelf schrijven
	Berekeningen niet mogelijk

2.3.9 Karate DSL

Figuur 31: Karate logo

Het grote voordeel van Karate is ook een groot nadeel. Met Karate worden de testen geschreven in BDD, maar bij Karate moet in tegenstelling tot de meeste andere testtools geen *step definitions* geschreven worden. Dit maakt Karate een tool die simpel en gemakkelijk werkt voor mensen die geen programmeerachtergrond hebben, maar dit geeft ook beperkingen. Doordat geen *step definitions* geschreven kunnen worden zijn de mogelijkheden van de tool beperkter.

Karate is een tool die ontwikkeld is om testers met een technische achtergrond maar geen sterke programmeerachtergrond, toch testen te laten schrijven. De technische kennis is nodig omdat het toch geen echte volledige BDD is met volledig leesbare zinnen. Het is gebaseerd op het BDD-principe, maar met vastgelegde stappen die de gebruiker moet volgen.

2.3.9.1 Waarom past Karate wel of niet binnen Argeüs?

Doordat Karate BDD werkt, maar zonder *step definitions* zal de mogelijkheid die deze tool heeft te klein worden en niet gebruikt kunnen worden binnen Argeüs. De testen die gebruikt worden binnen Argeüs zijn vaak uitgebreide testen met veel rekenwerk waar de tester met Karate in de problemen zal komen.

Tabel 10: Voor- en nadelen Karate

Voordelen	Nadelen
Opensource	Integratie Testrail zelf schrijven
	Berekeningen niet mogelijk
	BDD zonder step definitions

2.3.10 CitrusFramework

Figuur 32: CitrusFramework logo

Het CitrusFramework is een opensource tool die werkt in elke Java IDE dat Maven ondersteund. Hierdoor ondersteund CitrusFramework de drie grote besturingssystemen: Windows, Linux en Mac.

Testen in het CitrusFramework worden geschreven in Java of *Extensible Markup Language* (XML). Dit geeft dan ook weer hetzelfde nadeel als bij Rest-Assured. De *JSON-body* die meegegeven wordt aan het POST *request* moet omgezet worden van het JSON-formaat naar het een Java string formaat. [16]

CitrusFramework is gelauncht in 2009 en is sindsdien met regelmaat geüpdatet naar een goed werkende en stabiel framework met zeer veel mogelijkheden. Verder is er ook een zeer uitgebreide documentatie te vinden op hun website.

2.3.10.1 Waarom past CitrusFramework wel of niet binnen Argeüs?

CitrusFramework voldoet aan alle eisen van Argeüs. CitrusFramework is in mogelijkheden sterk te vergelijken met Rest-Assured alleen de uitwerking van de testen verschilt.

Tabel 11: Voor- en nadelen CitrusFramework

Voordelen	Nadelen
Opensource	Integratie Testrail zelf schrijven
Berekenen mogelijk	
Java	

3 Prototypes

3.1 Aanpak

Uit hoofdstuk 2 zijn vier tools gekozen die verder uitgewerkt gaan worden. Voor deze tools zal een prototype ontworpen worden. Eén testcase zal uitgewerkt worden met elke tool om te vergelijken hoe deze tools juist werken en zo zal er een conclusie genomen worden welke tool het beste zal zijn voor Argeüs. De tools zijn Postman, Katalon, Rest-Assured en CitrusFramework.

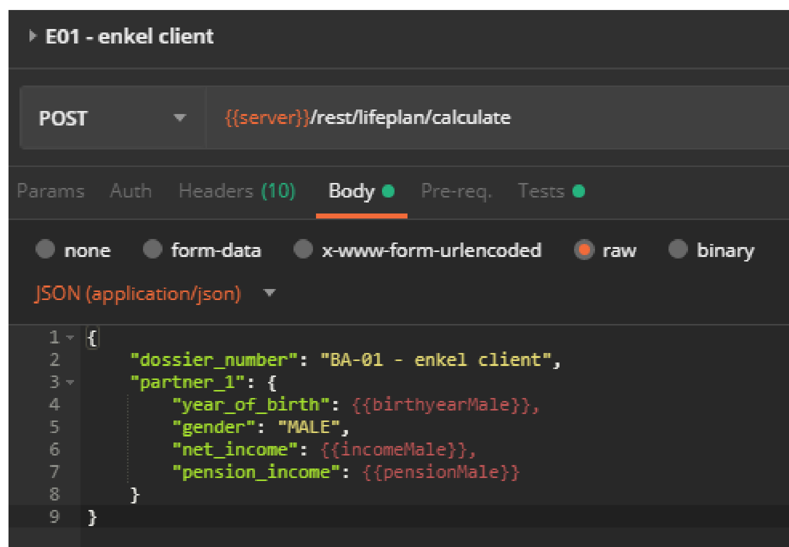
Postman is de baseline. Alle andere tools worden vergeleken met deze tool. Postman heeft alles wat Argeüs nodig heeft en is daardoor de ideale vergelijkingstool. Daarnaast is de werking van de deze tool gekend.

3.2 De uitwerking

Voor de uitwerking wordt er enkel gekeken naar de verschillende werkingen van de tools. Zo wordt bekeken wat de verschillen zijn tussen de tools.

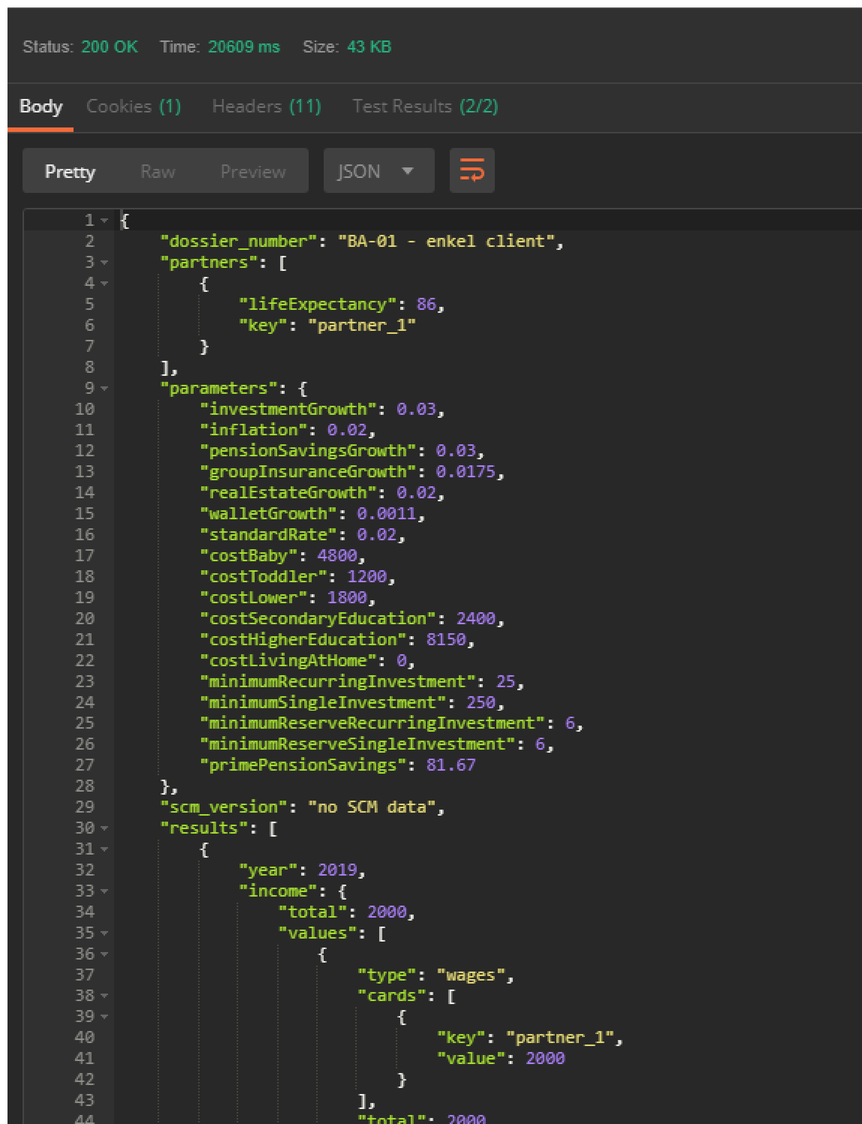
3.2.1 Postman

Voor het POST-request body op te bouwen in Postman moet de JSON meegegeven worden. Dit kan gemakkelijk gedaan worden door onder het tab Body, het formaat om te zetten naar “raw” en “JSON(application/json)” en daarna de json van de request te zetten. Zoals te zien is in Figuur 33 wordt de JSON opgebouwd met variabelen. Zo kan eenzelfde variabele meerdere malen gebruikt worden en op één plaats veranderd worden om alle testen die gebruik maken van deze variabelen aan te passen. Alle variabelen moeten tussen dubbele accolades gezet worden.



Figuur 33: Postman request body

Na het meegeven van de body kan het request verstuurd worden en komt er een overzicht van de response. Dit zorgt ervoor dat het antwoord bekeken kan worden en hierin gezocht kan worden wat getest moet worden. Zo kan er ook gezien worden hoe er genavigeerd moet worden door de JSON-structuur om aan dit antwoord te komen. In figuur 34 is te zien hoe dit antwoord leesbaar is in Postman.



Figuur 34: Response body in Postman

Nu kunnen de testen geschreven worden. Zoals eerder vermeld, in hoofdstuk 3.3.2.2 van deel I stageverslag, worden alle functies apart gezet in één request zodat ze opnieuw gebruikt kunnen worden in andere testcases. In Figuur 35 is te zien hoe een test opgebouwd wordt binnen Postman. Zo is te zien dat om te beginnen de "eval" functie opgeroepen wordt met alle functies zodat deze gebruikt kunnen worden in deze testcase. In deze case worden drie functies opgeroepen uit de lijst van functies: getIncomeForPartner, calculateInflationOverYears en roundToEven. Deze testcase is iets uitgebreider dan de andere prototypes. De calulateInflationOverYears en roundToEven functies worden niet toegepast in de andere prototypes omdat deze hier niet nodig zijn. Deze functies zijn gebouwd om veranderingen naar de toekomst aan te kunnen met de automatische testen.

E01 - enkel client

POST {{server}}/rest/lifeplan/calculate

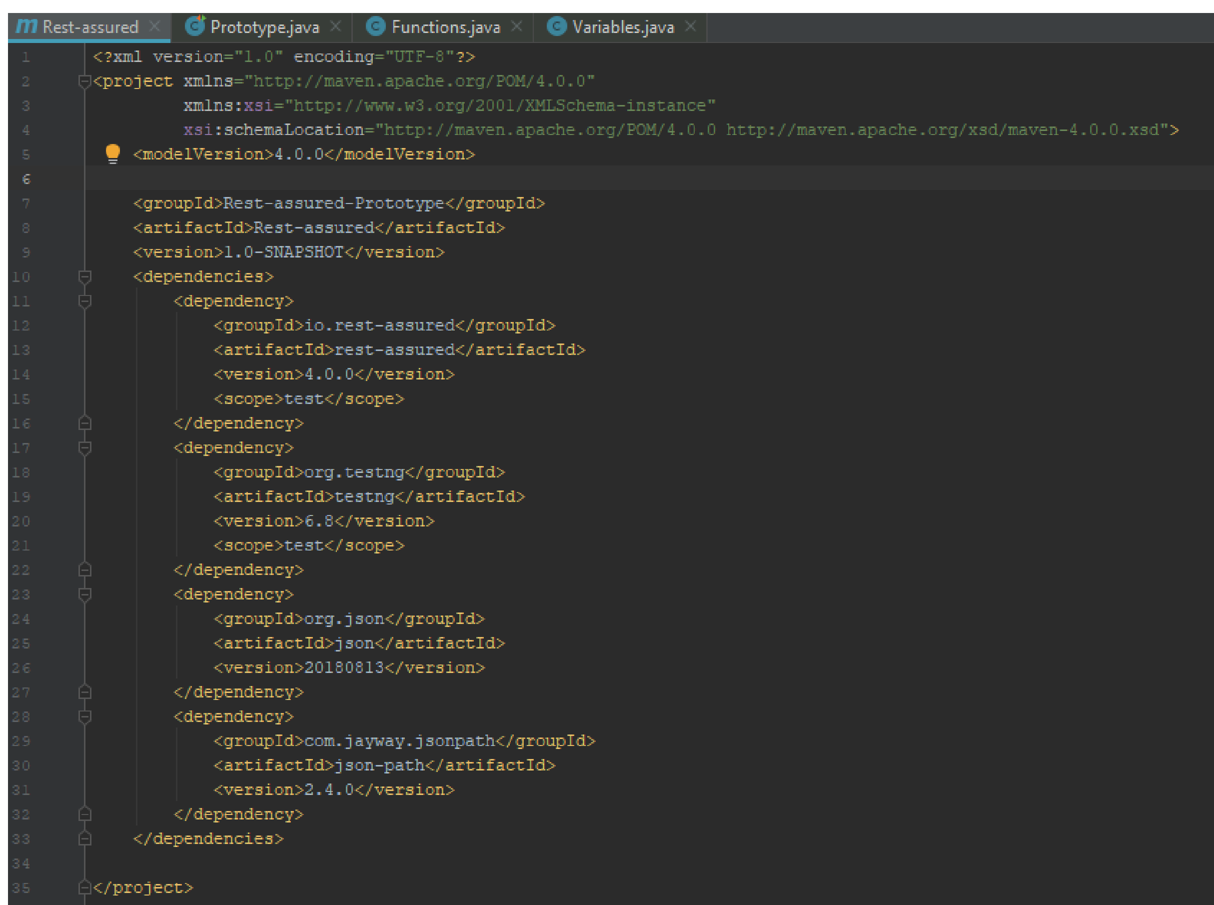
Params Auth Headers (10) Body Pre-req. Tests

```
1  eval(pm.environment.get("functies"));
2
3      var data = getBody();
4      var thisYear = parseInt(pm.environment.get("thisYear"));
5      var yearOfBirth = parseInt(pm.environment.get("birthyearMale"));
6      var defaultPensionAge = parseInt(pm.environment.get("defaultPensionAge"));
7
8  pm.test("C2 De client krijgt het juiste pensioen als hij zijn pensioenleeftijd heeft bereikt.", function () {
9      const pensionMale = parseInt(pm.environment.get("pensionMale"));
10     const pensionYear = yearOfBirth + defaultPensionAge;
11
12     var amountOfYearsTillPension = pensionYear - thisYear;
13     var dataPension = getDataForYear(pensionYear ,data);
14     var dataPensionIncome = dataPension.income;
15
16     var pensionFromEngine = getIncomeForPartner("pension", dataPensionIncome, "partner_1");
17
18     var calculatedPension = calculateInflationOverYears(pensionMale, amountOfYearsTillPension);
19
20     calculatedPension = roundToEven(calculatedPension);
21
22     pm.expect(pensionFromEngine).equal(calculatedPension);
23 });
24
25 pm.test("C3 De client krijgt het juiste loon het jaar voor zijn pensioen", function () {
26     const incomeMale = parseInt(pm.environment.get("incomeMale"));
27
28     var pensionYear = yearOfBirth + defaultPensionAge;
29     var dataYearBeforePension = getDataForYear(pensionYear -1 ,data);
30     var dataIncomeYearBeforePension = dataYearBeforePension.income;
31
32     var amountOfYearsTillLastWage = pensionYear - thisYear - 1;
33
34     var wageFromEngine = getIncomeForPartner("wages", dataIncomeYearBeforePension, "partner_1");
35
36     var calculatedWage = calculateInflationOverYears(incomeMale, amountOfYearsTillLastWage);
37
38     calculatedWage = roundToEven(calculatedWage);
39
40     pm.expect(wageFromEngine).equal(calculatedWage);
41 });
```

Figuur 35: Prototype Postman

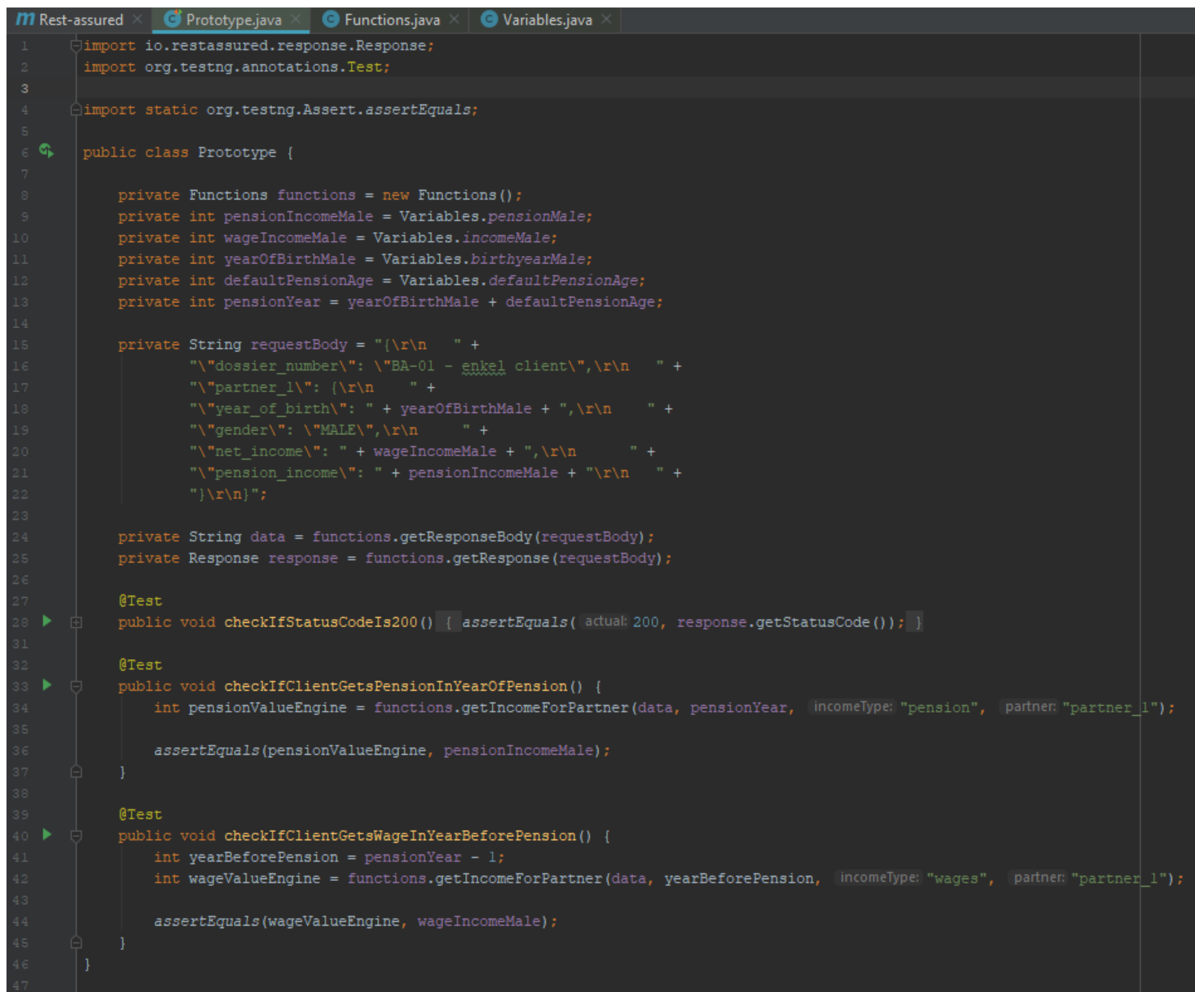
3.2.2 Rest-Assured

Rest-Assured wordt in tegenstelling tot Postman geschreven in Java en niet in Javascript. Daarnaast worden de Postman testen geschreven in de Postman IDE. Rest-Assured heeft geen eigen IDE, voor Rest-Assured werd er gekozen om te werken in de IDE van InteliJ. Hierdoor is de opstart voor Rest-Assured anders dan de opstart voor Postman. Postman heeft geen voorbereiding nodig. Alles is aanwezig in de Postman Sandbox, de IDE van Postman. Hiertegenover moet voor Rest-Assured nog enkele onderdelen toegevoegd worden aan het project in InteliJ. Hiervoor wordt gebruik gemaakt van Maven. Met Maven kunnen alle benodigdheden voor een Rest-Assured test project op te bouwen geïmporteerd worden. Hiervoor moeten wel de juiste Maven *dependencies* ingehaald worden. Dit wordt gedaan door een nieuw Maven project aan te maken en dan alle *dependencies* toe te voegen aan de pom.xml file, zoals te zien in figuur 36.



Figuur 36: Rest-Assured pom.xml dependencies

Als dit gedaan is kan er begonnen worden aan het schrijven van de testen. Om te beginnen moet er voor een testcase dus ook een request body meegegeven worden. Dit moet deze keer in stringvorm en niet in JSON-formaat. Gelukkig kan InteliJ JSON-formaat omzetten naar string formaat door het gewoon te kopiëren van een JSON-file naar de InteliJ IDE. Zoals te zien is in figuur 37 in de variabele requestBody.



Figuur 37: Prototype Rest-Assured

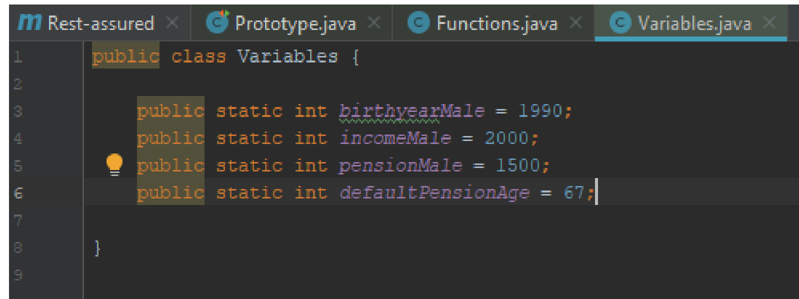
In Rest-Assured is te zien hoe de response eruitziet. Hierdoor kan het gemakkelijker zijn om de request eerst door te sturen via Postman en de testen te schrijven in Rest-Assured of om via de GCFT-testtool samen met chrome developer tools te kijken naar het antwoord in JSON-formaat. Zo kan er gezien worden welk pad genomen moet worden om het juiste antwoord te vinden.

Bij Rest-Assured wordt er ook gebruik gemaakt van een andere Javafile voor alle functies. Dit zorgt ervoor dat de functies vanuit elke testcase opgeroepen kunnen worden. Zo kan er in Rest-Assured gewerkt worden met packages en kan alles mooi opgedeeld worden, zoals we in Postman gebruik maken van collections. Zo blijft het overzicht mooi behouden en hoeft er geen tijd gestoken te worden in het zoeken van de juiste testcase of functie. Deze functie file is deels te zien in figuur 38.

```
import java.util.List;

public class Functions {

    public String getResponseBody(String requestBody) {
        Response response = getResponse(requestBody);
        String data = response.asString();

        return data;
    }

    public Response getResponse(String requestBody) {
        RestAssured.baseURI = "http://localhost:8082/rest/lifeplan";

        Response response = null;

        try {
            response = RestAssured.given()
                    .contentType(ContentType.JSON)
                    .body(requestBody)
                    .post( s: "/calculate");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return response;
    }

    public JSONObject getYearResult(JSONArray results, int year) {
        for (Object objResult : results) {
            JSONObject yearResult = (JSONObject) objResult;
            if (yearResult.getInt( key: "year") == year) {
                return yearResult;
            }
        }
        throw new IllegalArgumentException("can't find year " + year);
    }

    public int getIncomeForPartner(String allData, int year, String incomeType, String partner) {
        return getValue(allData, getYearIncomePartner(year, incomeType, partner), prop: "value");
    }
```

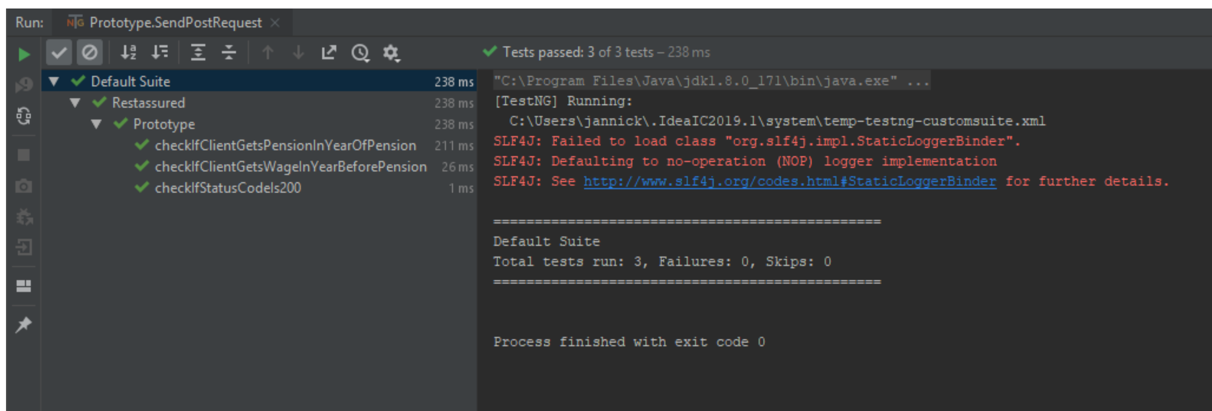
Figuur 38: Rest-Assured functie file

Als laatste maken we binnen Rest-Assured ook gebruik van globale variabelen. Deze zitten ook in een aparte Javafile. Zo kunnen deze ook opgeroepen worden vanuit elke testcase. Deze file is te zien in figuur 39.



Figuur 39: Rest-Assured variabelen file

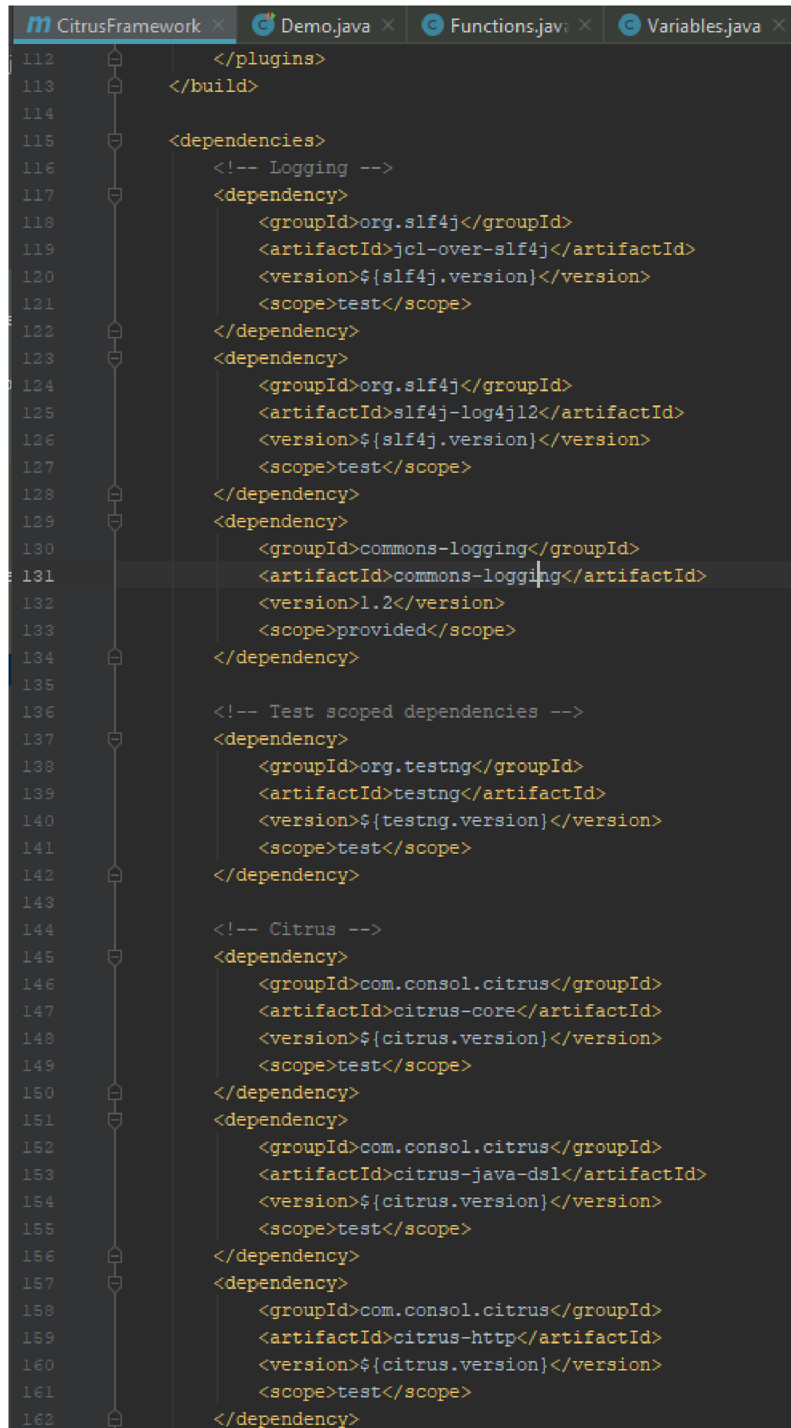
Het resultaat van deze testen is te zien in Unit-test vorm. Zo is te zien in figuur 40 hoe Rest-Assured het resultaat teruggeeft in InteliJ.



Figuur 40: Resultaat Rest-Assured

3.2.3 CitrusFramework

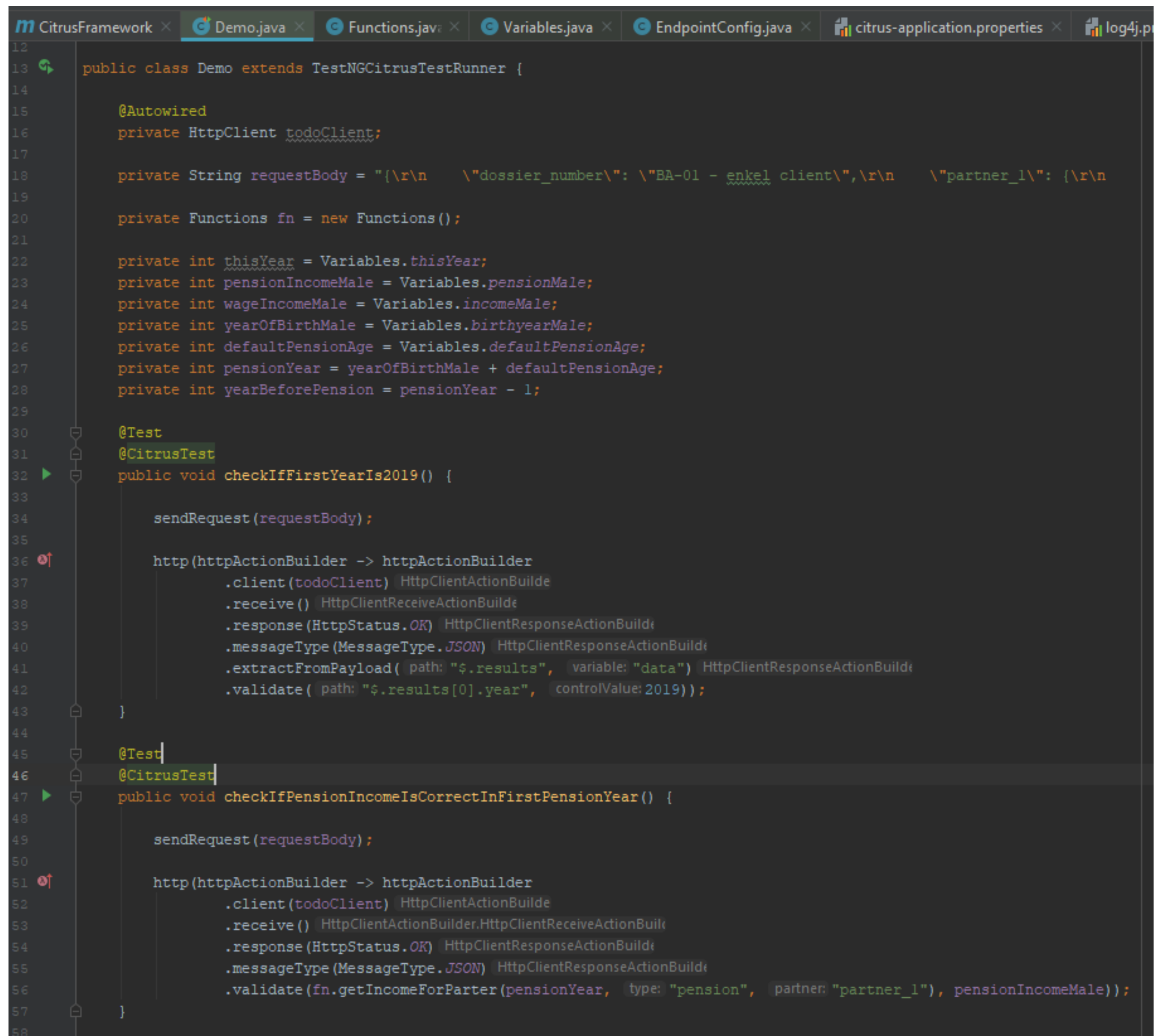
CitrusFramework heeft ook geen eigen IDE. Hiervoor wordt dus opnieuw gebruik gemaakt van InteliJ met dezelfde voor- en nadelen als bij Rest-Assured. Net als bij Rest-Assured maakt CitrusFramework ook gebruik van Maven. Echter is de pom.xml file van CitrusFramework veel uitgebreider dan die van Rest-Assured. Een deel van deze pom.xml file is te zien in figuur 41.



```
        </plugins>
    </build>

    <dependencies>
        <!-- Logging -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${slf4j.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
            <scope>provided</scope>
        </dependency>

        <!-- Test scoped dependencies -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>${testng.version}</version>
            <scope>test</scope>
        </dependency>

        <!-- Citrus -->
        <dependency>
            <groupId>com.consol.citrus</groupId>
            <artifactId>citrus-core</artifactId>
            <version>${citrus.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.consol.citrus</groupId>
            <artifactId>citrus-java-dsl</artifactId>
            <version>${citrus.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.consol.citrus</groupId>
            <artifactId>citrus-http</artifactId>
            <version>${citrus.version}</version>
            <scope>test</scope>
        </dependency>
```

Figuur 41: CitrusFramework pom.xml

Daarnaast is de werking van CitrusFramework zeer verschillend dan Rest-Assured terwijl ze beiden in Java geprogrammeerd zijn. In figuur 42 is te zien hoe het prototype opgebouwd is in CitrusFramework.

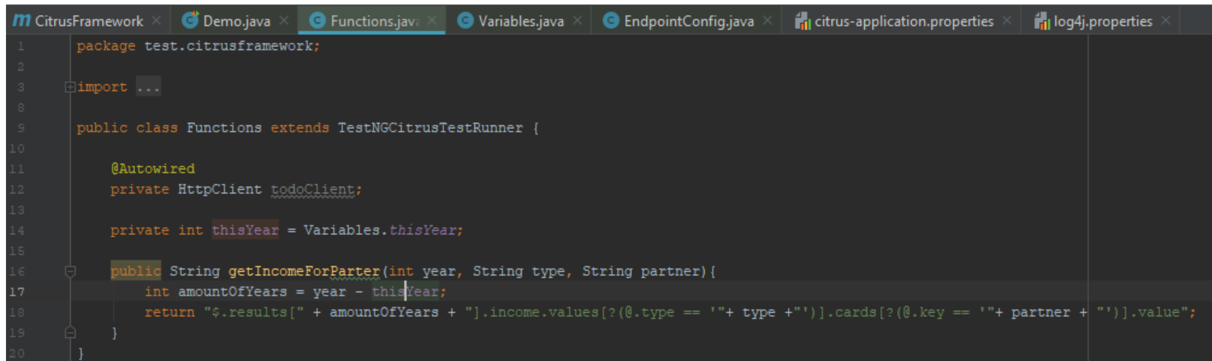


Figuur 42: Prototype CitrusFramework

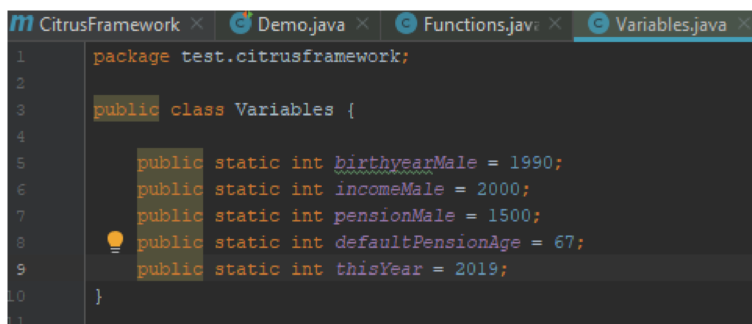
Bij CitrusFramework kan maar één test gebeuren per request dat gestuurd wordt naar de Lifeplanner, toch als je wilt zien welke test faalt en welke niet. Er kunnen meerdere “validates” gebeuren per request, maar deze worden allemaal toegevoegd aan dezelfde test, dan kan er alleen gezien worden als ze ofwel allemaal slagen of als één faalt dan faalt de hele test.

Alle testen moeten beginnen met de annotaties @Test en @CitrusTest. Zo kan TestNG zien bij het runnen wat allemaal testen zijn.

Net zoals in Postman en in Rest-Assured maken we gebruik van functies die in een aparte functiefile zitten en variabelen die in een aparte variabelen file zitten. In figuur 43 is de functiefile te zien en in figuur 44 is de variabelen file te zien.

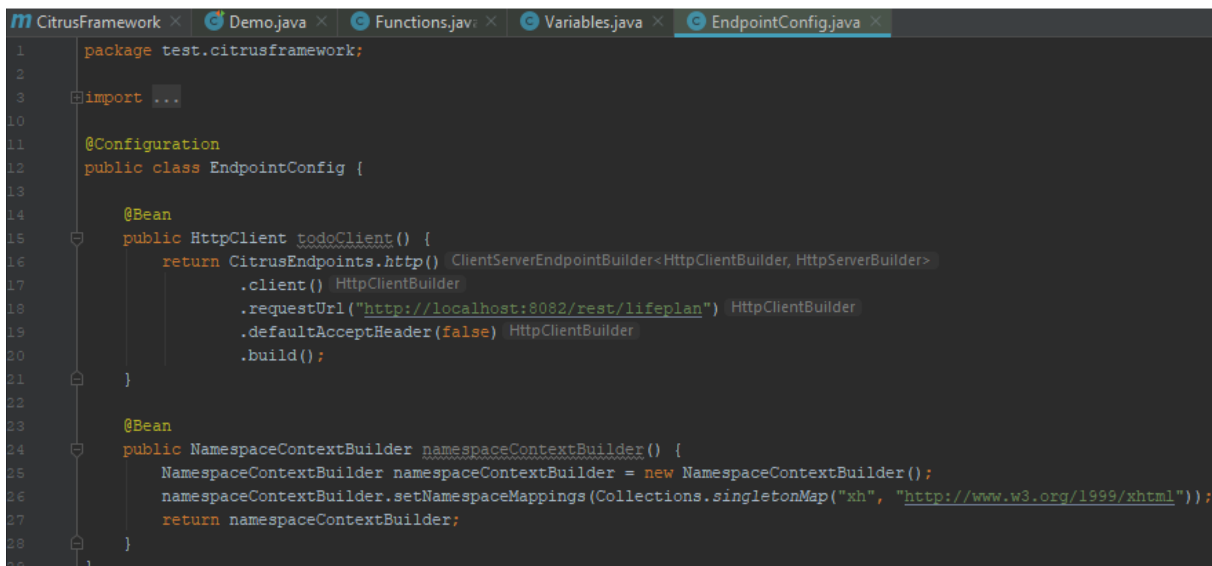


Figuur 43: CitrusFramework functiefile



Figuur 44: CitrusFramework variabelenfile

In tegenstelling tot Rest-Assured heeft CitrusFramework nog een andere file nodig om de testen te kunnen runnen. Dit is de EndpointConfig.Java file. In deze file wordt de connectie met de API opgebouwd. In figuur 45 is te zien hoe deze file opgebouwd is voor het prototype.



Figuur 45: CitrusFramework EndpointConfig.Java

CitrusFramework heeft minder mogelijkheden dan Rest-Assured en Postman. Dit heeft vooral te maken met de opbouw en manier van werken in CitrusFramework.

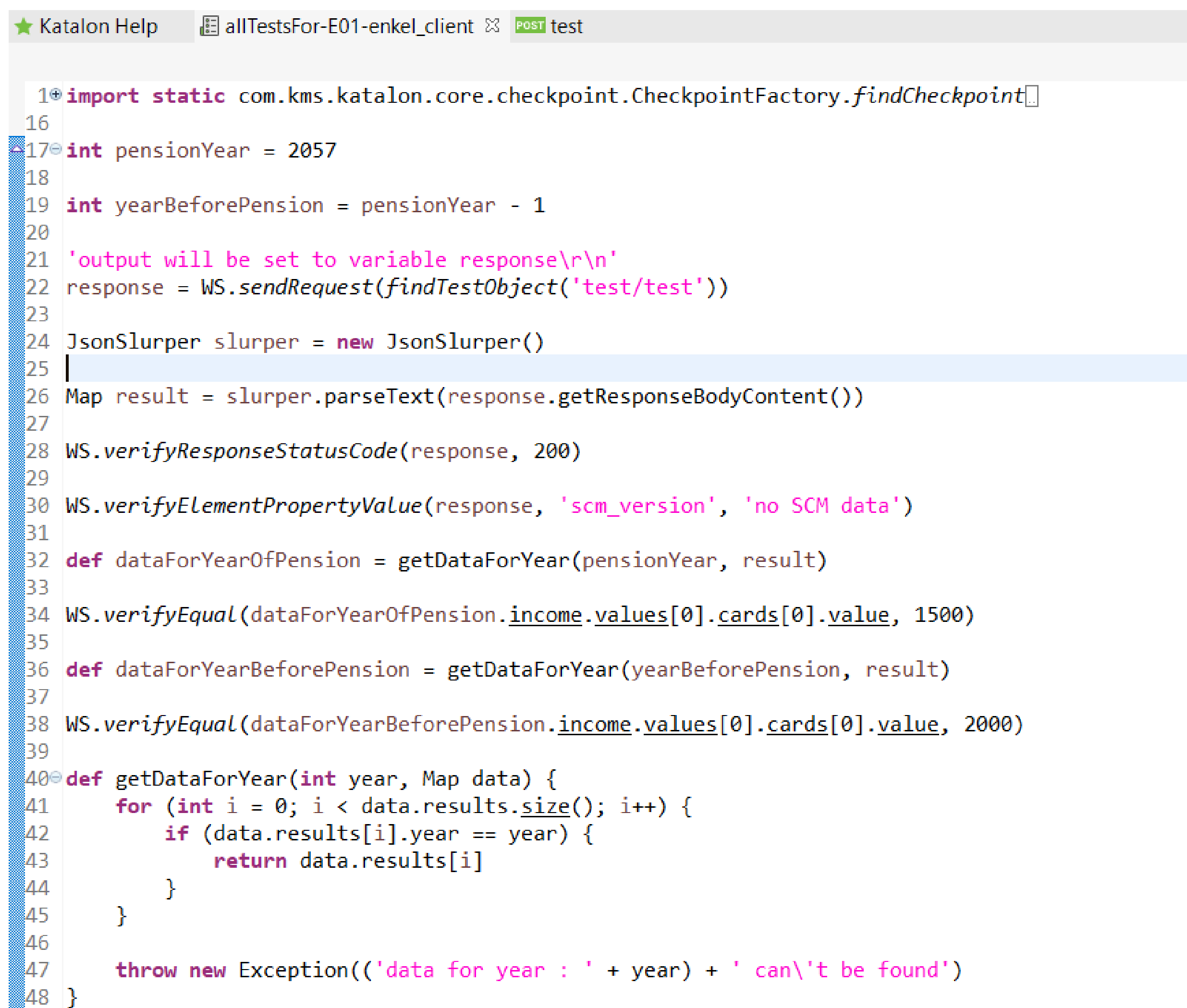
3.2.4 Katalon

Katalon werkt op een heel andere manier dan de vorige drie tools. Katalon maakt gebruikt van twee manieren om testen te schrijven. Deze twee manieren zijn scripting mode en manual mode. In manual mode kan via toevoegen van sleutelwoorden zonder iets te programmeren een test opgebouwd worden. Deze sleutelwoorden worden dan omgezet, in scripting mode, naar groovy code. Dit wil dus zeggen als er iets toegevoegd wordt in scripting mode, door het zelf te programmeren in groovy, dit ook toegevoegd wordt in manual mode.

Dit is dus een heel andere manier van werken en dit zorgt wel voor een leercurve om deze taal aan te leren.

Door te werken met Groovy zijn de mogelijkheden van Katalon ook veel kleiner dan Postman en Rest-Assured.

In figuur 46 is te zien hoe het prototype opgebouwd is in scripting mode in Katalon. Vervolgens is te zien in figuur 47 hoe dit eruit ziet in manual mode.



Figuur 46: Prototype Katalon Scripting mode

Katalon Help | allTestsFor-E01-enkel_client | test

Add Web Service Keyword | Add | Recent keywords | Delete | Move up | Move down | Edit tags | Add to test suite

Item	Object	Input	Output	Description
1 - Binary Statement		pensionYear = 2057		
2 - Binary Statement		yearBeforePension = pension		
3 - Send Request	test		response	output will be set to variable response
4 - Binary Statement		slurper = new JsonSlurper()		
5 - Binary Statement		result = slurper.parseText(resp		
6 - Verify Response Stat		response; 200		
7 - Verify Element Prope		response; "scm_version"; "no :		
8 - Binary Statement		dataForYearOfPension = getD		
9 - Verify Equal		dataForYearOfPension.income		
10 - Binary Statement		dataForYearBeforePension =		
11 - Verify Equal		dataForYearBeforePension.inc		
getDataForYear()				

Figuur 47: Prototype Katalon manual mode

Katalon maakt gebruik van test cases waar alle testen in staan, Object Repositories waar de requesten in komen en Test Suites waar een groepering van test cases in komt, om zo verschillende testen te kunnen runnen.

Zo is er voor dit prototype ook een object repository gemaakt die het request stuurt met de request body erbij. Deze is te zien in figuur 48.

Katalon Help | allTestsFor-E01-enkel_client | test

POST | http://localhost:8082/rest/lifeplan/calculate

Customize API methods

Query Parameters

Add | Remove

Name	Value

Authorization | HTTP Header | HTTP Body | Verification | Variables | Variables Editor | Configuration

text | x-www-form-urlencoded | form-data | file

```
{
    "dossier_number": "BA-01 - enkel client",
    "partner_1": {
        "year_of_birth": 1990,
        "gender": "MALE",
        "net_income": 2000,
        "pension_income": 1500
    }
}
```

Figuur 48: Katalon object repository

Katalon laat zoals Postman het antwoord wel zien. Hierdoor is het gemakkelijker om in het antwoord te kijken welk pad nodig is om aan het juiste JSON-onderdeel te komen. Hoe katalon dit weergeeft is te zien in figuur 49.

Response HAR

Status: 200 OK Elapsed: 4991 ms Size: 43 KB

Body Header Verification Log

(•) pretty ( ) raw ( ) preview


```
{
  "dossier_number":"BA-01 - enkel client",
  "partners":[
    {
      "lifeExpectancy":86,
      "key":"partner_1"
    }],
  "parameters":{
    "investmentGrowth":0.0300,
    "inflation":0.02000000000000,
    "pensionSavingsGrowth":0.03,
    "groupInsuranceGrowth":0.0175,
    "realEstateGrowth":0.02,
    "walletGrowth":0.00110000000000,
    "standardRate":0.02,
    "costBaby":4800.00,
    "costToddler":1200,
    "costLower":1800,
    "costSecondaryEducation":2400,
    "costHigherEducation":8150,
    "costLivingAtHome":0,
    "minimumRecurringInvestment":25.00,
    "minimumSingleInvestment":250.00,
    "minimumReserveRecurringInvestment":6.0000,
    "minimumReserveSingleInvestment":6.0000,
    "primePensionSavings":81.67
  },
  "scm_version":"no SCM data",
  "results":[
    {
      "year":2019,
      "income":{
        "total":2000,
        "values":[
          {
```


Select JSON or XML response data and press Ctrl/Command + K to add verification scripts directly.

(•) JSON ( ) XML ( ) HTML ( ) JavaScript [x] Wrap Line

Figuur 49: Katalon response body

De resultaten komen per testcase als één test. Worden er meerdere variabelen gecontroleerd, telt dit nog altijd maar als één test. Faalt een onderdeel van de test dan faalt alles. Katalon maakt wel een onderverdeling dus uiteindelijk kan wel teruggekeken worden welk onderdeel gefaald is. In figuur 50 is te zien hoe Katalon het resultaat weergeeft.

Figuur 50: Katalon resultaat

3.3 Vergelijking na prototypes

Nadat alle prototypes gebouwd zijn, is volgende tabel opgemaakt. In de onderstaande tabel zijn alle verschillen tussen de verschillende tools opgesomd. Deze verschillen zeggen niet welke tool dat beter is, maar geven aan wat de mogelijkheden en de werkingen zijn van de tools.

Tabel 12: Vergelijking na prototypes

	Postman	**Rest-Assured**	**CitrusFramework**	**Katalon**
Taal	Javascript	Java	Java	Groovy
Navigeren binnen JSON-Objecten	Zeer gemakkelijk	JsonPath of JsonObjects en JsonArrays	JsonPath ingebakken in validate	JsonSlurper
Integratie met Testrail	Plugin in Newman	Zelf schrijven	Zelf schrijven	Zelf schrijven/plugin kopen ($199/jaar)
Debugging	Beperkt tot console.log	Mogelijk	Mogelijk	Mogelijk
Globale functies	Mogelijk met *workaround*	Mogelijk	Mogelijk	Niet gevonden
Autocomplete	Enkel environment variabele	Mogelijk	Mogelijk	Mogelijk
Afzonderlijke *assertions* per *request*	Meerdere	Meerdere	Eén	Eén
IDE	Postman sandbox	Zelfgekozen Java IDE	Zelfgekozen Java IDE	Katalon studio

Conclusie

Er is geen tool die de beste is. Elke tool heeft zijn voor- en nadelen. Het komt er uiteindelijk op neer om een tool te nemen waar de nadelen niet opwegen tegen de voordelen voor elke specifieke case. Net zoals elke API anders is, heeft dus elke API ook een andere manier van testen.

Mijn persoonlijke voorkeur gaat naar Postman of Rest-Assured. De mogelijkheden van Rest-Assured zijn iets groter door de *autocomplete* die zeer handig kan zijn. Ook doordat debuggen mogelijk is en niet enkel met console.log, maar ook met breakpoints. Hierdoor worden fouten veel gemakkelijker en sneller gevonden. Vooral fouten naar namen die niet goed overgenomen zijn en dergelijke zullen veel sneller gevonden worden in Rest-Assured als in Postman.

Daarentegen heeft Postman een bestaande integratie met Testrail, deze zal voor Rest-Assured nog zelfgeschreven moeten worden.

Uiteindelijk is de keuze voor een testtool een keuze die aan verschillende factoren vasthangt. Als eerste is er het budget. Een betalende tool kan meer voordelen bieden, maar wegen deze voordelen hard genoeg door ten opzichte van een gratis tool. Dit zijn afwegingen die altijd gemaakt moeten worden bij het kiezen van een testtool.

Daarnaast is er de voorkeuren van de gebruiker van de tool. De ene persoon zou eerder voor Rest-Assured kiezen omdat deze werkt met Java. De andere persoon kiest juist voor Postman omdat deze persoon veel liever werkt met Javascript.

Voor Argeüs zijn er uiteindelijk twee winnaars. Zowel Postman als Rest-Assured zijn tools die perfect passen binnen Argeüs. Het zijn tools die aan alle voorwaarden voldoen die Argeüs heeft en daarnaast zijn het gebruiksvriendelijke tools met een lagere leercurve dan de andere twee. Ook de mogelijkheden zijn bij deze twee tools het grootst.

CitrusFramework en Katalon vallen na de prototypes uit de boot. Voor CitrusFramework is het vooral het nadeel dat er per test een nieuwe request gestuurd moet worden. Dit zou voor een zware belasting zorgen voor de testserver. Bij Rest-Assured en Postman gebeurt het vaak dat er meer dan tien testen uitgevoerd worden per request. Dit wil zeggen dat de load op de testserver met CitrusFramework tot wel tien keer zo hoog kan liggen dan voor Postman en Rest-Assured. Ook de manier van testen schrijven is niet zo duidelijk als bij Postman en Rest-Assured.

Daarnaast is Katalon een volledig andere manier van werken. Met het bouwen van het prototype voor Katalon waren zeer veel frustraties gemoeid. Katalon heeft een compleet andere manier van werken wat voor een hogere leercurve zorgt. Dit zorgt ervoor dat Katalon al snel uit de boot viel als mogelijke tool voor Argeüs.

Bibliografie

[1] Belfius, „Belfius YuMe,” [Online]. Available: https://www.belfius.be/retail/nl/sleutelmomenten/yume/index.aspx. [Geopend 25 april 2019].

[2] A. Aldaine. [Online]. Available: https://medium.com/@alicealdaine/top-10-api-testing-tools-rest-soap-services-5395cb03cfa9. [Geopend 04 maart 2019].

[3] Alex, „smartplate,” [Online]. Available: https://www.smartspate.com/top-10-tools-for-testing-apis-in-2018/. [Geopend 14 maart 2019].

[4] J. Colantonio, „joecolontonio,” [Online]. Available: https://www.joecolantonio.com/12-open-source-api-testing-tools-rest-soap-services/. [Geopend 04 maart 2019].

[5] A. Walling, „quora,” [Online]. Available: https://www.quora.com/What-are-the-best-tools-for-API-testing. [Geopend 14 maart 2019].

[6] J. Colantonio, „Linkedin,” [Online]. Available: https://www.linkedin.com/in/joecolantonio/. [Geopend 07 mei 2019].

[7] G. Bigby, „Dynomapper,” [Online]. Available: https://dynomapper.com/blog/513-top-30-application-programming-interface-api-testing-tools. [Geopend 30 maart 2019].

[8] „getpostman,” [Online]. Available: https://www.getpostman.com/. [Geopend 04 maart 2019].

[9] „insomnia,” Floating Keyboard Software Inc., [Online]. Available: https://insomnia.rest/. [Geopend 19 maart 2019].

[10] „SoapUI,” SMARTBEAR, [Online]. Available: https://www.soapui.org/. [Geopend 19 maart 2019].

[11] „Katalon,” [Online]. Available: https://www.katalon.com. [Geopend 28 maart 2019].

[12] „Tricentis,” [Online]. [Geopend 04 april 2019].

[13] „JMeter,” [Online]. Available: https://jmeter.apache.org/. [Geopend 14 mei 2019].

[14] „Rest-Assured,” [Online]. Available: http://rest-assured.io/. [Geopend 14 mei 2019].

[15] Assertible, „Assertible,” [Online]. Available: https://assertible.com/. [Geopend 07 mei 2019].

[16] „Citrus Framework,” [Online]. Available: https://citrusframework.org/. [Geopend 14 mei 2019].