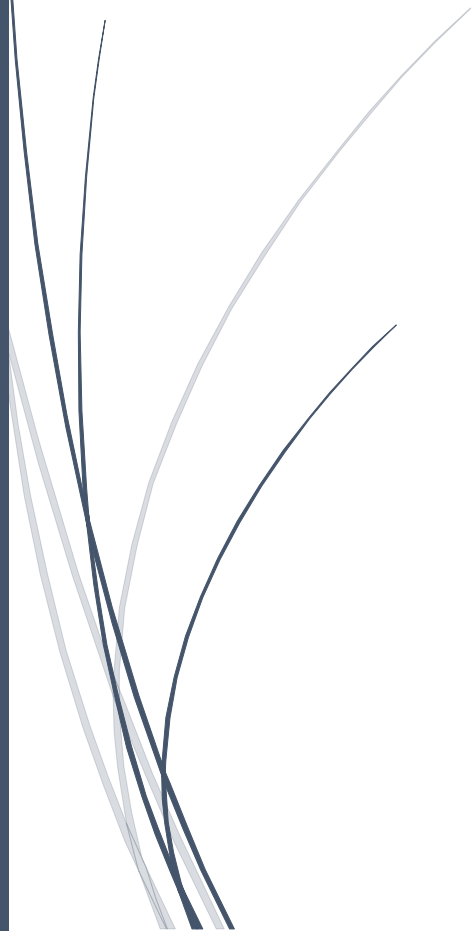
Den Haag, 20/03/2017

Versie 1.5

Afstudeerverslag

Quick Events Xamarin Mvvmcross app

- **School**: De Haagse Hogeschool
- **Opleiding**: Informatica
- **Examinatoren**:
  - G.A. Mijnarends
  - E.M van Doorn



Ravi Gajadin, 13031198
WEBBEAT PRODUCTS BV

Versiebeheer

Versie	Datum	Aanpassingen
0.1	12-12-2016	Hoofdstukken ingedeeld.
0.2	21-12-2016	Organisatiebeschrijving en de opdracht toegevoegd
0.3	06-01-2017	Sprint 1 toegevoegd
0.4	18-01-2017	Sprint 2 ingedeeld
0.5	25-01-2017	Binding library toegevoegd
0.6	05-02-2017	Hoofdstukken ingedeeld
0.7	10-02-2017	Vergelijking hoofdstuk toegevoegd en gebruikte tools
0.8	12-2-2017	Styling toegevoegd en sprint 1 verbeterd
0.9	16-02-2017	Hoofdstukken 2.3 ,3.1, 3.4 ,3.3 verbeterd of tekst aangepast voor duidelijkheid
1.0	03-03-2017	Sprint 3 hoofdstukken toegevoegd
1.1	06-03-2017	Sprint 1 bijgewerkt
1.2	09-03-2017	Sprint 2 bijgewerkt
1.3	10-03-2017	Sprint 3 bijgewerkt en bijlages toegevoegd
1.4	14-03-2017	4.2.9 en 4.2.8 geupdate
1.5	15-03-2017	Sprint 3 laatste hoofdstukken ingevuldt en beoordelingen

Referaat

Afstudeeropdracht “Quick Events Xamarin Mvvmcross app”
Gajadin R. 13031198
Webbeat Products BV, Wateringen 14 November 2016 – 20 Maart 2017

Dit document geeft inzicht in de werkzaamheden van het afstudeerproject van Ravi Gajadin bij Webbeat Products BV te Wateringen, waarbij een Xamarin Mvvmcross Android app gemaakt is waarmee snel afspraken gemaakt kunnen worden voor in de Agenda. Het document gaat in op de werkzaamheden, op welke manier ze zijn uitgevoerd en waarom deze manier gekozen is.

Descriptoren:

- Webbeat
- Xamarin
- Mvvmcross
- Android
- C#
- .NET
- API
- JSON
- Scrum
- UML
- MVVM

Voorwoord

Dit afstudeerverslag is geschreven in het kader van mijn opleiding informatica aan de Haagse Hogeschool, Academie voor ICT & Media. Dit verslag geeft inzicht in de afstudeerperiode van 14 nov. 2016 tot en met 20 mrt. 2017 welk ik heb doorlopen bij Webbeat Products BV.

Ik wil Webbeat bedanken voor de afstudeermogelijkheid die voor mijn gecreëerd is. Ik wil Geert Merkelbach en Leon de Brabander bedanken voor de opdracht en de ondersteuning hiervan. En ten slotte wil ik Maarten van der Meer en Maurice Timp bedanken voor het creëren van en het helpen met het design voor de app.

Dhr. G.A. Mijnarends en Dhr. E.M van Doorn wil ik bedanken voor de ondersteuning vanuit school. Dhr. G.A. Mijnarends als begeleidend examinator en Dhr. E.M van Doorn als tweede examinator.

Ravi Gajadin

Den Haag, Maandag 20 maart 2017

Inhoudsopgave

Versiebeheer i
Referaat ii
Voorwoord iii
Inhoudsopgave iv
Inleiding 1
1. Organisatiebeschrijving 2
1.1. Plaats binnen het bedrijf 2
2. De opdracht 3
2.1. Probleemstelling 3
2.2. Doelstelling 3
2.3. Resultaat 3
2.4. Producten 3
3. Oriënterende fase 4
3.1. Ontwikkelmethode 4
3.2. Planning 5
3.3. Vergelijking Xamarin varianten 5
3.4. Gebruikte tools en methoden 7
3.5. Initiële user stories 9
3.6. Initiële wireframes 10
3.6.1. Categorie en tag stap (US1) 10
3.6.2. Datum en tijd stap (US2) 11
3.6.3. Agenda selectie stap (US3) 11
3.7. Ontwerp 12
3.7.1. Design patterns 13
3.8. Initieel ontwerp database 16
4. Sprints 18
4.1. Sprint 1: Afspraak Aanmaken 18
4.1.1. Periode 18
4.1.2. User stories 18
4.1.3. Definition of Done 18
4.1.4. Contentprovider 19
4.1.5. Extension methods 20
4.1.6. Tonen iconen en eigen lettertype 21
4.1.7. Overige keuzes en problemen 22

4.1.8. Resultaat 23
4.2. Sprint 2: Intelligentie toevoegen 25
4.2.1. Periode 25
4.2.2. User stories 25
4.2.3. Definition of Done 25
4.2.4. Database ontwerp 25
4.2.5. Binding library 26
4.2.6. Kalender component 28
4.2.7. Intelligentie 29
4.2.8. Testen 30
4.2.9. Backend Web API 33
4.2.10. Resultaat 34
4.3. Sprint 3: App klaar maken voor release 36
4.3.1. Periode 36
4.3.2. User stories 36
4.3.3. Definition of Done 37
4.3.4. Database migraties 37
4.3.5. Repository pattern 39
4.3.6. Performance 40
4.3.7. Crashlytics 43
4.3.8. Resultaat 45
5. Evaluatie 47
5.1. Productevaluatie 47
5.1.1. Plan van aanpak 47
5.1.2. Requirements 47
5.1.3. Ontwerp 47
5.1.4. Test rapport 47
5.2. Procesevaluatie 47
5.3. Evaluatie beroepstaken 48
5.3.1. Uitvoeren analyse door definitie van requirements (1.4) 48
5.3.2. Ontwerpen, bouwen en bevragen van een database (2.2) 48
5.3.3. Ontwerpen systeemdeel (3.2) 49
5.3.4. Bouwen applicatie (3.3) 49
5.3.5. Uitvoeren van en rapporteren over het testproces (3.5) 49
6. Begrippenlijst 50

7. Bronnen 51
8. Bijlages 52
Bijlage 1) Afstudeerplan .....
Bijlage 2) Plan van aanpak .....
Bijlage 3) Requirements document .....
Bijlage 4) Functioneel & technisch ontwerp .....
Bijlage 5) Testplan en rapportage .....

Inleiding

Het verslag is opgedeeld in drie delen, in het eerste deel zal er gekeken worden naar het afstudeerbedrijf, wat voor diensten er worden geleverd en zal de stagiair zich plaatsen in het bedrijf. Daarnaast zal de opgestelde afstudeeropdracht worden besproken, welke probleem- en doelstellingen er zijn opgesteld.

Het tweede gedeelte van het verslag wordt opgedeeld in de sprints die er zijn gelopen en wordt er per sprint verteld wat er gedaan is, welke keuzes er gemaakt zijn en welke problemen er waren.

In het laatste deel van dit verslag, zal de evaluatie naar voren komen. Hier is besproken hoe het project is verlopen, zijn de doelstellingen behaald en het uiteindelijk resultaat van de producten.

Deel 1

Het bedrijf en de afstudeeropdracht

1. Organisatiebeschrijving

Webbeat Products B.V. is opgericht in 2000 in Den Haag, nu gevestigd in Wateringen. Webbeat is sinds 2012 onderdeel van Nalta Group te Almere. Samen bieden zij een complete dienstverlening voor projecten op het gebied van hardware en software. Webbeat heeft op het moment van schrijven 20 medewerkers in dienst die onder een platte organisatiestructuur samenwerken aan verschillende projecten.

Hierbij wordt er gebruik gemaakt van een vorm van Scrum als softwareontwikkelmethode, waarbij (interne) communicatie en overleg centraal staat. De medewerkers werken vaak in kleine groepjes of alleen aan verschillende opdrachten.

Webbeat is gespecialiseerd in het ontwikkelen van webapplicaties op basis van Microsoft technologie. Er worden daarnaast ook mobiele app's ontwikkeld voor iOS en Android. De meeste projecten worden in opdracht van andere bedrijven uitgevoerd. Enkele bekende klanten waar projecten voor zijn uitgevoerd zijn Heineken, PostNL en NPO, maar Webbeat ontwikkeld ook eigen producten die worden aangeboden als onlinedienst (SAAS).
Een van deze producten is Datumprikker. Datumprikker is erop gericht om zo snel en efficiënt mogelijk afspraken te plannen met andere personen. Er zijn inmiddels al meer dan 8 miljoen afspraken gemaakt met Datumprikker.

1.1. Plaats binnen het bedrijf

Ik heb de functie van App Developer vervuld en werd ondersteund door Leon de Brabander die dezelfde functie vervult. Omdat er een platte organisatiestructuur is, was er geen sprake van een afdeling. Verder was het altijd mogelijk om vragen te stellen via e-mail, Slack of andere communicatiekanalen.

2. De opdracht

2.1. Probleemstelling

De standaard agenda app die wordt meegeleverd met Android wordt door onvoldoende mensen gebruikt, doordat het maken van een afspraak saai is en vaak ook erg lang duurt, doordat er veel getypt moet worden.

2.2. Doelstelling

De doelstelling van het project is een Xamarin Android app die zich alleen richt op het aanmaken van een afspraak. Dit aanmaak proces moet leuk en aantrekkelijk zijn waardoor er bijvoorbeeld in minder dan 10 seconden een afspraak gemaakt kan worden.

2.3. Resultaat

Het resultaat is een Android Xamarin app, die op zichzelf staand aangeboden kan worden in de Google Play Store. In de app is het alleen de bedoeling om afspraken te maken die in de lokale agenda app opgeslagen kunnen worden of via een API-koppeling gedeeld kunnen worden.

Deze afspraken kunnen gemaakt worden door een aantal stappen door de app te volgen, waarin de belangrijkste onderdelen van een afspraak ingevuld worden zoals de titel, datum en dergelijke. De app moet responsive zijn en een goede gebruikers ervaring bieden.

2.4. Producten

De volgende(tussen)producten zullen opgeleverd worden tijdens het project:

- Plan van aanpak
- De Requirements
- Ontwerp
- Testrapportage en plan
- De applicatie

3. Oriënterende fase

In de eerste twee weken van het project heb ik me voornamelijk gericht op wat ik precies ging maken en hoe ik dat ging aanpakken. De planning die daarbij gebruikt is ziet er als volgt uit:

	X - 14-11-2016 / 20-03-2017				Week 1 14-18					Week 2 21-25				
Nr	**Taken**	**Begindatum**	**Einddatum**	**Weken**	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr
1	**Planning**	**14-11-2016**	**25-11-2016**	**2,00**										
1.1	Werkomgeving instellen	14-11-2016	15-11-2016											
1.2	Plan van aanpak maken	15-11-2016	18-11-2016											
1.3	Vaststellen specificaties en requirements	17-11-2016	22-11-2016											
1.4	Use cases maken	17-11-2016	22-11-2016											
1.5	Meeting met Leon	14-11-2016	25-11-2016											
1.6	Onderzoek MvvmCross of Xamarin.Forms of Native Android	22-11-2016	25-11-2016											
1.7	Taken indelen sprint 1 Leon & Geert	24-11-2016	25-11-2016											

Figuur 3-1, planning eerste 2 weken

3.1. Ontwikkelmethode

Om ervoor te zorgen dat de communicatie tussen mij en Webbeat zo soepel mogelijk kon verlopen, was het verstandig om de ontwikkelmethodes zo veel mogelijk te laten overeenkomen.

Als ontwikkelmethode had ik zelf een voorkeur voor SCRUM. SCRUM is een framework voor agile management van softwareontwikkeling. Omdat het een methode is die ik ook op school veel heb gebruikt bij projecten en ik daarom weet waar ik aan toe ben. Ook Webbeat gebruikt een vorm van SCRUM en daardoor was de keuze snel gemaakt.

In tegenstelling tot SCRUM waar sprints meestal een lengte van twee of drie weken hebben. Heb ik gekozen om sprints van vier weken te hanteren, omdat er met nieuwe technieken gewerkt zou worden en ik alleen de app zou moeten maken. Vier weken zou mij genoeg tijd geven om aan het eind van elke sprint een goed product op te leveren waarin de meeste taken voor die sprint in verwerkt zijn. De definitie van het product wat opgeleverd zal worden aan het eind van elke sprint zal afgesproken worden in de Definition of Done. Voor elke sprint zal er aan het begin vastgesteld worden welke taken er behandeld gaan worden. Deze taken worden door de opdrachtgever geprioriteerd door middel van de MoSCoW methode (Hoofdstuk 3.4 Gebruikte tools en methoden).

Er zal in tegenstelling tot Scrum niet gewerkt worden in teamverband, maar door alles goed vast te leggen kan er wel een goed overzicht gehouden worden door de bedrijfsmentor en de opdrachtgever. Ook zijn er geen daily meetings maar is er elke week op woensdag een vast moment waarop iedereen verteld waaraan hij of zij bezig is en of er problemen zijn. Aan het eind van elke sprint zal er een sprint retrospective plaatsvinden waarbij gekeken wordt of behaald is wat is afgesproken in de Definition of Done.

Ter ondersteuning van het SCRUM-proces zal er gebruik gemaakt worden van JIRA, meer uitleg over wat JIRA is en hoe het gebruikt is staat in Hoofdstuk 3.4 Gebruikte tools en methoden

3.2. Planning

De planning voor sprint 1 ziet er als volgt uit:

Nr	Taken	Week 3 28-02 Ma	Di	Wo	Do	Vr	Week 4 05-09 Ma	Di	Wo	Do	Vr	Week 5 12-16 Ma	Di	Wo	Do	Vr	Week 6 19-23 Ma	Di	Wo	Do	Vr
2	**Sprint 1**																				
2.1	Meeting met designers om 13:00	■					■					■					■				
2.2	Meeting met Leon	■		■			■		■			■		■					■		
2.3	Documentatie maken	■	■	■	■	■	■	■	■	■	■										
2.4	App maken	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■				
2.5	Testen maken			■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
2.6	Feedback en bugs verwerken																■	■	■	■	■
2.7	App testen								■	■	■	■	■	■	■	■	■	■	■	■	■
2.8	Vaststellen requirements en taken Sprint 2 Leon & Geert																			■	■

Figuur 3-2, planning sprint 1

Deze planning is voortgezet in het plannen van de overige sprints. Het idee achter deze planning was om eerst te beginnen met het maken van een gedeelte van de benodigde documentatie en die om te zetten naar functionaliteit voor de app. Dit zorgde ervoor dat een groot gedeelte van de documentatie meestal in de eerste week klaar was waarna er gefocust kon worden op de app zelf.

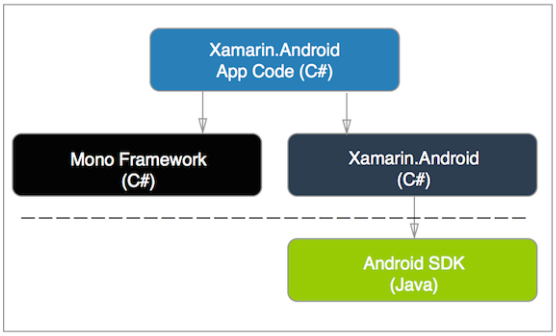
3.3. Vergelijking Xamarin varianten

Omdat er al afgesproken is in het afstudeerverslag dat de app in Xamarin gemaakt zal worden. Was het nu alleen nog nodig om te kijken welke variant van Xamarin er gebruikt zou worden, Om die rede is er een vergelijking gemaakt tussen de volgende meest gebruikte Xamarin varianten:

- Xamarin Android
- Xamarin Forms
- Xamarin en Mvvmcross

Er was ook een mogelijkheid om Xamarin met ReactiveUI te combineren. Deze is alleen niet meegenomen in de vergelijking, omdat die variant minder gebruikt werd dan de andere drie.

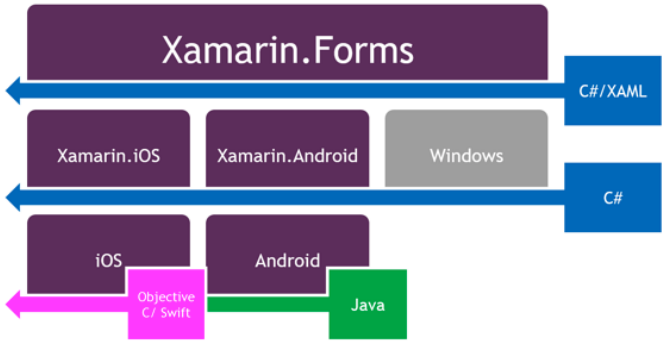
Xamarin Android



Figuur 3-3, Overzicht Xamarin.Android app [http://sharpmobilecode.com/nuts-bolts-of-xamarin-android/]

Xamarin.Android is een C# wrapper voor de Android SDK. Dit houdt in dat het dezelfde mogelijkheden heeft als Java Android zolang er een wrapper gemaakt is voor de betreffende SDK, het enige verschil is de programmeertaal die gebruikt wordt en dat het soms een tijdje kan duren voordat je de nieuwste Android SDK kan gebruiken.

Xamarin Forms

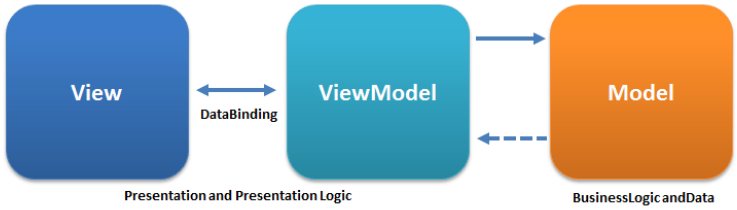


Figuur 3-4, Overzicht Xamarin.Forms app

Xamarin.Forms is een framework die ervoor zorgt dat ontwikkelaars snel cross platform user interfaces kunnen maken. Dit houdt in dat Xamarin.Forms zelf een aantal UI-componenten heeft die op Android, iOS, Windows en Windows Phone werken wat ervoor zorgt dat er maar één user interface gemaakt hoeft te worden.

Xamarin Mvvmcross

Mvvmcross is een Open source cross platform framework, opgestart in november 2011 door Stuart Lodge. Voor Xamarin.iOS, Xamarin.Android, Windows en Mac. Het maakt gebruikt van het MVVM pattern wat door Microsoft ontwikkeld is.



Figuur 3-5, Mvvmcross Architectuur

MVVM bestaat uit drie onderdelen: Model(M), View(V) en de View-Model (VM). De View bevat de platform (Android, iOS, Windows) specifieke code zoals: Views en platform specifieke API's (Kalender, Contacten). De View-Model en Model zijn crossplatform. Waarbij de View-Model ervoor zorgt dat er communicatie mogelijk is tussen de View en het crossplatform gedeelte van de app en de Model bestaan uit modellen.

Dit allemaal zorgt ervoor dat er met behulp van Mvvmcross apps gemaakt kunnen worden die een native UI hebben maar dat er toch vrij veel code hergebruikt kan worden.

Voor- en nadelen

Xamarin.Android kan het best gebruikt worden voor:

- Apps die een complex design hebben
- Apps die veel platform specifieke functionaliteit nodig hebben
- Apps waar de UI belangrijker is dan het kunnen hergebruiken van code

Xamarin.Forms kan het beste gebruikt worden voor:

- Data invoer apps
- Prototypes en proof of concept apps
- Apps die bijna geen platform specifieke functionaliteit nodig hebben
- Apps waar het hergebruiken van code belangrijker is dan de UI

Xamarin en Mvvmcross kan het beste gebruikt worden voor:

- Apps waar een complexe UI gebruikt gaat worden, maar waar er ook hergebruik van code moet plaatsvinden.
- Apps die veel platform specifieke functionaliteit nodig hebben
- Apps waar het design van de app niet direct beschikbaar is

Resultaat

De volgende punten kwamen bij de vergelijking naar voren:

- Doordat ze allemaal gebruik maken van Xamarin moet er gewerkt worden met C#.
- Van de drie lijkt Xamarin Android het meest op Java Android
- Xamarin Mvvmcross kan gebruikt worden om apps te maken met een native UI per platform en een cross platform gedeelte voor het delen van code.
- Xamarin Forms kan het best gebruikt worden voor prototypes en proof of concept apps.
- Xamarin Forms en Xamarin Mvvmcross hebben hetzelfde doel met betrekking tot het snel crossplatform ontwikkelen van apps, alleen doen zei dit op een verschillende manier. Mvvmcross zorgt ervoor dat er een native UI gemaakt kan worden waar Xamarin Forms ervoor kiest om de UI te maken van componenten die op alle platformen gebruikt kunnen worden.

Op basis van de resultaten en de vergaarde informatie is de keuze gemaakt om Xamarin.Android te laten vallen, omdat als de Android app in de toekomst aan zou slaan er zo snel mogelijk een iOS app gemaakt zou kunnen worden, iets wat met Xamarin.Android niet mogelijk is.

Er is uiteindelijk gekozen voor Xamarin en Mvvmcross, omdat de UI zo vloeiend mogelijk moet werken om de gebruiker zo snel mogelijk een afspraak te laten maken. En omdat er samengewerkt zou worden met designers, waardoor het design van de app niet direct beschikbaar en er gebruik gemaakt gaat worden van platform specifieke api's voor het werken met de kalender. Wat minder eenvoudig is met Xamarin Forms doordat er gebruikt gemaakt moet worden van plugins en die worden niet altijd even goed onderhouden.

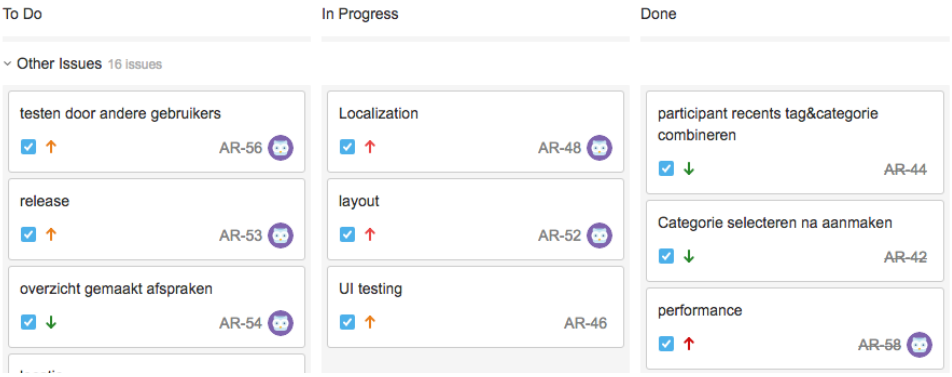
3.4. Gebruikte tools en methoden

Naast Visual Studio en Visual Paradigm voor het ontwikkelen van de app en het maken van diagrammen en SCRUM als ontwikkelmethodologie en UML als modeleer techniek zijn er ook andere softwareprogramma's en technieken gebruikt die in dit hoofdstuk nader toegelicht zullen worden.

JIRA

JIRA is een SAAS-pakket waarmee het SCRUM-proces ondersteund kan worden. Het vervult de rol van supportsysteem en planningssysteem. In JIRA worden alle facetten van alle projecten bijgehouden, ook als er dingen misgaan. Zo kan bijvoorbeeld teruggezocht worden of en wanneer requirements zijn toegevoegd. JIRA is gekozen, omdat dat ook bij Webbeat gebruikt wordt.

Een voorbeeld van de taken is te zien in Figuur 3-6.



Figuur 3-6, JIRA-taken sprint 3

Bitbucket

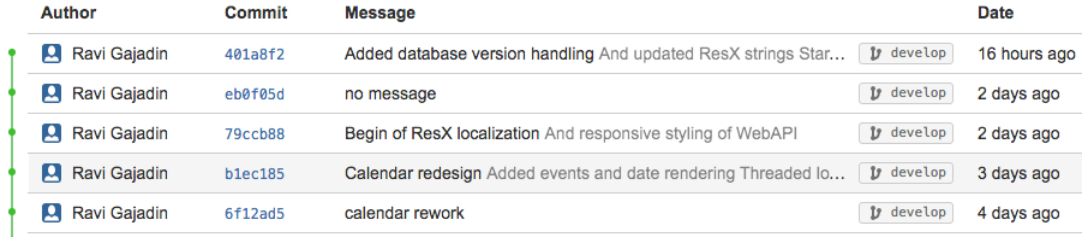
Met versiebeheer is het mogelijk om veranderingen in een bestand of een groep van bestanden bij te houden, zodat later specifieke versies opgevraagd kunnen worden.

In het project heb ik versiebeheer toegepast met behulp van BitBucket, omdat het ook gebruikt wordt door Webbeat. Hierbij heb ik de code ontwikkeld in een development branch en heb ik na elke sprint een release gemaakt. Een release is een stabiele versie die naar de master branch gepushed wordt. Hierdoor was de master branch altijd gevuld met een stabiele versie en heeft de development branch de laatste wijzigingen.

Om nieuwe functionaliteiten uit te proberen, heb ik een enkele keer een feature branch aangemaakt waarin dit gebouwd werd. Hierdoor kon ik deze tijdelijke branch eenvoudig verwijderen als een oplossing niet bleek te werken en samengevoegd worden met de development branch als het wel bleek te werken.

Door versiebeheer te gebruiken binnen mijn project heb ik de volgende voordelen ervaren:

1. De code kon op verschillende ontwikkelomgevingen opgehaald en opgeslagen worden.
2. Als er aanpassingen gedaan waren die achteraf niet goed werkte, kon er eenvoudig een vorige versie van een bestand of het project teruggezet worden.



Figuur 3-7, Lijst met commits in development branch

MoSCoW methode
De MoSCoW methode is een manier van prioriteren vaak gebruikt bij het ontwikkelen van software, waarmee een taak of eis(requirement) een prioriteit heeft.
MoSCoW is een afkorting en staat voor:

- **M**ust have**:** deze eisen moeten in het eindresultaat terugkomen, zonder is het product niet bruikbaar.
1. **S**hould have**:** Deze eisen zijn gewenst, maar zonder zal het product ook wel bruikbaar zijn
2. **C**ould have: Er zal alleen naar deze eisen gekeken worden als er tijd over is.
3. **W**ould have: Deze eisen komen niet aan bod, maar zijn handig om te weten voor een eventueel vervolgproject of in de toekomst.

De letters 'o' hebben geen betekenis. Een project waar niet alle Must haves zijn gehaald wordt gezien als een gefaald project.

3.5. Initiële user stories

Hieronder staan alle user stories die tijdens de oriënterende fase naar voren zijn gekomen. Er is gekozen voor user stories, omdat een user story makkelijker omgezet kan worden naar een taak voor in de sprint backlog door de wijze waarop user stories opgebouwd worden, namelijk Als xxx wil ik xxx, zodat/omdat xxxx.

Deze user stories zijn samen met de opdrachtgever en begeleider opgesteld. Door het vergelijken van de standaard agenda app (Google Calendar) met andere agenda apps en het kijken hoe die een afspraak maken en door zelf te kijken wat er echt nodig is in het proces voor het maken van een afspraak, wat minder belangrijk is en de voorkeur voor een aantal stappen van de opdrachtgever.

Er zijn in de user stories enkele termen gebruikt om de onderlinge communicatie makkelijker te maken, omdat die ook gebruikt worden bij Webbeat voor de Datumprikker applicatie namelijk: Categorie en Tag.

- Categorie heeft dezelfde betekenis als het Nederlands woord Categorie namelijk. Soort of dingen die bij elkaar horen. Een categorie kan meerdere tags bevatten.
- Tag is een ander woord voor titel. Een tag kan bijvoorbeeld het volgende zijn: Sporten, Eten, Vergadering enzovoort. Een tag hoort bij een categorie.

Requirement	ID	Omschrijving
FR1	US1	Als gebruiker wil ik een categorie en tag selecteren, zodat ik een afspraak aan kan maken
FR1	US2	Als gebruiker wil ik een dag en tijdstip selecteren, zodat ik een afspraak aan kan maken
FR1	US3	Als gebruiker wil ik een agenda kunnen selecteren, zodat ik de afspraak aan een agenda kan toevoegen
FR2	US4	Als gebruiker wil ik een categorie kunnen aanmaken, omdat de bestaande categorieën niet voldoen.
FR3	US5	Als gebruiker wil ik een categorie kunnen bewerken, omdat de naam van de categorie niet klopt.

FR4	US6	Als gebruiker wil ik een categorie kunnen verwijderen, omdat ik de categorie niet nodig heb.
FR5	US7	Als gebruiker wil ik een tag kunnen aanmaken, omdat de bestaande tags niet voldoen.
FR6	US8	Als gebruiker wil ik een tag kunnen bewerken, omdat de naam van de tag niet klopt.
FR7	US9	Als gebruiker wil ik een tag kunnen verwijderen, omdat ik de tag niet nodig heb.
FR8	US10	Als gebruiker wil ik een standaard agenda kunnen selecteren, omdat ik in de app altijd maar 1 agenda ga gebruiken.
FR9	US11	Als gebruiker wil ik een afspraak kunnen delen met andere, zodat ik iemand kan uitnodigen die geen contact is.

Tabel 3-1, Initiele user stories

Extra uitleg US3
US3 is een stap in de applicatie, omdat één Google account meerdere agenda's kan hebben. Zo kan een gebruiker met maar één Google account een agenda hebben voor privé en zakelijk gebruik. Er moet dus een optie zijn om te kiezen waar de afspraak aan toegevoegd wordt, doordat er meerdere agenda's zijn kan een afspraak ook toegevoegd worden aan alle agenda's.

3.6. Initiële wireframes

In deze paragraaf zullen er een aantal wireframes getoond worden die de basis van de app vormen en gebruikt zijn bij het bouwen van de app. Deze wireframes zijn samen met designers gemaakt en zij hebben op basis van deze wireframes het uiteindelijke design gemaakt.

3.6.1. Categorie en tag stap (US1)

Dit is de eerste stap die de gebruiker zal zien als hij of zij de app opstart. Waarin er een categorie en tag geselecteerd moet worden.

De categorieën zullen weergegeven worden op de volgende wijze:

- In rijen van 3 of 4 afhankelijk van de grootte van het scherm
- Elke categorie heeft een icoon
- Na X rijen categorieën, wordt het categorieën gedeelte schuifbaar.
- Een lijn na de laatste rij met categorieën
- De knop om een nieuwe categorie aan te maken wordt weergegeven aan het eind van een rij als dat past en anders gecentreerd in een nieuwe rij.

Figuur 3-8, stap 1 wireframe

De tags zullen weergegeven worden op de volgende manier:

- X tags naast elkaar afhankelijk van de grootte van het scherm en de grootte van de tags.
- In rijen onder elkaar.
- Na X rijen tags, wordt het tags gedeelde schuifbaar
- De knop om een nieuwe tag aan te maken wordt altijd gecentreerd in een nieuwe rij weergegeven.

Onderaan het scherm zullen de navigatie knoppen getoond worden.

3.6.2. Datum en tijd stap (US2)

Dit is de 2de stap die de gebruiker te zien krijgt waarin er een dag en tijd geselecteerd moet worden.

Dag selectie wordt op de volgende manier weergegeven:

- In de vorm van een maand kalender
- Er kan heen en weer genavigeerd worden in de kalender

De tijden worden op de volgende manier weergegeven:

- Standaard drie tijden voor de drie dag delen: Ochtend, Middag en Avond.
- Er is ook een 4de optie beschikbaar om zelf een tijd in te voeren.

Onderaan het scherm zullen de navigatie knoppen getoond worden.

Figuur 3-9, stap 2 wireframe

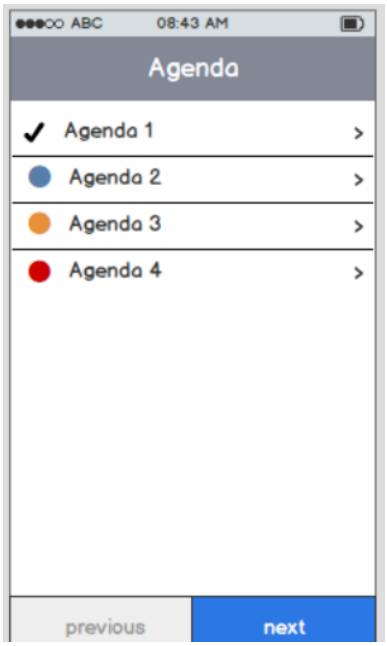
3.6.3. Agenda selectie stap (US3)

Dit is de 3de stap die de gebruiker te zien krijgt waarin er minimaal één agenda geselecteerd moet worden waaraan de afspraak toegevoegd wordt.

De agenda's worden op de volgenden manier weergegeven:

- Elke agenda waar de gebruiker dingen aan kan bewerken, wordt getoond met de naam ervan en een cirkel ervoor met de agenda kleur.
- Geselecteerde agenda's krijgen een vink achter hun naam.

Onderaan het scherm zullen de navigatie knoppen getoond worden.

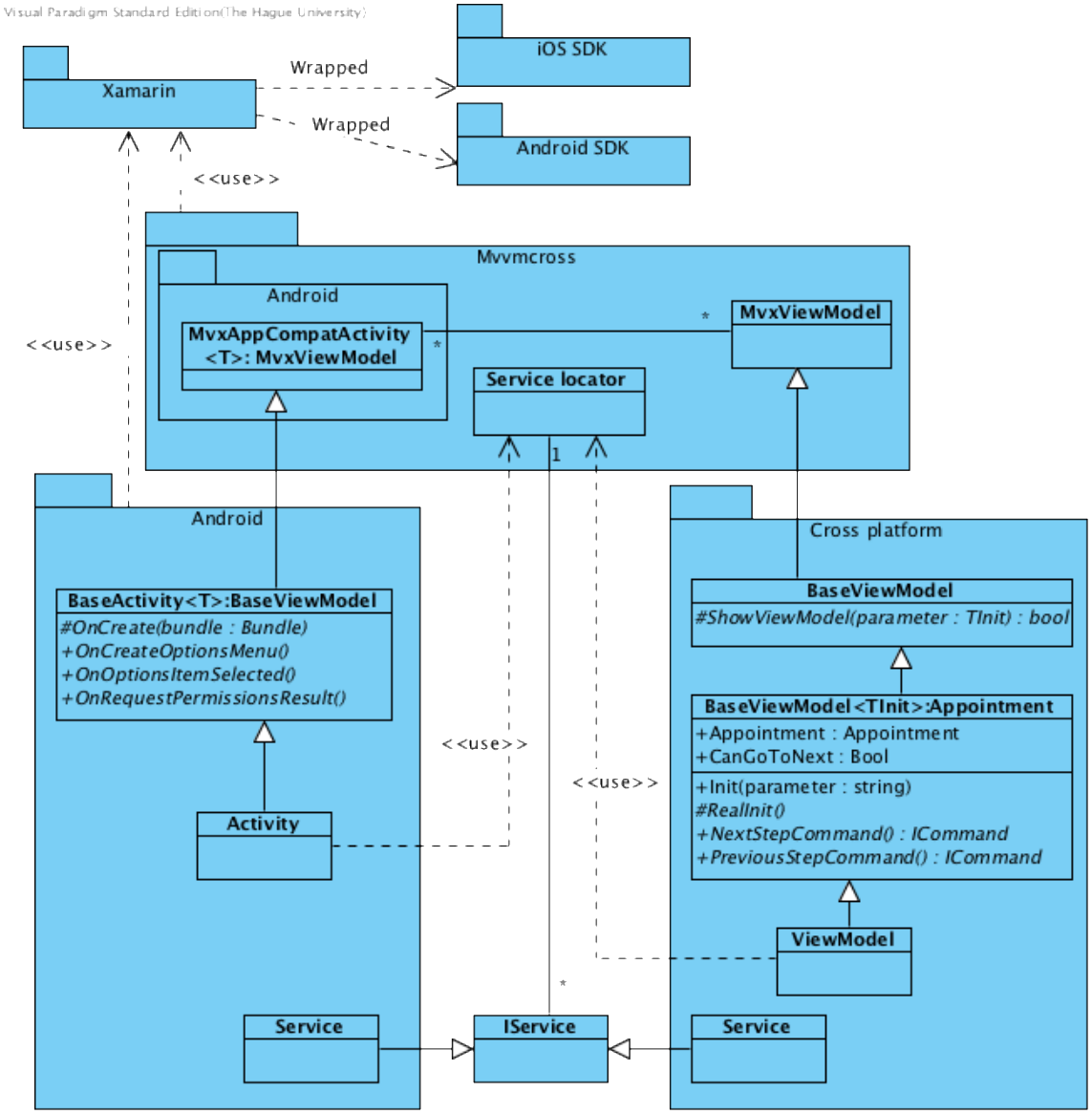


Figuur 3-10, stap 3 wireframe

3.7. Ontwerp

Nadat de initiële requirements opgesteld waren, kon er een ontwerp gemaakt worden. Hierbij is rekening gehouden met de volgende aspecten:

- Er moet een scheiding zijn tussen het platform specifieke en crossplatform gedeelte.
- Er moet gebruik gemaakt worden van de design patterns die Mvvmcross aanbiedt.
- Het moet mogelijk zijn om crossplatform en platform specifieke services te hebben.
- Het moet een generiek ontwerp zijn wat hergebruikt kan worden voor de verschillende views in de app.



Figuur 3-11, Ontwerp app

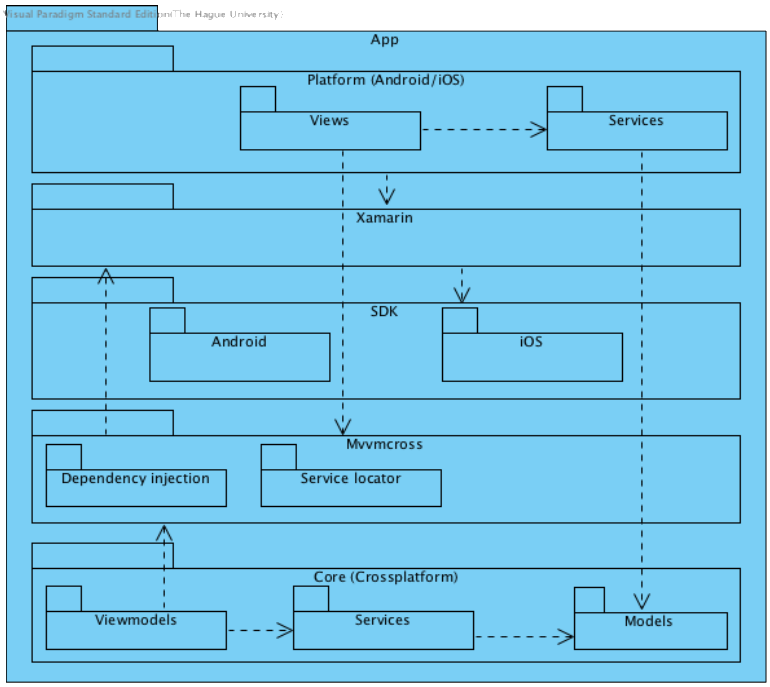
Op basis van deze aspecten en de beschikbare requirements, is er een ontwerp opgesteld (Figuur 3-12). Dit ontwerp bestaat uit 3 gedeeltes namelijk het Android (platform specifiek) gedeelte, het crossplatform gedeelte en Mvvmcross.

In het Android (platform specifiek) gedeelte komt alle code te staan wat afhankelijk is van het platform waar de app op draait, In dit geval dus Android, maar dit kan ook iOS of zelfs Windows Phone zijn.

In het cross platform gedeelte komt de code te staan die onafhankelijk is van het platform, je kan hierbij denken aan de ViewModels, Models en database services.

Mvvmcross zit tussen het platform en cross platform gedeelte en zorgt ervoor dat ze met elkaar kunnen communiceren en van elkaar gebruik kunnen maken.

Dit ontwerp is zo gemaakt dat het hergebruikt kan worden voor alle views in de app. In het ontwerp wordt ook gebruik gemaakt van het Service locator pattern (hoofdstuk 3.7.1 Design patterns) voor het runtime verkrijgen van services die platform onafhankelijk zijn of juist platform afhankelijk. Met dit ontwerp is het ook vrij eenvoudig om code te hergebruiken, omdat het enige wat verandert hoeft te worden de platform specifieke delen zijn. Een versimpelde weergave is te zien in figuur 3-12



Figuur 3-12, Lagen model app

3.7.1. Design patterns

Bij het maken van het ontwerp en de realisatie van de applicatie zal er gebruik gemaakt worden van design patterns. Design patterns zijn generieke oplossing voor veelvoorkomende software ontwerp problemen.

De patterns die gebruikt gaan worden in de applicatie worden in de volgende paragrafen besproken.

3.7.1.1. Singleton pattern

Het singleton pattern zorgt ervoor dat er maar één object van een klasse kan bestaan. Het singleton pattern zal in de app onder andere gebruikt worden voor het verkrijgen van een connectie met de database en het eenmalig inladen van Typefaces (Hoofdstuk 4.1.6 tonen iconen en eigen lettertype). Het eenmalig inladen van Typefaces is nodig, omdat een typeface meerdere keren per view gebruikt wordt en het dan niet nodig is om telkens weer een aantal bestanden in te laden en op te slaan in het geheugen.

Het toepassen van het singleton pattern kan met behulp van Mvvmcross op een vrij eenvoudige manier namelijk:

```
Mvx.RegisterSingleton<IFontService>(new FontService);
```

Hetgeen wat bovenstaande methode doet is een object koppelen aan een interface.

Er kan vervolgens een object verkregen worden met de volgende methode:

```
Mvx.Resolve<IFontService>();
```

Deze methode zorgt ervoor dat het object wat in het eerste voorbeeld is toegekend geretourneerd wordt inplaats van een nieuw object.

3.7.1.2. Dependency injection pattern

Dependency injection is een vorm van het inversion of control pattern. Het inversion of control pattern wordt gebruikt om de koppeling tussen klassen te verminderen.

Dependency injection kan op 3 verschillende manieren toegepast worden:

- Constructor injection
- Setter injection
- Interface injection

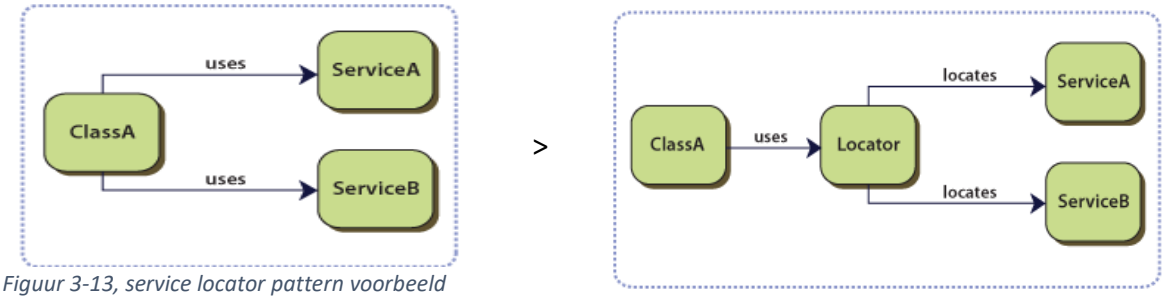
In de app zal gebruik gemaakt worden van constructor injection, omdat het ervoor zorgt dat klassen zich niet druk hoeven te houden met de vereiste parameters van de klassen die het gebruikt.
Een voorbeeld van constructor injection in het project ziet er als volgt uit:

```
private readonly ITagService tagService;
private readonly ICategoryService categoryService;
public DateAndTimeViewModel(ITagService tagService,
                            ICategoryService categoryService)
{
       CanGoToNext = false;
       this.tagService = tagService;
       this.categoryService = categoryService;
}
```

Het gebruiken van constructor injection zorgt ervoor dat in dit geval mijn DateAndTime-ViewModel zich niet druk hoeft te maken om eventuele parameters die mijn services nodig hebben, omdat dat allemaal afgehandeld wordt door het dependency injection framework. Dit zorgt er ook voor dat mijn DateAndTimeViewModel een lage koppeling heeft met mijn services waardoor ik de implementatie van mijn services kan aanpassen zonder iets in de viewmodel aan te passen.

3.7.1.3. Service locator pattern

Het service locator pattern zorgt ervoor dat een klasse, toegang krijgt tot objecten van service klassen zonder dat de klasse deze zelf hoeft te instantiëren. Dit is vooral handig als je services hebt die een andere implementatie hebben afhankelijk van het platform. Zo wordt er in de app gebruik gemaakt van onder andere een CalendarService klasse waar de implementatie afhankelijk is van het platform (Android, iOS) waar de app op draait. Dit pattern zorgt er tevens ook voor dat er een lage koppeling is tussen een service en de klasse die het gebruikt, omdat de service implementatie aangepast kan worden zonder andere klassen aan te passen.



Figuur 3-13, service locator pattern voorbeeld [https://msdn.microsoft.com/en-us/library/ff648968.aspx

Figuur 3-13 laat het verschil zien tussen een klasse die niet gebruikt maakt van een service locator en dus zelf de juiste services moet instantiëren en een klasse die daar wel gebruik van maakt.
Het service locator pattern wordt gebruikt, omdat er een aantal klassen zullen zijn die gebruik maken van services. Waar handmatig een object van aangemaakt zal worden, waardoor het niet mogelijk is om Constructor injection toe te passen. Het service locator pattern zal dus vrij schaars gebruikt worden, maar ondanks dat is het alsnog nodig voor de functionaliteit.

Met behulp van mvvmcross is het vrij eenvoudig om dit pattern te gebruiken doordat er maar drie dingen gedaan hoeven te worden om een service toe te voegen aan de service locator:

- Er moet een interface gemaakt worden van de service
- Er moet een klasse zijn die het implementeert
- Interface en klasse moeten geregistreerd worden door het gebruiken van de volgende Mvvmcross methode waarde betreffende interface en implementatie aan meegegeven wordt.

```
Mvx.RegisterType<ICalendarService,CalendarService>();
```

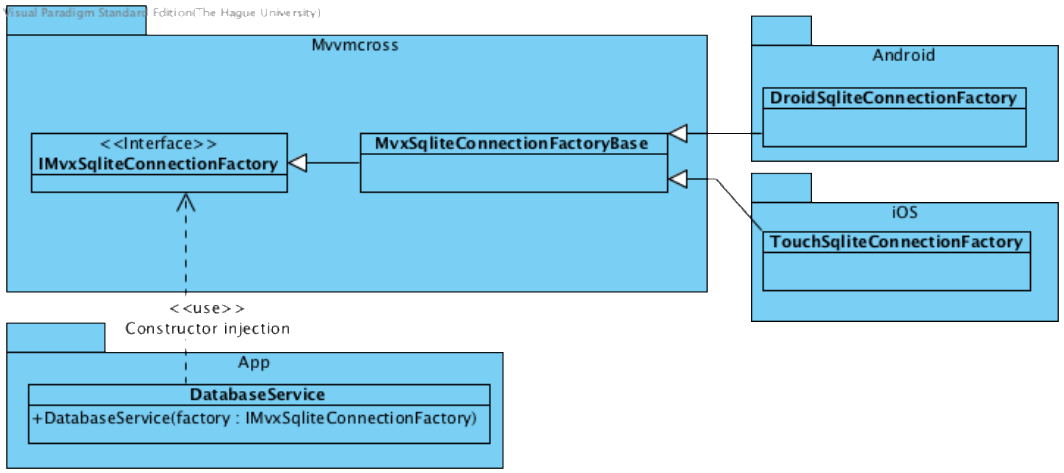
Waarna er doormiddel van onderstaande methode een object van de betreffende service verkregen kan worden.

```
Mvx.Resolve<ICalendarService>();
```

De objecten verkregen worden met deze methode zijn altijd hetzelfde, doordat Mvvmcross onderwater het singleton pattern implementeert in het service locator pattern.

3.7.1.4. *Abstract factory pattern*

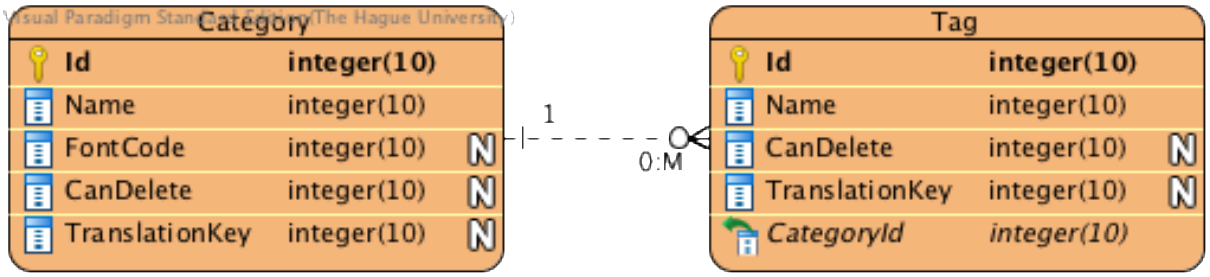
Het abstract factory pattern wordt gebruikt om runtime de juiste sqliteconnectionfactory te instantieren, omdat die een object aanmaken waarmee een connectie gemaakt kan worden met de SQLite database, dit is nodig omdat het maken van een connectie verschilt per platform. Een voorbeeld van hoe dit eruit ziet is te zien in figuur 3-8. In de realisatie daarvan wordt dit pattern gecombineerd met het dependency injection pattern en het singleton pattern.



Figuur 3-14, Voorbeeld factory pattern in app

3.8. Initieel ontwerp database

Het initiële ontwerp voor de database zoals die gebruikt is in sprint 1 is te zien in Figuur 3-10.



Figuur 3-15, Initieel database ontwerp

Bij dit ontwerp hebben de tabellen category en tag extra velden (CanDelete, TranslationKey) die in de toekomst gebruikt zouden kunnen worden om te zorgen dat een gebruiker een category of tag niet kan verwijderen of om te zorgen dat de categorieen en tags die meegeleverd worden met de app in meerdere talen getoond kunnen worden.

Deel 2

De sprints

4. Sprints

In dit hoofdstuk zullen de sprints besproken worden.

Per sprint zullen de volgende dingen besproken worden:

- Periode waarin de sprint plaatsvond
- De user stories voor de sprint
- De Definition of Done
- Technieken die gebruikt zijn tijdens de sprint inclusief de keuzes en problemen
- Overige keuzes en problemen die gemaakt en of ondervonden zijn
- Resultaat van de sprint

4.1. Sprint 1: Afspraak Aanmaken

4.1.1. Periode

Week 3 28-02					Week 4 05-09					Week 5 12-16					Week 6 19-23				
Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr

Figuur 4.1-1, periode sprint 1

4.1.2. User stories

De user stories die in deze sprint geïmplementeerd gaan worden staan hieronder. Deze user stories komen voort uit de oriënterende fase en zijn gekozen, omdat het in deze sprint belangrijk was om de basisstructuur van de app te hebben.

Requirement	ID	Omschrijving
FR1	US1	Als gebruiker wil ik een categorie en tag selecteren, zodat ik een afspraak aan kan maken
FR1	US2	Als gebruiker wil ik een dag en tijdstip selecteren, zodat ik een afspraak aan kan maken
FR1	US3	Als gebruiker wil ik een agenda selecteren, zodat ik een afspraak aan kan maken
FR2	US4	Als gebruiker wil ik een categorie kunnen aanmaken, omdat de bestaande categorieën niet voldoen.
FR5	US7	Als gebruiker wil ik een tag kunnen aanmaken, omdat de bestaande tags niet voldoen.

Tabel 4.1-1, user stories sprint 1

4.1.3. Definition of Done

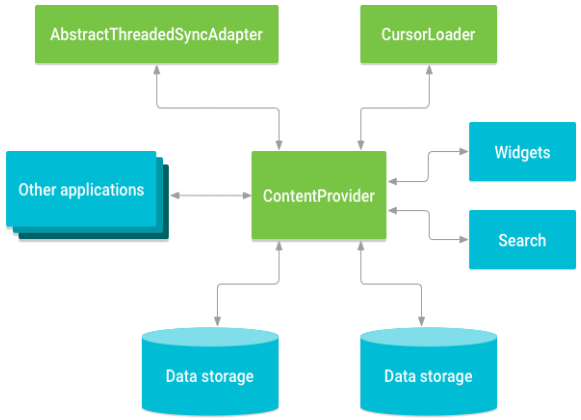
De definition of done voor deze sprint was: Het kunnen aanmaken van een afspraak in de Google calendar door de volgende stappen te doorlopen:

1 Selecteren van een categorie
2 Selecteren van een tag
3 Selecteren van een datum
4 Selecteren van een tijd
5 Selecteren van een of meerdere agenda's waar de afspraak aan toegevoegd wordt

4.1.4. Contentprovider

Om communicatie mogelijk te maken met de Google Calendar voor het ophalen van afspraken of het toevoegen van afspraken. Moest er gebruik gemaakt worden van de Xamarin Android Calendar Content Provider. Wat een wrapper is voor de Android Calendar Content Provider.

Een content provider beheert de toegang tot een centrale verzameling van data, afhankelijk van de content provider kan deze verzameling uiteenlopen van een simpel tekst bestandje tot een complexe database. Een content provider kan gemaakt worden om data beschikbaar te maken aan een eigen applicatie en deze data beschikbaar te maken voor andere apps, zodat die toegang kunnen krijgen tot de data. Door het gebruiken van een provider client object, die methoden bevat om data op te halen of om die op te slaan en te bewerken.

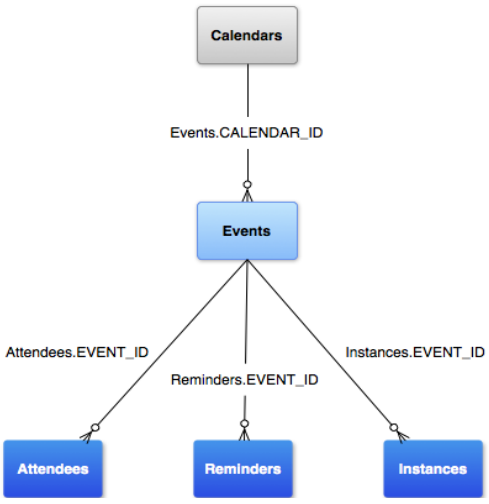


Figuur 4.1-2, relatie content provider en andere componenten [https://developer.android.com/guide/topics/providers/content-provider-basics.html]

Figuur 4.1-2 toont de relatie die een content provider heeft met andere componenten.

De content providers die aangeboden worden door Android bieden data aan als een verzameling van tabellen gebaseerd op een relationeel database model, waar elke rij een record is en elke kolom data bevat van een bepaald type. Door het gebruiken van de Calendar Provider API kunnen applicaties deze data gebruiken en bewerken.

Figuur 4-1-3 toont een grafische representatie van het datamodel voor de Calendar Provider en toont de hoofd tabellen en velden die ze met elkaar verbindt.



Figuur 4.1-3, Data model calendar provider [https://developer.android.com/guide/topics/providers/calendar-provider.html]

Doordat er gekozen is om minimaal Android API 16 (Hoofdstuk 4.1.7.1 Minimale Android versie) te ondersteunen. Kan er op een relatief eenvoudige wijze data opgehaald worden van de Calendar Provider. Het communiceren met de Calendar Provider gebeurt op onderstaande wijze.

```
var cursor = Application.Context.ContentResolver.Query(
                    CalendarContract.Events.ContentUri,
                    EVENT_PROJECTION,
                    searchString,
                    searchParameter,
                    null);
```

De ContentUri is een link om bepaalde data op te vragen aan een Content Provider. Elke Content Provider heeft een unieke link waarmee gecommuniceerd kan worden.

```
content://com.android.calendar/events
```

Bovenstaande is de link om afspraken op te halen van de Calender Provider.

EVENT_PROJECTION is een string array met de velden die je terug wilt krijgen. Een gedeelte van de EVENT_PROJECTION die ik gebruikt heb ziet er als volgt uit:

```
private static readonly string[] EVENT_PROJECTION ={
        CalendarContract.Events.InterfaceConsts.Id,
        CalendarContract.Events.InterfaceConsts.Title,
        CalendarContract.Events.InterfaceConsts.Description,
```

De Query methode maakt van de parameters een SQL query de EVENT_PROJECTEN kan je dus zien als de velden van een SELECT query die je in een normale SQL query hebt.

Als API 16 niet het minimum geweest was maar een versie lager dan dat. Dan zou er inplaats van de Query methode zelf een SQL query geschreven moeten worden en zou het dus nodig zijn om zelf te achterhalen hoe de verschillende velden heten in de verschillende kalender gerelateerde tabellen.

4.1.5. Extension methods

Omdat er op meerdere plekken in de app gebruik gemaakt werd van methodes die hetzelfde doen en bij elke wijziging aan de werking van deze methode moest ik meerdere bestanden langs om de betreffende methode te wijzigen. Leek het mij handig om deze methoden op een centrale plek neer te zetten waardoor ze door alles gebruikt kunnen worden. Om dit voor elkaar te krijgen zijn er twee mogelijkheden namelijk:

- Een statische klasse waar de methodes in staan
- Gebruiken van Extension methods

Ik heb gekozen om extension methods te maken, omdat de methoden waar het over gaat altijd maar één parameter hebben van een bepaald type (string of int). Extension methods zijn methoden die je kan toevoegen aan bestaande klassen, zonder dat je deze hoeft te subklassen of hercompileren. Zo kan er een methode toegevoegd worden aan de string of int klasse en kan deze methode overal in de app gebruikt worden.

Extension methods maak je door een static methode te definieren in een static class waarbij de eerste parameter met behulp van het “this” keyword het type is waar de extension aan toe gevoegd wordt.

Een voorbeeld hiervan is de volgende methode:

```
public static string ToHexColor(this int color)
{
        return Java.Lang.String.Format("#%06X", (0xFFFFFF & color));
}
```

Bovenstaande methode wordt gebruikt om de integer waarde die ik terugkrijg van de Calendar Provider om te zetten naar een hexadecimale waarde die ik kan meegeven aan de Java Color klasse die er vervolgens een kleur van kan maken die getoond kan worden.

Deze methode kan aangeroepen worden op de volgende manier:

```
int getal = 123456;
string kleur = getal.ToHexColor();
```

Dit is tevens ook wat makkelijker om te lezen dan bijvoorbeeld het volgende:

```
int getal = 123456;
string kleur = StringUtils.ToHexColor(getal);
```

4.1.6. Tonen iconen en eigen lettertype

Omdat er voor de DatumPrikker app al lettertypes en iconen gemaakt zijn door de designers van Webbeat en mijn app dezelfde kleur stijl zou krijgen als DatumPrikker is er ook gekozen om hetzelfde te doen voor de lettertypes en categorie iconen.

Voor het tonen van de categorie iconen en eigen lettertype is gebruik gemaakt van een Typeface. Een Typeface is een bestand wat bestaat uit een collectie van characters in verschillende grootten en stylen. Er is gekozen om de categorie iconen met behulp van een typeface te tonen, omdat er anders tientallen afbeeldingen gemaakt zouden moeten worden wat de app vrij groot zou maken. De iconen Typeface is ongeveer 57 KB groot en bevat ongeveer 120 iconen. Voor een kwalitatieve afbeelding die op verschillende scherm groottes werkt zit je al gauw op de 500 KB tot 1 MB.

Voor het tonen van de iconen wordt er gebruik gemaakt van de DatumPrikkerCat2.ttf Typeface. Waar de iconen in staan. Elk icoontje heeft een zogeheten unieke fontcode, zoals te zien is in Figuur 4.1-4 heeft het mes en vork icoon de code “1f374”.

Figuur 4.1-4, Typeface icoon

In tegenstelling tot iOS is er in Android geen mogelijkheid om eenvoudig een zelfgemaakte lettertype te gebruiken. Dat moet of handmatig aan elke UI-component toegewezen worden of automatisch gedaan worden door het gebruiken van plug-ins, omdat ik geen werkende plug-ins kon vinden.
Was het nodig om het toewijzen van de typefaces zelf af te handelen dit heb ik gedaan door eerst een klasse te maken die het singleton pattern (Hoofdstuk 3.7.1 Design patterns) implementeert, zodat de typefaces maar één keer ingeladen hoeven te worden inplaats van elke keer als er typefaces nodig zijn. Wat voorkomt dat het geheugen gebruik van de app flink omhoog gaat, doordat al deze typeface objecten in het geheugen bewaard moeten worden.

```
public void LoadFonts()
{
datumprikkerCat2 = Typeface.CreateFromAsset(Application.Context.Assets, "fonts/DatumprikkerCat2.ttf");
dtpFont = Typeface.CreateFromAsset(Application.Context.Assets, "fonts/DtpFont.otf");
datumprikker17 = Typeface.CreateFromAsset(Application.Context.Assets, "fonts/Datumprikker17.ttf");
}
```

Het inladen van de typefaces gebeurt op bovenstaande manier. Één van deze typefaces kan vervolgens aan een textveld of iets anders toegewezen worden. Of door het overschrijven van een aantal methoden van de TextView klasse kan dit ook gedaan worden op een iets natuurlijkere manier.

```
public class FontTextView : TextView
{
  public FontTextView(Context context, IAttributeSet attrs, int defStyle)
       : base(context, attrs, defStyle)
   {
      string name = attrs.GetAttributeValue("http://schemas.android.com/apk/res-
auto", "fontname");
      init(name);
    }
    private void init(string fontname)
    {
      if (!IsInEditMode) {
           Typeface tf = Mvx.Resolve<IFontService>().GetFont(fontname);
            Typeface = tf;
          }
    }
```

Bovenstaand klasse is hier een voorbeeld van. Deze klasse kan gebruikt worden in de view layouts inplaats van de standaard Android textview. Met onderstaande kan bovenstaande klasse in de view xml layouts neergezet worden waarbij er met de optie fontname gespecifieerd kan worden welke typeface er gebruik zal worden bij de betreffende text view

```
    <Afstudeerproject_Ravi.Droid.Views.Components.FontTextView
        android:text="Text"
        local:fontname="DtpFont"/>
```

Voor het tonen van iconen moet er naast het toewijzen van de datumprikkerCat2 typeface ook iets extra's gebeuren. Namelijk het omzetten van een fontcode (zie figuur 4.1-4) naar een hexadecimale waarde, waar de betreffende typeface mee om kan gaan. Het omzetten gebeurt met de volgende extension methode (hoofdstuk 4.1.5 Extension methods):

```
public static string ToHexFontCode(this string code)
{
    int value = int.Parse(code, System.Globalization.NumberStyles.HexNumber);
    string symbol = char.ConvertFromUtf32(value);
    return symbol;
}
```

Deze functie zet eerst een fontcode (bijv. "1f374") om naar een integer (127860), waarna deze wordt omgezet naar een string ("\ud83c\udf74\").

4.1.7. Overige keuzes en problemen

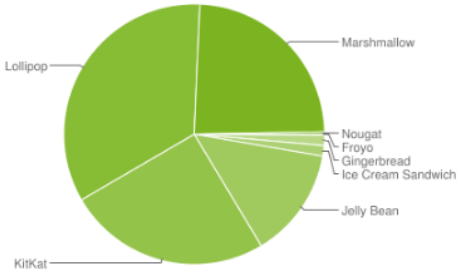
De overige keuzes en problemen die gemaakt en of ondervonden zijn in sprint 1 worden in deze paragraaf besproken.

4.1.7.1. Minimale Android versie

Bij het bouwen van de functionaliteit om met de calendar Contentprovider te praten moest er een keuze gemaakt worden wat de definitieve minimale Android versie zou zijn die ondersteund moest worden. Omdat er verschillende manieren zijn om te communiceren met een contentprovider afhankelijk van de Android API-versie, bijvoorbeeld: het gebruiken van de CalendarContract of handmatig een SQL query te definieren.

Bij het maken van deze keuze is er gebruik gemaakt van Google Dashboard waarop je kan zien wat het percentage gebruikers is voor de verschillende Android versies. Zoals te zien is op Figuur

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.3%
4.1.x	Jelly Bean	16	4.9%
4.2.x		17	6.8%
4.3		18	2.0%
4.4	KitKat	19	25.2%
5.0	Lollipop	21	11.3%
5.1		22	22.8%
6.0	Marshmallow	23	24.0%
7.0	Nougat	24	0.3%



Data collected during a 7-day period ending on November 7, 2016.
Any versions with less than 0.1% distribution are not shown.

Figuur 4.1-5, Android dashboard
[https://developer.android.com/about/dashboards/index.html]

De keuze die uiteindelijk genomen is dat: Android 4.1.x (API 16) minimaal ondersteund moet worden, aangezien het aantal gebruikers van die versie nog relatief hoog is.

4.1.7.2. Testen

Deze sprint had ik ook ingepland om een gedeelte van de app te testen, maar dat is niet gelukt doordat de app functioneel zo ver mogelijk afmaken belangrijker was voor het uiteindelijke resultaat dan het testen ervan. Testen zullen in een volgende sprint opgepakt worden.

4.1.8. Resultaat

Voor deze sprint is de Definition of Done gehaald en kan er dus een afspraak aangemaakt worden door:

- Het selecteren van een categorie en tag
- Het selecteren van een datum en tijd
- Het selecteren van een of meerdere agenda's

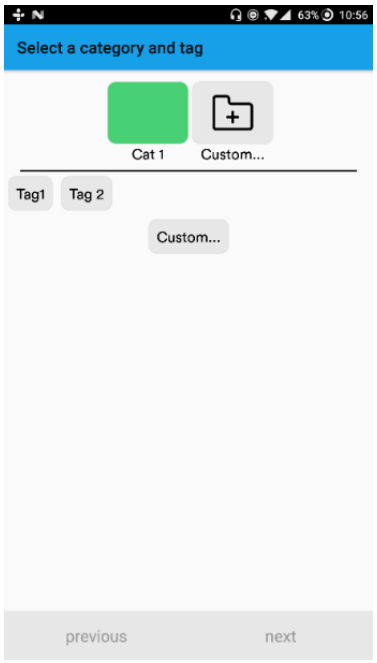
De app is na deze sprint geen shippable product, omdat de styling in alle views ontbreekt en niet werkt op kleinere schermen.

Een aantal dingen die op dit moment nog gedaan moeten zijn onder andere:

- Het design implementeren
- Zorgen dat de bestaande layout ook op kleinere schermen werkt.

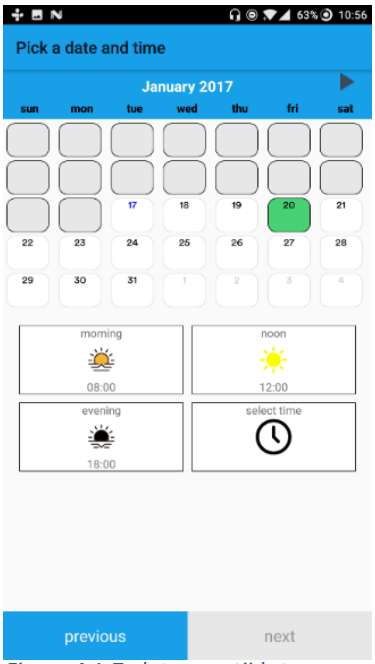
Hieronder staat een overzicht van een aantal schermen in de app zoals die er aan het eind van deze sprint uitzagen:

Categorie en tag stap



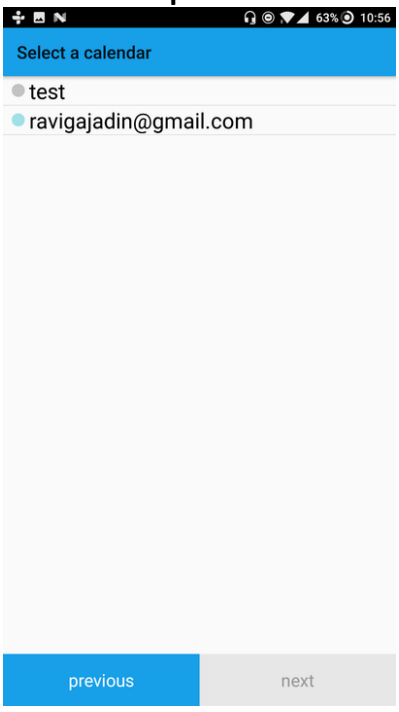
Figuur 4.1-6, categorie en tag stap

Datum en tijd stap



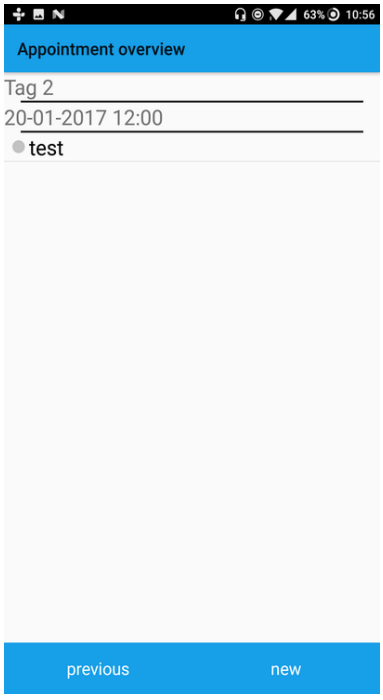
Figuur 4.1-7, datum en tijd stap

Kalender stap



Figuur 4.1-8, kalender stap

Overzicht scherm



Figuur 4.1-9, overzicht scherm

4.2. Sprint 2: Intelligentie toevoegen

4.2.1. Periode

In deze sprint waren er 5 weken inplaats van 4, omdat ik in week 7 en 8 een aantal vrije dagen had.

Week 7 26-30					Week 8 02-06					Week 9 09-13					Week 10 16-20					Week 11 23-27				
Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr

Figuur 4.2-1, Sprint periode

4.2.2. User stories

In deze sprint zullen er nieuwe user stories geimplementeerd worden. Deze user stories komen voort uit de sprint retrospective van sprint 1 en sprint planning voor deze sprint.

Requirement	ID	Omschrijving
FR9	US11	Als gebruiker wil ik een afspraak kunnen delen met andere, zodat ik iemand kan uitnodigen die geen contact is.
FR1	US12	Als gebruiker wil ik deelnemers kunnen selecteren, omdat ik een afspraak wil inplannen met meerdere personen.
FR10	US13	Als gebruiker wil ik dat de app mij suggesties toont voor afspraak tijden, omdat ik dan sneller een afspraak kan inplannen
FR1	US14	Als gebruiker wil ik dat de app mijn recent gebruikte deelnemers toont, zodat ik sneller mijn meest gebruikte deelnemers kan toevoegen
FR1	US15	Als gebruiker wil ik dat de app de afspraken per dag toont, zodat ik beter een keuze kan maken.

Tabel 4.2-1, User stories sprint 2

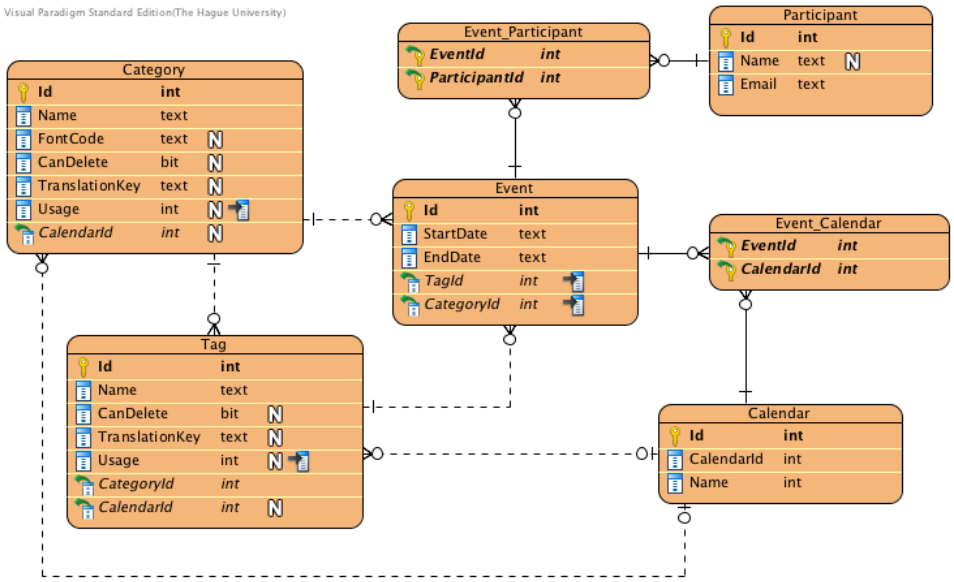
4.2.3. Definition of Done

De definition of done voor deze sprint was:

- Het kunnen selecteren van deelnemers tijdens het maken van een afspraak
- Zorgen dat de app suggesties kan tonen voor afspraak tijden en deelnemers

4.2.4. Database ontwerp

Om het mogelijk te maken voor de app om snel suggesties te tonen was het nodig om de huidige database uit te breiden. Het nieuwe ontwerp is te zien in figuur 4.2-2.



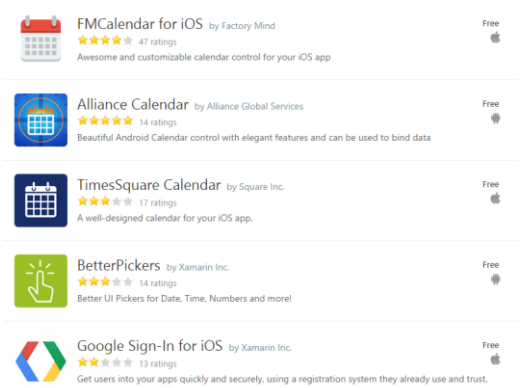
Figuur 4.2-2, Sprint 2 database ontwerp

In het nieuwe ontwerp is gekozen om de nodige data van een afspraak op te slaan namelijk de afspraak starttijd en eindtijd, de agenda's waar deze afspraak aan toegevoegd is, de deelnemers van een afspraak en de tag en categorie. Dit ontwerp zorgt ervoor dat er snel afspraken gevonden kunnen worden die voldoen aan bepaalde criteria, zodat er suggesties getoond kunnen worden zonder dat de gebruiker hoeft te wachten.

4.2.5. Binding library

Omdat in deze sprint het design geïmplementeerd zou worden was het nodig om de simpele kalender die in sprint 1 gebruikt werd te vervangen met iets wat moderner (swipende navigatie e.d.) was en makkelijker aangepast zou kunnen worden.

Het originele idee was om een kalender plug-in te gebruiken die op de Xamarin store stond. Nagenoeg waren de geschikte gratis plug-ins alleen gemaakt om gebruikt te worden op iOS. De Android plug-ins hadden vrijwel dezelfde functionaliteit als de kalender die ik al gebruikte. Er waren nu nog maar twee echte mogelijkheden waaruit ik kon kiezen: Zelf een kalender component maken met functionaliteit die ik nodig heb of met behulp van een Binding library een Android Java kalender over zetten naar C#, zodat het gebruikt kan worden binnen Xamarin.

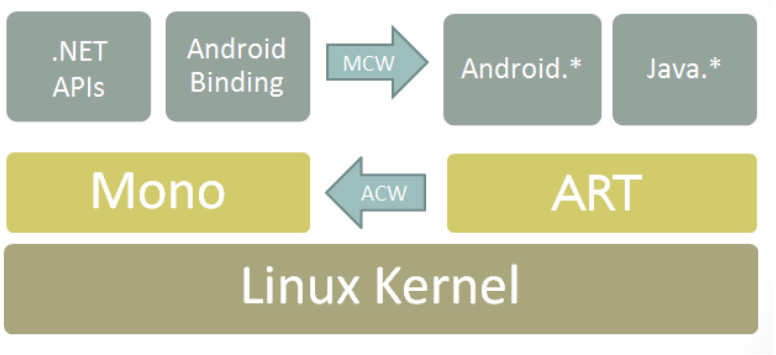


Figuur 4.2-3, Overzicht kalender componenten

Ik had dus gekozen om een binding library te maken wat mij sneller leek om te maken dan een complete kalendar. Achteraf bleek dit dus niet zo te zijn.

Een binding library is niet anders dan een wrapper wat ervoor zorgt dat Java code aangeroepen kan worden via C#.

Xamarin.Android implementeert deze bindings door het gebruiken van Managed Callable Wrappers (MCW). MCW functioneren als een brug tussen de C# en Java code, die gebruikt wordt als er vanuit de C# omgeving Java code wordt aangeroepen. Deze MCW werkt alleen één kant op namelijk vanuit C# naar Java. Om ervoor te zorgen dat er ook communicatie mogelijk is de andere kant op kan, wordt er gebruik gemaakt van Android Callable Wrappers (ACW). Deze architectuur ziet er als volgt uit:



Figuur 4.2-4, Architectuur binding library [https://developer.xamarin.com/guides/android/advanced_topics/binding-a-java-library/]

Om een binding library te maken is er gecompileerde Java code nodig in de vorm van een JAR of AAR bestand. JAR-bestanden zal je tegenwoordig voor Android bijna niet meer vinden tenzij het voor een oude Android versie gemaakt is of gecompileerd is met een ander programma dan Android Studio.

Vervolgens moet er in Visual Studio of Xamarin Studio een Binding Library project aangemaakt worden waar de JAR of AAR aan toegevoegd wordt. Dit project kan dan gecompileerd worden en als resultaat heb je dan de Binding library die in Xamarin gebruikt kan worden.

Het process zoals hierboven bechreven is hoe het normaal zou moeten werken. Alleen hadden de kalender componenten waar ik een binding voor probeerde aan te maken ook nog referenties naar andere Java library's waardoor het al vrij snel erg complex werd en er problemen optraden waar ik geen oplossingen voor kon vinden.

Onderstaande figuur toont een aantal foutmeldingen waar in bovenstaande alinea naar gerefereerd wordt.

```
Couldn't load class com/antonyt/infiniteviewpager/InfinitePagerAdapter : java.lang.NoClassDefFoundError: android/support/v4/view/PagerAdapter (J2X9001)

Couldn't load class com/antonyt/infiniteviewpager/InfiniteViewPager : java.lang.NoClassDefFoundError: android/support/v4/view/ViewPager (J2X9001)

Couldn't load class com/roomorama/caldroid/CaldroidFragment$DatePageChangeListener : java.lang.NoClassDefFoundError: android/support/v4/view/ViewPager$OnPageChangeListener (J2X9001)
```

Figuur 4.2-5, foutmeldingen binding library

Waar deze meldingen voor staan is dat de binding library referenties mist naar de Android Support SDK. Dit zou dus opgelost kunnen worden door het toevoegen van deze referenties, alleen was dat niet het geval en kreeg je bovendien nog meer meldingen na het toevoegen van de referentie dat er referenties miste die de Android Support SDK nodig heeft. Het oplossen van deze fouten zou wel mogelijk geweest zijn als er genoeg tijd beschikbaar voor zou zijn, maar of dit een verstandige keuze geweest zou zijn.

4.2.6. Kalender component

Zoals in hoofdstuk 4.2.5 Binding library is uitgelegd is het binden van een library niet gelukt. Ik moest daarom zelf een kalender component gaan maken die voldeed aan een aantal eisen:

1 Met swipes door de kalender te kunnen bewegen
2 Afspraak indicatie te tonen per dag
3 Compleet te kunnen stylen

Met swipes door de kalender heen te gaan

De oude kalendar maakte gebruik van knoppen om door de kalender heen te gaan. Dit werkte alleen niet echt fijn en het past niet echt bij een mobiele applicatie, daarom is er gekozen om de gebruiker met swipende bewegen door de kalender te laten bewegen.

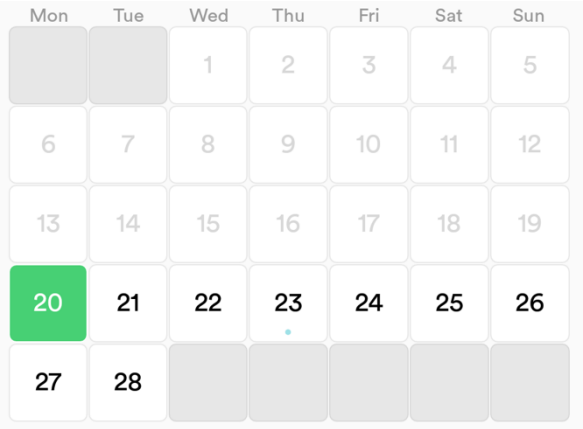
Figuur 4.2-6, voorbeeld swipe navigatie

Om de illusie te wekken dat de gebruiker oneindig door de kalender kan swipen was het nodig om een aantal aanpassingen in te voeren, aan hoe Android omgaat met swipende navigatie. Een andere rede voor deze aanpassing is dat de gebruiker op een redelijk snelle tempo door de kalender heen kan swipen. Wat er met de aanpasing gebeurt is dat Android nu kalender maanden(views) gaat recyclen als ze niet meer gebruikt worden (figuur 4.2-7)

Figuur 4.2-7, indeling kalender pagina's

De kalender heeft vier views de gebruikt worden om de huidige maand en de vorige/volgende maand te onthouden. Dit zorgt ervoor dat de gebruiker niet eerst hoeft te wachten voordat er een view opgebouwd is. De 4de view wordt gebruikt als "recycle" view, dit houdt in dat die alleen gebruikt wordt als er geen view beschikbaar, wat kan gebeuren als de gebruiker te snel terug of verder swiped.

Afspraak indicatie te tonen per dag
Om het maken van afspraken makkelijker te maken, zal er een indicatie getoond moeten worden bij de dagen die een afspraak hebben. Voor de app is gekozen om dit een klein cirkeltje te maken met de kleur van de agenda waar de afspraak instaat. Dit kan natuurlijk later eenvoudig aangepast worden.



Figuur 4.2-8, kalender component

Compleet te kunnen stylen
Om alle elementen van de kalender een andere style te kunnen geven is de kalender modulair opgebouwd, zodat het bestaat uit de rij met dag afkortingen en individuele blokken voor elke dag. Deze blokken kunnen onafhankelijk van elkaar opgemaakt worden.
Door de modulaire opbouw van de kalender kan deze zichzelf schalen naar de grootte van het scherm.

4.2.7. Intelligentie

Om de app wat slimmer te maken is er gekozen om de app suggesties te laten zien bij het selecteren van een dag en tijdstip. Deze suggesties zijn onder andere de meest gebruikte tijden afhankelijk van de tag en categorie en het overnemen van tags die veel voorkomen in de agenda's van de gebruiker.

Deze suggesties kunnen getoond worden, doordat de app de agenda's van de gebruiker doorleest in een periode van 6 maanden (3 maanden terug, 3 maanden heen) vanaf de huidige datum. Dit doorlezen gebeurt alleen als de app detecteert dat er geen afspraken in de interne database staan, dit komt namelijk alleen voor als de app voor de eerste keer opgestart wordt of als de gebruiker de data van de app gewist heeft.

Na het verkrijgen van alle afspraken worden de afspraken verwerkt op de volgende manier:

```
for (int i = 0; i < pastEvents.Count; i++) {
      int count = 0;
      var name = pastEvents[i].Title;
      name = name.Substring(0, 1).ToUpper() + name.Substring(1).ToLower();
      occurrences.TryGetValue(name, out count);
      occurrences[pastEvents[i].Title] = count + 1;
}
```

Bovenstaand stukje code gaat door de lijst met afspraken en houdt bij hoe vaak een tag voorkomt. Het bijhouden van tags gebeurt door de afspraak naam om te zetten naar kleine letters en van de eerste letter een hoofdletter te maken. Op deze manier kunnen namen gevonden die hetzelfde zijn maar verschillen in hoofd/kleine letters. De reden om de eerste letter als hoofdletter neer te zetten, is dat het netter is om afspraak namen met een hoofdletter te beginnen.

```
List<string> tags = occurrences.Where(x => x.Value >= 3).Select(x => x.Key).ToList();
List<int> usageCount = occurrences.Where(x => x.Value >= 3).Select(x => x.Value).ToList();
```

Vervolgens wordt er een lijst met tags die meer dan drie keer voorkomen en het totaal aantal met behulp van LINQ verkregen. LINQ biedt de mogelijkheid zoals te zien is in bovenstaand stukje code, query's te schrijven in code. Er is gekozen voor drie als minimum, omdat het suggereert dat het meer gebruikt kan worden in tegenstelling tot één of twee keer.

Als laatste wordt er een lijst met afspraken aangemaakt die aan de SQLite database toegevoegd kunnen worden. Als de afspraken in de database toegevoegd zijn kan de app op basis van de beschikbare gegevens een suggestie doen aan de gebruiker per categorie en tag, met de tijden waarop er afspraken gemaakt zijn met die categorie of tag. Het verkrijgen van deze lijst gebeurt met hulp van een SQL-query. De query voor het ophalen van de tijden suggesties voor tags ziet er zo uit:

```
"SELECT * " +
"FROM Event " +
"WHERE TagId = ? " +
"GROUP BY SUBSTR(StartDate,12,5) " +
"HAVING COUNT(*) >= 1 " +
"ORDER BY SUBSTR(StartDate,12,5) ASC " +
"LIMIT 3"
```

Met deze query worden alle events opgehaald die gemaakt zijn met de betreffende tag id vervolgens worden de resultaten gegroepeerd per tijdstip. Het is bij het groeperen nodig om de SUBSTR sql functie te gebruiken, omdat datums in het formaat "maand-dag-jaar uren:minuten" opgeslagen zijn en het in dit geval alleen nodig is om de tijden terug te krijgen en niet ook de datums. Vervolgens worden de tijden gesorteerd en worden de top drie tijden teruggegeven die meer dan één keer voorkomen.

4.2.8. Testen

Omdat de applicatie nu al vrij groot was en het steeds doorlopen van schermen om te kijken of alles nog werkte vrij vermoeiend was. Heb ik UI testen gemaakt die de UI voor mij doorlopen en controleren of alles werkt en getoond wordt zoals het zou moeten.

De UI testen worden gemaakt met behulp van het Xamarin.UITest test framework. Xamarin.UITest is een geautomatiseerde testing framework gebaseerd op Calabash wat ervoor zorgt dat er testen geschreven en uitgevoerd kunnen worden in C#. Er wordt ook gebruik gemaakt van NUnit voor het valideren van de gemaakte testen.

Deze testen kunnen uitgevoerd worden op een echt apparaat of met behulp van een emulator.

Een voorbeeld van hoe de UI test voor het aanmaken van een categorie eruit ziet is als volgt.

```
app.WaitForElement(c => c.Marked("category_name").Text("Activity"));
app.ScrollDownTo(c => c.Marked("category_name").Text("New..."), c => c.Marked("sv_category"));
app.Tap(c => c.Marked("category_name").Text("New..."));
```

Met bovenstaande code wordt het proces gestart door het volgende te doen:

- Er wordt eerste gewacht of er een categorie met de naam “Activity” getoond wordt, Omdat deze categorie in de test als laatste ingeladen wordt.
- Vervolgens scrolled de app naar de “New” knop.
- En wordt er op de knop gedrukt.

Vervolgens wordt het volgende uitgevoerd:

```
app.WaitForElement(c => c.Marked("fet_categoryname"));
Assert.IsFalse(app.Query(c => c.Marked("btn_save")).First().Enabled);
```

- Hier wordt er weer gewacht tot de app klaar is met laden
- Er wordt een vergelijking gedaan om te kijken of de opslaan knop klik baar is of niet

Als de opslaan knop hier klikbaar is wordt de test gestopt met resultaat: Failed, anders gaat de test door met:

```
app.EnterText(c => c.Marked("fet_categoryname"), "abcdefghijklmn");
app.Tap(c => c.Id("rv_cat_icon").Child(0));
Assert.IsTrue(app.Query(c => c.Marked("btn_save")).First().Enabled);
```

- Hier worden eerst de letters “abcdefghijklmn” ingevoerd.
- Daarna wordt het eerste icoontje geselecteerd.
- Vervolgens wordt er weer gekeken op de opslaan knop klikbaar is of niet

Als de opslaan knop hier onklikbaar is wordt de test gestopt met resultaat: Failed, anders gaat de test door met:

```
app.ClearText(c => c.Marked("fet_categoryname"));
Assert.IsFalse(app.Query(c => c.Marked("btn_save")).First().Enabled);

app.EnterText(c => c.Marked("fet_categoryname"), "abcdefgh");
Assert.AreEqual(8, app.Query(c => c.Marked("fet_categoryname")).First().Text.Length);
app.Tap(c => c.Marked("btn_save"));

app.WaitForElement(c => c.Marked("category_name"));
Assert.IsTrue(app.Query(c => c.Marked("category_name").Text("abcdefgh")).First().Enabled);
```

Dit is het laatste stukje van de test waar het volgende wordt gedaan:

- Het invoerveld wordt leeg gemaakt.
- Er wordt gekeken of de opslaan knop onklikbaar is.
- Er wordt een reeks letters ingevoerd.
- Er wordt gekeken of de lengte van de tekst in het invoerveld gelijk is aan 8, omdat dat het maximum is wat ingevoerd kan worden.
- Er wordt op de opslaan knop gedrukt
- Er wordt gewacht tot de applicatie terug is bij het categorie en tag selectie scherm.
- Er wordt gekeken of de aangemaakt categorie getoond wordt.

Alle UI testen worden op dezelfde manier gemaakt, waar er in stappen gekeken wordt of alles goed werkt of niet. Met automatische testen kan je alles doen wat een gebruiker ook

kan doen met de app, waardoor gebruikers interactie testen vrij eenvoudig gemaakt kunnen worden.

Mocking

Voor het unit testen is gebruik gemaakt van mocking. Mocking is het simuleren van gedrag en eigenschappen van objecten tijdens een test. Omdat er bijvoorbeeld bij het delen van een afspraak gebruik gemaakt wordt van een service die communiceert met de web api (hoofdstuk 4.2.9 Backend Web Api). Door deze service te mocken kan er gekeken worden of de app de juiste data verstuurd en de data die ontvangen wordt goed afhandelt zonder dat de web api daarvoor moet draaien.

Voor het mogelijk maken om objecten te mocken is gebruik gemaakt van het Moq framework. Wat op de volgende manier gebruikt is.

```
private Mock<IShareService> mockShare;
mockShare = new Mock<IShareService>();
mockShare.Setup(m => m.GetShareLink(It.IsAny<ShareEvent>())).Returns(ser);
mockShare.Setup(m => m.ParseAppointment(It.IsAny<Appointment>())).Returns("data");
```

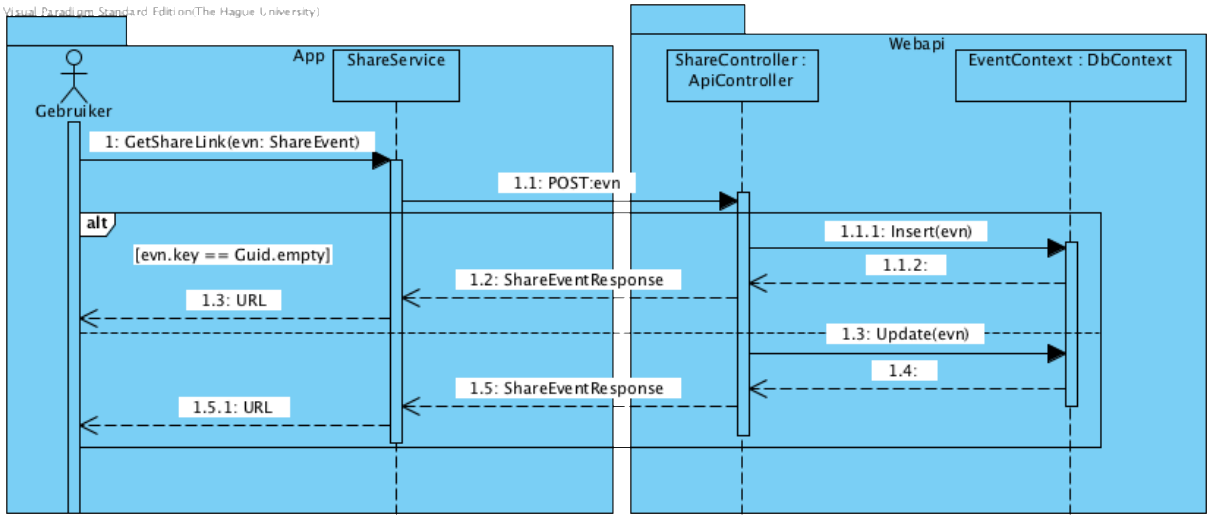
Bovenstaande code is nodig om de gebruikte methoden van ShareService te mocken. Het mocken gebeurt door de Setup methoden waar er met behulp van een lambda expressie (Een lambda expressie wordt ook wel een anonieme functie genoemd, omdat het in tegenstelling tot een normale methode geen naam heeft), gedefinieerd kan worden wat de betreffende functie als retour waarde heeft.

Het object wat aangemaakt is wordt vervolgens mee gegeven aan de overview viewmodel, aangezien de ShareService alleen daar wordt gebruikt.

Mocking kan niet toegepast worden bij het UI testen, omdat er dan niet getest wordt wat er in de app gebeurt maar alleen wat de app laat zien.

4.2.9. Backend Web API

Om het mogelijk te maken dat er een afspraak gedeeld kan worden via de app, was het nodig om een systeem te gebruiken wat ervoor zorgde dat de gebruiker niet zelf bestanden hoeft te delen met mensen, daarom is de keuze gemaakt om het delen van een afspraak, mogelijk te maken met behulp van een web api. Deze web api moet de mogelijkheid hebben om de gebruiker een URL te geven die opgestuurd kan worden naar andere mensen die vervolgens een bestandje kunnen downloaden om de afspraak aan hun eigen agenda toe te voegen.



Figuur 4.2-9, web api interactie voor delen

Figuur 4.2-9 laat zien hoe de interactie eruitziet tussen de app en web api als de gebruiker op de delen knop drukt. Het versturen van data tussen de app en web api gebeurd met behulp van JSON. De app en web api hebben identieke klassen (ShareEvent en ShareEventResponse) deze klassen worden respectievelijk gebruikt voor het versturen van een afspraak naar de web api en het terug krijgen van een unieke link. Deze link is uniek doordat er gebruik gemaakt wordt van een Guid. Een Guid is een willekeurige reeks van unieke letters en cijfers.

Een afspraak wordt aan de web api kant omgezet van JSON naar een ShareEvent en vervolgens toegevoegd of geupdate afhankelijk van of het een nieuwe afspraak is of niet. Of het een nieuwe afspraak is of niet bepaald de app.

t_evn_event	
evn_id	**int**
evn_key	uniqueidentifier
evn_title	varchar(255)
evn_category	varchar(255)
evn_categoryicon	varchar(255)
evn_startdate	datetime
evn_enddate	datetime
evn_data	varchar(max)

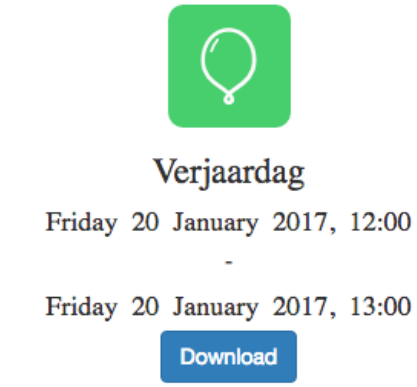
Figuur 4.2-10, web api database

Voor het opslaan van data is gebruik gemaakt van een Microsoft SQL-database, omdat dat als development database server gebruikt wordt bij Webbeat. Voor het maken van de database was het nodig om enkele conventies te volgen, die bij Webbeat gebruikt worden namelijk:

- Tabel begint met een ‘t' daarna een drie letterige afkorting met als laatste de tabel naam
- Alle velden in een tabel hebben voor hun naam de drie letterige afkorting als begin.

Er is ook gebruik gemaakt van drie stored procedures: het toevoegen van data, bewerken van data en het ophalen van data.

De link die een gebruiker terug krijgt heeft als parameter de unieke sleutel van de betreffende afspraak, deze kan opgestuurd worden naar andere mensen. Als iemand naar het linkje toe browsed krijgt diegene een scherm te zien zoals op figuur 4.2-11 getoond wordt, waarin enkele punten van de afspraak getoond worden zoals: de titel en de start en eind tijd van de afspraak.
Wanneer er op de download knop gedrukt wordt. Krijgt de gebruiker een event.ics bestand die geïmporteerd kan worden in vrijwel elke agenda app.



Figuur 4.2-11, web api afspraak info

Een ICS bevat onder andere de titel van een afspraak en de start en einddatum.

4.2.10. Resultaat

Voor deze sprint is de Definition of Done gehaald, wat betekent dat de app naast hetgeen wat in sprint 1 behaald is nu ook:

- Bij de stap “datum tijd”, tijd suggesties kan tonen aan de gebruiker op basis van de meest gebruikte tijden van de geselecteerde categorie en tag.
- De gebruiker de mogelijkheid geven om contacten die op het apparaat staan toe te voegen aan een afspraak.
- Afspraken te delen via bijvoorbeeld social media
- Volledig gestyled is

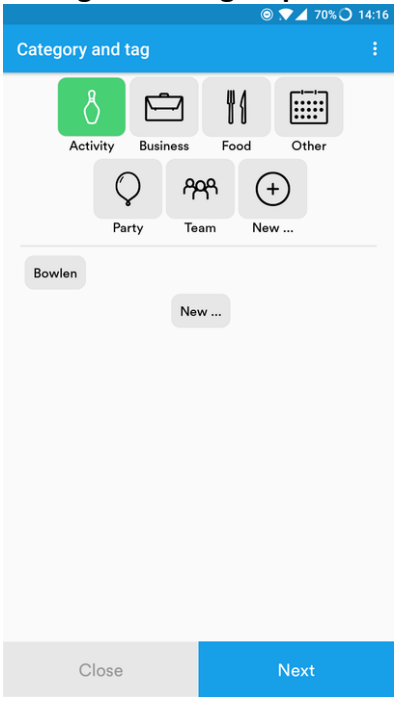
De app kan nu gezien worden als een shippable product, omdat het functioneel ver genoeg is om bijvoorbeeld als beta versie aan te bieden op de Google PlayStore.

Een aantal dingen die nog aan de app gedaan moeten worden zijn onder andere:

- Het vertalen van de app in het Nederlands en zorgen dat overal de juiste Engelse woorden staan.
- Hier en daar nog de styling nog een beetje aanpassen.
- Er waren aan het eind van deze sprint een aantal GUI gerelateerde bugs

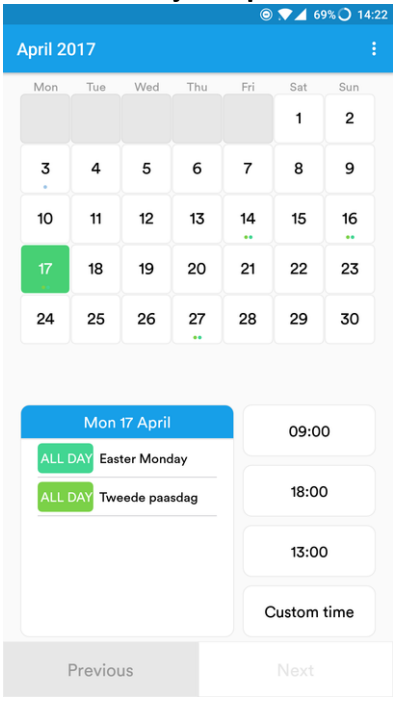
Hieronder zal er een overzicht getoond worden van een aantal schermen en hoe ze er aan het eind van sprint 2 uitzagen.

Categorie en tag stap



Figuur 4.2-12, categorie en tag stap

Datum en tijd stap

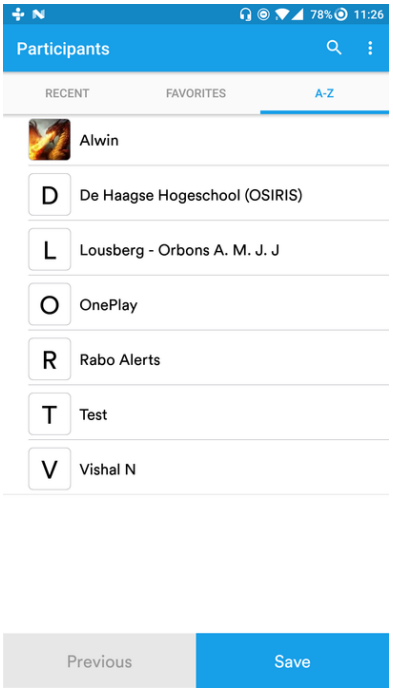


Figuur 4.2-13, datum en tijd stap

Agenda selectie scherm

Figuur 4.2-14, kalender scherm

Deelnemer selectie scherm



Figuur 4.2-15, deelnemer selectie scherm

4.3. Sprint 3: App klaar maken voor release

4.3.1. Periode

Week 12 30-03					Week 13 06-10					Week 14 13-17					Week 15 20-24				
Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr	Ma	Di	Wo	Do	Vr

Figuur 4.3-1, periode

4.3.2. User stories

In deze sprint zullen de resterende user stories geimplementeerd worden.

Requirement	ID	Omschrijving
FR3	US5	Als gebruiker wil ik een categorie kunnen bewerken, omdat de naam van de categorie niet klopt.
FR4	US6	Als gebruiker wil ik een categorie kunnen verwijderen, omdat ik de categorie niet nodig heb.
FR6	US8	Als gebruiker wil ik een tag kunnen bewerken, omdat de naam van de tag niet klopt.
FR7	US9	Als gebruiker wil ik een tag kunnen verwijderen, omdat ik de tag niet nodig heb.

Tabel 4.3-1, user stories sprint 3

Bij de sprint 2 retrospective en sprint 3 planning zijn er in overleg met de opdrachtgever een aantal nieuwe user stories bijgekomen die de gebruikers ervaring zouden moeten verbeteren en zijn er ook een aantal user stories gewijzigd, voor het verbeteren van de gebruikers ervaring.

Toegevoegd

Requirement	ID	Omschrijving
	US16	Als gebruiker wil ik een afspraak aanmaken om alleen te delen, omdat de afspraak niet in mijn agenda hoeft te staan.
	US17	Als gebruiker wil ik de optie hebben om bepaalde agenda's niet kiesbaar te maken, omdat ik die agenda's niet gebruik
	US18	Als gebruiker wil ik de standaard tijden kunnen aanpassen, omdat de standaard keuzes niet kloppen met mijn schema.
	US19	Als gebruiker wil ik de optie om van taal te kunnen veranderen, omdat ik de huidige taal niet ken.
	US20	Als gebruiker wil ik de optie hebben om te kiezen tussen 24 uur of 12 uurs formaat voor tijden.
	US21	Als gebruiker wil ik een lijst kunnen zien van afspraken die gemaakt zijn met de app, omdat ik een bestaande afspraak wil delen.
	US22	Als gebruiker wil ik een locatie kunnen selecteren, omdat de afspraak op een bepaalde locatie plaatsvindt.

Tabel 4.3-2, toegevoegde user stories

Gewijzigd

Requirement	ID	Omschrijving
FR1	US1	Als gebruiker wil ik een categorie en tag selecteren, zodat ik een afspraak aan kan maken
FR1	US3	Als gebruiker wil ik een agenda selecteren, zodat ik een afspraak aan kan maken

Tabel 4.3-3, gewijzigde user stories

- US1 is het selecteren van een tag nu optioneel geworden, omdat de categorie naam ook genoeg kan zijn voor als afspraak titel.
- US3 door de komst van US13 is het selecteren van een agenda nu optioneel.

4.3.3. Definition of Done

De definition of done voor deze sprint was:

- Het implementeren van het instellingen scherm waarbij de volgende dingen gedaan kunnen worden.
    - Categorie bewerken/verwijderen
    - Tag bewerken/verwijderen
    - Kiezen tussen Nederlands en Engels als app taal
    - Kiezen van agenda's die getoond moeten worden
    - Instellen van standaard tijden
    - Kiezen om tijd als 24 uur of 12 te weergeven
    - Kiezen om de app Tijd suggesties te laten maken.
- Afmaken van de andere user stories.

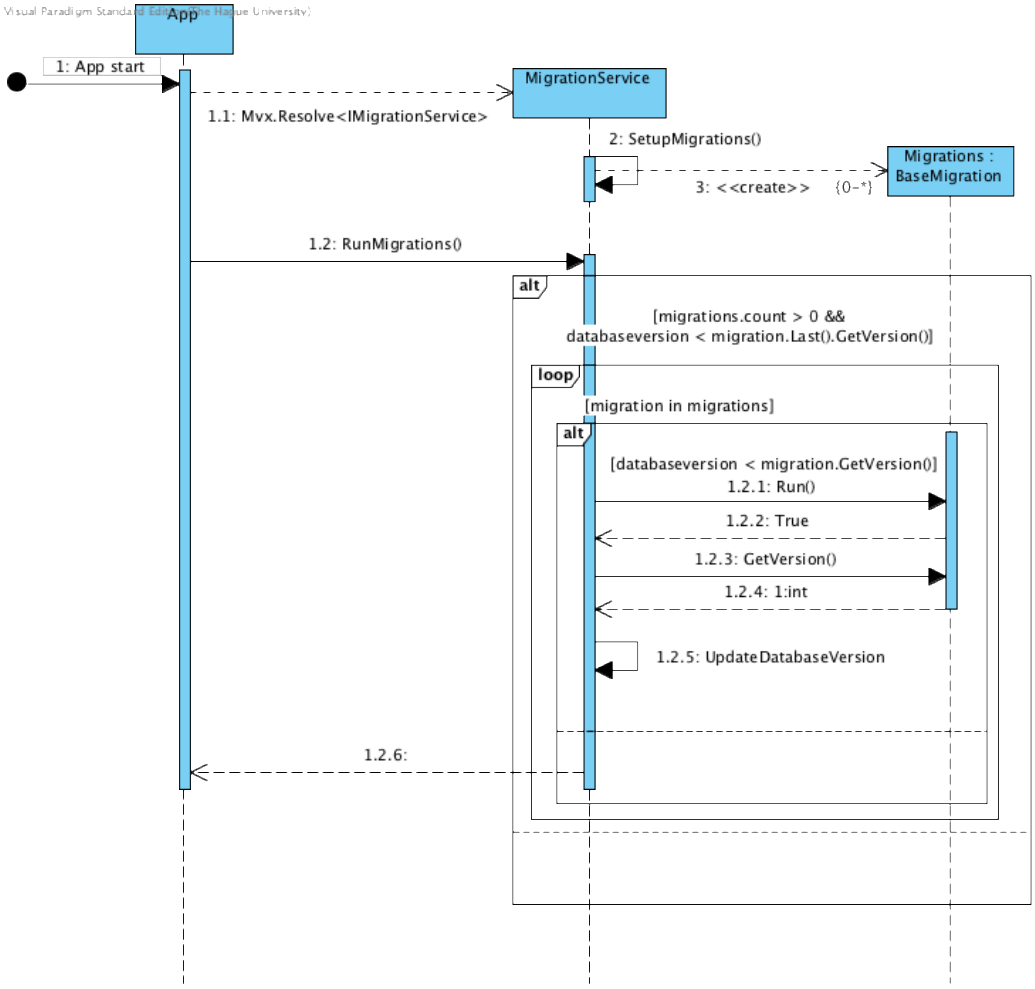
4.3.4. Database migraties

Omdat het in de toekomst voor kan komen dat de database aangepast zal worden. Wat op dit moment gedaan kan worden door de database te overschrijven, iets wat niet handig is als de app eenmaal uitgebracht is, doordat de gebruiker dan al zijn data kwijt is. Was het nodig om een systeem te bedenken die het mogelijk maakt om de database te bewerken of aan te passen zonder dat de gebruiker er iets van merkt.

Tijdens het bedenken van een systeem hiervoor ben ik op de volgende twee ideeën gekomen die makkelijk te implementeren waren en eenvoudig te gebruiken:

- Database versie bijhouden en SQL-bestanden meesturen met de app met de volgende naamgeving: migration_1.sql, migration_2.sql, etc..
- Database versie bijhouden en klassen gebruiken als containers met sql code met de volgende naamgeving: Migration1.cs, Migration2.cs, etc..

Er was ook een extra eis die ik belangrijk vond voor de werking en dat was dat het volledig crossplatform kan werken. Idee 1 viel daardoor meteen af, omdat het meeleveren van bestanden die ingeladen moeten worden niet crossplatform gedaan kan worden maar platform specifiek zou moeten gebeuren. Wat ervoor zorgt dat ik zelf zou moeten bijhouden of ik aan elk platform (Android, iOS) het laatste migratie bestandje toegevoegd heb.
Er is daarom voor Idee 2 gekozen wat op onderstaande manier geïmplementeerd is:



Figuur 4.3-2, gedeelte sequentie diagram migratieproces

In het sequentie diagram hierboven wordt het belangrijkste gedeelte van het proces getoond. Het update proces wordt gestart, zodra de app bezig is met het tonen van het splashscreen en is klaar voordat de app het categorie en tag scherm laat zien. Wat er gebeurt tijdens het proces is dat er een MigrationService object aangemaakt wordt die vervolgens gaat zoeken in de app of er migraties aanwezig zijn en of deze uitgevoerd moeten worden aan de hand van de migratie versie en database versie.

```
var currentDomain = typeof(string).GetTypeInfo().Assembly.GetType("System.AppDomain")
                                                  .GetRuntimeProperty("CurrentDomain")
                                                  .GetMethod
                                                  .Invoke(null, new object[] { });

var getAssemblies = currentDomain.GetType().GetRuntimeMethod("GetAssemblies", new Type[] { });
var assemblies = getAssemblies.Invoke(currentDomain, new object[] { }) as Assembly[];

var allTypes = assemblies.SelectMany(a => a.DefinedTypes);
var typesWithRegisterAttributes = allTypes
                                  .Select(t => new { TypeInfo = t })
                                  .Where(x => x.TypeInfo.BaseType == typeof(BaseMigration));

foreach (var pair in typesWithRegisterAttributes) {
    MvxTrace.TaggedTrace("MIGRATION", "Migration found {0}", pair.TypeInfo.AsType().ToString());
    migrations.Add((BaseMigration)Activator.CreateInstance(pair.TypeInfo.AsType(), null));
}
```

Bovenstaand stukje code wordt gebruikt voor het vinden van klassen die overerven van

BaseMigration en daar objecten van te creëren. Het vinden van deze klassen gebeurt door het stukje code boven de foreach, wat deze code doet is eerst runtime de methode GetAssemblies opvragen. Dit moet runtime gebeuren, omdat deze methode pas bestaat in xamarin apps als de app eenmaal gestart is. Vervolgens wordt met deze methode alle klassen opgevraagd, waarna er gekeken wordt of deze overerven van BaseMigration.

De enige klassen die BaseMigration overerven zijn de migration klasses. Deze klassen zien er als volgt uit:

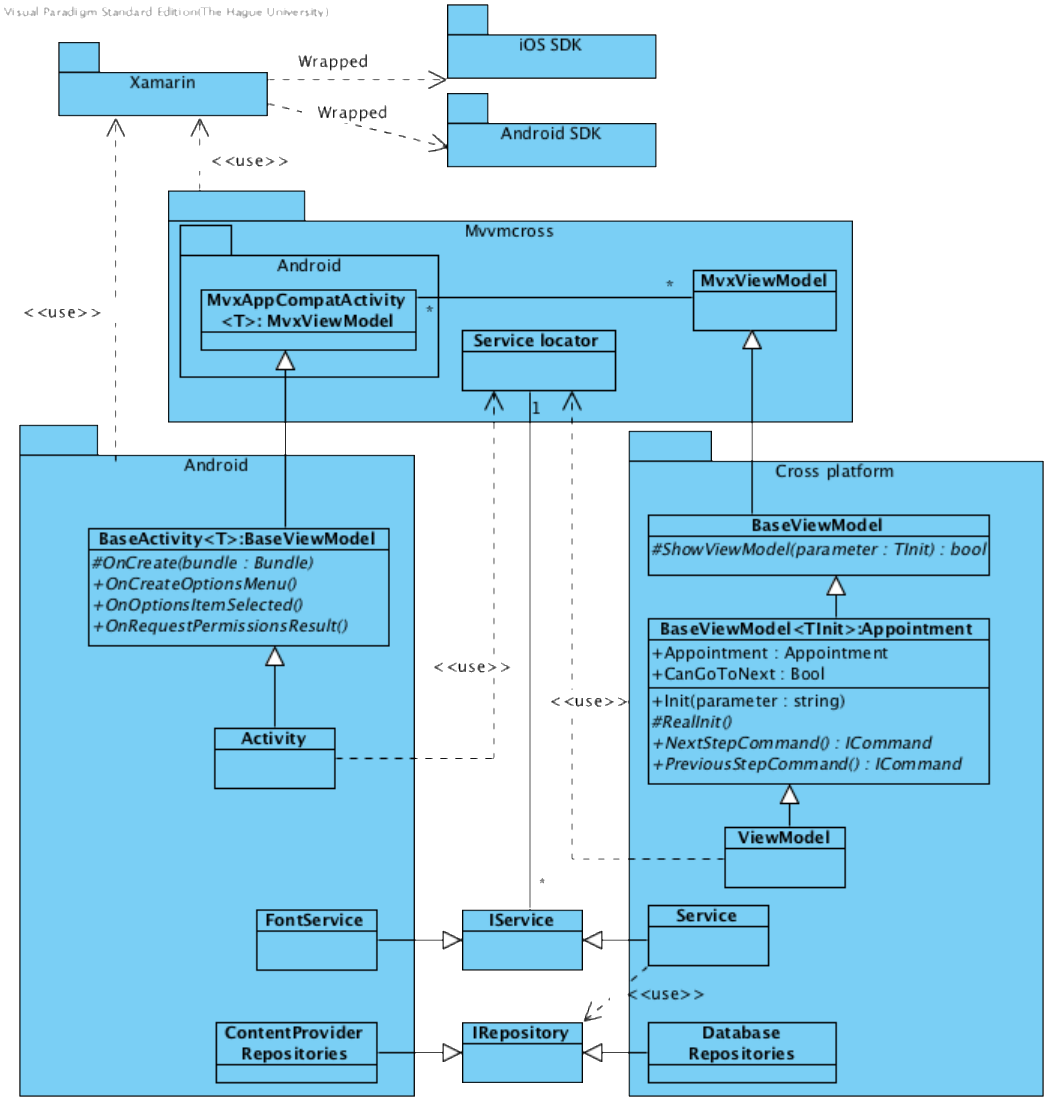
```
public class Migration3 : BaseMigration
{
    override protected List<string> GetCommands()
    {
          return new List<string> {
          "ALTER TABLE Category ADD COLUMN Visible boolean NOT NULL DEFAULT '1';",
          "ALTER TABLE \"Tag\" ADD COLUMN Visible boolean NOT NULL DEFAULT '1';",
          "ALTER TABLE Event ADD COLUMN MadeInApp boolean NOT NULL DEFAULT '0';"
          };
    }

    public override int GetVersion()
    {
          return 3;
    }
}
```

Alle migration klassen moeten de methodes GetCommands en GetVersion overriden, omdat zonder deze methoden een migration niet kan werken. In de methode GetCommands staan de sql statements die uitgevoerd zullen worden. En in GetVersion de versie van de betreffende migratie. Deze versie is altijd hoger dan voorgaande migraties, omdat migraties met een versie lager dan de database versie niet worden uitgevoerd. De SQL statements worden één voor één uitgevoerd waardoor het vrij eenvoudig is om fouten te vinden in een migratie.
Ten slotte wordt na het uitvoeren van elke Migration de database versie opgehoogd in het configuratie bestand van de app.

4.3.5. Repository pattern

Omdat ik al bezig was met het schrijven van de database migratie code, leek het mij ook handig om voor de duidelijkheid en leesbaarheid van de verschillende services om het repository pattern te implementeren. Het repository pattern heeft ervoor gezorgd dat ik inplaats van één grote klasse voor het communiceren met de database nu voor elke tabel in de database een klasse heb. Dit maakt het makkelijker om wijzigingen aan te brengen, doordat elke klasse nu verantwoordelijk is voor zijn eigen gedeelte in de database.

Het was hiervoor wel nodig om het ontwerp van de app iets aan te passen, waardoor het Er nu uitziet zoals in figuur 4.3.3.



Figuur 4.3-3, Nieuw ontwerp met repository pattern

In het nieuwe ontwerp heeft elke service een repository behalve de FontService (hoofdstuk 4.1.6 tonen iconen en eigen lettertype). Een repository bevat de code die communiceert met of de SQLite database of een Content provider (hoofdstuk 4.1.4 Contentprovider). De code om te communiceren met de SQlite database staat in het crossplatform gedeelde, omdat dit niet afhankelijk is van een platform en het Content provider gedeelde staat in het Android gedeelde. Het repository pattern zorgt ervoor dat de services niet aangepast moeten worden bij het veranderen van platform van Android naar bijvoorbeeld iOS, maar dat alleen de repositories aangepast hoeven te worden.

4.3.6. Performance

Omdat de app soms vrij traag reageerde, vooral bij de overgang naar het datum en tijd scherm, waarbij ik niet kon achterhalen wat de rede was voor deze vertragingen. Heb ik gebruik gemaakt van Xamarin profiler.

Xamarin profiler is een programma waarmee de volgende aspecten van een app geanalyseerd kunnen worden:

- Het geheugen gebruik

- Uitvoer tijd van methoden
- Hoevaak er naar het geheugen geschreven en vanaf gelezen wordt.

Voor het analyseren van het probleem heb ik gekeken naar het geheugen gebruik van de app en de uitvoer tijd van methoden. Tijdens het analyseren is gebruik gemaakt van het volgende toestel:

- OnePlus 3, Android 7.0

Voor het verkrijgen van nuttige resultaten zijn de volgende stappen vijf keer herhaald per toestel:

- App opstarten.
- Categorie selecteren.
- Naar het datum en tijd scherm gaan.
- 5 keer de kalender naar rechts swipen.
- Tijd selecteren.
- Naar het kalender selectie scherm gaan.
- Terug naar het datum en tijd scherm.
- 10 keer swipen in verschillende richtingen.

Stap 6 tot en met 8 werden elke keer gedurende één minuut herhalend uitgevoerd.

Ik heb voor deze stappen gekozen, omdat je een gedeelte van deze stappen kan vergelijken met de normale manier van interactie die een gebruiker zal hebben met de app. En het probleem kwam goed naar voren.

De resultaten van de analyse worden hieronder weergegeven.

4.3.6.1. Voor de wijzigingen

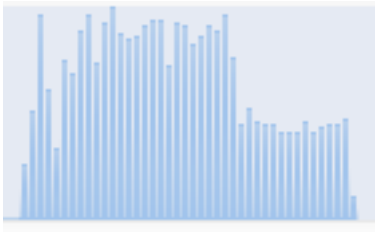
Door het plaatsen van stopwatches in de code. Een stopwatch meet de tijd tussen het starten en stoppen van de betreffende stopwatch. Deze stopwatches zijn geplaats voordat de view begint met laden en als de view klaar is. De tijden die hieruit zijn gekomen zijn te vinden in tabel 4.3-4.

Test	Tijd in MS (Milliseconden)
1	3535
2	3962
3	3323
4	3661
5	3411

Tabel 4.3-4, stopwatch tijden voor wijzigingen

CPU gebruik app

In het grafiekje hiernaast wordt het CPU gebruik van de app weergegeven voor het laden van het datum en tijd scherm in de schaal 0 tot 100 procent. Zoals te zien valt is het CPU gebruik vrij hoog aan het begin, dit komst overeen met het laden van de datum en tijd stap. Het CPU gebruik zakt direct weer omlaag, zodra het klaar is met laden.



Figuur 4.3-4, CPU gebruik app, voor wijzigingen

Geheugen gebruik OnePlus 3

Class	Count	Size
System.**String**	65.348	4.1 MB
System.**RuntimeType**	19.532	915.6 KB
System.Collections.Generic.Dictionary.**Entry<System.IntPtr,**	12	788.3 KB
Java.Interop.**JniMethodInfo**	14.550	682 KB
System.**Int32[]**	4.123	674 KB
<>__AnonType0<System.Reflection.TypeInfo>	17.852	418.4 KB
System.**Object[]**	5.227	403.2 KB

Figuur 4.3-5, geheugen gebruik, voor wijzigingen

Het geheugen gebruik van de app is vrij normaal en lager dan ik had verwacht ongeveer 9 MB. Dit kan dus uitgesloten worden als oorzaak.

Wat duidelijk geworden was met deze resultaten was dat er teveel gebeurde bij het laden van het datum en tijd scherm waardoor de app eventjes niet meer reageert totdat het weer het werk ingehaald heeft. Om dit op te lossen is er gebruik gemaakt van de Task klasse de Task klasse zorgt ervoor dat stukjes code op een andere thread uitgevoerd worden dan de hoofd thread van de app zelf, dit zorgt ervoor dat de app actief blijft terwijl er in de achtergrond allerlei taken uitgevoerd worden.

```
Task.Run(() => {
      ShowViewModel<DateAndTimeViewModel, Appointment>(Appointment);
});
```

Bovenstaande is een klein voorbeeld van een Task die asynchroon het Datum en tijd scherm laad. Dit kan in een methode neergezet worden waardoor het laden van de view in de achtergrond gebeurt en de rest van de methode uitgevoerd wordt zonder dat er gewacht moet worden totdat het laden van het scherm klaar is.

De Task klasse is toegepast bij het laden van datum en tijd scherm en het opbouwen van de kalender schermen (hoofdstuk 4.2.6 kalender component)

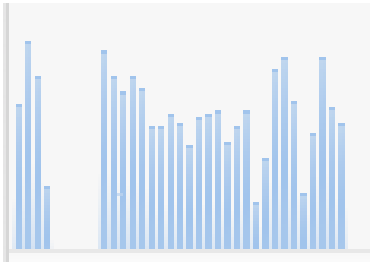
4.3.6.2. Na de wijzigingen

Ook na het toevoegen van de Task klasse zijn de test stappen weer uitgevoerd. De resultaten daarvan zullen hier besproken worden. Zoals te zien is in tabel 4.3-5 zijn de laadtijden flink naar beneden gezakt. Voorheen was de gemiddelde tijd om het datum en tijd scherm volledig te laden 3578 MS, dat is nu 1491 MS. Wat een flinke vooruitgang is vanuit het perspectief van een gebruiker, aangezien die niet meer zit te kijken naar een app die misschien vastgelopen is.

Test	Tijd in MS (Milliseconden)
1	1535
2	1461
3	1323
4	1682
5	1456

Tabel 4.3-5, stopwatch tijden na wijzigingen

CPU gebruik app
In figuur 4.3-5 wordt het CPU gebruik getoond na het aanbrengen van de wijzigingen en hier valt op dat de app een stuk minder gebruikt dan voorheen.



Figuur 4.3-5, CPU gebruik na wijzigingen

Bovenstaande test stappen zijn eveneens uitgevoerd met een Motorola XT waarop Android 4.1 staat. De resultaten van dit toestel zijn niet getoond, omdat de resultaten overeenkwamen met de OnePlus 3 zowel voor als na met betrekking tot reactie snelheid van de app.

4.3.7. Crashlytics

Voor het verkrijgen van verschillende statistieken over de app is gebruik gemaakt van Crashlytics. Crashlytics is een systeem voor het rapporteren van crashes wat gebruikt kan worden voor Android, iOS apps en games gemaakt met de Unity game engine. Crashlytics is geïnstalleerd op meer dan 2.5 biljoen apparaten.

Ik heb Crashlytics aan de app toegevoegd voor het verkrijgen van meldingen als de app crasht. Op deze manier kunnen deze crashes snel opgelost worden, waardoor een gebruiker minder snel de neiging krijgt om een één ster review achter te laten. Ook kan crashlytics verschillende andere statistieken tonen over een app waaronder: aantal actieve gebruikers op dit moment, groei van het aantal gebruikers per dag, locatie van gebruikers.

Omdat Crashlytics niet officieel beschikbaar is voor Xamarin was het nodig om zelf een Binding library (hoofdstuk 4.2.5 Binding library) te maken of een bestaande library te gebruiken, doordat er meer mensen zijn die Crashlytics op Xamarin gebruiken was er vrij veel keuze tussen Binding library's en heb ik gekozen voor de library met de meeste downloads. Na het importeren daarvan was het enige stukje code wat nodig om Crashlytics te activeren het volgende.

```
Fabric.With(applicationContext, new Bindings.CrashlyticsKit.Crashlytics());
```

Bovenstaande regel code moest in de Setup klasse neergezet worden, omdat daar de Android app gestart wordt. Voor Android zou het dus in de Application klasse neergezet moet worden.

Naast de betreffende regel code moesten er ook nog twee dingen toegevoegd worden aan Strings.xml en de Androidmanifest.

```
<string name="com.crashlytics.android.build_id">e9e6beb9c4284289ac68b9ab76a9ee56</string>
```

Bovenstaande string met willekeurige letters en cijfers dit koppelt Crashlytics aan een versie nummer dus zou dit veranderd moeten worden naar iets anders als de app versie veranderd.

```
<meta-data
      android:name="io.fabric.ApiKey"
      android:value="XXXXXX" />
```

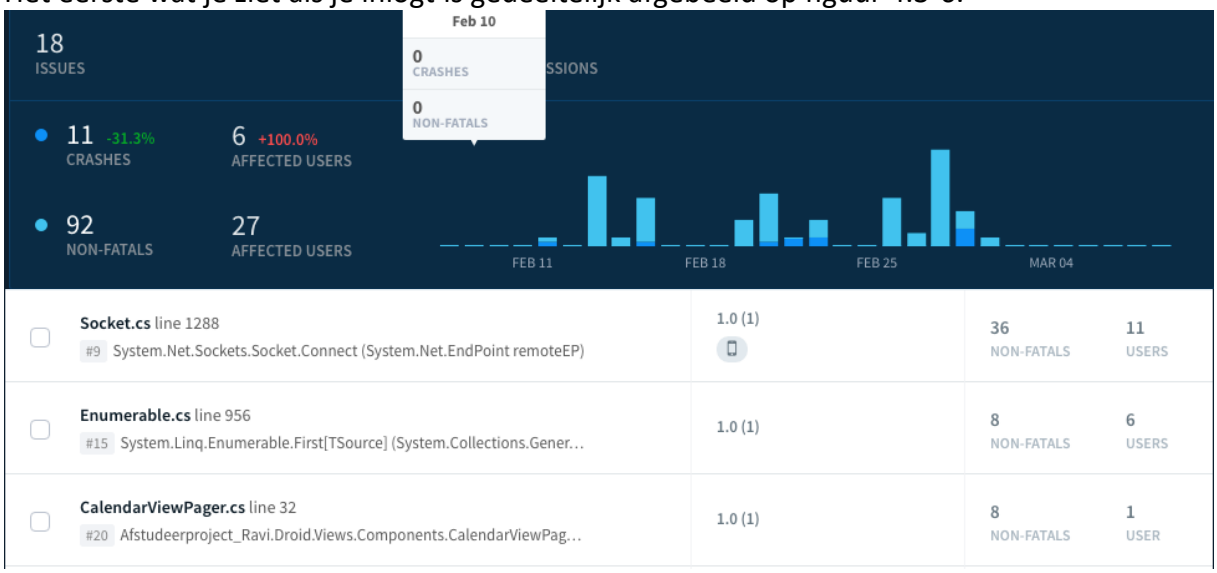
En bovenstaand stukje tekst in de Application tag in Androidmanifest. Om Crashlytics te gebruiken is er een ApiKey nodig die daarin geplaatst dient te worden.

Om handmatig dingen te registreren bijvoorbeeld in een catch statement, iets wat Crashlytics niet zal rapporteren omdat het systeem de exceptie afhandelt. Kan het volgende gebruikt worden, om alsnog de fout op te sturen.

```
Crashlytics.Instance.RecordException(ex);
```

Nadat Crashlytics is geconfigureerd kan er op de web portal ingelogd worden om de verschillende statistieken te bekijken.

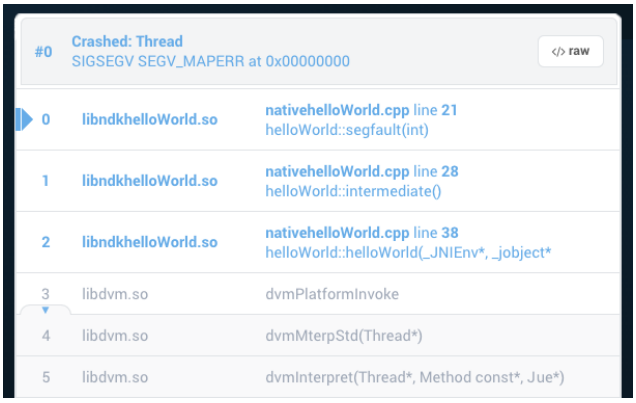
Het eerste wat je ziet als je inlogt is gedeeltelijk afgebeeld op figuur 4.3-6.



Figuur 4.3-6, Crashlytics dashboard

Dit scherm laat een lijst met crashes zien en de statistieken daarvan zoals, hoeveel gebruikers een crash hebben meegemaakt, of je app meer crasht dan voorheen of minder en het aantal crashes per dag.

De lijst met crashes laat voor elke crash een stacktrace (figuur 4.3-7) zien waarop te zien is waar in de app de fout heeft plaatsgevonden en welke methoden er vooraf zijn aangeroepen.

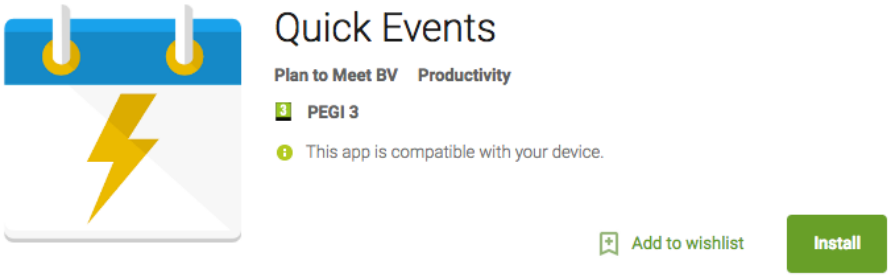


Figuur 4.3-7, voorbeeld stacktrace

4.3.8. Resultaat

In deze sprint is de definition of done gehaald, doordat het instellingen scherm geïmplementeerd is en behalve US22 en US23 zijn alle resterende user stories geïmplementeerd. Deze twee user stories zijn niet geïmplementeerd, omdat deze een lage prioriteit hebben. En dus opgepakt zullen worden als er tijd over is aan het eind van het project.

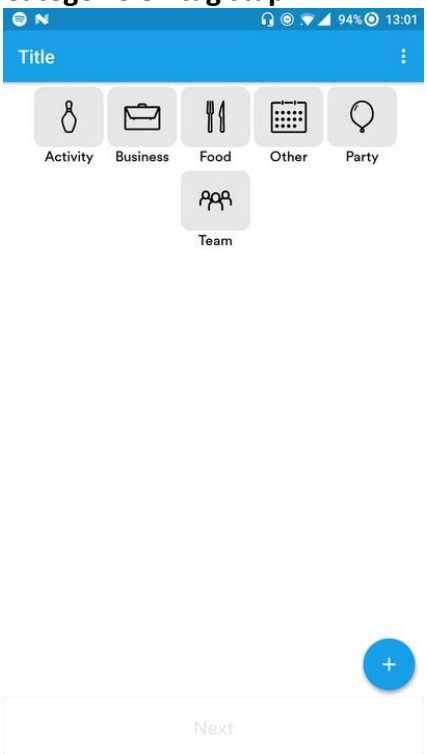
Na het verwerken van de feedback die ik gekregen had aan het eind van sprint 3, staat de app inmiddels ook op de Google Play Store.



Figuur 4.3-8, app pagina in Google Play Store [https://play.google.com/store/apps/details?id=com.cally.quickevents]

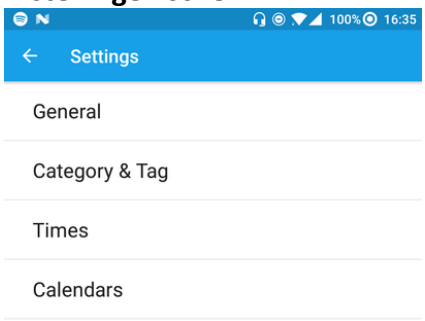
Hieronder worden een aantal schermen getoond waaronder de instellingen en het nieuwe categorie en tag scherm waarbij er gebruik gemaakt wordt van een zogeheten FloatingActionButton voor het toevoegen van categorieën en tags.

Categorie en tag stap



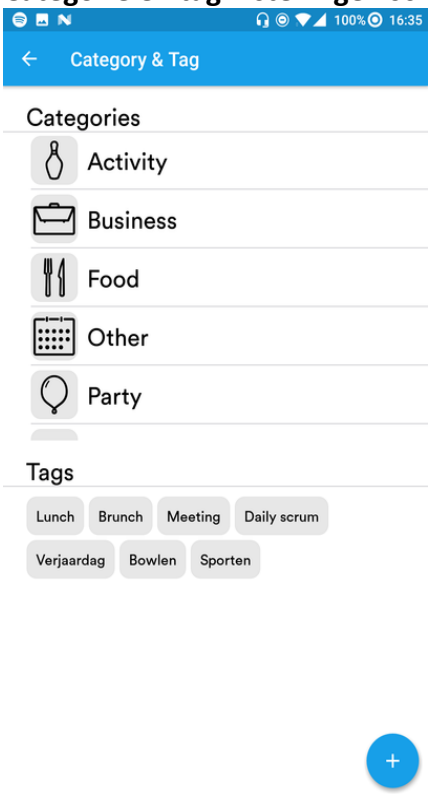
Figuur 4.3-9, categorie en tag stap

Instellingen scherm



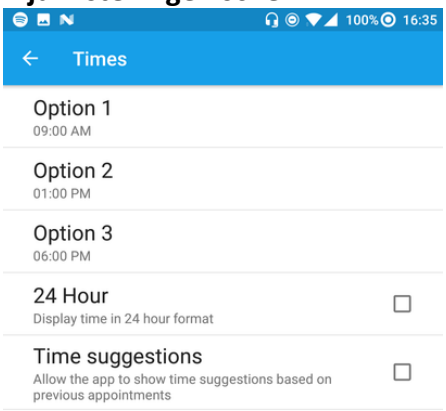
Figuur 4.3-10, Instellingen scherm

Categorie en tag instellingen scherm



Figuur 4.3-11, categorie en tag instellingen

Tijd instellingen scherm



Figuur 4.3-12, tijd instellingen

Deel 3

De evaluatie

5. Evaluatie

In dit hoofdstuk zullen de verschillende evaluaties plaatsvinden.

5.1. Productevaluatie

Het eindproduct is vrijwel hetzelfde als is besproken in het afstudeerplan, waarbij het een stand alone applicatie is waar snel een afspraak in gemaakt kan worden. Het integreren als datum prikker module is geschrapt, omdat de app meer prioriteit had. De app staat nu ook al een aantal dagen in de Google Play Store, wat een mooie afsluiting is van het project.

5.1.1. Plan van aanpak

Het opstellen van het plan van aanpak zorgde niet voor veel problemen, dit kwam doordat voor een overgrote deel het afstudeerplan aangehouden kon worden.

5.1.2. Requirements

Tijdens het project zijn er redelijk wat requirements en user stories bijgekomen na de initieel opgestelde. Dit kwam doordat er bij elke sprint aan het begin en eind gekeken werd wat er in de huidige en volgende sprint gedaan zou worden en uit deze gesprekken voortkwam wat anders moet of wat er miste.

Dit is vrij normaal bij een Scrum project en zou dus zo weer kunnen gebeuren bij een volgend project.

5.1.3. Ontwerp

Zoals wel vaker gebeurt bij een project met sprints vinden er tussendoor wijzigingen plaats waardoor ook het ontwerp aangepast moet worden. Dit waren echter kleine veranderingen en zorgde ervoor dat het ontwerp wat aan het begin gemaakt was niet echt veel verschillen heeft met het eind ontwerp.

Het ontwerp wat voor dit project gebruikt is zou hergebruikt kunnen worden voor andere Xamarin Mvvmcross apps, aangezien de opbouw hetzelfde zal zijn.

5.1.4. Test rapport

Het uitvoeren van testen was in vergelijking met andere projecten op school niet veel anders. Dit was dan ook niet heel erg uitdagend. Het hebben van de test cases was wel handig, omdat er dan duidelijk was wat er allemaal getest moest worden.

5.2. Procesevaluatie

In deze paragraaf zal ik het proces evalueren.

Tijdens het project heb ik sprints bijgehouden. Wat ik hieraan kon verbeteren was het beter bijhouden van de sprints, omdat ervan tevoren duidelijk is wat er in de sprint gemaakt moet worden. Waardoor sommige taken die ingepland waren niet op tijd uitgevoerd konden

worden zoals het testen in sprint 1, dit had voor de rest geen verdere volgen voor het project doordat ik er zelfstandig aan werkte.

Naar mijn gevoel paste de ontwikkelmethode perfect bij de het project, omdat er in sprints gewerkt werd was er altijd een specifiek doel wat gehaald moest worden. Dit doel is ook altijd gehaald wat een mooi pluspunt is.
die gebruikt is paste perfect bij het project, doordat er genoeg contact momenten waren maar er niet zoals bij een normaal scrum proces dagelijks korte vergaderingen zijn hoe het project verloopt maar minstens één keer per week of meer als het nodig is.

Voor een volgend project als het mogelijk is om Scrum als ontwikkelmethode te kiezen zal dat dus ook gebeuren.

Het werken met Mvvmcross was wel even wennen, maar na de eerste week was het wel duidelijk hoe er met Mvvmcross gewerkt moest worden en wat je er allemaal mee kan. Mvvmcross zou ik zeker nog een keer willen gebruiken bij een toekomstige app.

5.3. Evaluatie beroepstaken

In de paragrafen hieronder staan alle competenties die binnen het project gebruikt zijn. Deze competenties zijn allemaal zelfstandig uitgevoerd. Dit houdt in dat het niveau van een competentie tussen de 2 en 4 kan liggen.

Context		Taakrol		
		Geleid	zelfstandig	Sturend
	simpel	1	2	3
	lastig	2	3	4
	complex	3	4	5

Figuur 5.3-1, Competentie matrix,
[Beroepstaken van de opleiding Informatica –Academie voor ICT & Media (juni 2009, versie 1.1)]

5.3.1. Uitvoeren analyse door definitie van requirements (1.4)

Hoewel er geen grote hoeveelheid requirements opgesteld hoefde te worden gedurende dit project, waren er wel twee bronnen waaruit deze eisen naar voren moesten komen (hoofdstuk 3.5 Initiële user stories). Daarnaast kwamen er bij elke sprint nieuwe requirements en user stories bij of werden bestaande requirements en user stories gewijzigd (hoofdstuk 4.1.2, 4.2.2, 4.3.2), waardoor er continu versiebeheer plaats moest vinden.

Al deze punten samen zorgde voor een complexiteit van ‘lastig’, waardoor niveau 3 behaald is.

5.3.2. Ontwerpen, bouwen en bevragen van een database (2.2)

Voor dit project is er gewerkt met een SQlite database die bestaat uit meerdere tabellen die een relatie met elkaar hebben (hoofdstuk 3.8, 4.2.4). Bij het communiceren met de database is het regelmatig zo dat er meerdere tabellen aangesproken worden (hoofdstuk 4.2.7 Intelligentie).

Ook is er een database gemaakt voor de web api waarbij gebruik gemaakt wordt van stored procedures (hoofdstuk 4.2.9 backen web api, Bijlage 4).

Al deze punten samen zorgde voor een complexiteit van ‘lastig’, waardoor niveau 3 behaald is.

5.3.3. Ontwerpen systeemdeel (3.2)

Doordat er een mobiele app gemaakt werd en er gewerkt werd met het Mvvm design pattern was het nodig om een ontwerp te bedenken wat zo generiek mogelijk was (hoofdstuk 3.7) voor dit ontwerp is ook rekening gehouden met een aantal design patterns (hoofdstuk 3.7.1). Waarbij er in de laatste sprint een design pattern is bijgekomen (hoofdstuk 4.2.6). Voor het maken van de verschillende diagrammen is gebruik gemaakt van de UML ontwerp techniek waarbij de diagrammen gemaakt zijn in Visual Paradigm (hoofdstuk 3.4 gebruikte tools).

Deze punten samen zorgen voor een complexiteit van ‘lastig, waardoor niveau 3 behaald is.

5.3.4. Bouwen applicatie (3.3)

Doordat er gebruik gemaakt is van het Xamarin en Mvvmcross framework, waarbij er rekening gehouden is met platform specifieke en crossplatform functionaliteit. En er naast de app ook nog een Web API (hoofdstuk 4.2.9 Backend Web API) gemaakt is. En er voor de app gebruik gemaakt is van verschillende design patterns (hoofdstuk 3.7.1 en 4.3.5) en technieken (hoofdstuk 4.1.6 en 4.2.6).

Zorgen deze punten samen voor een complexiteit van ‘complex’, waardoor niveau 4 behaald is.

5.3.5. Uitvoeren van en rapporteren over het testproces (3.5)

Voor het testen van de app is gebruik gemaakt van verschillende test frameworks en technieken (hoofdstuk 4.2.8 testen), waarbij er bij elke sprint testen uitgevoerd zijn behalve bij sprint 1, de resultaten hiervan staan in bijlage 5.

Al deze punten samen zorgde voor een complexiteit van ‘lastig’, waardoor niveau 3 behaald is.

6. Begrippenlijst

Begrip	Betekenis
Assembly	Gecompileerde stukjes code, een programma kan uit meerdere assemblies bestaan.
Categorie	Soort of verzameling van het zelfde
Inversion of control	Design pattern voor het weghalen van afhankelijkheden in code
Native	Gemaakt voor een specifiek besturingssysteem (Android, iOS)
Overerven	Klasse erft eigenschappen en methoden van een super klasse
Runtime	Wanneer een programma uitgevoerd wordt
Saas pakket	Een Saas pakket is een software pakket die aangeboden wordt door een externe partij waarbij zij de software beheren en jij als gebruiker alleen gebruik kan maken van de betreffende software.
SDK	Verzameling hulpmiddelen die gebruikt worden bij het ontwikkelen van software voor een bepaald besturingssysteem, hardware e.d.
Tag/s	Een tag is een ander woord voor een titel. Een tag kan bijv. het volgende zijn: Sporten, Eten, Vergadering enzovoort.
Wrapper	Laag van abstractie over bestaande functionaliteit.

7. Bronnen

Nuts & Bolts of Xamarin.Android. (z.j.). Geraadpleegd van http://sharpmobilecode.com/nuts-bolts-of-xamarin-android/

Soroka, T. (z.j.). Xamarin.Forms – mobile revolution. Geraadpleegd van https://www.linkedin.com/pulse/xamarinforms-mobile-revolution-tomasz-soroka

Content Provider Basics. (z.j.). Geraadpleegd van https://developer.android.com/guide/topics/providers/content-provider-basics.html

Calendar Provider. (z.j.). Geraadpleegd van https://developer.android.com/guide/topics/providers/calendar-provider.html

The Service Locator Pattern. (z.j.). Geraadpleegd van https://msdn.microsoft.com/en-us/library/ff648968.aspx

NUnit, (z.j.). Geraadpleegd van https://www.nunit.org/

8. Bijlages

1) Afstudeerplan
2) Plan van aanpak
3) Requirements document
4) Technisch & functioneel ontwerp
5) Testplan en rapportage

Bijlage 1) Afstudeerplan

Afstudeerplan

Informatie afstudeerder en gastbedrijf (*structuur niet wijzigen*)

Afstudeerblok: 2016-2.2 (start uiterlijk 14 november 2016)
Startdatum uitvoering afstudeeropdracht: 14-11-2016
Inleverdatum afstudeerdossier volgens jaarrooster: 20 maart 2017

Studentnummer: 13031198
Achternaam: dhr Gajadin (*) *weghalen niet van toepassing*
Voorletters: R
Roepnaam: Ravi
Adres: Lau Mazirellaan 191
Postcode: 2525ZM
Woonplaats: Den Haag
Telefoonnummer:
Mobiel nummer: 0631000441
Privé emailadres: ravi-g@live.nl

Opleiding: Informatica
Locatie: Den Haag (*) *weghalen niet van toepassing*
Variant: voltijd

Naam studieloopbaanbegeleider: Anneke Wieman
Naam begeleidend examinator: G.A. Mijnarends
Naam tweede examinator: E.M. van Doorn

Naam bedrijf: Webbeat Products BV
Afdeling bedrijf:
Bezoekadres bedrijf: Turfschipper 7-9
Postcode bezoekadres: 2292 JC
Postbusnummer:
Postcode postbusnummer:
Plaats: Wateringen
Telefoon bedrijf: 088 88 22 280
Telefax bedrijf:
Internetsite bedrijf: http://www.webbeat.nl

Achternaam opdrachtgever: dhr Merkelbach (*) *weghalen niet van toepassing*
Voorletters opdrachtgever: G
Titulatuur opdrachtgever:
Functie opdrachtgever: Directeur
Doorkiesnummer opdrachtgever:
Email opdrachtgever: geert.merkelbach@webbeat.nl

Achternaam bedrijfsmentor: dhr de Brabander (*) *weghalen niet van toepassing*
Voorletters bedrijfsmentor: L
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: App Developer
Doorkiesnummer bedrijfsmentor:
Email bedrijfsmentor: leon.de.brabander@webbeat.nl
NB: bedrijfsmentor mag dezelfde zijn als de opdrachtgever

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:
Ontwikkelen van een Android app voor het maken van afspraken bij Webbeat Products BV.

Opdrachtomschrijving *(toelichtende tekst verwijderen)*

1. **Bedrijf**

   Webbeat Products BV is een profit bedrijf in de ICT branche, die sinds 2000 bestaat en opgericht is door Geert Merkelbach. De diensten die Webbeat aanbiedt zijn het ontwerpen en ontwikkelen van apps, responsive websites en online platformen. Op dit moment zijn er ongeveer 20 werknemers
   Eén voorbeeld van een product dat gemaakt is door Webbeat is de internationale tracking portal van PostNL.
   In 2012 zijn Nalta en Webbeat volledig gefuseerd. Nalta doet aan software development en professional IT services. Daarnaast zijn ze ook verbonden aan verschillende online diensten zoals: Filterworks, Optispeech, Datumprikker en Week Calendar.

2. **Probleemstelling**

   De standaard Android agenda app wordt door onvoldoende mensen intensief gebruikt, doordat het maken van een afspraak een saaie puur functionele invuloefening van een standaard scherm is. Om extra functionaliteit toe te voegen aan de agenda, zoals bijvoorbeeld plannen o.b.v. Artificial Intelligence, is een beter bijgehouden agenda heel waardevol.

   Om het aanmaken van een afspraak op een mobiele telefoon zo laagdrempelig en attractief mogelijk te maken, moet er een Xamarin android app gemaakt worden die zich alleen richt op het aanmaken van een afspraak. Dit aanmaakproces moet net zo leuk en aantrekkelijk zijn als het plaatsen van een social media bericht.

3. **Doelstelling van de afstudeeropdracht**

   Het ontwikkelen van een Android Xamarin app waarin afspraken kunnen worden aangemaakt; die lokaal of via een API koppeling in de bestaande back-end applicatie van Datumprikker opgeslagen moeten worden. De app zal responsive moeten zijn, zodat deze een goede gebruikerservaring zal bieden.
   Verder zal de app gegevens moeten gebruiken die lokaal of via een API binnen gehaald moeten worden.

4. **Resultaat**

   Het resultaat is een Android Xamarin app, die op zichzelf staand aangeboden kan worden in de app store en daarnaast de mogelijkheid om deze als module te integreren in de Datumprikker app.

   De app moet de mogelijkheid hebben om afspraken in te maken die lokaal of via een API koppeling opgeslagen moeten worden.

   Ook moet er een mogelijkheid zijn om beschikbare tijden, locaties en deelnemers te selecteren die lokaal opgeslagen staan of via een API binnen te halen.

   Deze afspraken moeten ook gedeeld kunnen worden via onder anderen social media.

   De app moet responsive zijn en een goede gebruikers ervaring bieden.

   De waarde van de app is dat de gebruiker gemakkelijk in één app zowel groepsafspraken als normale afspraken kan aanmaken en hier tussen ook eenvoudig kan switchen. Door beide type afspraken te combineren, zullen de gebruikers de app vaker gebruiken en kunnen zij op 1 plek hun agenda bijhouden. Met als doel meer (actieve) gebruikers te krijgen.

5. **Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten**

   - Plan van aanpak 3 dagen
   - Vaststellen requirements 3 dagen
   - Vaststellen huidige situatie 3 dagen
   - Beschikbare documentatie raadplegen 3 dagen
   - Uitzoeken werking backend API’s 3 dagen

- Ontwerpen 25 dagen
- Ontwikkelen 25 dagen
- Testen 5 dagen
- Schrijven eindverslag 15 dagen

Binnen het bedrijf wordt de agile ontwikkelmethode gebruikt en daar zal tijdens deze opdracht ook mee gewerkt worden.

6. Op te leveren (tussen)producten

- Plan van aanpak
- Volledige lijst met functionele en niet functionele requirements
- Ontwerp
- De app
- Code & testen

7. Te demonstreren competenties en wijze waarop

1.4 Uitvoeren analyse door definitie van requirements Niveau 3
Requirements moeten goed en duidelijk beschreven worden.

3.2 Ontwerpen systeemdeel Niveau 3
Ontwerpen van de app.

3.3 Bouwen applicatie Niveau 4
De app moet zo gebouwd worden zodat het zowel standalone en als een module kan functioneren.

3.5 Uitvoeren van en rapporteren over het testproces Niveau 3
Voor het testen van de app zullen er technieken als mocking en automatische testen gebruikt worden.

Bijlage 2) Plan van aanpak

PLAN VAN AANPAK

V1.0

Ravi Gajadin

Quick Events | Webbeat Products BV

Inhoudsopgave

1. Inleiding 2
2. Opdrachtbeschrijving 2
1.1. Probleemstelling ..... **Error! Bookmark not defined.**
1.2. Doelstelling ..... **Error! Bookmark not defined.**
1.3. Resultaat ..... **Error! Bookmark not defined.**
1.4. Projectorganisatie 2
3. Op te leveren producten 3
4. Ontwikkelmethode 3
5. Planning 5

1. Inleiding

In dit document wordt beschreven wat de opdracht is die gemaakt gaat worden en de planning die daarbij gebruikt wordt.

2. Opdrachtbeschrijving

In dit hoofdstuk worden de probleem en doelstelling besproken en het resultaat van deze opdracht.

1.1. Probleemstelling

De standaard agenda app die wordt meegeleverd met Android wordt door onvoldoende mensen gebruikt, doordat het maken van een afspraak saai is en vaak ook erg lang duurt, doordat er veel getypt moet worden.

1.2. Doelstelling

De doelstelling van het project is een Xamarin Android app die zich alleen richt op het aanmaken van een afspraak. Dit aanmaak proces moet leuk en aantrekkelijk zijn waardoor er bijvoorbeeld in minder dan 10 seconden een afspraak gemaakt kan worden.

1.3. Resultaat

Het resultaat is een Android Xamarin app, die op zichzelf staand aangeboden kan worden in de Google Play Store. In de app is het alleen de bedoeling om afspraken te maken die in de lokale agenda app opgeslagen kunnen worden of via een API-koppeling gedeeld kunnen worden.

Deze afspraken kunnen gemaakt worden door een aantal stappen door de app te volgen, waarin de belangrijkste onderdelen van een afspraak ingevuld worden zoals de titel, datum en dergelijke. De app moet responsive zijn en een goede gebruikers ervaring bieden.

1.4. Projectorganisatie

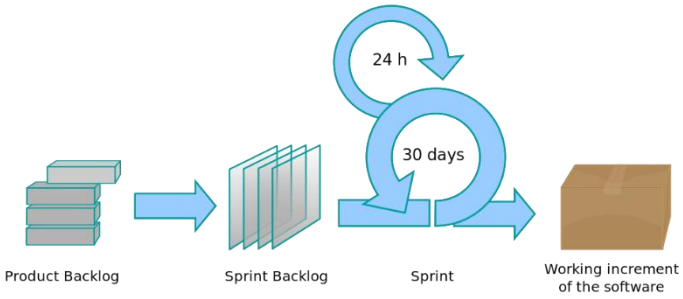
De opdrachtgever is Geert Merkelbach.
De begeleider is Leon de Brabander.

3. Op te leveren producten

De producten die opgeleverd gaan worden aan het eind van het project zijn:

- Plan van aanpak
- Planning
- De Requirements
- Use Case tabellen
- Use Case diagram
- Architectuur document
- Database diagram
- Testplan
- Test resultaten
- De app

4. Ontwikkelmethode



De ontwikkel methode die gebruikt gaat worden is scrum. Er is gekozen voor scrum, omdat niet alle requirements aan het begin bekend zijn. Op deze manier kan er na elke sprint bekeken worden wat er toegevoegd kan worden aan de app en heeft het voor de rest geen invloed op de kwaliteit, doordat na elke sprint een werkend product opgeleverd zal worden.

Gedurende het project zal er een product backlog zijn waar alle features instaan die gedurende het project gemaakt zullen worden. De product backlog wordt gebruikt om de sprint backlog te vullen. Elke sprint heeft een sprint backlog met features die in de sprint gemaakt worden. Deze features worden gerangschikt volgends de MoSCoW methode, zodat de belangrijkste features als eerste gemaakt worden.

Elke scrum team bestaat uit de volgende personen:
De product owner is Geert Merkelbach en bepaalt wat er gemaakt zal worden in elke sprint.
De scrum master is Leon de Brabander en hij stuurt het scrum team aan en is een buffer tussen het scrum team en de product owner.
Het scrum team bestaat uit Ravi Gajadin en zorgt ervoor dat de product owner na elke sprint een werkend product heeft.

Elke sprint zal bestaan uit een analyse, ontwerp, implementatie en testfase.

Er zullen sprints van 4 weken gehanteerd worden, omdat de opdrachtgever dan zeker kan zijn dat er na elke sprint een werkend product is wat gebruikt kan worden. Als na de eerste sprint blijkt dat 4 weken toch te veel is kan er een sprint toegevoegd om sprints van 3 weken te krijgen.

Gebruikte methoden en technieken

- Bitbucket als version control system
- Uml voor het modelleren van diagrammen
- Sqlite voor de database

Afhankelijk van het resultaat van het onderzoek zal er gewerkt worden met een van de onderstaande technieken en frameworks:

- Xamarin en C#
- Xamarin, C# en MvvmCross
- Xamarin.Forms en C#
- Java Android

5. Planning

Nr	Taken	Begindatum	Einddatum	Weken
1	**Planning**	**14-11-2016**	**25-11-2016**	**2,00**
1.1	Werkomgeving instellen	14-11-2016	15-11-2016	
1.2	Plan van aanpak maken	15-11-2016	18-11-2016	
1.3	Vaststellen specificaties en requirements	17-11-2016	22-11-2016	
1.4	Use cases maken	17-11-2016	22-11-2016	
1.5	Meeting met Leon	14-11-2016	25-11-2016	
1.6	Onderzoek MvvmCross of Xamarin.Forms of Native Android	22-11-2016	25-11-2016	
1.7	Taken indelen sprint 1 Leon & Geert	24-11-2016	25-11-2016	
2	**Sprint 1**	**28-11-2016**	**23-12-2016**	**4,00**
2.1	Meeting met designers om 13:00	28-11-2016	19-12-2016	
2.2	Meeting met Leon	28-11-2016	21-12-2016	
2.3	Documentatie maken	28-11-2016	09-12-2016	
2.4	App maken	28-11-2016	19-12-2016	
2.5	Testen maken	30-11-2016	23-12-2016	
2.6	Feedback en bugs verwerken	20-12-2016	23-12-2016	
2.7	App testen	08-12-2016	23-12-2016	
2.8	Vaststellen requirements en taken Sprint 2 Leon & Geert	22-12-2016	23-12-2016	
3	**Sprint 2**	**28-12-2016**	**27-01-2017**	**4,00**
3.1	Documentatie maken	28-12-2016	30-12-2016	
3.2	Meeting met designers om 13:00	28-12-2016	27-01-2017	
3.3	Meeting met Leon	28-12-2016	27-01-2017	
3.4	App maken	02-01-2017	20-01-2017	
3.5	Testen maken	16-01-2017	23-01-2017	
3.6	Feedback en bugs verwerken	18-01-2017	24-01-2017	
3.7	App testen	05-01-2017	27-01-2017	
3.8	Vaststellen requirements en taken Sprint 3 Leon & Geert	27-01-2017	27-01-2017	
4	**Sprint 3**	**30-01-2017**	**24-02-2017**	**4,00**
4.1	Documentatie maken	30-01-2017	02-02-2017	
4.2	Meeting met designers om 13:00	30-01-2017	24-02-2017	
4.3	Meeting met Leon	30-01-2017	24-02-2017	
4.4	App maken	30-01-2017	24-02-2017	
4.5	Testen maken	13-02-2017	24-02-2017	
4.6	Feedback en bugs verwerken	16-02-2017	24-02-2017	
4.7	App testen	16-02-2017	24-02-2017	
5	**School**	**05-12-2016**	**20-03-2017**	**15,00**
5.1	Afspraak maken voor	05-12-2016	27-02-2017	
5.1.1	Bedrijfsbezoek - 25% 14:30	21-12-2016	21-12-2016	
5.1.2	Voortgangsverslag inleveren - 45%	27-12-2016	05-01-2017	

5.1.3	Bespreking van het concept afstudeerdossier - 60%	16-01-2017	24-01-2017	
5.1.4	Tussentijds assessment - 3 weken voor eind	20-02-2017	27-02-2017	
5.2	Afstudeerdossier afmaken	27-02-2017	17-03-2017	
5.3	Afstudeerdossier inleveren - Voor 12:00	20-03-2017	20-03-2017	

Bijlage 3) Requirements document

REQUIREMENTS

v2.0

Ravi Gajadin

Quick Events | Webbeat Products BV

Versiebeheer

Versie	Datum wijziging	Beschrijving van veranderingen
1.0	17-11-2016	Initiële versie
1.1	18-11-2016	Requirements toegevoegd
1.2	21-11-2016	Use cases toegevoegd
1.3	22-11-2016	Requirements, Use cases, Flow van schermen toegevoegd.
1.4	23-11-2016	Requirements, Use cases, Flow van schermen, Activity diagram, use case lijst toegevoegd en document opgemaakt
1.5	24-11-2016	Activity diagram bewerkt en use case 1
1.6	25-11-2016	Requirements aangepast
1.7	02-12-2016	Taken toegevoegd
1.8	19-12-2016	Requirements bijgewerkt
1.9	28-12-2016	Sprint 2 requirements en user stories toegevoegd
2.0	08-03-2017	Sprint 3 requirements en user stories toegevoegd

Inhoudsopgave

Versiebeheer 1
Inhoudsopgave 2
1. Requirements 3
1.1. Functionele requirements 3
1.2. Niet functionele requirements 3
1.3. User stories 4
2. Use Cases 5
2.1. Diagram 5
2.2. Use Case lijst 5
2.3. Tabellen 6
3. Flow van schermen 111
4. Activity diagram aanmaken afspraak 122

1. Requirements

1.1. Functionele requirements

ID	Omschrijving	MoSCoW
FR1	De app heeft de mogelijkheid om een afspraak in aan te maken.	Must have
FR2	De app heeft de mogelijkheid om categorieën toe te voegen.	Must have
FR3	De app heeft de mogelijkheid om categorieën te bewerken.	Should have
FR4	De app heeft de mogelijkheid om categorieën te verwijderen	Should have
FR5	De app heeft de mogelijkheid om tags aan te maken.	Must have
FR6	De app heeft de mogelijkheid om tags te bewerken.	Should have
FR7	De app heeft de mogelijkheid om tags te verwijderen.	Should have
FR8	De app heeft de mogelijkheid om een standaard agenda te kunnen instellen.	Could have
FR9	De app heeft de mogelijkheid om een afspraak te kunnen delen	Could have
FR10	De app heeft de mogelijkheid om suggesties te tonen voor tijden	Must have
FR11	De app heeft de mogelijkheid om te kunnen kiezen tussen Nederlands en Engels als taal	Could have
FR12	De app heeft de mogelijkheid om standaardtijden in te stellen	Should have
FR13	De app heeft de mogelijkheid om tijden in 24 uurs of 12 uurs formaat weer te geven.	Should have
FR14	De app heeft de mogelijkheid om agenda's te kiezen de gebruikt gaan worden	Should have

1.2. Niet functionele requirements

Inclusief kwaliteitseigenschappen van de ISO 9126 standaard.

ID	Omschrijving	MoSCoW
NF1	De app moet binnen 2 seconde reageren op input van de gebruiker.	Must have
NF2	De app moet laten zien dat het bezig is als het meer dan 2 seconden nodig heeft om te reageren op een actie van de gebruiker.	Must have
NF3	De app moet zo gemaakt zijn dat iemand die de app voor de eerste keer gebruikt, binnen 2 minuten weet hoe de app werkt en wat je er allemaal mee kan.	Must have
NF4	Een afspraak maken moet ongeveer 10 seconden duren.	Must have
NF5	De app moet overgezet kunnen worden naar iOS door alleen de iOS GUI te implementeren.	Must have
NF6	De app moet minimaal Android API 16 ondersteunen	Must have

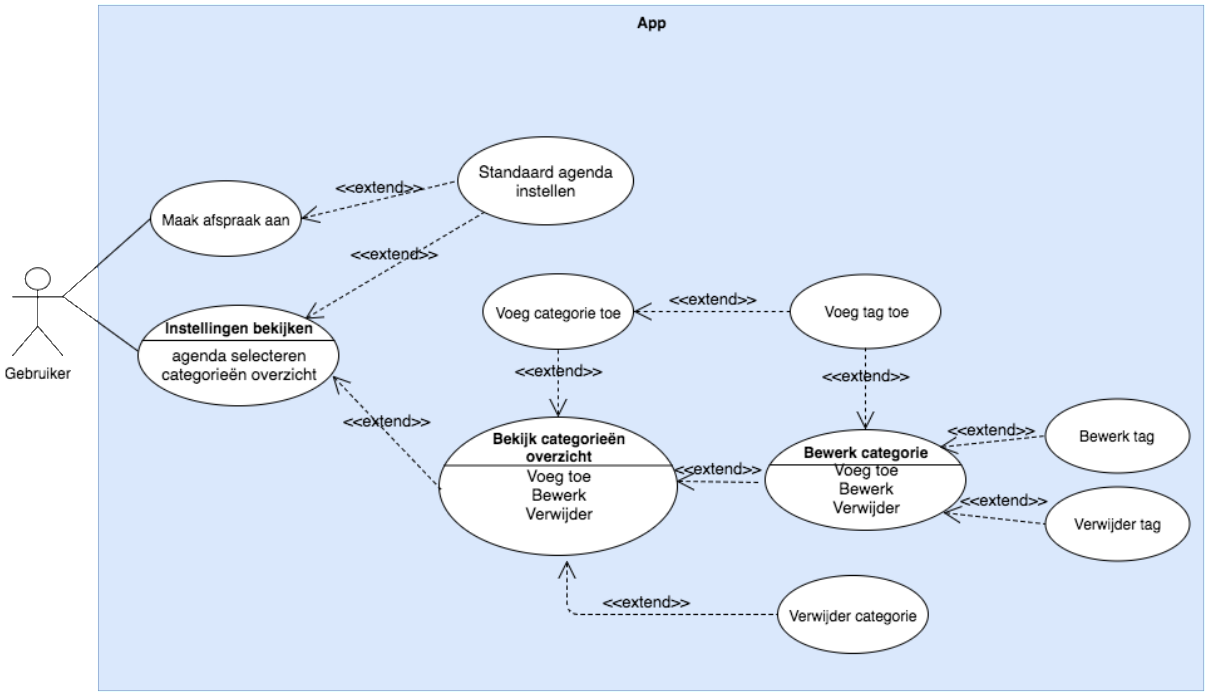
1.3. User stories

Requirement	ID	Omschrijving
FR1	US1	Als gebruiker wil ik een categorie en of tag selecteren, zodat ik een afspraak aan kan maken
FR1	US2	Als gebruiker wil ik een dag en tijdstip selecteren, zodat ik een afspraak aan kan maken
FR1	US3	Als gebruiker wil ik een kunnen agenda selecteren, zodat ik een afspraak aan kan maken
FR2	US4	Als gebruiker wil ik een categorie kunnen aanmaken, omdat de bestaande categorieën niet voldoen.
FR3	US5	Als gebruiker wil ik een categorie kunnen bewerken, omdat de naam van de categorie niet klopt.
FR4	US6	Als gebruiker wil ik een categorie kunnen verwijderen, omdat ik de categorie niet nodig heb.
FR5	US7	Als gebruiker wil ik een tag kunnen aanmaken, omdat de bestaande tags niet voldoen.
FR6	US8	Als gebruiker wil ik een tag kunnen bewerken, omdat de naam van de tag niet klopt.
FR7	US9	Als gebruiker wil ik een tag kunnen verwijderen, omdat ik de tag niet nodig heb.
FR8	US10	Als gebruiker wil ik een standaard agenda kunnen selecteren, omdat ik in de app altijd maar 1 agenda ga gebruiken.
FR9	US11	Als gebruiker wil ik een afspraak kunnen delen met andere, zodat ik iemand kan uitnodigen die geen contact is.
FR1	US12	Als gebruiker wil ik deelnemers kunnen selecteren, omdat ik een afspraak wil inplannen met meerdere personen.
FR10	US13	Als gebruiker wil ik dat de app mij suggesties toont voor afspraak tijden, omdat ik dan sneller een afspraak kan inplannen
FR1	US14	Als gebruiker wil ik dat de app mijn recent gebruikte deelnemers toont, zodat ik sneller mijn meest gebruikte deelnemers kan toevoegen
FR1	US15	Als gebruiker wil ik dat de app de afspraken per dag toont, zodat ik beter een keuze kan maken.
FR9	US16	Als gebruiker wil ik een afspraak aanmaken om alleen te delen, omdat de afspraak niet in mijn agenda hoeft te staan.
FR14	US17	Als gebruiker wil ik de optie hebben om bepaalde agenda's niet kiesbaar te maken, omdat ik die agenda's niet gebruik
FR12	US18	Als gebruiker wil ik de standaard tijden kunnen aanpassen, omdat de standaard keuzes niet kloppen met mijn schema.
FR11	US19	Als gebruiker wil ik de optie om van taal te kunnen veranderen, omdat ik de huidige taal niet ken.
FR13	US20	Als gebruiker wil ik de optie hebben om te kiezen tussen 24 uur of 12 uurs formaat voor tijden.

	US21	Als gebruiker wil ik een lijst kunnen zien van afspraken die gemaakt zijn met de app, omdat ik een bestaande afspraak wil delen.
	US22	Als gebruiker wil ik een locatie kunnen selecteren, omdat de afspraak op een bepaalde locatie plaatsvindt.

2. Use Cases

2.1. Diagram



2.2. Use Case lijst

In onderstaande lijst worden alle use cases beschreven.

ID	Beschrijving	Hoort bij requirement
1	Aanmaken van een afspraak.	FR1
2	Het instellingen scherm bekijken.	FR2, FR3, FR4, FR8
3	Overzicht van categorieën bekijken.	FR3, FR4
4	Het toevoegen van een categorie.	FR2, FR5
5	Het bewerken van een categorie.	FR4, FR5, FR6, FR7
6	Het verwijderen van een categorie.	FR4
7	Het toevoegen van een tag.	FR5
8	Het bewerken van een tag.	FR6
9	Het verwijderen van een tag	FR7
10	Het instellen van een standaard agenda	FR8

2.3. Tabellen

Hieronder staan de uitgeschreven use cases.

Use Case: Maak afspraak aan	
ID	1
Beschrijving	Aanmaken van een afspraak.
Primaire Actor	Gebruiker
Secundaire Actor	
Precondities	App is opgestart
Main flow	1. De use case start als de gebruiker een categorie selecteert. 2. Er wordt een scherm getoond met de tags die bij de geselecteerde categorie horen. 3. Optioneel: De gebruiker selecteert een tag en drukt op volgende. 4. Er wordt een scherm getoond met een maand kalender en tijden, inclusief een indicatie of een dag beschikbaar is of niet met data uit de eigen agenda. 5. De gebruiker selecteert een datum en tijd en drukt op volgende. 6. Er wordt een scherm getoond met agenda's. 7. De gebruiker selecteert één of meerdere agenda's en drukt op volgende. 8. Optioneel: De gebruiker selecteert een of meerdere deelnemers en drukt op opslaan. 9. De app voegt de afspraak toe aan de geselecteerde agenda/agenda's 10. Het overzicht scherm wordt getoond.
Postconditie	Afspraak is aangemaakt in de gekozen agenda/'s.
Alternatieve Flow	

Use Case: Instellingen bekijken	
ID	2
Beschrijving	Het instellingen scherm bekijken.
Primaire Actor	Gebruiker
Secundaire Actor	
Precondities	App is opgestart
Main flow	1. De use case start als de gebruiker op "instellingen" drukt 2. Het instellingen scherm wordt getoond. Extensionpoint: Bekijk categorieën overzicht
Postconditie	Het instellingen scherm is geopend
Alternatieve Flow	

Use Case: Bekijk categorieën overzicht	
ID	3
Beschrijving	Overzicht van categorieën bekijken.
Primaire Actor	Gebruiker
Secundaire Actor	
Preconditions	Het instellingen scherm wordt getoond
Main flow	1. De use case start als de gebruiker op "Categorieën" drukt 2. Een scherm met een lijst van de categorieën wordt getoond. Extensionpoint: Voeg toe Extensionpoint: Bewerk Extensionpoint: Verwijder
Postconditie	Scherm met categorieën wordt getoond.
Alternatieve Flow	

Use Case: Voeg categorie toe	
ID	4
Beschrijving	Het toevoegen van een categorie.
Primaire Actor	Gebruiker
Secundaire Actor	
Preconditions	Het categorieën overzicht wordt getoond
Main flow	1. De use case start als de gebruiker op "nieuwe categorie" drukt 2. Een scherm met invoervelden wordt getoond. 3. De gebruiker vult een naam in en selecteert een icon. 4. De gebruiker drukt op opslaan [1][2]. 5. De app voegt de categorie aan de database. Extensionpoint: Voeg tag toe
Postconditie	De categorie is toegevoegd.
Alternatieve Flow	[1] Er is geen naam ingevoerd, de app toont een waarschuwing en laat zien dat de naam ingevuld moet worden. Use case gaat verder bij 3. [2] Er is geen icon geselecteerd, de app zal een standaard icon gebruiken.

Use Case: Bewerk categorie	
ID	5
Beschrijving	Het bewerken van een categorie.
Primaire Actor	Gebruiker
Secundaire Actor	
Preconditie	Het categorieën overzicht wordt getoond
Main flow	1. De use case start als de gebruiker op "bewerken" drukt bij een categorie 2. Een scherm met ingevulde invoervelden wordt getoond. 3. De gebruiker veranderd de naam en of afbeelding en drukt op opslaan [1]. Extensionpoint: Voeg toe Extensionpoint: Bewerk Extensionpoint: Verwijder 4. De gebruiker drukt op opslaan [1].
Postconditie	De categorie is bewerkt.
Alternatieve Flow	[1] Er is geen categorie naam ingevuld, de app toont een waarschuwing en laat zien dat de naam ingevuld moet worden. Use case gaat verder bij 3.

Use Case: Verwijder categorie	
ID	6
Beschrijving	Het verwijderen van een categorie.
Primaire Actor	Gebruiker
Secundaire Actor	
Preconditie	Het categorieën overzicht wordt getoond
Main flow	1. De use case start als de gebruiker op “verwijderen” drukt bij een categorie. 2. Er wordt een waarschuwing getoond of de gebruiker het zeker weet. 3. De gebruiker drukt op “ja” [1]. 4. De app verwijdert de categorie en bijbehorende tags uit de database.
Postconditie	De categorie is verwijderd
Alternatieve Flow	[1] Er is op iets anders dan “ja” gedrukt. Use case stopt

Use Case: Voeg tag toe	
ID	7
Beschrijving	Het toevoegen van een tag.
Primaire Actor	Gebruiker
Secundaire Actor	
Precondities	Het categorie bewerken scherm of nieuwe categorie scherm wordt getoond
Main flow	1. De use case start als de gebruiker op "Nieuwe tag" drukt. 2. Er wordt een scherm met invoerveld getoond. 3. De gebruiker vult een naam in en drukt op opslaan [1]. 4. De app voegt de tag toe aan de categorie in de database.
Postconditie	Tag is toegevoegd.
Alternatieve Flow	[1] Er is geen naam ingevoerd, de app toont een waarschuwing en laat zien dat de naam ingevuld moet worden. Use case gaat verder bij 3.

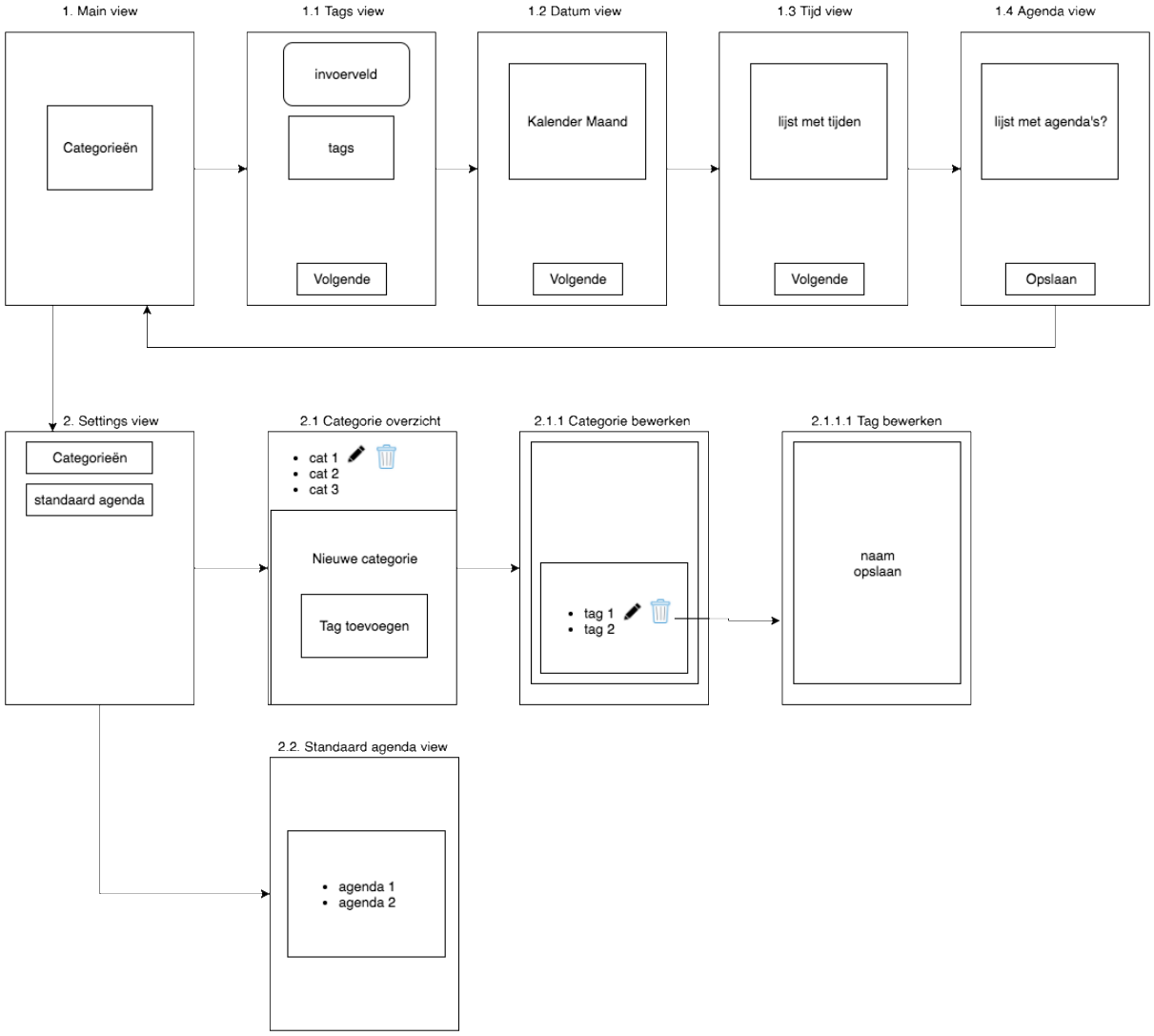
Use Case: Bewerk tag	
ID	8
Beschrijving	Het bewerken van een tag.
Primaire Actor	Gebruiker
Secundaire Actor	
Precondities	Het categorie bewerken scherm wordt getoond
Main flow	1. De use case start als de gebruiker op "bewerken" drukt bij een tag. 2. Er wordt een scherm getoond met een ingevulde invoerveld. 3. De gebruiker wijzigt de tag en drukt op opslaan [1]. 4. De app wijzigt de tag in de database.
Postconditie	De tag is bewerkt.
Alternatieve Flow	[1] Er is geen naam ingevoerd, de app toont een waarschuwing en laat zien dat het invoerveld ingevuld moet worden. Use case gaat verder bij 3.

Use Case: Verwijder tag	
ID	9
Beschrijving	Het verwijderen van een tag
Primaire Actor	Gebruiker
Secundaire Actor	
Precondities	Het categorie bewerken scherm wordt getoond
Main flow	1. De use case start als de gebruiker op "verwijderen" drukt bij een tag 2. Er wordt een waarschuwing getoond of de gebruiker het zeker weet. 3. De gebruiker drukt op “ja” [1]. 4. De app verwijdert de tag uit de database.
Postconditie	De tag is verwijderd
Alternatieve Flow	[1] Er is op iets anders dan “ja” gedrukt. Use case stopt

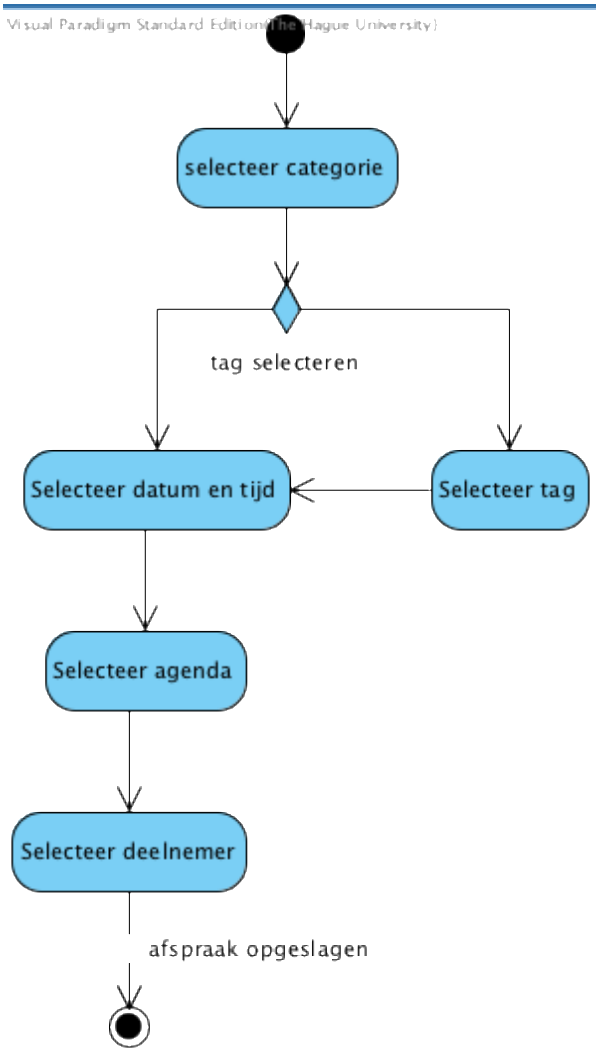
Use Case: Standaard agenda instellen	
ID	10
Beschrijving	Het instellen van een standaard agenda
Primaire Actor	Gebruiker
Secundaire Actor	
Precondities	Het instellingen scherm wordt getoond
Main flow	1. De use case start als de gebruiker op “Standaard agenda” drukt 2. Er wordt een lijst getoond met alle agenda’s 3. De gebruiker selecteert een of meerdere agenda’s.
Postconditie	Er is een standaard agenda ingesteld.
Alternatieve Flow	

3. Flow van schermen

Onderstaand concept beeld een gedeelte van de flow door de app uit



4. Activity diagram aanmaken afspraak



Bijlage 4) Functioneel & technisch ontwerp

FUNCTIONEEL & TECHNISCH ONTWERP

v1.4

Ravi Gajadin

Quick Events | Webbeat Products BV

Versiebeheer

Versie	Datum wijziging	Beschrijving van veranderingen
1.0	17-11-2016	Initiële versie
1.1	18-11-2016	App ontwerp toegevoegd
1.2	21-11-2016	Database toegevoegd
1.3	28-12-2016	Sprint 2 ontwerp aangepast
1.4	08-03-2017	Sprint 3 Ontwerp geupdate

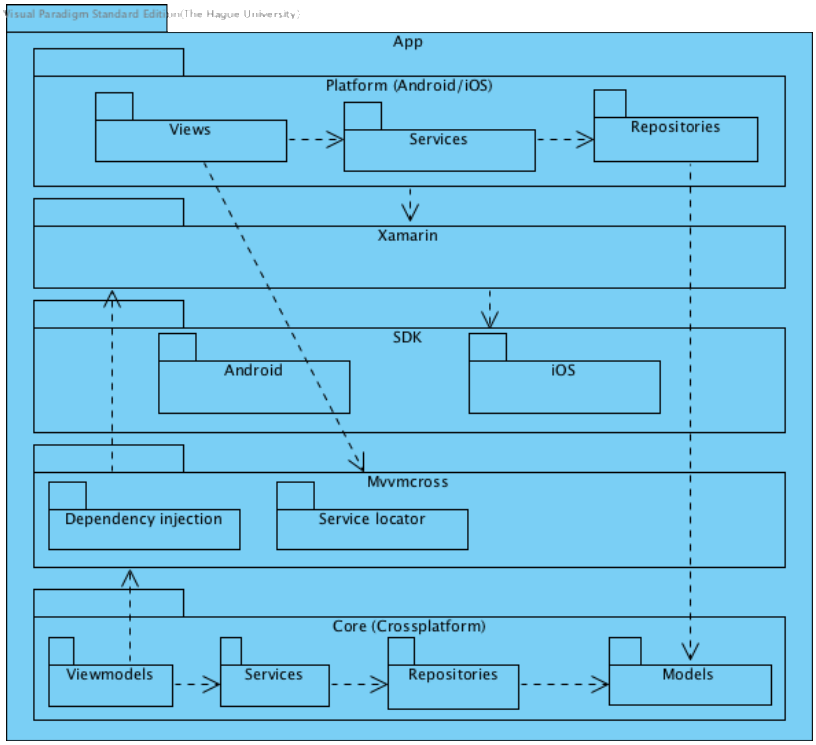
Inhoudsopgave

Versiebeheer 1
Inhoudsopgave 2
1. Architectuur app 3
1.1. Ontwerp app 3
1.2. Ontwerp Web API 5
1.3. Communicatie app en web api 5
2. Database intern 6
2.1. ERD 6
2.2. RRM 6
2.3. Datadictionary 7
3. Database Web Api 9
3.1. ERD 9
3.2. Datadictionary 9
3.3. Stored Procedures 10
4. WireFrames 12
4.1. Categorie en tag stap (US1) 12
4.2. Datum en tijd stap (US2) 13
4.3. Agenda selectie stap (US3) 13
4.4. Afspraak overzicht scherm 14

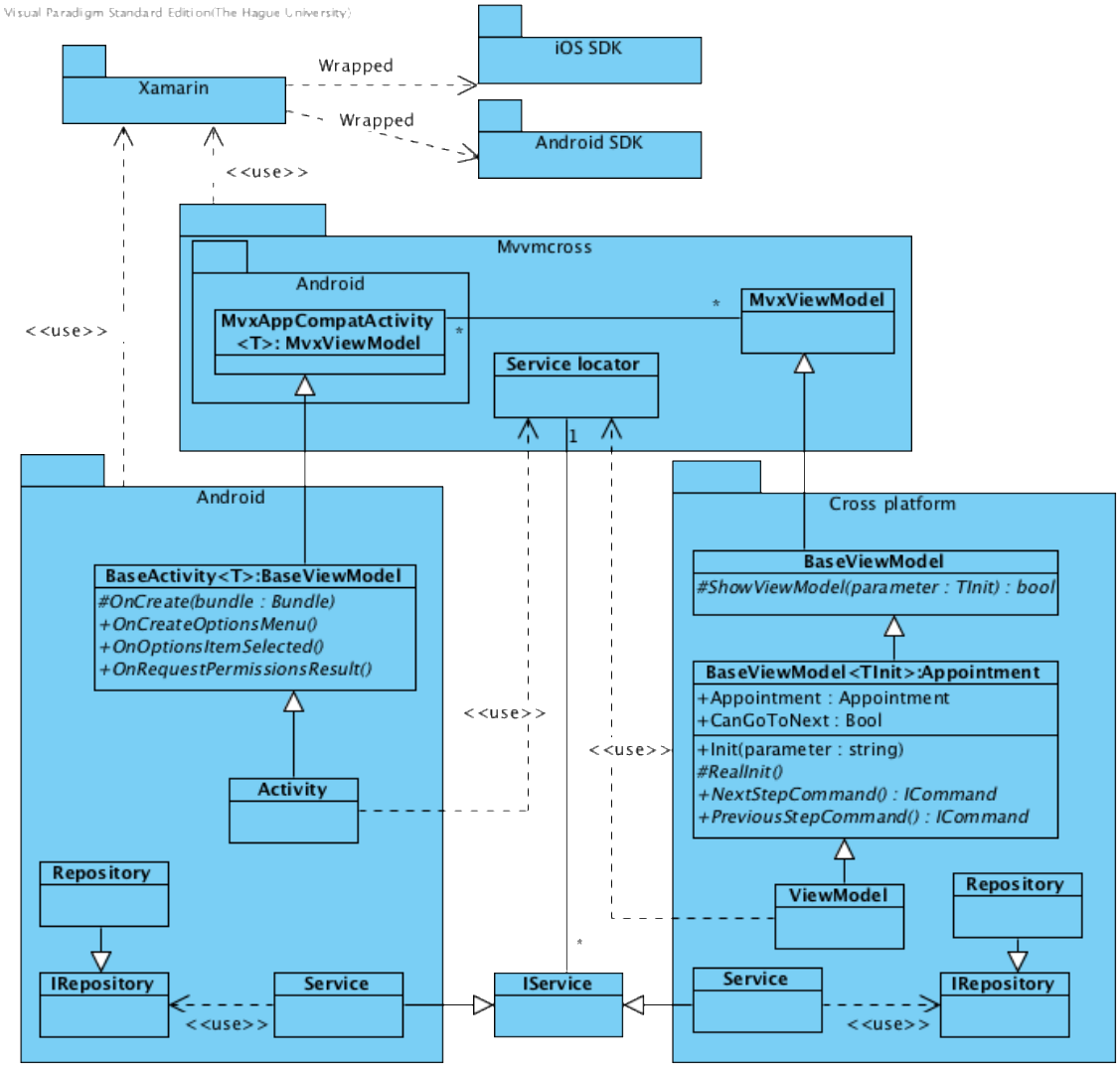
1. Architectuur app

In dit hoofdstuk wordt de architectuur van de app getoond.

1.1. Ontwerp app

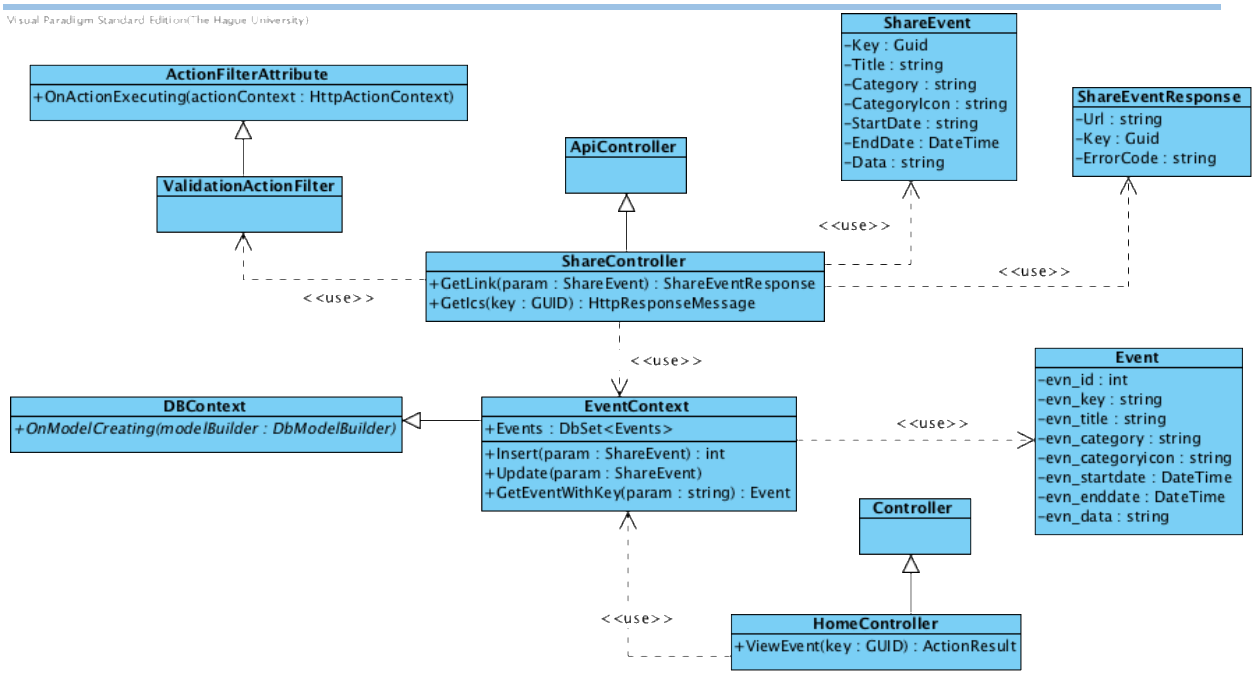


Bovenstaand is een simpele weergave van de app in de vorm van een lagen model met de belangrijkste lagen van de app. Dit diagram laat ook gelijk zien hoe de communicatie tussen deze lagen eruitziet.



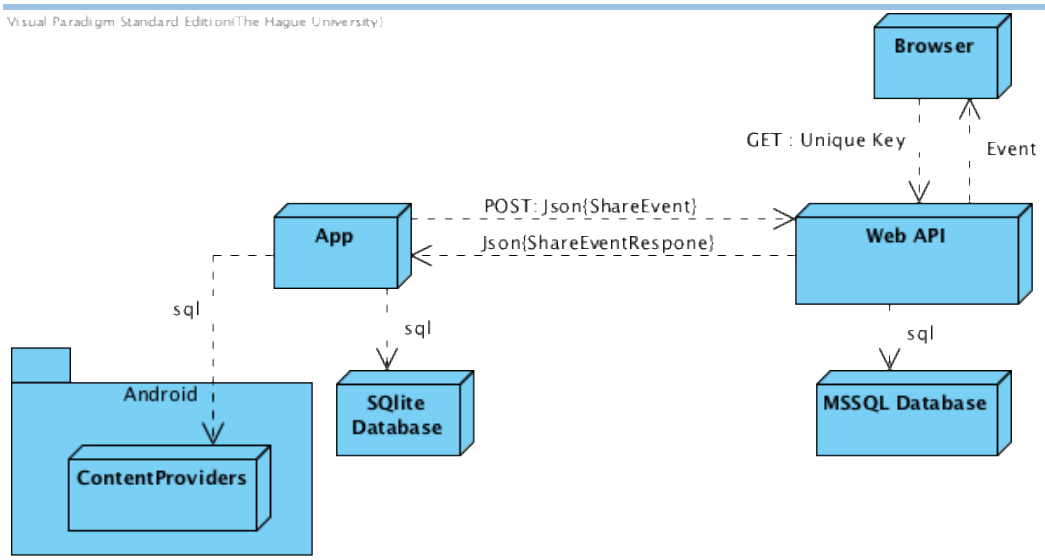
Bovenstaand diagram is een generiek ontwerp gebaseerd op het lagen model wat voor alle views in de app gebruikt zal worden. Dit diagram is specifiek voor Android gemaakt, maar door alle Android delen te vervangen met iOS equivalenten kan het ook voor iOS of een ander platform gebruikt worden.

2.2. Ontwerp Web API



*ShareEvent en ShareEventResponse zijn de objecten waarmee gecommuniceerd zal worden met de app.

2.3. Communicatie app en web api

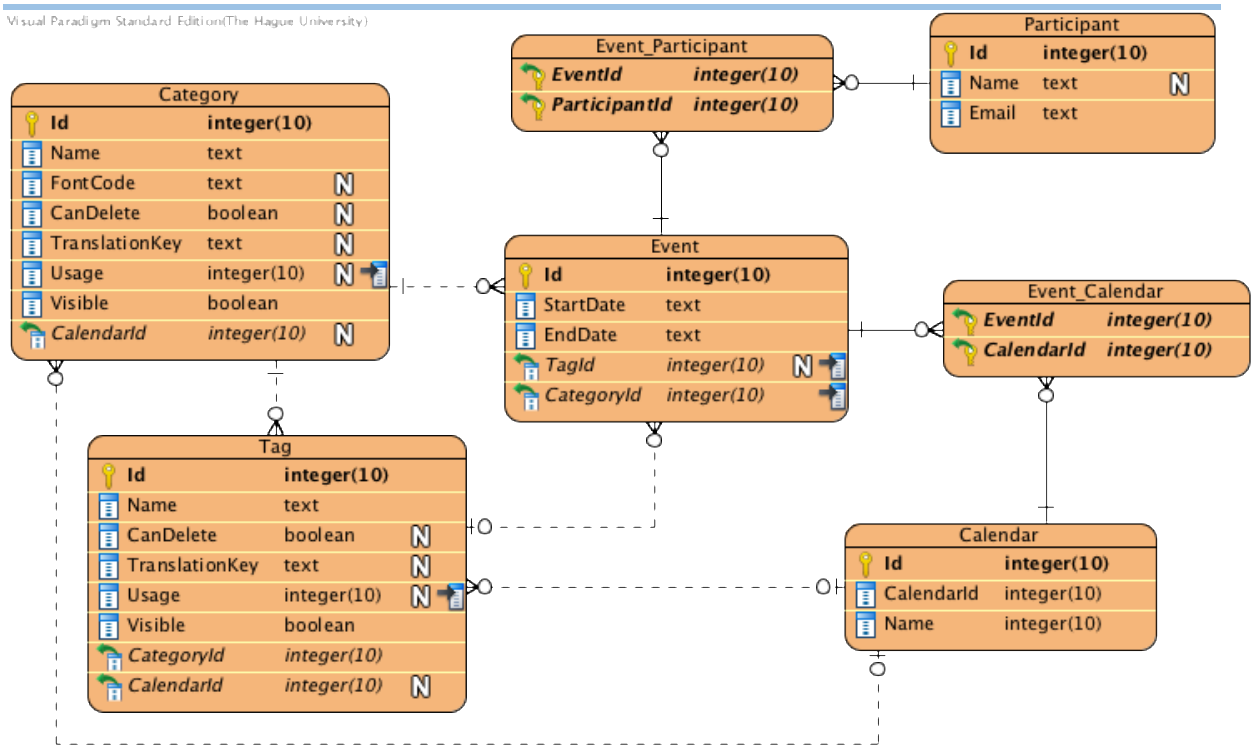


De communicatie met externe componenten ziet eruit zoals hierboven afgebeeld. Waarbij de app en web api communiceren door het converteren van de volgende 2 klassen naar json: ShareEvent, ShareEventResponse. Er kan ook via een willekeurige internet browser met de web API gecommuniceerd worden, Door het versturen van de unieke sleutel die een afspraak heeft om de afspraak te kunnen inzien.

3. Database intern

In dit hoofdstuk wordt uitgelegd hoe de sqlite database eruit komt te zien die gebruikt gaat worden in de app.

3.1. ERD



3.2. RRM

Tabellen:
Category (Id, Name, FontCode, CanDelete, TranslationKey, Usage, Visible)

Tag (Id, Name, CanDelete**,** TranslationKey, Usage, Visible, **CategoryId**)
CategoryId is een vreemde sleutel naar Category, NOT NULL

Event (Id, StartDate, EndDate, **TagId**, **CategoryId**)
TagId is een vreemde sleutel naar Tag, NULL
CategoryId is een vreemde sleutel naar Category, NOT NULL

Participant (Id, Name, Email)
Calendar (Id, CalendarId, Name)

Event_Participant (**EventId**, **ParticipantId**)
EventId is een vreemde sleutel naar Event, NOT NULL
ParticipantId is een vreemde sleutel naar Participant, NOT NULL

Event_Calendar (**EventId**, **CalendarId**)
EventId is een vreemde sleutel naar Event, NOT NULL
CalendarId is een vreemde sleutel naar Calendar, NOT NULL

Normalisatiecheck:
Alle tabellen staan in de 3de normaalvorm, aangezien er geen attributen afhankelijk zijn van een deel van de PK en ook niet afhankelijk zijn van andere attributen.

3.3. Datadictionary

Tabel: **Category**

Kolomnaam	**Datatype**	**Voorbeeld inhoud**	**Null?**	**Default**	**Lengte**	**Betekenis**
Id	INTEGER	0, 1, 2	N		10	Identificatie van de category **-> PK**
Name	TEXT	Overige, Business	N		80	Naam van de category
FontCode	TEXT	1f356	N		10	Fontcode van de category
CanDelete	boolean	1, 0	N	0	1	Of de categorie verwijderbaar is
TranslationKey	Text	cat1, cat2	Y		20	Vertalingscode voor de categorie
Usage	INTEGER	1, 2, 3	Y		10	Gebruik van de category
Visible	Boolean	1, 0	N	True	1	Of een category zichtbaar is in de app of niet

Tabel: T**ag**

Kolomnaam	**Datatype**	**Voorbeeld inhoud**	**Null?**	**Default**	**Lengte**	**Betekenis**
Id	INTEGER	0, 1, 2	N		10	Identificatie van de tag **-> PK**
Name	TEXT	Vergadering, afspraak	N		100	Naam van de tag
CanDelete	boolean	1, 0	N	0	1	Of de tag verwijderbaar is
TranslationKey	Text	Tag1, tag2	Y		20	Vertalingscode voor de tag
Usage	INTEGER	1, 2, 3	Y		10	Gebruik van de tag
Visible	Boolean	1, 0	N	True	1	Of een tag zichtbaar is in de app of niet
CategoryId	INTEGER	0, 1, 2	N		10	Identificatie van een category -> **FK**

Tabel: **Event**

Kolomnaam	Datatype	Voorbeeld inhoud	Null?	Default	Lengte	Betekenis
Id	INTEGER	0, 1, 2	N		10	Identificatie van de event **-> PK**
StartDate	TEXT	“25-12-2017 08:55 “	N		50	Start datum en tijd van een event
EndDate	TEXT	“25-12-2017 08:55”	N		50	Eind datum en tijd van een event
TagId	INTEGER	1, 2, 3	Y		10	Identificatie van een tag -> **FK**
CategoryId	INTEGER	1, 2, 3	N		10	Identificatie van een category -> **FK**

Tabel: **Participant**

Kolomnaam	Datatype	Voorbeeld inhoud	Null?	Default	Lengte	Betekenis
Id	INTEGER	0, 1, 2	N		10	Identificatie van de participant **-> PK**
Name	TEXT	test	Y		100	Naam van een participant
Email	TEXT	test@test.nl	N		100	Email adres van een participant

Tabel: **Calendar**

Kolomnaam	Datatype	Voorbeeld inhoud	Null?	Default	Lengte	Betekenis
Id	INTEGER	0, 1, 2	N		10	Identificatie van de calendar **-> PK**
CalendarId	INTEGER	659	N		10	Google id van een calendar
Name	TEXT	business	N		100	Naam van een calendar

Tabel: **Event_Participant**

Kolomnaam	Datatype	Voorbeeld inhoud	Null?	Default	Lengte	Betekenis
EventId	INTEGER	0, 1, 2	N		10	Identificatie van een event -> **FK**
ParticipantId	INTEGER	659	N		10	Identificatie van een participant -> **FK**

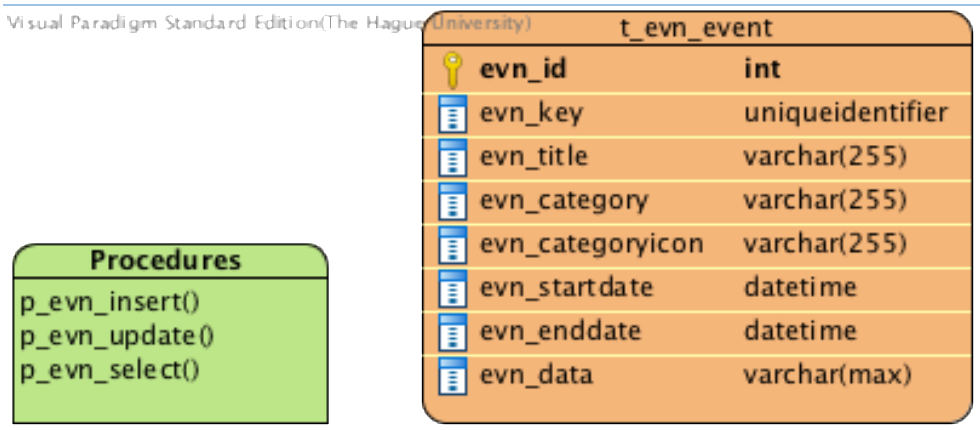
Tabel: **Event_Calendar**

Kolomnaam	Datatype	Voorbeeld inhoud	Null?	Default	Lengte	Betekenis
EventId	INTEGER	0, 1, 2	N		10	Identificatie van een event -> **FK**
CalendarId	INTEGER	659	N		10	Identificatie van een calendar -> **FK**

4. Database Web Api

In dit hoofdstuk wordt uitgelegd hoe de database eruit komt te zien die gebruikt gaat worden voor de web api.

4.1. ERD



4.2. Datadictionary

Tabel: **t_evn_event**

Kolomnaam	Datatype	Voorbeeld inhoud	Null?	Default	Lengte	Betekenis
evn_id	INTEGER	0, 1, 2	N		10	Identificatie van de event **-> PK**
evn_key	GUID	Overige, sport	N		max	Unieke sleutel van de afspraak
evn_title	varchar		N		255	Titel van de afspraak
evn_category	varchar	1, 0	N		255	Naam van de categorie
evn_categoryicon	varchar	1f356	N		255	Categorie fontcode
evn_startdate	datetime		N			Start datum afspraak
evn_enddate	datetime		N			eind datum afspraak
evn_data	varchar		N		max	ICS in json formaat

4.3. Stored Procedures

Voor de web api zal er ook gebruik gemaakt worden van een aantal Stored procedures namelijk:

p_evn_insert

Voor het toevoegen van nieuwe data deze stored procedure returned het id van de record.

```
CREATE PROCEDURE p_evn_insert
                @evn_title varchar(255),
                @evn_category varchar(255),
                @evn_categoryicon varchar(255),
                @evn_startdate datetime,
                @evn_enddate datetime,
                @evn_data varchar(max),
                @new_identity int=null OUTPUT
AS
BEGIN

INSERT INTO t_evn_event
                (
                        evn_key,
                        evn_title,
                        evn_category,
                        evn_categoryicon,
                        evn_startdate,
                        evn_enddate,
                        evn_data
                )
VALUES
                (
                        NEWID(),
                        @evn_title,
                        @evn_category,
                        @evn_categoryicon,
                        @evn_startdate,
                        @evn_enddate,
                        @evn_data
                )
set @new_identity = SCOPE_IDENTITY();
END
```

p_evn_select

Voor het selecteren van een event op basis van de key

```
CREATE PROCEDURE p_evn_select @evn_key varchar(255)
AS
BEGIN
SELECT *
FROM t_evn_event
WHERE evn_key = @evn_Key
END
```

p_evn_update

Voor het updaten van een bestaand event

```
CREATE PROCEDURE p_evn_update
                        @evn_key varchar(255),
                        @evn_title varchar(255),
                        @evn_category varchar(255),
                        @evn_categoryicon varchar(255),
                        @evn_startdate datetime,
                        @evn_enddate datetime,
                        @evn_data varchar(max)
 AS
   BEGIN
    UPDATE t_evn_event
    SET      evn_title = @evn_title,
                                    evn_category = @evn_category,
                                    evn_categoryicon = @evn_categoryicon,
                                    evn_startdate = @evn_startdate,
                                    evn_enddate = @evn_enddate,
                                    evn_data = @evn_data
    WHERE evn_key = @evn_key
END
```

5. WireFrames

5.1. Categorie en tag stap (US1)

Dit is de eerste stap die de gebruiker zal zien als hij of zij de app opstart. Waarin er een categorie en tag geselecteerd moet worden.

De categorieën zullen weergegeven worden op de volgende wijze:

- in rijen van 3 of 4 afhankelijk van de grootte van het scherm
- Elke categorie heeft een icoon
- Na X rijen categorieën, wordt het categorieën gedeelte schuifbaar.
- Een lijn na de laatste rij met categorieën
- De knop om een nieuwe categorie aan te maken wordt weergegeven aan het eind van een rij als dat past en anders gecentreerd in een nieuwe rij.

De tags zullen weergegeven worden op de volgende manier:

- X tags naast elkaar afhankelijk van de grootte van het scherm en de grootte van de tags.
- In rijen onder elkaar.
- Na X rijen tags, wordt het tags gedeelde schuifbaar
- De knop om een nieuwe tag aan te maken wordt altijd gecentreerd in een nieuwe rij weergegeven.

Onderaan het scherm zullen de navigatie knoppen getoond worden.

5.2. Datum en tijd stap (US2)

Dit is de 2de stap die de gebruiker te zien krijgt waarin er een dag en tijd geselecteerd moet worden.

Dag selectie wordt op de volgende manier weergegeven:

- In de vorm van een maand kalender
- Er kan heen en weer genavigeerd worden in de kalender

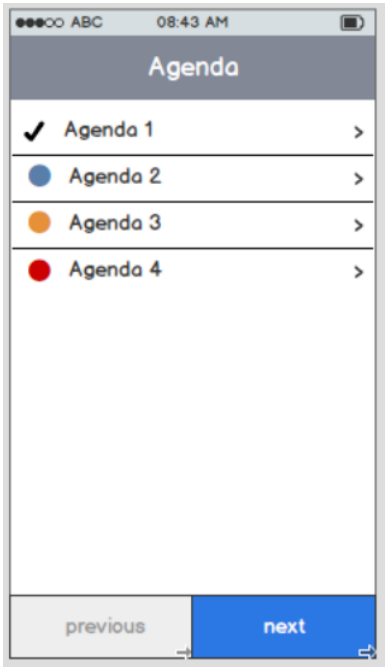
De tijden worden op de volgende manier weergegeven:

- Standaard drie tijden voor de drie dag delen: Ochtend, Middag en Avond.
- Er is ook een 4de optie beschikbaar om zelf een tijd in te voeren.

Onderaan het scherm zullen de navigatie knoppen getoond worden.

5.3. Agenda selectie stap (US3)

Dit is de 3de stap die de gebruiker te zien krijgt waarin er minimaal één agenda geselecteerd moet worden waaraan de afspraak toegevoegd wordt.



De agenda's worden op de volgenden manier weergegeven:

- Elke agenda waar de gebruiker dingen aan kan bewerken, wordt getoond met de naam ervan en een cirkel ervoor met de agenda kleur.
- Geselecteerde agenda's krijgen een vink achter hun naam.

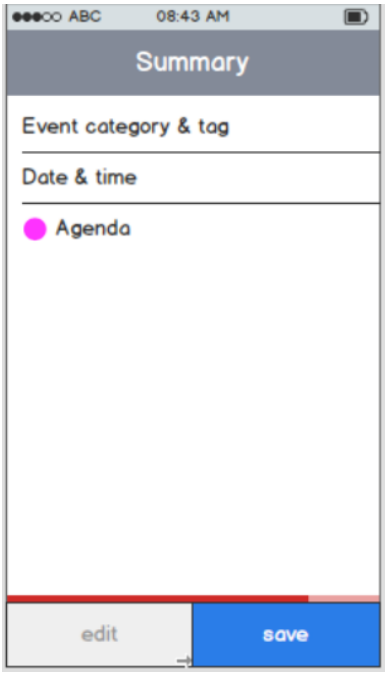
Onderaan het scherm zullen de navigatie knoppen getoond worden.

5.4. Afspraak overzicht scherm

Dit is de laatste stap in de applicatie waar een overzicht van de afspraak getoond wordt.
De volgende dingen worden getoond:

- Naam van de afspraak
- Datum en tijd van de afspraak
- En de geselecteerde agenda's

Onderaan het scherm zullen de navigatie knoppen getoond worden.



Bijlage 5) Testplan en rapportage

TEST PLAN EN RAPPORTAGE

V3.0

Ravi Gajadin

Quick Events | Webbeat Products BV

Inhoudsopgave

Inhoudsopgave 1
1. Introductie 2
1.1. Scope 2
2. Test plan en strategie 2
2.1. Unit testen 2
2.1.1. Definitie 2
2.1.2. Start criteria 2
2.1.3. Stop criteria 2
2.1.4. Defect afhandeling 2
2.2. Systeem testen 2
2.2.1. Definitie 2
2.2.2. Test procedure 2
2.3. Regressie testen 3
2.4. Pass/ Fail condities 3
3. Schema voor het testen 3
3.1. Risico's en Aannames 3
3.2. Risico's 3
3.3. Aannames 3
4. Start en Stop criteria 3
4.1. Start criteria 3
4.2. Stop criteria 3
5. Deliverables 3
6. Test cases 4
6.1. Sprint 1 4
6.2. Sprint 2 6
6.3. Sprint 3 8
7. Test resultaten 11
7.1. Sprint 1 11
7.2. Sprint 2 11
7.3. Sprint 3 11

1. Introductie

Dit document legt uit hoe de app getest gaat worden en welke test methodes er gebruikt gaan worden.

2. Test plan en strategie

2.1. Unit testen

2.1.1. Definitie

Het doel van Unit testen is om te verifiëren dat een stuk broncode goed werkt. Unit testen worden ontworpen om een klasse, component of module geïsoleerd te testen.

2.1.2. Start criteria

Unit testen kan gestart worden als aan de volgende voorwaarden is voldaan.

1. Test cases zijn gemaakt.
2. Build is voltooid zonder errors.

2.1.3. Stop criteria

Er kan gestopt worden met Unit testen als aan de volgende voorwaarden is voldaan:

1. Alle geplande test cases zijn uitgevoerd.
2. Alle defecten gevonden tijdens het testen zijn genoteerd.
3. Alle defecten zijn opgelost.

2.1.4. Defect afhandeling

Defecten die gevonden worden tijdens het Unit testen zullen direct opgelost worden.

2.2. Systeem testen

2.2.1. Definitie

Systeem testen is het testen van de complete app in tegenstelling tot Unit testen waar alleen delen van de app getest worden. Het doel is het valideren van de gehele applicatie.

2.2.2. Test procedure

Test stappen:

- Maken van test cases
- Rapporteren van defecten

Die testen die uitgevoerd gaan worden:

1. Functionaliteits testen – Gedocumenteerde features zijn geïmplementeerd in de app en werken zoals verwacht. Bron voor deze features is het requirements document.
2. UI testen – Het testen van de User Interface

2.3. Regressie testen

Dit is een extra stap en wordt gedaan voordat de systeem testen worden uitgevoerd. Regressie testen is het opnieuw testen van de app nadat er bugs opgelost zijn of er nieuwe functionaliteit is toegevoegd.

2.4. Pass/ Fail condities

Er wordt verwacht dat de app 95% van de testen moet halen en er geen kritische bugs zijn.

3. Schema voor het testen

Het testen zal in de laatste 1.5 weken van elke sprint plaatsvinden.

3.1. Risico's en Aannames

3.2. Risico's

De volgende dingen kunnen een impact hebben op de test cyclus:

- Beschikbaarheid van apparaten
- Nieuwe features of veranderingen die niet op tijd zijn gecommuniceerd.
- Een vertraging in het ontwikkelproces inclusief het oplossen van fouten.

3.3. Aannames

- Apparaten, Emulators en andere support tools zullen volledig functioneel zijn voordat het project begint.
- Alle bugs die gevonden worden zullen opgelost worden.

4. Start en Stop criteria

4.1. Start criteria

- De test cases zijn gemaakt
- Unit testen is succesvol voltooid.

4.2. Stop criteria

- Alle test cases zijn uitgevoerd en tenminste 95% van de testen zijn succesvol voltooid. De overige 5% bevat testen die geen impact hebben op kritische functionaliteit.
- Er zijn geen kritische fouten over of andere bugs waardoor de app niet meer kan werken.

5. Deliverables

- Test plan
- Test cases – Document met een beschrijving en verwachte resultaten voor elke test
- Test resultaten – De pass/fail status voor elke test case

6.1. Test cases

In dit hoofdstuk zullen de test cases geïdentificeerd worden.

6.1. Sprint 1

De user stories die bij de test cases horen

Requirement	ID	Omschrijving
FR1	US1	Als gebruiker wil ik een categorie en tag selecteren, zodat ik een afspraak aan kan maken
FR1	US2	Als gebruiker wil ik een dag en tijdstip selecteren, zodat ik een afspraak aan kan maken
FR1	US3	Als gebruiker wil ik een agenda selecteren, zodat ik een afspraak aan kan maken
FR2	US4	Als gebruiker wil ik een categorie kunnen aanmaken, omdat de bestaande categorieën niet voldoen.
FR5	US7	Als gebruiker wil ik een tag kunnen aanmaken, omdat de bestaande tags niet voldoen.

De volgende gegevens zullen gebruikt worden voor de testen.
Afspraak:

- Categorie: Business
- Tag: Meeting
- Datum: 25-12-2016
- Tijd: 13:00
- Agenda: Test

Nieuwe categorie:

- Naam: Test Categorie
- Icoon: 1ste icoon (kalender)

Nieuwe tag

- Naam: Test Tag

ID	User story	Stap	Pre conditie	Omschrijving	Verwacht Resultaat
TC1	US1	1	Categorieën worden getoond	Categorie selecteren	Geselecteerde categorie wordt groen
		2	Tags worden getoond	Tag selecteren	Geselecteerde Tag wordt groen
		3	Volgende knop wordt klikbaar	Op Volgende knop drukken	Datum en tijdstap wordt geladen, categorie en tag objecten worden meegestuurd
TC2	US2	1		Datum selecteren	Geselecteerde datum wordt gemarkeerd

		2	Tijden worden zichtbaar	Tijd wordt geselecteerd	Geselecteerde tijd wordt groen
		3	Volgende knop wordt klikbaar	Op Volgende knop drukken	Agenda selectie stap wordt getoond, datum en tijd worden meegestuurd inclusief objecten van TC1
TC3	US3	1	Agenda's worden getoond	Selecteer een agenda	Geselecteerde agenda wordt gemarkeerd
		2	Volgende knop wordt klikbaar	Op Volgende knop drukken	Overzicht stap wordt geladen, agenda/'s worden meegestuurd inclusief objecten van TC2
		3		Druk op opslaan	Afspraak wordt opgeslagen in kalender van gebruiker
TC4		1	Overzicht stap wordt getoond	Druk op vorige	Agenda stap wordt getoond. Geselecteerde agenda's worden gemarkeerd
		2	Agenda stap wordt getoond	Druk op vorige	Datum en tijd stap wordt getoond. Gekozen Datum en tijd zijn gemarkeerd
		3	Datum en tijd stap wordt getoond	Druk op vorige	Categorie en tag stap wordt getoond. Gekozen categorie en tag zijn gemarkeerd.
TC5	US4	1	Categorie en tag scherm wordt getoond	Druk op nieuwe categorie	Nieuw categorie scherm wordt getoond
		2	Categorie aanmaken scherm wordt getoond	Vult een naam in	-
		3	Naam is ingevuld	Selecteert een icoon	Icoon wordt gemarkeerd en opslaan knop wordt klikbaar
		4	Opslaan knop is klikbaar	Druk op opslaan knop	Categorie is opgeslagen in de database

TC6	US7	1	Categorie en tag scherm wordt getoond	Selecteer een categorie	Tags worden getoond inclusief nieuwe tag knop
		2		Druk op nieuwe tag	Nieuw tag scherm wordt getoond
		3	Nieuw tag scherm wordt getoond	Vul naam in	Opslaan knop wordt klikbaar
		4	Naam is ingevuldt	Druk op opslaan knop	Tag is opgeslagen in de database

6.2. Sprint 2

De user stories die bij de test cases horen zijn.

Requirement	ID	Omschrijving
FR9	US11	Als gebruiker wil ik een afspraak kunnen delen met andere, zodat ik iemand kan uitnodigen die geen contact is.
FR1	US12	Als gebruiker wil ik deelnemers kunnen selecteren, omdat ik een afspraak wil inplannen met meerdere personen.
	US13	Als gebruiker wil ik dat de app mij suggesties toont voor afspraak tijden, omdat ik dan sneller een afspraak kan inplannen
	US14	Als gebruiker wil ik dat de app mijn recent gebruikte deelnemers toont, zodat ik sneller mijn meest gebruikte deelnemers kan toevoegen
	US15	Als gebruiker wil ik dat de app de afspraken per dag toont, zodat ik beter een keuze kan maken.

De volgende gegevens zullen gebruikt worden voor de testen.

Afspraak:

- Categorie: Business
- Tag: Meeting
- Datum: 25-12-2016
- Tijd: 13:26, 14:26, 19:35, 13:26, 14:26, 19:35, 13:26
- Agenda: Test
- Deelnemer: Test

Nieuwe categorie:

- Naam: Test Categorie
- Icoon: 1ste icoon (kalender)

Nieuwe tag

- Naam: Test Tag

ID	User story	Stap	Pre conditie	Omschrijving	Verwacht Resultaat
TC7	US11				
TC8	US12	1	Deelnemers stap wordt getoond, en app bevat data van vorige stappen	Selecteer deelnemer	Deelnemer wordt gemarkeerd
		2	Deelnemer is geselecteerd	Druk op opslaan	Overzicht scherm wordt getoond, afspraak is opgeslagen in de agenda van de gebruiker
TC9	US13	1	Database bevat geen afspraken voor categorie en tag	7 afspraken invoeren in database	Database bevat 7 afspraken
		2	Database bevat afspraken van stap 1	Gaat naar datum en tijd stap en selecteert datum	13:26 wordt getoond als tijd optie
TC10	US14	1		Selecteer een agenda	Geselecteerde agenda wordt gemarkeerd
		2	Volgende knop wordt klikbaar	Op Volgende knop drukken	Overzicht stap wordt geladen, agenda/'s worden meegestuurd inclusief objecten van TC2
		3		Druk op opslaan	Afspraak wordt opgeslagen in kalender van gebruiker
TC11	US15	1	Datum en tijd stap wordt getoond	Selecteer een dag met afspraken	Lijst met afspraken wordt getoond.

6.3. Sprint 3

De user stories die bij de test cases horen zijn.

Requirement	ID	Omschrijving
FR1	US1	Als gebruiker wil ik een categorie en of tag selecteren, zodat ik een afspraak aan kan maken
FR1	US3	Als gebruiker wil ik een agenda selecteren, zodat ik een afspraak aan kan maken
FR3	US5	Als gebruiker wil ik een categorie kunnen bewerken, omdat de naam van de categorie niet klopt.
FR4	US6	Als gebruiker wil ik een categorie kunnen verwijderen, omdat ik de categorie niet nodig heb.
FR6	US8	Als gebruiker wil ik een tag kunnen bewerken, omdat de naam van de tag niet klopt.
FR7	US9	Als gebruiker wil ik een tag kunnen verwijderen, omdat ik de tag niet nodig heb.
	US16	Als gebruiker wil ik een afspraak aanmaken om alleen te delen, omdat de afspraak niet in mijn agenda hoeft te staan.
	US17	Als gebruiker wil ik de optie hebben om bepaalde agenda's niet kiesbaar te maken, omdat ik die agenda's niet gebruik
	US18	Als gebruiker wil ik de standaard tijden kunnen aanpassen, omdat de standaard keuzes niet kloppen met mijn schema.
	US19	Als gebruiker wil ik de optie om van taal te kunnen veranderen, omdat ik de huidige taal niet ken.
	US20	Als gebruiker wil ik de optie hebben om te kiezen tussen 24 uur of 12 uurs formaat voor tijden.

De volgende gegevens zullen gebruikt worden voor de testen.

Afspraak:

- Categorie: Business
- Tag: Meeting
- Datum: 25-12-2016
- Tijd: 13:26, 14:26, 19:35, 13:26, 14:26, 19:35, 13:26
- Agenda: Test
- Deelnemer: Test

Nieuwe categorie:

- Naam: Test Categorie
- Icoon: 1ste icoon (kalender)

Nieuwe tag

- Naam: Test Tag

Tijden

- 09:25
- 14:32
- 19:59

ID	User story	Stap	Pre conditie	Omschrijving	Verwacht Resultaat
TC12	US1	1	Categorieën worden getoond	Categorie selecteren	Geselecteerde categorie wordt groen en volgende knop wordt klikbaar
		2	Categorie is geselecteerd	Op Volgende knop drukken	Datum en tijdstap wordt geladen, categorie object wordt meegestuurd
TC13	US3	1	Agenda selectie scherm wordt getoond	Druk op volgende	Deelnemer schem wordt getoond, afspraak wordt meegestuurd zonder agenda's
TC14	US5	1	Categorie en tags instellingen scherm wordt getoond	Druk op een categorie	Categorie bewerk scherm wordt getoond
		2	Categorie bewerk scherm wordt getoond	Bewerk de naam	
		3		Selecteer een ander icoon en druk op opslaan	Categorie wordt opgeslagen en categorie en tags instellingen scherm wordt getoond
TC15	US6	1	Categorie en tags instellingen scherm wordt getoond	Druk op een categorie	Categorie bewerk scherm wordt getoond
		2	Categorie bewerk scherm wordt getoond	Druk op verwijderen	Categorie wordt op invisible gezet in de database en categorie en tags instellingen scherm wordt getoond
TC16	US8	1	Categorie en tags instellingen scherm wordt getoond	Druk op een tag	tag bewerk scherm wordt getoond
		2	tag bewerk scherm wordt getoond	Bewerk de naam	
		3		Druk op opslaan	tag wordt opgeslagen en categorie en tags

					instellingen scherm wordt getoond
TC17	US9	1	Categorie en tags instellingen scherm wordt getoond	Druk op een tag	tag bewerk scherm wordt getoond
		2	tag bewerk scherm wordt getoond	Druk op verwijderen	tag wordt op invisible gezet in de database en categorie en tags instellingen scherm wordt getoond
TC18	US16	1	agenda selectie scherm wordt getoond	Selecteer geen agenda's en druk op volgende	Deelnemer selectie scherm wordt getoond
		2	Deelnemer selectie scherm wordt getoond	Druk op opslaan	Afspraak wordt niet opgeslagen in de agenda, maar is alleen deelbaar
TC19	US17	1	Agenda instellingen scherm wordt getoond	Deselecteer een agenda	Agenda wordt verwijderd uit instellingen bestand
		2		Agenda selectie scherm wordt getoond	Agenda van stap 1 wordt niet getoond
TC20	US18	1	Tijd instellingen scherm wordt getoond	Tijden worden ingevoerd	Nieuwe tijden zijn opgeslagen in instellingen
		2	Datum en tijd stap wordt getoond	Selecteer datum	Tijden van stap 1 worden getoond
TC21	US19	1	Algemeen instellingen scherm wordt getoond	Selecteer een andere taal dan de huidige	Taal wordt opgeslagen
		2		Controleer taal door hele applicatie	App wordt getoond in de taal van stap 1
TC22	US20	1	Tijd instellingen scherm wordt getoond	Selecteer de optie 24 of 12 uur	Keuze wordt opgeslagen
		2			

7. Test resultaten

7.1. Sprint 1

Geen testen uitgevoerd.

7.2. Sprint 2

Hieronder zullen de test resultaten voor de test cases getoond worden.

Test case	Soort test	Resultaat	Opmerking
TC1	Unit test + UI test	Test geslaagd	
TC2	UI test	Test geslaagd	
TC3	Unit test + UI test	Test geslaagd	
TC4	UI test	Test geslaagd	
TC5	Unit test + UI test	Test geslaagd	
TC6	Unit test + UI test	Test geslaagd	
TC7	Unit test + UI test	Test geslaagd	
TC8	Unit test + UI test	Test geslaagd	
TC9	Unit test + UI test	Test geslaagd	
TC10	UI test	Test geslaagd	
TC11	Unit test + UI test + Systeem test	Test geslaagd	Systeem testen kunnen falen doordat het soms even kan duren voor een response van de web api, kan opgelost worden door het deployen van de api

7.3. Sprint 3

Hieronder zullen de test resultaten voor de test cases getoond worden.

Test case	Soort test	Resultaat	Opmerking
TC1	Regressie test	gefaald	time out door het niet kunnen vinden van UI component
TC2	Regressie test	Test geslaagd	
TC3	Regressie test	Test geslaagd	
TC4	Regressie test	Test geslaagd	
TC5	Regressie test	gefaald	Time out door het niet kunnen vinden van nieuwe categorie knop
TC6	Regressie test	gefaald	Time out door het niet kunnen vinden van nieuwe tag knop
TC7	Regressie test	Test geslaagd	
TC8	Regressie test	Test geslaagd	
TC9	Regressie test	Test geslaagd	
TC10	Regressie test	Test geslaagd	
TC11	Regressie test	Test geslaagd	

TC12	UI test	Test geslaagd	
TC13	UI test	Test geslaagd	
TC14	UI test	Test geslaagd	
TC15	UI test	Test geslaagd	
TC16	UI test	Test geslaagd	
TC17	UI test	Test geslaagd	
TC18	UI test	Test geslaagd	
TC19	UI test	Test geslaagd	
TC20	UI test	Test geslaagd	
TC21	handmatig	Test geslaagd	Vertalingen worden overal getoond, sommige vertalingen kloppen niet helemaal, maar zijn acceptabel.
TC22	UI test	Test geslaagd	

User acceptance testen

Voor de user acceptance testen is de app doorgelopen door de designers en leon de brabander, De feedback die hieruit kwam ging vooral over de UI die op een aantal plekken in de app beter kon en ontbrekende vertalingen. Deze fouten zijn voor het eind van sprint 3 opgelost en geïmplementeerd voor de release.

Conclusie

Ondanks het falen van 3 regressie testen in Sprint 3 is er na het oplossen van de fouten en feedback besloten om de app te releasen, omdat alle functionaliteit voldoende geverifieerd is.