



Professionele Bachelor Toegepaste Informatica



Flight Helper: hulp bij vluchtvertraging

Joas Röthig

Promotoren:

Rob Kubben
Arno Barzan

Transform Data International
Hogeschool PXL Hasselt





Professionele Bachelor Toegepaste Informatica



Flight Helper: hulp bij vluchtvertraging

Joas Röthig

Promotoren:

Rob Kubben
Arno Barzan

Transform Data International
Hogeschool PXL Hasselt



Dankwoord

Voor het volbrengen van mijn stageopdracht zou ik graag verschillende mensen willen bedanken. Ten eerste wil ik mijn bedrijfspromotor Rob Kubben bedanken om mij de kans te geven om mijn stage bij Transform Data International te doen. Alsook voor de goede opvolging en begeleiding tijdens de stage.

Daarnaast wil ik ook mijn hogeschoolpromotor Arno Barzan bedanken voor de feedback op zowel mijn eindwerk als mijn stageportfolio. Dankzij deze feedback heb ik mijn bachelorproef tot een goed einde kunnen brengen.

Verder zou ik ook graag iedereen binnen Transform Data International willen bedanken om mijn stageperiode zo aangenaam mogelijk te maken. In het bijzonder zou ik nog René Beulen willen bedanken om mij te betrekken bij verschillende projecten binnen Transform Data International. Eveneens wil ik Ramon Rollinger bedanken die mij altijd heeft geholpen met mijn technische vragen. En als laatste ook Walter Henrard bij wie ik met mijn vragen of problemen altijd terecht kan.

Abstract

Een probleem dat zich regelmatig voordoet en zeer frustrerend kan zijn, is de vertraging van een vlucht. Wat veel mensen niet beseffen, is dat ze mogelijk recht hebben op een compensatie bij de vertraging van hun vlucht. Zo zijn er velen die geen idee hebben wat die vergoeding inhoudt of welke personen of organisaties ze moeten inlichten. Bij de vertraging van een vlucht kan de persoon in kwestie recht hebben op een compensatie tot maar liefst 600 euro. Zo blijkt uit onderzoek dat 63% van de beklagden in Europa geen officiële klacht indient, met de voornaamste reden dat ze denken dat het niet de moeite waard is.

Hierbij kan de ‘flight helper’-applicatie hulp bieden. ‘Flight helper’ is een webapplicatie ontwikkeld met Neota Logic, een *no-code* platform, dat aan de hand van enkele eenvoudige vragen, de gebruiker helpt bij het aanvragen van een compensatie en het opstellen van de brief aan desbetreffende luchtvaartmaatschappij. Tijdens mijn stageopdracht bij Transform Data wordt alles vanaf het begin uitgewerkt.

Een *no-code* platform wordt door velen gezien als iets dat zeer beperkt is voor het ontwikkelen van applicaties. Wat zijn nu de voordelen van het *no-code* platform Neota Logic ten opzichte van programmeertalen zoals Java en C#? In welke mate is Neota beperkt? Welke voordelen kunnen bedrijven uit een *no-code* platform halen? Dat zijn vragen die onderzocht worden in de onderzoeksopdracht.

Aan de hand van een vergelijkend onderzoek met onder andere een aantal literatuurstudies worden deze vragen beantwoord. Door het maken van een prototype in een programmeertaal zoals Java alsook met Neota Logic worden de krachten en beperkingen van *no-code* duidelijk.

Uit onderzoek blijkt dat Neota Logic het beste geoptimaliseerd is voor het maken van de ‘flight helper’-applicatie. Verder is gebleken dat bedrijven voordelen uit *no-code* platformen kunnen halen, maar dat er enkele risico’s aan verbonden zijn. Door *no-code* te ontwikkelen kunnen applicaties sneller ontwikkeld worden, maar daarvoor moet het gekozen project wel bij het gekozen *no-code* platform passen.

Inhoudsopgave

Dankwoord	ii
Abstract	iii
Inhoudsopgave	iv
Lijst van gebruikte figuren	vi
Lijst van gebruikte tabellen	viii
Lijst van gebruikte afkortingen	ix
Inleiding	1
I. Stageverslag	2
1 Bedrijfsvoorstelling	2
2 Voorstelling stageopdracht	3
2.1 Vergemakkelijken van aanvragen van compensatie bij vluchtvertraging	3
3 Uitwerking stageopdracht	3
3.1 Leren werken met Neota Logic	3
3.2 Onderzoek over compensatie bij vluchtvertraging	4
3.3 Opstellen van beslisboom	5
3.4 Maken van prototype met behulp van de Canvas prototyping tool	6
3.5 Ontwikkelen in Neota Logic	10
3.5.1 Neota Logic werkbank	11
3.5.2 Structuur binnen Neota Logic Studio	12
3.5.3 Verschillende tools binnen Neota Logic Studio	13
3.5.4 Vormgeving binnen Neota Logic Studio	18
3.6 Lay-out met behulp van HTML, CSS en bootstrap	19
3.6.1 Thema editor	20
3.6.2 CSS-bestanden importeren	21
3.7 Resultaten	21
4 Extra activiteiten	26
4.1 Corona advisor	26
4.2 Onderzoek Canvas	28
4.3 Onderzoek <i>open negotiations</i>	29
4.4 Smart task list	30
5 Besluit	31
II. Onderzoekstopic	33
1 Vraagstelling onderzoek	33
2 Onderzoeksmethode	33

3	<i>No-code</i>	33
3.1	Neota Logic.....	34
3.1.1	<i>No-code</i> ontwikkelen.....	34
3.1.2	Expertise Automatisatie	35
3.1.3	Proces automatisatie.....	35
3.1.4	Document automatisatie.....	35
3.1.5	Applicatie design	35
3.1.6	Applicatie bouwblokken	36
3.1.7	Third-Party integratie	36
3.1.8	Datamanagement.....	36
3.1.9	Secure Cloud.....	36
4	Low-code	37
5	Traditioneel programmeren.....	37
5.1	Java	38
6	<i>No-code</i> ten opzichte van traditioneel programmeren	38
6.1	Voordelen van <i>no-code</i>	38
6.1.1	Snelheid	39
6.1.2	Lagere kosten	39
6.1.3	Hogere productiviteit	39
6.1.4	Makkelijk aanpassen	39
6.2	Nadelen van <i>no-code</i>	39
6.2.1	Vereisten	40
6.2.2	Limieten bij het ontwikkelen	40
6.2.3	Niet beheren van de <i>source code</i>	40
6.3	Neota Logic ten opzichte van Java	40
6.3.1	If-else statements.....	40
6.3.2	Databases	42
6.3.3	Document generatie.....	43
6.3.4	Documentatie	44
6.4	Key advantages of No Code for complex decision support.....	45
	Conclusie	47
	Bibliografie	48
	Bijlagen	50

Lijst van gebruikte figuren

Figuur 1 Het business gebouw van de ENCI met de kantoren van TDI op de vijfde verdieping	2
Figuur 2 Logo van Transform Data International	2
Figuur 3 Onlinecursus van Neota Logic Academy	4
Figuur 4 Deel van de beslisboom van de 'flight helper'-applicatie	6
Figuur 5 Mogelijkheden van blokken in Canvas	7
Figuur 6 Begin van de Canvas applicatie	8
Figuur 7 if/else in Canvas.....	8
Figuur 8 E-mail met rapportering van bugs.....	9
Figuur 9 Antwoord op e-mail met rapportering van bugs	9
Figuur 10 Werkbank van Neota Logic.....	11
Figuur 11 Voorbeeld van een URL in Neota	11
Figuur 12 Structuur van de 'flight-helper'-applicatie	12
Figuur 13 Tools in Neota Logic	13
Figuur 14 Variabelen in Neota.....	13
Figuur 15 Variabele waarde binnen een word document	14
Figuur 16 Actie en goal Figuur 17 Actie voor compensatie.....	14
Figuur 18 Beslisboom in Neota Logic	15
Figuur 19 Formulieren in Neota	15
Figuur 20 Question flow in Neota	16
Figuur 21 API binnen Neota Logic Studio	17
Figuur 22 Inputs en parameters van API	17
Figuur 23 Headers binnen Neota Logic Studio	17
Figuur 24 Outputs van API's in Neota Logic Studio	18
Figuur 25 Onduidelijke beslisboom in Neota	18
Figuur 26 Duidelijke beslisboom binnen Neota Logic	19
Figuur 27 Form in Neota Logic Studio	19
Figuur 28 HTML-code van form.....	20
Figuur 29 Thema editor van Neota	20
Figuur 30 Scherm voor het wijzigen van CSS in Stylizer 7	21
Figuur 31 Introductiescherm van de 'flight helper'-applicatie.....	22
Figuur 32 Vragenreeks in de 'flight helper'-applicatie	22
Figuur 33 Berekening van compensatie	23
Figuur 34 Voorbeeld van een formulier in de 'flight helper'-applicatie.....	23
Figuur 35 Eindscherm van de 'flight helper'-applicatie.....	24
Figuur 36 Mail met bijlages als resultaat van de 'flight helper'-applicatie	24
Figuur 37 Startscherm op een gsm.....	25
Figuur 38 Scherm voor bepaling van compensatie op een gsm.....	25
Figuur 39 Eindscherm op een gsm	26
Figuur 40 Beslisboom van RIVM over coronavirus.....	27
Figuur 41 NLS-database van de corona advisor	27
Figuur 42 Voorbeeld van vragen van de corona hulpverlener.....	28
Figuur 43 Resultaat van de corona hulpverlener	28
Figuur 44 'Sweet-sixteen'-applicatie in Canvas	29
Figuur 45 <i>Open negotiations</i> van Neota Logic	29
Figuur 46 Bugs van <i>open negotiations</i> in Neota Logic	30
Figuur 47 NaaS die taken uit de database kan lezen.....	30

Figuur 48 Verschillende onderdelen van het Neota Logic platform	34
Figuur 49 Document automatisatie in Neota Logic.....	35
Figuur 50 Grafiek van abstractie ten opzichte van ontwikkeltijd.....	39
Figuur 51 Voorbeeld if-then-else	40
Figuur 52 Flowchart van een <i>nested if-else statement</i>	41
Figuur 53 <i>Nested if-else</i> in Java	41
Figuur 54 Laden van Oracle driver	42
Figuur 55 Connectie maken met de database.....	42
Figuur 56 Creëren van een statement in Java.....	42
Figuur 57 Ophalen en afdrukken van gegevens	42
Figuur 58 Ophalen van gegevens uit NDS	43
Figuur 59 Generen van Word document met behulp van Apache POI.....	44
Figuur 60 Teksteditor van Neota Logic.....	44

Lijst van gebruikte tabellen

Tabel 1 vluchtvertraging.....	5
-------------------------------	---

Lijst van gebruikte afkortingen

AI	Artificial Intelligence
API	Application Programming Interface
CSS	Cascading Style Sheets
GUI	Graphical User Interface
HTML	HyperText Markup Language
JDBC	Java DataBase Connectivity
NaaS	Neota as a Service
NDM	Neota Data Manager
NDS	Neota Data Store
PaaS	Platform as a Service
REST	Representational State Transfer
RIVM	Rijksinstituut voor Volksgezondheid en Milieu
SQL	Structured Query Language
TDI	Transform Data International
URL	Uniform Resource Locator
VPC	Virtual Private Cloud
VPL	Visual Programming Language
WORA	Write Once, Run Anywhere

Inleiding

Veel mensen beseffen niet dat ze mogelijk recht hebben op een compensatie bij de vertraging van hun vlucht. Vluchtvertraging is een probleem dat zich regelmatig voordoet en dat voor velen frustraties kan veroorzaken. Mensen hebben geen idee wat hun vergoeding inhoudt of welke organisaties ze moeten inlichten. Een compensatie bij vluchtvertraging kan al snel oplopen tot 600 euro. Doordat betrokken personen vaak denken dat het niet de moeite waard is om een vergoeding aan te vragen, dienen ze vaak geen klacht in.

‘Flight helper’ is een webapplicatie gecreëerd met het *no-code* platform Neota Logic. Door de gebruiker enkele eenvoudige vragen te stellen, helpt deze applicatie bij het aanvragen van een compensatie ten gevolge van vluchtvertraging. Bovendien helpt de ‘flight helper’-applicatie bij het opstellen van de brief aan desbetreffende luchtvaartmaatschappij. Tijdens mijn stage bij Transform Data International wordt alles vanaf het begin uitgewerkt.

Een *no-code* platform wordt door velen gezien als iets dat zeer beperkt is voor het ontwikkelen van applicaties. Wat zijn de voordelen van het *no-code* platform Neota Logic? In welke mate is Neota beperkt? Welke voordelen kunnen bedrijven uit een *no-code* platform halen? Dit zijn de hoofdvragen voor het onderzoek van de stageopdracht.

Een vergelijkend onderzoek met onder andere een aantal literatuurstudies zal deze vragen beantwoorden. Hiermee zullen de krachten en beperkingen van een *no-code* platform in kaart worden gebracht.

I. Stageverslag

1 Bedrijfsvoorstelling

Transform Data International (TDI) is opgericht in 2014 en is gelegen aan de Lage Kanaaldijk 115 in Maastricht. Hun kantoren zijn gevestigd op de vijfde verdieping van het business center van de ENCI. Het Transform Data team bestaat uit een tiental medewerkers.



Figuur 1 Het business gebouw van de ENCI met de kantoren van TDI op de vijfde verdieping

Het doel van Transform Data is het helpen van organisaties bij het beheren en optimaliseren van documenten, dossiers en processen. Dit streven ze na door “business and user first” te denken en ernaar te handelen zodat innovaties en nieuwe bedrijfsmodellen worden gewaardeerd door de eindgebruikers. Ze focussen zich op advocatenkantoren en juridische afdelingen, ondernemingen die gebruikmaken van Office 365, Sharepoint, MS Teams en organisaties die digitale transformatie doormaken.

Transform Data zorgt voor het adviseren, implementeren, en beheren van de applicatie. Transform Data beschikt over de laatste kennis, ruime en brede ervaring en is daarmee in staat organisaties te begeleiden bij het opstellen van een IT-strategie. Ze profileren zich door gebruik te maken van hun Custodian for Legal, een dossiermanagement gebaseerd op Office 365, om documenten, contracten en e-mails te beheren, efficiënt te zoeken en informatie te delen met derden. Dit in combinatie met de Neota Logic AI, een automatiseringsplatform, dat complexe en geavanceerde beslistools biedt. Ze leveren efficiëntie, transformeren legal services naar digitale oplossingen en creëren nieuwe, rendabele processen voor dienstverlening.



Figuur 2 Logo van Transform Data International

2 Voorstelling stageopdracht

2.1 Vergemakkelijken van aanvragen van compensatie bij vluchtvertraging

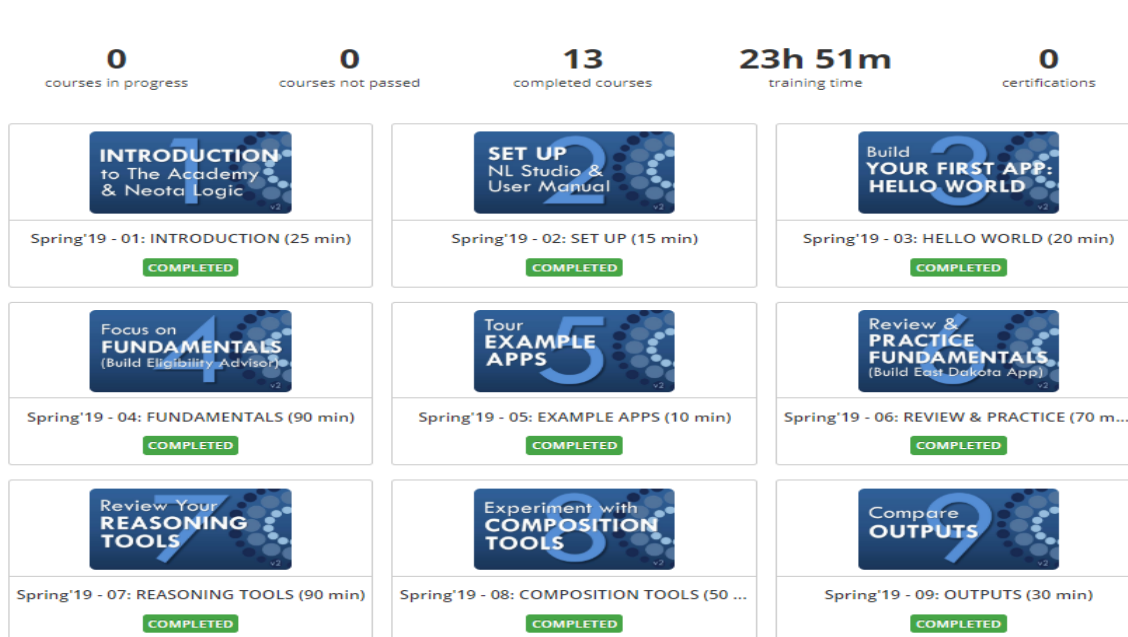
Het doel van deze stageopdracht is om het aanvragen van compensatie bij vluchtvertraging te vereenvoudigen. Uit onderzoek blijkt namelijk dat maar liefst 63% van de beklagden geen compensatie aanvraagt [1]. Het is de bedoeling dat een gebruiker door enkele duidelijke en simpele vragen te beantwoorden een brief kan genereren die aan de betrokken luchtvaartmaatschappij gestuurd kan worden en op die manier compensatie kan aanvragen. De applicatie berekent op basis van de ingevoerde gegevens van de gebruiker ook het bedrag van de compensatie.

De ontwikkelomgeving van de logica achter de applicatie is Neota Logic. Neota Logic is een *no-code* automatisatie platform dat aan de hand van een brede waaier aan tools ervoor zorgt dat er op een snelle manier applicaties gebouwd kunnen worden voor het automatiseren van diensten. Neota Logic heeft ook een prototype tool genaamd Canvas voor het maken van een prototype van de applicatie. Deze tool is op het moment van schrijven wel nog maar een bètaversie, maar zal toch gebruikt worden om een eerste prototype van de applicatie te maken. Voor de rest worden bootstrap, HyperText Markup Language (HTML) en Cascading Style Sheets (CSS) gebruikt voor de layout en styling van de ‘flight helper’ webapplicatie.

3 Uitwerking stageopdracht

3.1 Leren werken met Neota Logic

Aangezien het *no-code* platform Neota Logic nog vrij nieuw is, is het natuurlijk noodzakelijk om met deze technologie te leren werken. Neota Logic heeft een onlinecursus voor beginners die alle onderdelen binnen het platform behandelt. De tutorials gebeuren aan de hand van tekstuele uitleg alsook begrijpelijke video’s die concepten binnen Neota uitleggen. Bovendien wordt de kennis van bepaalde onderdelen tussendoor getest door extra oefeningen of challenges. Deze tutorials zijn te vinden bij Neota Logic Academy en geven ook op een visuele manier weer hoever de gebruiker met de cursus staat. Op de figuur hieronder zijn de hoofdstukken van de onlinecursus van Neota Logic Academy te zien.



Figuur 3 Onlinecursus van Neota Logic Academy

3.2 Onderzoek over compensatie bij vluchtvertraging

Aangezien deze applicatie volledig vanaf nul begint, is het natuurlijk belangrijk om eerst onderzoek te doen over compensatie bij vluchtvertraging. Het werd toen ook al snel duidelijk dat dit een vrij complex onderwerp is. Aangezien elk continent zijn eigen wetgevingen heeft, is er besloten om het bij vluchtvertragingen van vluchten binnen, naar en van Europa te beperken, omdat die allemaal onder de Europese wetgeving vallen. Alle bronnen die gebruikt zijn voor het onderzoek, zijn bronnen van de officiële website van de Europese Unie [2] [3]. Zo is het gegarandeerd dat alle informatie over vluchtvertragingen up-to-date is.

Uit dit onderzoek zijn verschillende voorwaarden gehaald. Aan die voorwaarden moet voldaan worden om recht te hebben op een compensatie ten gevolge van een vluchtvertraging. Een voorbeeld hiervan is dat de vertraging niet het gevolg mag zijn van buitengewone omstandigheden. Om de 'flight helper'-applicatie voor de gebruiker zo duidelijk mogelijk te houden, is het zeer belangrijk om de gebruiker duidelijk te maken wat precies onder buitengewone omstandigheden valt. Zo werd ook onderzocht welke gevallen precies tot deze buitengewone omstandigheden behoren.

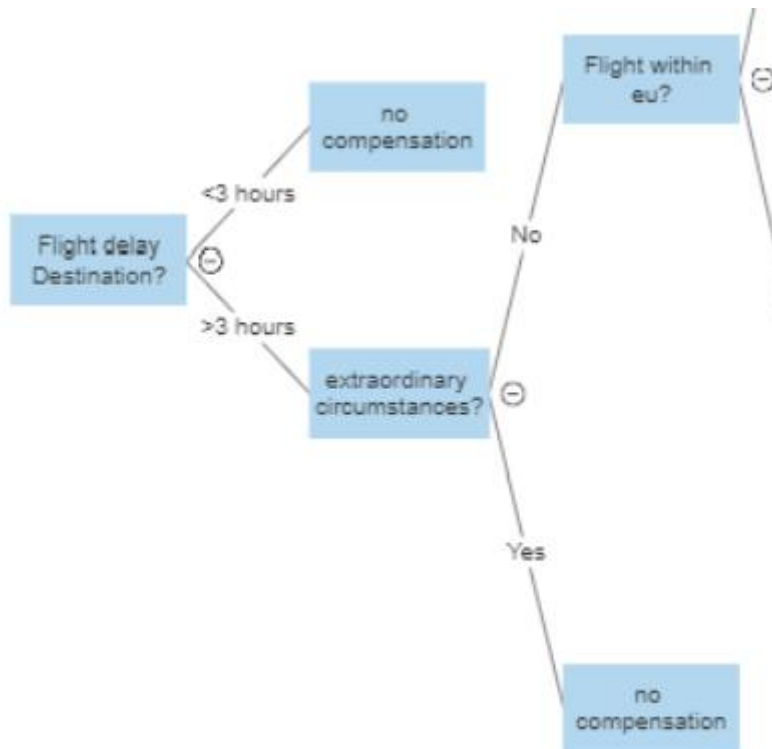
De vergoeding waar de gebruiker recht op heeft hangt ook af van de duur van de vertraging alsook de afstand van de vlucht. Onderstaande tabel geeft dit weer.

Type of flight	Distance	How long you have to wait
Short-haul	Up to 1,500km (932 miles) Flight time is usually about 2 hours or less	2 hours or more
Medium-haul	Between 1,500km - 3,500km (932-2,175 miles) Flight time is usually between 2 about 4 hours	3 hours or more
Long-haul	More than 3,500km (2,175 miles) Flight time is usually more than 4 hours	4 hours or more

Tabel 1 Regels voor vergoeding bij vluchtvertraging

3.3 Opstellen van beslisboom

Om alle gegevens en voorwaarden op een duidelijke en overzichtelijke manier weer te geven is er een beslisboom gemaakt. Een beslisboom of *decision tree* is een wetenschappelijk model voor de weergaven van alternatieven en keuzes in een besluitvormingsproces. Voor de ‘flight helper’-applicatie geeft dit weer welke vragen op welk moment gesteld moeten worden. Dat is zeer belangrijk, omdat de gebruiker daarom niet iedere vraag hoeft te beantwoorden. Hierdoor blijft het doorlopen van de applicatie korter en makkelijker en worden enkel de vragen die nodig zijn voor de eindconclusie gesteld. Hieronder is een gedeelte van de beslisboom te zien en onder bijlage A is de volledige beslisboom te vinden.



Figuur 4 Deel van de beslisboom van de 'flight helper'-applicatie

Zoals te zien is in bijlage A, geeft deze beslisboom duidelijk aan welk pad de gebruiker doorloopt. De eerste vraag die altijd gesteld wordt is: 'Hoeveel later kwam u aan op uw eindbestemming?' Afhankelijk van het antwoord dat de gebruiker hierop geeft, wordt de volgende vraag gesteld. Wanneer er 'langer dan drie uur' geantwoord wordt dan is de volgende vraag: 'Waren buitengewone omstandigheden de reden van uw vluchtvertraging?'. Het doorlopen van dit soort vragen wordt gedaan totdat er een conclusie gevormd wordt. In dit geval is dat het bedrag van de compensatie.

3.4 Maken van prototype met behulp van de Canvas prototyping tool

Om niet meteen te beginnen met ontwikkelen, zonder een goed idee te hebben over hoe de 'flight helper'-applicatie er uiteindelijk uit gaat zien, raadde TDI me aan om eerst een prototype te maken. Toevallig was Neota Logic juist bezig met het maken van een prototyping tool. Die tool heet Canvas en zit op het moment van schrijven nog in de bètaversie [4].

Canvas is een tool die kan helpen met het maken van applicaties door het simpele drag-and-drop principe. Het hulpprogramma kan vragen stellen, formules toepassen en nog veel meer. De voornaamste reden voor het gebruik van Canvas is dat het de flow van de applicatie op een duidelijke manier kan weergeven.

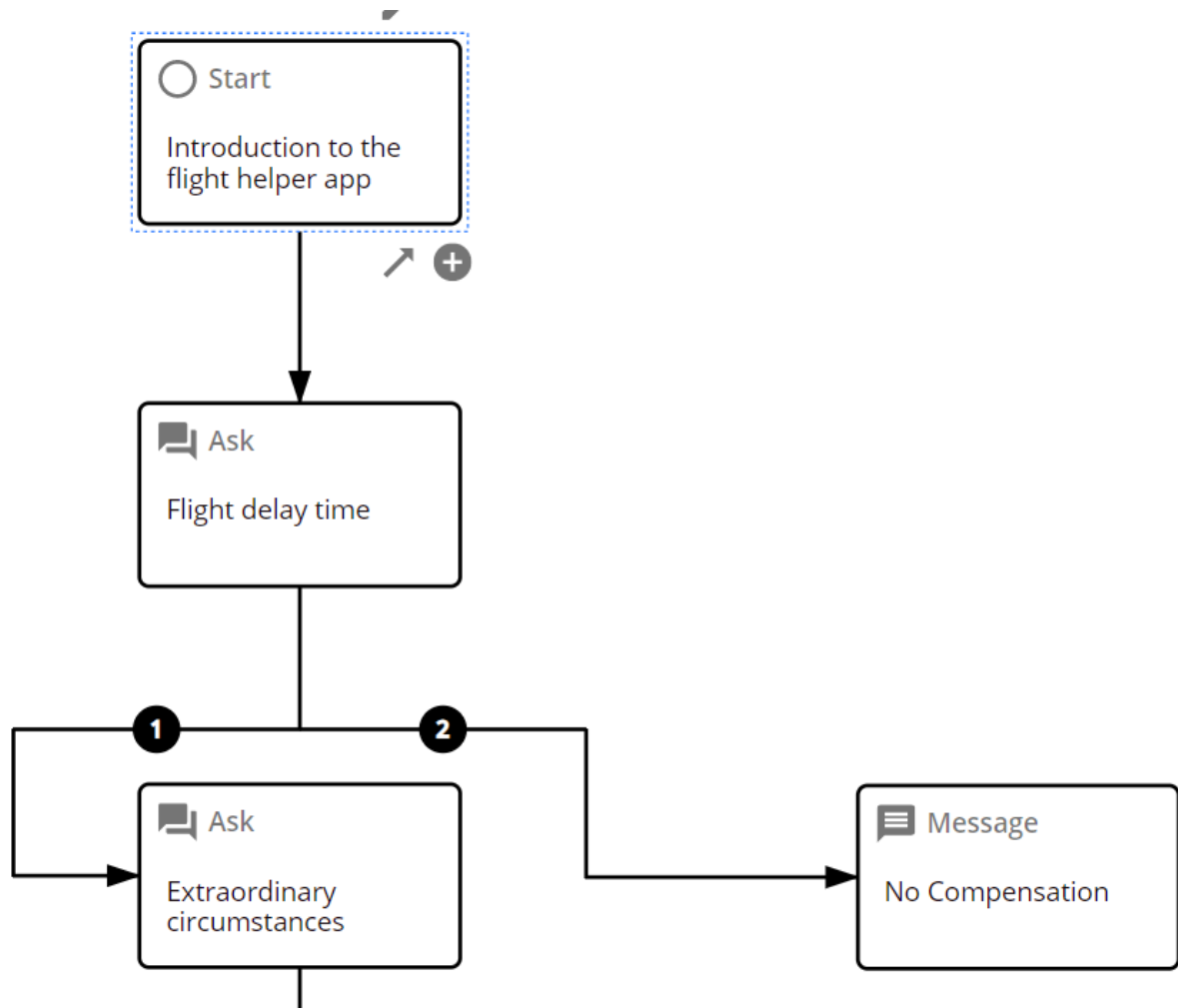
Na het opstellen van de beslisboom werd er gekeken naar de mogelijkheden van de Canvas. Omdat de tool in het begin nog zeer beperkt was, was het niet ideaal om er een prototype mee te maken. De tool werd al snel geüpdatet, waardoor een aantal bestaande functionaliteiten werden verbeterd en nieuwe mogelijkheden werden toegevoegd. Zo was het nu bijvoorbeeld mogelijk om een lijst van antwoorden te laten zien bij het stellen van vragen, waar het voorheen enkel mogelijk was om ja-neevragen te stellen.

Canvas gebruikt blokken die verbonden worden met pijlen om zo de flow van de applicatie duidelijk te maken. Die blokken kunnen vragen, berichten, formules en nog meer zijn. Op figuur 5 zijn alle mogelijke functionaliteiten van de blokken in Canvas te zien.



Figuur 5 Mogelijkheden van blokken in Canvas

Op de figuur hieronder is te zien hoe het begin van de flow van de 'flight helper'-applicatie er in de Canvas tool uitziet.



Figuur 6 Begin van de Canvas applicatie

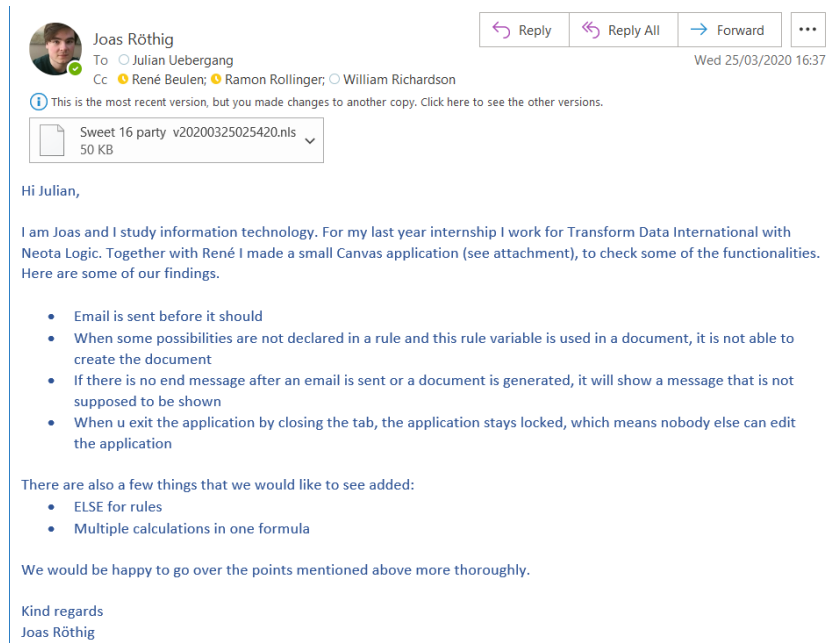
In dit schema wordt duidelijk dat er eerst een introductie tot de applicatie is. Hierna wordt de vraag in verband met de duur van de vertraging gesteld. Op basis van het gegeven antwoord wordt ofwel een bericht getoond dat zegt dat de gebruiker geen recht heeft op compensatie oftewel wordt de vraag over buitengewone omstandigheden gesteld. Op onderstaande figuur is dus te zien dat als de variabele 'Flight Delay' niet gelijk is aan 'Less than 3 hours later' dat dan de eerste situatie gevolgd wordt. Als dit niet het geval is dan wordt de tweede situatie uitgevoerd.

1	Situation 1 Logic for following path to next node	<input type="checkbox"/> ELSE situation
IF (Flight Delay != List Less than 3 hours later ...) ×		
2	Situation 2 Logic for following path to next node	<input checked="" type="checkbox"/> ELSE situation
ELSE		×

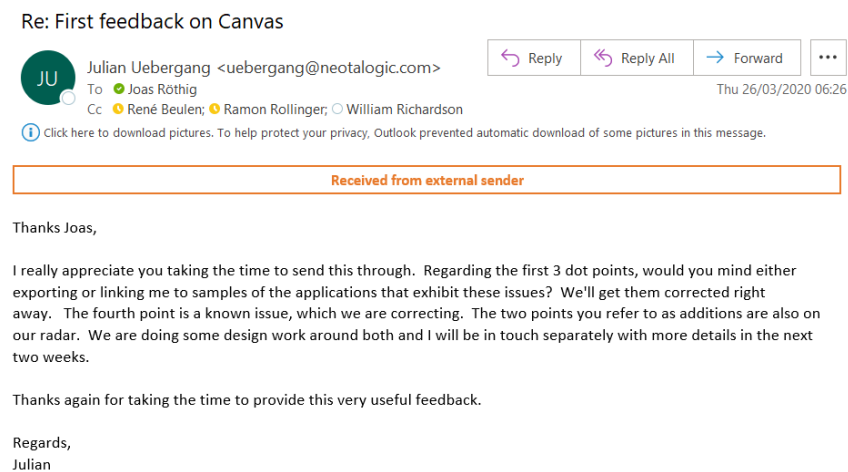
Figuur 7 if/else in Canvas

Aangezien Canvas zich nog in de bètaversie bevindt, zijn er tijdens het werken met de tool een aantal bugs gevonden. TDI heeft daarom gevraagd om een kleine applicatie te maken die deze programmafouten aantoont. Op deze manier kregen de makers van Canvas feedback over de

bevindingen met de nieuwste update. Daarom is er tijdens de stage ook tijd besteedt aan het maken van een kleine 'sweet sixteen'-applicatie die de eerder gevonden bugs duidelijk in kaart brengt. De flow van deze 'sweet sixteen'-applicatie is te vinden in bijlage 2. Een e-mail die niet verzonden wordt als er geen eindbericht aanwezig is, is een voorbeeld van zo een bug. Op onderstaande figuren is de rapportering van deze bugs te zien.



Figuur 8 E-mail met rapportering van bugs



Figuur 9 Antwoord op e-mail met rapportering van bugs

3.5 Ontwikkelen in Neota Logic

Nadat de 'flight helper'-applicatie als een prototype bestond met Canvas en het idee van de applicatie nu ook duidelijker was, kon er begonnen worden aan het ontwikkelen in de ontwikkelomgeving van Neota Logic. Neota Logic Studio wordt gebruikt voor het snel ontwikkelen van applicaties die expertise automatiseren. Organisaties gebruiken het Neota Logic platform voor het repliceren van het denken en de acties van experts. Applicaties die worden gebouwd met Neota vallen meestal onder minstens één van de volgende drie types:

- informatie van deskundige adviseurs;
- administratieve processen;
- automatiseren van documenten.

Het doel is om de toegankelijkheid, efficiëntie en effectiviteit van professionele hulp en diensten te verhogen.

Het ontwikkelproces van een Neota applicatie is meestal verdeeld in vier fases. De eerste fase is het definiëren van de functionaliteit. Het is daarom belangrijk om eerst onderzoek te doen en een prototype of dergelijke te maken. Hierdoor worden de functionaliteiten duidelijker en wordt er dus niet ontwikkeld zonder dat het uiteindelijke doel gekend is. De tweede fase omvat het voorbereiden van de inhoud. Dit houdt in om na te denken over de logica van de applicatie. Ook het opstellen van de nodige teksten en het opzoeken van afbeeldingen die gebruikt worden in de applicatie behoren tot deze fase. Het opstellen van de beslisboom valt hier ook onder. Door het maken van deze beslisboom is een deel van de logica van de applicatie al zeer duidelijk. Tijdens de derde fase wordt de applicatie gebouwd in de ontwikkelomgeving van Neota Logic, namelijk Neota Logic Studio. In dit onderdeel wordt de logica uit vorige fases omgezet in de applicatie. Ook wordt de algemene flow van de applicatie gemaakt en verschijnt er al ruwe output. Bovendien wordt ook de kern van de applicatie getest tijdens deze fase. Ten slotte is het belangrijk om de applicatie te verfijnen. Dit gebeurt in stadium vier. Onder dit stadium vallen vooral zaken die te maken hebben met de gebruiksvriendelijkheid. Het is de bedoeling dat de gebruiker de best mogelijke ervaring met de applicatie heeft. Het verbeteren van de lay-out, het optimaliseren van output en het toevoegen van input validatie behoren allemaal tot deze laatste fase.

3.5.1 Neota Logic werkbank

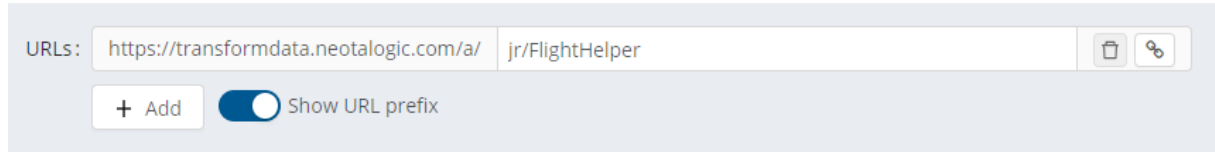
Met Neota Logic ontwikkelen begint altijd vanuit de werkbank. In de figuur hieronder is de werkbank van Neota te zien.

Applications

<

Figuur 10 Werkbank van Neota Logic

Vanuit deze werkbank is het mogelijk een applicatie te starten, te verwijderen of aan te passen alsook een nieuwe applicatie te maken of de naam van de Uniform Resource Locator (URL) te veranderen zoals te zien is op figuur 11.



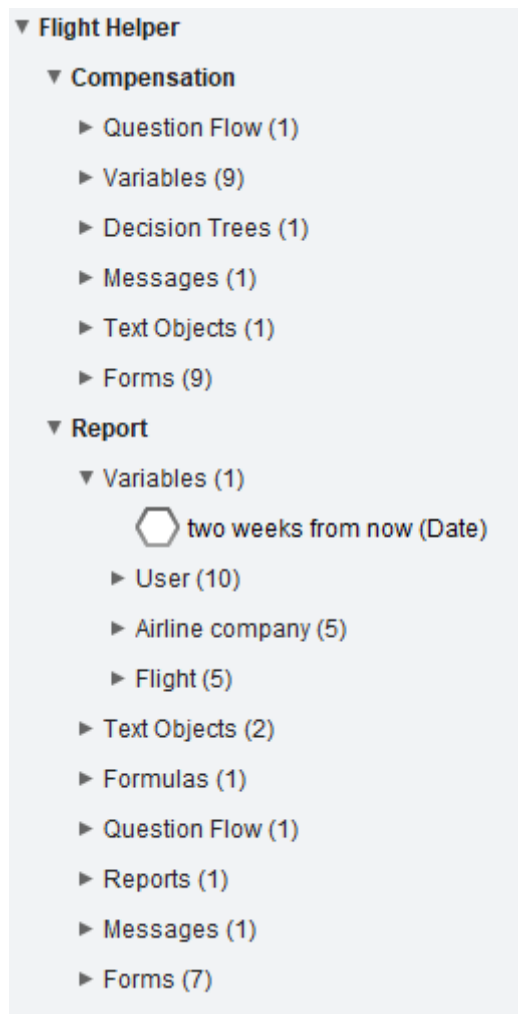
URLs:

+ Add ☒ Show URL prefix

Figuur 11 Voorbeeld van een URL in Neota

3.5.2 Structuur binnen Neota Logic Studio

In de studio van Neota Logic wordt alles wat te maken heeft met het ontwikkelen van de applicatie en zijn logica gedaan. Een applicatie in Neota heeft altijd één *main issue* die zelf verschillende *sub-issues* kan hebben. Deze *sub-issues* bevatten alle variabelen, formules en logica van de applicatie. Zoals te zien is op figuur 12 heet de *main issue* van de applicatie 'Flight Helper' en heeft deze 'Compensation' en 'Report' als twee *sub-issues*. De *sub-issues* bevatten op hun beurt weer variabelen die ook onderverdeeld zijn in verschillende folders. Op figuur 12 is er ook te zien dat de folder 'Variables' van 'Report' is onderverdeeld in 'User', 'Airline Company' en 'Flight'.

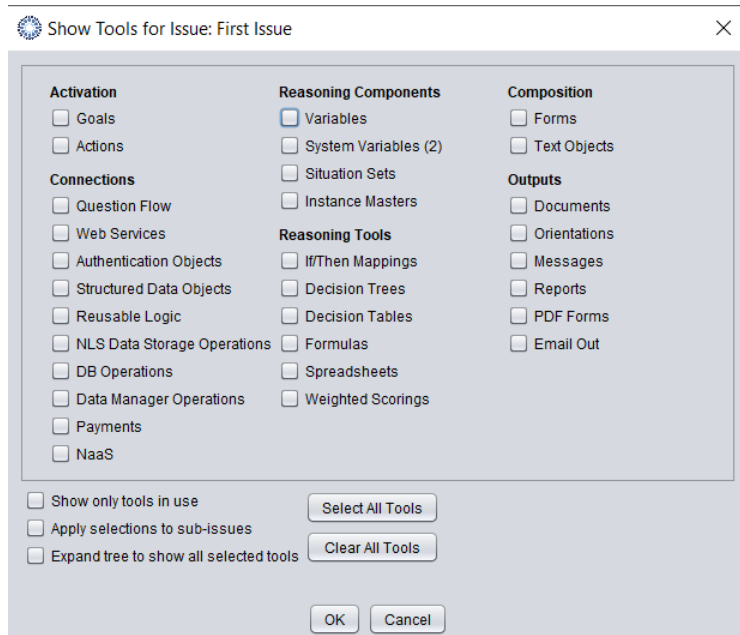


Figuur 12 Structuur van de 'flight-helper'-applicatie

Het is belangrijk om op te merken dat Neota, op niveau van issues, altijd van boven naar onder werkt. Dit wil zeggen dat eerst de issue 'Compensation' volledig wordt afgewerkt voordat er aan de volgende issue begonnen wordt. Dit betekent dus dat wanneer 'Report' boven 'Compensation' zou staan, eerst 'Report' volledig wordt uitgevoerd en dus ook de flow van de applicatie helemaal verandert.

3.5.3 Verschillende tools binnen Neota Logic Studio

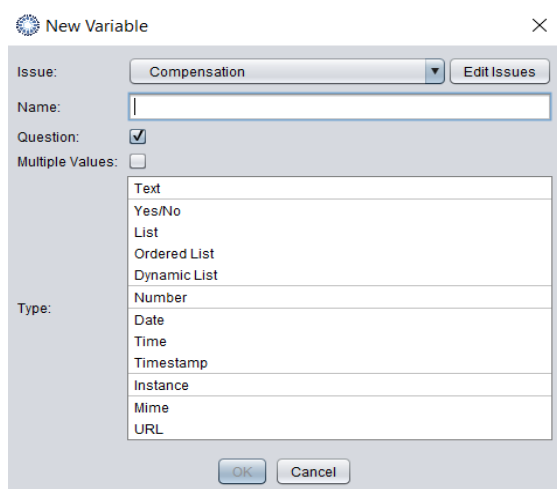
Neota heeft allerlei verschillende tools om je te helpen bij de logica van jouw applicatie. Elke *sub-issue* heeft toegang tot deze tools. Op onderstaande figuur is te zien welke tools elk *sub-issue* tot zijn beschikking heeft.



Figuur 13 Tools in Neota Logic

3.5.3.1 Variabelen

Variabelen kunnen aangemaakt worden binnen een *sub-issue*. Bij het maken van een variabele bestaat er de keuze uit verschillende types zoals te zien is op figuur 14. Om het overzicht te bewaren kunnen deze variabelen nog onderverdeeld worden in folders. Zo blijft het overzichtelijk wanneer er veel variabelen in één *sub-issue* zitten. Aan een variabele kan een vraag meegegeven worden die samen met een invoerveld, afhankelijk van het geselecteerde type, in de applicatie getoond wordt. Op het moment dat een gebruiker iets invult wordt dit opgeslagen in de variabele en kan het door de rest van de applicatie gebruikt worden. Op figuur 15 is te zien hoe de waarde van een variabele gebruikt wordt tijdens het genereren van een word document.



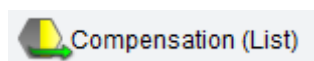
Figuur 14 Variabelen in Neota



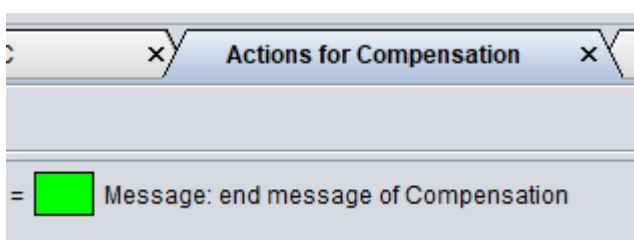
Figuur 15 Variabele waarde binnen een word document

3.5.3.2 Goals en acties

Aan een variabele kunnen ook goals en acties meegegeven worden. Een goal wordt aangegeven met een groene pijl en een actie met een gedeeltelijk gele achtergrondkleur zoals te zien is op figuur 16. Een goal is vaak het einde van de issue. Hierdoor weet Neota dat hij verder naar de volgende issue kan gaan. Een actie wordt uitgevoerd op het moment dat de variabele een waarde bevat. Op figuur 17 heeft 'Compensation' een actie die een bericht laat zien. Op het moment dat 'Compensation' een waarde bevat, zal dit bericht dus getoond worden.



Figuur 16 Actie en goal



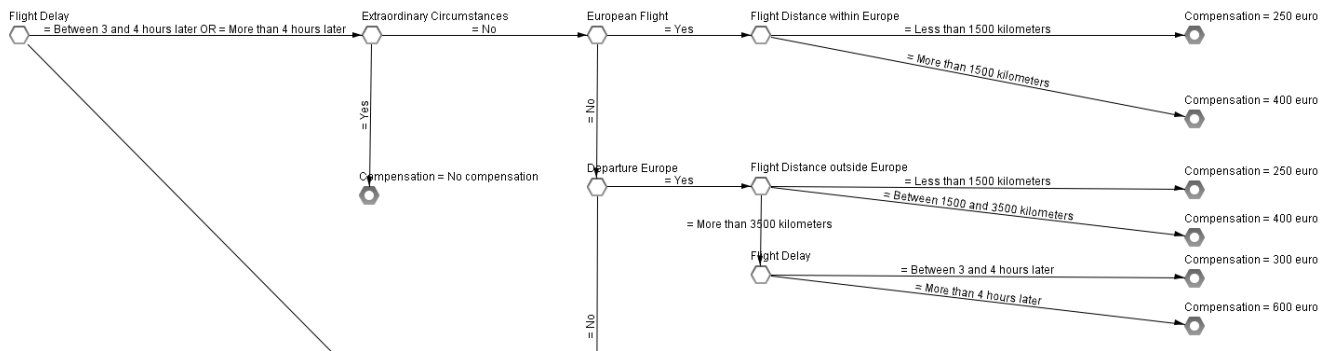
Figuur 17 Actie voor compensatie

3.5.3.3 Tekst objecten

Wanneer een stuk tekst vaker dan één keer gebruikt wordt, is het makkelijk om dit in een tekst object te zetten. Zo kan één stuk tekst op meerdere plaatsen voorkomen, wat typewerk bespaart. Een andere reden voor het gebruik van tekst objecten is het feit dat het de applicatie beter en makkelijker onderhoudbaar maakt. Als een stuk tekst verandert, hoeft dit slechts op één plaats aangepast te worden. Een goede naamgeving van deze tekst objecten is dan ook essentieel. Door een duidelijke naamgeving is het makkelijker om stukken tekst, die aangepast moeten worden, terug te vinden. Dit is extra belangrijk bij grotere applicaties.

3.5.3.4 Beslisbomen

Voor *if-then* condities in Neota bestaan er verschillende mogelijkheden. Eén van de opties is een beslisboom. Een beslisboom geeft op een duidelijke manier weer wat er na een bepaalde actie moet gebeuren. In Neota worden deze beslisbomen gecreëerd met knooppunten en linken. De knooppunten zijn de variabelen en de linken bevatten de condities. Op figuur 18 is een gedeelte van een beslisboom in Neota Logic te zien.



Figuur 18 Beslisboom in Neota Logic

Enkele andere mogelijkheden voor *if-then* condities binnen Neota zijn *if-then mappings* en beslistabellen. Voor de ‘flight-helper’-applicatie was het gebruik van beslisbomen het meest voor de hand liggend, omdat er voor het ontwikkelen in Neota al een beslisboom was gemaakt.

3.5.3.5 Formulieren

Om meerdere vragen op één pagina te tonen, is het makkelijk om formulieren te gebruiken. In formulieren kunnen de vragen en invoervelden van de variabelen gebruikt worden om deze op een verzorgde manier onder elkaar te laten verschijnen.

FORM

Please fill in the necessary info about your flight. ¶

QUESTIONS

question-of:Departure-airport

valueInput-of:Departure-airport ¶

question-of:Destination-airport

valueInput-of:Destination-airport ¶

question-of:Date-flight

valueInput-of:Date-flight ¶

question-of:Booking-reference-number-airline

valueInput-of:Booking-reference-number-airline ¶

question-of:Flight-number

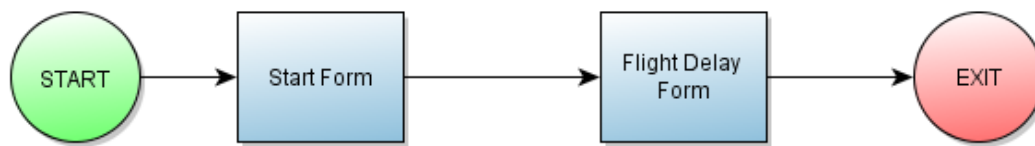
valueInput-of:Flight-number ¶

Figuur 19 Formulieren in Neota

3.5.3.6 Question Flows

Voor het volgen van de flow van de applicatie heeft Neota enkele regels. Zo wordt er eerst gekeken naar de issues. De bovenste issue wordt als eerste uitgevoerd, daarna die eronder enzovoort. Binnen deze issues kijkt Neota als eerste naar de *question* flow. Een *question* flow geeft, zoals het woord zegt, de flow van de vragen weer. Een *question* flow heeft een begin- en eindknooppunt en hiertussen variabelen of formulieren. De applicatie zal de vragen stellen aan de hand van de volgorde in de *question* flow. Als een variabele voorkomt in bijvoorbeeld een beslisboom, dan zal Neota eerst

deze beslisboom overlopen alvorens verder te gaan met de *question* flow. Daardoor is het niet nodig om alle variabelen in de *question* flow te zetten.



Figuur 20 Question flow in Neota

Op bovenstaande figuur is te zien dat de applicatie begint met een startknooppunt. Hierna worden de twee formulieren getoond en wordt de flow geëindigd met een eindknooppunt. Toch verschijnen na het vluchtvertragingsformulier nog eerst allerlei vragen. Dit komt omdat in dit formulier de variabele 'Flight Delay' gebruikt wordt. Neota ziet dit en weet dat deze variabele voorkomt in de beslisboom en zal dus ook eerst heel de beslisboom afdraaien voordat de applicatie daadwerkelijk wordt beëindigd. Het is dus ontzettend belangrijk om te weten welke prioriteiten Neota stelt op gebied van de flow van de applicatie. Anders bestaat er de kans dat een applicatie eindigt, zonder dat alle vragen gesteld zijn geweest.

3.5.3.7 Webservices

Een webservice is een manier om twee systemen gegevens te laten uitwisselen. De verzender stuurt hierbij data naar de ontvanger en de ontvanger stuurt gelijk een reactie terug [5].

Om in de 'flight helper'-applicatie een lijst van alle luchthavens terug te krijgen, is er gebruikgemaakt van een webservice. Hiervoor moest er eerst een goede Application Programming Interface (API) gevonden worden. Nadat er onderzoek is gedaan naar enkele API's die luchthavens als reactie terugsturen, was het resultaat uiteindelijk drie API's. De API's van Airportsfinder, Amadeus en OurAirports. Deze drie API's zijn alle drie binnen de Neota Logic Studio getest. Zowel Airportsfinder en Amadeus gaven alle luchthavens terug op basis van de stad die de gebruiker ingeeft. Het probleem met deze API's was echter dat er veel luchthavens niet als resultaat teruggegeven werden. Bovendien moest op de exacte stad gezocht worden. Brussel gaf de luchthaven in Brussel terug, maar wanneer Zaventem werd ingegeven kon geen luchthaven gevonden worden. Het is daarom dat er uiteindelijk voor de API van OurAirports is gekozen. Deze API geeft alle luchthavens van een land terug. Hierdoor moet de gebruiker het land van de luchthaven invoeren en deze krijgt dan een lijst van alle luchthavens uit dat land terug, waaruit de juiste luchthaven gekozen kan worden.

Binnen Neota Logic Studio bestaat de tool 'Web Services'. In deze tool kan de URL van de API meegegeven worden. Hierna kan er aangeduid worden of het gaat over een 'GET', 'POST', 'PUT', 'PATCH' of 'DELETE' request. Hieronder kan ook nog de methode van authenticatie aangegeven worden.

Inputs **Headers** **Outputs**

URL

https://ourairport-data-search.p.rapidapi.com/airports-in/belgium/belgium

HTTP Method: ☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE

Authentication: ☒ None ☐ Basic ☐ NTLM ☐ SSL Certificate

Test URL Synchronize

Figuur 21 API binnen Neota Logic Studio

De inputs voor de API kunnen ook ingegeven worden. Hierdoor is het mogelijk om bijvoorbeeld variabelen als parameter mee te geven. Op onderstaande afbeelding is te zien hoe de variabele 'Country' als waarde voor een parameter meegegeven wordt.

Inputs		Add Named Param	Add Unnamed Param	Delete
Input Name	Value			
BASE_URL	ourairport-data-search.p.rapidapi.com			
TRANSFER_PROTOCOL	https			
Url_Param_1	airports-in			
Url_Param_2	Country of Destination of Report			
Url_Param_3	Country of Destination of Report			

Figuur 22 Inputs en parameters van API

Ook de 'Headers' die nodig zijn voor de API kunnen ingevuld worden. Onder de 'Headers' vallen vaak de *API-keys* zoals ook op figuur 23 te zien is.

Headers	
Name	Value
x-rapidapi-host	ourairport-data-search.p.rapidapi.com
x-rapidapi-key	d1ec4f7ec9msh8f18c96145b9de0p181020jsn76fd2117f31b

Figuur 23 Headers binnen Neota Logic Studio

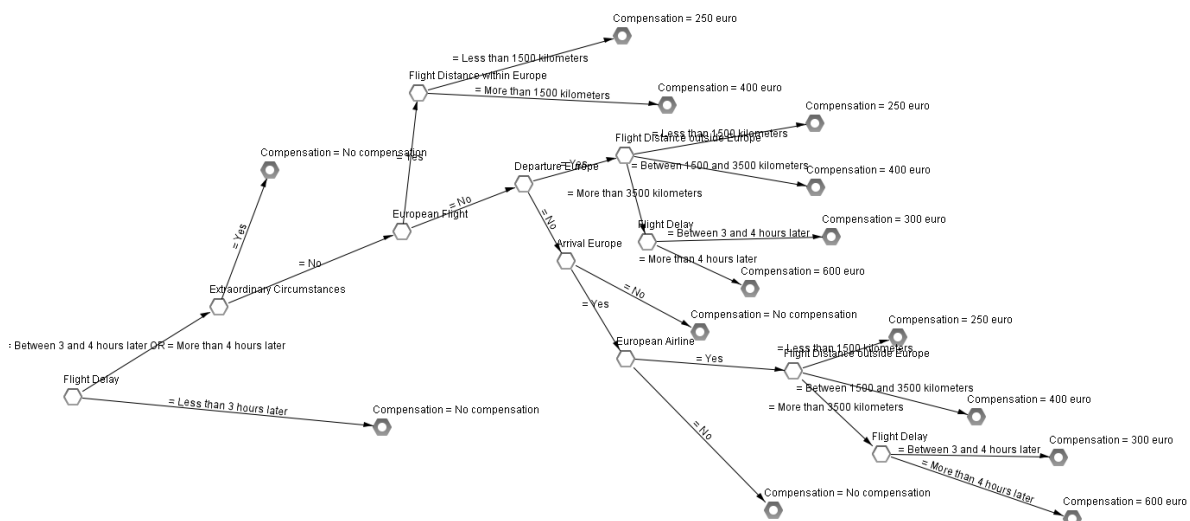
Als laatste is er het tabblad voor outputs. In dit tabblad wordt de teruggegeven data opgeslagen in variabelen. Op onderstaande figuur 24 is te zien hoe de namen van luchtvelden wordt opgeslagen in de instantie 'FName of Destination Airport'. Instanties houden een array van waarden vast. Deze instanties in Neota worden vaak gebruikt voor externe data. Dit kan data uit een database of in dit geval data van een webservice zijn.

Outputs	
Output Name	Variable Name
lookup	
isError	
query	
original	
isId	
isNameOrKeyword	
errorMessage	
count	
results	
continent	
country	
keywords	
municipality	
lon	FLongitude of Destination Airport of Report
gps	
type	
iata	
elev	
hasScheduledService	
name	FName of Destination Airport of Report
icao	
localCode	
id	
wikipedia	
region	
lat	FLatitude of Destination Airport of Report
homepage	

Figuur 24 Outputs van API's in Neota Logic Studio

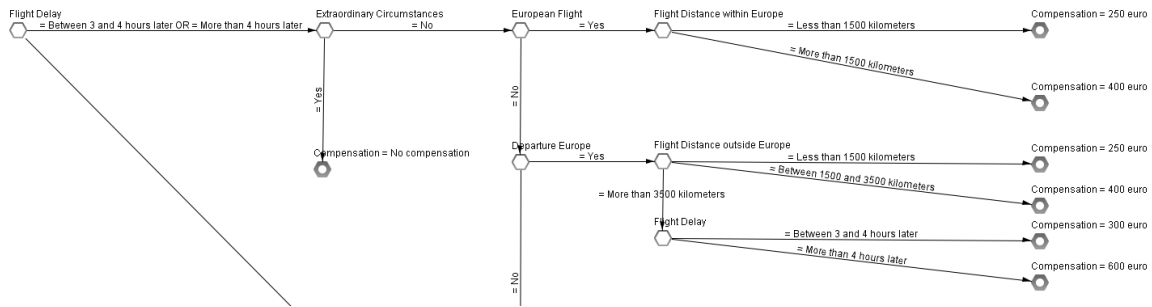
3.5.4 Vormgeving binnen Neota Logic Studio

Hoewel Neota Logic geen speciale regels heeft, zoals Java bijvoorbeeld *pascal casing heeft*, zijn er toch een aantal best practices. Deze best practices, rond het duidelijk en overzichtelijk houden van de applicatie, zijn duidelijk geworden door feedback van de collega's van TDI. Er zijn enkele tips gegeven toen er een presentatie gegeven moest worden over de voorlopige stand van de 'flight helper'-applicatie. Eén van de punten van de feedback had te doen met de vormgeving van mijn beslisboom. Voor de persoon die de beslisboom opstelt, is deze duidelijk en goed te begrijpen. Echter voor iemand van buitenaf, bijvoorbeeld een klant, kan het onoverzichtelijk overkomen waardoor het al snel ingewikkeld lijkt. Op onderstaande figuur is te zien hoe de beslisboom er voor de herwerking uitzag.



Figuur 25 Onduidelijke beslisboom in Neota

Deze beslisboom is zeer onduidelijk en onoverzichtelijk. Soms doorkruisen delen tekst elkaar. In een beslisboom zoals deze is het natuurlijk ontzettend moeilijk om later veranderingen in aan te brengen. Daarom ziet de beslisboom na enkele tips van mijn collega's eruit zoals te zien is op onderstaande figuur. De volledige beslisboom is te vinden onder bijlage C.



Figuur 26 Duidelijke beslisboom binnen Neota Logic

Hier is wel alle tekst duidelijk leesbaar en is ook makkelijk te zien hoe de flow van de beslisboom in elkaar zit. De techniek die hiervoor is toegepast, is door bovenaan met het eerste knooppunt te beginnen en altijd verder naar onder te werken. Hierdoor blijft alles overzichtelijk en is het makkelijk om er later veranderingen in aan te brengen of om knooppunten en linken toe te voegen.

Een tweede zaak om de applicatie overzichtelijk en onderhoudbaar te houden, is het gebruik van een goede naamgeving voor variabelen. Een goede naamgeving houdt in dat het omschrijft wat de variabele doet of inhoudt. Een formulier met de naam 'Gebruiker' bevat het best enkel vragen en gegevens over de gebruiker zelf. Als er dan toch nog andere variabelen zoals vluchtgegevens van de gebruiker nodig zijn, is de best practice om een tweede formulier aan te maken.

3.6 Lay-out met behulp van HTML, CSS en bootstrap

Elk element dat aangemaakt wordt in Neota Logic is in feite een stukje HTML-code. Wanneer er bijvoorbeeld een formulier wordt aangemaakt, is dit op de achtergrond eigenlijk een stuk HTML-code. Het is daarom ook altijd mogelijk om de HTML-code handmatig te wijzigen. Op figuur 27 is een formulier dat in Neota is gemaakt te zien en op figuur 28 de code die ervoor op de achtergrond gegenereerd wordt.

FORM

Please fill in the necessary info about your flight.

question-of:Date-flight

valueInput-of:Date-flight

question-of:Booking-reference-number-airline

valueInput-of:Booking-reference-number-airline

question-of:Flight-number

valueInput-of:Flight-number

question-of:Departure-Airports

valueInput-of:Departure-Airports

question-of:Destination-Airports

valueInput-of:Destination-Airports

<-Back

Next->

Figuur 27 Form in Neota Logic Studio

```

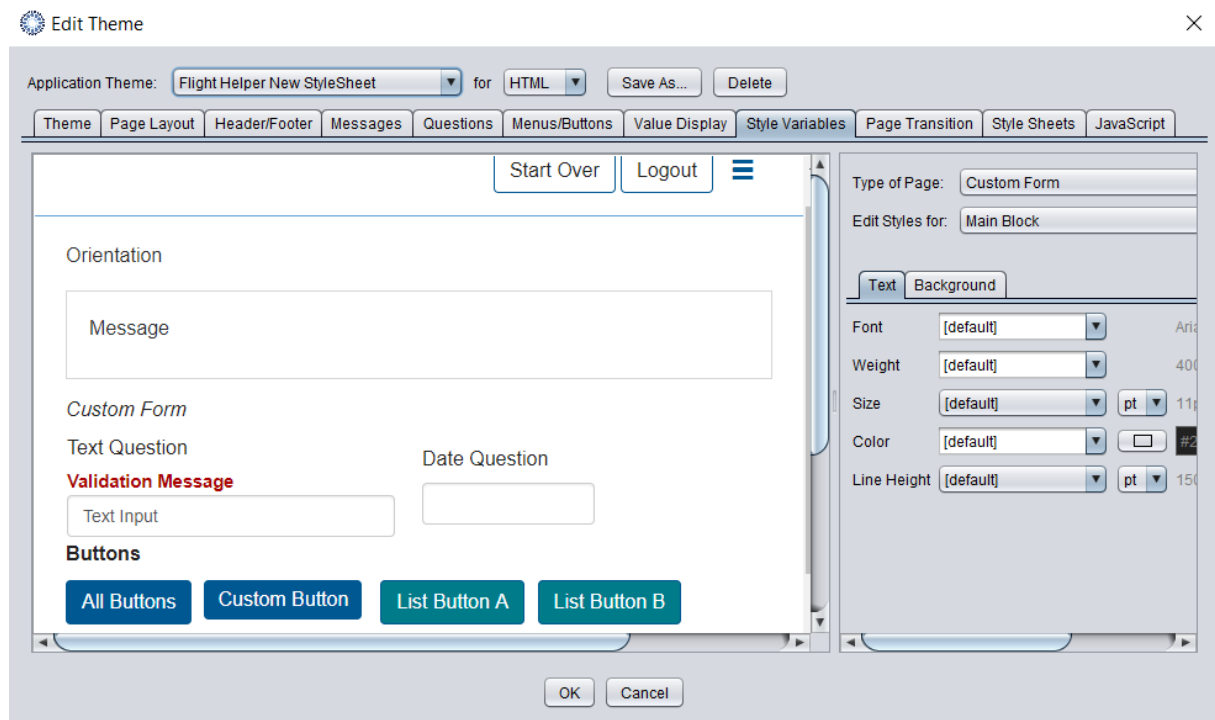
<?xml version="1.0" encoding="ISO-8859-1" ?>
<html lang="en-US" xml:lang="en-US" xmlns="http://www.w3.org/1999/xhtml" xmlns:ro="http://www.realobjects.com/review">
  <head>
    <meta content="text/html; charset=ISO-8859-1" http-equiv="Content-Type"/>
    <base href="https://transformdata.neotalogic.com"/>
    <link href="/assets/TaggedTextCSS/Standard.css" rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <form fid="697" style="width: 100%;">
      <div class="card" style="text-align: left;">
        <div class="card-header">Please fill in the necessary info about your flight.</div>
        <div class="card-body">
          <p><variable aspect="question" attrname="Date flight" ctxt="0,252" fact="true"></variable><variable
            aspect="valueInput" attrname="Date flight" ctxt="0,252" fact="true"></variable></p>
          <p><variable aspect="question" attrname="Booking reference number airline" ctxt="0,263"
            fact="true"></variable><variable aspect="valueInput" attrname="Booking reference number airline"
            ctxt="0,263" fact="true"></variable></p>
          <p><variable aspect="question" attrname="Flight number" ctxt="0,374"
            fact="true"></variable><variable aspect="valueInput" attrname="Flight number" ctxt="0,374"
            fact="true"></variable></p>
          <p><variable aspect="question" attrname="Departure Airports" ctxt="0,1091"
            fact="true"></variable><variable aspect="valueInput" attrname="Departure Airports" ctxt="0,1091"
            fact="true"></variable></p>
          <p><variable aspect="question" attrname="Destination Airports" ctxt="0,970"
            fact="true"></variable><variable aspect="valueInput" attrname="Destination Airports"
            ctxt="0,970" fact="true"></variable></p>
        </div>
        <div class="form-buttons" style="text-align: left;">
          <p><input id="PreviousSubmit" systemcmd="Go+Back" txt="&lt; Back" type="submit"
            value="&lt; Back"/> <input class="submit" id="NextSubmit" type="submit" value="Next >"/></p>
        </div>
      </div>
    </form>
  </body>
</html>

```

Figuur 28 HTML-code van form

3.6.1 Thema editor

De lay-out binnen Neota Logic kan gebeuren aan de hand van de thema editor. Deze editor is op onderstaande figuur te zien.



Figuur 29 Thema editor van Neota

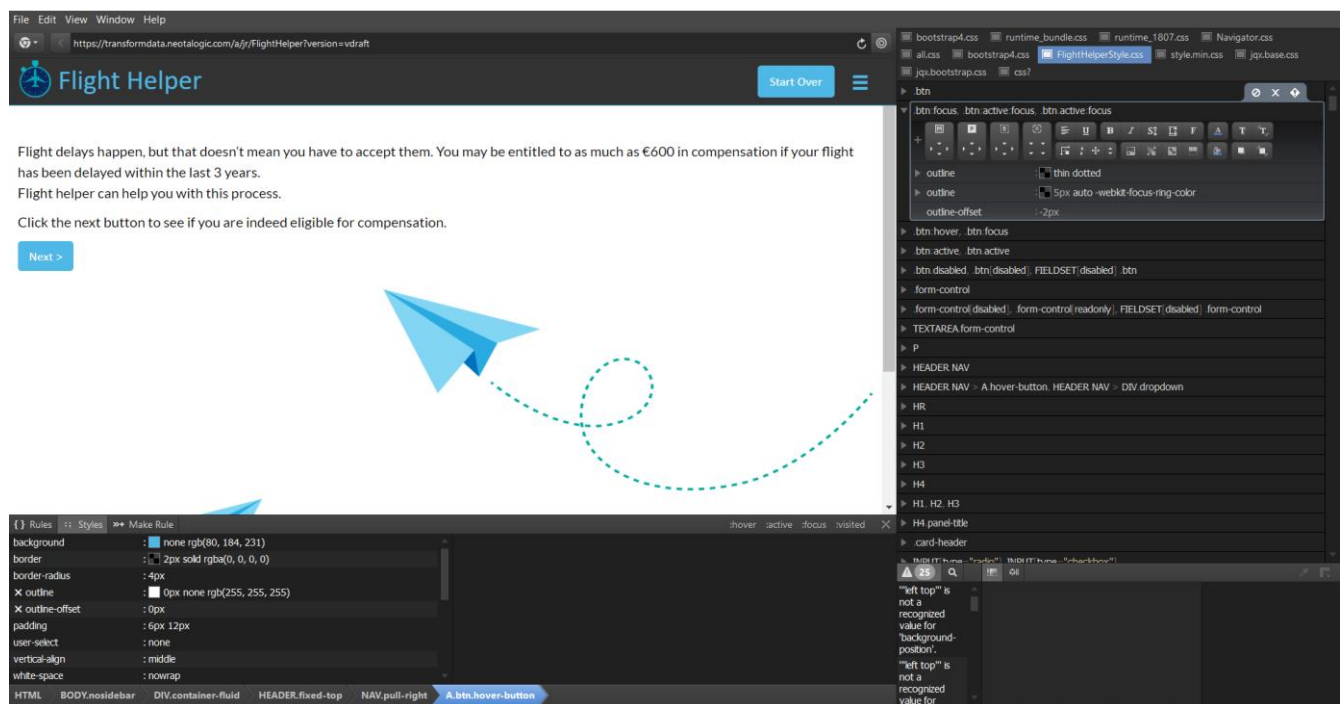
Binnen deze editor is het mogelijk om onder andere het uiterlijk van knoppen, inputvelden en de navigatiebalk te veranderen. Bovendien biedt Neota de optie om je eigen CSS-bestanden toe te voegen.

3.6.2 CSS-bestanden importeren

Een andere optie voor de lay-out van een Neota applicatie is om een eigen CSS-bestanden te maken en te importeren naar de thema editor. Dit is ook het meest gebruikt voor de 'flight helper'-applicatie. De lay-out van de applicatie bestaat uit CSS-bestanden met de bootstrap 4 *library*.

Bovendien is er gebruikgemaakt van Stylizer 7. Stylizer 7 is een CSS-editor die helpt met het schrijven van CSS-code. De editor laat onmiddellijk alle veranderingen zien zonder dat hiervoor de pagina eerst opnieuw geladen moet worden. Het biedt enkele tools die het schrijven van CSS-code vergemakkelijkt. Enkele handige functionaliteiten zijn onder andere:

- online CSS-bestanden aanpassen;
- visuele tools zoals een vergrootglas;
- 'bullseye': elementen selecteren en onmiddellijk aanpassen

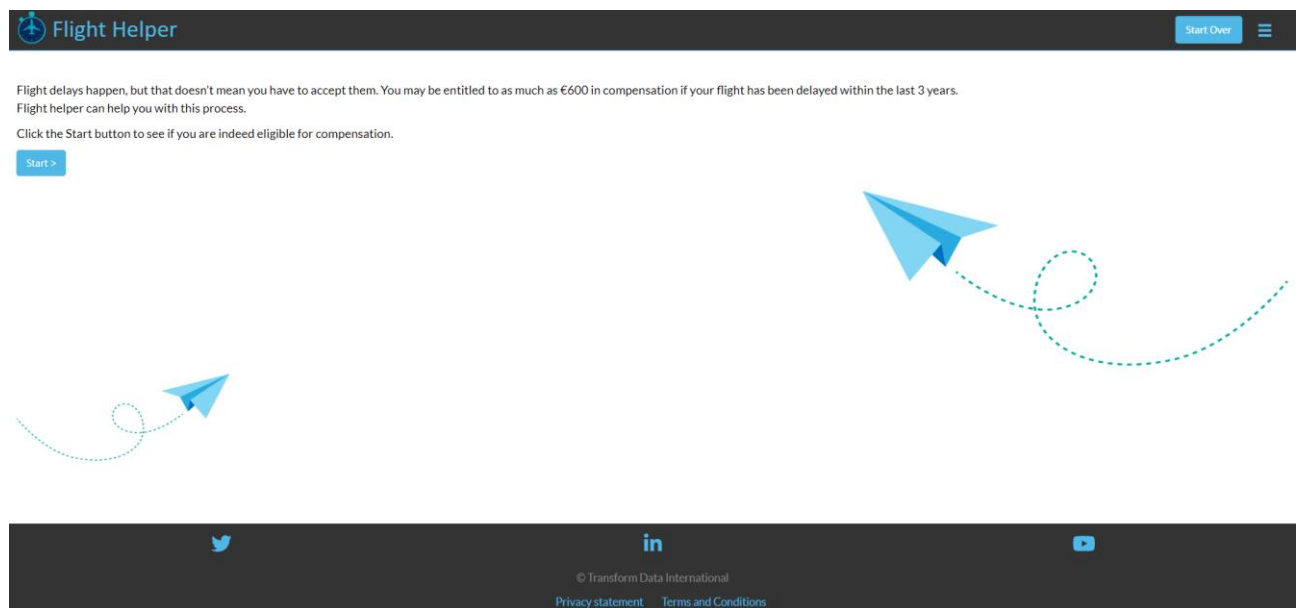


Figuur 30 Scherm voor het wijzigen van CSS in Stylizer 7

3.7 Resultaten

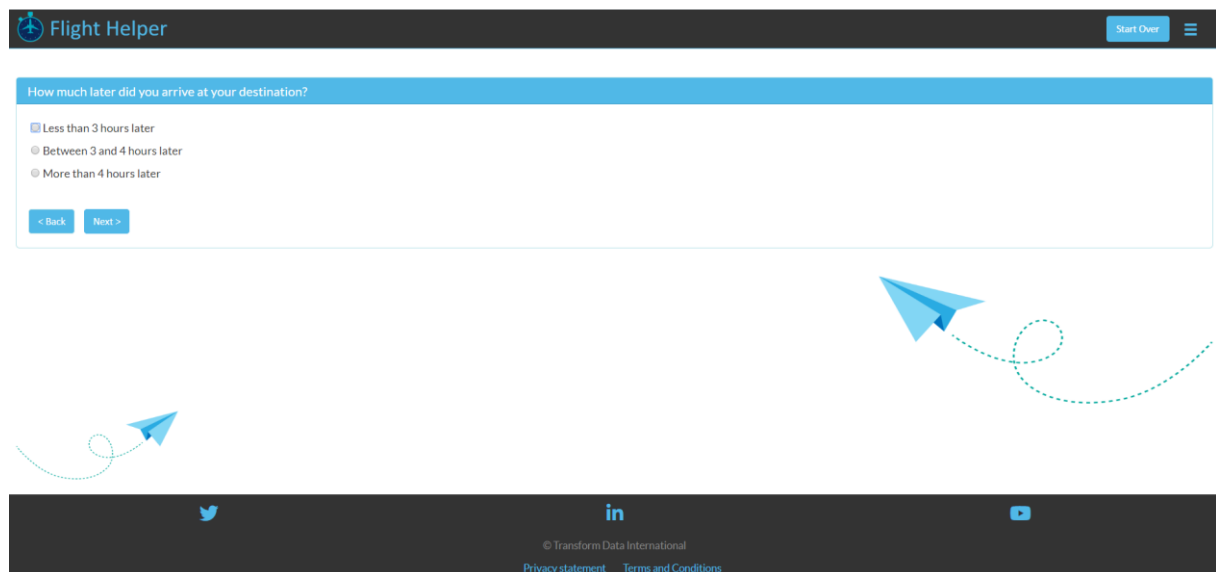
Het opgeleverde product is een webapplicatie waarbij een gebruiker door enkele vragen te beantwoorden een compensatiebrief bij vluchtvertraging kan creëren. Eerst wordt er gekeken of de gebruiker recht heeft op compensatie. Als dit het geval is kan de gebruiker ervoor kiezen om een vergoedingsbrief te genereren. Dit kan gebeuren door enkele persoonlijke gegevens alsook gegevens van de vlucht met vertraging in te geven. De brief zal gegenereerd worden als een pdf en als een Word bestand. De inhoud van de brief hangt af van de ingevulde antwoorden en maakt op basis hiervan ook enkele berekeningen.

Het eerste scherm is een introductie tot de 'flight helper'-applicatie.



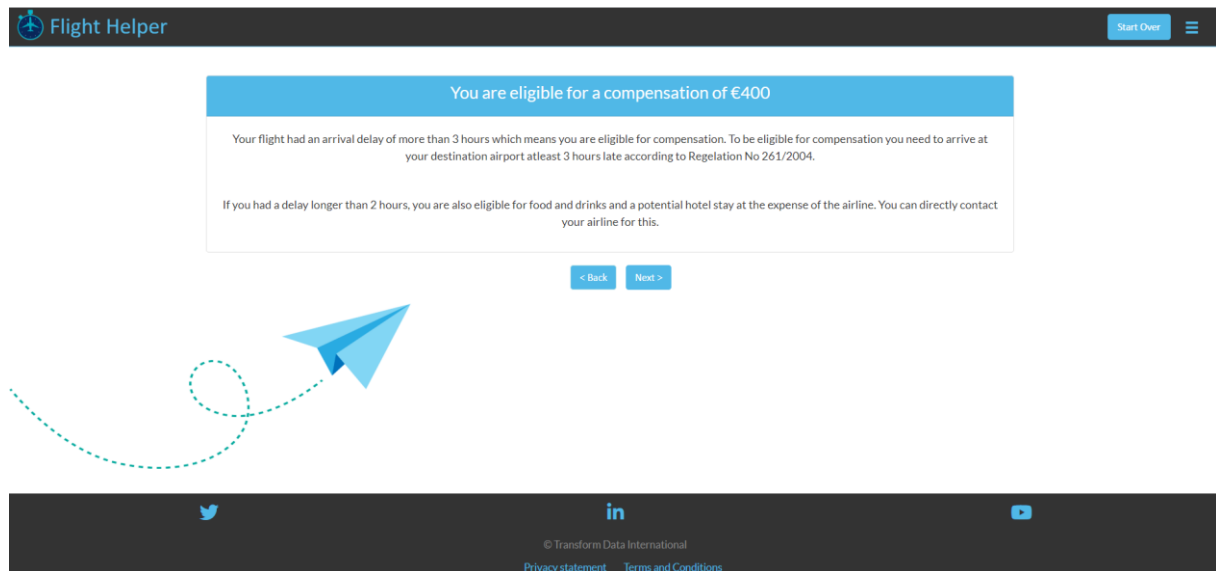
Figuur 31 Introductiescherm van de 'flight helper'-applicatie

Hierna zal een reeks van vragen aan de gebruiker gesteld worden. Op basis van de antwoorden die de gebruiker op deze vragen geeft zal de eventuele compensatie berekend worden.



Figuur 32 Vragenreeks in de 'flight helper'-applicatie

Als de gebruiker recht heeft op compensatie verschijnt er ook een *next button*. Als de gebruiker hier op klikt wordt de diegene geholpen bij het creëren van zijn of haar compensatiebrief.



Flight Helper

Start Over

You are eligible for a compensation of €400

Your flight had an arrival delay of more than 3 hours which means you are eligible for compensation. To be eligible for compensation you need to arrive at your destination airport at least 3 hours late according to Regulation No 261/2004.

If you had a delay longer than 2 hours, you are also eligible for food and drinks and a potential hotel stay at the expense of the airline. You can directly contact your airline for this.

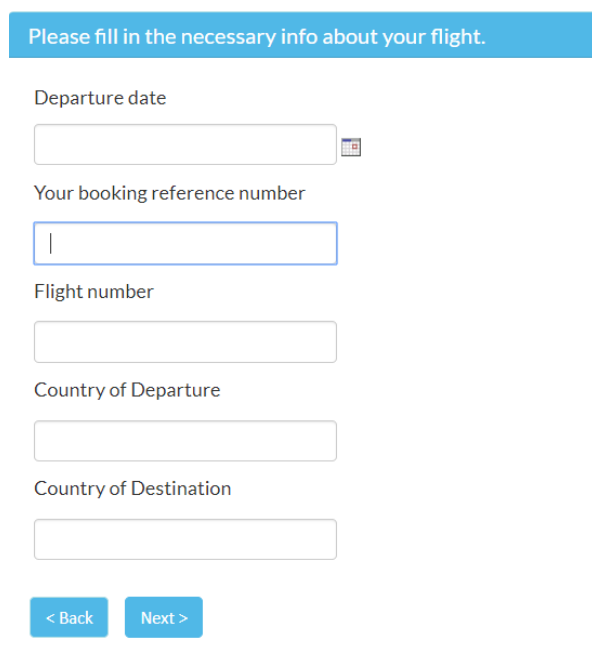
< Back Next >

Twitter LinkedIn YouTube

© Transform Data International
Privacy statement Terms and Conditions

Figuur 33 Berekening van compensatie

De gebruiker zal door enkele formulieren geleid worden die nodig zijn voor het correct genereren van de vergoedingsbrief. Op onderstaande afbeelding is een voorbeeld van zo een formulier te zien.



Please fill in the necessary info about your flight.

Departure date

Your booking reference number

Flight number

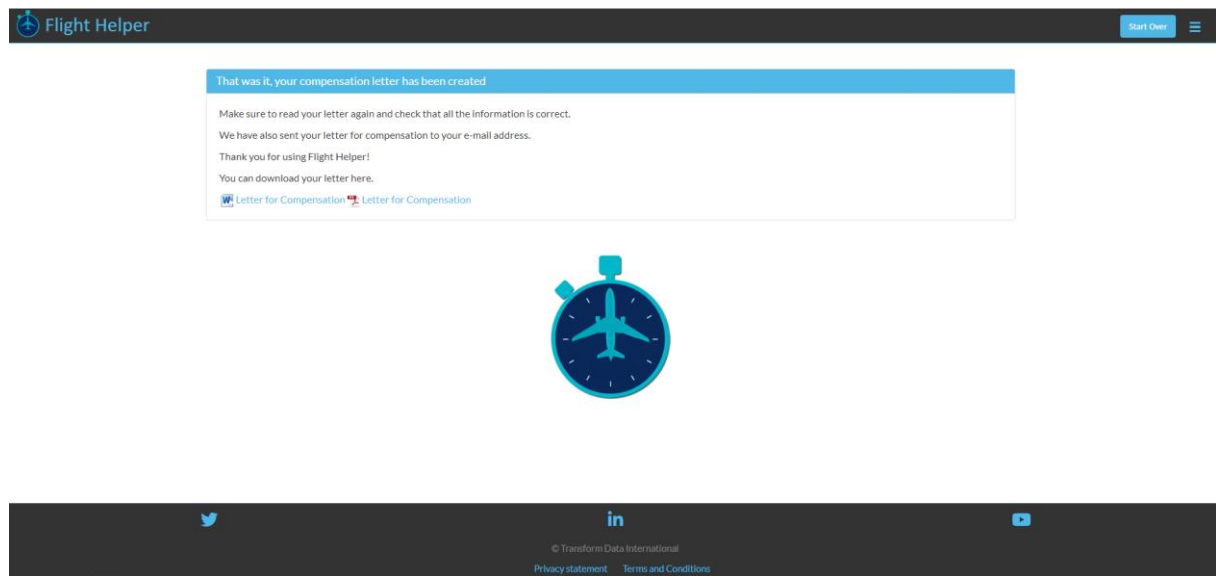
Country of Departure

Country of Destination

< Back Next >

Figuur 34 Voorbeeld van een formulier in de 'flight helper'-applicatie

Het eindscherm geeft de gebruiker de mogelijkheid om de brief voor compensatie te downloaden als een pdf of een Word document. Bovendien wordt er ook een e-mail naar de gebruiker gestuurd met daarin beide documenten als bijlage.



Figuur 35 Eindscherm van de 'flight helper'-applicatie

Compensation letter Flight Helper

info@flighthelper.be <info@flighthelper.be>
Aan: joas.rothig@gmail.com

Dear John Doe,

You have recently used Flight Helper to help you create your compensation letter.

You can find your compensation letter in the attachment of this mail.


If there are any questions regarding your compensation letter, you can write them as a reply to this mail.


Thank you for using Flight Helper!

Kind Regards,

Joas Röthig

2 bijlagen — [Alle bijlagen downloaden](#)

 **Letter for Compensation.pdf**
37K [Weergeven als HTML](#) [Downloaden](#)

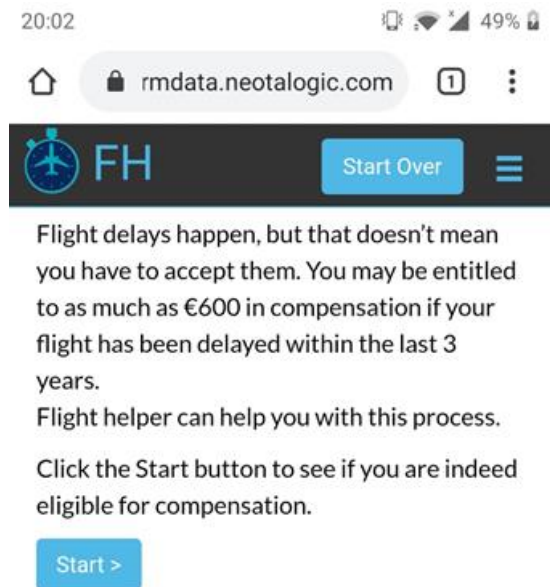
 **Letter for Compensation.docx**
22K [Weergeven als HTML](#) [Downloaden](#)

Figuur 36 Mail met bijlages als resultaat van de 'flight helper'-applicatie

Een voorbeeld van de compensatiebrief is te vinden in bijlage D.

De applicatie is ook beschikbaar op mobiele toestellen. Dit komt omdat de applicatie volledig *responsible* is. Enkel sommige afbeeldingen zijn weggehaald door middel van *media queries* om de applicatie zo overzichtelijk mogelijk voor de gebruiker te houden.

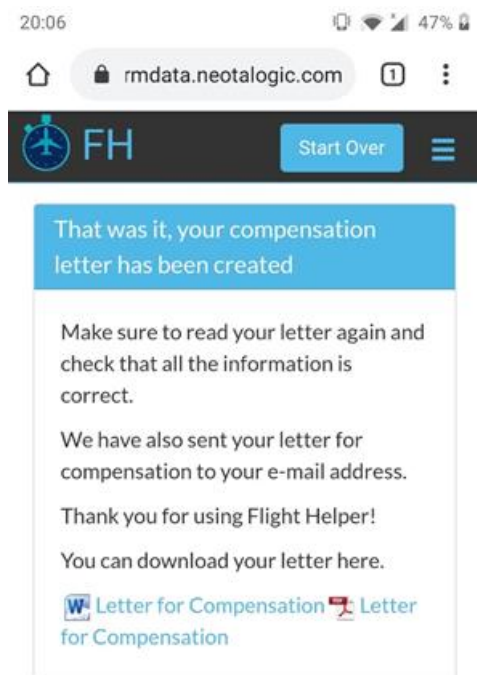
Hieronder zijn enkele afbeeldingen van de applicatie op een mobiel toestel te zien.



Figuur 37 Startscreen op een gsm



Figuur 38 Scherm voor bepaling van compensatie op een gsm



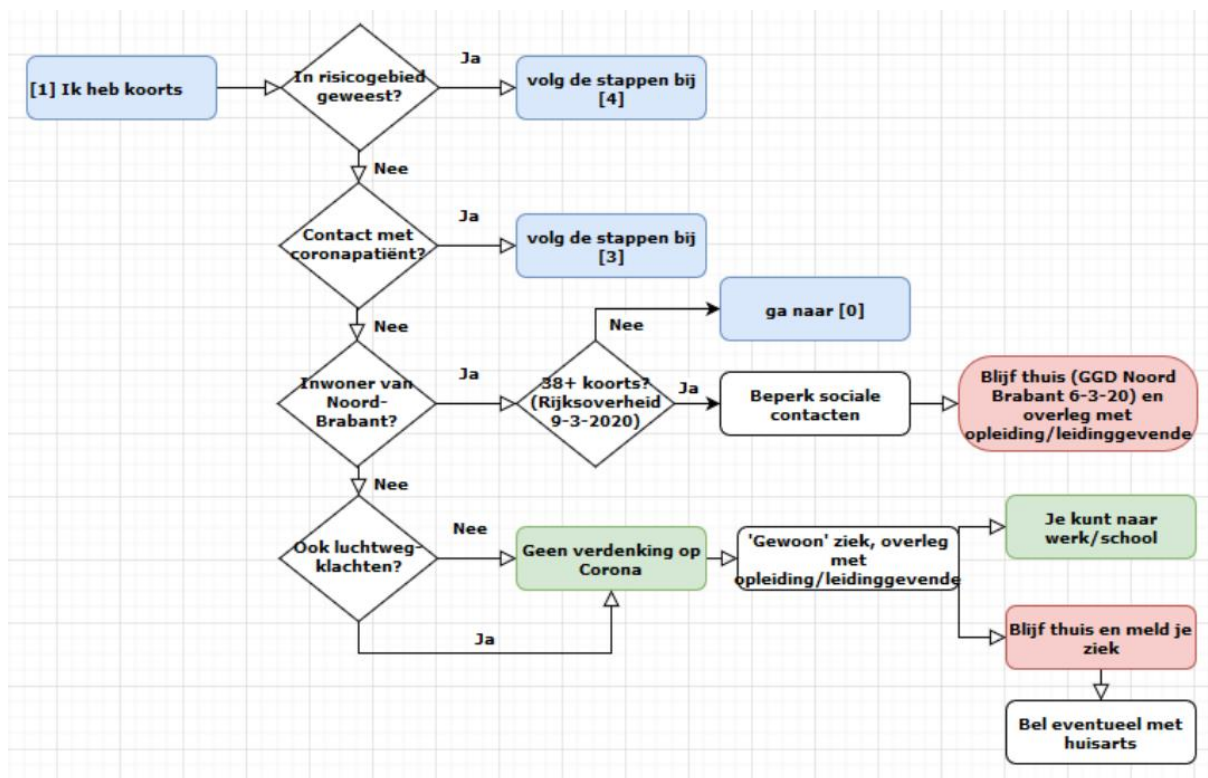
Figuur 39 Eindscherm op een gsm

4 Extra activiteiten

Gedurende de stageperiode kwamen enkele extra opdrachten aan bod. De opdrachten hielpen bij het beter begrijpen van de technologieën of onderzochten een bepaald onderdeel van Neota. Met de resultaten van de extra activiteiten is ook altijd iets gedaan. De applicaties werden gebruikt als een demo of duiden enkele problemen aan. Bovendien zijn de resultaten van de onderzoeken doorgestuurd naar de makers van Neota Logic. De feedback werd erg geapprecieerd en heeft ook geleid tot enkele verbeteringen van het Neota Logic platform.

4.1 Corona advisor

Tijdens de beginfase van het coronavirus bestonden er veel onduidelijkheden over het virus. In Nederland had elke provincie zijn eigen regels voor de aanpak van het virus. Aan de hand van enkele vragen maakt de 'corona advisor'-applicatie gebruikers duidelijk welke stappen ze moeten ondernemen als ze denken dat ze besmet zijn met het virus. De applicatie was onder andere gebaseerd op symptomen en een beslisboom van het Rijksinstituut voor Volksgezondheid en Milieu (RIVM). Op figuur 40 is een deel van die beslisboom te zien.



Figuur 40 Beslisboom van RIVM over coronavirus

Na het beantwoorden van enkele vragen krijgt de gebruiker te zien of hij al dan niet thuis moet blijven. Bovendien worden de resultaten van de applicatie opgeslagen in een NLS-database zoals te zien is op figuur 41.

<input type="checkbox"/>	Row	Geboortedatum	Provincie	Huisarts contact	Conclusie
	<input type="checkbox"/> 1	03/02/2020 00:00:00	Drenthe	Nee	Blijf thuis en meld je ziek
	<input type="checkbox"/> 2	03/02/2020 00:00:00	Flevoland	Nee	Je kunt naar werk/school
	<input type="checkbox"/> 3	03/02/2020 00:00:00	Flevoland	Nee	Je kunt naar werk/school
	<input type="checkbox"/> 4	03/04/2020 00:00:00	Overijssel	Nee	Je kunt naar werk/school
	<input type="checkbox"/> 5	03/02/2020 00:00:00	Limburg	Ja	Je kunt naar werk/school
	<input type="checkbox"/> 6	03/02/2020 00:00:00	Friesland	Ja	Je kunt naar werk/school
	<input type="checkbox"/> 7	03/02/2020 00:00:00	Zeeland	Ja	Je kunt naar werk/school
	<input type="checkbox"/> 8	03/01/2020 00:00:00	Utrecht	Nee	Je kunt naar werk/school
	<input type="checkbox"/> 9	03/01/2020 00:00:00	Flevoland	Nee	Volg aanwijzingen GGD
	<input type="checkbox"/> 10	03/03/2020 00:00:00	Zeeland	Ja	Blijf thuis en meld je ziek

Figuur 41 NLS-database van de corona advisor

Op figuur 42 is een voorbeeld van de vragen die de gebruiker dient te beantwoorden te zien. Het resultaat met de conclusie is te zien op figuur 43.

Corona Hulpverlener

Gelieve onderstaande vragen over uzelf in te vullen

Wat is uw geboortedatum?

In welke provincie woont u?

Heeft u onlangs nog contact gehad met uw huisarts?

☐ Ja

☐ Nee

Herkent u één of meerdere van onderstaande symptomen bij uzelf?

☐ Koorts

☐ Luchtwegklachten

☐ Geen van bovenstaande

Figuur 42 Voorbeeld van vragen van de corona hulpverlener

U kunt zich het beste ziek melden en thuis blijven.

We adviseren u onderstaande hygiënevoorschriften na te leven:

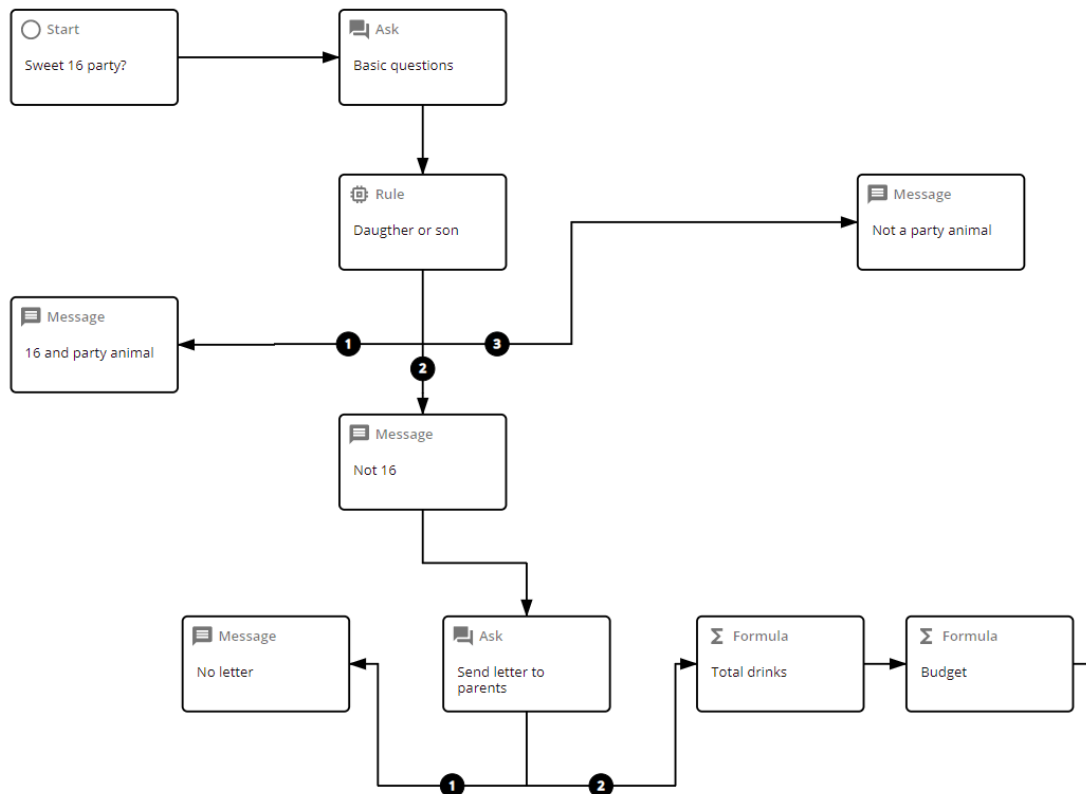
- Vermijd dicht contact met mensen die ziek zijn.
- Vermijd het aanraken van uw neus, ogen en mond.
- Blijf thuis als u ziek bent.
- Gebruik een zakdoek om uw mond te bedekken wanneer u hoest of niest, gooi daarna het zakdoekje in de vuilbak.
- Desinfecteer en maak objecten die vaak aangeraakt worden schoon met een huishoudelijke reinigingspray of doekje.
- Volg CDC's adviezen voor het gebruik van een gezichtsmasker.
 - CDC adviseert mensen die niet ziek zijn niet om een gezichtsmasker te dragen om zichzelf te beschermen tegen luchtweginfecties
 - Gezichtsmaskers moeten gebruikt worden door mensen die symptomen van COVID-19 vertonen om de verspreiding van de ziekte te voorkomen.
 - Voor gezondheidswerkers en mensen die zorgen voor iemand in nauwe omgeving (thuis of in een verzorgtehuis) is het gebruik van een gezichtsmasker ook cruciaal.
- Was uw handen vaak met zeep en water voor minstens 20 seconden, zeker na toiletgebruik, voor het eten en na het hoesten, niezen of snuiten van uw neus
 - Indien zeep en water niet beschikbaar zijn, gebruik dan een handdesinfecterend middel met minstens 60% alcohol. Was uw handen altijd met zeep als ze zichtbaar vuil zijn.

[Advies zwangerschap](#) [Advies andere gezondheidsproblemen](#)

Figuur 43 Resultaat van de corona hulpverlener

4.2 Onderzoek Canvas

Tijdens de stageperiode is er onderzoek gedaan naar de Canvas tool van Neota Logic. Het voornaamste doel van dit onderzoek was het vinden van bugs. Bovendien zijn er enkele suggesties aangegeven die de tool gebruiksvriendelijker maken en de mogelijkheden ervan uitbreiden. Er is een voorbeeldapplicatie gemaakt die de bugs in kaart brengt. Op de figuur hieronder is deze applicatie te zien.



Figuur 44 'Sweet-sixteen'-applicatie in Canvas

4.3 Onderzoek open negotiations

Open negotiations in Neota Logic is een onderdeel dat vrij recent is toegevoegd en dat nog niet veel gebruikt is. Daarom heeft TDI me gevraagd om onderzoek te doen naar dit deel. Het voornaamste doel was het in kaart brengen van de mogelijkheden en de beperkingen van open negotiations. Het kan vergeleken worden met Word review. Commentaren en reviews kunnen op een bepaald stuk tekst achtergelaten worden. Om de beperkingen en bugs aan TDI duidelijk te kunnen maken, is er een demo-applicatie gemaakt. Op onderstaande figuur is een voorbeeld van *open negotiations* te zien.

Make Some Changes

Track Changes is ON. Edit the text to see the markup for deletions and additions. To comment, click on the blue icon in the bottom toolbar:

What is Lorem Ipsum?

Lorem Ipsum is **simply dummy** text of the printing and typesetting industry. **It is used as a placeholder for text.** Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker **including versions of Lorem Ipsum.**

body div p span

Joas Rothig
now
It's not simply dummy text

Edit Delete Resolve

What would you like to do next?

continue to edit and comment

accept or reject changes and resolve comments

Figuur 45 *Open negotiations* van Neota Logic

Op de figuur is te zien dat er commentaren op bepaalde stukken tekst kan achtergelaten worden en dat er tekst verwijderd of toegevoegd kan worden. Er waren echter ook een heel aantal bugs en onduidelijkheden. Op onderstaande figuur is een e-mail naar ontwikkelaars van Neota te zien die de bugs in kaart brengt. Enkele van deze bugs zijn ondertussen al verbeterd, wat aantoont dat het onderzoek nut heeft gehad.

- Comments:
 - The comment button in standard UI is invisible, it's there but not blue as mentioned.
 - Comments in the value-display are shown as a dropped pin, the highlighted text however (which the comment is about) is hidden.
- Tracking changes/redlining:
 - Changes aren't caught in certain situations:
 - Mark-up changes (eg from **BOLD** to *Italic underlined*)
 - [ctrl]+[backspace] to remove an entire word
 - Can we track changes in a more advanced manner?
 - Whom made the change
 - At what time the change was made
 - Whom approved/rejected the change
 - At what time the approval was made
- Other:
 - Can we review a text with multiple people in parallel?

Figuur 46 Bugs van *open negotiations* in Neota Logic

4.4 Smart task list

De 'smart task list' is een applicatie die taken kan aanmaken, updaten en verwijderen. Het doel is dat verschillende applicaties de 'smart task list' kunnen aanspreken door middel van Neota as a Service (NaaS). De hoofdapplicatie slaat een taak op in een database en meerdere andere applicaties kunnen deze hoofdapplicatie aanspreken om een taak in een database op te slaan of op te halen. Deze extra opdracht begon met het opstellen van enkele *user stories*. De gehele applicatie is niet volledig afgewerkt tijdens de stageperiode. Er zijn twee NaaS-applicatie gemaakt, één die de taken uit de database kan halen en één die een taak naar een database kan aanspreken. Een volgende versie van de smart task list zou een taak kunnen updaten. Hieronder is een figuur te zien die alle taken uit de database ophaalt.

Id	Priority	Initial date	Due Date	Write to database	Owner	Responsible	App-function	Description	Status	Action	Edit
15	Low	2020-05-31 19:30	2020-06-02	Yes	Joas	René	Test app functie	Dit is een test descriptie	Open	Test action	Edit
16	Normal	2020-05-31 19:33	2020-06-16	Yes	Rob Kubben	Joas Röthig	Updaten van taak	Dit kan een taak updaten	In Progress	Database aanspreken	Edit
17	High	2020-05-31 19:35	2020-06-09	Yes	Walter Henrard	Rob Kubben	Opslaan van taak	Dit kan een taak opslaan	Closed	Aanspreken database	Edit

Create a Task

Figuur 47 NaaS die taken uit de database kan lezen

5 Besluit

Tijdens mijn stage bij Transform Data International heb ik ontzettend veel ervaringen opgedaan. Ik heb de kans gekregen om te zien hoe het er in het bedrijfsleven aan toe gaat. Bovendien heb ik zelfstandig aan een project kunnen werken met een technologie waar ik, voordat ik aan mijn stage begon, nog totaal geen ervaring mee had. Ik heb met collega's kunnen overleggen en discussiëren over ideeën. Zo heb ik ook enkele gesprekken met klanten kunnen bijwonen. Door het coronavirus heb ik remote leren werken. Dit was in het begin zeker een uitdaging. Een zekere discipline en goede werkhouding is hiervoor heel belangrijk. Dit ging steeds beter en beter. Ook door mezelf enkele regeltjes op te leggen zoals mijn gsm aan de kant te leggen terwijl ik aan het werken was. Bepaalde onderwerpen onderzoeken en hier een demo of presentatie over geven, is ook één van de zaken waar ik heel veel van heb bijgeleerd. Presentaties geven is nooit mijn sterkste punt geweest, maar door ze meer en meer te geven word je er wel beter in en begin je je er ook comfortabeler bij te voelen. Dit is absoluut nog een van mijn werkpunten. Zo moet ik bijvoorbeeld meer de leiding nemen en zelfverzekerder zijn over de dingen die ik heb onderzocht.

Mijn stage begon met het volgen van de onlinecursus van Neota Logic, die mij de basisbegrippen en technieken van het platform leerde. Daarna heb ik samen met mijn bedrijfspromotors gediscussieerd en gebrainstormd over de stageopdracht en de onderzoeksopdracht. In de tweede week ben ik begonnen met de stageopdracht, maar al vrij snel kreeg ik kans om een kleine tussenopdracht te maken. Dit was het maken van de Corona Advisor. Dit was een heel aangename opdracht aangezien ik de technieken die ik in de onlinecursus geleerd had, kon toepassen in een echt project. Hier heb ik een presentatie over gegeven en heb ik ook feedback van gekregen die ik uiteindelijk in mijn 'flight helper'-applicatie heb kunnen gebruiken. Een paar weken na de paasvakantie kreeg ik opnieuw de kans om aan nieuwe opdrachten te werken. Zoals het onderzoeken van bepaalde onderwerpen binnen Neota Logic. Open negotiations en de Canvas tool waren de voornaamste. Hier heb ik een presentatie en demo over gegeven. De conclusies die ik uit deze onderzoeken trok, heb ik als feedback naar de makers van het Neota platform kunnen sturen. Het was dan ook zeer fijn om een bericht terug te krijgen waarin aangegeven werd dat de feedback zeer bruikbaar was en dat ze enkele van deze zaken in hun platform hebben toegepast. Nadat de applicatie over vergoeding bij vluchtvertraging af was, kon ik nog beginnen aan een taken applicatie. Met deze opdracht leerde ik werken met databases binnen Neota Logic. Het eindwerk is gedurende de stage geüpdatet samen met het stageportfolio. Hierin is regelmatig feedback van zowel de lectoren als de stage- en bedrijfspromotor toegepast.

Tijdens mijn stage ben ik natuurlijk ook tegen een aantal problemen gelopen. Dingen die niet wilden werken, onzekerheden over bepaalde zaken, ook het remote leren werken tijdens de corona periode. De meeste problemen zijn opgelost door duidelijk te overleggen en vragen te stellen aan mensen van Transform Data. Ze stonden altijd klaar om mij met mijn problemen en vragen te helpen. Door zelf veel opzoekwerk te doen en trial-and-error kwam ik bijna altijd tot een oplossing.

Terugkijkend op de stageperiode kan ik besluiten dat ik de stage goed heb afgerond. Ik heb me goed aan de deadlines gehouden en heb geprobeerd de feedback altijd op te nemen en toe te passen. Tijdens mijn opdrachten ben ik altijd gedisciplineerd te werk gegaan. In het begin wilde ik vaak dingen zelf uitzoeken, wat niet per se slecht is, maar wat er wel voor zorgde dat ik vaak lang met bepaalde dingen bleef vastzitten. Dit kon makkelijk opgelost worden door vragen te durven stellen aan mijn begeleiders en dit heb ik ook meer proberen te doen. Het geven van presentaties en het overbrengen van mijn ideeën is iets waar ik naar toekomst toe wel nog aan wil werken. Hier is zeker nog ruimte voor verbetering.

Het eindresultaat is een afgewerkte opdracht: een webapplicatie die een gebruiker kan helpen bij het aanvragen van een vergoeding bij vluchtvertraging.

Ik heb ontzettend veel bijgeleerd tijdens mijn stageperiode en het heeft me ook erg gemotiveerd. Je leert veel dingen in school, maar het was heel leerrijk om te kunnen ervaren hoe het er in het bedrijfsleven aan toe gaat. Alleen vind ik het jammer dat ik het grootste deel van de stage, door de maatregelen rondom COVID-19, thuis heb moeten afwerken. Maar ook dit heeft me weer nieuwe dingen geleerd, zoals remote werken en mezelf gedisciplineerd aan mijn planning houden.

II. Onderzoekstopic

1 Vraagstelling onderzoek

Er bestaan ontelbaar veel mogelijkheden voor het maken van een webapplicatie. Voor dit project wordt gebruikgemaakt van het *no-code* platform Neota Logic, maar was dit de beste optie voor het maken van de ‘flight helper’-applicatie? Op welke manier verschilt Neota Logic met een traditionele programmeertaal zoals Java? Was Neota het beste geoptimaliseerd voor het maken van de ‘flight helper’-applicatie? In welke mate is Neota beperkt? Welke voordelen kunnen bedrijven uit een *no-code* platform halen?

2 Onderzoeksmethode

Voor de stageopdracht wordt alles vanaf het begin uitgewerkt. De technologie die gebruikt wordt, is Neota Logic. Voor het onderzoek ligt de focus op die technologie en op mogelijke alternatieven hiervoor.

Om te bepalen welke technologie het meest aangewezen is voor dit project, wordt eerst en vooral onderzocht wat de voor- en nadelen zijn van Neota Logic ten opzichte van een traditionele programmeertaal zoals Java. Ook wordt er gekeken naar low-code. Aan de hand hiervan wordt dan bepaald wanneer Neota Logic meer aangewezen is en wanneer een traditionele programmeertaal of low-code een beter alternatief is. Er wordt gekeken naar de gebruiksvriendelijkheid ervan en welke extra functionaliteiten deze technologieën hebben of juist ontbreken. Hiervoor worden artikels en reviews gelezen, maar wordt ook rekening gehouden met de eigen ervaringen. Uiteindelijk wordt er bepaald of Neota Logic inderdaad de beste keuze was voor dit project en of *no-code* ontwikkelen daadwerkelijk voordelen heeft voor bedrijven.

3 No-code

“The future of coding is no coding at all”. Dat zei de GitHub-CEO Chris Wanstrath [6]. Dit geeft eigenlijk in een notendop de essentie van *no-code* programmeren weer. Door middel van *no-code* programmeren bestaan er de mogelijkheden en middelen om softwaretoepassingen te ontwikkelen zonder daarbij ingewikkelde codes te hoeven schrijven. De meeste platforms die gebruikmaken van *no-code* bieden een reeks aan basisbouwblokken die ervoor zorgen dat er snel complexe applicaties in elkaar kunnen worden gezet [7].

No-code wordt gezien als de volgende stap in de ontwikkeling van slimme software- en applicatieoplossingen. De ontwikkelmethode stelt namelijk ook *citizen developers*, businessmensen met weinig IT-kennis die zelf software ontwikkelen, in staat om applicaties en software te bouwen.

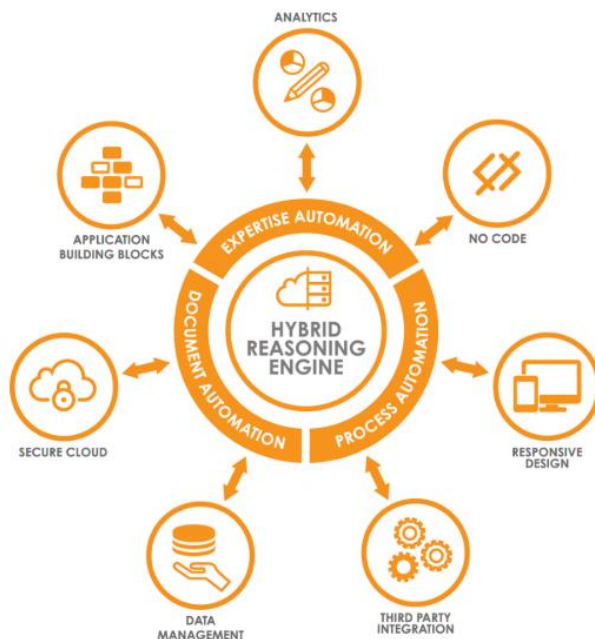
No-code legt een sterke focus op innovatie. Het wil ervoor zorgen dat gebruikers ideeën direct om kunnen zetten naar slimme applicaties. Door de diverse achtergronden van de *citizen developers* krijg je bovendien een brede en diverse innovatiestroom.

De gebruikte technologieën binnen *no-code*platformen verschillen enigszins. De meeste *no-code*platformen zijn gebaseerd op webtechnologieën als HTML en Javascript, maar er zijn ook platformen die vooral gebruikmaken van Java en .NET.

3.1 Neota Logic

Voor de stageopdracht is er gebruikgemaakt van Neota Logic. Neota Logic is een prijswinnend *no-code* artificial intelligence (AI) platform, dat ervoor zorgt dat professionals aan de hand van enkele makkelijk te gebruiken tools snel applicaties laat ontwikkelen. Hierdoor is het mogelijk voor de professionals om verschillende aspecten van hun diensten te automatiseren.

Op figuur 48 worden de verschillende onderdelen van het Neota platform getoond.



Figuur 48 Verschillende onderdelen van het Neota Logic platform

In de volgende hoofdstukken worden deze onderdelen uitgelegd alsook hoe ze in Neota gerealiseerd worden [8].

3.1.1 *No-code* ontwikkelen

Ontwikkelaars die werken met Neota Logic maken gebruik van Neota Studio. Neota Studio biedt de mogelijkheid tot volgende functionaliteiten:

- Stellen van vragen;
- Bereiken van conclusies;
- Volgen van logica;
- Genereren van onder andere documenten;
- Aanpassen van kleuren, afbeeldingen en andere elementen die zorgen voor een betere ervaring voor de gebruikers;
- Opzetten van databases.

In de studio van Neota wordt elk element van de applicatie gecreëerd en bijgehouden binnen zijn eigen visuele editor. Op die manier wordt het maken van applicaties met een heel complexe logica een stuk eenvoudiger.

3.1.2 Expertise Automatisatie

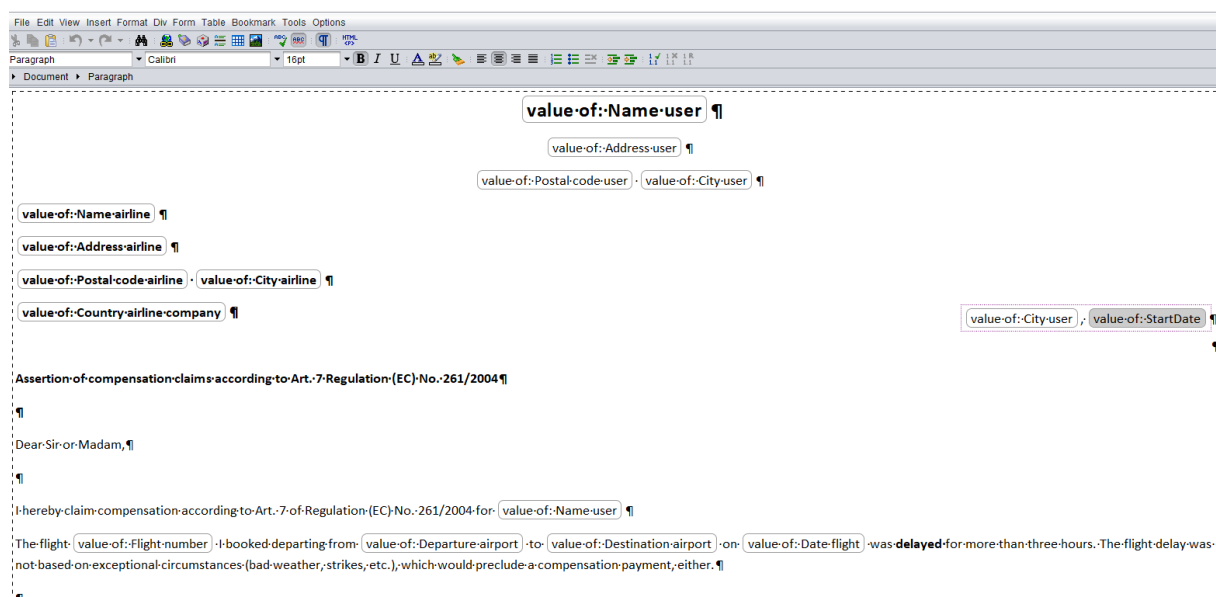
De *reasoning engine* combineert *booleans* van eender welke complexiteit met wiskundige formules, *multi-factor reasoning* en een uitgebreid aantal externe tools. Deze *reasoning engine* integreert automatisch al deze vormen van logica. Op het moment dat een applicatie aangestuurd wordt en er een probleem opgelost moet worden, wordt de *reasoning engine* aangeroepen. Ontwikkelaars kunnen grote en complexe problemen verdelen in kleinere problemen, waardoor de applicatie duidelijk blijft en de onderhoudbaarheid eenvoudiger wordt. Hierdoor kunnen ook de professionals zich focussen op kritieke problemen doordat routines geautomatiseerd worden.

3.1.3 Proces automatisatie

Expertise en documenten automatiseren is uitdagend en de mogelijkheden verschijnen meestal in de context van een businessproces. Een opdracht voltooien, een output bezorgen of een transactie uitvoeren zijn meestal processen die meer dan één stap hebben en door meer dan één persoon uitgevoerd moeten worden. Neota heeft de mogelijkheid tot integratie waardoor applicaties gestart kunnen worden of data kunnen gebruiken van business systemen zoals Salesforce.

3.1.4 Document automatisatie

Neota Logic biedt ook de mogelijkheid tot het automatiseren van documenten. Er kan logica in de documenten geïmplementeerd worden waardoor het flexibeler wordt om documenten te genereren. Bovendien zijn de bestanden ook aanpasbaar door de thema-editor en is het ook mogelijk om met CSS-bestanden te werken. Op figuur 49 is een voorbeeld van een flexibel document binnen Neota Logic Studio te zien.



Figuur 49 Document automatisatie in Neota Logic

3.1.5 Applicatie design

Ieder element van de userinterface wordt gedreven door de *reasoning engine* van Neota en door data van de applicatie. Dat wil zeggen dat de volgorde van vragen, de tekst van vragen, antwoordmogelijkheden en de lay-out van de pagina allemaal worden bepaald door regels die door de maker van de applicatie gevormd worden. Wanneer een gebruiker door een applicatie gaat, wordt het volgende scherm getoond op basis van vorige antwoorden die de gebruiker gegeven heeft

alsook op basis van het profiel van de gebruiker en andere databronnen. Enkel de informatie die nodig is, wordt aan de gebruiker gepresenteerd.

Neota heeft ook de mogelijkheid om het design van een userinterface aan te passen. Dat kan door middel van de thema editor binnen Neota, maar ook door zelfgemaakte CSS-bestanden. Met de thema editor is het onder andere mogelijk om de lay-out, kleuren, randen, achtergronden en logo's aan te passen. Doordat Neota Logic Studio (NLS) gebruikmaakt van HTML5 wordt de userinterface correct getoond op desktop, tablet, laptop en gsm.

3.1.6 Applicatie bouwblokken

Neota maakt gebruik van modulaair programmeren. Het gezegde 'divide and conquer' oftewel 'verdeel en heers' is een veelgebruikte aanpak die gebruikt wordt voor het oplossen van grote problemen. Eén groot probleem verdelen in kleinere problemen, deze kleinere problemen dan onafhankelijk van elkaar oplossen en uiteindelijk alle delen bij elkaar voegen, wordt veel gebruikt bij het ontwikkelen van software. Dit wordt ook wel modulaair programmeren genoemd.

Neota Logic zorgt voor modulaair programmeren op drie manieren. Ten eerste door de applicatie op te delen in 'Issues'. Elk 'Issue' staat voor één aspect van het probleem. Ten tweede kan een 'Issue' gedefinieerd worden als 'Reusable'. Op deze manier kan één actie op meerdere delen van de applicatie uitgevoerd worden. Ten derde is er Neota as a Service (NaaS). Met NaaS kan een applicatie door één persoon gemaakt worden en deze is daarna als module beschikbaar voor meerdere andere ontwikkelaars. Neota heeft ook al een kleine *library* van NaaS-applicaties voor veelgebruikte functies die door ontwikkelaars op elk moment gebruikt kunnen worden.

3.1.7 Third-Party integratie

Eén van de kernprincipes van Neota Logic is dat goede oplossingen vaak gebruikmaken van meerdere tools. Daarom is Neota Logic gedesignd om *third-party* tools te integreren. Applicaties kunnen data lezen van en schrijven naar onder andere Salesforce en SharePoint, maar ook naar en van alles met een Representational State Transfer (Rest) API.

3.1.8 Datamanagement

Business operaties omvatten heel veel data. Data van klanten, werknemers, beslissingen, acties en nog veel meer. Neota applicaties verkrijgen data door gebruikers vragen te stellen, logica op deze data toe te passen en door nieuwe data zoals documenten en conclusies te creëren. Daarom heeft Neota Logic een Neota Data Manager (NDM). Met NDM kunnen ontwikkelaars applicaties ontwikkelen die kunnen lezen van en schrijven naar Microsoft SQL Server zonder dat er code geschreven moet worden. NDM genereert automatisch de database met alle tabellen en relaties die nodig zijn voor het beheren van de data van iedere Neota applicatie. Als de structuur van de applicatie wordt aangepast, bijvoorbeeld wanneer een variabele wordt toegevoegd of verwijderd, dan wordt de database automatisch geüpdatet en gesynchroniseerd. Er is dus ook geen kennis van Structured Query Language (SQL) nodig.

3.1.9 Secure Cloud

Neota is een Platform as a Service (PaaS). Alle elementen van het platform worden beheerd door Neota Logic. Klanten kunnen hosting aanvragen bij Amazon Web Services (AWS) of Microsoft Azure in elke regio waar deze diensten beschikbaar zijn.

Data wordt geëncrypteerd tijdens de doorvoer en in rust. Data van klanten kan opgeslagen worden in databases die beheerd worden door Neota in hun Virtual Private Cloud (VPC), AWS, Microsoft Azure of in databases van klanten in hun eigen VPC's.

Checkpoints van applicaties worden automatisch opgeslagen wanneer aanpassingen worden gemaakt en kunnen ook altijd manueel aangemaakt worden door de ontwikkelaar zelf. Hierdoor kan er verzekerd worden dat ontwikkelaars altijd naar een vorige versie kunnen terugkeren.

4 Low-code

Low-code is een vorm van softwareontwikkeling die gericht is op het visuele ontwerpen van applicaties. Bij low-code wordt net zoals bij *no-code* gebruikgemaakt van een Graphical User Interface (GUI) en het instellen van configuraties in plaats van het schrijven van traditionele computercode.

Het proces van softwareontwikkeling kan versneld worden, omdat applicaties opgebouwd worden uit Visual Programming Language (VPL) blokken. De mogelijkheid voor ontwikkelaars om handgeschreven code in te voegen blijft bestaan in tegenstelling tot *no-code*.

De bekendste platforms voor het ontwikkelen met low-code zijn Mendix en OutSystems.

Low-code is kort uitgelegd aangezien het hetzelfde doel heeft als *no-code*. Beiden vormen van ontwikkeling willen organisaties helpen door een mogelijkheid te bieden voor het sneller implementeren van nieuwe functionaliteiten. Beiden hebben hun voor- en nadelen, maar deze zijn vaak afhankelijk van project tot project. *No-code* wordt het meest gebruikt voor simpelere business cases of manuele processen die geen *3rd party* systemen nodig hebben, alhoewel dit vaak toch kan in *no-code* platformen. Zo biedt Neota Logic bijvoorbeeld ook de mogelijkheid tot *3rd party* integratie. Low-code biedt een mechanisme voor programmeurs om eigen code toe te voegen om een functionaliteit te creëren die nog niet in het platform beschikbaar was. [9]

Er zijn een heel aantal verschillende *no-code* en low-code platformen. Elk van deze platformen heeft zijn eigen functionaliteiten en het is dus ook afhankelijk van elk platform alsook de organisatie en het project om te kunnen constateren wat de beste oplossing is voor een bepaald probleem.

5 Traditioneel programmeren

Het traditionele programmeren is het schrijven van een computerprogramma. Dit is een reeks van instructies die een computer kan uitvoeren. Softwareontwikkelaars of programmeurs zijn de personen die zorgen voor het schrijven van deze computerprogramma's. De programmacode die geschreven wordt heet broncode. Deze broncode wordt door een assembler, compiler of *interpreter* omgezet in machinecode.

Door compilers die dezelfde programmeertaal omzetten naar verschillende machinetalen, wordt het mogelijk om programma's te schrijven die niet gebonden zijn aan één specifieke processor. Bovendien zijn programma's geschreven met een programmeertaal makkelijker te begrijpen en te wijzigen aangezien een programmeertaal een hoger niveau van abstractie heeft dan een machinetaal.

Er bestaan duizenden programmeertalen [10], waarvan de meesten, die nog in gebruik zijn, nog voortdurend gewijzigd worden.

Doorheen het onderzoek en de vergelijking van een traditionele programmeertaal met het *no-code* platform van Neota Logic, wordt vooral gekeken naar Java. Java is één van de populairste programmeertalen [11]. Het is dan ook daarom dat er gekozen wordt voor Java tijdens dit onderzoek. Bovendien zijn er ook eigen ervaringen met deze programmeertaal.

5.1 Java

Java is een platformonafhankelijke taal die op gebied van syntaxis voor een groot deel is gebaseerd op de programmeertaal C++. Java beschikt echter over een grotere klassenbibliotheek dan C++.

De objectgeoriënteerde programmeertaal ontstond in het begin van de jaren 90 in een klein dochterbedrijf van Sun Microsystems onder leiding van de softwareontwikkelaar James Gosling. Java begon als een project genaamd 'Oak' in juni 1991. Deze naam 'Oak' werd gekozen toen men bij het zoeken van een naam naar buiten keek en daar een eik zag staan. Er bleek echter al een programmeertaal met die naam te bestaan, dus werd er gekozen voor 'Java', een verwijzing naar koffie. Later bouwden ze verder op deze terminologie met onder andere 'JavaBeans' wat staat voor koffiebonen.

De eerste publieke implementatie van Java was in 1995. Hierbij werd het principe van Write Once, Run Anywhere (WORA) beloofd. Het was behoorlijk veilig en de beveiliging was configureerbaar. Dit kon ervoor zorgen dat toegang tot netwerk en bestanden gelimiteerd was.

Aanvankelijk wilde men Java promoten als programmeertaal voor allerlei elektronische apparaten, zoals televisies, afstandsbedieningen en koelkasten. Toen het wereldwijde web steeds populairder werd, bedacht Sun dat we hun taal goed konden gebruiken in een webomgeving. Dankzij de open specificatie en de mogelijkheid om een Javaprogramma te embedden was de hype al snel gecreëerd.

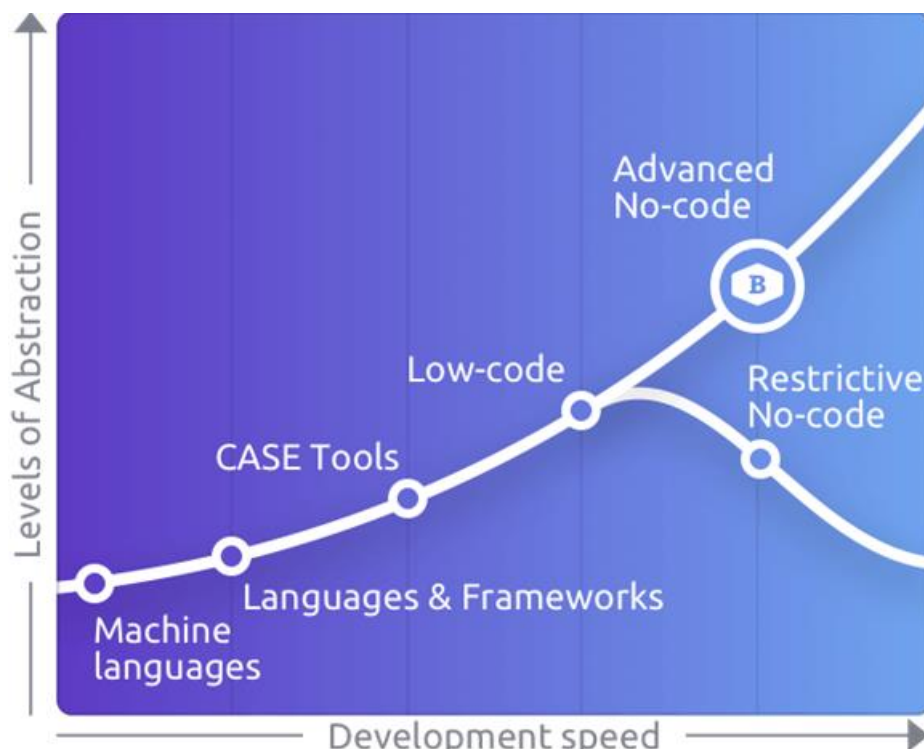
Op 13 november 2006 gaf Sun delen van Java vrij als opensourcesoftware, onder de GNU General Public License (GPL). Op 8 mei 2007 gaf Sun de laatste delen van Java vrij onder de GPL, op enkele kleine delen na waar Sun niet het auteursrecht op heeft [12].

6 *No-code* ten opzichte van traditioneel programmeren

6.1 Voordelen van *no-code*

No-code ontwikkelen is aantrekkelijk, omdat het gemakkelijk te gebruiken is en het ervoor zorgt dat er sneller applicaties gemaakt kunnen worden. Het kan zeer bruikbaar zijn voor zowel ontwikkelaars als de business [13].

In het algemeen zorgt het ontwikkelen met een *no-code* platform voor een snellere ontwikkeltijd, maar in ruil daarvoor is er een hoger abstractieniveau. Hierdoor is er vaak minder ruimte voor vrijheid bij het ontwikkelen. Dit is ook wat te zien is op onderstaande grafiek.



Figuur 50 Grafiek van abstractie ten opzichte van ontwikkeltijd

6.1.1 Snelheid

Meeste *no-code* platformen maken gebruik van visuele bouwblokken. Dit betekent dat het bouwen van applicaties een stuk sneller is. In het algemeen is het testen geautomatiseerd, waardoor de ontwikkeltijd ook naar beneden gaat.

6.1.2 Lagere kosten

Om te werken met een *no-code* platform is er in tegenstelling tot traditioneel programmeren vaak een licentie nodig. Deze licenties zijn vaak niet goedkoop, maar ontwikkelaars zijn duur. *No-code* kan een manier zijn om dit te ontwijken, omdat er niet altijd een team van ervaren ontwikkelaars nodig is. Hierdoor kunnen sneller applicaties gebouwd worden, wat ervoor zorgt dat het op de lange termijn vaak goedkoper is.

6.1.3 Hogere productiviteit

Omdat applicaties sneller gemaakt kunnen worden, gaat ook de productiviteit omhoog. Wat voorheen enkele maanden duurde, kan nu in enkele dagen of weken.

6.1.4 Makkelijk aanpassen

Het probleem met traditioneel programmeren is dat het vaak moeilijk is om een bepaalde functionaliteit aan te passen. Zeker als het gaat om een programmeertaal die onbekend is voor de ontwikkelaar. Met *no-code* is het aanpassen van bepaalde logica eenvoudiger, omdat er nieuwe logica geïmplementeerd kan worden binnen een zeer korte tijd.

6.2 Nadelen van *no-code*

De voordelen van *no-code* zijn aantrekkelijk, maar het is belangrijk om te weten dat risico's aan verbonden zijn. Het kan zijn dat iets op de korte termijn zeer kostenefficiënt is, maar dat het

financiële plaatje op lange termijn flink kan oplopen. Het is daarom altijd cruciaal om de totale kosten van het eigendom bij elk softwareproject te overwegen.

6.2.1 Vereisten

Alle *no-code* platformen zijn verschillend. Het is daarom belangrijk dat elke gebruiker eerst bepaald of zijn vereisten passen binnen de beperkingen van een bepaald platform. Zelfs dan bestaat er het risico dat de vereisten veranderen en niet meer binnen het platform passen.

6.2.2 Limieten bij het ontwikkelen

Als het gaat om het uitbouwen van bepaalde functionaliteiten van een applicatie dan hebben *no-code* platformen verschillende templates en componenten die geconfigureerd kunnen worden om bepaalde cases uit te voeren. Maar het risico bestaat dat een bepaalde functionaliteit niet binnen deze case valt. Het is dan ook niet altijd een optie om deze functionaliteit te laten vallen. Hierdoor kan de focus veranderen van wat het idee is om te bouwen, naar wat mogelijk is om te bouwen.

6.2.3 Niet beheren van de *source code*

Wanneer er bepaald wordt om weg te stappen van het *no-code* platform dan gaat dit vaak gepaard met het onvermogen om van provider te veranderen. Zelfs als het mogelijk is om van provider te veranderen, zorgt dit vaak voor een heel aantal extra kosten. Alle kansen voor het onderhouden van de applicatie liggen in de handen van de originele verkoper.

6.3 Neota Logic ten opzichte van Java

Om de voor- en nadelen van het *no-code* platform Neota Logic te vinden, worden enkele onderdelen vergeleken met die binnen een traditionele programmeertaal. Er is gekozen om Neota Logic te vergelijken met Java, omdat Java een van de populairste programmeertalen is.

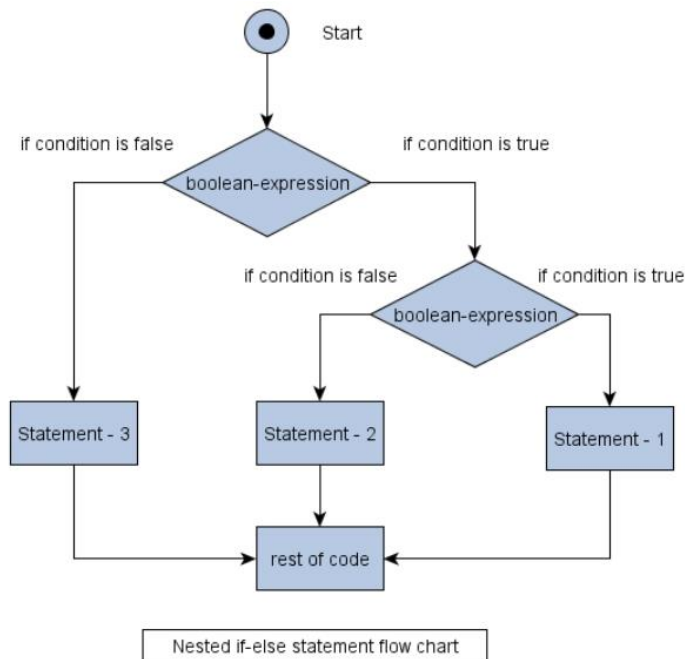
6.3.1 If-else statements

De *if-else* statements zijn de eenvoudigste vorm om een bepaalde flow te controleren. Het vertelt het programma om een bepaald stuk van de code uit te voeren als een bepaald deel van het statement waar of onwaar is. Op onderstaande figuur is een voorbeeld uit de Java documentatie te zien [14]. Als 'isMoving' waar is, dan wordt de 'currentspeed' verminderd. Indien 'isMoving' onwaar is dan wordt het gedeelte van het *else* statement uitgevoerd.

```
if (isMoving) {  
    currentSpeed--;  
} else {  
    System.err.println("The bicycle has already stopped!");  
}
```

Figuur 51 Voorbeeld if-then-else

De *nested if-else* statements worden gebruikt als er een serie van beslissingen gemaakt moet worden. Dit wil zeggen dat één *if-else* statement binnen een andere *if-else* statement zit. Op onderstaande figuur is een flow chart van een *nested if-else* te zien.



Figuur 52 Flowchart van een *nested if-else statement*

Nested if-else statements kunnen ook in Neota Logic Studio gebruikt worden met behulp van beslisbomen. Het idee hierachter is precies hetzelfde als in Java. Afhankelijk van de conditie wordt een bepaald gedeelte uitgevoerd. Het grote verschil tussen Java en Neota Logic hierin is dat het op een visuele manier in Neota Logic wordt getoond. Dit zorgt voor een duidelijker beeld als het gaat om situaties met meerdere *if-else* statements. Op figuur 53 is een situatie in Java te zien met meerdere geneste *if-else* statements. Dit kan voor een onduidelijk beeld zorgen dat moeilijk te begrijpen is. Hoe meer *if-else* statements hoe onduidelijker de code wordt. Bovendien wordt het moeilijker om statements toe te voegen of om condities aan te passen. Dit is zeker het geval voor personen die de code niet zelf hebben geschreven.

```

System.out.println("Beslissingsboom Studievoortgang\n");
Scanner keyboard = new Scanner(System.in);
System.out.println("Studeerde je reeds aan Hogeschool FXL?");
String antwoord = keyboard.nextLine();
if (antwoord == "ja") {
    System.out.println("Had je reeds individueel bindende voorwaarden?");
    antwoord = keyboard.nextLine();
    if (antwoord == "ja") {
        System.out.println("Heb je individueel bindende voorwaarden behaald?");
        antwoord = keyboard.nextLine();
        if (antwoord == "ja") {
            System.out.println("Je kan je onlibne herinschrijven via mijn FXL, indien je leerkrediet te klein is");
        } else {
            System.out.println("Kreeg je positief advies van de voortgangscommissie?");
            antwoord = keyboard.nextLine();
            if (antwoord == "ja") {
                System.out.println("behaalde je tenminste 60% studierendement?");
                antwoord = keyboard.nextLine();
                if (antwoord == "ja") {
                    System.out.println("Je kan je opnieuw herinschrijven via mijn FXL tenzij je een tekort aan leerkrediet hebt");
                } else {
                    System.out.println("Je moet een afspraak maken bij de dienst studievoortgang");
                }
            }
        }
    }
}
  
```

Figuur 53 *Nested if-else* in Java

In Neota Logic worden de statements en condities aan de hand van een visuele beslisboom gedaan. Door het visuele aspect is de flow duidelijker. Mensen kennen het principe van een beslisboom

waardoor het makkelijker wordt om die flow te begrijpen. Hierdoor is het ook gemakkelijker om aanpassingen aan te brengen of om de beslisboom uit te breiden.

6.3.2 Databases

Om een Java-applicatie met een database te connecteren via Java DataBase Connectivity (JDBC) zijn er vijf stappen [15]:

- Registreer de driver
- Verkrijg een connectie
- Creëer een statement
- Voer de query uit
- Sluit de connectie

De 'forName()' methode van 'Class' wordt gebruikt om de driver klasse te registreren. De methode wordt gebruikt om de klasse dynamisch te laden. Op de figuur hieronder is een voorbeeld te zien waar de Oracle driver geladen wordt om een database connectie te verkrijgen.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Figuur 54 Laden van Oracle driver

Hierna moet er een connectie met database verkregen worden. Dit gebeurt aan de hand van de 'getConnection()' methode. Hiervan is op onderstaande figuur een voorbeeld te zien.

```
Connection con=DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

Figuur 55 Connectie maken met de database

Als derde moet er een statement gecreëerd worden. Het object van het statement is verantwoordelijk om query's met de database uit te voeren. Hieronder is een voorbeeld van het creëren van een statement te zien.

```
Statement stmt=con.createStatement();
```

Figuur 56 Creëren van een statement in Java

Stap vier is het uitvoeren van de query. Hiervoor wordt de 'executeQuery()' methode gebruikt. Deze methode geeft het object van de 'ResultSet' terug die gebruikt kan worden om alle data van een tabel te verkrijgen. Op figuur 57 is te zien dat alle gegevens van de tabel 'emp' opgevraagd worden. Hierna worden al de resultaten afgeprint.

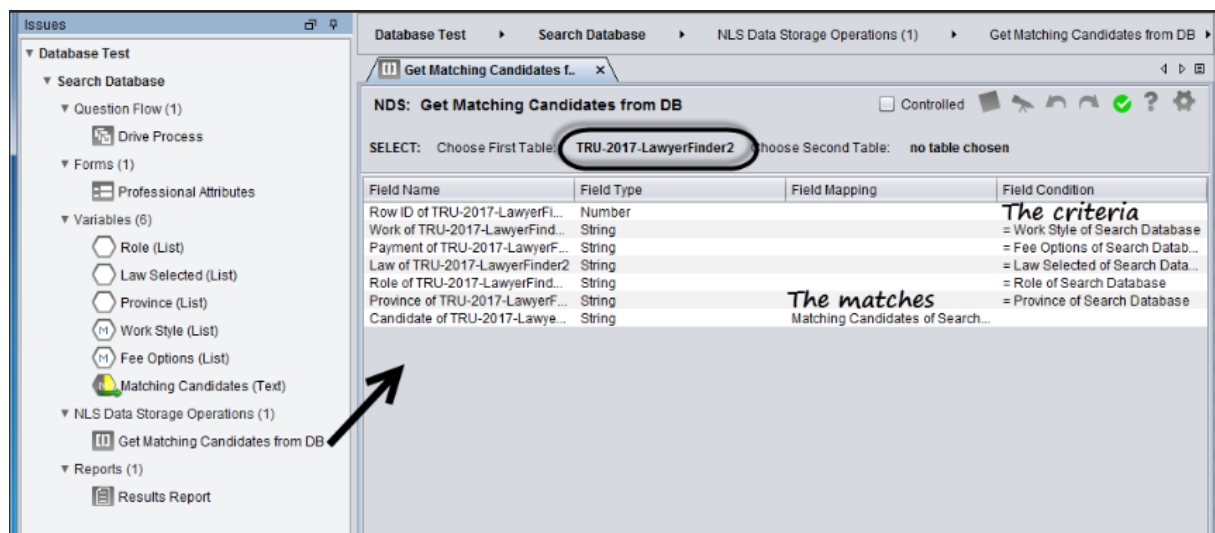
```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Figuur 57 Ophalen en afdrukken van gegevens

Als laatste moet de connectie gesloten worden. Dit kan gedaan worden door de connectie te sluiten, waardoor ook automatisch het statement en de 'ResultSet' gesloten worden. 'Con.close()' is hoe de connectie gesloten wordt.

In Neota Logic het mogelijk om te werken met de Neota Data Store (NDS) of de Neota Data Manager (NDM). NDS is een database voor Neota Logic applicaties. De Neota applicaties kunnen data in een NDS-tabel zetten en opvragen. Het is mogelijk om de NLS-database vanuit de werkbank van Neota Logic te beheren. Tijdens het maken van de corona applicatie, die beschreven is onder extra activiteiten, is gebleken dat NDS duidelijk en makkelijk is om mee te werken. Bij de corona applicatie werden gegevens van de applicatie in een NDS-database opgeslagen. Hieronder is te zien hoe gegevens uit een NDS gehaald kunnen worden binnen de Neota Logic Studio.



Figuur 58 Ophalen van gegevens uit NDS

De tweede mogelijkheid voor databases In Neota is NDM. Met NDM is het mogelijk om een Neota applicatie te connecteren met een externe database. Dit is gedaan tijdens de extra activiteit van de Smart Task App. In tegenstelling tot NDS is de documentatie van NDM zeer beperkt. Dit maakt het ook moeilijker om mee te werken. Bovendien is er ook een bug waardoor de data niet in de Neota werkbank getoond wordt. De data kan niet geladen worden wat werken met NDM vermoelijkijkt. De bug is gerapporteerd aan Neota en staat op de lijst om verbeterd te worden. Bovendien wordt er ook geen Unique Identifier (UID) bij objecten aangemaakt, wat het moeilijk maakt om een bepaald object te updaten.

6.3.3 Document generatie

In Java kunnen Microsoft documenten gegenereerd worden met behulp van Apache POI. Apache POI is een populaire API die programmeurs toelaat om MS Office bestanden te creëren en aan te passen door gebruik te maken van programma's in Java. Op figuur 59 is te zien hoe een leeg Word document gemaakt kan worden met behulp van de open source library Apache POI.

```

import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.XWPFDocument;

public class CreateDocument {

    public static void main(String[] args) throws Exception {

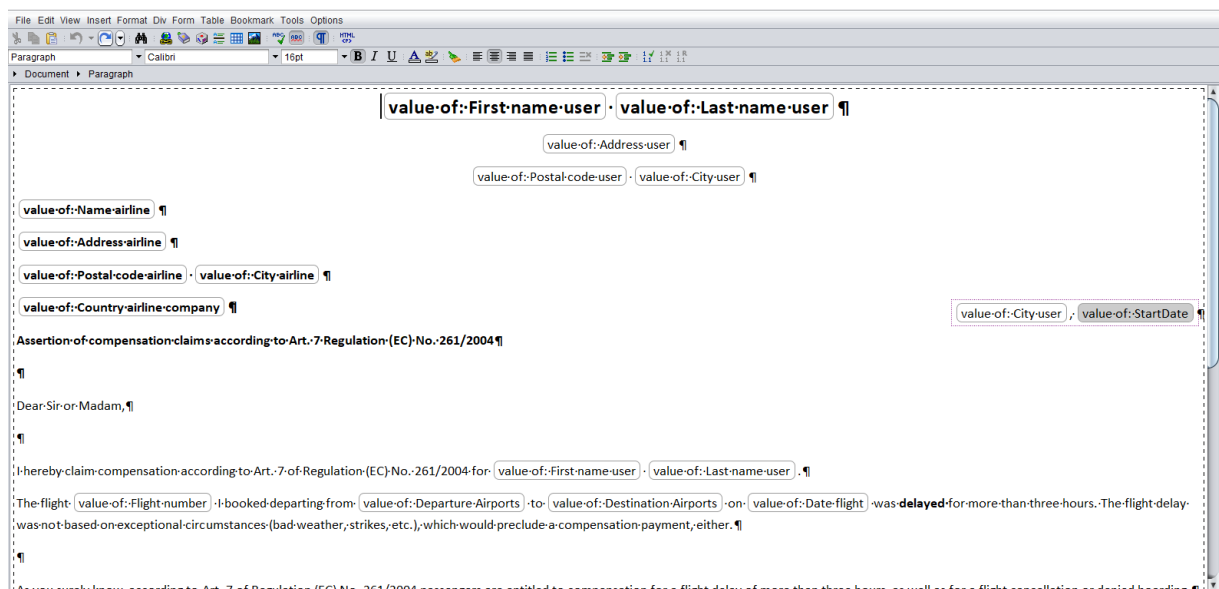
        //Blank Document
        XWPFDocument document = new XWPFDocument();

        //Write the Document in file system
        FileOutputStream out = new FileOutputStream( new File("createdocument.docx"))
        document.write(out);
        out.close();
        System.out.println("createdocument.docx written successully");
    }
}

```

Figuur 59 Generen van Word document met behulp van Apache POI

Neota Logic heeft een eigen tool ‘Reports’ voor het genereren van zowel Word documenten als PDF-documenten. Het heeft de standaardfunctionaliteiten van elke teksteditor zoals het aanpassen van het lettertype en het veranderen van de lettergrootte. Het is duidelijk om mee te werken en het kan gemakkelijk waardes van variabelen in het document verwerken. Op onderstaande figuur is de editor van Neota Logic te zien.



Figuur 60 Teksteditor van Neota Logic

6.3.4 Documentatie

Zowel Java als Neota Logic hebben een uitgebreide documentatie. Enkel is Neota Logic soms wat beperkt bij bepaalde onderdelen zoals bijvoorbeeld bij NDM. Het grote verschil is echter dat er veel minder informatie over Neota Logic op het internet gevonden kan worden in vergelijking met Java. Dit komt omdat Java één van de populairste programmeertalen is en omdat Neota Logic een heel aantal minder gebruikers heeft. Neota moet het daarom hebben van zijn eigen handleiding

aangezien er geen informatie over staat op websites zoals StackOverflow, een vraag- en antwoordenwebsite voor enthousiaste programmeurs.

6.4 Key advantages of No Code for complex decision support

Iets programmeren vanaf het begin is een heel werkintensief proces. Het vereist een set van skills waar een grote vraag naar is. In het artikel “Key advantages of No Code for complex decision support”, geschreven door Tom Routen op het *online publishing platform* Medium, wordt programmeren vergeleken met het bouwen van de Notre Dame kathedraal met lucifers. Het kan goed zijn dat er een duidelijk beeld is over hoe mooi de kathedraal er gaat uitzien, maar dit betekent ook dat het proces van het maken van de kathedraal een fragiel en tijdrovend werk is [16].

Onder het hoofdstuk “No code pros and cons” maakt Tom in zijn artikel duidelijk dat *no-code* niet alleen, of zelfs meestal, voor niet-programmeurs is. Hij schrijft dat *no-code* een manier kan zijn om software sneller en robuuster te schrijven. Het kan de productiviteit van een programmeur verbeteren.

Wanneer er over de nadelen van een *no-code* aanpak verteld wordt, wordt er teruggekeken naar het voorbeeld van de Notre Dame. “It can feel a bit like building a Notre Dame model with bigger blocks than matchsticks: you can build the cathedral more quickly and easily, but you can’t portray the gargoyles in loving detail.”. Het kan in feite gezien worden als een trade-off. Er wordt een deel van de flexibiliteit ingeleverd, maar in plaats daarvan kan een programma sneller en gemakkelijker ontwikkeld worden.

Ook het argument van een trade-off wordt weerlegd. Er zijn verschillende *no-code* platformen, sommigen laten meer flexibiliteit en expressiviteit toe. Zo zijn er omstandigheden waar *no-code* helemaal geen trade-off is. Soms is *no-code* gewoonweg de beste manier om bepaalde software te creëren.

Een taak waarbij de opdracht het bouwen van een medisch diagnose en behandelingssysteem is, wordt als voorbeeld genomen. Dit systeem moet mensen helpen bij het volgen van klinische richtlijnen en best practices bij het behandelen en verzorgen van een ziek kind. Hierbij is de medische dokter de specialist die de kennis heeft waarop de applicatie gebaseerd moet worden. In het hoofdstuk “The Doctor, the Programmer and the validation gap between them” wordt aangetoond hoe het softwareontwikkelp proces hiervan eruit ziet.

In normale programmeermethodes bestaan er meestal minstens twee mensen. In dit geval de dokter en de programmeur. We gaan ervan uit dat de dokter op één of andere manier haar kennis overbrengt naar de programmeur. De programmeur begint erna met het coderen van deze kennis. Hierbij wordt ervan uitgegaan dat de dokter haar kennis overdraagt door een formeel document zonder enige vaagheid of dubbelzinnigheid. De code die door de programmeur is geproduceerd is voor de dokter onbegrijpbaar, omdat ervan uitgegaan wordt dat de dokter niet kan programmeren. Hierdoor is het niet mogelijk voor haar om de correctheid van de implementatie te valideren door de code te lezen.

De correctheid van de implementatie kan beoordeeld worden door het resultaat van het systeem te testen. Maar Tom Routen beschrijft het probleem waar dan tegenaan gelopen wordt als de regels van schaken. Iemand kan de regels van schaken begrijpen zonder alle mogelijke zetten te kennen. Voor het voorbeeld met de programmeur en de dokter betekent dit dat alle richtlijnen, met veel logica en afhankelijk van verschillende variabelen, zoals symptomen en medische geschiedenis, moeilijk zijn om te valideren. Dit komt doordat het aantal mogelijke wegen om tot het resultaat te

komen veel te groot is. Hierdoor is het onhaalbaar om de correctheid aan de hand van testen te valideren. Validatie door middel van testen wordt nog moeilijker als er regelmatig veranderingen aan de richtlijnen worden aangebracht. Het is op die manier dus ook zeer moeilijk of onmogelijk om de validatiekloof tussen programmeur en dokter te verkleinen.

Bij een *no-code* aanpak kan in de plaats van een document waar de logica in beschreven wordt, een *no-code* omgeving gebruikt worden, waarin meteen de logica kan overgebracht worden. Hierdoor is er geen ruimte meer voor het verkeerd interpreteren van de logica. Doordat er geen code gebruikt wordt in *no-code* platformen is het voor de dokter veel gemakkelijker om het programma te begrijpen, wat ook betekent dat de logica transparant en begrijpbaar is. Dit zorgt ervoor dat de correctheid van de implementatie gevalideerd kan worden. Tom maakt duidelijk dat de taak van de dokter niet het schrijven van code is en dat dit ook totaal niet de bedoeling is.

Kort samengevat wil dit zeggen dat *no-code* niet het definitieve antwoord is om correctheid te garanderen, maar het zorgt er wel voor dat de logica transparanter is. Hierdoor wordt de validatiekloof geëlimineerd wat een grote stap in de juiste richting is.

In sommige gevallen is *no-code* niet gewoon efficiënter dan traditioneel programmeren. Het is in afdelingen zoals recht of wetgeving gewoon beter.

Conclusie

Er bestaan ontelbaar veel mogelijkheden voor het maken van een webapplicatie. Voor dit project wordt gebruikgemaakt van het *no-code* platform Neota Logic, maar was dit de beste optie voor het maken van de ‘flight helper’-applicatie? Op welke manier verschilt Neota Logic met een traditionele programmeertaal zoals Java? Was Neota het beste geoptimaliseerd voor het maken van de ‘flight helper’-applicatie? In welke mate is Neota beperkt? Welke voordelen kunnen bedrijven uit een *no-code* platform halen?

Uit onderzoek is gebleken dat Neota Logic de beste optie was voor de ‘flight helper’-applicatie. Dit komt doordat het project gefocust is op beslissingen. Een beslisboom staat centraal voor de beslissingen van de applicatie. Bovendien is het genereren van documenten makkelijk in Neota Logic.

Bedrijven kunnen voordelen halen uit een *no-code* platform. Een hogere productiviteit, snellere ontwikkelingstijd en het makkelijk aanpassen van de applicatie zijn hierbij de belangrijkste voordelen. Het is wel belangrijk om te weten dat hier ook enkele risico’s aan zijn verbonden. Er moet goed afgewogen worden wat het doel van de applicatie is. Elk *no-code* platform heeft zijn eigen functionaliteiten en tools. Afhankelijk moet het juiste platform gekozen worden.

Het is moeilijk om een *no-code* platform met een traditionele programmeertaal te vergelijken. Het is afhankelijk van project tot project om te zien of *no-code* of ontwikkelen met een programmeertaal zoals Java beter is. Neota Logic is beperkt ten opzichte van een taal zoals Java, maar de snellere ontwikkelingstijd en de handige en visuele tools zijn, afhankelijk van het doel, zeker het overwegen waard.

Mijn doel tijdens deze stage was om te ondervinden wat er allemaal mogelijk is met een *no-code* platform. In het begin dacht ik dat slechts kleine en simpele applicaties gebouwd konden worden met het gebruik van *no-code*. Maar door uitspraken zoals “The future of coding is no coding at all” wilde ik zelf ervaringen opdoen met *no-code* ontwikkelen. Door onder andere het onderzoek en het werken met Neota Logic ben ik positief verrast geworden door de verschillende mogelijkheden van *no-code*. Er is zeker meer mogelijk dan dat ik voorheen dacht en elk *no-code* platform heeft zo zijn eigen functionaliteiten en tools die zich specialiseren op bepaalde gebieden.

Bibliografie

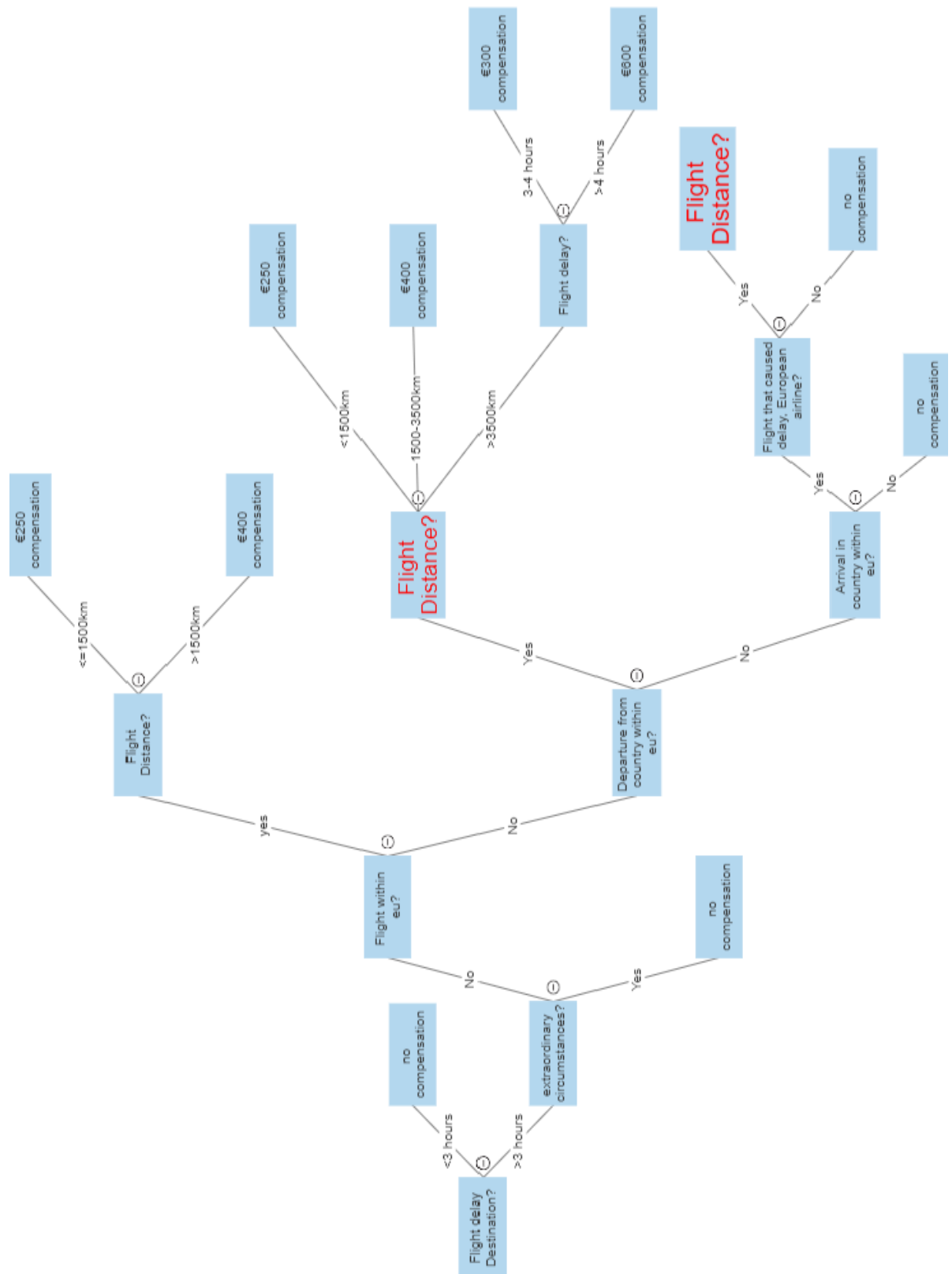
- [1] S. d. Keersmaecker, „Less than half of EU travellers are aware of EU Passenger Rights,” European commission, 13 januari 2020. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/ip_19_6814. [Geopend 3 februari 2020].
- [2] „Air passenger rights,” European Union, 05 maart 2020. [Online]. Available: https://europa.eu/youreurope/citizens/travel/passenger-rights/air/index_en.htm#delay. [Geopend 09 maart 2020].
- [3] „EU air passenger rights in case of denied boarding, a delayed flight or a cancelled flight,” 17 februari 2004. [Online]. Available: https://eur-lex.europa.eu/resource.html?uri=cellar:439cd3a7-fd3c-4da7-8bf4-b0f60600c1d6.0009.02/DOC_1&format=PDF. [Geopend 10 maart 2020].
- [4] „Neota Logic Canvas,” Neota Logic, 2020. [Online]. Available: <https://www.neotalogic.com/canvas/>. [Geopend 17 mei 2020].
- [5] Y. v. Roozendaal, „Wat is een webservice?,” Tektonikos, 18 oktober 2019. [Online]. Available: <https://www.tektonikos.eu/nl/artikelen/wat-is-een-webservice>. [Geopend 05 mei 2020].
- [6] B. Peterson, „The CEO of GitHub, which caters to coders, thinks automation will bring an end to traditional software programming,” Business Insider, 11 Oktober 2017. [Online]. Available: <https://www.businessinsider.com/github-ceo-wanstrath-says-automation-will-replace-software-coding-2017-10?r=US&IR=T>. [Geopend 13 april 2020].
- [7] B. Berkelaar, „Wat is no code en hoe werkt het,” esperantoxl, 13 augustus 2019. [Online]. Available: <https://esperantoxl.nl/blog/wat-is-no-code>. [Geopend 13 april 2020].
- [8] „The Neota platform,” Neota Logic, [Online]. Available: <https://www.neotalogic.com/platform/>. [Geopend 14 april 2020].
- [9] R. Novak, „What is the Difference Between No-Code and Low-Code Platforms?,” SegueTechnologies, 17 april 2019. [Online]. Available: <https://www.seguetech.com/difference-between-no-code-and-low-code-platforms/>. [Geopend 29 april 2020].
- [10] D. Pigott, „HOPL: An interactive Roster of Programming Languages,” WayBackMachine, 1995-2006. [Online]. Available: <https://web.archive.org/web/20110220044217/http://hopl.murdoch.edu.au/>. [Geopend 18 mei 2020].
- [11] P. v. Leemputten, „Scratch bij 20 populairste programmeertalen,” DataNews, 21 april 2020. [Online]. Available: https://datanews.knack.be/ict/nieuws/scratch-bij-20-populairste-programmeertalen/article-news-1590609.html?cookie_check=1590753040. [Geopend 14 mei 2020].

- [12] „History of Java programming Language,” Freejavaguide, 2013. [Online]. Available: <https://www.freejavaguide.com/history.html>. [Geopend 23 mei 2020].
- [13] D. Healay, „What is no code? The pros and cons of no code for software development.,” codebots, 23 maart 2020. [Online]. Available: <https://codebots.com/low-code/what-is-no-code-the-pros-and-cons-of-no-code-for-software-development>. [Geopend 24 april 2020].
- [14] „Java Tutorials,” Oracle, [Online]. Available: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html>. [Geopend 23 mei 2020].
- [15] „java database connectivity with 5 steps,” JavaTPoint, [Online]. Available: <https://www.javatpoint.com/steps-to-connect-to-the-database-in-java>. [Geopend 29 mei 2020].
- [16] T. Routen, „Key advantages of No Code for complex decision support,” Medium, 29 februari 2020. [Online]. Available: <https://medium.com/@routen/key-advantages-of-no-code-for-complex-decision-support-fc1145d8a87b>. [Geopend 13 maart 2020].

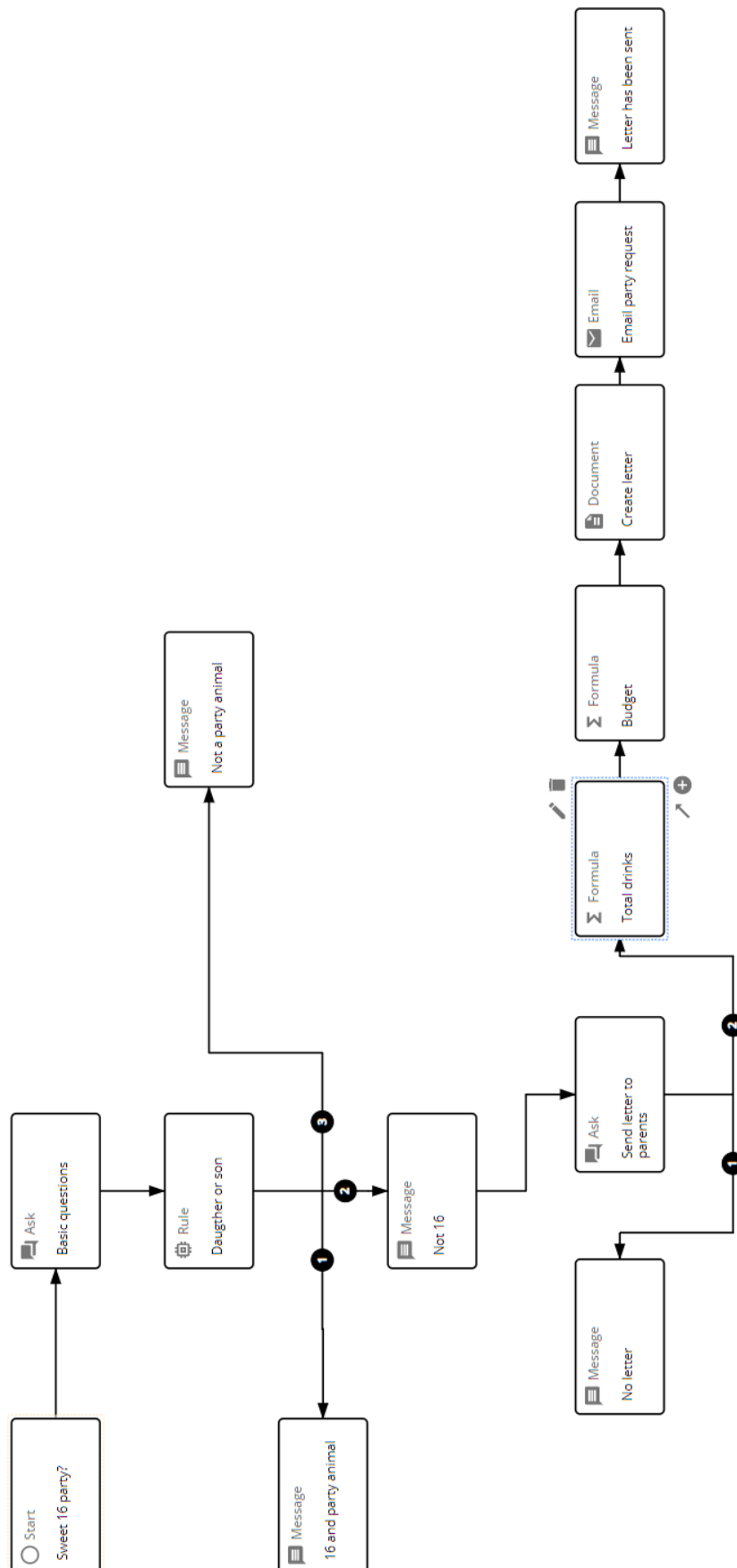
Bijlagen

- A. Beslisboom van de 'flight helper'-applicatie**
- B. 'Sweet sixteen'-applicatie in Canvas**
- C. Beslisboom in Neota Logic**

A. Beslisboom van de 'flight helper'-applicatie



B. Sweet sixteen applicatie in Canvas



C. Beslisboom in Neota Logic

