“Ontwikkelen van een webapplicatie (dashboard en API), die de visuele interactie tussen de chatbot en mantelzorger verzorgt.”

Afstudeerverslag
HBO ICT - Software Engineering
De Haagse Hogeschool

Student: Jeffrey Dufour (15036278)
Bedrijf: TrueLime
Bedrijfsmentor: Wesley van Strien
Examinatoren: Rianne Bechet – Tjoonk & Gustaaf Vöcking

Datum: 27 mei 2019
Plaats: Breda

Ontwikkelen van een webapplicatie (dashboard en API), die de visuele interactie tussen de chatbot en mantelzorger verzorgt.
Breda, 2019

Afstudeerverslag van Jeffrey Dufour, geschreven in het kader van het afstuderen bij de opleiding Software Engineering aan de Haagse Hogeschool.

In dit verslag wordt het ontwikkelproces van een dashboard (prototype) beschreven dat is uitgevoerd in opdracht van TrueLime, een ICT-bedrijf in Breda. Het dashboard is een Single-Page Application (SPA) dat de huidige functionaliteiten van de chatbot visualiseert. Mantelzorgers en hun cliënten kunnen gebruik maken van het dashboard met als doel om de zorgverlening te verbeteren. Voor het realiseren van een SPA is er, op verzoek van TrueLime, een onderzoek uitgevoerd naar populaire frameworks.

Descriptoren:

- Afstudeeropdracht
- Chatbot
- Microsoft Bot Framework
- .NET Core
- Azure
- JavaScript
- Single-Page Application
- Vue

Voorwoord

Voor u ligt de scriptie ‘Ontwikkelen van een webapplicatie (dashboard en API), die de visuele interactie tussen de chatbot en mantelzorger verzorgt’. Tijdens het project heb ik onderzoek gedaan naar diverse Single-Page Application frameworks om vervolgens een prototype te ontwikkelen voor de mantelzorgers en hun cliënten. Het prototype heeft als doel om de zorgverlening van de mantelzorgers te verbeteren. Het project is uitgevoerd in opdracht van TrueLime, een ICT-bedrijf in Breda. Deze scriptie is geschreven in het kader van mijn afstuderen aan de opleiding Software Engineering aan de Haagse Hogeschool. Van 4 februari tot en met 31 mei ben ik bezig geweest met het onderzoeken en ontwikkelen van een dashboard voor mantelzorgers en hun cliënten.

Samen met Wesley van Strien, de bedrijfsmentor, heb ik binnen 17 weken een volwaardig prototype mogen ontwikkelen. Hierbij wil ik hem ook extra bedanken voor de begeleiding en tijd die hij in mij en de scriptie heeft geïnvesteerd. Ik heb een hoop kennis opgedaan in het beheren van een project. Verder wil ik de product owner, Hans van der Linden, bedanken voor de prettige samenwerking en goede feedback die tijdens de Sprint Planning en Demo zijn gegeven. Verder wil ik Team Raptors bedanken voor de betrokkenheid in het project en voor het open staan voor technische en inhoudelijke vragen.

Tevens wil ik de examinatoren Rianne Bechet-Tjoonk en Gustaaf Vöcking bedanken voor de begeleiding bij het schrijven van dit verslag. De feedback heeft mij enorm geholpen om de scriptie zo volledig mogelijk op te stellen.

Ik wens u veel leesplezier toe.

Jeffrey Dufour
Breda, 24 mei 2019

Samenvatting

TrueLime heeft een chatbot ontwikkeld voor mantelzorgers en hun cliënten. De spraakgestuurde applicatie helpt mantelzorgers bij het verzorgen van hun naasten. Een chatbot bevat verschillende vaardigheden, zoals het beheren van afspraken, taken en het bijhouden van een logboek.

Probleemstelling
In de praktijk is het probleem ontstaan dat de spraakgestuurde applicatie niet toegankelijk is voor de doelgroep. Mantelzorgers en de cliënten hebben 'visuele' begeleiding nodig voor de interactie met de chatbot. Hiermee moet het mogelijk zijn om het dashboard ook zonder spraak te kunnen bedienen.

Doelstelling en onderzoeksvraag
Het doel is om een dashboard (webapplicatie) te ontwikkelen voor het visualiseren van de interactie met de chatbot om zo de toegankelijkheid voor de mantelzorgers en hun cliënten te verbeteren. Om het dashboard zo gebruiksvriendelijk mogelijk te maken is er bij TrueLime een wens ontstaan om een Single-Page Application (SPA) te ontwikkelen. Een SPA zorgt voor een betere gebruikerservaring, omdat de volledige pagina van een webapplicatie niet bij elke interactie van de gebruiker (opnieuw) ingeladen hoeft te worden. Hiervoor is de volgende hoofdvraag ontstaan, namelijk: *Hoe kan een dashboard voor TrueLime ontwikkeld worden, met het gebruik van een Single-Page application framework, om de interactie met de chatbot te verbeteren?*

Aanpak
Voor het realiseren van het dashboard is Scrum als ontwikkelmethodiek ingezet. Er zijn in totaal zeven sprints van twee weken ingepland. Samen met de productowner is er besproken welke functionaliteiten er in het dashboard moet komen. De requirements zijn geprioriteerd volgens de MoSCoW-methodiek, omdat het helpt met het bepalen van wat er in een product moet komen om het project te laten slagen. Met de MoSCow-methode krijgen de opgestelde requirements een label (must have, should have, could have en won't have) en zijn hierdoor makkelijker te prioriteren (ToolsHero, 2018).

Uitvoering
Het dashboard moet communiceren met de chatbot, hiervoor is de huidige situatie, domein en technische omgeving onderzocht. Uit het onderzoek is een architectuur bepaald en is per sprint verder aan het prototype ontwikkeld. Het dashboard en chatbot zijn twee losse applicaties en delen een codebase met elkaar.

Uit het onderzoek van diverse SPA frameworks is Vue.js als beste uitgekomen voor het project. Het framework scoort goed op performance en is compact. Verder heeft Vue een simpele integratie, en wordt mede hierdoor als minder complex ervaren. In combinatie met goede documentatie en een laag instapniveau heeft ervoor gezorgd dat er vlot aan het dashboard ontwikkeld kon worden.

Resultaat
Na 17 sprints is er een dashboard ontwikkeld waar alle vaardigheden van de chatbot zijn gevisualiseerd. Hiermee is het mogelijk om taken, afspraken en logberichten te beheren in het dashboard. Verder worden data realtime uitgewisseld met elkaar en kan er rechtstreeks met de chatbot gecommuniceerd worden via een chat interface. TrueLime is tevreden met het prototype. Het dashboard wordt ingezet voor sales pitches. Ze hopen om een zorginstantie te kunnen helpen door het inzetten van chatbots.

Inhoud

1. Inleiding

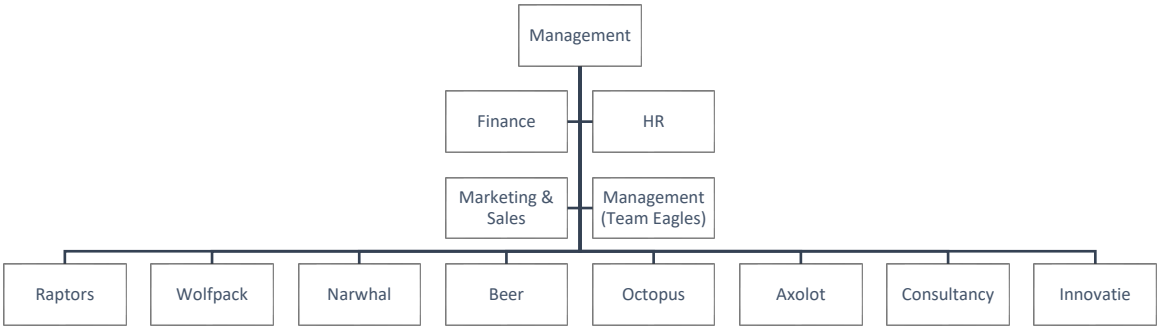
TrueLime, een ICT-bedrijf gevestigd in Breda, heeft een chatbot ontwikkeld voor mantelzorgers en hun cliënten. De virtuele assistent moet de samenwerking tussen mantelzorgers en hun cliënt verbeteren. De chatbot is een applicatie die wordt aangestuurd middels spraak. De mantelzorgers en de cliënt kunnen opdrachten inspreken en de chatbot verwerkt de gegevens. Een mantelzorger kan taken toevoegen, afspraken inplannen en een logboek beheren. TrueLime is ervan bewust dat een spraakgestuurde applicatie niet toegankelijk is voor de doelgroep. Het bedrijf is van mening dat mantelzorgers en hun cliënten 'visuele' begeleiding nodig hebben voor de interactie met de chatbot. TrueLime wil dat de chatbot zowel een visueel als spraakinterface gaat krijgen om zo beide doelgroepen te kunnen ondersteunen. Het resultaat gaat een dashboard worden dat in het huis van de cliënt geplaatst kan worden. Daarnaast moet het mogelijk zijn dat de mantelzorger het dashboard kan gebruiken op zijn of haar telefoon, tablet of desktop.

In dit document wordt het proces van het afstuderen beschreven. Allereerst vertel ik over de organisatie, opdracht en de aanpak van het project. Vervolgens ga ik de huidige situatie en technieken onderzoeken om zo een aansluitende architectuur op te zetten. TrueLime wil graag een applicatie die een spraakinterface en een visuele interface heeft. Beide interfaces ga ik onderzoeken om zo tot een besluit te komen voor het ontwerp van het dashboard. Vervolgens start ik mijn onderzoek naar Single-Page Applications frameworks. Met het resultaat van dit onderzoek kan de ontwikkeling van het dashboard van start gaan. Het ontwikkelproces wordt iteratief uitgevoerd. Er zijn in totaal zeven sprints ingepland en beschreven. Tot slot eindig ik het document met een evaluatie van het proces en product.

2. Organisatiebeschrijving

TrueLime is een ICT-bedrijf dat in 1997 is opgericht. Het bedrijf is gevestigd in Breda en heeft 63 medewerkers in dienst. TrueLime creëert diverse producten, zoals: intranetten, websites, portalen en applicaties. Ze zijn gespecialiseerd in softwareoplossingen voor werkprocessen van juridische aard. TrueLime staat bekend om het zaaksysteem Octopus, dat administratieve en juridische werkprocessen automatiseert, beheert en optimaliseert. Daarnaast adviseren en begeleiden de consultants van TrueLime organisaties bij het ontwikkelen van een strategie naar succesvolle online toepassingen. De klanten van TrueLime opereren in de zorg, in het onderwijs, of zijn gemeentes en overheidsinstanties.

TrueLime is een Gold partner van Microsoft, dit komt mede door dat alle softwareoplossingen ontwikkeld wordt met hun ecosysteem. Het bedrijf heeft zeven Agile teams, die allemaal zelfstandig aan hun projecten werken. In Figuur 1 is er een organogram van de organisatie te zien.



Figuur 1: Organogram

Tijdens het afstudeertraject maak ik onderdeel uit van Team Innovatie en mijn werkplek is bij Team Raptors. Het team focust zich vooral op het ontwikkelen van software voor onderwijs en zorginstellingen. Het team bevat zes leden, namelijk: Wesley van Strien (bedrijfsmentor, scrum master), Sietse Trommelen (architect), Leroy Ketelaars (ontwikkelaar), Roel de Bruijn (ontwikkelaar) en Nick van der Raaf (ontwikkelaar).

3. Opdrachtomschrijving

TrueLime heeft een chatbot ontwikkeld voor mantelzorgers en hun cliënten. De applicatie helpt mantelzorgers bij het verzorgen van hun naasten. Een chatbot bevat verschillende vaardigheden, zoals het beheren van afspraken, taken en het bijhaouden van een logboek. De cliënt kan zelf ook gebruik maken van de chatbot om bijvoorbeeld klusjes toe te voegen voor de mantelzorger.
De chatbot is een spraakgestuurde applicatie waarmee de mantelzorger en hun cliënten kunnen communiceren.

In de praktijk is het probleem ontstaan dat de spraakgestuurde applicatief niet toegankelijk is voor de doelgroep. TrueLime is van mening dat mantelzorgers en de cliënten 'visuele' begeleiding nodig hebben voor de interactie met de chatbot. TrueLime wil dat de chatbot zowel een visueel als spraakinterface gaat krijgen om zo beide doelgroepen te kunnen ondersteunen.

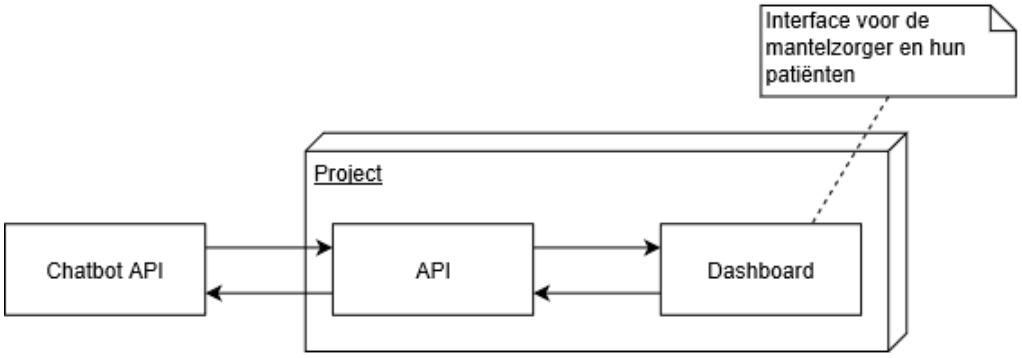
Het doel is om een dashboard (webapplicatie) te ontwikkelen voor het visualiseren van de interactie met de chatbot om zo de toegankelijkheid voor de mantelzorgers en hun cliënten te verbeteren. Hiermee moet het mogelijk zijn om het dashboard ook zonder spraak te kunnen bedienen.

Onderzoek
Om het dashboard zo gebruiksvriendelijk mogelijk te maken is er bij TrueLime een wens ontstaan om een Single-Page Application (SPA) te ontwikkelen. Een SPA zorgt voor een betere gebruikerservaring, omdat de volledige pagina van een webapplicatie niet bij elke interactie van de gebruiker (opnieuw) ingeladen hoeft te worden. De applicatie reageert hierdoor sneller op de input van een gebruiker (Ismail, 2018). Er zijn diverse (populaire) SPA frameworks in opkomst en TrueLime wil graag meer weten over de voor- en nadelen van elk framework voordat de ontwikkeling van het dashboard van start gaat. Hiervoor is de volgende hoofdvraag opgesteld, namelijk: *Hoe kan een dashboard voor TrueLime ontwikkeld worden, met het gebruik van een Single-Page Application framework, om de interactie met de chatbot te verbeteren?*

Om antwoord te krijgen op de hoofdvraag heb ik de volgende deelvragen opgesteld:

1. Welke functionaliteiten moet het dashboard bevatten om de interactie met de chatbot te verbeteren?
2. Wat is de huidige situatie bij TrueLime?
    - Wat is een chatbot?
    - Welke technieken worden er gebruikt om de chatbot te realiseren?
    - Wat is een virtuele assistent?
3. Hoe kan de communicatie tussen het dashboard en chatbot gerealiseerd worden?
4. Hoe kan een visuele- en spraakinterface gerealiseerd worden in het dashboard?
5. Welke Single-Page Application framework is geschikt voor het ontwikkelen van het dashboard?

Context
Na het uitvoeren van de opdracht verwachten we een dashboard gebouwd te hebben dat gebruiksvriendelijker zal zijn voor de mantelzorgers en hun cliënten. Het dashboard gaat ervoor zorgen dat de doelgroep zowel via de visuele interface en via spraak kan interacteren met de chatbot. Verder wordt het dashboard een webapplicatie die toegankelijk is op een telefoon, tablet en desktop. Hiermee is het product op meerdere kanalen beschikbaar. Doordat het dashboard geen onderdeel uitmaakt van de chatbot, zal er een API ontwikkeld worden voor de communicatie tussen de chatbot en het dashboard. In Figuur 2 is een globale weergave van de applicatie te zien.



Figuur 2: Globale weergave van applicatie

4. Aanpak

Het ontwikkelen van het dashboard gebeurt via de Scrum-methodiek. TrueLime wilt graag dat ik hun manier van werken leer en dat toepas in het project. Door het project incrementeel op te zetten, zal per iteratie de requirements (User Stories) opgesteld worden. Met als doel om na 17 weken tot een volwaardig prototype te komen, dat voldoet aan de acceptatiecriteria van de product owner.

Er zullen sprints van twee weken plaatsvinden, waarin we twee keer per week een stand-up houden om de voortgang te bespreken. Aan het eind van een Sprint zal er een review moment plaatsvinden met de product owner. Tijdens de Sprint Review is er ruimte om de volgende User Stories in te richten voor de volgende iteratie (Sprint Planning). Het project zal niet volledig volgens de standaarden van Scrum gevolgd worden, en wijkt af op het volgende:

1. Daily stand-up meetings worden om de twee dagen met de bedrijfsmentor ingepland;
2. Scrum team bestaat uit een product owner, ontwikkelaar en een bedrijfsmentor;
3. Product owner werk niet fulltime op het project, hierdoor is afgesproken dat de bedrijfsmentor, indien nodig, de rol van hem overneemt.

Samen met de product owner en bedrijfsmentor zijn de Definition of Ready en Definition of Done besproken en vastgelegd. In Tabel 1 zijn de eisen vastgelegd waar een Product Backlog Item (PBI)[1] aan moet voldoen. Dit zijn de eisen die TrueLime ook in hun eigen Scrum teams stellen.

Definition of Ready	**Definition of Done**
Een PBI is "ready" en kan opgepakt worden in een sprint wanneer: 1. Een PBI is geschreven in User Story format "als een - wil ik - zodat"; 2. Acceptatiecriteria zijn toegevoegd (indien nodig); 3. User Story is kort genoeg om in een sprint te passen; 4. De beschrijving is duidelijk voor eenieder die het werk moet oppakken; 5. Het backlog item is voorzien van een (relatieve) inschatting; 6. Er zijn taken aangemaakt voor alle benodigde werkzaamheden voor dit item;	Een PBI is "done" en kan potentieel naar productie gereleaset worden wanneer: 1. De functionaliteit voldoet aan de acceptatiecriteria; 2. Alle taken van het betreffende Backlog Item zijn uitgevoerd; 3. De product owner gaat akkoord met de functionaliteit. 4. De product owner heeft een demo gehad van de ontwikkelde functionaliteit. 5. De documentatie is geüpdatet (indien nodig);

[1] De Product Backlog in Scrum bestaat uit een lijst met items (User Stories) die uitgevoerd moeten worden tijdens de ontwikkeling van het product (Scrumguide, sd).

7. Het item heeft geen afhankelijkheden, of deze afhankelijkheden zijn geïdentificeerd.	

Tabel 1: Eisen van een PBI

Prioriteren van requirements

Voor het beantwoorden van mijn eerste deelvraag 'Welke functionaliteiten moet het dashboard bevatten om de interactie met de chatbot te verbeteren?' heb ik aan de product owner gevraagd welke functionaliteiten er in de chatbot zitten en wat de wensen zijn voor het dashboard. Uit het gesprek zijn de requirements naar voren gekomen, besproken en vastgelegd. Voor het prioriteren van de requirements heb ik gebruik gemaakt van de MoSCoW-methode, omdat het helpt met het bepalen van wat er in een product moet komen om het project te laten slagen. Met de MoSCow-methode krijgen de opgestelde requirements een label en zijn hierdoor makkelijker te prioriteren (ToolsHero, 2018).

De requirements zijn ingedeeld in de volgende categorieën:

- **Must have**: Deze eisen moeten in het eindresultaat komen anders is het product niet bruikbaar.
- **Should have:** Deze eisen zijn gewenst, maar het product is alsnog bruikbaar als ze niet worden gerealiseerd.
- **Could have**: Deze eisen worden gerealiseerd als er nog tijd over is.
- **Won't have:** Deze eisen zullen waarschijnlijk niet gerealiseerd worden.

In Tabel 2 staan de geprioriteerde functionaliteiten. Met de 'must have' en 'should have' requirements als doel voor ogen is er een planning opgesteld. In Figuur 3 is de planning te zien zoals het is opgenomen in het plan van aanpak.

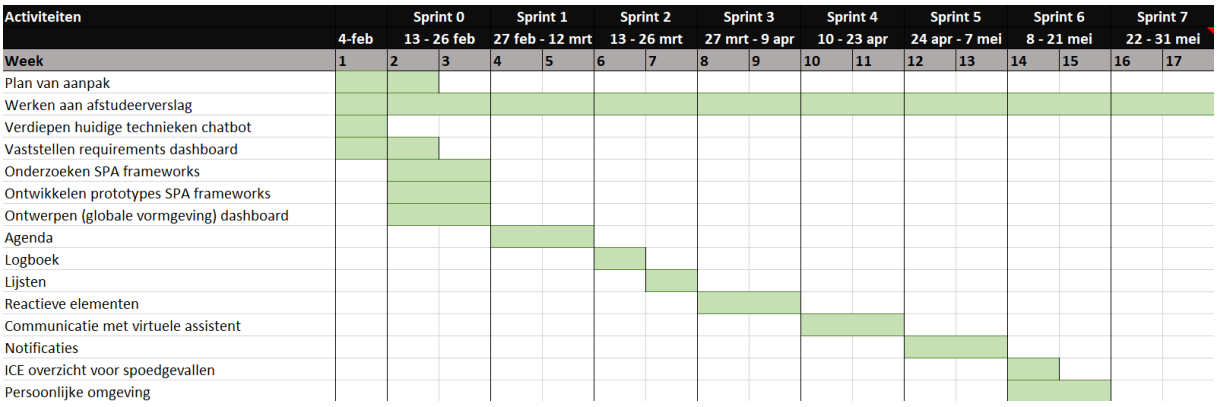
Must have	1. Beheren van logboek 2. Agenda (afspraken) 3. Lijsten (taken) 4. Reactieve elementen (realtime dataverwerking) 5. Communicatie met virtuele assistent (vraag en antwoord)
Should have	6. Persoonlijke omgeving 7. Notificaties 8. ICE-overzicht voor spoedgevallen
Could have	9. Afgeschermde omgeving 10. Stemherkenning 11. Vervoer regelen voor patiënt
Won't have	12. Chatten met mantelzorgers *Chatten binnen het dashboard is tegen de visie van het product. Het aanbieden van een chat mogelijkheid geeft de gebruikers de mogelijkheid om de geïntegreerde functionaliteiten, zoals taken en afspraken in de chatbot te ontwijken. De reden waarom het*

	dashboard wordt ontwikkeld is om structuur te krijgen in de communicatie.

Tabel 2: Prioriteren requirements

De requirements zijn inmiddels bekend, geprioriteerd en ingepland voor het dashboard en hierbij is de deelvraag 'Welke functionaliteiten moet het dashboard bevatten om de interactie met de chatbot te verbeteren?' beantwoord. Tijdens de ontwikkelfase van het dashboard worden de requirements geïmplementeerd, waarbij de focus steeds ligt op het verbeteren van de interactie met de chatbot.

In de voorbereidingsfase van het project (Sprint 0) start mijn onderzoek naar Single-Page Application (SPA) frameworks. Daarnaast ga ik me verdiepen in de huidige situatie en technieken van de chatbot en wordt er een globaal ontwerp gemaakt van het dasboard. De voorbereidingen zorgen ervoor dat ik in Sprint 1 direct aan de slag kan gaan met het ontwikkelen van het dashboard.



Figuur 3: Globale planning zoals opgenomen in het plan van aanpak

In het plan van aanpak (Bijlage 1) is er meer informatie te vinden over de planningsfase. Elke Sprint heeft zijn eigen hoofdstuk, waarin de uitvoering nader wordt beschreven.

5. Dashboard - Architectuur

In de planningsfase heb ik de functionaliteiten van de chatbot onderzocht om meer inzichten te krijgen over de scope van het project. Aan het begin van mijn onderzoek heb ik samen met de ontwikkelaar van de chatbot gezeten. Tijdens die sessie ben ik meer te weten gekomen over de huidige functionaliteiten en de technieken van de applicatie. Na de sessie ben ik de technieken van de chatbot verder gaan bestuderen. Hiermee wilde ik op een globaal niveau begrijpen hoe de applicatie is geconfigureerd en wat voor invloeden dat zou kunnen hebben op het dashboard. In dit hoofdstuk beschrijf ik de theorie en uitvoering van de architectuur.

5.1 Theorie

Voor het uitdenken van de architectuur heb ik de deelvraag 'Wat is de huidige situatie bij TrueLime?' opgesteld (zie hoofdstuk 3 voor alle deelvragen). Om antwoord te krijgen op deze vraag heb ik het volgende onderzocht:

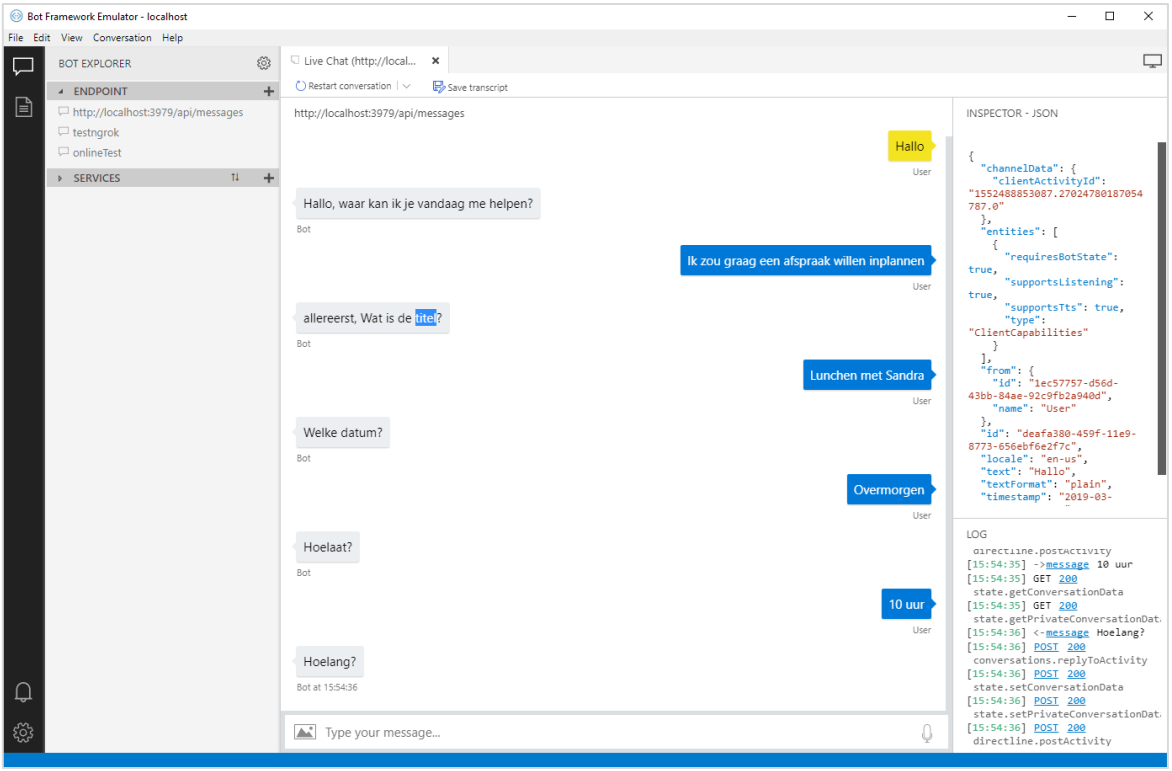
1. Wat is een chatbot?
2. Welke technieken worden er gebruikt om de chatbot te realiseren?
3. Wat is een virtuele assistent?

Wat is een chatbot?

Een chatbot is een geautomatiseerd softwareprogramma. Bots worden vaak ingezet bij een helpdesk-omgeving om op een snel en effectieve wijze een klant te helpen. Het juist inzetten van de chatbot geeft de klant de ervaring dat hij of zij met een persoon, in dit geval een medewerker aan het praten is (Microsoft, 2019). Het toepassen van een chatbot zorgt voor het verbeteren van de gebruikservaring, omdat ze altijd beschikbaar zijn voor het beantwoorden van vragen. De chatbots begrijpen de context uit een vraag en kunnen hierdoor ook sneller antwoord geven. Door de chatbot te implementeren, worden handmatige handelingen geautomatiseerd. Hiermee zal er kostenreductie plaatsvinden (Den Arend, 2018).

Welke technieken worden er gebruikt om de chatbot te realiseren?

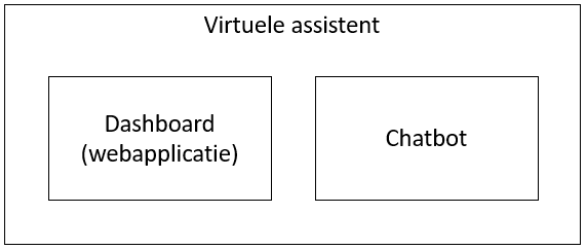
TrueLime heeft met Azure Bot Services van Microsoft een chatbot kunnen ontwikkelen voor mantelzorgers en hun cliënten. De chatbot kan je inzetten op verschillende kanalen, zoals: Skype, Slack, Facebook en Cortana. Het is op dit moment mogelijk om via een chat interface (chat emulator) en met Google Assistant (spraak) te communiceren met de bot. In Figuur 4 wordt de interactie met de chatbot weergegeven via een emulator.



Figuur 4: Gesprek met een chatbot via een emulator

Wat is een virtuele assistent?

De termen chatbot, dashboard en virtuele assistent kunnen lastig van elkaar te onderscheiden zijn, omdat ze veel raakvlakken met elkaar hebben. Het is belangrijk om vast te leggen wat ze betekenen voor dit project. Samen met de product owner is afgesproken dat de chatbot wordt gezien als de achterliggende techniek van de virtuele assistent, terwijl het dashboard de visualisatie en interactie verzorgd. Eigenlijk worden beide projecten, chatbot en dashboard, samen als de 'virtuele assistent' gezien, zoals in Figuur 5 is weergegeven.



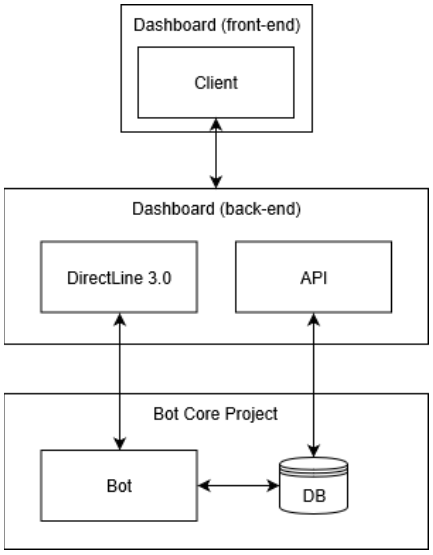
Figuur 5: Virtuele assistent

In de simpelste vorm stuurt de chatbot alleen tekst naar de aanvrager, de aangesloten kanalen bepalen wat ermee gebeurt. Het dashboard is een webapplicatie en wordt in dit geval een nieuw kanaal voor de chatbot.

5.2 Uitvoering

Voor de deelvraag 'Hoe kan de communicatie tussen het dashboard en chatbot gerealiseerd worden?' heb ik onderzoek gedaan naar de technieken waar de chatbot mee is ontwikkeld, namelijk Azure Bot Services van Microsoft. Met de zogeheten DirectLine API is het mogelijk om een communicatie laag te creëren naar de chatbot. De API zorgt ervoor dat het dashboard een nieuw kanaal voor de chatbot

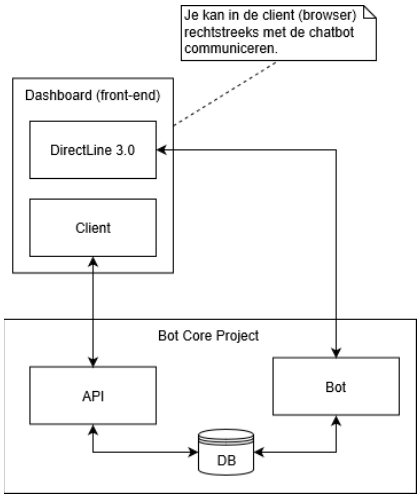
wordt. Er zijn twee opties om de Direct Line API te integreren in een project, namelijk rechtstreeks in de client (browser) of in de server. Ik heb drie architecturen bedacht over hoe de communicatie tussen het dashboard en chatbot geregeld kan worden. Hieronder zijn de drie voorstellen met hun betreffende omschrijving opgenomen:



Figuur 6: Koppeling dashboard back-end

Architectuur #1

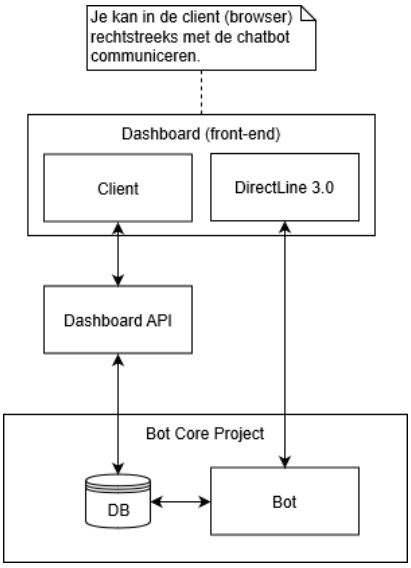
Het dashboard is op te splitsen in twee omgevingen, namelijk het front-end (client) en back-end (server). In Figuur 6 bevindt de DirectLine API zich in de back-end van het dashboard. Hiermee is het dashboard vanaf de server in verbinding met de chatbot. Voor overige acties, zoals het ophalen van data uit de chatbot, wordt er een Dashboard API ontwikkeld dat in verbinding staat met de database. Het voordeel is dat de chatbot geen kennis heeft van het dashboard. Dit zorgt voor een lage koppeling, wat uiteindelijk de aanpasbaarheid van de software bevorderd. Een nadeel is dat er een extra communicatie laag tussenkomt. Dit kan nadelig zijn voor de snelheid van de applicatie.



Figuur 7: Koppeling chatbot-omgeving

Architectuur #2

In Figuur 7 bevindt de DirectLine API zich aan het front-end van het dashboard. De client staat direct verbonden met de chatbot. Dit is ideaal voor het snel afhandelen van conversaties. Er is namelijk geen extra communicatie laag tussen de client en chatbot. Verder wordt er in de chatbot-omgeving een API ontwikkeld om zo data op te halen vanuit de client. Met deze aanpak is er geen back-end voor het dashboard nodig. Een nadeel is dat de API ontwikkeld wordt voor het dashboard. Hierdoor worden er functionaliteiten toegevoegd aan de chatbot wat niet voor de chatbot van toepassing is.



Figuur 8: DirectLine in client

Architectuur #3

In Figuur 8 bevindt de API zich in het dashboard-omgeving. Met dit voorstel blijft het voordeel bestaan dat de communicatie tussen het dashboard en chatbot-omgeving geen extra communicatie laag bevat. Verder is er een back-end voor het dashboard aanwezig wat uiteindelijk de data uit de chatbot verwerkt. Een nadeel is dat het Dashboard API en de back-end omgeving van de chatbot overeenkomsten hebben, zoals database modellen. Dit leeft dan in twee omgevingen en verreist twee handelingen mocht er iets gewijzigd worden. Wel blijft het voordeel bestaan dat de chatbot niks van het dashboard weet.

Conclusie

In Tabel 3 heb ik alle voor- en nadelen van de architectuur ontwerpen onder elkaar gezet. De architecturen heb ik gedeeld met de product owner, bedrijfsmentor en architect binnen TrueLime en alle drie gingen akkoord met het derde voorstel. Het dashboard is gescheiden van de chatbot-omgeving en heeft alsnog het voordeel dat het vanuit de client direct verbonden staat met de chatbot. In Sprint 1, voor de ontwikkeling van de back-end, moet er een oplossing komen voor de aanpasbaarheid van de (redundante) code dat in beide omgevingen leeft.

Voorstel	Voordeel	Nadeel
#1	• De chatbot-omgeving is niet op de hoogte van het dashboard.	• Extra communicatie laag tussen chatbot en dashboard.
#2	• Directe verbinding (via de client) met de chatbot.	• Dashboard specifieke implementaties kunnen in de back-end van de chatbot beschikbaar zijn. Er is een hoge koppeling tussen beide omgevingen.
#3	• Directe verbinding (via de client) met de chatbot. • Dashboard heeft een eigen back-end waar dashboard-gerelateerde data verwerkt kan worden. • Chatbot weet niks van het dashboard-omgeving.	• Mogelijkheid tot redundante code dat zowel in het dashboard als chatbot kan ontstaan.

Tabel 3: Voor- en nadelen voorstellen architecturen

6. Dashboard - Ontwerp

De chatbot is een applicatie waar een gebruiker een gesprek mee kan voeren. Op dit moment, zoals eerder beschreven in hoofdstuk 5.1, is het alleen mogelijk om de chatbot via spraak (of via een chat emulator) te benaderen. Voor het dashboard heeft de product owner aangegeven een combinatie te willen zien tussen een zogeheten conversational interface en een traditionele userinterface.

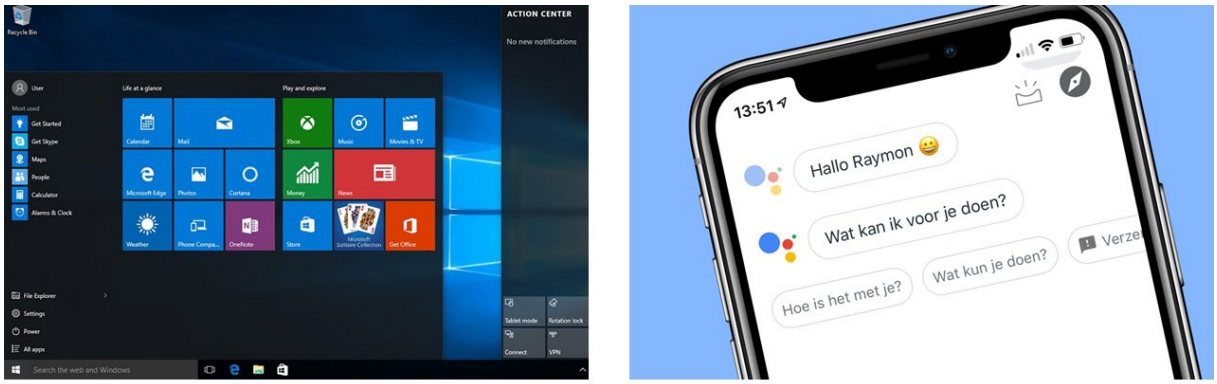
6.1 Theorie

Om de deelvraag 'Hoe kan een visuele- en spraakinterface gerealiseerd worden in het dashboard?' te beantwoorden heb ik onderzocht wat een conversational en traditionele interface is en wat voor invloed dit heeft op het uiterlijk van het dashboard.

Conversational interface

In 1981 heeft het bedrijf Xerox PARC de eerste commerciële computer met een grafische interface uitgebracht. In plaats van de toen gebruikte command-line-interface was het nu mogelijk om de computer te besturen door op visuele elementen te klikken (Štolfa, 2016).

Er is sinds 2016 een andere trend aan het groeien, namelijk de conversational interface. Hiermee is het mogelijk om via spraak de applicatie te besturen. Ontwerpers krijgen de mogelijkheid om de wat complexere interface te versimpelen door elementen te verbergen, die pas worden getoond door middel van een specifieke spraakopdracht. De gebruiker hoeft niet meer een grafische omgeving te leren, omdat zij al gewend is om in haar eigen taal te kunnen communiceren. Dit is ook direct de reden waarom conversational interfaces zo populair aan het worden zijn (Brownlee, 2016). Implementaties van zo'n interface zijn bijvoorbeeld virtuele assistenten en chatbots (zie Figuur 9).



Figuur 9: Verschil tussen een grafische en conversational interface (One More Thing, 2018)

Virtuele assistenten verbeteren de gebruikerservaring op een telefoon en ook in huis. Zo is het mogelijk om bijvoorbeeld met Google Home, een smart speaker, alle lichten in het huis te besturen met één spraakopdracht. Bij een chatbot is het mogelijk dat elke bericht dat de gebruiker stuurt de potentie heeft om een miniapplicatie te worden. De applicatie zou bijvoorbeeld een kaart kunnen tonen met de dichtstbijzijnde restaurants (Štolfa, 2016).

Hybride interface

TrueLime speelt in op de huidige trends. Het dashboard wordt de communicatielijn tussen de gebruikers (mantelzorgers en hun cliënten) en de chatbot. Het is mogelijk om het dashboard te beheren via spraak, maar ook de traditionele userinterface is aanwezig. Het is weliswaar de bedoeling om een soort hybride interface te ontwerpen. De applicatie moet reactief zijn, wat betekent dat het

inspeelt op veranderingen. Als een cliënt bijvoorbeeld een nieuwe klus heeft toegevoegd, dan moet de mantelzorger dat direct zien verschijnen op het dashboard.

De reden waarom de product owner een hybride interface wilt is eenvoudig. Niet alle patiënten zijn in staat om via spraak het dashboard te beheren. Het zou kunnen dat een patiënt slechthorend is en hierdoor niet via spraak kan communiceren met de chatbot. Wel kan de patiënt gebruik maken van de grafische interface van het dashboard.

6.2 Uitvoering

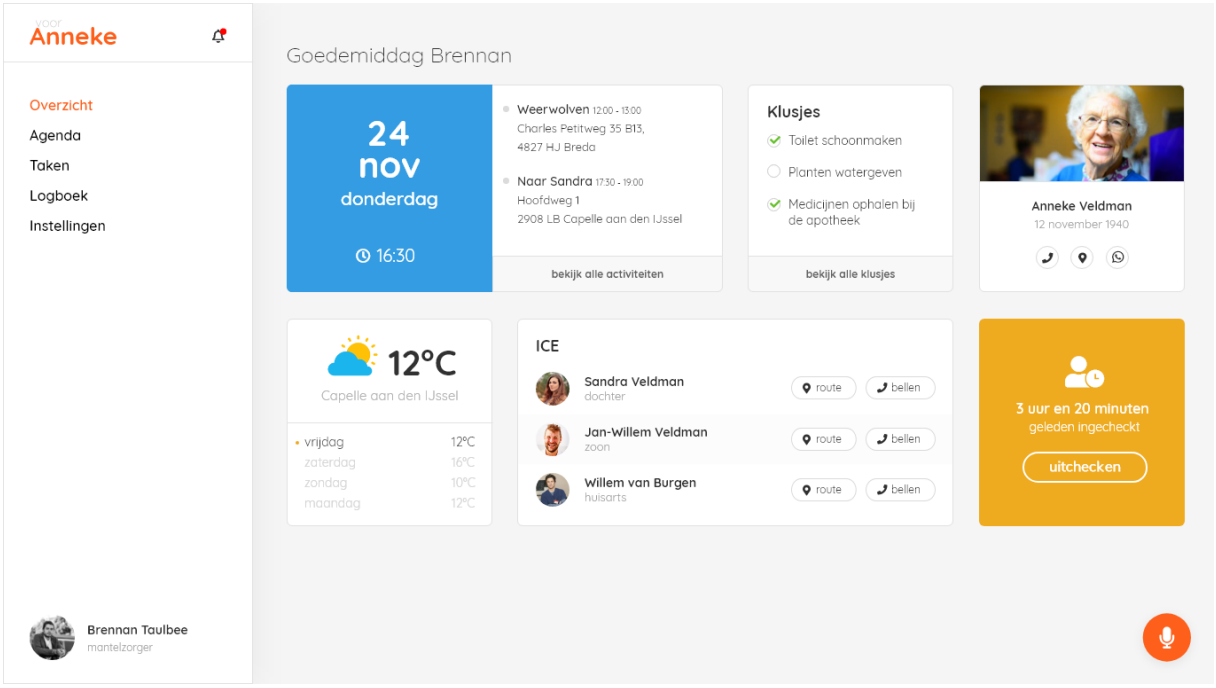
Het dashboard wordt de visuele begeleiding voor de mantelzorgen en hun naasten. Er waren geen ontwerpen van het dashboard beschikbaar. Hierdoor had ik twee opties voor de userinterface (UI) van de applicatie, namelijk:

1. Gebruik maken van een framework, zoals Bootstrap of Material Design: Hiermee is de interface voor een groot deel bepaald;
2. Maatwerk: Vormgeving en de bijbehorende sjablonen zelf bedenken, ontwerpen en uitwerken.

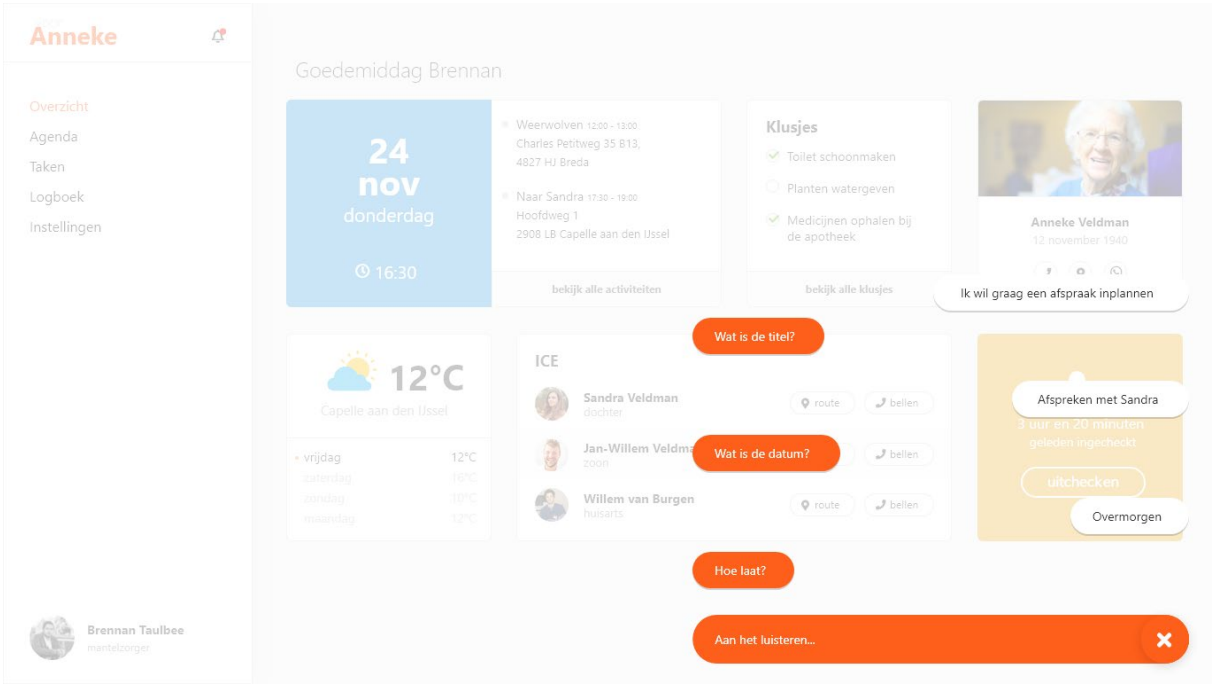
Een UI framework kan de ontwikkelaar helpen om snel een UI te realiseren, mits je niet afwijkt van de gekozen structuur en sjablonen (Andrew, 2011). Aangezien het dashboard een wat complexere interface krijgt is het verstandig om een eigen ontwerp te bepalen en uit te werken. De product owner heeft hier goedkeuring voor gegeven. Het realiseren van een eigen interface heeft de volgende voordelen:

- Een unieke uitstraling;
- De UI wordt ontworpen met de requirements van het project in het achterhoofd;
- Volledig controle over hoe het dashboard er uit gaat zien;
- Het ontwerp is voordat de ontwikkeling van start gaat al goedgekeurd door de product owner. Dit voorkomt plotselinge (cruciale) UI-wijzigingen in het ontwikkelproces;
- Aangezien ik het dashboard zelf in code uitdien te werken weet ik waar ik rekening mee kan houden in het ontwerpproces;
- Betere tijdschatting voor het realiseren van de sjablonen in het ontwerp.

Aangezien het project in iteraties wordt opgeleverd heb ik afgesproken met de product owner om per sprint een deel van het ontwerp uit te werken. Aan het begin van de planningsfase heb ik een standaard structuur en sjablonen ontworpen en goed laten keuren door de product owner. In het ontwerpproces heb ik gekeken hoe ik de conversationele en traditionele userinterface kan combineren. In Figuur 10 en 11 wordt de startpagina van het dashboard weergegeven zoals het eruit gaat zien voor een mantelzorger. In het ontwerp is te zien dat er geschakeld kan worden tussen de spraak en traditionele interface. In Bijlage 2 zijn alle schetsen van het dashboard opgenomen.



Figuur 10: Ontwerp dashboard (traditionele interface)



Figuur 11: Ontwerp dashboard (spraakinterface)

7. Onderzoek Single-Page Application frameworks

Een Single-Page Application (SPA) zorgt voor een betere gebruikerservaring, omdat de volledige pagina van een webapplicatie niet bij elke interactie van de gebruiker (opnieuw) ingeladen hoeft te worden. Om het visuele gedeelte van het dashboard zo gebruiksvriendelijk mogelijk te maken is er, zoals beschreven in hoofdstuk 3, bij TrueLime een wens ontstaan om een SPA te ontwikkelen. De applicatie reageert hierdoor sneller op de input van een gebruiker. Verder worden SPA-frameworks gebruikt om front-end ontwikkelaars meer te laten focussen op het bouwen van complexere interfaces door de ingebouwde patterns en functies van de gekozen framework te gebruiken (FusionCharts, 2018).

Er zijn diverse SPA-frameworks in opkomst en TrueLime wil graag meer weten over de voor- en nadelen van elk framework alvorens het ontwikkelen van het dashboard van start gaat. Om de deelvraag 'Welke Single-Page Application framework is geschikt voor het ontwikkelen van het dashboard' te beantwoorden is er tijdens het gesprek met de product owner drie populaire frameworks naar voren gekomen waar TrueLime meer over wil weten, namelijk: React, Angular en Blazor**.** Daarnaast heb ik door positieve verhalen op het internet Vue aangedragen om mee te nemen in het onderzoek.

Deze vier frameworks heb ik onderzocht om te achterhalen welk framework het meest relevant is voor het dashboard. De relevantie voor het dashboard bepaal ik met de volgende criteria:

1. Is het framework goed gedocumenteerd?
2. Hoe wordt het framework algemeen ontvangen in de community (populariteit)?
3. Wat is het instapniveau van het framework? Hoe eenvoudiger het instapniveau is, des te eerder ik kan beginnen met het ontwikkelen van het dashboard.
4. Hoe scoort het framework qua performance?

Figuur 12: Gekozen SPA-frameworks voor onderzoek

Van de bovengenoemde criteria vind ik goede documentatie en een eenvoudig instapniveau het meest belangrijk. Mocht een framework daar niet aan voldoen, dan resulteert dat tot een negatief advies. Per hoofdstuk vertel ik meer over het framework en eindig ik het onderzoek met een conclusie.

7.1 React

React is op het moment de leider van de JavaScript frameworks. Het framework is gemaakt door Facebook en wordt nog actief aan doorontwikkeld. React heeft een component-gebaseerd architectuur. Een component kan zowel business als HTML-opmaak bevatten. React heeft verder speciale objecten geïntroduceerd namelijk: state en props. Met beide objecten heb je de mogelijkheid om data door de hele applicatie te sturen. Een voordeel van React is dat het flexibel is. Je kan verschillende bibliotheken aan het framework koppelen om zo te voldoen aan de eisen van een project. Hier schuilt overigens ook een nadeel, namelijk dat er zoveel keuze is dat je niet altijd weet wat de juiste opties zijn voor het framework. React heeft verder niet echt een gedefinieerde development workflow. Een programmeur moet dat zelf gaan inrichten (Sviatoslav, 2017). React is verder alleen de 'View' van de welbekende Model-View-Controller-architectuur. Om de staat en

modellen van een applicatie te onderhouden moet er extra bibliotheken toegevoegd worden, zoals Redux, wat de complexiteit van de architectuur verhoogt (FusionCharts, 2018).

7.2 Angular

Angular is een grote concurrent van React. Het framework is ontwikkeld door Google en is mede hierdoor populair geworden. Een van de eerdere versie van Angular had een Model-View-Controller-architectuur. Door de hype van React hebben ze ook een component-based architectuur geïmplementeerd om te concurreren. Dit had overigens grote gevolgen voor het updaten van bestaande applicaties. Programmeurs moesten namelijk hun applicatie opnieuw ontwikkelen. Hiermee zijn veel programmeurs overgestapt naar concurrenten (FusionCharts, 2018).

Wanneer er een applicatie wordt gebouwd met Angular, dan split je een component in drie bestanden, namelijk: logica, layout en styling. Naast componenten heb je ook services, dependency injection en speciale HTML-attributen. Een voordeel is dat Angular al een workflow voor de programmeur heeft bepaald. Echter wordt de gedefinieerde workflow als heel complex ervaren door beginners (Sviatoslav, 2017).

7.3 Vue

Vue wordt gezien als een mix tussen Angular en React. De maker van Vue heeft beide concepten gepakt van de frameworks. Vue wilt dat logica en layout in één bestand wordt geplaatst. Net als bij React kunnen componenten met elkaar praten door de speciale objecten: props en state. Een mogelijke reden om voor Vue te kiezen is dat een React applicatie samen met Redux ingewikkeld en lastig te onderhouden is. Vue heeft zijn eigen state manager genaamd Vuex en wordt als minder complex ervaren, omdat het speciaal voor het framework is ontwikkeld.

Het verschil tussen Angular en Vue is dat Angular wordt gezien als een volwaardig, maar complex framework met een beperkt karakter. Vue heeft net als Angular een workflow, maar is wel flexibeler opgesteld. Een bijkomend voordeel is dat Vue geen superset[2] van JavaScript forceert. Hierdoor is de leercurve minder steil (Sviatoslav, 2017). Verder is het framework compact en scoort hierdoor ook beter qua performance (Schae, 2018).

Vue is verder een nieuwe framework en hierdoor ontbreken er standaard componenten en plugins. Het integreren van bijvoorbeeld Google Maps zou alsnog zelfgeschreven moeten worden. React en Angular hebben, door hun grote community en code base, al standaardoplossingen klaarliggen (Roberts, 2018).

7.4 Blazor

Microsoft heeft sinds 2010 een template engine Razor ontwikkelt dat wordt gebruikt in ASP.NET, een framework voor het ontwikkelen van webapplicaties. Door WebAssembly[3] is het mogelijk om hun

[2] Een superset van JavaScript is een nieuwe laag bovenop de programmeertaal. Het biedt de programmeur meer opties in de taal en een specifiek workflow.
[3] WebAssembly is een nieuw soort (low-level) code dat in moderne browsers kan draaien. Hierdoor is het mogelijk om in plaats van Javacript andere programmeertalen te gebruiken zoals C/C++.

eigen programmeertaal aan te bieden in de browser. Blazor staat dus eigenlijk voor Browser + Razor en past perfect in hun .NET ecosysteem. Het is net als Angular, React en Vue SPA-georiënteerd. Blazor bevat componenten wat eigenlijk een collectie is van afgeschermde C# en HTML-pagina's (Fortech, 2019).

Een groot voordeel van Blazor is dat je functionaliteiten kan delen tussen de server en client. Overigens zit Blazor (en WebAssembly) in een experimentele fase. Microsoft heeft aangegeven dat Blazor nog niet klaar is voor project dat in productie moet draaien. Verder zijn er ook wat valkuilen aanwezig, namelijk dat er geen (rechtstreekse) toegang is tot het Document Object Model[4] en de bijbehorende API. Dit betekent dat alle events in de browser alsnog via JavaScript afgehandeld moet worden. Dit resulteert in ongestructureerde code dat in twee omgevingen leeft. Verder is gebleken dat het wisselen tussen de twee programmeertalen ten kostte gaat van performance (Strahl, 2018).

7.5 Resultaat

Mijn advies is om Vue te gebruiken voor het dashboard, om de volgende punten:

- Scoort van de vier frameworks het beste op performance
- Framework is compact (Vue is gecomprimeerd maar 18 kilobytes groot)
- Simpele integratie (het toevoegen van bijvoorbeeld een state manager brengt weinig complexiteit met zich mee, wat in meeste frameworks wel het geval is)
- Goede documentatie
- Instapniveau voor beginners is laag
- Groeit per jaar in populariteit

Aangezien het project binnen een aantal weken gerealiseerd moet worden lijkt het me een verstandige keuze om vooral door bovengenoemde punten met Vue aan de slag te gaan. Hierdoor is het mogelijk om zo efficiënt en vlot mogelijk het dashboard te realiseren, waarbij de gebruikerservaring van de applicatie goed mee groeit. Mijn advies heb ik overlegd met de product owner. Uiteindelijk hebben we samen besloten om het dashboard te ontwikkelen met Vue als SPA-framework.

Waarom React niet geschikt is voor het dashboard

React is het populairste framework op dit moment. TrueLime heeft zelf ook meerdere keren gebruik gemaakt van React in projecten. Het is hierdoor een interessante kandidaat. Het framework is flexibel en is eigenlijk alleen de 'weergave' van de welbekende Model-View-Controller-architectuur. Zoals eerder aangegeven kan de staat van een applicatie beheerd worden door bijvoorbeeld Redux (een state manager), waarbij de complexiteit van de architectuur wordt verhoogd. Hierdoor is het instapniveau lastiger en resulteert in een negatief advies.

Waarom Blazor niet geschikt is voor het dashboard

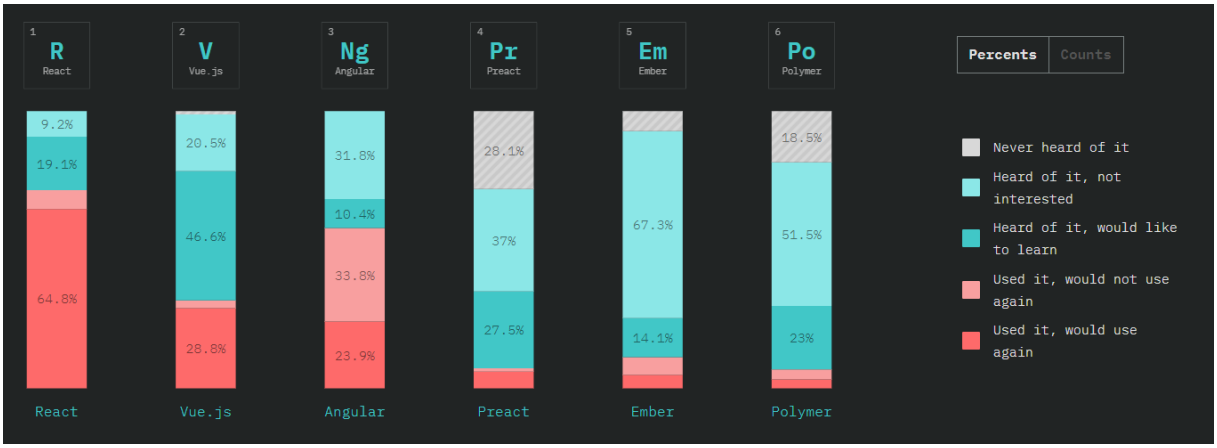
Blazor is mede mogelijk gemaakt door Web Assembly wat nog in een experimentele fase zit. Hierdoor zijn niet alle functionaliteiten beschikbaar die in JavaScript wel zitten. Zo is het bijvoorbeeld niet

[4] Het Document Object Model zorgt voor een structurele representatie van HTML-documenten, wat het mogelijk maakt de inhoud en visuele presentatie te veranderen (Mozilla, sd).

mogelijk om (rechtstreeks) gebruik te maken van het Document Object Model en de bijbehorende API. Er zijn omwegen mogelijk, maar dit gaat ten kostte van performance. Ten slotte geeft Microsoft zelf ook aan dat Blazor nog niet klaar is om in productie gebruikt te worden. Wel is dit een interessant techniek om in de gaten te houden voor in de toekomst! Want ik denk dat het wel veel voor TrueLime kan betekenen, omdat ze de ecosysteem van Microsoft gebruiken.

Waarom Angular niet geschikt is voor het dashboard

Angular is al jaren een grote concurrent van React. Om met de nieuwe trends mee gaan in de front-end wereld hebben ze grote wijzigingen in hun interne structuur aangebracht. Programmeurs moesten hun applicatie opnieuw schrijven om te kunnen updaten. Hierdoor zijn er veel ontwikkelaars overgestapt naar React. Verder is het instapniveau voor beginners hoog, met name omdat het gebruik maakt van TypeScript, complexe terminologieën en technieken. Tot slot zijn er 20,000 JavaScript ontwikkelaars geïnterviewd waaruit gebleken is dat een hoog percentage (33.8%) niet meer gebruik willen maken van het framework (The State of JavaScript, 2018). Aangezien het project binnen een korte tijd gerealiseerd moet worden lijkt het me niet verstandig om met Angular in zee te gaan. In Figuur 13 is te zien dat Angular niet goed scoort (33,8%) op populariteit bij de ontwikkelaars.



Figuur 13: Statistieken frameworks (The State of JavaScript, 2018)

Met Vue als gekozen framework kan het ontwikkeltraject beginnen. In het volgende hoofdstuk beschrijf ik hoe de eerste sprint is verlopen.

8. Afspraken - Sprint 1

Voor de deelvraag ‘Welke functionaliteiten moet het dashboard bevatten om de interactie met de chatbot te verbeteren’ heeft de product owner bepaald wat er als eerst gerealiseerd moet worden. Een mantelzorger kan voor zijn patiënt afspraken bijhouden. Dit kan de mantelzorger (en patiënt) op dit moment via de chatbot vragen. In deze sprint ga ik de bestaande afspraken ophalen van de chatbot en tonen in het dashboard. Daarnaast is het de bedoeling om in het dashboard zelf afspraken te kunnen toevoegen, bewerken en verwijderen. In Tabel 4 zijn de User Stories met bijbehorende story points beschreven.

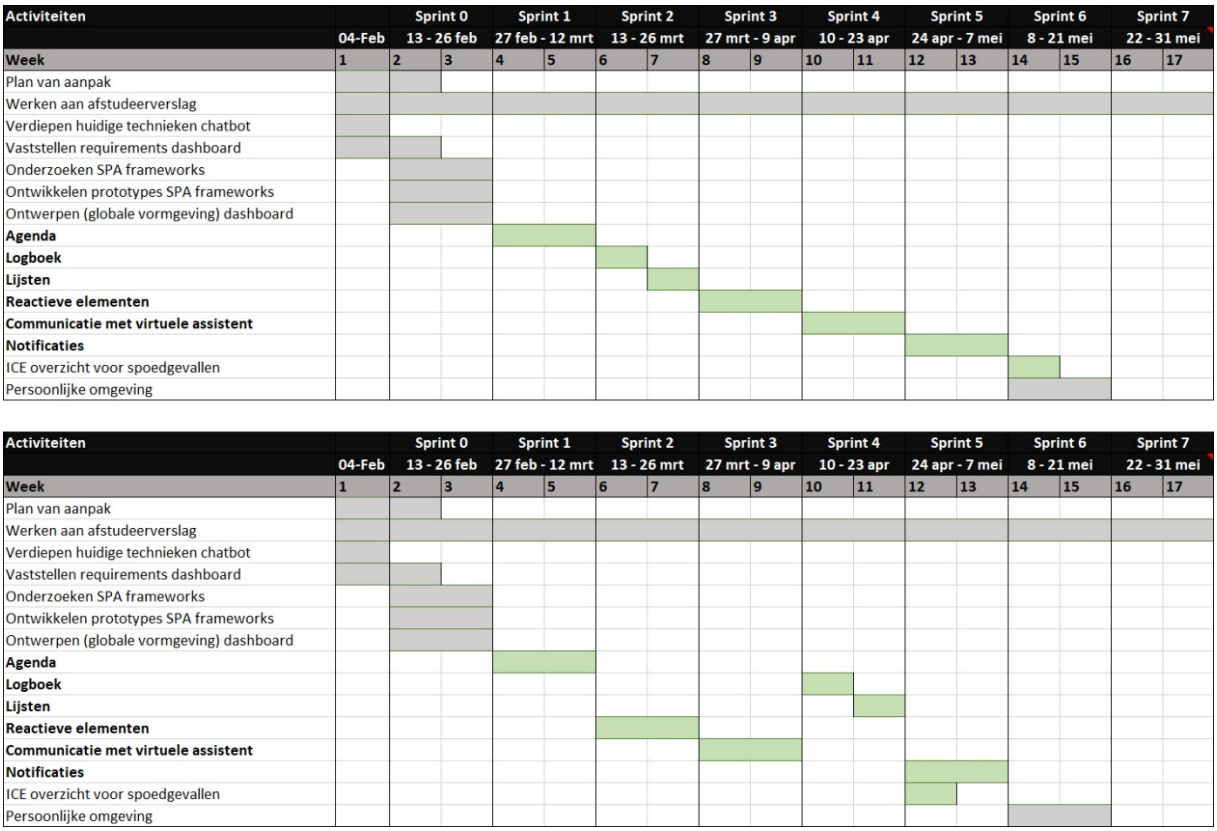
ID	User Story	Points
31705	Als ontwikkelaar wil ik een ontwikkelomgeving opzetten, zodat ik de virtuele assistent kan ontwikkelen.	3
32646	Als ontwikkelaar wil ik dat het gedeelde domein (package) automatisch wordt gebuild en gereleaset, zodat wijzigingen niet handmatig overgezet hoeft te worden bij de chatbot en dashboard project.	3
31723	Als mantelzorger wil ik alle afspraken per maand kunnen inzien, zodat ik meer inzichten heb over wat er gaat gebeuren met mijn patiënt	8
31726	Als mantelzorger wil ik via een kalender het aantal afspraken kunnen inzien, zodat ik in één oogopslag weet hoeveel afspraken er zijn gemaakt in een maand	8
31727	Als mantelzorger wil ik afspraken kunnen inplannen, zodat iedereen die gebruik maakt van het dashboard op de hoogte is van wat er gaat komen	8
31729	Als mantelzorger wil ik een afspraak kunnen bewerken, zodat ik makkelijker de gewijzigde afspraken kan doorgeven	8
31730	Als mantelzorger wil ik een afspraak kunnen verwijderen, zodat verkeerde afspraken niet meer worden getoond in mijn agenda	1

Tabel 4: User Stories voor sprint 1 (zie Bijlage 3 voor acceptatiecriteria)

Aan het begin van de sprint is de planning, na overleg met de product owner, gewijzigd. In het volgende hoofdstuk vertel ik waarom de wijziging een positief effect heeft op het project.

8.1 Aangepaste planning

Tijdens de Sprint Planning-sessie is er een wijziging in de planning gekomen. Na overleg met de product owner leek het ons verstandig om eerst één functionaliteit van de chatbot uit te werken om vervolgens te starten met het ontwikkelen van reactieve elementen en de koppeling tussen de chatbot en dashboard. Hierbij is de hele integratie stap gerealiseerd en kan de laatste skills toegevoegd worden. Het voordeel van de nieuwe planning is dat mogelijke complicaties in een vroeg stadium gesignaleerd kunnen worden. Hierdoor is er ook ruimte ontstaan om in Sprint 4 het dashboard daadwerkelijk te laten testen door de doelgroep. De bevindingen kunnen we goed gebruiken om in de laatste sprints de overige functionaliteiten en uiteraard de feedback van de tester te verwerken. Zie Figuur 14 voor de aangepaste planning.



Figuur 14: De aangepaste planning (onderaan)

Samen met de product owner hebben we besloten om eerst te focussen op het beheren van afspraken, hiermee wordt het volgende traject bewandeld:

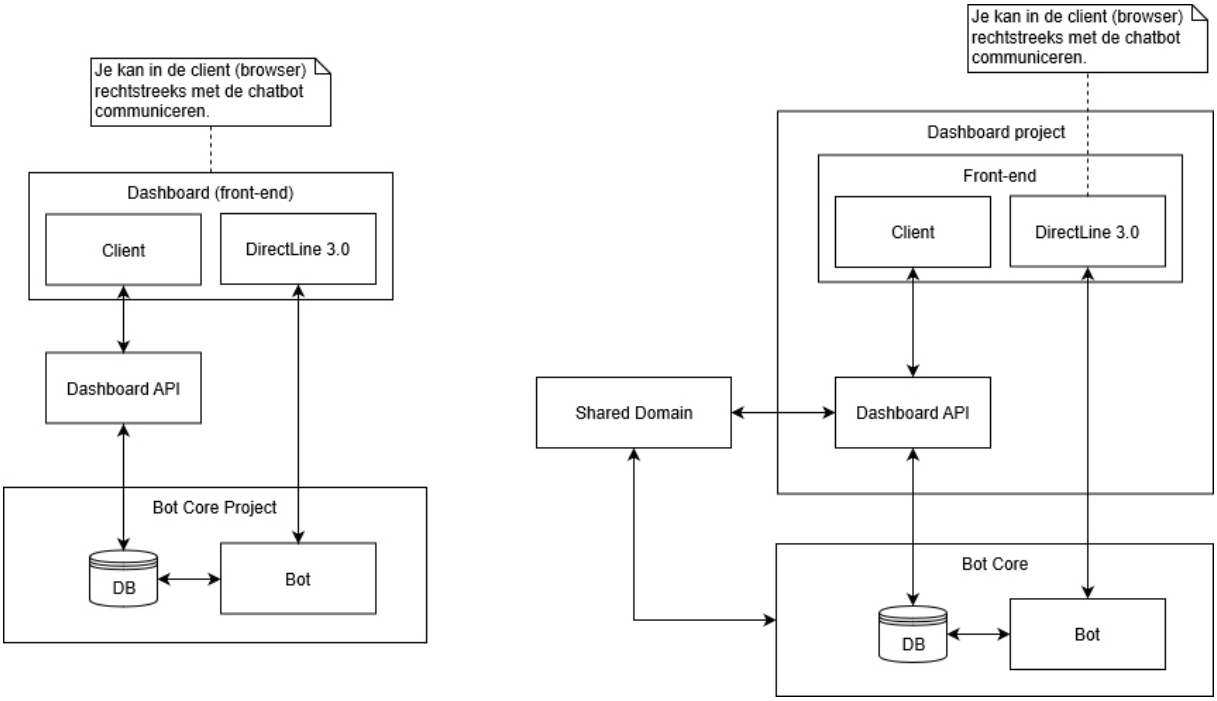
- De afspraken van de chatbot visualiseren naar het dashboard (sprint 1);
- Het beheren van afspraken mogelijk maken in het dashboard (sprint 1);
- Het dashboard moet reactief zijn, dat betekent dat wijzigingen in afspraken direct getoond wordt aan de gebruiker (sprint 2);
- Koppeling realiseren tussen de chatbot en dashboard (sprint 3);
- Het rechtstreeks kunnen communiceren van het dashboard naar de chatbot (sprint 3);

De overige skills (logboek en lijsten) van de chatbot zijn verplaatst naar de Sprint 4. In het volgende hoofdstuk vertel ik meer over hoe en waarom de architectuur van het dashboard gewijzigd is tijdens het opzetten van de ontwikkelomgeving.

8.2 Architectuur

In de planningsfase zijn er verschillende architectuurschetsen gemaakt om de deelvraag ‘Hoe kan de communicatie tussen het dashboard en chatbot gerealiseerd worden?’ te onderzoeken. Na overleg met de product owner is er één architectuur gekozen zoals beschreven in hoofdstuk 5.2. Met de DirectLine API van Microsoft is het mogelijk om rechtstreeks met de chatbot te communiceren. Hierdoor wordt het dashboard weliswaar als een nieuw kanaal gezien. Het ophalen van data wordt mogelijk gemaakt door de Dashboard API. Een groot voordeel is dat dashboard specifieke functionaliteiten niet in de chatbot-omgeving bestaat. Een nadeel van de gekozen architectuur is dat het dashboard en chatbot wel overeenkomsten met elkaar hebben, zoals database modellen. Mocht er een wijziging plaatsvinden aan zo’n model, dan moet de programmeur twee omgevingen gaan aanpassen. In Figuur 15 is er een gedeeld domein (model rechts) toegevoegd aan de architectuur. In

het domein staan alle gedeelde functies en modellen waar zowel de chatbot en het dashboard gebruik van kunnen maken. Hiermee is het nadeel van de gekozen architectuur opgelost.



Figuur 15: Architectuur met gedeelde domein

In het volgende hoofdstuk vertel ik meer over hoe het gedeelde domein is ingericht.

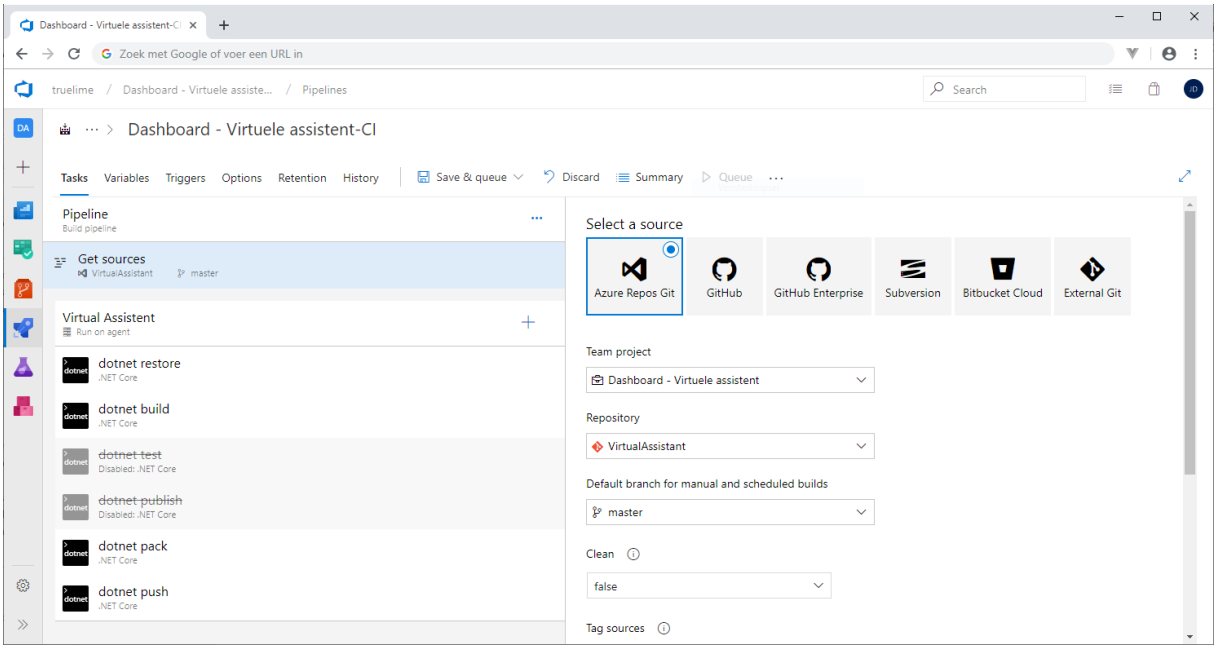
8.3 Gedeeld domein

Het gedeelde domein moet beschikbaar worden gesteld voor de chatbot en dashboard-omgeving. Beide omgevingen zijn ontwikkeld met de .NET framework van Microsoft. Het framework biedt een package manager aan, genaamd NuGet, om met programmeurs code te creëren, te delen en te gebruiken. Het gedeelde domein aanbieden als een NuGet-package is daarom een ideale oplossing (Microsoft, 2018). Hierdoor is het mogelijk om de modellen en functies in het gedeelde domein te gebruiken in jouw 'afgeschermde' omgeving.

Azure DevOps

Het (handmatig) uitbrengen van nieuwe versies van het gedeelde domein kost tijd en is hierdoor niet efficiënt. Daarnaast moet de programmeur ook de nieuwe versie uitrollen bij het dashboard en chatbot project. Er zijn veel stappen nodig om de gedeelde modellen en functies beschikbaar te stellen. Het automatiseren van het uitbrengen van een nieuwe versie zal een logische stap zijn om het gehele proces te versimpelen en dus ook te versnellen (User Story 32646, Tabel 4)

Het gedeelde domein is een .NET Core applicatie dat wordt gehost in de Azure DevOps omgeving. Hierdoor is het mogelijk om een build pipeline[5] in te richten, waarmee nieuwe wijzigingen in de applicatie automatisch worden gedetecteerd.

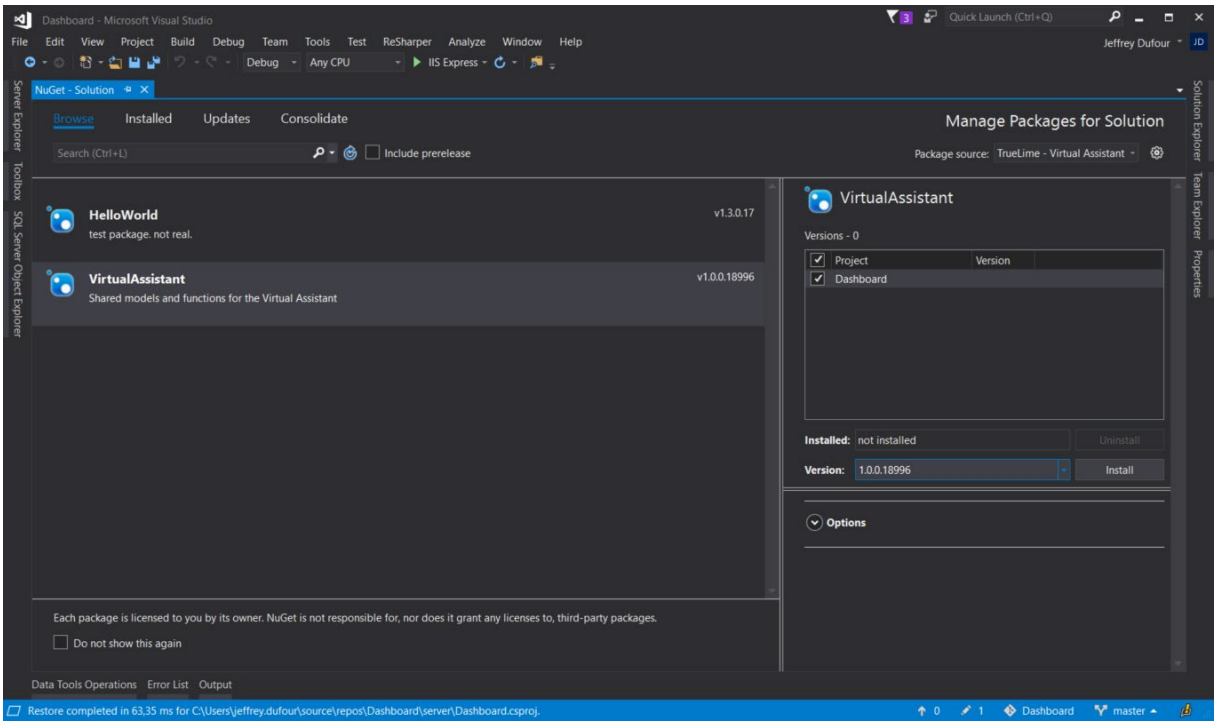


Figuur 16: Build pipeline van het gedeelde domein

In Figuur 16 is aan de rechterzijde te zien dat de repository[6] van het gedeelde domein als bron is geselecteerd. Hierdoor is het mogelijk dat elke nieuwe wijziging in de code automatisch de build flow start. Aan de linkerzijde staan de taken die worden uitgevoerd als een build van start gaat. Het project wordt met een virtuele machine ingesteld en uitgebracht als een nieuwe NuGet package. Hierdoor kan je, zoals in Figuur 17 wordt weergegeven, een nieuwe versie van het gedeelde domein updaten in de chatbot of het dashboard project.

Na het inrichten van de ontwikkelomgeving (User Story 31705, Tabel 4) ben ik begonnen met het ophalen en tonen van afspraken in het dashboard (User Story 31723 & 31726, Tabel 4). Tijdens de ontwikkeling zijn er wat technische uitdagingen naar voren gekomen, namelijk het afhandelen en doorsturen van data binnen de componenten. Om de kwaliteit van het front-end te waarborgen heb ik een state manager geïmplementeerd. In het volgende hoofdstuk vertel ik daar meer over.

[5] Een pipeline is een set van geautomatiseerde processen.
[6] Een repository is een opslagplaats voor code met versiebeheer.

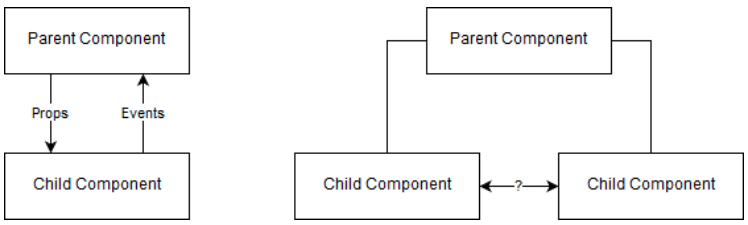


Figuur 17: Het gedeelde domein als NuGet package

8.4 State manager

Vue heeft een component-based architectuur, dat eigenlijk een methode is om onderdelen uit een groot userinterface om te zetten naar zelfonderhoudende, onafhankelijke microsystemen. Een component kan gezien worden als een kleine feature, dat een deel uitmaakt van een userinterface. Het is hierbij ook belangrijk om te weten dat een component ook onafhankelijk uitgevoerd kan worden. Hierdoor is het mogelijk componenten te hergebruiken (Shapiro, 2016).

Data dat in een component leeft wordt ook wel **component-level data** genoemd. In het framework is het mogelijk om data van een parent component door te sturen naar zijn child componenten. Het doorsturen van data is een eenrichtingscommunicatie. Wel is het mogelijk om een bevestiging terug te sturen naar de parent component via een event. Het is niet mogelijk om twee child componenten rechtstreeks met elkaar te laten communiceren. In Figuur 18 is de communicatie tussen de componenten gevisualiseerd.



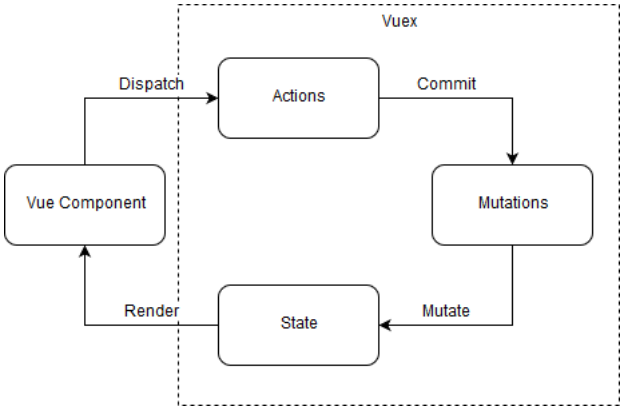
Figuur 18: Communicatie binnen Vue

Voor **application-level data** kan dit resulteren in een chaotische structuur. Wat er namelijk kan ontstaan is dat componenten kennis krijgen van data om het vervolgens alleen door te sturen naar een onderliggende component. Het is zonde om een componenten kennis te laten hebben van data die ze niet zelf gebruiken. Een bijkomend nadeel is dat wanneer data door veel componenten heen stroomt, het lastig te detecteren is waar data vandaan komt en welke component het heeft aangepast (Djirdeh, 2018).

Een oplossing voor het beheren van application-level data is Vuex, een state manager. De data worden opgeslagen in een store dat vervolgens binnen componenten opgehaald kan worden. Er zijn twee grote voordelen aan het gebruik nemen van Vuex:

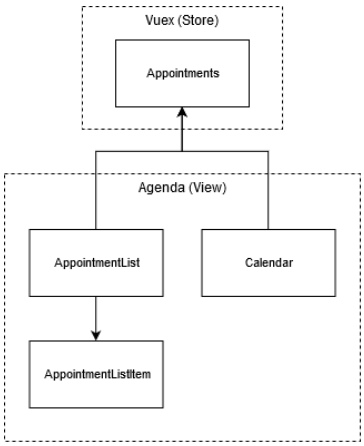
1. Vuex stores zijn reactief. Aangepaste data in de store worden efficiënt en reactief aangepast bij de componenten die er gebruik van maken.
2. Het is niet mogelijk om rechtstreeks data aan te passen in de store. Om data te muteren (aanpassen) moet er een aanvraag gestuurd worden naar Vuex. Met resultaat dat wijzigingen goed te traceren zijn en helpt om de structuur van de applicatie beter te begrijpen.

In Figuur 19 is te zien hoe data wordt aangepast binnen een Vuex store. Een component verstuurd (dispatch) een verzoek naar een action. Actions worden gebruikt om asynchrone operaties af te handelen en kan business logica bevatten. Een action houdt zich niet bezig met het muteren van de Vuex store (ook wel een state genoemd), maar biedt juist de data aan een mutation. Met een mutation is het alleen mogelijk data van de store te wijzigen (Vuex, n.d.).



Figuur 19: Flow van Vuex

Voor het project is Vuex ingesteld. Nu is het mogelijk om afspraken vanuit de API op te slaan in de een data store. Vervolgens kan mijn agenda en kalender component gebruikmaken van de data, zonder dat de bovenliggende componenten ervan af weten. In Figuur 20 is er een globale weergave van de Agenda view te zien.



Figuur 20: Globale weergave Agenda

In het volgende hoofdstuk bespreek ik hoe de gehele sprint is verlopen.

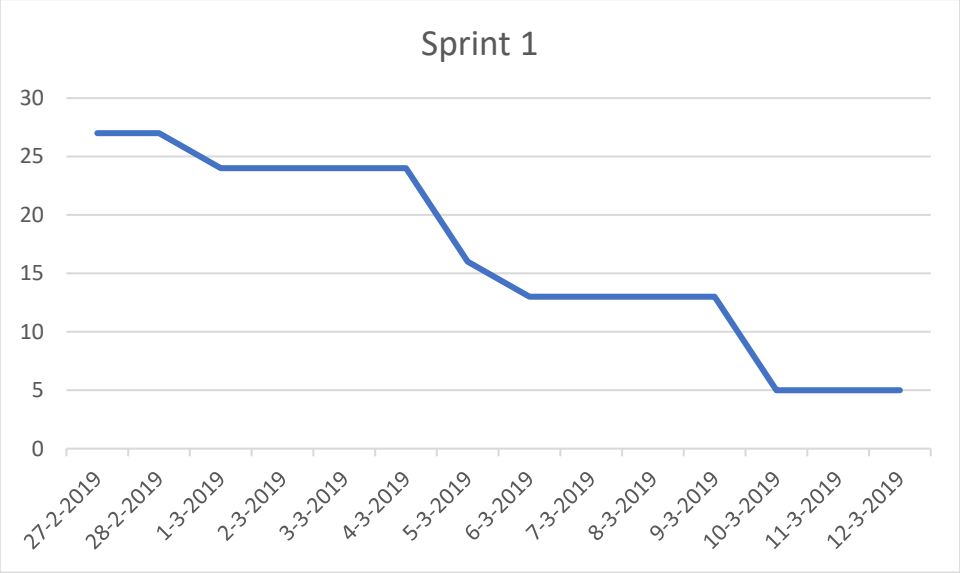
8.5 Resultaat

Voor de eerste sprint waren er 27 story points ingepland. Uiteindelijk heb ik 20 punten kunnen afronden (aldus mijn velocity). In de eerste iteratie heb ik me vooral beziggehouden met het opzetten van de ontwikkelomgeving, een gedeeld domein aangemaakt en worden afspraken uit de chatbot getoond in het dashboard. In Tabel 5 is de huidige status van het prototype weergegeven.

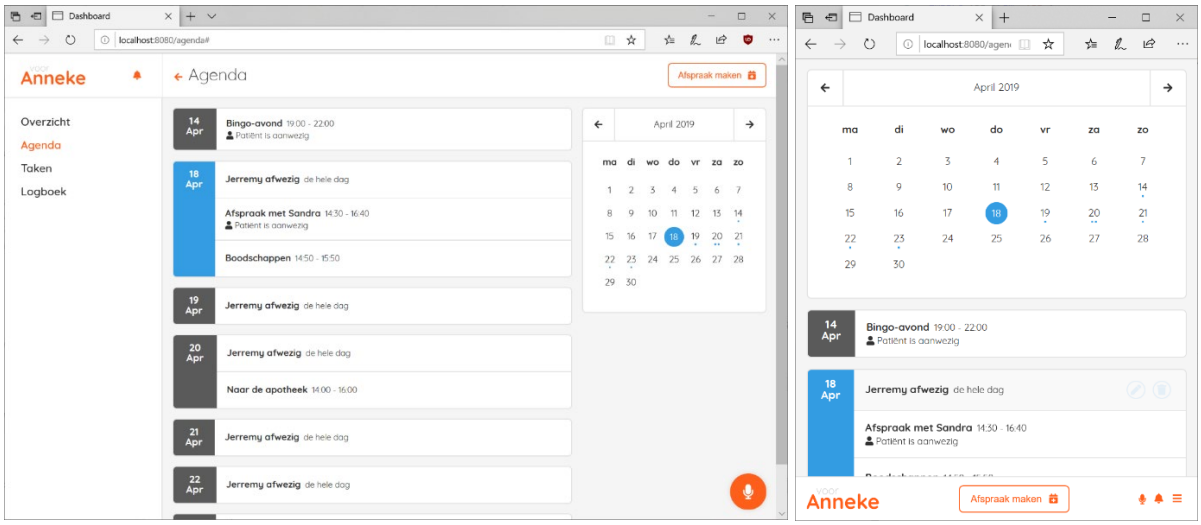
Requirement	**Status**
Must have	
Beheren van logboek	Ingepland
Agenda (afspraken)	In ontwikkeling
Lijsten (taken)	Ingepland
Reactieve elementen (realtime dataverwerking)	Ingepland
Communicatie met virtuele assistent (vraag en antwoord)	Ingepland
Should have	
Persoonlijke omgeving	Ingepland
Notificaties	Ingepland
ICE-overzicht voor spoedgevallen	Ingepland

Tabel 5: Huidige status prototype na sprint 1 (zie hoofstuk 4 voor de beschreven requirements)

In de Burndown Chart voor deze sprint (Figuur 21) is te zien dat ik niet alle punten heb afgerond. Dit komt omdat tijdens de sprint een belangrijke requirement bij is gekomen, namelijk het build proces van het gedeelde domein automatiseren (zie hoofdstuk 8.3). Vervolgens ben ik langer bezig geweest met het opzetten van de front-end framework, namelijk met de state manager en het animeren van bepaalde elementen. Mijn schatting was te positief op basis van de taken die ik eerder in gedachten had. Hierdoor is het niet mogelijk om afspraken te beheren, dit neem ik mee naar de volgende sprint. In Figuur 22 heb ik een impressie van de afspraken pagina toegevoegd. In het volgende hoofdstuk vertel ik meer over hoe de tweede iteratie van het project is verlopen.



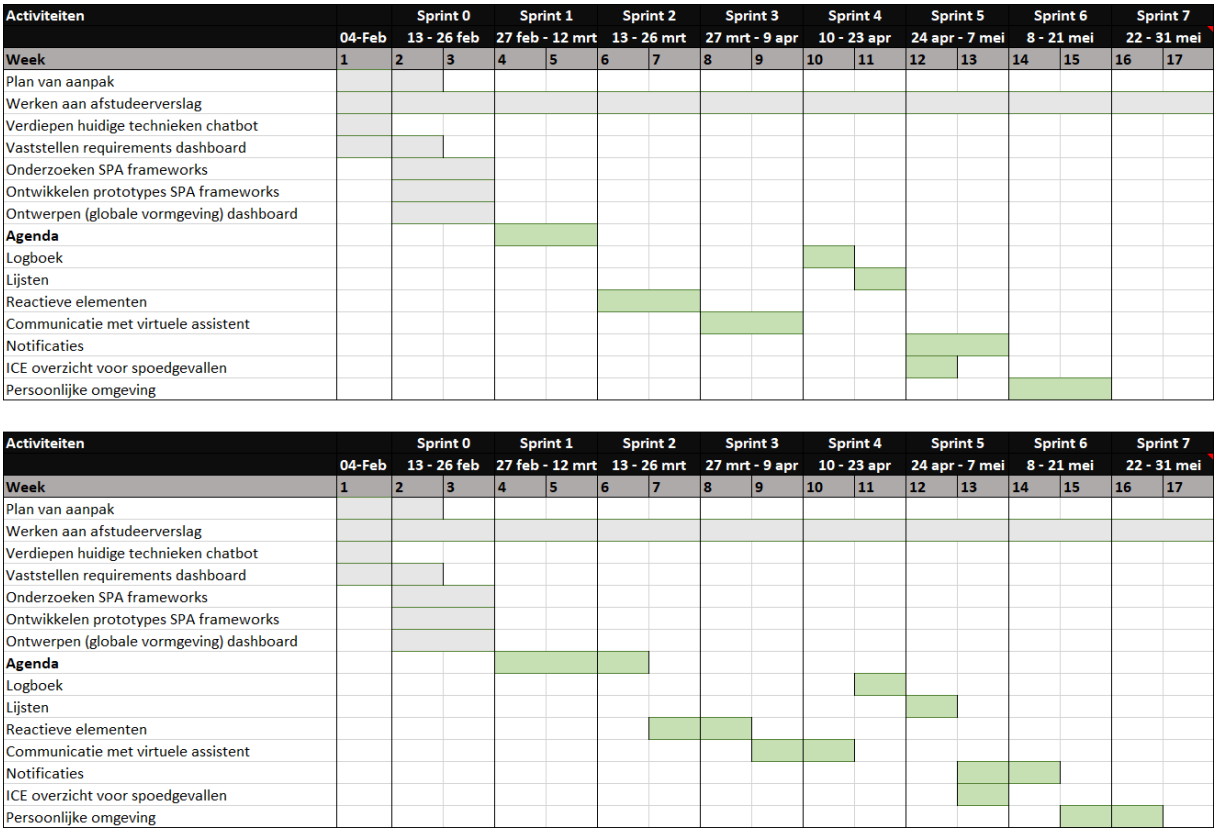
Figuur 21: Burndown Chart



Figuur 22: Afspraken pagina dashboard

9. Beheer omgeving en reactieve elementen - Sprint 2

Voor de tweede sprint ga ik werken aan de beheeromgeving voor de gemaakte afspraken en ga ik de huidige functionaliteiten van het dashboard reactief maken. Zoals in hoofdstuk 8.5 is beschreven heb ik de sprint niet gehaald. In de aangepaste planning (Figuur 23) zijn alle activiteiten een halve sprint naar voren geschoven. Sprint 7 zal voornamelijk voor uitlopende taken ingezet worden. Hierdoor zijn er geen complicaties ontstaan voor de deadline van het project. Sprint 2 wil ik een velocity van 22 story points bereiken, dit zijn twee punten meer dan dat ik vorige sprint had behaald.



Figuur 23: Aangepaste planning voor sprint 2

In Tabel 6 heb ik de User Stories opgenomen die ik samen met de product owner heb opgesteld voor de requirements ‘agenda’ en ‘reactieve elementen’ (zie hoofdstuk 4 voor een overzicht van alle requirements).

ID	User Story	Points
31727	Als mantelzorger wil ik afspraken kunnen inplannen, zodat iedereen die gebruik maakt van het dashboard op de hoogte is van wat er gaat komen	3
31729	Als mantelzorger wil ik een afspraak kunnen bewerken, zodat ik makkelijker de gewijzigde afspraken kan doorgeven	8
31730	Als mantelzorger wil ik een afspraak kunnen verwijderen, zodat verkeerde afspraken niet meer worden getoond in mijn agenda	1

33162	Als ontwikkelaar wil ik een architectuur met WebSockets, zodat het mogelijk is om real-time wijzigingen te kunnen sturen en ontvangen tussen dashboards	5
33118	Als mantelzorger wil ik na het maken van een afspraak direct de wijzigingen in de lijst weergaven kunnen zien, zodat ik weet dat de afspraak goed is verwerkt	5

Tabel 6: User Stories voor sprint 2 (zie Bijlage 4 voor acceptatiecriteria)

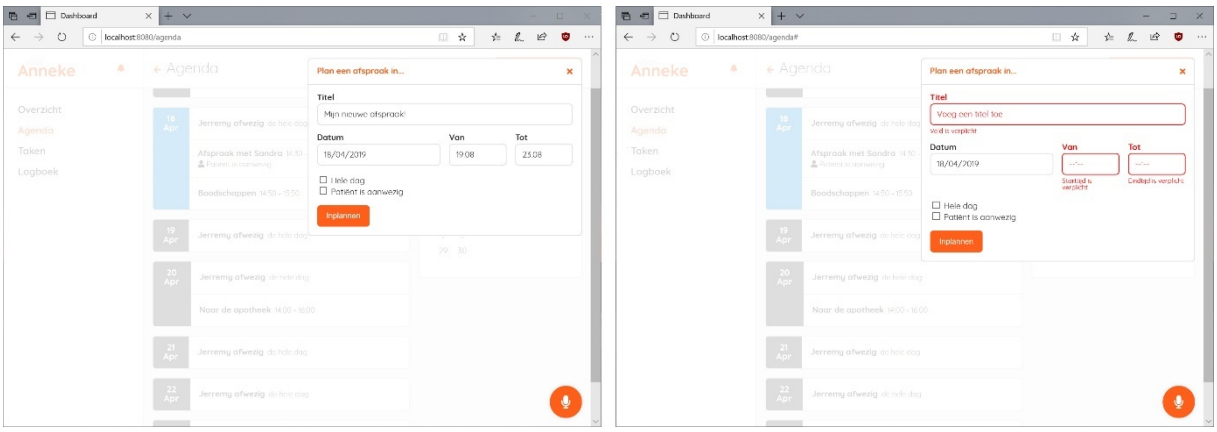
9.1 Beheeromgeving

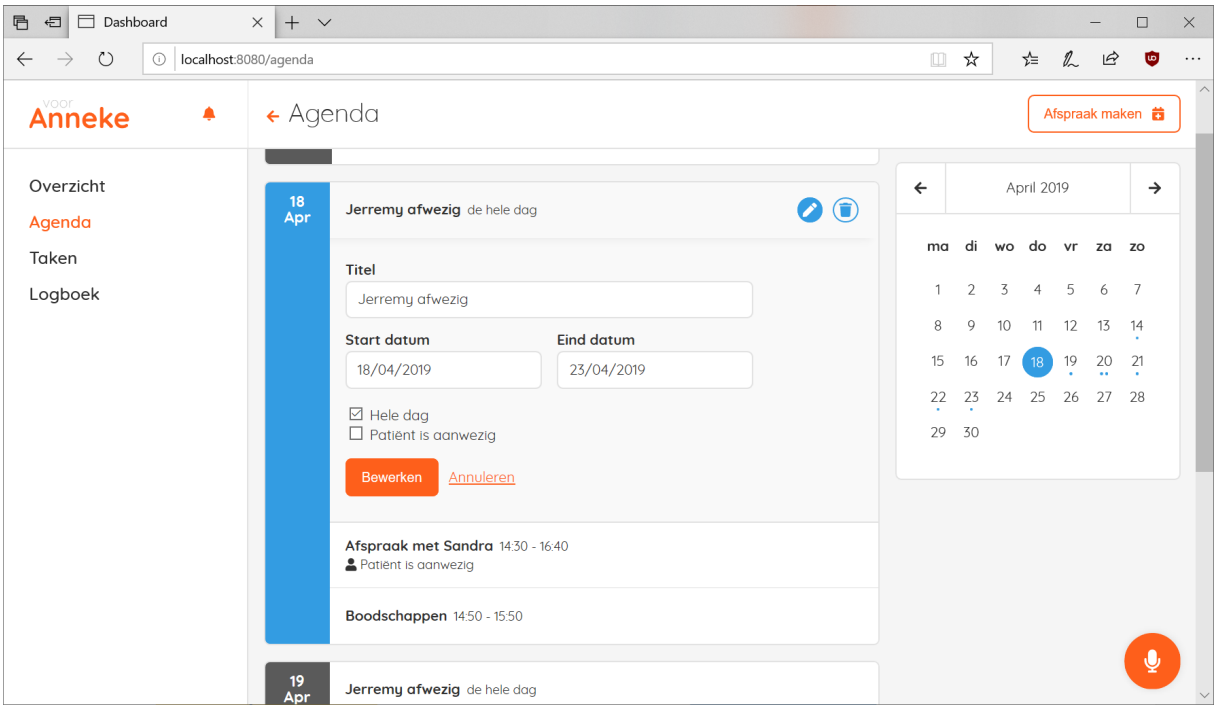
Voor de beheeromgeving van afspraken wilde ik een formulier component ontwikkelen, dat zowel items kan bewerken als toevoegen. Dit is om redundante code te voorkomen, want de velden van het formulier komen overeen, alleen wat er met de data gebeurt is anders. Two-way data binding is mogelijk in Vue. Dat wil zeggen dat data zowel in de Document Object Model (DOM) als in JavaScript in verbinding staat met elkaar. Wijzigingen worden direct verwerkt en getoond.

Voor de formulier velden heb ik eigen componenten gemaakt, zodat ik het structuur en vormgeving eenmalig hoef te definiëren (User Story 31727, 31729 & 31730, Tabel 6). Om data makkelijk te synchroniseren met elkaar heb ik bovengenoemde (two-way binding) techniek toegepast. Het resultaat is dat tijdens het bewerken van een afspraak, de wijziging direct getoond wordt in het item. Dit zorgt voor een responsieve ervaring. Naast data synchronisatie heb ik me ook gefocust op validatie. Ik hou rekening met het volgende:

1. Titel en datum zijn verplicht;
2. Het wordt niet toegestaan om een latere ‘van’ tijd aan te geven dan een ‘tot’ tijd;
3. Het wordt niet toegestaan om een eerdere ‘tot’ tijd aan te geven dan een ‘van’ tijd;
4. Het wordt niet toegestaan om een latere ‘van’ datum aan te geven dan een ‘tot’ datum;
5. Het wordt niet toegestaan om een eerdere ‘tot’ datum aan te geven dan een ‘van’ datum;
6. Titel moet langer dan 5 karakters zijn.

Het is nu mogelijk om in het dashboard afspraken te beheren (inclusief validatie), zoals in Figuur 24 te zien is. In het volgende hoofdstuk vertel ik meer over het reactief maken van de elementen.

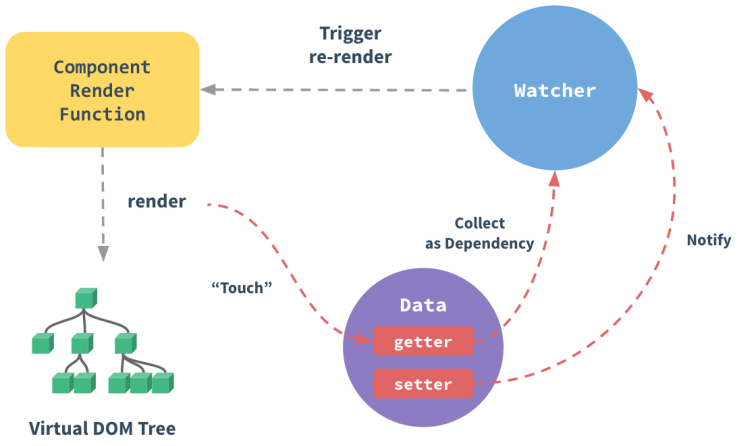




Figuur 24: Afspraken beheren

9.2 Reactieve elementen

Zoals eerder besproken in hoofdstuk 8.4 kan er binnen een component data opgevraagd worden uit de Vuex store, maar is het ook mogelijk om de opgehaalde data door te sturen naar de onderliggende componenten. Vue en Vuex hebben beiden een reactief systeem, dat wil zeggen dat datawijzigingen worden gedetecteerd. De componenten die de data in gebruik hadden worden op een efficiënte manier bijgewerkt. Tijdens het ontwikkelen van de beheeromgeving kwam ik tegen het probleem aan dat een component niet de juiste informatie liet zien, terwijl de data wel was aangepast in de store. Om meer inzicht te krijgen op de situatie heb ik onderzocht hoe Vue onder de motorkap zijn data detecteert en verwerkt. Na het definiëren van een component zorgt Vue ervoor dat er een watcher wordt aangemaakt die gedefinieerde data in de gaten gaat houden. Na elke wijziging gaat de watcher vragen aan een render functie om de component opnieuw te tekenen op het scherm. Dit proces is in Figuur 25 gevisualiseerd.



Figuur 25: Detecteren van wijzigingen in Vue (Vue.js, sd)

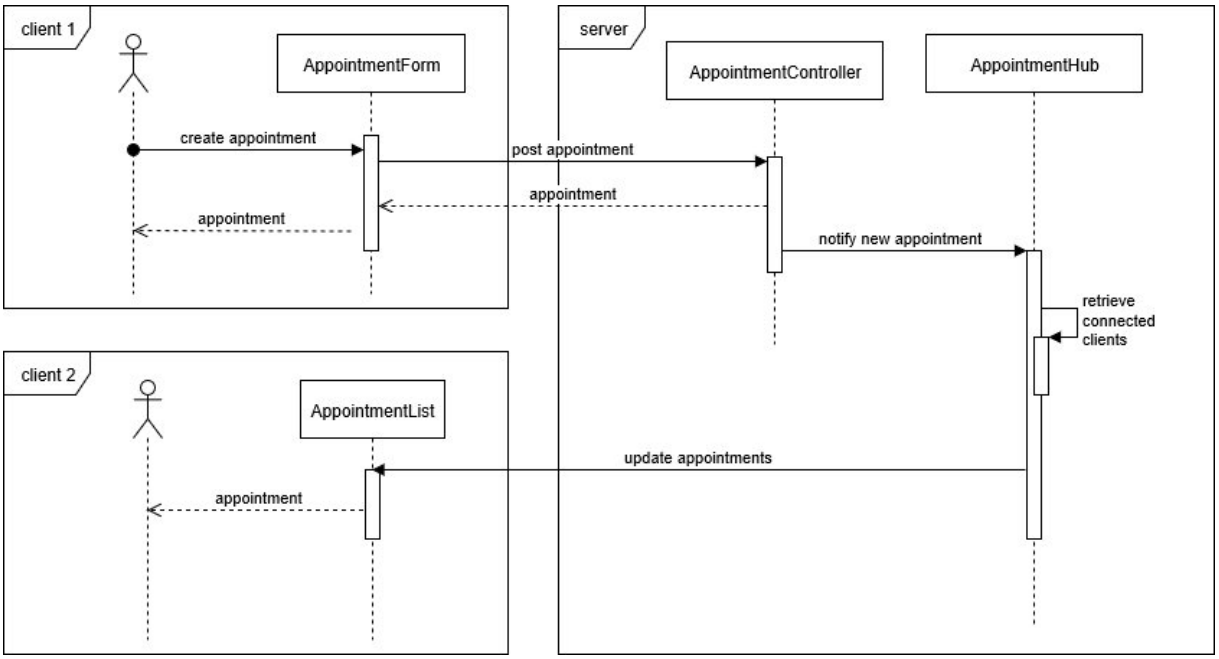
Het is belangrijk dat de datastructuur van een Vue component al van tevoren wordt vastgelegd. Dit zorgt er namelijk voor dat er een watcher aan gekoppeld kan worden. Verder moet de data binnen een component onveranderlijk (immutable) zijn, dat wil zeggen dat er geen directe wijzigingen in een object of lijst mogen plaatsvinden. Een oplossing is om een nieuwe lijst of object aan te bieden met de aangepaste gegevens. Hierdoor weet Vue, door middel van de watcher, dat de component opnieuw getekend moet worden. Met deze nieuwe inzichten heb ik uiteindelijk het probleem met de beheeromgeving kunnen oplossen. Data werd namelijk rechtstreeks aangepast en niet als een nieuw object aangeboden. Hierdoor wist Vue niet dat er een aanpassing had plaatsgevonden. Hieronder heb ik een versimpelde versie als code snippet toegevoegd om het probleem en oplossing te visualiseren.

```
// Aanmaken van een data object
let data = { title: 'Hello world!' };

// Dit wordt niet gedetecteerd in Vue (past rechtstreeks het object aan)
data.title = "Goodbye!";

// Dit wordt wel gedetecteerd (object wordt opnieuw aangeboden)
data = { title: 'Goodbye!' };
```

Elementen worden reactief afgehandeld binnen Vue, mits je objecten op de juiste manier aanmaakt en bewerkt. Dit zorgt er overigens niet voor dat data realtime uitgewisseld kan worden tussen verschillende apparaten. In het volgende hoofdstuk vertel ik meer over de techniek WebSockets dat realtime dataverwerking mogelijk maakt.

9.3 WebSockets

Een WebSocket is een netwerkprotocol dat bidirectionele dataverkeer mogelijk maakt tussen webbrowsers en webservers. Nadat er een verbinding tussen een client en server is gemaakt kan er continue berichten uitgewisseld worden, aldus realtime dataverwerking. Microsoft heeft voor zijn .NET ecosysteem de library SignalR ontwikkeld dat gebruik maakt van dit protocol (Microsoft, sd). Met SignalR is het mogelijk om een hub aan te maken dat berichten kan versturen en ontvangen via events. Het resultaat is dat een webbrowser (client) een bericht stuurt naar de server en vervolgens stuurt hij, via de hub, het bericht door naar alle actieve clients. In Figuur 26 is te zien wat er gebeurt als een mantelzorger een nieuwe afspraak toevoegt vanuit het dashboard.



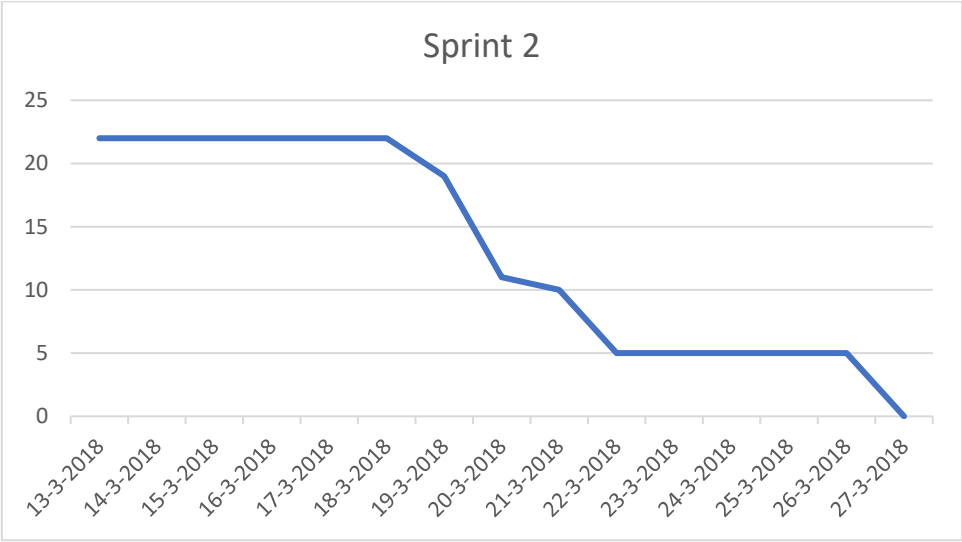
Figuur 26: Toevoegen van een afspraak

Met WebSockets hoeft een mantelzorger niet meer haar pagina opnieuw te verversen om nieuwe afspraken te zien (User Story 33118 & 33162, Tabel 6). Aangepaste data worden nu direct op het dashboard weergegeven.

In het volgende hoofdstuk bespreek ik hoe de gehele sprint is verlopen.

9.4 Resultaat

Voor de tweede iteratie van het project waren er 22 story points ingepland. Dit zijn twee story points meer dan mijn velocity. Dit is bewust ingepland om mijn achterstand van de vorige sprint een deel in te halen. Ik heb de beheeromgeving van afspraken afgerond en is het nu mogelijk om realtime data uit te wisselen tussen verschillende webbrowsers (clients). In de Burndown Chart (Figuur 27) is te zien dat ik deze sprint goed heb afgerond.



Figuur 27: Burndown Chart sprint 2

Voor deze sprint is er is een ‘must have’ requirements afgerond voor het prototype, namelijk agenda (zie Tabel 7). Requirement ‘reactieve elementen’ is voor een groot deel gerealiseerd. Zoals eerder beschreven is het mogelijk om realtime data te sturen en ontvangen tussen webbrowsers. Het detecteren van nieuwe wijzigingen vanuit een andere client, dus bijvoorbeeld een Google Home apparaat moet nog worden gerealiseerd. Volgende sprint ga ik verder met ‘reactieve elementen’ en wordt er een start gemaakt om de koppeling (communicatielijn) tussen de chatbot en het dashboard te realiseren.

Requirement	**Status**
Must have	
Beheren van logboek	Ingepland
Agenda (afspraken)	Afgerond
Lijsten (taken)	Ingepland
Reactieve elementen (realtime dataverwerking)	In ontwikkeling*
Communicatie met virtuele assistent (vraag en antwoord)	Ingepland
Should have	
Persoonlijke omgeving	Ingepland
Notificaties	Ingepland
ICE-overzicht voor spoedgevallen	Ingepland

Tabel 7: Huidige status prototype na sprint 2 (zie hoofstuk 4 voor de beschreven requirements)

10. Communicatie met de chatbot - Sprint 3

Voor de derde sprint ga ik de laatste functionaliteiten van 'reactieve elementen' oppakken, namelijk het detecteren van wijzigingen die rechtstreeks in de chatbot gebeuren (zie hoofdstuk 4 voor alle requirements). In hoofdstuk 5 heb ik voor de deelvraag 'Hoe kan de communicatie tussen het dashboard en chatbot gerealiseerd worden?' onderzoek gedaan. In deze sprint wil ik de bevindingen gaan realiseren om te kijken of het daadwerkelijk mogelijk is om het dashboard als een nieuw kanaal toe te voegen aan de chatbot. Om mijn achterstand volledig in te halen, wil ik voor deze sprint 24 story points inplannen. Dat zijn twee punten meer dan vorige Sprint. In Tabel 8 heb ik de User Stories opgenomen die ik samen met de product owner heb opgesteld voor de requirement 'reactieve elementen' (zie hoofdstuk 4 voor een overzicht van alle requirements).

ID	User Story	Points
33163	Als ontwikkelaar wil ik een architectuur opstellen waarin het mogelijk is om nieuwe wijzigingen in de chatbot-omgeving door te sturen naar het dashboard-omgeving, zodat het mogelijk is beide afgeschermde omgevingen toch met elkaar te laten communiceren	8
34028	Als ontwikkelaar wil ik dat het dashboard een eigen kanaal wordt van de chatbot, zodat het mogelijk is om met de chatbot te communiceren	8
34012	Als mantelzorger wil ik vanuit het dashboard rechtstreeks kunnen communiceren met de chatbot, zodat ik een vraag en antwoord interactie (via een chat interface) kan hebben zonder in het dashboard te hoeven zoeken naar een antwoord	8

Tabel 8: User Stories voor sprint 3 (zie Bijlage 5 voor acceptatiecriteria)

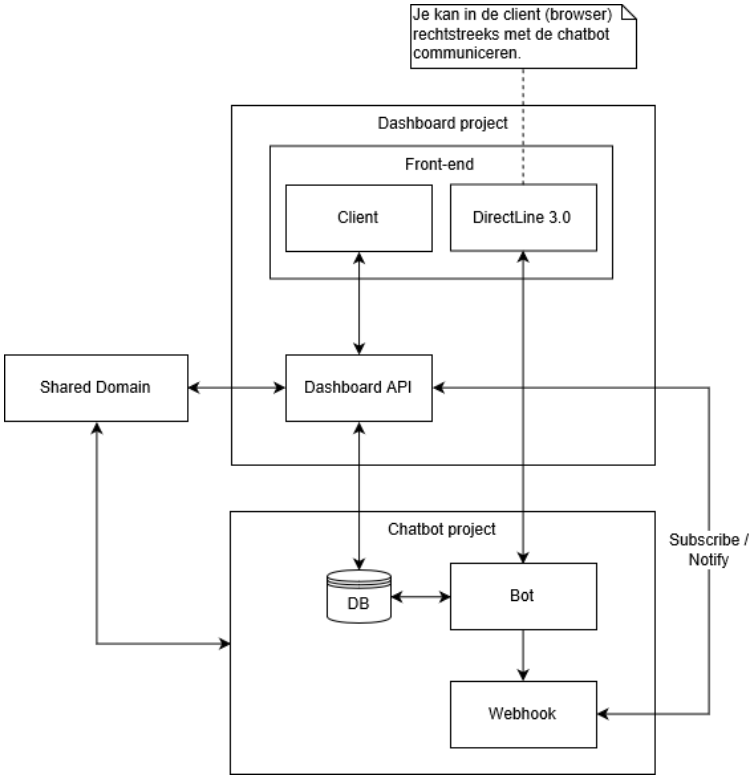
10.1 Webhooks

Het is na Sprint 2 mogelijk om aangepaste afspraken direct te kunnen zien in het dashboard. Door het toevoegen van WebSockets kunnen wijzigingen realtime gedeeld worden tussen alle actieve gebruikers van het dashboard. De wijzigingen worden nu alleen binnen het dashboard gedetecteerd. Wijzigingen van buitenaf, dus via de chatbot, worden nu niet gedetecteerd door het dashboard. Het dashboard is pas 'volledig' reactief wanneer de chatbot een signaal kan sturen wanneer er wijzigingen heeft plaatsgevonden.

Wat is een webhook?
Een webhook is een API-concept. Door het inzetten van webhooks is het mogelijk om van een bron applicatie nieuwe data te sturen naar een doelapplicatie. Het voordeel hiervan is dat er weinig tot geen resources nodig zijn. Het is relatief eenvoudig om een webhook te realiseren. Een doel applicatie kan zich aanmelden op een onderwerp en zodra er wijzigingen plaatsvinden in de bron applicatie, dan worden alle 'abonnees' op de hoogte gebracht (i-Reserve, sd).

Het verschil tussen een API en webhook kan het best beschreven worden met een voorbeeld: Een API kan gezien worden als een tekstbericht dat naar een vriend wordt gestuurd om meer informatie te krijgen over een evenement. De persoon in kwestie stelt een vraag en zijn vriend geeft vervolgens antwoord. Met een webhook wordt er aan een vriend gevraagd of hij een update wilt sturen zodra er nieuwe evenementen worden georganiseerd. De persoon in kwestie heeft zich aangemeld en zijn vriend blijft hem nieuwe evenementen doorsturen in de toekomst (Jin, 2017).

Architectuur
In Figuur 28 is de nieuwe situatie van de virtuele assistent (het dashboard en chatbot omgeving) te zien. De architectuur heb ik besproken en goed laten keuren door een architect van TrueLime. In de chatbot wordt de webhook geïntegreerd, waar het dashboard zich op kan abonneren. Als een mantelzorger een afspraak maakt via de chatbot, dan gaat de webhook kijken naar alle actieve abonnees en stuurt vervolgens de nieuwe afspraak naar hun door. De nieuwe afspraak wordt in dit geval via WebSockets, zoals beschreven in hoofdstuk 9.3, realtime doorgestuurd naar de actieve clients (User Story 33163, Tabel 8).



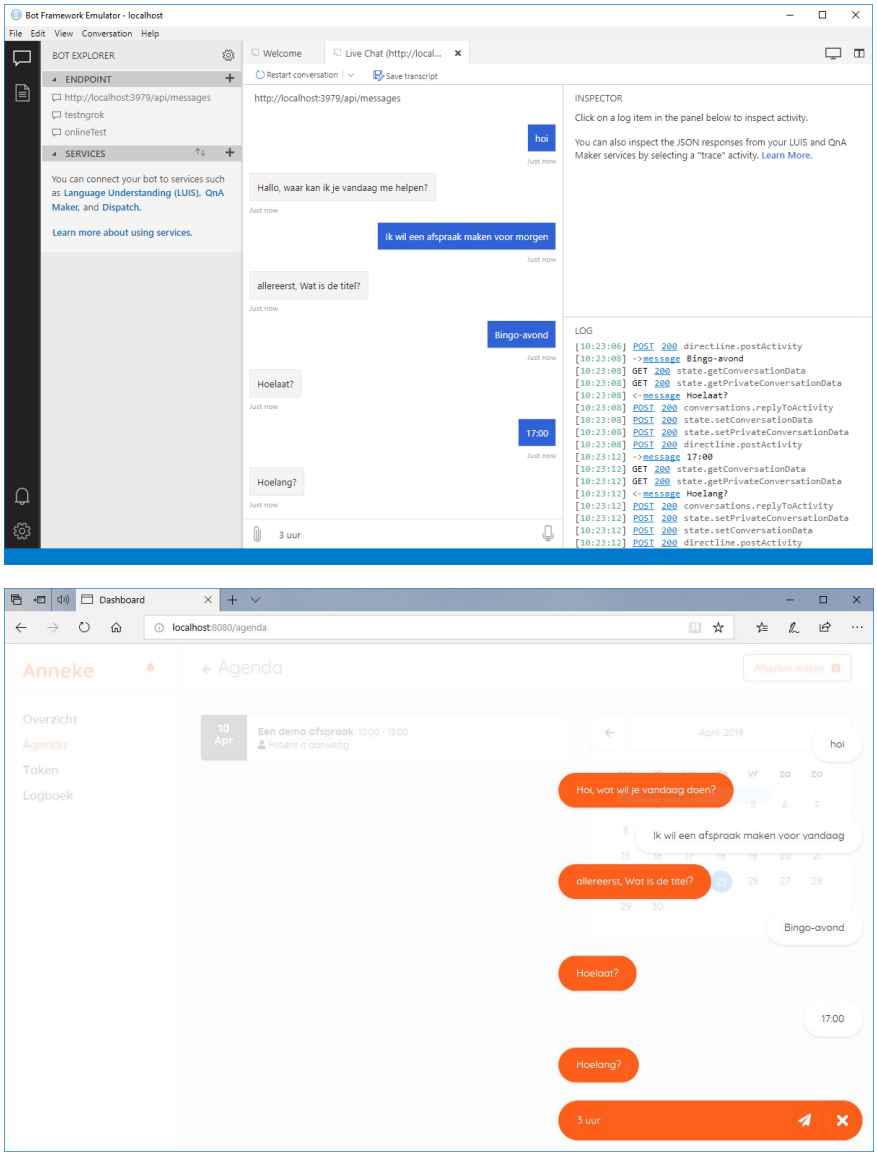
Figuur 28: Architectuur dashboard

10.2 Communicatie met de chatbot

Om de koppeling tussen de chatbot en het dashboard te realiseren moet het dashboard, zoals in hoofdstuk 8.2 is beschreven, een nieuw kanaal worden van de chatbot. In Vue heb ik een component gemaakt dat fungeert als een Event Bus[7], die door heel mijn applicatie beschikbaar is. De Event Bus bevat de benodigde configuraties van de DirectLine API en heeft een subscribe en publish methode. Met de methodes is het mogelijk om berichten te ontvangen en te versturen (User Story 34028, Tabel 8). Uiteindelijk heb ik in mijn Chat component een connectie gemaakt met de chatbot om vervolgens berichten te kunnen uitwisselen (User Story 34012, Tabel 8). In Figuur 29 is te zien dat het nu

[7] Een event bus is een mechanisme waarin het mogelijk is om verschillende componenten met elkaar te laten communiceren, zonder dat ze van elkaar af weten.

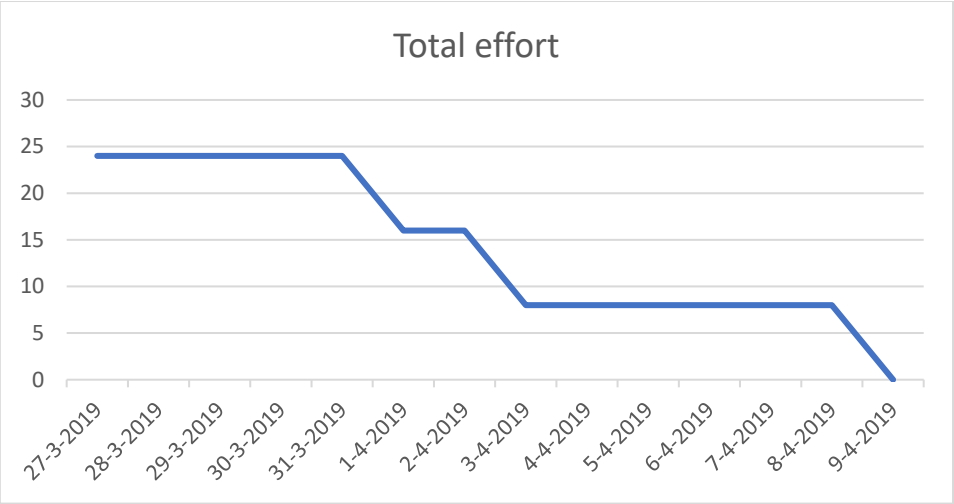
mogelijk is om een gesprek te voeren op het dashboard. De deelvraag ‘Hoe kan de communicatie tussen het dashboard en chatbot gerealiseerd worden?’ was in hoofdstuk 5 onderzocht en inmiddels gerealiseerd.



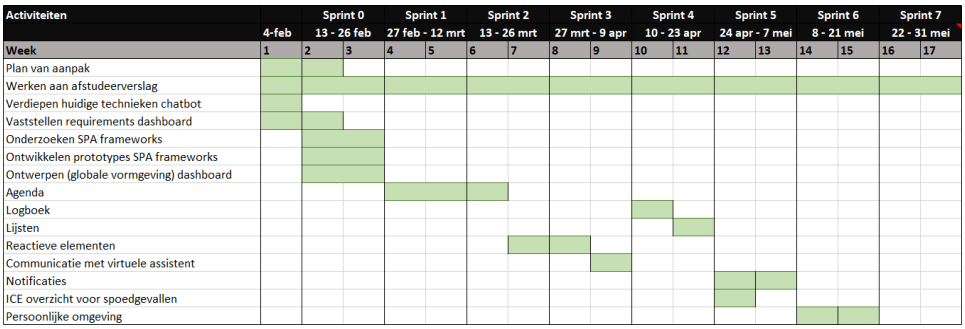
Figuur 29: Voorbeeld dashboard interface

10.3 Resultaat

Voor de derde iteratie van het project waren er 24 story points ingepland en gerealiseerd (zie Figuur 30). De koppeling tussen chatbot en dashboard is deze sprint afgerond. Hierdoor is Sprint 7 weer vrijgekomen en is mijn achterstand ingehaald, zoals op de planning (Figuur 31) te zien is.



Figuur 30: Burndown Chart sprint 3



Figuur 31: Nieuwe planning

Voor deze sprint is er is een nieuwe ‘must have’ requirement afgerond voor het prototype, namelijk ‘Communicatie met virtuele assistent’ (zie Tabel 9).

Requirement	**Status**
Must have	
Beheren van logboek	Ingepland
Agenda (afspraken)	*Afgerond*
Lijsten (taken)	Ingepland
Reactieve elementen (realtime dataverwerking)	*In ontwikkeling**
Communicatie met virtuele assistent (vraag en antwoord)	*Afgerond*
Should have	
Persoonlijke omgeving	Ingepland
Notificaties	Ingepland
ICE-overzicht voor spoedgevallen	Ingepland

Tabel 9: Huidige status prototype na sprint 3 (zie hoofstuk 4 voor de beschreven requirements)

Volgende Sprint ga ik me focussen op het afronden van ‘reactieve elementen’, ‘lijsten’ en ‘beheren van logboek’.

11. Lijsten en logboek – Sprint 4

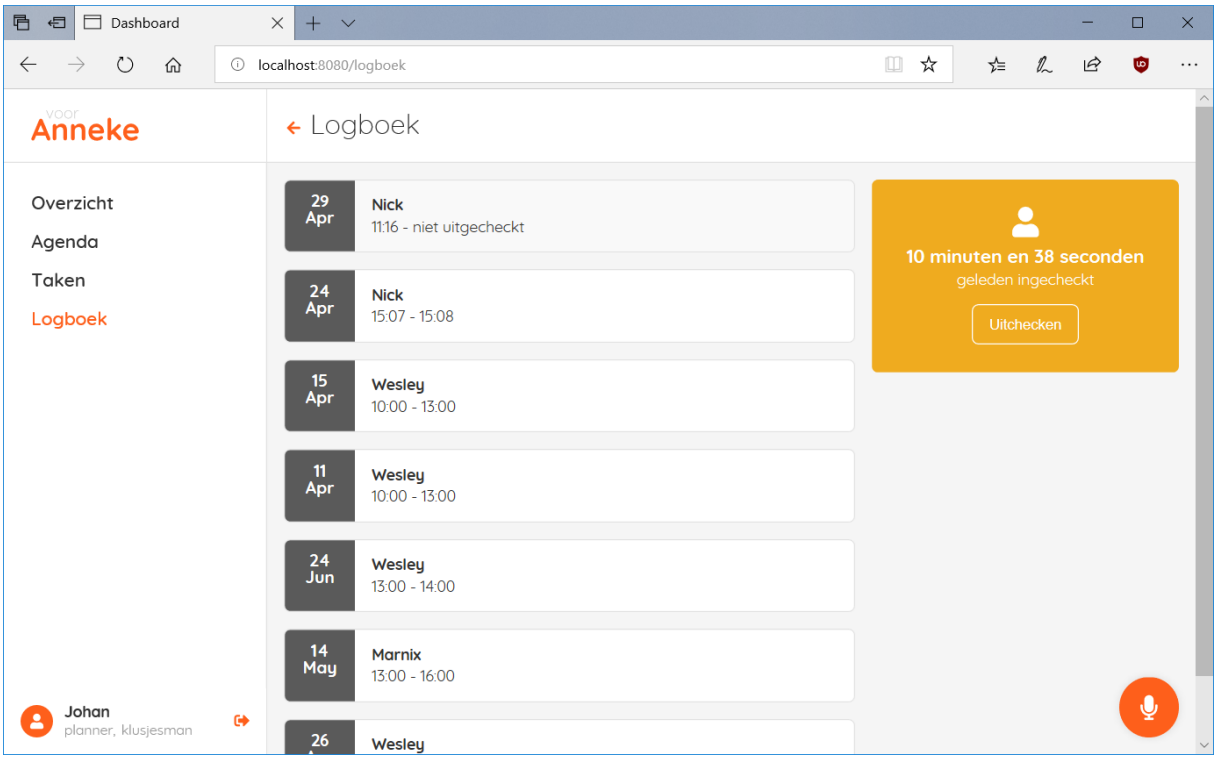
Voor de vierde Sprint ga ik aan de slag met de laatste twee skills van de chatbot, namelijk lijsten (taken) en het beheren van een logboek. Voor deze sprint heb ik 21 story points ingepland. Hierdoor heb ik meer ruimte om het project te documenteren, omdat ik de vorige sprint vooral had gefocust op het inhalen van mijn achterstand. In Tabel 10 heb ik de User Stories opgenomen die ik samen met de product owner heb opgesteld voor de requirements 'lijsten' en 'logboek' (zie hoofdstuk 4 voor een overzicht van alle requirements).

ID	**User Story**	**Points**
35083	Als mantelzorger wil ik een overzicht kunnen zien van alle logberichten die zijn aangemaakt in het verleden, zodat ik op de hoogte ben van de situatie rondom de cliënt	3
35082	Als mantelzorger wil ik mezelf kunnen uitchecken, zodat de chatbot weet wanneer mijn zorgverlenging is beëindigd	2
35081	Als mantelzorger wil ik mezelf kunnen inchecken, zodat de chatbot weet wanneer mijn zorgverlening van start is gegaan is	3
35080	Als mantelzorger wil ik alle (dus ook afgeronde) taken binnen een lijst kunnen zien, zodat ik weet wat voor werkzaamheden er in het verleden heeft plaatsgevonden	3
35079	Als mantelzorger wil ik taken kunnen verwijderen, zodat foutieve taken niet meer in de lijsten staan	1
35077	Als mantelzorger wil ik taken kunnen afvinken, zodat ik kan bijhouden dat de taak is afgerond	1
35076	Als mantelzorger wil ik een lijst met taken bijhouden, zodat ik niet vergeet wat voor werkzaamheden ik moet verrichten voor mijn cliënt	8

Tabel 10: User Stories voor sprint 4 (zie Bijlage 6 voor de acceptatiecriteria)

11.1 Logboek

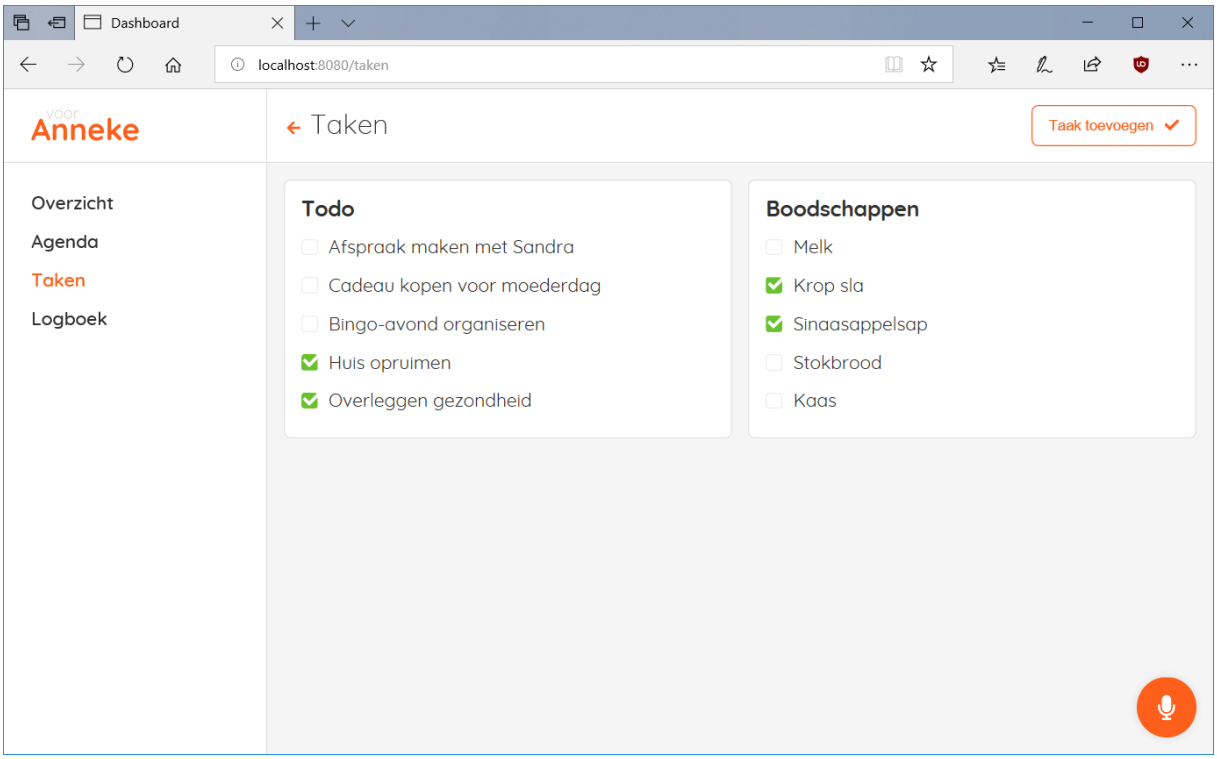
Een mantelzorger kan zich in- en uitchecken in het dashboard wanneer hij of zij de cliënt aan het helpen is (User Story 35081 & 35082, Tabel 10). Dit is vooral handig voor cliënten die lijden aan geheugenverlies. Hierdoor is er een archief beschikbaar van alle incheckbewijzen van de mantelzorgers die de afgelopen tijd bij de cliënt waren (User Story 35083, Tabel 10). In het dashboard wordt een startdatum toegevoegd. De widget laat vervolgens, zoals aangegeven in Figuur 32, zien hoe lang geleden de mantelzorger ingecheckt is.



Figuur 32: Logboek in het dashboard

11.2 Lijsten

In de chatbot is het mogelijk om taken toe te voegen, dit is handig voor wanneer een mantelzorger of de cliënt zelf een geheugensteuntje nodig heeft. Nu is het mogelijk om taken te beheren in het dashboard (User Story 35076, 35077, 35079 & 35080, Tabel 10). Het eindresultaat is in Figuur 33 te zien. Voorheen werd in de chatbot taken verwijderd als ze opgepakt waren. In de nieuwe situatie is met de product owner besproken dat een taak ook afgevinkt kan zijn, zonder dat het item verwijderd moet worden. Hiervoor heb ik de modellen moeten veranderen in het gedeelde domein en werkt het functioneel bij de chatbot en dashboard (zie hoofdstuk 8.3 voor meer informatie over het gedeelde domein).



Figuur 33: Taken in het dashboard

Voor het beheren van de lijsten moet, net als bij agenda, wijzigingen realtime uitgewisseld worden. Voor het reactief maken van de nieuwe elementen heb ik structurele wijzigingen in het project gemaakt. In het volgende hoofdstuk wordt er meer over de wijzigingen toegelicht.

11.3 Hub builder

Voor het beheren van taken is er afgesproken met de product owner om de data, net zoals bij agenda, realtime uit te wisselen (zie hoofdstuk 9.3 voor meer informatie over hubs). Om dit te realiseren moet ik een nieuwe hub aanmaken en vervolgens opnieuw inrichten aan het front-end. Het configureren van een hub bevat veel boilerplate code[8]. Mocht er later meer skills voor de chatbot bijkomen dat direct afgehandeld moet worden, dan wordt de logica redundant en onoverzichtelijk.

Back-end

Voor de back-end is het mogelijk om met SignalR, een framework voor het realiseren van realtime data-afhandeling, een hub te creëren. Elke hub kan zijn eigen definitie bevatten. Dit is overigens niet gunstig voor het dashboard, omdat er alleen nieuwe, aangepaste en verwijderde data uitgewisseld moeten worden. Verder is het belangrijk dat de front-end weet wat de definitie is van een hub. Dit probleem heb ik opgelost door twee interfaces met de drie acties (toevoegen, bewerken en verwijderen) toe te voegen die nodig zijn om data te kunnen ontvangen en versturen. Voor elke nieuwe hub hoef ik alleen twee interface te implementeren en is er verder geen wijzigingen nodig aan

[8] Boilerplate code is code die regelmatig wordt gebruikt maar minimale wijzigingen nodig heeft op de daadwerkelijke inhoud (Techopedia, sd).

het front-end. In de code snippet hieronder heb ik met relatief weinig code een afspraak hub aangemaakt die de data doorstuurt naar alle actieve clients.

```
public class AppointmentHub : Hub<IHubClient>, IHub {
  public async Task Delete(string id) {
    await Clients.Others.Deleted(id);
  }
  public async Task Insert(object data) {
    await Clients.Others.Inserted(data);
  }
  public async Task Update(object data) {
    await Clients.Others.Updated(data);
  }
}
```

Front-end

Voor het front-end heb ik een builder gemaakt, waar je met twee parameters een nieuwe hub kan aanmaken die vervolgens de juiste functionaliteiten en configuratie bevat; Waaronder het toevoegen, bewerken en verwijderen van items. Een bijkomend voordeel is dat structurele wijzigingen in de hub op één centrale plek aangepast kan worden. Het aanmaken van een nieuwe hub is mogelijk met de volgende snippet:

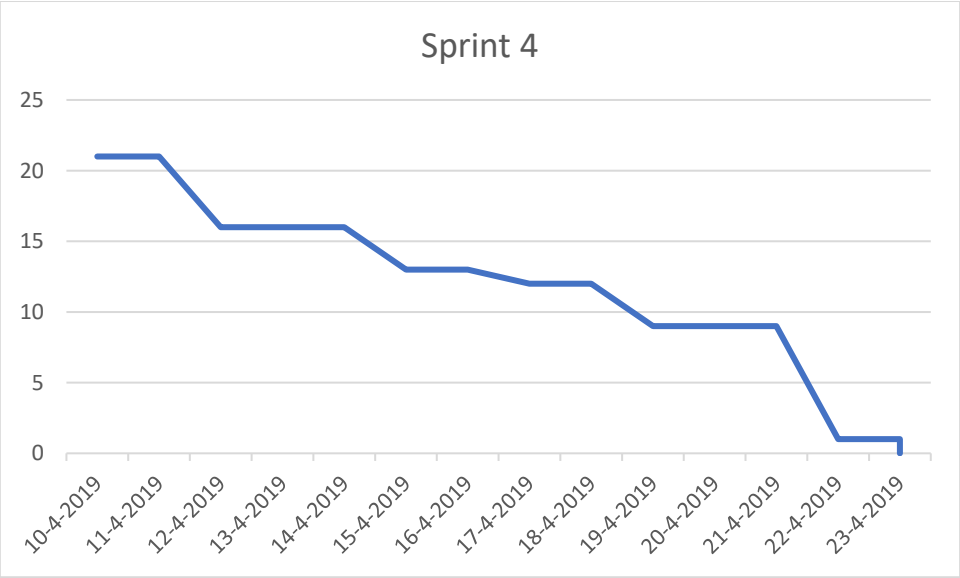
```
import build from './hubBuilder';
export default build('appointmentHub', 'https://url-to-hub');
```

In de snippet is te zien hoe er met twee regels code een afspraken hub is gemaakt met de benodigde methodes. Verder zorgt de builder ervoor dat het object wordt gekoppeld aan Vue en is het hierdoor mogelijk om in de componenten zelf te abonneren voor de realtime wijzigingen. In het geval van de snippet hieronder krijgt de component alle nieuwe toegevoegde afspraken (insert) binnen.

```
this.$appointmentHub.$on('insert', function(appointment) {
  // Alle nieuwe afspraken worden in deze functie aangeroepen.
});
```

11.4 Resultaat

Deze sprint heb ik de laatste twee skills van de chatbot afgerond. Het was voor mij een verademing om te zien dat ik binnen een rappe tijd twee skills heb kunnen afronden, zoals in de Burndown Chart (Figuur 34) te zien is.



Figuur 34: Burndown Chart van sprint 4

Zoals aangegeven in Tabel 11 zijn alle 'must have' requirements van het dashboard afgerond. Deze sprint heb ik 'beheren van logboek', 'lijsten (taken)' en 'reactieve elementen' kunnen afsluiten.

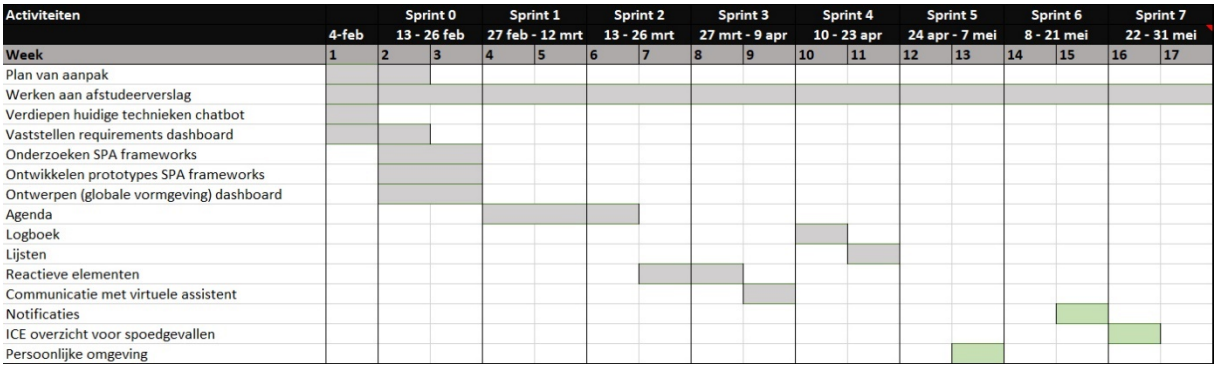
Requirement	**Status**
Must have	
Beheren van logboek	*Afgerond*
Agenda (afspraken)	*Afgerond*
Lijsten (taken)	*Afgerond*
Reactieve elementen (realtime dataverwerking)	*Afgerond*
Communicatie met virtuele assistent (vraag en antwoord)	*Afgerond*
Should have	
Persoonlijke omgeving	Ingepland
Notificaties	Ingepland
ICE-overzicht voor spoedgevallen	Ingepland

Tabel 11: Prototype na sprint 4 (zie hoofstuk 4 voor de beschreven requirements)

In de volgende sprint ga ik aan de slag met de persoonlijke omgeving van het dashboard.

12. Persoonlijke omgeving & interview mantelzorger – Sprint 5

Voor de vijfde sprint hebben de product owner en ik de planning aangepast. Initieel was het idee om in deze sprint notificaties te realiseren. Het leek ons belangrijker om de persoonlijke omgeving eerder te ontwikkelen, omdat het meer waarde oplevert dan het inbouwen van notificaties. Een persoonlijke omgeving zorgt er namelijk voor dat het mogelijk is om te schakelen tussen verschillende mantelzorgers. De mantelzorger kan zijn eigen inhoud beheren en hierdoor gaat de samenwerking op het dashboard pas echt tot leven komen (zie Figuur 35 voor de aangepaste planning).



Figuur 35: Aangepaste planning (sprint 5)

Zoals in hoofdstuk 11.4 is beschreven zijn de 'must have' requirements afgerond. Hierdoor kunnen we het dashboard, na het realiseren van de persoonlijke omgeving, gaan testen door de doelgroep. Ik zou graag een mantelzorger willen interviewen om zo de behoefte te achterhalen en ook om de applicatie te testen qua gebruiksvriendelijkheid. De bevindingen uit de test zou eventueel een nieuwe richting kunnen geven aan de laatste sprint.

In deze sprint staat ook het tussentijdse assessment voor het afstuderen ingepland. Om die reden heb ik samen met de product owner besloten om 12 Story Points in te plannen. Met de resterende tijd kan ik het interview voorbereiden, uitvoeren en aan mijn afstudeerverslag werken. In Tabel 12 heb ik de User Stories opgenomen die ik samen met de product owner heb opgesteld voor de requirement 'persoonlijke omgeving' (zie hoofdstuk 4 voor een overzicht van alle requirements).

ID	User Story	Points
35972	Als gebruiker wil ik de huidige actieve gebruiker en rol in het dashboard zien, zodat ik weet wie er op dit moment actief is in het dashboard	1
35971	Als gebruiker wil ik vanuit een lijst van gebruikers mijn eigen account selecteren, zodat ik kan werken in mijn eigen omgeving	4
35973	Als gebruiker wil ik kunnen uitloggen, zodat iemand anders in zijn of haar eigen omgeving kan	1
35974	Als gebruiker wil ik dat tijdens het beheren van taken mijn naam aan de taak wordt gekoppeld, zodat iedereen kan zien dat ik de eigenaar ben van het item	2
35975	Als gebruiker wil ik dat tijdens het beheren van afspraken mijn naam aan de afspraak wordt gekoppeld, zodat iedereen kan zien dat ik de eigenaar ben van het item	2

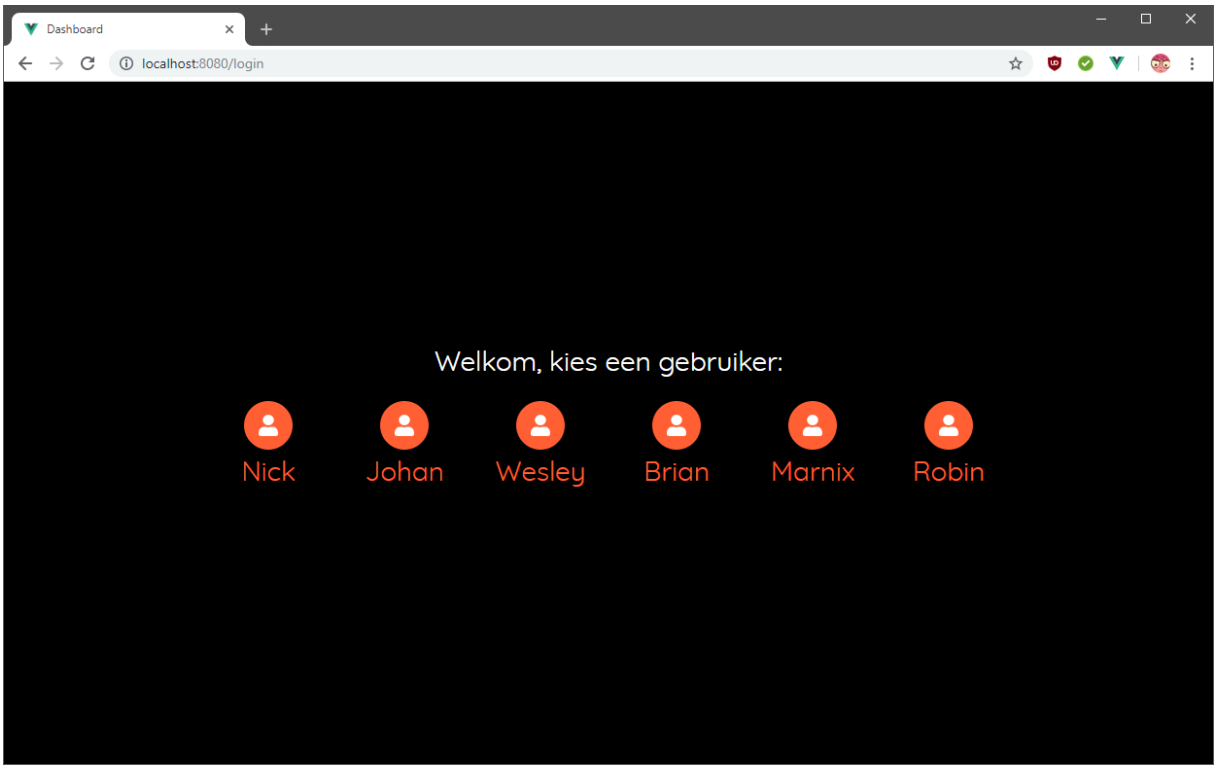
35976	Als gebruiker wil ik dat tijdens het inchecken en uitchecken mijn naam aan het log item wordt gekoppeld, zodat iedereen kan zien dat ik de eigenaar ben van het items	2

Tabel 12: User Stories voor sprint 5 (zie Bijlage 7 voor de acceptatiecriteria)

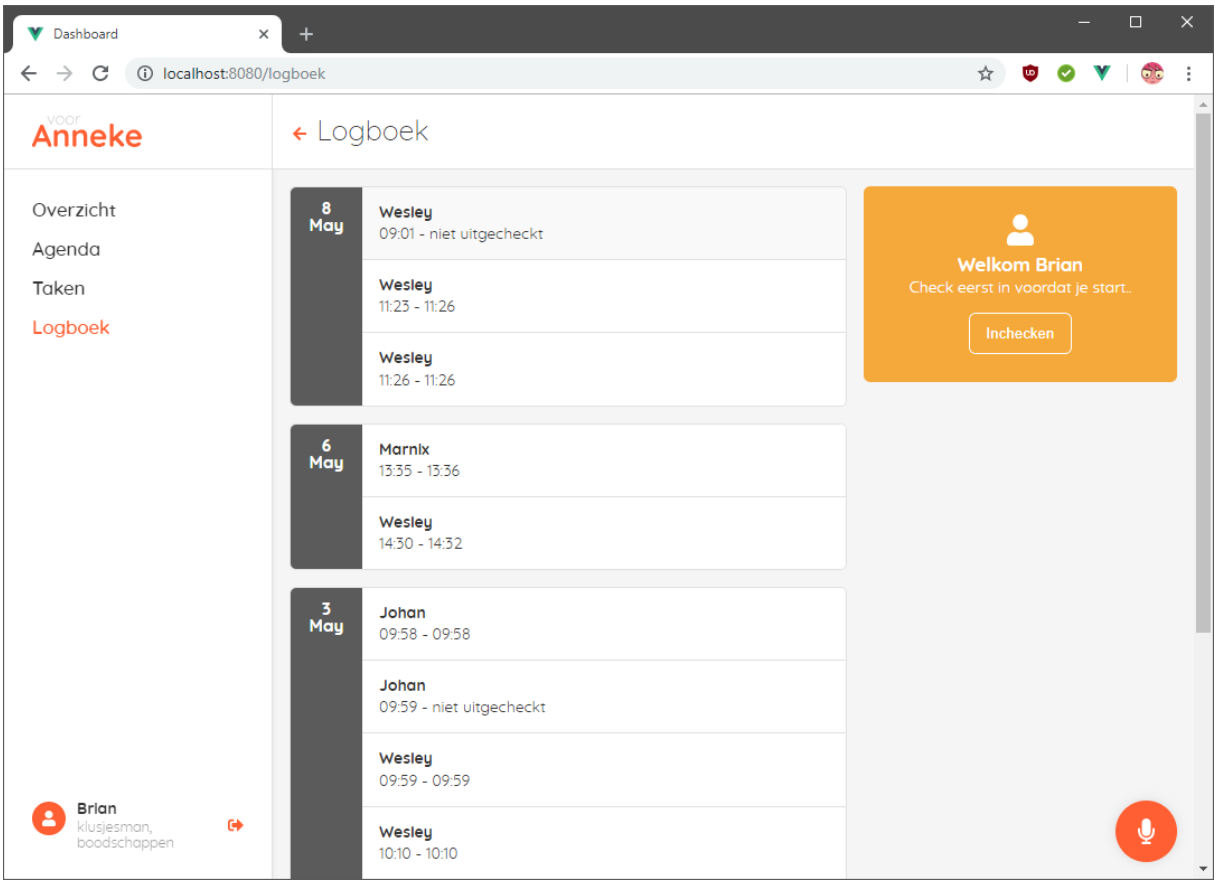
In het volgende hoofdstuk vertel ik meer over de 'persoonlijke omgeving' requirement.

12.1 Persoonlijke omgeving

Mantelzorgers en de cliënt kunnen inloggen op hun eigen omgeving. Aangezien er meerdere gebruikers zich kunnen aanmelden op het dashboard moet het beheren van taken, logboeken en afspraken terug te herleiden zijn aan de juiste gebruiker (User Story 35974 & 35975, Tabel 12). Hiervoor heb ik een aanmeldscherm gemaakt, waar alle gebruikers van het dashboard worden getoond. De mantelzorger kan zijn profiel selecteren en wordt vervolgens doorgestuurd naar de startpagina (User Story 35971 & 35973, Tabel 12). Verder is het noodzakelijk dat een mantelzorger zichzelf kan inchecken (logboek) wanneer hij of zij bij de cliënt is (User Story 35976, Tabel 12). In Figuur 36 en Figuur 37 wordt het resultaat van het aanmeldproces weergegeven.



Figuur 36: Aanmeldscherm dashboard



Figuur 37: Een mantelzorger (Brian) is ingelogd en kan inchecken

12.2 Interview

Nu de 'must have' requirements zijn afgerond is het dashboard klaar om getest te worden. In deze sprint is het me gelukt om een interview te regelen met een mantelzorger, mevrouw R. Een bijkomend voordeel is dat mevrouw R naast mantelzorger ook werkzaam is als verpleegkundige. Hiermee heeft zij ervaring op het gebied van dossiers bijhouden van patiënten en communiceren met meerdere verzorgers in een ziekenhuis.

Voorbereiding

Het interview is opgedeeld in drie onderdelen. Het eerste deel is het interview zelf, waar ik vragen stel over de huidige situatie en waar de behoefte is om de communicatie te verbeteren. In het tweede deel vertel ik kort over het dashboard en wat voor problemen het moet gaan oplossen. Hiermee hoop ik de mantelzorger een juist beeld te geven voordat zij het dashboard gaat testen. Tijdens het testen zal ik de mantelzorger gaan observeren om te kijken of het dashboard makkelijk in gebruik te nemen is. Dit kan ik achterhalen door te vragen of het dashboard op de manier reageert zoals zij verwacht. Na het testen ga ik bij het laatste deel van het interview vragen naar verbeterpunten en eventueel wat voor functionaliteiten er ontbrak in het dashboard. In Bijlage 10 is de structuur van de drie onderdelen verder toegelicht met de bijbehorende vragen.

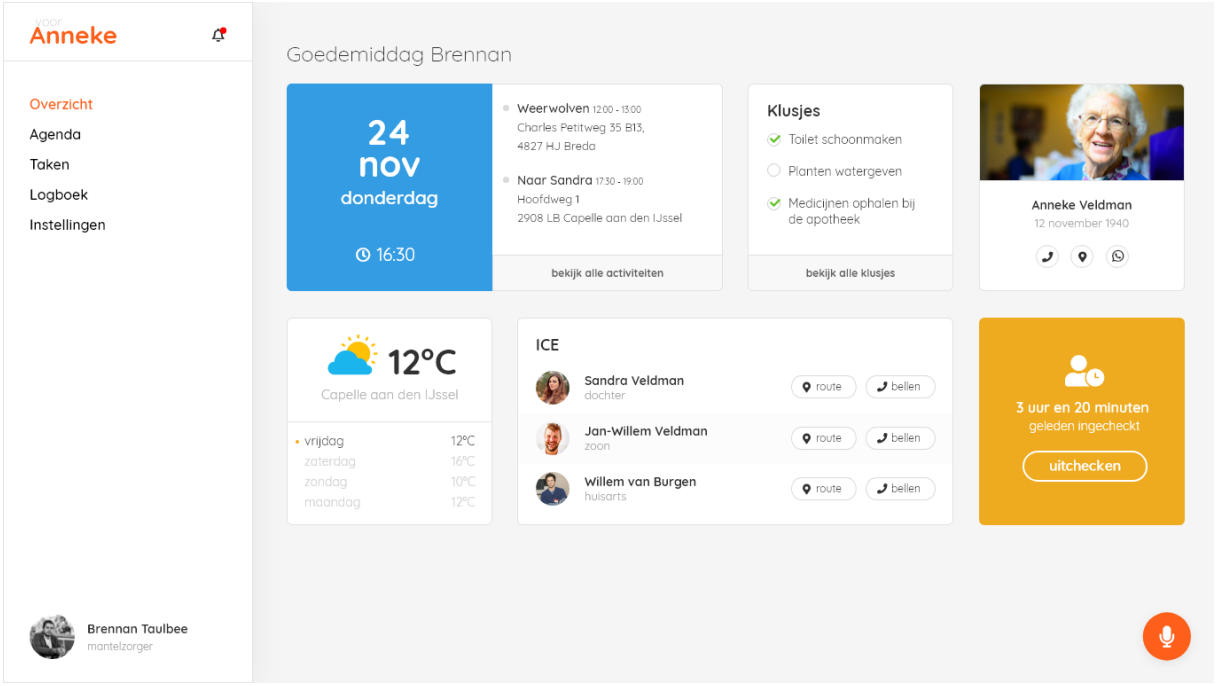
Resultaat

Het interview heeft nieuwe inzichten gegeven over het dashboard. Voor de huidige functionaliteiten heb ik in Tabel 13 de verbeterpunten van mevrouw R samengevat. Een van de belangrijkste functionaliteiten dat mevrouw R vond dat ontbrak in het dashboard was een startpagina waar ze direct alle informatie kan inzien en beheren. Logboek, agenda en logboek zou dan als een widget

toegevoegd moeten worden. Hier was in het ontwerpproces al rekening mee gehouden (zie Figuur 38), maar nooit ingepland om te realiseren. Het realiseren van de widgets wil ik dan ook verder bespreken met de product owner tijdens de Sprint Demo.

Requirements		
Afspraken (agenda)	**Logboek**	**Taken**
• Afspraken kunnen inplannen via de kalender • Categorieën toevoegen (het liefst met kleuren) voor beter overzicht en beheer • Herhaalpatronen toevoegen aan een afspraak (bijvoorbeeld wekelijks, maandelijks en jaarlijks) • De meest recente afspraken tonen op het dashboard voor snel beheer	• Samenvatting kunnen toevoegen aan een logbericht (na het uitchecken), zodat andere mantelzorgers ook weet wat de mantelzorger heeft gedaan • Incheck systeem toevoegen op het dashboard	• Deadline toevoegen aan een taak • Taken zijn nu standaard publiekelijk, graag standaard afschermen voor andere gebruikers • Optie toevoegen om en taak publiekelijk te maken • Taken kunnen koppelen aan andere mantelzorgers • De meest recente taken tonen op het dashboard

Tabel 13: Verbeterpunten huidige functionaliteiten



Figuur 38: Startpagina voor een mantelzorger

Naast bovengenoemde verbeterpunten kwam mevrouw R ook met nieuwe ideeën om het dashboard uit te breiden:

- Omgeving voor vragen over de cliënt;
- Omgeving voor alarmerende zaken (bijvoorbeeld dat de cliënt regelmatig is gevallen en dat de andere mantelzorgers daar direct van af moeten weten, zodat iedereen alert is als ze zorg verlenen);
- Medicijnoverzicht (widget) op het dashboard, zodat de mantelzorger weet wat voor medicatie de cliënt moet innemen.

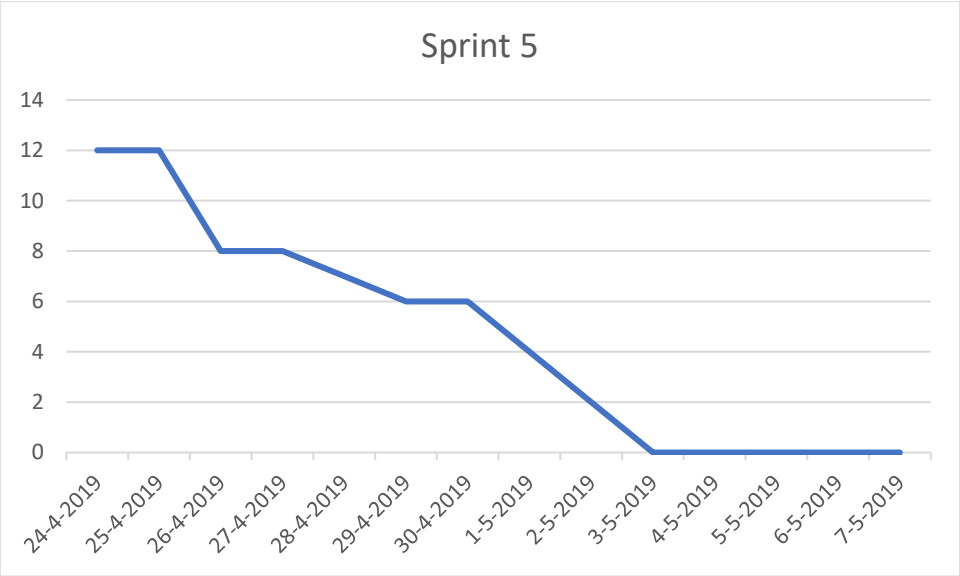
Haar input heb ik verwerkt in een presentatie en ben ik van plan om te delen met de product owner tijdens de Sprint Demo. In het volgende hoofdstuk vertel ik meer over het resultaat van deze sprint.

12.3 Resultaat

Voor deze sprint heb ik de requirement 'persoonlijke omgeving' afgerond (zie Tabel 14). Er waren 12 Story Points ingepland en zoals wordt weergegeven in de Burndown Chart (Figuur 39) zijn alle taken in een korte tijd gerealiseerd. De rest van de sprint heb ik gewerkt aan mijn afstudeerverslag en heb ik een interview gehad met een mantelzorger. Het interview heeft nieuwe inzichten gegeven over de gebruiksvriendelijkheid en wensen over het dashboard (zie hoofdstuk 12.2). In een later stadium, wat na mijn afstuderen zal plaatsvinden, moet het dashboard door meer gebruikers getest worden om het prototype verder te verbeteren.

Requirement	**Status**
Must have	
Beheren van logboek	*Afgerond*
Agenda (afspraken)	*Afgerond*
Lijsten (taken)	*Afgerond*
Reactieve elementen (realtime dataverwerking)	*Afgerond*
Communicatie met virtuele assistent (vraag en antwoord)	*Afgerond*
Should have	
Persoonlijke omgeving	*Afgerond*
Notificaties	Ingepland
ICE-overzicht voor spoedgevallen	Ingepland

Tabel 14: Overzicht van sprint 5 (zie hoofstuk 4 voor de beschreven requirements)

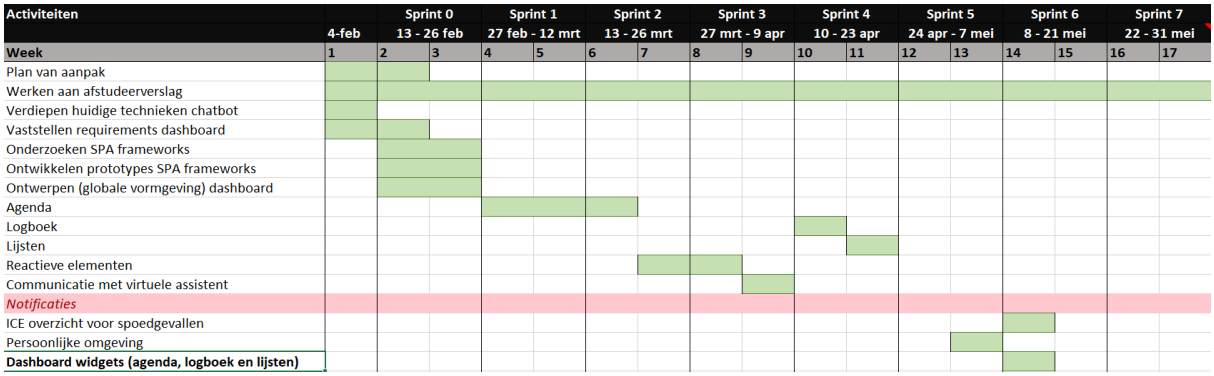


Figuur 39: Burndown Chart van sprint 5

Met de bevindingen van het interview ben ik de Sprint Demo en Sprint Planning sessie ingegaan. In het volgende hoofdstuk vertel ik meer over hoe de planning eruit gaat zien voor de laatste sprint en of het interview veranderingen heeft gebracht aan de geprioriteerde requirements.

13. Startpagina – Sprint 6

De bevindingen uit het interview met de mantelzorger (hoofdstuk 11.2) zijn meegenomen naar de Sprint Planning sessie. Samen met de product owner hebben we naar de verbeterpunten gekeken, zoals in Tabel 13 staat beschreven. Ik had tijdens het gesprek aangegeven dat het ontwikkelen van een startpagina het belangrijkste verbeterpunt is uit het interview. De mantelzorger wilt namelijk een centrale plek waar ze haar taken, afspraken en logboek kan beheren. Hierdoor zal het prototype ook gebruiksvriendelijker worden, omdat de mantelzorger de meeste handelingen vanaf de startpagina kan uitvoeren. Dit heeft consequenties voor de planning. Notificaties stond oorspronkelijk voor Sprint 6 ingepland, maar dit is nu komen te vervallen omdat de startpagina een hogere prioriteit heeft gekregen door de product owner. Sprint 7, zoals beschreven in het plan van aanpak (Bijlage 1), is een uitloop week waar ik voornamelijk bezig gaat zijn met het afronden van mijn afstudeerverslag. In de nieuwe planning (Figuur 40) is te zien hoe de laatste weken van het project eruitziet.



Figuur 40: Aangepaste planning sprint 6

In Tabel 15 heb ik de User Stories opgenomen die ik samen met de product owner heb opgesteld voor de startpagina. Er zijn deze sprint in totaal 14 story points ingepland. Dat zijn twee punten meer dan vorige sprint. Ik wil de requirement 'ICE-overzicht' deze sprint meenemen, omdat het ook een widget is voor de startpagina.

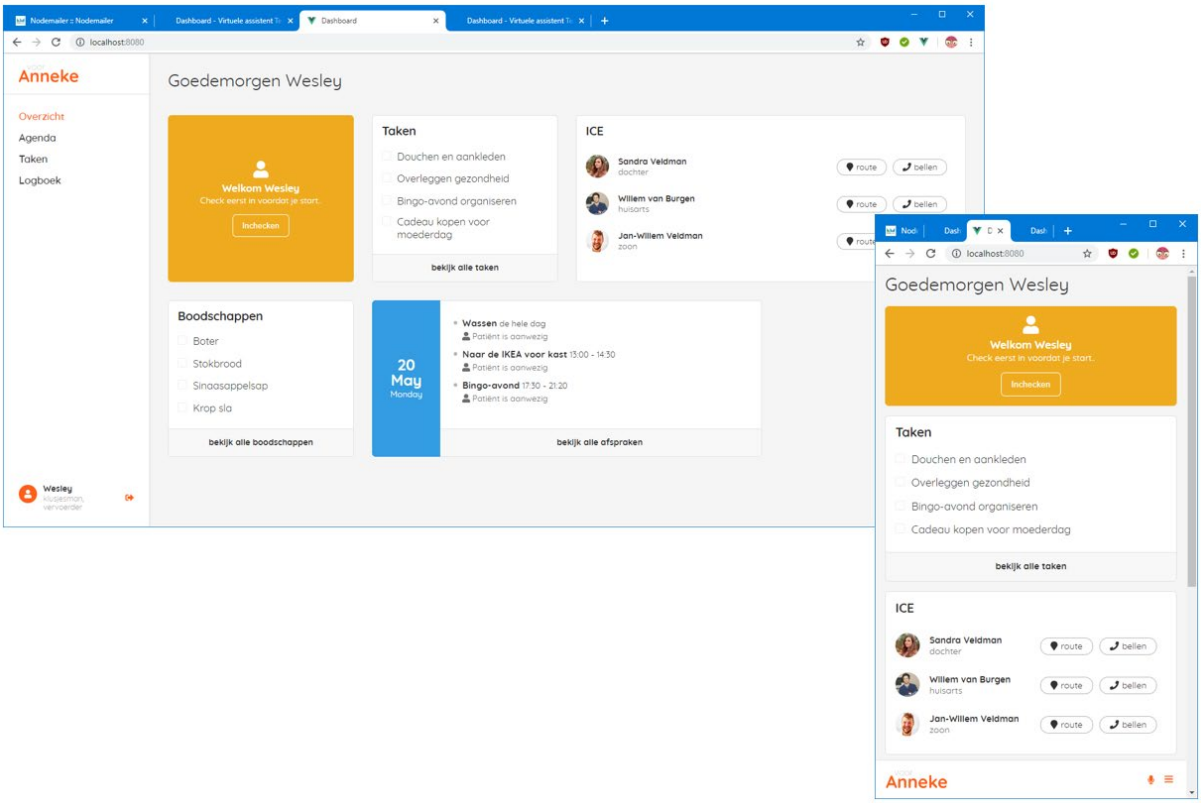
ID	User Story	Points
35083	Als mantelzorger wil ik de afspraken van vandaag kunnen inzien op mijn dashboard, zodat ik weet wat ik moet gaan uitvoeren	6
36936	Als mantelzorger wil ik op het dashboard direct kunnen inchecken, zodat ik het niet vergeet wanneer ik voor het eerst aanmeld op het dashboard	3
31722	Als mantelzorger wil ik op het dashboard de actuele taken kunnen inzien, zodat ik direct weet wat ik vandaag moet oppakken	3
36938	Als mantelzorger wil ik een ICE-overzicht zien, zodat als er wat misgaat met de cliënt dat ik iemand kan benaderen	2

Tabel 15: User Stories voor sprint 6 (zie Bijlage 6 voor de acceptatiecriteria)

Technische realisatie
Voor het realiseren van de widgets op de startpagina heb ik een standaard component ontwikkeld. Met deze component delen alle widgets (afspraken, logboek en taken) dezelfde structuur. Een voordeel is dat eventuele wijzigingen maar op één plek aangepast hoeft te worden.

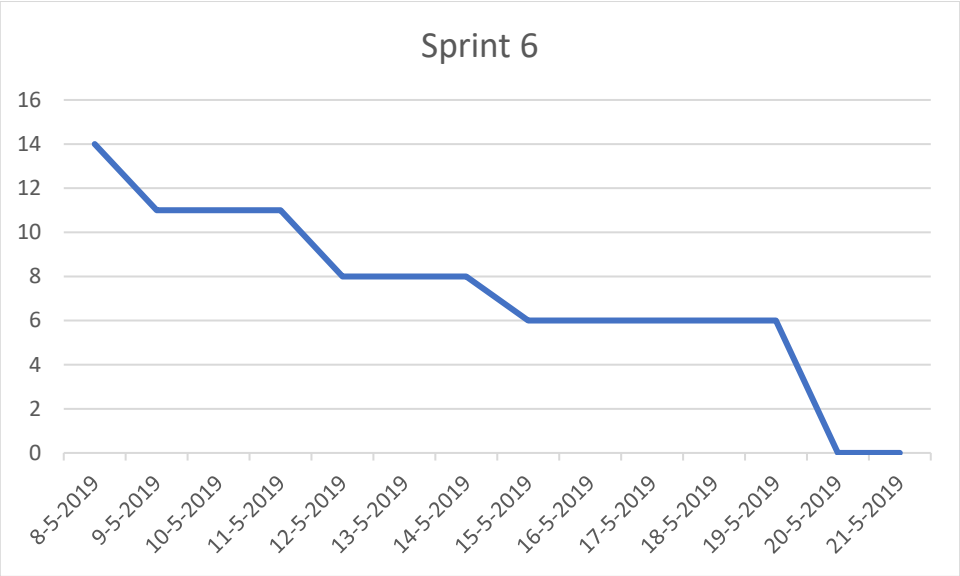
De startpagina bevat nu de volgende functionaliteiten:

- Alle afspraken voor de huidige dag worden weergegeven, zo kan de mantelzorger direct zien wat er op de planning staat;
- Taken kunnen op de startpagina afgevinkt worden;
- Boodschappen kunnen op de startpagina afgevinkt worden;
- Een mantelzorger kan na het inloggen direct inchecken;
- Er is een ICE-overzicht geplaatst voor spoedgevallen;
- Startpagina is volledig responsive (beschikbaar op mobiel, tablet en desktop).



Figuur 41: Startpagina met widgets

Resultaat
Het afronden van het dashboard ging voorspoedig. Uiteindelijk heb ik, zoals te zien is in Figuur 41, 14 punten kunnen realiseren deze sprint. In Tabel 16 is te zien dat alle ‘must have’ en ‘should have’ requirements zijn afgerond. De product owner is tevreden met het prototype en wilt, zoals eerder aangegeven in hoofdstuk 4, het product gaan inzetten voor sales pitches.



Figuur 42: Burndown Chart sprint 6

Requirement	**Status**
Must have	
Beheren van logboek	*Afgerond*
Agenda (afspraken)	*Afgerond*
Lijsten (taken)	*Afgerond*
Reactieve elementen (realtime dataverwerking)	*Afgerond*
Communicatie met virtuele assistent (vraag en antwoord)	*Afgerond*
Should have	
Persoonlijke omgeving	*Afgerond*
Notificaties	Geannuleerd
ICE-overzicht voor spoedgevallen	*Afgerond*
Startpagina met widgets (afspraken, taken en logboek)	*Afgerond*

Tabel 16: Overzicht requirements sprint 6

De laatste sprint is zojuist afgerond. In het volgende hoofdstuk evalueer ik het afstudeerproces.

14. Evaluatie

De laatste sprint is voorbij. Alle belangrijke functionaliteiten zijn gerealiseerd voor het dashboard. Beide projecten, chatbot en dashboard, werken naadloos samen om mantelzorgers en hun cliënten te kunnen helpen. Het ontwikkelde dashboard wordt nu al in sales pitches gebruikt als prototype. Naast het onderzoek en implementatie van het dashboard evalueer ik in dit hoofdstuk de opgeleverde producten en hoe het gehele proces is verlopen. Verder licht ik toe waarom ik denk dat ik voldoe aan de gekozen beroepstaken.

14.1 Productevaluatie

Tijdens het opstellen van mijn afstudeerplan had ik een aangegeven welke producten ik zou opleveren aan het eind van het project. Hieronder heb ik voor elk product een evaluatie geschreven.

Bevindingen front-end frameworks

Voordat het ontwikkelen van het dashboard begon heb ik een onderzoek gestart naar vier populaire Single-Page Application frameworks. In het onderzoek heb ik criteria opgesteld waar een framework aan moet voldoen. Met de vastgestelde criteria ging ik gefocust op zoek naar de voor- en nadelen van elk framework. Als ik kritisch terugkijk naar de uitvoering, dan zou ik de volgende keer een weging toevoegen aan de criteria. Hiermee kan ik naast de geschreven bevindingen ook met cijfers aantonen waarom een framework goed of slecht scoort.

Uit het onderzoek kwam Vue als beste kandidaat naar voren. Wel had Vue één valkuil, namelijk dat niemand in het bedrijf ervaring met het framework heeft. Mocht ik tegen problemen aanlopen, wat vooral framework specifiek was, dan was het lastig om daar ondersteuning voor te krijgen. Uiteindelijk heb ik toch voor Vue gekozen, omdat de resultaten uit het onderzoek positief zijn. Verder was ik ook nieuwsgierig waarom Vue, een relatief nieuwe framework, zo populair is geworden. Al ben ik af en toe tegen wat problemen aangelopen tijdens het ontwikkelen (zie hoofdstuk 9.2), door de goede documentatie van het framework heb ik met een vlot tempo het dashboard kunnen realiseren. In hoofdstuk 7 kunt u het onderzoek lezen.

Prototypes front-end frameworks

In mijn afstudeerplan had ik aangegeven prototypes te willen ontwikkelen van elk onderzocht Single-Page Application framework. In overleg met de bedrijfsmentor is het realiseren van prototypes komen te vervallen. Het uitwerken van alle prototypes zou veel tijd in beslag genomen hebben. Dit zou uiteindelijk ten kostte gaan van het daadwerkelijke product (het dashboard). Het raadplegen van artikelen, in combinatie met de eigen documentatie van het framework, was voldoende om tot een weloverwogen besluit te komen.

Ontwerpen userinterface Dashboard

Voor de front-end ontwikkeling had ik twee opties, namelijk het in gebruik nemen van een userinterface (UI) framework of een eigen ontwerp maken. In hoofdstuk 6.2 heb ik beschreven waarom ik uiteindelijk, uiteraard in overleg, voor een ontwerp had gekozen. Er zijn geen cruciale UI-wijzigingen plaatsgevonden tijdens de ontwikkelfase en dat kwam vooral omdat de product owner van tevoren wist wat hij kon verwachten. Ik heb positieve reacties gehad op het ontwerp en ik ben zelf ook wel trots op wat ik in een korte tijd heb kunnen realiseren.

Dashboard

Het belangrijkste product van het project is natuurlijk het dashboard zelf. Voor het dashboard heb ik een front en back-end applicatie ontwikkeld. Er waren wel wat uitdagingen met het inrichten van de back-end Web API, namelijk het omgaan met bestaande functionaliteiten. Het dashboard heeft

namelijk veel data van de chatbot nodig. Door het toevoegen van een gedeeld domein (beschreven in hoofdstuk 8.3) en het reactief maken van elementen (beschreven in hoofdstuk 10) heb ik ook aanpassingen moeten maken in de chatbot project. Dit was een leerzaam moment, voornamelijk omdat het project op een oudere versie van ASP.NET draaide. Uiteindelijk vind ik het leuk om te zien hoe twee 'afgeschermde' omgevingen toch met elkaar kunnen samenwerken.

Voor het front-end heb ik geprobeerd zoveel mogelijk de gebruikerservaring te verbeteren door het toevoegen van form validatie, animaties en meer feedback te geven aan de gebruiker als hij of zij het dashboard gebruikt. Verder heb ik de layout en elementen responsive gemaakt. Het dashboard is hierdoor toegankelijk op een telefoon, tablet en desktop. Ik kijk tevreden terug naar wat ik in een aantal weken heb mogen realiseren, zowel voor front als back-end. In Bijlage 9 kunt u korte videofragmenten bekijken van het product. Verder is het leuk te benoemen dat de videofragmenten door de CTO van TrueLime gebruikt wordt voor sales pitches.

Architectuurontwerpen
Het van tevoren in kaart brengen van een architectuur, om vervolgens te bespreken met de architect van TrueLime was voor mij een prettige werkwijze. Het verwerken van de feedback, maar ook goedkeuring krijgen voor mijn voorstellen had een positief effect op het ontwikkelproces. Ik wist namelijk precies wat ik moest uitwerken en mogelijke complicaties waren al van tevoren gesignaleerd. Verder heb ik schetsen gemaakt in mijn verslag om bepaalde concepten, zoals bijvoorbeeld de state manager Vuex en WebSockets (hoofdstuk 8.4 en 9.3), beter toe te kunnen toelichten.

14.2 Procesevaluatie

Voor het project waren er sprints van twee weken ingepland. Dit vond ik zelf prettig, omdat er genoeg tijd was om te ontwikkelen, maar ook om voorbereidingen te treffen voor de demo en het inrichten van de volgende sprint. Het samen prioriteren van requirements en bepalen wat er in het prototype moet komen met de product owner had ook een positief effect op de planning. Hierdoor had ik altijd voor ogen wat er uiteindelijk gebouwd moest gaan worden. Spontane veranderingen in de requirements konden we veel beter managen en dus ok direct de complicaties voor de deadline beperken. Het project was hierdoor goed afgebakend en heeft tot een prettig ontwikkelproces gezorgd. Verder had ik in de planning ook rekening gehouden met een 'uitloop' sprint. Hierdoor had een niet gehaalde sprint minder impact op de deadline van het dashboard. In een latere sprint kon ik mijn achterstand weer inhalen. Dit verminderde ook de werkdruk en had ik genoeg ruimte om te onderzoeken en ontwikkelen.

14.3 Reflectie op beroepstaken

Tijdens het opstellen van mijn afstudeerplan heb ik zeven beroepstaken aangegeven waar ik tijdens het afstuderen op ga focussen. Hieronder beschrijf ik per beroepstaak wat ik heb gedaan om eraan te voldoen.

A-1 Analyseren probleemdomein & opstellen probleemstelling
Na het ontmoetingsgesprek met de bedrijfsmentor heb ik een afstudeerplan opgesteld. In het gesprek is op een globaal niveau het probleem toegelicht. Tijdens het opstellen van het plan van aanpak (Bijlage 1) heb ik diverse afspraken gehad met de product owner en bedrijfsmentor. Hierin heb ik de probleemdomein en probleemstelling verder kunnen specificeren, zie hoofdstuk 2 en 3 voor meer informatie.

A-3 Vergaren en analyseren van requirements
Tijdens het opstellen van het plan van aanpak heb ik samen met de bedrijfsmentor en product owner

gezeten om het gehele project in detail te bespreken. Uit de gesprekken zijn er requirements naar voren gekomen, die ik vervolgens heb geprioriteerd met de MoSCoW-methode. Hoe het proces is verlopen is te lezen in hoofdstuk 4. Met de geprioriteerde requirements is er bepaald wat er in het prototype moet komen. Bij elke sprint heb ik de gekozen requirements, op basis van de feedback van de product owner, omgezet naar User Stories met de bijbehorende acceptatiecriteria. In Bijlage 3 tot en met 8 zijn de User Stories per sprint te vinden.

B-4 Gemotiveerd selecteren van ICT-gerelateerde oplossingen
Aan het begin van mijn afstudeeropdracht heb ik, zoals besproken in hoofdstuk 7, diverse Single-Page Application frameworks onderzocht. Na overleg met de opdrachtgever is er uit het onderzoek een framework gekozen voor de ontwikkeling van het dashboard. In de eerste Sprint, beschreven in hoofdstuk 8, is er een gedeeld domein bijgekomen voor het dashboard en chatbot. Het handmatig releasen van het domein naar de twee projecten was tijdrovend. Ik heb hiervoor Azure gebruikt om het automatisch builden en releasen mogelijk te maken. Verder heb ik in het ontwikkelproces meerdere technieken en concepten ontdekt, geselecteerd en toegepast. Waaronder de Vuex state manager (hoofdstuk 8.4), WebSockets (hoofdstuk 9.3) en Webhooks (hoofdstuk 10.1).

C-6 Ontwerpen software
In het afstudeerproces zijn er diverse architectuurontwerpen gemaakt van het dashboard. In de planningsfase heb ik de huidige situatie onderzocht. Met de vergaarde kennis heb ik diverse architectuurplaten opgesteld, waarin ik de bijbehorende voor- en nadelen had beschreven. Uiteindelijk zijn de variaties besproken met de product owner tot er een basisarchitectuur werd gekozen (hoofdstuk 5.2). In het ontwikkelproces is de architectur verder uitgebreid, zoals beschreven is in hoofdstuk 8.2, 9.3 en 10.1.

Verder is er een eigen ontwerp (userinterface) gemaakt van het dashboard, dat uiteindelijk mij heeft geholpen om de ontwikkeling van het front-end te versnellen. In hoofdstuk 6.2 beargumenteer ik in detail waarom ik voor een eigen vormgeving heb gekozen en in Bijlage 2 kunt u het eindresultaat bekijken.

D-14 Realiseren van software
Na mijn onderzoek voor Single-Page Application frameworks (hoofdstuk 7) ben ik begonnen met het ontwikkelen van het dashboard. Elke sprint werd de applicatie verder uitgebreid met de vooraf vastgestelde requirements. In hoofdstuk 8 tot en met 13 wordt het hele proces van het realiseren van software in zijn compleetheid besproken. Aangezien beschrijvingen en figuren van het eindresultaat niet de complete ervaring geeft van mijn opdracht, verwijs ik u graag naar Bijlage 9 waar korte videofragmenten bekeken kan worden.

Gc Kritisch, onderzoekend en methodisch werken
Tijdens mijn afstuderen heb ik gewerkt met de Scrum-methodiek. Hiervoor heb ik veel overleg gehad met de product owner en bedrijfsmentor. Per Sprint had ik een demo voorbereid en stelde ik mezelf kritische vragen hoe ik de volgende sprint beter kon inrichten. Verder was ik altijd voorbereid en actief bij de sessies en probeerde ik zoveel mogelijk in te spelen op de feedback van mijn bedrijfsmentor en product owner om een prototype te realiseren waar TrueLime tevreden mee is.

Naast het methodisch en kritisch werken heb ik ook een onderzoekende houding gehad. Aan het begin van het project heb ik de huidige situatie onderzocht, zoals beschreven in hoofdstuk 5.1, om zo tot een beter architectuur keuze te komen voor het dashboard. Verder heb ik een onderzoek gestart

naar diverse front-end frameworks, om tot het juiste framework te komen voor het ontwikkelen van het dashboard (hoofdstuk 7).

Gf Leren leren: voorbereiden op volgende studiefase en beroep
Tijdens mijn afstuderen heb ik kennis gemaakt met nieuwe technieken en concepten. Het inlezen over populaire Single-Page Application (SPA) frameworks heeft me veel inzichten gegeven over wat er momenteel speelt in de front-end wereld. Verder ben ik meer te weten gekomen over hoe en waarom state managers ingezet worden om een SPA beter in te richten (zie hoofdstuk 8.4). Het implementeren van WebSockets heeft ervoor gezorgd dat data realtime in het dashboard afgehandeld kan worden. Ik heb ervaring opgedaan in het ontwikkelen van een applicatie dat moet samenwerken met een bestaand systeem. Door kritisch te kijken hoe ik het dashboard kon inrichten, zonder dat het impact had op de bestaande functionaliteiten was een leuke uitdaging, waar ik in een volgende studiefase of beroep veel aan ga hebben.

14.4 Toekomst voor het dashboard

Het prototype is gebouwd om de interactie met de chatbot te verbeteren. TrueLime wilt het prototype gebruiken voor sales pitches. Ze hopen om een zorginstantie te kunnen helpen door het inzetten van chatbots. Mocht TrueLime het prototype in de toekomst willen gebruiken bij klanten, dan moet er wel rekening gehouden worden met het volgende:

- Het prototype bevat geen authenticatie, het is aanbevolen om dit te implementeren om de gegevens van de mantelzorgers en de cliënt te kunnen afschermen;
- Het dashboard is ontwikkeld met moderne technologieën, het is raadzaam om goed in kaart te krijgen wat de technische limitatie is van de infrastructure alvorens het dashboard ingezet wordt;
    - Het prototype werkt niet in Internet Explorer 8, omdat Vue bepaalde technieken van JavaScript gebruikt dat niet door de browser wordt ondersteund.
- Als het dashboard wordt uitgebreid, dan is het een goed moment om unit testen te schrijven om de kwaliteit van de code te bewaken. Mocha is een unit testing framework dat eenvoudig te integreren is in Vue;
- Voor grotere projecten is het verstandig om TypeScript, een superset van JavaScript, toe te voegen aan het dashboard, omdat het helpt potentiële runtime fouten te voorkomen (Vue, sd);
- In het kader van privacy is het verstandig om de data van de cliënt te anonimiseren, verder moet de gebruikers ook de mogelijkheid krijgen hun data te kunnen inzien, aanpassen of verwijderen.

In het volgende hoofdstuk vertel ik de conclusie van de onderzoeksvragen.

15. Conclusie onderzoeksvragen

Voor het beantwoorden van mijn onderzoeksvraag '*Hoe kan een dashboard voor TrueLime ontwikkeld worden, met het gebruik van een Single-Page application framework, om de interactie met de chatbot te verbeteren?'*. In dit hoofdstuk geef ik een terugkoppeling op de deelvragen.

1. **Welke functionaliteiten moet het dashboard bevatten om de interactie met de chatbot te verbeteren?**

   De volgende functionaliteiten zijn vastgesteld om de interactie met de chatbot te verbeteren:
   - Beheren van logboek
   - Agenda (afspraken)
   - Lijsten (taken)
   - Reactieve elementen (realtime dataverwerking)
   - Communicatie met virtuele assistent (vraag en antwoord)
   - Persoonlijke omgeving
   - Notificaties
   - ICE-overzicht voor spoedgevallen
   - Afgeschermde omgeving
   - Stemherkenning
   - Vervoer regelen voor patiënt
   - Chatten met mantelzorgers

   In hoofdstuk 4 is te lezen hoe de requirements zijn achterhaald, geprioriteerd en ingepland.

2. **Wat is de huidige situatie bij TrueLime?**
   a. **Wat is een chatbot?**

      Een chatbot is een geautomatiseerd softwareprogramma. Bots worden vaak ingezet bij een helpdesk-omgeving om op een snel en effectieve wijze een klant te helpen. Het juist inzetten van de chatbot geeft de klant de ervaring dat hij of zij met een persoon, in dit geval een medewerker aan het praten is (Microsoft, 2019). Het toepassen van een chatbot zorgt voor het verbeteren van de gebruikservaring, omdat ze altijd beschikbaar zijn voor het beantwoorden van vragen. De chatbots begrijpen de context uit een vraag en kunnen hierdoor ook sneller antwoord geven. Door de chatbot te implementeren, worden handmatige handelingen geautomatiseerd. Hiermee zal er kostenreductie plaatsvinden (Den Arend, 2018).

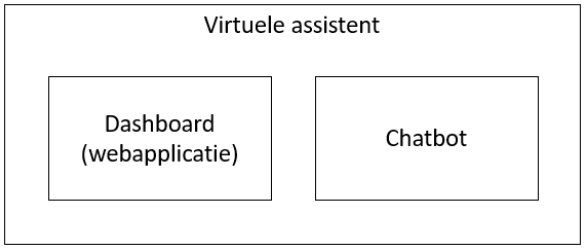
   b. **Welke technieken worden er gebruikt om de chatbot te realiseren?**

      TrueLime heeft met Azure Bot Services van Microsoft een chatbot kunnen ontwikkelen voor mantelzorgers en hun cliënten. De chatbot kan je inzetten op verschillende kanalen, zoals: Skype, Slack, Facebook en Cortana. Het is op dit moment mogelijk om via een chat interface (chat emulator) en met Google Assistant (spraak) te communiceren met de bot.

   c. **Wat is een virtuele assistent?**

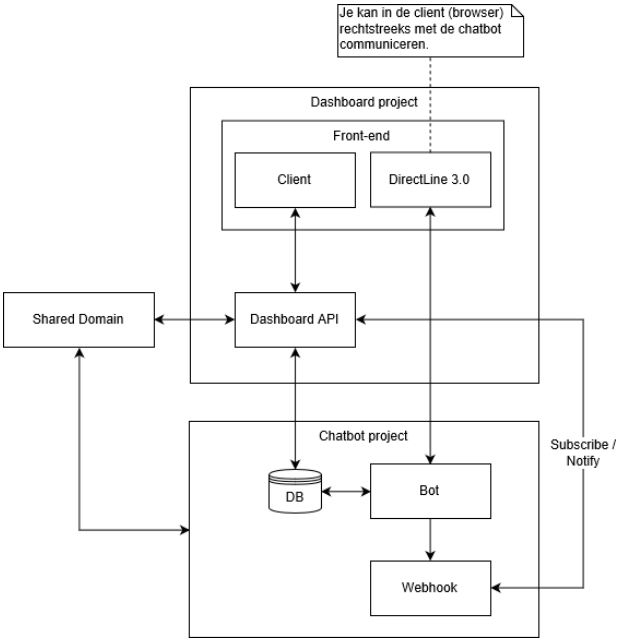
      De termen chatbot, dashboard en virtuele assistent kunnen lastig van elkaar te onderscheiden zijn, omdat ze veel raakvlakken met elkaar hebben. Het is belangrijk om vast te leggen wat ze betekenen voor dit project. Samen met de product owner is afgesproken dat de chatbot wordt gezien als de achterliggende techniek van de virtuele assistent, terwijl het dashboard de visualisatie en interactie verzorgd. Eigenlijk worden

beide projecten, chatbot en dashboard, samen als de 'virtuele assistent' gezien, zoals in Figuur 43 is weergegeven.



Figuur 43: Virtuele assistent

3. **Hoe kan de communicatie tussen het dashboard en chatbot gerealiseerd worden?**
   Graag verwijs ik u naar hoofdstuk 5.2 waar ik drietal architecturen heb bedacht om de communicatie tussen het dashboard en chatbot te realiseren. In hoofdstuk 8.2 is er een gedeeld domein toegevoegd aan beide omgevingen, om zo geen dubbele code te hebben. Hierdoor staan alle gedeelde functionaliteiten op één plek wat de aanpasbaarheid verbeterd. In hoofdstuk 10.1 is er een webhook toegevoegd aan de chatbot omgeving. Met een webhook is het mogelijk om de dashboardomgeving op de hoogte te stellen van nieuwe data, wanneer wijzigingen rechtstreeks in de chatbot plaatsvinden. In Figuur 44 is het eindresultaat van de architectuur te zien.



Figuur 44: Architectuur

4. **Hoe kan een visuele- en spraakinterface gerealiseerd worden in het dashboard?**
   Het uiterlijk van het dashboard is niet gebaseerd op een userinterface framework zoals Bootstrap en Material Design. Het ontwerpen en uitwerken van een eigen dashboard heeft de volgende voordelen:

   - Een unieke uitstraling;
   - De userinterface wordt ontworpen met de requirements van het project in het achterhoofd;
   - Volledig controle over hoe het dashboard er uit gaat zien;

- Het ontwerp is voordat de ontwikkeling van start gaat al goedgekeurd door de product owner. Dit voorkomt plotselinge (cruciale) UI-wijzigingen in het ontwikkelproces;
- Aangezien ik het dashboard zelf in code uitwerk, weet ik waar er rekening gehouden moet worden in het ontwerpproces;
- Betere tijdschatting voor het realiseren van de techniek.

In hoofdstuk 6 wordt in detail toegelicht hoe de vormgeving van het dashboard tot stand is gekomen. In hoofdstuk 10 beschrijf ik meer over hoe ik de communicatie met de chatbot op technisch niveau heb geïmplementeerd.

5. **Welke Single-Page Application framework is geschikt voor het ontwikkelen van het dashboard?**
   Uit het onderzoek van diverse SPA frameworks is Vue.js als beste uitgekomen voor het project. Het framework scoort goed op performance en is compact. Verder heeft Vue een simpele integratie, en wordt mede hierdoor als minder complex ervaren. In combinatie met goede documentatie en een laag instapniveau heeft ervoor gezorgd dat er vlot aan het dashboard ontwikkeld kon worden. Zie hoofdstuk 7 om meer te lezen over het onderzoek.

Begrippenlijst

In de tabel hieronder zijn alle begrippen met hun omschrijving opgenomen.

Begrip	**Omschrijving**
Boilerplate code	Boilerplate code is code die regelmatig wordt gebruikt maar minimale wijzigingen nodig heeft op de daadwerkelijke inhoud (Techopedia, sd).
Chatbot	Een chatbot is een geautomatiseerd softwareprogramma. Bots worden vaak ingezet bij een helpdesk-omgeving om op een snel en effectieve wijze een klant te helpen. Het juist inzetten van de chatbot geeft de klant de ervaring dat hij of zij met een persoon, in dit geval een medewerker aan het praten is (Microsoft, 2019).
Document Object Model	Het Document Object Model zorgt voor een structurele representatie van HTML-documenten, wat het mogelijk maakt de inhoud en visuele presentatie te veranderen (Mozilla, sd).
Event Bus	Een event bus is een mechanisme waarin het mogelijk is om verschillende componenten met elkaar te laten communiceren, zonder dat ze van elkaar af weten.
Product Backlog Item	De Product Backlog bestaat uit een lijst met items (User Stories) die uitgevoerd moeten worden tijdens de ontwikkeling van het product (Scrumguide, sd).
Reactief	De applicatie moet reactief zijn, wat betekent dat het inspeelt op veranderingen. Als een patiënt bijvoorbeeld een nieuwe klus heeft toegevoegd, dan moet de mantelzorger dat direct zien verschijnen op het dashboard.
Single-Page Application	Een Single-Page Application zorgt voor een betere gebruikerservaring, omdat de volledige pagina van een webapplicatie niet bij elke interactie van de gebruiker (opnieuw) ingeladen hoeft te worden. De applicatie reageert hierdoor sneller op de input van een gebruiker.
Superset (JavaScript)	Een superset van JavaScript is een nieuwe laag bovenop de programmeertaal. Het biedt de programmeur meer opties in de taal en een specifiek workflow.

Bibliografie

Andrew, P. (2011, Maart 14). *Discussing the Pros and Cons of Using a CSS Framework*. Opgehaald van Speckyboy: https://speckyboy.com/discussing-the-pros-and-cons-of-using-a-css-framework/

Brownlee, J. (2016, April 4). *Conversational Interfaces, Explained*. Opgehaald van Fast Company: https://www.fastcompany.com/3058546/conversational-interfaces-explained

Den Arend, C. (2018, November 17). *Wat zijn chatbots?* Opgehaald van Sogeti: https://www.sogeti.nl/expertises/chatbots

Djirdeh, H. (2018, Juni 26). *Managing State in Vue.js*. Opgehaald van Medium: https://medium.com/fullstackio/managing-state-in-vue-js-23a0352b1c87

Fortech. (2019, Januari 15). *Blazor, a C# Friendly Single-Page Application (SPA) Framework*. Opgehaald van Fortech: https://www.fortech.ro/blazor-spa-framework/

FusionCharts. (2018). *Top JavaScript Frontend Frameworks Comparison in 2018*. Opgehaald van FusionCharts: https://www.fusioncharts.com/resources/developers/js-frontend-frameworks-comparison

i-Reserve. (sd). *Wat is een webhook*. Opgehaald van i-Reserve: https://support.i-reserve.net/nl/ontwikkelaar/webhooks/wat-is-een-webhook/

Ismail, K. (2018, November 2). *What Is a Single Page Application?* Opgehaald van CMS Wire: https://www.cmswire.com/digital-experience/what-is-a-single-page-application/

Jin, R. (2017, Juli 10). *Webhook vs API: What's the Difference?* Opgehaald van Hackernoon: https://hackernoon.com/webhook-vs-api-whats-the-difference-8d41e6661652

Microsoft. (2018, Oktober 1). *An introduction to NuGet*. Opgehaald van Microsoft Docs: https://docs.microsoft.com/en-us/nuget/what-is-nuget

Microsoft. (2019, oktober 1). *About Azure Bot Service*. Opgehaald van Microsoft Docs: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0

Microsoft. (sd). *Real-time ASP.NET with SignalR*. Opgehaald van .NET: https://dotnet.microsoft.com/apps/aspnet/real-time

Mozilla. (sd). *Document Object Model (DOM)*. Opgehaald van MDN: https://developer.mozilla.org/nl/docs/Web/API/Document_Object_Model

One More Thing. (2018, Augustus 3). *Concurrentie voor Siri: Nederlandse Google Assistant nu op iOS*. Opgehaald van One More Thing: https://www.onemorething.nl/2018/08/google-assistant-nederlands-nu-ios/

Roberts, M. (2018, Juni 3). *Vue.js: Pros, cons and things I learned along the way.*. Opgehaald van Medium: https://medium.com/@michealjroberts/vue-js-pros-cons-and-things-i-learnt-along-the-way-eb724a412113

Schae, J. (2018, Maart 31). *A Real-World Comparison of Front-End Frameworks with Benchmarks*. Opgehaald van Medium: https://medium.freecodecamp.org/a-real-world-comparison-of-front-end-frameworks-with-benchmarks-2018-update-e5760fb4a962

Scrumguide. (sd). *Product Backlog*. Opgehaald van Scrumguide: https://scrumguide.nl/product-backlog

Shapiro, D. (2016, Juni 16). *Understanding Component-Based Architecture*. Opgehaald van Medium: https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238

Štolfa, T. (2016, Maart 10). *The Future of Conversational UI Belongs to Hybrid Interfaces*. Opgehaald van Medium: https://medium.com/the-layer/the-future-of-conversational-ui-belongs-to-hybrid-interfaces-8a228de0bdb5

Strahl, R. (2018, Juli 31). *Web Assembly and Blazor: Re-assembling the Web*. Opgehaald van West-Wind: https://weblog.west-wind.com/posts/2018/Jul/31/Web-Assembly-and-Blazor-Reassembling-the-Web

Sviatoslav, A. (2017, Juni 8). *The Best JS Frameworks for Front End*. Opgehaald van RubyGarage: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end

Techopedia. (sd). *Boilerplate*. Opgehaald van Techopedia: https://www.techopedia.com/definition/1259/boilerplate

The State of JavaScript. (2018). *Front-end Frameworks*. Opgehaald van The State of JavaScript: https://2018.stateofjs.com/front-end-frameworks/overview/

ToolsHero. (2018, Juni 14). *MoSCoW Methode*. Opgehaald van ToolsHero: https://www.toolshero.nl/project-management/moscow-methode/

Vue. (sd). *TypeScript Support*. Opgehaald van Vue: https://vuejs.org/v2/guide/typescript.html

Vue.js. (sd). *Reactivity in Depth*. Opgehaald van Vue.js: https://vuejs.org/v2/guide/reactivity.html

Vuex. (sd). *What is Vuex?* Opgehaald van Vuex: https://vuex.vuejs.org/

Bijlage 1: Plan van aanpak

(Voorblad en inhoudsopgaven zijn niet opgenomen i.v.m. complicaties paginanummering en inhoudsopgaven in Microsoft Word.)

1. Aanleiding en context

TrueLime is een ICT-bedrijf dat in 1997 is begonnen met het maken van softwareapplicaties. Het bedrijf is gevestigd in Breda en heeft 63 medewerkers in dienst. TrueLime creëert diverse producten, zoals: intranetten, websites, portaalontwikkelingen en (maatwerk) applicaties. Ze zijn vooral gespecialiseerd in softwareoplossingen voor werkprocessen van juridische aard. Daarnaast adviseren en begeleiden de consultants van TrueLime organisaties bij het ontwikkelen van een strategie naar succesvolle online toepassingen. Het bedrijf focust zich vooral op klanten in de zorg, bij gemeente en overheid. TrueLime staat bekend om het zaaksysteem Octopus, dat administratieve en juridische werkprocessen automatiseert, beheert, en optimaliseert.

TrueLime heeft een chatbot ontwikkeld voor mantelzorgers en cliënten. De applicatie helpt mantelzorgers met het verzorgen van hun naasten. Een chatbot heeft verschillende skills, met name voor het beheren van afspraken, taken en logboek. De patiënt kan zelf ook klusjes toevoegen voor de mantelzorger. De chatbot is een spraakgestuurde applicatie waarmee de mantelzorger en hun cliënten kunnen communiceren.

2. Probleemstelling

In de praktijk is het probleem ontstaan dat de spraakgestuurde app niet toegankelijk is voor de doelgroep. Patiënten en mantelzorgers zijn vaak wat ouder en hebben meer visuele begeleiding nodig voor de interactie met de chatbot. *TrueLime wilt dat de chatbot zowel een visueel als conversational[9] interface gaat krijgen om zo beide doelgroepen te kunnen ondersteunen.*

3. Doelstelling en eindresultaat

Het doel is om een dashboard (webapplicatie) te ontwikkelen voor het visualiseren van de interactie met de chatbot om zo de toegankelijkheid voor de mantelzorgers en hun patiënten te verbeteren.

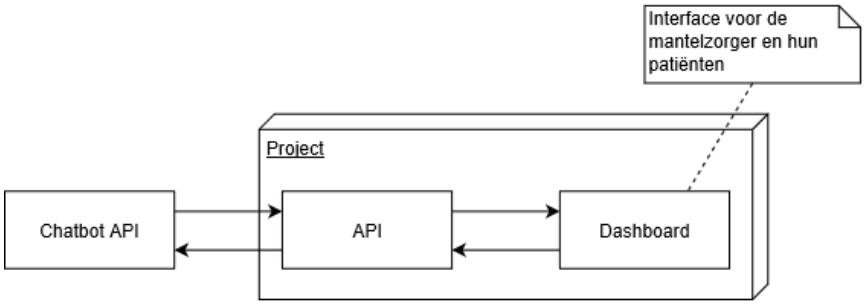
Onderzoek

Om het visuele gedeelte van het dashboard zo gebruiksvriendelijk mogelijk te maken is er bij TrueLime een wens ontstaan om een Single-Page Application (SPA) te ontwikkelen. Een SPA zorgt voor een betere gebruikerservaring, omdat de volledige pagina van een webapplicatie niet bij elke interactie van de gebruiker (opnieuw) ingeladen hoeft te worden. De applicatie reageert hierdoor sneller op de input van een gebruiker. Er zijn diverse SPA frameworks in opkomst en TrueLime wilt graag meer weten over de voor- en nadelen van elk framework alvorens het ontwikkelen van het dashboard van start gaat.

[9] Een conversational interface zorgt ervoor dat een gebruiker met spraak een applicatie kan besturen.

Het eindresultaat

Na het uitvoeren van de opdracht zal er een dashboard beschikbaar zijn voor de mantelzorgers en hun patiënten. Het dashboard gaat ervoor zorgen dat de doelgroep zowel visueel als via spraak kan interacteren met de chatbot. De webapplicatie wordt een SPA die toegankelijk is op een telefoon, tablet en computer. Hiermee bereik je meer 'kanalen' waar de applicatie gebruikt kan worden. Aangezien het dashboard op het moment geen onderdeel uitmaakt van de chatbot, zal er een API ontwikkeld worden voor de communicatie tussen de chatbot en het dashboard[10]. In Figuur 1 ziet u een globale weergave van de totale applicatie.



Figuur 1: Globale weergave van applicatie

Afbakening

Het ontwikkelen van het dashboard is het primaire doel van de opdracht. De bestaande skills van de chatbot moeten gevisualiseerd worden. Samen met de product owner zijn er globale requirements vastgesteld en geprioriteerd volgens de MoSCoW-methodiek, zoals in Tabel 1 is weergegeven.

Must have Deze eisen moeten in het eindresultaat komen anders is het product niet bruikbaar.	• Logboek • Agenda • Lijsten • Reactieve elementen • Communicatie met virtuele assistent (vraag en antwoord)
Should have Deze eisen zijn gewenst, maar het product is alsnog bruikbaar als ze niet worden gerealiseerd.	• Persoonlijke omgeving • Notificaties • ICE-overzicht voor spoedgevallen
Could have Deze eisen worden gerealiseerd als er nog tijd over is.	• Afgeschermde omgeving • Stemherkenning • Vervoer regelen voor patiënt

[10] Tijdens het uitvoeren van de opdracht wordt bepaald hoe de daadwerkelijke communicatie tussen het dashboard en huidige app gaat verlopen. De structuur zou eventueel nog kunnen wijzigen.

Won't have Deze eisen zullen waarschijnlijk niet gerealiseerd worden.	• Chatten met mantelzorgers *Chatten in het dashboard is tegen de visie van het product. Het aanbieden van een chat mogelijkheid geeft de gebruikers de mogelijkheid om de geïntegreerde skills in de chatbot te ontwijken.*

Tabel 17: Prioriteren requirements

De belangrijkste functionaliteiten moeten gerealiseerd worden in het prototype. Per sprint wordt er opnieuw naar de eisen gekeken en indien nodig opnieuw geprioriteerd en aangescherpt. Graag verwijs ik u naar Bijlage A voor meer informatie over het prioriteringsproces.

4. Randvoorwaarden

Voor het project zijn de volgende randvoorwaarden opgesteld:

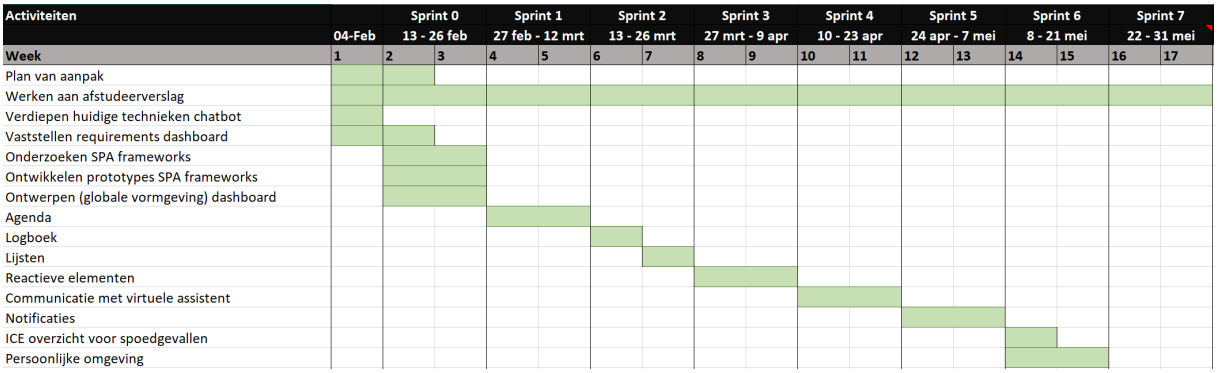
Huisvesting	Het dashboard wordt ontwikkeld op het kantoor van TrueLime.
Soort	De chatbot is een innovatieproject.
Tijd	Het prototype moet binnen 17 weken gerealiseerd zijn.
Meetings	Per week zal er een tweetal momenten ingepland worden om het project te bespreken.
Expertise	• Nick van der Raaf heeft de chatbot ontwikkeld. In het project is hij de contactpersoon voor technische vragen; • Wesley van Strien is mijn bedrijfsmentor tijdens het uitvoeren van het project; • Hans van der Linden is de product owner. Samen bepalen we welke functionaliteiten er gerealiseerd moet worden en zal het prototype per iteratie uitgebreid worden op basis van zijn feedback en wensen.
Programmeertalen	• C# (back-end); • JavaScript (front-end).
Tools en omgeving	• Visual Studio voor het ontwikkelen van het dashboard; • GIT voor versiebeheer; • Azure DevOps voor het beheren van de Scrum-omgeving; • Microsoft Azure voor het beheren van de chatbot; • Draw.io voor het modeleren van de architectuur; • Adobe XD voor het ontwerpen van het dashboard.

Tabel 18: Randvoorwaarden

5. Aanpak

Voor het realiseren van het dashboard wordt Scrum als ontwikkelmethodiek ingezet. Door het project incrementeel op te zetten zal per iteratie de requirements (User Stories) aangescherpt worden om zo na 17 weken tot een goed product te komen. Verder gebruikt TrueLime Scrum voor al hun projecten en willen graag dat ik hierin meedraai om zo hun processen beter te volgen.

Aangezien het project alleen met de bedrijfsmentor wordt uitgevoerd is het niet volledig volgens de standaarden van Scrum. Er zullen Sprints van twee weken plaatsvinden, waarin we twee keer per week samenkomen om de voortgang te bespreken. Aan het eind van een Sprint zal er een review moment plaatsvinden met de product owner. Tijdens de Sprint Review is er ruimte om de volgende User Stories in te richten voor de volgende iteratie. In Figuur 2 is er een globale planning van de gewenste functionaliteiten uit het prototype te zien.



Figuur 2: Globale planning

Sprint 7 is bewust leeg gelaten om ruimte te creëren voor het verfijnen van de applicatie en mogelijke vertragingen.

6. Risico's

Het is belangrijk om mogelijke risico's in kaart te brengen, al voordat het project van start gaat. Hieronder heb ik een aantal risico's opgesomd met een maatregel die genomen kan worden om het te vermijden.

1. Het dashboard en de chatbot hebben overeenkomende functies. Dit kan een negatieve invloed hebben op de aanpasbaarheid van beide applicaties.

   Maatregel: Voor de ontwikkeling moet er goed nagedacht worden over hoe de communicatie tussen beide applicaties verloopt. De oplossing is om een globale architectuur te bedenken en te bespreken met de product owner en architect om het probleem te verhelpen.

2. De chatbot is afhankelijk van het dashboard.

   Maatregel: Tijdens het ontwikkelproces moet gekeken worden hoe het dashboard zo modulair mogelijk opgezet kan worden. Structuur van het dashboard wordt bepaald op basis van de chatbot. De chatbot mag niks te weten komen over het dashboard. Dit moet tijdens de ontwerpfase van het architectuur al meegenomen worden.

3. Het "SPA-frameworks" onderzoek duurt langer dan nu is opgenomen in de globale planning, hierdoor is het niet mogelijk om alle functionaliteiten van het dashboard te implementeren.

Maatregel: Aangezien het project in Sprints wordt opgezet, zal een mogelijke achterstand vroegtijdig naar voren komen. Tijdens de Sprint Planning en Sprint Review is er een mogelijkheid om de User Stories dusdanig aan te passen om zo een volwaardig prototype op te leveren na 17 weken.

Bijlage A: Prioriteren requirements

Tijdens de kick-off zijn diverse wensen besproken. De requirements zijn vervolgens geprioriteerd volgens de MoSCoW-methodiek, omdat het project binnen 17 weken afgerond moet zijn. Door het prioriteren is er bepaald wat in het prototype gaat komen. De volgende (globale) requirements zijn opgesteld:

- Persoonlijke omgeving
- Afgeschermde omgeving:
    - Inloggen
    - Gebruikersrollen
- Agenda:
    - Agenda overzicht
    - Afspraken beheren
- Logboek:
    - Inchecken
    - Uitchecken (met verslag)
- Lijsten:
    - Boodschappenlijst beheren
    - Klussenlijst beheren
- Notificaties:
    - Notificatie sturen van afspraak patiënt
    - Notificatie ontvangen zodra er een nieuwe activiteit is toegevoegd
- ICE-overzicht voor spoedgevallen
- Reactieve elementen
- Communicatie met virtuele assistent (vraag en antwoord)
- Stemherkenning
- Vervoer regelen voor een patiënt
- Chatten met mantelzorgers

Verdeling volgens MoSCoW:

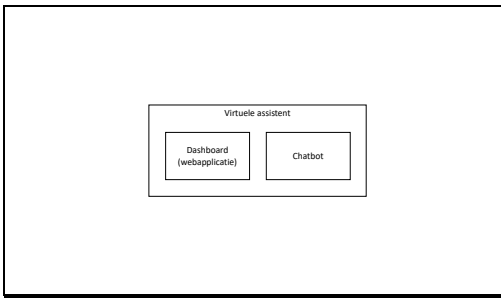
Must have	• Logboek • Agenda • Lijsten • Reactieve elementen • Communicatie met virtuele assistent (vraag en antwoord)
Should have	• Persoonlijke omgeving • Notificaties • ICE-overzicht voor spoedgevallen
Could have	• Afgeschermde omgeving • Stemherkenning • Vervoer regelen voor patiënt
Won't have	• Chatten met mantelzorgers *Chatten binnen het dashboard is tegen de visie van het product. Het aanbieden van een chat mogelijkheid geeft de gebruikers de mogelijkheid om de geïntegreerde skills in de chatbot te ontwijken.*

Presentatie:

Tijdens de kick-off met de bedrijfsmentor en product owner zijn er diverse functionaliteiten naar voren gekomen. Het was belangrijk om alle input om te zetten naar features (oftewel globale requirements) en te prioriteren om zo de scope te kunnen bepalen voor het project.

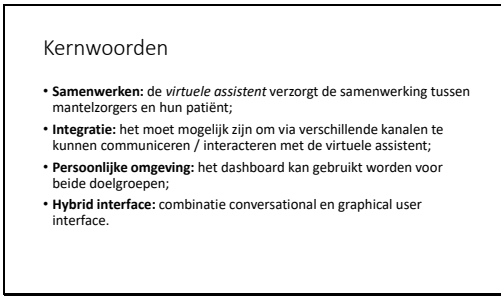
Ter voorbereiding heb ik een presentatie opgesteld om de product owner te informeren, maar ook te overtuigen waarom ik voor deze prioritering heb gekozen. De presentatie was een fijn hulpmiddel om mijn verhaal goed te kunnen overbrengen. Hierdoor is in het gesprek een scope bepaald voor het project. De slides zijn hieronder als hand-out opgenomen:

Dia 1



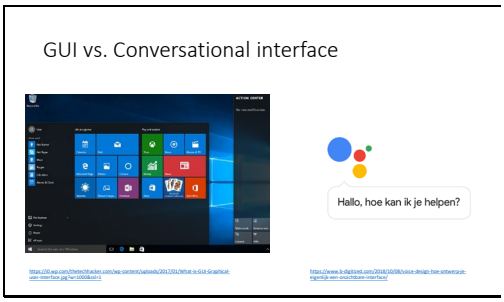
Afgesproken dat we voortaan ‘virtuele assistent’ zeggen in plaats van ‘chatbot’. Wel is het belangrijk om te weten dat het dashboard en chatbot technisch twee verschillende entiteiten zijn. Samen vormen ze de virtuele assistent.

Dia 2



Uit het gesprek zijn bij mij vier kernwoorden naar voren gekomen, namelijk: samenwerken, integratie, persoonlijke omgeving en een hybrid interface. Dit moet naar mijn mening ook de focus worden van het project.

Dia 3



GUI, oftewel een graphical userinterface, biedt de mogelijkheid om een applicatie te beheren via visuele elementen. Een conversational interface verbergt meeste visuele aspecten en is juist via spraak te benaderen.

Dia 4

Chatbot

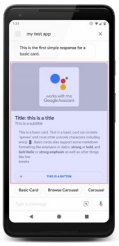
- Chatbot is de techniek achter de virtuele assistent;
- Chatbot heeft kanalen, zoals: Google Assistant, Skype*, Slack* **en het dashboard**;
- Chatbot stuurt alleen tekst terug naar de aanvrager, de kanalen bepaalt wat ermee gebeurt.

* *Niet relevant voor de virtuele assistent, maar geeft meer context wat een kanaal is.*

Dia 5

Google Assistant

- **Huidige functionaliteit:** De chatbot geeft nu een stukje tekst terug aan Google Assistent
- **Uitbreiding:** We kunnen ook een zogeheten card terugsturen naar de device:
- **Voorbeeld:** patiënt maakt een afspraak via Google Assistent. Vervolgens krijgt hij of zij de melding dat de afspraak succesvol is verwerkt met een verwijzing naar de agenda pagina van het dashboard om zo alle afspraken te kunnen inzien.

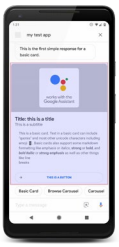


Wat heeft Google Assistant te bieden voor de virtuele assistent?

Dia 6

Google Assistant

- **Mogelijke knelpunt:** Chatbot moet weten van het dashboard (of moet vragen aan het Dashboard API wat het moet terugsturen naar de device);



Dia 7

Dashboard voor **mantelzorger & patiënt**

- **Eén webapplicatie** toegankelijk op telefoon, tablet en desktop;
- Het dashboard **visualiseert de data** van de chatbot;
- Maar, het moet mogelijk zijn om **spraakopdrachten** te sturen naar de chatbot (dus zonder andere kanalen, zoals Google Assistant).

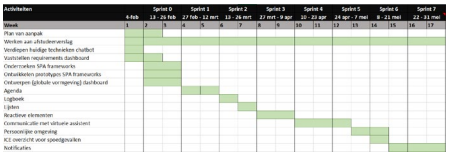
Dia 8

Dashboard voor **mantelzorger & patiënt**

- **Mogelijke knelpunt:** er is een Web Speech API beschikbaar dat in de browser *(Javascript)* spraak kan omzetten naar tekst. Het is op het moment alleen beschikbaar in Google Chrome. Nederlands wordt wel ondersteund!

Dia 9

Globale planning



Dia 10

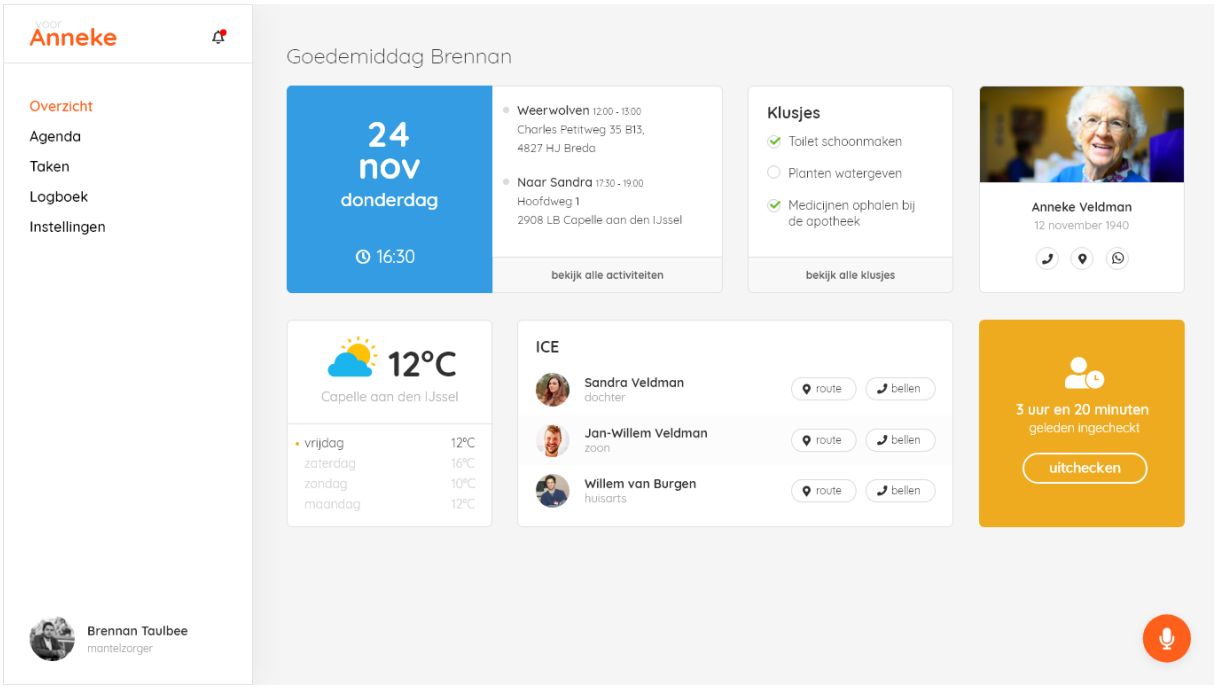
Mantelzorgers en hun patiënt kunnen dus:

- Communiceren via het dashboard (mobiel, tablet en desktop);
- Communiceren met Google Assistant (mobiel, tablet, Google Home):
  - Elementen op het dashboard zijn reactief en passen aan zodra er wijzigingen heeft plaatsgevonden

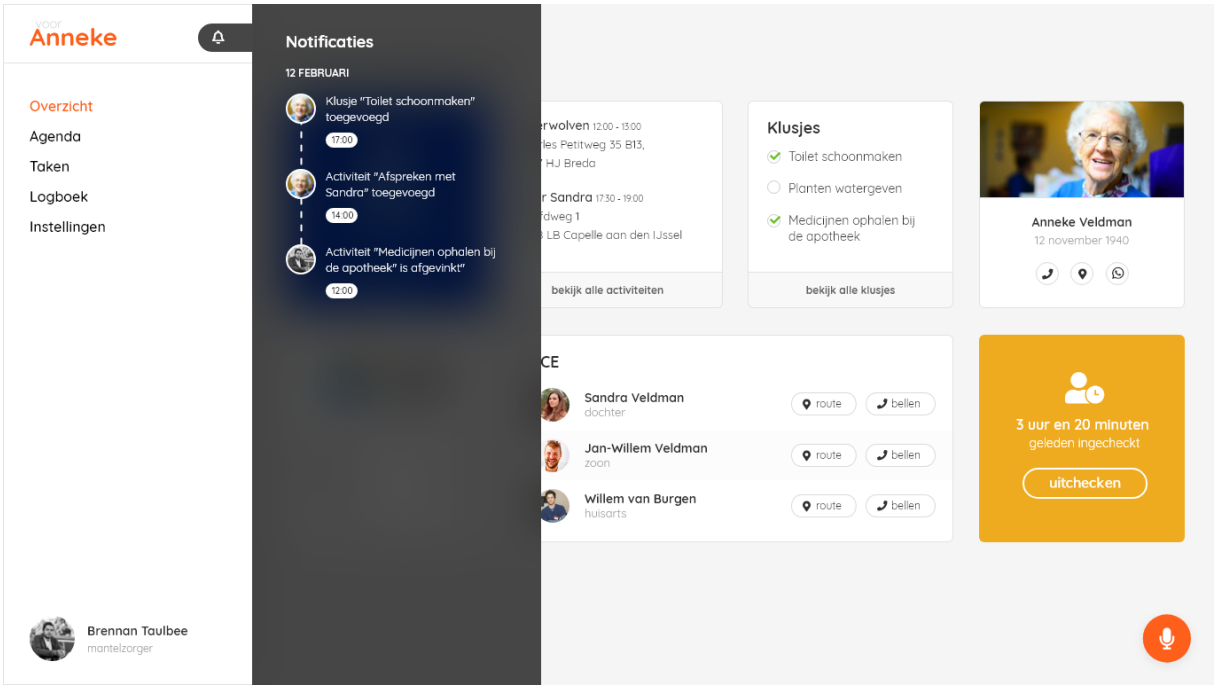
Korte samenvatting hoe er nu gecommuniceerd kan worden met de virtuele assistent.

Bijlage 2: Ontwerp van het dashboard

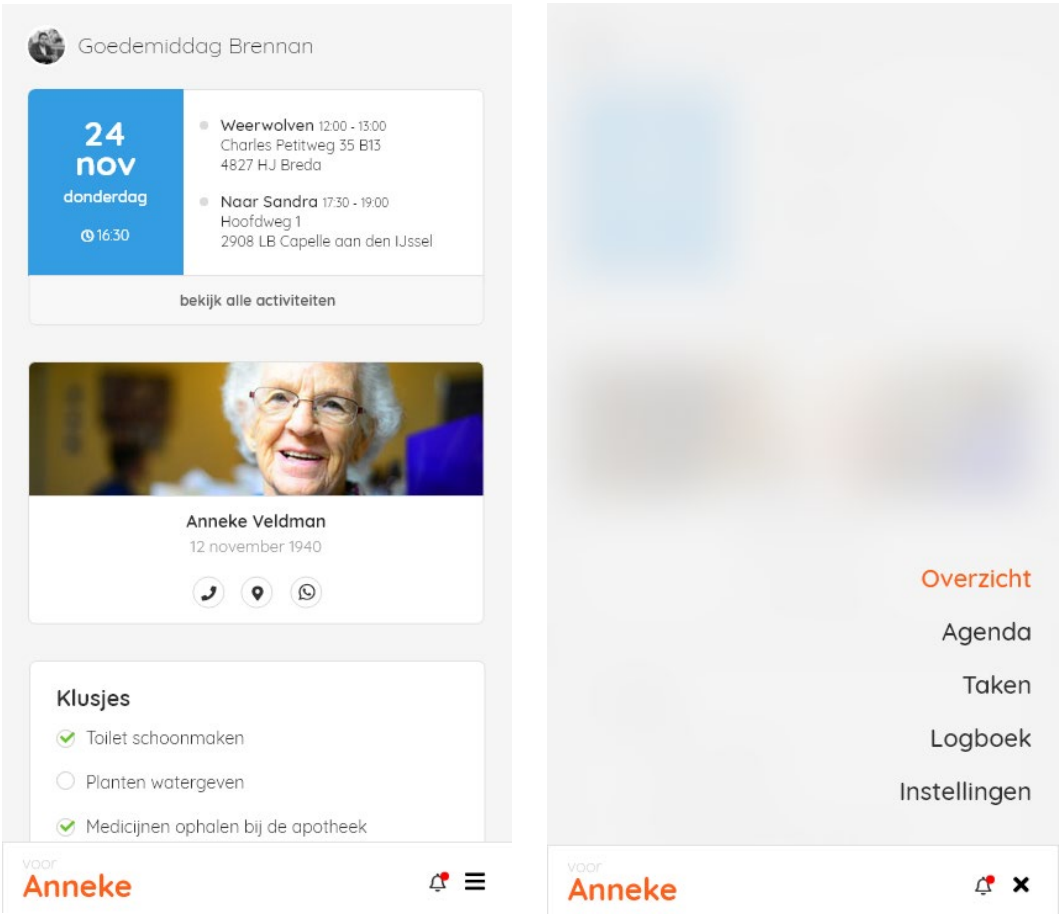
Op het moment dat de globale requirements bekend waren is er een start gemaakt aan het ontwerp. Dit is al vanaf Sprint 0 gevisualiseerd en is per sprint uitgebreid op basis van de nieuwe functionaliteiten. De schetsen zijn een goed hulpmiddel geweest voor het ontwikkelen van het front-end. De product owner weet namelijk van tevoren hoe een bepaald functionaliteit eruit gaat zien en ik weet hoe ik het technisch moet realiseren.



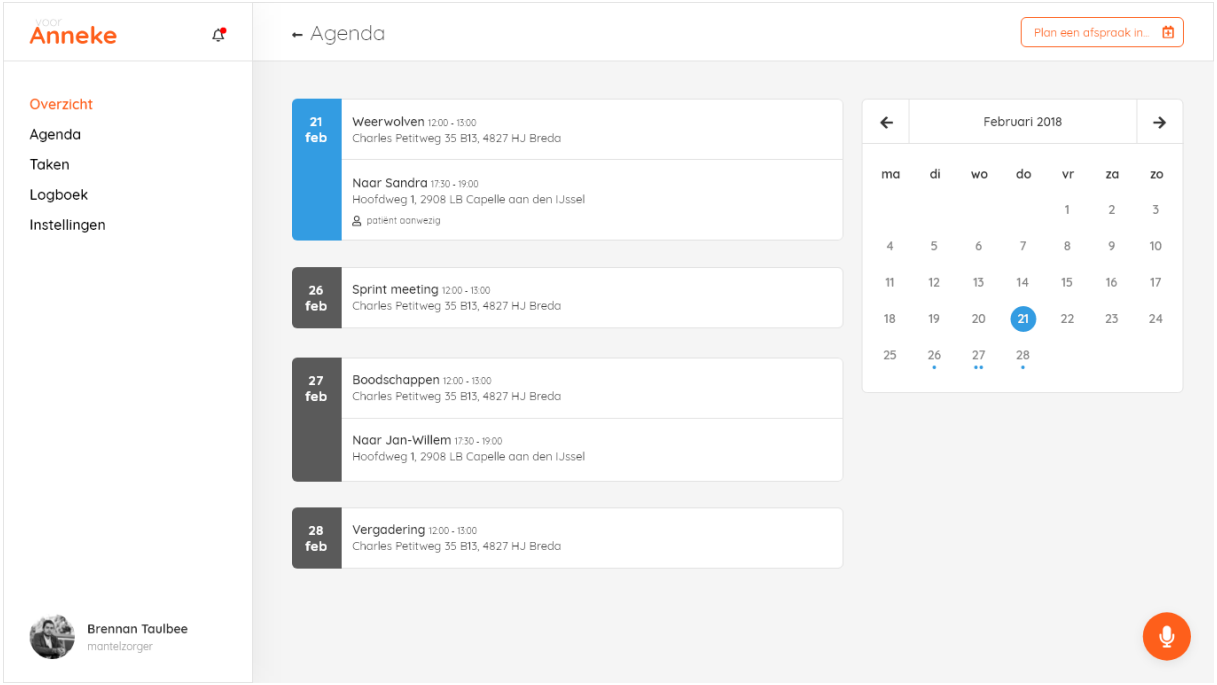
Figuur 1: Overzicht van dashboard



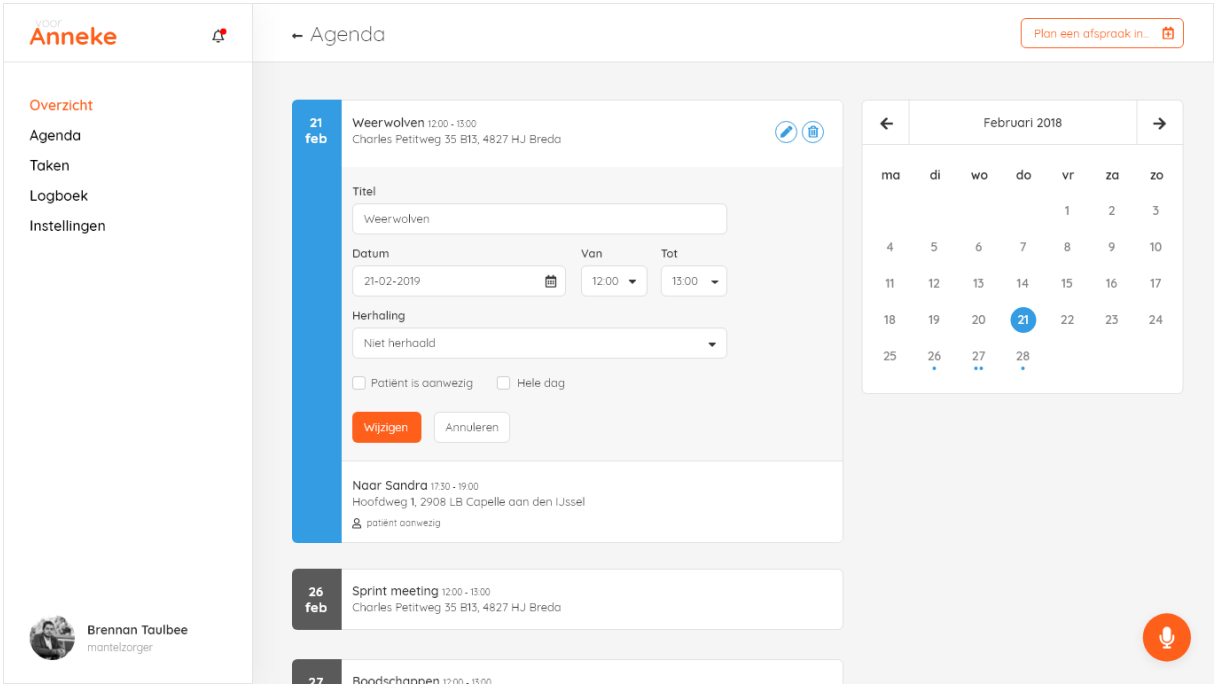
Figuur 2: Notificatiescherm



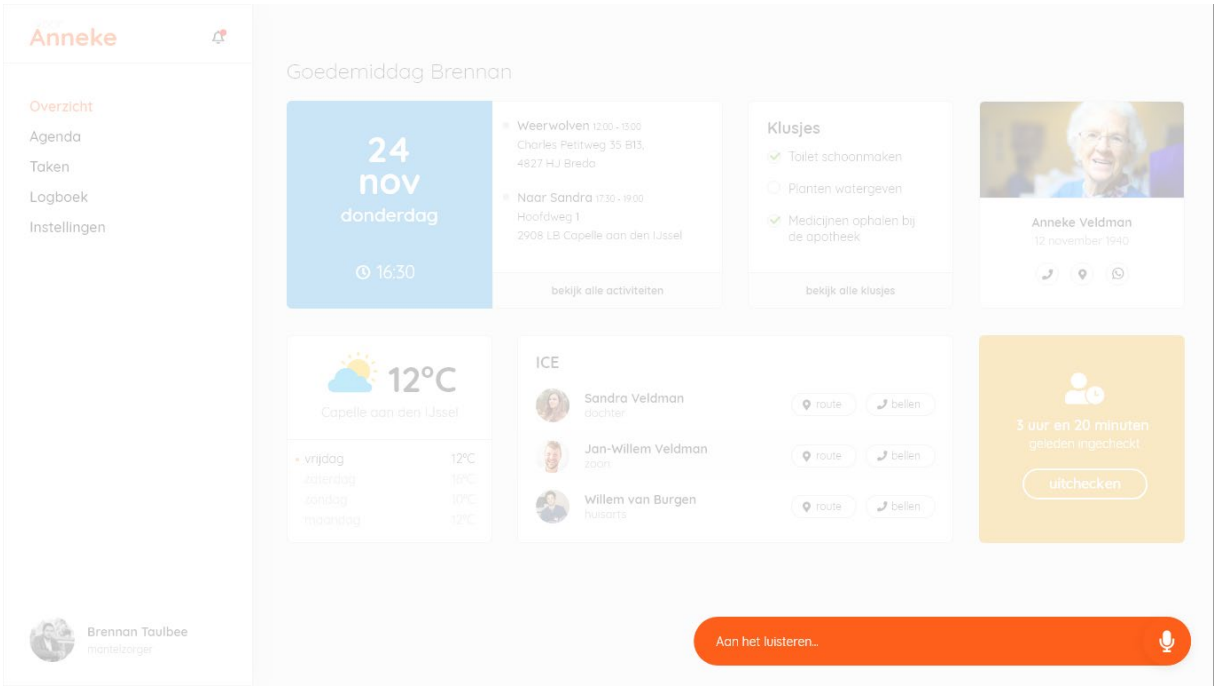
Figuur 3: Dashboard op mobiel



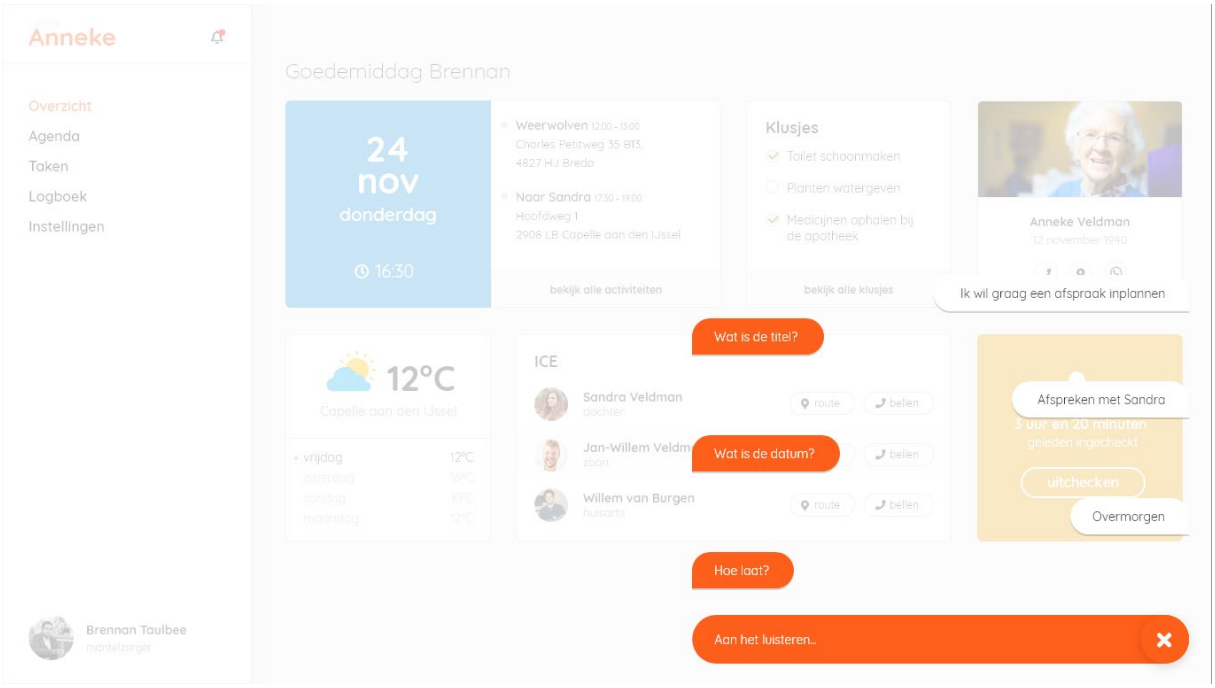
Figuur 4: Agenda overzicht



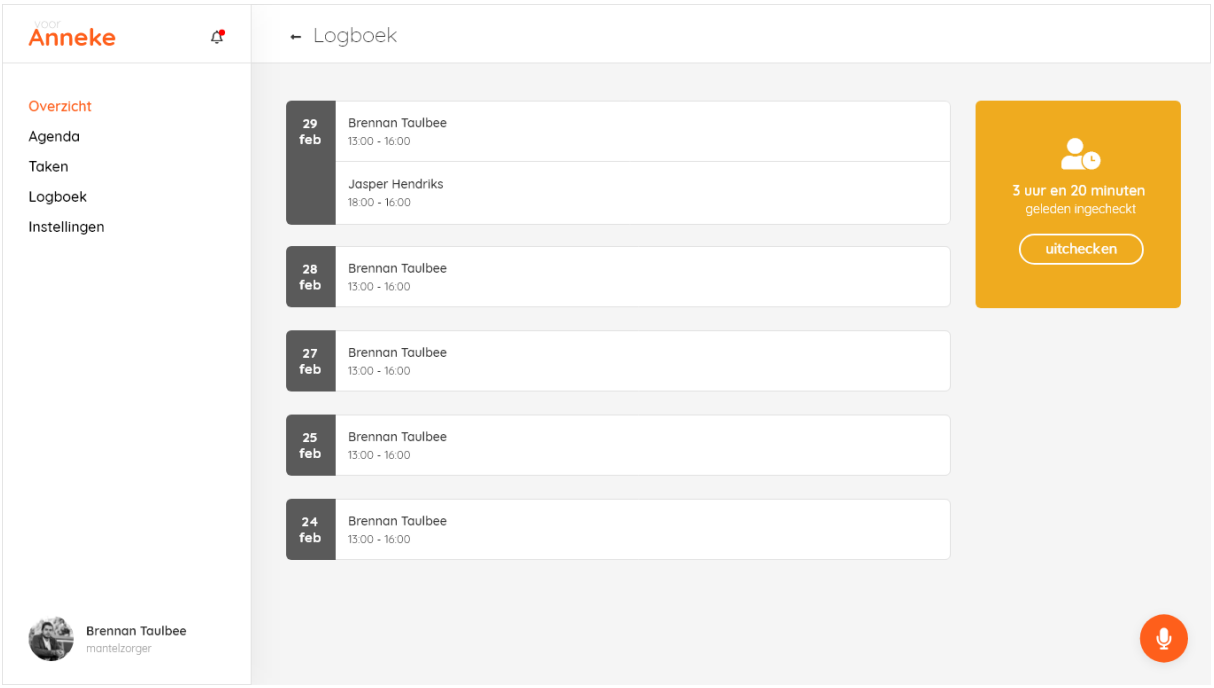
Figuur 5: Bewerken van een afspraak



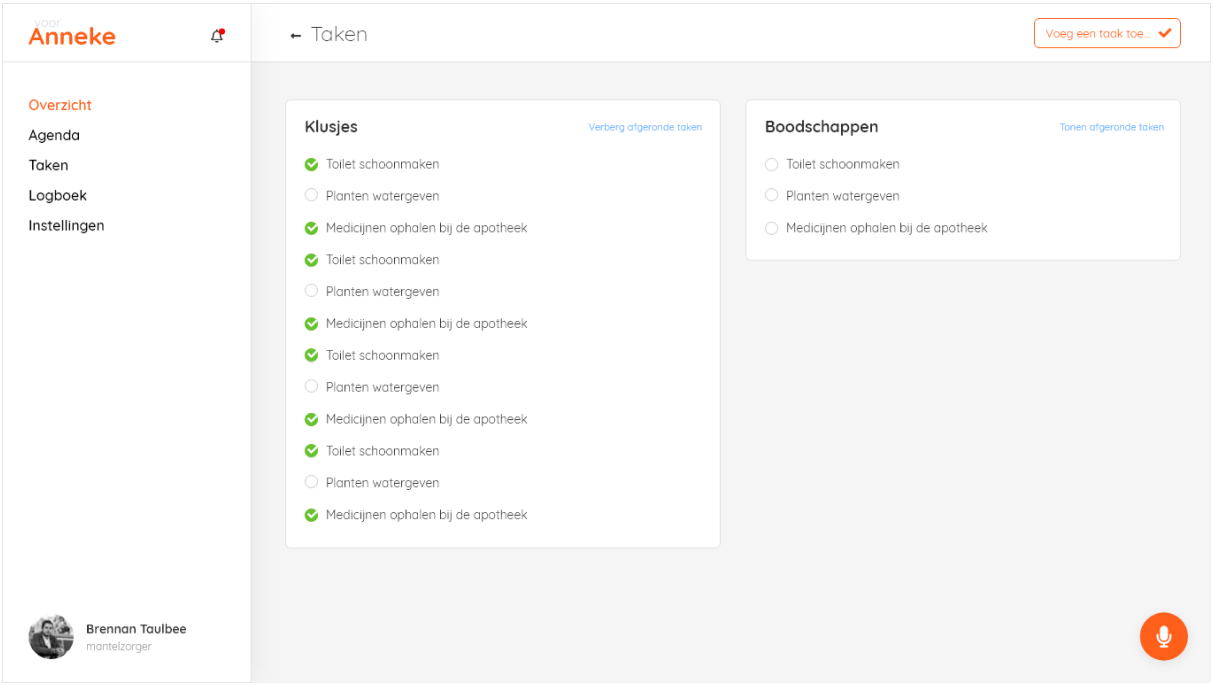
Figuur 6: Communiceren met chatbot



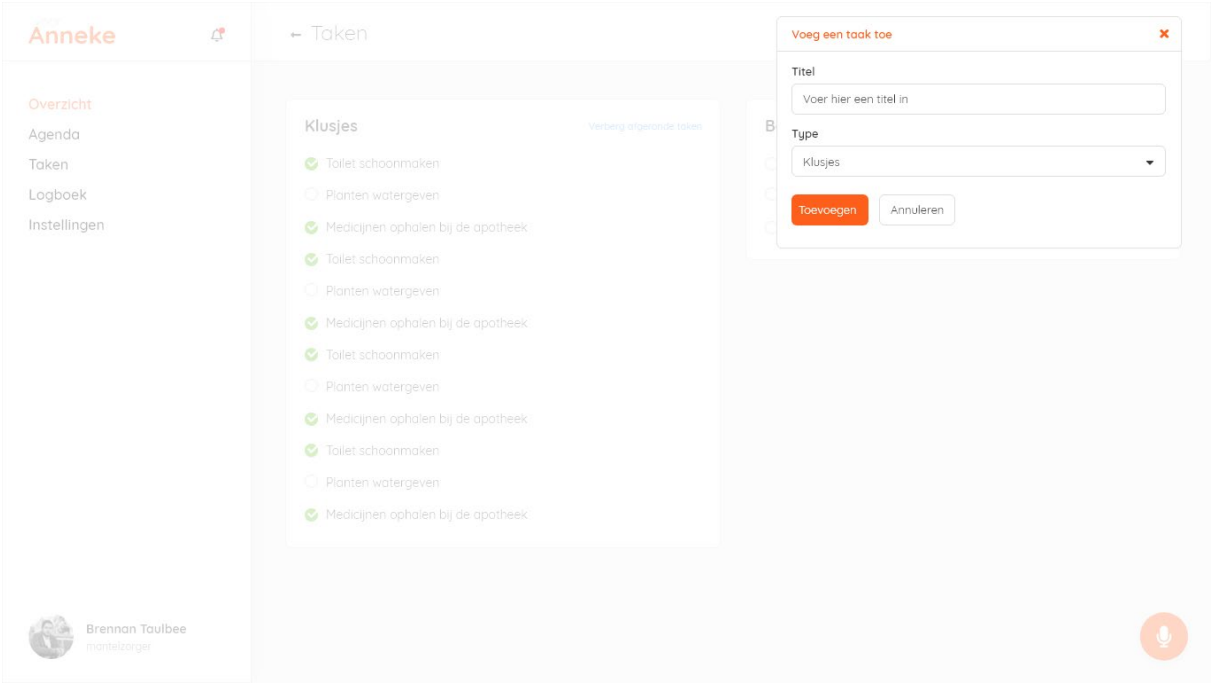
Figuur 7: Communiceren met chatbot vervolg



Figuur 8: Logboek overzicht



Figuur 9: Taken overzicht



Figuur 10: Taken toevoegen

Bijlage 3: User Stories – Sprint 1

In deze bijlage zijn alle opgestelde User Stories en bijbehorende acceptatiecriteria van de eerste sprint opgenomen.

ID	User Story	Acceptatiecriteria	Points
31705	Als ontwikkelaar wil ik een ontwikkelomgeving opzetten, zodat ik de virtuele assistent kan ontwikkelen.		3
32646	Als ontwikkelaar wil ik dat het gedeelde domein (package) automatisch wordt gebuild en gereleaset, zodat wijzigingen niet handmatig overgezet hoeft te worden bij de chatbot en dashboard project.		3
31723	Als mantelzorger wil ik alle afspraken per maand kunnen inzien, zodat ik meer inzichten heb over wat er gaat gebeuren met mijn patiënt	• Er is een agenda pagina aangemaakt en moet toegankelijk zijn via het menu • Het moet mogelijk zijn om afspraken per maand op te vragen • Meerdere afspraken onder hetzelfde datum moeten gebundeld getoond worden (zie ontwerp) • Overzicht afspraken moet responsive zijn (goed weergegeven worden op mobiel, tablet en desktop) • Indien er geen afspraken zijn voor de gekozen maand, een melding tonen dat er geen afspraken zijn.	8
31726	Als mantelzorger wil ik via een kalender het aantal afspraken kunnen inzien, zodat ik in één oogopslag weet hoeveel afspraken er zijn gemaakt in een maand	• Alle afspraken van de gekozen maand moet binnen twee seconden getoond worden • Zodra er in de kalender een dag wordt aangeklikt, dan moeten de afspraken van die maand getoond worden	8
31727	Als mantelzorger wil ik afspraken kunnen inplannen, zodat iedereen die gebruik maakt van het dashboard op de hoogte is van wat er gaat komen	• Het formulier bevat de volgende velden: titel, datum, van, tot, patiënt aanwezig en hele dag. • Verplichte velden: Titel, datum, van, tot	8

		• Indien een verplicht veld niet is ingevuld, melding tonen en de ontbrekende veld rood kleuren	
31729	Als mantelzorger wil ik een afspraak kunnen bewerken, zodat ik makkelijker de gewijzigde afspraken kan doorgeven	• Hetzelfde formulier moet gebruikt worden (zoals bij het toevoegen van een afspraak) • Verplichte velden: Titel, datum, van, tot • Indien je het formulier hebt ingevuld en opslaat moet je de wijzigingen direct zien • Indien een verplicht veld niet wordt ingevuld, melding tonen en veld rood maken • Met annuleren wordt het formulier weer verborgen	8
31730	Als mantelzorger wil ik een afspraak kunnen verwijderen, zodat verkeerde afspraken niet meer worden getoond in mijn agenda	• Na het verwijderen van een afspraak moet het item ook uit het overzicht verdwijnen.	1

Bijlage 4: User Stories – Sprint 2

In deze bijlage zijn alle opgestelde User Stories en bijbehorende acceptatiecriteria van de tweede sprint opgenomen.

ID	User Story	Acceptatiecriteria	Points
31727	Als mantelzorger wil ik afspraken kunnen inplannen, zodat iedereen die gebruik maakt van het dashboard op de hoogte is van wat er gaat komen	• Het formulier bevat de volgende velden: titel, datum, van, tot, patiënt aanwezig en hele dag. • Verplichte velden: Titel, datum, van, tot • Indien een verplicht veld niet is ingevuld, melding tonen en de ontbrekende veld rood kleuren	3
31729	Als mantelzorger wil ik een afspraak kunnen bewerken, zodat ik makkelijker de gewijzigde afspraken kan doorgeven	• Hetzelfde formulier moet gebruikt worden (zoals bij het toevoegen van een afspraak) • Verplichte velden: Titel, datum, van, tot • Indien je het formulier hebt ingevuld en opslaat moet je de wijzigingen direct zien • Indien een verplicht veld niet wordt ingevuld, melding tonen en veld rood maken • Met annuleren wordt het formulier weer verborgen	8
31730	Als mantelzorger wil ik een afspraak kunnen verwijderen, zodat verkeerde afspraken niet meer worden getoond in mijn agenda	• Na het verwijderen van een afspraak moet het item ook uit het overzicht verdwijnen.	1
33162	Als ontwikkelaar wil ik een architectuur met WebSockets, zodat het mogelijk is om real-time wijzigingen te kunnen sturen en ontvangen tussen dashboards	• Er is een connectie aanwezig tussen de API en client • Er kan een bericht gestuurd worden vanuit de server naar de client • Er kan een bericht gestuurd worden vanuit de client naar de server	5
33118	Als mantelzorger wil ik na het maken van een afspraak direct de wijzigingen in de lijst weergaven kunnen zien, zodat ik weet dat de afspraak goed is verwerkt	• Een afspraak kan rechtstreeks via het dashboard of chatbot toegevoegd zijn. Het is de bedoeling dat, ongeacht waar, de nieuwe afspraak wordt toegevoegd aan de lijst.	5

Bijlage 5: User Stories – Sprint 3

In deze bijlage zijn alle opgestelde User Stories en bijbehorende acceptatiecriteria van de derde sprint opgenomen.

ID	User Story	Acceptatiecriteria	Points
33163	Als ontwikkelaar wil ik een architectuur opstellen waarin het mogelijk is om nieuwe wijzigingen in de chatbot-omgeving door te sturen naar het dashboard-omgeving, zodat het mogelijk is beide afgeschermde omgevingen toch met elkaar te laten communiceren	• De chatbot mag niks weten van het dashboard. Hiervoor moet dus een generieke oplossing komen • Er moet een architectuurontwerp aanwezig en beschreven zijn	8
34028	Als ontwikkelaar wil ik dat het dashboard een eigen kanaal wordt van de chatbot, zodat het mogelijk is om met de chatbot te communiceren	• Verbinding tussen dashboard en chatbot (DirectLine API) moet client-side geregeld worden • Er moet succesvol een bericht gestuurd kunnen worden naar de chatbot	8
34012	Als mantelzorger wil ik vanuit het dashboard rechtstreeks kunnen communiceren met de chatbot, zodat ik een vraag en antwoord interactie (via een chat interface) kan hebben zonder in het dashboard te hoeven zoeken naar een antwoord	• Er is een chat interface gemaakt in het dashboard, waarin het mogelijk is om berichten te sturen naar de chatbot • Er moet rekening gehouden worden met meerkeuzevragen van de chatbot	8

Bijlage 6: User Stories – Sprint 4

In deze bijlage zijn alle opgestelde User Stories en bijbehorende acceptatiecriteria van de vierde sprint opgenomen.

ID	User Story	Acceptatiecriteria	Points
35083	Als mantelzorger wil ik een overzicht kunnen zien van alle logberichten die zijn aangemaakt in het verleden, zodat ik op de hoogte ben van de situatie rondom de cliënt	• Log items moeten geordend zijn op aanmaakdatum • Als er geen log items zijn moet er een melding getoond worden dat er geen items beschikbaar zijn	3
35082	Als mantelzorger wil ik mezelf kunnen uitchecken, zodat de chatbot weet wanneer mijn zorgverlenging is beëindigd	• Na het uitchecken moet het totaalaantal uren getoond worden • Na het inchecken moet de timer gereset worden	2
35081	Als mantelzorger wil ik mezelf kunnen inchecken, zodat de chatbot weet wanneer mijn zorgverlening van start is gegaan is	• Er moet constant een timer lopen, zelfs als de mantelzorger op andere pagina's van het dashboard zit	3
35080	Als mantelzorger wil ik alle (dus ook afgeronde) taken binnen een lijst kunnen zien, zodat ik weet wat voor werkzaamheden er in het verleden heeft plaatsgevonden	• Taken moeten op aanmaak datum gesorteerd zijn • De status van de taak moet zichtbaar zijn (actief, afgerond)	3
35079	Als mantelzorger wil ik taken kunnen verwijderen, zodat foutieve taken niet meer in de lijsten staan	• Verwijderde taken moeten realtime getoond worden met andere (actieve) gebruikers	1
35077	Als mantelzorger wil ik taken kunnen afvinken, zodat ik kan bijhouden dat de taak is afgerond	• Aangepaste taken moeten realtime getoond worden met andere (actieve) gebruikers	1
35076	Als mantelzorger wil ik een lijst met taken bijhouden, zodat ik niet vergeet wat voor werkzaamheden ik moet verrichten voor mijn cliënt	• Nieuwe taken moeten realtime getoond worden met andere (actieve) gebruikers • Het moet mogelijk zijn om een taak onder een categorie te plaatsen	8

Bijlage 7: User Stories – Sprint 5

In deze bijlage zijn alle opgestelde User Stories en bijbehorende acceptatiecriteria van de vijfde sprint opgenomen.

ID	User Story	Acceptatiecriteria	Points
35972	Als gebruiker wil ik de huidige actieve gebruiker en rol in het dashboard zien, zodat ik weet wie er op dit moment actief is in het dashboard	• Indien er geen foto is graag een icoon naast de gebruiker tonen • Indien een mantelzorger geen rollen heeft, geen melding geven • Component moet responsive zijn	1
35973	Als gebruiker wil ik kunnen uitloggen, zodat iemand anders in zijn of haar eigen omgeving kan	• Na het uitloggen moet de gebruiker weer naar het aanmeldpagina gaan	1
35971	Als gebruiker wil ik vanuit een lijst van gebruikers mijn eigen account selecteren, zodat ik kan werken in mijn eigen omgeving	• Alleen de voornaam moet getoond worden • Indien er geen foto is graag een icoon naast de gebruiker tonen • Component moet responsive zijn	4
35974	Als gebruiker wil ik dat tijdens het beheren van taken mijn naam aan de taak wordt gekoppeld, zodat iedereen kan zien dat ik de eigenaar ben van het item		2
35975	Als gebruiker wil ik dat tijdens het beheren van afspraken mijn naam aan de afspraak wordt gekoppeld, zodat iedereen kan zien dat ik de eigenaar ben van het item		2
35976	Als gebruiker wil ik dat tijdens het inchecken en uitchecken mijn naam aan het log item wordt gekoppeld, zodat iedereen kan zien dat ik de eigenaar ben van het items		2

Bijlage 8: User Stories – Sprint 6

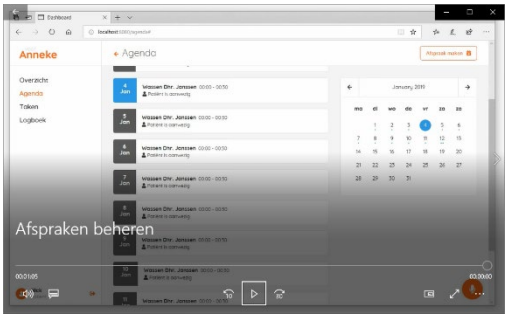
In deze bijlage zijn alle opgestelde User Stories en bijbehorende acceptatiecriteria van de zesde sprint opgenomen.

ID	User Story	Acceptatiecriteria	Points
35083	Als mantelzorger wil ik de afspraken van vandaag kunnen inzien op mijn dashboard, zodat ik weet wat ik moet gaan uitvoeren	• Recente afspraken worden getoond in een widget op het dashboard • Naast de recente afspraken moet ook de datum en tijd getoond worden • Indien er geen afspraken zijn gemaakt, graag een bericht tonen dat er geen afspraken voor vandaag zijn • Indien er meer dan drie afspraken zijn gemaakt, niet meer tonen in de widget, maar een verwijzing plaatsen naar de dag zelf • Verwijzing naar agenda pagina om alle afspraken te bekijken • Widget moet responsive zijn voor mobiel en tablet	6
36936	Als mantelzorger wil ik op het dashboard direct kunnen inchecken, zodat ik het niet vergeet wanneer ik voor het eerst aanmeld op het dashboard	• Widget moet responsive zijn voor mobiel en tablet	3
31722	Als mantelzorger wil ik op het dashboard de actuele taken kunnen inzien, zodat ik direct weet wat ik vandaag moet oppakken	• Recente taken worden getoond in een widget op het dashboard • Indien er geen taken beschikbaar zijn, graag een bericht tonen dat er geen taken voor vandaag zijn • Indien er meer dan drie taken zijn gemaakt, niet meer tonen in de widget, maar een verwijzing plaatsen naar de dag zelf • Widget moet responsive zijn voor mobiel en tablet	3
36938	Als mantelzorger wil ik een ICE-overzicht zien, zodat als er wat misgaat met de cliënt dat ik iemand kan benaderen	• Widget moet responsive zijn voor mobiel en tablet	2

Bijlage 9: Videofragmenten

De focus van het project lag vooral op het front-end. In het ontwikkelproces heb ik mijn best gedaan om het gemaakte ontwerp zo goed mogelijk te vertalen naar code en doormiddel van animaties een betere ervaring te geven aan de bezoeker. Helaas is dit in mijn verslag moeilijk aan te tonen, vandaar dat ik in dit hoofdstuk beknopt alle interessante functionaliteiten benoem met een aansluitend videofragment.

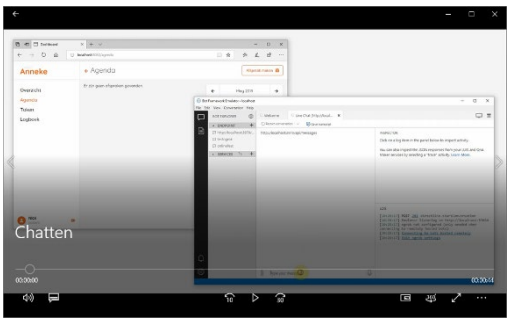
9.1 Afspraak beheren



Een mantelzorger kan in het dashboard afspraken beheren. Zo is het mogelijk om een activiteit op één dag of op meerdere dagen in te plannen. Verder is het mogelijk om via de kalender per maand alle afspraken in te zien.

Klik hier om het videofragment te bekijken

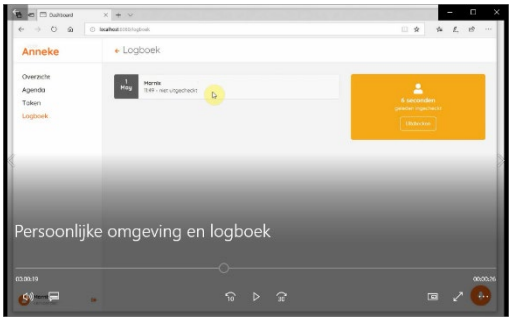
9.2 Communicatie met de chatbot



Het is mogelijk om in het dashboard rechtstreeks met de chatbot te communiceren. In het videofragment wordt getoond hoe er in een externe programma, in dit geval een emulator, een gesprek gestart kan worden met de chatbot. Vervolgens wordt het gesprek ook in het dashboard vervolgd. De aangemaakte afspraak wordt direct in de afspraken lijst weergegeven.

Klik hier om het videofragment te bekijken

9.3 Persoonlijke omgeving en logboek



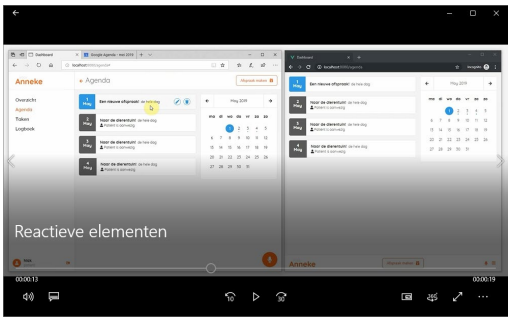
In het dashboard is het mogelijk om ‘in te loggen’ als een gebruiker. De mantelzorger gaat dan naar zijn eigen omgeving. Voor de mantelzorger start met de zorgverlening moet hij of zij inchecken. Via de widget wordt de start tijd geregistreerd en wordt de huidige tijd getoond. Met het logboek kan een archief bijgehouden worden wie en wanneer een mantelzorger de cliënt heeft geholpen.

Klik hier om het videofragment te bekijken

9.4 Reactieve elementen

In het dashboard is het mogelijk om wijziging realtime af te handelen. Nieuwe afspraken die door een mantelzorger worden toegevoegd, worden direct getoond bij andere actieve gebruikers op het dashboard getoond (dus zonder de browser te hoeven vernieuwen).

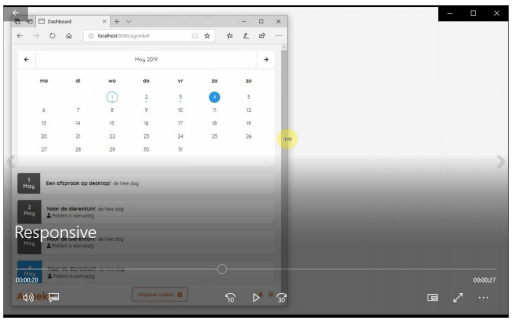
Klik hier om het videofragment te bekijken



9.5 Responsive

Het dashboard is responsive opgezet. Dat wil zeggen dat de applicatie zich aanpast op de grootte van het scherm. Het dashboard is hierdoor toegankelijk op zowel een mobiel, tablet en desktop.

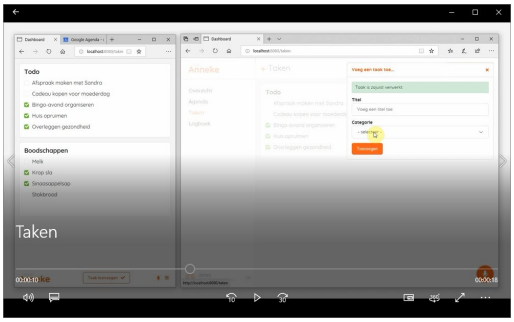
Klik hier om het videofragment te bekijken



9.6 Taken

Een mantelzorger kan taken beheren in het dashboard. In het videofragment is verder ook te zien hoe wijzigingen realtime worden afgehandeld.

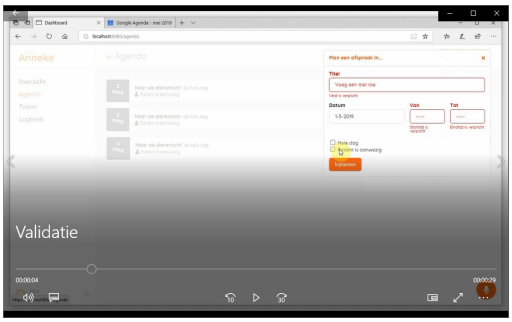
Klik hier om het videofragment te bekijken



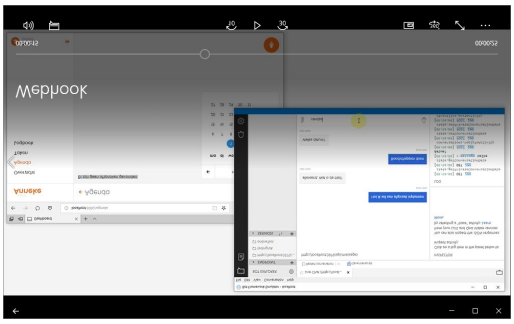
9.7 Validatie

Voor het beheren van afspraken (en overige formulieren op het dashboard) is er validatie toegevoegd.

Klik hier om het videofragment te bekijken



9.8 Webhooks



Het dashboard is pas ‘volledig’ reactief wanneer de chatbot een signaal kan sturen wanneer er wijzigingen heeft plaatsgevonden. In het videofragment wordt via een emulator (externe programma om met de chatbot te kunnen communiceren) een afspraak ingepland. Na het aanmaken van de afspraak ziet u de wijzigingen direct op het dashboard verschijnen. Dit is mogelijk gemaakt door het toevoegen van een webhook in de chatbot-omgeving.

Klik hier om het videofragment te bekijken

Bijlage 10: Interview

Voor het gesprek met de mantelzorger heb ik een aantal vragen voorbereidt. Hieronder treft u de structuur van het interview.

Interview

1. Allereerst, waren er meer betrokkenen die zorgden voor uw cliënt?
2. Zo ja, hoe ging de samenwerking precies?
3. Als er meer betrokkenen waren, was de cliënt zelf ook betrokken in de communicatie met de mantelzorgers?
4. Welke kanaal werd er ingezet om met elkaar te communiceren?
5. Hoe was het overzicht van wat er allemaal speelt voor de cliënt?
6. (Indien deze vraag nog niet is beantwoord) Is er een behoefte bij u ontstaan om de communicatie op een centrale plek te hebben? Waar u, de overige mantelzorgers en zelfs de cliënt op kan samenwerken?
7. Zo ja, wat voor functionaliteiten zou je graag willen zien in zo'n centrale plek?

Dashboard testen

Vertel kort aan de mantelzorger hoe de chatbot is ontstaan. Waarom het dashboard erbij is gekomen en wat voor functionaliteiten het biedt. Vertel verder dat het een prototype is en dat het vooral bedoeld is om later verder uit te werken. Mantelzorger mag nu het dashboard testen.

Afloop

1. Wat is uw ervaring met het dashboard?
2. Wat zou beter kunnen op dit moment?
3. Toen u het in gebruik nam, ontbrak er nog iets?
4. Wilt u verder nog wat kwijt?