Professionele Bachelor Toegepaste Informatica

Afsprakenbeheer vanuit een nieuw perspectief

Kiryl Maltsav

Promotoren:

Tom Schuyten & Stéphane Jacobs — Gift2Give
Sam Vanderstraeten — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Afsprakenbeheer vanuit een nieuw perspectief

Kiryl Maltsav

Promotoren:

Tom Schuyten & Stéphane Jacobs
Gift2Give
Sam Vanderstraeten
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Als eerste wil ik mijn junior-collega, Nathan bedanken om met volle motivatie en inzet te werken aan dit project. Zonder Nathan zou ik nooit het risico genomen hebben om dit project te beginnen.

Vervolgens wil ik onze bedrijfspromotoren Tom Schuyten en Stéphane Jacobs bedanken om ons de mogelijkheid te geven om onder Gift2Give ons project te mogen beginnen. Zij hebben onze droom om Luci Assistant tot leven te brengen werkelijkheid kunnen maken. Ook hebben zij ons advies gegeven en gemotiveerd om Luci blijven uit te werken.

Als laatste wil ik Sam Vanderstraeten bedanken voor de inzet die hij heeft geleverd met de ondersteuning en verbetering. Sam is de grootste factor geweest dat dit eindwerk, in mijn ogen tot een succesvol resultaat heeft gemaakt.

Niet te vergeten wil ik ook mijn ouders bedanken om mij te steunen doorheen de jaren. Het was niet altijd even gemakkelijk maar zij hebben mij de mogelijkheid gegeven om deze studie te kunnen doen.

Abstract

Luci Assistant is een student start-up in de IT-sector die ervoor wil zorgen dat online afspraken maken de standaard wordt in de dienstensector. Het is op de dag van vandaag nog zeer gebruikelijk om dienstverleners (kapper, tandarts, kinesist enzovoort) te contacteren via verschillende kanalen, zoals gsm, e-mail of *social media*.

Omdat er zo veel verschillende communicatiekanalen bestaan, wordt er veel tijd verspild. Hiervoor heeft Luci Assistant een innovatieve oplossing waarbij een generisch platform aangeboden wordt aan zowel de klanten als aan de dienstverleners. Dit geeft hun de mogelijkheid om op een makkelijke en efficiënte manier hun afspraken te maken en te beheren.

Het *minimum viable product* bestaat uit een mobiele applicatie voor de klanten waarmee ze afspraken kunnen maken, documenten kunnen uitwisselen en nieuwe dienstverleners kunnen vinden. Vervolgens wordt er ook een dashboard voorzien voor de dienstverleners. Met deze applicatie kunnen afspraken en andere bedrijfsprocessen volledig beheerd worden.

Er wordt gebruik gemaakt van verschillende technologieën om de MVP te ontwikkelen.

- Angular (dashboard)
- NativeScript (mobiele app)
- Java Spring Boot (backend)
- MongoDB (database)
- Pusher (real-time data)
- Heroku (DevOps, CI/CD)
- Firebase (authenticatie)
- Github (versiebeheer)

Tijdens het onderzoek wordt er nagegaan welk framework het best gebruikt wordt om een mobiele applicatie te ontwikkelen voor Luci Assistant. De meest voor de hand liggende optie is NativeScript. Dit is een framework om *cross platform apps* te ontwikkelen in JavaScript. Maar aan de andere kant is er een nieuwe en opkomende technologie (Progressieve Web App) die gebruik maakt van *service workers* om zo een betere mobiele ervaring aan te bieden.

Ook wordt er tijdens het onderzoek een *proof of concept* ontwikkeld in Angular PWA en Angular NativeScript. Verder wordt het hele proces gedocumenteerd over hoe de ontwikkeling verloopt, wat er goed gaat, wat de valkuilen zijn enzovoort. Ook wordt er een vergelijkingsmatrix opgesteld van de verschillende functies die wel en niet mogelijk zijn met Native Script en/of PWA.

Het doel is om met de resultaten van het onderzoek te bepalen welke technologie de beste is voor dit project. De gekozen technologie wordt volledig uitgewerkt en geïmplementeerd in de MVP.

Luci assistant

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

Tabel 1 Gebruikte afkortingen

PWA	Progressive Web App	PWAs zijn websites gebouwd met web technologie die zich gedragen als een app.
OS	Besturingssysteem	Het besturingssysteem is het hoofdprogramma dat geïnstalleerd staat op je computer of smartphone.
CSS	Cascading style sheet	CSS is bedoeld voor het opmaken en vormgeven van HTML-pagina's oftewel uw website.
UI	Userinterface	Een grafisch vormgegeven schil waarmee gebruikers en apparaten met elkaar kunnen communiceren
UX	User Experience	Dit is het gevoel dat een bezoeker meekrijgt bij het bezoeken van bijvoorbeeld een website of app.
CI	Continuous integration	Het is de gewoonte om verschillende werkkopieën meerdere malen per dag samen te voegen tot een gedeelde hoofdlijn.
CD	Continuous Delivery	Een software-engineeringaanpak waarbij teams in korte cycli software produceren, zodat de software op elk moment betrouwbaar kan worden vrijgegeven en, wanneer de software wordt vrijgegeven, dit automatisch gedaan wordt.
MVP	Minimum viable product	Een MVP is een vroege, uitgeklede versie van een product, waarmee bepaald wordt of dat product rendabel is.
DevOps	Samentrekking van "development" en "operations	Een praktijk binnen software engineering die tot doel heeft softwareontwikkeling (Dev) en softwareoperaties (Ops) samen te brengen.
API	Application programming interface	Een API is een set aan definities waarmee softwareprogramma's onderling kunnen communiceren.
Git	/	Een open source gedistribueerd versiebeheersysteem.
REST	Representational state transfer	Een software-architectuur voor gedistribueerde mediasystemen zoals het wereldwijde web.
NoSql	Not Only SQL	Een database die mechanisme biedt voor het opslaan en terugvinden van gegevens die zijn gemodelleerd in andere middelen dan de tabellarische relaties die worden gebruikt in relationele databases.
HTML	Hypertext Markup Language	Een taal die gebruikt wordt voor de opmaak van websites.
HTTPS	Hypertext Transfer Protocol Secure	Een technologie voor beveiligde communicatie via het internet.
CORS	Cross-Origin Resource Sharing	Een mechanisme waarmee beperkte bronnen op een webpagina kunnen worden opgevraagd bij een ander domein buiten het domein van waaruit de eerste bron is aangeboden.

Inleiding

Een innoverend idee, dat dienstverleners helpt om afspraken te beheren, tot werkelijkheid maken. Dit eindwerk laat zien wat er komt kijken bij de ontwikkeling van een volledig functionerend, productieklaar, online platform.

Als eerste zal er besproken worden hoe de analyse verlopen is. Vervolgens bekijken we de kleine maar belangrijke aspecten voor het ontwerpen van de mock-ups. Aan de hand van de mock-ups worden de applicaties ontwikkeld en terwijl wordt er een omgeving opzet, waar de ontwikkelde applicaties getest en gereleaset kunnen worden naar een productie omgeving.

Als laatste wordt er bekeken wat tegenwoordig de beste oplossing is om een cross-platform mobiele applicatie te ontwikkelen. Dit zal gedaan worden door een onderzoek dat de vergelijking maakt tussen een Progressive Web App en een JavaScript framework genaamd NativeScript. Aan de hand van de conclusie wordt de gekozen technologie gebruikt om de mobiele applicatie van Luci Assistant te ontwikkelen.

I. Stageverslag

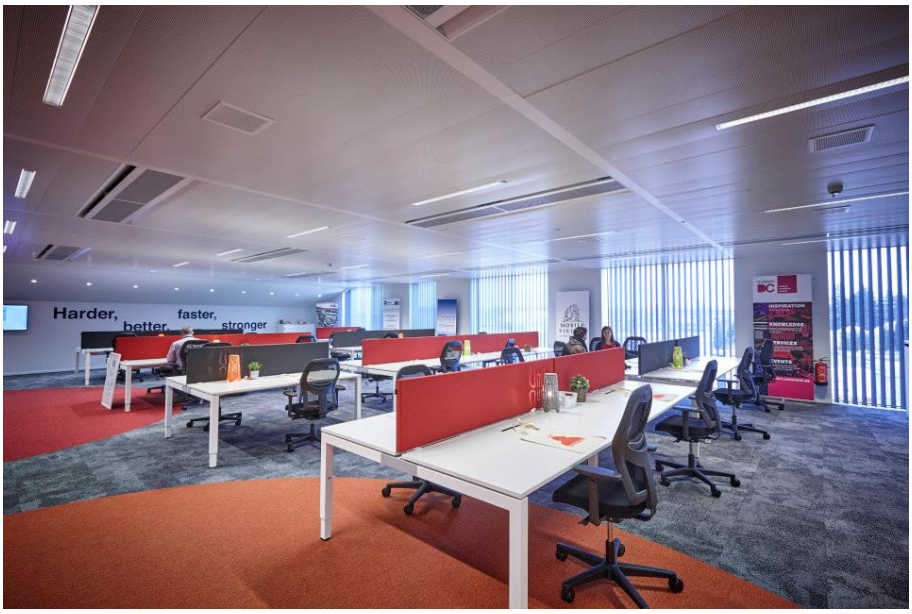
1 Bedrijfsvoorstelling

1.1 Situering

Luci Assistant is een startup in de IT-sector. Momenteel bestaat het bedrijf uit twee PXL studenten Toegepaste Informatica. Het bedrijf heeft als doel om een innovatieve toepassing te ontwikkelen voor de Belgische markt en wil in de nabije toekomst uitbreiden om een voet tussen de deur te plaatsen in Europa.

Figuur 1 Corda Campus

De stage vond plaats op de Corda INCubator waar Luci Assistant zich in de eerste fase gevestigd heeft. De incubator is een super initiatief om starters te helpen hun droom waar te maken. Er worden voordelige werkplaatsen aangeboden in een zeer professionele maar toch toffe werkomgeving. Ook biedt de incubator vergaderzalen aan om meetings te houden met collega's en/of klanten.



Figuur 2 Corda INCubator werkruimte

1.2 Unique selling points

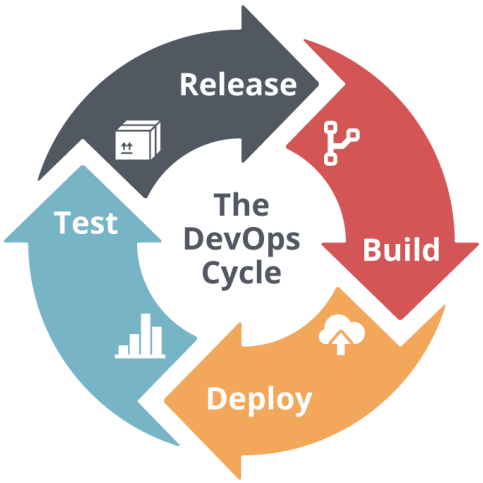
De manier waarop Luci Assistant zich wil distantiëren van de concurrentie is om nadruk te leggen op een perfecte *User Experience.*

De focus ligt op het aanbieden van een unieke en innoverende ervaring aan de klanten. Het doel is dat de gebruikers zich afvragen waarom er nog nooit eerder een dergelijke applicatie ontwikkeld was, die hun het leven werkelijk beter en makkelijker maakt.

Ook gelooft Luci Assistant in het aanbieden van een product dat voortdurend beter en beter wordt. Er wordt daarom veel aandacht besteed aan DevOps, waarbij *Continious Integration* (CI) en *Continious Deployment* (CD) de kern zijn.

Het product biedt een *Software as a service* (SaaS) solution, hiervoor zijn web technologieën zeer belangrijk. Bij Luci is er een IT-cultuur die zich focust op full-stack ontwikkeling. De belangrijkste technologieën zijn Angular, Spring en MongoDB.

Natuurlijk is een product pas perfect als er gegarandeerd kan worden dat de applicaties foutloos en 24/7 beschikbaar zijn. Daarvoor wordt er nadruk gelegd op het testen van de applicaties in zowel de frontend maar ook de backend.



Figuur 3 DevOps cycle

1.3 Beschrijving IT-afdeling

De startup bestaat uit twee oprichters. Beide zijn ze gespecialiseerd in applicatieontwikkeling. Het bedrijf wil in de volgende fases uitbreiden door experts in verschillende vakgebieden aan te nemen. Deze kunnen de bestaande applicaties verbeteren en naar een hoger niveau brengen zodat het bedrijf voorbereid is op de toekomst.

2 Voorstelling stageopdracht

2.1 Probleemstelling

2.1.1 Situering

Luci Assistant wil het beheer van afspraken tussen klant en onderneming vergemakkelijken. Het ultieme doel is om de ervaring van zowel de klant als het bedrijf te verbeteren. Zo is er voor beide een toepassing voorzien om hun afspraken te beheren.

Het aangeboden platform is in tegenstelling tot vergelijkbare tools (die vooral gericht zijn op de bedrijven zelf en niet de consumenten) generisch. Dat betekent dat het toepasbaar is op verschillende industrieën. De consument heeft slechts één toepassing nodig om al zijn/haar praktijken te beheren. In tegenstelling tot meerdere verschillende toepassingen. De tool geeft de bedrijven meer *exposure*, en beginnende praktijken besparen vele kosten op vlak van personeel en management. Dit door een bijna volledig geautomatiseerde tool waar gebruiksvriendelijkheid op de eerste plaats staat.

Bovendien kan de consument ook zijn openstaande rekeningen en tal van andere features raadplegen via deze tool. Zo kan er heel wat papierwerk omzeild worden. Op dit ogenblik bestaan er toepassingen die specifiek zijn aan een industrie, maar Luci Assistant wil ervoor zorgen dat de dienstverleners overschakelen naar een generisch platform.

2.1.2 Achtergrondinformatie

Het concept van de bestaande toepassingen is niet verkeerd, toch valt het op dat er vaak iets ontbreekt**.** Bij Luci Assistant is de focus gericht op de executie. Daarom ligt de nadruk op de gebruiksvriendelijkheid en de ervaring voor de klanten. De toepassing moet overzichtelijk en makkelijk te gebruiken zijn.

De voordelen van Luci Assistant zijn:

- Het bedrijf meer exposure geven
- Het automatiseren van bedrijfsprocessen
- Het toelaten om reviews te schrijven
- Het delen van documenten met de klant

Eén van de grote voordelen is dat er initieel geen noodzaak is om rechtstreeks de consumenten over te halen om de toepassing te gebruiken (netwerkeffect). De toepassing stuurt nog steeds de traditionele sms- en email notificaties. Het doel is dat de praktijken merken dat het gehele proces beter wordt als de klanten gebruik maken van de mobiele applicatie.

Ook blijkt uit een marktonderzoek dat de meeste tools die gebruikt worden door praktijken op de dag van vandaag verouderd zijn.

2.1.3 Stakeholders

Het product dat ontwikkeld wordt, is voornamelijk een B2B oplossing. De stakeholders zijn de dienstverleners die afspraken gebruiken om hun dienst te verlenen. Omdat Luci Assistant een mobiele applicatie voorziet voor de consumenten, zijn deze in de toekomst ook een stakeholder.

Ten slotte zijn de uitvoerders van de stageopdracht stakeholders, als het project succesvol is dan kunnen zij dit verder uitwerken.

2.2 Doelstellingen

Het doel van dit project is om een Minimum Viable Product (MPV), te ontwikkelen. Deze bestaat uit een applicatie voor zowel klanten als dienstverleners. Tijdens het onderzoek wordt er nagegaan welke technologie het best gebruikt wordt voor de ontwikkeling van de mobiele applicatie van Luci Assistant.

De voorkeur voor Angular als framework komt uit vorige ervaringen die hebben laten zien dat het een sterk platform is om mee te werken. Angular ondersteunt ook enkel cross-platform technologieën. Specifiek wordt tijdens het onderzoek een vergelijking gemaakt tussen NativeScript en Progressive Web Apps (PWA).

NativeScript is een vrij bekende speler en biedt enorm veel functionaliteiten aan. PWA daarentegen is een nieuwe speler die gebruik maakt van *service workers*. Deze *service workers* helpen om de app extra functies te voorzien zoals offline gebruik, *data caching*, mogelijkheid tot *fullscreen* applicaties, enz.

Met de resultaten van het onderzoek wordt er beslist welke technologie het beste is voor dit project. Deze wordt dan ook gebruikt om de MVP te ontwikkelen. Het doel van het onderzoek is om na te gaan of de technologie de belangrijkste features succesvol kunnen aanbieden. Enkele van deze features zijn het maken van compleet geautomatiseerde afspraken, deze afspraken beheren, zoeken en vinden van nieuwe dienstverleners, documenten delen tussen klant en bedrijf enz. Toekomstgericht moet er ook een goede integratie zijn met bestaande agenda's.

Het doel wordt verwezenlijkt door gebruik te maken van agile development. Als tweede is er de keuze gemaakt om gebruik te maken van DevOps. Dit biedt de mogelijkheid om bij elke aanpassing het project meteen te *deployen* op een productieserver. Hierdoor is het programma meteen beschikbaar voor de gebruikers. Als dit bij aanvang goed verloopt is er de mogelijkheid om de applicatie aan het einde van de stage vlekkeloos te lanceren op de markt.

2.3 Omgeving

2.3.1 Technologieën

Er worden verschillende technologieën gebruikt om een *production ready* MVP te ontwikkelen.

- Angular (Dashboard en mobiele applicatie)
- Java Spring Boot (backend)
- MongoDB (database)
- Pusher (real-time data en events)
- Google Cloud (DevOps, CI/CD)
- Firebase (authenticatie)

2.3.2 Tools

De tools die gebruikt worden tijdens de stage.

- VS Code (Typescript/ Angular)
- Intellij (Java)
- Mongo Compass (Database beheer)
- Google Cloud Console (DevOps)
- Heroku (DevOps)
- Adobe Xd (Design mockups)
- Github (version control)

2.3.3 Diensten

- Agenda
  - Zowel klanten als bedrijven kunnen op ieder moment een afspraken maken, raadplegen en beheren via een easy-to-use applicatie.
- Analytics
  - Bedrijven kunnen op ieder moment a.d.h.v. grafieken hun groei/ontwikkeling meten en raadplegen.
- Exposure en Marketing
  - Het platform biedt bedrijven de kans om in de “spotlight” te komen waardoor ze *leads* kunnen binnenhalen.
  - De consumenten hebben een overzicht van alle bedrijven per categorie en op basis van hun locatie.

3 Uitwerking stageopdracht

De uitwerking van de stageopdracht bestond uit meerdere fases. Het eindresultaat bestaat uit een dashboard voor de bedrijven en een mobiele applicatie voor de klanten. Beide communiceren ze met een gezamenlijke backend REST API. Voordien is er een globale analyse uitgevoerd om te bepalen welke features de MVP moest bevatten. Verder wordt er een grondige technische analyse uitgevoerd om de technologieën en architectuur te bepalen.

De flow van het ontwikkelingsproces bestaat uit de volgende punten:

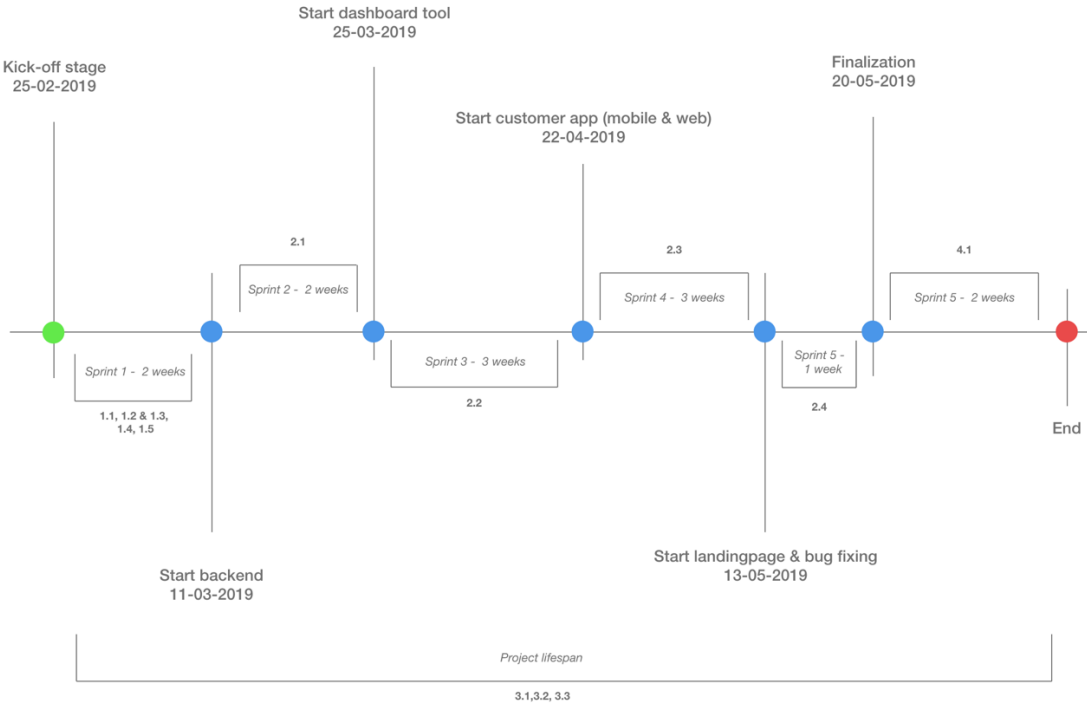
- Uitvoeren van een technische analyse
- Ontwerpen van mockups
- Aanmaken van tickets in Trello voor mobiele app en dashboard
- Opzetten DevOps
- Ontwikkelen van de backend
- Ontwikkelen van het dashboard
- Ontwikkelen van de mobiele applicatie
- Ontwikkelen van een info site en portal
- Finaliseren van de MVP

Luci

1. Analyse
2. Development
3. Research & Bachelorproject
4. Finalization

Milestone	Description	Est. Time *(days)*
1.1	Bespreking/analyse over alle technologieën die we gebruiken in ons project	5
1.2	Mockup's dashboard	2
1.3	Mockup's customer app	2
1.4	Technische analyse	7
1.5	Algemene afspraken, Devops, Agile	1
2.1	Ontwikkeling backend	14
2.2	Ontwikkeling dashboard tool	21
2.3	Ontwikkeling klanten toepassing (mobile & web)	21
2.4	Ontwikkeling landing page	7
3.1	Portfolio	*Project lifespan*
3.2	Projectomschrijving	
3.2	Eindwerk	
4.1	Finalization	14

Timeline



Figuur 4 Geschatte tijdplanning

3.1 Analyse

3.1.1 Scope

Het doel van deze stage is om een MVP te ontwikkelen die productieklaar is en gebruikt zou kunnen worden door bedrijven. Omdat het een groot project is, er maar twaalf weken beschikbaar zijn en er maar twee ontwikkelaars aan werken, moet er een duidelijke scope ingesteld worden. Daarom is er besloten om een grondige analyse te doen naar welke features het belangrijkste zijn om de kans op slagen zo hoog mogelijk te maken. De features die zeker in de MVP moeten zitten zijn:

- Mobiele applicatie
    - Beheren van afspraken
    - Zoeken naar bedrijven
    - Plaatsen van reviews
    - Beheren van persoonlijk profiel
- Dashboard
    - Beheren van afspraken
    - Wijzigen van bedrijfsinstelling
    - Beheren van klantenbestand
    - Uploaden van documenten

3.1.2 Mockups en prototype

Aan de hand van de vereiste features kan er een eerste versie gemaakt worden van hoe de applicaties er uit gaan zien. De tool die gebruikt wordt om de mockups te ontwerpen is Adobe XD. Dit is een gratis tool die enorm veel functionaliteit aanbiedt om *pixel perfect* te werken. Ook geeft deze de mogelijkheid om ontwerpen te maken voor verschillende resoluties. Hierdoor kan er voor elke schermgrootte de juiste lay-out gemaakt worden.

Deze tool is een cruciaal aspect ideeën tot leven te brengen. Het is belangrijk om een idee snel te kunnen ontwerpen en dan stap voor stap verder in detail uit te kunnen werken.

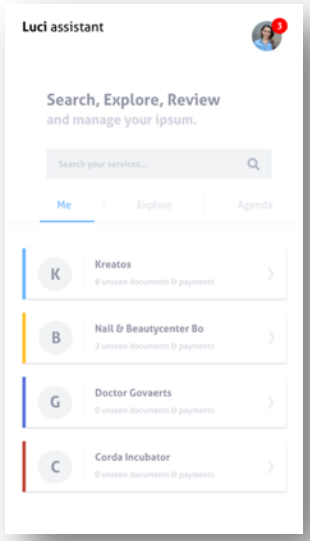
Nog een handige feature die Adobe XD biedt is de mogelijkheid om prototypes te maken. Door middel van de prototype functie kunnen de verschillende views gekoppeld worden door de knoppen een actie te geven. Na het verbinden van de views is er de mogelijkheid om een live preview te tonen op een smartphone. Dit zorgt ervoor dat de look en feel getest kan worden. Als er iets niet goed aanvoelt, kunnen er onmiddellijk aanpassingen doorgevoerd worden om een bepaalde lay-out en/of knop aan te passen.

3.1.2.1 Designs mobiele applicatie

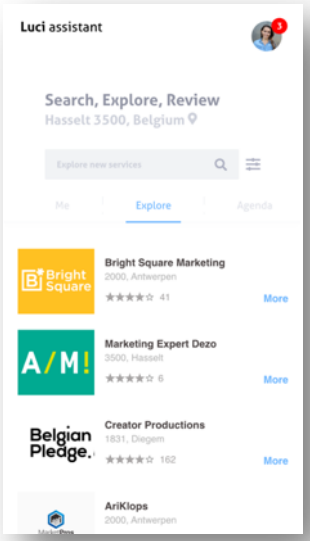
Voor de designs van de mobiele applicatie was er een visie, een heel overzichtelijke en simpele UI die tegelijk enorm veel functionaliteit aanbiedt. De gebruiker moet de applicatie meteen begrijpen. Voor de designs is er veel inspiratie gehaald uit hedendaagse applicaties zoals FB Messenger, Appstore, AirBnb, TakeAway enz.

Deze moderne applicaties hebben enkele dingen gemeen. Een voorbeeld is dat er weinig gebruik gemaakt wordt van borders. De meeste onderdelen zijn los van elkaar en worden enkel gescheiden door een open ruimte. Wel is het belangrijk dat de elementen die samen horen goed gegroepeerd en uitgelijnd worden.

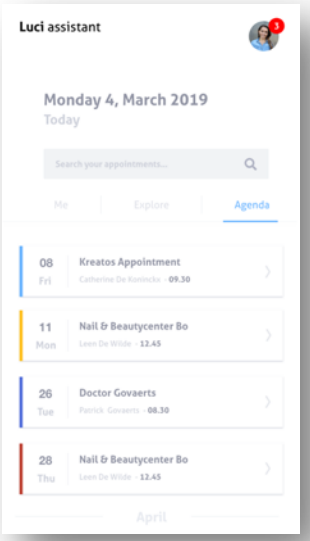
Het hoofdscherm bevat de belangrijkste features voor de gebruiker, zoals het zien van zijn afspraken, zijn opgeslagen bedrijven, maar ook kan de gebruiker hier meteen op zoek gaan nieuwe bedrijven.



Figuur 5 App Home - Me



Figuur 6 App Home - Explore



Figuur 7 App Home - Agenda

Het ontwerpen van een goede UI die tegelijkertijd een goede UX aanbiedt is geen makkelijke opdracht. Elke knop, zoekbalk, foto, enz. heeft een bepaalde functie. Een valkuil is dat er vaak te veel functionaliteit verwerkt wordt in een view. Hierdoor kan een applicatie snel onoverzichtelijk worden voor de gebruikers.

3.1.2.2 Designs dashboard

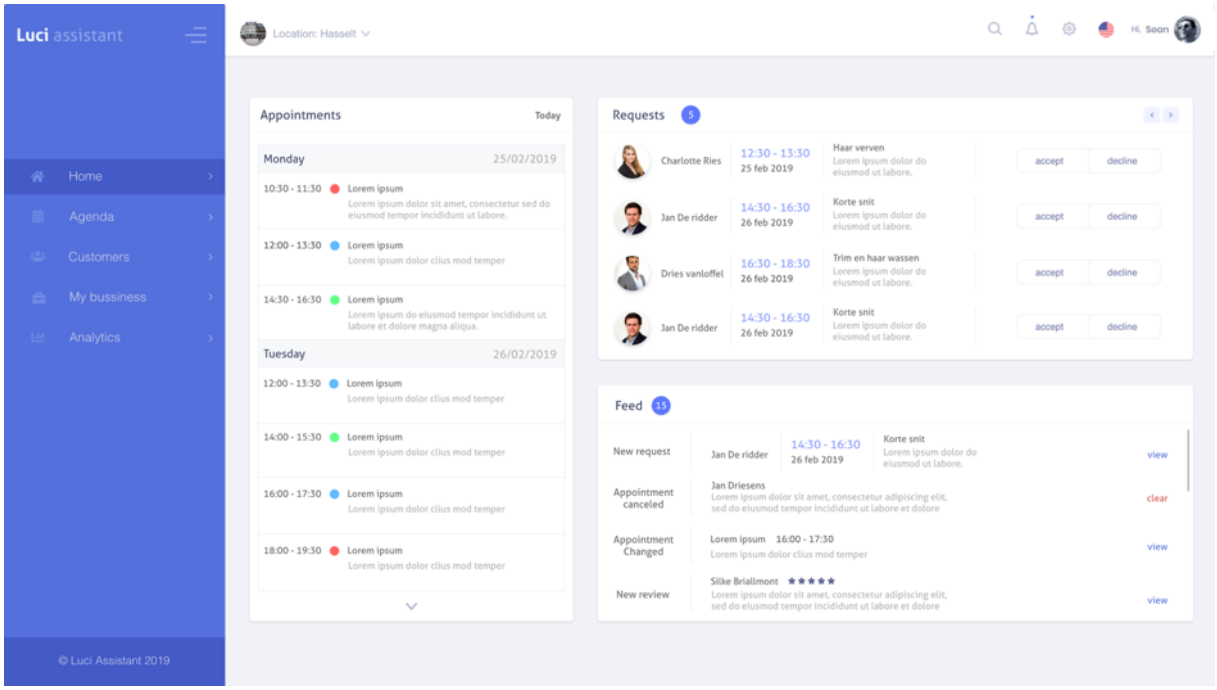
Het dashboard dat gebruikt wordt door de dienstverleners moet er professioneel uitzien. Tegelijk moet de gebruiker snel kunnen navigeren tussen de verschillende views. Er is veel inspiratie opgedaan uit bestaande dashboards.

Het is belangrijk dat de gebruiker een goed onderscheid heeft tussen de gegevens en instellingen die hij vaak gebruikt, en die hij minder vaak nodig heeft. Zo moet een gebruiker in elke view de mogelijkheid hebben om te zoeken naar klanten en afspraken. Ook is ervoor gezorgd dat de gebruiker snel en makkelijk een nieuwe klant of afspraak kan aanmaken vanuit de navigatiebalk.

Op figuur 9 wordt er een voorbeeld getoond van hoe een bedrijf zijn home view er uit kan zien. Deze bevat een lijst met kort samengevatte info van zijn eerstkomende afspraken.

Verder kan de gebruiker binnenkomende afspraken inplannen of afwijzen. Dit wordt later in meer detail besproken.

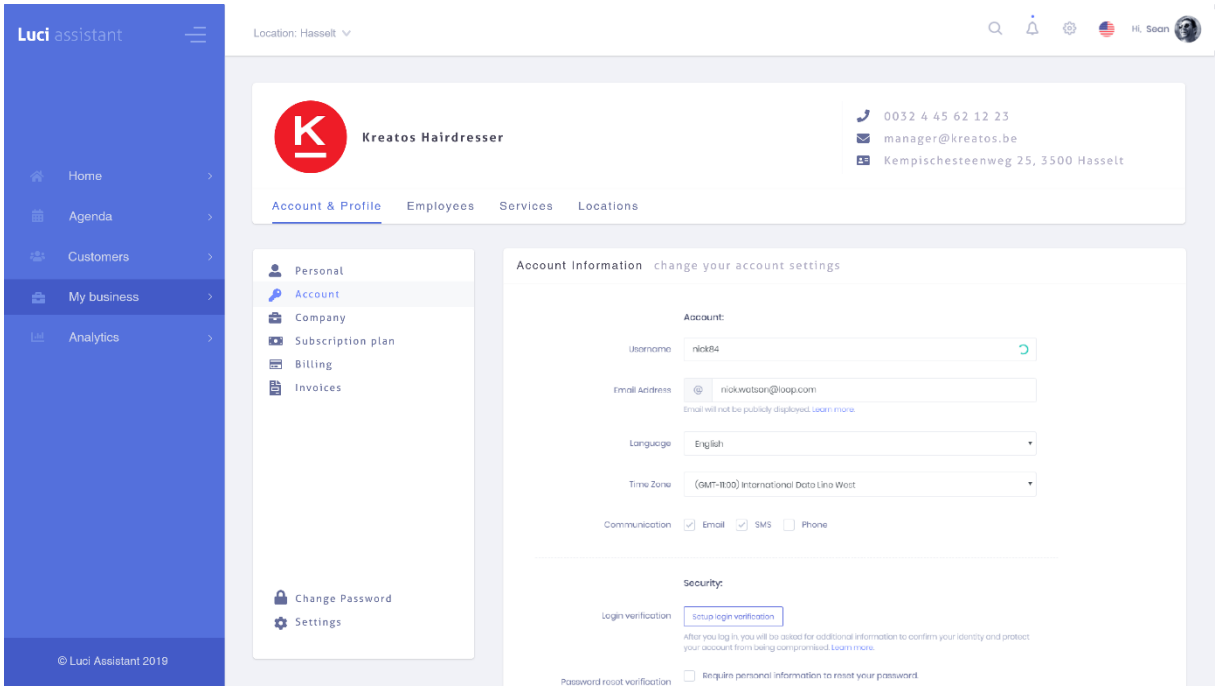
Als laatste heeft de gebruiker een *live feed* van alle binnenkomende events zoals een nieuwe review, wijziging van een afspraak, enz.



Figuur 8 Dashboard Home

De *My Business view* (figuur 10) geeft de gebruiker de mogelijkheid om zijn bedrijf te beheren. Hier kan de gebruiker zijn profiel en accountgegevens aanpassen, werknemers beheren, diensten toevoegen, reviews lezen, enz.

Het instellen van de dienst van een bedrijf moet maar één keer gebeuren, daarom zijn deze instellingen verder weg gestoken. Dit verbetert de UX van de gebruiker omdat deze niet in de weg zitten bij het dagelijks gebruik van de tool.



Figuur 9 Dashboard My Bussiness

3.2 DevOps

Een belangrijk aspect van het ontwikkelproces, is het opvolgen ervan. Een Agile manier van werken is op de dag van vandaag een noodzaak om een project in goede banen te leiden. Hiervoor zijn de juiste tools een belangrijke factor. Om zo veel mogelijk tijd te besparen is er gebruik gemaakt van simpele applicaties zoals Trello.

3.2.1 Projectmanagement

Trello is een tracking tool om issues aan te maken en op te volgen. Ook is er aandacht besteed om alle vereisten uit te schrijven zodat deze tijdens de ontwikkeling gecontroleerd en opgevolgd kunnen worden.

3.2.2 Communicatie

De stage vindt zich plaats op de INCubator, hierdoor kan de communicatie vlot verlopen. Toch is het handig om een communicatie tool te hebben om snel bestanden, informatie, enz. te kunnen delen. Hiervoor wordt er Slack gebruikt, dit is een van de meest gebruikte communicatietools in de ICT-wereld.

3.2.3 Continious Integration (CI)/ Continious Deployment (CD)

Sinds de afgelopen jaren is er een opmars in de IT-wereld op vlak van updates. Vroeger werden alle kleine aanpassingen verzameld in één grote update, die vervolgens handmatig werd opgestuurd naar de productieservers. Dit was vroeger geen enkel probleem, maar op de dag van vandaag gaat de technologie zo snel vooruit, dat updates nu wekelijks of zelf dagelijks doorgevoerd worden.

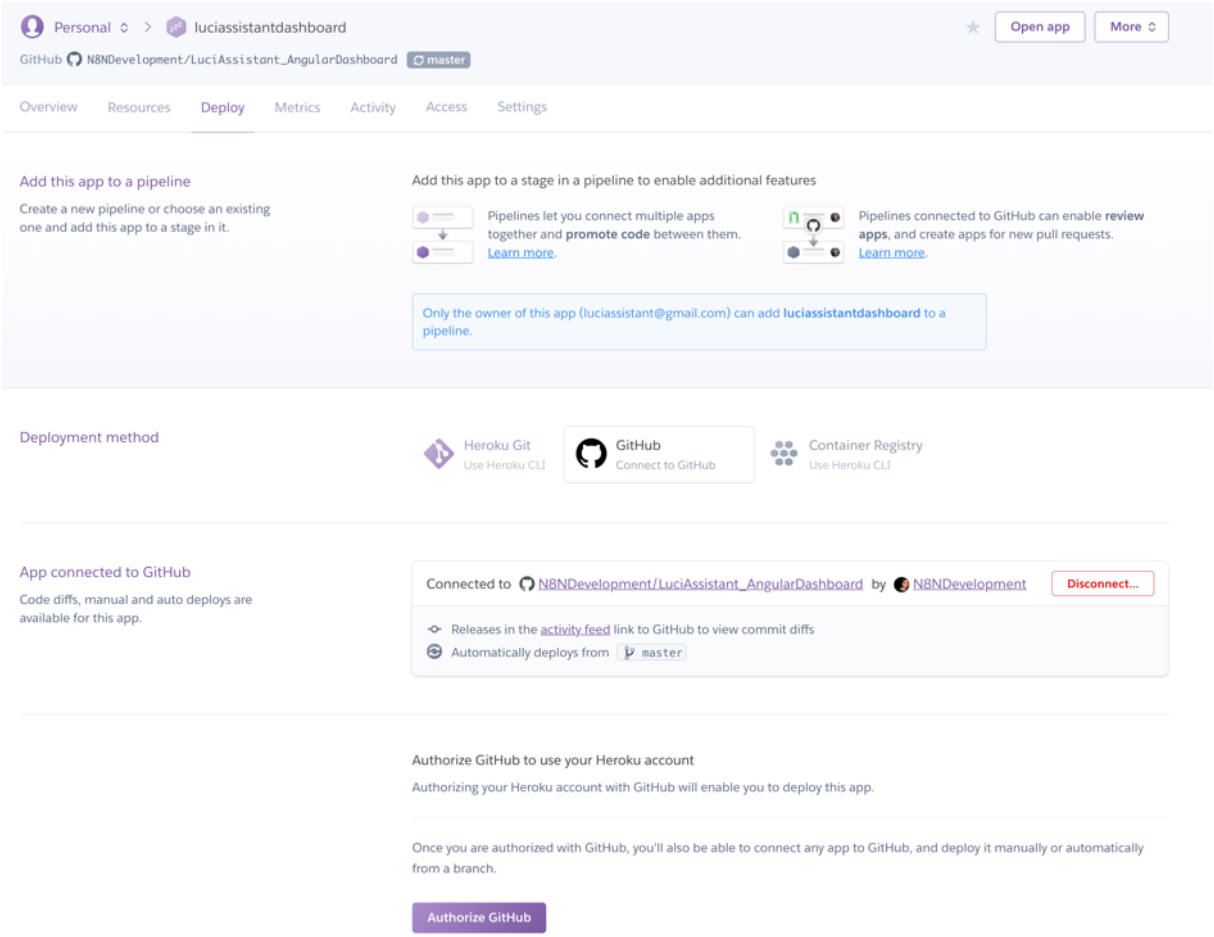
Om dit probleem op te lossen is er CI en CD in het leven geroepen. Dit zorgt er voor dat elke aanpassing onmiddellijk en automatisch toegevoegd kan worden aan de productieversie. Natuurlijk zijn er vele checks die ervoor zorgen dat het allemaal getest is en geen fouten bevat.

Voor het opleveren van een *production ready* applicatie, wordt er ook een dergelijke tool gebruikt. Na een kort onderzoek is er besloten om Heroku te gebruiken.

Heroku is een bekende SaaS, die ervoor zorgt dat bij elke trigger op een bepaalde “git branch”, de code automatisch geüpload wordt naar een productieserver.

De setup van Heroku is makkelijk en kan een bestaande applicatie binnen de 15 minuten draaiend krijgen. De stappen bestaan uit:

- Aanmaken van een *pipeline* (optioneel)
- Instellen van GitHub
    - Autoriseren van het account
    - Kiezen van project
    - Bepalen op welke *branch* de trigger zit



Figuur 10 Deployment configuratie van Heroku

3.3 Ontwikkeling

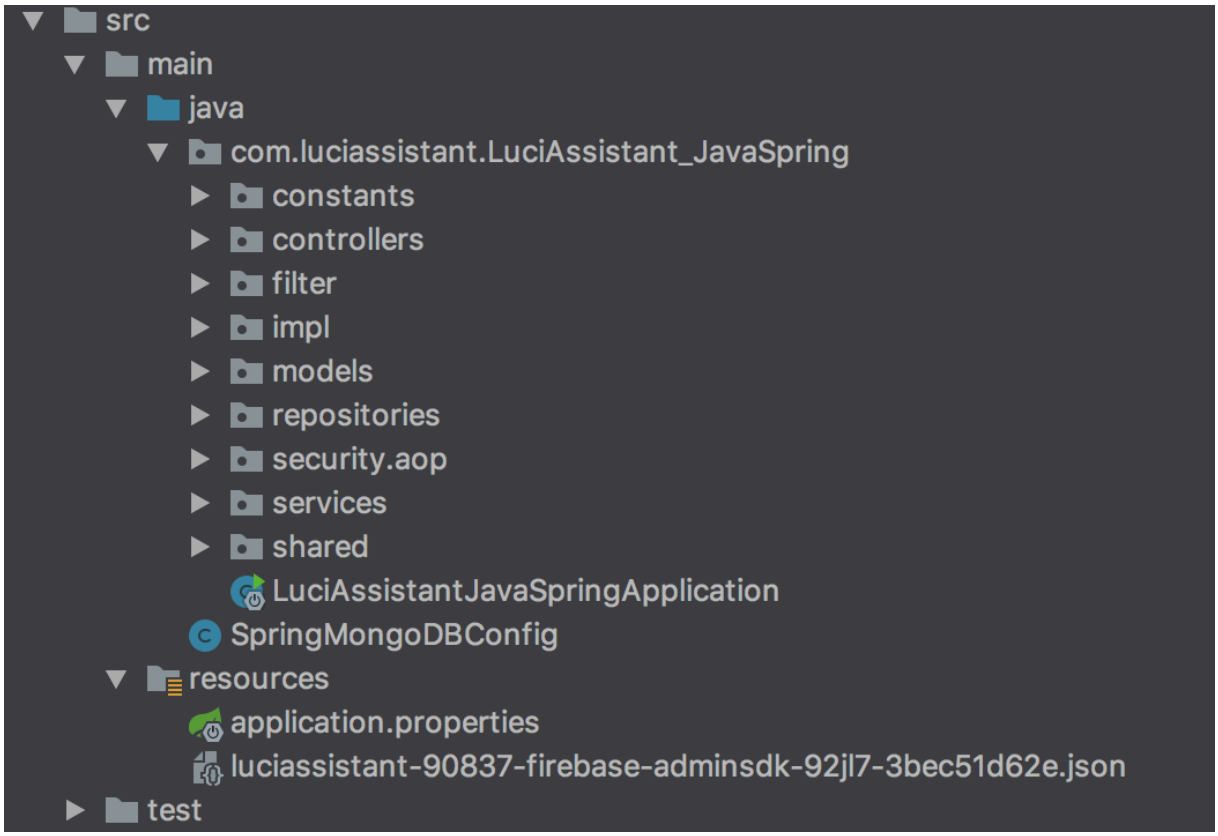
3.3.1 REST API Backend: Spring Boot

De backend waarop alle applicaties hun API-calls maken is ontwikkeld in Spring Boot. Dit is een krachtig *framework* dat op Java steunt.

3.3.1.1 Structuur

Spring Boot biedt enorm veel functionaliteiten aan. Om hier optimaal gebruik van te maken, is het essentieel dat het project goed gestructureerd wordt.

Het project bestaat uit de volgende mappen:



Figuur 11 Structuur project Spring Boot

De “constants” map bevat alle waardes die vaker gebruikt worden in het project zoals de API-routes, categorieën, error berichten, enz. Dit zorgt voor een extra laag veiligheid en consistentie doorheen het project.

De controllers zijn de aanspreekpunten. Deze bepalen welke methodes uitgevoerd en welke berichten teruggestuurd worden. Ook is er veel beveiliging voorzien zoals het bepalen van welke gebruikers deze methodes mogen aanspreken.

De “filters” map bevat momenteel enkel de *CORS-filter* configuratie. Dit is om te vermijden dat onbekende IP-adressen een *request* mogen sturen naar deze applicatie.

De “impl” map bevat alle implementaties van de service *interfaces*. Deze implementaties worden gebruikt om controller *requests* te behandelen. Ze spreken de *repositories* aan en kunnen aan de hand van de parameters de juiste waardes terugsturen naar de controllers.

De “models” map bevat alle domein klassen. Deze bepalen de structuur en welke waardes een model moet hebben, bv. Alle informatie dat van een klant moet bijgehouden worden:

- Naam
- Adress
- Email
- Telefoon
- Opgeslagen locaties
- Enz.

Vervolgens heb je de “repositories” map. Dit is het aanspreek punt van de database. Hier wordt later meer info over gegeven.

De “*security*” map bevat alle functionaliteit om de applicatie veilig te maken tegen ongewenste verzoeken. Enkel als de juiste authenticatie sleutel mee gestuurd wordt naar de backend, is er de mogelijkheid om data op te vragen. Ook worden hier de rollen nagekeken. Zo kunnen enkel administrators hun bedrijf instelling aanpassen.

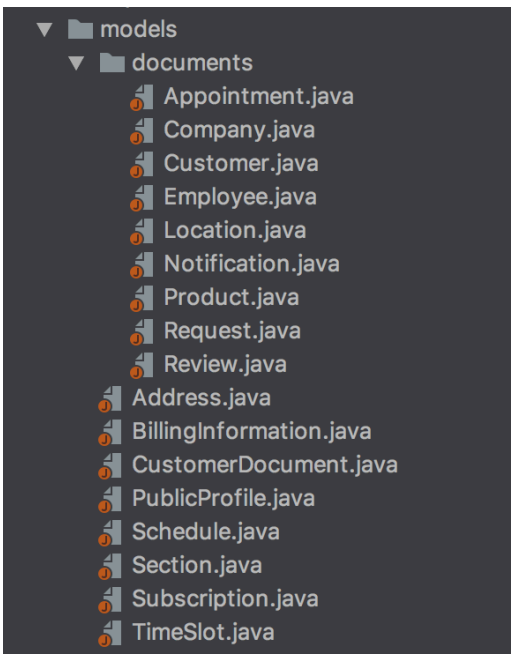
De “services” map bevat alle *interfaces* die geïmplementeerd moeten worden door de *service* implementaties.

Als laatste bevat de “shared” map enkele belangrijke configuraties en/of klassen die doorheen heel de applicatie gebruikt worden.

3.3.1.2 Database

Omdat er veel data opgeslagen moet worden door de backend, wordt er een database gebruikt. Voor dit project is er gekozen om MongoDb te gebruiken, dit is een NoSql database. Typisch is dat er gebruik gemaakt wordt van een boomstructuur. Deze soort database biedt vele voordelen. De voornaamste reden is de enorme flexibiliteit tijdens het programmeren. Aan de andere kant zijn er weinig concrete regels, waardoor de ontwikkelaar goed moet oppassen dat het project gestructureerd blijft.

Het moeilijkste aan een NoSql database is een datastructuur te ontwerpen die de data zo snel en efficiënt mogelijk kan ophalen. Voor dit project krijgen alle grote entiteiten een eigen repository. Al de kleinere data objecten zoals adressen, timeslots, documenten enz. bevinden zich in de entiteit zelf.



Figuur 12 structuur entiteiten

3.3.1.3 Beveiliging

Het doel is om een veilige REST-API te ontwikkelen die onafhankelijk en publiek beschikbaar is via een webserver. Om te vermijden dat er geknoeid gaat worden met de gegevens moet er een goede beveiliging zijn. Wegens de beperkte beschikbare tijd is er de keuze gemaakt om gebruik te maken van Firebase, deze zal het opzetten van de authenticatie vlot laten verlopen.

Firebase is een *third-party library* die een heel groot deel van de authenticatie afhandelt waardoor er geen eigen authenticatie server ontwikkeld moet worden. Dit bespaart veel tijd en biedt tegelijk een beveiliging die beter is dan een zelfgemaakte authenticatie en autorisatie server. Indien er nog enkele weken extra beschikbaar waren, dan was het een goede optie geweest om dit wel zelf te maken. Dit zou een betere integratie bieden met functionaliteit in de backend en database. Ook zou dit ervoor zorgen dat alle gebruikers hun inloggegevens op eigen servers beschikbaar zouden zijn.

De scope van deze stage is vooral bedoeld om te laten zien dat het platform kan werken en bedrijven er gebruik van kunnen maken. Beveiliging is belangrijk in de toekomst als er meerdere gebruikers zijn.

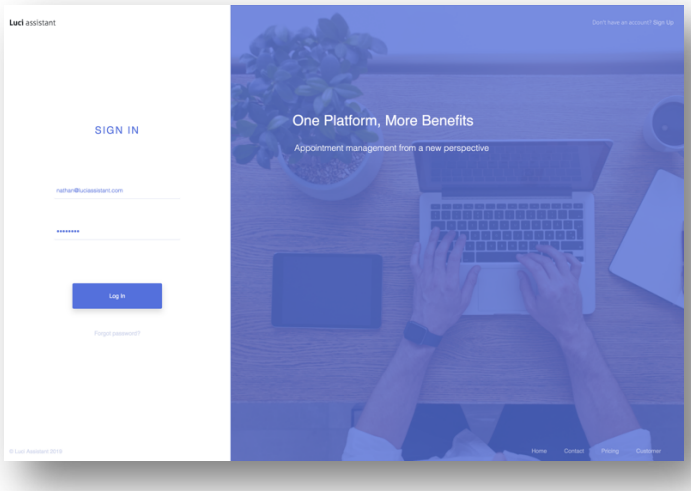
3.3.1.4 Agenda algoritme

De kern van de applicatie draait rond het maken van afspraken. Hiervoor is er een eigen algoritme ontwikkeld dat aan de hand van verschillende variabelen de klant de mogelijkheid geeft om een afspraak te maken bij een dienstverlener.

Als eerste kijkt het algoritme per werknemer wanneer hij wel en niet werkt. Vervolgens zal er nagegaan worden wanneer er al afspraken ingepland zijn. Als laatste worden alle beschikbare tijden weergegeven aan de gebruiker.

Ook worden deze beschikbare tijden vergeleken met de duur van de gewenste behandeling van de klant. Hierdoor gaat heel het systeem van afspraken maken automatisch kunnen functioneren en wordt er veel tijd bespaard aan beide kanten.

3.3.2 Dashboard



Figuur 13 Login scherm dashboard

Het dashboard wordt gebruikt door de dienstverleners. Een dienstverlener kan zich toevoegen aan het platform door een bedrijf aan te maken. Dit bedrijf bevat één of meerdere locaties, één of meerdere werknemers, en elke locatie bevat de verschillende diensten die ze daar aanbieden.

Dit dashboard is specifiek ontwikkeld om zo snel en efficiënt mogelijk afspraken te kunnen beheren. Als een klant geen gebruik wil maken van de mobiele applicatie, kan hij nog altijd contact opnemen met de dienstverlener. Vervolgens kan de dienstverlener het dashboard gebruiken om in een paar klikken een afspraak in te plannen.

Natuurlijk werkt niet elke onderneming volledig automatisch en is er de mogelijkheid om het type automatisatie te bepalen. De keuze bestaat uit een volledig automatische agenda waarbij klanten rechtstreeks afspraken kunnen maken. En een semiautomatisch systeem waarin elke afspraak geaccepteerd moet worden door de dienstverlener.

Het semiautomatisch systeem vereist wel meer input van de dienstverlener, maar het geeft hem wel meer controle en zekerheid over wanneer alle afspraken worden ingeboekt.

Voor de ontwikkeling van het dashboard wordt er gebruik gemaakt van Angular. Dit is een *open source framework* ontwikkeld door het Angular Team van Google. Angular is een complete herwerking van AngularJS, dat voordien door hetzelfde team ontwikkeld werd.

Angular maakt gebruik van TypeScript. Dit is een *open source* programmeertaal die ontwikkeld en beheerd wordt door Microsoft. Het is een *strict syntactical superset* van Javascript. Het geeft de mogelijkheid om *static typing* toe te voegen aan variabelen.

3.3.2.1 Structuur

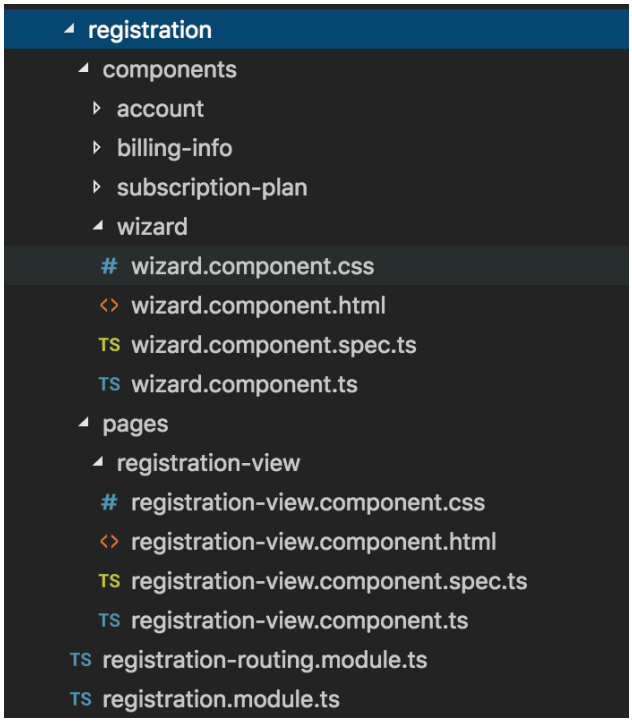
De kern van elke Angular applicatie bestaat uit het “app.ts” bestand dat alle modules inlaadt. Vervolgens wordt het “main.js” bestand ingeladen door de browser. Als laatste geeft de browser de applicatie weer aan de gebruiker. Angular heeft zijn eigen code conventies die makkelijk zijn om gebruiken. De gebruikte structuur doorheen heel de applicatie bestaat uit de volgende mappen:

- Node_modules
    - Deze map bevat alle *dependencies* die gebruikt worden doorheen heel het project.
    - Vele *dependencies* zijn noodzakelijk voor de basis werking van het programma. Maar het komt ook voor dat een speciale *library* of component al door iemand ontwikkeld is, en dat er veel tijd bespaard kan worden door deze *dependency* te gebruiken.
- Constants
    - Hier worden alle constant bijgehouden zodat je fouten minimaliseert.
    - Enkele van deze constanten zijn de routes om te navigeren, de API-constanten voor de http-calls, pusher, foutmeldingen enz.
- Core
    - De “core” map bevat alles dat te maken heeft met authenticatie. Waaronder een *guard* class om te controleren of een user ingelogd is.
    - Ook bevat deze map enkele services om de user aan te melden. Deze *guards* kunnen ervoor zorgen dat er grote controles uitgevoerd kunnen worden op het volledige programma.
- Login, Registratie
    - Dit zijn de componenten die zorgen voor de login en registratie. Hier bevinden zich de bestanden voor de layout, styling en de business logica.
- Shared
    - Deze map bevat de meeste logica, hier zijn alle services gevestigd. De services zijn het contact punt met de backend, maar er zijn natuurlijk ook helper functies zoals *pipes* om data te manipuleren en weer te geven. Ook is hier een *store* voorzien om doorheen heel de applicatie dezelfde data te kunnen gebruiken. De *store* is een *singleton* die variabelen onthoudt zoals de ingelogde werknemer, het bedrijf van deze werknemer, de geselecteerde locatie enz. Ook kunnen deze waardes geüpdatet worden met de voorziene functies.
    - Als laatste worden hier ook de interfaces van de gebruikte modellen gedefinieerd.
- Main-screens
    - Deze map bevat de componenten en containers voor de hoofdschermen van het dashboard (home, my business, customers, agenda). Elke map heeft zijn eigen sub folders om zijn eigen componenten te bewaren.

Deze structuur zorgt voor een sterke *seperation of concerns* waardoor zo goed als niets afhankelijk is van elkaar. Elk component kan bestaan en werken zonder afhankelijk te zijn van een andere. Ook biedt deze structuur de mogelijkheid om makkelijk te werken met versiebeheer. Er zijn weinig tot geen *merge conflicts* mogelijk en dat zorgt voor een vlotte ontwikkeling in een team. Een voorbeeld is dat twee verschillende ontwikkelaars enkele weken aan een stuk kunnen werken aan verschillende features. Als het tijd is om de features samen te voegen is er nauwelijks een probleem. Hierdoor kan het programma en de nieuwe features meteen gebruikt worden in een productie omgeving.

3.3.2.2 Component

Een Angular applicatie bestaat volledig uit componenten. Zoals eerder vermeld is er geen vaste structuur om een applicatie te maken, waardoor de ontwikkelaar zelf moet beslissen hoe uitgebreid hij te werk wil gaan. Een view kan bestaan uit één component waarbij alle HTML, businesslogica en CSS in dit component wordt gezet. Maar het is ook mogelijk om een container component te maken die meerdere *child* componenten bevat. De *child* componenten bevatten kleinere stukken code, waardoor het programma veel overzichtelijker wordt. Angular voorziet veel functionaliteiten zoals Input en output tags om makkelijk data en events uit te wisselen tussen componenten. Op afbeelding 17 wordt getoond hoe de structuur van componenten in de *registration* map er uitziet. Op afbeelding 16 wordt html-file weergegeven van de *registration*-view component. Dit is de *parent* die alle *child* componenten inlaadt.



Figuur 14 Structuur registration component

Figuur 15 Html registration-view

3.3.2.3 Services

Een frontend applicatie die gebruik maakt van een backend om zijn data op te halen heeft *services* nodig. Angular heeft een slim systeem om services makkelijk te kunnen gebruiken doorheen heel de applicatie. Bijvoorbeeld als er een *service* wordt gemaakt om data op te halen uit de backend, kan de gebruiker aan de hand van *dependency injection* die *service* inladen in zijn component. Dit wordt gedaan via de *constructor*. Op afbeelding 18 is een voorbeeld te zien van hoe deze services worden ingeladen.

```
constructor(
  private _storeService: StoreService,
  private _countrySerivce: CountriesService
) { }
```

Figuur 16 Inladen services via constructor

3.3.2.4 Authenticatie

Om tijd te besparen is er gekozen om gebruik te maken van Firebase als onze authenticatie methode. Het is een simpele en krachtige service voorzien door Google met een uitgebreide documentatie en goede integratie met JavaScript. Firebase is een package dat eerst toegevoegd moet worden aan het project. Als dit gebeurd is moeten de *keys* in orde gebracht worden om de applicatie te koppelen aan het juiste Firebase project.

Als de setup voltooid is, kan er heel vlot een functie geschreven worden om een gebruiker te authenticeren. Op afbeelding 19 wordt een stuk code getoond waar op de Firebase *service* een methode wordt aangeroepen om een user in te loggen.

```
//Authenticate User
var userCredentials = this.authenticationForm.value;
firebase.auth().signInWithEmailAndPassword(userCredentials.email, userCredentials.password).catch((error) => {
  // Handle Errors here.
  var errorCode = error.code;
  this.updateErrorMessage(false, ErrorConstants.INVALID_CREDENTIALS_FIREBASE);
  this.isAuthenticating = false;
```

Figuur 17 Firebase login user

3.3.2.5 State management

Een belangrijk aspect bij het ontwikkelen van een web app is het weergeven van data op een vlotte manier. Omdat de applicatie draait in een browser moet er altijd geprobeerd worden om gebruikt te maken van enkele technieken zoals *reactive programming*, asynchroon data weergeven en statemanagement.

De meeste webapps bestaan uit meerdere componenten die met elkaar moeten communiceren. Gegevens moeten van de ene naar de andere view gestuurd worden. Bepaalde variabelen moeten doorheen de applicatie beschikbaar zijn enz. Hiervoor kan een store gebruikt worden. Een store is niets meer dan een singleton klasse die variabelen bevat. Deze variabelen kunnen geüpdatet worden door functies aan te roepen op deze singelton. Ook kan een component zich *subscriben* op een *observable* in deze store, waardoor er bij elke verandering iets automatisch getriggerd wordt. Op figuur 20 is de *store service* weergegeven. Als de "updateLocation" methode aangeroepen wordt, gaat de "selectedLocationObserable" geüpdatet worden met de meegegeven "locationId" en wordt iedereen die geabonneerd is op de "selectedLocationObserable" verwittigd dat er een wijziging is.

```
@Injectable({
  providedIn: 'root'
})
export class StoreService {

  authenticatedEmployee: IEmployee;
  authorizationSessionToken: string;
  authenticatedCompany: ICompany;
  selectedLocation: ILocation;

  selectedLocationObservable = new Subject<ILocation>();
  loadingObservable = new Subject<string>();
  loadingCount: number = 0;

  constructor(private _locationService : LocationService, private _router: Router,
    private _authService: AuthService, private _employeeService: EmployeeService, private _companyService: CompanyService) {

    }

  updateLocation(locationId: string){
    this._locationService.getLocationById(locationId).subscribe(location => {
      this.selectedLocation = location;
      this.selectedLocationObservable.next(location);
      this.selectedLocation = location;
    },
    error => {
      console.log("Failed updating location with id: " + locationId);
    });
  }
```

Figuur 18 StoreService

3.3.2.6 Modellen

In tegenstelling tot JavaScript biedt Typescript de mogelijkheid om eigen interfaces te maken voor modellen. Dit zorgt voor een veel betere structuur in je applicatie. Dit geeft de mogelijkheid om eigen objecten aan te maken die specifieke variabelen bevatten. Ook zorgt dit voor een veel betere programmeer ervaring omdat er minder fouten gemaakt kunnen worden, het duidelijker is welke type een variabele is enz.

```
import { ITimeSlot } from './timeslot.interface';

export interface IRequest {
  _id: string;
  locationId: string;
  productId: string[];
  employeeId: string;
  customerId: string;
  createdAtDateTime: string;
  timeSlot: ITimeSlot[];
  note: string;
  accepted: boolean;
}
```

Figuur 19 IRequest interface

3.3.2.7 Routing

Om tussen verschillende views te kunnen navigeren is er gebruik gemaakt van routing. Angular voorziet vele extra functionaliteiten om routes op te stellen, *guards* toe te voegen, *child* routes op te stellen enz. Op figuur 22 wordt getoond hoe de structuur van de routes opgesteld is. Als eerste is er een routing module die ingeladen wordt in de “app.ts” *file*. Vervolgens bevat de “app-routing” module een *array* van Route objecten. Elke object bestaat uit een “path”, een component en nog eventuele extra optionele velden.

Een voorbeeld is de dashboard route, daar wordt uit de route constante het pad mee gegeven, de component dat getoond moet worden als er naar die route wordt genavigeerd. Vervolgens is er een “authguard” *service* gekoppeld aan deze route. Deze *guard* wordt elke keer aangeroepen en controleert of er een gebruiker werkelijk is ingelogd. Tenslotte bevat deze route ook *child* routes, dit geeft de mogelijkheid om *sub routing* te doen waardoor er meerdere router-outlets gebruikt kunnen worden.

```
const routes: Routes = [
    { path: RouterConstants.ROOT_ROUTE, component: DashboardComponent, canActivate: [CanactivateAuthguardService]},
    { path: RouterConstants.REGISTRATION_ROUTE, component: RegistrationViewComponent },
    { path: RouterConstants.LOGIN_ROUTE, component: LoginComponent},
    { path: RouterConstants.ERROR_ROUTE, component: Error404Component},
    { path: RouterConstants.DASHBOARD_ROUTE, component: DashboardComponent, canActivate: [CanactivateAuthguardService],
        children: [
            {
                path:RouterConstants.CUSTOMERS_ROUTE,
                component: CustomercontainerComponent,
                canActivate: [CanactivateAuthguardService]
            },
            {
                path:RouterConstants.MY_BUSINESS_ROUTE,
                component: MybusinesscontainerComponent,
                canActivate: [CanactivateAuthguardService],
                children: [
                    {
                        path:RouterConstants.ACCOUNT_ROUTE,
                        component: AccountComponent,
                        canActivate: [CanactivateAuthguardService]
                    },
```

Figuur 20 Routing module

II. Onderzoek

4 Probleemstelling onderzoek

Op de dag van vandaag heeft bijna iedereen een smartphone. De kern van deze smartphones zijn de mobiele apps die erop draaien. Deze moeten natuurlijk ook door iemand ontwikkeld worden, hier hoort dan ook bij dat er twee verschillende besturingssystemen heersen (iOS en Android).

Een vaak voorkomend probleem is dat de ontwikkelaars hun applicaties altijd moeten voorzien in deze twee verschillende besturingssystemen. Dit is noodzakelijk want hierdoor is er een veel groter publiek dat de applicatie kan gebruiken. Dit is natuurlijk geen makkelijke taak, omdat er 2 compleet verschillende programmeertalen en omgevingen nodig zijn om deze applicaties native te ontwikkelen.

Sinds enkele jaren is er een opmars in cross-platform technologieën. Deze helpen de ontwikkelaars om hun programma maar 1 keer te moeten schrijven en dat het onmiddellijk gebruikt kan worden op beide platformen.

In dit onderzoek wordt nagegaan of de klanten applicatie van Luci Assistant ontwikkeld kan worden met zo een cross-platform technologie. De twee verschillende technologieën die onderzocht zullen worden, zijn: NativeScript in combinatie met Angular en PWA in combinatie met Angular.

NativeScript is een *framework* dat de mogelijkheid biedt om in combinatie met enkele andere talen (Angular, Vue.js, Javascript) een native applicatie te ontwikkelen die zowel op iOS en Android werkt.

Progressive web apps zijn vrij nieuw en opkomend, maar deze zijn zeer interessant voor de toekomst. Ze bieden enorm veel functionaliteiten aan en ze versnellen het ontwikkelingsproces terwijl ze de look en feel geven van een native applicatie.

De belangrijkste aspecten die er onderzocht gaan worden zijn:

- Documentatie
- Community
- Performantie
- Deployment
- Features
- Styling

De resultaten van dit onderzoek gaan bepalen met welke technologie de mobiele applicatie ontwikkeld gaat worden.

5 Onderzoeksmethode

Het onderzoek wordt uitgevoerd door middel van een POC uit te werken in twee technologieën. De grootste verschillen begeven zich in de implementatie van de views (HTML, CSS, NativeScript tags, …).

Als eerste wordt er een vergelijkingsmatrix opgesteld, deze wordt doorheen het hele onderzoek verder aangevuld. Er worden zo veel mogelijk aspecten bekeken en gedocumenteerd in deze matrix. Een voorbeeld van hoe deze matrix wordt opgesteld is zichtbaar in tabel 2.

Tabel 2 Voorbeeld vergelijkingsmatrix

	PWA	NativeScript
App store	Ondersteund door Google Play Store Geen mogelijkheid om een PWA aan te bieden in de App Store van Apple.	Volledige ondersteuning om cross-platform de applicaties aan te bieden om de verschillende app stores.
Push notificaties	Deze worden door de browser afgehandeld en enkel in Android is er een integratie voorzien.	Volledige support op beide OS'en door gebruik te maken van extra libraries.
…	…	…

Vervolgens gaat de ontwikkeling van de twee POC-applicaties van start. Beide gaan de homeview en enkele belangrijke features bevatten. De view die ontwikkeld wordt komt overeen met de designs van de werkelijke applicatie. Er wordt grondig bekeken hoe efficiënt/makkelijk de lay-out en styling is. Ook wordt er rekening gehouden met hoe responsief de views zijn op verschillende toestellen.

NativeScript maakt gebruik van zijn eigen componenten, die helpen met de responsiviteit van de applicatie.

De achterliggende code wordt hergebruikt door beide POC's, omdat deze allebei in Angular ontwikkeld worden. De data is voorzien door de REST API Java Spring backend. Deze is al reeds ontwikkeld vooraleer het onderzoek van start gaat. Ook worden enkele belangrijke features getest zoals het voorzien van *push notifications* om de gebruikers een melding te kunnen geven. Als laatste wordt er rekening gehouden met de *deployment* van beide POC's. Een PWA kan bijvoorbeeld niet zomaar gedownload worden vanuit de app store. Deze factoren worden grondig gedocumenteerd en weergegeven in de vergelijkingsmatrix.

6 Uitwerking onderzoek

6.1 Inleiding

De eerste stap die er wordt gedaan om het onderzoek goed te laten verlopen, is een grondige kennis te verkrijgen van de technologieën die vergeleken worden. Voor beide keuzes komen onder andere een korte historiek, specifieke kenmerken, werking en marktaandeel aan bod.

6.1.1 Progressive Web App

6.1.1.1 Historiek [1]

De geschiedenis van progressieve webapps begint met een sleutelmoment in de geschiedenis van de industrie, 2007. Het jaar van de eerste iPhone-lancering.

Dit kwam op een moment toen Web 2.0 vorm kreeg, en de HTML5-standaard nog steeds gedefinieerd werd. Webpagina's werden steeds dynamischer met interactieve kaarten, hoge kwaliteit *videostreams* en lokale browseropslag, waardoor de manier waarop we het web op desktopapparaten gebruikten, veranderde.

Veel van de kenmerken van PWA's zijn een voortzetting van de ontwikkeling van die krachtige web technologieën. In die tijd was er echter nog steeds een grote kloof tussen wat mogelijk was op desktops en mobiele apparaten.

Toen de originele iPhone werd gelanceerd was het vrijwel verplicht om *native* apps te ontwikkelen in traditionele talen zoals Java, C en C++. Tegenwoordig is ontwikkeling met HTML, CSS en JavaScript prominenter.

Wegens technische beperkingen en bezorgdheid over app-kwaliteit, beveiliging en privacy bleef het *native* model echter rondhangen, ondersteund door een eigen en gecureerde app-ecosysteem dat wordt beheerd door de telefoon en OS-leveranciers.

6.1.1.2 Kenmerken [2]

- Responsief
- Veilig
- Up-to-date
- Offline
- Installeerbaar

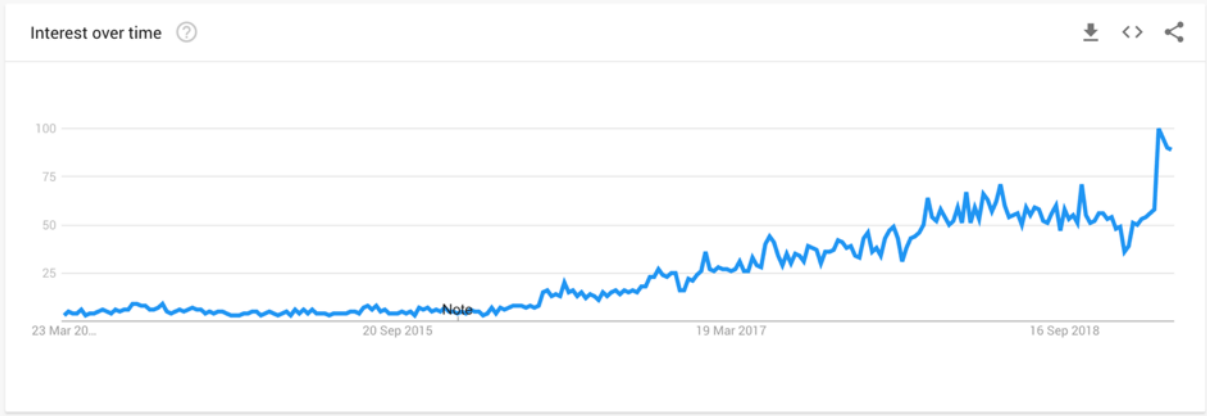
6.1.1.3 Werking

Een PWA heeft drie vereisten:

- HTTPS: Het moet een certificaat hebben en gebruikmaken van een SSL-verbinding.
- Web App Manifest: Er moet een manifest bestand staan in het project met daar alle metadata in.
- Service Worker: Als laatste moet er een *service worker* geconfigureerd zijn in het project. Deze gaat communiceren met de *browser* en de applicatie om belangrijke features te kunnen implementeren.

6.1.1.4 Interesse

Google Trends biedt de mogelijkheid om op te zoeken wat de interesse is van een bepaalde zoekterm. Figuur 25 toont de “interest over time” grafiek van “PWA” van de afgelopen vijf jaar, wereldwijd. De afgelopen vier jaar is er een sterke groei zichtbaar, met een sterke piek tijdens het afgelopen jaar.



Figuur 21 Google Trends PWA

6.1.1.5 Toekomstvisie

Volgens een survey gelooft 46% dat PWA de toekomst is. De verklaring hiervoor is dat het een natuurlijke keuze is als een ontwikkelaar wenst om een web/mobiele app te bouwen. De tweede grootste groep (24%), zegt dat het compleet afhangt van de applicatie en de functies die er gebouwd moeten worden. [3]

6.1.2 NativeScript

6.1.2.1 Historiek [4]

NativeScript werd in maart 2015 voor het eerst vrijgegeven. Versie 1.0.0 volgde twee maanden later. Het won al snel aan populariteit dankzij de 3000 GitHub-sterren en meer dan 1500 volgers op Twitter kort na de publieke *release*. Ondertussen zijn er meer dan 700 plug-ins beschikbaar, die officieel worden ondersteund door Progress, en/of voortkomen uit de *open source* community.

6.1.2.2 Kenmerken

- Toegang tot native API en componenten via JavaScript/ TypeScript
- Gebruikerservaring
    - Biedt een bijna perfecte gebruikerservaring omdat de applicaties gecompileerd worden naar *native code*.
- Eén codebasis
    - *Write once, run anywhere*. Er moet maar één keer ontwikkeld worden.
- Community
    - *Open source* project dat zorgt voor een sterke en groeiende *community*.
- App store
    - Mogelijkheid om de applicaties te publiceren in de Playstore en App Store.
- Officiële marktplaats
    - Vele plug-ins beschikbaar die ontwikkeld zijn door de community.

6.1.2.3 Werking [5]

6.1.2.3.1 Runtimes

Met de *runtimes* is het mogelijk om API's in de Android- en iOS-frameworks aan te roepen met behulp van JavaScript-code. Om dit te doen wordt er gebruikt gemaakt van JavaScript Virtual Machines.

6.1.2.3.2 Core Modules

De *Core Modules* zijn er om de abstracties te bieden die nodig zijn om toegang te krijgen tot de onderliggende *native* platformen.

Ook bieden de Core Modules een op basis van XML-gebaseerde manier voor het definiëren van gebruikersinterface, databinding en navigatie.

6.1.2.3.3 NativeScript CLI

De NativeScript CLI is de *command line* waarmee de apps gemaakt, gebouwd en uitgevoerd kunnen worden. De CLI werkt op Windows, macOS of Linux.

6.1.2.4 Interesse [6]

Sinds de lancering van de eerste versie van NativeScript is er een geleidelijke stijging van de interesse. Momenteel is de interesse op een evenwichtig balans gekomen, wat betekent dat het zijn plaats in de markt gevonden heeft.

Figuur 22 Google Trends NativeScript

6.1.2.5 Toekomstvisie [7]

De roadmap voor NativeScript in 2019 bestaat vooral uit verbeteringen van de bestaande functies. Ook is er een experimentele functie die “nativescript-web” noemt. Deze zou 100% *code-sharing* bieden op alle platformen. Ook zou dit toelaten om applicaties te ontwikkelen voor macOS, Windows en Linux via Electron.

Het tweede aspect dat verbeterd wordt, is fouten blootstellen in de Javascript code aan de foutrapportageservices. Deze bevatten meestal de bedrijfslogica van de app en deze zijn momenteel volledig verborgen.

Fouten, hoewel niet gecrasht, zijn nog steeds belangrijk voor de gezondheid van een applicatie. Bovendien wordt de integratie met bestaande analyse *services* beter gedocumenteerd.

6.2 Ontwikkeling

Nu dat beide technologieën volledig uitgeklaard zijn, gaat er voor beide een POC ontwikkeld worden. Deze POC bestaat uit de homeview en enkele features die belangrijk zijn om de juiste keuze te kunnen maken voor de verdere uitwerking van de stageopdracht.

De homeview is gekozen omdat deze de meeste informatie bevat en enkele complexe functionaliteiten bevat zoals push notificaties, camera en locatie. Met de ontwikkeling van deze view wordt er snel gemerkt of de componenten, styling en werking van de technologie makkelijk is om te gebruiken.

De homeview bestaat uit een “top-nav” waarin zich een knop bevindt om naar de profiel pagina te gaan. Deze knop gaat in de POC vervangen door een knop die de camera gaat openen. De gebruiker kan dan een foto maken en achteraf wordt deze foto getoond op de knop. Zo is meteen de functionaliteit van het gebruiken van een camera getest.

Vervolgens bevat de homeview enkele regels tekst met info. Ook bevindt zich hier de zoekbalk om snel tussen de onderstaande lijst te kunnen zoeken. Als laatste zijn de 3 tabbladen aanwezig waar de gebruiker kan navigeren tussen de verschillende lijsten. De tabbladen zijn:

- Agenda
    - Hier kan de gebruiker een overzicht zien van al zijn afspraken.
- Explore
    - Deze lijst bevat alle bedrijven die zich in de buurt bevinden en waar de klant een afspraak kan maken.
- Me
    - Deze lijst bevat een overzicht van alle bedrijven waar de gebruiker eerder is geweest, een afspraak heeft vastliggen.

Push notificaties zijn vrij belangrijk voor de werking van de applicatie. Als een bedrijf een afspraak heeft geaccepteerd dan moet de gebruiker zo snel mogelijk verwittigde kunnen worden. Ook als er een afspraak binnenkort plaats zou vinden, dan zou de applicatie een notificatie moeten kunnen sturen om de gebruiker te informeren.

De camera zal gebruikt worden voor het maken van een profiel foto van de gebruiker. Deze foto gaat vervolgens getoond worden op de homeview en op de profiel pagina van de gebruiker.

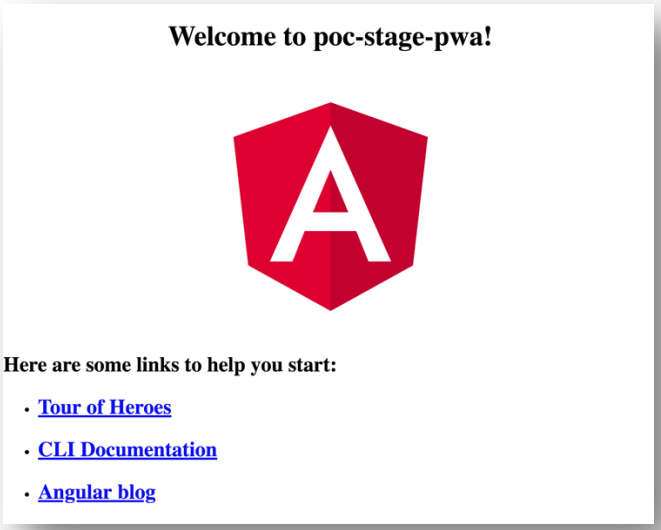
Als laatste moet ook de locatie van de gebruiker beschikbaar zijn. Deze zal gebruikt worden om in de *Explore* view de dichtstbijzijnde bedrijven te vinden.

6.2.1 Ontwikkeling PWA

6.2.1.1 Setup

Om een nieuwe Angular project te maken, kan er via de console een commando uitgevoerd worden om een nieuw project te genereren. Het is noodzakelijk om via npm de globale angular-cli te installeren. Het commando om een project te maken is:

- ng new poc-stage-pwa



Figuur 23 Nieuw Angular project POC PWA

In het genereerde project kan vervolgens de module geïnstalleerd worden die zorgt voor de noodzakelijke klassen en modules om PWA mogelijk te maken. Het commando hiervoor is:

- ng add @angular/pwa --project poc-stage-pwa

Dit commando zorgt voor de volgende acties:

- Voegt de @angular/service-worker package toe aan het project.
- Geeft de mogelijkheid om *service worker builds* te ondersteunen in de CLI.
- Importeert en registreert de *service worker* in de app module.
- Updatet de index.html file:
    - Voegt een link toe naar de manifest.json file.
    - Voegt meta tags toe voor theme-color.
- Installeert icon bestanden voor de PWA.
- Maakt een *service worker* configuratie bestand aan genaamd “ngsw-config.json”, deze specifieert de *caching* en andere instellingen.

Omdat een *service worker* enkel werkt via https moet het project op een SSL-secured domain en server draaien. Hiervoor is Heroku een handig hulpmiddel om snel en gemakkelijk een project te *deployen*. Om dit mogelijk te maken moet er eerst een project aangemaakt worden in Heroku. Vervolgens wordt het project in Github gekoppeld en wordt er een trigger ingesteld die zorgt voor *automatic deployment* bij het pushen naar de "master branch".

Tenslotte moeten er nog enkele kleine aanpassingen gedaan worden in de "package.json", zodat de *deployment* naar de servers van Heroku werkt en de *service worker* actief wordt. Een van de vereisten is om een "express server" op te zetten die de bestanden uit de "dist folder" *deployed*. Op figuur 28 is de basis setup voltooid en kan de PWA op de "homescreen" opgeslagen worden. Opslaan op de "homescreen" van het toestel kan op iOS enkel gedaan worden via safari. Dit is een maatregel van Apple om zo veel mogelijk controle te behouden van wat er gedaan is op het toestel. Dit heeft vooral te maken met security.

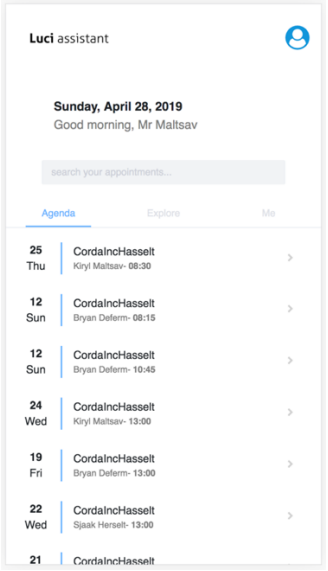
Nu dat de applicatie zich op de "homescreen" bevindt, kan deze vervolgens gebruikt worden als een normale applicatie. De app kan ook gebruikt worden zonder internet omdat de *service worker* zorgt voor *caching*.

Figuur 24 Angular applicatie als PWA op gsm

6.2.1.2 Views

Nu dat de basis setup compleet is, kan de ontwikkeling van de view beginnen. De eerste view die ontwikkeld wordt is de “home view”. Deze toont de lijst van afspraken en bedrijven. Vooraleer deze view gemaakt wordt, moet er eerst enkele services geschreven worden om de data op te kunnen halen uit de backend.

Vervolgens worden deze services aangesproken in de view en wordt de data getoond. Omdat onderzoek draait rond de designs en functionaliteit van specifieke features, wordt er gebruik gemaakt van statische inloggegevens.



Figuur 25 POC PWA Homeview

Het ontwikkelen van de views wordt in Angular gedaan met HTML en CSS. Ook wordt Bootstrap gebruikt in dit project. Bootstrap helpt met de lay-out en zorgt ervoor dat de views responsief zijn.

Op afbeelding 29 is de screenshot van de afgewerkte homeview zichtbaar. Deze wordt weergeven in Chrome gebruik makend van de *developer tools*. Het formaat van de gsm is dezelfde als die van een iPhone 6/7. Het ontwikkelingsproces van deze view gaat vlot en heeft weinig knelpunten. De structuur bestaat uit de “top-nav” en de 3 onderstaande lijsten die worden weergeven aan de hand van welke “navItem” geselecteerd is.

Voor het inladen van de gegevens wordt er eerst een lijst opgehaald van alle afspraken. Vervolgens wordt deze lijst doorlopen met een “foreach” lus die dan per afspraak de extra detail gegevens gaat ophalen.

De view zelf is zeer responsief en werkt perfect op elke resolutie. Wat wel meteen merkbaar is, is dat de lijst niet *native* aanvoelt. Het scrollen is niet zo vloeiend als een lijst die ontwikkeld is met NativeScript. Op vlak van design, is HTML en CSS makkelijk om te gebruiken. De mogelijkheid om *pixel perfect* te werken en te bepalen hoe alles werkt, is zeker geen knelpunt.

Als er graag een meer uniforme styling gebruikt wilt worden zoals “Material design”, dan is dit ook perfect mogelijk.

Op Afbeelding 30 wordt de code voor de “home-container” en “appointment-list” weergegeven. De home-container bevat alle *child* componenten. De “app-top-nav” component laat weten aan de container welk “navItem” hij geselecteerd heeft.

De “appointment-list” laat zien hoe weinig code er nodig is om de volledige lijst te renderen met alle styling, lay-out en gegevens. De lijst bestaat uit “rows” die door bootstrap voorzien worden. De “*ngFor *directive*“ zorgt ervoor dat er per afspraak een rij gemaakt wordt met de nodige gegevens.

Omdat er gebruik gemaakt wordt van HTML en CSS, kunnen er heel snel aanpassingen doorgevoerd worden aan het volledige programma of aan één specifiek component.

Wat Angular nog een leuke *framework* maakt is de mogelijkheid om objecten weer te geven in de HTML-code. Zo kan een “appointment” object zijn gegevens meteen opgehaald worden uit de business-code laag en per veld van het object weergegeven worden op de gewenste positie.

home-container.component.html

```
<app-top-nav (selectedNavItemEvent)="onSelectedNavItemChange($event)"></app-top-nav>

<app-appointment-list *ngIf="selectedNavItem === 'AGENDA'"></app-appointment-list>

<app-companies-list *ngIf="selectedNavItem === 'EXPLORE'"></app-companies-list>

<app-saved-companies-list *ngIf="selectedNavItem === 'ME'"></app-saved-companies-list>
```

appointment-list.component.html

```
<div class="container">
  <div class="row align-items-center row-appointment" *ngFor="let appointment of displayAppointmentList">
    <div class="col-2 text-center date">
        <div class="row">
            <div class="col">
                <strong>{{appointment.dayOfMonth}}</strong>
            </div>
        </div>
        <div class="row">
            <div class="col">
              {{appointment.dayOfWeek}}
            </div>
        </div>
    </div>
    <div class="col">
      <div class="row">
        <div class="col">
          {{appointment.locationName}}
        </div>
      </div>
      <div class="row">
        <div class="col sub-text">
          {{appointment.employeeName}}- <strong> {{appointment.from}}</strong>
        </div>
      </div>
    </div>
    <div class="col-2 text-center angle">
        <i class="fas fa-angle-right"></i>
    </div>
  </div>
</div>
```

Figuur 26 POC PWA html code van de home-container

6.2.1.3 Functionaliteit

6.2.1.3.1 Camera

Nu dat de lay-out van de homeview klaar is kunnen er enkele functionaliteit getest worden. De eerste functionaliteit is de camera. Als er op de profiel knop gedrukt wordt, moet de camera geopend worden en moet de gebruiker een foto maken of meegeven. Als dit succesvol gebeurd is, gaat de foto weergeven worden op de profiel knop.

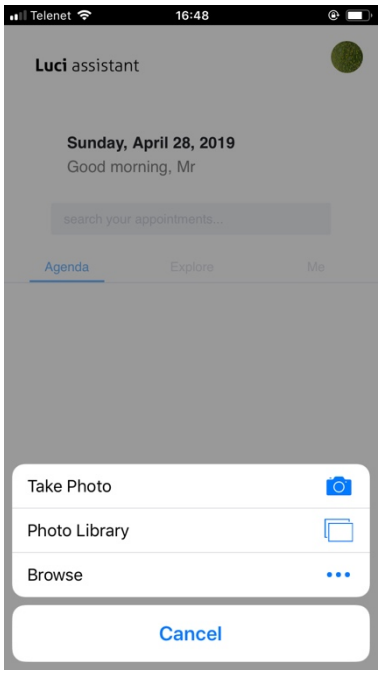
Het openen van de camera is makkelijk. Het enige dat er gedaan moet worden, is het voorzien van een "input" tag. Deze tag staat toe om de camera te gebruiken en kan vervolgens de afbeeldingen verwerken.

```
<input type="file" accept="image/*">
```

Bij het drukken op deze input wordt het venster op afbeelding 31 weergeven op de mobiele app. Hier krijgt de gebruiker de mogelijkheid om zijn foto te kiezen uit zijn bibliotheek. Ook kan de gebruiker een nieuwe foto nemen. Als er een foto genomen is dan wordt deze meteen getoond op de "homeview".

De *browser* zorgt voor de afhandeling waardoor de aantal regels code minimaal blijven. Dit maakt dat de toekomst voor PWA wel effectief vele voordelen kan bieden. Ook niet te vergeten is dat dit onmiddellijk werkt op zowel Android als iOS toestellen.

De foto kan nu opgeslagen worden in de backend en is dan altijd beschikbaar.



Figuur 27 PWA Profiel foto image picker

6.2.1.3.2 Push notificaties

Omdat het belangrijk is om de gebruiker van de applicatie altijd up-to-date te houden, bijvoorbeeld als er iets veranderd is aan zijn afspraak, moeten er push notificaties voorzien worden.

Push notificaties zijn een manier om real-time berichten te kunnen sturen van een server naar een applicatie. Dit wordt mogelijk gemaakt door een design patroon dat het “Pub-Sub pattern” noemt.

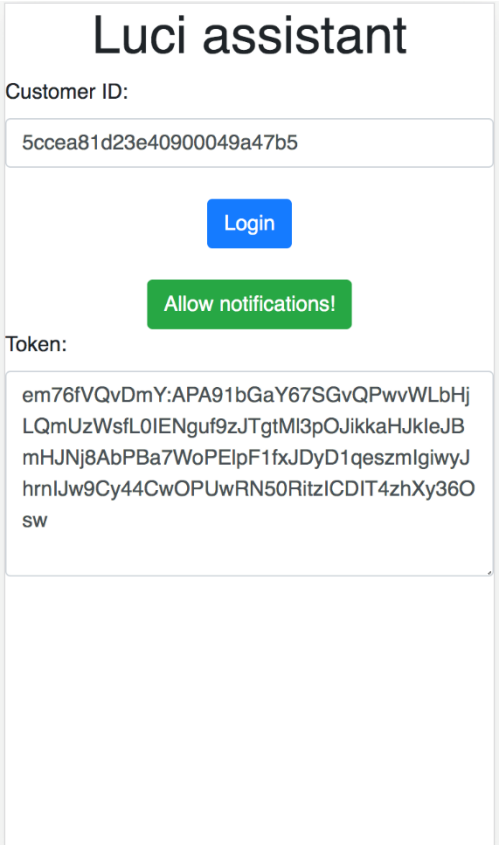
Bij dit patroon is er altijd een iemand die luistert en iemand die het bericht verstuurt. Om ervoor te zorgen dat het netwerk niet overbelast wordt, wordt er gebruikt gemaakt van unieke ID’s en kanalen. Een bekend platform dat dit doet is “Pusher”. Pusher wordt ook gebruikt in de backend van Luci.

In dit scenario zal Firebase Messaging gebruikt worden. Dit is de *cloud messaging solution* ontwikkeld door het team van Firebase. Angular en Firebase werken vlot samen, dat zorgt ervoor dat de ontwikkeling van een POC met push notificaties vlot zal verlopen.

De eerste stap is het implementeren van de “Firebase messaging” module. Deze moet toegevoegd worden in de “app.module”. Vervolgens moet er een nieuwe *service worker* aangemaakt worden, dit is de “firebase-messaging-sw.js”.

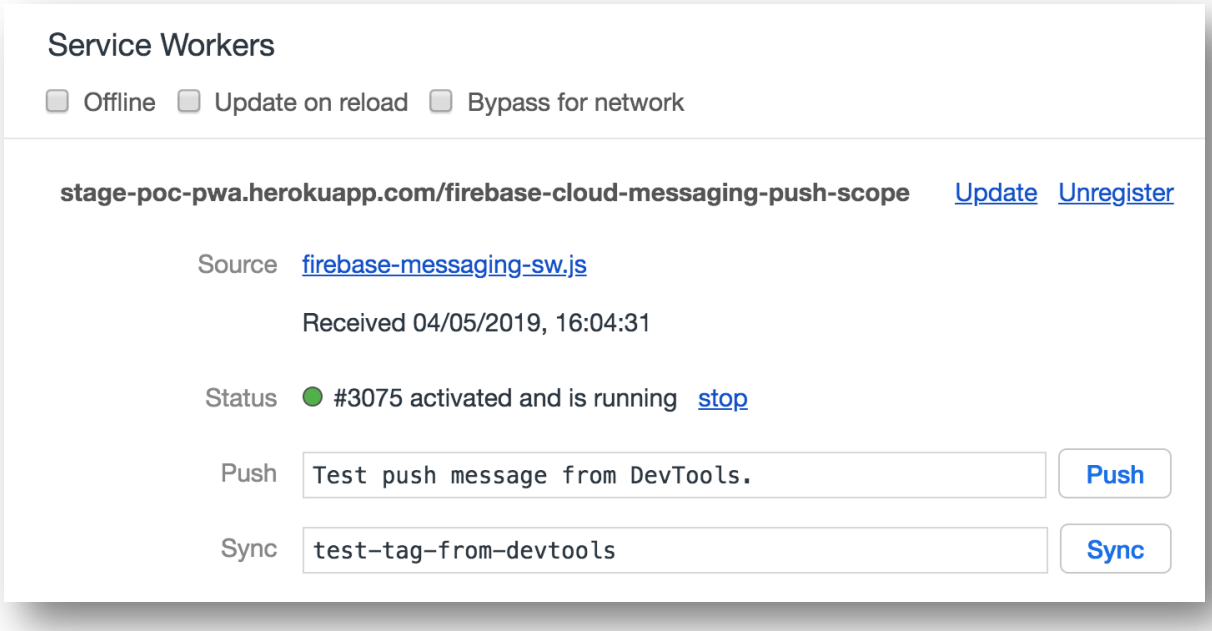
Deze *service worker* zorgt voor de afhandeling van binnenkomende berichten. Deze *worker* is vrij intelligent en weet ook wanneer de applicatie op de voorgrond openstaat of dat de melding op de achtergrond verstuurd moet worden.

Om de *service worker* aan te zetten moet de gebruiker eerst toegang verlenen. Op afbeelding 32 heeft de gebruiker de toegang verleend en is er een token door Firebase gegeneerd, en vervolgens teruggestuurd naar de gebruiker. Deze token zal gebruikt worden om de push notificaties naartoe te sturen.



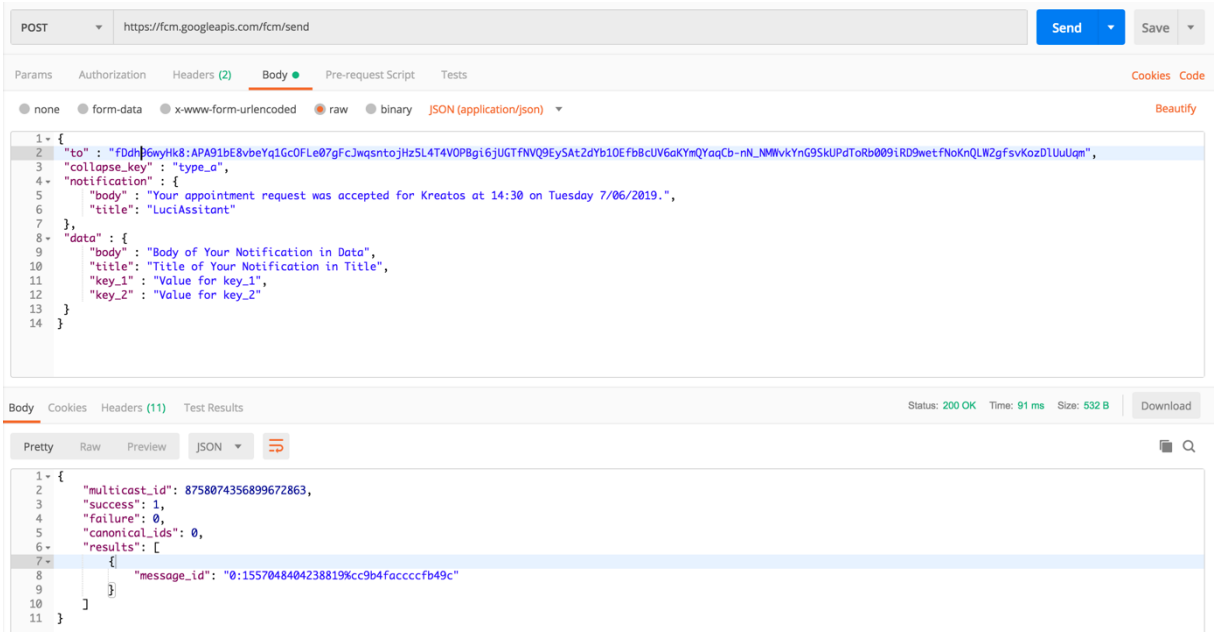
Figuur 28 Toelaten van push notificaties en terug sturen van de token

Op afbeelding 33 zien we dat de *service worker* actief is in de browser. Elke keer als er een bericht gestuurd wordt naar de unieke messaging token, zal de *service worker* het bericht opvangen en tonen op het scherm. Deze berichten kunnen gesimuleerd worden met “Postman”. Dit is een veel gebruikte tool om HTTP *requests* te sturen en te ontvangen.



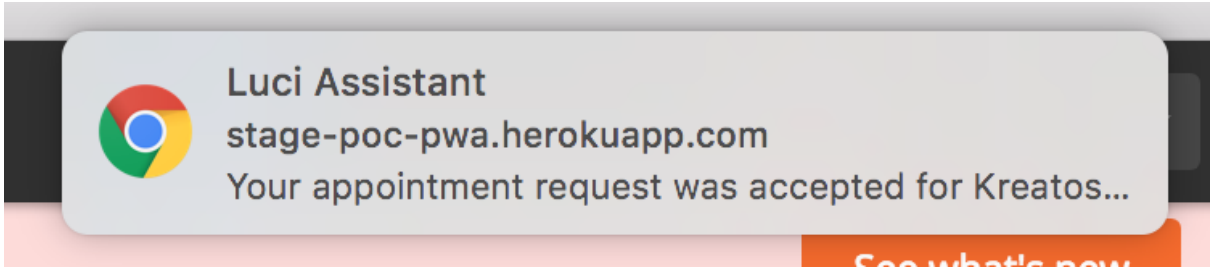
Figuur 29 Firebase messaging service worker

Op Afbeelding 34 wordt een “Firebase message” gestuurd naar de token van de gebruiker. Als resultaat krijgt de gebruiker de notificatie binnen, en de *response body* laat weten dat het succesvol verlopen is.



Figuur 30 Versturen van een firebase message met postman

Als resultaat krijgt de gebruiker een push notificatie binnen. Als de applicatie van de gebruiker openstaat, zal het bericht op de applicatie getoond worden. Als de applicatie in de achtergrond openstaat dan krijgt de gebruiker de notificatie te zien als een desktop melding. De melding wordt door de browser afgehandeld. Op afbeelding 35 is te zien hoe deze notificatie door Chrome weergegeven wordt op het bureaublad.



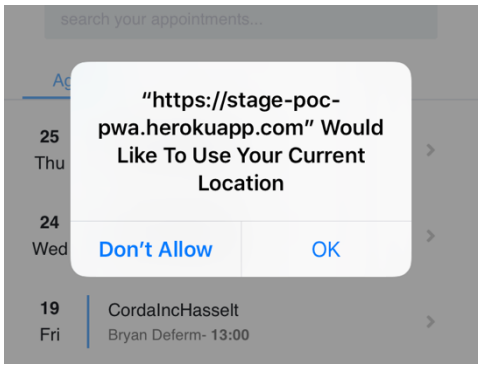
Figuur 31 Push notificatie op het bureaublad

6.2.1.3.3 Locatievoorzieningen

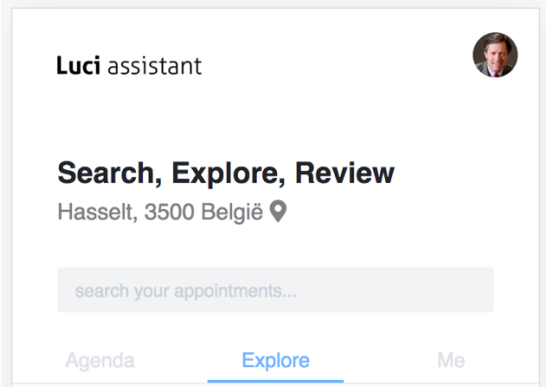
Als laatste functionaliteit gaat de locatievoorziening bekeken worden. Het doel is om de stad, postcode en land van een gebruiker op te vragen en weer te geven op zijn scherm.

De eerste stap is om de locatie van de gebruiker op te vragen. Dit is een simpele regel code die de coördinaten van de gebruiker terugstuurt. Deze coördinaten worden door de browser opgehaald. Op afbeelding 36 moet de gebruiker eerst toestemming geven voordat de browser de coördinaten mag geven.

Dit vergemakkelijkt het volledige proces en er moet geen extra module gebruikt worden voor dit resultaat te behalen. Op afbeelding 37 is het resultaat getoond waarbij de locatie zeer accuraat opgehaald en verwerkt wordt.



Figuur 32 POC PWA Toestemming locatievoorzieningen



Figuur 33 POC PWA locatie

In afbeelding 38 worden de coördinaten omgezet naar een bruikbare locatie via de Google Maps API. Vervolgens kan er uit de teruggeven JSON, de nodige data genomen worden. Uit deze data kan de stad, postcode en land gehaald worden en in variabelen gestoken worden. De JSON bevat ook bezienswaardigheden en de namen van de bedrijven in de buurt. Bijvoorbeeld Corda 1 of CordaINC.

```
this._http.get("https://maps.googleapis.com/maps/api/geocode/json?latlng="
+ lat + "," + long + "&sensor=false&key=XXXXXXXXXXXXXXXXXXXXXXXXXXXX").subscribe(val => {
  console.log(val);

  this.city = val['results'][0]['address_components'][0]['long_name'];
  this.postalcode = val['results'][0]['address_components'][2]['long_name'];
  this.country = val['results'][0]['address_components'][1]['long_name'];

});
```

Figuur 34 Google Maps API

6.2.1.4 Conclusie

Een PWA heeft vele voor- en nadelen. Het grootste voordeel is de tijd die er bespaard wordt met het ontwikkelen van een PWA. Omdat het een web app is en enorm veel functionaliteiten door de browser zelf worden voorzien, kan de ontwikkelaar zich meer focussen op de designs, lay-out en business logica van de applicatie.

Op vlak van designs is er natuurlijk ook meer vrijheid omdat de ontwikkelaar met HTML en CSS werkt. Dit zorgt ervoor dat de ontwikkelaar zo goed als alles kan doen wat hij wil. Om hetzelfde design te behalen met de componenten van NativeScript moet er veel meer tijd besteed worden.

Het grootste nadeel van een PWA is tegenstrijdig met het grootste voordeel, maar het is de afhankelijkheid van de browser. Bijvoorbeeld de Push API, deze is nodig om push notificaties te kunnen sturen naar een mobiele telefoon. “Firebase messaging” werkt niet zonder deze API en alle browsers ontwikkeld voor iOS ondersteunen nog geen Push API. Dit zorgt ervoor dat er op geen enkele manier een melding gestuurd kan worden naar een PWA op iOS.

Als laatste is er nog een probleem dat PWA niet onmiddellijk kan oplossen en dat is de *native feel* van een applicatie. Heel veel kleine details zorgen er voor dat NativeScript applicaties beter aanvoelen t.o.v. webapplicaties. Zaken zoals het scrollen van een lijst en *swipe gestures* zijn niet te evenaren met een PWA.

Op vlak van performance zijn PWA’s zeer nauw met de varianten. Het opstarten van de applicatie, inloggen en inladen van alle gegevens duurt nog geen 3 seconden. Dit is even snel als NativeScript.

De PWA maakt het makkelijk om een applicatie te ontwikkelen die cross-platform is. Maar er is op het moment nog geen mogelijkheid om een PWA in de app store te plaatsen. Hierdoor zou de Luci Asssistant app niet aanvoelen als een echte app.

6.2.2 Ontwikkeling NativeScript

Vervolgens zal de NativeScript applicatie ontwikkeld worden met dezelfde view en functionaliteiten.

6.2.2.1 Setup

Voor de ontwikkeling van een NativeScript applicatie moet er nieuw project gegenereerd worden. Dit is mogelijk met één commando:

```
tns create StagePocNativeScript --template tns-template-blank-ng
```

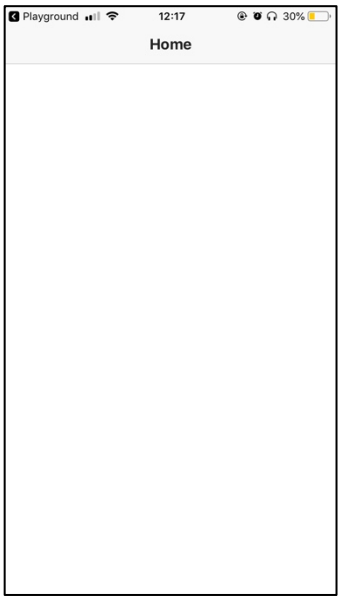
Nadat het commando het project heeft gegenereerd kan er een ander commando gebruikt worden om de NativeScript applicatie meteen te testen op een mobiele telefoon. NativeScript voorziet een applicatie genaamd “Preview” waardoor de applicatie gecompileerd wordt en meteen getest kan worden.

Als de ontwikkelaar navigeert naar het project en het volgende commando invoert:

```
tns preview
```

Zal er een QR-code weergeven worden in de terminal die gescand kan worden en vervolgens meteen bruikbaar is op de mobiele telefoon.

Figuur 36 tns preview QR-code



Figuur 35 Preview App

Het volledige proces duurt de eerste keer langer omdat de NativeScript CLI en de emulators eerst geïnstalleerd moeten worden, maar als dit al gebeurd is duurt de setup van een nieuwe NativeScript applicatie minder dan 2 minuten. Nu dat de setup compleet is kunnen de views ontwikkeld worden.

6.2.2.2 Views

NativeScript maakt gebruik van zijn eigen componenten. Hierdoor moet de ontwikkelaar zich in het begin eerst bekend maken met de verschillende componenten. De reden waarom deze componenten gebruikt worden, is belangrijk voor de onderliggende werking van NativeScript.

Bij het builden en compileren wordt de “JavaScript code” omgezet naar “native code” die op het device zal draaien. Hierdoor zijn NativeScript applicaties even snel als “native apps”.

Op afbeelding 41 is de HTML-code getoond van de home view. Er wordt een combinatie gebruikt van verschillende soorten “Lay-outs” en ander componenten zoals “Labels”, “Progress bar”, “Scroll pickers”, “Activity indicators” enz.

home-view.component.html

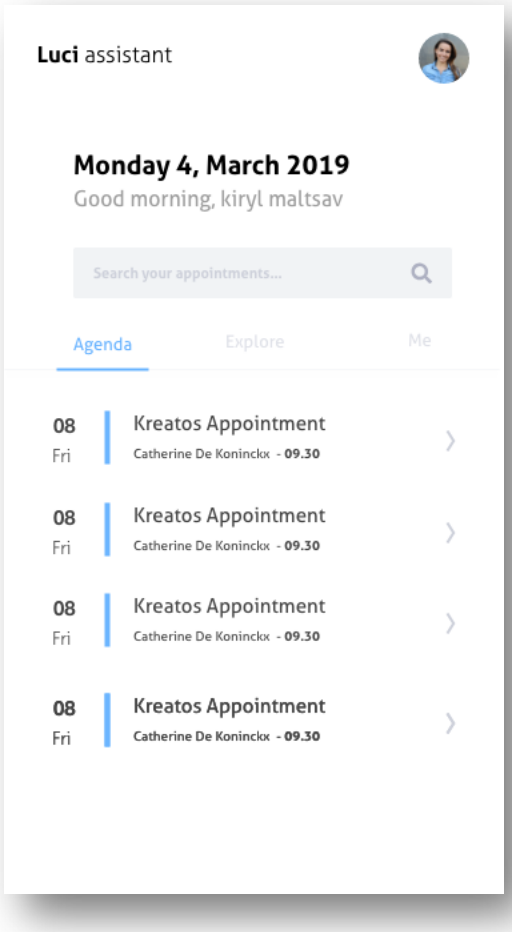
```
<GridLayout rows="auto,*" columns="auto">
    <GridLayout row="0" rows="*,*,*,*" class="navbar" verticalAlignment="top" columns="*,auto,auto,auto">
        <StackLayout col="0" row="0" class="logo-title-container">
            <Label class="logo-title">
                <FormattedString>
                    <Span text="Luci " class="bolded-text"></Span>
                    <Span text="assistant"></Span>
                </FormattedString>
            </Label>
        </StackLayout>
        <Image *ngIf="(authenticatedCustomer == undefined) || (authenticatedCustomer == null) || (authenticatedCustomer.photoUrl == '')"
            col="1" row="0" stretch="aspectFill" class="profile-img" src="~/app/assets/shared/dummycustomer.png"></Image>
        <Image *ngIf="(authenticatedCustomer != undefined) && (authenticatedCustomer != null) && (authenticatedCustomer.photoUrl != '')"
            col="1" row="0" stretch="aspectFill" class="profile-img" [src]="authenticatedCustomer.photoUrl"></Image>
        <StackLayout row="1" colSpan="2" class="homeview-title-container">…
        </StackLayout>
        <StackLayout *ngIf="selectedTabview == 0" row="2" colSpan="2" class="search-textfield-container" orientation="horizontal">…
        </StackLayout>
        <StackLayout *ngIf="selectedTabview == 1" row="2" colSpan="2" class="search-textfield-container" orientation="horizontal">…
        </StackLayout>
        <StackLayout *ngIf="selectedTabview == 2" row="2" colSpan="2" class="search-textfield-container" orientation="horizontal">…
        </StackLayout>
        <StackLayout orientation="horizontal" row="3" colspan="4" class="navigation-items-container">…
        </StackLayout>
    </GridLayout>
    <GridLayout *ngIf="selectedTabview == 0" row="1" width="100%" backgroundColor="white">…
    </GridLayout>
    <GridLayout *ngIf="selectedTabview == 1" row="1" width="100%" backgroundColor="white">…
    </GridLayout>
    <GridLayout *ngIf="selectedTabview == 2" row="1" width="100%" backgroundColor="white">…
    </GridLayout>
</GridLayout>
```

Figuur 37 NativeScript POC Homeview html

Vervolgens kunnen alle componenten gestyled worden door gebruik te maken van een CSS-file of *inline* styling op de component zelf.

Dit zorgt voor een heel vlotte ontwikkeling van de views. Wat wel vaak een probleem is bij het ontwikkelen van mobiele apps, is dat mobiele telefoons verschillende resoluties en aspect ratio's hebben. Hierdoor is het niet makkelijk om alle applicaties responsief te maken. Met het ontwikkelen van deze view was het duidelijk dat de "lay-out" componenten enorm helpen met dit probleem op te lossen. De "lay-out" componenten behouden op elk toestel de gewenste hoogte en breedte.

Het resultaat van dit proces is op afbeelding 38 te zien. Er kan ook wel gezegd worden dat de view sterk lijkt op de designs. Dit is een goed teken dat NativeScript makkelijk is om mobiele applicaties mee te ontwikkelen.



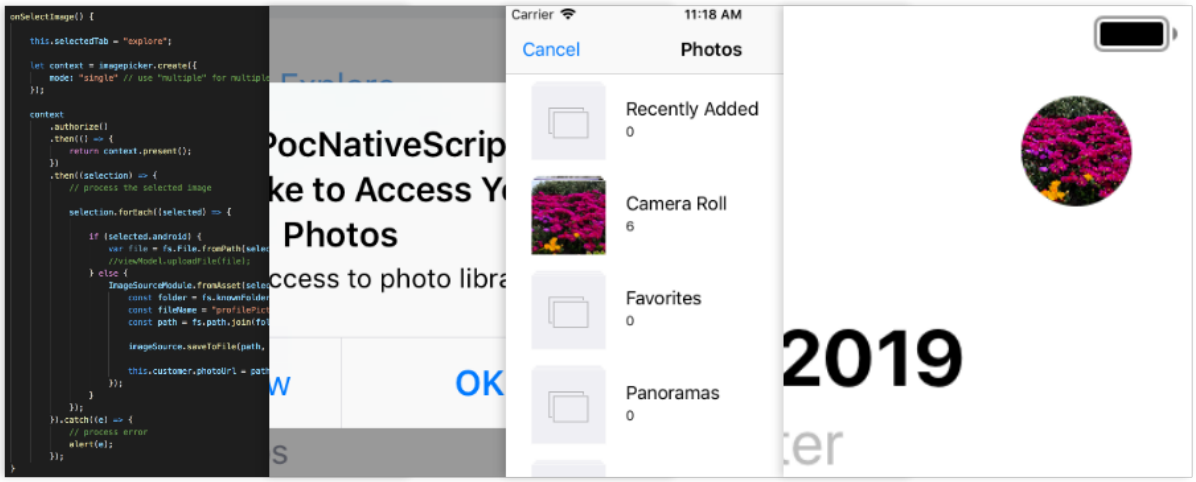
Figuur 38 NativeScript POC Homeview

6.2.2.3 Functies

6.2.2.3.1 Photo library

Voor het gebruiken van de “photo library” moet er eerst een plug-in geïnstalleerd worden die dit mogelijk maakt. De plug-in noemt “nativescript-imagepicker” en zorgt ervoor dat de volledige flow van een afbeelding te selecteren afgehandeld wordt. Voor het gebruiken van de camera moet er nog een extra plug-in geïnstalleerd worden, maar dit gaat in dit onderzoek niet verder uitgewerkt worden omdat het hetzelfde is als het gebruiken van de “photo library”.

Afbeelding 42 toont de volledige flow voor het selecteren en weergeven van een afbeelding. Ook is er te zien dat er weinig code nodig is om een gebruiker een afbeelding te laten kiezen uit zijn bibliotheek en deze weer te geven op het scherm. De code vraagt eerst om toestemming van gebruiker. Dit is nodig om de afbeelding op het toestel te kunnen gebruiken. De volledige UI voor het kiezen van de afbeelding wordt door het toestel zelf voorzien. Vervolgens stuurt de “photo-library” de afbeelding naar de business logica en gaat de code de afbeelding opslaan op de voorziene locatie. Als laatste wordt de afbeelding getoond in de mobiele applicatie.

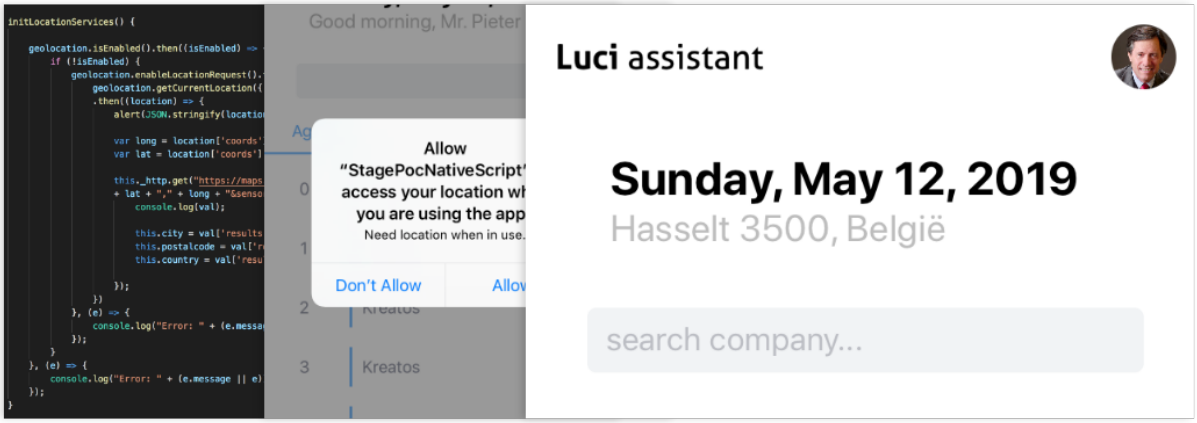


Figuur 39 POC NativeScript Photo library flow

6.2.2.3.2 Locatievoorzieningen

Het opvragen van de locatie van een gebruiker wordt afgehandeld door de “nativescript-geolocatie plug-in”. Deze handelt net zoals de “photo-library” de volledige flow af. Dit houdt in dat er eerst toestemming gevraagd wordt aan de gebruiker om zijn locatie te mogen weten. En vervolgens kunnen de coördinaten opgehaald worden. Vervolgens kan de stad, postcode en land van de gebruiker opgehaald worden door de coördinaten via een API te sturen. Hier wordt de Google Maps API gebruikt.

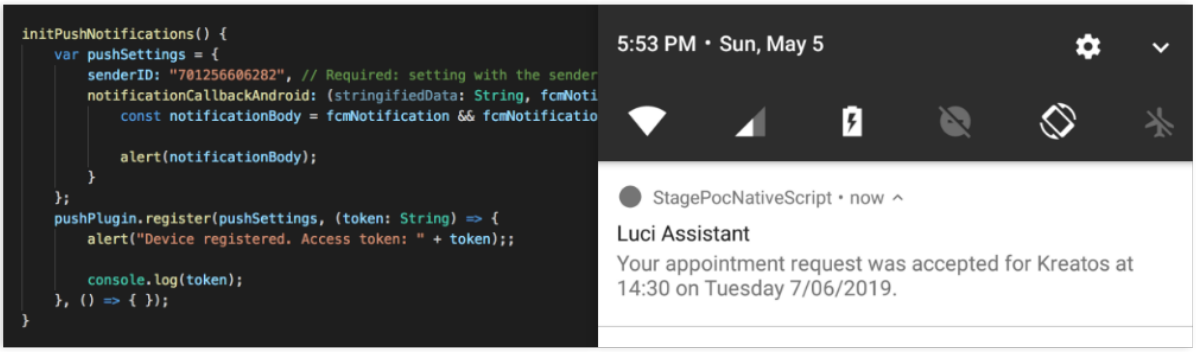
Afbeelding 43 toont de volledige flow van de “locatievoorziening” functionaliteit. De code die de coördinaten opvraagt en toestemming vraagt aan de gebruiker. En vervolgens wordt de locatie door de google API omgezet naar een werkelijke locatie.



Figuur 40 POC Nativescript Locatievoorziening flow

6.2.2.3.3 Push notificaties

De push-notificaties worden net zoals de andere twee features ook mogelijk gemaakt door een plug-in. Dit is de “nativescript-push-notifications” plug-in. Om de notificaties zelf te versturen wordt er gebruik gemaakt van “Firebase Messaging”. Deze is ook gebruikt geweest om de PWA POC push notificaties mogelijk te maken. Ook bij NativeScript wordt er veel logica door het toestel zelf afgehandeld. De gebruiker moet toestemming geven aan de applicatie om de notificaties te mogen sturen. Indien dit gebeurd is krijgt de gebruikt een unieke id. Deze “id” wordt gebruikt om een “request” te kunnen sturen via Firebase naar een specifieke gebruiker. Als laatste wordt de notificatie weergegeven in de applicatie zelf of als de applicatie uit staat in de notificatie lijst van het toestel van de gebruiker.



Figuur 41 POC Nativescript push-notifications

6.2.2.4 Conclusie

NativeScript is een speciale manier van ontwikkelen. Voor zowel alles moet er gebruik gemaakt worden van extra plug-ins. Deze zijn gelukkig goed ondersteund en er is een enorme community die voor de meeste problemen een oplossing klaar heeft staan. Omdat er voor alles een plug-in nodig is, zorgt dat wel voor een veel performantere applicatie. Indien een bepaalde feature niet nodig is dan moet de applicatie deze ook niet implementeren.

Het is soms ongelooflijk hoe vlot de ontwikkeling gaat met NativeSript. Met de Playground app van NativeScript kan er vlot een lay-out ontworpen worden. Ook kunnen er verschillende features/functies uitgetest worden voordat ze toegepast worden in de werkelijke app.

De performance van NativeScript applicaties is opmerkelijk en vergelijkbaar met die van een native app. Dit zorgt ook voor de look en feel van native apps. Het beste voorbeeld is het scrollen door lijsten en het zien van animaties. Alles is vloeiend en voelt goed aan.

Op vlak van features is NativeScript zeker niet af te keuren. Voor zo goed als elke mogelijke feature, bestaat er een plug-in die helpt om de ideeën van een ontwikkelaar tot leven te brengen.

Als laatste is het belangrijk dat er een grote en actieve community bestaat die kan helpen met problemen op te lossen. Dit is bij NativeScript zeker het geval omdat het een *open source* project is, en dat door enorm veel enthousiaste ontwikkelaars onderhouden wordt. Elke week zijn er nieuwe “pull-requests” en “issues” op de Github van NativeScript.

6.3 Vergelijkingsmatrix

Als laatste is er deze vergelijkingsmatrix om concreet te tonen wat de verschillen zijn tussen deze technologieën. Indien er interesse is om één van deze opties te gebruiken, dan kan de volgende tabel gebruikt worden om per project te bepalen welke de beste optie zou zijn. De informatie komt deels van eigen ervaringen, maar is grotendeels ondersteund door externe bronnen en documentatie van de technologieën zelf.

6.3.1 Matrix

Tabel 3 Vergelijkingsmatrix

Features	Progressive Web App [8]	NativeScript [9]
Camera en microfoon		
Audio en video opname	✅	✅
Geavanceerde camera controls	❌	❌
Opname van media/audio	❌	✅
Real-time communicatie	✅	✅
Native behaviors		
Locale Notificaties	❌	✅
Push berichten	❌	✅
Home scherm installatie	✅	✅
Permissies	❌	✅
Operating systeem		
Offline storage	✅	✅
File access	✅	✅
Contacten	❌	✅
SMS	❌	✅

Input		
Touch gestures	✅	✅
Klembord	✅	✅
Pointing device adaptation	✅	✅
Seamless experience		
Offlinemode	✅	✅
Achtergrond Synchronisatie	❌	✅
Betalingen	❌	✅
Credentials	✅	✅
Location en positie		
Geolocatie	✅	✅
Screen en output		
Fullscreen	✅	✅
Scherm oriëntatie en vergrendelen	✅	✅

Conclusie

De keuze tussen NativeScript en PWA is niet zo moeilijk als eerder gedacht. De “schrik” lag bij het ontwerpen van de lay-outs in NativeScript omdat er nieuwe componenten aangeleerd moesten worden. Dit blijkt achteraf niet het geval te zijn, waardoor het uiteindelijk juist makkelijker was om mobiele applicaties te ontwikkelen in NativeScript.

Ook voelen NativeScript applicaties veel beter aan dan een PWA. De *user experience* van een NativeScript app voelt bijna 100% aan als een *native* applicatie. Terwijl de UX van een PWA blijft aanvoelen als een website. Daarom is NativeScript op vlak van *native feel* en *performance* een veel betere keuze.

Het verschil tussen de documentatie tussen de twee is enorm. NativeScript heeft een enorme *dedicated* site voor de documentatie. Terwijl er voor PWA geen enkele site bestaat met meer dan een pagina uitleg. Hier is NativeScript weer de betere keuze om makkelijker app te kunnen ontwikkelen.

Nog een belangerijk vereiste was om de app te kunnen plaatsen in de app-store. Dit zou niet mogelijk zijn met een PWA. NativeScript ondersteunt om de app zowel in de Playstore als de App Store te plaatsen. NativeScript is daarom de betere oplossing voor Luci Assistant.

Wat voor vele ontwikkelaars de doorslaggevende beslissing is, is de community die er bestaat bij een programeer taal/framework. Het verschil tussen PWA en NativeScript is op vlak van community, dag en nacht. Als er een bepaald probleem is bij het maken van een PWA, is het vrijwel onmogelijk om een oplossing te vinden. Er moet al zeer specifiek en lang gezocht worden naar iemand die een gelijkaardig probleem heeft. Terwijl bij NativeScript er een enorme community heerst van ontwikkelaars die helpen aan het onderhouden van het *open-source* project, maar ook elkaar helpen op de forums om een oplossing te vinden voor problemen.

De keuze is duidelijk en de mobiele applicatie van Luci Assistant zal ontwikkeld worden met NativeScript.

Bibliografie

[1] D. Webster, „Progressive Web Apps - adpot, adapt or reject,” 4 October 2018. [Online]. Available: https://www.venndigital.co.uk/blog/progressive-web-apps-adopt-adapt-or-reject-93069/. [Geopend 17 March 2019].

[2] A. Johanna, „seriously-though-what-is-a-progressive-web-app,” 6 Aug 2018. [Online]. Available: https://medium.com/@amberleyjohanna/seriously-though-what-is-a-progressive-web-app-56130600a093. [Geopend 17 March 2019].

[3] E.-E. Papadopoulou, „Native apps or PWAs: Who will prevail? — Poll respondents choose the PWA way,” 15 November 2018. [Online]. Available: https://jaxenter.com/react-native-tool-native-vs-pwa-poll-150667.html. [Geopend 17 Maart 2019].

[4] Wikipedia, „NativeScript,” 14 Maart 2019. [Online]. Available: https://en.wikipedia.org/wiki/NativeScript#Development. [Geopend 17 Maart 2019].

[5] NativeScript, „How NativeScript Works,” [Online]. Available: https://docs.nativescript.org/core-concepts/technical-overview. [Geopend 17 Maart 2019].

[6] Google, „NativeScript,” [Online]. Available: https://trends.google.com/trends/explore?date=today%205-y&q=NativeScript. [Geopend 17 maart 2019].

[7] V. Radeva, „What's on the NativeScript Roadmap for 2019?,” 5 december 2018. [Online]. Available: https://www.nativescript.org/blog/whats-on-the-nativescript-roadmap-for-2019. [Geopend 17 maart 2019].

[8] A. Bar, „What Web Can Do Today,” [Online]. Available: https://whatwebcando.today/. [Geopend 7 April 2019].

[9] NativeScript, „NativeScript Docs,” [Online]. Available: https://docs.nativescript.org. [Geopend 7 April 2019].