Professionele Bachelor Toegepaste Informatica

# Geautomatiseerde integratietesten financiële berekeningen

Daan Vankerkom

Promotoren:

Reinaut Krekels — Argeüs cvba
Lowie Vangaal — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

# Geautomatiseerde integratietesten financiële berekeningen

Daan Vankerkom

Promotoren:

Reinaut Krekels — Argeüs cvba
Lowie Vangaal — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

# Dankwoord

Als eerste wil ik Vik Nelissen, Pieter De Backer en Davy Palmans bedanken om mij deze stage aan te bieden bij Argeüs.

Daarnaast wil ik mijn bedrijfs- en hogeschoolpromotoren: Reinaut Krekels en Lowie Vangaal bedanken voor hun feedback, steun en motivatie gedurende mijn stage. Dankzij hun feedback ben ik sterk gegroeid.

Ook wil ik mijn collega's bedanken voor hun feedback over mijn bachelorpaper.

Tot slot wil ik Berkan Aktag, Dries Steegmans en Jonas Gerits bedanken voor de steun en motivatie gedurende mijn opleiding aan de Hogeschool PXL.

# Abstract

Argeüs, gelegen in Lummen, biedt samen met experten uit de praktijk software aan. Ze wil op die wijze de adviesverlening in nichesegmenten van de financiële, notariële, juridische en accountancysector rentabiliseren. De huidige operationele softwareproducten zijn de modulaire successiecalculator (MSP) en de modulaire financiële planner (MFP).

Momenteel worden bij Argeüs regressietesten manueel uitgevoerd voordat er een release gebeurt. Dit manueel werk is arbeidsintensief, neemt veel tijd in beslag en kan foutieve resultaten door het vangnet laten glippen omdat kleine veranderingen de resultaten sterk kunnen beïnvloeden.

In het kader van de stageopdracht heeft Argeüs dan ook de vraag gesteld om deze regressietesten te automatiseren in combinatie met een programma dat de resultaten van de successieberekeningen vergelijkt met voorafgaande resultaten en bijkomend de verschillen visualiseert.

De tool voor deze geautomatiseerde testen bestaat uit een webbased interface waarin de resultaten en wijzigingen visueel weergegeven worden in tabellen. Achterliggend wisselt de applicatie informatie uit met Jenkins aan de hand van een REST-API. Voor de gebruikersinterface zijn Angular en Semantic UI gebruikt om snel een kwalitatief resultaat te bekomen. Developers en testers kunnen de resultaten van de testruns bekijken en testcases uploaden en verwijderen. De applicatie is geschreven in Java met behulp van de Spring Boot, Apache Commons en de Guava *libraries*. Jenkins informeert de applicatie op welke testomgeving de berekeningen moeten worden uitgevoerd en brengt hem op gang na een wijziging in de code.

Aansluitend kadert het onderzoek binnen innovatie. Er wordt nagegaan welke performantievoordelen de programmeertaal Rust kan bieden. Ook wil Argeüs weten of deze programmeertaal integreerbaar is in het ontwikkelproces van nieuwe applicaties. Een *proof of concept* van het onderzoek bestaat uit een REST API in Java en Rust die het verworven kapitaal per jaar met een jaarlijks vast inkomen berekent.

# Inhoudsopgave

# Lijst van gebruikte figuren

# Lijst van gebruikte tabellen

# Lijst van gebruikte afkortingen

| | |
|---|---|
| API | Application programming interface |
| ARM | Acorn RISC Machine |
| CLI | Command-line-interface |
| DTO | Data transfer object |
| GDPR | De algemene verordening gegevensbescherming |
| GUID | Globally unique identifier, een getal dat verondersteld wordt wereldwijd uniek te zijn. |
| HTTP | Hypertext Transfer Protocol |
| JSON | Javascript Object Notation |
| MFP | Modulaire financiële planner; een softwarepakket van Argeüs om aan persoonlijke financiële planning te doen. |
| MD5 | Message Digest Algorithm 5 |
| MSP | Modulaire successiecalculator, een softwarepakket van Argeüs voor notarissen om erfbelastingen te berekenen. |
| npm | Node.js package manager |
| ORM | Object-relational mapping |
| RAM | Random-access memory |
| REST | Representational state transfer |
| RISC | Reduced instruction set computer |
| SOLID | Acroniem voor: *Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion*. |
| VPN | Virtual private network |

# Lijst van belangrijkste begrippen

| | |
|---|---|
| Dossier | Een collectie van persoonsgegevens om berekeningen uit te voeren. |
| Dossiersleutel | Een ID-nummer van een dossier bestaande uit een GUID. |
| Successiecalculator | Een softwarepakket van Argeüs voor notarissen om erfbelastingen te berekenen. |
| Testrun | Een testiteratie gestart door Jenkins. |
| Testcaserun | Een testcase die uitgevoerd is tijdens een testrun. |
| Testcase | Een dossier waarvan de resultaten vergeleken worden. |

# Inleiding

Het testen van grote applicaties met veel rekenresultaten is heel uitdagend. In de successiecalculator maakt een eenvoudig dossier meer dan 500 berekeningen. Het manueel controleren van de resultaten van één dossier neemt veel tijd in beslag. In deze paper wordt het uitwerkingsproces beschreven van een testtool. Deze tool vergelijkt JSON-structuren en visualiseert de verschillen aan de hand van kleuren in de webbrowser.

Recent is er een nieuwe API gelanceerd door Argeüs. Deze API maakt het mogelijk voor externe partijen om de financiële planningssoftware aan te spreken vanuit websites. In het kader van optimalisatie is de vraag gesteld of de programmeertaal Rust een goed alternatief is voor het bouwen van API's. Hiervoor is er onderzoek verricht naar de voor- en nadelen van de programmeertaal Rust in combinatie met een toegepast onderzoek onderzocht. In het toegepast onderzoek werden performantiemetingen uitgevoerd op twee *proof of concept* API's geschreven in Rust en Java.

# I. Stageverslag

## 1 Bedrijfsvoorstelling

Argeüs is een fintech-bedrijf opgericht in 2011 als financial consultingbedrijf. In de loop van 2012 is het bedrijf uitgegroeid naar een softwareontwikkelingsbedrijf. Momenteel is het gevestigd in Lummen, België. Het bedrijf bestaat uit een team van 15 professionals die samen praktijkgerichte expertisesoftware aanbieden voor de financiële, notariële en juridische sector in België. Dit gebeurt in samenwerking met experten uit deze sectoren om een oplossing op maat te bekomen.

De huidige operationele softwareproducten zijn de modulaire successiecalculator (MSP) en de modulaire financiële planner (MFP).

*Figuur 1 Huidige locatie Argeüs*

### 1.1 Successiecalculator

De successiecalculator is een ondersteuningssoftware voor notarissen om erfenissen te kunnen verwerken. Met deze krachtige rekenmotor kunnen de kosten en de waarde van erfenissen en nalatenschappen berekenend worden. Ook kan deze applicatie de wettelijke aangiftedocumenten genereren. De successiecalculator is het eerste commerciële product dat ontwikkeld is door Argeüs als softwarebedrijf.

### 1.2 Financiële planningssoftware

Ondersteuningssoftware voor persoonlijke financiële planning. De software maakt een kritische analyse van de vermogens- en cashflowsituatie. Dit product was oorspronkelijk als interne tool gebouwd en is na de commerciële lancering van de successiecalculator opnieuw gebouwd en gecommercialiseerd.

# 2 Voorstelling opdracht

## 2.1 Probleemstelling

Momenteel worden bij Argeüs regressietesten manueel uitgevoerd voordat er een release gebeurt. Dit manueel werk is langzaam, neemt veel tijd in beslag en er kunnen foutieve resultaten door het vangnet glippen. Dit kan gebeuren omdat lichte veranderingen de resultaten sterk kunnen beïnvloeden.

Ontwikkelaars kunnen de ongewenste wijzigingen in de lijst van rekenresultaten niet altijd juist in kaart brengen omdat ze in een grote lijst zitten. Daardoor is volledige manuele controle onhaalbaar en kunnen fouten in de productie terechtkomen.

Het zoeken en oplossen van die fouten kan veel tijd en energie in beslag nemen wat kan leiden tot frustratie van de ontwikkelaars. Soms worden fouten pas duidelijk lang nadat de code gewijzigd is. Dat maakt het veel moeilijker en tijdrovender om de oorzaak op te sporen. Ook kan het gebeuren dat de oorspronkelijke ontwikkelaar niet meer beschikbaar is waardoor vragen onbeantwoord achterblijven.

Het is voor ontwikkelaars en het bedrijf vaak moeilijk om correct in te schatten hoeveel werk een aanpassing vraagt omdat verborgen gebreken plots een grote impact kunnen hebben. Dit kan leiden tot onverwachte tijdsnood. Wijzigingen en nieuwe features moeten dan soms uitgesteld worden door de oude fouten omdat die gecorrigeerd moeten worden.

Tijdens softwareontwikkeling is het vaak zinvol om de structuur van de code aan te passen aan nieuwe vereisten zodat deze nieuwe onderdelen in de toekomst makkelijk uitbreidbaar zijn. Deze wijzigingen kunnen ongewild leiden tot gebreken. Een beperkt zicht op deze gebreken kan leiden tot risico's. Om die risico's te vermijden kan het management structurele verbeteringen gaan uitstellen of verbieden waardoor de flexibiliteit van de code daalt en aanpassingen duurder worden.

Fouten hebben ook impact op de werkingsprocessen van de applicatiegebruikers. Als een fout in productie terechtkomt, dan kan het voorkomen dat de gebruiker terug moet schakelen naar manuele handelingen om de fouten te corrigeren. Daardoor is de gebruikerservaring dan niet optimaal en zou het kunnen voorkomen dat de gebruiker de applicatie niet gaat aanraden aan potentiële nieuwe klanten.

## 2.2 Doelstellingen

De menselijke fouten moeten worden opvangen door de regressietesten te automatiseren zodat het testen sneller en efficiënter gedaan kan worden. In combinatie met deze testen moet er een hulpprogramma ontwikkeld worden dat de ontwikkelaars en testers inzicht kan geven in wijzigingen bij het aanpassen van de successiecalculator. Het programma moet de resultaten van de successieberekeningen vergelijken met voorafgaande resultaten en meldingen uitsturen als er een resultaat gewijzigd is zodat rekenfouten vroegtijdig opgespoord kunnen worden.

De focus ligt op het controleren en visualiseren van wijzigingen in de rekenresultaten. Wijzigingen in de resultaten worden visueel voorgesteld aan de hand van drie kleuren: rood voor verwijderde, groen voor nieuwe en geel voor gewijzigde resultaten zodat ze makkelijker te herkennen zijn.

Resultaten worden weergeven per testcase in een testrun. Elke testrun en elk testcaserunresultaat heeft een statusindicator die aangeeft of er een resultaat gewijzigd is of er een fout gebeurd is tijdens het uitvoeren van de testcase.

Op de detailpagina worden de resultaten gevisualiseerd met het gepaste kleurenschema. Het is mogelijk om door de resultaten te zoeken om zo de ontwikkelaars en testers een inzicht te geven bij gewijzigde resultaten. Ontwikkelaars en testers kunnen testcases uploaden en verwijderen. Nieuwe testcases worden uitgevoerd wanneer er een nieuwe testrun gestart is.

De geschiedenis bijhouden en notificaties in het chatsysteem, Mattermost, zijn *nice-to-haves* als er tijd over is.

## 2.3 Omgeving

De testtool bestaat uit twee onderdelen; een webapplicatie en een API. De webapplicatie is geschreven in Typescript en maakt gebruikt van de Angular en Semantic UI frontend frameworks.

De backendapplicatie is geschreven in Java 8 en maakt gebruik van het Spring Boot-framework waaraan een MariaDB-database gekoppeld is door het Hibernate-framework. De ontwikkeling van de applicatie gebeurt in Intellij IDEA Community Edition in een Ubuntu-omgeving.

In de applicatie zijn REST-API *endpoints* geïntegreerd om te communiceren met de frontend. Voor het uitrekenen en opvragen van de resultaten van een dossier in de successiecalculator is er een POST-API-*endpoint* voorzien. Dit *endpoint* is enkel toegankelijk op testenvironments. Bij het uitvoeren van een POST-verzoek met een geldig dossier zal de applicatie dit dossier importeren en uitrekenen. De uitgerekende resultaten worden teruggestuurd in een JSON-formaat. Bij een ongeldig dossier wordt door het API-*endpoint* een foutcode teruggeven.

Om de API-*endpoints* te testen tijdens het ontwikkelen is het programma Postman gebruikt om de resultaten van de REST-API te controleren.

Voor het ontwikkelen van de webapplicatie zijn de volgende tools gebruikt: Angular-CLI, Firefox-webbrowser, Node.js, Node.js package manager (npm), en Visual Studio Code.

# 3 Uitwerking opdracht

## 3.1 Uitwerking

Er is gekozen om de Kanban methode te gebruiken om het werk te visualiseren en te beheren. Het uitwerken van de stageopdracht heeft in verschillende fases plaatsgevonden zodat er in iteraties gewerkt kon worden. Elke fase heeft het ontwikkelingsteam feedback en tips gegeven.

### 3.1.1 Fase 0

Fase 0 heeft plaatsgevonden voordat de ontwikkeling van de applicatie gestart was. Voordat deze fase begon heeft er een meeting met het ontwikkelingsteam en een klein onderzoek plaatsgevonden. Ook zijn de databasestructuur en enkele wireframes van de applicatie gemaakt.

#### 3.1.1.1 Meeting

De eerste meeting heeft plaatsgevonden om de stageopdracht te verduidelijken en de verwachtingen af te stemmen in het team. In deze meeting waren alle stakeholders aanwezig voor de eerste keer.

Uit deze meeting is voortgekomen dat de eerste iteratie van de applicatie een webapplicatie is die veranderingen in de output van de successiecalculator detecteert en visualiseert aan de hand van kleuren. Ook zijn de doelstellingen hier aan bod gekomen, zie 2.2 Doelstellingen voor de volledige lijst van de doelstellingen.

#### 3.1.1.2 Onderzoek

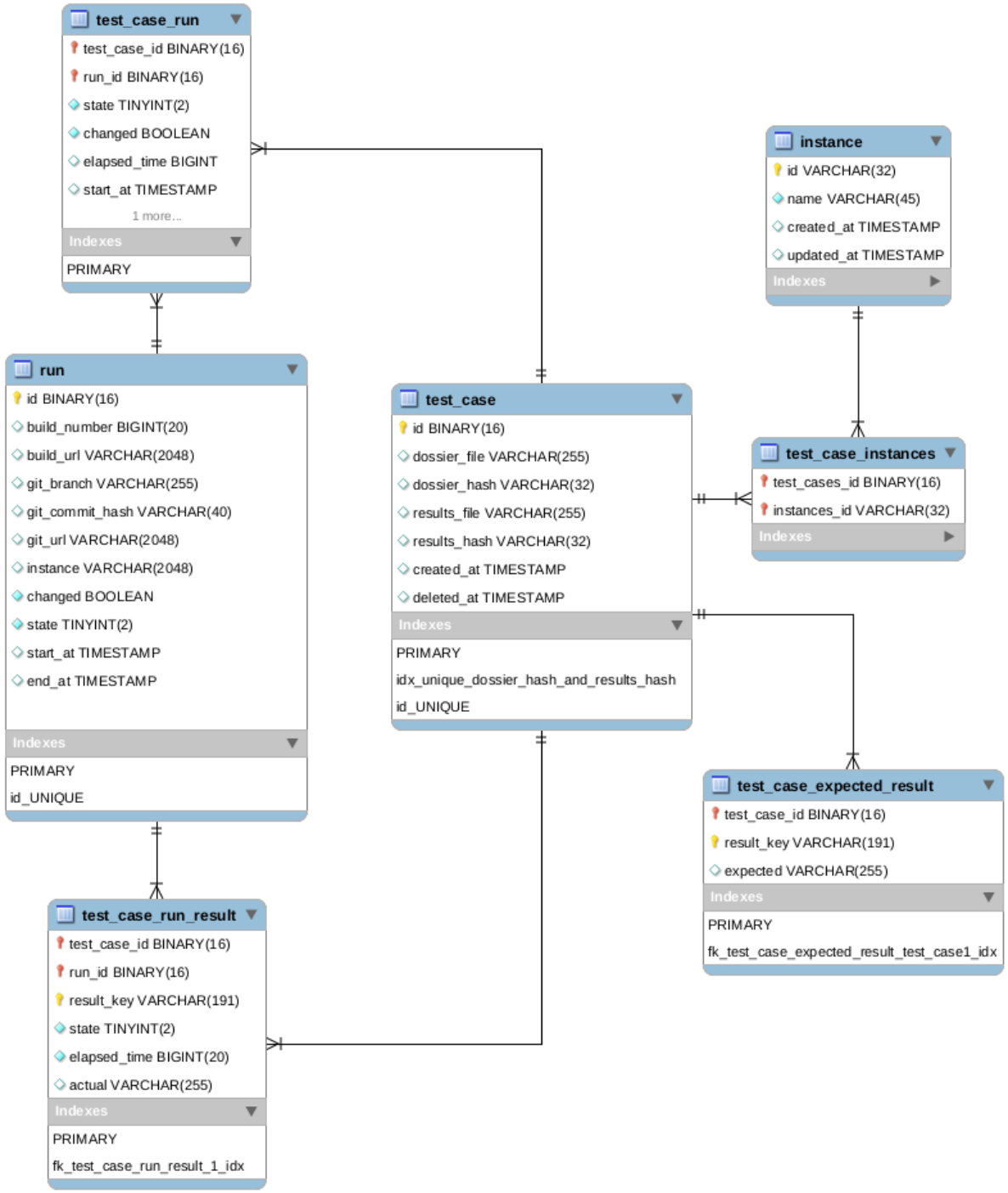
Voor de uitwerking van de stageopdracht was er een klein onderzoek gepland waarin vergelijkingslibraries onderzocht zouden worden om te kijken welke *library* het makkelijkste te implementeren en onderhouden was maar na enkele Google-zoekopdrachten was er al een simpele oplossing ontdekt.

De oplossing voor het vergelijken van twee JSON-structuren in Java was terug te vinden in een blogpost. Deze oplossing maakte gebruik van het *MapDifference*-object in de Guava *library*. Deze component berekent het verschil tussen twee Java-map-objecten en geeft zowel de gemeenschappelijke en verschillende waardes terug alsook de waardes die enkel in de linker- en rechtermap zitten als resultaat. [1] De verschillen in het resultaat komen overeen met ongewijzigd, gewijzigd, toegevoegd en verwijderd respectievelijk.

Het *MapDifference*-object kon dus gebruikt worden om de vergelijkingen te maken van de twee JSON-bestanden. Uiteindelijk is er gekozen voor deze oplossing omdat ze makkelijk verstaanbaar was en omdat de Guava *library* actief onderhouden wordt door Google [2].

### 3.1.1.3 Databasestructuur

De databasestructuur is opgebouwd in samenwerking met het ontwikkelingsteam. Voor de uitwerking heeft er een brainstormsessie plaatsgevonden. Onderstaande figuur bevat de genormaliseerde structuur van de database.

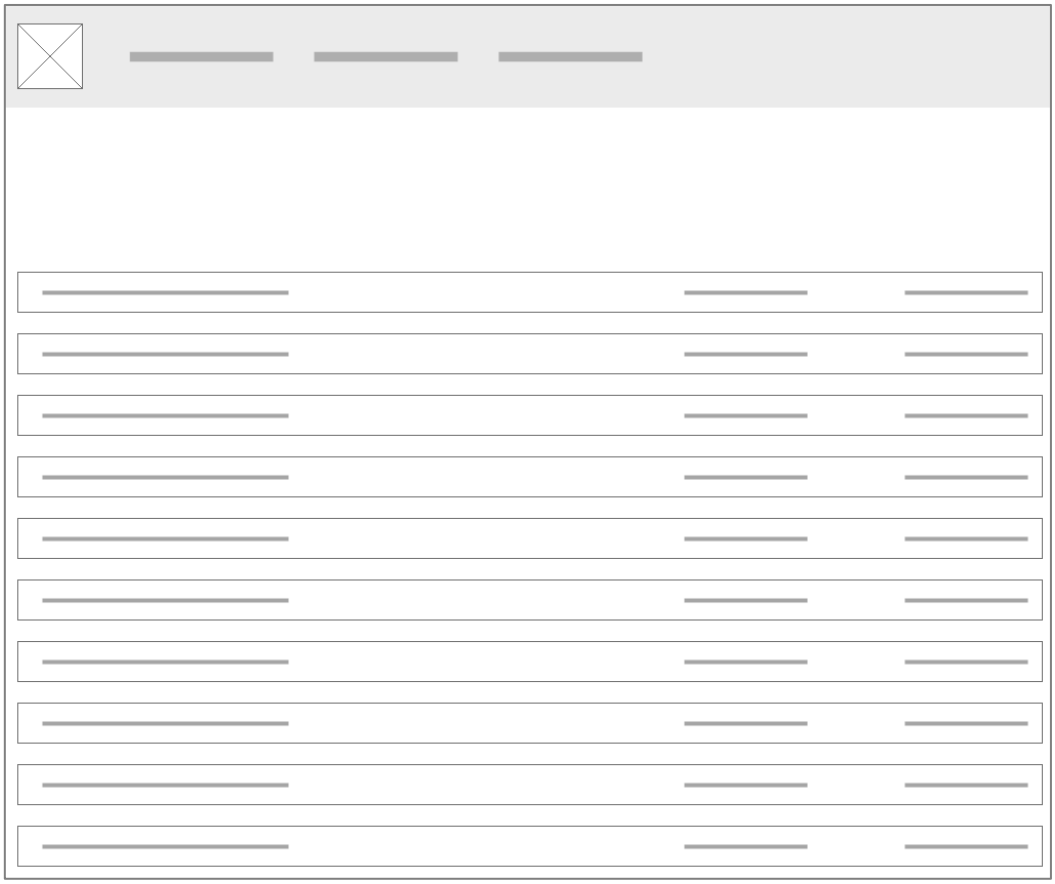



*Figuur 2 Database structuur*

#### 3.1.1.4 Wireframes

Om de inhoud van de webpagina's te bepalen zijn er enkele *low-fidelity* wireframes uitgewerkt op het whiteboard. Hieronder is de lijst van de drie uitgewerkte wireframes met een beschrijving van hun inhoud.

Alle wireframes bevatten bovenaan een navigatie menu. Dit navigatie menu bevat een logo met links daarnaast een lijst van navigatie elementen. In het midden van het hoofscherm is er een lijst waarin elke testrun weergegeven wordt. Voor elke testrun wordt de naam links getoond in het element. De rechterkant van het element bevat de status van de testrun zoals op onderstaande figuur.



*Figuur 3 Hoofscherm wireframe*

Het testrundetailscherm en testresultatenscherm bevatten een header sectie onder het navigatie menu. Dit is te zien op twee onderstaande figuren. Deze sectie bestaat uit twee delen, een linker en rechterdeel. Het linkerdeel bevat de naam en gegevens van de testrun of testcase in het testrundetailscherm en testresultatenscherm, respectievelijk. Het rechterdeel bevat statusindicatoren over de testcase of testrun. Het midden van het testrundetailscherm bevat een lijst met testcases die dezelfde componenten bevat als de lijst van de hoofdpagina zoals in de vorige paragraaf beschreven staat.

*Figuur 4 Testrundetailscherm wireframe*

Structureel is het testresultatenscherm gelijkaardig aan het testrunddetailscherm (zie Figuur 4). Het verschil is dat het testresultatenscherm een zoekbalk bevat tussen de header en lijst van testresultaten. De zoekbalk bevat aan de linkerkant een tekstveld waarin getypt kan worden en rechts een filter waarmee de status van de resultaten gefilterd kan worden zoals te zien is op Figuur 5.

*Figuur 5 Testresultatenscherm wireframe*

### 3.1.2 Fase 1

Gedurende fase 1 lag de focus op de uitwerking van een *minimum viable product* waarin de kernfunctionaliteiten van de applicatie ontwikkeld zijn. Om het ontwikkelingsproces te versnellen was er gebruikgemaakt van bestaande kennis van Spring Boot, Angular en Semantic UI waardoor er minder problemen belemmeringen veroorzaakten tijdens deze fase.

#### 3.1.2.1 Database implementatie

Om de database aan de applicatie te koppelen zijn de volgende *Maven dependencies* toegevoegd aan het project: Spring Data en de MariaDB Java Client.

Door de onderstaande instellingen te plaatsen in het application.properties-bestand kon de verbinding met de database gelegd worden.

```
# Database Configuration
spring.datasource.url=jdbc:mariadb://127.0.0.1:3306/potato
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.hibernate.ddl-auto=validate
```

*Figuur 6 Database configuratie*

Voor het mappen en uitlezen van de gegevens is er gebruikgemaakt van Hibernate waardoor er voor elke database tabel of view een *entity*-klasse gedefinieerd was.

### 3.1.2.2 Successiecalculator API

Om de laatste rekenresultaten te bekomen is in de successiecalculator een nieuw API *endpoint* toegevoegd waarmee de resultaten van een dossier uitgerekend worden. Dit *endpoint* accepteert een dossierbestand en importeert het in het systeem. Na een succesvolle import gaat het systeem de gegevens berekenen. De rekenresultaten bevatten veel gegevens die intern gebruikt worden om keuzes te maken zodat de correcte wetgeving gehanteerd is in de berekeningen. Deze interne gegevens bevatten te veel informatie en moeten gefilterd worden om relevant te zijn. Er is een filter voorzien die een beperkt aantal gegevens exporteert. Deze filter bestaat uit een DTO-klasse waarin de gegevens gekopieerd worden door een *factory*-klasse. Voor het genereren van de JSON-uitvoer is de library GSON gebruikt. GSON laat de ontwikkelaar toe om klassen te serialiseren zonder veel code te schrijven.

Na het uitvoeren van de berekeningen en bij een fout worden de dossiergegevens verwijderd om te voorkomen dat de database snel gevuld geraakt met dezelfde data. Het verwijderen van de gegevens is verwezenlijkt door de rekenfunctie in een *try*-blok te hullen met een *finally*-blok onderaan. Dit *finally*-blok is verantwoordelijk voor het verwijderen van de gegevens.

```
// Calculate the results based on the provided input.
try {
  return resultsService.calculateResults(importedDossier);
}
finally {
  // Force delete after it has been processed.
  hardDeleteDossier(importedDossier);
}
```

*Figuur 7 Rekenmethode met finally-blok*

### 3.1.2.3 Starten van testen

Voor het starten van de testen is er een GET-*endpoint* voorzien om een nieuwe testrun te starten van een extern systeem. Er was gekozen voor een GET-*endpoint* omdat er tijdens fase 1 geen extra informatie naar het start-*endpoint* verzonden moest worden.

#### 3.1.2.4 Vergelijken van resultaten

Het vergelijken van de resultaten bestaat uit twee stappen. In de eerste stap worden de resultaten van de bekomen en opgeslagen JSON-resultatenbestanden omgezet in twee *flattend maps*. Gedurende dit omzettingsproces worden de elementen in de JSON-boomstructuur geprojecteerd tot een platte Java-map. De sleutels van de nieuwe geprojecteerde resultaten zijn de namen van de elementen aaneengehecht met de punt als koppelingsteken.

Stap twee wordt afgehandeld door de *DifferenceMap* van de Guava-*library* zoals beschreven in het onderzoek van fase 0. De resultaten die in de *DifferenceMap* zitten worden hierna opgeslagen in de database.

#### 3.1.2.5 Webapplicatie

Het opzetten van de applicatie ging met behulp van de Angular-CLI. Hierna is met de Node.js package manager (npm) Semantic UI geïnstalleerd.

Semantic UI heeft componenten die een Javascript implementatie hebben en hun eigen state bijhouden. De state van deze componenten kan niet worden bijgehouden door Angular omdat deze niet in hetzelfde ecosysteem geïntegreerd zijn. Om te voorkomen dat alles manueel geïmplementeerd moet worden en toch de componenten te gebruiken is de ng2-semantic-ui *library* gebruikt. Deze was geïnstalleerd door gebruik te maken van de npm-CLI.

#### 3.1.2.6 Communicatie webapplicatie

De communicatie tussen de backend en frontend gebeurt aan de hand van een REST-API.

Voor de API-endpoints zijn er verschillende Controllers voorzien die een @RestController *annotation* bevatten. Deze *controllers* spreken een servicemethode aan die een *optional*-object teruggeeft.

Als het *optional*-object inhoud bevat, dan wordt deze inhoud omgezet naar een *data transfer object* (DTO) klasse om te voorkomen dat te veel informatie in het resultaat komt. De onderstaande figuur bevat voorbeeldcode van een implementatie van de data-*mapping*.

Er is geen authenticatie voorzien in de applicatie omdat de tool enkel intern met VPN-toegang gebruikt kan worden. Daardoor is in tegenstelling tot bij een gewone website, dit minder belangrijk.

```
/**
 * Returns the details of a single test run.
 */
@GetMapping("/{runId}")
public RunDetailedDto get(@PathVariable final UUID runId) throws EntityNotFoundException {
    return runService.findById(runId)
            .map(run -> modelMapper.map(run, RunDetailedDto.class))
            .orElseThrow(() -> new EntityNotFoundException(runId));
}
```

*Figuur 8 Controllermethode met optional-object mapping*

Bij een leeg *optional*-object zal een *exception* opgegooid worden. Deze exception bevat een @ResponseStatus *annotation* die de HTTP-statuscode aangeeft aan het Spring Boot framework. De gebruikte techniek zorgt ervoor dat deze statuscode en *exception* door het framework in het antwoord op het verzoek geplaatst wordt. Daardoor kunnen in webapplicatie gedetailleerde foutmeldingen getoond worden.

#### 3.1.2.7 Problemen

Tijdens het ontwikkelen van het testcase-uploadscherm is er een fout ontdekt in de ng2-semantic-ui *library*. Deze *library* kon niet overweg met de nieuwste versie van Semantic UI [3]. Hierdoor werden de *modals* op de verkeerde plaats in de pagina weergegeven. Door ng2-semantic-ui te upgrade naar de laatste *alpha* versie van de *library* was dit probleem opgelost.

### 3.1.3 Fase 2

Fase 2 was de oppoets- en uitbreidingsfase. Gedurende deze fase hebben er twee uitbreiding plaatsgevonden. De eerste uitbreiding was een wijzigingen van het teststart-*endpoint* zodat Jenkins de applicatie kan aanspreken en nieuwe testenruns kan starten. Ook is er ondersteuning voor meerdere testomgevingen toegevoegd zodat er maar één instantie van de applicatie nodig is.

#### 3.1.3.1 *Endpoint* wijzigingen

Het bestaande GET-*endpoint* is vervangen door een POST-*endpoint* omdat Jenkins extra informatie verstuurd over de *build*. Deze informatie bestaat uit het *build*-nummer, de *build*-URL de GIT-*commit-hash*, *branch* en *repository*-URL ook worden de gebruikersnaam, wachtwoord, omgevings-ID en *endpoint* van het te testen environment mee gestuurd.

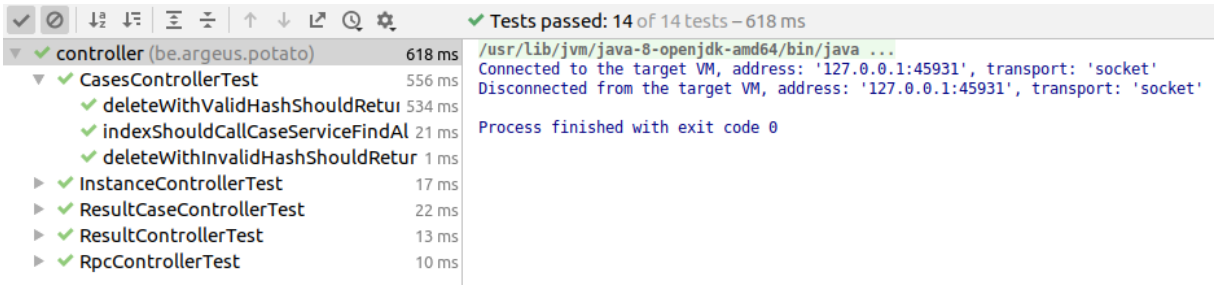
#### 3.1.3.2 Meerderen testomgevingen

Na het uitwerken van fase één was er ontdekt dat de tool geen onderscheid maakte tussen de verschillende testomgevingen. Hierdoor kon de tool enkel gegevens van één testenvironment opslaan per instantie van de applicatie. Het was onpraktisch om voor elke testomgeving een instantie van de applicatie te moeten voorzien dus werd er gekozen om meerdere testomgevingen te ondersteunen.

De oplossing voor dit probleem was een uniek omgevingsnummer toevoegen aan elke testcase en aan het teststart-*endpoint* zodat de applicatie kan detecteren over welke testomgeving het gaat. Het implementeren van de oplossing heeft enkele dagen in beslag genomen. Oorspronkelijk werden de testcases als bestanden opgeslagen op het bestandssysteem zonder extra metadata. Voor het ophalen van de testcases werd een *test ware* map gescand op het bestandsysteem. Bij het uitvoeren van een testrun werden nieuwen testcases weggeschreven naar de database. De koppeling met de records in de database werd gemaakt met de MD5-hash van de geüploade bestanden. Nu worden de testcases aangemaakt bij het uploaden van de bestanden. De oude koppeling is vervangen door de bestandsnamen te hernoemen naar het record-ID van de testcase in de database. Hierdoor kan de metadata gemakkelijk gekoppeld worden aan nieuwe testcases.

### 3.1.3.3 Refactor en unittesten

Eer groot deel van de code is herschreven om de leesbaarheid van de code te verhogen. De SOLID-principes zijn toegepast tijdens het *refactoren* in combinatie met het schrijven van unittests. Onderstaande figuur is een screenshot van de controller testen.



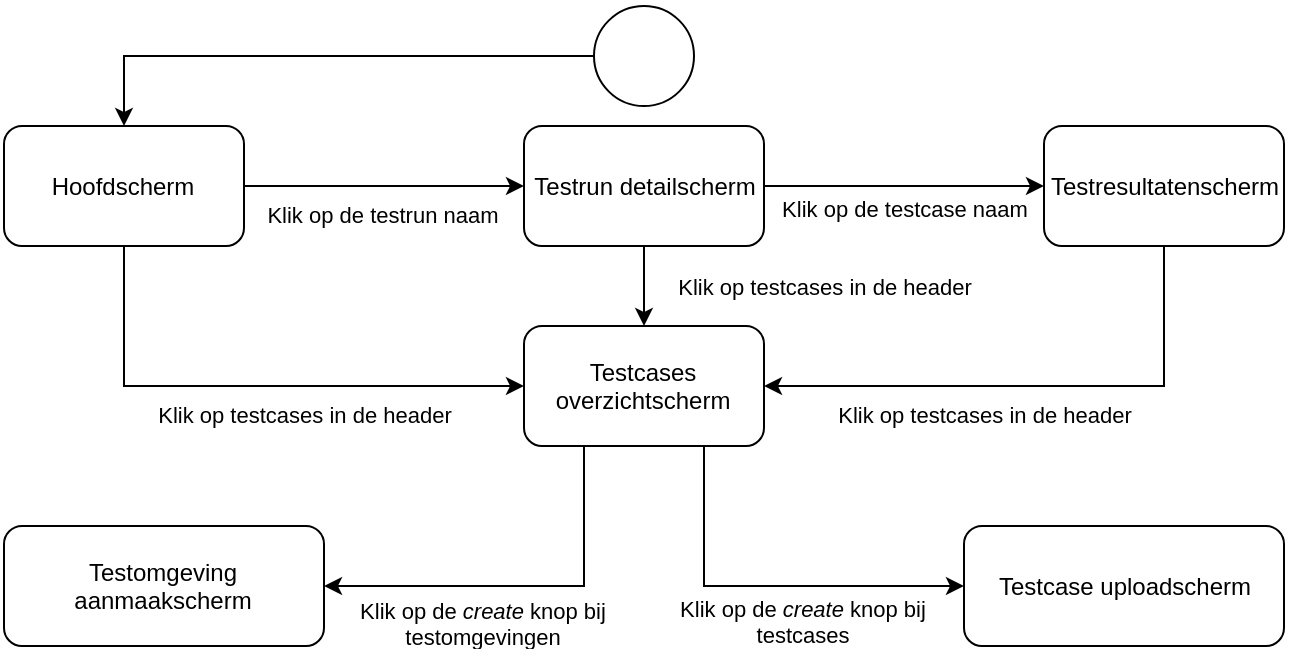

*Figuur 9 Unittesten van alle controllers*

## 3.2 Resultaat

Het resultaat van de stageopdracht wordt weergegeven aan de hand van onderstaande schermen met een korte beschrijving.

### 3.2.1 Schermtoestandsdiagram

Het onderstaande toestandsdiagram bevat de stroom van de schermen in de applicatie.

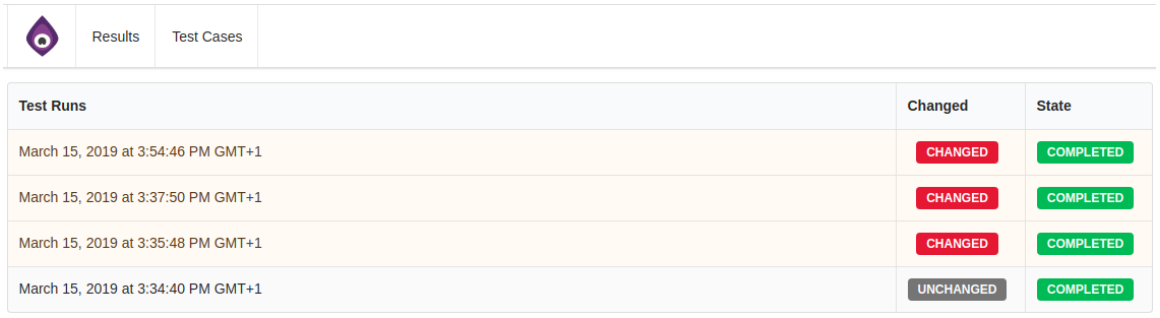



*Figuur 10 Toestandsdiagram van de schermen*

### 3.2.2 Hoofdscherm

Het hoofdscherm is het eerste scherm dat de gebruiker ziet als hij naar de webapplicatie navigeert. Dit scherm bevat een lijst van alle testruns, aflopend gesorteerd op datum zodat de meest recente testrun eerst te zien is.

De testruns bevatten twee statusindicatoren, een om aan te geven of er een resultaat gewijzigd is en een om de status van de testrun weer te geven. De status van de testrun kan *completed*, *running* of *error* zijn, wat voor afgerond, bezig of fout staan respectievelijk.



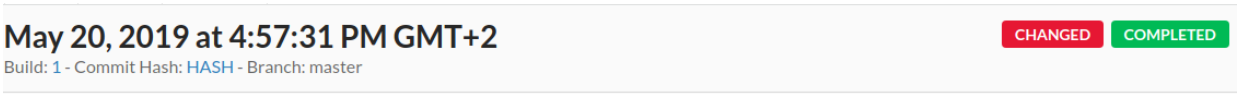

*Figuur 11 Testruns overzicht*

### 3.2.3 Testrundetailscherm

Het testrundetailscherm bevat een lijst van alle testcases die zijn uitgevoerd tijdens de testrun.

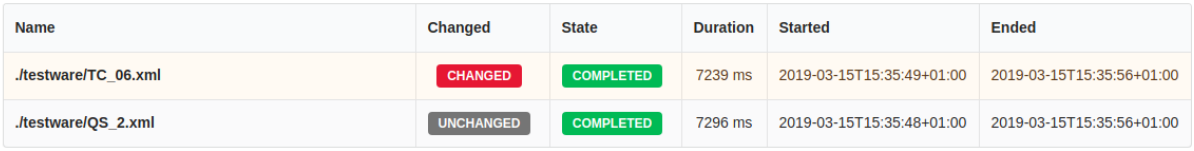
In de linkerkant van de hoofding staan de datum en de versiebeheergegevens van een testrun. De gegevens bestaan uit het *build*-nummer, de *commit hash* en de branch. Ze worden automatisch door Jenkins meegeleverd bij het starten van een testrun. De gebruiker kan op de *commit hash* klikken om deze in detail te bekijken. Bij het klikken wordt de gebruiker doorverwezen naar het versiebeheersysteem.

Rechts staan twee statusindicatoren. De linkerindicator geeft aan of er één van de resultaten van de alle testresultaten is gewijzigd. De rechterindicator geeft de uitvoeringsstatus van de testrun weer.




*Figuur 12 Testrun hoofding*

De lijst met testresultaten bestaat uit een tabel. Als er één van de testcaseresultaten gewijzigd is dan verandert de achtergrondkleur van de gewijzigde testcase naar oranje. Zijn er geen wijzigingen, dan is de achtergrondkleur een grijstint.



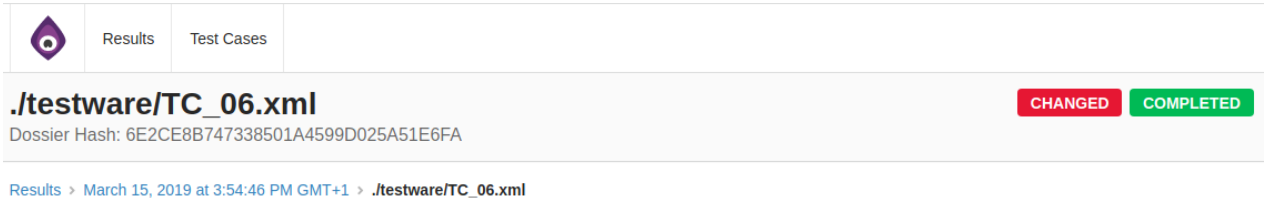

*Figuur 13 Testrun resultatenlijst*

### 3.2.4 Testresultatenscherm

Het testresultatenscherm bestaat uit drie componenten. Een hoofding met informatie over de resultaten, een zoekbalk en een lijst van alle vergeleken testresultaten.

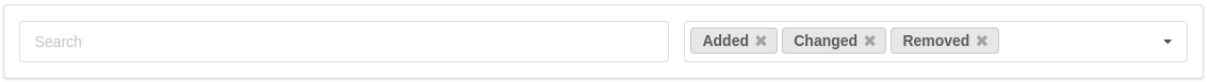
De hoofding bevat de naam van de testcase en dossier-*hash* links. Rechts bevinden zich twee statusindicatoren. De linkse statusindicator geeft aan of er minsten een resultaat gewijzigd is of niet. De rechterstatusindicator geeft aan of de testrun succesvol beëindigd of actief is.

Onder de hoofding is een broodkruimelspoor waar er genavigeerd kan worden naar het hoofdscherm of naar het testrundetailscherm door op *results* of de datum van de testrun te klikken respectievelijk.



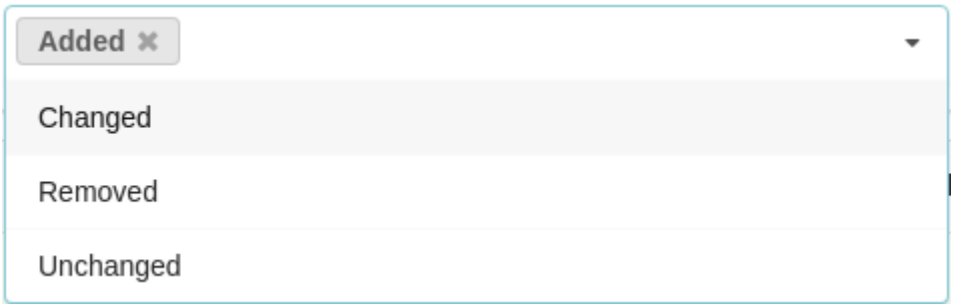

*Figuur 14 Testresultatenhoofding*

De zoekbalk bevat twee filters. Links kan een waarde ingevuld worden waarmee de resultaten gefilterd worden op de sleutel, effectieve en verwachte waarden. Door in het veld te typen worden de resultaten automatisch geüpdatet. Rechts is een meerkeuzemenu om te filteren op de status van de resultaten. Standaard zijn de volgende statussen geselecteerd: *added*, *changed* en *removed*.




*Figuur 15 Testresultatenscherm zoekbalk*

Het meerkeuzemenu kan opengeklapt worden door in het veld te klikken. In dit menu kan de gebruiker een status toevoegen aan de lijst door op de gewenste status te klikken. Statusfilters verwijderen kan door op het kruisje rechts van de status te klikken.




*Figuur 16 Opengeklapt status meerkeuzemenu*

De resultaten worden weergeven in een tabel en zijn gesorteerd op de dossiersleutel. Voor elk resultaat is er een rij voorzien met dossiersleutel, status, huidige en verwachte waardes. Als er een resultaat gewijzigd, toegevoegd of verwijderd is uit de reeks, dan verandert de achtergrondkleur naar oranje, groen en rood respectievelijk.

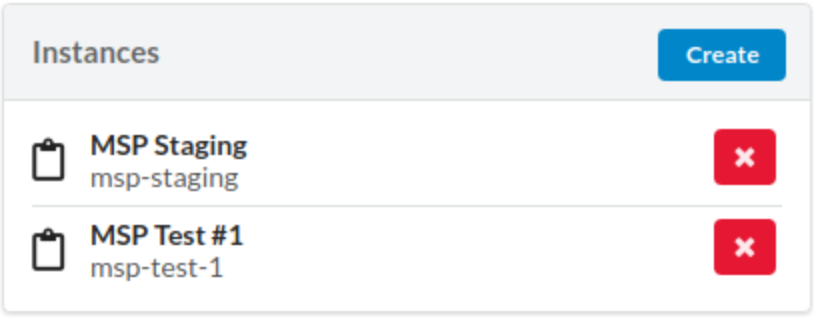
| Key | State | Actual | Expected |
|---|---|---|---|
| dossier.name | CHANGED | TC_06 | TC_06 (EDITED) |
| dossier.type | CHANGED | PRO | PRO (EDITED) |
| succession.0.children.0.children.0.children.1.values.successionValue | CHANGED | -22612.17 | -22612.179 |
| succession.0.children.0.children.0.values.successionValue | CHANGED | 22612.17 | 225612.17 |
| succession.0.children.0.children.1.children.0.values.usufructOwnValue | CHANGED | 113281.92771084337 | 500.0 |
| succession.0.children.0.children.1.children.1.values.fullCommonValue | CHANGED | 1000000.0 | 999.0 |
| succession.0.children.0.description | ADDED | Subtotaal | |
| succession.0.description | ADDED | Totaal | |
| succession.0.potato | REMOVED | | Totaal |

*Figuur 17 Testresultatenlijst*

### 3.2.5 Actieve testcasesscherm

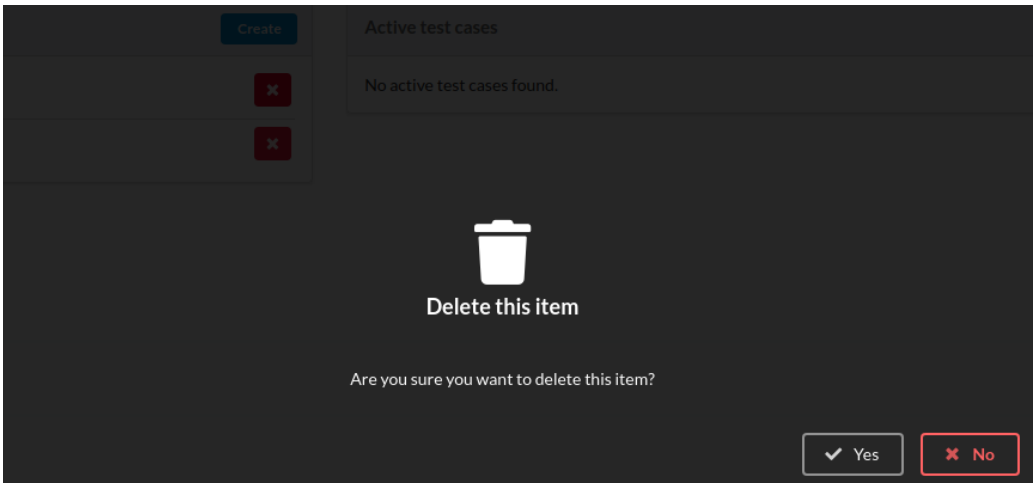
Op dit scherm kan een ontwikkelaar of tester bestaande testcases verwijderen of nieuwe aanmaken en testomgevingen beheren.

Rechts bevat het scherm een lijst van testomgevingen die aanwezig zijn in het systeem. Voor elk testomgeving in de lijst wordt de naam, ID-tekst en een verwijderknop getoond. Bij het klikken op de verwijderknop krijgt de gebruiker een bevestigingsscherm te zien.




*Figuur 18 Testomgevingenlijst*

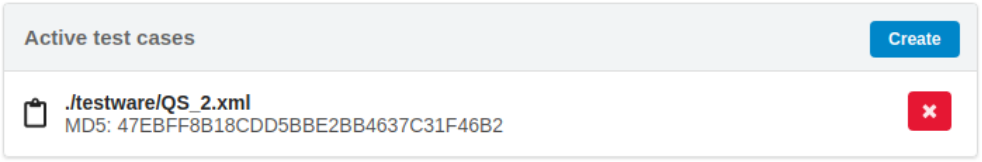
Het bevestigingsscherm bestaat uit een melding die heel de pagina inneemt en twee knoppen. De *yes*-knop met een grijze kleur om de verwijdering te bevestigen en een rode *no*-kop die automatisch gefocust is zodat de melding niet per ongeluk bevestigd kan worden.




*Figuur 19 Bevestigingscherm bij verwijderen van een item*

Rechts van de testomgevingenlijst is er een lijst van testcases waar voor elke testcase de naam en MD5-hash van het dossierbestand weergeven worden.

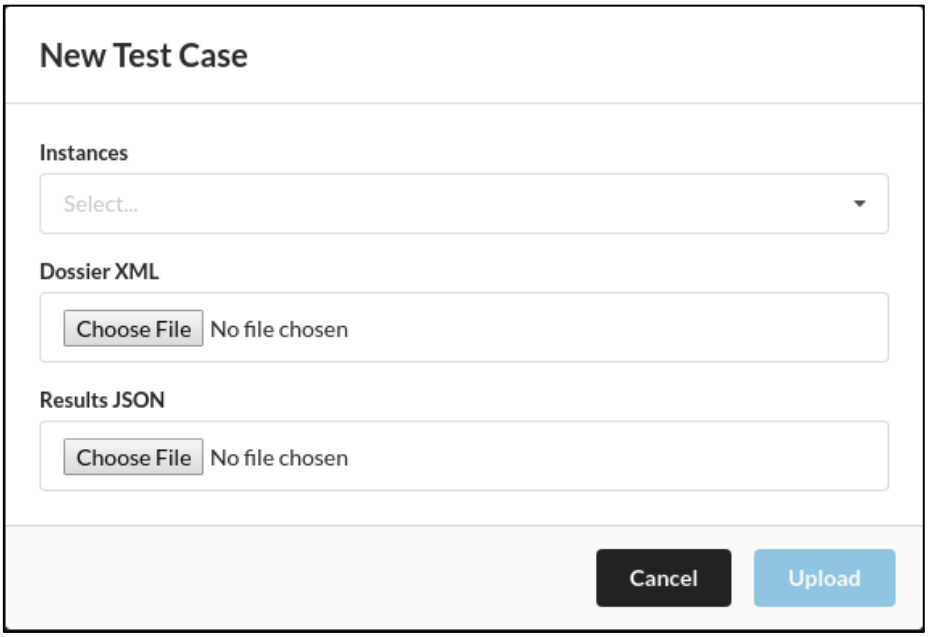
Testcases kunnen worden aangemaakt door op de *create*-knop te klikken. Verwijderen van bestaande testcases kan door op de rode knop te klikken rechts van de testcase. Bij het verwijderen van een testcase blijft de historiek in de testruns bewaard.




*Figuur 20 Testcases overzicht*

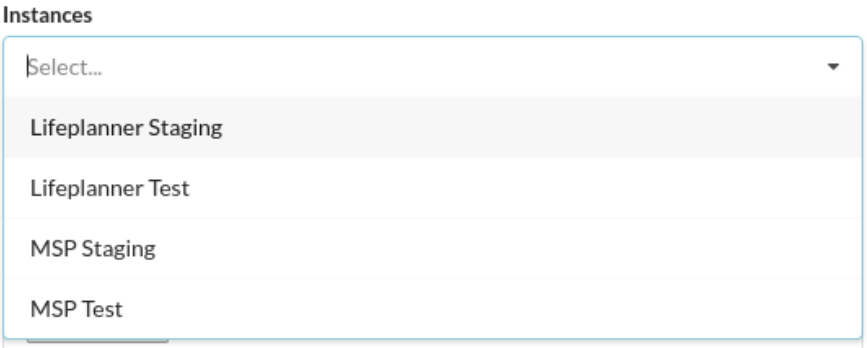
### 3.2.6 Testcase uploadscherm

Om een nieuwe testcase toe te voegen moeten twee bestanden geüpload worden. Een dossier als xml-bestand en de resultaten in een JSON-bestand. Onderstaande figuur is een representatie van het upload *modal*. Er kan een keuze gemaakt worden uit meerdere testomgevingen. De uploadknop is enkel aanklikbaar als er twee geldige bestanden en minstens één testomgeving geselecteerd zijn.




*Figuur 21 Testcase uploadscherm*

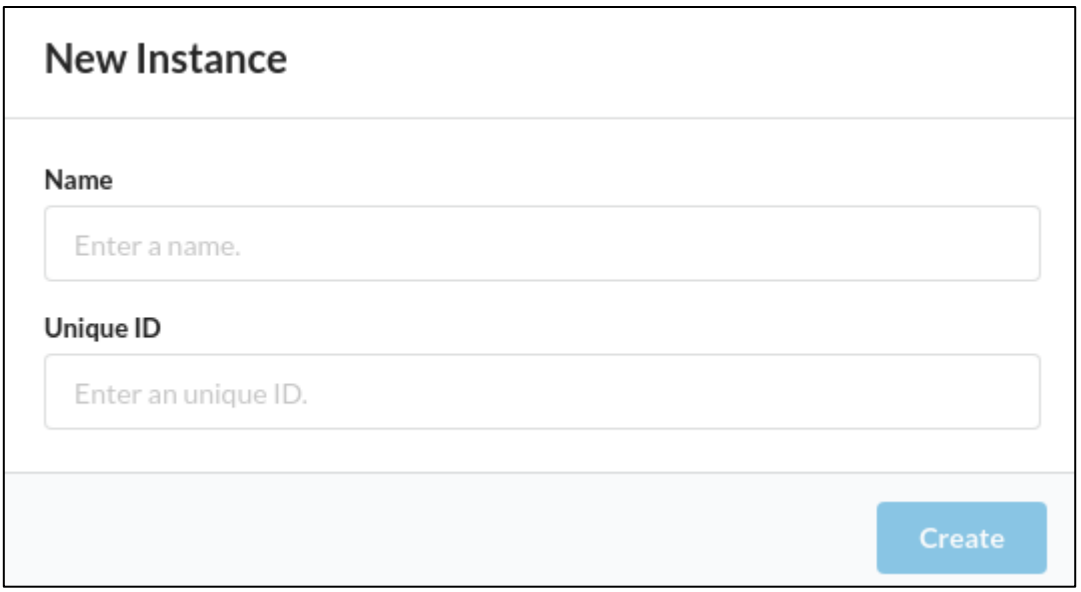
De gebruiker kan meerdere testomgevingen selecteren door op het vak onder *instances* te klikken. Er zal dan een lijst openen waarin items geselecteerd kunnen worden. In dit veld kan er getypt worden om de resultaten te filteren. Een geselecteerde testomgeving kan uit de lijst verwijderd worden door op het kruisje rechts van een geselecteerde testomgeving te klikken.




*Figuur 22 Opengeklapt menu met testomgevingen*

### 3.2.7 Testomgeving aanmaakscherm

Op dit scherm kunnen nieuwe testomgevingen worden toegevoegd. Het scherm bestaat uit een pop-up met een formulier waarin een unieke sleutel en naam ingevuld moeten worden. De *create* knop is uitgeschakeld als beide velden niet ingevuld zijn.




*Figuur 23 Testomgeving aanmaakscherm*

# 4 Besluit

Het doel van de stageopdracht was het automatisch detecteren en visualiseren van veranderingen in de resultaten van de successiecalculator. Dit is gelukt. Er is een vangnet gecreëerd dat in de *continuous integration* van de successiecalculator geïntegreerd is. De tool is ontwikkeld met onderhoud en uitbreidingen in het achterhoofd.

Met de applicatie kunnen de ontwikkelaars en testers van Argeüs hun regressietesten automatiseren door resultaten te exporteren uit de successiecalculator. Dit kan met behulp van de nieuwe integratie die voorzien is in deze stageopdracht. Hierdoor zijn veel manuele stappen uit het testproces geautomatiseerd. Ook kan deze tool gebruikt worden met andere applicaties als ze resultaten produceren in JSON-formaat.

Met deze applicatie kan Argeüs garanderen aan hun klanten dat de resultaten van de formules niet gewijzigd zijn bij het uitrollen van een nieuwe update. Als er toch wijzigingen zijn, dan kan de tool deze vroegtijdig detecteren en in detail rapporteren.

Als uitbreiding op de stageopdracht zou er een reviewsysteem toegevoegd kunnen worden. Met dit systeem zouden ontwikkelaars of testers de resultaten kunnen nakijken en aanpassen via de webapplicatie.

# 5 Persoonlijke reflectie

Ik ben twaalf weken bezig geweest met het ontwikkelen van deze stageopdracht.

Door tijd te reserveren voor een klein onderzoek naar bestaande *libraries* heb ik veel bijgeleerd en veel tijd uitgespaard. Dit heb ik gedaan tijdens fase 0 bij de uitwerking van mijn stageopdracht. Een persoonlijk doel tijdens mijn stageopdracht was om zoveel mogelijk code van opensourceprojecten te gebruiken en zo weinig mogelijk code zelf te schrijven. Hierdoor moest ik mij verdiepen in de bestaande frameworks die ik kende en heb ik veel nieuwe kennis opgedaan. De opgedane kennis heb ik toegepast in dit en nieuwe projecten.

Ik vind het jammer dat ik tijdens de initiële ontwikkeling weinig tijd gespendeerd had aan het schrijven van unittests. Er waren een zeer beperkt aantal testen geschreven waardoor ik een klein vangnet had toen ik de applicatie ging herstructureren in fase 2. Ik had niet verwacht dat testen mij konden helpen bij het herstructureren van een applicatie. In de toekomst ga ik test-driven development beter toepassen in nieuwe projecten.

Er is veel tijd gestoken in het vergelijken van de resultaten. Vooral in de koppeling tussen de tool en de successiecalculator. Ik heb met *multithreading* proberen te werken zodat meerdere testcases gelijktijdig uitgevoerd kunnen worden. Dit was een uitdaging voor mij omdat ik enkel schoolkennis had over threading. Uiteindelijk is dit gelukt en heeft het mijn kennis over *multithreading* vergroot.

Ik ben tevreden van het opgeleverde eindresultaat.

# II. Onderzoeksopties

## 1 Vraagstelling

Argeüs is een pionier op het vlak van financiële software en is constant op zoek naar nieuwe technologieën en methodieken die de ontwikkeling van haar software kunnen ondersteunen.

De huidige applicaties zijn ontwikkeld zonder performantie in het achterhoofd. Tot recent werden er geen performantiemetingen uitgevoerd op de testenvironments. Tijdens het eerste gebruik is ontdekt dat de performantie goed genoeg is voor de vereisten van de huidige gebruikers. Bij het verhogen van het aantal gebruikers neemt de performantie sterk af.

De applicatie is gebouwd in een monolithische-architectuur die niet goed schaalt naar meerdere omgevingen omdat de berekeningen afhankelijk zijn van veel gegevens die opgeslagen zijn in een centrale database.

Er zijn plannen om *calculation as a service* op grote schaal aan te bieden aan de grote spelers in de financiële markt. Door deze plannen moet de capaciteit van de huidige API's, die ontwikkeld zijn voor een niche doelgroep, verhoogd worden.

Recent is door een ontwikkelaar van Argeüs ontdekt dat de programmeertaal Rust de focus op performantie legt. De innovatiemanager heeft de vraag gesteld of de programmeertaal Rust als alternatief voor Java gebruikt kan worden voor het bouwen van applicaties met een REST-API. Hij wil graag weten of een implementatie in Rust performanter en makkelijker te gebruiken is in de productie dan de huidige Java-implementatie.

# 2 Methode van onderzoek

In het onderzoek wordt er een literatuurstudie gedaan. Vervolgens is er een implementatie van een REST-API in Rust en Java die het verworven kapitaal per jaar met een jaarlijks vast inkomen berekent als toegepast onderzoek.

Er is gekozen om een literatuurstudie te doen naar de sterktes en zwaktes van de programmeertaal Rust. Java wordt gebruikt om Rust te vergelijken ten opzichte van een andere programmeertaal omdat Argeüs deze gebruikt om hun producten te ontwikkelen.

In het toegepast onderzoek zullen er benchmarks worden uitgevoerd op twee verschillende processorarchitecturen: ARM en Intel x86-64. Deze benchmarks worden uitgevoerd op twee implementaties van dezelfde REST-API. Deze REST-API's zijn vanaf nul in de verschillende programmeertalen geschreven. Hierbij is gebruikgemaakt van twee vooraf gekozen frameworks omdat de focus op het uitwerken van een werkende *proof of concept* ligt.

Voor de Java-implementatie is er gebruikgemaakt van Jersey omdat dit framework ook gebruikt wordt in de successiecalculator. Voor de Rust-implementatie is er gekozen voor Rocket.rs omdat dit framework actief in ontwikkeling is, makkelijk te configureren is en het vertrouwen van de community heeft.

De performantiemetingen van de REST-API's worden gemeten aan de hand de reactietijd op de ARM- en Intel x86-64-architectuur. Argeüs zorgt voor een Raspberry Pi 3B+ zodat de ARM-architectuur gecoverd is.

# 3 Resultaten

## 3.1 Literatuurstudie

### 3.1.1 The Rust Programming Language

Dit is het officiële boek om de Rust programmeertaal te leren [4] (het Rust boek). Het boek is gratis digitaal beschikbaar en bestaat in paperback formaat. Het is geschreven door Steve Klabnik, Carol Nichols en bevat bijdragen vanuit de Rust community. Carol Nichols is een van de Rust Core Team members en is betrokken bij de ontwikkeling van de programmeertaal. Beginners kunnen met dit boek stapsgewijs Rust leren.

Dit boek bevat verschillende kernconcepten van Rust met uitleg van de werking. De basis van het *ownership and borrwing*-concept is uitgewerkt in rubriek 3.2.1.1.

### 3.1.2 Rust in Production at Figma

Deze sectie bevat een samenvatting van de casestudie *Rust in Production at Figma*. *In de casestudie van het bedrijf Figma werd ontdekt* dat het bedrijf een schaalprobleem had in haar software. [5]

Het probleem werd veroorzaakt door een sterke groei aan gebruikers. De servers konden de load niet bijhouden als er door veel gebruikers gelijktijdig aan één project gewerkt werd. In deze casestudie tonen ze de resultaten van het gebruik van Rust in hun product om de performantie te verbeteren.

Het bedrijf was van plan om hun serverarchitectuur te herschrijven in Rust om het schaalprobleem op te lossen. Later is het hierop teruggekomen en is er een cruciaal onderdeel van de backend vervangen. Onderstaande figuur bevat de veranderingen in de piekmetingen bij Figma voor en na het implementeren van Rust.

| Metric | Old server | | New server | Improvement |
|---|---|---|---|---|
| Peak average per-worker memory usage | 4.2gb | ⟶ | 1.1gb | 3.8x smaller |
| Peak average per-machine CPU usage | 24% | ⟶ | 4% | 6x smaller |
| Peak average file serve time | 2s | ⟶ | 0.2s | 10x faster |
| Peak worst-case save time | 82s | ⟶ | 5s | 16.4x faster |

*Figuur 24 Verandering in piekmetingen bij Figma voor en na het implementeren van Rust*

In dit artikel werden de voor- en nadelen van het implementeren van Rust op een rijtje gezet. De voordelen waren laag geheugengebruik, geweldige performantie, goede tools en gebruiksvriendelijke foutmeldingen. De nadelen: de *lifetimes* van variabelen kunnen verwarrend zijn, fouten zijn moeilijk te debuggen, veel *libraries* zijn nog in een vroeg ontwikkelingsstadium en een standaardisatie ontbreekt.

### 3.1.3 Rust Case Study: Community makes Rust an easy choice for npm

Deze sectie bevat een samenvatting van een casestudie geschreven door de npm-ontwikkelaars over hoe Rust hun CPU-schaalprobleem heeft opgelost. Over de afgelopen jaren is het gebruik van npm exponentieel gegroeid. Hierdoor zag men bij npm potentiële CPU-problemen in de authenticatie service die de controle voor het publiceren van nieuwe packages afhandelt. Hun oude systeem, geschreven in Javascript, was aan vervanging toe. Het npm-team zag deze opportuniteit om de code en de performantie te verbeteren. [6]

De programmeertalen C, C++ en Java werden meteen overboord gegooid omdat deze niet voldeden aan de volgende criteria:

> *"Given the criteria that the programming language chosen should be:*
>
> ● *Memory safe*
>
> ● *Compile to a standalone and easily deployable binary*
>
> ● *Consistently outperform JavaScript*
>
> *[..]." [6]*

Uiteindelijk bleven Go en Rust over. Het npm-team besloot de authenticatie service te herschrijven in Node.js, Go en Rust. De Node.js variant nam een uur in beslag. De Go variant twee dagen. De Rust variant nam meer tijd in beslag dan de Node.js en Go varianten. Het duurde ongeveer een week om de applicatie te implementeren. Uiteindelijk is er gekozen voor Rust omdat het npm-team het dependentiebeheer in Rust interessanter vond dan die van Go en omdat de Rust community zeer behulpzaam was.

Het dependentiebeheersysteem heeft het de ontwikkelaars van npm gemakkelijk gemaakt om de service te ontwikkelen op hun eigen ontwikkelomgeving. Ook voorkomt het systeem versiebeheerproblemen omdat de dependenties beschikbaar zijn op één centraalpunt.

Sinds het uitrollen van deze service heeft deze nog geen problemen veroorzaakt. Het onderhoud van de applicatie vraagt amper aandacht omdat deze service weinig operationele problemen heeft.

### 3.1.4 Gemeenschappelijk

#### 3.1.4.1 Performantie

Uit alle bronnen kan afgeleid worden dat Rust een performante taal is die compileert naar een binair uitvoerbaar bestand. Dit gebeurt aan de hand van LLVM zoals beschreven staat in *Rust in Production at Figma*. LLVM is een compiler infrastructuur die een programmeertaal omzet naar machinecode. [7]

> *"Rust definitely delivered on its promise of optimal performance, both because it can take advantage of all of LLVM's optimizations and because the language itself is designed with performance in mind." [5]*

In *The Rust Programming Language* is er rubriek gewijd aan het doelpubliek. Hierin wordt beschreven dat Rust voor mensen is die snelheid en stabiliteit willen. Ook is er een beknopte uitleg van hoe dit verwezenlijkt wordt in de praktijk:

> *"Rust is for people who crave speed and stability in a language. By speed, we mean the speed of the programs that you can create with Rust and the speed at which Rust lets you write them. [..] By striving for zero-cost abstractions, higher-level features that compile to lower-level code as fast as code written manually, Rust endeavors to make safe code be fast code as well*
>
> *[..] Rust's greatest ambition is to eliminate the trade-offs that programmers have accepted for decades by providing safety and productivity, speed and ergonomics." [8]*

#### 3.1.4.2 Laag geheugengebruik

Naast het efficiënt gebruik van processor maakt Rust ook efficiënt gebruikt van het RAM-geheugen. Dit staat vermeld in de conclusie van de casestudie van npm: "It keeps resource usage low without the possibility of compromising memory safety." [6]. In de casestudie van Figma staat de performantie in de lijst van positieve eigenschapen van Rust met de volgende beschrijving:

> *"Rust combines fine-grained control over memory layout with the lack of a GC and has a very minimal standard library. It used so little memory that it was actually practical to just start a separate Rust process for every document." [5]*

In de casestudie van Figma en het Rust boek kaarten ze ook aan dat Rust geen *garbage collector* heeft. Rubriek 3.2.2.3 bevat meer informatie over de *garbage collector* in Rust.

#### 3.1.4.3 Solide toolchain en dependentiebeheer

Rust heeft verschillende command-line-interfaces die gebruik kunnen worden tijdens het ontwikkelen en onderhouden van applicaties geschreven in Rust. De meest gebruikte CLI's zijn Rustup en Cargo. De rubrieken 3.2.1.3 en 3.2.1.2 bevatten informatie over Rustup en Cargo respectievelijk.

Ook heeft Rust een solide dependentiebeheer. Het is makkelijk om verschillende dependenties toe te voegen aan een Rust project zonder dat het problemen geeft bij een update. In de casestudie van npm was het solide dependentiebeheer een belangrijke factor voor het kiezen voor Rust. In de casestudies wordt er aangekaart dat Rust nog te weinig gestandaardiseerde *libraries* heeft. Ondanks dit probleem wordt er lof uitgesproken over Rust. Een citaat uit casestudie van npm: "As a young language, Rust doesn't yet have industry-standard libraries and best practices for these purposes, but hopefully will in the future." [6]

#### 3.1.4.4 Vriendelijke community en goede toekomst

Het Rust boek bevat feedback van community leden. In de casestudies wordt de ondersteuning van de community vermeld. Meer informatie over de community is te vinden in rubriek 3.2.2.1. Naast de vriendelijke community wordt Rust positief onthaald bij de bedrijven waar de casestudies plaatsvonden. De bedrijven verzekeren dat Rust een positieve toekomst tegemoet gaat.

### 3.1.5 Verschillen

De verschillen tussen de bronnen zitten in de casestudie van Figma.

Het eerste verschil is een probleem dat de ontwikkelaars ondervonden tijdens het uitwerken van een Rust programma. Het *lifetime* systeem van *ownership en borrowing* systeem is verwarrend. De *lifetimes* is een extra check in de compiler om *dangling* pointers te voorkomen. [9]

Ook heeft Rust volgens de casestudy goede foutmeldingen waardoor Rust leren makkelijker is. Ondanks dat er goede foutmeldingen zijn, is het moeilijk om fouten te debuggen zoals vermeld in de paper:

> *"Error-handling in Rust is intended to be done by returning a value called "Result" that can represent either success or failure. Unlike with exceptions, creating an error value in Rust does not capture a stack trace so any stack traces you get are for the code that reported the error instead of the code that caused the error." [5]*

Het laatste verschil is dat asynchroon programmeren in Rust moeilijk is. Asynchroon programmeren in Rust bestaat niet in de programmeertaal. Dit is een gekend probleem en er is een implementatie in de aanmaak. [10] Er is een alternatief dat op asynchroon programmeren lijkt: *the futures API*. Bij Figma vonden ze deze API te complex dus hebben ze hun netwerking in Node.js gehouden met de volgende oplossing:

*“Instead of going all-in on Rust, we decided to keep the network handling in node.js for now. The node.js process creates a separate Rust child process per document and communicates with it using a message-based protocol over stdin and stdout. All network traffic is passed between processes using these messages.” [5]*

### 3.1.6 Conclusie

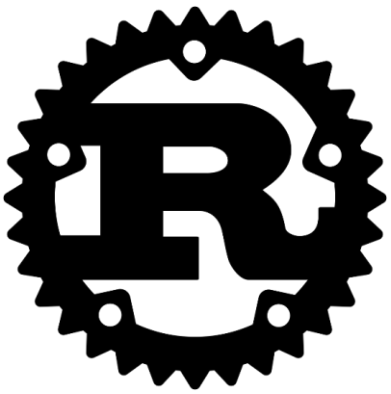
Uit de bronnen kan worden geconcludeerd dat de Rust compiler performante applicaties produceert met een laag geheugengebruik. In de casestudies zijn applicaties vervangen die schaalproblemen veroorzaakten door veel processorgebruik. Ook kan uit de casestudies afgeleid worden dat Rust een sterke community heeft waaraan vragen in verschillende communicatiekanalen gesteld kunnen worden. De tools en het dependentiebeheersysteem maken het makkelijk om applicaties in Rust te ontwikkelen in verschillende ontwikkelingsomgevingen.

De casestudie van npm bevat voornamelijk positieve punten over Rust. Het enige nadeel in deze casestudie is dat de standaardisatie van *design patterns* en *best practices* beperkt zijn. De sterktes en zwaktes komen overeen met die van de casestudie van Figma. In de casestudie van Figma worden er meer probleempunten aangekaart onder anderen dat debuggen en asynchroon programmeren nog niet op punt staan

Ondanks dat Rust een jonge programmeertaal is, heeft hij veel groeipotentie en zien npm en Figma een goede toekomst in Rust. Rust is een goede kandidaat om te introduceren als een applicatie performant en een laag geheugengebruik moet hebben. Onderstaande deskresearch zal bepalen of Rust nuttig is voor Argeüs.

## 3.2 Deskresearch

### 3.2.1 Wat is Rust?



*Figuur 25 Rust logo*

Rust is een systeemprogrammeertaal oorspronkelijk ontwikkeld als een persoonlijk project van Graydon Hoare met de focus op veiligheid, snelheid en *concurrency*. In 2009 is de Mozilla Foundation sponsor geworden van het Rust project. Een jaar later, in 2010 maakte Mozilla Rust bekend aan het publiek. Daarna is de taal nog in ontwikkelingsfase gebleven tot de eerste stabiele versie 1.0 uitgebracht was op 15 mei 2015.

Sinds de 1.0 release wordt er maandelijks minstens één nieuwe functionaliteit toegevoegd aan de stabiele versie van Rust. De nieuwe functionaliteit is achterwaarts compatibel met bestaande Rust code zodat oude projecten blijven werken met de nieuwste versie van de compiler.

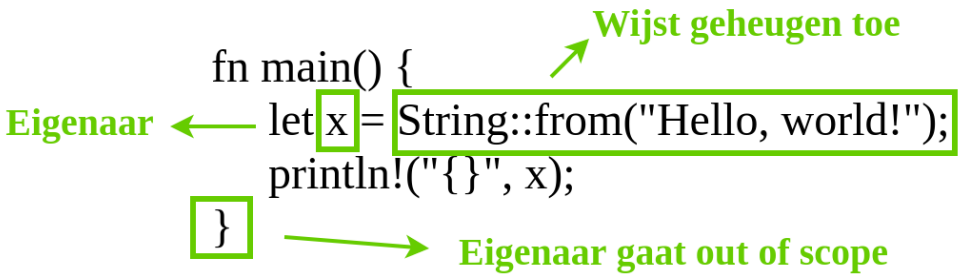
Eind december 2018 heeft het Rust *core team* een nieuwe editie van Rust gelanceerd genaamd: Rust 2018. Dit is de eerste grote wijziging sinds 1.0 waarvan de editienaam Rust 2015 is.

#### 3.2.1.1 Ownership and borrowing

Alle computerapplicaties doen aan een vorm van geheugenbeheer zolang ze actief zijn. Sommige programmeertalen maken gebruik van een *garbage collector* die regelmatig niet meer gebruikt geheugen vrijmaakt. In andere programmeertalen moet de programmeur zelf aan geheugenbeheer doen door expliciet een blok geheugen te gaan verzamelen en vrij te geven wanneer nodig. Rust heeft een uniek concept voor geheugenbeheer genaamd het *ownership*-systeem. Dit systeem is een kernfeature van Rust om *memory safety* te garanderen. Het systeem gaat tijdens de compileertijd het geheugen uitlijnen door gebruik te maken van de onderstaande regels zoals beschrijven staat in *The Rust Programming Language* [11]:

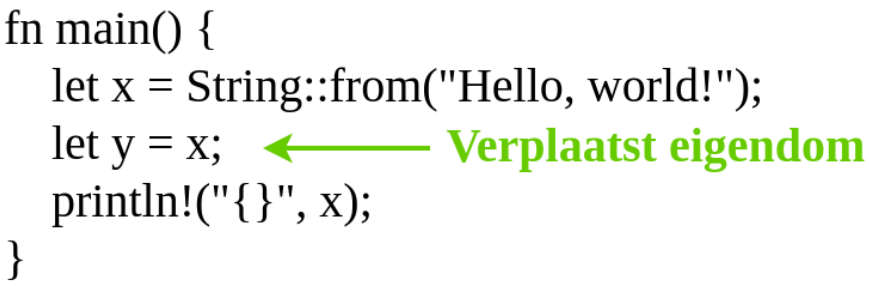
- Elke waarde in Rust heeft een variabele die de eigenaar genoemd wordt.
- Er kan slechts één eigenaar tegelijk zijn.
- Als de eigenaar out of scope gaat, dan wordt de waarde verwijderd.

In de onderstaande figuur bevat een voorbeeld van hoe het *ownership*-systeem werkt en geheugen toewijst. In de code wordt er een String-object aangemaakt in het *heap*-geheugen en de eigendom toegewezen aan de variabele x. Als de variabele x de scope verlaat, dan wordt het geheugen van het String-object vrijgegeven.



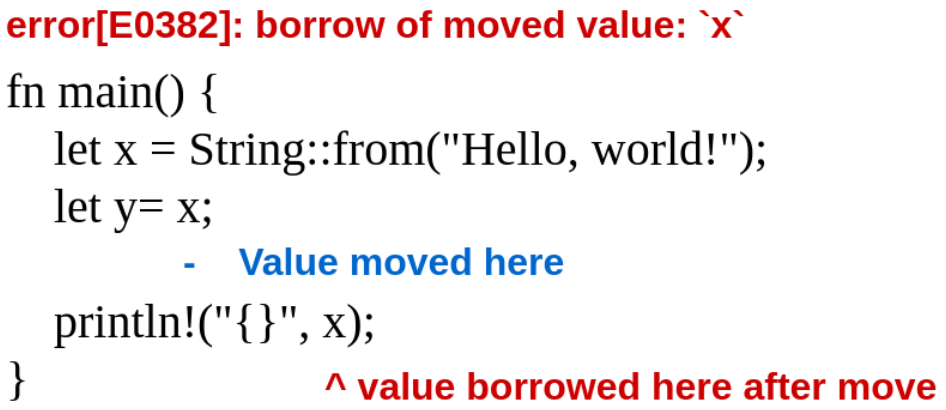

*Figuur 26 Ownership voorbeeld*

Het is mogelijk om de eigendom van een waarde over te dragen naar een andere variabele. Dit kan door een nieuwe variabele te declareren en de voorafgaande variabele hieraan toe te kennen. Bij het toekennen van een variabelen zal deze de nieuwe eigenaar worden van de waarden. Hierdoor wordt de waarde verplaatst van de oude variabele naar de nieuwe omdat er maar één variabele eigenaar kan zijn van een waarde om te voldoen aan de regels.



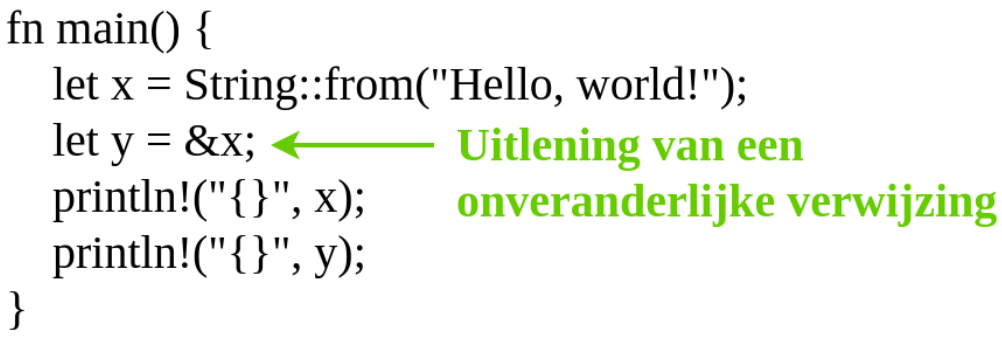

*Figuur 27 Voorbeeld van eigendom doorgeven in Rust*

De bovenstaande code compileert niet omdat de eigenaar doorgegeven is van de variabele x naar y waardoor de variabele x geen eigendom meer heeft. De compiler detecteert dit en zal een foutmelding geven zoals in de onderstaande figuur.




*Figuur 28 Foutmelding "borrow of moved value"*

Standaard wordt de eigendom van een waarde overgedragen naar de functie parameters bij het aanroepen van een functie. Soms moet de eigendom niet overgedragen worden naar een andere functie omdat de waarde later nog gebruikt moet worden. Als de eigendom niet overgedragen mag worden, dan is het mogelijk om een referentie naar deze waarde uit te lenen aan een functie. Dit is het *borrow*-systeem. Dit systeem maakt het mogelijk om een referentie naar een waarden tijdelijk uit te lenen en zo *data races* te vermijden. Om het uitleningssysteem te laten werken zijn de volgende twee regels [12] van toepassing: Onveranderlijke referenties kunnen oneindig aantal keren uitgeleend worden. Veranderlijke referenties kunnen maximaal eenmaal uitgeleend worden.




*Figuur 29 Uitlening van een onveranderlijke verwijzing in Rust*

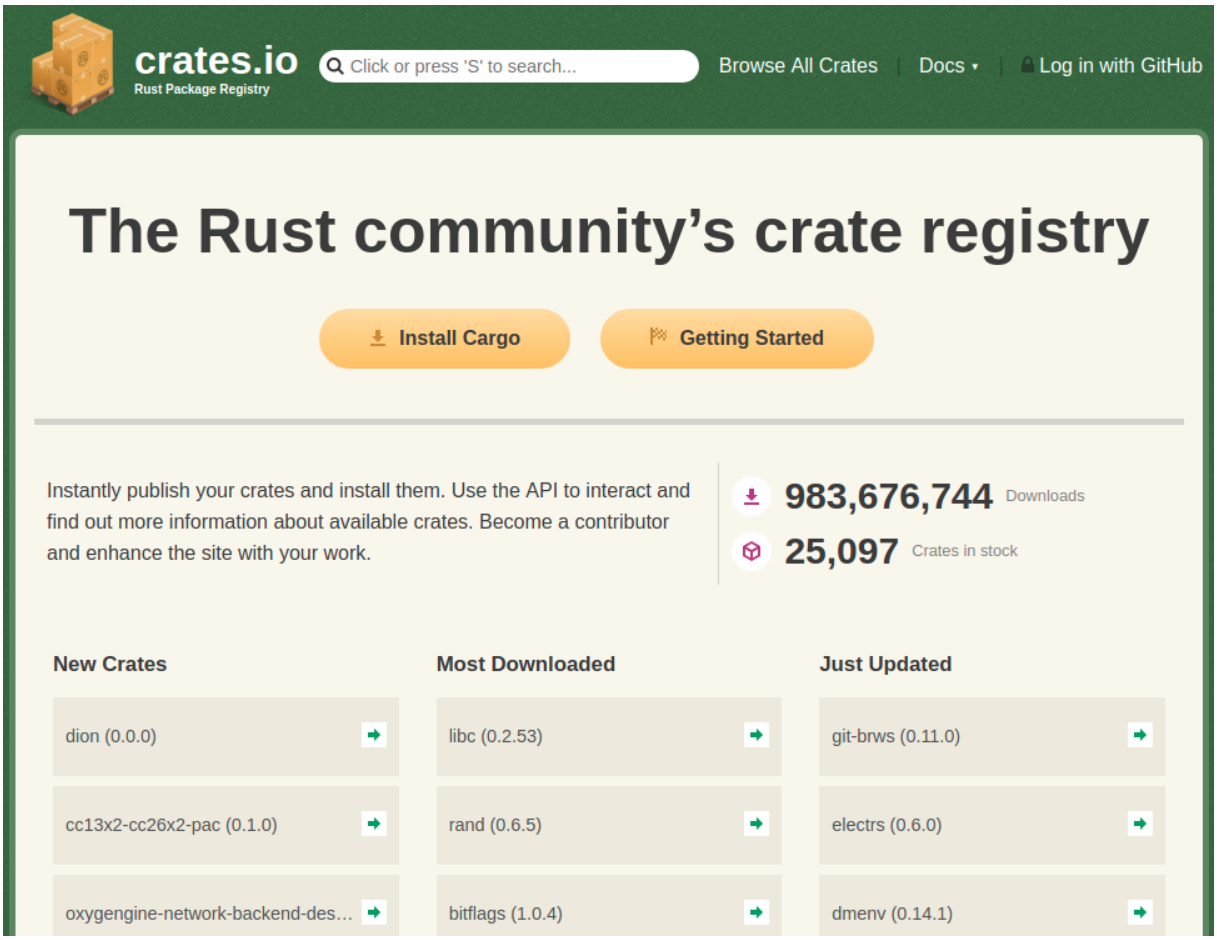
Dit is de basis van het *ownership and borrowing*-systeem. De code moet voldoen aan de *ownership*- en *borrow*-regels om te compileren anders wordt er een foutmelding weergegeven en compileert de code niet.

#### 3.2.1.2 Rustup

Rustup is CLI die verantwoordelijk is voor het installeren en updaten van de Rust programmeertaal. Ook maakt deze CLI het mogelijk om makkelijk te wisselen tussen de *stable*, beta en *nightly* Rust-compilers en maakt het *cross-compiling* van de bekende platformen simpeler. [13]

#### 3.2.1.3 Cargo

Cargo is een CLI die de *dependencies* in een Rust project beheerd. De *dependencies* van een Rust-project worden *packages* genoemd. *Packages* kunnen worden aangemaakt, gedownload en geregistreerd door ontwikkelaars op de *Rust Package Registry* website genaamd crates.io. Op deze website kan er door alle Rust-*crates* gezocht en gebladerd worden. [14]




*Figuur 30 Crates.io homepage*

### 3.2.2 Sterktes

#### 3.2.2.1 Sterke community

In alle bronnen wordt vermeld dat Rust een geweldige community heeft. In de casestudie van npm wordt de community aangehaald als één van de beslissingsfactoren om te kiezen voor Rust:

> *"npm called out the Rust community as a positive factor in the decision-making process. Particular aspects they find valuable are the Rust community's inclusivity, friendliness, and solid processes for making difficult technical decisions. These aspects made learning Rust and developing the Rust solution easier, and assured them that the language will continue to improve in a healthy, sustainable fashion." [6]*

Er zijn verschillende manieren om met community leden in contact te komen. Dit kan via de Rust fora of via de verschillende chatplatformen waaronder Discord. Deze platformen staan vermeld op de Rust website [8]. Er wordt veel gebruikgemaakt van deze communicatiekanalen. Op enkele jaren tijd zijn er verschillende conferenties ontstaan over hele de wereld die over Rust gaan. In 2019 stonden de onderstaande evenementen op de agenda [15].

*Tabel 1 Lijst van aangekondigde Rust evenementen in 2019*

| Naam | Datum | Locatie |
|---|---|---|
| **FOSDEM Rust Room** | **02/02/2019 – 03/03/2019** | **Brussel, België** |
| Rust Latam | 29/03/2019 - 30/03/2019 | Montevideo, Uruguay |
| RustCon Asia | 20/04/2019 - 30/04/2019 | Peking, China |
| Oxidize | 26/04/2019 - 29/04/2019 | Berlijn, Duitsland |
| RustLab | 28/06/2019 - 29/06/2019 | Florence, Italië |
| RustConf | 22/08/2019 - 23/08/2019 | Portland, Oregon, Verenigde Staten |
| Colorado Gold Rust | 20/09/2019 - 21/09/2019 | Denver, Colorado, Verenigde Staten |
| Rust Belt Rust | 18/10/2019 - 19/10/2019 | Dayton, Ohio, Verenigde Staten |

In België wordt jaarlijks FOSDEM georganiseerd aan de Vrije Universiteit Brussel. Dit is een gratis evenement voor softwareontwikkelaars om elkaar te ontmoeten, ideeën te delen en samen te werken [16]. In 2019, zoals het voorafgaande jaar, was er een Rust room voorzien waarin onderwerpen omrent Rust besproken werden.

Rust is verkozen tot meest geliefde programmeertaal vier jaar op rij in 2016 [17], 2017 [18],2018 [19] en 2019 [20] door de gebruikers van Stack Overflow.

#### 3.2.2.2 Multiplatform ondersteuning

Het Rust Compiler Team deelt *multiplatform* ondersteuning op in verschillende *tiers*. De lijsten van de verschillende tiers staan op de Rust Forge website [21]. Voor *Tier 1* garandeert het Team dat Rust op deze platformen werkt zonder problemen en automatisch getest is. Ook worden officiële *builds* voor deze platformen aangeboden via de Rustup-CLI.

*Tabel 2 Rust tier 1 platformen*

| Target | std | rustc | cargo | Notities |
|---|---|---|---|---|
| i686-apple-darwin | ✓ | ✓ | ✓ | 32-bit OSX (10.7+, Lion+) |
| i686-pc-windows-gnu | ✓ | ✓ | ✓ | 32-bit MinGW (Windows 7+) |
| i686-pc-windows-msvc | ✓ | ✓ | ✓ | 32-bit MSVC (Windows 7+) |
| i686-unknown-linux-gnu | ✓ | ✓ | ✓ | 32-bit Linux (2.6.18+) |
| x86_64-apple-darwin | ✓ | ✓ | ✓ | 64-bit OSX (10.7+, Lion+) |
| x86_64-pc-windows-gnu | ✓ | ✓ | ✓ | 64-bit MinGW (Windows 7+) |
| x86_64-pc-windows-msvc | ✓ | ✓ | ✓ | 64-bit MSVC (Windows 7+) |
| x86_64-unknown-linux-gnu | ✓ | ✓ | ✓ | 64-bit Linux (2.6.18+) |

*Builds* voor de Tier 2 platformen worden officieel aangeboden via de Rustup-CLI. Het Rust Team garandeert dat Rust compileert voor alle platformen in deze tier. De processor van de Raspberry Pi die gebruikt werd in het toegepast onderzoek valt in deze Tier. Dit Raspberry Pi model bevat een ARMv8-processor die ARMv7-instructies ondersteunt. Rust heeft ondersteuning voor dit platform onder het target *armv7-unknown-linux-gnueabihf*. Zie de bijlage voor een gedetailleerde tier 2 lijst.

Voor tier 2.5 en 3 worden er geen officiële *build* aangeboden. Voor tier 2.5 garandeert het team dat Rust op deze platformen compileert. Voor tier 3 is er ondersteuning van Rust maar deze worden niet automatisch getest en is er geen garantie dat het werkt.

#### 3.2.2.3 Geen garbage collector

Door het gebruik van *ownership and borrowing* bestaat er geen *garbage collector*. [22] Tijdens het compileren wordt het toewijzen en vrijgeven van het geheugen automatisch in de applicatie geïntegreerd. Hierdoor kan de compiler het geheugengebruik optimaliseren. De applicatie gaat minder geheugen gebruiken en is sneller tijdens *run-time*. Er moet geen processortijd vrijgemaakt worden om aan geheugenbeheer te doen in tegenstelling tot Java waar de *garbage collector* het geheugenbeheer doet [23]. In Java gebeurt het geheugenbeheer door een blok geheugen toe te wijzen en hierin objecten te beheren. Als er naar één van deze objecten geen referenties meer zijn, dan wordt het object gemarkeerd om opgeruimd te worden.

De *borrow checker* in Rust past de regels zeer strikt toe. Het systeem laat niet toe om een programma te compileren als er niet voldaan wordt aan de regels. Hierdoor worden *data races* vermeden en moet de programmeur tijd en aandacht besteden aan de implementatie en code.

#### 3.2.2.4 Typesysteem

Rust is *strongly typed* en heeft een geavanceerd typesysteem. Standaard zijn variabelen onveranderlijk. Het is mogelijk om deze variabelen veranderlijk te maken door dit aan te geven met het keyword *mut*. Als een variabele veranderlijk is, dan moet het gebruik ervan voldoen aan de *ownership and borrow*-regels zoals beschreven in de *ownership and borrowing* rubriek.

In Java zijn variabelen standaard veranderlijk. Variabele moeten expliciet als *final* gedefinieerd worden om aan te duiden dan de waarde onveranderlijk is. In de programmeertaal Rust is dit concept omgedraaid.

Ook bestaat In Rust de waarde *null* niet. Om aan te duiden of een variabele bestaat kan er gebruikgemaakt worden van het type Option. Dit type kan een *Some(value)* of *None* waarde bevatten. Dit concept kan vergeleken worden met de *Optional*-klasse in Java waar een *null* waarde wordt gebruikt om aan te geven of het onderliggende object bestaat.

#### 3.2.2.5 Grote bedrijven gebruiken Rust

Grote bedrijven maken gebruik van Rust om hun producten te ontwikkelen, enkele succesverhalen staan vermeld [24] op de Rust website. Hieronder is een oplijsting van bedrijven en enkele van hun producten die Rust gebruiken.

*Tabel 3 Bedrijven en hun producten die Rust gebruiken*

| Bedrijf | Producten |
|---|---|
| Google | Google Fuchia |
| Dropbox [25] | DivAns [26] |
| NPM [6] | Shelby [27] |
| Mozilla Corporation | Servo |

### 3.2.3 Zwaktes

#### 3.2.3.1 Steile leercurve

Rust staat gekend voor moeilijk leerbaar te zijn ondanks dat er een sterke inhaalbeweging [28] is gemaakt door de verschillende teams die aan Rust werken om de documentatie te verbeteren. Het is een jonge programmeertaal die nog maar vier jaar oud is. Het grootste struikelblok is het *ownership*-systeem.

Een groot aantal programmeurs struikelen over het *ownership*-systeem omdat het een nieuw concept [29]. Dit concept vraagt tijd om aan te wennen. Ook past het systeem zijn regeltjes strikt toe waardoor er veel tijd in het correct toepassen van de regels gestoken moet worden om de *borrow*-checker geen foutmeldingen meer te laten weergeven. Hiernaast kunnen *lifetimes* de ontwikkeling moeilijk maken:

> *"In Rust, storing a pointer in a variable can prevent you from mutating the thing it points to as long as that variable is in scope. This guarantees safety but is overly restrictive since the variable may not be needed anymore by the time the mutation happens. [..] it's still frustrating to have to pause your work to solve the little unnecessary borrow checker puzzles that can come up regularly as you work." [5]*

#### 3.2.3.2 Weinig standaardisatie

Zoals eerder vermeld heeft het Rust team een inhaalbeweging gedaan om hun documentatie te verbeteren. Dit is hun goed gelukt voor de programmeertaal te leren begrijpen maar niet voor het toepassen ervan. Veel documentatie ontbreekt over de best practices en *design patterns*. In het *Rust book* en *Learn Rust by Example* worden enkele voorbeelden gegeven over kleine onderdelen van Rust. Deze voorbeelden zijn meestal ter illustratie.

Uit de conclusie van de literatuurstudie (rubriek 3.1.6) blijkt dat dat er nog veel standaard *libraries* ontbreken. Dit komt omdat Rust nog een vrij jonge programmeertaal is. Bij elke update van Rust worden er steeds nieuwe dingen geïntroduceerd in de standaard zoals vermeld in rubriek 3.2.1.

#### 3.2.3.3 Rapid prototyping is trager

Het snel ontwikkelen van applicaties in Rust is moeilijker dan in Java omdat de compiler code zal afwijzen als er onduidelijkheden in aanwezig zijn. Dit is goed voor de kwaliteit van de code maar niet om prototypes te ontwikkelen. Fouten moeten correct worden afgehandeld voordat de applicatie compileert.

## 3.3 Toegepast onderzoek

Het toegepast onderzoek bestaat uit twee onderdelen: het uitwerken van de *proof of concept* implementaties en het benchmarken ervan. De uitwerking van de *proof of concept* bestaat uit een REST-API die het verworven kapitaal per jaar met een jaarlijks vast inkomen berekent aan de hand van vaste parameters gedefinieerd in onderstaande rubriek.

#### 3.3.1.1 Formule

Onderstaande formule is een goede maatstaf omdat deze een goede weergave is van een gemiddelde berekening in de financiële planner. Om het verworven kapitaal per jaar met een jaarlijks vast inkomen te simuleren, zijn de volgende parameters nodig: een startkapitaal $S_0$, jaarlijks inkomen I, groeivoet g en inflatiepercentage i.

$$S_{j+1} = S_j * \frac{1+g}{1+i} + I$$

Hieronder is een voorbeeld om het verworven kapitaal op drie jaar te bereken met een startkapitaal van 100, een jaarlijks inkomen van 1200, een groeivoet van vijf percent en inflatie van twee percent:

Eerst rekenen we de vaste coëfficiënt uit voor de groeivoet: (1 + 0.05 / 1 + 0.02) = 1,0294

| | | |
|---|---|---|
| Jaar 1: | 100,0000 | * 1,0294 + 1200 ~= 1302,9400 |
| Jaar 2: | 1302,9400 | * 1,0294 + 1200 ~= 2541,2464 |
| Jaar 3: | 2541,2464 | * 1,0294 + 1200 ~= 3815.9590 |

De bovenstaande getallen zijn benaderingen die afgerond zijn tot vier cijfers na de komma.

### 3.3.2 Meetresultaten

#### 3.3.2.1 Reactietijden

Voor elke benchmark van de Java-implementatie is er een run van één minuut gestart om de Java JIT-compilatie te optimaliseren. De reactietijd werd gemeten en gesimuleerd met Apache JMeter™. De REST-API werd maximaal belast gedurende de benchmark om een systeem onder zware load te simuleren. De grafieken in de onderstaande rubrieken zijn gegenereerd met Apache JMeter™.

##### 3.3.2.1.1 Overzicht

Rust is de winnaar op de Intel x86-64 architectuur met een constante reactiesnelheid die meer dan twaalf keer sneller is dan de Java-implementatie op dezelfde architectuur. De reactietijd van de Java-implementatie schommelde tussen de anderhalve tot acht milliseconden maar bleef gemiddeld

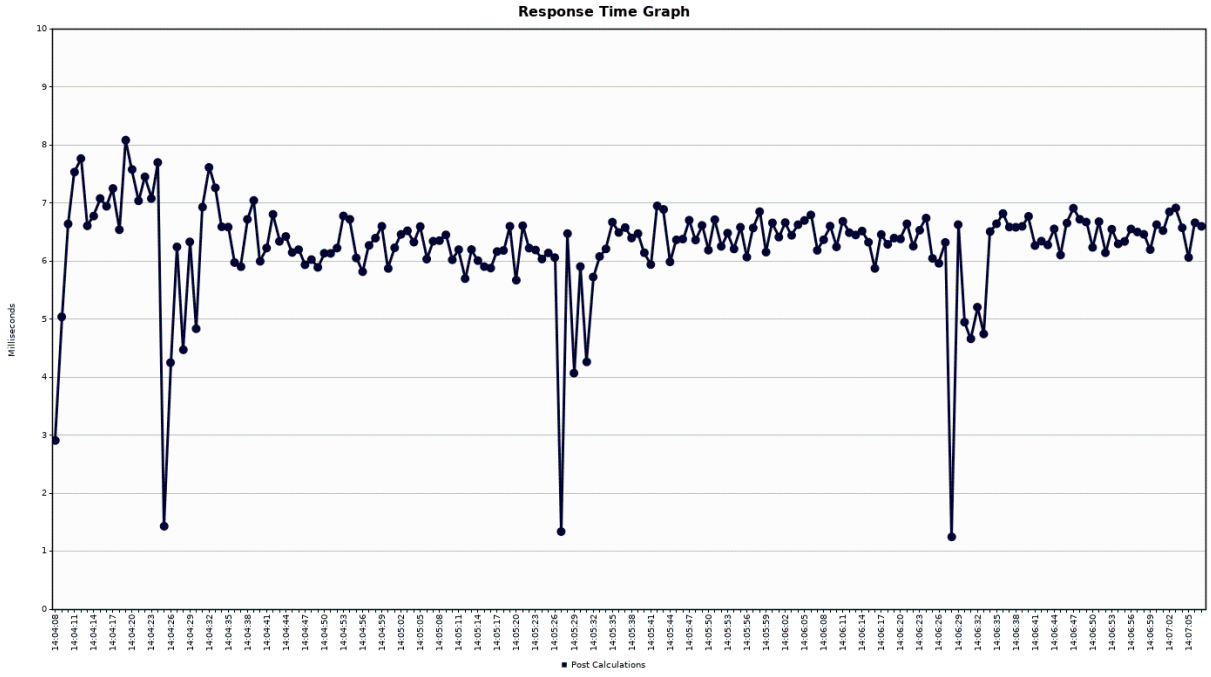
constant tussen de zes en zeven milliseconden terwijl de reactietijd van de Rust-implementatie constant onder een halve milliseconde bleef.

De reactietijd van de ARM-Rust-implementatie ligt in hetzelfde bereik als die van Java-implementatie op de Intel x86-64 architectuur.

#### 3.3.2.1.2 Intel x86-64 architectuur

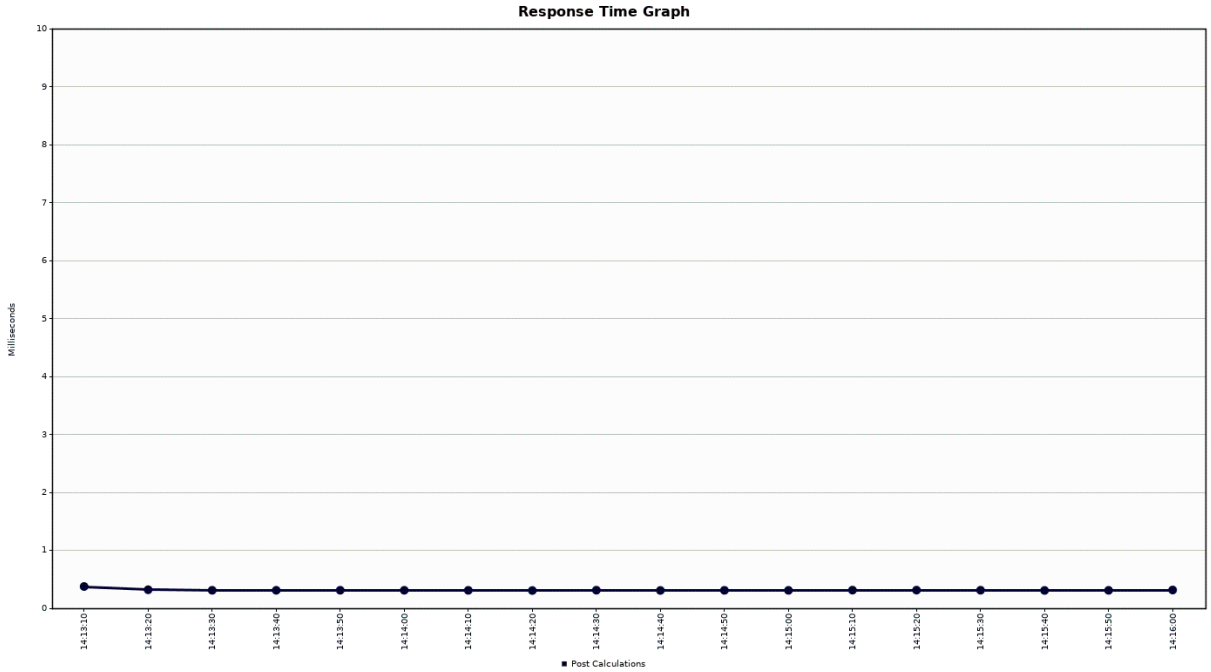
Deze benchmarks werden uitgevoerd op een Intel® Core™ i7-8700K Processor.

Onderstaande grafiek is een weergave van de reactietijd van de Java-implementatie op de Intel x86-64 architectuur met de horizontale as als de tijdsduur met een frequentie van 1 seconde en de verticale as als de reactietijd met een bereik van nul tot tien milliseconden en een frequentie van één milliseconde.



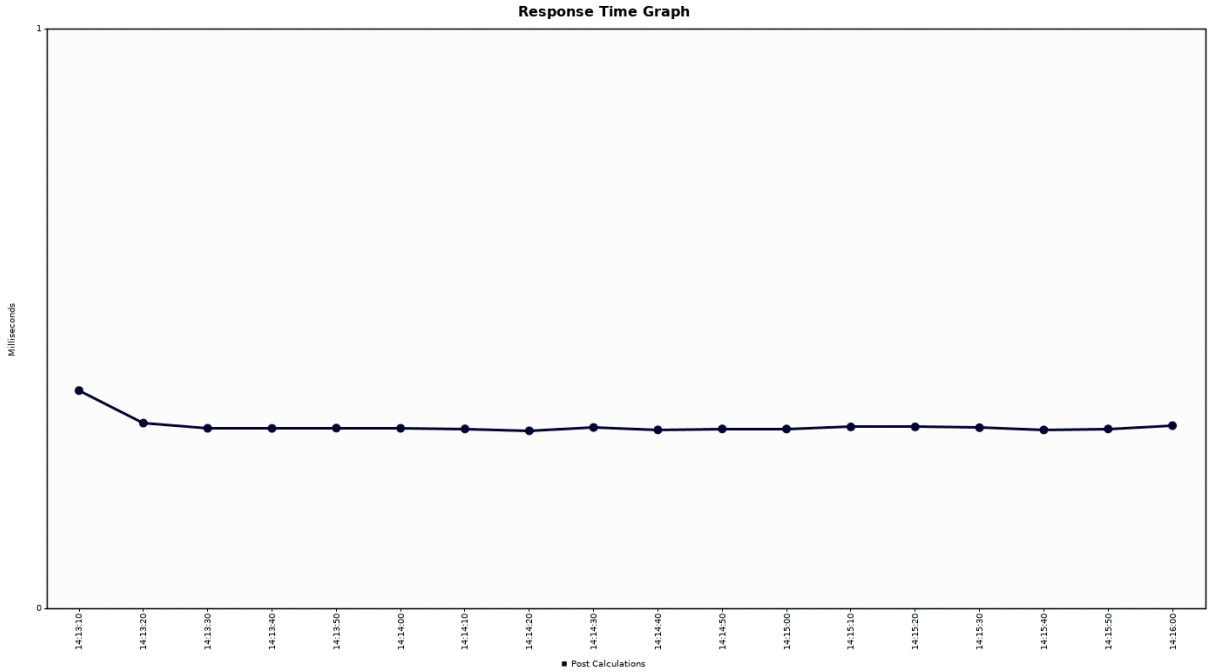

*Figuur 31 Reactietijd Java Intel op x86-64 architectuur*

Onderstaande grafiek is een weergave van de reactietijd van de Rust-implementatie op de Intel x86-64 architectuur met de horizontale as als de tijdsduur met een frequentie van één seconde en de verticale as als de reactietijd met een bereik van nul tot tien milliseconden en een frequentie van één milliseconde.




*Figuur 32 Reactietijd Rust op Intel x86-64 architectuur*

De reactietijd van de Rust-implementatie is onder één milliseconde zoals afgeleid kan worden uit de bovenstaande grafiek. In de onderstaande grafiek is de schaal van de verticale as herleid naar één milliseconde om een beter beeld te krijgen van de constante reactietijd.

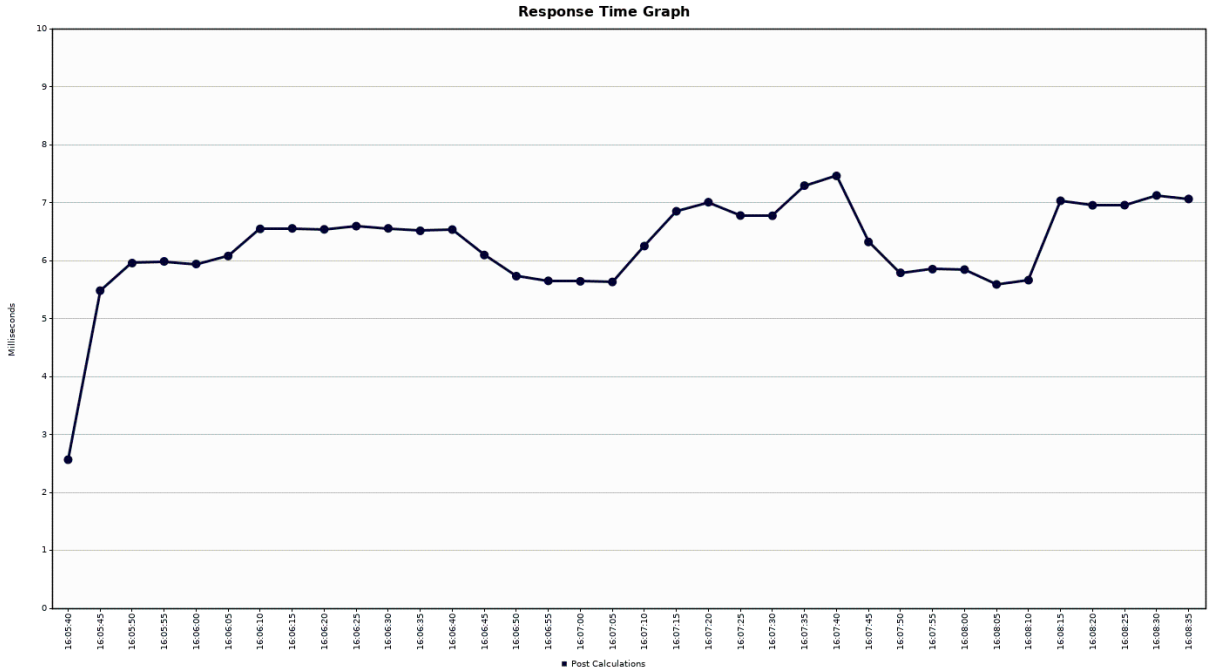



*Figuur 33 Reactietijd Rust op Intel x86-64 architectuur met een schaal van 0 tot 1 milliseconde*

#### 3.3.2.1.3 ARM-architectuur

De ARM-architectuur benchmarks werden uitgevoerd op een Raspberry Pi 3 B+ met een ARMv8 processor.

Onderstaande grafiek is een weergave van de reactietijd van de Rust-implementatie op de ARM-architectuur met de horizontale as als de tijdsduur met een frequentie van één seconde en de verticale as als de reactietijd met een bereik van nul tot tien milliseconden en een frequentie van één milliseconde.




*Figuur 34 Reactietijd Rust op ARM-architectuur*

Voor de Java-applicatie zijn er geen resultaten beschikbaar. Tijdens het uitrollen gaf de applicatie een foutmelding met de HTTP-statuscode 406 ondanks dat alle HTTP-headers correct geconfigureerd waren. Er zijn verschillende methodes geprobeerd om de API aan het werken te krijgen. Wegens beperkte tijd is er uiteindelijk na een week besloten dit niet uit te werken. Uit dit voorval kan men wel afleiden dat Rust makkelijker uit te rollen is naar ondersteunde platformen.

# Conclusie

Het doel van het onderzoek was het in kaart brengen van de sterktes en zwaktes van programmeertaal Rust in combinatie met een performantievergelijking van de twee uitgewerkte *proof of concepts*.

Uit het literatuuronderzoek kan geconcludeerd worden dat Rust een programmeertaal is waar de focus op performantie ligt, een laag geheugengebruik heeft en memory safe is. Rust kan dit verzekeren dankzij zijn uniek kernconcept: *ownership and borrowing*. Het uitrekenen van formules in Rust is sneller van Java zoals aan getoond in het toegepast onderzoek.

Bij de performantiemetingen zijn er twee computerarchitecturen gebruikt. Er is een Raspberry Pi gebruikt voor het meten van resultaten voor de ARM-architectuur. Voor de Intel x64-86 architectuur is de lokale ontwikkelingsomgeving gebruikt. Uit de resultaten van het onderzoek kan worden afgeleid dat de reactietijden van de Rust-implementaties opmerkelijk sneller waren dan die van de Java-implementaties. Op de Intel architectuur waren de verzoeken van de Rust-API twaalfmaal sneller dan de gelijkaardige Java implementatie. Het toegepast onderzoek toont aan dat Rust uitstekend anderen computerplatformen ondersteund.

Rust heeft een steile leercurve omdat het een jonge programmeertaal is maar groeit sterk door de actief betrokken community. Een programmeur kan de taal leren door het Rust boek te lezen.

# Aanbevelingen

Ondanks de snelle performantie van Rust is het beter voor Argeüs om deze taal momenteel niet te introduceren. Rust vraagt veel tijd om te leren. Veel extra tijd is er niet beschikbaar in het kleine team van Argeüs omdat iedereen verantwoordelijk is voor het onderhouden van de bestaande applicaties. Omdat de Java kennis beter is dan de Rust kennis is het beter voor de ontwikkelaars om de bestaande code te optimaliseren.

Als er in de toekomst een project is dat zeer snelle berekeningen nodig heeft, dan raad ik Argeüs aan om Rust eens te bekijken. Rust is goed gedocumenteerd en *strongly typed*. Daarnaast moeten fouten correct afgehandeld worden. Hierdoor kan de ontwikkelaar overhaaste beslissingen nemen bij het schrijven van code. Hierdoor zal er meer tijd en aandacht gestoken worden in de code waardoor onderhoud makkelijk wordt. De onderhoudskost daalt zoals beschreven in de casestudie van npm.

Als aansluiting op dit onderzoek kan een nieuw onderzoek uitgevoerd worden naar de impact van architecturale wijzigingen in de bestaande applicaties op de performantie.

# Persoonlijke reflectie

In het begin van mijn stage dacht ik dat Rust nuttig kon zijn voor Argeüs. Na enkele gesprekken met mijn bedrijfspromotor mocht ik onderzoek doen naar de programmeertaal Rust. Dit vond ik een geweldige ervaring. Ik heb veel bijgeleerd over deze programmeertaal.

Bij het uitwerken van de *proof of concept* API's had ik ontdekt dat *rapid prototyping* in Rust niet makkelijk is. Ik had problemen met het *ownership and borrowing*-systeem omdat dit een nieuw concept voor mij was. Dit leidde tot frustratie omdat ik niet begreep wat er fout was in de code.

Na enkele dagen experimenteren begon ik het *ownership and borrowing*-systeem beter te begrijpen. Hierna heb ik mij verdiept in de programmeertaal waardoor ik technisch sterk gegroeid ben. Ik vond het jammer dat de *best practices* van Rust nog niet gestandaardiseerd zijn. Hierdoor heb ik lang zitten zoeken voor *best pratices* voor een groot project.

Ik vond het tof om het groeitraject van een nieuwe programmeertaal te zien. Rust is in 2015 uitgerold. Op enkele jaren tijd hebben ze hun initiële tekortkomingen weggewerkt. Tijdens mijn onderzoek ben ik door verschillende artikels en documentatie gegaan. In deze artikels heb ik ontdekt dat de ontwikkelaars de programmeertaal meer toegankelijker hebben gemaakt en dat ze dit willen blijven doen.

Buiten mijn stage was ik ook bezig met Rust. Ik had een vraag omrent het gebruik van een *library*. Ik heb deze vraag gesteld in het officiële Discord-chatkanaal van Rust. In enkele minuten kreeg ik een gedetailleerde uitleg van een van de community leden. Ik was verbaasd van de snelle reacties.

In mijn privéprojecten ga ik Rust zeker gebruiken. Dit onderzoek heeft mij een fan gemaakt van de programmeertaal.  Het was een zeer interessant onderzoek voor mij. Ik vond het jammer dat ik Rust moest afraden voor Argeüs. Aangezien Argeüs een klein ontwikkelingsteam heeft en de steile leercurve van de programmeertaal vind ik dit een wijze beslissing.

# Bibliografie


[1] C. M. Molin, „Comparing JSON documents in Java | Insights,” 23 07 2018. [Online]. Available: https://cassiomolin.com/2018/07/23/comparing-json-documents-in-java/. [Geopend 6 03 2019].

[2] Google, „Commits · google/guava · GitHub,” 2019. [Online]. Available: https://github.com/google/guava/commits/master. [Geopend 08 06 2019].

[3] E. Carroll, „sui.modal Incompatiple with Semantic UI CSS 2.3.3 · Issue #407 · edcarroll/ng2-semantic-ui · GitHub,” 10 08 2018. [Online]. Available: https://github.com/edcarroll/ng2-semantic-ui/issues/407. [Geopend 14 03 2019].

[4] The Rust Team, „Learn - Rust programming language,” 2019. [Online]. Available: https://www.rust-lang.org/learn. [Geopend 03 05 2019].

[5] E. Wallace, „How Mozilla’s Rust dramatically improved our server-side performance,” Figma, 02 06 2018. [Online]. Available: https://www.figma.com/blog/rust-in-production-at-figma/. [Geopend 08 05 2019].

[6] The Rust Project Developers, „Rust Case Study: Community makes Rust an easy choice for npm,” 25 02 2019. [Online]. Available: https://www.rust-lang.org/static/pdfs/Rust-npm-Whitepaper.pdf. [Geopend 25 05 2019].

[7] LLVM-admin Team, „The LLVM Compiler Infrastructure,” 2019. [Online]. Available: https://llvm.org/. [Geopend 06 06 2019].

[8] The Rust Community Team, „Community - Rust programming language,” 2019. [Online]. Available: https://www.rust-lang.org/community. [Geopend 27 05 2019].

[9] „Validating References with Lifetimes - The Rust Programming Language,” 2019. [Online]. Available: https://doc.rust-lang.org/book/ch10-03-lifetime-syntax.html. [Geopend 06 06 2019].

[10] „rfcs/2033-experimental-coroutines.md at master · rust-lang/rfcs · GitHub,” 15 06 2017. [Online]. Available: https://github.com/rust-lang/rfcs/blob/master/text/2033-experimental-coroutines.md#asyncawait-syntax. [Geopend 06 06 2019].

[11] The Rust Team, „What is Ownership? - The Rust Programming Language,” [Online]. Available: https://doc.rust-lang.org/book/second-edition/ch04-00-understanding-ownership.html. [Geopend 13 05 2019].

[12] The Rust Team, „References and Borrowing - The Rust Programming Language,” 2018. [Online]. Available: https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html. [Geopend 13 05 2019].

[13] The Rustup contributors, „rust-lang/rustup.rs: The Rust toolchain installer,” 2019. [Online]. Available: https://github.com/rust-lang/rustup.rs. [Geopend 29 04 2019].

[14] „Cargo: packages for Rust,” 09 06 2019. [Online]. Available: https://crates.io/crates. [Geopend 09 06 2019].

[15] Rust Community Team, „The 2019 Rust Event Lineup | Rust Blog,” 20 05 2019. [Online]. Available: https://blog.rust-lang.org/2019/05/20/The-2019-Rust-Event-Lineup.html. [Geopend 27 05 2019].

[16] „FOSDEM 2019 - Home,” 2019. [Online]. Available: https://fosdem.org/2019/. [Geopend 27 05 2019].

[17] Stack Overflow, „Stack Overflow Developer Survey 2016 Results,” Stack Overflow, 2016. [Online]. Available: https://insights.stackoverflow.com/survey/2016#technology-most-loved-dreaded-and-wanted-loved. [Geopend 29 04 2019].

[18] Stack Overflow, „Stack Overflow Developer Survey 2017,” Stack Overflow, 2017. [Online]. Available: https://insights.stackoverflow.com/survey/2017#technology-most-loved-dreaded-and-wanted-loved. [Geopend 29 04 2019].

[19] Stack Overflow, „Stack Overflow Developer Survey 2018,” Stack Overflow, 2018. [Online]. Available: https://insights.stackoverflow.com/survey/2018#overview. [Geopend 29 04 2019].

[20] Stack Overflow, „Stack Overflow Developer Survey 2019,” Stack Overflow, 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019#technology-most-loved-dreaded-and-wanted-loved. [Geopend 29 04 2019].

[21] „Rust Platform Support · The Rust Programming Language,” [Online]. Available: https://forge.rust-lang.org/platform-support.html. [Geopend 13 05 2019].

[22] the Rust Team, „Fearless Concurrency with Rust | Rust Blog,” 10 04 2015. [Online]. Available: https://blog.rust-lang.org/2015/04/10/Fearless-Concurrency.html. [Geopend 15 05 2019].

[23] J. Kulandai, „How Java Garbage Collection Works? - Javapapers,” 13 10 2014. [Online]. Available: https://javapapers.com/java/how-java-garbage-collection-works/. [Geopend 27 05 2019].

[24] The Rust Team, „Production - Rust programming language,” [Online]. Available: https://www.rust-lang.org/production. [Geopend 22 05 2019].

[25] „Dropbox,” Dropbox, [Online]. Available: https://github.com/dropbox?language=rust. [Geopend 25 05 2019].

[26] D. R. H. a. J. Baek, „Building better compression together with DivANS | Dropbox Tech Blog,” Dropbox, 19 06 2018. [Online]. Available: https://blogs.dropbox.com/tech/2018/06/building-better-compression-together-with-divans/. [Geopend 21 05 2019].

[27] NPM, „npm/shelby: a system agent in rust for the numbat metrics system,” [Online]. Available: https://github.com/npm/shelby. [Geopend 27 05 2019].

[28] C. Nichols, „Rust: A Language for the Next 40 Years,” in *Emering techonologies for the enterprise conference*, Philadelphia, PA, USA, 2019.

[29] A. Nowell, „Why Rust?,” 29 12 2016. [Online]. Available: http://anowell.com/posts/why-rust.html. [Geopend 16 05 2019].

[30] The Rust Core Team, „Announcing Rust 1.31 and Rust 2018 | Rust Blog,” 6 12 2018. [Online]. Available: https://blog.rust-lang.org/2018/12/06/Rust-1.31-and-rust-2018.html#rust-2018. [Geopend 26 04 2019].

[31] D. Madunuwan, „Why Rust? | Learning Rust,” 2019. [Online]. Available: https://learning-rust.github.io/docs/a1.why_rust.html. [Geopend 26 03 2019].

[32] P. Ruffwind, „Graphical depiction of ownership and borrowing in Rust - Rufflewind's Scratchpad,” 15 02 2017. [Online]. Available: https://rufflewind.com/2017-02-15/rust-move-copy-borrow. [Geopend 13 05 2019].

[33] The Rust Team, „References and Borrowing - The Rust Programming Language,” 2018. [Online]. Available: https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html. [Geopend 13 05 2019].

[34] „Fear not the Rust Borrow Checker,” 31 05 2018. [Online]. Available: http://squidarth.com/rc/rust/2018/05/31/rust-borrowing-and-ownership.html. [Geopend 14 05 2019].

[35] K. H. &. A. B. Cyrille Artho, „High-Level Data Races,” [Online]. Available: https://people.kth.se/~artho/papers/vveis03.pdf. [Geopend 27 05 2019].

# Bijlage

## A. Rust tier 2 platformen

## A. Rust tier 2 platformen

*Tabel 4 Rust tier 2 platformen*

| Target | std | rustc | cargo | Notities |
|---|---|---|---|---|
| aarch64-apple-ios | ✓ | | | ARM64 iOS |
| aarch64-fuchsia | ✓ | | | ARM64 Fuchsia |
| aarch64-linux-android | ✓ | | | ARM64 Android |
| aarch64-unknown-linux-gnu | ✓ | ✓ | ✓ | ARM64 Linux |
| aarch64-unknown-linux-musl | ✓ | | | ARM64 Linux with MUSL |
| arm-linux-androideabi | ✓ | | | ARMv7 Android |
| arm-unknown-linux-gnueabi | ✓ | ✓ | ✓ | ARMv6 Linux |
| arm-unknown-linux-gnueabihf | ✓ | ✓ | ✓ | ARMv6 Linux, hardfloat |
| arm-unknown-linux-musleabi | ✓ | | | ARMv6 Linux with MUSL |
| arm-unknown-linux-musleabihf | ✓ | | | ARMv6 Linux, MUSL, hardfloat |
| armv5te-unknown-linux-gnueabi | ✓ | | | ARMv5TE Linux |
| armv7-apple-ios | ✓ | | | ARMv7 iOS, Cortex-a8 |
| armv7-linux-androideabi | ✓ | | | ARMv7a Android |
| armv7-unknown-linux-gnueabihf | ✓ | ✓ | ✓ | ARMv7 Linux |
| armv7-unknown-linux-musleabihf | ✓ | | | ARMv7 Linux with MUSL |
| armv7s-apple-ios | ✓ | | | ARMv7 iOS, Cortex-a9 |
| asmjs-unknown-emscripten | ✓ | | | asm.js via Emscripten |
| i386-apple-ios | ✓ | | | 32-bit x86 iOS |
| i586-pc-windows-msvc | ✓ | | | 32-bit Windows w/o SSE |
| i586-unknown-linux-gnu | ✓ | | | 32-bit Linux w/o SSE |
| i586-unknown-linux-musl | ✓ | | | 32-bit Linux w/o SSE, MUSL |
| i686-linux-android | ✓ | | | 32-bit x86 Android |
| i686-unknown-freebsd | ✓ | ✓ | ✓ | 32-bit FreeBSD |
| i686-unknown-linux-musl | ✓ | | | 32-bit Linux with MUSL |
| mips-unknown-linux-gnu | ✓ | ✓ | ✓ | MIPS Linux |
| mips-unknown-linux-musl | ✓ | | | MIPS Linux with MUSL |
| mips64-unknown-linux-gnuabi64 | ✓ | ✓ | ✓ | MIPS64 Linux, n64 ABI |
| mips64el-unknown-linux-gnuabi64 | ✓ | ✓ | ✓ | MIPS64 (LE) Linux, n64 ABI |
| mipsel-unknown-linux-gnu | ✓ | ✓ | ✓ | MIPS (LE) Linux |
| mipsel-unknown-linux-musl | ✓ | | | MIPS (LE) Linux with MUSL |
| powerpc-unknown-linux-gnu | ✓ | ✓ | ✓ | PowerPC Linux |
| powerpc64-unknown-linux-gnu | ✓ | ✓ | ✓ | PPC64 Linux |
| powerpc64le-unknown-linux-gnu | ✓ | ✓ | ✓ | PPC64LE Linux |
| riscv32imac-unknown-none-elf | * | | | Bare RISC-V (RV32IMAC ISA) |
| riscv32imc-unknown-none-elf | * | | | Bare RISC-V (RV32IMC ISA) |
| riscv64gc-unknown-none-elf | * | | | Bare RISC-V (RV64IMAFDC ISA) |
| riscv64imac-unknown-none-elf | * | | | Bare RISC-V (RV64IMAC ISA) |
| s390x-unknown-linux-gnu | ✓ | ✓ | ✓ | S390x Linux |
| sparc64-unknown-linux-gnu | ✓ | | | SPARC Linux |
| sparcv9-sun-solaris | ✓ | | | SPARC Solaris 10/11, illumos |
| wasm32-unknown-unknown | ✓ | | | WebAssembly |
| wasm32-unknown-emscripten | ✓ | | | WebAssembly via Emscripten |

| Target | std | rustc | cargo | Notities |
|---|---|---|---|---|
| x86_64-apple-ios | ✓ | | | 64-bit x86 iOS |
| x86_64-fortanix-unknown-sgx | ✓ | | | Fortanix ABI for 64-bit Intel SGX |
| x86_64-fuchsia | ✓ | | | 64-bit Fuchsia |
| x86_64-linux-android | ✓ | | | 64-bit x86 Android |
| x86_64-rumprun-netbsd | ✓ | | | 64-bit NetBSD Rump Kernel |
| x86_64-sun-solaris | ✓ | | | 64-bit Solaris 10/11, illumos |
| x86_64-unknown-cloudabi | ✓ | | | 64-bit CloudABI |
| x86_64-unknown-freebsd | ✓ | ✓ | ✓ | 64-bit FreeBSD |
| x86_64-unknown-linux-gnux32 | ✓ | | | 64-bit Linux |
| x86_64-unknown-linux-musl | ✓ | ✓ | ✓ | 64-bit Linux with MUSL |
| x86_64-unknown-netbsd | ✓ | ✓ | ✓ | NetBSD/amd64 |
| x86_64-unknown-redox | ✓ | | | Redox OS |

* These are bare-metal microcontroller targets that only have access to the core library, not std. [21]