Professionele Bachelor Toegepaste Informatica

Een mobiel portaal tussen cliënt en behandelaar met MyClient

David Randisi

Promotoren:

Michel Claassen
Carine Derkoningen

Yunify
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Een mobiel portaal tussen cliënt en behandelaar met MyClient

David Randisi

Promotoren:

Michel Claassen
Carine Derkoningen

Yunify
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Op een event van de PXL heb ik kennis kunnen maken met het Nederlands bedrijf Yunify. Vervolgens heb ik de mogelijkheid gekregen om mijn stage te doen in dit bedrijf waarvoor ik zeer dankbaar ben. Yunify heeft er voor gezorgd dat ik mijn stage heb kunnen uitvoeren in een zeer aangename werkomgeving. Verder ben ik ook dankbaar voor de vrijheid die ik heb ervaren tijdens het uitwerken van de stageopdracht.

Mijn collega's tijdens de stageperiode hebben me ook goed ontvangen in hun team. Voornamelijk Roy Bos en Michel Claassen waren betrokken bij mijn opdracht en hebben er voor gezorgd dat ik de opdracht met zo veel mogelijk ondersteuning heb kunnen uitvoeren.

Ook zou ik graag mijn hogeschoolpromotor, Carine Derkoningen, willen bedanken voor de hulp die ze mij aangeboden heeft gedurende de volledige stageperiode. Zij heeft me wekelijks de nodige feedback gegeven rond de stageportfolio en het eindwerk. Verder was ze steeds weer beschikbaar voor al mijn vragen via mail of via de contactmomenten.

Tenslotte wil ik graag mijn familie en vrienden bedanken om mij te ondersteunen tijdens de stageperiode. Ondanks dat ze geen kennis van IT hebben waren ze wel steeds bereid om te luisteren naar mijn verhalen na een lange werkdag.

Abstract

Yunify is een bedrijf gevestigd in Nederland met een sterke focus op *healthcare*. Eén van de e-health-applicaties in hun assortiment is MyClient. Dit is een online portaal tussen behandelaar en cliënt in de vorm van een webapplicatie. Ondanks dat de webapplicatie een *responsive* lay-out heeft voor het gebruik op mobiele toestellen is toch de beslissing genomen om de functionaliteiten van MyClient ook aan te bieden in de vorm van een mobiele *native* applicatie.

Na de beslissing om een *native* applicatie te gaan bouwen kan het ontwikkelproces nog niet van start gaan. Een *native* applicatie kan namelijk op verschillende manieren gebouwd worden. De juiste technologie kiezen voor de vereisten van een project is niet altijd evident en brengt vaak verwarring met zich mee. Zo bestaan er tal van frameworks die het mogelijk maken om platformonafhankelijke mobiele applicaties te bouwen met één enkele *codebase*.

Voor de stageopdracht is één van de platformonafhankelijke technologieën, Flutter, gebruikt om de functionaliteiten van MyClient te ontwikkelen. In het onderzoek wordt deze keuze kritisch bekeken. Naast Flutter wordt er ook aandacht besteed aan het Ionic-framework. Ionic zorgt uiteindelijk ook voor platformonafhankelijke applicaties maar is een voorbeeld van de hybride technologie. Hier ontstaat er vaak al verwarring omdat het verschil tussen de diverse platformonafhankelijke technologieën niet altijd duidelijk is.

Ten eerste wordt er onderzoek gedaan naar de effectieve werking van beide frameworks. Hier wordt er beschreven hoe Flutter een uniek systeem aanbiedt voor het tekenen van de componenten die op het scherm getoond worden. Voor Ionic wordt er besproken hoe dit framework gebruik maakt van een *web view* om mobiele applicaties weer te geven.

Vervolgens wordt er een literatuurstudie opgesteld door enkele bronnen te raadplegen die ook Flutter met Ionic vergelijken. Hier wordt het al snel duidelijk dat er enkele meningsverschillen ontstaan tussen de gekozen artikels. Uit de tegenstrijdigheden en gelijkenissen worden er een conclusies opgesteld.

Tenslotte worden de bevindingen en conclusies uit de literatuurstudie op de proef gesteld door middel van een *proof of concept*. Hierbij wordt een applicatie gebouwd die enkele simpele functionaliteiten bevat. Dezelfde applicatie wordt gebouwd met beide frameworks waardoor er ook een praktische vergelijking opgesteld wordt. Het PoC wordt ook gebruikt om meetresultaten op te stellen omtrent de performantie van beide applicaties. De praktische vergelijking wordt samen met de meetresultaten gebruikt om overeen te stemmen met de bevindingen van de literatuurstudie.

Inhoud

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

AOT	Ahead-of-Time
API	Application Programming Interface
APK	Android Package
CSS	Cascading Style Sheets
CB	Content Block
FAQ	Frequently Asked Questions
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IPA	iOS App Store Package
JIT	Just-in-Time
JSON	JavaScript Object Notation
JWT	JSON Web Token
MB	Megabyte
NPM	Node Package Manager
OEM	Original Equipment Manufacturer
PoC	Proof of Concept
REST	Representational State Transfer
RxJS	Reactive Extensions Library for JavaScript
SDK	Software Development Kit
SSMS	SQL Server Management Studio
UI	User Interface
VM	Virtual Machine

Inleiding

In de voorbije jaren is er een serieuze opmars geweest in de mobiele wereld. Een wereld zonder smartphones is de dag van vandaag bijna ondenkbaar. Hierdoor willen bedrijven meer en meer investeren in de mobiele markt. De vraag is niet langer meer of het een goed idee is om hun services aan te bieden op mobiele toestellen, maar er wordt eerder gekeken naar de beste mogelijkheid om dit te verwezenlijken. Dit laatste is niet altijd een gemakkelijke keuze en brengt een hele hoop verwarring met zich mee.

Tijdens het ontwikkelproces van webapplicaties wordt er in eerste instantie vaak aandacht besteed aan een *responsive layout*. Dat wil zeggen dat de webapplicatie zich correct schaalt afhankelijk van het toestel waarop de applicatie gebruikt wordt. Dit ziet er op het eerste zicht een aantrekkelijke optie uit, omdat een applicatie op tal van toestellen werkt met een minimale inspanning. Dit is namelijk ook het geval bij de MyClient-applicatie van Yunify.

Toch is de beslissing genomen om de applicatie nogmaals te ontwikkelen maar dan als een *native* mobiele applicatie. De reden hiervoor is dat deze manier van ontwikkelen op een mobiel platform functionaliteiten aanbiedt die op een webapplicatie helemaal niet mogelijk zijn. Zo kan er bijvoorbeeld gebruik gemaakt worden van pushnotificaties, de trilfunctie, de camera en nog veel meer.

Nadat de keuze gemaakt is om van een webapplicatie over te schakelen naar een *native* applicatie kan het ontwikkelingsproces nog niet van start gaan. Een *native* applicatie kan namelijk ontwikkeld worden met behulp van verschillende technologieën. Deze technologieën hebben op hun beurt ook nog tal van frameworks waarmee applicaties gebouwd kunnen worden.

Hier ontstaat er vaak verwarring omdat de voor- en nadelen van deze technologieën en frameworks minder duidelijk zijn. In dit onderzoek wordt er dieper onderzocht in welke maten twee van de populairste frameworks om platformonafhankelijke *native* applicaties te bouwen verschillen. Dit gebeurt enerzijds aan de hand van een literatuurstudie. Vervolgens worden mijn bevindingen ook eens op de proef gesteld door een *proof of concept* uit te werken met beide frameworks.

I. Stageverslag

1 Bedrijfsvoorstelling

1.1 Situering bedrijf

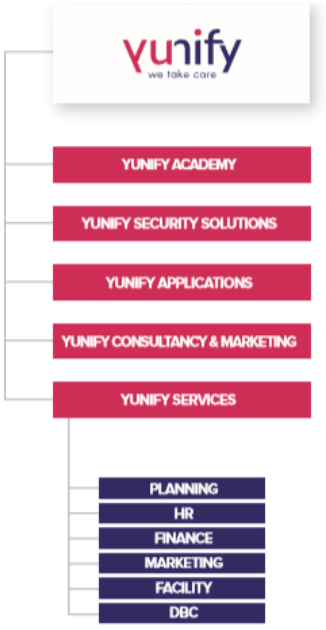
Yunify is een IT-bedrijf dat gevestigd is in Nederland. Het is één van de meest succesvolle en snelst groeiende bedrijven in het zuiden van Nederland. Yunify maakt deel uit van de IIC Group, een investeringsmaatschappij met een sterke focus op *healthcare.*

Figuur 1 toont de positie van Yunify in het IIC organigram.

Figuur 1 IIC Group organigram

Yunify zelf is nog eens ingedeeld in verschillende afdelingen. De afdeling waarin de stageopdracht plaatsvindt is *Yunify Applications*. Dit is de afdeling die zich focust op diverse e-health-oplossingen.

Figuur 2 toont de verschillende afdelingen binnen Yunify.



Figuur 2 Organigram Yunify

Het bedrijf is opgericht in 2012 en blijft groeien door steeds nieuwe werknemers aan te werven. Momenteel wordt er gewerkt in hun kantoor op de Mijnweg te Geleen.

1.2 Cultuur

Zoals eerder vermeld heeft Yunify een sterke focus op *healthcare*. Yunify verenigt passie voor zorg, wetenschappelijke kennis en creativiteit op een manier die uniek is in de gezondheidszorg. Er wordt dagelijks gewerkt aan e-health-oplossingen die de kwaliteit van leven en vitaliteit verbeteren voor hun cliënten.

Met deze oplossingen is het de bedoeling om de oorzaak van medische problemen aan te pakken. In de gezondheidszorg wordt er vaak gekeken naar manieren om gezondheidsproblemen zoals rugpijn, burn-outs en insomnia op te lossen in plaats van de oorzaak direct aan te pakken. Gevolgen hiervan zijn stijgingen van gezondheidskosten voor de cliënten en dalingen in werkkracht.

De oplossingen die aangeboden worden in verband met e-health zijn *hightech*-oplossingen voor smartphones, tablets en computers. In plaats van te investeren in ééndimensionale fitness-applicaties of *tracking devices* wordt de nadruk gelegd op multidimensionale en complete applicaties.

In het ontwikkelingsproces wordt de agile methodologie gehanteerd. Er zijn dagelijkse stand-ups zodat de ontwikkelaars een duidelijker zicht krijgen van de activiteiten van hun teamleden. Ook wordt er gewerkt met de technologieën van *Azure DevOps* voor een efficiënte samenwerking.

De gebruikte technologieën staan niet vast waardoor de ontwikkelaars op de hoogte moeten blijven van de laatste nieuwe technologieën in een snel evoluerende IT-wereld. Binnen Yunify zijn er experts die gespecialiseerd zijn in technologieën zoals .NET zonder het principe van levenslang leren te verwaarlozen. Er wordt aandacht besteed aan de nieuwste technologieën zoals *Virtual Reality* (Unity) en Flutter.

1.3 Bedrijfsgegevens

Yunify investeert in zijn werknemers en is sinds 2012 gegroeid tot een onderneming van meer dan 150 mensen. Dit zijn mensen van meer dan 15 nationaliteiten die gebruik maken van talloze disciplines om dag na dag grenzen te verleggen van wat mogelijk is. Omdat Yunify bestaat uit werknemers van diverse origine is de voertaal bijgevolg Engels.

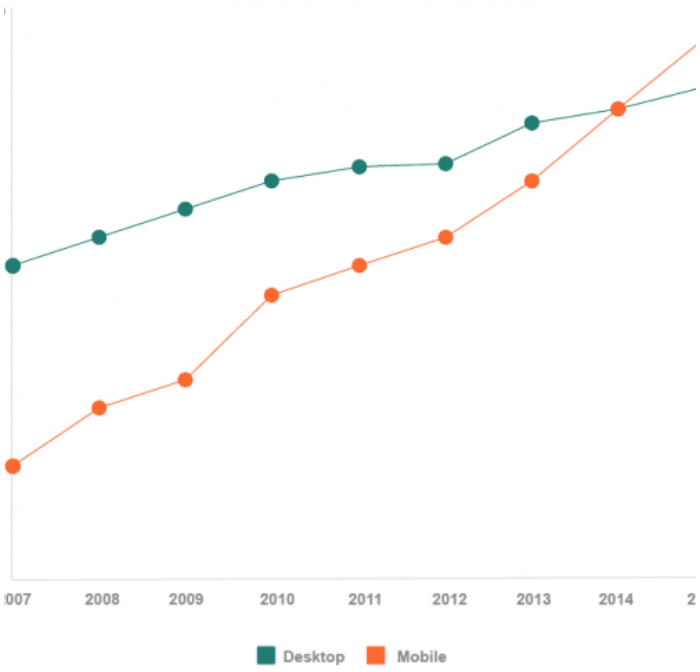
2 Voorstelling stageopdracht

2.1 Probleemstelling

2.1.1 Situering van het probleem

Yunify heeft momenteel een webapplicatie genaamd MyClient. Dit is een applicatie die gebruik maakt van technologieën zoals ASP.NET en Razor. Om het gebruik van de applicatie te optimaliseren is er gewerkt met een *mobile first approach*. Hierdoor kan de applicatie zonder neveneffecten gebruikt worden op allerlei toestellen van diverse groottes. Dit is een goede manier van ontwikkelen, omdat dezelfde code gebruikt kan worden terwijl de applicatie zowel op een computer als op een smartphone draait. Jammer genoeg is dit niet voldoende om de mobiele markt te kunnen veroveren.

In zijn studie ‘Mobile marketing statistics compilation’ concludeert Dave Chaffey dat mobiele toestellen meer gebruikt worden dan computers op het internet sinds 2014. Zijn conclusie wordt verder ook gevisualiseerd in het grafiek van figuur 3. [1]



Figuur 3 Internetgebruikers doorheen de jaren

Het is dus een noodzaak om sterk te investeren in de mobiele markt om succesvol te blijven naar de toekomst toe.

Om een goede indruk te maken bij de mobiele gebruikers moet er aandacht besteed worden aan hun eisen. Onderzoek toont aan dat gebruikers vaak de voorkeur geven aan *native* applicaties tegenover webapplicaties. Zo blijkt dat mobiele gebruikers 90% van hun tijd spenderen in *native* applicaties en slechts 10% in een browser. [2] Uiteindelijk heeft Yunify de beslissing genomen om de overschakeling te maken naar een *native* applicatie.

2.1.2 Achtergrondinformatie

De applicatie waarrond dit project draait is MyClient. Dit is een e-health-applicatie die gebruikt wordt als een portaal tussen behandelaar en cliënt. De applicatie kan zowel door behandelaars als cliënten gebruikt worden. Afhankelijk van de inloggegevens biedt de applicatie verschillende functionaliteiten aan.

Een behandelaar kan onder andere cliënten opvolgen, communiceren met cliënten en inhoud aanmaken. Cliënten kunnen bijgevolg communiceren met de behandelaar, inhoud consumeren en persoonlijke data ingeven. Dit is de kern van de applicatie zonder te veel op de details in te gaan.

2.2 Doelstellingen

In dit project is het de bedoeling om de uitgebreide MyClient-applicatie om te bouwen naar een *native* applicatie. MyClient is momenteel nog geen volledig afgewerkt product waardoor de andere teamleden verder werken aan de webapplicatie. Bijgevolg koppelt het team geen extra ontwikkelaars aan dit project, maar wordt er natuurlijk wel de nodige ondersteuning aangeboden.

Hierdoor wordt de opdracht beperkt tot slechts enkele functionaliteiten van de webapplicatie. Naast de functionaliteiten is het ook belangrijk om een duidelijke documentatie bij te houden. Omdat dit project na de stageperiode nog verder ontwikkeld zal worden is het belangrijk om aan het eind van de stage de opbouw van de applicatie over te dragen. De ontwikkelaars van Yunify die dit project verder zullen uitwerken moeten op de hoogte zijn van de applicatiestructuur zodat er gemakkelijk nieuwe functionaliteiten ontwikkeld kunnen worden.

Samen met de documentatie worden ook verschillende onderzoeksonderwerpen uitgeschreven. Dit heeft ook betrekking tot de overdracht van de applicatie aan het einde van de stage. MyClient zit momenteel in een uitgebreide infrastructuur en communiceert met verschillende externe systemen. Om dezelfde functionaliteiten uit te werken moet de mobiele applicatie ook samenwerken met deze externe systemen. Bijgevolg wordt er onderzocht in welke mate dit mogelijk is en of er andere oplossingen bestaan die beter zouden zijn voor de mobiele specifieke eisen.

2.3 Omgeving

Om dit project te kunnen realiseren wordt er gebruik gemaakt van verschillende technologieën, tools en diensten.

2.3.1 Communicatie

Microsoft Teams is de communicatietool die gebruikt wordt tijdens de stageperiode. Deze tool maakt het mogelijk om een eigen omgeving aan te maken voor het team. Een omgeving wordt gebruikt om te communiceren, te integreren met andere Office-365-applicaties en bestanden uit te wisselen.

2.3.2 Ontwikkeling

Dit project kan opgedeeld worden in twee delen. Eerst en vooral is er de mobiele applicatie. Hiervoor wordt Flutter gebruikt, een framework uitgebracht door Google om mobiele applicaties te bouwen. Voor de ontwikkelomgeving wordt *Visual Studio Code* gebruikt, een IDE uitgebracht door Microsoft.

Verder wordt er ook een simpele backend gebouwd. Dit is nodig om data op te halen vanuit de databank en vervolgens aan te bieden aan de mobiele applicatie. De server wordt gebouwd met ASP.NET Core. Dit is een versie van ASP.NET die op meerdere besturingssystemen kan draaien dan enkel Windows. De server wordt gebouwd in Visual Studio, een IDE uitgebracht door Microsoft die ASP.NET Core-applicaties ondersteunt.

2.3.3 Data

Een applicatie zoals MyClient maakt gebruik van veel data die opgeslagen wordt in SQL-server, een beheersysteem van databanken ontwikkeld door Microsoft. Hiernaast wordt ook gebruik gemaakt van SSMS. Dit is een applicatie die samen gebouwd is met SQL-server in 2005 en wordt gebruikt voor de configuratie van alle componenten binnen SQL-server.

De backend dient data op te halen vanuit de databank. Om deze communicatie te voorzien wordt er gebruik gemaakt van Entity Framework Core. Dit is een technologie die het mogelijk maakt om data op te halen vanuit een databank om deze vervolgens om te vormen tot objecten waarmee gewerkt kan worden in een ASP.NET Core-applicatie.

2.3.4 Versiebeheer

Voor de versiebeheer van het project wordt Azure Devops gebruikt. Dit is een set van tools die het mogelijk maakt om applicaties te ontwikkelen in teamverband op een gestructureerde manier. Er kunnen onder andere *kanban boards* en automatische *build pipelines* opgezet worden, maar in dit project worden voornamelijk de *repositories* benut. Azure Devops biedt namelijk gratis privé *repositories* aan die gebruikt worden voor de versiebeheer van dit project.

3 Uitwerking stageopdracht

3.1 Methodologieën

De *agile* methodologie is momenteel één van de meest gebruikte methodologieën voor softwareontwikkeling. In deze stageopdracht worden enkele aspecten van *agile* gebruikt om het project te beheren.

Iedere ochtend wordt er een dagelijkse stand-up gehouden zodra iedereen aanwezig is. Hier reflecteert iedereen in het team om de beurt kort over zijn persoonlijke stand van zaken. In een stand-up wordt kort toegelicht waaraan de dag ervoor gewerkt is, wat de problemen waren en wat de planning voor de huidige dag is. Op deze manier houdt iedereen in het team elkaar op de hoogte.

Verder wordt er in dit project beroep gedaan op Azure Devops. Dit platform biedt tal van tools om een project op een *agile* manier te begeleiden. In dit project wordt dit platform gebruikt voor de versiebeheer van de code.

Buiten deze twee aspecten wordt de *agile* methodologie niet zo veel gebruikt. Dit komt grotendeels omdat er geen andere ontwikkelaars van Yunify werken aan de functionaliteiten van de mobiele applicatie. Dit maakt het een stuk minder zinvol om bijvoorbeeld een ticketsysteem op te stellen voor de resterende functionaliteiten.

Er wordt ook niet gewerkt met sprints, maar in plaats hiervan worden er meetings georganiseerd met de stagebegeleider om de vooruitgang en het verder verloop van het project te overlopen. Bij sprints wordt er gewerkt met verschillende periodes van een gekozen aantal weken. In dit project zijn er geen afspraken rond tijd gemaakt, maar worden de meetings georganiseerd als er bepaalde onduidelijkheden of suggesties ontstaan.

3.2 Startfase

In de startfase van de stage heeft Yunify beslist om het Flutter-framework te kiezen voor dit project. Een eerste belangrijke taak van de stage is bijgevolg dit framework verkennen voordat er effectief geprogrammeerd wordt. De officiële documentatie van Flutter maakt het ontwikkelproces een stuk aangenamer. Naast de documentatie wordt er ook een demoapplicatie voorzien die gebruikt kan worden voor een eerste aanraking met Flutter.

Een ander belangrijk aspect van het project is de MyClient infrastructuur. Omdat er in de stageopdracht enkele functionaliteiten van de MyClient applicatie worden vertaald in een mobiele applicatie is het noodzakelijk om de onderliggende infrastructuur te begrijpen. De mobiele applicatie moet namelijk ook samenwerken met dezelfde externe systemen waarmee de webapplicatie mee samenwerkt.

De MyClient-applicatie wordt al een hele tijd uitgewerkt door verschillende ontwikkelaars dus is het niet zo makkelijk om uit te zoeken hoe de verschillende systemen samenhangen. Hierdoor worden

enkel de nodige delen bestudeert die noodzakelijk zijn voor de mobiele applicatie. Dit gebeurt door de databank structuur grondig te bestuderen en de werking van de webapplicatie te ondernemen.

3.3 State management

Een belangrijk onderdeel van de applicatie is de *state*. Binnen de applicatie wordt dit concept op verschillende manieren toegepast.

3.3.1 Stateful en Stateless

Enerzijds kunnen Flutter-widgets *stateful* of *stateless* zijn. Dit gedrag wordt vastgelegd afhankelijk van de klasse waar de widget van afgeleid is. Als de widget overerft van 'StatefulWidget' is er een *state* aanwezig waardoor de widget veranderlijk is. Als de *state* gewijzigd wordt, gaat de logica om de inhoud te tonen opnieuw uitgevoerd worden. Als een widget afgeleid is van 'StatelessWidget' is er geen *state* aanwezig waardoor de widget onveranderlijk is. De logica om de inhoud te tonen zal bijgevolg slechts één keer uitgevoerd worden.

3.3.2 Scoped Model

Anderzijds is het ook handig om een *state* te hebben over de hele applicatie in plaats van per individuele widget. Hierdoor kan er data opgeslagen worden die beschikbaar is over de volledige applicatie. Ter gevolg is het niet meer nodig om op een onoverzichtelijke manier data uit te wisselen tussen diep geneste widgets via de *constructors*. Eender welke widget in de applicatie kan op deze manier beroep doen op de *state* van de applicatie en zelfs reageren als deze *state* zou veranderen. Om dit te realiseren wordt er gebruik gemaakt van een *library*.

Ten eerste biedt de *library* een klasse genaamd 'Model' aan die de data of de *state* van de applicatie beheert. Er kunnen klassen aangemaakt worden afgeleid van '*Model*' om gegroepeerde data op te slaan.

Ten tweede wordt er een widget genaamd 'Scoped Model' aangeboden. Als widgets beroep moeten doen op de data van een 'Model' dan moeten deze widgets kinderen zijn van een 'Scoped Model'. Omdat het de bedoeling is om de data van een 'Model' overal in de applicatie aan te bieden is het logisch om de hele applicatie in te pakken in een 'Scoped Model'. In de declaratie van de 'Scoped Model' kan een eigen gedefinieerde klasse, afgeleid van 'Model', meegegeven worden. Hierdoor hebben deze widgets toegang tot de data die opgeslagen is in een instantie van de meegegeven 'Model'.

Ten laatste wordt er een 'Scoped Model Descendant' aangeboden. Deze widget kan gebruikt worden als kind van een 'Scoped Model'. Het bevat een 'builder'-methode die uitgevoerd wordt als de *state* van de meegegeven 'Model' aangepast wordt.

Een praktisch voorbeeld van het gebruik van de *library* in de applicatie is te zien in figuur 12.

```
final MainModel model

@override
Widget build(BuildContext context) {
    return ScopedModel<MainModel>(
        model: model,
        child: MaterialApp(
            title: 'MyClient',
            theme: createThemeData(model.selectedTheme),
            routes: {
              '/': (BuildContext context) => ScopedModelDescendant(
                    builder: (BuildContext context, Widget widget, MainModel model) {
                        return model.user == null ? AuthPage(model) : Home();
                    }
                )
            }
        ));
    }
```

Figuur 4 Voorbeeld van de Scoped Model library

In dit voorbeeld is 'MainModel' een eigen gedefinieerde klasse afgeleid van 'Model'. Het bevat een eigenschap genaamd 'user' die pas opgevuld wordt als een gebruiker ingelogd is.

De volledige applicatie wordt ingepakt in een 'Scoped Model' met als concrete implementatie een 'MainModel' waardoor de data van de 'MainModel' beschikbaar is over de hele applicatie.

Er is één standaard route aanwezig die gebruik maakt van een 'Scoped Model Descendant'. In de 'builder'-methode wordt eerst gekeken of de 'user' van de 'MainModel'-instantie aanwezig is. Als dit het geval is wordt de startpagina getoond, anders zal de loginpagina getoond worden. Zodra een gebruiker inlogt, wordt de 'user'-eigenschap ingevuld waardoor ook de 'builder'-methode opnieuw uitgevoerd wordt.

3.4 Authenticatie

Wanneer de applicatie opgestart wordt, komt de gebruiker op de loginpagina terecht. Nadat de gebruiker probeert in te loggen met een e-mailadres en wachtwoord wordt er een HTTP-aanvraag gestuurd naar de MyClient-server. De server is verantwoordelijk voor het afhandelen en valideren van de inloggegevens.

Als de logingegevens ongeldig zijn komt er een gepaste boodschap op het scherm. Bij het ingeven van geldige logingegevens krijgt de gebruiker een JWT-token toegewezen en toegang tot de applicatie. De JWT-token moet vervolgens meegegeven worden bij elke nieuwe HTTP-aanvraag. Aan de hand van de token kan de server controleren of de aanvraag van een geldige gebruiker komt.

Om er voor te zorgen dat de gebruiker niet elke keer opnieuw moet inloggen, wordt er gebruik gemaakt van een *library* uitgebracht door de ontwikkelaars van Flutter. Deze *library* maakt het mogelijk om met dezelfde code de *Shared Preferences* van zowel Android als iOS aan te spreken. *Shared Preferences* is een collectie van simpele *key-value pairs* die blijven bestaan nadat de applicatie afgesloten wordt. Wanneer een gebruiker inlogt worden er enkele gebruikersgegevens opgeslagen in de *Shared Preferences*. Dit zijn gegevens zoals de e-mail, gebruikersnaam en JWT-*token*.

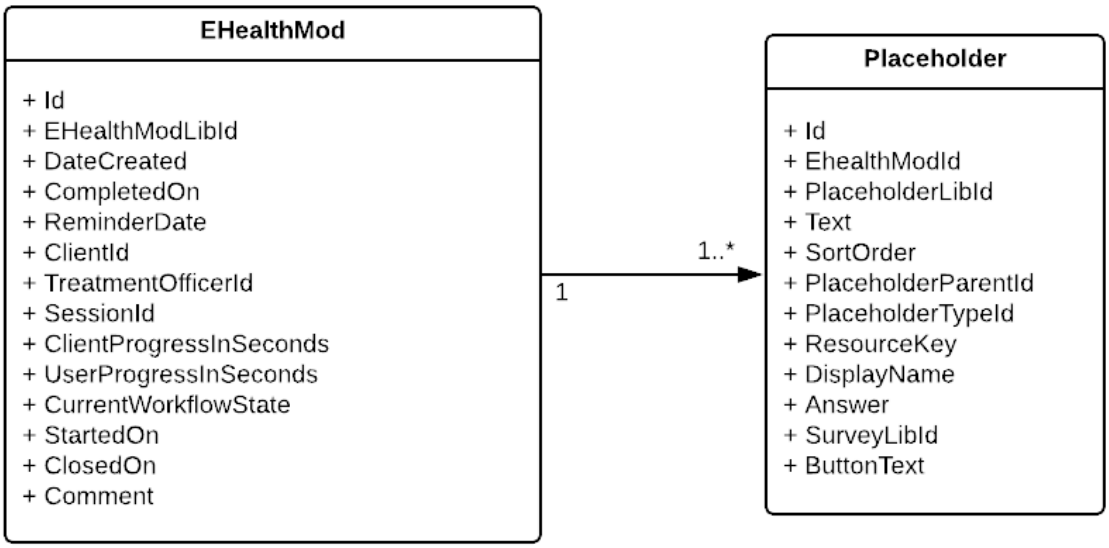
Wanneer de applicatie vervolgens opgestart wordt, wordt er een methode uitgevoerd die controleert of deze gegevens aanwezig zijn. Als dit effectief het geval is krijgt de gebruiker toegang tot de applicatie zonder opnieuw te moeten inloggen. De gebruiker heeft ook de mogelijkheid om zichzelf uit te loggen. Hierbij worden de *Shared Preferences* leeg gemaakt en wordt er terug naar de loginpagina genavigeerd.

3.5 E-health

De functionaliteit van MyClient die eerst en vooral in de mobiele applicatie verwerkt wordt is het overzicht van de e-health-modules van een cliënt. Een behandelaar kan aan de hand van de webapplicatie een module opbouwen en koppelen aan een cliënt. Een module bestaat uit allerlei verschillende componenten die betrekking hebben tot de behandeling van een cliënt. Dit zijn onder andere componenten zoals informatie over een bepaald onderwerp, oefeningen die een cliënt moet uitvoeren, tabellen met data over een onderwerp, foto's voor meer informatie over een onderwerp en invuloefeningen.

Een module kan dus bij gevolg dynamisch opgebouwd worden aan de hand van de verschillende componenten. Voordat modules kunnen worden weergegeven in de applicatie is het belangrijk om te weten hoe de inhoud van een module wordt opgeslagen. Na zelf onderzoek te doen en wat uitleg te krijgen van het team werd het duidelijk hoe de inhoud opgeslagen wordt in de databank.

Het probleem hierbij is dat de dataopslag geoptimaliseerd is voor het gebruik in een webapplicatie. Figuur 4 toont de databank structuur van de opmaak van een e-health-module



Figuur 5 Structuur e-health-module

Een module bestaat uit verschillende *placeholders*. Dit zijn de componenten waaruit een module is opgebouwd (informatie, foto's, tabellen, …). Een *placeholder* bevat een 'ResourceKey' die gebruikt wordt om een *localization* op te vragen. Een *localization* bevat de uiteindelijke inhoud die weergeven moet worden in een component.

De structuur van een *localization* is te zien in figuur 5.

Figuur 6 Structuur localization

De inhoud wordt opgeslagen in de vorm van HTML-code. HTML wordt gebruikt door browsers om webapplicaties te tonen en is ter gevolg niet optimaal in een *native* mobiele applicatie. Er moet dus onderzocht worden hoe de inhoud van een module het beste getoond kan worden in de *native* applicatie.

3.5.1 HTML Rendering

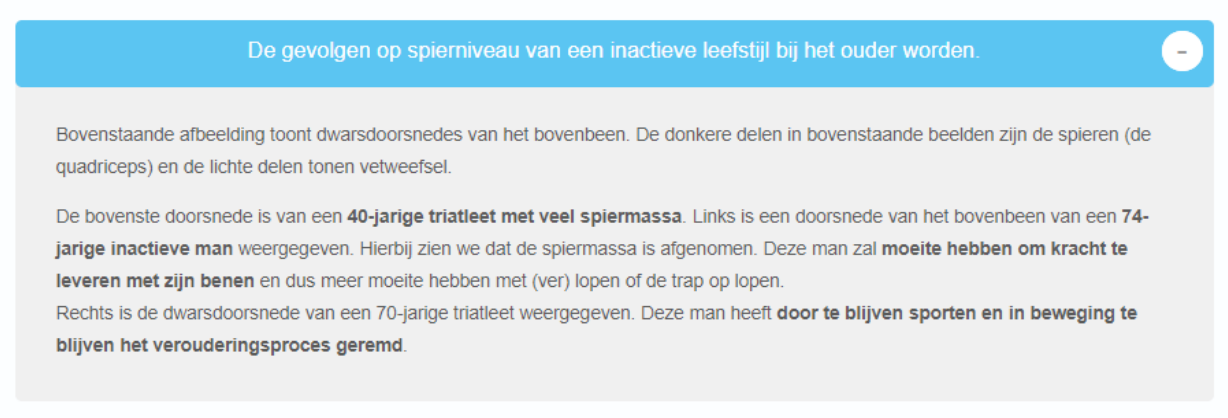
Een eerste mogelijkheid is om de HTML-code te gebruiken in de *native* applicatie. Zoals eerder vermeld, is dit geen ideale oplossing, maar dat wil niet zeggen dat het niet mogelijk is. Er bestaat een open source *library* die het mogelijk maakt om HTML-code in een Flutter-applicatie te tonen.

Het voordeel hiervan is dat de code snel en eenvoudig weergegeven kan worden in de applicatie. Er zouden geen aanpassingen nodig zijn in de datastructuur waardoor Yunify geen extra werk moet verrichten.

Uiteindelijk wordt er niet voor deze oplossing gekozen, omdat de nadelen groter zijn dan de voordelen. HTML-code maakt namelijk gebruik van CSS om styling te voorzien in een webapplicatie. Dit is iets waar geen rekening mee wordt gehouden in de gebruikte *library*. Styling is uiteindelijk wel mogelijk als de broncode van de *library* aangepast wordt.

Dit is jammer genoeg niet voldoende om deze oplossing te gebruiken, want de mogelijkheden van styling zijn toch een stuk beperkter dan in de webapplicatie. Zo zijn enkele technieken om bepaalde componenten te tonen zelfs onmogelijk in de *native* applicatie.

Bijvoorbeeld het tonen van de 'ButtonCollapse'-component, zoals te zien in figuur 6, is niet mogelijk op deze manier in de *native* applicatie.



Figuur 7 de ButtonCollapse-component van de webapplicatie

Door op de titel te klikken kan de inhoud van een 'ButtonCollapse' verborgen en vervolgens weer getoond worden. Hiervoor maakt de webapplicatie gebruik van Javascript om logica aan een component toe te voegen. De beschreven *library* van deze oplossing biedt geen mogelijkheden om logica aan componenten toe te voegen waardoor het niet haalbaar is om de component op deze manier na te bouwen.

3.5.2 WebView

Een volgende mogelijkheid is om de HTML-code te tonen aan de hand van een *web view*. Een *web view* maakt het mogelijk om de inhoud van eender welke webapplicatie in een mobiele applicatie te tonen. Een gebruiker kan dus via een mobiele applicatie interageren met een andere webapplicatie. In het geval van MyClient komt dit goed uit, omdat er al een webapplicatie voorzien is.

Bij deze oplossing wordt er dus beroep gedaan op de MyClient-webapplicatie om een e-health-module in te vullen. Een eerste probleem hierbij is dat de gebruiker op de webapplicatie zou blijven nadat een module ingediend wordt. Er is geen manier om de gebruiker terug naar de mobiele applicatie te navigeren nadat de module ingediend is. Hierdoor zou de gebruiker waarschijnlijk de webapplicatie van MyClient blijven gebruiken waardoor de mobiele versie niet meer benut wordt.

Nog een probleem is dat Flutter op het moment van de stage nog geen werkende *web view* heeft. De meeste mobiele ontwikkelingstechnologieën voorzien standaard een ingebakken *web view* in hun softwarepakket, maar bij Flutter is dit nog niet het geval. Dit komt omdat Flutter nog in een vroege fase zit waardoor de uitgevers nog volop nieuwe functionaliteiten zijn aan het ontwikkelen.

Bijgevolg is de beslissing genomen om geen *web view* te gebruiken in de applicatie.

3.5.3 Flutter Widgets

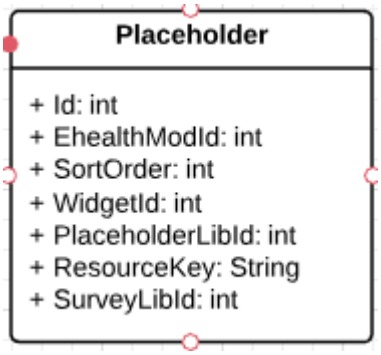
De oplossing die uiteindelijk gebruikt wordt in de applicatie maakt gebruik van Flutter-widgets. Widgets vormen de kern van Flutter en kunnen gezien worden als herbruikbare componenten die toegepast worden om een applicatie op te bouwen. Op deze manier wordt de Flutter-technologie optimaal benut. Flutter voorziet tal van kant en klare widgets waarmee inhoud verwerkt kan worden op het scherm maar er is natuurlijk ook de mogelijkheid om eigen widgets te definiëren.

Elke component van een e-health-module krijgt zijn eigen widget in de Flutter-applicatie. Op deze manier is de applicatie ook makkelijk uitbreidbaar want bij het ontwikkelen van een nieuwe

component in de webapplicatie moet er enkel een nieuwe widget in de mobiele applicatie ontwikkeld worden op dezelfde manier.

Het enige nadeel dat deze oplossing met zich meebrengt is dat de structuur van de inhoud aangepast moet worden. De inhoud kan niet meer in de vorm van HTML opgeslagen worden. Om dit nadeel op te vangen wordt er in de stageopdracht gewerkt met dummy data. Als de applicatie na de stageperiode verder uitgewerkt wordt, moet de dataopslag aangepast worden met de structuur van de dummy data.

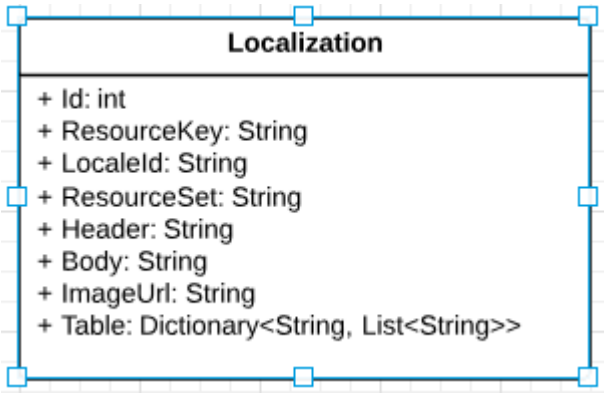
Eerst en vooral wordt de structuur van een *placeholder* aangepast zoals getoond in figuur 7.



Figuur 8 Aangepaste placeholder structuur

Het nieuwe veld 'WidgetId' wordt gebruikt om te identificeren welke widget ingezet moet worden voor het weergeven van de huidige inhoud. De rest van de structuur kan behouden worden. De 'ResourceKey' wordt nog steeds gebruikt om een *localization* aan te spreken.

De structuur van een *localization* wordt ook aangepast zoals te zien is in figuur 8.



Figuur 9 Aangepaste localization structuur

Eerst bevatte een *localization* een 'value'-veld met daarin de HTML code van een component. In deze versie staat de *localization* zelf in om de inhoud in verschillende velden op te slaan. Het nieuwe veld 'Header' staat in voor de titel van een component en het veld 'Body' voor de effectieve inhoud. Het veld 'ImageUrl' kan een referentie bevatten naar een foto die gebruikt moet worden in de component. Tenslotte wordt het 'Table'-veld gebruikt om een tabel van data op te slaan die gebruikt kan worden in een component. Met deze nieuwe velden kunnen dynamische Flutter-widgets opgebouwd worden om uiteindelijk een volledige module te representeren.

3.5.3.1 Voorbeeld widget

Eén van de simpelste widgets is de 'Content Block' (CB). Een voorbeeld van de nieuwe datastructuur die gebruikt wordt voor een CB wordt weergeven in figuur 9. Dit is de data van een *placeholder* die vervolgens beroep doet op een *localization* voor de effectieve inhoud.


```
{
  "Id": 1,
  "EHealthModId": 1,
  "SortOrder": 1,
  "WidgetId": 1,
  "PlaceholderLibId": 1,
  "ResourceKey": "dcp18-Ehealth_mod2_01",
  "SurveyLibId": null
}
```


Figuur 10 JSON formaat van een placeholder

Omdat de *placeholder* een *id* heeft van 1, wordt de inhoud weergegeven in de vorm van een CB. Deze inhoud moet eerst opgevraagd worden aan de hand van de 'ResourceKey'. De structuur van de *localization* met de voorziene 'ResourceKey' wordt weergeven in figuur 10.


```
{
  "Id": 1,
  "ResourceKey": "dcp18-Ehealth_mod2_01",
  "LocaleId": "nl-NL-rug",
  "ResourceSet": null,
  "Header": "Inspanning – "Thuis aan de slag"",
  "Body": "Een belangrijk onderdeel van het behandeltraject van het Rughuis is gericht op het
    onderhouden van de behaalde resultaten. Hiervoor is het belangrijk om ook na afronding van
    het behandeltraject lichamelijk actief te blijven. In deze module wordt stil gestaan bij de
    voordelen van lichamelijke activiteit en worden ook praktische tips gegeven hoe jij in jouw
    thuissituatie in beweging kunt blijven.",
  "ImageUrl": null,
  "Table": null
}
```


Figuur 11 JSON formaat van een localization

De CB toont de header vervolgens in een speciale weergave met de body van de *localization* eronder. Het resultaat van de data die uiteindelijk in de applicatie getoond wordt is te zien in figuur 11.

Figuur 12 Voorbeeld van een Content Block

De verschillende koppelingen tussen widget-*ids* en widgets samen met gedetailleerde beschrijvingen van de verschillende widgets zijn te vinden in de documentatie van dit project.

3.6 Vragenlijsten

Een volgende belangrijke functionaliteit die in de mobiele applicatie wordt verwerkt is de mogelijkheid om cliënten vragenlijsten te laten invullen. De vragenlijsten in de webapplicatie maken gebruik van verschillende vraagsoorten die ook beschikbaar moeten zijn op de mobiele applicatie.

Vragenlijsten zijn op dezelfde manier opgebouwd als de e-health-modules maar doen beroep op een eigen set van widgets die de verschillende vraagsoorten vertegenwoordigen. Voor de datastructuur wordt de structuur van de webapplicatie behouden.

Een vragenlijst is opgebouwd uit verschillende secties die in de webapplicatie dynamisch onder elkaar getoond worden. In de Flutter-applicatie wordt dit op een andere manier aangepakt. Iedere sectie krijgt zijn eigen pagina met de mogelijkheid om vooruit te navigeren naar de volgende sectie. Als er geen secties meer beschikbaar zijn, ziet de cliënt een eindboodschap met een knop om de vragenlijst af te ronden.

Zoals eerder vermeld, maakt een vragenlijst gebruik van een eigen set van widgets. Omdat een vragenlijst simpelweg een formulier is, kunnen de reeds bestaande widgets gebruikt worden die Flutter aanbiedt voor het bouwen van een formulier. Deze widgets zijn subklassen van de 'FormField'-klasse en worden vervolgens gebruikt met een aangepaste styling. Aan de hand van de 'FormField'-klasse worden er functionaliteiten toegevoegd aan de widget zoals het valideren en opslaan van de input. Voor de meeste vraagsoorten bieden de bestaande 'FormField'-implementaties niet genoeg functionaliteiten aan. In dat geval worden eigen subklassen geschreven van 'FormField' om extra functionaliteiten toe te voegen. De vraagsoorten worden uiteindelijk omgezet tot bestaande of eigen gedefinieerde formulier widgets.

Eens een client een vragenlijst afrondt worden alle antwoorden gebundeld in een lijst. De antwoordenlijst wordt omgezet tot JSON-formaat en wordt vervolgens als HTTP-aanvraag doorgestuurd naar de server. De afhandeling van deze data in de server valt buiten de scope van de stageopdracht.

3.7 Multitenancy

De applicatie wordt gebouwd voor meerdere *tenants* die elk een gepersonaliseerde versie van de applicatie krijgen. Dit wordt gerealiseerd door thema's en logo's te koppelen aan de verschillende *tenants* en deze vervolgens te gebruiken in de applicatie. Later wordt dit systeem uitgebreid door verschillende databases te koppelen aan de verschillende *tenants*, maar dit valt buiten de scope van de opdracht.

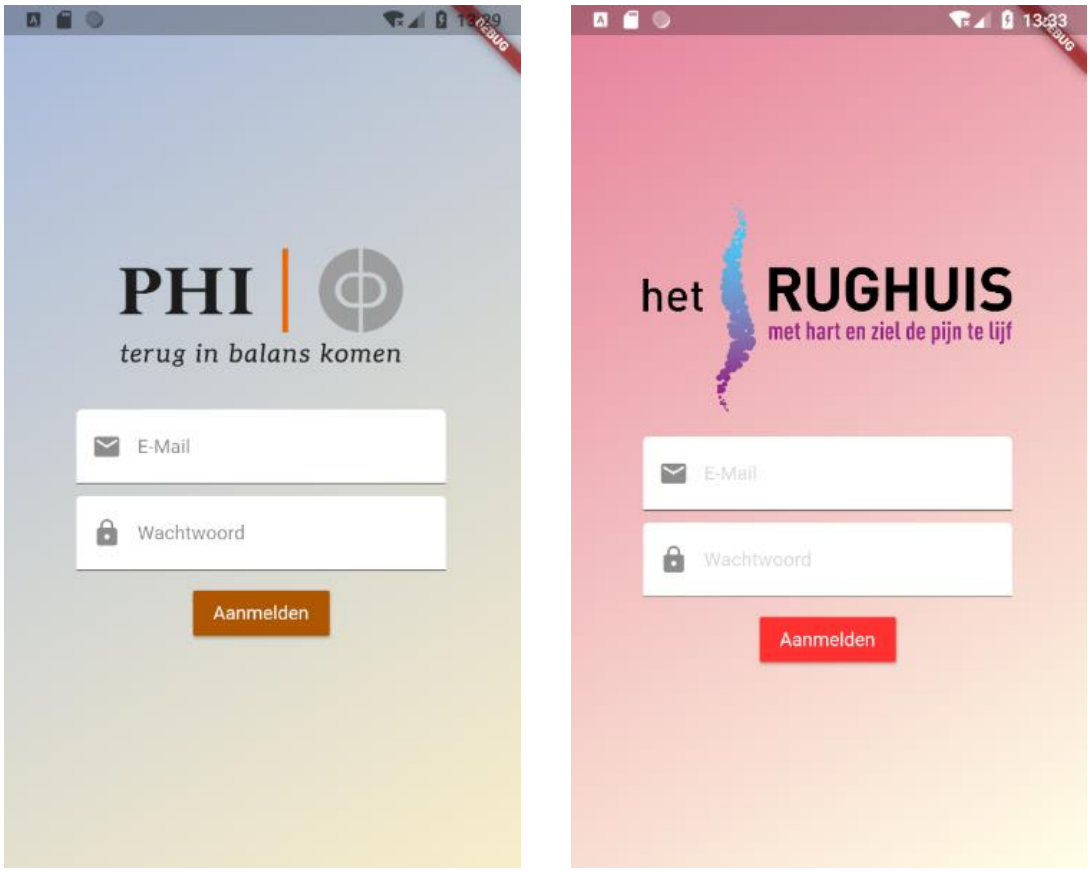
3.7.1 Theming

Elke *tenant* heeft een eigen set van thema's beschikbaar. Flutter ondersteunt *theming* door middel van een widget genaamd 'ThemeData'. Een thema is een instantie van deze widget en wordt meegegeven als kenmerk van de applicatie wanneer de applicatie opstart. Vervolgens wordt dit thema door bijna elke klasse gebruikt voor het renderen van de UI.

Een thema bestaat uit verschillende kenmerken die geconfigureerd kunnen worden in de instantie van 'ThemeData'. In de Flutter-applicatie wordt er gebruik gemaakt van een subset van de verschillende mogelijkheden. Er is een klasse aangemaakt die de subset van kenmerken bevat zodat

JSON-data gemapt kan worden naar een object van dit type. Dit object wordt vervolgens gebruikt om de kenmerken van 'ThemeData' op te vullen.

In dit project heeft elke *tenant* een eigen set van één of meer gepersonaliseerde thema's ter beschikking, zoals te zien in figuur 13.



Figuur 13 Voorbeeldthema's van de applicatie

Bij het opstarten van de applicatie worden alle beschikbare thema's van de huidige *tenant* ingelezen. Vervolgens wordt er gekeken als er al een thema geselecteerd is aan de hand van de *Shared Preferences*. Als er al een thema geselecteerd is dan wordt dit thema gebruikt, anders wordt er een standaard thema gekozen voor de cliënt. Verder is er een scherm gebouwd met de functionaliteit om een ander thema te kiezen die aangeboden wordt door de huidige *tenant*.

3.8 Documentatie

Naast het PoC wordt er ook een documentatie opgeleverd tijdens de stageperiode. Omdat ik als enige ontwikkelaar aan dit project werk, vindt de opdrachtgever het belangrijk dat de verschillende implementaties correct gedocumenteerd worden. Naast documentatie staan er ook enkele vergelijkingen en adviezen beschreven in dit document. Het vormt dus zowel een beeld van de geïmplementeerde functionaliteiten als de redenering waarom bepaalde implementatiedetails zijn uitgevoerd.

Bij de beschreven functionaliteiten staat er telkens vermeld wat de werking is maar nog belangrijker in welke maten het mogelijk is om de functionaliteit verder uit te breiden. Na de stageperiode wordt er door de opdrachtgevers beslist als ze resources gaan investeren om dit project volledig af te werken. Als het project verder uitgewerkt wordt dan zal de documentatie een cruciale rol spelen.

4 Besluit

4.1 Reflectie stageopdracht

Vóór de start van de stage was de technologie van het project nog niet gekozen. In de stageopdracht stond vermeld dat het React Native of Flutter zou worden. Omdat React Native eerder gebruikt werd in mijn IT-Project van de PXL-opleiding zou dit een interessante keuze geweest zijn. Hierdoor zou ik snellere resultaten opleveren en zou ik de kans krijgen om mezelf nog meer te verdiepen in de geavanceerde werkingen van dit framework.

Uiteindelijk heeft het stagebedrijf zelf de knoop doorgehakt en was de beslissing genomen om het Flutter-framework te gebruiken. Persoonlijk vind ik dat andere alternatieve zoals React Native of Ionic interessanter zouden zijn geweest voor de specifieke vereisten van dit project en dit heb ik ook aangehaald in mijn onderzoek.

Bij de start van de opdracht was er nog geen duidelijke scope gedefinieerd. Het was dus moeilijk om de functionaliteiten van de mobiele applicatie te bespreken omdat zowel ik als de stagebegeleider nog geen ervaring had met Flutter. Naarmate ik beter bekend geraakte met de mogelijkheden van het Flutter-framework ontstonden er verschillende vereisten voor dit project.

Aan het einde van de stage zijn alle doelstellingen van de opdracht voltooid. Het bedrijf heeft nu een *proof of concept* in handen voor een mobiele applicatie die gebaseerd is op hun huidige webapplicatie. Verder hebben ze ook een documentatie gekregen waarin vermeld staat hoe de applicatie is opgebouwd en welke mogelijkheden er bestaan om de applicatie nog uit te breiden.

4.2 Reflectie eigen functioneren

Op technisch vlak heb ik vooral leren werken met het Flutter-framework en met de programmeertaal Dart. Ook heb ik voor een eerste keer gewerkt aan een project waarvan een groot deel van de infrastructuur reeds bestond. Dit vond ik ook een grote uitdaging omdat ik mezelf heb moeten verdiepen in een zeer uitgebreide datastructuur.

Bij al mijn voorgaande projecten ben ik altijd met een schone lei mogen beginnen maar dat was in dit project niet het geval. Dit heb ik persoonlijk niet positief ervaren omdat ik het een leuke uitdaging vind om zelf een datastructuur op te zetten in overleg met het team. Hierdoor zou het ook niet nodig zijn om veel tijd te investeren in het ontcijferen van de ideeën van een andere ontwikkelaar omdat er geen documentatie beschikbaar was.

Naast het technisch vlak heb ik een eerste ervaring opgedaan als een softwareontwikkelaar in het bedrijfsleven. Het was een interessante ervaring om te werken in de werkomgeving van het bedrijf. Wat ik wel jammer vond was het feit dat ik als enige persoon aan dit project heb gewerkt. Verder zat er zelfs niemand in mijn team die ervaring had met Flutter. Hierdoor heb ik de kans niet gekregen om ideeën uit te wisselen rond de Flutter-technologie en was het moeilijker om bepaalde technische problemen op te lossen.

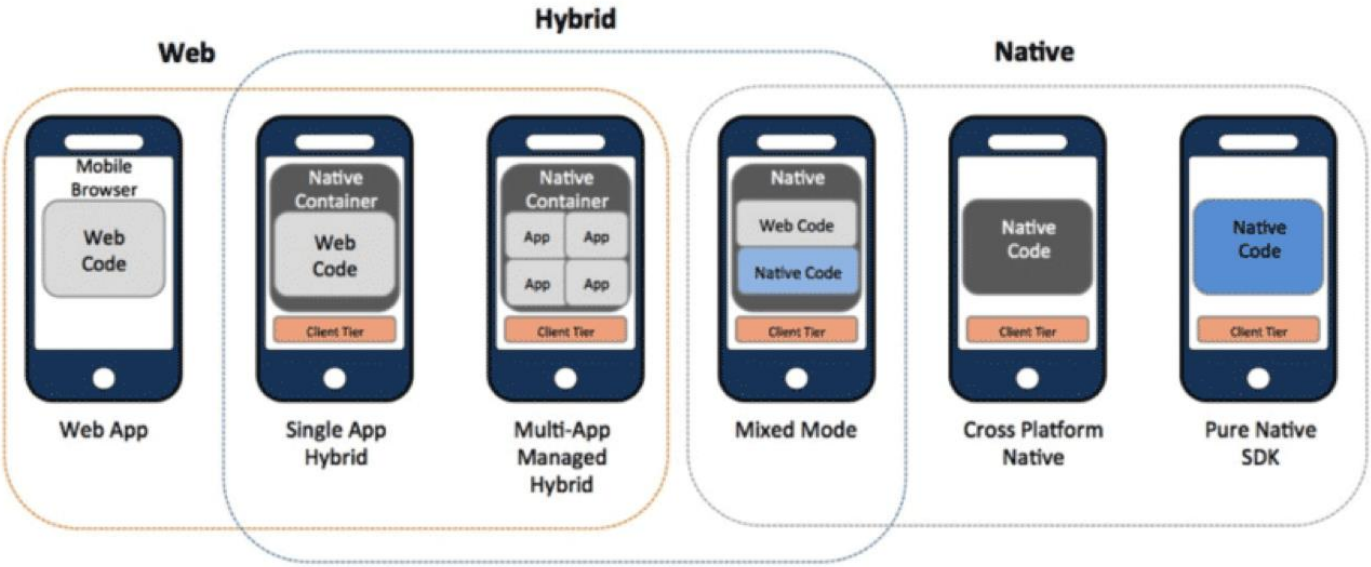
II. Onderzoekstopic

1 Probleemstelling

1.1 Mobiele ontwikkeling

Mobiele applicaties worden tegenwoordig op veel verschillende manieren gebouwd. Bedrijven komen steeds met nieuwe innovatieve ideeën om het mobiele ontwikkelingsproces te vergemakkelijken bij ontwikkelaars. De meeste van die ideeën zorgen er voor dat één enkele *codebase* gebruikt kan worden om dezelfde applicatie te draaien op meerdere besturingssystemen. Ondertussen kunnen mobiele applicaties al opgesplitst worden in drie hoofdcategorieën: web, hybride en *native* applicaties.

Figuur 14 toont een overzicht tussen de drie hoofdcategorieën.



Figuur 14 Overzicht mobiele ontwikkelingsmethodes

De web-categorie maakt gebruik van de browser op een mobiel toestel en is uiteindelijk niets meer dan een website die weergeven wordt in een ander formaat. Hierdoor kan een applicatie van deze categorie ook niet beschouwd worden als een mobiele applicatie waardoor er geen rekening mee wordt gehouden in het onderzoek.

De hybride categorie kan nog onderverdeeld worden in drie subcategorieën, maar omdat deze maar in kleine mate verschillen wordt er in dit onderzoek simpelweg gefocust op de hoofdcategorie. Bij deze technologie wordt er gewerkt met dezelfde programmeertalen als in de web-categorie, maar het verschil is dat hybride applicaties vertaald en gebruikt worden als *native* applicaties. [3] Dit is mogelijk door gebruik te maken van een *native* container. Hierdoor kunnen er met deze oplossing platformonafhankelijke *native* applicaties gebouwd worden.

Uiteindelijk is er nog de *native* categorie. Hier is het mogelijk om enerzijds gebruik te maken van een *native* SDK. Dit wil zeggen dat een applicatie gebouwd wordt voor een specifiek besturingssysteem. Anderzijds bestaan er ook *cross-platform native* technologieën die het mogelijk maken om platformonafhankelijke *native* applicaties te bouwen. De programmeertaal die hier gebruikt wordt

om een applicatie te bouwen hangt af van het gekozen framework. Zo kunnen *cross-platform* mobiele applicaties gebouwd worden met bijvoorbeeld C# of Dart.

1.2 Onderzoeksvraag

Als de beslissing genomen wordt om een platformonafhankelijke applicatie te bouwen, dan kan dit zowel met de hybride technologie als met de *cross-platform* technologie. Beide oplossingen hebben hun eigen manier om applicaties, geschreven in een gekozen taal, te vertalen naar een *native* applicatie. Vervolgens ontstaat er de vraag over de achterliggende technieken die gebruikt worden om dit proces te realiseren en in welke maten deze verschillen van elkaar.

Een bekend framework dat gebruik maakt van de *cross-platform* technologie is Flutter. Verder bestaat er ook het framework Ionic dat de hybride technologie benut. Beide oplossingen kunnen gebruikt worden bij de creatie van platformonafhankelijke *native* applicaties.

In dit onderzoek wordt er onderzocht in welke mate Flutter en Ionic verschillen in hun aanbod van platformonafhankelijke *native* applicaties..

1.3 Relevantie stageopdracht

In de stageopdracht wordt er gewerkt met Flutter. Uit de opdracht volgt een PoC die als basis gebruikt kan worden voor toekomstige uitbreidingen. Het bedrijf heeft momenteel geen toegewijd team voor dit framework. De ontwikkelaars evolueren mee met de snel veranderende IT-wereld.

Na de stageopdracht wordt er besloten of de applicatie verder ontwikkeld wordt. Voordat de ontwikkelaars hun tijd investeren in het leerproces van het gekozen framework, is het belangrijk om te weten of de gekozen technologie wel de juiste keuze is voor dit project.

Het eerste deel van dit onderzoek toont aan hoe beide frameworks achter de schermen verschillen, omdat ze uiteindelijk hetzelfde opleveren, een platformonafhankelijke *native* applicatie. Verder wordt er ook gefocust op de voor- en nadelen die deze verschillen aanbieden. Deze resultaten kunnen gebruikt worden om een beter beeld te creëren van beide keuzes.

Tenslotte wordt er in dit onderzoek ook gewerkt aan zowel een PoC gemaakt met Flutter als een PoC gemaakt met Ionic. Hierdoor kan een standaard Flutter-applicatie vergeleken worden met een Ionic-applicatie op basis van gebruik en performantie. Verder wordt er aandacht besteed aan factoren zoals de leercurve, de beschikbare documentatie, de populariteit en nog andere nuttige informatie van beide technologieën waardoor de ontwikkelaars weten wat er verwacht kan worden.

2 Methode van onderzoek

Eerst en vooral worden de technische kenmerken van beide frameworks met elkaar vergeleken. Flutter is namelijk een voorbeeld van een *cross-platform* technologie, terwijl Ionic het toelaat om hybride applicaties te bouwen. Beide opties zorgen uiteindelijk voor een *native* applicatie die op meerdere besturingssystemen draait. In het eerste deel van het onderzoek worden de achterliggende technologieën van beide mogelijkheden onderzocht. Eens deze verschillen duidelijker zijn, worden de voor- en nadelen die deze verschillen aanbieden onderzocht. Dit gebeurt aan de hand van een literatuurstudie waarbij er verschillende bronnen geraadpleegd worden.

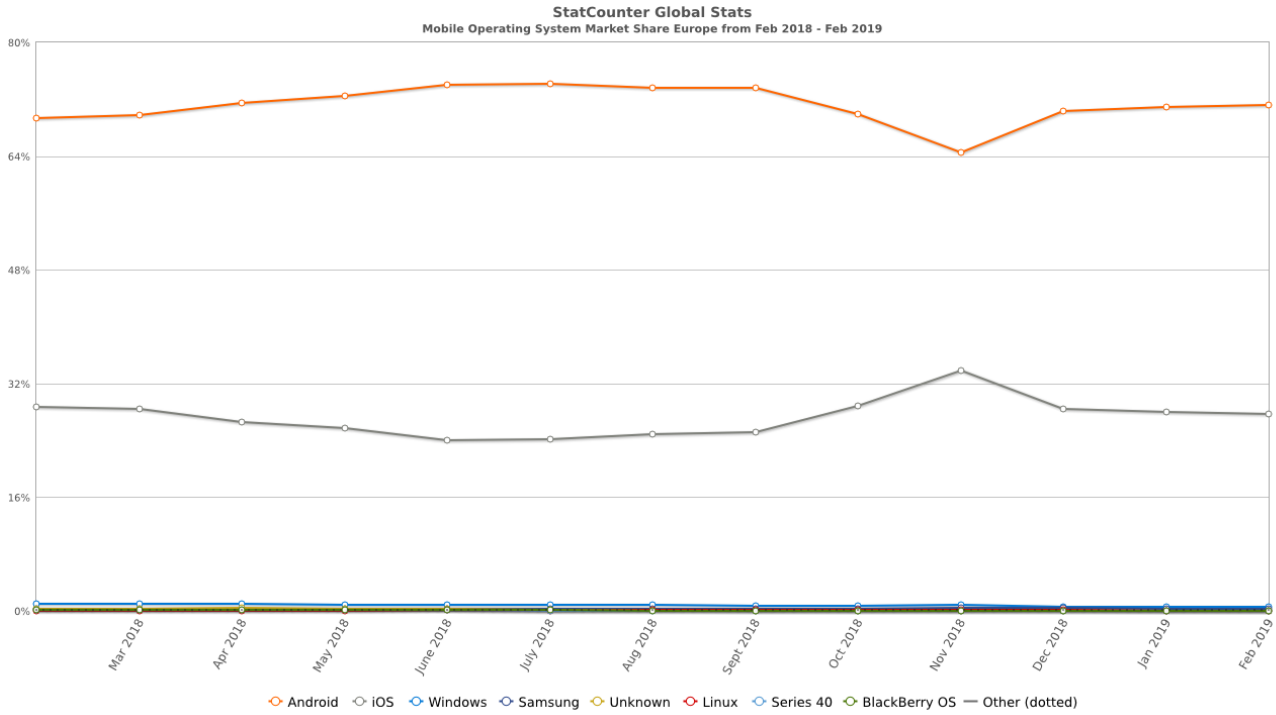
Vervolgens worden de conclusies uit de literatuurstudie op de proef gesteld aan de hand van een PoC. Er wordt een applicatie gebouwd die enkele functionaliteiten bevat die gebruikt worden om

verschillende aspecten van de applicatie te testen. Uiteindelijk worden de opvattingen uit de literatuurstudie kritisch vergeleken met mijn eigen ondervindingen.

3 Resultaten

3.1 *Native* ontwikkeling

Een *native* applicatie is de eenvoudigste en meest voorkomende vorm van een mobiele applicatie. Het is een standaard mobiele applicatie die gebouwd wordt voor een specifiek besturingssysteem. De verschillende mobiele besturingssystemen kunnen beperkt worden tot de twee hoofdspelers Android en iOS. Figuur 15 toont aan dat deze twee besturingssystemen gezamenlijk bijna de volledige mobiele markt van het voorbije jaar in handen hadden. [4]



Figuur 15 Marktaandeel van de beschikbare mobiele besturingssystemen van het voorbije jaar

Hierdoor wordt er vaak gesproken over een Android- of iOS-app. Voor de verschillende besturingssystemen bestaan er ook verschillende ontwikkelingsomgevingen en programmeertalen. Zowel Apple als Google voorziet een eigen IDE, SDK en set van ontwikkelingshulpmiddelen.

iOS-applicaties kunnen bijvoorbeeld gemaakt worden met de programmeertalen Objective-C en Swift, terwijl Android applicaties gebruik maken van Java of Kotlin. De applicatie is beschikbaar via de 'Google Play store' (Android) of de 'App store' (iOS) en is toegankelijk op het startscherm van het mobiel toestel.

Omdat deze manier van ontwikkelen de technische en gebruiksvriendelijke richtlijnen van het gekozen besturingssysteem volgt, biedt de applicatie een natuurlijk gevoel aan voor de gebruiker. Met dit natuurlijk gevoel wordt bedoeld dat het uitzicht en het gebruik van de applicatie consistent is tegenover andere *native* applicaties op het toestel. De eindgebruiker is uiteindelijk sneller op weg met de navigatie en het algemeen gebruik van de applicatie.

3.2 Platformonafhankelijkheid

Naast het natuurlijk gevoel dat een *native* applicatie aanbiedt, bestaan er nog tal van voordelen. Langs de andere kant zijn er natuurlijk ook nadelen verbonden aan *native* ontwikkeling waarvan de grootste is dat de applicatie voor een specifiek besturingssysteem gebouwd moet worden. Zoals eerder vermeld, hebben de twee grootste besturingssystemen, Android en iOS, de markt bijna volledig in handen. Als een *native* SDK wordt gebruikt moet de keuze gemaakt worden tussen Android of iOS, waardoor een grote groep van eindgebruikers de applicatie niet kan gebruiken. De applicatie kan ook tweemaal gebouwd worden, maar hiervoor zijn er meer resources nodig die in de meeste gevallen niet aanwezig zijn.

De nood voor mobiele applicaties die eenmaal gebouwd worden en vervolgens op zowel Android en iOS draaien, is de reden voor de opkomst van diverse platformonafhankelijke ontwikkelingstechnologieën. De platformonafhankelijke technologieën werden tot voor kort vaak bekritiseerd, omdat er redelijk wat consequenties aan vasthingen. De performantie van platformonafhankelijke applicaties was niet optimaal. Verder ging er meestal meer tijd verloren door bugs op te lossen dan als de applicatie tweemaal gebouwd werd voor beide besturingssystemen.

Ondertussen is de vraag naar platformonafhankelijkheid nog steeds krachtig en heeft de technologie genoeg tijd gehad om zichzelf te verbeteren. Technologieën zoals Flutter en Ionic zorgen voor mobiele applicaties die via dezelfde *codebase* op zowel Android als iOS draaien. De technologieën zijn uiteraard nog niet perfect waardoor er nog steeds nadelen blijven bestaan.

Een gedetailleerde vergelijking tussen de voor- en nadelen van het gebruik van een *native* SDK tegenover een platformonafhankelijke SDK is te zien in tabel 1.

Kenmerk	*Native*	Platformonafhankelijk
Kosten	Meerdere ontwikkelaars nodig met de juiste kennis dus een verhoogde kost.	Eén ontwikkelingsteam nodig wat voor een gereduceerde kost zorgt.
Ontwikkelingstijd	Hoog als er gekozen wordt voor een applicatie die zowel op Android als iOS draait.	Laag omdat er maar één *codebase* is.
Doelpubliek	Als er maar één applicatie gebouwd wordt zal een keuze gemaakt moeten worden tussen Android of iOS, waardoor tot 50% van de eindgebruikers kunnen wegvallen.	Laat het toe om met één applicatie een veel grotere doelpubliek aan te spreken.
Uitdagingen	Makkelijk ontwikkelingsproces omdat de technische richtlijnen van het besturingssysteem worden opgevolgd.	Sommige functionaliteiten eisen een uitgebreidere technische kennis.
Expertise	Moeilijker om experts van de verschillende besturingssystemen te vinden die om kunnen gaan met de aangeboden ontwikkelingsomgevingen.	De meeste platformonafhankelijke mogelijkheden zijn gebaseerd op een bekende programmeertaal waardoor er makkelijker experts gevonden kunnen worden.

Tabel 1 Vergelijking native en platformonafhankelijke applicaties

Welke keuze nu de beste is, hangt af van diverse factoren die gebonden zijn aan het project. Omdat verschillende projecten verschillende eisen hebben is het moeilijk om een algemeen besluit te vormen over de beste mogelijkheid. Uit de gegevens van tabel 1 kan er wel geconcludeerd worden dat platformonafhankelijke technologieën zeker het overwegen waard zijn in veel gevallen. [5]

De twee frameworks die in dit onderzoek besproken worden maken beide gebruik van platformonafhankelijkheid. Een volgende vraag die gesteld kan worden is op welke manier platformonafhankelijkheid gerealiseerd wordt.

Flutter wordt gezien als een *cross-platform* technologie terwijl Ionic gezien wordt als een hybride technologie. Beide technologieën zorgen voor platformonafhankelijke applicaties op een andere manier.

Cross-platform applicaties worden gebouwd aan de hand van een tussenliggende taal zoals Dart die niet *native* is tegenover het besturingssysteem van het toestel. Dit zorgt ervoor dat een groot deel of

zelfs alles van de *codebase* gebruikt kan worden over verschillende besturingssystemen heen. Bij *cross-platform* technologieën wordt de geschreven code vertaald naar code die wel *native* is tegenover het besturingssysteem. Als de applicatie geïnstalleerd wordt op een Android toestel dan zal de code ter gevolg dus vertaald worden naar *native* Android code en hetzelfde gebeurt op een iOS toestel.

De manier waarop de code precies vertaald wordt hangt af van het gekozen framework. Er kan wel besloten worden dat de code die op het toestel draait honderd procent *native* is, waardoor de UI even snel werkt als bij een pure *native* applicatie.

Een hybride applicatie is een mix tussen enerzijds een webapplicatie en anderzijds een *native* applicatie. Het voelt aan als een *native* applicatie maar gebruikt achterliggend web technologieën zoals HTML5 en Javascript voor de opbouw van de applicatie. Dit gebeurt door de applicatie in te pakken in een *native* wrapper of container.

Een artikel uitgebracht door het Ionic-framework, geschreven door Chris Griffith, beschrijft de technische kenmerken van hybride ontwikkeling in meer detail. [6]

Het artikel vermeld dat de applicatie geschreven wordt zoals een standaard webapplicatie met HTML, CSS en Javascript. De applicatie wordt echter niet getoond in de browser van het toestel maar binnen een *native* applicatie die verantwoordelijk is voor een eigen *embedded* browser die praktisch onzichtbaar is voor de eindgebruiker.

Een iOS applicatie maakt bijvoorbeeld gebruik van 'WKWebView' om een hybride applicatie te tonen, terwijl Android beroep doet op 'WebView' voor dezelfde functionaliteit. Dit zijn beide technologieën die het mogelijk maken om web data in een mobiele applicatie weer te geven zonder de overschakeling te moeten maken naar een browser.

De code wordt vervolgens verwerkt door een *native* wrapper zoals Apache Cordova of Ionic Capacitor. Met deze technologieën worden *native shell* applicaties gecreëerd die simpelweg de vernoemde *webview*-technologieën, 'WKWebview' en 'WebView', gebruiken om de applicatie in te laden. Op deze manier kunnen *native* applicaties gebouwd worden die ook toegankelijk zijn via zowel de 'App store' als de 'Google Play store'.

3.3 Flutter

De officiële documentatie van Flutter biedt een zeer uitgebreide FAQ aan, waarin onder andere wordt uitgelegd wat Flutter precies is en hoe het achter de schermen werkt. [7] Flutter is een open source mobiele SDK die gebruikt wordt om kwaliteitsvolle *native* applicaties te bouwen. Dit zijn applicaties die gebruikt kunnen worden op zowel Android als iOS. Het framework is uitgebracht door Google en wordt gezamenlijk ondersteunt door Google en de Flutter-*community*.

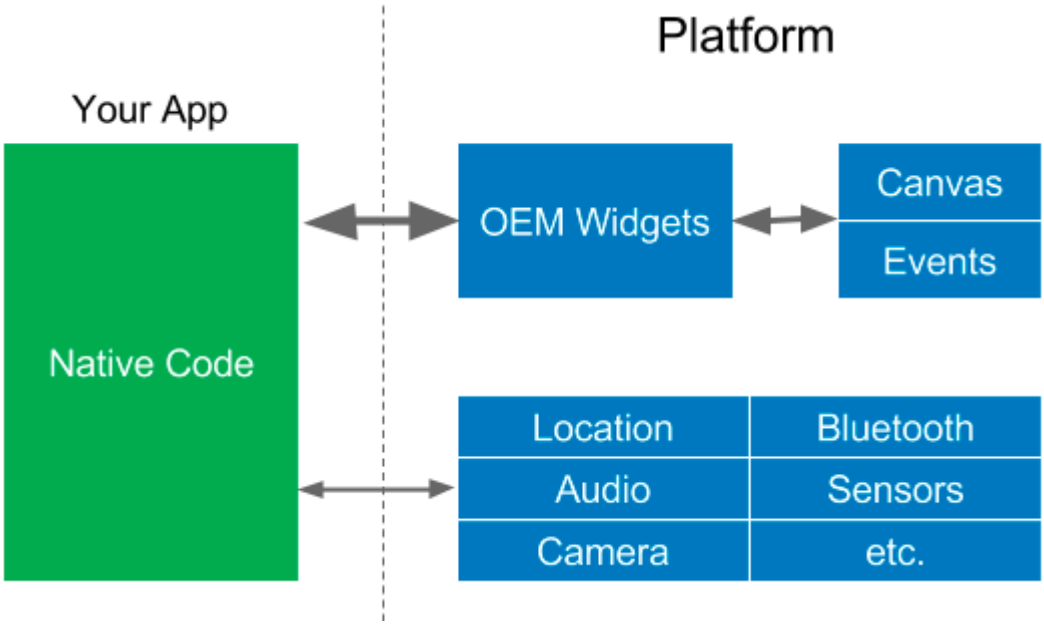
Flutter maakt gebruik van de programmeertaal Dart en heeft hier zijn redenen voor. In de FAQ vermeldt het team van Flutter dat de programmeertaal Dart, ook uitgebracht door Google, het hoogste scoort in verband met hun eisen en criteria. De *runtime*-omgevingen en compilers die Dart aanbieden ondersteunen twee van de kritische functionaliteiten waarop Flutter gebaseerd is. Het eerste is een JIT-gebaseerde ontwikkelingscyclus die het mogelijk maakt om *stateful hot reloading* uit te voeren in een getypeerde programmeertaal. Het tweede is een AOT-compiler die efficiënte ARM-code genereert. Verder heeft het team van Flutter de mogelijkheid om nauw samen te werken met Dart omdat ze beide uitgebracht zijn door hetzelfde bedrijf. Hierdoor investeert Dart resources om het gebruik in het Flutter-framework te optimaliseren.

De zelf geschreven Dart-code wordt gezamenlijk met de SDK-code AOT gecompileerd in *native*-, ARM- en x86-*libraries*. Deze *libraries* worden inbegrepen in een Android- en/of iOS-project. Vervolgens wordt alles bijeen omgevormd tot een APK voor Android en/of een IPA voor iOS. Wanneer de applicatie opstart, wordt eerst en vooral de *library* ingeladen. Al het *render* werk, de input en de events worden gedelegeerd naar de gecompileerde code. Als de applicatie in debug mode gebouwd wordt, wordt er een VM gebruikt die de Dart code zal draaien om *stateful hot reloading* te ondersteunen.

Flutter is anders dan de meeste mogelijkheden om mobiele applicaties te bouwen. Het maakt geen gebruik van de technologie rond *WebView* zoals vermeld in hybride ontwikkeling. Verder gebruikt het ook geen OEM-widgets om de UI op te bouwen. In plaats hiervan heeft Flutter een eigen *rendering engine* om widgets te tekenen. Er wordt dus een eigen set van widgets aangeboden zoals Material Design en Cupertino die onderhouden worden door het framework.

Flutter heeft gekozen om niet te werken met de OEM-widgets met het vooruitzicht op applicaties van hogere kwaliteit. Als de OEM-widgets worden hergebruikt, dan zou de kwaliteit en performantie van Flutter-applicaties afhankelijk zijn van de kwaliteit van de gebruikte widgets. Android biedt bijvoorbeeld een vaste set van *gestures* aan. Als de OEM-widgets gebruikt worden zit Flutter vastgebonden aan deze set, maar nu is het mogelijk om met Flutter een eigen *gesture* herkenner te schrijven.

In het artikel 'What's Revolutionary about Flutter' geeft Wm Leler, Senior Developer bij Google, een beter overzicht tussen het gebruik van OEM-widgets en de Flutter-structuur. [8] Normaal gezien communiceert een applicatie met het platform om widgets te creëren en services zoals de camera aan te spreken. De widgets worden gerenderd op een canvas en events worden doorgestuurd naar de widgets. Dit is een simpele structuur maar er moeten nog steeds twee verschillende applicaties gebouwd worden omdat ieder platform een eigen set van widgets aanbiedt. Figuur 16 geeft de besproken structuur visueel weer. [8]

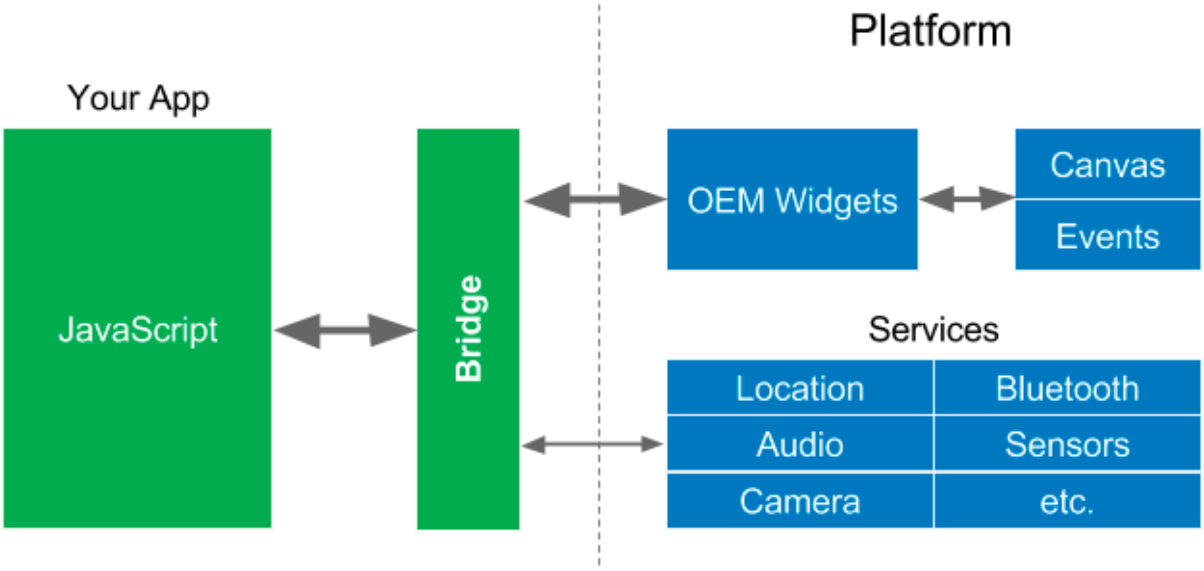


Figuur 16 Overzicht mobiele applicatiestructuur

Een volgende mogelijkheid zijn Reactive Views. Enkele populaire concurrenten van Flutter, zoals React Native, maken gebruik van een *bridge* om de *native* componenten van het platform aan te spreken. Dit is een tool die de communicatie tussen het platform en de code voor zich neemt. Een

nadeel van een *bridge* is dat er een verlies aan performantie ontstaat. Widgets kunnen tot zestig keer per seconde aangesproken worden gedurende een animatie waardoor de bridge zeer veel werk moet verrichten.

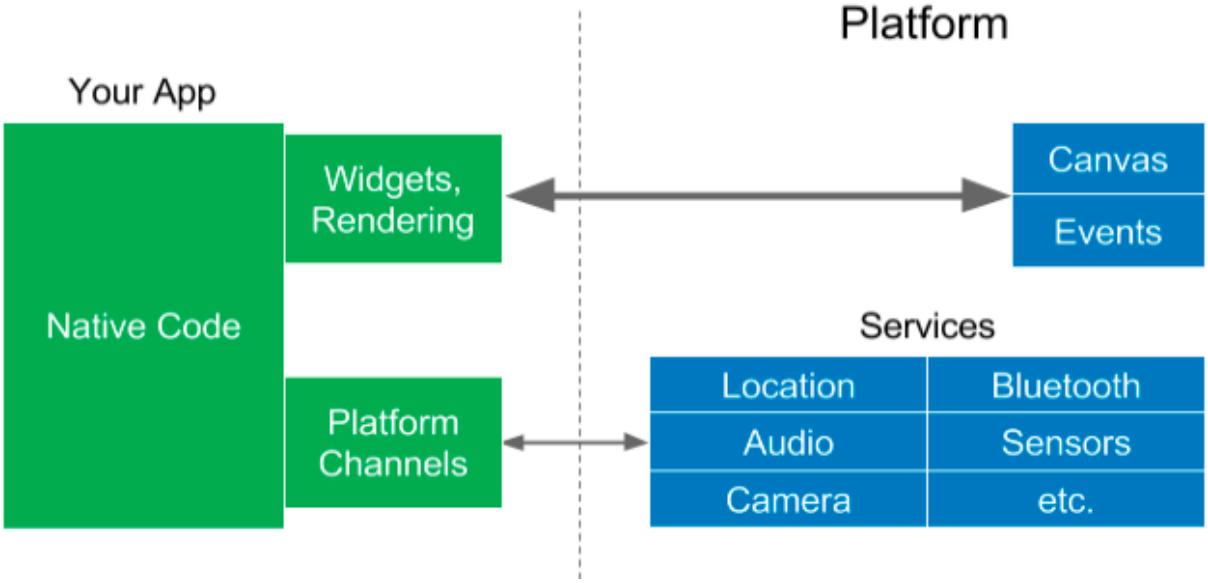
In figuur 17 wordt de aangepaste applicatiestructuur getoond die gebruik maakt van een *bridge*. [8]



Figuur 17 Reactive Views applicatiestructuur

Zoals React Native maakt ook Flutter gebruik van *reactive-style* views. Flutter doet dit wel op een andere manier om het verlies aan performantie aan te pakken. De programmeertaal Dart, waar Flutter gebruik van maakt, wordt AOT gecompileerd in *native* code voor meerdere platformen. Dit zorgt ervoor dat Flutter kan communiceren met het platform zonder hulp van een JavaScript *bridge* die moet switchen tussen contexten. Verder wordt de opstartsnelheid van de applicatie ook verbetert op deze manier. Flutter is momenteel de enige mobiele SDK die *reactive-style* views aanbiedt zonder gebruik te maken van een *bridge*.

Flutter heeft dus een eigen architectuur waarin widgets inbegrepen zijn die snel, aanpasbaar en uitbreidbaar zijn. Deze architectuur is te zien in figuur 18. [8]



Figuur 18 Flutter reactive-style views architectuur

De widgets en de functionaliteit om ze te renderen worden opgenomen in de applicatie in plaats van in het platform. Het enige dat Flutter nodig heeft van het platform is een canvas en een manier om events en services aan te spreken. Een nadeel van dit systeem is dat de applicatiegrootte groter zal zijn dan normaal omdat Flutter de widgets zelf voorziet. Hierdoor is de minimumgrootte van een Android applicatie ongeveer 4,7 MB.

3.4 Ionic

Zoals eerder vermeld, maakt Ionic gebruik van hybride ontwikkeling waarbij web technologieën zoals HTML5, CSS en Javascript worden gebruikt om een mobiele applicatie op te bouwen. De geschreven code wordt vervolgens getoond in een *web view* van een *native* applicatie. Er bestaan verschillende frameworks rond de hybride technologie. In dit onderdeel wordt er besproken hoe het Ionic-framework de hybride manier toepast om mobiele applicaties te bouwen. Net zoals Flutter heeft Ionic ook een eigen uitgebreide documentatie waarin onder andere beschreven staat hoe Ionic-applicaties opgebouwd zijn en hoe ze verschillen van andere hybride frameworks. [9]

In het gedeelte 'What is Ionic Framework?' wordt besproken wat het framework effectief aanbiedt. Het framework wordt beschreven als een open source UI toolkit die het toelaat om performante, kwaliteitsvolle mobiele en desktop applicaties te bouwen die gebruik maken van de web technologieën. Ionic focust op de front-end of de UI interactie van een applicatie. Momenteel biedt Ionic een officiële integratie aan met het Javascript framework Angular. Angular heeft altijd al een grote rol gespeeld binnen Ionic maar vanaf Ionic V4 zijn er enkele grote aanpassingen doorgevoerd in de onderliggende technologie en mogelijkheden van Ionic. Hoewel V4 nog steeds een sterke focus heeft op de integratie met Angular, is het nu ook mogelijk om te werken met andere Javascript frameworks zoals Vue, React of helemaal geen framework.

In het volgend deel genaamd 'Core Concepts' wordt er een overzicht gegeven van de kernfilosofie, concepten en hulpmiddelen van Ionic. Het framework biedt een *library* van UI componenten aan. Dit zijn herbruikbare elementen, geschreven met HTML, CSS en Javascript, die gebruikt kunnen worden voor de opbouw van een applicatie. Hoewel dit vooraf gemaakte componenten zijn, is het mogelijk om de componenten aan te passen en te personaliseren zodat ze gebruikt kunnen worden in applicaties met verschillende stijlen. Verder wordt er een concept genaamd 'Platform Continuity' gehanteerd binnen Ionic dat ervoor zorgt dat dezelfde *codebase* gebruikt kan worden op meerdere platformen. Iedere Ionic-component zal zijn visuele representatie aanpassen naargelang het platform, bijvoorbeeld iOS, waarop de applicatie draait.

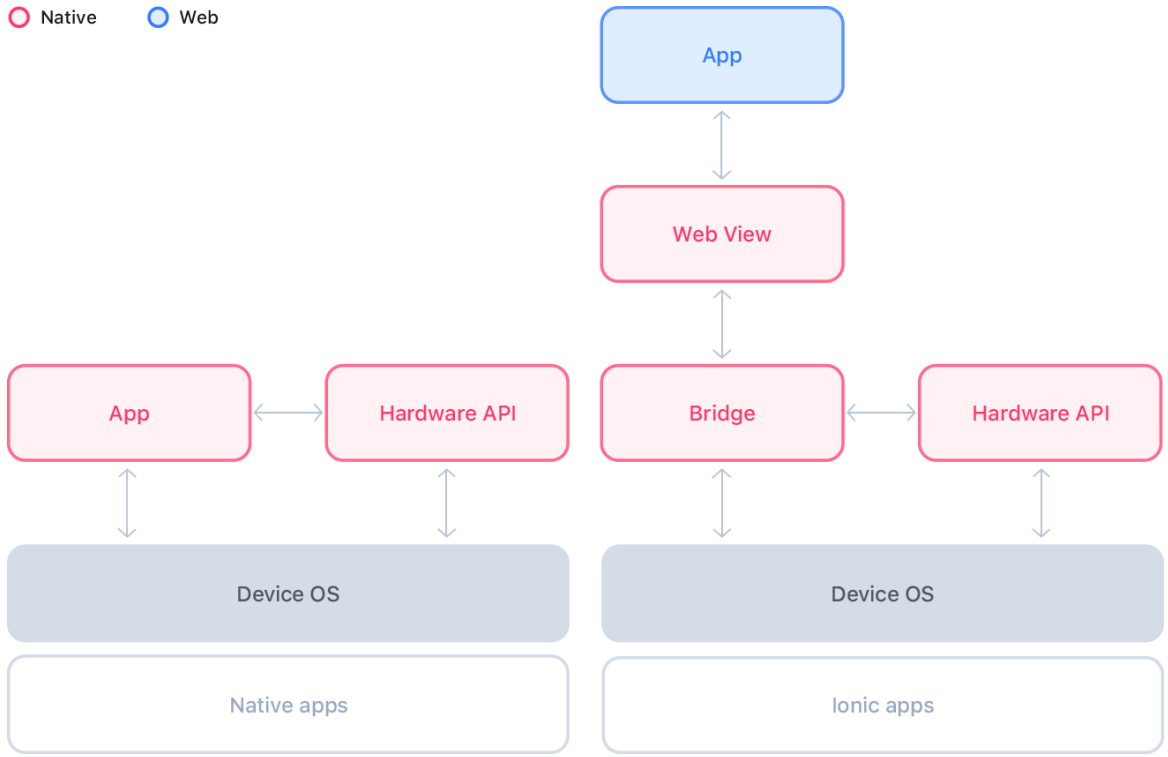
Traditionele webapplicaties gebruiken een lineaire geschiedenis voor de navigatie. Dit wil zeggen dat een gebruiker naar een pagina kan navigeren en vervolgens met de terug knop terug kan navigeren naar de oorspronkelijke pagina. In tegenstelling tot de lineaire geschiedenis utiliseren mobiele applicaties vaak een parallelle vorm van navigatie. Een tabblad navigatiestructuur maakt bijvoorbeeld gebruik van verschillende navigatiestapels voor elk tabblad zodat de gebruiker nooit zijn of haar plek verliest tijdens het navigeren en wisselen tussen tabbladen. Ionic-applicaties gebruiken deze manier van mobiele navigatie en ondersteunen dus parallelle navigatiegeschiedenissen die ook kunnen worden genest.

Applicaties die gebouwd worden met de web technologieën, zoals Ionic-applicaties, hebben het grote voordeel dat ze vrijwel op elk platform kunnen worden uitgevoerd. Zoals eerder vermeld kan dezelfde *codebase* gebruikt worden op meerdere platformen omdat deze applicaties gebaseerd zijn op de web standaarden en APIs die gedeeld worden ondereen de verschillende platformen. Dit systeem wordt ook gebruikt om een applicatie te bouwen die gedownload kan worden op een iOS-

en/of Android-toestel. Zowel iOS als Android biedt een *web view* aan die gebruikt kan worden om een Ionic-applicatie te tonen terwijl het nog steeds mogelijk is om de *native* SDK aan te spreken door middel van technologieën zoals Capacitor of Cordova.

In het gedeelte 'Web View' wordt verder uitgelegd hoe de *web views* achter de schermen werken. De *web views* die gebruikt worden om Ionic-applicaties te renderen zijn *full screen* webbrowsers. Veel moderne *web views* bieden ingebouwde HTML5 APIs aan voor hardware functionaliteiten zoals camera, sensoren, GPS, speakers en bluetooth maar soms is het ook nodig om platform specifieke hardware APIs aan te spreken. Binnen Ionic-applicaties kunnen hardware APIs, die meestal gebruik maken van *native* plug-ins, aangesproken worden via een *bridge* laag.

Een overzicht van dit systeem is te zien in figuur 19. [9]



Figuur 19 Web view structuur Ionic-applicatie

De Ionic *web view* plug-in is gespecialiseerd voor moderne Javascript applicaties. Voor zowel iOS als Android worden bestanden altijd gehost door gebruik te maken van het HTTP-protocol met een geoptimaliseerde HTTP-server die draait op het lokale toestel.

3.5 Vergelijking

Nu dat het duidelijker is hoe Flutter en Ionic achter de schermen platformonafhankelijke mobiele applicaties aanbieden, worden enkele voor-en nadelen van deze verschillen vergeleken aan de hand van diverse bronnen. Omdat Flutter redelijk nieuw op de markt is, zijn er momenteel niet veel onderzoeken te vinden die Flutter vergelijken met Ionic aan de hand van effectieve meetresultaten. Daarom worden er bronnen geraadpleegd die beide frameworks vergelijken aan de hand van enkele subjectieve opvattingen. In een volgend deel van dit onderzoek worden een aantal van deze opvattingen samen met nog nieuwe aspecten op de proef gesteld door zelf te experimenteren met beide frameworks.

In een artikel geschreven door professionele web ontwikkelaar Maximilian Schwarzmüller worden enkele platformonafhankelijke technologieën met elkaar vergeleken waaronder Flutter en Ionic. [10]

Een eerste factor waarop Schwarzmüller beide frameworks evalueert is het *"Write once, use everywhere"* principe. Hiermee wordt er gekeken naar de herbruikbaarheid van de geschreven code. Hoewel beide frameworks gebaseerd zijn op platformonafhankelijkheid kan het toch voorkomen dat aangepaste code geschreven moet worden naargelang het actieve platform. Het kan bijvoorbeeld voorkomen dat verschillende UI-componenten gebruikt moeten worden omdat niet alle UI-elementen bestaan op beide platformen.

Standaard webapplicaties zien er hetzelfde uit op Android en iOS en omdat Ionic achterliggend gebruik maakt van een webapplicatie scoort het op dit vlak zeer goed. Hiernaast maakt Ionic ook gebruik van componenten die automatisch aangepast worden naargelang het platform waarop de applicatie draait.

Flutter scoort op dit vlak ook goed maar net iets minder dan Ionic. Dit komt omdat Flutter widgets gebruikt die niet automatisch aangepast worden aan de hand van het platform. Als dit nodig is zullen de widgets manueel aangepast moeten worden, wat natuurlijk extra werk vereist.

Het tweede vergelijkingspunt gaat over de voorgedefinieerde componenten die aangeboden worden op beide frameworks. Hier gaat het over de moeilijkheidsgraad om een prachtige UI te bouwen en als het nodig is om de componenten zelf te stylen.

De kern van Ionic is een grote set van voorgedefinieerde en gestylde componenten. Zoals eerder vermeld, worden deze componenten ook nog eens automatisch aangepast naargelang het platform. Hierdoor maakt Ionic het mogelijk om prachtige, *native* applicaties te bouwen met een kleine moeite.

Flutter scoort ook goed maar nogmaals iets minder dan Ionic op dit vergelijkingspunt. Flutter biedt ook een uitgebreid pakket aan van ingebouwde widgets. Met deze widgets kunnen snel mooie gebruikersinterfaces gebouwd worden zonder al te veel handmatige styling. Alleen als er verschillende looks voor verschillende platformen nodig zijn kan enige inspanning vereist zijn.

Het derde punt bespreekt het ecosysteem van beide frameworks. Hieronder vallen aspecten zoals het aantal beschikbare *third-party libraries*, de mogelijkheid om makkelijk hulp te vinden op platformen zoals Stackoverflow en de populariteit.

Ionic gebruikt Javascript voor de logica van de applicatie en hoewel het mogelijk is om diverse of zelfs geen Javascript framework te gebruiken, biedt Ionic een zeer goede integratie met Angular aan. Hierdoor kan een ontwikkelaar profiteren van al deze ecosystemen. Zowel het Javascript als het Angular ecosysteem is gigantisch groot met enorm veel vragen en antwoorden op Stackoverflow. Zo zijn er op dit moment 1 795 807 vragen gesteld op Stackoverflow met de Javascript tag en 162 933 met een Angular tag. Verder kan een Ionic-applicatie ook gebruik maken van het uitgebreid aanbod aan npm-pakketten voor het uitbreiden van de functionaliteiten van de applicatie. Voor de populariteit wordt beroep gedaan op het aantal GitHub sterren dat het framework verzameld heeft. De officiële GitHub *repository* van Ionic bevat momenteel 37 731 sterren.

Bij Flutter gaat het er op dit moment wat rustiger aan toe maar dit komt omdat dit framework nog relatief nieuw is. Flutter is wel redelijk trending en heeft ondertussen al 14 340 vragen op Stackoverflow. Tenslotte wordt Flutter sterk geadverteerd door de uitgever Google die veel aandacht geeft aan dit framework. Hierdoor is Flutter geliefd door ontwikkelaars en heeft het framework op dit moment 60 399 GitHub sterren verzameld op hun officiële *repository*.

Het vierde punt gaat het verschil in performantie van applicaties ontwikkeld met Ionic en Flutter vergelijken. Hier scoort Ionic het slechtst omdat het uiteindelijk een gewrapte webapplicatie oplevert. Maar omdat een Ionic-applicatie slechter presteert dan een Flutter-applicatie betekent het niet dat de performantie uiteindelijk slecht is. Met de moderne smartphones van vandaag valt het verschil in performantie nauwelijks of helemaal niet op en zal een Ionic-applicatie zeer snel reageren. Flutter biedt volledig *native* applicaties aan waardoor de Flutter-applicaties een zeer goede performantie kennen.

Het vijfde vergelijkingspunt gaat over de mogelijkheid om *native* functionaliteiten aan te spreken zoals de camera of de GPS van het toestel. Ionic gebruikt oplossingen zoals Cordova of Capacitor om *native* functionaliteiten aan te spreken. Deze oplossingen zorgen voor een degelijke set van *packages* om bijvoorbeeld de camera of de GPS aan te spreken. Bij Flutter is het bijna een noodzaak om gebruik te maken van *third-party packages*. In de meeste gevallen zijn deze *packages* voorzien maar omdat het ecosysteem van Flutter nog nieuw en opkomend is kan het mogelijk zijn dat er voor enkele *native* functionaliteiten geen *packages* bestaan. Verder is een applicatie dan steeds afhankelijk van de uitgever van de *package* wat ervoor kan zorgen dat een *package* op een gegeven moment niet meer ondersteund wordt.

De argumenten en vergelijkingspunten die besproken worden in een artikel uitgebracht op Apptunix komen grotendeels overeen met de argumenten van het eerste artikel, maar toch zijn er enkele meningsverschillen. [11]

Naast de besproken vergelijkingspunten in het eerste artikel geeft dit artikel ook nog een inzicht over de gebruikte programmeertalen en de eventuele kosten van Flutter- en Ionic-applicaties. Er wordt besproken hoe Flutter gebruik maakt van Dart, een programmeertaal die naar de toekomst toe zeer succesvol kan worden. Flutter en Ionic zijn beide open source frameworks maar Ionic biedt ook een betaalde pro versie aan die het ontwikkelingsproces zou moeten versnellen.

Verder beschrijft dit artikel Flutter als de ultieme winnaar als het gaat over performantie. Het feit dat Ionic de web technologieën op een *native* manier gebruikt zou de performantie sterk negatief beïnvloeden. Flutter zou langs de andere kant de beste performantie aanbieden in vergelijking met Ionic en zelfs met veel andere platformonafhankelijke technologieën. Dit omdat Flutter geen gebruik maakt van een Javascript *bridge* om de *native* componenten aan te spreken.

Een derde artikel uitgebracht door newgenapps vergelijkt ook platformonafhankelijke technologieën waaronder Flutter en Ionic. In dit artikel zijn er geen opvallende afwijkingen te vinden in de vergelijking tussen beide frameworks. De aangehaalde argumenten en inzichten komen dus overeen met de andere twee besproken artikels. [12]

Al de besproken artikels concluderen hetzelfde. Algemeen bekeken is er geen slechte keuze en zijn Flutter en Ionic beide goede alternatieven waardoor de keuze die gemaakt moet worden projectgebonden is. Naargelang de vereisten van een bepaald project moeten de aangehaalde verschillen tussen Flutter en Ionic geëvalueerd worden. Als de meeste ontwikkelaars van een team bijvoorbeeld web experts zijn en performantie niet van hoogste belang is, dan kan Ionic een betere optie zijn. Als performantie wel van hoogste belang is, zou Flutter gebruikt kunnen worden. Dit zijn uiteraard maar enkele simpele voorbeelden om te tonen hoe het denkproces uitgevoerd kan worden.

3.6 Persoonlijke reflectie

Persoonlijk vind ik de manier waarop Flutter volledige *native* applicaties produceert een zeer interessante optie met veel potentie. Het feit dat Flutter niet afhankelijk is van een *bridge* zorgt ervoor dat dit framework zeker het overwegen waard is. Dat gezegd zijnde, is het wel nog steeds een risico om resources te gaan investeren in Flutter. Het framework wordt ondersteund door Google maar zit wel nog in een vroege fase. Een eerste stabiele versie van het framework is op dit moment, 19 april 2019, ongeveer 5 maanden uit. Dit wil dus zeggen dat het mogelijk is dat het framework nog grote bugs kan bevatten of dat er nog grote veranderingen gepland staan die het ontwikkelingsproces volledig kunnen veranderen. Flutter maakt verder ook gebruik van de programmeertaal Dart die niet super populair is. De kans is dus groot dat ontwikkelaars ook tijd moeten gaan investeren in het leerproces van Dart. Dit is nogmaals een bijgevoegd risico want als Flutter een hype blijkt te zijn en Dart is nog steeds niet geliefd bij ontwikkelaars, dan is het mogelijk dat de geïnvesteerde tijd in het leerproces van Dart ook verloren gaat. Langs de andere kant zijn er veel meer ontwikkelaars met kennis van de web technologieën die meteen aan de slag kunnen gaan aan de opbouw van Ionic-applicaties. Omdat Ionic al een langere tijd bestaat en gebruik maakt van de enorme ecosystemen van Javascript en Angular, is het een stuk makkelijker om op weg te geraken met dit framework. Hier zit dan wel weer het nadeel aan verbonden dat Ionic-applicaties niet 100% *native* zijn, maar ik vind dat dit argument niet doorslaggevend mag zijn.

Omdat het project van de stageopdracht speciale vereisten heeft rond HTML-code, kan Ionic ook een voordeel bieden. De code die gebruikt wordt om de inhoud van de webapplicatie te tonen is opgeslagen in de vorm van HTML. Bij het gebruik van Flutter moeten er grote aanpassingen doorgevoerd worden om dezelfde inhoud in de mobiele applicatie te tonen. Ionic-code wordt geschreven in de vorm van HTML waardoor de bestaande inhoud beter hergebruikt kan worden zonder verplicht te zijn om grote aanpassingen te moeten doorvoeren.

Tenslotte ben ik het grotendeels mee eens met de aangehaalde argumenten van het artikel geschreven door web ontwikkelaar Schwarzmüller. De keuze kan het best gemaakt worden met de besproken vergelijkingspunten bij de hand en met de gedachte dat Flutter eventueel een hype kan zijn met enkele verbonden risico's.

3.7 Proof of Concept

3.7.1 Omgeving configureren

Vooraleer het ontwikkelingsproces van start kan gaan moet er een omgeving geconfigureerd worden. In dit gedeelte wordt er onderzocht hoe snel een omgeving opgezet kan worden en of er veel benodigdheden zijn voor beide frameworks. In dit onderzoek wordt de omgeving geconfigureerd op een Windows besturingssysteem.

3.7.1.1 Flutter-omgeving

Flutter biedt een zeer overzichtelijke handleiding aan om het installatieproces correct uit te voeren. Eerst en vooral moet de Flutter-SDK geïnstalleerd worden op het systeem. Op Windows vereist de Flutter-SDK een 64-bit versie van het besturingssysteem van Windows 7 SP1 of later. Het zip-bestand van de SDK kan gedownload worden op de officiële Flutter-site. In dit bestand is een console te vinden waarin Flutter-commando's uitgevoerd kunnen worden. De Flutter commando's kunnen ook in de standaard console programma's gebruikt worden als de omgevingsvariabelen van het systeem aangepast worden.

In de Flutter console, of eender welke console die Flutter ondersteunt, kan het '*flutter doctor*'-commando uitgevoerd worden. Dit commando geeft een overzicht van de installatie op het huidig systeem met aanwijzingen over de resterende software die nog geïnstalleerd moet worden of taken die nog uitgevoerd moeten worden. De uitleg die dit commando aanbiedt is onmiskenbaar en zorgt voor een eenvoudig installatieproces. Eens het installatieproces voltooid is, kan een project gecreëerd worden. Hiervoor kan een IDE naar keuze gebruikt worden. Voor het PoC in dit onderzoek wordt gebruik gemaakt van Visual Studio Code.

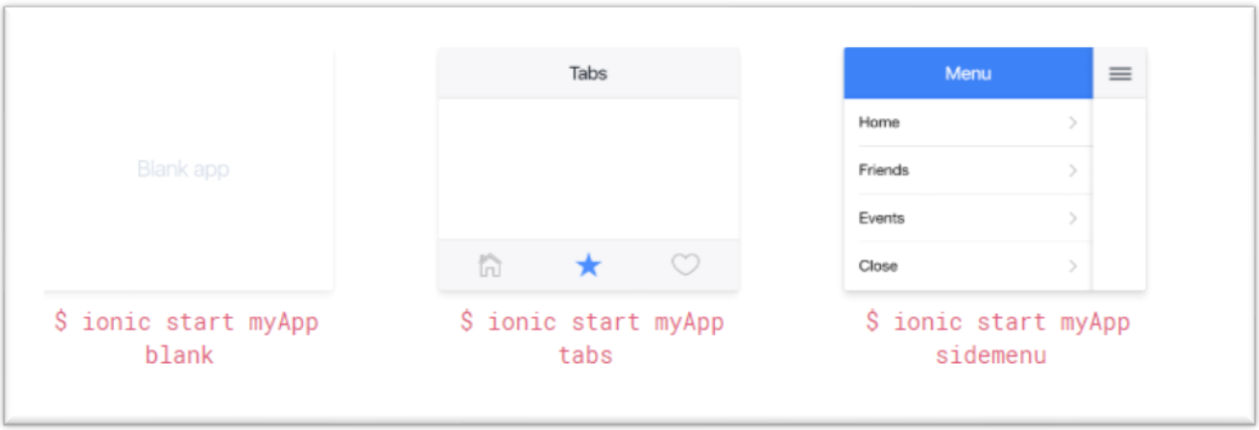
Om Flutter-applicaties te bouwen met de gekozen IDE moeten de Flutter en Dart plug-ins geïnstalleerd worden. In Visual Studio Code kan dit zeer gemakkelijk door te navigeren naar '*extensions*' en te zoeken naar Flutter.

Vervolgens kan een project gecreëerd worden via het '*flutter create myapp*'-commando waarbij '*myapp*' de naam van het project wordt. Hierdoor wordt er een demoapplicatie aangemaakt die geopend kan worden in de gekozen IDE. De code van de demoapplicatie is te vinden in de hoofdklasse '*main.dart*'. De demoapplicatie bevat een simpele applicatie waarin een teller verhoogd en verlaagd kan worden. In de hoofdklasse staan ook handige blokken commentaar met uitleg over de gebruikte code. De applicatie kan gestart worden door het '*flutter run*'-commando uit te voeren. Hiermee is in enkele eenvoudige stappen een Flutter-omgeving gezamenlijk met een demoapplicatie opgezet en kan het ontwikkelingsproces van start gaan.

3.7.1.2 Ionic-omgeving

Een Ionic-omgeving kan op verschillende manieren opgezet worden wat voor enige verwarring zorgt.

Omdat dit framework achterliggend webapplicaties oplevert is de eenvoudigste manier om Ionic-applicaties te bouwen in een browser. Eerst en vooral moet de pakketbeheerder voor JavaScript applicaties Node geïnstalleerd worden. Vervolgens kunnen de *command-line tools* van Ionic geïnstalleerd worden via een npm-commando. Om een Ionic-applicatie te genereren kunnen verschillende templates gebruikt worden. De templates zorgen voor een snelle start en zijn te zien in figuur 20 samen met de bijhorende npm-commando's.



Figuur 20 Ionic-applicatie templates

Eens een project gecreëerd is, kan de applicatie gestart worden in zijn ontwikkelomgeving via het gepaste '*ionic serve*'-commando. Dit gaat ervoor zorgen dat de applicatie in een webbrowser draait. Via bijvoorbeeld de 'Chrome DevTools' kan de browser geschaald worden op mobiel formaat. Als de applicatie *native* functionaliteiten moet aanspreken dan is het natuurlijk ook slim om de applicatie op een effectief Android- of iOS-toestel te testen. Dit kan op twee verschillende manieren. Een eerste

mogelijkheid is om afhankelijk van het gekozen platform de Android of iOS SDK te installeren. Hierna kan de Ionic-applicatie gestart worden op het toestel via een bijhorende npm-commando. Een andere manier is om op het gekozen toestel de 'DevApp'-applicatie te installeren. Dit is een applicatie aangeboden door het team van Ionic, die het mogelijk maakt om zonder een *native* SDK de applicatie te draaien. Eender welke keuze zorgt uiteindelijk voor een ontwikkelomgeving waarin de applicatie verder uitgebouwd kan worden.

3.7.2 Implementatiedetails

De applicatie die gebouwd wordt bevat drie functionaliteiten om zowel het ontwikkelproces als de performantie te evalueren. Eerst en vooral wordt er een scherm gebouwd waarmee het mogelijk is om foto's te creëren. Hiermee wordt de manier waarop *native* functionaliteiten aangesproken worden getest. Het tweede scherm gaat een lijst van 5000 items binnenkrijgen via een API-aanvraag. Er wordt onderzocht hoe de applicaties omgaan met communicatie over het netwerk en ook hoe ze omgaan met grote lijsten van data. Als laatste wordt er een scherm gebouwd waarmee een stopwatch aangestuurd kan worden. Dit zorgt ervoor dat de applicaties getest worden op gebied van performantie.

3.7.2.1 Aanspreken *native* functionaliteit

In het eerste scherm wordt dus een camera geïntegreerd in de applicatie. Foto's kunnen op twee manieren gecreëerd worden. Het is mogelijk om de applicatie te laten communiceren met de geïnstalleerde camera-applicatie van het toestel. Tijdens het maken van de foto neemt de camera-applicatie de voorgrond en als de foto genomen is wordt er terug genavigeerd naar de oorspronkelijke applicatie met de fotogegevens. Een andere manier is om de effectieve camera te gebruiken in de applicatie zonder beroep te moeten doen op een andere applicatie. Dit is de manier waarop de camera gebruikt wordt in het PoC-project omdat de camera op deze manier rechtstreeks aangesproken wordt.

Het team achter Flutter biedt een *package* aan die gebruikt wordt om de camera van het toestel te beheren. Verder is er ook een officiële handleiding beschikbaar in de documentatie omtrent het gebruik van de camera in Flutter. Op basis van de handleiding is deze *native* functionaliteit zonder enig probleem geïmplementeerd.

Voor Ionic wordt er gebruik gemaakt van Cordova om *native* functionaliteiten aan te spreken. Cordova biedt een standaard set van plug-ins aan voor de meest voorkomende functionaliteiten waaronder de camera. Deze officiële plug-in is zeer gebruiksvriendelijk maar maakt gebruik van de geïnstalleerde camera-applicatie waardoor het niet gebruikt wordt in het PoC-project.

Verder is er een andere plug-in te vinden die het mogelijk maakt om de camera in de applicatie te integreren. Omdat dit niet één van de officiële Cordova plug-ins is, is de documentatie een stuk onduidelijker en minder uitgebreid. Hierdoor heeft het langer geduurd om dezelfde functionaliteit te implementeren in de Ionic-applicatie. Bij het gebruik van de gedocumenteerde code werkte de camera niet. Na meerdere bugs te fixen en hulp te zoeken op platformen zoals Stackoverflow en GitHub is het uiteindelijk gelukt om de camera te integreren in de Ionic-applicatie.

3.7.2.2 Lijsten & Data

Het tweede scherm leest 5000 JSON-objecten in over het netwerk via REST. Eens de data verwerkt is wordt het weergeven in een lijst. Op deze manier wordt zowel de mogelijkheid om veel data te verwerken over een netwerk als de performantie van grote lijsten getest.

Voor deze functionaliteit biedt Flutter nogmaals een uitgebreide documentatie aan die het ontwikkelingsproces een stuk eenvoudiger maakt. Voor communicatie over het netwerk wordt de officiële HTTP-*package* gebruikt die het mogelijk maakt om met een simpele syntax te communiceren over het netwerk. Omdat Flutter de *object-oriented* programmeertaal Dart gebruikt is het nodig om de JSON-objecten eerst om te zetten naar zelf gedefinieerde Dart-objecten. Vervolgens worden de objecten getoond in een lijst aan de hand van de 'ListView'-widget. Deze widget bevat een 'builder'-methode die er voor zorgt dat een grote lijst van data dynamisch aangemaakt wordt als de gebruiker doorheen de lijst scrolt. Hierdoor wordt enkel de nodige data gerenderd zodat de applicatie performant blijft.

In het geval van Ionic verloopt het proces in grote lijnen op dezelfde manier. Voor de communicatie over het netwerk zijn er verschillende mogelijkheden. Ionic biedt een eigen *native* HTTP-*package* aan naast de beschikbare Javascript en Angular mogelijkheden. Er wordt aangeraden om de Ionic HTTP-*package* enkel te gebruiken als de minimale voordelen noodzakelijk zijn voor het project, maar dit is in dit project niet het geval waardoor de Angular HTTP-*package* wordt gebruikt. Hierdoor komt de uitgebreide Angular documentatie goed van pas. Dit is een goed voorbeeld van het voordeel waarbij Ionic kan profiteren van het ecosysteem van Angular. Net zoals Flutter biedt Ionic een component aan die grote lijsten van data dynamisch rendert.

3.7.2.3 Stopwatch

De stopwatch wordt gebouwd om factoren zoals het batterij- en geheugenverbruik te meten voor beide applicaties.

In Flutter worden er twee klassen gebruikt voor de stopwatch. De 'Stopwatch'-klasse is een eenvoudige interface om de verstreken tijd te meten. De 'Timer'-klasse zorgt er voor dat na elke zelf gekozen tijdsinterval een actie uitgevoerd wordt. In dit project gaat de timer na elke tijdsinterval van 30 ms de UI updaten waardoor er een vloeiende stopwatch gecreëerd wordt. In de Flutter-applicatie is het belangrijk om de logica van de volledige stopwatch inclusief start- en stopknop te scheiden van de tekstcomponent die de verstreken tijd toont. Dit is noodzakelijk voor een betere performantie omdat na elke 30 ms de UI opnieuw gerenderd wordt. Omdat de tekstcomponent als een aparte widget gedefinieerd wordt, kan de tekstcomponent verantwoordelijk gesteld worden voor het beheren van de timer. Dit zorgt er voor dat enkel de tekst opnieuw gerenderd wordt elke 30 ms in plaats van de hele stopwatch.

Ionic biedt standaard geen hulpklassen aan voor deze functionaliteit dus is er beroep gedaan op de timer van RxJS om hetzelfde concept te implementeren. De timer wordt geconfigureerd met een interval van 30 ms zoals in de Flutter-applicatie. Na Iedere interval wordt een eigen gedefinieerde variabele verhoogd met 30 zodat de tijd steeds beschikbaar is in milliseconden. Vervolgens wordt deze variabele omgevormd tot het correcte tijdsformaat met behulp van een library genaamd 'momentjs'.

3.7.3 Ontwikkelproces

Tijdens het ontwikkelproces van beide applicaties heb ik notities genomen van diverse raakpunten. Beide frameworks zijn beoordeeld op verschillende factoren rond *developer-friendliness*.

De resultaten zijn opgesteld in de vergelijkingsmatrix die te zien is in tabel 2.

Kenmerk	Flutter	Ionic
Testen	*Stateful hot-reloading* zorgt er voor dat aanpassingen aan de code doorgevoerd worden op het testtoestel zonder dat de state van de applicatie verloren gaat.	Applicatie kan getest worden in de browser wat zeer handig is. Voor *native* functionaliteiten biedt Ionic ook *hot-reloading* aan op een mobiel toestel maar zonder de state te bewaren.
Documentatie	Zeer duidelijk en uitgebreid.	De verschillende Ionic versies hebben verschillende documentaties wat voor verwarring zorgt.
Ecosysteem	Minder hulp online te vinden. Beperkt aantal artikels, *libraries*, video materiaal en stackoverflow vragen.	Veel hulp online te vinden aan de hand van het ecosysteem van Angular en Javascript.
Leercurve	De opbouw van de applicatie via het widget-systeem kan verwarrend zijn voor beginners.	Met Angular ervaring is Ionic makkelijk om te leren. Zonder Angular ervaring kan het ook uitdagend zijn.
Applicatiestructuur	Eenvoudige structuur. Alles kan in principe in een enkele klasse. De structuur is zelf te bepalen.	Een simpele applicatie kan al snel een zeer uitgebreide structuur krijgen. Het componentenstructuur van Angular wordt toepast.
Debuggen	Goede debug functionaliteiten met breakpoints. Logs worden rechtstreeks naar de console gestuurd.	Extra configuratiewerk nodig om debug functionaliteiten te voorzien.
Styling	Widgets kunnen makkelijk gestyled worden via hun *properties*.	Deels via styling componenten zoals een '*ion-grid*'-tag en deels via CSS.
Voorgedefinieerde componenten	Groot aanbod van Android- of iOS-componenten.	Groot aanbod van componenten die dynamisch uitgewisseld worden voor Android en iOS.

Tabel 2 Vergelijking developer-friendliness van Flutter en Ionic

3.7.4 Meetresultaten

Beide applicaties worden getest op basis van verschillende factoren omtrent performantie. De opstarttijd, het batterij- en geheugenverbruik en de applicatiegrootte worden geëvalueerd. Voor het batterij- en geheugenverbruik wordt de stopwatch functie van beide applicaties gebruikt. De stopwatch wordt actief gesteld voor 30 minuten voordat het batterij- en geheugenverbruik geëvalueerd worden.

De testen worden uitgevoerd op een Samsung Galaxy S8. De Android versie die draait op het testtoestel is 8.0.0. De Flutter-applicatie is ontwikkeld met versie 1.3.14 en de Ionic-applicatie met versie 4.12.0.

De resultaten worden weergeven in tabel 3.

Kenmerk	Flutter	Ionic
Opstarttijd	0,71s	2,38s
Batterijverbruik	23 mAh	376 mAh
Geheugenverbruik	Gemiddeld 81 MB Maximum 416 MB	Gemiddeld 35 MB Maximum 277 MB
Applicatiegrootte	57,06 MB	8,26 MB

Tabel 3 Vergelijking meetresultaten PoC-applicaties

4 Conclusie

4.1 Bespreking resultaten

In het onderzoek is er grondig bestudeerd in welke maten Flutter en Ionic verschillen in hun aanbod van *native* applicaties.

Flutter maakt gebruik van Dart en biedt een uniek systeem aan om *native* applicaties te creëren in vergelijking met andere *cross-platform* varianten zoals React Native. Ionic voorziet een sterke integratie met Angular en zorgt voor een typische hybride opbouw waarbij beroep gedaan wordt op Cordova om Javascript applicaties de mogelijkheid te geven om *native* functionaliteiten aan te spreken in de browser.

Beide frameworks zijn gebouwd met dezelfde filosofie, het zorgen voor prachtige platformonafhankelijke applicaties die op een eenvoudige manier gebouwd kunnen worden. De leercurve voor beide frameworks is niet te steil wat er voor zorgt dat de grootste verschillen te vinden zijn in zowel de meetresultaten en performantie als de *developer-friendliness*.

Naast de verschillen zijn er dus ook redelijk wat gelijkenissen. Beide frameworks bieden een grote hoeveelheid gepersonaliseerde UI-componenten aan. Verder zorgen beide frameworks voor één enkele codebase voor Android en iOS. Het aanspreken van *native* functionaliteiten gebeurt voor de ontwikkelaar in grote lijnen ook hetzelfde aan de hand van de beschikbare *packages*.

De bronnen uit de literatuurstudie zijn gebaseerd op subjectieve opvattingen waardoor deze opvattingen grondig vergeleken kunnen worden met mijn eigen ondervindingen en meetresultaten. Oorspronkelijk zat ik met de gedachte dat het grootste verschil tussen beide mogelijkheden was dat Ionic-applicaties er minder *native* uitzien. De resultaten uit de literatuurstudie samen met mijn eigen ondervindingen tonen aan dat dit helemaal niet het geval is. Ionic besteedt zelfs extra aandacht aan een *native* lay-out door middel van een geautomatiseerde wisseling tussen Android- en iOS-componenten.

Verder komen de resultaten van het volledig onderzoek grotendeels overeen met mijn eigen ondervindingen. Zo is een Flutter-applicatie inderdaad een stuk groter door middel van de manier waarop het framework beroep doet op eigen gedefinieerde widgets en kent Ionic slechtere resultaten op het gebied van performantie. Een opvallende opmerking is wel dat uit de meetresultaten van het PoC blijkt dat Flutter gemiddeld meer geheugen verbruikt. Voor de andere performantie raakpunten heeft Flutter dan weer wel het voordeel.

4.2 Reflectie onderzoeksitem

In het onderzoek heb ik er voor gekozen om twee frameworks met elkaar te vergelijken die zich focussen op platformonafhankelijke mobiele applicaties. De opdrachtgevers hebben aan het begin van de stageopdracht gekozen voor het Flutter-framework en in het onderzoek is dit framework vergeleken met een framework van de hybride technologie genaamd Ionic.

Het doel van het onderzoek was om een heldere blik te creëren van de verschillen en gelijkenissen tussen platformonafhankelijke applicaties die gebouwd zijn met Flutter of met Ionic. Mijn taak bestond uit het onderzoeken van de achterliggende werking van beide frameworks. Bij de ontwikkeling van mobiele applicaties zijn er tal van mogelijkheden en door dit onderzoek is het een stuk duidelijker geworden waar twee bekende frameworks zoals Flutter en Ionic nu effectief in verschillen.

Verder had ik ook de opdracht om verschillende bronnen te raadplegen en een literatuurstudie op te stellen. Hiervoor is er beroep gedaan op drie verschillende bronnen die Flutter met Ionic vergelijken op basis van bepaalde aspecten. De drie bronnen zijn kritisch vergeleken en de tegenstrijdigheden en gelijkenissen zijn vervolgens aangehaald.

Als laatste heb ik een eigen project opgezet met zowel Flutter als Ionic om de conclusies en waarnemingen te testen in een praktisch voorbeeld. Deze twee applicaties heb ik vervolgens ook kunnen gebruiken om de frameworks te testen op basis van factoren zoals performantie en *developer-friendliness*. Hierdoor heb ik een beter beeld gekregen van de verschillende ontwikkelprocessen van beide frameworks.

4.3 Aanbevelingen

Het stagebedrijf heeft er voor gekozen om Flutter te gebruiken voor deze opdracht maar Ionic is een sterke concurrent als framework voor dit project omdat er bepaalde vereisten bestaan omtrent het renderen van HTML. Ondertussen is er een grote bron aan informatie beschikbaar die gebruikt kan worden als basis voor de keuze tussen Flutter en Ionic.

In een volgend onderzoek zou er een ander alternatief onderzocht kunnen worden die eventueel de voordelen van beide opties combineert zoals React Native. Dit framework maakt ook grotendeels gebruik van de web technologie maar dit keer zonder het gebruik van een *web view*. React Native is ook een zeer bekend framework dat geliefd is bij vele ontwikkelaars en is dus zeker ook het overwegen waard.

5 Bibliografie

[1] D. Chaffey, „Mobile marketing statistics compilation,” smartinsights, 11 7 2018. [Online]. Available: https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. [Geopend 23 5 2019].

[2] L. Barnaart, „Mobile App versus Mobile Website Statistics,” JMANGO360, 30 11 2017. [Online]. Available: https://jmango360.com/wiki-pages-trends/mobile-app-vs-mobile-website-statistics/. [Geopend 8 3 2019].

[3] Gigvy, „What’s the Difference between Native vs. Web vs. Hybrid Apps?,” 29 4 2019. [Online]. Available: https://getgist.com/difference-between-native-vs-web-vs-hybrid-apps/. [Geopend 23 3 2019].

[4] statcounter, „Mobile Operating System Market Share Worldwide,” [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide. [Geopend 29 3 2019].

[5] M. Dennis, „Native vs. Cross-Platform Apps – You’ll Be the Winner,” 21 12 2018. [Online]. Available: https://www.zeolearn.com/magazine/native-vs-cross-platform-apps-youll-be-the-winner. [Geopend 6 4 2019].

[6] C. Griffith, „What is Hybrid App Development?,” 7 2 2019. [Online]. Available: https://ionicframework.com/enterprise/resources/articles/what-is-hybrid-app-development. [Geopend 29 3 2019].

[7] Flutter, „FAQ,” Google, [Online]. Available: https://flutter.dev/docs/resources/faq. [Geopend 7 4 2019].

[8] W. Leler, „What’s Revolutionary about Flutter,” 15 8 2017. [Online]. Available: https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514?gi=2de7c8ecb784. [Geopend 7 4 2019].

[9] Ionic, „Ionic Framework,” Ionic, 23 1 2019. [Online]. Available: https://ionicframework.com/docs. [Geopend 16 4 2019].

[10] M. Schwarzmüller, „React Native vs Flutter vs Ionic vs NativeScript vs PWA,” 2 7 2018. [Online]. Available: https://www.academind.com/learn/flutter/react-native-vs-flutter-vs-ionic-vs-nativescript-vs-pwa/. [Geopend 17 4 2019].

[11] N. Sharma, „React Native Vs. Xamarin Vs. Ionic Vs. Flutter: Which Is Best For Cross-Platform Mobile App Development?,” 17 9 2018. [Online]. Available: https://www.apptunix.com/blog/frameworks-cross-platform-mobile-app-development/. [Geopend 18 4 2019].

[12] newgenapps, „Flutter vs React Native vs Ionic - the best tool for cross-platform apps,” newgenapps, 27 9 2018. [Online]. Available: https://www.newgenapps.com/blog/flutter-react-native-ionic-cross-platform-apps. [Geopend 18 4 2019].