Professionele Bachelor Toegepaste Informatica

Retroid

Sergei Belov

Promotoren:

Bert Roex
Jan Willekens

XTi
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Retroid

Sergei Belov

Promotoren:

Bert Roex
Jan Willekens

XTi
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Ik vroeg een uitdaging, XTi gaf me er genoeg. Tijdens de 12 weken periode waarin ik stage mocht doen heb ik de opportuniteit gekregen om nieuwe technologie te leren en toe te passen. Problemen kwamen en problemen verdwenen, maar toch kon ik altijd rekenen op steun en kennis dat het gehele team van XTi te bieden had.

Daarom wil ik geheel XTi bedanken voor de kans die mij is gegeven, alle steun die ik gekregen heb en kennis en ervaring die ik heb mogen opdoen tijdens de stageperiode.
Bedankt Bert Roex, Maarten Tutak, Calvin Swinnen, Nick De Kock en Johnny Ressen.

Daarnaast wil ik Jan Willekens bedanken die mij heeft bijgestaan als PXL promotor en mijn eindwerk meermaals nalas en verbeterde.

Abstract

Retrospectives binnen een agile proces nemen veel tijd in beslag. Het aanmaken, sorteren en overlopen van tickets kan makkelijk uren duren. Op dit moment zijn er weinig tools op de markt die een oplossing bieden die ook mobielvriendelijk tewerk gaan. Daarom wil XTi de perfecte tool ontwikkelen die door iedereen gebruikt kan worden. Het doel van de stage is om een oplossing te bieden die zowel op een computer als op een mobiel apparaat gebruikt kan worden. Aan de hand van deze tool kunnen gebruikers, zonder zich te registreren, tickets aanmaken binnen hun eigen of een gedeelde werkplek. Met één knop kunnen alle tickets samengevoegd worden met dezelfde context en sentiment, en dit met behulp van *machine learning.* Zo kunnen alle tickets vlot met het hele team besproken worden en gaat er minder tijd verloren. De besproken tickets kunnen ook actiepunten omvatten die verplicht volgens het SMART-principe opgesteld moeten zijn. Het SMART-principe houdt in dat de actiepunten specifiek, meetbaar, haalbaar, relevant en tijdgebonden moeten zijn.

De technologieën die gebruikt worden voor de frontend zijn Vue.js ondersteund door Vuetify. Vue.js is een Javascript-framework dat in 2014 gestart is. Het framework probeert de beste eigenschappen van React.js en Angular te combineren en de gebruiker zelf de keuze te geven over de functionaliteiten die gebruikt worden. Samen met Vue.js is er gekozen voor Vuetify in plaats van het meer gebruikte Bootstrap4 framework omdat Vuetify specifiek gemaakt is voor Vue.js en gekend staat voor de mooie resultaten.

In de backend werd gekozen voor Spring Boot geschreven in Kotlin samen met Gradle. Kotlin is een recente technologie. Het is een framework dat concurreert met Java en meer en meer aan het opkomen is. Het doel van Kotlin is om code zoveel mogelijk te minimaliseren en te optimaliseren. Daarnaast lost het veel kleine problemen eigen aan Java 8.

Gebaseerd op zijn voorgangers Maven en Ant probeert Gradle het beste van beide over te nemen zonder de uitgebreidheid van de configuratie in XML. De Gradle-files zijn korter en overzichtelijker dan de veel gebruikte Maven-files.

Het Spring Boot-framework is tot de dag van vandaag een van de meest gebruikte frameworks om een krachtige en stabiele backend op te zetten.

Voor de implementatie van *machine learning* in het project wordt onderzocht of er gebruikgemaakt kan worden van *Natural Language Processors* om taal te ontleden. Het doel is om zowel het onderwerp als het sentiment te verkrijgen van korte zinnen.

Inhoudsopgave

1. Inhoud

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

AFKORTING	BETEKENIS
AI	Artificiële Intelligentie
IDE	Integrated Development Environment: Programma waarin geprogrammeerd wordt. Doel is om efficiënter te programmeren.
URL	Uniform Resource Locator: Een webadres.
SCRUM	Ontwikkeling strategie.
NLP	Natural Language Processor: Machine learning-framework die zich specialiseert in taal.
SMART	Actiepunten dienen specifiek, meetbaar, haalbaar, relevant en tijdgebonden te zijn.
API	Application Programming Interface: Een methode om verschillende applicaties met elkaar te laten communiceren.
NER	Named Entity Recognition: Herkenning van zelfstandige naamwoorden en waar ze voor staan. bv: Vliegtuig -> Transport
IDE	Integrated Development Environment: Het programma dat gebruikt wordt waarin geprogrammeerd kan worden.
CI/CD	Continuous Integration/Continuous Deployment
NPM	Node Packet Manager
LSA	Latent Semantic Analysis

Inleiding

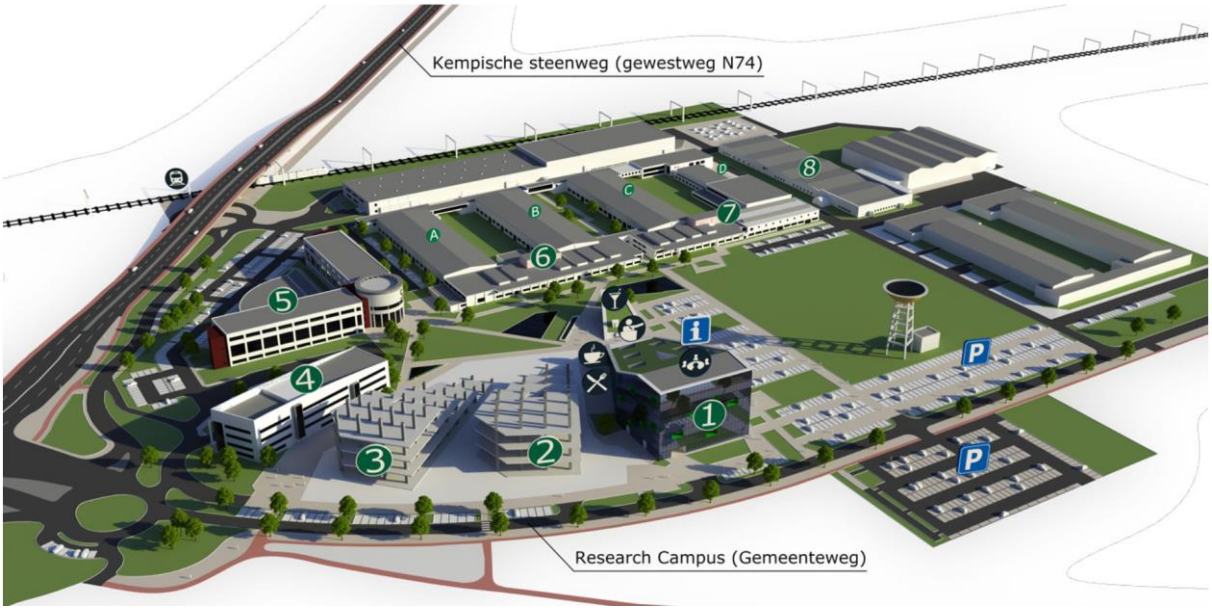
Deze stage vindt plaats in het bedrijf XTi die onderdeel uitmaakt van de Xplore Group en de Xplore Group maakt deel uit van de Cronos Groep. De stage focust zich op sprint *retrospectives* binnen een agile werk omgeving. De manier waarop het nu gebeurt vraagt veel kostbare tijd aan heel het team. Een mogelijke oplossing die wordt onderzocht is het automatiseren van een deel van het proces.

Een deel van het eindwerk omvat het maken van een digitaal platform waar teams hun *retrospectives* kunnen houden. Dit gaat gepaard met recente technologieën zoals Vue.js en Kotlin samen met Spring Boot. Het ander gedeelte van het eindwerk houdt onderzoek naar AI en *machine learning* in en dit in verband met taal. Er wordt onderzocht of het mogelijk is om zinnen te ontleden en mogelijke gelijkenis te herkennen met behulp van bestaande technologieën. Dit met als doel gelijkaardige tekstjes of zinnen te groeperen en terug weer te geven.

I.Stageverslag

2. Bedrijfsvoorstelling

XTi is een dynamisch bedrijf met meerdere vestigingen. Het biedt software op maat aan de klant met behulp van moderne technologieën en Java en open source software als basis. XTi telt 50 werknemers verspreid over Merelbeke, Kontich en Hasselt en maakt deel uit van de Xplore Group die onderdeel is van de Cronos Groep. Zoals getoond in figuur 1 vindt de stage zich plaats op Corda 2, gelegen aan de Corda Campus te Hasselt.



Figuur 1: Grondmap Corda
(figuur van Corda Campus [1])

3. Stageverloop

1.1 Het gebruik van agile methodologie binnen de werkomgeving

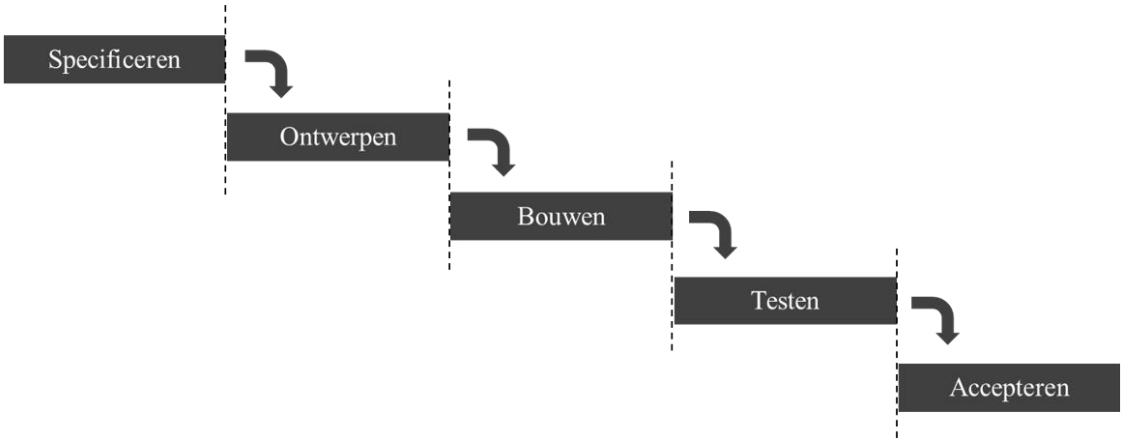
1.1.1 Wat is agile?

In de software development wereld wordt meer en meer de overstap gemaakt naar een agile manier van werken waarbij de klant een deel is van het proces. Een van de meer bekende en meest gebruikte methoden is de SCRUM-methodologie. Korte sprints en constante feedback momenten zorgen ervoor dat er sneller aanpassingen kunnen gebeuren tijdens het ontwikkelen van het product. Meer inspraak van de klant zorgt voor een product waar de klant tevreden mee is.

Een fase binnen het SCRUM-proces omvat zogenaamde sprints en deze zijn meestal twee weken lang. In zo een sprint gebeuren drie handelingen om agile te werk te kunnen gaan. Om te beginnen wordt een sprint planning gehouden met het team om in te plannen wat er binnen de twee weken gerealiseerd moet worden. Daarnaast worden elke dag *daily standups* gehouden met het hele team en mogelijk ook met de klant, waarbij wordt gekeken wat er gebeurd is, wat er die dag gedaan zal worden en of er problemen zijn. Het doel hiervan is om snel in te grijpen als er problemen zijn. Ten slotte wordt er aan het einde van de sprint de sprint-*review* ofwel sprint-*retrospective* gehouden. Deze dient om een zicht te krijgen op de afgelopen twee weken om aan het team te vragen wat er goed ging, wat er minder goed ging en wat er totaal anders moet gebeuren. Het doel hiervan is om fouten te herkennen en oplossingen toe te passen bij de volgende sprint.

1.1.2 Een andere methodologie

Naast de agile methodologie zijn er ook andere methodologieën die worden gebruikt in de softwarewereld. In figuur 2 is er een visueel voorbeeld van een andere veel gebruikte methode genaamd de watervalmethode. Bij deze methode wordt er niet gewerkt met sprints, maar in fases. Binnen een fase worden de benodigdheden gespecifieerd tussen het bedrijf en de klant. Deze worden ook wel *requirements* genoemd. Als de *requirements* zijn vastgelegd, kunnen er ontwerpen gemaakt worden om als richtlijn te dienen voor de programmeurs. Eens deze zijn opgemaakt, beginnen de programmeurs te ontwikkelen aan de applicatie en volgt het testen. Het gemaakte onderdeel of de applicatie wordt uitvoerig getest waarna de klant een goedkeuring kan geven of deze geïmplementeerd mag worden.



Figuur 2: Watervalmethode visuele weergave
(Figuur overgenomen van House Of Control [2])

1.1.3 Vergelijking agile en watervalmethode

De watervalmethode is een alternatief voor de agile ontwikkelingsmethode, maar hoewel de twee methodes veel gebruikt worden in de softwarewereld verschillen de twee toch wel wat. Om een overzicht te verkrijgen worden de twee methodologieën vergeleken.

Een van de grootste verschillen tussen watervalmethode en agile scrum zijn de fases. Hoewel in een agile sprint er aan alles wordt gewerkt wat er op de planning staat, volgt de watervalmethode de fases waar er niet op terug gekeerd kan worden. Eens het ontwikkelen gedaan is en er getest gaat worden kan er niet terug gekeerd worden naar het ontwikkelen. Een ander verschil is de klant. Bij de agile methodologie maakt de klant deel uit van het team en doet mee aan de verscheidenen activiteiten. Bij de waterval methode maakt de klant alleen deel uit bij het opstellen van de eisen en de demo. Maar één van de belangrijkste verschillen is de flexibiliteit tussen de twee. De wensen van de klant kunnen tijdens het ontwikkelen veranderen. De agile methodologie gaat flexibel om met veranderingen terwijl de watervalmethode er geen mogelijkheid ertoe is.

1.2 Stageopdracht

Het doel van de stage is om een deel van het agile proces te versnellen en te vereenvoudigen voor de deelnemende partijen te maken. Het deel waar de stage precies op focust zijn de *retrospectives* binnen een sprint. Waar de *scrum-master* de aangemaakte tickets handmatig moet sorteren op welk ticket inhoudelijk overeenkomt met andere tickets. Hier bestaat de mogelijkheid om het manueel werk te vervangen met technologieën zoals machine learning. Het gebruik van *machine learning* wordt in het research gedeelte besproken, terwijl in dit deel het aanmaken en onderhouden van een mobielvriendelijk platform besproken wordt.

1.3 Technologieën

1.3.1 Vue.js

Voor de grafische weergave van de ontwikkelde software is er gekozen voor het Javascript-Framework Vue.js, dat bekend staat als het beste van Angular en React.js samengebundeld in een framework. Binnen Vue.js bestaat zowel de mogelijkheid om Javascript te gebruiken als programmeertaal alsom, zoals in Angular, gebruik te maken van Typescript. Het framework is ontwikkeld door voormalig Google-werknemer Evan You in 2014 en is de jongste van de drie Javascript-frameworks.

Een aantal voordelen van Vue.js ten opzichte van de andere bekende Javascript frameworks zijn:

- Component-gebaseerd programmeren, wat inhoudt dat stukken code in verschillende componenten gestoken kunnen worden en deze op elk moment opgehaald kunnen worden aan de hand van de component waarin ze zich bevinden.
- Gedetailleerde documentatie waardoor startende ontwikkelaars een snelle start kunnen maken waardoor de leercurve lager ligt. De gelijkenis met de andere bekende frameworks zoals Angular en React.js helpt om ontwikkelaars een snelle start te laten maken.
- Kleine grootte voor een Javascript-framework waardoor de componenten sneller geladen worden.

1.3.2 Vuetify

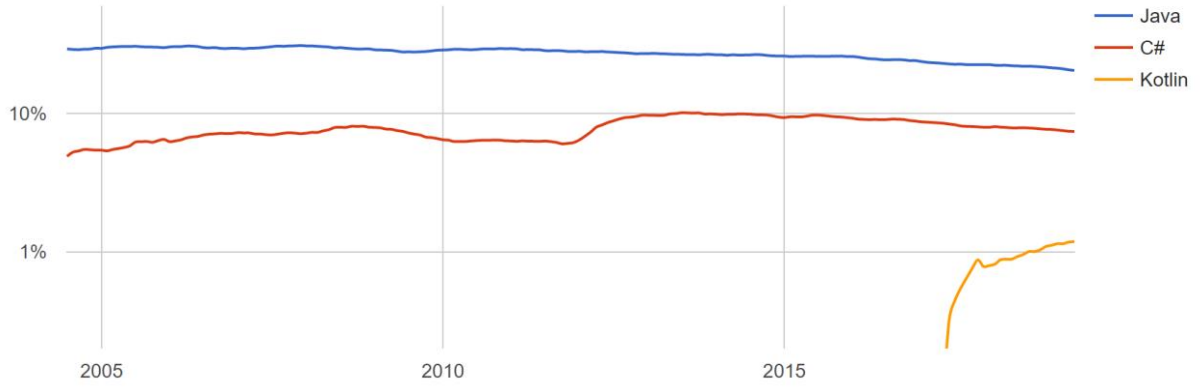
In dit project is er gekozen voor Vuetify in plaats van Bootstrap4 als *design* framework. Vuetify is een *material design* framework en dient als hulpmiddel voor het ontwerpen van een website. Het is speciaal ontworpen voor het Vue.js-framework en bevat een rijke documentatie die voorzien is van voorbeelden om ontwikkelaars te bemiddelen.

Enkele redenen dat er gekozen is voor Vuetify in plaats van Bootstrap4 zijn:

- Gemakkelijke integratie in Vue.js.
- Gericht op Material Design.
- Simpel toepassen van responsive design.
- Zeer uitgebreide documentatie.

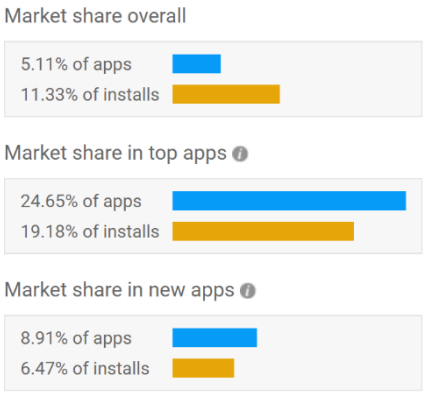
1.3.3 Kotlin

Kotlin is, net zoals Vue.js, een nieuwe speler tussen de programmeertalen en concurreert met het verouderde Java. Ontworpen door JetBrains, een bedrijf dat bekend staat om zijn IDE's, is Kotlin een programmeertaal die in populariteit aan het stijgen is zoals te zien is in figuur 3.



Figuur 3: Grafiek populariteit programmeertalen (Java, Kotlin, C#)
(figuur overgenomen van Pierre Carbonnelle [3])

Met de komst van Android Studio 3.0, een IDE van Google om mobiele applicaties te maken voor Android, wordt Kotlin officieel ondersteund als vervanger voor Java. Dit heeft ook invloed gehad op de populariteit van Kotlin ten opzichte van de klassieke Java. In figuur 4 worden de statistieken weergegeven van apps die gemaakt zijn in Kotlin, en kan vastgesteld worden dat Java in populariteit aan het dalen is.



Figuur 4: Gebruik Kotlin in Android applicaties
(figuur overgenomen van appbrain [19])

1.3.4 Spring Boot

Spring is een framework voor Java dat bekend staat voor zijn simpliciteit. Zonder moeite en al te veel configuratie kan er een stabiele *enterprise level* applicatie opgezet worden op een korte tijd. Het doel is om het ontwikkelen van een stabiele *backend* te vergemakkelijken. Door veel onnodig werk te automatiseren, kan er meer focus gelegd worden op business logica en het creëren van functionele applicaties, dit aan de hand van modules en annotaties die voorgeschreven *boilercode* bevatten.

Met Spring Boot wordt het proces van het opzetten van een applicatie nog eenvoudiger. De bedoeling van Spring Boot is de programmeur zo snel mogelijk startend te krijgen met de backend. Het verschil tussen Spring en Spring Boot is dat veel van de configuratie die bij Spring geconfigureerd moet worden reeds voorgedaan is.

Een aantal voordelen van Spring Boot:

- *Security* kan op verschillende manieren geconfigureerd worden waardoor alle communicatie van en naar Spring op een veilige manier gebeurt.
- Spring zorgt voor alle *dependencies* binnen het project.
- *Dependency Injection* is makkelijk toe te passen binnen Spring aan de hand van de *@Autowired* annotatie.
- Design patronen vormen de basis binnen Spring wat het ontwikkelen van applicaties vergemakkelijkt.

1.3.5 Google Cloud

Als CI/CD pipeline voor het *deployen* van de applicatie wordt er gebruik gemaakt van Google Cloud waarbij er via Google Build een trigger wordt aangemaakt. De trigger activeert wanneer er een nieuwe versie van de applicatie beschikbaar komt op de Bitbucket. Deze versie wordt dan gebouwd en de geschreven *unit*-testen binnen de applicatie worden getest. Eens alles slaagt wordt de applicatie live gezet.

1.4 Analyse

Voor er gestart wordt met ontwikkelen wordt er eerst een test analyse opgemaakt. In dit document wordt de *test strategy* gedefinieerd. Hier wordt vastgelegd wat er getest moet worden en met welke technologieën er getest wordt. Naast de *test strategy* worden ook de *functional requirements en non-functional requirements* opgesteld. Bij de functionele eisen van het project wordt gekeken hoe de applicatie moet reageren als er een actie wordt uitgevoerd. Bij de non-functionele eisen worden alle eisen gedefinieerd die niet afkomstig zijn vanuit de applicatie zelf. Hier kan bepaald worden hoelang de applicatie onbereikbaar mag zijn per dag of jaar. Ten slotte worden ook de best practices genoteerd waaraan het team zich moet houden tijdens de ontwikkeling van de applicatie.

1.4.1 Functional requirements

In deze sectie worden de functionele eisen besproken van het project. Het geeft een weergave hoe de applicatie moet reageren als de gebruiker een actie verricht. De applicatie wordt van begin tot eind gedefinieerd voordat er geprogrammeerd wordt.

- START
- De gebruiker krijgt een veld te zien waarin hij zijn naam moet ingeven.
- Eens de gebruiker op “Next” duwt, krijgt hij het CHOOSE-venster te zien.
- CHOOSE
- De gebruiker ziet twee opties met:
- “Create session”
- “Join session” en een inputveld voor een unieke URL
- Bij het drukken op “Create session” gaat de gebruiker naar het CREATE-scherm.
- Bij het drukken op “Join session” gaat de gebruiker, indien een juiste URL, naar SPACE.
- Bij een foutieve URL krijgt de gebruiker een notificatie dat de sessie niet gevonden is.
- CREATE
- De gebruiker kan een titel invoeren.
- Er wordt gecontroleerd of deze titel is ingevuld.
- De gebruiker krijgt een foutmelding als de titel leeg is.
- De gebruiker krijgt een foutmelding als de titel minder dan 5 karakters lang is.
- De gebruiker kan een *retrospective* model kiezen.
- Bij het drukken op de knop “Next” wordt er nogmaals gecontroleerd
- De gebruiker gaat naar SPACE als alles correct ingevuld is.
- De gebruiker krijgt een foutmelding met de fout.
- SPACE
    - Een ticket kan gesleept worden naar een andere kolom binnen de applicatie.
    - Een ticket kan gesleept worden naar een ander ticket om ze te groeperen.
    - Een groep kan gesleept worden naar een andere kolom binnen de applicatie.
    - Binnen een groep kan een ticket uit de groep gesleept worden waardoor dit wordt verwijderd van de groep.
    - Een ticket kan geüpvotet worden.
    - Bij een geüpvotet ticket moet visueel zichtbaar zijn dat dit geüpvotet is.
    - Een geüpvotet ticket kan nog eens gedownvotet worden waardoor dit niet meer geüpvotet is.
    - Bij een ticket dat niet meer geüpvotet is moet zichtbaar duidelijk zijn dat dit niet geüpvotet is.
    - De waarde van het aantal upvotes moet zichtbaar zijn bij het ticket.

    - Een ticket kan verwijderd worden; hierdoor verschijnt een pop-up waar de gebruiker moet antwoorden op de vraag of hij zeker is over deze actie.
    - Het ticket kan een actie krijgen door een gebruiker.
    - Bij het toevoegen van een actie krijgt de gebruiker een pop-upvenster waar hij verdere info moet invullen:
        - Titel
        - Deadline datum
        - Omschrijving
    - Een actie kan worden aangevinkt als deze compleet is, waardoor deze verdwijnt.
- Algemeen
    - Een gebruiker kan meteen naar een URL gaan om bij een SPACE terecht te komen waarmee de URL gelinkt is.

1.4.2 Non-functional requirements

- De applicatie mag niet meer dan één keer op vierentwintig uur down zijn wegens onderhoud.
- De frontend van de applicatie moet binnen de seconden op 1gb/s geladen en klaar voor gebruik zijn.
    - Uitgezonderd Edge en Internet Explorer
- De webapplicatie moet geschaald worden naar het scherm van de gebruiker.
- De webapplicatie moet bruikbaar zijn op schermen van 5 inch en groter.
- De webapplicatie moet beveiligd zijn tegen SQL-injectie.
- De API moet minimum 25 requests per seconde kunnen verwerken.
- De applicatie mag niet langer dan 24 uur onbereikbaar zijn.

1.5 Domeinkennis opdoen

Een applicatie kan niet gebouwd worden rond een onderwerp waar de programmeur niks vanaf weet. Daarom is er naast de analyse tijd voorzien om wat domeinkennis op te doen. Het onderwerp van deze stage zijn *retrospectives* binnen een agile Scrum omgeving. In dit gedeelte van het eindwerk wordt er uitgelegd wat *retrospectives* zijn en hoe ze gehouden worden.

1.5.1 Retrospectives

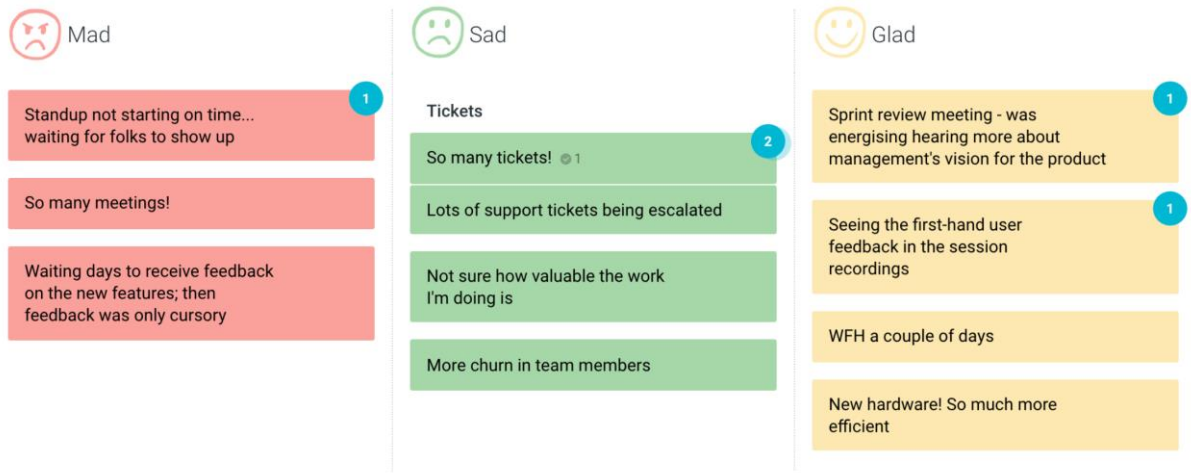
Een *retrospective* binnen een agile Scrum omgeving biedt het team een moment waar er gereflecteerd kan worden over de sprint die geweest is. Het houden van een *retrospective* kan op verschillende manieren gebeuren. Meestal gebeurt dit met Post-its op een bord of muur maar kan ook digitaal gebeuren. Er zijn een aantal bekende sjablonen die gebruikt worden om het proces te vergemakkelijken. Een *retrospective* verloopt in verschillende fases. Er is een fase voor het creëren van tickets en alle ideeën neer te pennen. Dan komt de volgende fase waar de gebruikers kunnen stemmen op de tickets. Meestal krijgt elke gebruiker vijf stemmen die verdeeld kunnen worden onder de tickets. Er kan ook meerdere keren gestemd worden op een ticket om het belang te benadrukken. In laatste fase worden de tickets besproken.

Een *retrospective* heeft een bepaalde duur waarin de tickets besproken moeten worden. Het is mogelijk dat de tickets niet allemaal worden besproken, daarom is een stemming een belangrijk onderdeel van een *retrospective*. De tickets met de meeste stemmen krijgen prioriteit om besproken te worden. Het doel is om een manier te bieden waar er uitgebreid gecommuniceerd kan worden met het team en fouten in de toekomst te vermijden en de dingen die goed gingen te behouden. Hieronder worden een paar veel gebruikte sjablonen besproken.

1.5.2 Mad Sad Glad

Zoals getoond in figuur 5, worden er bij de Mad Sad Glad methode drie kolommen voorzien voor elke titel: Mad, Sad en Glad waaronder het team zijn tickets kan plaatsen.

Bij de Mad categorie is het de bedoeling om iets aan te halen wat er mogelijk dwars zit. Een probleem die ervoor zorgt dat de gebruiker besproken wil hebben. Bij de Sad categorie is het de bedoeling om iets aan te halen wat de gebruiker teleur heeft gesteld en bij de Glad categorie kan worden aangehaald wat er goed ging tijdens de twee weken.

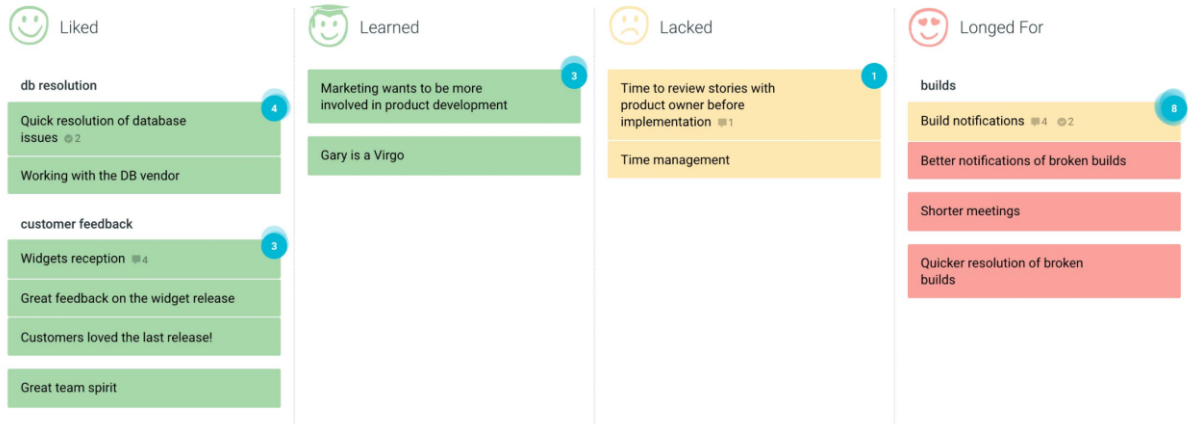


Figuur 5: Mad Sad Glad visueel voorbeeld
(figuur overgenomen van teamretro [4])

1.5.3 4Ls

De 4Ls methode bestaat uit 4 kolommen met als categorieën Liked, Learned, Lacked, & Longed For zoals getoond in figuur 6.

Liked staat voor wat de gebruiker goed vond doorheen de twee weken, waarbij bij Lacked het tegenovergestelde wordt genoteerd. Bij de Longed For categorie kan er aangehaald kan worden wat er ontbrak in het team. Learned is een categorie voor wat het team geleerd heeft tijdens de twee weken.



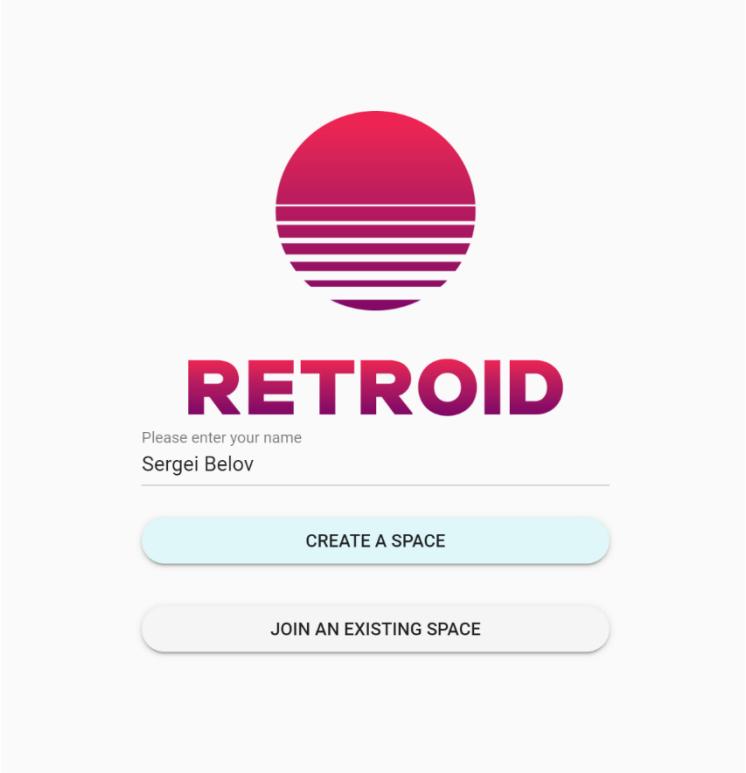
Figuur 6: 4Ls visueel voorbeeld
(figuur overgenomen van teamretro [5])

1.6 De applicatie

Één van de belangrijkste eisen waar de applicatie aan moet voldoen is dat het mobiel vriendelijk moet zijn. De applicatie moet zich kunnen aanpassen aan het toestel waarop het gebruikt wordt. Voor zowel laptops als computers, alsook Gsm's en tablets. Daarnaast is het doel van de applicatie om reeds aangemaakte tickets automatisch te laten groeperen. Als start wordt zowel de frontend als de backend opgezet en geconfigureerd. Om Vue.js op te zetten werd de tutorial gevolgd die beschikbaar is op de website van Vue.js zelf. [6] Naast Vue.js is het ook belangrijk dat de backend opgezet wordt. Dit gebeurt aan de hand van de Spring Initializr functie op de website van Spring zelf. [7] Voor de frontend wordt Vuetify.js geïnstalleerd via de NPM-commando van Node.js. De installatie gebeurt vanzelf en er hoeft niks niet meer geconfigureerd te worden. De eerste pagina die wordt aangemaakt is het startscherm van de applicatie.

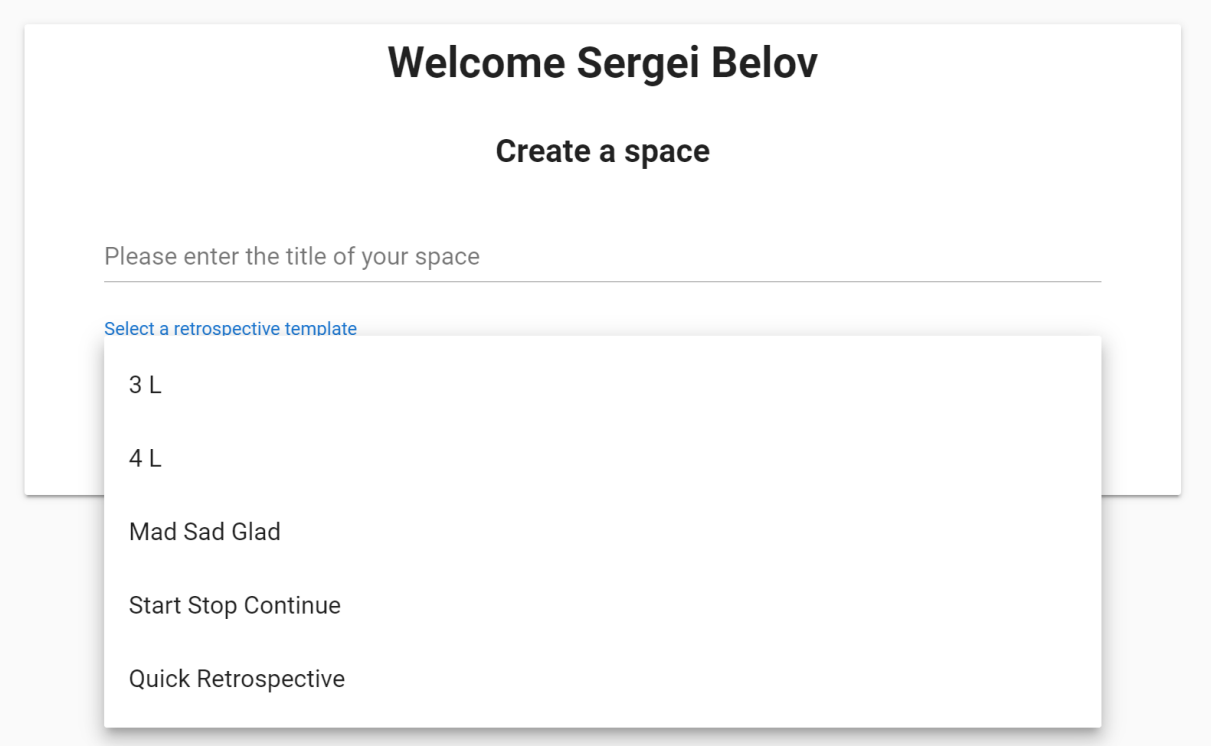
1.6.1 Het startscherm

In het startscherm moet de gebruiker zijn naam ingeven voordat hij door mag gaan. In figuur 7 is het design te zien van de pagina. In het startscherm ligt de focus op functionaliteit en een simpel design die de gebruiker zo min mogelijk afleidt. De gebruiker krijgt de keuze of hij of zij een *retrospective* omgeving wil aanmaken of wilt deelnemen aan een reeds bestaande omgeving. Bij het creëren van



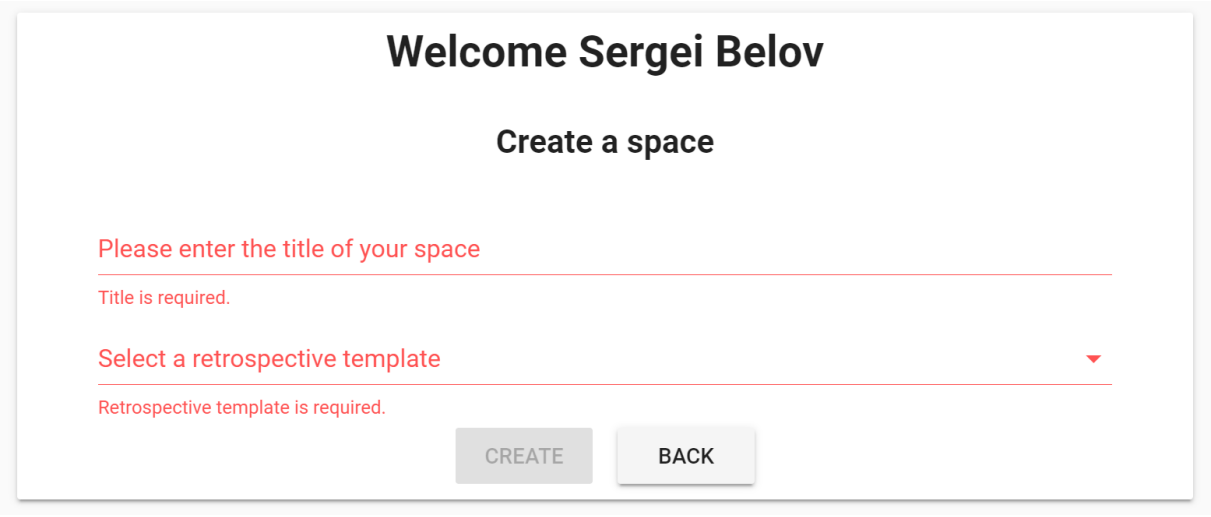
Figuur 7: Startscherm

een nieuwe omgeving dienen er een aantal instellingen geconfigureerd te worden. In figuur 8 is te zien wat er allemaal ingegeven moet worden voordat er een omgeving aangemaakt kan worden.



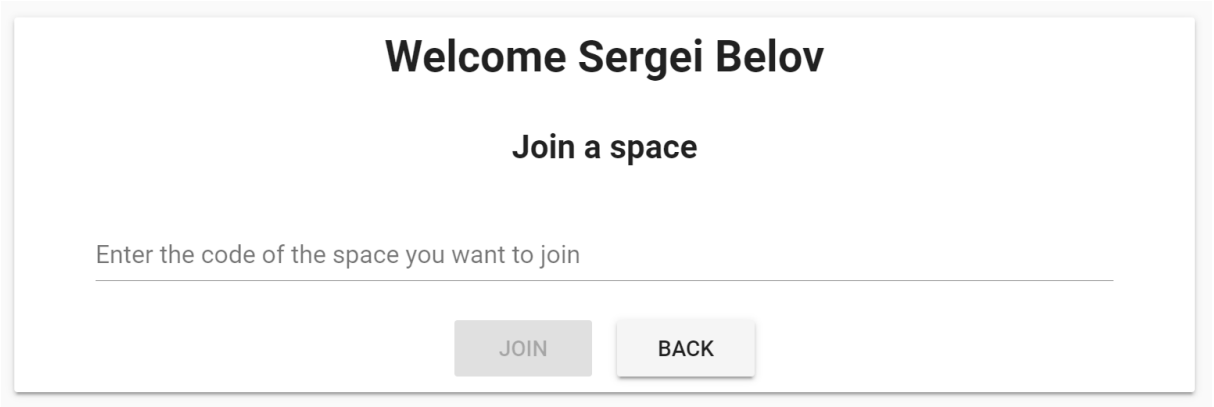
Figuur 8: Creëer een omgeving pagina

De gebruiker dient een titel in te geven voor de omgeving. Daarnaast moet er ook een keuze gemaakt worden welke retrospective template er gebruikt gaat worden. Er is een keuze uit 3L, 4L, Mad Sad Glad, Start Stop Continue en Quick Retrospective. Als de nodige velden niet ingevuld zijn wordt de gebruiker gewaarschuwd en kan er geen omgeving aangemaakt worden, zoals weergegeven in figuur 9. Eens alle nodige data ingegeven is kan er een omgeving aangemaakt worden.



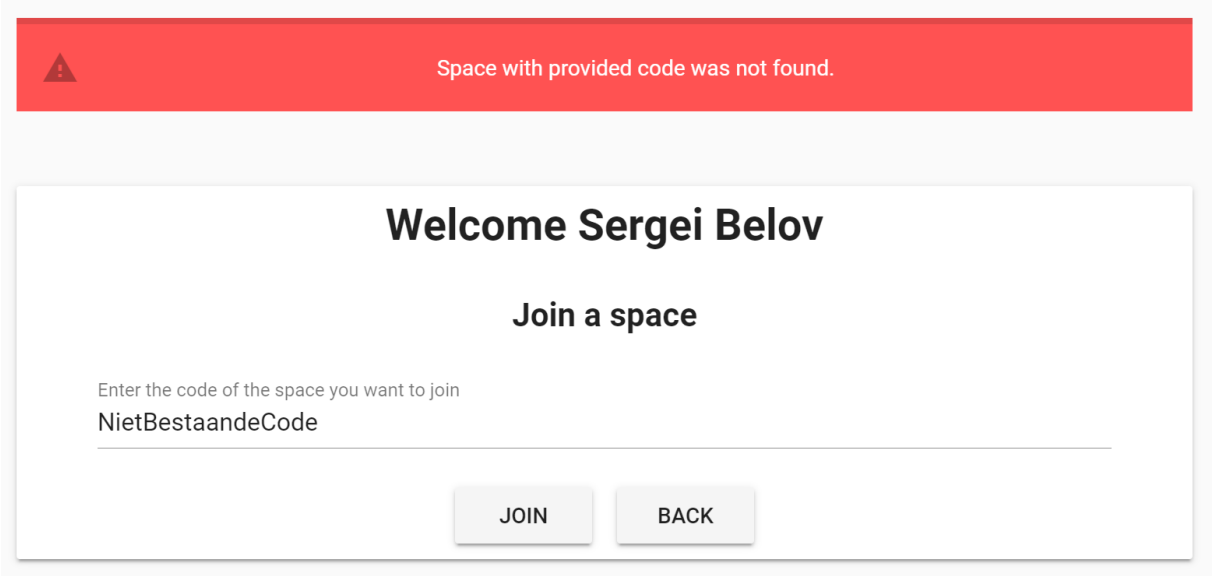
Figuur 9: controle ingevoerde data

Naast het creëren van een omgeving, kan een gebruikt zich toevoegen aan een bestaande omgeving zoals te zien is in figuur 10. Hierbij dient de gebruiker een unieke code in te geven die te vinden is in de titelbalk van een bestaande omgeving.



Figuur 10: Join a space pagina

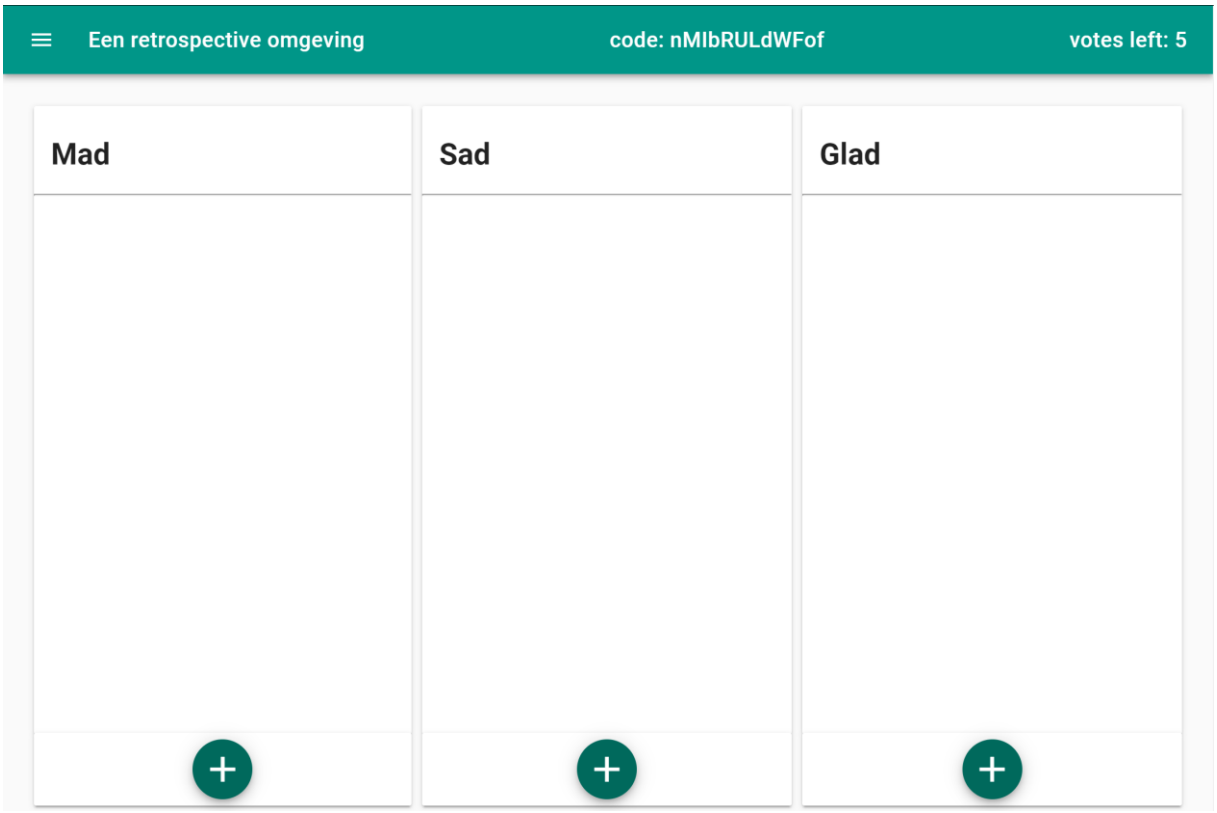
Als er een code wordt ingegeven die niet bestaat wordt de gebruiker gewaarschuwd aan de hand van een bericht. Deze is duidelijk zichtbaar aan de hand van een rode error veld zoals te zien is in figuur 11.



Figuur 11: error van niet gevonden omgeving

1.6.2 De retrospective omgeving

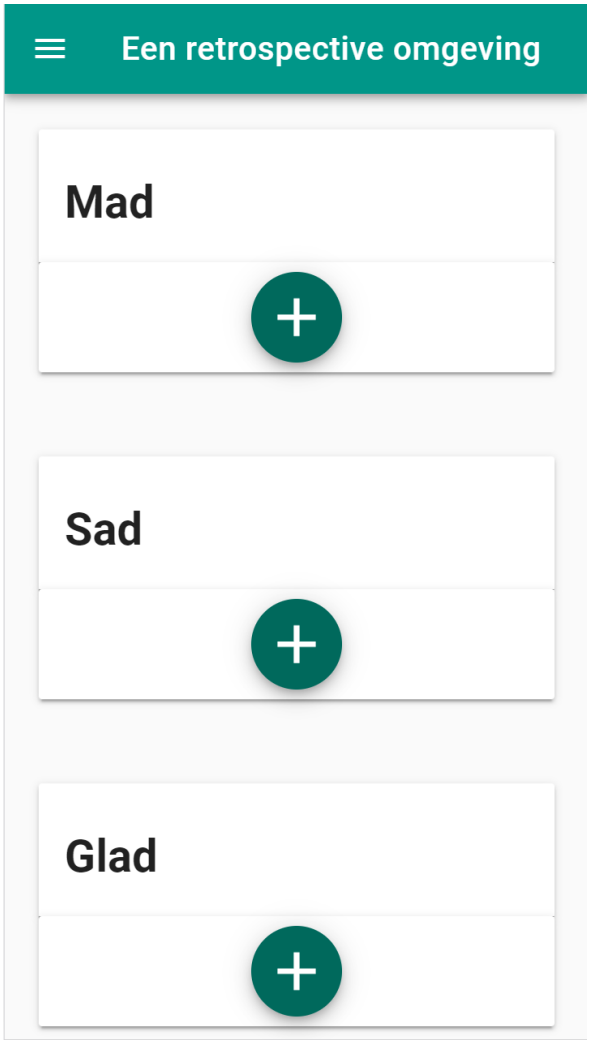
Bij het succesvol aanmaken van of deelnemen aan een omgeving, wordt de gebruiker verwezen naar de omgeving. Hierbij is er ook gestreefd naar een design die simpel in gebruik is en duidelijk is. In figuur 12 is een weergave van een lege retrospective omgeving.



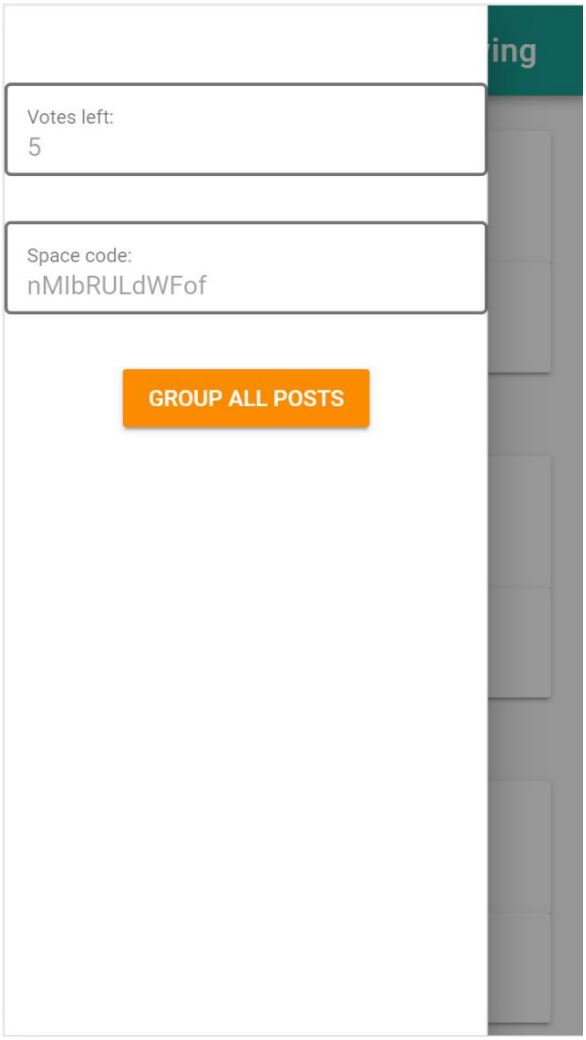
Figuur 12: Een retrospective omgeving (Mad, Sad, Glad)

De titel wordt links weergegeven en maakt plaats voor de code van de omgeving. Daarnaast is een visuele representatie van het aantal keren dat de gebruiker nog mag stemmen. Voor gebruiksvriendelijke redenen is er gekozen om de code van omgeving klikbaar te maken. Zo hoeft de gebruiker niet de code zelf te kopiëren en wordt deze automatisch in zijn clipboard gezet.

De mobiele versie van de omgeving ziet er wat anders uit. Drie horizontale kolommen op het scherm van een GSM of tablet is niet overzichtelijk. Daarom, zoals te zien is in figuur 14, worden de kolommen onder elkaar geplaatst.



Figuur 14: Mobiele versie retrospective omgeving

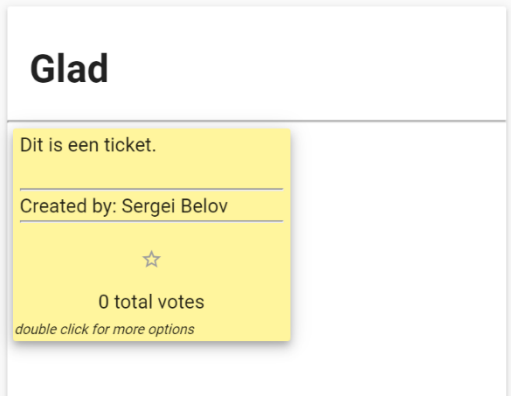


Figuur 13: Verplaatsing aantal stemmen en code

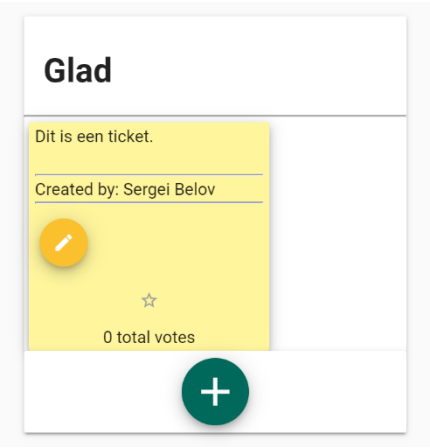
Wegens plaats gebrek is er in de titelbalk van het platform niet genoeg plaats voor alle belangrijke informatie. Daarom, zoals aangetoond in figuur 13, wordt op de mobiele versie van de omgeving, het aantal stemmen en de omgevingscode verplaatst naar de hamburger menu. Net zoals de desktopversie van de site, is hier ook de code klikbaar en heeft het dezelfde functionaliteiten. Eens de omgeving aangemaakt is, kunnen er tickets aangemaakt worden op het bord. Dit gebeurt aan de hand van de plus knoppen die zichtbaar zijn onder elk kolom. Eens er op de knop wordt gedrukt, verschijnt er een pop-up scherm met een voorbeeldweergave van de ticket. Figuur 15 bevat een visuele weergave van deze pop-up scherm.

Figuur 15: pop-up voor het creëren van een ticket

In het ticket is een veld voorzien waar de gebruiker zijn tekst kan schrijven, met een minimum van vier tekens. Als de tekst niet aan deze regel voldoet, wordt er een melding getoond en krijgt het tekstveld een rode rand. De gebruiker kan het ticket niet aanmaken zolang deze niet aan de voorwaarden voldoet. Eens een ticket is aangemaakt, wordt deze in de gekozen kolom geplaatst. Als voorbeeld wordt er hier een ticket gemaakt onder de Glad kolom zoals te zien is in figuur 17, of voor de mobiele weergave figuur 16.

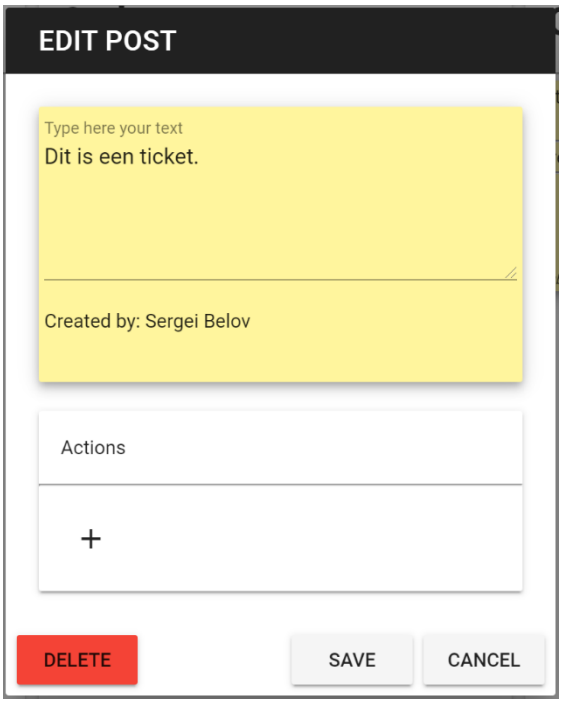


Figuur 17: aangemaakt ticket



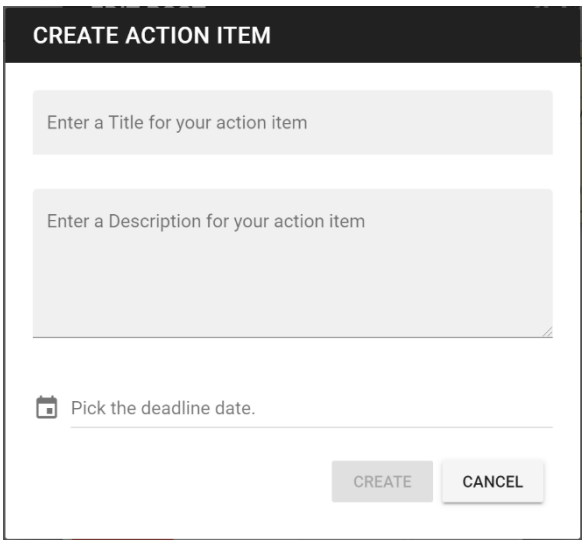
Figuur 16: aangemaakt ticket, mobiele weergave

Door te dubbelklikken worden bijkomende mogelijkheden getoond. Omwille van usability werd bij de mobiele versie gekozen om deze mogelijkheden aan te bieden via een knop en niet via dubbelklikken.

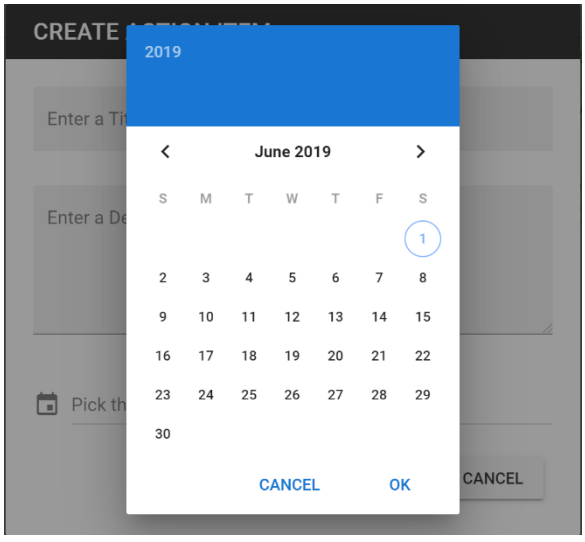


Figuur 18: editeer ticket

Figuur 18 toont een weergave van de edit pop-up. Hierbij is er weer een voorbeeldweergave van het ticket zichtbaar en een lijst met actiepunten. Het ticket kan hier ook verwijderd worden, waardoor er een nieuwe pop-up verschijnt die vraagt of de gebruiker zeker is. Actiepunten kunnen aangemaakt worden door op de plus te drukken onder “Actions”. Hierbij verschijnt er een nieuwe pop-up voor het aanmaken van een actiepunt. In figuur 19 wordt getoond hoe deze pop-up eruitziet, en in figuur 20 wordt getoond hoe een datum gekozen kan worden.

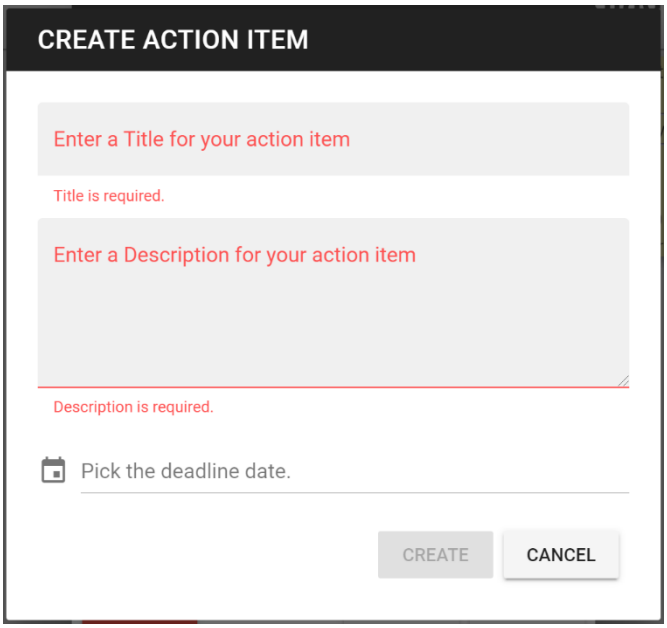


Figuur 19: Aanmaken actiepunt



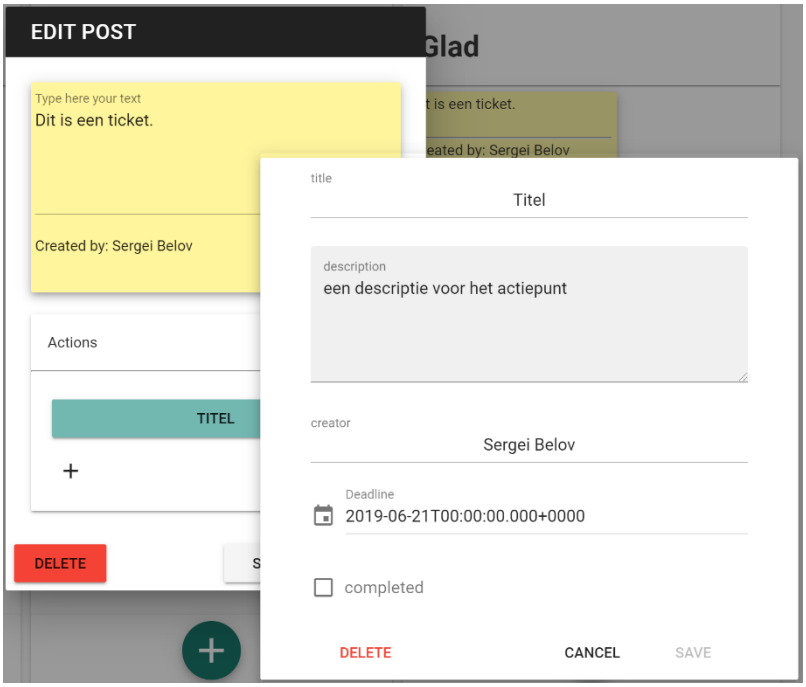
Figuur 20: Keuze datum

Ook voor het aanmaken van actiepunten wordt er gekeken of de invoervelden voldoen aan de opgelegde regels. Hierbij moet elk veld ingevuld zijn, waaronder de deadline datum. Als deze te kort zijn wordt de gebruiker gewaarschuwd zoals zichtbaar in figuur 21.



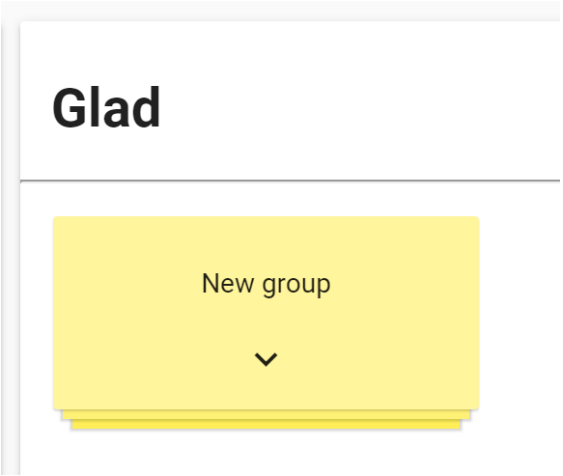
Figuur 21: input regels

De titel dient minimum vier tekens lang te zijn en maximum 25. De descriptie van het actiepunt moet minimum 10 tekens lang zijn. Eens het actiepunt aangemaakt is, is deze zichtbaar bij het edit venster tussen de lijst van actiepunten. Er kan op geklikt worden om meer informatie te verkrijgen en het actiepunt te wijzigen. Deze handeling wordt aangetoond in figuur 22.

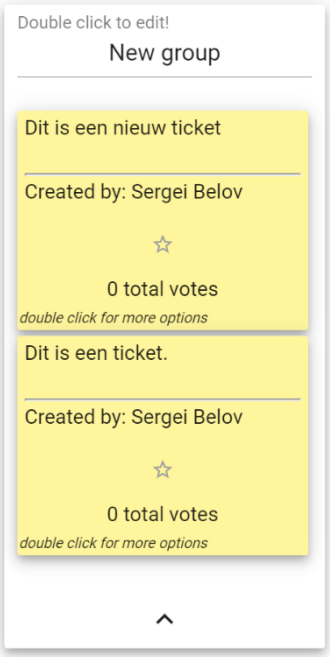


Figuur 22: actiepunt detail weergave

De applicatie is voorzien van een sleep functie voor de aangemaakte tickets. Als er een ticket onder de verkeerde kolom wordt geplaatst kan deze zo versleept worden. Ook kunnen aangemaakte tickets hiermee gegroepeerd worden door een ticket over een ander ticket te slepen. Een manueel aangemaakte groep krijgt als titel “New group”. In figuur 24 is een weergave te zien van een groep binnen de applicatie, en in figuur 23 als deze groep opengeklapt is.



Figuur 24: gesloten groep



Figuur 23: open groep

De titel kan aangepast worden door dubbel te klikken op de titelbalk, waarna in het veld geschreven kan worden. Als er op enter geduwd wordt, wordt de nieuwe titel opgeslagen. Tickets kunnen verder nog aan de groep toegevoegd worden door ernaar gesleept te worden. Een groep verwijderen gebeurt door alle tickets uit de groep te slepen.

1.6.3 Automatisch groeperen

Het uiteindelijke doel is een mogelijkheid te voorzien om tickets automatisch te laten groeperen. Hierbij wordt beroep gedaan op CoreNLP, een *natural language processor* geschreven door Stanford University en Semilar, een Java *library* om berekeningen uit te voeren op zinnen en woorden. Als algoritme voor de zinnen, is er gekozen voor de LSA-methode, wat staat voor *Latent Semantic Analysis.* Hierbij worden zinnen of woorden met elkaar vergeleken op basis van wat het woord of zin betekend. Dit is anders dan de meeste methodes die vandaag beschikbaar zijn. De andere methodes berekenen hoever de gegeven zinnen van elkaar afwijken. Dit op basis van de woorden die ze gebruiken en niet de betekenis daarachter. Meer uitleg over LSA en CoreNLP is te vinden in het onderzoek gedeelte van dit eindwerk. In dit gedeelte wordt er besproken hoe het uiteindelijk geïmplementeerd is.

De bedoeling van automatisch groeperen is tickets, die op elkaar lijken op basis van betekenis, samen te zetten in een groep. Om het niet ingewikkeld te maken, worden er nieuwe groepen aangemaakt. Er wordt niet gekeken of de tickets misschien ook in andere, reeds bestaande groepen passen. Ook wordt er niet constant gezocht om te groeperen, maar wordt dit gedaan aan de hand van een trigger. Dit om de performantie van de applicatie zo hoog mogelijk te houden. De trigger die gebruikt wordt is een simpele knop die een verzoek stuurt naar de backend.

Voor de implementatie wordt er bij het creëren van een ticket het sentiment berekend en de zelfstandige naamwoorden ontleed door CoreNLP. De sentiment analyse wordt uitgevoerd zodat tickets die onder een verkeerde kolom staan, niet mee gegroepeerd worden met andere tickets. Deze wordt op voorhand berekenen zodat het uiteindelijke groeperen minder tijd in beslag neemt. De zelfstandige naamwoorden worden opgeslagen als mogelijke tags van het ticket. Eens er op de trigger knop geduwd wordt, wordt er een GET-verzoek gedaan naar de backend om de geüpdatet omgeving op te vragen. Terwijl dat frontend wacht op resultaat, wordt er in de backend een zelfgeschreven algoritme uitgevoerd die te zien is in figuur 25.

```
fun group(posts: MutableList<Post>): MutableList<MutableList<Post>> {
    val similarPosts : MutableList<MutableList<Post>> = mutableListOf()

    // Double for loop to avoid comparing same posts with each other
    for (i :Int in 0 until posts.size - 1) {
        val groupList = ArrayList<Post>()
        groupList.add(posts[i])
        for (j :Int in i + 1 until posts.size) {
            // avoid comparing posts that are already grouped
            if (posts[i].groupID == null) {
                if (posts[j].groupID == null) {
                    if (posts[i].currentLane === posts[j].currentLane) {
                        if (posts[i].sentiment == posts[j].sentiment) {
                            val similarity :Float? = lsaService.computeLSA(posts[i].content!!, posts[j].content!!)
                            if (similarity!! >= 0.6) {
                                groupList.add(posts[j])
                            }
                        }
                    }
                }
            }
        }
        similarPosts.add(groupList)
    }
    return similarPosts
}
```

Figuur 25: Algoritme automatisch groeperen (code)

Hierbij wordt er gebruik gemaakt van twee for-lussen om te voorkomen dat tickets met zichzelf vergeleken worden. Dan wordt er gekeken of het ticket reeds tot een groep behoord. Als deze tot een groep behoord wordt deze niet meer vergeleken. Ook wordt er gekeken of de tickets tot dezelfde kolom behoren. Als dat zo is, gaat er gekeken worden naar het sentiment van de tickets. Zoals eerder aangehaald, is er een mogelijkheid dat tickets onder een verkeerde kolom geplaatst zijn. Om te voorkomen dat er groepen aangemaakt worden met positieve tickets en negatieve tickets tezamen, wordt er ook naar het sentiment gekeken. Als alle tickets aan de opgesomde voorwaarden voldoen, wordt er een LSA-berekening uitgevoerd door de Semilar *library*. Dit retourneert een waarde tussen nul en één. Hoe hoger de geretourneerde waarde is, hoe groter de kans de inhoud van de tickets hetzelfde betekenen. In de code is er een drempel gedefinieerd van 0.6, oftewel 60% gelijkheid. Alle tickets die hoger, of gelijk aan 60% scoren, worden samen in een lijst gezet. De lijst wordt na deze methode overlopen en als er lijsten zijn met meer dan één ticket, wordt hiervan een groep gemaakt.

Om te vermijden dat de aangemaakte groepen dezelfde titel krijgen, is er een functie voorzien die aan de hand van de reeds berekende tags, een titel genereerd. Deze functie is weergegeven in figuur 26.

```
findSuitableTitleForGroup(toBeGroupedPosts: List<Post>): String {
val allTags = ArrayList<String>()
val uniqueAllTags: Set<String>
var mostFrequentTag = ""
var mostFrequent = 0

// add all tags to a list
for ((_ : String , _ : Int , _ : String? , _ : MutableList<Action>? , _ : User? , _ : Int? , _ : String? , tags : MutableList<String>? ) in toBeGroupedPosts){
    for (tag : String in tags!!) {
        allTags.add(tag)
    }
}

uniqueAllTags = HashSet(allTags)

for (tag : String! in uniqueAllTags) {
    val frequency : Int = Collections.frequency(allTags, tag)
    if (frequency > mostFrequent) {
        mostFrequent = frequency
        mostFrequentTag = tag
    }
}

if (uniqueAllTags.isEmpty()) {
    mostFrequentTag = "New Group"
}

return mostFrequentTag
```

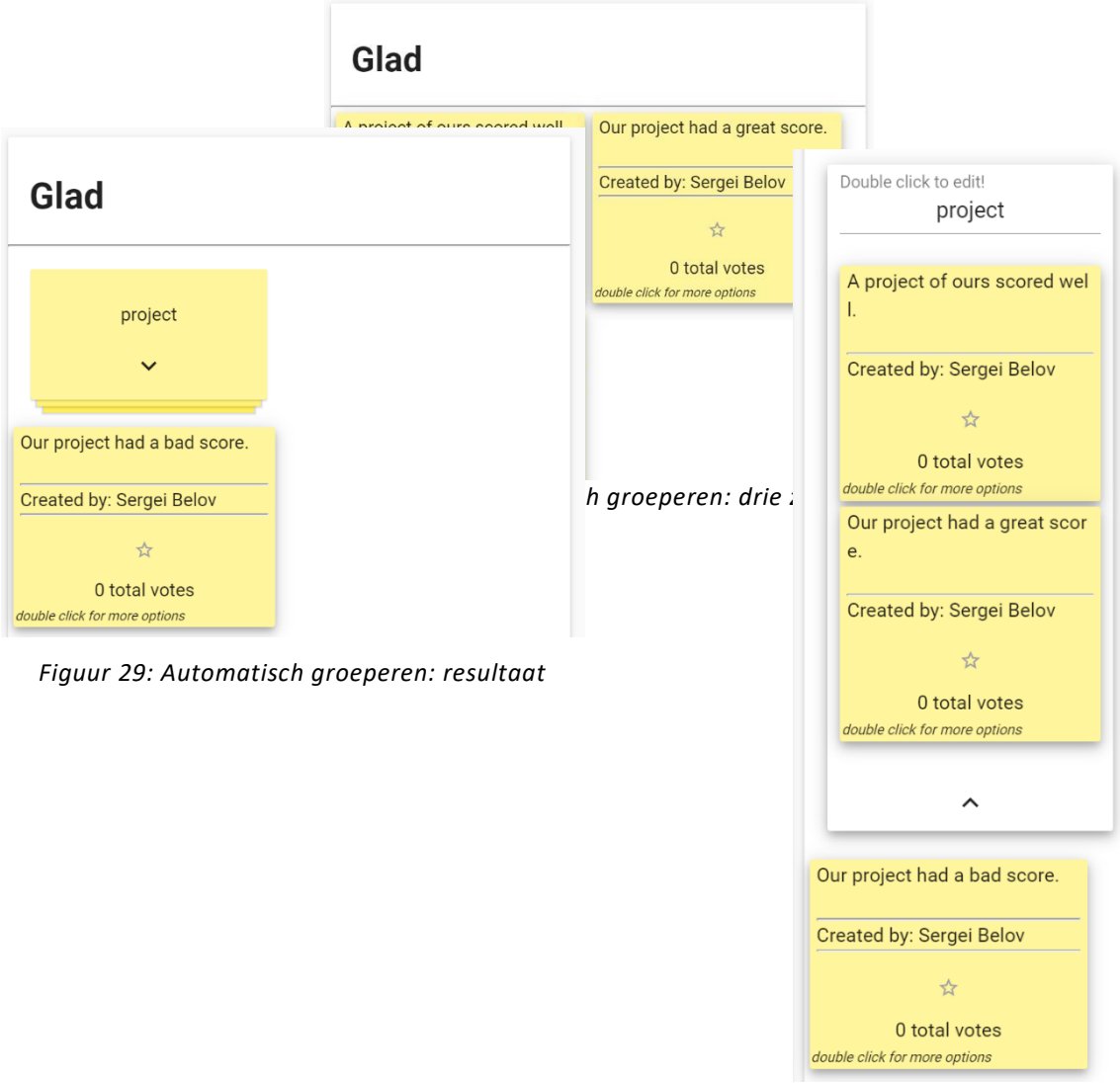
Figuur 26: Methode voor het berekenen van een titel (code)

Hierbij worden alle tags opgehaald van de tickets in de groep, en samen gezet in één lijst. Er wordt ook een andere *HashSet* klaargezet met alle unieke tags van de tickets in de groep. Eens de twee lijsten opgesteld zijn, wordt er berekend welke tag het meeste voorkomt tussen alle tags. Deze tag wordt dan gebruikt als titel van de groep.

Als voorbeeld worden er in de frontend drie tickets aangemaakt met volgende inhoud:

- *A project of ours scored well.*
- *Our project had a great score.*
- *Our project had a bad score.*

Twee van de drie tickets betekenen contextueel hetzelfde terwijl de laatste zin het tegenovergestelde betekend van de eerste twee zinnen. De zinnen zijn in het Engels vanwege de gelimiteerde ondersteuning van de Nederlandse taal onder NLP's. Een visuele weergave is te zien in figuur 27.



Figuur 29: Automatisch groeperen: resultaat

Figuur 28: Automatisch groeperen: resultaat opengevouwen

Als er op de knop "*Group all posts*" wordt gedrukt, te vinden in de hamburger menu, start het automatisch groeperen. Het resultaat van deze handeling is te zien in figuur 29 en in figuur 28 is de opengevouwen groep te zien.

De tickets zijn met elkaar gegroepeerd volgens de voorwaarden van de zelfgeschreven algoritme. Hierbij is te zien dat een ticket zich in dezelfde kolom bevond, maar een andere sentiment waarde had verkregen, niet gegroepeerd is. De eerste twee tickets zijn onder éénzelfde groep geplaatst met als titel "project", een titel die gegenereerd is met de tags die de tickets met zich meedragen.

II.Onderzoekstopic

1 Onderzoeksvragen

1.1 Omschrijving

In dit deel van de tekst wordt onderzocht of het mogelijk is om zinnen te ontleden en op basis van inhoud te groeperen. Verder wordt aan de hand van de inhoud het sentiment van de zin afgeleid en wordt er geprobeerd om de zin te groeperen volgens sentiment. Om te werken met taal moet er gezocht worden naar een *natural language processor*. Er worden twee oplossingen met elkaar vergeleken met name CoreNLP, ontworpen door Stanford University, en OpenNLP, ontworpen door Apache. Het doel van een *Natural language processors,* of NLP in het kort, is het lezen, ontcijferen en begrijpen van menselijke taal.

Hierbij worden de volgende vragen beantwoord:

Is het mogelijk om met behulp van *machine learning* zinnen te groeperen op basis van inhoud?
Is het ook mogelijk om het sentiment te bepalen van zinnen?
Is CoreNLP sneller en accurater dan OpenNLP?

2 Onderzoeksmethode

2.1 Aanpak

Voor de aanvang van de experimentele fase van het onderzoek moet er eerst een dataset aangemaakt worden. De dataset bestaat uit een lijst van één of meerdere zinnen. De *natural language processors* maken gebruik van de aangemaakte dataset om de context en het sentiment te ontleden. De gegenereerde data van beide NLP's wordt vergeleken met elkaar. Er wordt gekeken of de juiste inhoud gevonden is van de zinnen en het sentiment accuraat berekend is. De tijd die elke NLP nodig heeft om tot het resultaat te bekomen wordt in kaart gebracht.

Ten slotte volgt er een algemene vergelijking en wordt de bekomen data in een tabel gezet. Daarna wordt er een persoonlijk cijfer gegeven op basis van snelheid, accuraatheid en hoe gebruiksvriendelijk de gebruikte technologieën zijn.

3 Literatuurstudie

Vandaag wordt er meer en meer gebruikgemaakt van AI en *machine learning* om hedendaagse problemen op te lossen of te vergemakkelijken. Grote bedrijven zoals Google en Microsoft investeren miljoenen in het onderzoek naar het gebruik van AI en *machine learning* om deze technologieën beter te kunnen benutten. Mogelijke doeleinden van deze technologie zijn landbouw, financiën, onderwijs en zo veel meer. Dit onderzoek focust op het gebruik van dergelijke frameworks om taal te ontleden met als doel de inhoud te begrijpen en het sentiment te verkrijgen om daarna zinnen inhoudelijk met elkaar te vergelijken. Maar één van de grootste struikelpunten aan heel dit verhaal is de complexiteit van taal. Taal is veel uitgebreider dan dat erbij stil gestaan wordt. Volgens Venkat Srinivasan, CEO van RAGE Frameworks, zijn er drie uitdagingen die overkomen moeten worden als er met taal gewerkt wil worden namelijk de manier waarop we taal zien, context en de gedachten erachter. Naast deze uitdagingen is er ook de extra uitdaging door het gebruik van synoniemen en proxy's. [8] Zo kan er binnen een conversatie verwezen worden naar objecten of personen die eerder in de conversatie zijn aangehaald. Of kan een gesprek gehouden worden over een onderwerp die al eerder besproken is geweest. Daardoor kan een machine moeilijk uitmaken waar het overgaat. Dit is de context binnen een conversatie, en omdat een mens voorafgaande kennis heeft of een link kan leggen, kan een ongetrainde computer er niet aan uit. Dit heeft ook deels te maken met de manier hoe taal wordt gezien door een machine. Veel gebruikte technieken om taal te analyseren, analyseren elk woord individueel waarna er wiskundige berekeningen worden uitgevoerd. Uit de uitgevoerde berekeningen wordt er gezocht naar patronen om te berekenen wat er bedoeld wordt. Met deze technieken wordt taal niet gezien als "taal" maar als data. [8]

Extra uitdagingen die komen kijken bij het vinden van de context binnen een tekst zijn de woorden die gebruikt worden. Woorden hebben vaak ook andere betekenissen en betekenen dan ook iets anders afhankelijk van hun plaatsing in een zin. Proxy's binnen een tekst maken het ingewikkeld. Waar er in een tekst Amazon wordt gebruikt, weet een mens dat het over een online webwinkel gaat, maar een computer die niet getraind zal de link hier niet in kunnen terugvinden. Naast context is het ook belangrijk om te begrijpen met welke gedachtegang de tekst geschreven is. In de meeste toepassingen binnen AI en *machine learning* is dit geen vereiste zoals bijvoorbeeld bij *image recognition*. Om *image recognition* uit te kunnen voeren wordt er door het framework niet gekeken naar de gedachtegang die er achter een afbeelding zit.

Ondanks de uitdagingen zijn er al vooruitgangen gemaakt met *machine learning* en taal. Doorheen de jaren zijn chatbots meer en meer in populariteit gestegen. Webwinkels implementeren ze voor hun klantendiensten. Hoewel de technologie nog niet vlekkeloos werkt, is IBM er verzekerd van dat de toekomst AI gaat zijn. Naast chatbots wordt de technologie ook gebruikt voor persoonlijk assistentes. Een aantal voorbeelden hiervan zijn Siri, ontwikkeld door Apple, Google Assistant die met Siri concurreert en Cortana die ontwikkeld is door Microsoft. Maar er groeit meer teleurstelling onder de gebruikers in de technologie, ondanks alle vooruitgangen.

"Chatbots were super-hot and now not-quite-as-much."
"They're seeing some early successes in a few narrow applications like customer support and smart appliances, but people are getting frustrated because they have overly high expectations." [9]

Volgens IBM is één van de hoofdredenen van alle teleurstelling de snelheid waarmee de machines kunnen antwoorden. De verwachting wordt geschept dat de machines ook standaard conversaties kunnen houden. Emotie en empathie verstaan en ermee antwoorden is ook een uitdaging voor de huidige technologie.

De vraag die nog onbeantwoord is, is hoe *machine learning*-frameworks op dit moment taal analyseren. Hiervoor moet er gekeken worden naar *natural language processors, machine learning*-frameworks die zich specialiseren in taal. Volgens Tim Mohler van Lexalytics is het doel van een NLP om volgende vier vragen te beantwoorden. Wie spreekt, Waar spreken ze over, Hoe voelen ze zich erbij en waarom voelen ze zich er zo bij. De laatste vraag is een contextuele vraag. Hiervoor wordt er een manier gezocht om de context van een zin te vinden. De basis van alles context is volgens Tim Mohler het zelfstandig naamwoord. Deze wordt ook gebruikt voor het toepassen van *named entity recognition*, maar NER gaat een stap verder en probeert het gevonden woord te classificeren onder een thema. Een aantal voorbeelden die Tim Mohler aanhaald is een Cessna en een vliegtuig. Beide kunnen geclassificeerd worden onder het thema "transport". *Het artikel "Context Analysis in NLP: Why It's Valuable and How It's Done" bespreekt vier mogelijke manieren waarop context gevonden kan worden in een zin of een tekst.* [10]

De eerste manier die wordt besproken zijn de n-grams, waarbij een zin wordt ontleed in groepen van woorden. Het aantal woorden in een groep is “n”. Drie veel voorkomende n-grammen zijn de mono-gram, bi-gram en de tri-gram, elk gepaard met sterktes en zwaktes. In het artikel wordt de voorbeeldzin *“President Barack Obama did a great job with that awful oil spill.”* gebruikt om de drie n-grams voor te stellen. In figuur 30 zijn de resultaten te zien van een mono-gram, bi-gram en een tri-gram analyse.

Mono-grams	Bi-grams	Tri-grams
a	a great	a great job
awful	awful oil	awful oil spill
barack	barack obama	barack obama did
did	did a	did a great
great	great job	great job with
job	job with	job with that
obama	obama did	obama did a
oil	oil spill	president barack obama
president	president	that awful oil
spill	barack	with that awful
that	that awful	
with	with that	

Figuur 30: Voorbeeld monogram, bigram en trigram [10]

Bij de mono-gram is het al snel duidelijk dat er veel te weinig informatie mee komt omdat de zin woord voor woord wordt opgesplitst. Mono-grams worden daarom zelden gebruikt bij het ontleden van zinnen. Volgens het artikel zijn tri-grammen ook niet van toepassing omdat ze juist veel te veel informatie bieden. Uiteindelijk worden tri-grams wel nog gebruikt voor het analyseren van zinnen, maar niet zo frequent als bi-grams. Bi-grams ontleden de woorden op de juiste lengte om genoeg context te bieden zodat er sentiment gevonden kan worden. Met het gebruik van n-gram kunnen er woordgroepen zijn die geen waarde hebben voor het achterhalen van context noch sentiment. Zoals bijvoorbeeld de bi-gram “voor het”. Een oplossing hiervoor is een dataset aanmaken met daarin een lijst van woorden die geen bijgevoegde waarde bieden in een zin. Zo kunnen deze woorden gefilterd worden voor analyse. De voordelen van het gebruik van n-grams zijn dat het makkelijk te begrijpen is en dat er een inzicht kan geschept worden op verschillende niveaus door het gebruik van verschillende n-grams. De nadelen zijn dat een dataset, met woorden die geen waarde bevatten, een noodzaak is en dat er geen overzicht is naar het belang van een woord binnen de zin of tekst.

Een veelgebruikte manier en ook tweede in de lijst is volgens het artikel “*Noun Phrase Extraction*” waarbij er wordt gekeken naar het zelfstandig naamwoord en de woorden er rond. Het wordt veel gebruikt in *natural language processors* om te achterhalen waarover het gaat. *Noun phrase extraction* maakt gebruik van spraak patronen om de relevantie van een zin binnen een tekst te achterhalen. Een nadeel van deze methode is dat ze beperkt is tot de woorden die gebruikt worden in de tekst.

Themes and Theme Extraction with Relevancy Scoring is een methode waarbij het onderwerp bepaald wordt van een tekst of document. Het maakt gebruik van *noun phrasing* maar voegt een contextuele relevantie score toe. In plaats van zinnen apart te nemen, worden de zinnen naast elkaar geplaatst op basis van het zelfstandig naamwoord en wordt er een score geplakt op de ketting. De hoogst scorende ketting krijgt de hoogste relevantie score. *Theme extraction* erft dezelfde voor- en nadelen over van *noun phrase extraction*, maar voegt er een aantal voordelen bij. Zo worden er scores gegeven aan de contextuele relevantie en belang van zinnen.

De laatste methode die wordt besproken in het artikel noemt *facets* die geen gebruik maakt van *noun phrase extraction*. Ze wordt gebruikt wanneer er geen goede *noun phrases* zijn. De zinnen die hier in aanmerking komen zijn bijvoorbeeld recensies waar de zinnen kort en bondig zijn. De zinnen dragen dan wel betekenis en hebben een doel maar kunnen niet door een *theme extraction* gefilterd worden. Het is dan ook bijvoorbeeld handig om een groot aantal reviews over een product of service door een *facet* te laten analyseren waarna er een tabel weer gegeven kan worden om meer inzicht te krijgen over het onderwerp.

Er kan geconcludeerd worden dat de technologie om taal te analyseren aan het verbeteren is. Er wordt actief onderzoek verricht om bestaande methodes te verbeteren en nieuwe, betere methodes te verzinnen. Voor dit onderzoek wordt er gekeken om wat er hier geleerd is toe te passen en de inhoud van korte zinnen te verkrijgen, het sentiment te berekenen. Dit met behulp van een aantal NLP’s die voorzien zijn van uitgebreide datasets om te trainen of NLP’s die reeds getraind en klaar voor gebruik zijn.

4. Sentiment analyse

4.1 Introducuctie

Het sentiment van een zin vinden is verre van triviaal. Veel hangt af van de woorden die gebruikt worden en waar ze gebruikt worden of de zin positief gestemd is of negatief. Daarnaast zijn er zinnen die neutraal zijn en geen sentiment dragen. In dit gedeelte gaan we in praktijk kijken hoe OpenNLP en CoreNLP deze uitdaging aangaan. Gedetailleerd wordt er besproken welke stappen genomen worden om een resultaat te verkrijgen. De eerste NLP die vergeleken wordt is OpenNLP, een NLP opgenomen door Apache via de Apache Incubator programma. Daarna wordt er gekeken naar CoreNLP waarna de resultaten samen worden vergeleken en een conclusie volgt. Er is een lijst gemaakt met 100 willekeurige Engelse zinnen als test data voor beide NLP's.

4.2 OpenNLP

Na uitgebreid onderzoek is gebleken dat OpenNLP geen ingebouwde functie heeft om sentiment van een document of zin te berekenen. De ingebouwde functie is sinds 13 maart 2016 in ontwikkeling en is nog niet uitgebracht zoals te zien is in figuur 31.

Figuur 31: OpenNLP sentiment jira status [11]

Ondanks de afwezigheid van een ingebouwde functie voor sentiment analyse, is er een andere manier waarop het sentiment berekend kan worden. Via de OpenNLP *Document Categorizer* is het mogelijk om twee verdelingen te maken, positief en negatief. Daarna kan de *categorizer* van OpenNLP getraind worden aan de hand van eigen training data. Voor dit voorbeeld is er gebruikgemaakt van een voorgemaakte dataset te vinden in het artikel 'sentiment analysis using OpenNLP Document Categorizer', geschreven door Leonard Giura. [12] De dataset bevat 100 tweets die reeds geclassificeerd zijn als positief of negatief. Bij het opstarten van het project moet de *categorizer* getraind worden vooraleer deze gebruikt kan worden. Object georiënteerd gericht wordt een Java klasse gemaakt genaamd 'Sentiment'. De code voor het trainen van de *categorizer* is weergegeven in figuur 32.

```java
public Sentiment() {
    try {
        InputStreamFactory isf = () -> new FileInputStream( name: "en-sentiment.train");

        ObjectStream<String> lineStream =
                new PlainTextByLineStream(isf,  charsetName: "UTF-8");
        ObjectStream<DocumentSample> sampleStream = new DocumentSampleStream(lineStream);

        TrainingParameters params = new TrainingParameters();
        params.put(TrainingParameters.CUTOFF_PARAM, Integer.toString( i: 0));
        params.put(TrainingParameters.ALGORITHM_PARAM, NaiveBayesTrainer.NAIVE_BAYES_VALUE);

        // Training a maxent model by default!!!
        doccatModel = DocumentCategorizerME.train( languageCode: "en", sampleStream, params, new DoccatFactory());
    } catch (IOException e) {
        // Failed to read or parse training data, training failed
        e.printStackTrace();
    }
}
```

Figuur 32: OpenNLP sentiment training

Om een paar lijnen code te besparen gebeurt de training van de *categorizer* in de constructor van de Java klasse. Hierbij wordt het bestand waarin de training data zich bevindt opgehaald en ingelezen. Vooraleer de training gestart wordt, worden er een aantal parameters meegegeven. Eerst wordt de *cutoff* parameter gedeclareerd. Hierbij worden de aangemaakte n-grams ingekort waarbij woorden die minder waarde hebben en weinig voorkomen gefilterd worden. In het geval van sentiment is dit geen vereiste dus wordt deze op nul gezet. Daarna wordt het algoritme dat gebruikt gaat worden gespecifieerd, genaamd *Naive Bayes*. De parameters worden meegegeven aan de *DocumentCategorizerME* die het *categorizer* model teruggeeft. De *categorizer* is getraind en is klaar voor gebruik. Figuur 33 bevat de resterende code om een zin een sentiment toe te kennen.

```java
public void classifyNewString(String input) {
    DocumentCategorizerME myCategorizer = new DocumentCategorizerME(doccatModel);
    SimpleTokenizer tokenizer = SimpleTokenizer.INSTANCE;
    String[] tokenizedInput = tokenizer.tokenize(input);
    double[] outcomes = myCategorizer.categorize(tokenizedInput);
    String category = myCategorizer.getBestCategory(outcomes);

    System.out.println(input);
    if (checkOutcomesIfNeutral(outcomes)) {
        category = "1";
    }

    System.out.println(stringifyCategory(category));
}
```

Figuur 33: Sentiment analyse OpenNLP

Om het sentiment te kunnen berekenen dient er een *categorizer* aangemaakt te worden met het getraind model als parameter. Voor het classificeren verwacht de *categorizer* een string die in stukken is gesplitst. Hierbij wordt een *tokenizer* mee aangemaakt die de ingegeven zin splitst. Eens de gesplitste zin meegegeven is aan de aangemaakte *categorizer*, wordt er berekend waar de ingegeven zin toebehoord. De *outcomes* variabelen bevat een collectie van komma getallen die weergeeft hoe zeker OpenNLP is met zijn beslissing.

Als bijkomende stap wordt de teruggegeven data omgevormd tot tekst. Vanwege de training data, waarbij de positieve zinnen gelabeld zijn als '2' en de negatieve zinnen als '0', is de uitvoer van de *categorizer* ook '2' of '0'. Daarnaast herkent de *categorizer* geen neutrale zinnen. Daarom, zoals aangegeven in figuur 34 wordt er code voorzien die de teruggegeven data omzet in '*Negative*', '*Positive*' en '*Neutral*'.

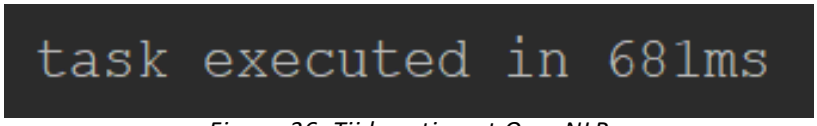
```
private boolean checkOutcomesIfNeutral(double[] outcomes) {
    if (outcomes[0] >= 0.75) {
        return false;
    } else if (outcomes[1] >= 0.75) {
        return false;
    } else {
        return true;
    }
}

private String stringifyCategory(String category) {
    switch (category) {
        case "0":
            return "Negative";
        case "1":
            return "Neutral";
        case "2":
            return "Positive";
        default:
            return "";
    }
}
```

Figuur 34: Resterende code voor output

De code vertrekt van de uitkomsten die berekend zijn door OpenNlp, en beslist of het framework zeker is met de beslissing. Als beslist wordt dat het framework niet zeker genoeg is, wordt de zin als neutraal beschouwd. De tweede methode neemt de beslissing en zet ze om in mensen taal. Om het resultaat te bekomen wordt er gebruik gemaakt van 100 willekeurige Engelse zinnen waarop de sentiment analyse wordt uitgevoerd. De uitkomsten van OpenNLP zijn teleurstellend maar niet onverwacht. Zoals getoond in figuur 35, zijn de beslissingen die de NLP maakt incorrect op een aantal na. De snelheid waarin het framework tot de resultaten bekomen is, is wel indrukwekkend. In maar liefst 600 milliseconden is OpenNLP erin geslaagd om alle 100 zinnen te overlopen en te classificeren. Zoals aangetoond in figuur 36.

```
I haven't got it.
Negative

This is such a sad story.
Positive

Get your filthy hands off her.
Neutral

My little sister doesn't like the taste of cottage cheese.
Neutral

Can you tell my age?
Neutral

What is this the abbreviation for?
Negative

I think it's very impressive.
Negative
```

Figuur 35: Fragment uitkomsten sentiment OpenNLP



Figuur 36: Tijd sentiment OpenNLP

4.2.1 Besluit

Het besluit is dat een sentiment analyse uitvoeren met OpenNLP jammer genoeg niet ingebouwd is in het framework. Hierdoor moet er naar andere manieren gezocht worden om dit te verwezenlijken. Aan de hand van de *Document Categorizer* van OpenNLP kan er een tussenweg voorzien worden om toch aan sentiment analyse te kunnen doen. Maar de *categorizer* zoekt naar patronen eerder dan naar de betekenis van de woorden binnen de zin. Maar het kan evengoed voor sentiment analyse gebruikt worden. Een voordeel dat patroonherkenning heeft tegenover het analyseren van individuele woorden is de snelheid, maar hiervoor wordt de accuraatheid voor ingeruild. OpenNLP *categorizer* is snel, maar moet voorzien worden van een grote dataset met training data om beter te worden.

4.3 CoreNLP

CoreNLP, gemaakt door Stanford University, is voorzien van een ingebouwde sentiment functie. Volgens de website van CoreNLP wordt het sentiment berekend door naar de gehele zinstructuur te kijken en te begrijpen wat er gezegd wordt. Het model is getraind aan de hand van een *Recursive Neural Network*. [13] De methode die gebruikt wordt door CoreNLP om sentiment te berekenen is verder uitgelegd in het onderzoek paper '*Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*' te vinden op nlp.stanford.edu. [14]

CoreNLP heeft twee versies van zijn framework. Zo is er een CoreNLP en CoreNLP Simple, voor wanneer er weinig aangepast moet worden aan de *pipeline.* Volgens de website is het voor mensen die CoreNLP zo snel en makkelijk mogelijk het framework functioneel willen hebben. Daarom voor de testen gaat er ook getest worden met CoreNLP Simple om de uitkomsten van de twee te vergelijken.

In CoreNLP wordt er gebruik gemaakt van een pipeline waarbij wordt meegegeven welke onderdelen gebruikt gaan worden van het framework. In figuur 37 is een voorbeeld van een pipeline waarbij wordt meegegeven welke functies nodig zijn. Een aantal functies hebben andere functies nodig om te kunnen werken. De volledige lijst met functies, *annotators* genoemd, is te vinden op de site van CoreNLP zelf. [15]

```
Properties props = new Properties();
props.setProperty("annotators", "tokenize, ssplit, parse, sentiment, pos, lemma, ner, regexner");
props.put("regexner.mapping", "jg-regexner.txt");
props.put("regexner.ignorecase","true");
StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
```

Figuur 37: CoreNLP pipeline declaratie

Hier worden de functies in String formaat meegegeven als *properties*. Daarna wordt deze aan de pipeline gegeven waardoor de pipeline klaar is voor gebruik. Om de pipeline op een zin of document toe te voegen, dient deze verwerkt te worden door de pipeline. Om dit te realiseren wordt code uitgevoerd te zien in figuur 38.

```
Annotation annotation = pipeline.process(string);
```

Figuur 38: ingevoerde tekst annoteren

De ingevoerde tekst wordt geannoteerd waardoor het sentiment, en andere functies, opgehaald kunnen worden. De code in figuur 39 biedt een kijk naar hoe het sentiment opgevraagd wordt. De berekende sentiment score is een cijfer van nul tot en met 4 en betekenen:

- 0: Zeer negatief
- 1: Negatief
- 2: Neutraal
- 3: Positief
- 4: Zeer positief

```
for (CoreMap sentence : annotation.get(CoreAnnotations.SentencesAnnotation.class)) {
    Tree tree = sentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);

    System.out.println(string);
    System.out.println(AssignValuesToEnum.assign(RNNCoreAnnotations.getPredictedClass(tree)));
}
```

Figuur 39: Sentiment analyse CoreNLP

Eerst wordt er gekeken naar de ingevoerde tekst of het meerdere zinnen zijn of enkel één zin. Als het meerdere zinnen zijn worden de zinnen gesplitst zodat er zin per zin het sentiment van gevonden kan worden. CoreNLP biedt geen mogelijkheid om teksten of documenten te analyseren, alleen maar een zin. Eens de zin verkregen is, wordt de sentiment klasse aangesproken om het sentiment op te vragen van de zin. Hier wordt berekend met welke zekerheid de NLP de score gaat toewijzen. Deze kunnen ook opgevraagd worden voor meer informatie. De score is een numerieke waarde die nog omgezet moet worden. Dit gebeurt aan de hand van een zelfgeschreven klasse te zien in figuur 40.

```
public class AssignValuesToEnum {
    public static Enum<SentimentScore> assign(int input) {
        switch (input) {
            case 0:
                return SentimentScore.VERY_NEGATIVE;
            case 1:
                return SentimentScore.NEGATIVE;
            case 2:
                return SentimentScore.NEUTRAL;
            case 3:
                return SentimentScore.POSITIVE;
            case 4:
                return SentimentScore.VERY_POSITIVE;
            default:
                return null;
        }
    }
}
```

Figuur 40: numerieke waarde naar mensen taal

CoreNLP zijn resultaten zijn accuraat maar traag. In afbeelding 41 is een overzicht van een aantal zinnen die bestempeld zijn waarna bij afbeelding 42 de tijd wordt weergegeven die CoreNLP nodig heeft om de 100 zinnen te labelen.

```
I haven't got it.
NEGATIVE

This is such a sad story.
NEGATIVE

Get your filthy hands off her.
NEGATIVE

My little sister doesn't like the taste of cottage cheese.
NEGATIVE

Can you tell my age?
NEUTRAL

What is this the abbreviation for?
NEUTRAL

I think it's very impressive.
POSITIVE
```

Figuur 41: CoreNLP resultaten sentiment analyse

Figuur 42: Tijd sentiment CoreNLP

Uit het fragment van figuur 41 is te zien dat de zinnen juist gelabeld zijn, maar dat het 5 seconden de tijd nodig heeft om de 100 zinnen te overlopen.

4.3.1 Besluit

De ingebouwde functie van CoreNLP, die op voorhand getraind is door Stanford University zelf, ziet er veelbelovend uit. De sentiment analyse van de framework is accuraat, maar niet 100 proces. Hier en daar kunnen er zinnen gevonden worden die fout gelabeld zijn. Qua snelheid is het niet het snelste, maar dit ligt aan de annotaties die mee gegeven worden. Bij elke stukje test dat meegegeven is aan het framework, dient deze het te annoteren met de functies die in de pipeline zijn meegegeven, en te splitsen. Hoe meer functies er meegegeven worden, hoe trager de applicatie de zinnen overloopt.

4.4 CoreNLP Simple

CoreNLP Simple is uiteraard de simpelere versie van CoreNLP. Hier wordt geen gebruik gemaakt van een pipeline die ingesteld moet worden. Achterliggend wordt deze nog uitgevoerd maar het is niet aan de gebruiker om functies toe te voegen of weg te nemen. Het gebruik ervan is veel simpeler en veel extra stappen moeten niet gezet te worden. Om het sentiment te berekenen zijn minder stappen vereist in vergelijking met CoreNLP en OpenNLP. In totaal zijn het vier lijnen code om alles te verkrijgen zoals te zien is in figuur 43. De uitvoer van de functie is het sentiment in geschreven vorm, in tegenstelling tot de numerieke vorm bij de volledige CoreNLP.

```
Document doc = new Document( text: "Teamwork went up and our team is perfoming well.");
for (Sentence sent : doc.sentences()) {
    System.out.println(sent.sentiment());
}
```

Figuur 43: Code CoreNLP Simple sentiment analyse

CoreNLP Simple zijn resultaten zijn accuraat maar de snelheid ervan neemt toe. In figuur 44 is een klein segment van de resultaten van CoreNLP Simple waarna in figuur 45 de tijd weergegeven wordt die het nodig had.

```
I haven't got it.
NEGATIVE

This is such a sad story.
NEGATIVE

Get your filthy hands off her.
NEGATIVE

My little sister doesn't like the taste of cottage cheese.
NEGATIVE

Can you tell my age?
NEUTRAL

What is this the abbreviation for?
NEUTRAL

I think it's very impressive.
POSITIVE
```

Figuur 44: CoreNLP Simple resultaten sentiment analyse

```
task executed in 7115ms
```

Figuur 45: Tijd sentiment CoreNLP Simple

4.4.1 Besluit

CoreNLP Simple is simpeler in gebruik dan de standaardversie. Er is geen configuratie vereist en is perfect voor mensen die pas met NLP's beginnen te werken. Het declareren van een document en de zinnen ophalen is simpel. De *sentence* klasse van CoreNLP Simple bevat alle basismethodes die nodig zijn en die de NLP te bieden heeft. Wat wel een verassing is, is de tijd die het nodig heeft om 100 zinnen te overlopen en te labelen. Dit kan te wijten zijn aan CoreNLP die alle functies gaat nemen zonder de gebruiker een keuze te geven. De accuraatheid van CoreNLP Simple is parallel met de accuraatheid van CoreNLP, de complete versie. Dit is geen verassing omdat beide dezelfde manier gebruiken met dezelfde modellen. In tabel 1 is een overzicht met de uitslagen van alle vergeleken NLP's.

4.5 Resultaten sentiment analyse

Tabel 1: Resultaten sentiment analyse

	JUIST	TIJD
OPENNLP	33/100	0.7s
CORENLP	81/100	5.3s
CORENLP SIMPLE	81/100	7.1s

Afgerond op één cijfer na de komma.

5. Inhoud analyse

Uit dit onderzoek is gebleken dat context halen uit zinnen een grote uitdaging is, tot zelfs onmogelijk. Een methode om toch erachter te komen waar het over gaat is de *Named Entity Recognition*. Bij de NER-methode zoekt de NLP naar zelfstandige naamwoorden en probeert hij te herkennen waarmee het verbonden is. Zo kan gisteren gezien worden als een tijdsbepaling of Alexander kan herkend worden als een naam van een persoon. Hierdoor kan herkend worden waar het overgaat en de algemene thema gevonden worden. In dit gedeelte wordt er gekeken hoe de NER uitgevoerd kan worden bij OpenNLP, CoreNLP en CoreNLP Simple.

5.1 OpenNLP

OpenNLP biedt de mogelijkheid om NER toe te passen. Maar de methode waarop is niet performant. Zoals in figuur 46 te zien is, zijn alle *models* van de NER gesplitst in zeven bestanden. De modellen kunnen niet samen gebruikt worden, tenzij er via de code zeven aparte *models* aangemaakt worden en de tekst over elk model overlopen wordt. Dit is ook de manier die gebruikt wordt om te achterhalen welke entiteiten er in een zin te vinden zijn.

en	Name Finder	Date name finder model.	en-ner-date.bin
en	Name Finder	Location name finder model.	en-ner-location.bin
en	Name Finder	Money name finder model.	en-ner-money.bin
en	Name Finder	Organization name finder model.	en-ner-organization.bin
en	Name Finder	Percentage name finder model.	en-ner-percentage.bin
en	Name Finder	Person name finder model.	en-ner-person.bin
en	Name Finder	Time name finder model.	en-ner-time.bin

Figuur 46: OpenNLP NER Model bestanden

Bij de uitvoering van de NER worden alle zeven modellen binnengehaald door de code en in modellen gestoken. Dat is jammer genoeg de enige manier tenzij er zelf een NER-model getraind wordt of manueel samengevoegd wordt. Dit kan een beetje omslachtig worden maar naast de benoemde opties zijn er geen andere manieren dat dit kan gebeuren. Zoals te zien is in figuur 47 worden de zeven verschillende modellen individueel ingeladen waarna de modellen klaar zijn om de tekst te analyseren.

```
InputStream inputStream = getClass().getResourceAsStream( name: "/en-ner-person.bin");
personModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-ner-date.bin");
dateModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-ner-location.bin");
locationModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-ner-money.bin");
moneyModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-ner-organization.bin");
organizationModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-ner-percentage.bin");
percentageModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-ner-time.bin");
timeModel = new TokenNameFinderModel(inputStream);

inputStream = getClass().getResourceAsStream( name: "/en-token.bin");
tokenizerModel = new TokenizerModel(inputStream);
```

Figuur 47: Werkwijze NER OpenNLP

Het zoeken van de NER gebeurt vervolgens door een zelfgeschreven methode waarbij de *tokenized* zin aan wordt meegegeven aan het model en die de gevonden entiteiten teruggeeft. Een belangrijk onderdeel is dat er zelf gezorgd moet worden voor deze *tokenization*, wat weer een extra stap is binnen OpenNLP. Dit gebeurt voor elk model, en in totaal wordt de zin zeven keer doorzocht op entiteiten die dan worden opgeslagen in een Java Map. Het uiteindelijke resultaat is in het volgend gedeelte te vinden.

De resultaten van de NER-onderzoek zijn belovend. Op het eerste zicht naar de uitvoer van de IDE valt op dat er vooral namen gevonden worden en niet meer. Uit verder onderzoek blijkt dat er ook tijden gevonden worden en landen maar niet meer dan dat. Dat is voldoende informatie om te achterhalen waar het overgaat of over wie het gaat. In figuur 48 is een voorbeeld van de uitvoer van OpenNLP. In figuur 49 is te zien dat ondanks het framework een aantal zware taken met herhalen per zin, de snelheid waarop het gebeurt er weinig onder lijdt. Alle 100 zinnen worde geanalyseerd in maar liefst zeven seconden.

```
Are you in Australia?
Australia: [3..4) location

Tom spends a lot of time in the library.
Tom: [0..1) person

We should give Tom a chance to finish the report.
Tom: [3..4) person

I'm going to the beach this afternoon.
this: [6..8) time
```

Figuur 48: Resultaten NER OpenNLP

```
task executed in 7060ms
```

Figuur 49: Performantie NER OpenNLP

5.2 CoreNLP

De modellen van CoreNLP zijn ingebouwd en klaar voor gebruik. Het is relatief makkelijk om de NER toe te passen op een zin en de gevonden entiteiten weer te geven. In figuur 50 wordt de methode weergegeven hoe de NER-resultaten ontleed kunnen worden. Eerst moet de zin omgevormd worden naar een *CoreDocument* om functies toe te kunnen passen. In figuur 51 wordt weergegeven hoe de CoreDocument bekomen kan worden. Eens de entiteiten gevonden zijn worden deze uitgeprint in de console van de IDE.

```
CoreDocument coreDoc = new CoreDocument( documentText: "tekst of zin");
pipeline.annotate(coreDoc);
```

Figuur 50: CoreNLP CoreDocument aanmaken

```
public static void getEntities(CoreDocument coreDoc) {
    System.out.println("---");
    System.out.println("entities found");
    for (CoreEntityMention em : coreDoc.entityMentions())
        System.out.println("\tdetected entity: \t"+em.text()+"\t"+em.entityType());
    System.out.println("---");
}
```

Figuur 51: CoreNLP NER

Om de frameworks met elkaar te kunnen vergelijken wordt de dataset van 100 willekeurige Engelse zinnen gebruikt om te kijken welke entiteiten gevonden kunnen worden. Al snel is duidelijk dat er meer NER-labels zijn en ook verschillende varianten, zoals te zien is in figuur 53. De snelheid niet veel hoger dan OpenNLP zijn snelheden van rond de zeven seconden. In figuur 52 wordt weergegeven de tijd die CoreNLP nodig heeft om de functie uit te voeren. Het is in CoreNLP mogelijk om specifiekere NER-entiteiten op te vragen. Hiervoor is er een methode voorzien om dit aan te zetten. Volgens de website van CoreNLP zal dit wel de snelheid van functie doen vertragen omdat er meer modellen ingeladen moeten worden. [16]

```
This book is missing two pages.
two: NUMBER

Have you been to the barber?
barber: TITLE

"We know that a cat

Are things bad now?
now: DATE

Her anger is understandable.
Her: PERSON

I've learned a lot about him.
him: PERSON
```

```
task executed in 9672ms
```

Figuur 52: Performantie NER CoreNLP

Figuur 53: Resultaten NER CoreNLP

5.3 CoreNLP Simple

Bij CoreNLP Simple, die zich promoot als sneller en makkelijk dan CoreNLP, is het ook mogelijk om entiteiten te vinden aan de hand van de NER. Als er naar de code gekeken wordt kan er duidlelijk besloten worden dat CoreNLP Simple duidelijk eenvoudiger is om op te stellen en uit te voeren. Net zoals bij de sentiment analyse is het maar één regel code die ervoor zorgt dat alle entiteiten van een zin ontleed worden en in een lijst worden gegooid. Er is wel één probleem die zich voordoet en dat is dat CoreNLP Simple ook de niet-entiteiten labelt die later gefilterd moeten worden. Hierdoor zal de code die nodig is om de entiteiten te vinden, een aantal lijnen langer worden.

Het uitvoeren van de NER binnen CoreNLP Simple gebeurt, net zoals bij de sentiment analyse, door bij de *Sentence* variabele een voorgemaakte functie uit te voeren. Deze functie geeft een lijst terug met de NER-objecten. Om de niet-entiteiten eruit te halen wordt er wat logica toegepast om de labels eruit te filteren. Niet-entiteiten krijgen de label “O” waarop er gezocht kan worden. Zoals in figuur 54 te zien is, is het totaalaantal code om tot een resultaat te bekomen beperkt.

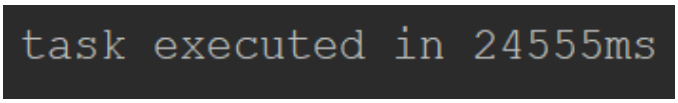
```
for (String text : data) {
    Document doc = new Document(text);
    for (Sentence sent : doc.sentences()) {
        System.out.println(sent.toString());

        for (int i = 0; i < sent.nerTags().size(); i++) {
            if (!sent.nerTag(i).equals("O")) {
                System.out.println(sent.word(i) + ": " + sent.nerTag(i));
            }
        }
        System.out.println();
    }
}
```

Figuur 54: CoreNLP filter niet-entiteiten

Wat opvalt bij het uitvoeren van de functie is de tijd die de functie nodig heeft om alle 100 zinnen te analyseren zeer hoog. Maar liefst 24 seconden zijn nodig. Dit is een resultaat die onverwachts komt sinds CoreNLP Simple gepromoot wordt als simpeler en efficiënter. De resultaten verschillen een klein beetje met die van CoreNLP, maar dit komt door de manier waarop gefilterd wordt door de zelfgeschreven logica. In het algemeen zijn de resultaten, zoals verwacht, gelijk aan de resultaten van CoreNLP aangezien beide gebruik maken van dezelfde modellen en executie-methode. In figuur 55 is een voorbeeld van de uitvoer van CoreNLP Simple en de zelfgeschreven logica. De tijd die de functie nodig heeft wordt weergegeven in figuur 56.

```
The travel agent suggested that we take some traveler's checks with us.
travel: TITLE
agent: TITLE

I still make mistakes.

The students pay keen attention.

We arrived three days ago.
three: DATE
days: DATE
ago: DATE

This a very significant discovery.

Don't tell me you've forgotten Tom.
Tom: PERSON
```

Figuur 55: Resultaten NER CoreNLP Simple



Figuur 56: Performantie NER CoreNLP Simple

5.4 Algemeen besluit

Het vinden van entiteiten is een uitdaging, zeker voor een computer. Aan de hand van voorgemaakte modellen kan er gezocht worden naar verschillende entiteiten binnen een tekst. Op die manier kan een algemeen beeld gevormd worden waar het overgaat. De twee frameworks die getest zijn, met de additie van CoreNLP Simple wat een onderdeel is van het framework CoreNLP. OpenNLP doet het wat minder goed waar er meer verwacht wordt dat er zelf gezorgd wordt voor de modellen. Op vlak van code is er ook een duidelijk verschil met CoreNLP. OpenNLP is dan ook het framework die ondermaats presteert maar wel het meest *customizable* is. Zelf modellen maken en trainen is geen onderdeel van dit onderzoek. In tabel 2 is een overzicht met de uitslagen van alle vergeleken NLP's.

5.5 Resultaten Inhoudsanalyse

Tabel 2: Resultaten inhoudsanalyse

	TIJD
OPENNLP	7s
CORENLP	9.7s
CORENLP SIMPLE	24.6s

Afgerond op één cijfer na de komma.

Conclusie gebruikte frameworks

In het onderzoek werd er gebruik gemaakt van de frameworks CoreNLP en OpenNLP, met als toevoeging CoreNLP Simple. CoreNLP is een *natural language processor*-framework ontwikkeld door Stanford University. OpenNLP is een framework die opgenomen is door Apache via het Apache Incubator programma. Beide frameworks focussen zich op het verwerken van taal en hebben een Java *library* waar vrij gebruik van gemaakt kan worden.

Tijdens het ontwikkelingsproces van dit onderzoek is er ondervonden dat het opzetten van CoreNLP moeilijker is dan het opzetten van OpenNLP, vanwege de pipeline die gebruikt wordt. Er zijn meerdere variabelen die dezelfde betekenis hebben maar toch niet met elkaar samenwerken. Een voorbeeld hiervan zijn de Java klassen CoreDocument en Document van CoreNLP. Een handige *feature* is CoreNLP Simple binnen CoreNLP, waarbij er makkelijker en met minder code een resultaat bereikt kan worden. Maar hiermee wordt de bruikbaarheid gelimiteerd. De uitgebreide gegevens van CoreNLP hebben meer stappen of complexere stappen nodig om dezelfde resultaten te bereiken als CoreNLP Simple. Maar de resultaten zijn flexibeler in gebruik. OpenNLP is een framework die niet van in het begin hoog getraind is, waardoor het kan leiden tot slechte resultaten. OpenNLP biedt uitgebreide documentatie om elk deel van de framework zelf te trainen naar wens waardoor dit aantrekkelijker kan zijn voor bedrijven die de resources hebben. Beide frameworks zijn krachtig maar voor verschillende doelpublieken bedoeld.

Conclusie onderzoek

Voor de eerste onderzoeksvraag werd er onderzocht of het mogelijk was om zinnen te groeperen op basis van inhoud. De gebruikte frameworks zijn voorzien van een NER-functie waardoor entiteiten uit een zin gehaald kunnen worden. Op basis van de gevonden entiteiten kan er aan groeperen gedaan worden. Voor de tweede onderzoeksvraag werd onderzocht of het mogelijk was om het gevoel die achter de zinnen schuilt te analyseren. Met andere woorden, er wordt gezocht naar het sentiment van een zin. Na onderzoek blijkt dat het zoeken van het sentiment mogelijk is met de ingebouwde functie van CoreNLP die met een accuraatheid van 89% volgens de gebruikte testdata. Simpele zinnen kunnen makkelijk en accuraat geanalyseerd worden terwijl het met complexere zinnen wat moeilijker wordt en er meer fouten gemaakt worden. De laatste onderzoeksvraag die onderzocht werd is of het framework CoreNLP sneller en accurater werkt dan het framework OpenNLP. Uit resultaten van de sentiment analyse en resultaten van de NER-analyse kan worden vastgesteld dat de CoreNLP accurater te werk gaat maar hiervoor snelheid opoffert. OpenNLP gaat snel tewerk maar de resultaten kloppen niet altijd. De mogelijkheid om in CoreNLP en OpenNLP eigen modellen te trainen is een pluspunt voor beide.

III. Conclusie stage

Tijdens de stage heb ik met behulp van nieuwe technologieën, een project mogen opbouwen. In het begin was het een uitdaging omdat ik de syntax of de werking ervan niet gewoon was, of totaal niet kende. Maar uiteindelijk kreeg ik de nieuwe frameworks onder de knie, waardoor het ontwikkelen van dit project kon starten. Het ontwikkelen van het project ging vlot vooruit, en tijdens het ontwikkelen leerde ik nieuwe dingen bij. Van best practices tot nieuwe manieren om resultaten efficiënter te bekomen. Tijdens de ontwikkelingsfase van het project waren er een aantal merkwaardige problemen die het project in gevaar brachten. Uiteindelijk was het de bedoeling om een platform te bouwen waar tickets automatisch gegroepeerd konden worden. Maar dit was niet mogelijk zonder extra libraries of frameworks. Om kennis op te doen over machine learning en advies te verkrijgen, heb ik gesproken met InfoFarm binnen de Xplore Group. InfoFarm specialiseert zich in datamining en machine learning. Zij adviseerden eens te kijken naar NLP's. Met behulp van deze informatie is er een aanzet gedaan tot het onderzoeken van CoreNLP en OpenNLP. Reeds getrainde machine learning-frameworks. Verder was de zoektocht naar een Java library die accuraat twee zinnen kon vergelijken aan de hand van de LSA-methode, een uitdaging op zichzelf. Er waren ook problemen die zich voordeden waarvan de oplossing meerdere weken op zich liet wachten. De CI/CD pipeline die via Google Cloud opgezet was, kreeg de libraries en de nodige bestanden niet opgehaald waardoor het build proces meermaals faalde. Hiervoor zijn er een groot aantal creatieve oplossingen toegepast waaronder de twee externe libraries in het project te plaatsen, in plaats van op te halen via Gradle. De nodige modellen die de twee libraries nodig hadden, in een zip bestand te steken en programmatisch uit te pakken tijdens het opstarten van het project op de plaats waar ze gevonden konden worden. Het laatste probleem die zich voordeed was een library die geschreven was in 2013, bestanden nodig had, maar dat geschreven klassen hiervoor incorrect geschreven waren. Om dit op te lossen moest het JAR bestand gedecompiled te worden, de Java klassen aangepast te worden en deze terug compilen. Dan de Java klassen terug in het JAR bestand gestoken te worden. Uiteindelijk resulteerde dit in een slagende pipeline en stond de lokale versie gesynchroniseerd met de live versie. Uiteindelijk kan ik vaststellen dat ik veel heb bijgeleerd tijdens deze stage. Nieuwe technologieën, CI/CD pipelines en een betere kennis rond JAR's en FAT-JAR's. Naast alle technische ervaring die ik heb mogen opbouwen, heb ik mij verder mogen verdiepen in de agile SCRUM-methodologie. Het eindresultaat is een platform waar retrospectives gehouden kunnen worden. Een applicatie die doet wat hij moet doen. Zelf ben ik tevreden over het resultaat. Ook XTi zelf is tevreden met het eindproduct en prestaties die geleverd zijn.

6. Bibliografie

[1] Corda Campus, „Floor plan,” Corda Campus, 21 5 2015. [Online]. Available: https://www.cordacampus.com/en/floor-plan. [Geopend 3 4 2019].

[2] S. O. Bijvank, „Watervalmodel / V-model,” House of Control, 29 10 2015. [Online]. Available: http://www.house-of-control.nl/watervalmodel-v-model.html. [Geopend 3 4 2019].

[3] P. Carbonnelle, „PYPL PopularitY of Programming Language,” /, 18 1 2015. [Online]. Available: http://pypl.github.io/PYPL.html. [Geopend 24 4 2019].

[4] TeamRetro, „Mad Sad Glad Retrospective,” TeamRetro, 22 8 2018. [Online]. Available: https://www.teamretro.com/techniques/mad-sad-glad-retrospective/. [Geopend 24 4 2019].

[5] TeamRetro, „What is a four Ls retrospective?,” TeamRetro - GroupMap Technology Pty. Ltd., 22 8 2018. [Online]. Available: https://www.teamretro.com/techniques/4ls-retrospective/. [Geopend 24 4 2019].

[6] Vue.js, „Introduction,” Vue.js, 8 11 2016. [Online]. Available: https://vuejs.org/v2/guide/#Getting-Started. [Geopend 25 2 2019].

[7] Spring, „Spring Initializr,” Spring, 13 1 2014. [Online]. Available: https://start.spring.io/. [Geopend 25 2 2019].

[8] V. Srinivasan, „Context, Language, and Reasoning in AI: Three Key Challenges,” MIT Technology Review, 1 1 1. [Online]. Available: 1. [Geopend 1 1 1].

[9] S. Zillis, „How conversation (with context),” IBM, 1 1 1. [Online]. Available: 1. [Geopend 1 1 1].

[10] lexalytics, „Context Analysis in NLP: Why It’s Valuable and How It’s Done,” Lexalytics, 18 2 2019. [Online]. Available: https://www.lexalytics.com/lexablog/context-analysis-nlp. [Geopend 27 3 2019].

[11] Apache Jira, „Sentiment Analysis,” Apache Jira, 13 3 2016. [Online]. Available: https://issues.apache.org/jira/browse/OPENNLP-840. [Geopend 30 4 2019].

[12] L. Giura, „SENTIMENT ANALYSIS USING OPENNLP DOCUMENT CATEGORIZER,” Technobium, 8 3 2015. [Online]. Available: http://technobium.com/sentiment-analysis-using-opennlp-document-categorizer/. [Geopend 30 4 2019].

[13] Stanford University, „Sentiment Analysis,” Stanford University, 1 8 2013. [Online]. Available: https://nlp.stanford.edu/sentiment/index.html. [Geopend 2 5 2019].

[14] Stanford University, „Recursive Deep Models for Semantic Compositionality,” Stanford University, 8 10 2013. [Online]. Available: https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf. [Geopend 13 5 2019].

[15] Stanford University, „CoreNLP,” Stanford University, 4 12 2014. [Online]. Available: https://stanfordnlp.github.io/CoreNLP/. [Geopend 24 4 2019].

[16] CoreNLP, „Named Entity Recognition,” Stanford University, 18 12 2015. [Online]. Available: https://stanfordnlp.github.io/CoreNLP/ner.html#fine-grained-ner. [Geopend 15 5 2019].

[17] IBM, „How conversation (with context),” IBM, 8 10 2017. [Online]. Available: https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-conversation.html. [Geopend 27 3 2019].

[18] OpenNLP, „OpenNLP Proposal,” Apache, 09 11 2010. [Online]. Available: https://wiki.apache.org/incubator/OpenNLPProposal. [Geopend 03 05 2019].

[19] appbrain, „Kotlin,” Appbrain, 23 4 2019. [Online]. Available: https://www.appbrain.com/stats/libraries/details/kotlin/kotlin. [Geopend 24 4 2019].