AFSTUDEER VERSLAG

Loki 1.0

Roy Klaassen

12-6-2020

1.0

1 Versiebeheer

VERSIE	BESCHRIJVING	PUBLICATIE DATUM
0.1	Concept verslag met alleen de probleemstelling en een afgerond vooronderzoek.	6-4-2020
0.2	Concept verslag	25-5-2020
0.3	Concept verslag met verwerkte feedback van bedrijfsbegeleider en team	1-6-2020
1.0	Definitieve versie met toevoegingen van project en verwerkte feedback van Afstudeerbegeleider	15-6-2020

2 Voorwoord

Dit document beschrijft het afstudeerverslag van Roy Klaassen. Het verslag is geschreven in verband met de afstudeerstage bij het Kadaster. Deze afstudeer stage dient als afsluiting van de opleiding HBO-ICT Software Engineering op het Saxion in Deventer. De stage is gelopen in de periode van 10 februari 2020 tot 26 juni 2020.

Tijdens de afstudeerstage heb ik onderzoek gedaan naar wat op dit moment de beste mogelijkheden zijn voor het realiseren van een chatbot. De uitkomsten van dit onderzoek heb ik toegepast in het maken van de chatbot Loki 1.0. Deze chatbot dient als eerste testbare versie voor de eindgebruiker.

De afstudeerstage heb ik gedaan onder begeleiding van Erwin Folmer van het Kadaster en Tristan Pothoven namens Saxion. Ik wil hen bedank voor de goede begeleiding en paraatheid tijdens mijn stage.

Roy Klaassen,

12-6-2020

3 Samenvatting

Het kadaster beschikt over veel data over alles wat met de openbare ruimte in Nederland te maken heeft. Deze data wordt op maar weinig manier ontsloten naar de eindgebruiker. Een van deze ontsluitingen, die het kadaster heeft geprobeerd in te zetten de afgelopen jaren, is een chatbot met de naam Loki. Deze chatbot was een experiment waarmee werd bewezen dat het mogelijk is om de data van het Kadaster met natuurlijke taal te kunnen bevragen.

3.1 Probleem

Omdat de chatbot slechts een experiment was zijn er verschillende “shortcuts” genomen tijdens de ontwikkeling. Hiermee is het concept bewezen maar betekende dit wel dat het niet bruikbaar was voor de eindgebruiker. Dit is wel de vervolg stap die het Kadaster graag met Loki wil zetten. Daarom was de opdracht om Loki door te ontwikkelen tot een testbare versie voor de eindgebruiker ofwel: Loki 1.0. Hierbij was het een vereiste dat er een nieuw front-end gerealiseerd zou worden

3.2 Onderzoek

Om tot Loki 1.0 te kunnen komen is eerst een onderzoek gedaan waarin de volgende hoofdvraag werd beantwoord:

- Wat is er nodig om Loki beschikbaar te stellen voor eindgebruikers?

Uit dit onderzoek is gebleken dat, Rasa; het chatbot framework waar de experimentele versie van Loki gebruikt van maakte, nog steeds een goede optie was om te blijven gebruiken. De reden hiervoor zijn:

- Er waren geen grote verschillen te zien in performance tussen Rasa en andere frameworks zoals Dialogflow van Google of Watson van IBM.
- Rasa bood meer mogelijkheden voor het configureren van de chatbot en het doen van toevoegingen zoals een kaartweergave.

3.3 Realisatie

Met de kennis van het onderzoek is begonnen aan de realisatie. Hierbij werd eerst een ontwerp gemaakt dat weergaf hoe Loki 1.0 eruit zou zien en welke functionaliteiten de chatbot zou hebben. Tijdens deze realisatie werd al snel duidelijk dat het veel tijd en werk zou kosten om van de experimentele versie van Loki een robuuste testbare versie te maken voor de eindgebruiker. Dit had de volgende redenen:

- Matige kwaliteit van code
- Keuzes die niet in lijn lagen met wat Rasa beschreef
- Geen documentatie van de code
- Responses speciaal voor Mendix
- Niet gebruikersvriendelijk

Daarom is besloten om net als met de front-end de backend ook opnieuw te maken. Dit betekende dat niet alle functionaliteiten konden worden geïmplementeerd maar zorgde wel voor een betere basis voor de testbare versie waar in de toekomst mogelijk op verder gebouwd kan worden.

4 Inhoudsopgave

1 Versiebeheer 1

2 Voorwoord 1

3 Samenvatting 2

3.1 Probleem 2

3.2 Onderzoek 2

3.3 Realisatie 2

4 Inhoudsopgave 3

5 Achtergrond 5

5.1 Knowledge graph 6

6 Probleemstelling 7

7 Vooronderzoek 9

7.1 Slimme chatbots 9

7.2 Chatbot implementaties 11

7.3 Kaartvisualisaties 13

7.4 Kadasterdata 15

7.5 Externe data 16

7.6 Conclusie 17

8 Aanpak 18

8.1 Tijdlijn 18

8.2 Tooling 19

8.3 Methoden 20

9 Functioneel ontwerp 21

9.1 Doel 21

9.2 Requirements 22

9.3 Chatfunctionaliteit 23

9.4 Chatflow 23

9.5 Designs 24

9.6 Overige functionaliteit 24

10 Implementatie 26

10.1 Repository Indeling 26

10.2 Front-end 27

10.3 Rasa backend 31

11 Conclusie 41

12 Aanbevelingen 42

13 Reflectie 43
13.1 STARR-methode reflecties 43
14 Bronnen 45

5 Achtergrond

De afstudeerstage wordt gelopen bij het Kadaster. Deze organisatie wort een zelfstandig bestuursorgaan genoemd, wat betekent dat ze overheidstaken uitvoeren maar niet direct onder het gezag van een ministerie vallen.

De wettelijke taak van het Kadaster is: "*Het houden van openbare registers en het bijwerken van de Basisregistratie Kadaster en Basisregistratie Topografie"*[26] Verder onderhoud het kadaster het Rijksdriehoekstelsel, dat is het coördinatenstelsel van Nederland.

In deze openbare registers staan gegevens van woningen, schepen, luchtvaartuigen, percelen, wegen en netwerken van kabels en leidingen onder de grond. Verder beheert het kadaster ook voorzieningen van andere organisaties. "*Zo beheren wij onder andere de WOZ Landelijke Voorziening en de Basisregistratie Adressen en gebouwen (BAG): alle adressen en gebouwen in Nederland, zoals bouwjaar, oppervlakte, gebruiksdoel en locatie op de kaart.*"[26]

Ook is het kadaster internationaal actief voor het adviseren van buitenlandse overheden bij de inrichting van landregistraties en geografische systemen.[26]

Het Kadaster heeft verder meerdere vestigingen op de volgende locaties:

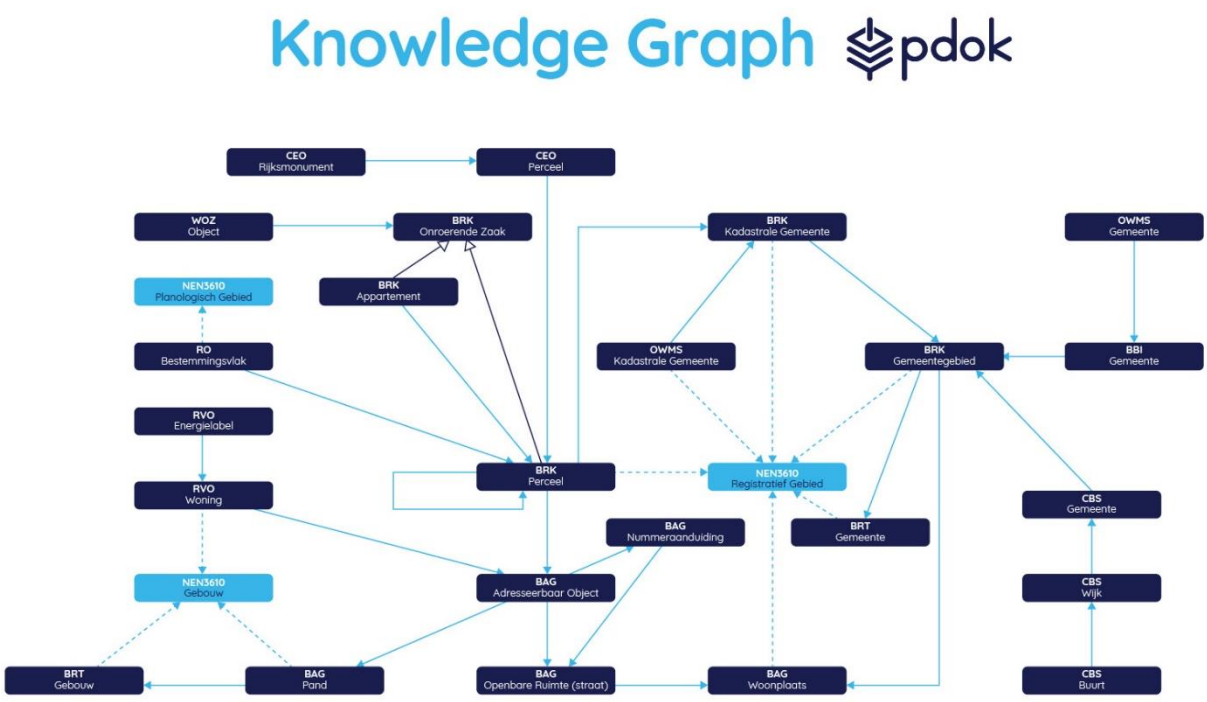
- Apeldoorn (Het hoofdkantoor, De Grift)
- Apeldoorn(De brug)
- Amsterdam
- Arnhem
- Eindhoven
- Groningen
- Rotterdam
- Zwolle

Er werken in totaal ongeveer 2800 mensen waarvan 500 op het hoofdkantoor en 500 op de IT-afdeling. Deze IT-afdeling is vooral gevestigd op De Brug in Apeldoorn

5.1 Knowledge graph

Veel van de data van het kadaster bevind zich in losstaande datasets die niet met elkaar zijn verbonden. Hierdoor is het vaak lastig om bijvoorbeeld alle informatie over een adres op te halen zoals WOZ-waarde, perceelnummer en bouwjaar. Omdat de WOZ-waarde in de “WOZ Landelijke Voorziening”, het perceelnummer in de BRK en het bouwjaar in de BAG is opgeslagen.

Om makkelijker met deze datasets om te kunnen gaan werkt het Kadaster aan een knowledge graph waarmee middels Linked-Data de datasets aan elkaar verbonden zijn. In figuur 1 is een overzicht te zien van de datasets die met de knowledge graph verbonden zijn.



Figuur 1 Kadaster knowledge Graph https://labs.kadaster.nl/cases/pdok-knowledge-graph.html

Om in deze datasets te zoeken wordt SPARQL gebruikt. Dat is een soort SQL maar dan voor Linked-Data en de Semantic Web.

6 Probleemstelling

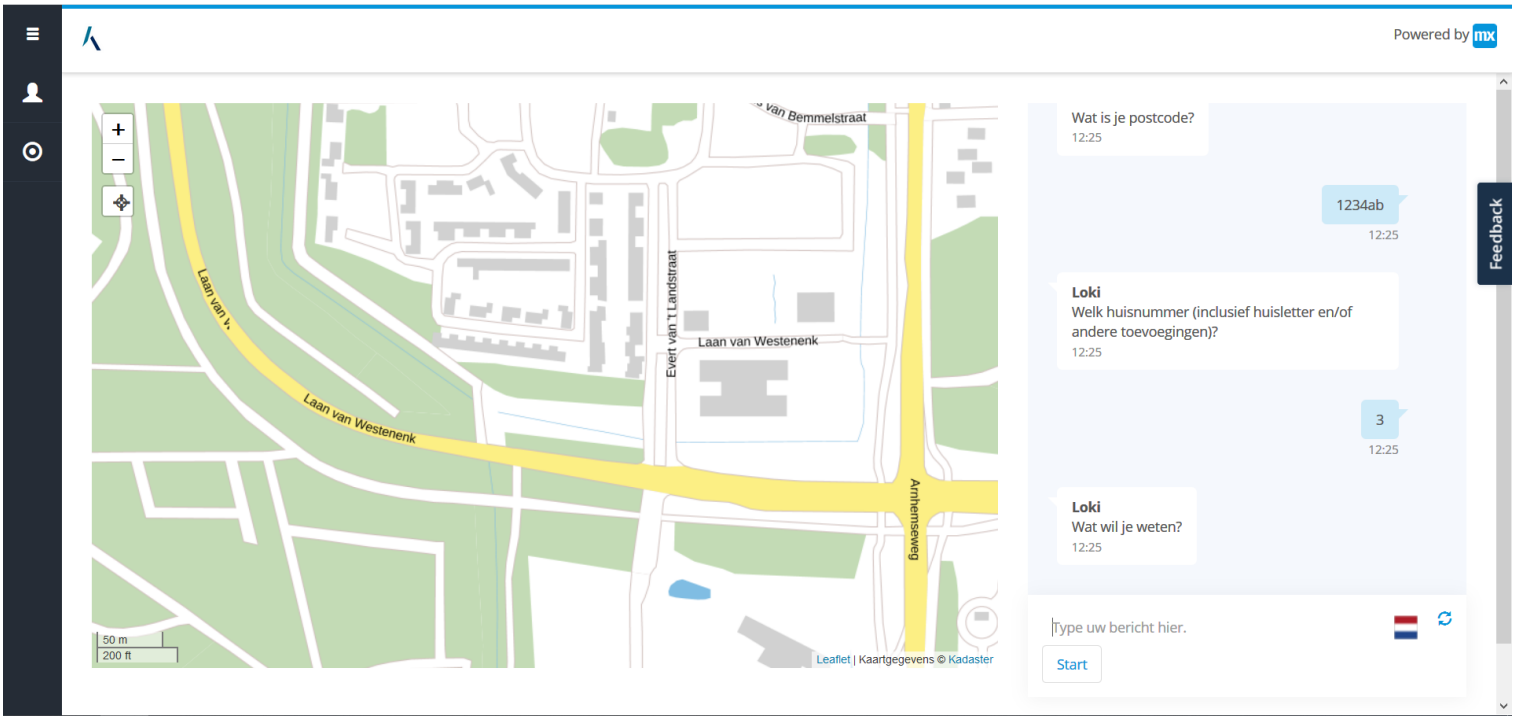
Het Kadaster heeft veel data over alles wat met de ruimte in Nederland te maken heeft. Echter zijn er niet veel toepassingen die deze data ontsluiten naar de burger en daar is het Kadaster wel naar opzoek.

Een van de ontwikkelingen op het gebied van deze ontsluitingen is “LOKI”. *“Loki is een experiment waarbij we de toegevoegde waarde aantonen van een chatbot voor Locatie-gebaseerde Kadaster Informatieverstrekking. Loki maakt gebruik van natuurlijke taalverwerking voor het beantwoorden van vragen.”*[1]

In de huidige versie van Loki zijn dit de type vragen die beantwoord kunnen worden:[24]

- “Wat is de WOZ waarde van mijn huis?”
- “Wat is het bouwjaar van mijn huis?”
- “Wat is de gemiddelde oppervlakte in mijn straat?”
- “Waar ligt mijn perceel?”
- “Wat zijn de huizen in Oranje gebouwd na 2000?”
- “Wat is het oudste huis in Haarlem?”
- “Geef mij alle kerken gebouwd voor 1500 in Dordrecht.”

Deze huidige versie heeft verder een front-end dat gemaakt is in Mendix, deze front-end is in figuur 2 weergegeven.



Figuur 2 Screenshot Loki 1.0

Omdat Loki slechts een experiment was, wordt het niet gebruikt in een context waar de eindgebruiker daadwerkelijk wat aan heeft. Het doel was namelijk om de technische haalbaarheid van het concept te toetsen. Zo is het in de huidige front-end bijvoorbeeld niet mogelijk om in je gesprek met Loki terug te scrollen. Verder kan de gebruiker slechts één keer een adres opgeven waarover vervolgens vragen gesteld kunnen worden. Als de gebruiker een ander anders wil bevragen moet de webpagina worden herladen. Loki geeft de gebruiker verder geen suggesties over wat er over een adres bevraagt kan worden. De gebruiker moet dus van tevoren al weten welke vragen Loki ondersteund. Tot slot is de zogenoemde “fallback policy” erg slecht, als de gebruiker namelijk iets intypt wat Loki niet begrijpt zegt Loki alleen “Ik heb je niet begrepen, probeer opnieuw”. Dit zou beter kunnen doormiddel van het weergeven van mogelijkheden waarvan Loki denkt dat ze het dichts in de buurt liggen van wat de gebruiker probeert te bereiken.

Het kadaster zou Loki graag beschikbaar maken voor eindgebruikers. Daarom is de opdracht om de huidige versie door te ontwikkelen tot een eerste, testbare, versie voor eindgebruikers, ofwel Loki 1.0. Hierbij is het ten eerste van belang dat de keuzes die zijn gemaakt in Loki 0.1 worden gevalideerd en waar nodig verbeterd. Zo kan er nu al gezegd worden dat de front-end een nieuwe implementatie nodig heeft maar is het nog onbekend welke verdere verbeteringen doorgevoerd kunnen worden. Om dat laatste te bepalen wordt er een vooronderzoek gedaan waarin o.a. wordt gekeken naar mogelijke manieren om chatbots slimmer te maken en wat we kunnen leren van op de markt aangeboden chatbot providers. Uit dit onderzoek moet blijken of de huidige versie van Loki genoeg basis biedt voor Loki 1.0. Mocht dit het geval zijn, dan ziet het kadaster graag mogelijke toevoegingen in de vorm van een integratie met Whatsapp of Telegram en een mogelijkheid om wandelroutes te genereren langs P.O.I’s(Points of interest). Deze P.O.I’s bestaan uit bijvoorbeeld monumenten, parken bijzondere gebouwen, etc. De gebruiker krijgt de route op een kaart te zien of kan deze met een link openen in Google Maps. Tijdens de wandelroute kan de gebruiker zijn locatie naar Loki sturen om informatie te krijgen over bijvoorbeeld het bouwjaar van woningen en andere kenmerken.

De integratie met Whatsapp of Telegram en wandelroute functionaliteit dient tevens als vangnet om er zeker van te zijn dat de beroepsactiviteit “realiseren” op niveau 3 kan worden afgerond. In het geval dat Loki 1.0 volledig herschreven moet worden zal dit vangnet komen te vervallen omdat het volledig herschrijven van Loki voldoende ruimte biedt om dit niveau van deze beroepsactiviteit aan te tonen. Voor de overige beroepsactiviteiten is er geen vangnet nodig omdat het vooronderzoek genoeg mogelijkheid bied om de beroeps activiteit “analyseren” op niveau 3 aan te tonen. En “ontwerpen” kan worden afgevangen middels het opstellen functioneel en technisch ontwerp. Waarin onder andere wordt beschreven hoe Loki 1.0 integreert in de bestaande systemen van het kadaster.

7 Vooronderzoek

Het onderzoek vindt plaats omdat het kadaster de huidige versie van Loki graag wil valideren en waar nodig verbeteren tot een state-of-the-art chatbot die op een logische manier gesprekken kan voeren. Daarbij vraagt het kadaster om een nieuwe front-end die net als de oude front-end kaartvisualisatie mogelijk maakt. Verder moet er gekeken worden naar wat de invulling van de Points of interest(POI) zal zijn voor de wandelroute functionaliteit. Deze drie aspecten vereisen allemaal een vooronderzoek.

De hoofdvraag waar in dit onderzoek antwoord op gegeven zal worden is:

- Wat is er nodig om Loki beschikbaar te stellen voor eindgebruikers?

De deelvragen hierbij zijn:

1. Hoe kan de chatbot, d.m.v. A.I, slim worden zonder dat elk mogelijke input van de gebruiker moet worden uitgeschreven in een datamodel?
2. Wat kunnen we leren van op de markt aangeboden chatbot providers?
3. Wat is de “state of the art” op het gebied van kaartvisualisatie en chatbots?
4. Welke kadaster data is interessant voor de invulling van de points of interest?
5. Is er, naast kadaster data, externe data die gebruikt kan worden voor de invulling van de points of interest?

In de volgende hoofdstukken zal antwoord op deze vragen gegeven worden.

7.1 Slimme chatbots

De meest simpele manier om een chatbot te realiseren is door een programma te maken wat voorgeprogrammeerde resultaten teruggeeft bij een bepaalde input. Het probleem hierbij is dat dit niet schaalt. Voor grote complexe gesprekken moeten alle mogelijke opties worden uitgeschreven in if-else statements. waarbij verdere diepgang in deze statements al snel erg onoverzichtelijk wordt. Dit wordt ook wel de “bag-of-rules” aanpak[1] of een “rule-based” chatbot[2] genoemd.

Een mogelijke oplossing hiervoor is het gebruik van machine learning. De meest gebruikte machine learning modellen voor chatbots zijn sequence-to-sequence en reinforcement learning[2]. Bij reinforcement learning probeert een bot zijn taak steeds opnieuw, waarbij er een beloning wordt gegeven als het goed gaat. Deze beloning kan gezien worden in de vorm van een score die omhoog gaat als er iets goed gaat en omlaag als de bot iets verkeerd doet, Het doel dat de bot vervolgens wordt gegeven is om een zo hoog mogelijke score te bemachtigen. Echter is er veel data nodig voordat een bot zinvolle gesprekken kan hebben. Daarbij is het bij een chatbot vaak moeilijk te bepalen wat de beloning kan zijn en dus wanneer de score omhoog moet gaan.[1]

Een andere aanpak voor het maken van een chatbot met machine learning is dus door het toepassen van sequence-to-sequence learning. Dit is ook de manier wat op dit moment de beste chatbots oplevert.[2][5] De methode maakt gebruik van Recurrent neural networks(RNN)[2] om tot het beste antwoord te komen. Echter heeft ook deze aanpak grote hoeveelheden bestaande gesprekken nodig om het model te kunnen trainen. Verder wordt zowel seq2seq als reinforcement learning vooral gebruikt voor het maken van chatbots die alledaagse gesprekken kunnen hebben.[2] Ze bepalen hun antwoord alleen op basis van de gesprekken waarop ze zijn getraind en kunnen deze antwoorden dus niet uit een database opvragen.

Chatbots die dat wel kunnen worden ook wel Information Retrieval(IR) gebaseerde chatbots genoemd. Deze chatbots hebben een grote database met mogelijke vragen en bijbehorende antwoorden. De bot zoekt voor elke vraag van de gebruiker welke vraag uit de database het meest overeenkomt en geeft het bijbehorende antwoord terug.[2] In principe lijkt deze vorm van chatbot heel erg op een zoekmachine. Het verschil is alleen dat een zoekmachine niet goed kan zoeken op basis van natuurlijke tekst omdat zoekmachines vaak zoeken op basis van trefwoorden. Deze chatbot doet dit aan de hand van similarities tussen de opgegeven zin en de zinnen die in de database aan antwoorden zijn gekoppeld. Ook de machine learning methodes werken op basis van natuurlijke tekst. Het begrijpen van deze tekst en welk antwoord er gegeven moet worden gebeurt daar volledig aan de hand van het getrainde model.

De hiervoor beschreven chatbot soorten hebben, met uitzondering van de rule-based chatbot, een groot probleem. Ze kunnen niet informatie ophalen of verzenden naar andere systemen omdat deze informatie in de training data of voor de IR-chatbot in de database moet zitten. Hierdoor is het doen van een API call niet mogelijk is. Je kan hierdoor bijvoorbeeld geen vlucht boeken, afspraak maken of levertijd opvragen. Omdat een rule-based chatbot geen schaalbare oplossing is zijn er frameworks bedacht die het mogelijk maken om een chatbot te realiseren zonder alle mogelijke uitkomsten uit te schrijven. Voorbeelden hiervan zijn: Microsoft's Bot Framework, IBM Watson, Dialogflow en kore.ai.[4]

Het idee bij deze frameworks is dat ze het doel van de gebruiker, ofwel "intent" bepalen. Een ontwikkelaar moet dan eerst lijsten definiëren met zinnen en woorden die bij een intent horen. Vervolgens kan het framework met machine learning voorspellen welke input van de gebruiker bij een intent hoort.[15] Het kan dus zo zijn dat een zin die oorspronkelijk niet is gedefinieerd door de ontwikkelaar wel wordt toegewezen aan een bepaald intent. Aan een intent zit verder een actie gebonden. Dit kan het teruggeven van een simpel statisch antwoord zijn maar een actie kan ook een API-call bevatten waardoor er een dynamisch antwoord gegenereerd kan worden.

```
## intent: weather
- wat is het weer?
- hoe is het buiten?
- schijnt de zon?
```

Figuur 3 Intent lijst voorbeeld

In figuur 3 is een voorbeeld van een intent weergegeven met drie mogelijke opties van zinnen die bij dit intent horen. Een mogelijke actie zou dan een statische zin met een verwijzing naar een weersvoorspelling kunnen zijn, bijvoorbeeld: "Je kunt de actuele weersverwachting bekijken op www.weeronline.nl". Een andere mogelijke actie bij dit intent zou een API-call kunnen zijn die de weersverwachting ophaalt en een samengesteld antwoord teruggeeft, bijvoorbeeld: "Het is 15 graden en bewolkt".

Vaak is het nodig om bepaalde informatie uit de input van een gebruiker te halen. bijvoorbeeld de tijd, datum of locatie. Hiervoor wordt "Named Entity Recognition" (NER) toegepast.

```
## intent: weather
- wat is het weer [vandaag](date)?
- wat is het weer in [Apeldoorn](location)?
- regent het [morgen](date) in [Amsterdam](location)?
```

Figuur 4 Intent lijst met entities voorbeeld

In figuur 4 is te zien hoe een ontwikkelaar in de intent lijst plekken in een zin aan kan geven waar een entity wordt verwacht. Vervolgens kan bij het uitvoeren van de actie die bij deze intent hoort worden gekeken wat er daadwerkelijk als entitie is opgegeven. De combinatie van NER, lijsten met intents en machine learning stelt een chatbot in staat om natuurlijke tekst van een gebruiker te begrijpen en de juiste actie toe te passen. Dit begrijpen van natuurlijke tekst wordt ook wel “Natural Language Understanding” of NLU genoemd.

Een antwoord op de eerste deelvraag is dus dat het in zekere zin mogelijk is om een chatbot te maken waarbij niet elke mogelijke input van de gebruiker moet worden gedefinieerd in een intent-lijst. Met het intent-systeem kan doormiddel van machine learning op termijn worden bepaald welke niet-gedefinieerde gebruikersinput bij een intent hoort. Echter betekent dit nog steeds dat er hoe dan ook lijsten moeten worden gedefinieerd met mogelijke gebruikers input. Alhoewel er mogelijkheden zijn om dit automatisch te generen[6] lijkt dit niet de juiste oplossing. Intents zullen voor nu altijd nodig zijn om zonder enige training data een chatbot te kunnen realiseren. Zodra deze chatbot dan met gebruikers communiceert geneerd het in feiten zijn eigen training data. Deze training data wordt in de huidige implementatie van het intent-systeem dus nog gebruikt om het machine learning model beter te trainen in het voorspellen van welke input van de gebruiker bij een intent hoort. Echter is het waarschijnlijk eind 2020 al mogelijk om de gegenereerde training data te gebruiken om van het intent-systeem af te stappen en naarmate je chatbot meer data genereerd een machine learning model te trainen die gebruikers input direct aan een actie of response kan koppelen zonder deze input eerst in een intent te moeten classificeren.[7] Met deze toekomstige nieuwe technologie is het dus belangrijk dat een chatbot al genoeg data heeft gegenereerd middels het intent-systeem.

7.2 Chatbot implementaties

Het intent-systeem wordt door verschillende partijen beschikbaar gesteld voor bedrijven of consumenten. Denk hierbij aan de eerder genoemde chatbotservices: Microsoft’s Bot Framework, IBM Watson, Dialogflow en kore.ai. Maar er zijn ook opensource mogelijkheden zoals Rasa[8][15] en Deeppavlov[9]. Dit laatste framework lijk nog in het begin van zijn ontwikkeling en niet beschikbaar voor Nederlandse implementaties[10].

Het verschil in de verschillende chatbotservices zit in de userinterface waarmee een bot gemaakt kan worden, de NLU die deze bots gebruiken en de talen die ze ondersteunen.[11] Verder bieden sommige van de chatbotservices built-in entities ook wel “system entities” dat zijn entities die je niet meer hoeft te trainen en het systeem dus al begrijpt. Denk hierbij aan objecten zoals locaties, datum, tijd of valuta. Deze system entities bestaan alleen in Dialogflow[12] en IBM Watson[13].

Zoals eerder beschreven biedt Rasa een opensource oplossing voor het maken van een chatbot. Het verschil van Rasa en chatbotservices zoals Dialogflow is dat je bij het gebruik van Rasa alle data, en dus gesprekken met je chatbot, zelf beheert. In het kader van data privacy is dit een groot voordeel. Daarbij gaan er bij chatbotservices vaak kosten gemoeid om de service te kunnen gebruiken. Verder biedt Rasa meer flexibiliteit voor het configureren van de NLU[14]. Zo is het, voor enkele talen, mogelijk om gebruik te maken van een bestaand NLU model genaamd Spacy waardoor er al enige kennis van taal is over bijvoorbeeld welke betekenissen van woorden erg op elkaar lijken. Het gebruik van Spacy helpt verder om Rasa goed te kunnen laten concurreren met chatbotservices zoals Dialogflow. In zowel een onderzoek uit 2017 als 2019 werd geconcludeerd dat ondanks dat chatbotservices veel meer data tot hun beschikking hebben, Rasa net zo goed of niet veel slechter scoort dan de commerciële alternatieven.[16][17] Deze onderzoeken hebben verder gebruik

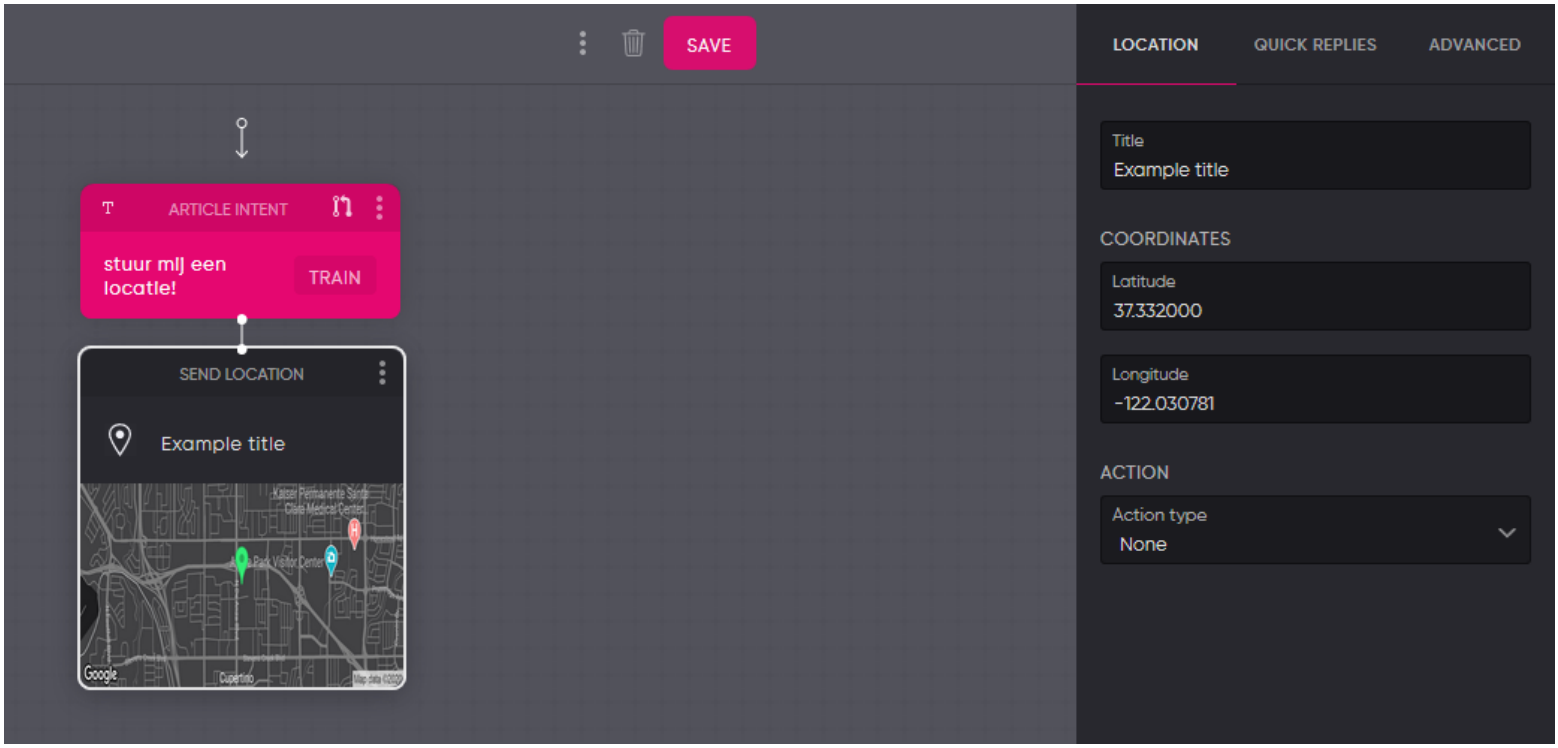
gemaakt van Spacy en Mitie als initieel taal model. Echter raad Rasa aan om bij zeer domein specifieke chatbots geen gebruik te maken van deze taalmodellen[14] omdat de domein specifieke terminologie mogelijk verbanden heeft met woorden die in het geval van een zeer domein specifieke chatbot buiten het domein vallen.

Een andere manier waarop Rasa meer flexibiliteit biedt voor het configureren van de NLU is de “fallback policy”. Dat zijn regels die een ontwikkelaar kan definiëren als de chatbot situaties tegenkomt waarbij de gebruiker niet wordt begrepen. Waar chatbotservices vaak een enkele fallback mogelijk maken, bijvoorbeeld “Ik heb je niet begrepen”. Biedt Rasa opties om dit uit te breiden. Een ontwikkelaar moet namelijk een threshold instellen wat bepaald vanaf wanneer de fallback moet worden geactiveerd. Als de threshold bijvoorbeeld op 0.6 staat en Rasa de hoogste intent scoort op 0.5, zal de fallback geactiveerd worden. Mocht er echter een two-stage fallback zijn ingesteld. Dan vraagt Rasa eerst of de intent met een classificatie score van 0.5 werd bedoeld. Als dit niet het geval is zal Rasa alsnog de standaard fallback gebruiken wat dan een zin kan zijn als “Ik begrijp je niet.”

Alle, op de markt aangeboden chatbotservices, werken dus volgens hetzelfde principe. Een interessant aspect van Dialogflow en Watson is de introductie van system entities. Dit kan ontwikkelaars helpen om een chatbot sneller beschikbaar te stellen voor de eindgebruiker. Ook Rasa introduceert dit idee doormiddel van lookup tables. Dit zijn grote lijsten met bijvoorbeeld alle plaatsen in Nederland of alle woorden die een datum kunnen beschrijven. Het verschil hiermee is dat de ontwikkelaar de data van Rasa, dus ook de lookup tables, zelf beheert en dus up-to-date moet houden. Dit zelf beheren van zowel de data, als het beschikbaar stellen van de chatbot is het belangrijkste verschil tussen Rasa en een chatbotservice.

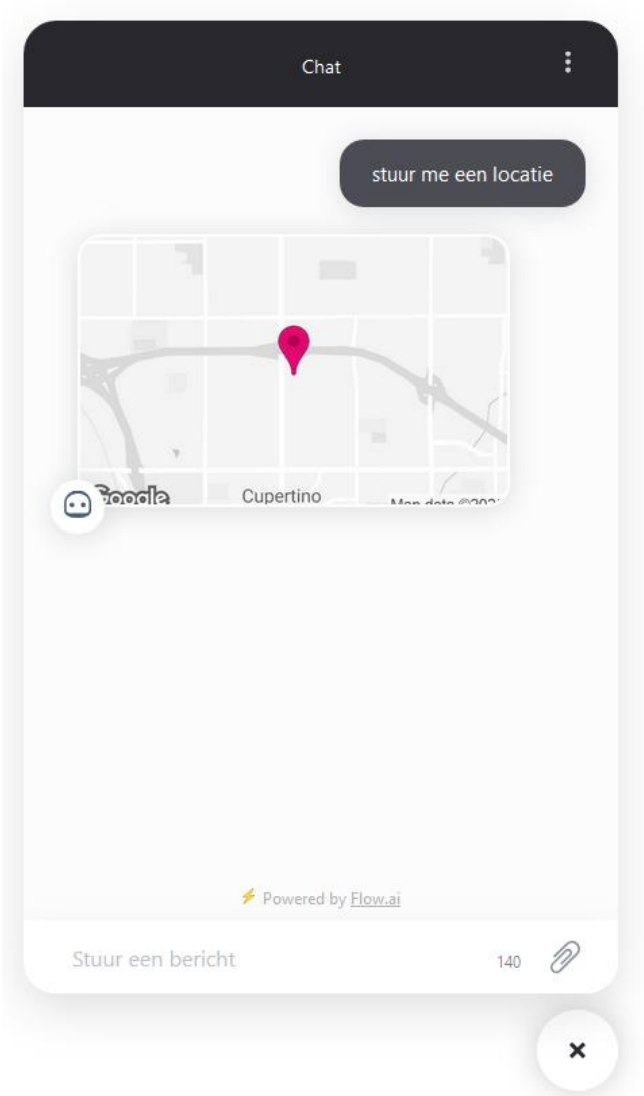
7.3 Kaartvisualisaties

Geen van de hiervoor beschreven chatbotservices biedt een standaard manier om kaarten weer te geven aan de gebruiker. Een service die dit wel kan is flow.ai.[18] Deze chatbotservice lijkt verder veel op de hiervoor genoemde services en is beschikbaar in het Nederlands. Ze stellen een standaard antwoord beschikbaar die in de web-interface te zien is als een afbeelding in de chat. Er is verder geen interactie met deze afbeelding mogelijk. In het figuur hieronder is de visualisatie van een simpel gesprek te zien waar als de gebruiker zegt : “stuur me een locatie” de chatbot een locatie stuurt.



Figuur 5 Screenshot flow.ai

Deze afbeelding laat een voorbeeld zien van het sturen van een locatie met statische coördinaten. Flow.ai biedt verder een mogelijkheid om deze coördinaten dynamisch te genereren op basis van het voorgaande gesprek met de gebruiker doormiddel van het programmeren van code blokken in javascript.



Figuur 6 Screenshot standaard web-interface van flow.ai

In figuur 6 is de standaard web-interface van flow.ai te zien waarin het hiervoor gedefinieerde gesprek is doorlopen. De locatie wordt dus enkel weergegeven als afbeelding waar verder geen interactie mee mogelijk is. Deze interactie kan, net als bij de dynamische coördinaten, door middel van het programmeren van javascript wel tot stand gebracht worden. Het lijkt verder niet mogelijk om bij het gebruik van deze web-interface informatie van de chat buiten de chat-window weer te geven. Zo kan er bijvoorbeeld geen ondersteunende kaart buiten de chat getoond worden die locaties weergeeft op basis van het gesprek. Dit kan natuurlijk wel geïmplementeerd worden als er gebruik gemaakt wordt van de API van flow.ai[18] net als bij de andere chatbotservices.

Samengevat biedt flow.ai de enige out-of-the-box oplossing voor het weergeven van kaarten. Deze kaarten zijn verder niet interactief en alleen aan te passen door het schrijven van code, wat betekent dat het weergeven van interactieve kaarten in een chatbot alleen mogelijk is met eigen gemaakte implementaties.

7.4 Kadasterdata

Het kadaster beheert verschillende open datasets die worden gepubliceerd op de website van PDOK, ofwel Publieke Dienstverlening Op de Kaart[20]. Hiervan zijn er 5 het meest belangrijk voor de dienstverlening van het kadaster, deze worden in tabel 1 weergegeven.

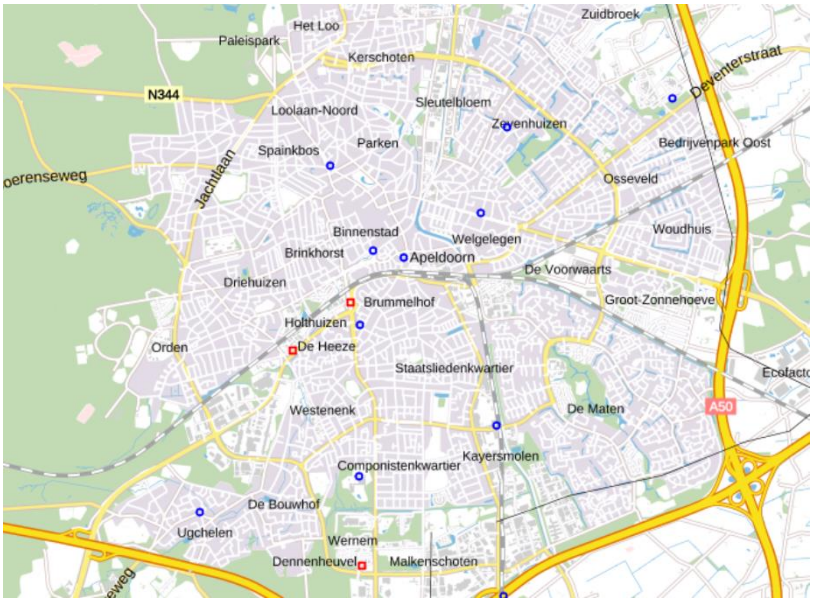
AFKORTING	NAAM	OMSCHRIJVING
BAG	Basisregistratie Adressen en Gebouwen	Gegevens van alle adressen en gebouwen in Nederland, zoals bouwjaar, oppervlakte, gebruiksdoel en locatie op de kaart.
BRK	Basisregistratie Kadaster	Bestaat uit de kadastrale registratie en de kadastrale kaart waarop perceel grenzen zijn weergegeven.
BRT	Basisregistratie Topografie	Bevat objectgerichte topografische bestanden op diverse schaalniveaus
BGT	Basisregistratie Grootschalige Topografie	De gedetailleerde grootschalige basiskaart (digitale kaart) van heel Nederland.
RD INFO	Rijksdriehoek informatie	Overzicht van de ligging van de Rijksdriehoek-punten

Tabel 1 Kadaster datasets en afkortingen

Met uitzondering van de BGT en RD Info, worden alle hierboven genoemde datasets o.a. ontsloten doormiddel van Linked Open Data. Wat betekent dat de data vrij te bevragen is en semantiek bevat.

Voor de invulling van de P.O.I's zijn vooral de BAG en de BRT interessant. In de BAG staan namelijk bouwjaren van huizen waarmee bijvoorbeeld gebouwen die gebouwd zijn vóór 1800 kunnen worden opgevraagd, hiermee kan een wandelroute langs oude gebouwen gerealiseerd worden. De BRT kan verder nuttig zijn omdat hierin de soort van het gebouw wordt beschreven. Hiermee kunnen de wandelroutes langs bijvoorbeeld kerken, scholen of bunkers worden gelopen.

De laatste in tabel 1 beschreven dataset is het Rijksdriehoeksstelsel(RD). Dit is een van de oudste datasets van het Kadaster en wordt sinds 1885 beheert. "*Het Rijksdriehoeksstelsel is onderdeel van de geodetische infrastructuur van Nederland. Dit landelijke netwerk van coördinaatpunten wordt gebruikt voor landmeetkundige werkzaamheden en plaatsbepaling. Het stelsel bestaat uit actieve GNSS-referentiestations (GNSS is de afkorting van Global Navigation Satellite System) die constant signalen uitzenden en 'passieve' punten zoals kerktorens en grondankers. Behalve het Rijksdriehoeksstelsel maken ook de referentiesystemen AGRS en NETPOS onderdeel uit van de geometrische infrastructuur van Nederland.*"[21]



Figuur 7 Rijksdriehoekspunten in Apeldoorn

Omdat er veel van deze punten in het land bestaan is het, net als de BAG en de BRT, een dataset waar mogelijk wandelroutes langs gegenereerd kunnen worden. Deze dataset is verder niet beschikbaar in Linked Data maar WMS(Web Map Service) en WFS(Web Feature Service).

7.5 Externe data

De website van PDOK wordt niet alleen gebruikt voor het publiceren van kadaster data, ook partijen als Rijkswaterstaat(RWS), het ministerie van Binnenlandse Zaken en Koninkrijkrelaties(BZK) en het centraal bureau voor de statistiek(CBS) publiceren datasets op PDOK. Maar ook stichtingen zoals Stichting Landelijk Fietsplatform en Stichting Wandelnet stellen datasets beschikbaar op de website.

Figuur 8 Stadsroutes in Apeldoorn van Stichting Wandelnet

In figuur 8 zijn stadswandelingen door Apeldoorn te zien opgesteld door stichting Wandelnet. Deze routes zouden in plaats van een dynamische route langs P.O.I's gebruikt kunnen worden voor de wandelroute functionaliteit. De datasets zijn niet ontsloten middels Linked data maar alleen WMS(Web Map Service).[20]

Nog een interessant, via PDOK beschikbaar gesteld, dataset is dat van de NAP-peilmerken. "*Om overal in Nederland de hoogte ten opzichte van het NAP te kunnen bepalen, zijn er door het hele land ongeveer 35.000 peilmerken aangebracht. Deze NAP-peilmerken hebben een hoogte ten opzichte van het NAP en zijn verankerd in onder meer woonhuizen, bruggen, viaducten. Zo kunnen we gemakkelijk de waterstand bepalen of de hoogte van een bouwwerk. Vrijwel overal in Nederland is binnen de afstand van 1 km een peilmerk te vinden.*"[22] Omdat deze peilmerken relatief dicht bij elkaar staan en verwerkt kan zijn in infrastructuur, zou het een mogelijke optie kunnen zijn voor de invulling van de P.O.I's. De dataset wordt verder net als het Rijksdriehoeksstelsel ontsloten in WMS en WFS.

Ook buiten PDOK bestaat er een interessante dataset. Namelijk die van de Rijksdienst voor het Cultureel Erfgoed(RCE). Ook deze dataset is net als de BAG en BRT ontsloten als Linked open data. Met deze dataset kunnen interessante wandelroutes gegenereerd worden langs archeologische vindplaatsen en terreinen of monumenten in Nederland die door het Rijk zijn aangewezen als beschermd monument. Dit zijn historische gebouwen, archeologische vindplaatsen of groene, aangelegde bouwwerken.[23]

NAAM	PDOK	OMSCHRIJVING	BEHEERDER	LINKEDDATA
LANDELIJKE WANDELROUTES	Ja	Dataset met wandelroutes verspreid over Nederland	Stichting Wandelnet	Nee
LANDELIJKE FIETSROUTES	Ja	Dataset met fietsroutes verspreid over Nederland	Stichting Landelijk Fietsplatform	Nee
NAP-PEILMERKEN	Ja	Locaties van Nap-peilmerken waarmee overal in Nederland de waterstand kan worden bepaald	RWS	Nee
CULTUREEL ERFGOED	Nee	Linked data over cultuurhistorische objecten	RCE	Ja

Tabel 2 Externe datasets.

7.6 Conclusie

Met dit onderzoek is een globaal beeld tot stand gekomen van zowel chatbots, en de implementatie daarvan, als P.O.I mogelijkheden. Dit wordt gebruikt om de beste invulling aan de criteria te geven die samen met het Kadaster voor Loki zijn opgesteld. Deze criteria zijn hieronder weergegeven.

1. De chatbot maakt gebruik van de nieuwste “proven” technologie. Dus technologie die schaalbaar is, en waarmee een betrouwbaar product gerealiseerd kan worden.
2. De taal van de chatbot moet Nederlands zijn.
3. De chatbot moet kaartvisualisaties mogelijk maken.
4. De chatbot maakt gebruik van bestaande Linked Data bronnen
5. De chatbot maakt zoveel mogelijk gebruik van bestaande tools en componenten om ontwikkeltijd en effort te minimaliseren.

Uit dit onderzoek is gebleken dat chatbotservices geen grote voordelen bieden ten opzichte van Rasa. Vooral als het gaat om performance zoals in bron 16 en 17 onderzocht, blijkt er geen groot verschil te zijn tussen Rasa en chatbotservices. Ook tussen de chatbotservices onderling zijn er weinig verschillen en voordelen te zien. Er kan dus geconcludeerd worden dat zowel Rasa als alle chatbotservices kunnen voldoen aan de beschreven criteria. Echter biedt Rasa wel meer mogelijkheden voor het configureren van de NLU waardoor dit ten alle tijden de “nieuwste technologie” kan zijn en in elk mogelijke taal. Daarbij wordt Rasa al sinds 2016 gebruikt voor het maken van chatbots waardoor het “proven technologie” genoemd kan worden.[1] Verder zal het voor elke chatbot implementatie noodzakelijk zijn om kaartvisualisaties mogelijk te maken, het maakt voor dit criteria dus niet uit of er een van de chatbotservices wordt gebruikt of Rasa. Dit geldt ook voor het Linked Data criteria. Het laatste criteria kan verder ook toegepast worden op Rasa omdat het vooral gaat om effort en ontwikkeltijd, aangezien er al een versie van Loki bestaat in Rasa zal het door ontwikkelen op deze versie minder tijd en effort kosten. Daarbij is Rasa opensource waardoor er een community achter zit die verschillende tools en componenten beschikbaar stellen.

De criteria beschrijven verder dat de chatbot alleen gebruik maakt van Linked Data. Dit betekent dat de invulling van de P.O.I’s uit Linked Data bronnen moet bestaan. De BRT, BAG en Rijksmonumenten zijn zulke databronnen. Omdat Loki verder al een connectie heeft met de BRT en de BAG en zojuist beschreven is dat deze implementatie door ontwikkeld wordt lijken deze databronnen het meest geschikt voor de invulling van de P.O.I’s. Het is verder een streven om de implementatie van deze P.O.I’s zo generiek mogelijk te maken zodat er gemakkelijk datasets aan toegevoegd kunnen worden. Zo zou het bijvoorbeeld mogelijk moeten zijn om de dataset van het Rijksdriehoeksstelsel om te zetten naar Linked Data waardoor er ook wandel routes langs deze punten gegenereerd kan worden.

8 Aanpak

Dit hoofdstuk beschrijft de daadwerkelijke aanpak van het realiseren van de opdracht. Allereerst wordt een tijdlijn beschreven van gemaakte keuzes en de onderbouwingen daarbij. Daarna volgt een beschrijving van de gebruikte tooling. En tot slot de manier waarop procesmatig is gewerkt door gebruik te maken van bepaalde methoden.

8.1 Tijdlijn

De uitkomst van het vooronderzoek betekende in eerste instantie dat grote delen van de experimentele versie van Loki hergebruikt konden worden omdat de chatbot gebruik bleef maken van Rasa. Dit betekende dat de focus voor Loki 1.0 vooral lag bij het maken van een front-end en het toevoegen van de wandelroute functionaliteit waarbij slechts kleine aanpassingen doorgevoerd moesten worden in de manier waarop de gebruiker wordt begeleid door een gesprek. Bijvoorbeeld de toevoeging van suggestie knoppen en een twostagefallbackpolicy.

Dit wetende is er begonnen aan het schrijven van het functioneel ontwerp. In de eerste versie hiervan is nog goed te zien hoe de functionaliteiten die in de experimentele versie van Loki zitten niet worden ontworpen, deze functies waren immers al gerealiseerd. Er werd vooral gekeken naar hoe de front-end eruit moest zien, en wat de invulling van de wandelroute functionaliteit was. Het ontwerp van de front-end heeft verder meerdere iteraties gekend omdat hierbij veel overleg heeft plaatsgevonden met leden uit het team, de bedrijfsbegeleider, een UX-designer en een medewerker van de afdeling: marketing en communicatie.

Na ongeveer 2 weken ontwerpen werd er gestart aan de realisatie. Tijdens deze realisatie werd al snel duidelijk dat het erg moeilijk zou zijn om de backend van de experimentele versie van Loki, in grote delen, te kunnen hergebruiken. Hiervoor waren de volgende redenen:

- Matige kwaliteit van code
    - Er bestonden stukken code die nooit zouden kunnen werken.
    - Meerder keren exacte dezelfde code geschreven
    - Een enkel bestand bestaande uit meer dan 2500 regels code voor het beschrijven van alle actions.
    - Er is nooit rekening gehouden met mogelijke doorontwikkelingen. Er bestond alleen het doel om het concept te bewijzen.
- Niet gebruikersvriendelijk
    - Zodra de chat begon vroeg de chatbot direct om een postcode en huisnummer van de gebruiker, zonder enige context. De gebruiker kon pas verder met het gesprek als deze informatie was opgehaald. Veel functionaliteiten waren vervolgens van deze informatie, die direct aan de start van een gesprek gegeven moest worden, afhankelijk.
- Keuzes die niet in lijn lagen met wat Rasa beschreef
    - Voor het herkennen van een huisnummer is een huisnummer entity getraind bestaande uit enkel een nummer, terwijl voor het detecteren van een nummer in de tekst al oplossingen bestaan.
    - Unhappy chatflows waren soms afgevangen in de actions. Actions mogen alleen gebruikt worden voor het doen van een actie en niet het bepalen van een response.
    - Er stonden veel dubbele zinnen en woorden in de NLU. Dit kan ervoor zorgen dat het model gaat overfitten.

- Geen documentatie van de code
    - Er bestond geen technische documentatie van de code waardoor het erg lastig was code, en de keuzes daarbij, te begrijpen.
- Responses speciaal voor Mendix
    - Sommige responses waren speciaal samengesteld zodat Mendix er wat mee kon.

Omdat het vrijwel onmogelijk zou zijn om de experimentele versie van Loki om te bouwen tot een bug-vrije testbare versie voor de eindgebruiker is er, samen met leden van het team en de bedrijfsbegeleider, voor gekozen om opnieuw te beginnen. Dit betekende dat het vangnet zou komen te vervallen omdat het volledig opnieuw opbouwen van de Rasa-backend voldoende ruimte biedt om de beroepsactiviteit “realiseren” op niveau 3 te kunnen aantonen. Verder betekende dit dat ook de functies van de backend eerst moesten worden ontworpen in het functioneel ontwerp. Er was dus een korte extra ontwerp fase nodig voordat de backend kon worden opgebouwd.

Na deze extra ontwerpfase is er begonnen aan het opzetten van een nieuw Rasa project waarin de ontworpen functionaliteiten toegevoegd konden worden. Het toevoegen van deze functionaliteiten is het resterende deel van de stage aan gewerkt.

8.2 Tooling

Tijdens de afstudeerstage is er gebruik gemaakt van verschillende tooling om het proces te ondersteunen.

8.2.1 Slack

Voor de snelle communicatie met teamleden werd gebruik gemaakt van Slack. Dit maakte het mogelijk om goed en snel contact te houden zonder de formaliteiten van email

8.2.2 Outlook

Als aanvulling op Slack is er ook veel gebruik gemaakt van outlook voor het email verkeer tussen de bedrijfsbegeleider, team leden of andere medewerkers van het kadaster.

8.2.3 Microsoft Teams

Voor online vergaderingen is er gebruik gemaakt van Teams. Vooral toen iedereen vanuit huis werkte is Teams veel gebruikt voor bijvoorbeeld Stand-ups of het voortgangsgesprek met de bedrijfsbegeleider.

8.2.4 Citrix

Dit programma is gebruikt om ook vanuit huis het kadasternetwerk te kunnen benaderen. Bijvoorbeeld voor bepaalde afgeschermde informatie die alleen via deze verbinding opgehaald kon worden.

8.2.5 Intergrated Development Environments(IDEs)

Voor het ontwikkelen van de python code van Rasa is gebruikt gemaakt van Pycharm. Dit was erg handig omdat het over in intellisense beschikt waardoor argumenten van methodes gemakkelijk achterhaald konden worden. Verder is voor de front-end gebruik gemaakt van Visual Studio Code omdat dit een goede integratie had met het ontwikkelen binnen een Angular project.

8.2.6 Jira

Voor het bijhouden van het scrumproces is Jira gebruikt. Dit omdat Jira ook gebruikt werd door het team waardoor het gebruik van Jira hen beter in staat stelde op de hoogte te blijven van de ontwikkelingen binnen het project.

8.2.7 Git en GitHub

Voor versiebeheer van de code is Git gebruikt. Verder is deze code beschikbaar gesteld in een eigen repository op de GitHub van het Kadaster: https://github.com/kadaster-it

8.2.8 Figma.com

Tijdens het ontwerp proces is voor het ontwerpen van de front-end gebruik gemaakt van Figma.com. Dit stelde het ontwerp online beschikbaar waardoor het gemakkelijk door verschillende partijen in te zien is. Verder beschikt Figma over de mogelijkheid om snel een klikbaar prototype te realiseren van het ontwerp.

8.3 Methoden

De methoden die zijn gebruikt voor het waarborgen van procesmatig werken zijn ten eerste het gebruik van scrum. Dit is op een manier gedaan die beter paste bij een project waar slechts 1 persoon aan werkt, daarbij is rekening gehouden met de manier waarop het team op de hoogte blijft van de ontwikkelingen. Zo is ervoor gekozen mee te doen met de sprints van het team die een lengte hebben van 3 weken, hierdoor kon er ook mee worden gedaan met de retrospectieves. De stories van het afstudeerproject vielen verder onder een eigen “Epic” zodat snel kon worden gezien of een story bij de stage hoorde of de reguliere taken van het team. Verder hadden de stories geen story points omdat dit voor het relatief korte stage project niet relevant zou zijn en daarbij zorgde dit er ook voor dat het de velocity van het team niet aangetast zou worden.

Verder is er voor het ontwikkelen van de code gebruik gemaakt van branches, codereviews en pull request. Voor de ontwikkeling van elke story werd een aparte branch aangemaakt. Als alle taken van deze story afgerond waren werd er een pull request aangemaakt zodat een lid van het team de code kon beoordelen. Waar nodig werden aanpassingen gemaakt waarna werd de code gemerged. Hierdoor kon niet alleen de kwaliteit van de code gewaarborgd worden, maar bleef ook het team op de hoogte van de codebase.

9 Functioneel ontwerp

In dit hoofdstuk volgt het functioneel ontwerp. Dit document wordt tevens als los bestand opgeleverd aan het Kadaster omdat het de beoogde functionaliteiten van het project beschrijft. De hoofdstukken die het functioneel ontwerp bevat zijn:

1. Doel
2. Requirements
3. Chatfunctionaliteit
4. Chatflow
5. Designs
6. Overige functionaliteit

Omdat “Chatflow” en “Designs” een uitwerking is van wat er in “requirements” en “chatfunctionaliteit” wordt beschreven worden deze hoofdstukken hier niet volledig behandeld. Hiervoor wordt verwezen naar het daadwerkelijke functioneel ontwerp dat als bijlage is toegevoegd. Tot slot beschrijft het laatste hoofdstuk de vangnet bepaling van de originele opdracht. Dit is dus komen te vervallen doordat er opnieuw is begonnen met het implementeren van de Rasa backend. Echter word dit hoofdstuk nog wel beschreven omdat het volledig inzicht kan geven in de ontwerp fase van het project.

9.1 Doel

Het doel van het eindproduct is een testbare versie voor de eindgebruiker. Burgers moeten in staat zijn om vragen te stellen omtrent zaken van het Kadaster. De chatbot zal vervolgens het beste antwoord aan de gebruiker terug geven op basis van meerdere data bronnen. De gebruiker hoeft hierdoor niet zelf deze bronnen te doorzoeken, dit wordt door de chatbot gedaan.

9.2 Requirements

In de tabel hieronder zijn de requirements weergeven. Alle requirements zijn SMART ofwel, Specifiek, Meetbaar, Acceptabel, Realistisch en Testbaar. Verder zijn ze volgens MoSCoW geprioriteerd in “Must have”, “Should have”, “Could have” en “Would have”. En tot slot wordt aangegeven of een requirement functioneel of non-functioneel is door middel van een ‘F’ voor functioneel en ‘NF’ voor non-functioneel.

Nr.	Requirement	Prioriteit	Type
1	De chatbot kan enkel vragen beantwoorden op basis van data in de kadaster knowledge graph.	Must	F
2	De chatbot heeft een startscherm waarop een introductie van de chatbot is weergegeven met suggesties van wat de gebruiker kan zeggen.	Must	F
3	Gesprekken van gebruikers kunnen worden bekeken om het mogelijk te maken de bot, bij fouten, te optimaliseren.	Must	F
4	De chatbot maakt gebruik van chatbot management tooling	Must	NF
5	De chatbot geeft, gedurende het gesprek, suggesties van wat de gebruiker kan zeggen in de vorm van knoppen.	Must	F
6	De chatbot kan locaties weergeven op een kaart in het chatvenster.	Must	F
7	De gebruiker kan een weergegeven locatie in het chatvenster, nader bekijken op een interactieve kaart.	Must	f
8	De chatbot refereert, middels een link, altijd naar de bron van de data waarop een antwoord is gebaseerd.	Must	F
9	De chatbot heeft een responsive front-end	Must	NF
10	De chatbot beschikt over een feedback mogelijkheid aan het eind van een gesprek	Must	F
11	De chatbot kan benaderd worden via whatsapp of telegram.	Could	NF
12	De chatbot kan wandelroutes genereren langs P.O.I’s op basis van de opgegeven locatie van de gebruiker.	Should	F
13	De chatbot geeft informatie over P.O.I’s als de gebruiker aangeeft zich in de buurt van dit punt te bevinden.	Should	F

Tabel 3 Requirements Loki 1.0

9.3 Chatfunctionaliteit

Requirement 1 beschrijft dat Loki alleen vragen mag beantwoorden op basis van de kadaster knowledge graph. Dit zijn de beoogde chatfunctionaliteiten die middels deze knowledge graph behaalt kunnen worden:

1. Weergeven van de volgende eigenschappen van een adres:
    a. Oppervlakte
    b. Bouwjaar
    c. WOZ-waarde
    d. Status (Bijv: “in gebruik”)
    e. Perceeloppervlakte (of kadastrale grootte)
    f. Perceelnummer (of kavelnummer)
2. Weergeven van de afstand(In km) tussen een adres en het dichtstbijzijnde gebouw van een bepaald type. (Bijv: Universiteit of ziekenhuis)
3. Weergeven van de uitkomst van aggregaat functies(min, max, som, gem) die enkel gedaan kunnen worden op de eigenschappen: oppervlakte, perceeloppervlakte of bouwjaar, van een adres in de context van een plaats, postcode of straat. (Bijv: wat is het oudste gebouw in Apeldoorn?” of “Wat is de gemiddelde oppervlakte in mijn straat?”)
4. Weergeven van de uitkomst van filters, die gedaan kunnen worden op de eigenschappen; oppervlakte, perceeloppervlakte of bouwjaar, van een adres, in de context van een plaats, postcode of straat. (Bijv: “Hoeveel panden zijn er in mijn straat met een bouwjaar vóór 1850? ”)
5. Inzien van de ligging van een gebouw/adres op de kaart.
6. Inzien van de ligging van percelen op de kaart.

9.4 Chatflow

Met de chatflow worden alle mogelijke goedweer-paden van de chat beschreven. Dit betekent dus alleen de paden waarop alles goed gaat en waar nog geen informatie bij Loki bekend is. Uiteraard worden er zoveel mogelijk slechtweer-paden afgevangen om de gebruiker weer terug op het goedweer-pad te krijgen. De gebruiker heeft verder altijd de mogelijkheid om uit een bepaalde flow te stappen als dit gewenst is. Op deze manier komt de gebruiker nooit vast te zitten in een flow die alleen kan worden verlaten als deze volledig wordt afgegaan. Ook zal er bij het ophalen van informatie zoals bijvoorbeeld de postcode worden gekeken of dit al bekend is bij de chatbot zodat dezelfde informatie niet telkens opnieuw wordt uitgevraagd.

Voor een uitwerking van chatflows wordt verder verwezen naar het functioneel ontwerp dat als bijlage is toegevoegd.

9.5 Designs

Voor het maken van de designs is gebruik gemaakt van Figma (https://www.figma.com). Hiermee kunnen de designs gemakkelijk worden gedeeld en bekeken. Daarbij heeft figma de mogelijkheid om een klikbaar prototype te realiseren waardoor de functionaliteit van sommige componenten al getoond kan worden. In tabel 4 worden links beschreven die verwijzen naar de project pagina in figma en het klikbaar prototype.

Project pagina	https://www.figma.com/file/S9quh4wUPoaCSPE9OQqybg/Loki-1.0
Klikbaar prototype	https://www.figma.com/proto/S9quh4wUPoaCSPE9OQqybg/Loki-1.0

Tabel 4 Verwijzingen naar figma project pagina en prototype

Om requirement 9 te kunnen waarborgen is er een design gemaakt voor zowel desktop- als mobiele-schermen. Het desktop ontwerp wordt alleen gebruikt bij schermen waarbij zowel het chatscherm als de interactieve kaart naast elkaar op het scherm passen. Dit is mogelijk tot een scherm breedte van **1080 pixels**. Met dit punt kan er onderscheid gemaakt worden tussen de twee designs en kunnen alle scherm groottes op de juiste manier bediend worden.

Alle desktop designs zijn verder voorzien van een standaard achtergrond in de vorm van de Kadaster.nl landingspagina. Hiermee kan de aanwezigheid van de chatbot op een pagina als deze beter geïllustreerd worden. Verder zal alle functionaliteit eerst worden uitgelegd aan de hand van de desktop ontwerpen waarna wordt beschreven hoe deze ontwerpen zich vertalen naar een mobiel(kleiner) scherm.

Voor verdere uitleg bij elk design wordt weer verwezen naar het functioneel ontwerp.

9.6 Overige functionaliteit

Naast een nieuwe front-end zijn er overige functionaliteiten die aan Loki toegevoegd moeten worden. Hieronder vallen het toevoegen van chatbot management tooling, een integratie met whatsapp of telegram en het genereren van wandelroutes langs P.O.I's.

9.6.1 Chatbot management tooling

Om aan requirement 3 en 4 te kunnen voldoen moet er gebruik gemaakt worden van chatbot management tooling die de gesprekken van gebruikers opslaat en het mogelijk maakt voor ontwikkelaars om de gesprekken te bekijken zodat de bot geoptimaliseerd kan worden.

9.6.2 Whatsapp en telegram integratie

Requirement 11 beschrijft een integratie met Whatsapp of Telegram. Om hieraan te voldoen zal er eerst een integratie met Telegram opgezet worden voordat dit met Whatsapp wordt geprobeerd omdat Telegram makkelijker om gaat met het integreren van chatbots ten opzichte van Whatsapp. In telegram is het bijvoorbeeld mogelijk om knoppen in de chat te tonen zoals de suggestie knoppen. In Whatsapp kan dit niet waardoor hier een alternatief voor gezocht moet worden. Verder heeft dit requirement de laagste prioriteit van alle requirements waardoor het gunstiger is om de exacte invulling van deze functionaliteit in te vullen zodra de andere requirement gewaarborgd zijn.

9.6.3 Wandelroutes

De invulling van de wandelroute functionaliteit gaat er als volgt uit zien:

1. De gebruiker stuurt zijn locatie naar Loki, dit kan ten eerste in de vorm van een adres met een mogelijke uitbreiding in de vorm van coördinaten.
2. De gebruiker krijgt een wandelroute opgestuurd langs religieuze gebouwen(kerken, moskeeën, etc.) binnen een straal van 10 kilometer. De route wordt weergegeven op de kaart in de chat van Loki met een link naar de route in GoogleMaps.
3. De gebruiker loopt de route en geeft aan als er in de buurt van een van de P.O.I's gelopen wordt doormiddel van het sturen van een adres van waar de gebruiker zich bevind.
4. Loki kijkt welk POI het dichts bij het opgestuurde adres is en geeft informatie weer over dit POI.

De invulling van P.O.I's is in dit voorbeeld religieuze gebouwen maar het is een streven om deze invulling zo generiek mogelijk te maken zodat gebruikers zelf zouden kunnen kiezen langs welke punten ze willen lopen. Verder geeft de gebruiker zijn locatie aan Loki via adressen maar dit kunnen als mogelijke uitbreiding ook coördinaten. Dit slot is het van belang te beschrijven dat deze functionaliteit alleen wordt geïmplementeerd als al de voorgaande requirements van Loki naar wens zijn omdat de requirements waar deze functionaliteit over gaat, nummer 12 en 13, een lagere prioriteit hebben.

10 Implementatie

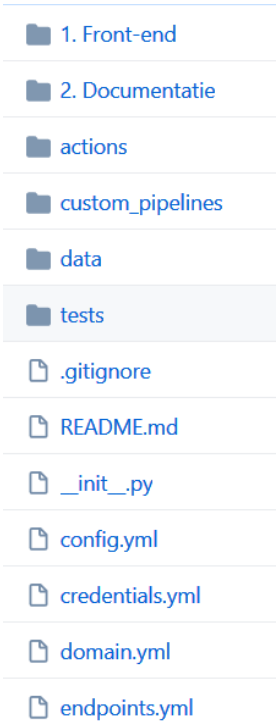
In dit hoofdstuk volgt het technisch document. Dit document wordt tevens als los bestand opgeleverd aan het Kadaster omdat het de inhoudelijke documentatie van de code betreft. Het kan gebruikt worden door andere ontwikkelaars om sneller een beeld van de code-base te krijgen. Verwijzingen naar libraries worden om deze reden dicht bij het onderwerp gezet om het voor volgende ontwikkelaars makkelijker te maken naar deze bronnen te navigeren.

10.1 Repository Indeling

De code wordt op de GitHub van de organisatie “Kadaster-IT” beschikbaar gesteld in een repository met de naam: “datascience-loki”. Op deze repository zijn drie delen van het project te vinden, namelijk:

- De front-end
- De backend
- De documentatie

Hierbij zitten alleen de front-end en documentatie in een eigen map omdat het voor de chatbot management tool genaamd “Rasa-X” vereist is om de Rasa project bestanden in de root van een repository te hebben. Om beter overzicht te kunnen bewaren zijn de namen van de front-end en documentatie mappen genummerd zodat deze altijd bovenaan de repository worden getoond.



Figuur 9 Repository indeling

10.2 Front-end

Voor de front-end is gebruik gemaakt van Angular. Er zijn slechts enkele wijzigingen gemaakt aan de bestandsstructuur van een standaard Angular project omdat dit overzichtelijker is. De bestandsstructuur wordt beschreven in het volgende hoofdstuk.

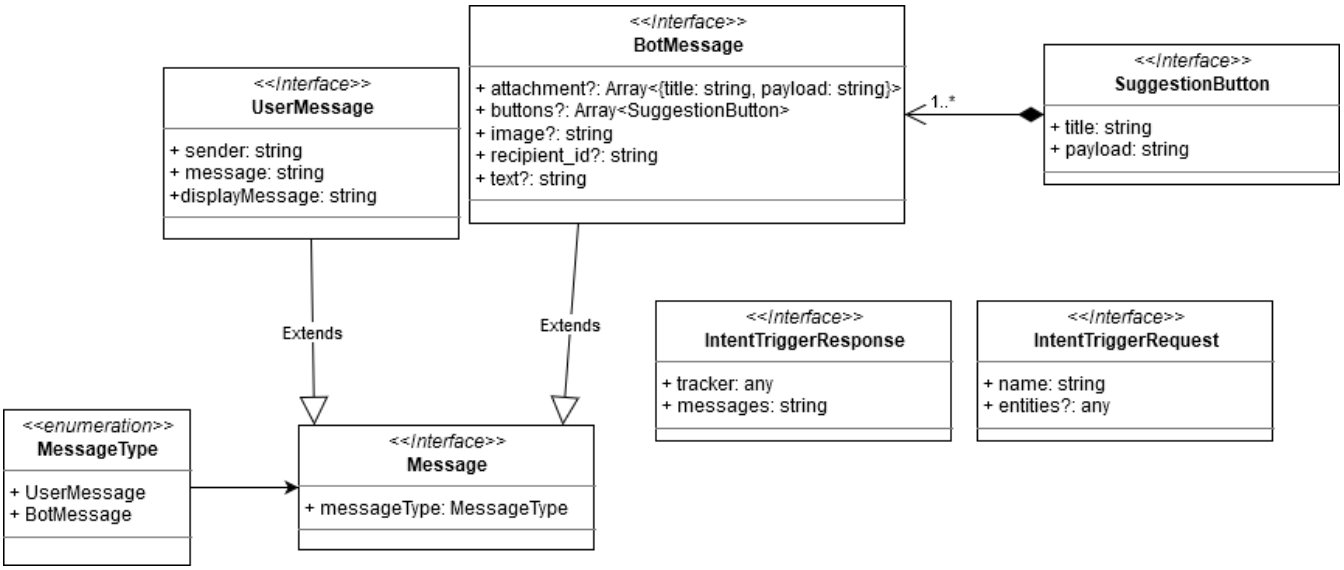
10.2.1 Bestandsstructuur

Map naam	Beschrijving
local_libs	Deze map bevat libraries die niet via npm toegevoegd konden worden.
src/app	Bevat de Angular componenten
src/assets	Bevat de assets van de app zoals bijvoorbeeld statische plaatjes
src/environments	Bevat de environment files die gebruikt worden bij het builden van de Angular app.
src/model	Bevat de model klassen.
src/scss	Bevat alle scss bestanden van de app.
src/services	Bevat bestanden waarin logica is beschreven waarvan het mooier was om ze los te halen van een component. Bijvoorbeeld: api-services of Angular-directives

Tabel 5 Bestandsstructuur Front-end

10.2.2 Model klassen

De model klassen bestaan uit typescript interfaces. Met deze interfaces worden invullingen van objecten types afgedwongen.



Figuur 10 Front-end klassendiagram

Een bijzonder aspect van dit klassen diagram is dat de “tracker” in “IntentTriggerResponse” van type “any”. Dit is gedaan omdat de “tracker” vanuit Rasa uit veel verschillende objecten bestaat die verder nooit in de front-end worden gebruikt. Waardoor het toevoegen van interfaces voor de “tracker” alleen maar complexiteit zou toevoegen aan de model klassen en geen voordelen zou bieden voor de verdere implementatie van de Front-end.

10.2.3 Componenten

De applicatie bestaat uit slecht 2 componenten namelijk:

1. App.component
2. Message.component

Het eerste component is het standaard root Angular component. Hierin is de floating-action-button en het chat-venster gedefinieerd. Het tweede component wordt in het root-component gebruikt als lijst. Deze lijst bestaat vervolgens uit alle berichten van de gebruiker of van Loki.

10.2.4 Implementatie keuzes en verduidelijkingen

Tijdens de realisatie zijn er verschillende implementatie keuzes gemaakt. De meest belangrijke worden hieronder toegelicht met daarbij verduidelijkingen van belangrijke stukken code.

10.2.4.1 Globale scss folder

Een standaard Angular project heeft voor elk component een eigen stylingsbestand. Omdat dit hergebruik moeilijker maakt en de stylings bestanden niet op één plek opslaat is ervoor gekozen om dit op een andere manier te doen.

In map “src/scss” bevind zich alle styling code van het volledige project. Deze stylings bestanden hebben verder een prefix bestaande uit een underscore(“_”). Dit betekent dat de inhoud van deze bestanden, tijdens het compilen, wordt gekopieerd op de plek van waar de bestanden worden geïmporteerd. In dit geval worden alle styles tot één bestand geaggregeerd in: “src/stylings.scss”. Hierdoor kan er ook gemakkelijk gebruik gemaakt worden van globale variabelen in combinatie met bootstrap.

10.2.4.2 Custom scrollbar

Om het scrollen in de chat mooier te maken is een custom scrollbar geïmplementeerd. Hiervoor is gebruik gemaakt van de “Simpelbar-Angular” library. Verder is voor het scrollen extra functionaliteit toegevoegd in de vorm van automatisch naar beneden scrollen zodra nieuwe berichten worden weergegeven zodat de gebruiker de nieuwste berichten altijd direct in het scherm te zijn krijgt.

- Simpelbar Library: https://www.npmjs.com/package/simplebar-angular

10.2.4.3 Directives

Dit hoofdstuk beschrijft alle geïmplementeerde Angular directives.

10.2.4.3.1 AppAutoSize

Omdat een html-textarea het niet mogelijk maakt automatisch van hoogte te veranderen naarmate er meer tekst wordt getypt is er een directive geschreven die deze logica implementeert. De naam van dit directive is “appAutoSize” en de code is grotendeels geïnspireerd vanuit een library genaamd “ngx-autosize”. Er is echter voor gekozen om de code van deze library zelf te implementeren in een directive omdat er aanpassingen aan gemaakt moesten worden om het scrollen optimaal te kunnen laten werken met de huidige toepassing.

- Ngx-autosize library: https://github.com/chrum/ngx-autosize

10.2.4.3.2 AppAutoFocus

Dit directive maakt het mogelijk de focus op te vragen zodra, het component waarop het directive is toegepast, is weergeven. Hierdoor kan de gebruiker direct typen in het tekstveld van de chat zodra het chat-venster wordt geopend. De gebruiker hoeft dus niet op het veld te klikken.

10.2.4.4 Font-awsome icons

Om iconen weer te kunnen geven wordt gebruik gemaakt van het Angular component van Font-awsome. Font-awsome beheert veel icoontjes die gemakkelijk kunnen worden ingeladen als SVG binnen Angular. Om een icoon toe te voegen moet het worden geimporteerd in "src/app/app.module.ts". Vervolgens moet het geïmporteerde object worden toegevoegd in de "library.addIcons()" methode onderaan de "app.module".

- Font-awsome Angular component: https://www.npmjs.com/package/@fortawesome/angular-fontawesome
- Font-awsome Icoon lijst: https://fontawesome.com/icons?d=gallery

10.2.4.5 Chat service

In de map "src/services" is een bestand genaamd "chat.service.ts" te vinden. Dit bestand bevat alle logica voor het communiceren met de Rasa backend en beheert de lijst met berichten zodat ieder component hier gemakkelijk bij kan. De service heeft vier belangrijke functionaliteiten:

1. Het genereren van een "conversationID" door middel van de uuid library.
2. Het verzenden van een trigger_intent request.
3. Het verzenden van gebruikers berichten.
4. Het afhandelen van Bot-responses.

Omdat de front-end aangeeft wanneer de gebruiker een gesprek begint moet ook de front-end zorgen voor een unieke identificatie string. Dit wordt gedaan middels de uuid library.

- UUID library: https://www.npmjs.com/package/uuid

Verder beschikt Rasa over een endpoint waarmee intents geforceerd kunnen worden aangeroepen zonder dat dit door de NLU wordt gehaald. Dit is handig voor bijvoorbeeld het starten van een gesprek in de front-end. De chatservice implementeert een methode, triggerIntent(), waaraan de naam van het intent en mogelijke entities meegegeven kunnen worden. Dit wordt vervolgens verzonden naar Rasa via de trigger_intent endpoint. Het resultaat van dit intent wordt in de vierde functionaliteit van deze service afgehandeld.

De derde functionaliteit van de chatservice is het verzenden van een bericht van de gebruiker. De methode "sendUserMessage()" kan worden aangeroepen met als argument een UserMessage object. De inhoud van het UserMessage object wordt vervolgens verzonden naar de Rasa backend waarna een response wordt teruggegeven op basis van wat de backend heeft voorspeld.

De response wordt vervolgens afgehandeld door de vierde functionaliteit. Deze functionaliteit kijkt of een van de berichten die de chatbot stuurt een "custom" object heeft. Dit houd in dat er bepaalde informatie wordt gestuurd die bijvoorbeeld een locatie op de kaart aangeeft. Rasa stuurt deze custom informatie niet mee met het bericht waarbij dit in de backend wordt aangegeven maar maakt hier een apart bericht voor aan. Om ervoor te zorgen dat de front-end vervolgens geen leeg bericht zonder tekst toont, moet het bericht waarbij de extra informatie wordt meegegeven in de custom tag worden toegevoegd aan het laatst gestuurde bericht met tekst. Dit is wat er in de functie: "_handleBotResponse()" wordt gedaan. Tevens vult deze functie ook de type van het bericht met "BotMessage".

10.2.4.6 Tekst formateringen

Soms is het mooier als het antwoord van Loki een enter of witregel bevat. Om dit mogelijk te maken kan er in het antwoord van Loki de volgende string worden geplaatst

- [br]

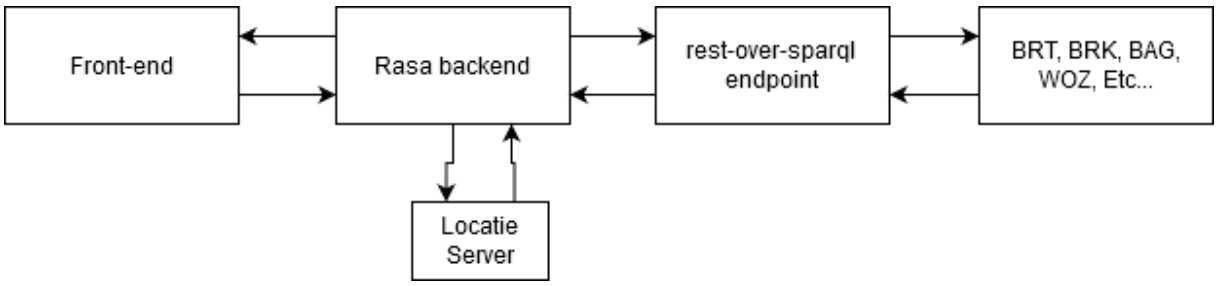
De front-end zal voor elk bericht van Loki het bericht splitten op deze string. Waarna elke split in een aparte paragraaf tag( ) wordt geplaatst. Hierdoor zal de tekst na “[br]” altijd op een nieuwe regel beginnen.

10.2.5 Local_libs map

In de bestandsstructuur is een map genaamd “local_libs” te vinden. Deze map bevat de code voor het Generieke Geo Component(GGC) van het Kadaster. Met dit component kunnen gemakkelijk kaarten worden weergeven in de front-end. Dit component is niet middels NPM toegevoegd aan de front-end omdat hier geen toegang tot was.

10.3 Rasa backend

De backend is gemaakt door middel van het Rasa chatbot framework. In feite is deze backend een soort tussen laag die middels natuurlijke taal tot een API-call komt waaraan bepaalde paramaters worden toegevoegd. Het resultaat van deze call wordt vervolgens weer omgezet naar natuurlijke taal of bijvoorbeeld een locatie op de kaart. In het figuur hieronder is weergegeven hoe het technische deel van de applicatie is ontwerpen en op welke manier het project met bestaande systemen verbind.



Figuur 11 Loki 1.0 Architectuur Ontwerp

Te zien is dat de Rasa-backend niet direct met de bron communiceert. Dit wordt gedaan door de “rest-over-sparql” endpoint. Dit endpoint is een soort service die voor verschillende url’s een statisch gedefinieerde SPARQL query doet, gebruik makend van de knowledge graph. Er kunnen waardes aan deze query worden meegegeven door deze waardes als parameters in de url te definiëren, zoals hieronder in een voorbeeld weergegeven.

https://api.labs.kadaster.nl/queries/kadaster/loki-rest-1/run?postcode=7417DH&huisnummer=75

Met het gebruik van dit Endpoint kan de implementatie en onderhoud van de queries uit handen worden gegeven. Hierdoor is het in de Rasa backend niet nodig om SPARQL queries op te bouwen en kan er simpelweg een request naar een endpoint worden gedaan met de bepaalde parameters.

10.3.1 Bestandsstructuur

Map naam	Beschrijving
actions	Bevat alle action bestanden of bestanden die deze actions ondersteunen
actions/resources	Bevat resources waar de actions gebruik van maken. Bijvoorbeeld het CSV bestand voor intent-mappings.
actions/services	Bestanden met code die op zichzelf een stuk logica bevatten waarvan het mooier was dit los te halen van een action file om bijvoorbeeld hergebruik makkelijker te maken.
custom_pipelines	Bevat eigengemaakte pipelines die worden gebruikt in de “config.yml”
data	Bevat alle data voor het trainen van het model.
data/lookup_tables	Bevat de lookup tables txt-bestanden die gebruikt worden in de NLU.
models	Bevat de getrainde modellen.
results	Map waarin Rasa de resultaten van het commando “rasa test” neer zet.
tests	Bevat het bestand met alle tests.

Tabel 6 Bestandsstructuur Rasa_backend

10.3.2 Intents

In de tabel hieronder worden alle geïmplementeerde intents weergeven met een beschrijving.

Intent naam	Beschrijving
introduce	Intent zonder NLU data, wordt alleen gebruikt voor het starten van een gesprek middels de trigger_intent endpoint.
greet	Een groet.
bye	Een afsluitende groet.
thank	Een bedankje.
affirm	Een bevestiging
deny	Een afwijzing
faq_or_chitchat	Een hoofd-intent met sub-intents voor veel gestelde vragen of chitchat(kletsen)
explain_slot	Een slot uitleggen. Wordt gebruikt wanneer bepaalde informatie wordt uitgevraagd en de gebruiker wil weten wat deze informatie inhoud of waarom het wordt gevraagd.
get_address_property	Het opvragen van adres eigenschappen
inform	Intent voor wanneer de gebruiker enkel informatie opgeeft.
out_of_scope	Intent voor vragen die niet in de scope van de bot zitten en waarbij het mooier is als dit aan de gebruiker wordt gemeld.
trigger_rephrase	Intent zonder NLU data, wordt alleen gebruikt door de 2stageFallbackPolicy wanneer de gebruiker op de knop “Iets anders” klikt.

Tabel 7 Intents

10.3.3 Entities

In de volgende tabel worden de entities beschreven.

Intent naam	Beschrijving
property	De eigenschap die de gebruiker over een adres wil weten.
postalcode	De postcode van een adres.
housenumberdesignation	De nummeraanduiding van een adres. Bestaat uit een nummer met mogelijk een huisletter en huisnummertoevoegingen.

Tabel 8 Entities

Alleen de entitie voor een postcode komt niet voor in de NLU data. Dit entitie wordt met een custom entitie extractor opgehaald genaamd “RegexEntitieExtractor”. In de “Implementatie keuzes” wordt hier verder op in gegaan.

10.3.4 Actions

De actions die worden geïmplementeerd worden hieronder weergegeven

Action naam	Beschrijving
respond_faq_or_chitchat	Action met “respond_” prefix om aan te geven dat deze action door de response selector wordt afgehandeld. Beschikt daarom ook niet over een custom action class, deze logica wordt standaard door Rasa geleverd.
action_greet	Groet de gebruiker
action_introduce	Stuurt introductie berichten naar de gebruiker
action_ask_use_same_address	vraagt of de gebruiker het opgeslagen adres opnieuw wil gebruiken
action_reset_address	Verwijderd het opgeslagen adres zodat een nieuw adres wordt uitgevraagd
action_default_ask_affirmation	Eerste fallback van de 2stageFallbackPolicy. vraagt welk intent de gebruiker bedoelde uit de best gescoorde intents.
action_default_fallback	Tweede fallback van de 2stageFallbackPolicy en tevens de fallback voor wanneer er geen opvolgende action bestond met een score boven de, in de config.yml aangegeven “core_treshold”.

Tabel 9 Actions

Enkele van deze actions zijn statisch gekoppeld aan bepaalde intents. Dit betekent dat wanneer dit intent wordt geclassificeerd het gekoppelde action altijd wordt aangeroepen. De actions met een statische koppeling zijn:

- “action_introduce” aan de “introduce” intent.
- “action_greet” aan de “greet” intent.

10.3.5 Slots

Slots zijn velden binnen Rasa waarin informatie van het gesprek kan worden opgeslagen zodat dit later in het gesprek kan worden gebruikt. De slots die worden geïmplementeerd worden in de tabel hieronder weergeven

Naam	Type	Beschrijving
property	unfeaturized	De adres eigenschap die een gebruiker wil ophalen
postalcode	tekst	De postcode
housenumberdesignation	unfeaturized	De nummeraanduiding
housenumber	tekst	Het huisnummer
houseletter	unfeaturized	De huisletter
housenumberaddition	unfeaturized	De huisnummertoevoegingen
requested_slot	categorical (property, postalcode, housenumberdesignation)	Standaard slot van rasa. Dit wordt gevuld met de naam van het slot dat wordt uitgevraagd.

Tabel 10 Slots

De meeste slots hebben een unfeaturized type omdat het niet relevant is voor het vervolg van het gesprek dat deze slots een waarde hebben. Dit is wel relevant voor de slots met het type: “tekst”. Dit geeft alleen aan of het slot een waarde heeft of niet. Voor slots waar ook de exacte waarde relevant is wordt het type: “categorical” toegepast met tussen haakjes de mogelijke waardes die dit slot kan hebben.

10.3.6 Forms

De chatbot maakt gebruik van form actions, dit is een speciaal soort actie die pas kan worden aangeroepen als alle benodigde data beschikbaar is. Mocht dit niet het geval zijn, dan wordt dit aan de gebruiker gevraagd totdat alle informatie valide is ingevuld.

Naam	Benodigde informatie	Beschrijving
address_property_form	De postcode De nummeraanduiding De eigenschap waarvan een waarde opgehaald moet worden	Dit form wordt gebruikt wanneer de gebruiker om een adres eigenschap vraagt.

Tabel 11 Forms

10.3.7 Testen

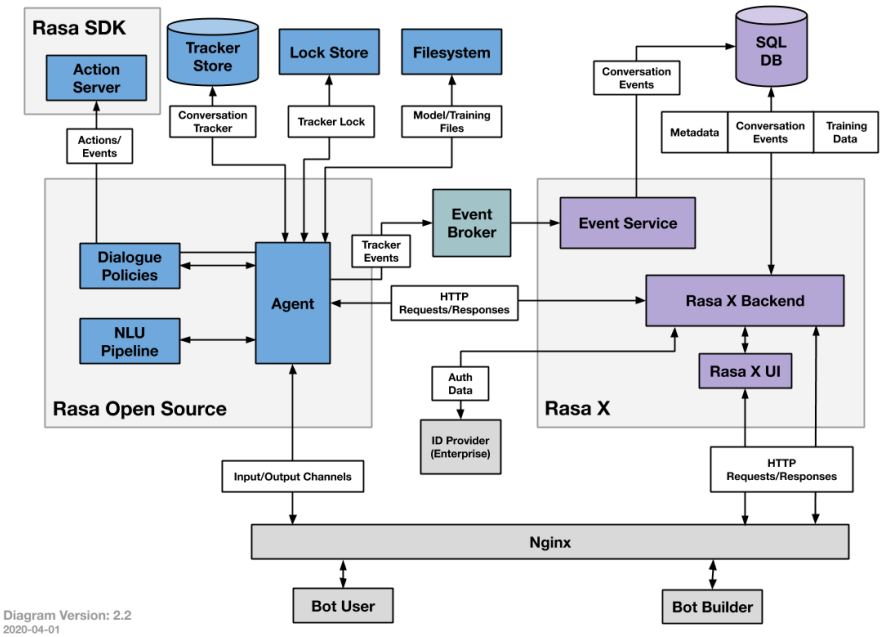
Rasa maakt het mogelijk om de chatbot geautomatiseerd te kunnen testen. De tests worden in de map “tests” gedefinieerd in het bestand “conversation_tests.md”

Door het commando “rasa test” uit te voeren zullen alle tests worden aangeroepen. De huidige tests beschrijven alleen paden die ook zijn gedefinieerd in de stories. Hierdoor kan bij elke nieuwe toevoeging snel worden getest of alle stories nog naar behoren werken. Ook bij wijzigingen van de pipeline of policies in het config.yml bestand, is het belangrijk om het test commando van Rasa uit te voeren. Op deze manier zal de kwaliteit van het product beter bewaard blijven.

10.3.8 Rasa X

De chatbot maakt gebruik van Rasa X als chatbot management tooling. Rasa X helpt een ontwikkelaar niet alleen met het daadwerkelijk deployen van de chatbot maar maakt het ook makkelijker om gesprekken in te zien en mogelijk aanpassingen te maken aan de chatbot middels een Userinterface.

Rasa X staat verder los van het rasa project en dient alleen ter ondersteuning ervan ingezet te worden op een server. Dit kan worden gedaan met een docker-compose bestand waarmee verschillende docker containers worden geactiveerd. Deze containers worden hieronder weergegeven in een architectuur plaat en beschreven in een tabel.



Figuur 12 Rasa-X Architectuur diagram https://rasa.com/docs/rasa-x/installation-and-setup/architecture/

Naam	Beschrijving	Component in diagram
rasa-x	Draait de rasa-x docker image. Dit is de daadwerkelijke chatbot management tool met UI.	Rasa-X
rasa-production	Draait een standaard rasa docker image. En wordt gebruikt om de chatbot beschikbaar te maken.	Rasa Open Source
rasa-worker	Draait een standaard rasa docker image. Hiermee kunnen models worden getrained terwijl de chatbot blijft draaien op de production container.	Rasa Open Source
app	Draait de action server docker image.	Action server
db	Draait een postgresql docker image waarmee de database benaderd kan worden. In deze database worden de gesprekken opgeslagen	SQL DB & Tracker Store
rabbit	Draait een rabbitmq docker image. Hiermee worden events in een gesprek doorgestuurd van rasa-production naar Rasa-X	Event Broker
nginx	Draait de nginx docker image van Rasa. Hiermee worden requests die op port 80 van de server worden gedaan doorgestuurd naar de juiste container	Nginx
redis	Draait de redis docker image. Houd bij welke rasa open source node aan welk gesprek werkt. Zonder deze container zijn er geen gesprekken mogelijk.	Lock Store

Tabel 12 Rasa X docker containers

Zodra de containers online zijn kan er naar het IP-adres van de server waarop deze containers draaien worden genavigeerd met een browser(Firefox wordt niet ondersteund), en kan worden ingelogd in Rasa-X. Het wachtwoord dat hierbij opgegeven moet worden is in de README.md van het project te vinden. Hier kan vervolgens verbinding worden gemaakt met de repository van het rasa project zodat Rasa-X altijd de laatste versie kan draaien. Zoals eerder beschreven is het hierbij noodzakelijk dat de Rasa project bestanden in de root van de repository staan. Anders kan Rasa-X de bestanden niet uitlezen.

Er zijn verder een aantal aanpassingen gedaan aan het standaard docker-compose bestand van Rasa. Deze aanpassingen zijn gedaan in een “docker-compose.override.yml” bestand zodat bij updates, van het docker-compose bestand van Rasa-X, de aanpassingen niet verloren gaan.

Ten eerste is de image van de app container vervangen met een image waarop de actions van Loki beschikbaar zijn. Als deze actions geüpdatet moeten worden is het dus noodzakelijk dat er een nieuwe docker image wordt gemaakt die vervolgens moet worden gepushed naar docker-hub. Waarna de server waarop de containers draaien deze image ook kan updaten.

De volgende aanpassing is het beschikbaar maken van de production container waarop de chatbot live staat. Dit is gedaan door de port van deze container gelijk te stellen aan port 5005 van de server waarop de docker containers worden gedraaid. Dit is nodig omdat Rasa-X geen mogelijkheid biedt om middels haar endpoints de “trigger_intent” endpoint aan te roepen van Rasa open source.

De derde aanpassing die is gemaakt aan het standaard docker-compose bestand is het vervangen van de image, die op de rasa-production en rasa-worker worden gedefinieerd, met een Dockerfile. Hiermee is het mogelijk om tijdens het bouwen van de container commando’s uit te voeren. Dit was noodzakelijk omdat de implementatie van de chatbot gebruik maakt van het Spacy language model, wat niet standaard is geïnstalleerd op de rasa docker image. In de docker file wordt dus nog steeds hetzelfde image aangegeven waarna commando’s worden uitgevoerd die Spacy op de container instaleren.

Deze Dockerfile is verder ook gebruikt om de RegexEntitieExtractor op de containers beschikbaar te maken. In Rasa-X is het namelijk alleen met een omweg mogelijk om custom pipelines te gebruiken. Het bestand van de custom pipeline moet op de server, waar Rasa-X draait, worden geplaatst waarna deze map wordt toegevoegd als volume aan de rasa-production en rasa-worker container. Vervolgens moet in de Dockerfile, de locatie van de custom pipeline, aan het “PYTHONPATH” worden toegevoegd zodat tijdens het draaien van de container rasa de python module kan vinden. Het nadeel aan deze implementatie is dat bij updates van een custom pipeline, deze pipeline opnieuw op de server van Rasa-X moet worden geplaatst. Dit is niet mogelijk met git.

De laatste aanpassing die wordt gedaan aan het standaard docker-compose.yml van Rasa-X is het verwijderen van de duckling service. Dit wordt niet in docker-compose.override.yml gedaan omdat het binnen docker niet mogelijk is om in een docker-override bestand aan te geven dat een container die in de docker-compose.yml staat gedefinieerd niet moet worden gebruikt. Het is hierbij dus belangrijk dat bij updates van het docker.compose.yml bestand, dat van Rasa-X wordt verkregen, de duckling service weer wordt verwijderd. Tenzij hier in het rasa project gebruikt van wordt gemaakt, wat op dit moment niet het geval is.

10.3.8.1 Aanpassingen doorvoeren naar production

Bij het updaten van het project zijn er dus voor verschillende delen andere stappen vereist.

- Updaten van NLU, Stories, config en domain.
    - Dit kan worden gedaan middels de UI van Rasa-X en vereist verder geen handelingen op de server waar Rasa-X draait. Door de aanpassingen op de master branch te zetten kan Rasa-X ze inlezen. Daarna Kan er op de “train” knop gedrukt worden. Na een aantal minuten is het model in het “Models” tab te vinden waar het als “active” getagged kan worden. De updates staan nu live.
- Updaten van Custom pipeline
    - Een custom pipeline wordt in Rasa-X niet ondersteund. Daarom moeten deze bestanden lokaal op de server gezet worden vanuit waar Rasa-X draait. Bij een update moet dit bestand dus worden vervangen waarna het direct is geüpdatet.
- Updaten van Actions server
    - Het doorvoeren van aanpassingen aan de actions server kan gedaan worden door lokaal een image te builden en deze image te pushen naar een container registry zoals bijvoorbeeld docker-hub. Daarna kan deze Image op de server waar Rasa-X draait worden opgehaald.

10.3.9 Implementatie keuzes en verduidelijkingen

Ook bij de backend zijn er tijdens de realisatie verschillende implementatie keuzes gemaakt. De meest belangrijke worden hieronder toegelicht met daarbij verduidelijkingen van belangrijke stukken code.

10.3.9.1 Opsplitsing van actions

Voor het implementeren van actions heeft Rasa in de standaard implementatie een enkele actions.py bestand waarin alle actions worden gedefinieerd. Om de structuur van het project beter te bewaren is er voor gekozen dit niet te doen en elke action in een apart bestand te implementeren. Deze bestanden hebben de naam van de action zonder de “action_” prefix.

10.3.9.2 Retrieval actions

Met retrieval actions maakt Rasa het mogelijk om simpele gesprekken op een efficiëntere manier te declareren. Een simpel gesprek is bijvoorbeeld wanneer de gebruiker een vraag stelt waar altijd hetzelfde antwoord op gegeven kan worden, zoals een veel gestelde vraag(faq) of geklets(chitchat). Met retrieval actions hoeft er slechts één story geschreven te worden voor het afhandelen van meerdere intents.

Het intent met de naam “faq_or_chitchat” hoort bij zo’n retrieval action. Door in de NLU een intent de volgende prefix te geven: “faq_or_chitchat/”, gaat Rasa opzoek naar responses voor dit intent. Deze responses worden gedeclareerd in “data/nlu/faq_or_chitchat_responses.md”. Hier worden de sub-intents gematched om tot het juiste response te komen. Het voordeel van het gebruik van retrieval actions is dat bij bijvoorbeeld een interruptie van een form er slecht één story geschreven hoeft te worden die deze interruptie afhandelt voor alle mogelijke intents die onder het hoofd-intent vallen.

10.3.9.3 RegexEntitieExtractor

Om het makkelijker te maken entities met een regex te extracten is een custom entitie extractor geïmplementeerd genaamd “RegexEntitieExtractor”. Deze extractor is te vinden in de map “custom_pipelines” en wordt gedeclareerd in de config.yml onder “pipeline”. De code voor de extractor is grotendeels geïnspireerd uit een artikel van Naoko Reeves.

Er is slechts een kleine aanpassing aan gedaan zodat alleen, in de NLU gedeclareerde regexen, met de prefix: “regex_extractor/” worden gebruikt voor het extracten van entities. verder wordt de tekst die na de “/” komt gebruikt als naam voor de entitie. Voor bijvoorbeeld: “regex_extractor/postalcode” wordt “postalcode” als entitie naam gebruikt.

Het is verder belangrijk dat de entities die door de regexExtractor worden opgehaald niet middels de standaard entitie declarering (bijv: [1234AB](postalcode)) worden aangegeven in de NLU omdat anders meerdere entitie extractors mogelijk hetzelfde entitie zullen extracten. Dit betekent nog wel dat de entities in de NLU voorkomen om de intent classificatie te kunnen trainen, ze worden enkel niet meer aangemerkt als entitie omdat dit dus door de RegexEntitieExtractor wordt gedaan.

- Artikel van Naoko Reeves: https://medium.com/@naoko.reeves/rasa-regex-entity-extraction-317f047b28b6

10.3.9.4 TwoStagefallback

De chatbot maakt gebruik van de twoStageFallbackPolicy van Rasa. Dit betekent dat wanneer de intent classificatie van Rasa onder een bepaalde nlu_threshold komt, rasa zal vragen welke van de 2 best gescoorde intents de gebruiker bedoelde of dat de gebruiker iets anders bedoelde. Als de gebruiker iets anders bedoelde dan de opties die Rasa voorspelde zal de gebruiker worden gevraagd om zijn bericht anders te formuleren. Als de intent classificatie score van het herformuleerde bericht opnieuw onder een bepaalde nlu_threshold valt, zal Rasa opnieuw vragen of het bericht een van de 2 best gescoorde intents betrof. Zodra de gebruiker hierna weer aangeeft dat het geen dat werd gezegd niet tussen de 2 best gescoorde intents staat, zal rasa “opgeven” en in dit geval de gebruiker vragen om de chat te herstarten. Echter zou dit ook een mogelijk punt kunnen zijn om het gesprek over te geven aan een echt persoon.

Zoals in de beschrijving van “action_default_fallback” verder al werd beschreven is er nog een mogelijkheid om direct tot de laatste fallback te komen. Dit is wanneer de intent classificatie score van rasa boven de bepaalde nlu_threshold is, en Rasa dus weet wat de gebruiker zegt, maar hier geen actie aan kan classificeren boven de bepaalde core_threshold.

De huidige thresholds staan aangegeven in de config.yml onder policies en hebben de volgende waardes:

- core_threshold = 0.3
- nlu_threshold = 0.8

10.3.9.5 Kleine letters

Om het aantal mogelijkheden van wat de gebruiker kan zeggen te reduceren wordt alle tekst in de NLU in kleine letters gedeclareerd. Ook de front-end stuurt op de achtergrond alles in kleine letters door onderscheid te maken tussen de “displayMessage” en het standaard “message” attribuut van Rasa. Op deze manier worden hoofdletter die de gebruiker typt nog wel in de chat getoond maar kan Rasa op de achtergrond altijd van kleine letters gebruik maken.

Met het gebruik van alleen kleine letters is het, mocht dit nodig zijn, niet meer mogelijk om bepaalde entitie extractors van het talen model van Spacy volledig te benutten omdat dit het enige component is dat onderscheid maakt tussen hoofd- en kleine letters.

10.3.9.6 PDOK Locatieserver

Omdat een huisnummer niet alleen uit een cijfer kan bestaan maar ook letters of andere toevoegingen, wordt er gebruik gemaakt van de machine learning van Rasa om alles wat met een huisnummer te maken heeft te extracten als “housenumberdesignation”. In combinatie met de PDOK locatieserver wordt deze “housenumberdesignation” vervolgens ontleed tot de relevante delen van het huisnummer en de eventuele huisletter en huisnummertoevoeging.

Zodra Rasa een nummeraanduiding van de gebruiker ontvangt wordt dit samen met de postcode naar de “suggest-service” endpoint van de locatieserver gestuurd. Dit is een soort zoekfunctie die het meest relevante object als eerste in een lijst teruggeeft. Vervolgens wordt, met de “lookup-service” endpoint, meer informatie van het meest relevante object opgehaald. In deze extra informatie bevinden zich attributen genaamd “huisnummer”, “huisletter” en “huisnummertoevoeging”. Deze attributen worden uitgelezen en opgeslagen in Rasa doormiddel van slots en daarbij worden de attributen gebruikt bij het ophalen van data uit de knowledge graph

- PDOK-locatie server: https://github.com/PDOK/locatieserver/wiki/API-Locatieserver

10.3.9.7 Housenumberdesignation slot

Het is in Rasa niet mogelijk om de waarde van een entitie in meerdere slots op te slaan. Dit betekent dat, voor de entitie die een nummeraanduiding beschrijft, een apart slot moet worden gedeclareerd terwijl deze nummeraanduiding altijd wordt ontleed in een huisnummer en eventuele huisletter en huisnummertoevoeging. De informatie in het slot met de naam “housenumberdesignation” wordt verder nooit meer in het gesprek gebruikt.

10.3.9.8 Suggestieknoppen

Suggestieknoppen worden gebruikt door de twoStageFallbackPolicy of kunnen worden gedeclareerd bij een response in de domain.yml. Een suggestie knop bestaat uit een “title” en een “payload”. De “title” is wat als tekst wordt weergeven in de front-end en de “payload” is wat wordt verstuurt naar Rasa. Deze payload bestaat vaak uit een “/” gevolgd door een intent naam en json met de entities. Het aanroepen van de “get_address_property” intent met “oppervlakte” als “property” entitie ziet er dan als volgt uit:

- /get_address_property{"property":"bouwjaar"}

10.3.9.9 Entities negeren

Rasa zal bij het voorspellen van de volgende actie altijd kijken naar welke entities zijn gevonden in het geclassificeerde intent. In sommige gevallen is het helemaal niet relevant welke entities gevonden zijn en moet er bij een bepaald intent altijd een bepaalde actie worden uitgevoerd ongeacht welke entities zijn gevonden. Dit wordt aangegeven in het domain.yml bestand met de volgende syntax:

- { use_entities: [] }

Een voorbeeld van waar dit gebruikt wordt is bij het intent met de naam “get_address_property”. Dit intent zal altijd een Form gebruiken waarin eerst alle informatie wordt verzameld die nog niet bekend is om daarna deze informatie te gebruiken voor het ophalen van data. Het is hierbij niet relevant welke informatie er in het initiële intent, “get_address_property”, heeft gezeten, dit wordt immers afgehandeld door de form-action. Het negeren van entities in intents reduceert het aantal verschillende mogelijkheden dat een intent kan hebben waardoor stories versimpeld kunnen worden.

11 Conclusie

De afstudeerstage heeft geresulteerd in een ander product dan wat er in de opdracht omschrijving is beschreven. Uiteindelijk zijn er namelijk drie belangrijke delen naar voren gekomen:

1. Een onderzoek naar verschillende mogelijkheden voor het maken van een chatbot.
2. Een front-end voor een chatbot.
3. Een backend van een chatbot die gebruik maakt van het Rasa framework.

Het onderzoek heeft een goede basis gelegd voor de kennis op het gebied van een chatbot, en heeft beschreven welke mogelijkheden er op dit moment zijn voor de ontwikkeling hiervan. Verder heeft het onderzoek duidelijk gemaakt dat het Rasa framework voor deze toepassing de beste keuze is omdat het meer flexibiliteit biedt aan een ontwikkelaar. Ook is er naar voren gekomen hoe partijen als Google en IBM niet altijd de beste op het gebied van machine learning hoeven te zijn ondanks dat ze beschikken over meer data.

Verder beschikken de front-end en backend niet over alle vooraf opgestelde must-requirements omdat er tijdens de ontwikkeling voor is gekozen om de al bestaande Rasa backend opnieuw te implementeren. Ondanks deze keuze is het toch gelukt om een goede basis van een chatbot neer te zetten waarbij de volgende requirements behaald zijn:

- De chatbot kan enkel vragen beantwoorden op basis van data in de kadaster knowledge graph.
- De chatbot heeft een startscherm waarop een introductie van de chatbot is weergegeven met suggesties van wat de gebruiker kan zeggen.
- Gesprekken van gebruikers kunnen worden bekeken om het mogelijk te maken de bot, bij fouten, te optimaliseren.
- De chatbot maakt gebruik van chatbot management tooling
- De chatbot geeft, gedurende het gesprek, suggesties van wat de gebruiker kan zeggen in de vorm van knoppen.
- De chatbot kan locaties weergeven op een kaart in het chatvenster.
- De gebruiker kan een weergegeven locatie in het chatvenster, nader bekijken op een interactieve kaart.
- De chatbot refereert, middels een link, altijd naar de bron van de data waarop een antwoord is gebaseerd.
- De chatbot heeft een responsive front-end

Verder is de eerste functionaliteit in de backend geïmplementeerd wat het weergeven van eigenschappen van een adres betrof.

12 Aanbevelingen

Een software product is natuurlijk nooit helemaal af. Zo is er ook bij Loki 1.0 ruimte voor verbeteringen of toevoegingen. Deze punten worden hieronder weergeven en daarna toegelicht.

1. Functionaliteiten toevoegen.
2. Laten overnemen door persoon als de chatbot er niet uit komt.
3. Veel gestelde vragen van het Kadaster toevoegen.
4. Toevoegen van GGC met NPM.
5. Laad icoontje tonen.
6. Docker-hub koppelen aan de repository.

De keus voor het opnieuw beginnen met de Rasa backend heeft ervoor gezorgd dat niet alle oude functionaliteiten ook werken in de nieuwe implementatie. Een belangrijke aanbeveling zou zijn om deze functionaliteiten toe te voegen zodat de gebruiker nog beter bediend kan worden.

In de twoStageFallbackPolicy zit een mogelijkheid om het gesprek over te geven aan een echt persoon. Als Loki er in de huidige implementatie niet uit komt zal de gebruiker worden gevraagd om de chat te herstarten. Dit kan worden vervangen met een hand-off naar een echt persoon zodat de gebruiker ten alle tijden geholpen kan worden met zijn probleem.

Loki biedt de ultieme oplossing voor het afhandelen van veel gestelde vragen aan het Kadaster. Dit is niet alleen handiger voor een gebruiker die op deze manier niet meer door de kadaster website hoeft te zoeken. Maar kan ook dienen als oplossing voor medewerkers die vaak dezelfde vragen krijgen. De retrieval actions van Rasa bieden kunnen hier zeer gemakkelijk voor gebruikt worden. Er hoeft voor het toevoegen van vraag- en antwoorden dan geen enkele regel code geschreven te worden.

Zoals beschreven eerder beschreven wordt het Generieke Geo Component van het Kadaster toegevoegd door middel van een statische kopie in de “local_libs” map. Uiteraard is het voor bijvoorbeeld updates van dit component gunstiger als er toegang is tot de packages van het Kadaster in NPM.

Punt vijf beschrijft het toevoegen van een laad icoon. Dit houd in dat bij het sturen van een bericht van de gebruiker een laad-animatie wordt getoond in de chat totdat er een reactie van de chatbot binnen is. Hiermee is het voor de gebruiker duidelijk dat er op een reactie van de chatbot wordt gewacht en dat Loki niet is vastgelopen.

Het laatste punt heeft te maken met hoe Rasa-X de actions van de chatbot toevoegt. Dit wordt namelijk gedaan met een docker image om updates van deze image makkelijker te maken wordt aangeraden om de image, die op docker-hub beschikbaar is, te koppelen aan de GitHub repository zodat er een nieuwe image beschikbaar wordt gesteld zodra er een update heeft plaats gevonden in de repository

13 Reflectie

Aan het begin van de afstudeerstage was het erg lastig om goed in kaart te brengen wat de opdracht was omdat ik natuurlijk wel de beroeps activiteiten op een juist niveau moest kunnen aantonen. Ondanks het feit dat de opdracht uiteindelijk anders heeft uitgepakt denk ik nog wel dat ik de beroepsactiviteiten “analyseren” “ontwerpen” en “realiseren” op niveau 3 heb aangetoond. Het vangnet is komen te vervalen omdat het opnieuw bouwen van een Rasa backend mij voldoende ruimte heeft geboden om “realiseren” op niveau 3 te kunnen laten zien. Echter vind ik het wel erg jammer dat ik niet de tijd heb gehad om alle beoogde functionaliteiten toe te voegen. Ik ben erg lang bezig geweest met de onderzoeks en ontwerp fase. En het koste veel tijd om het Rasa framework goed te begrijpen. In Rasa wil je namelijk zoveel mogelijk aan de machine learning overlaten, zelfs als het soms makkelijker was om oplossingen in de actions te programmeren. Dit was iets wat ik nooit wou doen omdat dit ook een van de redenen was om opnieuw te beginnen met de backend.

Ik heb verder wel onwijs veel geleerd over chatbots en de werking ervan en ben erg benieuwd naar wat de toekomst van chatbots te bieden heeft. Verder ben ik veel te weten gekomen over het Kadaster en vooral de data die ze (vaak gratis) beschikbaar stellen.

13.1 STARR-methode reflecties

De meest belangrijke gebeurtenissen in de afstudeerstage zijn de keuze voor het opnieuw beginnen met de Rasa backend. En de corona-crisis. Daarom wil ik hierop reflecteren middels de STARR-methode[25].

13.1.1 Nieuwe rasa Backend

13.1.1.1 Situatie

Na twee weken werken aan de front-end werd het tijd de beoogde aanpassingen te maken in de backend van de experimentele versie van Loki. Deze backend bleek erg rommelig waardoor het moeilijk te begrijpen was. Ook waren er fouten gemaakt in de manier waarop het Rasa framework werd gebruikt.

13.1.1.2 Taak

Mijn taak was om de backend van de experimentele versie van Loki grotendeels te behouden en slechts kleine aanpassingen in te doen. Op deze manier zou tijd worden bespaard en kon er sneller een volwaardig product worden neergezet met veel functionaliteiten

13.1.1.3 Actie

Het was voor mij dus erg lastig om de code te begrijpen, tevens zaten er fouten in waardoor ik ervan overtuigd begon te raken dat er nooit een robuuste gespreksassistent zou kunnen worden gemaakt uit de bestaande codebase. Ik ging in overleg met mijn bedrijfsbegeleider en de ontwikkelaars van de experimentele codebase. En we besloten dat het gunstiger zou zijn om opnieuw te beginnen met het risico dat niet alle functionaliteiten behouden bleven.

13.1.1.4 Resultaat

Dit resulteerde in een schone codebase die volgens de beschrijvingen van Rasa is geïmplementeerd. Verder zorgde dit ervoor dat ik het Rasa framework beter leerde kennen. De nieuwe codebase implementeert verder meerdere functionaliteiten die de gebruiker begeleiden door een gesprek, iets wat de experimentele versie compleet miste.

13.1.1.5 Reflectie

Ik denk dat dit een moeilijke maar goede keuze is geweest. De nieuwe code biedt veel meer perspectief voor toevoegen van functionaliteiten en is in de basis al veel robuuster dan de experimentele versie omdat het beter om kan gaan met onverwachte input van de gebruiker.

13.1.2 Covid-19

13.1.2.1 Situatie

Na het vooronderzoek en in de ontwerpfase van de stage werd het noodzakelijk om in verband met de corona crisis vanuit huis te werken. De ontwerpfase is een fase waarin er veel moet worden overlegd met verschillende partijen om zo goed mogelijk tot dezelfde visie te komen, waardoor het een erg slecht moment was om deze verandering te ondergaan. Verder woon ik nog thuis en bleek het soms lastig om een goede werkplek te kunnen vinden omdat natuurlijk iedereen thuis zat. Daarbij sloten de sportscholen, wat voor mij altijd een bron van motivatie was. En ook de sociale contacten vielen weg.

13.1.2.2 Taak

Mijn taak blijf natuurlijk het afstuderen. In dit geval dus het ontwikkelen van de testbare versie van de chatbot Loki.

13.1.2.3 Actie

In het begin was het onwijs moeilijk om een routine te kunnen vinden. Een werkdag van 8 uur zat er gewoon niet meer in. Ik kreeg van mijn bedrijfsbegeleider de tip om ook in de avonden langer door te werken zodat je overdag langere pauzes kon nemen. Dit werkte enigszins maar betekende wel dat er doordeweeks een constante druk ontstond om aan het werk te blijven. Zodra ik echter door de ontwerpfase heen was en begon met ontwikkelen lukte het om meer werk te verzetten. Verder hielpen de stand-ups elke ochtend goed als verplichting om aan het werk te gaan. En ook de wekelijkse voorgangsgesprekken met mijn bedrijfsbegeleider waren momenten waar ik hard naartoe werkte om ook daadwerkelijk nieuwe dingen te kunnen laten zien. Daarbij kreeg ik na ongeveer 5 weken thuiswerken de mogelijkheid om 2 dagen in de week op het kantoor van mijn vader te zitten. En tot slot heb ik tijdens de afstudeerstage nooit extra dagen “vrij” genomen buiten de verplichte vrije dagen.

13.1.2.4 Resultaat

De ondernomen acties om aan het werk te blijven zoals in de avonden door werken, stand-ups bijwonen, voortgangsgesprekken, geen extra vrije dagen en 2 dagen in de week een andere werkplek. Waren zeker nuttig en hebben mij erg geholpen. Echter heb ik in de eerste 5 weken nooit een goede werkplek kunnen vinden en ook daarna waren er nog 3 dagen in de week dat ik op mijn eigen kamer aan het werk moest. Verder betekende het thuiswerken minder contact met collega’s en andere stagairs waardoor het moeilijker was hen op de hoogte te houden van je werk en was het een grotere stap om daadwerkelijk om hulp te vragen als ik vastliep.

13.1.2.5 Reflectie

Tijdens de thuiswerk-periode heb ik gemerkt hoe belangrijk een aparte werkplek is voor je productiviteit. Verder viel het wegvallen van sport voor mij erg zwaar en was het moeilijker om daadwerkelijk dingen te ondernemen. Ik denk echter wel dat ik er het beste van heb gemaakt door de scheiding tussen werk en privé te vervagen en ook in de avonden hard door te werken. Daarbij heb ik elk mogelijke dag dat ik productief kon zijn benut door geen extra “vrije” dagen te nemen. Tot slot heb ik op het moment dat ik doorkreeg dat het erg lastig zou zijn om de code van de experimentele versie van Loki te blijven gebruiken, direct aan de bel getrokken waardoor ik daarmee slecht een week “verloor”.

14 Bronnen

Alle bronnen worden zoveel mogelijk weergegeven volgens de IEEE-norm.

1. Alan Nichol, " A New Approach to Conversational Software" Medium.com, 4 oktober 2017. [Online] Beschikbaar: https://medium.com/rasa-blog/a-new-approach-to-conversational-software-2e64a5d05f2a [Geraadpleegd op 18 februari 2020]
2. MNASRI, Maali “Recent advances in conversational NLP: Towards the standardization of Chatbot building” Arxiv.org, 21 maart 2019. [Online] Beschikbaar: https://arxiv.org/abs/1903.09025 [Geraadpleegd op 18 februari 2020].
3. Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan “A Neural Network Approach to Context-Sensitive Generation of Conversational Responses” Arxiv.org, 22 Juni 2015. [Online] Beschikbaar: https://arxiv.org/abs/1506.06714 [Geraadpleegd op 19 februari 2020].
4. Karri Sreedhar, “What it takes to build enterprise-class chatbots.” Chatbotsmagazine.com, 28 juni 2018. [Online] Beschikbaar: https://chatbotsmagazine.com/what-it-takes-to-build-enterprise-class-chatbots-db62227013c0 [Geraadpleegd op 24 februari 2020].
5. Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, Quoc V. Le “Towards a Human-like Open-Domain Chatbot” Arxiv.org, 31 Januari 2020 [Online] Beschikbaar: https://arxiv.org/abs/2001.09977 [Geraadpleegd op 25 februari 2020]
6. Umutcan Şimşek, Dieter Fensel, “Intent Generation for Goal-Oriented Dialogue Systems based on Schema.org Annotations” Arxiv.org, 3 juli 2018. [Online] Beschikbaar: https://arxiv.org/abs/1807.01292 [Geraadpleegd op 4 maart 2020]
7. Alan Nichol, “It’s about time we get rid of intents” blog.rasa.com, 5 december 2019. [Online] Beschikbaar: https://blog.rasa.com/its-about-time-we-get-rid-of-intents/ [Geraadpleegd op 4 maart 2020]
8. https://rasa.com/ [Geraadpleegd op 4 maart 2020]
9. https://deeppavlov.ai/ [Geraadpleegd op 4 maart 2020]
10. Yurakuratov, Yoptar, Mary Trofimova, Fedor Ignatov, AlexeySorokin “BERT in DeepPavlov” Docs.Deeppavlov.ai, 26 februari 2020. [Online] Beschikbaar: http://docs.deeppavlov.ai/en/master/features/models/bert.html [Geraadpleegd op 5 maart 2020]
11. Devashish Mamgain “Dialogflow vs Lex vs Watson vs Wit vs Azure Bot | Which Chatbot Service Platform To Use?” Kommunicate.io, 10 mei 2019. [Online] Beschikbaar: https://www.kommunicate.io/blog/dialogflow-vs-lex-vs-watson-vs-wit-vs-azure-bot/ [Geraadpleegd op 9 maart 2020]
12. “System entities reference” cloud.google.com, 24 februari 2020. [Online] Beschikbaar: https://cloud.google.com/dialogflow/docs/reference/system-entities [Geraadpleegd op 9 maart 2020]
13. “System entity details” cloud.ibm.com, 27 februari 2020. [Online] Beschikbaar: https://cloud.ibm.com/docs/assistant?topic=assistant-system-entities [Geraadpleegd op 9 maart 2020]
14. “Choosing a Pipeline” Rasa.com, 25 februari 2020. https://rasa.com/docs/rasa/nlu/choosing-a-pipeline/ [Geraadpleegd op 9 maart 2020]
15. Tom Bocklisch, Joey Faulkner, Nick Pawlowski, Alan Nichol “Rasa: Open Source Language Understanding and Dialogue Management” Arxiv.org, 15 December 2017 [Online] Beschikbaar: https://arxiv.org/abs/1712.05181 [Geraadpleegd op 9 maart 2020]

16. Daniel Braun, Adrian Hernandez Mendez, Florian Matthes, Manfred Langen “Evaluating Natural Language Understanding Services for Conversational Question Answering Systems” In Proc. of the SIGDIAL 2017 Conf. Saarbrücken, Germany, 15-17 August 2017. [Online] Beschikbaar: https://www.aclweb.org/anthology/W17-5522/ [Geraadpleegd op 10 maart 2020]
17. Xingkun Liu, Arash Eshghi, Pawel Swietojanski, Verena Rieser “Benchmarking Natural Language Understanding Services for building Conversational Agents” Arxiv.org, 26 maart 2019. [Online] Beschikbaar: https://arxiv.org/abs/1903.05566 [Geraadpleegd op 10 maart 2020]
18. https://flow.ai/nl/ [Geraadpleegd op 12 maart 2020]
19. “Flow ai api” Flow.ai. [Online] Beschikbaar: https://flow.ai/docs/api-docs/#introduction [Geraadpleegd op 12 maart 2020]
20. https://www.pdok.nl/datasets [Geraadpleegd op 16 maart 2020]
21. https://zakelijk.kadaster.nl/rijksdriehoeksmeting [Geraadpleegd op 18 maart 2020]
22. https://www.rijkswaterstaat.nl/zakelijk/open-data/normaal-amsterdams-peil/index.aspx [Geraadpleegd op 20 maart 2020]
23. https://linkeddata.cultureelerfgoed.nl/ [Geraadpleegd op 20 maart 2020]
24. “Use Case: Loki voor GEO Informatieverstrekking”, https://labs.kadaster.nl/cases/loki [Geraadpleegd op 12-02-2019]
25. Lou Benders, “Reflecteren met de STARR-methode”, Scribbr.nl, 18 november 2019 [Online] Beschikbaar: https://www.scribbr.nl/stage/starr-methode/
26. “Wat doet het kadaster”, Kadaster.nl, [Online] beschikbaar: https://www.kadaster.nl/over-ons/het-kadaster/wat-doet-het-kadaster