adres
Colosseum 13
7521 PV Enschede

www.ovsoftware.nl
info@ovsoftware.nl

telefoon
+31 (0)53 3033600

kvk
06049464

Afstudeerverslag

Afstudeerstage

Leo Geerdink

bewerkt door
Leo Geerdink

leo.geerdink@ovsoftware.com
461379@student.saxion.nl

documentversie
0.1

Colofon

Student
Naam: Leo Geerdink
Studentnummer: 461379
E-mail: 461379@student.saxion.nl

Stagebegeleider
Naam: Melanie Bonnes
E-mail: m.j.g.bonnes@saxion.nl

Bedrijfsbegeleider
Naam: Eddie Zeën
E-mail: eddie.zeen@ovsoftware.com
Telefoon: 06-14944134

Stagebedrijf
Naam: OVSoftware B.V.
Telefoon: +31 (0)53 3033600
Bezoekadres: Colosseum 13 Enschede
Website: https://www.ovsoftware.nl/

OVSoftware

Inhoud

1 Colofon 2

2 Definities 6

3 Samenvatting 8

3.1 Probleemstelling ..... **Error! Bookmark not defined.**

3.2 Onderzoek ..... **Error! Bookmark not defined.**

3.3 Realisatie ..... **Error! Bookmark not defined.**

3.4 Resultaat ..... **Error! Bookmark not defined.**

3.4.1 Onderzoek ..... **Error! Bookmark not defined.**

4 Inleiding 10

5 Opdracht 11

6 Bedrijf 11

7 Aanpak 14

7.1 Onderzoek 14

7.1.1 Deelvraag 1: Wat doet OVSoftware al op het gebied van codekwaliteit? 14

7.1.2 Deelvraag 2: Op welke manier kan code beoordeeld worden 14

7.1.3 Deelvraag 3: Hoe is machine learning relevant voor dit project? 14

7.1.4 Deelvraag 4: Welke bestaande datasets kunnen gebruikt worden? 14

7.1.5 Deelvraag 5: Hoe kan de dataset opgeslagen worden? 14

7.1.6 Deelvraag 6: Op welke manier kunnen de resultaten opgeslagen worden? 15

7.1.7 Deelvraag 7: Welke machine learning technieken kunnen gebruikt worden voor het maken van het model? 15

7.1.8 Deelvraag 8: Hoe kan een systeem gebouwd worden dat de resultaten van het model nuttig kan weergeven? 15

7.1.9 Deelvraag 9: Hoe kan de dataset verder uitgebreid worden? 15

7.1.10 Deelvraag 10: Hoe kan het model op een nuttige manier uitgevoerd worden? 16

7.2 Ontwerp 16

7.3 Realisatie 16

8 Resultaten 18
8.1 Onderzoek 18
8.2 Ontwerpen 24
8.2.1 Eisen 24
8.2.2 Front-End Design 27
8.2.3 Data Model 29
8.3 Maken van het model 29
8.3.1 Klaarmaken Dataset 29
8.3.2 Trainen van het model 34
8.4 Ontwikkelen van het dashboard 40
8.5 Implementeren van het model 43
8.6 Ontwikkelen van de pipeline 44

9 Evaluatie 45
9.1 Resultaten 45
9.2 Aanbevelingen 45
9.3 Evaluatie van het proces 45

10 Bronnen 47

11 Bijlagen 49
11.1 Bijlage 1: Download script 49
11.2 Bijlage 2: Krimp script 50
11.3 Bijlage 3: Syntax tree script 50
11.4 Bijlage 4: Gebruiksklaar maken script 52
11.5 Bijlage 5: Traditioneel model 53
11.6 Bijlage 6: Neural Network 54
11.7 Bijlage 7: TfidfVectorizer 54
11.8 Bijlage 8: Trainen en exporteren van het model voor het detecteren van long method 55
11.9 Bijlage 9: Data model 56
11.9.1 Code Segment 56
11.9.2 Issue 57
11.9.3 Rating 58

11.10 Bijlage 10: Voorbeeld van een API Controller 59
11.11 Bijlage 11: Lijst van issues 59
11.11.1 HTML 59
11.11.2 TypeScript 60
11.12 Bijlage 12: Code Display Component 61
11.12.1 HTML 61
11.12.2 TypeScript 61
11.13 Bijlage 13: Lijst van beoordelingen 62
11.13.1 HTML 62
11.13.2 TypeScript 62
11.14 Bijlage 14: Pipeline script 63
11.15 Hyperparameter tuning 64

OVSoftware

Definities

Term	Definitie
Framework	Een verzameling tools en voorgeschreven code om het maken van software te vereenvoudigen.
Front-end	Het visuele deel van een applicatie.
Back-end	Het niet-visuele deel van de applicatie. Deze communiceert met de database en zorgt dat de front-end alle nodige data heeft die laten zien moet worden.
Angular	Een framework voor het maken van de front-end van web applicaties.
ASP.NET	Een framework voor het maken van de back-end van web applicaties.
ANTLR	Een framework voor het maken en analyseren van programmeertalen.
MLCQ	Een dataset met informatie over de kwaliteit van verschillende Java projecten.
Sample	De term die gebruikt wordt voor het beschrijven van één element uit een dataset
Label	Een waarde in de dataset die aangeeft wat een sample representeert
Algoritme (machine learning)	Een functie die uitgevoerd wordt op een collectie data, om zo een model te kunnen maken.
Model	Een model is het product van een algoritme, die uitgevoerd kan worden op nieuwe data om hier de label bij te vinden.
Syntax tree	Een boom datastructuur om de syntax van computercode te weergeven.
Smell	Een aspect aan code die aangeeft dat er mogelijk een probleem is
Anti-pattern	Een architectonisch patroon dat tot problemen lijdt wanneer deze geïmplementeerd wordt.

Versiebeheer systeem	Een systeem dat gebruikt wordt door ontwikkelaars om de code die ze geschreven hebben op te slaan, eerdere veranderingen terug te kunnen kijken en het samenwerken met anderen te versoepelen.
Pushen	Code naar het online versiebeheer systeem toevoegen.
CI/CD pipeline	Een Continuous Integration / Continuous Deployment pipeline is een systeem waarmee code uitgevoerd kan worden bij elke push. Dit kan bijvoorbeeld gebruikt voor het automatisch uitvoeren van tests.
Statische Analyse	Analyse van code die uitgevoerd kan worden zonder de code uit te voeren, om objectieve metingen te doen zoals de hoeveelheid regels.

OVSoftware

Samenvatting

Onderhoudbare code schrijven betekent dat in de toekomst deze code makkelijk aangepast kan worden. Wanneer de code aangepast moet worden wordt zo de nodige moeite geminimaliseerd. Het is echter niet altijd even makkelijk om te beoordelen of code wel of niet onderhoudbaar is. OVSoftware wil een tool ontwikkelen die code kan analyseren met ondersteuning van een machine learning model. Hiermee moet de programmeur inzicht krijgen waar in de code problemen zijn. Er bestaan al systemen die code kunnen analyseren gebaseerd op regels, maar deze niet alle problemen kunnen in meetbare regels

Er moet onderzoek gevoerd worden naar bestaande datasets die gebruikt kunnen worden voor het trainen van het model. Voor het inzichtelijk maken van de resultaten wordt een dashboard gebruikt. Hiervoor wordt onderzoek gedaan naar de verschillende frameworks die gebruikt kunnen worden voor het ontwikkelen van deze applicatie.

Ook is er onderzoek gedaan naar het verbeteren van de prestaties van het model door het uitbreiden van de dataset. Elke keer dat het model uitgevoerd wordt, kan het resultaat hiervan beoordeeld worden. Als dit een goed resultaat is, kan deze bij de dataset toegevoegd worden.

Om het model uit te voeren is er gekozen om gebruik te maken van een CI/CD pipeline. Dit wil zeggen dat het model uitgevoerd moet kunnen worden op basis van toevoegingen aan het versiebeheersysteem. Op deze manier wordt het model regelmatig uitgevoerd zonder teveel computerkracht van de computer van de programmeur te eisen.

Ook is er onderzoek gedaan naar de mogelijkheid om een terugkoppeling te implementeren tussen het model en de dataset. Door een gebruiker een resultaat van het model te beoordelen op accuraatheid kan beoordeeld worden of een resultaat toegevoegd kan worden aan de dataset. Als het model opnieuw getraind wordt is er meer data beschikbaar. Dit zou de prestaties moeten verbeteren.

Gebaseerd hierop zijn de volgende deelvragen geformuleerd:

- Wat doet OVSoftware al op het gebied van codekwaliteit van codekwaliteit?
- Op welke manier kan code beoordeeld worden
- Hoe is machine learning relevant voor dit project?
- Welke bestaande datasets kunnen gebruikt worden?
- Hoe kan de dataset opgeslagen worden?
- Op welke manier kunnen de resultaten opgeslagen worden?
- Welke machine learning technieken kunnen gebruikt worden voor het maken van het model?
- Hoe kan een systeem gebouwd worden dat de resultaten van het model nuttig kan weergeven?

- Hoe kan de dataset verder uitgebreid worden?
- Hoe kan het model op een nuttige manier uitgevoerd worden?

Met als hoofdvraag:

- Hoe kan een systeem gebouwd worden om de kwaliteit van code te analyseren met machine learning?

Het project bestaat uit 3 onderdelen die gerealiseerd moeten worden:

- Het model
- Het dashboard
- De pipeline

Deze 3 onderdelen moeten samen kunnen werken om zo een geheel systeem te vormen: De pipeline stuurt code naar de back-end waar het geanalyseerd wordt. De resultaten van de analyse worden in het dashboard weergeven.

Tijdens het ontwikkelen van het model wordt er elke week 2 meetings gehouden met de begeleiders om de voortgang en eventuele problemen te bespreken. Voor het ontwikkelen van het dashboard en de pipeline wordt er gewerkt met dagelijkse sprint meetings en voortgangsgesprekken als er iets besproken moet worden dat niet in de sprint meetings kan. Voor werkmethode wordt scrum gehanteerd. Gezien er maar één persoon is die aan dit project werkt, worden er aanpassingen gemaakt aan de methode als dit handiger lijkt.

Uit onderzoek is gebleken dat de meest geschikte dataset de MLCQ dataset [1] is. Deze bestaat uit 4 criteria waarop code beoordeeld wordt, met 4 niveaus aan serieusheid voor elk criterium. De code in de dataset moet geconverteerd worden naar een syntax tree, wat uit onderzoek de meest nuttige manier is om code te representeren voor machine learning. [2]. Uit dit onderzoek zijn ook een aantal algoritmen bepaald die de meeste potentie hebben voor het maken van het model.

De code en syntax trees worden opgeslagen als losse bestanden met metadata in een spreadsheet. De resultaten van het model worden opgeslagen in een SQL database.

Voor het maken van het dashboard worden de frameworks Angular en ASP.NET gebruikt, respectievelijk voor de front-end en back-end. Voor de database wordt SQL Server gebruikt, met Entity Framework voor de interactie met de back-end.

Voor de pipeline wordt Azure Pipelines gebruikt.

Na de dataset te verwerken is er een model getraind. Dit model is geëxporteerd voor gebruik in de back-end, waar het aangeroepen kan worden via een API endpoint. De pipeline stuurt elk bestand naar dit endpoint. De resultaten worden opgeslagen in een SQL Server database.
Het dashboard vraagt aan de back-end om deze resultaten te sturen, zodat deze laten zien kunnen worden aan de gebruiker.
In het dashboard kan de gebruiker een resultaat beoordelen op accuraatheid. Wanneer een resultaat voldoende positieve beoordeling heeft, wordt deze niet meer in het dashboard laten zien en opgeslagen in de dataset. Als een resultaat teveel negatieve beoordelingen heeft, wordt deze niet meer laten zien in het dashboard en niet toegevoegd aan de dataset.

1 Inleiding

Dit document is het afstudeerverslag van Leo Geerdink, voor de afstudeerstage, uitgevoerd bij OVSoftware B.V. Het doel van dit document is om een overzicht te geven van het bedrijf waar de stage bij gelopen is, de uitgevoerde opdracht, de realisatie hiervan en het uiteindelijk resultaat.

De afstudeerstage vond plaats in de periode februari – juli, 2022.
De opdracht is geformuleerd in overleg met de afstudeerbegeleider Eddie Zeën, wegens de waarde die OVSoftware hecht aan de kwaliteit van geproduceerde code.

OVSoftware wil graag meer inzicht krijgen in de kwaliteit van de code die geschreven wordt, en hoe dit ontwikkelt tijdens een project. Hiervoor moet er een tool ontwikkelt worden die code kan analyseren en de resultaten inzichtelijk maakt voor de programmeur.

2 Bedrijf

OVSoftware B.V. is een bedrijf die maatwerk software produceert en onderhoudt voor verschillende bedrijven en ook medewerkers detacheert naar deze bedrijven. Met ervaring in Java, .NET, AI en Machine Learning, Low-code, Blockchain, en Virtual/Augmented reality technologiën.

OVSoftware hecht veel waarde aan kwaliteit, kennis, openheid en betrokkenheid. Hiermee hebben ze vertrouwen opgebouwd bij verschillende klanten. Om dit vertrouwen om te bewaren is het belangrijk dat ze de kwaliteit en onderhoudbaarheid van code waarborgen. Medewerkers en gebruikte tools moeten voldoen aan actuele certificeringen. Ook wordt het delen van kennis onder collega's gestimuleerd, bijvoorbeeld in de vorm van presentaties en het gezamenlijk leren van een nieuwe techniek.

Voor het vinden van problemen in code worden onder meer verschillende tools gebruikt, zoals Better Code Hub [3]. Better Code Hub is echter gelimiteerd tot analyse gebaseerd op regels. Om mogelijke oplossingen voor deze limitatie te onderzoeken is deze opdracht bedacht.

OVSoftware

3 Opdracht

OVSoftware wil de mogelijkheid onderzoeken om een tool te ontwikkelen die problemen die de onderhoudbaarheid van code verminderen te vinden. Hiervoor moet een systeem gemaakt worden die als prototype kan dienen.
Dit systeem bestaat minimaal uit de volgende onderdelen:

- Een systeem dat problemen in de code detecteert.
- Een manier om dit uit te voeren.
- Een interface die de resultaten inzichtelijk maakt voor de programmeur.

De onderhoudbaarheid van code analyseren kan ook uitgevoerd worden door een mens. Dit heeft echter het nadeel dat het tijd kost voor een mens om dit te doen. Een automatisch systeem is in staat om gehele codebases in seconden te analyseren.

Voor de ontwikkeling voor de tool is er vanuit OVSoftware de wens om machine learning te gebruiken. Machine learning is een techniek die moeilijk te definiëren algoritmen automatisch kan generen. Zo is het voor een mens bijvoorbeeld makkelijk om te zien of een foto een auto bevat of niet, maar dit uitdrukken in een functie die een computer kan gebruiken is bijna onmogelijk. Gezien mensen het kunnen kan het wel gesteld worden dat een algoritme die hiertoe in staat is kan bestaan. Met machine learning kan dit algoritme gemaakt worden.

Een machine learning algoritme moet getraind worden. Hiervoor is een dataset nodig. De dataset moet code bevatten en kwaliteitsproblemen die in de code zitten. Er kan een nieuwe dataset gemaakt worden, maar gezien de gelimiteerde tijd die beschikbaar is, moet er gezocht worden naar een al bestaande dataset.

De programmeur moet regelmatig feedback krijgen uit het systeem over de kwaliteit van zijn code. Door gebruik te maken van een CI/CD pipeline kan het model in de back-end automatisch geactiveerd worden op alle code die aan het versiebeheer systeem toegevoegd wordt. De pipeline stuurt de code naar de back-end die het analyseert.
De resultaten hiervan kunnen inzichtelijk gemaakt worden in een dashboard applicatie. Deze applicatie laat de programmeur zien waar in de code de problemen gedetecteerd zijn.

Ook is een systeem bedacht om de programmeur het resultaat van het model te laten beoordelen. Na de code gepusht te hebben, beoordeelt de gebruiker de analyse van het model op accuraatheid. Andere programmeurs kunnen dit ook doen. Als een bepaald aantal programmeurs het eens zijn dat een resultaat accuraat is, kan deze toegevoegd worden aan de dataset. Op deze manier wordt de dataset groter, wat de prestaties van het model verbeterd. Als het resultaat niet accuraat is of er ontstaat geen overeenstemming, wordt deze niet aan de

dataset toegevoegd. De gebruiker moet het resultaat dat het model geproduceerd heeft beoordelen, niet de code die beoordeeld is.
Ook forceert dit de gebruiker om kritisch na te denken over de code die geschreven is.

Om deze systemen te realiseren moet er onderzoek gedaan worden naar bestaande onderzoeken op dit gebied, geschikte datasets, welke tools en frameworks gebruikt kunnen worden voor het ontwikkelen van de applicatie.

Nadat er een dataset gevonden is, wordt deze gebruikt om een machine learning model te trainen. Hiervoor moet een onderzoek gedaan worden naar welk soort model het beste gebruikt kan worden voor deze opdracht. Er zijn verschillende soorten algoritmen beschikbaar, bijvoorbeeld neural networks en decision trees. Hiervoor moet gekeken worden naar welke algoritmen soortgelijke projecten gebruikt hebben, welke alternatieven er zijn en wat het meest geschikt is voor deze specifieke applicatie.

Om de prestaties van het model te verbeteren, moet de dataset uitgebreid worden. Als genoeg programmeurs het eens zijn over een beoordeling, kan deze aan de dataset toegevoegd worden. Hiermee kan de dataset verder uitgebreid worden. Dit moet via het dashboard kunnen. Hiervoor moet onderzoek gedaan worden naar hoe de validiteit van een beoordeling bepaald kan worden.

Voor het onderzoek is de volgende hoofdvraag geformuleerd:

> Hoe kan een systeem gebouwd worden om de kwaliteit van code te analyseren met machine learning?

Met deze hoofdvraag wordt onderzoek gedaan naar welke frameworks gebruikt worden, welke onderzoeken al uitgevoerd zijn, wat OVSoftware al doet op het gebied van codekwaliteit en hoe het systeem nuttig gebruikt kan worden. Hieruit zijn de volgende deelvragen opgesteld.

- Wat doet OVSoftware al op het gebied van codekwaliteit van codekwaliteit?
- Op welke manier kan code beoordeeld worden?
- Hoe is machine learning relevant voor dit project?
- Hoe kan de dataset opgeslagen worden?
- Op welke manier kunnen de resultaten opgeslagen worden?
- Welke machine learning technieken kunnen gebruikt worden voor het maken van het model?
- Hoe kan een systeem gebouwd worden dat de resultaten van het model nuttig kan weergeven?
- Hoe kan de dataset verder uitgebreid worden?
- Hoe kan het model op een nuttige manier uitgevoerd worden?

4 Aanpak

4.1 Onderzoek

Het onderzoek bestaat uit 10 deelvragen. Deze deelvragen zijn zo geformuleerd dat de nodige informatie voor het realiseren van het project verkregen wordt.

4.1.1 **Deelvraag 1: Wat doet OVSoftware al op het gebied van codekwaliteit?**

OVSoftware hecht veel waarde aan de kwaliteit van de geproduceerde code. Er zijn dus al een aantal dingen die gedaan worden om deze kwaliteit te waarborgen. Voor deze deelvraag wordt een gesprek gevoerd met collega's om de nodige informatie te krijgen.

4.1.2 **Deelvraag 2: Op welke manier kan code beoordeeld worden?**

Voor deze deelvraag is een literatuuronderzoek gedaan naar verschillende manieren om code te beoordelen. Zo is er gekeken naar het verschil tussen anti-patterns en code smells, welke soorten code smells bestaan en hoe deze beoordelingen tot stand komen.
Informatie wordt verkregen door het gebruik van zoekmachines om te zoeken naar relevante literatuur. Ook wordt het boek [4], die is aangeraden vanuit OVSoftware.

4.1.3 **Deelvraag 3: Hoe is machine learning relevant voor dit project?**

Voor deze deelvraag is er onderzoek gedaan naar welke mogelijkheden machine learning biedt voor dit project. Hiervoor is een literatuuronderzoek uitgevoerd om te bepalen welke sterke punten machine learning heeft en hoe deze relevant zijn voor dit project.

4.1.4 **Deelvraag 4: Welke bestaande datasets kunnen gebruikt worden?**

Voor deze deelvraag wordt gezocht naar bestaande datasets en de onderzoeken waar deze voor gebruikt zijn. Hiermee worden één of meer datasets gekozen die gebruikt kunnen worden voor het project.
De datasets worden gevonden door het gebruik van zoekmachines. Ook wordt er gekeken naar eventuele onderzoeken die bij de dataset horen.

4.1.5 **Deelvraag 5: Hoe kan de dataset opgeslagen worden?**

Voor deze deelvraag wordt gekeken naar beschikbare technieken voor data opslaan en hoe deze gebruikt kunnen worden voor de structuur van de dataset. Hiervoor wordt een literatuuronderzoek uitgevoerd om verschillende opties te vinden en gesproken met collega's

over welke technieken al gebruikt worden binnen OVSoftware. De beste optie wordt gekozen. Hiervoor moet er rekening gehouden worden met de gekozen dataset; de dataset moet met de techniek kunnen werken. Ook moet de gekozen techniek het uitbreiden van de dataset toestaan, voor deelvraag 9.

4.1.6 **Deelvraag 6: Op welke manier kunnen de resultaten opgeslagen worden?**

De resultaten van het model moeten ook opgeslagen worden. Er wordt gesproken met collega's over welke technieken al gebruikt worden binnen OVSoftware en hoe deze gebruikt kunnen worden voor de data die uit het model komt. Ook moet er rekening gehouden worden met welke techniek gebruikt wordt voor het opslaan van de dataset bepaald in deelvraag 5 omdat dit soortgelijke data is.

4.1.7 **Deelvraag 7: Welke machine learning technieken kunnen gebruikt worden voor het maken van het model?**

Voor deze deelvraag wordt er gekeken naar eerder gemaakt onderzoek, op het gebied van machine learning met code en tekst analyse om te kijken welke technieken het meest gepast zijn. Om deze onderzoeken te vinden wordt er gebruik gemaakt van zoekmachines en arXiv om artikelen te vinden over machine learning op code en tekst.
Ook kunnen de onderzoeken van de gevonden datasets van deelvraag 4 geraadpleegd worden.

4.1.8 **Deelvraag 8: Hoe kan een systeem gebouwd worden dat de resultaten van het model nuttig kan weergeven?**

Met collega's worden de frameworks en technieken die al binnen OVSoftware gebruikt worden voor het ontwikkelen van applicaties besproken. Deze worden verder onderzocht om te kijken naar functionaliteit om vervolgens de meest geschikten te kunnen selecteren, rekening houdend met bestaande kennis van de afstudeerder.

4.1.9 **Deelvraag 9: Hoe kan de dataset verder uitgebreid worden?**

De hoeveelheid data heeft veel invloed op de prestaties van het model. Er wordt dus gekeken naar de mogelijkheid om een terugkoppeling te implementeren in de front-end waarmee gebruikers de accuraatheid van resultaten van het model kunnen beoordelen, om hiermee de dataset uit te breiden.
Hiervoor moet bepaald worden hoe de back-end om moet gaan met beoordelingen, waarvoor rekening gehouden moet worden met de manier de dataset en resultaten opgeslagen worden zoals bepaald in deelvraag 5 en 6.

4.1.10 Deelvraag 10: Hoe kan het model op een nuttige manier uitgevoerd worden?

Om resultaten uit het model te krijgen, moet het uitgevoerd worden. Als het teveel moeite kost om het model uit te voeren, zal het niet uitgevoerd worden. Het is belangrijk dat het model op een eenvoudige en nuttige manier uitgevoerd kan worden, bijvoorbeeld door gebruik te maken van een CI/CD pipeline.
Aan collega's wordt gevraagd welke pipeline systemen al gebruikt worden binnen OVSoftware. Vervolgens wordt gekeken naar de mogelijkheid om deze te gebruiker om de code die geanalyseerd moet worden naar de back-end te sturen.

4.2 Ontwerp

De ontwerpen voor het project worden gemaakt tijdens de onderzoek en realisatiefase van het project. Een ontwerp is meestal fluïde, naarmate er meer kennis en duidelijkheid over het project ontstaat tijdens de ontwikkeling verandert het ontwerp.

Voor de volgende dingen moet er een ontwerp gemaakt worden:

- Front-end van het dashboard
- Database
- Communicatie tussen onderdelen
- API

De ontwerpen voor deze onderdelen worden gemaakt op basis van technische en functionele eisen die tijdens de onderzoeksfase vastgesteld worden in een Design Document. In ditzelfde document worden de ontwerpen voor deze onderdelen uitgewerkt.

De front-end van het dashboard bestaat uit een schets die beschrijft hoe dit er uit moet zien. Het ontwerp van de database gaat zowel om welke technieken gebruikt worden en hoe het data model er uit ziet.

De communicatie tussen verschillende onderdelen gaat om hoe data tussen de database, de front-end, de back-end en het model verplaatst wordt. Dit bestaat uit een data flow diagram.

Voor de communicatie tussen verschillende onderdelen wordt een API gebruikt. De API specificatie beschrijft welke onderdelen welke informatie versturen naar welke andere onderdelen.

4.3 Realisatie

Het project wordt opgedeeld in 3 delen:

- Model
- Dashboard

- Pipeline

Het ontwikkelen van deze onderdelen gebeurt in deze volgorde, gebaseerd op hoe belangrijk ze zijn en hoeveel tijd ze naar verwachting kosten.

Tijdens het ontwikkelen van het model is er de afspraak gemaakt om twee keer in de week een voortgangsgesprek te voeren met de begeleiders. In deze gesprekken wordt de voortgang gemaakt sinds het laatste gesprek besproken en het werk dat gedaan gaat worden tot het volgende gesprek. Voor dit onderdeel zijn 4 weken gepland.

Tijdens deze periode wordt er ook begonnen aan het maken van de ontwerpen voor de verschillende onderdelen die in de volgende fase uitgewerkt worden.

Voor de ontwikkeling van het dashboard en de pipeline wordt er SCRUM gebruikt. Elke ochtend wordt een stand-up gehouden waarin het werk dat de vorige dag gemaakt is en die dag gemaakt gaat worden besproken wordt. Het werk wordt bijgehouden op een Issue Board. Sprints worden niet strikt gehanteerd omdat dit niet voldoende meerwaarde heeft voor een project dat individueel uitgevoerd wordt. In de planning staan er voor het dashboard 8 weken en voor de pipeline 2.

5 Resultaten

5.1 Onderzoek

De eerste weken van de stage zijn besteed aan het uitvoeren van de onderzoeken, bekend worden met de werkomgeving en collega's en het schrijven van het plan van aanpak. Ook is er tijd gestopt in het leren van de frameworks die gebruikt worden voor het maken van het dashboard, Angular en .NET Core. Om meer kennis te krijgen over de kwaliteitseisen binnen OVSoftware is het boek [4] gelezen.

5.1.1.1 Hoofdvraag

Hoe kan een systeem gebouwd worden dat met machine learning de kwaliteit van code kan analyseren?

De hoofdvraag dient als overkoepelende onderzoeksvraag waar de deelvragen op gebaseerd zijn. De vraag die gesteld wordt is eigenlijk "Hoe kan het project gemaakt worden?". Het antwoord op de hoofdvraag is dus eigenlijk een samenvatting van de deelvragen.

Het systeem bestaat uit 3 onderdelen: het model, het dashboard en de pipeline. Het model stuurt code naar het dashboard waar het geanalyseerd wordt. Deze wordt gemaakt met Azure Pipelines. Het dashboard wordt gemaakt met Angular en ASP.NET, met SQL Server als database en Entity Framework voor de verbinding tussen de back-end en database.

De MLCQ dataset die gebruikt wordt voor het maken van het model is opgeslagen in een spreadsheet formaat. De code, waarnaar hyperlinks in de spreadsheet staan, wordt gedownload en geconverteerd naar een syntax tree. Dit wordt opgeslagen als losse bestanden.

De resultaten van het model worden op dezelfde manier opgeslagen, maar in plaats van een spreadsheet bestand in een SQL database.

5.1.1.2 Deelvraag 1: Wat doet OVSoftware al op het gebied van code kwaliteit

Om de kwaliteit te waarborgen hanteert OVSoftware code reviews. Bij een code review beoordeelt een programmeur de code die iemand anders heeft geschreven. OVSoftware medewerkers worden opgeleid binnen het bedrijf met de kwaliteitsstandaarden [4] opgezet door de Software Improvement Group.

Ook wordt er gebruik gemaakt van het SIG dashboard. Deze heeft ongeveer dezelfde functie als dit project; code analyseren en hier een beoordeling aan geven. Dit dashboard verschilt van dit project door het gebruik van analyse gebaseerd op regels, niet machine learning.

5.1.1.3 Deelvraag 2: Op welke manier kan code beoordeeld worden?

Code kan beoordeeld worden op objectieve, meetbare manieren en subjectieve manieren. Een voorbeeld van een objectieve meting is de hoeveelheid regels code in een functie. Een subjectieve meting is de leesbaarheid van code. Hoe leesbaar code is voor iemand verschilt per persoon.

Ook zijn er verschillende in de soorten problemen waarnaar gekeken kan worden. Deze worden opgedeeld in 'Code smell' en 'Anti-pattern'.
Een code smell is een aspect in de code die een hint geeft dat er misschien een probleem is, bijvoorbeeld een functie met heel veel regels. De specifieke problemen verschillen, maar iets klopt waarschijnlijk niet.

Een anti-pattern is een design pattern die altijd tot problemen zal lijden [5]. Een design pattern geeft voor een generiek probleem een generieke oplossing [6]. Een voorbeeld hiervan is een Singleton. De Singleton zorgt ervoor dat voor een gegeven class er maar één instantie kan bestaan. Dit is handig als er maar één instantie van een object nodig is, bijvoorbeeld de verbinding met de database. Een anti-pattern is een design pattern die in zijn implementatie ineffectief is of juist contraproductief. De Singleton is hier ook een voorbeeld van. Als er maar één instantie van een class kan bestaan introduceert dit een globale staat in het systeem, wat het gedrag van het systeem onvoorspelbaar maakt.

Code smells zijn interessant voor machine learning omdat deze niet altijd een probleem zijn. Waar een anti-pattern per definitie altijd tot problemen zal lijden, zijn er legitieme redenen om een implementatie te gebruiken die een code smell is. Deze scenario's zijn moeilijk om op basis van regels te detecteren, omdat ze grotendeels subjectief zijn. Machine learning kan hiervoor de oplossing bieden, omdat deze niet afhankelijk is van precies gedefinieerde regels.

5.1.1.4 Deelvraag 3: Hoe is machine learning relevant voor dit project?

Machine learning is een techniek om patronen te herkennen in data. Deze patronen zijn te ingewikkeld om handmatig te kunnen programmeren en te tijdrovend voor een mens om handmatig te bekijken. Op basis van een algoritme kan een model gemaakt worden die deze taken snel en accuraat uit kan voeren.

Er zijn verschillende soorten algoritmen. Voor het doel van ontwikkeling kunnen deze opgedeeld worden in twee categorieën: traditionele algoritmen en neural networks/deep learning. Deze scheiding bestaat door de bijbehorende ontwikkelingsstijl. Een traditioneel algoritme wordt gemaakt gemaakt door een instantie van een model te maken, deze een aantal parameters te geven en vervolgens te trainen op de dataset. Bij een neural network moet de structuur van de neuronen waar het netwerk uit elkaar gedefinieerd worden. Neural networks hebben het voordeel dat deze beter complexe relaties kunnen leren. In ruil hiervoor kost het meer

rekenkracht en dus tijd om een neural network te trainen en uit te voeren. Traditionele algoritmen hebben hier geen problemen mee.

Een mens zou in staat zijn om handmatig elke stuk code te beoordelen op problemen. Dit kost echter ontzettend veel tijd. Door gebruik te maken van machine learning kan in een aantal seconden de hele codebase beoordeeld worden.

5.1.1.5 Deelvraag 4: Welke bestaande datasets kunnen gebruikt worden?

Om een model te trainen is een dataset nodig. Deze bevat voorbeelden van wat er gedetecteerd moet worden. In dit geval een stuk code met eventuele problemen hierin.
Voor het onderzoek zijn meerdere datasets bekeken en beoordeeld op geschiktheid voor dit project. De enige dataset die gebruikt kan worden voor dit project is de MLCQ dataset. Deze dataset bestaat uit ongeveer 15000 samples, heeft 4 criteria waar code op beoordeeld wordt en geeft voor elk stuk code aan hoe serieus het probleem is [1].

Er zijn in de dataset twee waarden die als label gebruikt kunnen worden voor het model: smell en severity. Deze kolommen geven aan om wat voor probleem het gaat en hoe serieus dit is. Met deze waarden output en de code als input kan deze dataset gebruikt worden om een model te trainen om deze smells te detecteren en de serieusheid ervan bepalen.

De 4 smells die MLCQ bevat zijn:

- Blob
  Een class/component die alles doet.
- Data class
  Een class die enkel data bevat, zonder functionaliteit.
- Long method
  Een method die te lang is.
- Feature envy
  Een method in een class die meer data van andere classes gebruikt dan die van zichzelf.

Met 4 niveaus aan serieusheid:

- None
  Dit betekent dat de smell niet in het stuk code zit.
- Minor
- Major
- Critica

In deze tabel staan de hoeveelheden van elke smell en serieusheid [1]:

Smell	Totaal	Critical	Major	Minor	None
Blob	4019	127	312	535	3045
Data Class	4021	146	401	510	2964
Long Method	3362	78	274	454	2556
Feature Envy	3337	24	142	288	2883

5.1.1.6 Deelvraag 5: Hoe kan de dataset opgeslagen worden?

De MLCQ dataset is opgeslagen in een spreadsheet formaat, bestaande uit 15 kolommen.
Van deze kolommen zijn er 8 relevant:

- id
- Smell
- Severity
- Type
- Link
- Repository
- Path
- Start_line
- End_line

Smell en Severity zijn labels. Type geeft aan of het om een functie of class gaat. Link is een link naar de relevante code in de repository. Repository is een link naar de repository zelf. Path is het pad naar het bestand waar de sample over gaat in de repository. Start_line en end_line geven het begin en einde van het code fragment in het bestand aan.

Er zijn nog een aantal overige kolommen, maar deze zijn niet relevant voor dit project:

- reviewer_id
- sample_id
- review_timestampt
- code_name
- commit_hash
- is_from_indsutry_relevant_project

Om de code te kunnen gebruiken moet deze gedownload worden uit de repository. Deze code kan vervolgens opgeslagen worden in bestanden. De spreadsheet bevat een ID kolom. De naam van het bestand is dit ID. Op deze manier kan een bestand aan een kolom gekoppeld kan worden.

5.1.1.7 Deelvraag 6: Op welke manier kunnen de resultaten opgeslagen worden?

Een model produceert resultaten in hetzelfde formaat als de data waarmee het getraind is. Dit betekent dat de resultaten van het model op een soortgelijke manier opgeslagen worden als de originele dataset. In plaats van een spreadsheet, wordt er gebruik gemaakt van een SQL database. Deze database heeft ook een tabellen en kolommen structuur.

Een SQL database schaalt veel beter; de MLCQ dataset heeft 15000 samples, wat best te doen is met een spreadsheet, maar als het model veel gebruikt wordt en er veel resultaten worden gegenereerd, schaalt dit niet goed. Het is dan beter voor de prestaties van het systeem om een SQL database te gebruiken.

De database die gebruikt wordt is SQL Server. SQL Server wordt al gebruikt binnen OVSoftware.

5.1.1.8 Deelvraag 7: Welke machine learning technieken kunnen gebruikt worden voor het maken van het model?

Er zijn verschillende algoritmen waarmee machine learning modellen gemaakt kunnen worden. Deze algoritmen presteren op verschillende soorten problemen beter of slechter dan andere algoritmen.

Door te kijken naar eerdere onderzoeken naar het analyseren van code met machine learning is een selectie van algoritmen gemaakt die waarschijnlijk het beste werken voor dit project. Decision Trees, Support Vector Machines, Naïve bayes en Long Short Term Memory Neural Networks vertoonden de beste prestaties in eerdere onderzoeken [2]. Deze algoritmen worden getest op de dataset en van de beste wordt het uiteindelijke model gemaakt.

Uit ditzelfde onderzoek wordt duidelijk dat veel eerder pogingen tot machine learning op broncode een snytax tree gebruiken om de code te representeren. Een syntax tree is een structuur om code op een generieke manier te representeren. Zo is het overbodig om te weten welke naam een variabele krijgt, enkel dat de declaratie gebeurt is.

Voor frameworks die gebruikt worden voor machine learning, kennis bij de student ligt bij scikit-learn [7] en TensorFlow [8]. Deze frameworks worden ook al gebrtuikt binnen OVSoftware en dus zullen ze gebruikt worden voor ontwikkeling van het model.

5.1.1.9 Deelvraag 8: Hoe kan een systeem gebouwd worden dat de resultaten van het model nuttig kan weergeven

Om de programmeur inzicht te geven in de resultaten van het model wordt een dashboard gebruikt. Hiervoor moeten frameworks geselecteerd worden voor het bouwen van de front- en back-end, en database.

Voor de front- en back-end is gekozen voor het gebruik van Angular en ASP.NET, respectievelijk voor de front-end en back-end. Deze frameworks worden al gebruikt binnen het bedrijf en de student wil hier graag meer over leren.

Voor de database wordt SQL Server. Entity Framework wordt gebruik voor communicatie tussen ASP.NET en SQL Server. Ook hier is de keuze gebaseerd op wat al binnen het bedrijf gebruikt wordt.

Voor Authenticatie wordt gebruik gemaakt van IdentityServer. Dit framework is een implementatie van OpenID voor ASP.NET.

5.1.1.10 Deelvraag 9: Hoe kan de dataset verder uitgebreid worden?

De prestaties van een model worden grotendeels bepaald door de hoeveelheid data die gebruikt is om deze te trainen. Om de prestaties van het model beter te maken, kan het model gebruikt worden om de dataset uit te breiden. Hiervoor moet het model wel goede resultaten leveren en dat kan alleen een persoon beoordelen.

Wanneer een persoon de resultaten bekijkt in het dashboard, moet hij hier het resultaat kunnen beoordelen; aangeven of hij het er wel of niet mee eens is. Als deze gebruiker, of meerdere gebruikers, het eens zijn met de beoordeling is dit waarschijnlijk een goed resultaat. Het kan dan gebruikt worden in het opnieuw trainen van het model, deze keer met een grotere dataset. Dit zou op lange termijn de prestaties van het model moeten verbeteren.

5.1.1.11 Deelvraag 10: Hoe kan het model op een nuttige manier uitgevoerd worden?

Als het model niet uitgevoerd wordt, is het natuurlijk vrij nutteloos. Om het model regelmatig uit te voeren op een nuttig moment kan een CI/CD pipeline gebruikt worden. Een CI/CD pipeline is hiervoor handig omdat deze uitgevoerd wordt elkaar keer dat er code aan het versiebeheer systeem wordt toegevoegd. In de pipeline kan de nieuwe code naar het dashboard verstuurd worden, waar het geanalyseerd wordt.
Het CI/CD systeem dat binnen OVSoftware het meest gebruikt wordt is Microsoft Azure Pipelines.

Azure Pipelines bouwt en test code automatisch, zodat deze beschikbaar gemaakt kan worden voor anderen. Het werkt met vrijwel elke programmeertaal en projecttype. De artefacten die door dit systeem gemaakt worden kunnen gedeployed worden naar verschillende targets. [7] Meerdere testing frameworks kunnen gebruikt worden. Ook is er de mogelijkheid voor het uitvoeren van scripts. [8]

In principe kan dus alle code uitgevoerd worden in de pipeline. De code kan verstuurd worden naar het dashboard. Deze slaat de resultaten op in de database.

Om het model in het dashboard uit te voeren kunnen end-points gemaakt worden in de back-end. Wanneer code wordt verstuurd naar deze end-points wordt deze code verwerkt en geanalyseerd. De resultaten van de analyse worden opgeslagen in de database.

5.2 Ontwerpen

Na het onderzoek voltooid te hebben zijn de eerste versies van de ontwerpen voor de verschillende onderdelen gemaakt.
Als eerste zijn een aantal functionele en niet-functionele eisen opgesteld. Deze zijn vooral gebaseerd op de onderzoeken die uitgevoerd zijn. Op basis van deze eisen kon een ontwerp voor de front-end, de database, hoe alle onderdelen met elkaar communiceren en een API ontwerp. Deze ontwerpen zijn verwerkt in het Design Document.

5.2.1 Eisen

De Functionele en niet-functionele eisen zijn opgesteld om een overzicht te geven van welke functionaliteit het systeem moet hebben en hoe deze geïmplementeerd moeten worden.
De functionele eisen beschrijven wat het systeem moet doen, bijvoorbeeld: “Een beoordeeld resultaat van het model kunnen naar de database verstuurd worden.”.

De niet-functionele eisen beschrijven hoe het systeem gebouwd moet worden. Deze eisen worden hierom ook wel technische eisen genoemd.
De prioriteiten van de eisen zijn gebaseerd op de MoSCoW methode.

5.2.1.1 Functionele Eisen

Dashboard

Eis	Prioriteit
Code en de daarin gevonden problemen kunnen gevisualiseerd worden.	Must
De problemen gevonden door het model kunnen beoordeeld worden door de programmeur.	Must
Een beoordeeld resultaat van het model kunnen naar de database verstuurd worden.	Must
De gebruiker kan inloggen	Must
Een resultaat dat niet goed beoordeeld is kan apart opgeslagen worden in de database.	Must

De gebruiker nieuwe code uploaden	Should

Model

Eis	Prioriteit
Het model wordt in een CI/CD pipeline uitgevoerd op nieuwe code.	Must
Het model wordt uitgevoerd op elke commit.	Must
De resultaten van het model worden in het dashboard zichtbaar.	Must

Database

Eis	Prioriteit
Wanneer een resultaat voldoende beoordeeld is, wordt deze uit de lijst van problemen gehaald en aan de dataset toegevoegd.	Must
Andere resultaten worden apart opgeslagen	Must
Resultaten kunnen later bekeken worden	Must
Wanneer een resultaat voldoende beoordeeld is, wordt deze verplaatst naar de dataset, of juist niet.	Must

Pipeline

Eis	Prioriteit
De pipeline voert het model uit op de code die gecommit wordt.	Must
De pipeline stuurt de resultaten naar de database.	Must

5.2.1.2 Niet-functionele eisen

Dashboard

Eis	Prioriteit
De front-end is gemaakt met het Angular framework	Must
De back-end is gemaakt met het ASP.NET framework	Must
Het dashboard is een web applicatie.	Must
De resultaten worden opgehaald uit de database	Must
Beoordelingen zijn gekoppeld aan de ingelogde gebruiker	Must
De gebruiker moet ingelogd zijn om toegang te krijgen tot het dashboard	Must
Communicatie met de front-end gebeurt via HTTPS	Must
De API is afgeschermd voor geautoriseerden	Must

Model

Eis	Prioriteit
Het model is in staat om Java code te analyseren.	Must
Het model is in staat om de serieusheid problemen te bepalen	Should

Database

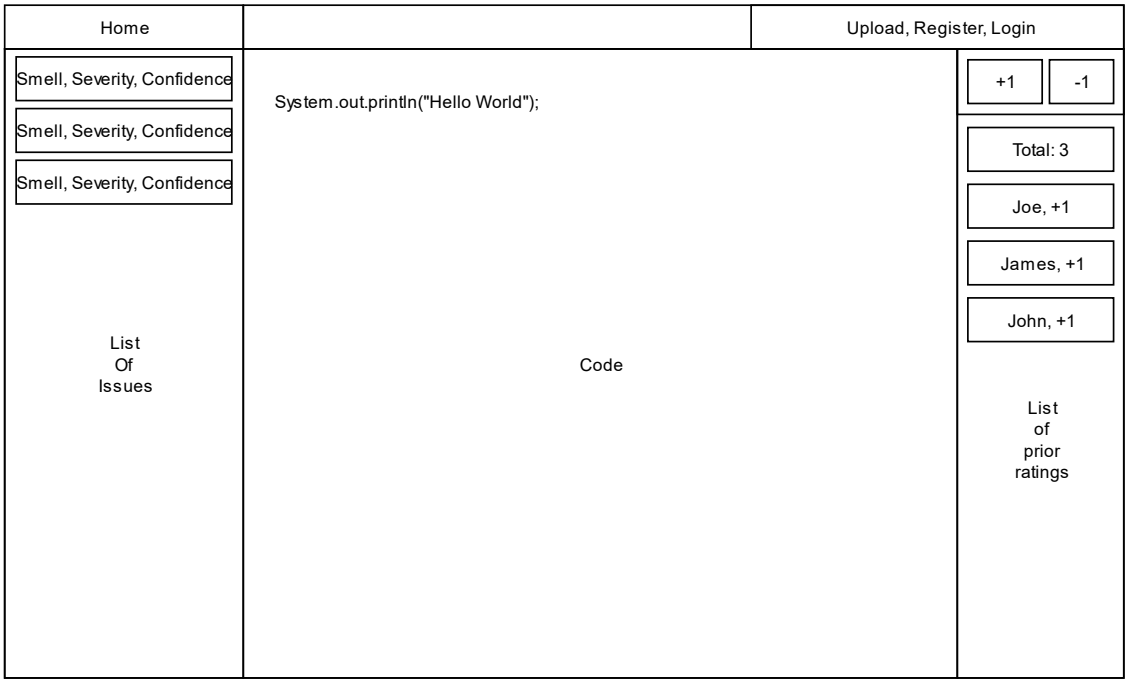
Eis	Prioriteit
De dataset kan opgeslagen worden	Must
De dataset kan uitgebreid worden	Must
De resultaten van het model kunnen opgeslagen worden	Must
Data kan later teruggehaald worden	Must

Pipeline

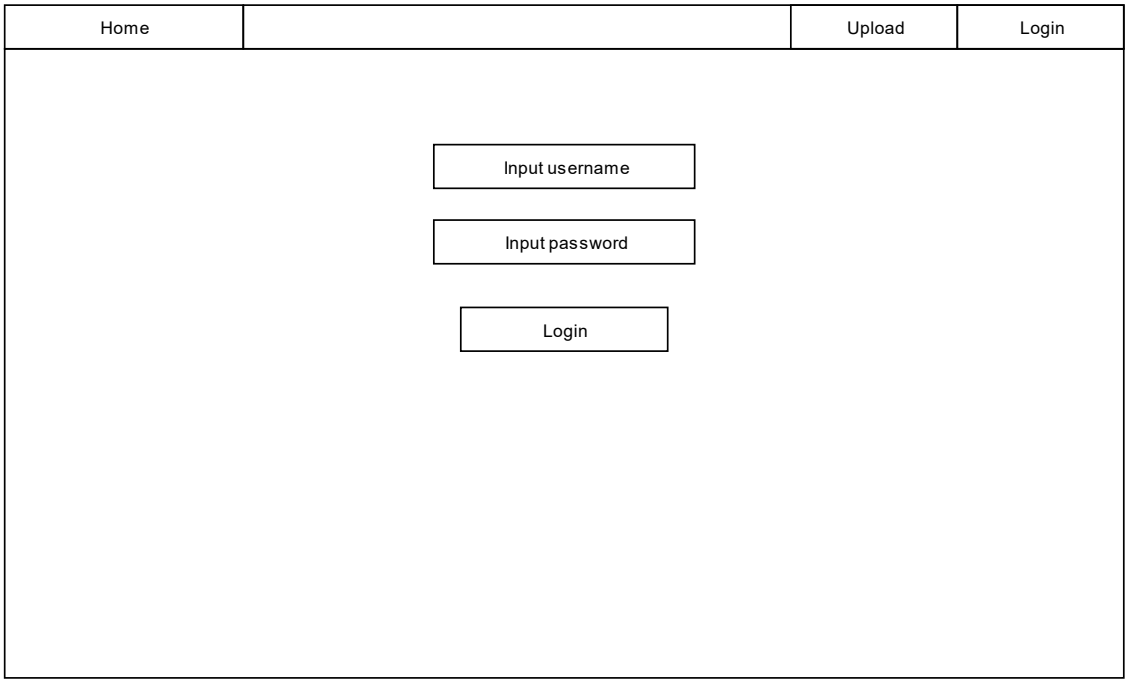
Eis	Prioriteit
De resultaten worden in de database opgeslagen	Must
De pipeline is een Azure pipeline	Must

5.2.2 Front-End Design

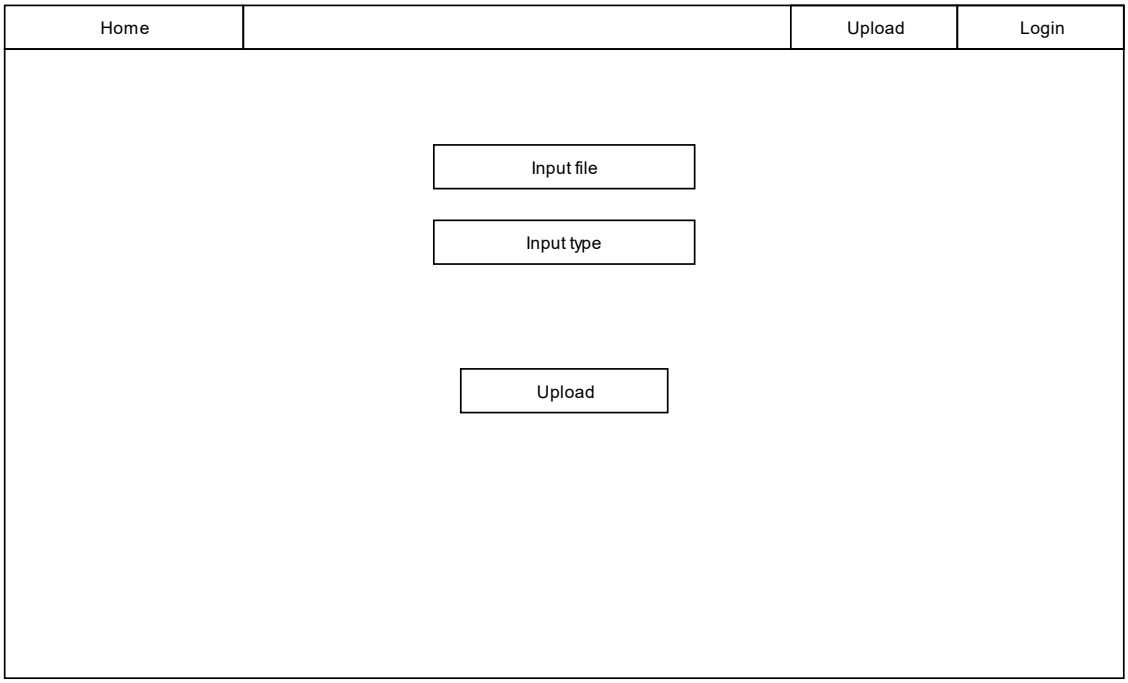
De design van het front-end is een wireframe schets met daarin de beoogde lay-out van de pagina. De verschillende elementen in de pagina, zoals knoppen, worden beschreven.



Figuur 1 Home pagina



Figuur 2: Login pagina

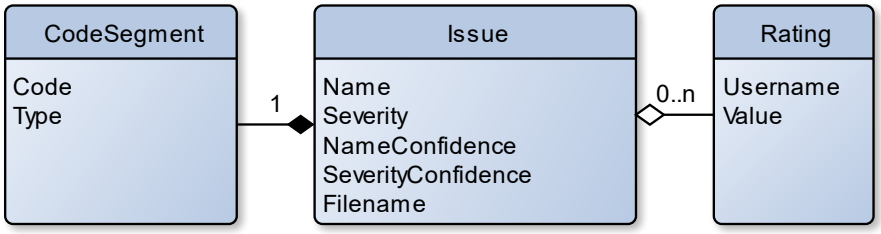


Figuur 3: Upload pagina

Deze ontwerpen zijn opgesteld op basis van de gestelde eisen

5.2.3 Data Model

Het data model beschrijft met welke structuur de data in de database opgeslagen wordt. In het ontwerp betekent dit dat er beschreven wordt welke objecten er in de database worden opgeslagen en wat de relaties tussen deze objecten zijn.



Figuur 4: Data model

Het centrale object in het model is een Issue. Dit representeert een gevonden probleem inde code. Deze heeft een

- Name, welk probleem het is
- Severity, hoe serieus het probleem is
- NameConfidence, hoe zeker het model het is van het probleem
- SeverityConfidence, hoe zeker het model het is over de serieusheid
- Filename, de naam van het bestand waar de code origineel uitkwam.

In de eerste poging tot het maken van dit data model was de relatie tussen Issues en CodeSegments omgedraaid; een CodeSegment heeft één of meerdere issues, en nul of meer ratings. Dit bleek echter niet een handige manier te zijn door de structuur van de front-end. Eerst wordt er een lijst van issues laten zien, waar vervolgens een CodeSegment bij hoort. De relatie is dus omgedraaid.

5.3 Maken van het model

Om het model te kunnen trainen moet de dataset klaargemaakt worden; het algoritme verwacht dat de data een bepaalde structuur heeft. Zodra dit gedaan is kan het model getraind worden.

5.3.1 Klaarmaken Dataset

Uit het onderzoek is gebleken dat de MLCQ dataset het meest geschikt is voor dit project, met een syntax tree als beste representatie voor de code [deelvraag 7]. Dit is een methode om de structuur van de code te weergeven in een boomstructuur, de namen die variabelen, functies, etc. krijgen maken dan niet uit.

In dit onderdeel wordt de structuur van de dataset, het downloaden, krimpen, parsen en gebruiksklaar maken van de dataset beschreven. Ook worden de ondervonden problemen beschreven.

5.3.1.1 Structuur

De dataset is opgeslagen in een spreadsheet formaat. Hierbij is elke rij een stuk code met bepaalde data die bij dat stukje code hoort.
De 4 belangrijkste kolommen zijn: link, smell, severity en type.
'Link' is een link naar de GitHub pagina waar de code op is geslagen. Er zijn ook twee kolommen die aangeven in welke regels het relevante stukje begint en eindigt.

De smell kolom geeft aan om welke smell het gaat. De dataset bevat: feature envy, long method, blob en data class.

Severity geeft voor het stuk code aan hoe serieus het probleem is. Dit kan vier waarden hebben: none, major, minor en critical.

Type geeft aan of het stuk code om een functie of een class gaat.

Feature envy en long method kunnen alleen in functies voorkomen. Blob en data class enkel in classes.

5.3.1.2 Downloaden

De code samples zijn dus opgeslagen in de vorm van een link naar een GitHub pagina. Deze pagina moet dus gedownload worden. Een voorbeeld van een link is:

> https://github.com/apache/syncope/blob/114c412afbfba24ffb4fbc804e5308a823a16a78/client/idrepo/ui/src/main/java/org/apache/syncope/client/ui/commons/ConnIdSpecialName.java/#L35-L37

Deze link is echter niet meteen geschikt om te downloaden. Naast de code is zijn er op deze pagina veel extra dingen zichtbaar. Wat we nodig hebben is een link naar de "raw" pagina, waar alleen de code instaat. Deze link ziet er zo uit:

> https://raw.githubusercontent.com/apache/syncope/114c412afbfba24ffb4fbc804e5308a823a16a78/client/idrepo/ui/src/main/java/org/apache/syncope/client/ui/commons/ConnIdSpecialName.java

Voor het converteren van de links en vervolgens het downloaden van de pagina is een Python script gemaakt [Bijlage 1]. De samples worden opgeslagen als bestanden in een map. Sommige samples verwijzen naar dezelfde code, met een andere smell. In dit geval wordt de code duplicaat opgeslagen.

Een probleem dat hier voor kan komen is dat er iets mis is gegaan in het formatteren van de bestanden tijdens het uploaden naar GitHub. Verschillende besturingssystemen geven op verschillende manieren aan dat een regel eindigt. Niet elk stuk code gebruikt dus dezelfde regeleindes. Wanneer dit bestand dan gedownload wordt, wordt er tussen elke regel een extra, lege regel toegevoegd. Dit levert problemen bij de volgende stap.

5.3.1.3 Krimpen

De eerste link die in de vorige stap genoemd wordt eindigt op #L35-L37. Op GitHub is het mogelijk om te linken naar specifieke regels binnen een bestand. Dit kan met de raw link echter niet. In principe hoeft alleen dit deel van de pagina gedownload te worden, maar omdat raw links het selecteren van regels niet ondersteund, wordt het hele bestand gedownload. In deze stap worden de bestanden gekrompen naar enkel de nodige regels. Hiervoor wordt weer een Python script gebruikt [Bijlage 2].

Het probleem dat in het vorige onderdeel genoemd is komt nu naar voren. Het besturingssysteem weet niet hoe het met andere regeleindes om moet gaan. Het kan dan voorkomen dat elk regeleinde als twee regeleindes gezien wordt. De regels die in de dataset vermeld staan verwijzen dan niet naar de regels die gebruikt moeten worden. De sample kan dan niet gebruikt worden. De totale hoeveelheid samples waar dit bij voorkomt is ongeveer 30.
Zie hieronder twee afbeeldingen met een voorbeeld ter illustratie:

```java
public void run() throws IOException {
    FileInputStream fis = new FileInputStream(file);
    XSSFWorkbook book = new XSSFWorkbook(fis);
    XSSFSheet sheet = book.getSheetAt(0);

    for (Row row : sheet) {
        makeSample(row);
        System.out.println();
    }
}
```

Figuur 5: Correcte code

```
public void run() throws IOException {

    FileInputStream fis = new FileInputStream(file);

    XSSFWorkbook book = new XSSFWorkbook(fis);

    XSSFSheet sheet = book.getSheetAt(0);



    for (Row row : sheet) {
```

Figuur 6: Foute code

In principe is het mogelijk om de extra regels te verwijderen, maar het is niet programmatisch mogelijk om te bepalen welke bestanden wel of niet aan dit probleem voldoen op het moment van opslaan. Verder, de lage hoeveelheid bestanden waar dit probleem in voorkomt, 30 van e 15000, betekent dat er de keuze is gemaakt om deze bestanden niet te gebruiken voor het trainen in plaats van tijd besteden aan een oplossing vinden.

5.3.1.4 Converteren naar syntax tree

Uit deelvraag 7 is gebleken is gebleken dat het gebruik van een syntax tree handig is om de code te representeren.
Voor het converteren naar een syntax tree wordt een tool genaamd ANTLR [9] gebruikt. Met deze tool kan code geconverteerd worden naar een syntax tree.

Om deze conversie uit te voeren werd een script gemaakt die de tool uitvoert op code. Dit script werd als eerst geschreven in de Python programmeertaal. Uiteindelijk bleek dit een slechte keuze te zijn, omdat de Python versie van ANTLR ontzettend traag is. Hierom moest het script geconverteerd worden naar Java [Bijlage 3].
De resultaten van dit proces wordt in losse bestanden opgeslagen.

```java
public class Example {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Figuur 7: Een Java class

```
(compilationUnit
    (typeDeclaration
        (classDeclaration
        (normalClassDeclaration
            (classModifier public)
            class Example
                (classBody
                {
                    (classBodyDeclaration
                        (classMemberDeclaration
                            (methodDeclaration
                            (methodModifier public
                            (methodModifier static
                            (methodHeader
                                (result void)
                            (methodDeclarator main
```

Figuur 8: Een deel van de resulterende syntax tree

Tijdens dit proces werd er tegen een aantal problemen aangelopen. De eerste was dat niet elke code sample in de dataset een class is. Sommige hiervan zijn functies. Wanneer ANTLR een functie vindt wanneer het een class verwacht, is er een error in de console, maar geen exceptie. Wanneer er sprake is van een serieuze error, is het gebruikelijk om een exceptie te gooien. Gezien er geen sprake was van een exceptie, was de assumptie dat ANTLR doorhad dat het niet om een class maar en functie ging. Dit bleek echter niet het geval te zijn: ANTLR geeft het gewoon op en het script gaat verder met het volgende deel, dus moest de code aangepast worden om hier rekening mee te houden. De dataset bevat informatie over welk stuk code een functie of een class is, dit kan gebruikt worden voor om te bepalen hoe ANTLR met het stuk code om moet gaan.

Het tweede probleem is dat niet elke functie in de dataset daadwerkelijk een functie is; sommige zijn constructors. Dit staat echter niet in de dataset.

Om ANTLR de exceptie te laten gooien moet de standaard error handler van ANTLR overschreven worden [bijlage 16]. Op deze manier wordt er wel een exceptie gegooid wanneer er een fout gevonden wordt. Door deze exceptie op te vangen weet het programma wanneer het stuk code een constructor is.

De error die ANTLR wil afhandelen is een NoViableAltException. Deze geeft aan dat er sprake is van een significante syntax error. Alle code in de dataset is gecheckt op correcte syntax door de makers ervan. Het geval met functies en constructors is dus het enige scenario waarin deze exceptie gegooid wordt.

5.3.1.5 Gebruiksklaar maken

Het gebruiksklaar maken van de dataset houdt in om de losse bestanden die tijdens het parsen geproduceerd zijn bij elkaar te bundelen om deze makkelijker te kunnen gebruiken. Hiervoor is wederom een Python script gebruikt [Bijlage 4]. Deze verzamelt alle bestanden en bundelt deze in een zogeheten Pickle bestand. Dit is een formaat bedoeld om Python objecten als bestanden op te slaan.

5.3.1.6 Vectorizen

De dataset is nu klaar voor gebruik, maar de data die erin staat nog niet helemaal. Een model is enkel in staat om getallen te begrijpen. De syntax tree, opgeslagen als tekst, moet dus geconverteerd worden naar getallen. Hiervoor worden twee algoritmen gebruikt, de CountVectorizer [10] en de TfidfTransformer [11]. De CountVectorizer en TfidfTransformer vormen samen de TfidfVectorizer. Deze algoritmen proberen op basis van de hoeveelheid dat een bepaald “eenheid die betekenis heeft”, of token, in de tekst voorkomt. In het geval van natuurlijke tekst (zoals dit document) gaat dit om woorden. In dit geval gaat het om elementen in de syntax tree. Dit proces is geïllustreerd in [bijlage 7].

De labels hoeven niet omgezet te worden, het framework dat gebruik

5.3.2 Trainen van het model

Wanneer een programmeur een machine learning model wil maken moet hij eerst een geschikt algoritme selecteren. Dit algoritme wordt getraind met de dataset. Dit proces zorgt ervoor dat het algoritme de patronen in de dataset herkent. Het resultaat van dit proces wordt een model genoemd.

Voor het maken van de modellen is gebruik gemaakt van twee frameworks: scikit-learn en TensorFlow [deelvraag 7]. Scikit-learn wordt gebruikt voor het trainen van “traditionele” modellen. TensorFlow wordt gebruikt voor het trainen van neural networks. Het maken van een

model met scikit-learn bestaat uit het kiezen van een algoritme, dit algoritme instantiëren en deze instantie vervolgens trainen met de dataset [bijlage 5].

Na het trainen moeten de prestaties van het model beoordeeld worden. Hiervoor wordt de F1-score [15] gebruikt. De F1-score is een metriek die de hoeveelheid echt-positieven, fout-positieven en fout-negatieven tot één waarde combineert. Hoe hoger, hoe beter, met een minimum van 0 en een maximum van 1.

Om deze waarde te bepalen moet een deel van de dataset apart worden gezet als validatie set. Dit is data waar de bijbehorende labels bekend zijn, maar het model nog niet eerder heeft te zien. Door het getrainde model op deze validatie set uit te voeren en de resultaten te vergelijken met de bekende waarden, kunnen de prestaties van het model beoordeelt worden.

Voor het optimaliseren van de algoritmen is een proces genaamd “hyperparamter tuning” [bijlage 15] gebruikt. De meeste klassieke algoritmen hebben verschillende parameters die het trainingsproces beïnvloeden. Dit gaat bijvoorbeeld om hoe streng het model moet zijn bij een fout tijdens het trainen. Hyperparameter tuning is het proces om voor elke mogelijke combinatie van parameters, binnen door de programmeur bepaalde limieten, een model te trainen en hiervan de prestaties te meten, bijvoorbeeld met de F1-score. Hiermee kunnen de beste mogelijke combinatie van parameters gevonden worden en dus kan er een beter model getraind worden.

Een neural network maken met TensorFlow bestaat uit het maken van een lege instantie van een neural network, hier verschillende lagen van neuronen aan toevoegen, het selecteren van activation functions voor deze lagen, het selecteren van een loss functie [deelvraag 7], het selecteren van een optimizer functie [12] en vervolgens het model compileren en trainen. [bijlage 6].

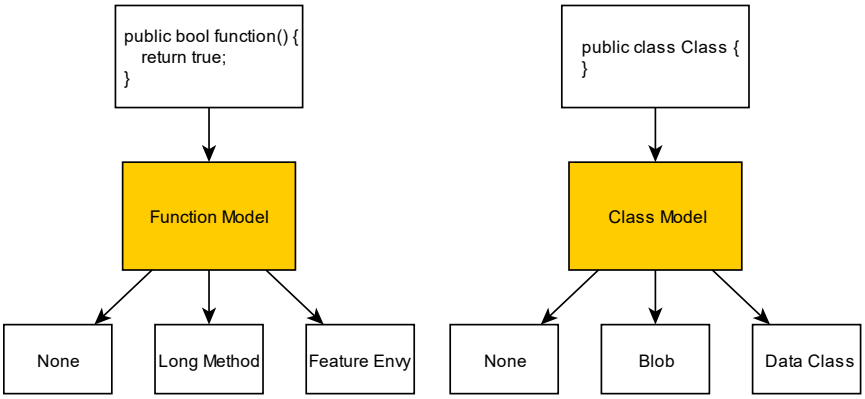
Neural networks kunnen niet automatisch geoptimaliseerd worden, de programmeur moet handmatig aanpassingen maken aan de structuur van het model op zo een optimale te vinden. Bij het implementeren van deze strategieën is er een neural network met één LSTM laag gebruikt, met hierna 3 lagen van krimpende grootte.

De traditionele modellen die getest worden zijn de Decision Tree Classifier, the Support Vector Classifier en de Naïve-Bayes Classifier, zoals besproken in [Deelvraag 7]. Naast deze 3 classifiers wordt ook de Dummy Classifier getest. De Dummy Classifier negeert de dataset en produceert willekeurige resultaten. Als een model beter presteert dan de Dummy Classifier, presteert deze beter dan kans. De prestaties van een Dummy Classifier kunnen variëren met geluk en de distributie van data.

Het doel van het model is om de aanwezigheid van problemen te detecteren in code.
De MLCQ dataset bevat hiervoor het stuk code, het probleem in deze code en de serieusheid van dit probleem. Met de beschikbare data zijn er drie strategieën bedacht om dit te doen.

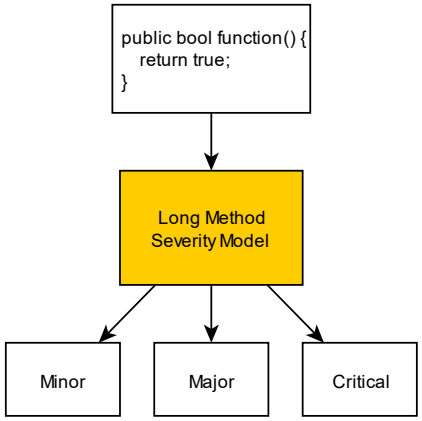
5.3.2.1 Strategie 1

Er zijn twee soorten code in de dataset, functies en classes. Voor beide soorten wordt een model gemaakt, die als resultaat heeft welke smell er gedetecteerd is. Er worden twee modellen gebruikt omdat twee van de problemen enkel in functies voor kunnen komen en twee enkel in classes.



Figuur 9: Schematische weergave van strategie 1

Om vervolgens de serieusheid te bepalen, wordt een tweede model gebruikt.



Figuur 10: Schematische weergave van strategie 1

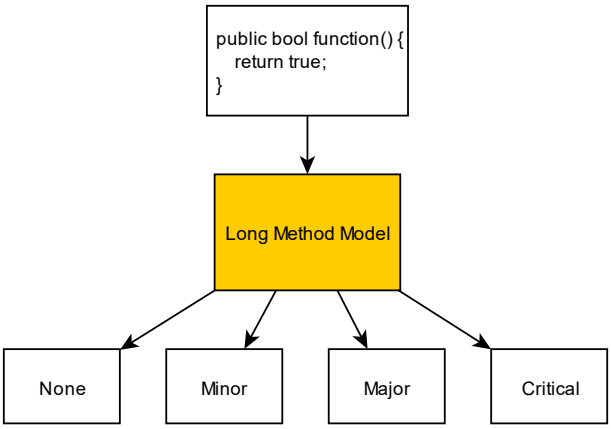
Als bepaald is dat in de het stukje code het “long method” probleem zit, wordt een ander model gebruikt om hiervoor de serieusheid hiervan te bepalen. Voor deze strategie moeten in totaal 5

modellen getraind worden. Het eerste model moet heeft 4 mogelijk resultaten, de andere 4 hebben 3 mogelijke resultaten.

De tweede deel van de strategie om de serieusheid te bepalen is niet uitgevoerd. Dit omdat de resultaten van het eerste deel van de strategie geen positieve indruk gaven.

5.3.2.2 Strategie 2

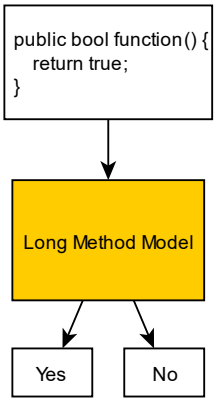
De tweede strategie bestaat uit 4 modellen. Elk model detecteert de serieusheid van één probleem op de 4 niveaus van serieusheid. Hieronder is het Long Method model afgebeeld.



Figuur 11: Schematische weergave van strategie 2

5.3.2.3 Strategie 3

De derde strategie is een vereenvoudigde versie van strategie 2. In plaats van verschillende niveau's van serieusheid te bepalen wordt enkel bepaald of er wel of niet een probleem in de code zit.



Figuur 12: Schematische weergave van strategie 3

Door serieusheid weg te laten wordt de effectieve hoeveelheid data groter. In plaats van de data te verdelen over meerdere niveau's van serieusheid, worden deze gecombineerd tot één niveau. Hierom is het de verwachting dat dit model beter zal presteren dan strategie 2.

Deze strategie is bedacht na het uitvoeren van een aantal tests met strategie 2 en hiervan de resultaten te zien. Er werd gekeken naar mogelijke aanpassingen aan de strategie om betere prestaties te leveren.

5.3.2.4 Implementatie

Voor de traditionele algoritmen zijn alle strategieën geïmplementeerd. De modellen zijn geoptimaliseerd met hyperparameter tuning [bijlage 15].

Neural networks waren echter problematisch. Deze kosten veel meer tijd, rekenkracht en RAM om te maken. Na een aantal eerste experimenten bleken de prestaties van neural networks niet significant beter dan traditionele algoritmen, ten koste van veel tijd tijdens het wachten op het trainen en constante problemen met een gebrek aan RAM. Hierom is besloten om te stoppen met het ontwikkelen van neural networks en te focussen op traditionele algoritmen.

5.3.2.5 Resultaten

De resultaten van de strategieën zijn opgenomen in de volgende tabellen, uitgedrukt in de F1-score, afgerond op twee getallen achter de komma.

Strategie 1

Model Type	Naïve-Bayes	Support Vector Classifier	Decision Tree Classifier	Dummy Classifier
Class Model	0.27	0.33	0.42	0.27
Function Model	0.21	0.31	0.46	0.26

Strategie 2

Model Type	Naïve-Bayes	Support Vector Classifier	Decision Tree Classifier	Dummy Classifier
Long Method	0.31	0.23	0.63	0.15
Blob	0.25	0.21	0.40	0.17
Feature Envy	0.12	0.23	0.31	0.18
Data Class	0.16	0.29	0.43	0.18

Strategie 3

Model Type	Naïve-Bayes	Support Vector Classifier	Decision Tree Classifier	Dummy Classifier
Long Method	0.70	0.84	0.86	0.47
Blob	0.55	0.43	0.74	0.41
Feature Envy	0.28	0.47	0.57	0.45
Data Class	0.41	0.72	0.75	0.47

Uit deze resultaten blijkt dat de derde strategie het beste presteert. Dit komt overeen met de verwachtingen dat door de effectief hogere hoeveelheid data die deze strategie levert voor betere prestaties zorgt.
Binnen de derde strategie levert de Decision Tree Classifier de beste prestaties. Deze classifier gaat gebruikt voor het analyseren van code.

De resultaten van deze classifier zijn echter niet al te goed. De prestaties zijn altijd beter dan kans, maar 0.57 tot 0.86 zijn niet hele hoge scores.

Wat betreft de andere classifiers, deze presteren ook niet goed. Opmerkelijk is de Naïve-Bayes classifier voor Feature Envy. Deze heeft een significant lage F1-score dan de Dummy Classifier. Een mogelijke verklaring hiervoor is dat het model een ander patroon in de dataset vindt dan wat het moet vinden.

De andere strategieën presteren allemaal niet goed. Gezien het grootste verschil tussen de tweede en derde strategie de effectieve grootte van de dataset per label is, kan dit verklaard worden door een gebrek aan data.

5.3.2.6 Conclusie

Op basis van deze resultaten is gekozen voor het implementeren van de derde strategie met de Decision Tree Classifier. Dit betekent dat serieusheid niet meer voorspeld wordt. Het is echter wel belangrijk dat de mogelijkheid hiertoe blijft bestaan in code, zodat deze in de toekomst toegevoegd zal kunnen worden als er meer data beschikbaar komt.

5.4 Ontwikkelen van het dashboard

Het dashboard bestaat uit drie onderdelen: de front-end, de back-end en de database. Omdat er gebruik gemaakt wordt van Entity Framework [deelvraag 6] gaat het ontwikkelen van de database gepaard met het ontwikkelen van de back-end.

De eerste stap is om het project te maken. Hiervoor is het Visual Studio template “ASP.NET Core with Angular” gebruikt, met als authenticatie type “Individual Account”. Dit template komt met SQL Server meegeleverd. Er is gekozen om Visual Studio te gebruiken omdat dit binnen OVSoftware dit vaak gebruikt wordt voor ontwikkeling met C#.

Het eerste dat gedaan is qua ontwikkeling is het maken van de het data model. Dit houdt in dat de C# objecten die corresponderen aan de database objecten gemaakt worden, waarna Entity Framework hier automatisch een database van genereerd [bijlage 9]. Op basis van deze modellen zijn ook de API controllers [bijlage 10] gemaakt. Deze controllers beheren de functionaliteit gekoppeld aan de API endpoints die gebruikt worden voor de communicatie tussen verschillende onderdelen.

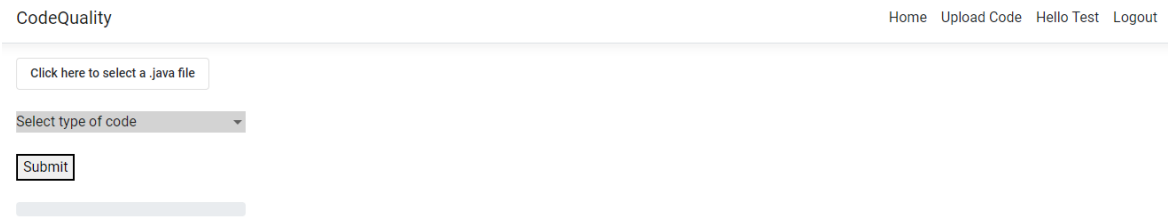
Na de basis van de back-end te maken is er begonnen met het maken van de front-end. Door tegelijk de front-end en back-end te maken kunnen deze makkelijk op elkaar afgesteld worden. De API endpoints in de back-end hoeven dan niet aangepast op te worden omdat de gekozen manier toch niet handig blijkt te zijn.

De eerste functionaliteit die in de front-end is geïmplementeerd is de lijst van issues [bijlage 11]. Vervolgens is er gewerkt aan het laten zien van de code die bij de geselecteerde issue hoort [bijlage 12], authenticatie via inloggen en als laatste het kunnen beoordelen van een issue en de lijst van ratings [bijlage 13].

Figuur 13: Hoofdpagina van de applicatie

Ook is besloten om de naam van het bestand waar het stuk code uit komt op te slaan. In het ontwerp werd hier geen rekening gehouden.

Tijdens de ontwikkeling is ook besloten om een nieuw onderdeel te maken waarin gebruikers code kunnen uploaden om te analyseren, zonder dat dit via de pipeline hoeft te gaan. De back-end ontvangt de code van de front-end, analyseert deze en slaat de resultaten op in de database.



Figuur 14: Upload Code Pagina

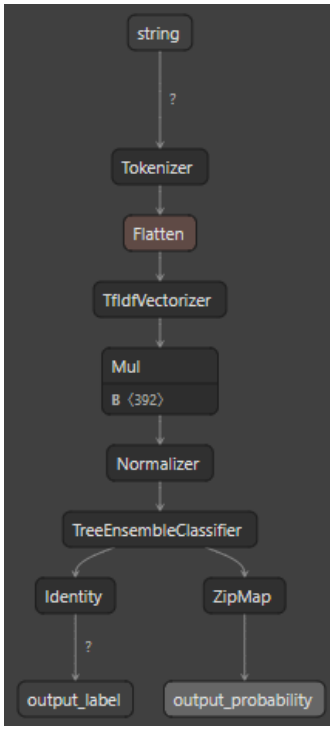
Om de front-end te testen wordt gebruik gemaakt van handmatige tests. De verschillende onderdelen van de front-end worden handmatig doorgeklik. Omdat het dashboard relatief gelimiteerd is in wat de gebruiker kan doen is dit voldoende om de functionaliteit van het

dashboard te testen. De back-end wordt getest met unit-tests waar dit kan. Anders worden de endpoints aangeroepen vanuit Postman om de functionaliteit te testen.

5.5 Implementeren van het model

Zoals genoemd in 8.3.1.6 moet de tekst waar de syntax tree uit bestaat eerst getransformeerd worden met een TfidfVectorizer. Zodra dit gedaan is, kan de dataset gebruikt worden om het model te trainen. Om dit proces te vereenvoudigen wordt een Pipeline gebruikt. Dit is een feature in scikit-learn waarmee verschillende stappen in het proces automatisch uitgevoerd kunnen worden. Zo kan de output van stap 1, het vectorizen, gebruikt worden in stap 2, het trainen. Zodra het model getraind is moet deze geëxporteerd worden zodat deze gebruikt kan worden in de back-end voor het analyseren van code. Het model is echter gemaakt in een Python omgeving en de back-end is een C# omgeving. Om deze reden wordt het model geëxporteerd in het ONNX [13] formaat. Het Open Neural Network Exchange formaat is een open-source formaat met als doel om machine learning platform onafhankelijk te maken. Zo kan een model gemaakt in Python gebruikt worden in C#. Er is gekozen om ONNX te gebruiken na aanleiding van een presentatie gegeven door een aantal collega's met als doel om kennis te delen binnen het bedrijf. ONNX

Ook is het mogelijk om de vectorizer in het ONNX model te integreren. Op deze manier kan de back-end een syntax tree aan het model geven. Het model vectorized en analyseert deze tree en geeft een resultaat terug. Het proces om het model te trainen en exporteren is geïllustreerd in bijlage 8.

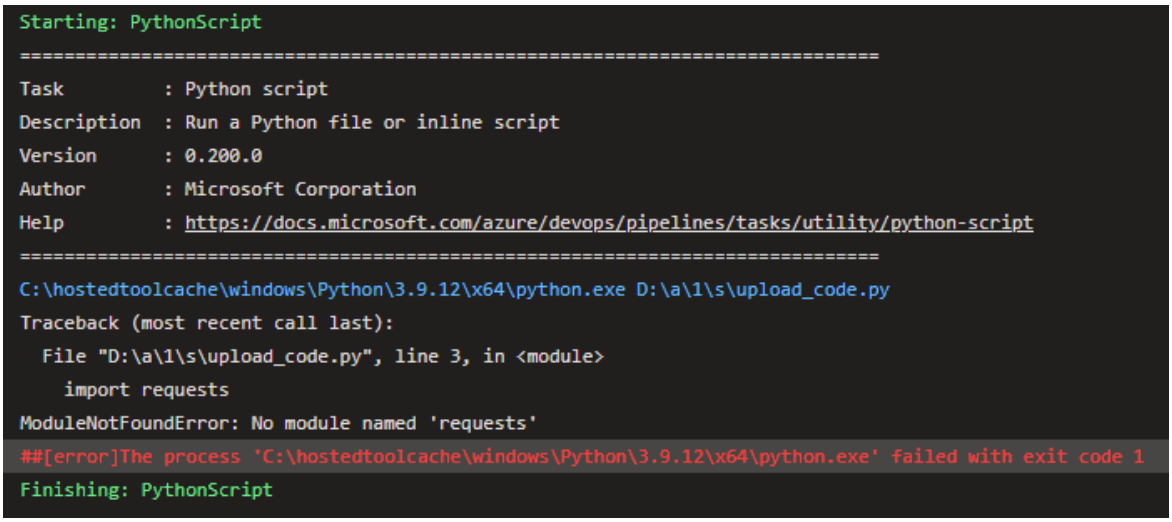


Figuur 15: Schematische weergave van het ONNX model.

5.6 Ontwikkelen van de pipeline

Wanneer code aan het versiebeheer systeem wordt toegevoegd kan er een pipeline uitgevoerd worden. Deze pipeline voert in een gevirtualiseerde omgeving bepaalde taken uit, zoals het uitvoeren van alle tests.
Voor dit project is er een taak gemaakt die een Python script uitvoert [bijlage 14]. Beschikbaar in deze omgeving zijn alle bestanden die gepusht zijn naar het systeem en alle bestanden die er al waren. Het script zoekt door al deze bestanden naar Java bestanden en verstuurt deze naar de back-end waar deze geanalyseerd worden.

Tijdens het ontwikkelen van de pipeline is er tegen een probleem aangelopen dat de package die Python nodig heeft voor het versturen van HTTP requests niet beschikbaar is, omdat de 'requests' package niet beschikbaar is in de executie omgeving. Hierdoor kon het script niet in de pipeline zelf uitgevoerd worden. Het script handmatig uitvoeren werkt echter goed.



Figuur 16: Uitvoeren in de pipeline

De pipeline wordt getest door het uit te voeren in de repository van dit project en het lokaal uit te voeren.

OVSoftware

6 Evaluatie

6.1 Resultaten

Het eindresultaat bestaat uit een model, een dashboard met database.

Zoals beschreven in 8.3.2.5, het gemaakte model is in staat om te bepalen of er een probleem. De nauwkeurigheid is niet heel hoog, maar wel beter dan kans. De strategie die gebruikt is om deze nauwkeurigheid te bereiken betekent dat het model niet in staat is om de serieusheid van het gedetecteerde probleem te bepalen. Het model is geëxporteerd als een ONNX model en wordt uitgevoerd in de back-end.

Het dashboard bestaat uit een database, een back-end en een front-end.
De front-end is gemaakt met Angular, de back-end met ASP.NET en de database met Entity Framework en SQL Server.

De database is in staat om de geproduceerde resultaten van het model op te slaan. De back-end communiceert met de database via Entity Framework. Andere onderdelen communiceren met de database via de API in de back-end, zolang ze de juiste inloggegevens hebben.

De front-end kan resultaten van het model van de back-end opvragen. De gebruiker kan deze resultaten en bijbehorende code bekijken. Ook is de gebruiker in staat om deze resultaten te beoordelen op geschiktheid voor toevoeging aan de dataset.

De pipeline bestaat uit een Python script dat uitgevoerd wordt in een Azure Pipeline. Dit script zoekt alle .java bestanden in de repository en verstuurd deze naar de back-end, die ze analyseert en de resultaten hiervan opslaat in de database.

Met dit resultaat zijn alle eisen zoals origineel opgesteld bereikt, behalve het functioneren van de pipeline.

6.2 Aanbevelingen

Uit de resultaten van 8.3.2.5 wordt duidelijk dat de modellen niet al te goed presteren. Gezien meerdere modellen zijn getest en er gebruik is gemaakt van hyperparameter tuning is de waarschijnlijkste reden voor deze gebrekkige prestaties een dataset van onvoldoende grootte.

Het pipeline script kan niet uitgevoerd worden in de pipeline, omdat hier niet de juiste Python packages beschikbaar zijn.

Het beoordelingssysteem is momenteel afhankelijk van het optellen van alle beoordelingen. Een positieve beoordeling is 1 waard, een negatieve beoordeling is -1 waard. Wanneer de totale beoordelingen boven 5 komen of onder -5 wordt deze of toegevoegd aan de dataset, of inactief gemaakt. Dit betekent dat als de beoordelaars het niet eens kunnen worden over het resultaat deze nooit verder verwerkt wordt. Hier moet misschien een betere oplossing voor bedacht kunnen worden.

6.3 Evaluatie van het proces

In de eerste paar weken van het project tijdens het onderzoek werden er elke week twee voortgangsgesprekken gehouden met de bedrijfsbegeleiders. Na het beginnen met actieve ontwikkelingen werd er begonnen met dagelijkse sprintmeetings om de voortgang te bespreken.

Dit systeem heeft goed gewerkt. Tijdens het de onderzoeksfase waren de twee gesprekken voldoende. Het gebruik van dagelijkse meetings zou zijn neergekomen op “ik ga verder met wat ik gisteren deed”. Na de onderzoeksfase tijdens het ontwikkelen waren er op een dag genoeg ontwikkelingen dat dagelijkse gesprekken nuttig waren.

De planning die gemaakt is voor het Plan van Aanpak is niet helemaal gevolgd. Dit is vooral te wijten aan een periode van twee weken, in de laatste week van het onderzoek en de eerste week van het ontwikkelen waarin ik ziek was en dus niet in staat om op volle capaciteit te werken. Ook is er besloten om eerder te beginnen met het ontwikkelen van de pipeline omdat er voldoende werk was gedaan aan het dashboard om daarmee te beginnen.

OVSoftware

7 Bronnen

[1] L. Madeyski en T. Lewowski, „MLCQ: Industry-relevant code smell data set,” [Online]. Available: https://doi.org/10.5281/zenodo.3666840. [Geopend 1 maart 2022].

[2] T. Sharma, M. Kechagia, S. Georgiou, R. Tiwari en F. Sarro, „A Survey on Machine Learning Techniques for Source Code Analysis,” 18 oktober 2021. [Online]. Available: https://arxiv.org/abs/2110.09610. [Geopend 15 februari 2022].

[3] Software Improvement Group, „better Code Hub,” [Online]. Available: https://bettercodehub.com/. [Geopend 7 juni 2022].

[4] J. Visser, Building Maintainable Software, Sebastopol, CA, USA: O'Reilley, 2016.

[5] 12 December 2014. [Online]. Available: http://wiki.c2.com/?CodeSmell. [Geopend 11 februari 2022].

[6] E. Gamma, R. Helm, R. Johnson en J. Vlissides, Design Patterns, Addison-Wesley, 1995.

[7] „What is Azure Pipelines?,” Microsoft, 11 februari 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops#:~:text=Azure%20Pipelines%20automatically%20builds%20and,ship%20it%20to%20any%20target.. [Geopend 21 februari 2022].

[8] Microsoft, 11 februari 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/command-line?view=azure-devops&tabs=yaml. [Geopend 21 februari 2022].

[9] T. Parr, „ANTLR,” ANTLR, [Online]. Available: https://www.antlr.org/. [Geopend 20 april 2022].

[10] scikit-learn, „CountVectorizer,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer. [Geopend 2022 mei 18].

[11] scikit-learn, „TfidfTransformer,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html#sklearn.feature_extraction.text.TfidfTransformer. [Geopend 18 mei 2022].

[12] A. Gupta, „A Comprehensive Guide on Deep Learning Optimizers,” 7 october 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy.. [Geopend 17 mei 2022].

[13] ONNX, „Open Neural Network Exchange,” [Online]. Available: https://onnx.ai/. [Geopend 18 mei 2022].

[14] scikit-learn, „f1_score,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. [Geopend 12 april 2022].

[15] T. Sharma, „QScored: A Large Dataset of Code Smells and Quality Metrics,” 2021. [Online]. Available: https://www.tusharma.in/preprints/qscored_data_preprint.pdf. [Geopend 10 februari 2022].

[16] scikit-learn, „scikit-learn: machine learning in Python,” [Online]. Available: https://scikit-learn.org/stable/index.html. [Geopend 17 mei 2022].

[17] TensorFlow, „TensorFlow,” [Online]. Available: https://www.tensorflow.org/. [Geopend 17 mei 2022].

8 Bijlagen

8.1 Bijlage 1: Download script

Dit script download alle bestanden waarnaar verwezen wordt in de dataset. De gedownloade code wordt opgeslagen in .java bestanden.

```python
import pandas as pd
import os
import requests


def parse_url(url):
    url = url.split("/")
    url[2] = "raw.githubusercontent.com"
    url.pop(5)  # Remove "blob"
    url.pop()  # remove #L...
    url = '/'.join(url)  # merge back
    return url


def download_page(row):
    url_path = 'download/{}.java'.format(row['id'])
    if os.path.exists(url_path):  # File already exists, skipping
        return
    link = parse_url(row['link'])
    page = requests.get(link)  # Get page from internet
    if page.text == '404: Not Found':
        return
    with open(url_path, 'w+', encoding='utf-8') as file:  # Create or overwrite
        print(url_path)
        file.write(page.text)


def main():
    data = pd.read_excel('mlcq/samples.xlsx')
    data.apply(download_page, axis=1)  # Download


if __name__ == "__main__":
    main()
```

Figuur 17: download_dataset.py

8.2 Bijlage 2: Krimp script

De code gedownload in bijlage 1 is het volledige bestand, inclusief niet relevante regels. Dit script selecteert alleen de relevante regels. Deze gekrompen bestanden worden opgeslagen in .java bestanden.

```
import glob
import os

import pandas as pd

data = pd.read_excel('mlcq/samples.xlsx')


def shrink(filename):
    sample_id = os.path.basename(filename)[:-5]  # Get the id from the file name
    row = data.loc[data['id'] == int(sample_id)]  # Get the dataset row belonging to this file
    if os.path.exists('small_dataset/{}.java'.format(sample_id)):  # Already shrunk
        return
    with open(filename, 'r', encoding='utf-8') as file:  # Read the file and shrink the code
        content = file.readlines()
        shrunk_code = content[int(row['start_line']) - 1:int(row['end_line'])]
    with open('small_dataset/{}.java'.format(sample_id), 'w+', encoding='utf-8') as file:
        print(sample_id)
        for line in shrunk_code:
            file.write(str(line))  # Save the shrunk code


def main():
    files = glob.glob(os.path.join('download/*.java'))
    for file in files:
        shrink(file)


if __name__ == "__main__":
    main()
```

Figuur 18: shrink_dataset.py

8.3 Bijlage 3: Syntax tree script

Dit script converteert de dataset zoals deze opgeslagen is in een .xlsx bestand naar .tree bestanden. Deze bestanden bevat de syntax tree als een string.

```
public void run() throws IOException {
    File file = new File("samples.xlsx");
    FileInputStream fis = new FileInputStream(file);
    XSSFWorkbook book = new XSSFWorkbook(fis);
    XSSFSheet sheet = book.getSheetAt(0);
    for (Row row : sheet) {
        makeSample(row);
        System.out.println();
    }
}
```

Figuur 19: Inlezen van het .xlsx bestand en loopen door alle rijen

```
DataFormatter formatter = new DataFormatter(); //Don't need to recreate this in every loop
public void makeSample(Row row) throws IOException {
    String id = formatter.formatCellValue(row.getCell(0));
    String treePath = makeTreePath(id); //Path to where the tree file will be
    String filename = makeFileName(id); //Path to where the code currently is

    //Check if we already did this one
    if (checkFileExists(treePath)) {
        System.out.println("Already did file: " + id);
        return;
    }
    System.out.println(id);

    //Create the parser
    Java8Parser parser = null;
    try {
        parser = makeParser(filename);
    } catch (IOException e) {
        System.out.println(id + " doesn't exist.");
        return;
    }

    //Create the tree
    String tree = null;
    if (formatter.formatCellValue(row.getCell(6)).equals("class")) { //Class
        try {
            tree = parseClass(parser);
        } catch (MyNoViableException mnve) {
            significantError(id);
        }
    } else { //Function or constructor
        try {
            tree = parseFunction(parser);
        } catch (MyNoViableException mnve) {
            significantError(id);
        }
    }
    //Save it to .tree file
    saveToFile(tree, treePath);
}
```

Figuur 20: De loop functie

```
public Java8Parser makeParser(String filename) throws IOException {
    Java8Parser parser = new Java8Parser(new CommonTokenStream(new Java8Lexer(CharStreams.fromFileName(filename))));
    parser.removeErrorListeners();
    parser.addErrorListener(new MyErrorListener());
    return parser;
}
```

Figuur 21: Maken van een parser

```
public String parseClass(Java8Parser parser) throws MyNoViableException {
    return parser.compilationUnit().toStringTree();
}
```

Figuur 22: Parsen van een class

```
public String parseFunction(Java8Parser parser) throws MyNoViableException { //Or constructor
    try { // Not function but a constructor block
        return parser.methodDeclaration().toStringTree();
    } catch (MyNoViableException e) {
        System.out.println("Continuing as constructor");
        //Catch block for unknown error. Not a constructor either
        return parser.constructorDeclaration().toStringTree();
    }
}
```

Figuur 23: Parsen van een functie

8.4 Bijlage 4: Gebruiksklaar maken script

Converteert alle .tree bestanden naar een Pandas DataFrame, opgeslagen in dataset_with_nones.pkl

```
import glob
import os
import sys

import pandas
import pandas as pd

data = pd.read_excel('mlcq/samples.xlsx')


def make_row(file, file_name):
    file_name = file_name.split('.')[0]  # Remove file extension
    row = data[data['id'] == int(file_name)]
    return [file_name, file.readlines()[0], row.iloc[0]['smell'], row.iloc[0]['severity'], row.iloc[0]['type']]


def main():
    files = glob.glob(os.path.join('trees/*'))
    rows = []
    for file in files:
        with open(file, 'r', encoding='utf-8') as opened_file:
            rows.append(make_row(opened_file, os.path.basename(file)))
    dataframe = pd.DataFrame(rows, columns=['id', 'code', 'smell', 'severity', 'type'])
    dataframe.to_pickle('dataset_with_nones.pkl')
    print(dataframe)


if __name__ == "__main__":
    main()
```

Figuur 24: shrink_dataset.py

8.5 Bijlage 5: Traditioneel model

Om een traditioneel model te maken, moet er een instantie van een algoritme gemaakt worden. Hier kunnen parameters die het trainingsproces beïnvloeden aan meegegeven worden.
Op deze instantie wordt de fit methode aangeroepen, met de X en Y van de dataset als parameters.

```
classifier = DecisionTreeClassifier(**{'criterion': 'gini', 'max_depth': 9, 'max_leaf_nodes': None})
classifier.fit(X, y)
```

OVSoftware

8.6 Bijlage 6: Neural Network

Om een neural network te trainen, moet er een instantie gemaakt worden van een leeg netwerk. Hier worden lagen aan toegevoegd, met de soort laag, hoeveelheid neuronen en activatie methode als parameter.

Deze instantie moet gecompileerd worden met de loss functie, de optimizer functie en metriek voor het meten van de prestaties.

De gecompileerde instantie wordt gefit met de X en Y van de dataset als parameters. Ook moet de hoeveelheid epochs, batch size en validation split als parameters. Dit zijn de tijd dat het model getraind wordt, de hoeveelheid data die tegelijk gebruikt wordt en de hoeveelheid data die apart gehouden wordt voor validatie van het model.

```
classifier = Sequential()
classifier.add(Dense(input_formaat))
classifier.add(Dense(30, activation="relu"))
classifier.add(Dense(20, activation="relu"))
classifier.add(Dense(10, activation="relu"))
classifier.add(Dense(output_formaat, activation="softmax"))

classifier.compile(
    loss=CategoricalCrossentropy(from_logits=False),
    optimizer='adam',
    metrics=["accuracy"]
)

classifier.fit(X, y, epochs=30, batch_size=32, validation_split=0.2)
```

8.7 Bijlage 7: TfidfVectorizer

De TfidfVectorizer wordt geïnstantieerd met een vocabulary parameter. Dit is de lijst van alle termen die gevectorized moeten worden.
De vectorizer wordt gefit op de X van de dataset en wordt vervolgens getransformeerd.

```
vectorizer = TfidfVectorizer(vocabulary = vocabulary)
vectorizer.fit(X)
getransformeerde_x = vectorizer.transform(X)
```

8.8 Bijlage 8: Trainen en exporteren van het model voor het detecteren van long method

Voor het trainen van dit model wordt een pipeline gebruikt, met daarin de vectorizer en classifier. Aan de classifier worden ook de parameters meegegeven die gevonden zijn met hyperparameter tuning.
Het doel van de pipeline is om het trainen van de vectorizer en classifier te combineren tot 1 actie. Dit werkt ook voor het uitvoeren hiervan.

Zodra de pipeline getraind is, wordt deze geconverteerd naar een ONNX model en opgeslagen in een bestand.

```
X, y = data_per_smell['long_method']
initial_type = [('string', StringTensorType([1, 1]))]

pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(vocabulary = vocabulary, lowercase = False)),
    ("clf", DTC(**long_method))
])

pipeline.fit(X, y)

onx = convert_sklearn(pipeline, initial_types=initial_type, dtype=np.float64)

with open("models/long_method.onnx", "wb+") as f:
    f.write(onx.SerializeToString())
```

8.9 Bijlage 9: Data model

Een model wordt gedefinieerd als een class met een aantal variabelen. Ook heeft Entity Framework een lege constructor nodig.

8.9.1 Code Segment

```
public class CodeSegment
    {
        public int Id { get; set; }
        public string Type { get; set; }
        public string Code { get; set; }

        //public string status { get; set; }

        public CodeSegment(string Code, string Type)
        {
            this.Code = Code;
            this.Type = Type;
        }

        public CodeSegment() {}
    }
```

8.9.2 Issue

```
public class Issue
    {
        public int? Id { get; set; }
        public string Name { get; set; }
        public string Severity { get; set; }
        public double NameConfidence { get; set; }
        public double SeverityConfidence { get; set; }
        public bool isActive {get; set;}

        public int? CodeSegmentId { get; set; }

        [ForeignKey("CodeSegmentId")]
        public virtual CodeSegment CodeSegment { get; set; }

        public virtual List<Rating> Ratings { get; set; }

        public Issue(string name, string severity, double nameConfidence,
double severityConfidence, CodeSegment segment)
        {
            Name = name;
            Severity = severity;
            NameConfidence = nameConfidence;
            SeverityConfidence = severityConfidence;
            CodeSegment = segment;
            isActive = true;

            Ratings = new List<Rating>();
        }

        public Issue() {
            isActive = true;
            Ratings = new List<Rating>();
        }
    }
```

8.9.3 Rating

```
namespace CodeQuality.Models
{
    public class Rating
    {

        public int? Id { get; set; }
        public string Username { set; get; }
        public int Value { get; set; }

        public Rating(string User, int Value)
        {
            this.Username = User;
            this.Value = Value;
        }

        public Rating()
        {

        }
    }
}
```

8.10 Bijlage 10: Voorbeeld van een API Controller

In de constructor van een controller wordt eerst een service gemaakt. Een service wordt gebruikt voor communicatie met de database.
Verder bevat deze controller twee endpoints: GetIssues() en PostIssue().

```
[Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class IssuesController : ControllerBase
    {
        private readonly IssueService _issueService;

        public IssuesController(ApplicationDbContext _context)
        {
            _issueService = new IssueService(_context);
        }

        // GET: api/Issues
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Issue>>> GetIssues()
        {
            return await _issueService.GetIssues();
        }

        // POST: api/Issues
        [HttpPost]
        public async Task<ActionResult<Issue>> PostIssue(Issue issue)
        {
            await _issueService.AddIssue(issue);
            await _issueService.Save();

            return CreatedAtAction("GetIssue", new { id = issue.Id }, issue);
        }
    }
```

8.11 Bijlage 11: Lijst van issues

8.11.1 HTML

Het HTML onderdeel van de lijst bestaat uit een div met hierin een Angular directive ngFor.
ngFor maakt een loop, voor elk issue in de lijst issues wordt er een div gemaakt.
In de div zit het element issue. De waarden hiervan worden opgehaald uit de issue geïnstantieerd in de ngFor directive. De waarden komen dus uit de lijst

```
<div *ngFor="let issue of issues"><issue [issue]="issue" (click)="onClick(issue)"></issue></div>
```

8.11.2 TypeScript

Het TypeScript onderdeel definieert het gedrag van de component. Wanneer er op dit component geklikt wordt stuurt deze een bericht naar de Active Issue Service, die andere onderdelen van het programma vertelt wat de actieve issue is.

```
@Component({
  selector: 'issue-list',
  templateUrl: './issue-list.component.html',
  styleUrls: ['./issue-list.component.css']
})
export class IssueListComponent implements OnInit {
  issues: Issue[];

  //Issue service is for getting the issues from the API, activeService makes the currently selected issue available to others.
  constructor(private issueService: IssueService, private activeService: ActiveIssueService, private authorizeService: AuthorizeService) { }

  ngOnInit(): void {
    this.issueService.getIssues().subscribe((issues) => {
      this.issues = issues;
    })
  }

  public onClick(issue: Issue) {
    this.activeService.setActive(issue.id);
  }

}
```

8.12 Bijlage 12: Code Display Component

8.12.1 HTML

Het HTML onderdeel van dit component is enkel een Angular statement om de variabele “code”, die in het TypeScript onderdeel gedefinieerd staat te laten zien.
{{code}}

8.12.2 TypeScript

Het TypeScript onderdeel krijgt de actieve issue uit de Active Issue service, die zijn data krijgt uit het component beschreven in [bijlage 11]. Op basis van deze Issue wordt aan de Code Segment Service gevraagd om de code die bij deze issue hoort te sturen. Deze wordt aan de code variabele toegewezen, die wordt laten zien in het HTML onderdeel.

```
@Component({
  selector: 'code-display',
  templateUrl: './code-display.component.html',
  styleUrls: ['./code-display.component.css']
})
export class CodeDisplayComponent implements OnInit {

  code: string = "Please select an issue.";

  constructor(private activeService: ActiveIssueService, private codeSegmentService: CodeSegmentService) { }

  ngOnInit(): void {
    //We first acquire the active issue from the activeSerivce
    //this value is then used to query for the codeSegment in the front end

    this.activeService.getActive().pipe(concatMap(active => {
      return this.codeSegmentService.getSegmentByIssueId(active);
    })).subscribe(segment => {
      this.code = segment.code;
    })
  }
}
```

8.13 Bijlage 13: Lijst van beoordelingen

8.13.1 HTML

De lijst van ratings bestaat uit een het component dat de knoppen voor het beoordelen laat zien, en een lijst van eerder gemaakte beoordelingen.

```
<upvote-downvote *ngIf="ratings"></upvote-downvote>

<div *ngFor="let rating of ratings"><rating [rating]="rating"></rating></div>
```

8.13.2 TypeScript

Het TypeScript onderdeel vraagt aan de Active Issue Service wat de actieve issue is en vervolgens aan de Rating Service om alle ratings die bij deze issue horen te sturen.

```
@Component({
  selector: 'rating-list',
  templateUrl: './rating-list.component.html',
  styleUrls: ['./rating-list.component.css']
})
export class RatingListComponent implements OnInit {

  constructor(private activeIssue: ActiveIssueService, private ratingService: RatingService) { }
  ratings: Rating[];


  getIssues() {
    this.activeIssue.getActive().pipe(
      concatMap(active => {
        return this.ratingService.getRatingsForIssue(active)
      })
    ).subscribe(ratings => {
      this.ratings = ratings;
    })
  }

  ngOnInit(): void {
    this.getIssues();
  }

}
```

OVSoftware

8.14 Bijlage 14: Pipeline script

Dit script wordt uitgevoerd in de pipeline. Het eerste wat moet gebeuren is een access token krijgen tot de API. Dit

```
import os
import glob
import requests
import json
import glob


def get_token():
    data = {'client_id': 'pipeline', 'grant_type': 'client_credentials',
            'client_secret': 'secret2', 'scope': 'CodeQualityAPI'}
    response = requests.post('https://localhost:44440/connect/token', data=data, verify=False)
    return response.json()['access_token']


if __name__ == "__main__":
    token = get_token()

    headers = {'Authorization': 'Bearer ' + token}
    for file in glob.glob("**/*.java")
        form_data = {'type': 'file'}
        files = {'codefile': open(file, 'rb')}
        response = requests.post('https://localhost:7123/api/CodeUpload', data=form_data,
                                 files=files, headers=headers, verify=False)

        print(response.status_code)
```

8.15 Bijlage 15: Hyperparameter tuning

Hier is zichtbaar het zoeken naar optimale hyperparameters voor een implementatie van Strategie 2 [8.3.2.2]. Eerst wordt een dictionary gemaakt om alle resultaten in op te slaan. De 2 loops zorgen ervoor dat alle combinaties van smells en classifiers uitgetest worden. In de binnenste loop wordt GridSearchCV aangeroepen. GridSearchCV test alle mogelijke combinaties van parameters, binnen limitieten gedefinieerd in de variabele “grid”, en beoordeelt de prestaties. Deze worden aan de dictionary van resultaten toegevoegd waar ze later bekeken kunnen worden.

```
results = dict()
results['feature envy'] = []
results['blob'] = []
results['data class'] = []
results['long method'] = []

for smell, X, y in data_per_smell:
    for clf_name, clf, grid in clfs:
        print(clf_name)

        gscv = GridSearchCV(clf, grid, cv=3, scoring='f1_macro', verbose=0, n_jobs=12)
        gscv.fit(vectorizer.fit_transform(X).toarray(), y)

        results[smell].append((smell, clf_name, gscv.best_score_, gscv.best_params_))
        print('done', clf_name)
    print('done', smell)
```

9 Bijlage 16: ANTLR Error handler

Wanneer ANTLR een probleem tegenkomt, wordt de syntaxError functie aangeroepen. Het soort error wordt bevestigt. Als er sprake is van een NoViableAltException, wat betekent dat het niet om een functie maar een constructor gaat, wordt MyNoViableException gegooid, die niet automatisch afgevangen wordt door ANTLR. Deze exception kan dan afgevangen worden en de juiste methoden voor het verwerken van constructors kunnen aangeroepen worden.

```
public class MyErrorListener extends BaseErrorListener {

    public void syntaxError(Recognizer<?, ?> recognizer, Object offendingSymbol,
    int line, int charPositionInLine, String msg, RecognitionException e) throws MyNoViableException{
        System.out.print("syntaxError: ");
        if (e instanceof NoViableAltException) {
            System.out.println("Not a function, line: " + line + " col: " + charPositionInLine + ": " + msg);
            throw new MyNoViableException();
        }
        if (e instanceof InputMismatchException) {
            System.out.println("Input mismatch, line: " + line + " col: " + charPositionInLine + ": " + msg);
        }
    }
}
```