Cross-platform udvikling

En eksplorativ undersøgelse af fordele og begrænsninger ved cross-platform tilgange

Projektgruppe	Alexander Ibsen, Mads Christiansen og Martin Laursen
Vejleder	Anders Lassen
Institut	Institut for mennesker og teknologi IMT, Roskilde Universitet
Dato	18-12-2018

Alexander Ibsen, Mads Christiansen og Martin Laursen
RUC – Semesterprojekt (kandidat)
18-12-2018

Abstract

Since the popularization of smartphones, we have seen a growing demand for mobile applications. As of now there are two predominant operating systems used for current smartphones, Android and iOS. This presents a problem for companies and developers, since the operating systems do not share a common coding language or other development qualities. Historically speaking companies have had to manage and spend resources on the development of two apps, one on each platform. However, since the introduction of mobile cross-platform development tools, developers can simultaneously develop for multiple platforms, which potentially reduces the cost of the production. The effectiveness of these cross-platform tools has been called into question in several studies, and many seem to think that cross-platform development compromises certain usability and performance aspects.

In this study we explore the newest generation of cross-platform development frameworks. We are using the Xamarin framework to develop a case application, in which we explore the concerns surrounding compromises when performing cross-platform development. We found that, although cross-platform development has improved dramatically since previous generations, we still see compromises being made toward usability and performance aspects. However, we conclude that current cross-platform tools, such as Xamarin, represent a viable and even preferred approach when developing for multiple platforms.

Indholdsfortegnelse

1. Introduktion

Vi vil i dette afsnit introducere projektet. Afsnittet inkluderer en beskrivelse af emnefeltet og en gennemgang af projektets omfang og begrænsninger. Formålet med afsnittet er at danne et overblik over emnet *cross-platform* udvikling, samt fokusere projektet imod en specifik case som er beskrevet i slutningen af afsnittet.

1.1 Emnefelt

Smartphones har set stor udvikling i løbet af de seneste ~10 år efter den første iPhone blev lanceret. Siden da, er der kommet en lang række leverandører af smartphones som alle vil have en del af markedet.

I takt med det voksende smartphone marked, har vi set en stor vækst inden for mobilapplikationer. Langt de fleste smartphones har adgang til enten Apple's 'App Store' eller Googles 'Play Store', hvor forbrugerne kan finde mobile 'Apps' passende til deres behov. Appmarkedet er åbent for offentligheden, og det er muligt for både virksomheder og private at udvikle deres egne applikationer og udgive dem. Udviklerne møder dog en udfordring hvis de gerne vil have deres app udgivet på to eller flere platforme. De to markedsdominerende platforme, er Android og iOS. Heri ligger problematikken at platformenes kodebase ikke har meget tilfælles, og derfor kan udviklerne møde en stor arbejdsbyrde når deres kode skal tilpasses til en ny platform. Med udgangspunkt i denne problemstilling er der udviklet nye frameworks, der forsøger at hjælpe udvikleren med at tage en cross-platform udviklingstilgang.

Formålet med cross-platform tilgangen er grundlæggende, at spare tid og ressourcer for udviklerne, ved at sørge for at størstedelen af arbejdet er kompatibel med alle platforme. Et populært cross platform-framework hos udviklere er Xamarin. Frameworket fungerer som led mellem de tre mest brugte styresystemer (Android, iOS, Windows). Xamarin gør dette ved at forsyne et interface der er ens på alle platforme med hensyn til funktionalitet. Det er dog blevet påpeget at, der ved cross-platform udvikling findes en række begrænsninger, eksempelvis pointerede Suyesh Amatya og Arianit Kurti, i en undersøgelse, at der på daværende tidspunkt, var optimeringsproblemer ved cross-platform udvikling, ydeevne var altså endnu ikke på niveau med de native udviklingsmiljøer. Desuden viste undersøgelsen også, at der var problemer med at tilgå nativ funktionalitet, såsom kamera (Amatya & Kurti, 2013). Et andet eksempel, er studiet af Esteban Angulo og Xavier Ferre, der påpegede at, en cross-platform tilgang kan have en negativ effekt på applikationens UX, specielt med hensyn til iOS brugere.

Cross-platform mobil udvikling kan stadig betragtes som et relativt nyt område, og er under hastig udvikling både i forhold til eksisterende og nye redskaber. Der har været flere tilgangsvinkler til cross-platform udvikling som, til en vis grad, kan betragtes som *generationer*. Undersøgelser og studier omkring effektiviteten af cross-platform redskaber er sparsomme, især hvis man leder efter information omkring nyere udgaver eller *generationer*. Det vi vil forsøge at gøre, er at give et bidrag til vidensfeltet omkring de nyere generationer af cross-platform udviklingsredskaber.

1.2 Problemformulering

Vi vil i vores undersøgelse tage udgangspunkt i tidligere studier og finde frem til de mest centrale problemer ved cross-platform udvikling. Ud fra disse problemstillinger vil vi opstille en række cases, som vi vil bruge til at teste Xamarin. Undersøgelsens perspektiv vil primært være fra udviklerens synspunkt. Vi fokuserer derfor på hvor nemt og hurtigt et (acceptabelt) resultat kan opnås ved at bruge Xamarin, i forhold til et nativt framework. Med henblik på dette har vi udformet en problemformulering vi vil forsøge at svare på i denne undersøgelse:

Hvordan forholder cross-platform udvikling sig til usability egenskaber, såsom efficiency og learnability, og i hvilken grad understøtter nutidige cross-platform frameworks anvendelsen af usability principper i udviklingen?

1.2.1 Arbejdsspørgsmål

Vi har udarbejdet 3 underspørgsmål, som skal bidrage til besvarelsen af vores problemformulering:

1. Hvad er forskellen på nativ og cross-platform udvikling?
Formålet med dette spørgsmål er at skabe et overblik over 'cross-platform emnet'. Vi undersøger; hvad det betyder at udvikle med en cross-platform tilgang, hvordan det bidrager til en udviklingsproces og selve formålet ved cross-platform udvikling. Vi vejer dette op imod den traditionelle native udviklingsproces.

2. Hvilke fordele og ulemper er der ved cross-platform tilgange?
Cross-platform har set en stor udvikling gennem de seneste år, derfor, for at kunne besvare dette spørgsmål mener vi, at det er nødvendigt at se på udviklingen af cross-platform redskaber; hvordan features og funktionaliteter har ændret sig. På denne måde danner vi os et overblik over fordelene og ulemperne ved cross-platform over en længere periode. Dette er specielt vigtigt, da mange studier er lavet med udgangspunkt i ældre cross-platform udviklingsmiljøer, og derfor kan have fundet problemer der ikke længere er gældende.

3. Hvad er de mest centrale usability egenskaber, der anvendes i udvikling af mobilapplikationer?
På baggrund af underspørgsmål 1 & 2, har vi fundet frem til nogle af de mest fremtrædende problemstillinger ved de nuværende og ældre cross-platform redskaber. Formålet med dette spørgsmål er at undersøge, hvilke usability egenskaber der potentielt bliver nedprioriteret ved brug af cross-platform udvikling. For at kunne besvare dette spørgsmål undersøger vi, hvilke usability-egenskaber der er mest fremtrædende ved udvikling af mobilapplikationer, og hvordan cross-platform forholder sig til dem.

1.3 Afgrænsning

Denne undersøgelse omhandler cross-platform udvikling, vi undersøger hvorvidt det er muligt at skrive en kodebase, der er delt på tværs af iOS, Android og Windows styresystemer uden at miste funktionaliteter der er tilstede ved nativ udvikling. Formålet med undersøgelsen er, at finde begrænsningerne ved at tage en cross-platform tilgang. Med dette i tankerne afgrænser vi os fra nativ udvikling i iOS, da formålet, som nævnt, ikke er at teste de native miljøer, men derimod cross-platform miljøet. Desuden vil vi inkludere nativ Android implementation for at sammenligne med Xamarin's platform specifikke redskaber. Vi mener derfor ikke at der er behov for yderligere nativ udvikling, da det ville overskride omfanget af fokusområdet i undersøgelsen.

Udvikling til UWP (Universal Windows Platform) vil være sekundært, da vores fokus primært omhandler iOS og Android grundet deres store markedsandel. Dette betyder at, vi ikke direkte udvikler med UWP i tankerne hvad angår design, optimering, konventioner osv.

Vi benytter Xamarin som udviklingsmiljø, og afgrænser os fra andre nye prævalente cross-platform udviklingsmiljøer såsom Flutter og React Native eller ældre miljøer såsom PhoneGap, Appcelerator og Cordova. Dette gøres fordi det ikke er muligt at teste flere miljøer inden for en rimelig tidsramme, som denne undersøgelse skal overholde. Vi har valgt Xamarin fordi det er populært (9. mest populære framework (StackOverFlow, 2018)), og fordi det aktivt bliver vedligeholdt og jævnligt ser opdateringer.

Vi forsøger at skabe et overblik over cross-platform udviklingens nuværende status og hvilke trin der, historisk set er blevet taget for at komme frem til de redskaber vi benytter nu. I denne del-undersøgelse vil vi se på hvilke fordele og ulemper der er fremtrædende, især med hensyn til usability og UX-egenskaber. Vi analyserer resultatet og udvælger de mest centrale problemstillinger og udformer en case på baggrund af valget. Dette betyder at vi ikke tager stilling til alle usability og UX-egenskaber, men prøver at forholde os til de mest problematiske, udvalgt på baggrund af tidligere studier. Vi er nødsaget til at tage denne fremgangsmåde, da det ikke er muligt at teste alle funktioner i Xamarin, og vi må derfor indskrænke systemets omfang til de områder vi tror er mest problematiske.

1.4 Case Beskrivelse

Casen er udformet på baggrund af problemstillinger der er blevet fremhævet som led i vores litteratursøgning. Vi har udført en omfattende research omkring cross-platform udvikling med fokus på dets fordele og ulemper. Vi vil tage udgangspunkt i litteraturen og drage på andres erfaringer, i forsøg på at skabe den mest grundige og repræsentative case, som vil teste relevante aspekter af cross-platform tilgangen.

Vil vil kort beskrive en række problemstillinger og derefter give en beskrivelse af indholdet af vores specifikke case. Vi har vurderet disse problemstillinger som de mest centrale:

1. "Cross-platform applikationer har dårligere *performance* end native"
2. "Cross-platform har begrænset adgang til native features"
3. "Cross-platform går på kompromis med usability-egenskaber såsom learnability og memorability "

4. “Cross-platform har begrænsninger i forhold til hvad der kan skrives i den delte kodebase.”

Vi har valgt at udvikle en mobil applikation vha. Xamarin, da vi mener at vi kan få pålidelig og korrekt information omkring effektiviteten af udviklingsværktøjet på denne måde. Vi vil desuden sammenligne med native udviklingsmiljøer såsom Android Studio. På baggrund af de ovenfor nævnte problemstillinger har vi besluttet indholdet af case applikationen.

- Applikationen vil indeholde et **nyhedsfeed** som bliver forsynet gennem en RSS-funktionalitet. Ved denne feature vil der ligge en stor udfordring i at formatere og opsætte indholdet på en attraktiv måde, som forholder sig til de platform-specifikke konventioner **[3]**.
- Vi vil indføre et **persistence lag**, som kan gemme data fra brugeren eksempelvis artikler, favorit feeds osv **[2]**.
- Applikationen vil indeholde en **gps-feature**, som kan vise brugerens position på et kort (Bing, Google Maps, Apple Maps) **[2]**.
- For at undersøge applikationens **ydeevne** vil vi udvikle software der kan måle på visse parametre såsom responstider mm. **[1]**.
- Et stort spørgsmål til effektiviteten af cross-platform udvikling er; hvor meget kode kan deles mellem platforme, og hvor meget skal udvikles nativt i et andet miljø, vi vil forsøge at **samle kodebasen** så vidt muligt, således at der bruges minimal tid i native miljøer **[4]**.

2. Teori

Dette afsnit danner det teoretiske grundlag for undersøgelsen. Afsnittet starter med at beskrive cross-platform som udviklingsredskab. Vi vil gennemgå tidligere undersøgelser og forsøge at uddrage de mest interessante og relevante resultater. Herefter går vi yderligere i dybden med formålet ved cross-platform og hvorfor det bliver benyttet. Det er vigtigt at kende udviklingen af cross-platform redskaber, for at forstå hvilke problemstillinger der hører under de forskellige iterationer, derfor har vi undersøgt hvilke 'generationer' af redskaber der eksisterer, og hvilke fordele og ulemper der associeres ved hver iteration. Til sidst redegører vi for usability og UX aspekter, der især skal tages forbehold for når en cross-platform tilgang benyttes.

2.1 Cross-platform udvikling

En cross-platform applikation er en applikation der kører på flere platforme, eksempelvis Android og iOS (Martinez, 2018). Der findes adskillige frameworks der kan hjælpe udvikleren med at udvikle til flere platforme på samme tid, eksempelvis Xamarin.

Formålet ved at tage en cross-platform tilgang, er at spare ressourcer i forhold til udvikling og vedligeholdelse (Martinez, 2018). Dette opnås ved at have en samlet kodebase, der er delt på tværs af de givne platforme. Eksempelvis indeholder Xamarin en række produkter, der giver mulighed for at udvikle applikationer skrevet i C#: Xamarin.iOS, Xamarin.UWP og Xamarin.Android (Peppers, 2014).

> *Xamarin gives you direct access to the native APIs on each platform and the flexibility to share C# code among platforms" (Peppers, 2014).*

At kunne dele kode mellem platforme vil naturligvis have en positiv effekt på produktiviteten, og spare virksomheder og private for udviklingsomkostninger.

2.1.1 Tidligere studier

Der er flere studier der har vist at en cross-platform tilgang ligeledes også kan have negative effekter på slutresultatet. Eksempelvis udførte Andreas Sommer og Stephan Krusche en undersøgelse af tre forskellige cross-platform frameworks (PhoneGap, Rhodes og Titanium), hvori de udviklede en 'sample' applikation i hvert framework, og sammenlignede deres erfaringer. Konklusionen var at, cross-platform løsninger generelt kan anbefales, men der er stadig begrænsninger i forhold til performance og mangel på native usability features (Sommer, Krusche, 2013).

> *We found out that cross-platform solutions can be recommended in general, but they are still limited if high requirements apply regarding performance, usability or native user experience" (Sommer, Krusche, 2013).*

De pointerer desuden at, ved ikke at have adgang til native usability features, såsom gestikulation funktionaliteter, kan det dette gå ud over applikationens learnability og satisfiability. Dette er fordi navigationen ikke stemmer overens med brugerens forventninger.

> *The support for touch gestures is important because gestures (potentially platform-specific) improve learnability and satisfaction by allowing users to achieve certain actions quicker and in a familiar way (e.g. "pull to refresh").*
> *(Sommer, Krusche, 2013)*

Lignende resultater kan findes i undersøgelsen af Esteban Angulo og Xavier Ferre der fandt frem til at cross-platform udvikling kan gå ud over brugeroplevelsen især for iOS brugere. I forsøget havde de udviklet to native applikationer i henholdsvis Android og iOS samt en cross-platform applikation i Titanium Appcelerator. Applikationerne havde samme indhold, men var udviklet af to 'teams' med lignende UX-færdigheder. De testede applikationerne på 37 testpersoner, og efter 5 dage med hver af udgaverne blev testpersonerne bedt om at give deres mening. Forsøget viste at 52% af testpersonerne foretrak applikationen skrevet i nativ Android fremfor Appcelerator, mens 86% af testpersonerne foretrak nativ iOS fremfor Appcelerator. Konklusionen var derfor at, man godt kunne opnå et godt UX-design via. en cross-platform tilgang, men det er muligt at opnå et højere UX-niveau ved at tage en nativ tilgang, især med hensyn til iOS (Angulo, Ferre, 2014). I artiklen beskriver de ikke specifikke UX-egenskaber, men de har stor fokus på overensstemmelse med platform specifikke konventioner og flydende UI-interaktion hvilket ofte kategoriseres som *performance* og *learnability*.

Undersøgelserne har vist; at der er store muligheder inden for cross-platform udvikling, og på trods af nogle begrænsninger kan det stadig anbefales. Desuden er cross-platform udvikling et hastigt udviklende felt, og der bliver jævnligt forbedret på eksisterende værktøjer eller udviklet nye, mere sofistekerede redskaber.

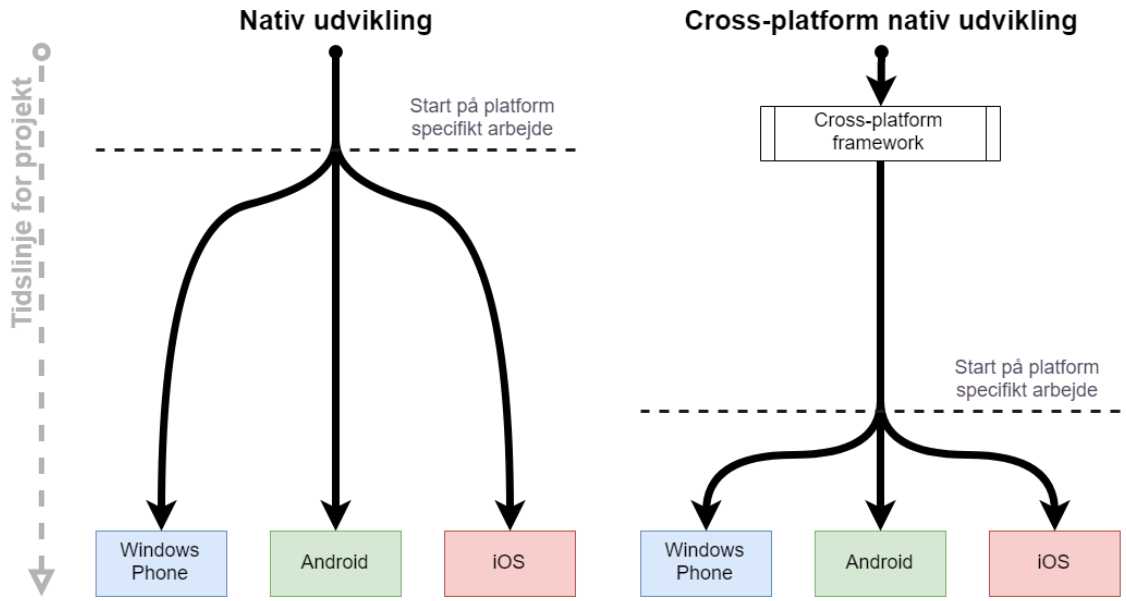
2.1.2 Formål ved cross-platform

Formålet ved at udvikle en 'cross-platform applikation' er, kort sagt, at dække en bredere gruppe af brugere, og på denne måde, potentielt øge omsætningen (Martinez, 2018). En virksomhed, der ønsker at udvikle en mobil applikation til flere operativsystemer og mobilmarkeder ville, traditionelt set, være nødsaget til at producere den samme applikation flere gange. Én gang for hvert operativsystem de målretter sig. Dette ville i hvert fald være tilfældet, hvis ikke for cross-platform udvikling.

Cross-platform udvikling kan have forskellige niveauer af kompleksitet, det kan desuden variere hvor meget kode der kan deles under et tag, alt efter type af projekt og hvilket udviklingsmiljø der bliver anvendt. Eksempelvis kan cross-platform udvikling bestå af en samlet kodebase, som kompileres til flere operativsystemer og bagefter støttes op af nativ interface design udviklet i de tilhørende native miljøer. Årsagen til at tage denne tilgang er for at drage fordel af native features og design, som ikke kan findes i cross-platform redskabet. Det er påpeget i tidligere studier at mange cross-platform redskaber endnu ikke indeholder alle features som er tilgængelig ved nativ udvikling, og derfor kan en kombination af cross-platform og nativ udvikling være en mere attraktiv tilgang.

Behovet for cross-platform udvikling er opstået pga. den store mængde ressourcer det kræver at udvikle og vedligeholde adskillige applikationer i deres native miljøer. De nyeste og mest lovende muligheder inden for cross-platform udvikling søger at holde ressourcerne

samlet i så meget af udviklingsprocessen som muligt. Xamarin er et af disse cross-platform udviklingsframework. Xamarin tillader at have omkring 75% af koden samlet i én kodebase, hvor størstedelen af logikken og UI vil kunne opbygges i samme programmeringssprog og udviklingsmiljø. Dette kan heftigt reducere de nødvendige ressourcer under udviklingsprocessen samt den efterfølgende vedligeholdelse når appen er i det operative stadie (Xamarin, 2018).



Figur 2.1 - Illustration af hvor længe ressourcer kan holdes samlet i udviklingsprocessen, ved nativ og cross-platform udvikling (Xamarin).

Figur 2.1.2 illustrerer hvordan ressourcer og arbejdsprocesser er fordelt over et tidsforløb under nativ og cross-platform udvikling. Det kan ses at vha. et cross-platform redskab, eksempelvis Xamarin, kan vi holde udviklingsressourcerne samlet i en længere periode. Dette kan naturligvis have store fordele for en virksomhed, da udviklerne er ikke nødsaget til at producere flere selvstændige produkter, én til hver platform, men i stedet kan udviklerne opbygge størstedelen af koden i et samlet miljø, og derefter tilpasse applikationen til de native platforme. Dette betyder at der skal produceres mindre kode, og derfor kan udviklingsomkostninger potentielt reduceres betydeligt. Frameworks som Xamarin lover en samlet kodebase og funktionalitet, hvilket betyder at platform specifikt arbejde (spredning af ressourcer) er en meget mindre del af det samlede tidsforløb.

Cross-platform udvikling er ikke et nyt fænomen. For mobilapplikationer har det været undervejs i takt med smartphonens udbredelse. Det følgende afsnit vil dykke ned i hvad vi ser som værende tre grove generationer af udviklingen af mobilapplikationer; vi beskriver processen fra nativ udvikling til cross-platform udvikling.

2.2 Tilgange til cross-platform udvikling

I takt med at mobilens udbredelse, udvikling og interesse er steget, har udvikling af mobilapps også været igennem en fase af forskellige tilgange til at optimere udviklingsprocessen til cross-platform. Alle tilgangene varierer mere eller mindre fra

hinanden, og det er svært at påpege præcist hvordan forskellige frameworks forholder sig til hinanden.

Selvom det har været en løbende udvikling der er sket inden for cross-platform udvikling, vil vi i grove træk inddele tilgangene i tre generationer; nativ udvikling, hybrid-udvikling og 'cross-platform nativ udvikling'. Hver af disse generationer har søgt at udforske og efterfølgende forbedre mangelfulde aspekter i den forrige generation. Vi ser opdelingen af tilgangene i tre generationer som en måde hvorpå det kan være lettere at få overblik over tilgangenes styrker og tilhørsforhold til hinanden.

2.2.1 Nativ udvikling - 1. generation

Den første generation af app udvikling er, udvikling i de native miljøer og programmeringssprog. Her har vi Android apps, der er kodet i Java og vha. f.eks. Android Studio. Ligeledes er der iOS apps, som er kodet i Objective-C vha. f.eks. Xcode.
Nativ udvikling har sine fordele i at koden er udviklet i og optimeret til et miljø, der udelukkende er tiltænkt det specifikke operativsystem. Ydeevnen er i top og udvikleren har direkte og nem adgang til at anvende systemets native funktionaliteter, såsom kamera, mikrofon, kontakter m.m. I nativ udvikling er det også det native brugerinterface for operativsystemet, der anvendes. De største operativsystemer, Android og iOS, har klare retningslinjer for hvordan UX skal udarbejdes for at følge de normale konventioner for operativsystemet.

En problematik ved nativ udvikling, er at der skal udvikles en unik og selvstændig app til hvert operativsystem hvor appen ønskes udgivet. Det kræver en stor mængde ressourcer, i form af tid og ekspertise. Ligeledes er vedligeholdning af flere native apps en større proces. Ved at udvikle nativt på hver platform, opnår man derimod den fulde støtte af native funktionaliteter og brugerinterface.

2.2 2 Hybrid-udvikling og web apps - 2. generation

Anden generation af app-udvikling søgte at standardisere udvikling ved at samle funktionaliteten i et enkelt programmeringssprog og udfra denne genererer fungerende apps til hvert operativsystem. Eksempler på platforme hvor det er muligt at lave hybrid-udvikling af apps er for eksempel PhoneGap, Appcelerator og Cordova.

2. generation bestod hovedsageligt af web-apps. Web-apps består udelukkende af HTML, CSS og Javascript og det er denne opbygning der tillader at de er cross-platform, da disse sprog kan fortolkes på alle enheder der har en browser. Web-apps havde dog manglende adgang til mange native og hardware features på enhederne. Det er på grund af dette at *hybride* tilgange til app udvikling opstod (Heitkötter et. al., 2013).

Hybrid-udvikling søger at kombinere styrken ved eksklusive web apps og native apps. Hybride apps er altså hovedsageligt web apps bestående af HTML, CSS og JavaScript, men med det ekstra lag at de er inkorporeret i en tynd app-skal for at komme tættere på de forskellige operativsystemer og deres funktionalitet. Yderligere er det en nemmere tilgang til at udgive direkte til markedet for hvert system; Google Play Store, Apple Store, Microsoft Store (Xanthopoulos & Xinogalos, 2013).

Hybride apps løser delvist problemet med manglen på en samlet kodebase, samt at tillade multiplatform udvikling. Løsningen kommer dog med sine kompromisser, både i form af at god performance og usability er sværere at opnå. Det er sværere at tilgå og arbejde med platform specifikke features og brugerinterface.

2.2.3 Cross-platform "nativ" udvikling - 3. generation

Cross-platform "nativ" udvikling er den seneste generation i rækken af tilgange til udviklingen af mobile applikationer. Her prøver man at bevare effektiviteten af hybrid-udvikling samtidig med at man bibeholder den native performance og brugerinterface. Der er dukket mange lovende cross-platform udviklingsframeworks op over de seneste år, bla. ReactNative, Xamarin og Flutter.

Cross-platform frameworks tillader udviklere at skrive kode én gang og anvende en større del af koden på alle platforme. Dette kommer med prisen af at man mister performance og et komplet nativt brugerinterface, men det skal dog siges at de nyeste frameworks inden for cross-platform udvikling, anses for at give en tæt på nativ performance og brugeroplevelse.

Tabellen nedenunder opsummerer kort de tre generationer. Den dækker hvilke udviklingsmiljøer der er/var mest udbredt under hver generation, samt opsummerer de styrker og svagheder som de specifikke generationer havde.

1. generation Nativ udvikling	**2. generation** Hybrid-udvikling	**3. generation** Cross-platform nativ udvikling
Udviklingsmiljøer:	***Udviklingsmiljøer:***	***Udviklingsmiljøer:***
Objective C / Swift Android Java Windows .NET	PhoneGap Cordova Appcelerator Ionic	Xamarin React Native NativeScript Flutter
Styrker og svagheder:	***Styrker og svagheder:***	***Styrker og svagheder:***
+ Høj performance + Nativ brugerflade og UX + Direkte adgang til platform specifikke egenskaber - Multiplatform - Samlet kodebase - Omkostningseffektiv	- Høj performance - Nativ brugerflade og UX - Direkte adgang til platform specifikke egenskaber + Multiplatform + Samlet kodebase + Omkostningseffektiv	+ Høj performance + Nativ brugerflade og UX + Direkte adgang til platform specifikke egenskaber + Multiplatform + Samlet kodebase + Omkostningseffektiv

Tabel 2.1 - Oversigt over de tre generationers udviklingsmiljøer, samt deres styrker og svagheder. Farverne indikerer i hvor stor en grad generationerne besidder de nævnte kvaliteter. Rød er totalt fraværende, gul er delvist fraværende, og grøn indikerer at kvaliteten er til stede. (Chauhan, 2018)

Tabellen er grov i sin opdeling af svagheder og styrker, samt de forskellige udviklingsmiljøers præcise tilhørsforhold til generationerne. Det er selvfølgelig en flydende udvikling der er sket over tid og udviklingsmiljøerne har haft varierende styrker og svagheder. Xamarin anser vi som en 3. generations cross-platform framework. Det følgende afsnit vil beskrive Xamarin som cross-platform framework, samt udforske hvorvidt Xamarin opfylder de normale konventioner for opbygning af brugerinterface på dets tre understøttede operativsystemer.

2.3 Xamarin

Xamarin er i blandt de mest populære frameworks for cross-platform udvikling (StackOverflow, 2018), men research omkring fordele og ulemper ved Xamarin er sparsomt (Martinez, 2018). Xamarin definerer vi som et 3. generations cross-platform framework. Vi har altså adgang til native features, og applikationen bliver ikke fortolket ved runtime. Det betyder at der ikke bliver gået på kompromis med ydeevnen/responstiden. Ifølge Microsoft gør Xamarin også brug af den platform specifikke hardware acceleration:

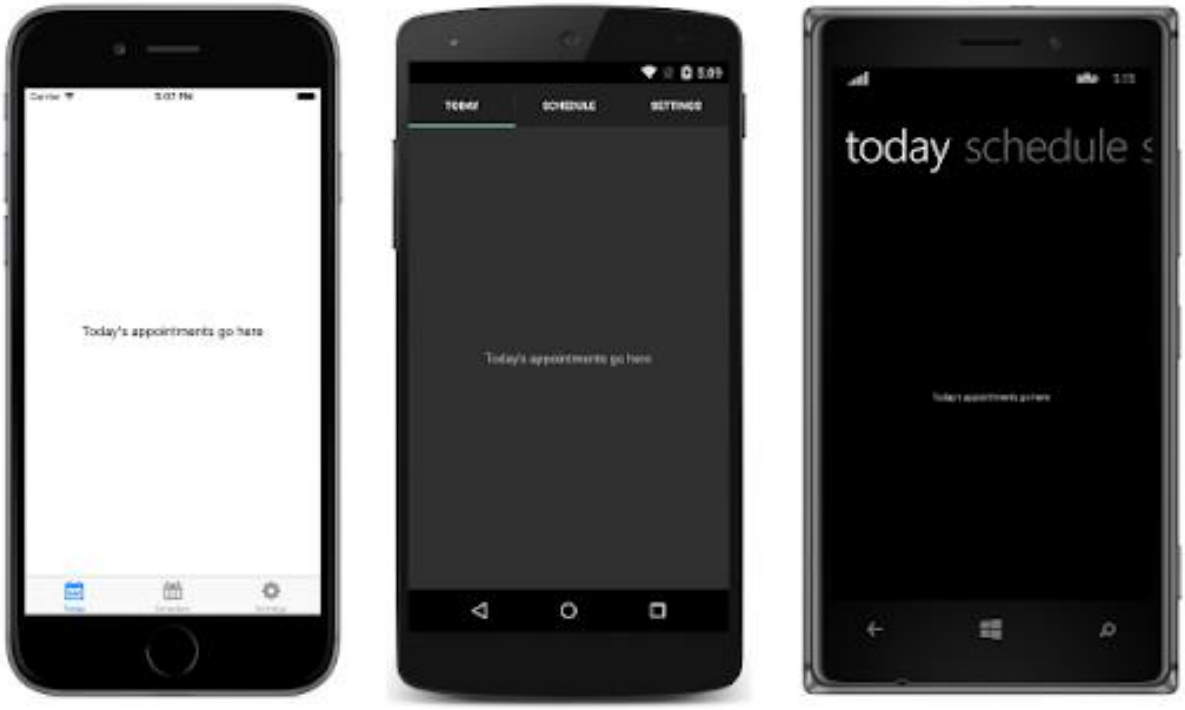
> *Apps built using Xamarin leverage platform-specific hardware acceleration and are compiled for native performance. This can't be achieved with solutions that interpret code at runtime." (Microsoft Visual Studio, 2018)*

Det er altså et stort salgspunkt for Xamarin at kunne kompilere og køre deres C# kode på platformene, i stedet for at skulle fortolke koden ved runtime. Xamarin opnår dette ved at bruge 'Mono' (Xamarin Developer [1], 2018). Mono er et C# framework, som kan bruges på en række forskellige platforme, herunder Android, iOS, Windows og Linux. Mono består, blandt andet, af en C# compiler og et .NET Framework klasse bibliotek, der er kompatibelt med Microsoft's .NET Framework (Mono Project, 2018). Xamarin bygger én eller flere udgaver af applikationen som bliver kompileret til nativ kode for den givne platform. På iOS opnås dette via Xamarins Ahead-of-Time (AOT) compiler, som gør det til nativ ARM assembly. På Android kompilerer Xamarin til et *Intermediate Language* (IL), som bliver kompileret til ARM assembly når applikationen åbnes, dette kaldes Just-In-Time (JIT) kompilation.

At have en delt kodebase mellem platforme kan lede til en høj grad af kompleksitet, et område hvor dette specielt er synligt er ved brugergrænseflade design. Xamarin gør brug af platformenes native UI-framework, frem for at udvikle deres eget. Det er derfor ikke er muligt at bruge den samme markup til at definere UI'et, da de forskellige platforme har forskellige UI-elementer. Desuden har platformene forskellige konventioner for hvordan UI'et skal udformes, i forhold til placering af elementer, navigation osv.

2.3.1 Design af brugergrænseflade

Xamarin er delt op i to kerneprodukter, Xamarin.iOS og Xamarin.Android. Disse er rettet mod at bygge grænsefladen på henholdsvis iOS og Android, da disse platforme ikke følger de samme konventioner om hvordan et UI skal opbygges.



Figur 2.2: Billede af hvordan TabbedPage UI elementet ser ud på iOS, Android og Windows Phone. (Microsoft [1], 2018)

Et tilføjelsesprodukt til Xamarin, der forsøger at gøre det muligt også at bruge den samme kode til flere grænseflader er Xamarin.Forms. Fordelene ved Xamarin.Forms er altså en forøget kode deling, dog resulterer dette i nogle kompromisser der skal tages i forhold til hvis man anvendte Xamarin.iOS og Xamarin.Android (også kendt som Xamarin Native).

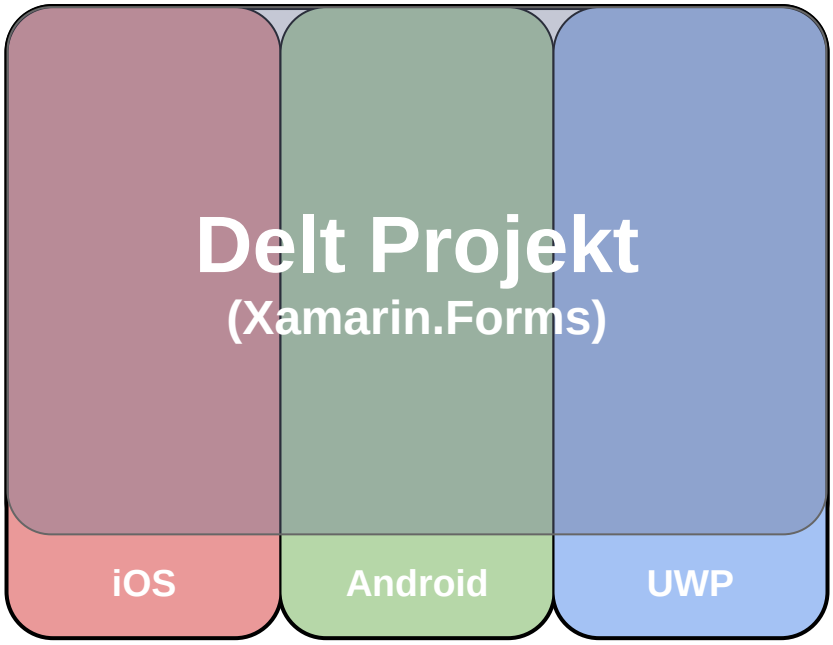
Problematikken ved at benytte Xamarin Native er naturligvis, at det kun er muligt at kode imod én platform ad gangen. Selvom det kun drejer sig om UI elementer kan det stadig medføre en betydelig platform specifik arbejdsbyrde. Xamarin har udviklet Xamarin.Forms for at afhjælpe denne problemstilling, da det er muligt at kode UI i en samlet kodebase, og ud fra denne kode generere platform specifikke UI-elementer. Xamarin.Forms fungerer altså som en UI mellemmand mellem platformene.

En ulempe ved denne tilgang er at genereret kode tit er uoverskuelig og forvirrende da det skal kunne virke i mange forskellige sammenhænge, hvor det er urentabelt for Xamarin holdet at forsøge at optimere til alle disse situationer. Dette kan resultere i en “tungere” App, som fylder mere og kører langsommere. Derudover er der nogle tilpasningsmuligheder som går tabt, i hvert fald i den delte kode, da de ikke vil være tilgængelige på alle platforme.

En Xamarin.Forms app er delt op i fire projekter;

- et til den samlede kode og UI
- et for hver af de tre platforme der understøttes;
    - iOS
    - Android
    - UWP (windows 10)

De platform specifikke projekter indeholder kode som tilgår native features og manifester der beskriver hvilke tilladelser app'en skal bede enheden om osv. Det kan ses som tre projekter, når man bygger appen, hvor indholdet af det delte projekt er en del af hver af de platform specifikke projekter (iOS, Android, UWP).



Figur 2.3 - Formålet med figuren er at illustrere hvordan iOS, Android og UWP projekterne er separate, men har en meget stor del tilfælles.

2.4 Usability

I en udviklingsproces der er brugercentreret, forholder termet *Usability* sig til, i hvor stor en grad, et produkt tillader en bruger at opnå fastsatte mål med effektivitet og tilfredsstillelse. En definition af usability bliver beskrevet af J. Robier:

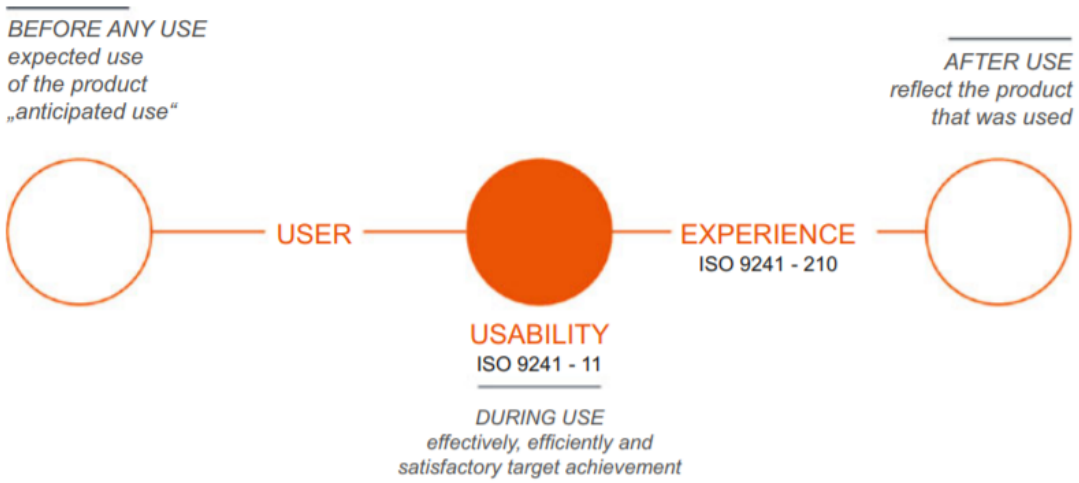
> *By definition, usability is all about developing products efficiently, effectively and to the highest satisfaction of the customer." (J. Robier, 2016)*

I denne definition fremhæver Robier tre kvaliteter, som usability søger gradvist at opnå; *efficiency*, *effectiveness* og *satisfaction*. I det danske ordforråd betyder *efficiency* og *effectiveness* det samme, men i forbindelse med usability skelnes der imellem dem. Vi vil anvende Robiers simple forklaring af hvad de tre begreber indebærer til at skelne mellem hvad efficiency og effektivitet er. En process er effektiv hvis man kan opnå sit mål. En process er *efficient*, hvis man opnår målet på den mest optimale måde; færrest ressourcer, korteste process og så videre.

> *Effective = The target can be reached.*
> *Efficient = The shortest path without any obstacles for the user.*
> *Satisfaction = With a smile.*
> *Usability = Quickly achieving the goal with a smile"*
> *(J. Robier, 2016)*

Denne opsummering, som Robier laver, er meget generisk og uspecifik, men vi mener at den giver et godt indblik i hvad usability indebærer.

Usability ses i dag som en del af det bredt dækkende begreb *User Experience* (herefter betegnet UX). I de seneste to årtier er begreberne usability og UX opstået og løbende fået større interesse. I starten blev begge begreber brugt til at beskrive det samme emne, men i og med at de begge blev mere veldefineret blev konsensussen at de ikke var ens, men delte elementer af hinanden. Denne opdeling blev beskrevet Preece, Rogers og Sharp i 2002. De angiver usability som værende den hårde kerne af målbare kvaliteter og UX som den blødere ydre skal uden lige så skarpt definerede kvaliteter (Preece et. al, 2002). Den nyere forståelse af usability følger den samme opbygning, men ser usability som et element af det bredere begreb; User Experience, og at usability er med til at opnå god UX.



Figur 2.4 - Viser sammenhængen mellem UX og usability. UX er den fulde process fra før til efter brugen af produktet. Usability er et mellemled i processen (J. Robier, 2016).

Kort fortalt så fokuserer usability på objektivitet, målbare faktorer, optimering af processer og en simpel forståelse af kundetilfredshed. UX fokuserer på den subjektive fornemmelse ved anvendelsen af et produkt og opfattelsen af dets kvalitet, hvori usability er et aspekt af og hjælper med at opnå UX. UX fokuserer ikke kun på anvendelsen af produktet, men også perioden op til anvendelse samt perioden efter anvendelse og den opfattelse/oplevelse brugeren har af produktet i den fulde periode.

I vores case vil vi forholde os til usability. Dette betyder at vi ikke forholder os til de UX aspekter, der falder udenfor de specifikke rammer for usability. Dette gør vi fordi vores case ikke er omfattende nok til at kunne skabe indsigt i de mere bløde og følelsesladede aspekter som UX i sin helhed kan bringe.

For at gribe yderligere fat i konceptet om usability vil vi anvende Jakob Nielsens tilgang til usability. Jakob Nielsen er medstifter af NNgroup; et konsulentfirma med ekspertise i usability.
Nielsen beskriver fem kvalitetsegenskaber som usability dækker over:

- **Learnability**: Hvor let er det for brugeren at klare basale opgaver første gang de anvender designet.

- **Efficiency**: Efter at brugeren har lært designet, hvor hurtigt kan de udføre opgaver?
- **Memorability**: Hvor let er det for brugeren at huske designets funktioner efter at de har været væk fra det i en periode?
- **Errors**: Hvor mange fejl laver brugeren, hvilke fejl og hvor let er det for dem at rette fejlene.
- **Satisfaction**: Hvor behageligt er designet at anvende?

(J. Nielsen, 2012)

Han påpeger at disse ikke er de eneste kvaliteter, og nævner f.eks “Utility” som forholder sig til om designet tillader brugeren at opnå det de vil. Denne kvalitet anser vi som værende magen til den kvalitet fremhævet af Robier som effektivitet, altså om målet kan nås.

Vigtigheden i at forholde sig til usability er fordi, det er med til at bestemme om et produkt bliver succesfuldt eller ej. Et dårligt design kan få folk til at opgive at fuldføre deres mål, eller ikke vende tilbage givet det var en ineffektiv proces. På grund af dette er det vigtigt at cross-platform frameworks har et udviklingsmiljø, der tillader udviklerne at skabe god usability. Igennem udviklingen af vores case app, vil vi så vidt som muligt vurdere hvordan og i hvilken grad Xamarin understøtter opbygningen af de essentielle usability kvaliteter.

3. Metode

Vores undersøgelse kan siges at være todelt. Vi undersøger hvorvidt Xamarin, et 3. generations cross-platform framework, er et godt redskab til at udvikle applikationer, som skal virke på flere platforme. Derfor må vi også undersøge om selve applikationen, udviklet i frameworket, lever op til en *nativ* standard. Vi vil, i dette afsnit, gennemgå hvilke fremgangsmåder vi benytter for at give en nøjagtig vurdering af Xamarin som framework og case applikationen.

3.1 Prototype design

Vi har udarbejdet en mobilapplikation, som har til formål at udfordre cross-platform frameworket Xamarin på de, historisk set, mest problematiske områder hvad angår cross-platform udvikling.

Figur 3.1 - Illustration af mobilapplikationens layout

Mobilapplikationen består af 5 *views* som det kan ses på figur 3.1 Nyheds layout, Artikel layout, Kort layout, Favorit layout og Indstillings layout er alle indlejret i Navigations layoutet, da alle *views* skal inkludere de samme navigationsmuligheder. Følgende er en kort beskrivelse af indholdet af de forskellige views:

- Nyheds layout
    - I dette view bliver et RSS-feed præsenteret i en kompakt udgave, med titel og beskrivelse. Ved hvert element er det muligt at vise den fulde artikel ved at trykke på et link. Linket navigerer brugeren hen til Artikel layout. Desuden kan brugeren angive favorit artikler. Artikler som er favoriseret gemmes lokalt på den mobile enhed, og kan hentes i favorit layoutet.
- Artikel layout
    - I dette view bliver en fuld artikel fra RSS-feed'et vist. Vi gør dette ved hjælp af WebView-funktionaliteten.
- Kort layout
    - I dette view har vi implementeret en 'kort' funktionalitet, fra udbyderne Apple, Bing eller Google afhængig af hvilken platform applikationen bliver kørt på.
- Favorit layout
    - Dette view er opsat på samme måde som Nyheds layoutet, men i stedet for at bruge RSS-provideren, henter vi data fra mobilens lager.
- Indstillingslayout
    - Dette view indeholder evt. indstillinger brugeren kan angive. Eksempelvis kan brugeren tilføje nye feeds og selv vælge hvilke der skal være synlige på Nyheds layoutet.

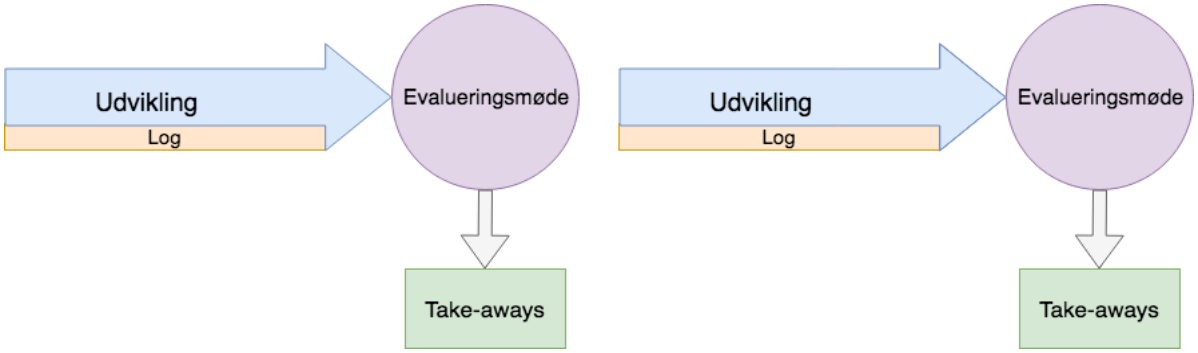
3.2 Undersøgelsesform

Vores undersøgelse består af to hovedelementer, en udviklingsproces og en produkt-evalueringsfase. Vi vil, i dette afsnit, beskrive hvordan vi har struktureret forløbene.

3.2.1 Udviklingsproces

En stor del af vores undersøgelse omhandler, hvordan et cross-platform udviklingsforløb forholder sig de native modparter. For at undersøge dette har vi udviklet to mobilapplikationer i henholdsvis Xamarin og Android Studio. Vi har dog nøjes med at udvikle delprodukter i Android Studio. Formålet ved denne fremgangsmåde er danne et overblik over forskellen ved nativ og cross-platform udvikling, og finde frem til de mest iøjnefaldende problemstillinger ved cross-platform tilgangen. Vi er opmærksomme på at omfanget af denne undersøgelse er relativt begrænset, og vi derfor er nødsaget til at fokusere på visse elementer af cross-platform udvikling og Xamarin frameworket. Vi har i denne anledning udformet en case som inkluderer nogle af de mest fremtrædende problemer ved cross-platform, baseret på tidligere studier.

Vi har i løbet af udviklingen ført en *log* hvor vi har noteret hvilke områder af udviklingsprocessen der har forårsaget problemer. Vi har desuden haft løbende evaluering gennem forløbet, hvor vi har forsøgt at klassificere/kategorisere udfordringer ved udviklingen. Xamarin har en anderledes udviklingstilgang end de native miljøer, og derfor kan det, i nogle tilfælde, være problematisk at fastslå hvor et givent problem stammer fra. Dette er grundet vores begrænset erfaring i miljøet, og kompleksiteten ved at udvikle til flere systemer på én gang, som forværres ved at fejlkoder ikke altid er konsistente på tværs af platformene, vi vil gennemgå dette i flere detaljer i senere afsnit.



Figur 3.2 - Illustration af udviklings- og evalueringsforløb

Forløbet er udformet som illustreret i figur 3.2. Vi har haft en række udviklingsfaser, under disse faser har vi ført en log. Loggen indeholder; beskrivelse af problemer, kommentarer, overvejelser, beslutninger osv. Efter en afsluttettet fase afholdte vi et evalueringsmøde. Under mødet har vi diskuteret og yderligere beskrevet loggens indhold, samt udarbejdet et mål for den kommende udviklingsfase. Evalueringsmødet resulterer i et dokument, der indeholder en række 'take aways' altså erfaringer og pointer vi har gjort os i løbet af fasen. Disse take aways vil beskrives og analyseres i sammenhæng med vores evalueringsstrategi, som beskrevet i afsnit 3.3.

3.2.2 Produkt evaluering

I produkt evalueringen har vi fokus på applikationens *performance*. Dette er grundet at performance har indflydelse på flere usability egenskaber, såsom effektivitet, efficiency og satisfiability. En hurtig og responsiv mobilapplikation er vigtigt for den overordnede brugeroplevelse af appen.

Performance

Vi har angivet en række punkter hvor det er muligt at måle applikationens *performance:*

- Hvor lang tid tager det at åbne applikationen?
    - Cold boot - Når app'en skal starte helt fra bunden
    - Warm boot - Når en del af app'en stadig kører, men resten skal genskabes
- Hvor meget hukommelse bruger applikationen?
- Hvor meget fylder applikationen?

For at give en mere detaljeret og kontekstuel evaluering, har vi udført forsøget ligesom Esteban Angulo og Xavier Ferre udførte deres studie i 2014, og har udviklet både en nativ og en cross-platform udgave af applikationen, for at danne et grundlag for sammenligning. Vi har dog kun udviklet udvalgte features i nativ Android.

Usability inspektion

Usability inspektion er en samling af metoder til at evaluere et brugerinterface. Usability inspektion står i kontrast til usability testing i og med at det ikke er rigtige brugere, der bliver inddraget til evalueringen, men derimod evaluatorer. Evaluatorerne kan være usability specialister, udviklere, slutbrugere med viden om produktet eller andre typer af

professionelle (Mack & Nielsen, 1994). Usability inspektion kan med fordel anvendes tidligt i og løbende med en udviklingsproces, hvor brugere endnu ikke er inddraget.
I de forskellige inspektionsmetoder baseres evalueringen af brugerfladerne på evaluatorernes velovervejede dom, men kriterierne denne dom skal baseres på varierer.

I vores produkt evaluering vil vi tage udgangspunkt i den uformelle *heuristiske evaluering* fremlagt af Jakob Nielsen. Denne metode indebærer at de udvalgte usability evaluatorer skal bedømme hvorvidt et brugerinterface lever op til etablerede usability principper. Nielsen præsenterer en liste af usability principper:

- Simpel og naturlig dialog
- Anvend brugernes sprog
- Minimér mængden af information brugerne skal huske.
- Konsistens
- Feedback
- Klart definerede udveje
- Genveje
- Præcise og konstruktive fejlbeskeder
- Forhindre fejl
- Hjælp og dokumentation
  (Mack & Nielsen, 1994)

I og med at vi ikke laver et produkt til slutbrugere, føler vi at usability inspektion kan give os en god indgangsvinkel til at evaluere på forskellige usability aspekter i Xamarin udviklingen. Den ovenstående liste af usability principper er indenfor rammerne af vores eksisterende forståelse af usability. Ved at inddrage dem i vores evaluering kan vi bedre vurdere hvorvidt Xamarin understøtter at udvikle med de anerkendte principper.

3.3 Udgangspunkt for evaluering

Vi vil evaluere Xamarin ud fra funktionaliteten af et cross-platform produkt udviklet i Xamarin vejet op imod enkeltstående funktionaliteter udviklet i et nativt Android miljø. Evalueringen baseres på vores egne erfaringer og opfattelse af udviklingsforløbet samt det endelige produkt. Evaluering af funktionaliteten skal bruges til at vurdere svagheder og styrker vi fandt ved at udvikle i Xamarin. Ud fra dette ønsker vi at undersøge om den store udvikling indenfor cross-platform frameworks har den eftertragtede effektivitet, og hvis ikke; undersøge hvor kompromisser bliver indgået.

Vores evalueringskriterier vil tage udgangspunkt i en artikel af Heitkötter et. al, hvori de opstiller en række kriterier til at evaluere cross-platform udviklingstilgange (f.eks. Xamarin). Kriterierne dækker en bred vifte af relevante punkter til at evaluere kvaliteten af den givne tilgang. Grundet vores begrænsede projektomfang har kriterierne forskellig vægtning i forhold til de brugsscenarier vi søger at fremstille med vores produkt. Vi har valgt ikke at fravælge kriterier, men gør opmærksom på at nogle af dem er sværere for os at opnå egen erfaring og indsigt i.

Heitkötter et. al. opdeler kriterierne i to perspektiver; et infrastruktur perspektiv og et udviklingsperspektiv. Begge perspektiver har syv vurderingskriterier.

3.3.1 Kriterier for infrastruktur perspektiv

Infrastruktur perspektivet opsummerer kriterier relateret til livscyklussen for en mobil applikation, dens anvendelse, operation og funktionelle evner (Heitkötter et. al, 2013).

1	**Licens og pris** Tilbydes framework som Open-source, gratis version og/eller licens?
2	**Understøttede platforme** Hvor mange platforme støttes og hvor godt støttes de (i forhold til andre frameworks)?
3	**Adgang til platform specifikke features** Hvor nem og hvor stor adgang er der til enheds specifikke features, såsom kamera eller GPS?
4	**Langsigtet gennemførlighed** Har frameworket indikatorer på langsigtet gennemførlighed? Indikatorer kan eksempelvis være hyppige opdateringer og bug-fixes, et aktiv community og support af selv de nyeste mobiloperativsystemer. Dette kriterie kan især være relevant for mindre virksomheder, hvor valg af framework er en betydelig investering.
5	**Udseende og følelse** Tillader frameworket et nativt udseende og en nativ fornemmelse når appen anvendes?
6	**Applikationshastighed** Dette kriterie forsøger at evaluere applikationens opstartshastighed, samt performance imens appen opererer og hvor responsiv den er overfor brugeren.
7	**Distribuering** Hvor let er det at distribuere apps, når de er udviklet i det givne framework. Tillader et framework distributionskanaler ud over det sædvanlige?

(Heitkötter et. al, 2013)

3.3.2 Kriterier for udviklingsperspektiv

Udviklingsperspektivet dækker alle kriterier der er direkte forbundet med udviklingen af appen, som f.eks. test, debugging og udviklingsværktøjer (Heitkötter et. al, 2013).

1	**Udviklingsmiljø** Dette kriterie forholder sig til platformens modenhed mht. features og værktøjer. (IDE, debugger, emulator, autocompletion osv.)
2	**GUI design** Dette kriterie forholder sig til processen at udvikle det grafiske brugerinterface.
3	**Lethed af udvikling** Hvor god er platformens kvalitet og kvantitet af dokumentation og er det en flydende læringsproces?
4	**Vedligeholdelsesevne** Tillader platformen god vedligeholdelse? Til at vurdere dette anvendes der en simpel parameter kaldet “LOC”, der står for “lines of

codes". Jo færre linjer kode, jo simplere er det at lære nye op og jo lettere er det at læse og forstå koden. Denne metode er simpel, men målbar.

5	**Skalérbarhed** Forholder sig til hvor let større udviklingshold og projekter kan anvende frameworket. Tillader frameworket mange forskellige muligheder for at skabe både store og små projekter uden stor ændring i grundstrukturen?
6	**Mulighed for videreudvikling** Dette kriterie forholder sig til hvor genbrugelig kildekoden er til andre tilgange end det valgte. Dette gøres for at vurdere hvor risikoen for at et "lock-in" kan opstå, skulle projektet være startet med en tilgang men ikke er muligt at overføre til en anden tilgang.
7	**Hastighed og pris for udvikling** Evaluerer hastigheden af udviklingsprocessen. Er der faktorer der forhindrer en hurtig og ligefrem tilgang til udviklingen? Hvad er omkostningerne for at udvikle i frameworket - denne er ofte bundet op på hastigheden for udvikling?

(Heitkötter et. al, 2013)

De 14 kriterier i tabellerne ovenover vil vi bruge som led i vores evaluering og diskussion. Indholdet af tabellerne forholder sig hovedsageligt til udviklingsmiljøer, i vores tilfælde, Xamarin. Vi vil, i senere afsnit, følge evalueringsmetoderne for performance, usability og naturligvis inddrage evalueringsmetoderne fra Heikøtter et. al, for at give et overblik over de nuværende tilgange til cross-platform udvikling og hvordan de forholder sig til tidligere generationer og nativ udvikling.

4. Udvikling

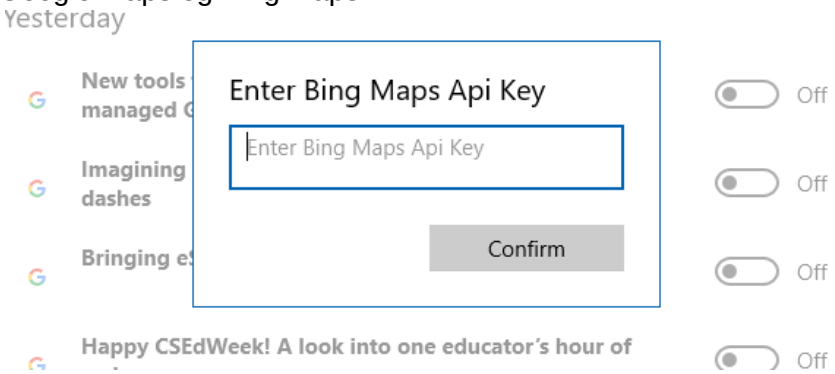
Udviklingen af mobilapplikationen er foregået i faser. Vi har udformet det på denne måde for at sikre en grundig og velovervejet evaluering. Vi vil i dette afsnit gennemgå faserne, og beskrive de mest centrale 'takeaways' fra hver udviklingsfase. Udviklingen er forløbet som beskrevet i afsnit 3.2.1. Vi har udført 5 faser som hver har haft et specifikt mål, med udgangspunkt i at teste cross-platform frameworket Xamarin. Kildekode kan findes i bilag 2.

4.1 Første udviklingsfase

Status
I denne fase var målet at blive familiær med Xamarin. Det involverede at installere Xamarin, samt at lave en test app ud fra en skabelon. Vi forsøgte at bygge kort-viewet, hvis formål var at undersøge adgangen til native features.

Udvikling
Selve installationen af Xamarin gik meget smertefrit; det blev gjort gennem Visual Studio Installer, som inkluderer en standardpakke. At sætte en test app op var også ret nemt. Vi løb dog ind i et par problemer med Android emulatoren. I ét tilfælde var alle UI elementerne alt for store i forhold til skærmen, hvilket gjorde at nogle af billederne i test projektet ikke kunne passe på skærmen, det blev dog løst ved at omkonfigurere emulatoren. Et problem der viste sig ikke at være til at løse så nemt, var at en bærbar ikke har nok ydeevne til at køre en emulator, Visual Studio og bygge en APK på samme tid, hvilket gjorde at det tog meget lang tid (nogle gange mere end 5 minutter) at køre en test efter vi havde lavet ændringer i app'en. Vi kom uden om det problem, ved at teste med UWP udgaven i stedet, da den åbnede meget hurtigere, og så kun åbne i Android en gang imellem for at sikre at det stadig virkede efter hensigten. Dette problem forholder sig naturligvis ikke kun til Xamarin og Visual Studio, problemet ved at køre en emulator sideløbende er et generelt problem ved App udvikling på computere uden høj ydeevne.

Vi anvendte Xamarin.Forms.Maps pakken til at få maps-funktioner på alle platforme, hvilket forløb mere eller mindre smertefrit, da der var en grundig guide at følge i Xamarin dokumentationen. Den mest besværlige del af at bruge denne pakke, var at skulle oprette API nøgler til Google Maps og Bing Maps.



Figur 4.1 - Prompt der beder om Bing API key og gemmer denne i Windows' password vault.

Takeaways

Adgang til Maps

Xamarin indeholder ikke direkte en funktion til kort, men de har en udviklet en pakke, "Xamarin.Forms.Maps" som gør det muligt at bruge hver platforms native API til at vise kort. Apple Maps på iOS, Google Maps på Android og Bing Maps på UWP. Ulempen ved dette er at der er behov for flere API-nøgler som potentielt kan øge omkostningen af udviklingen.

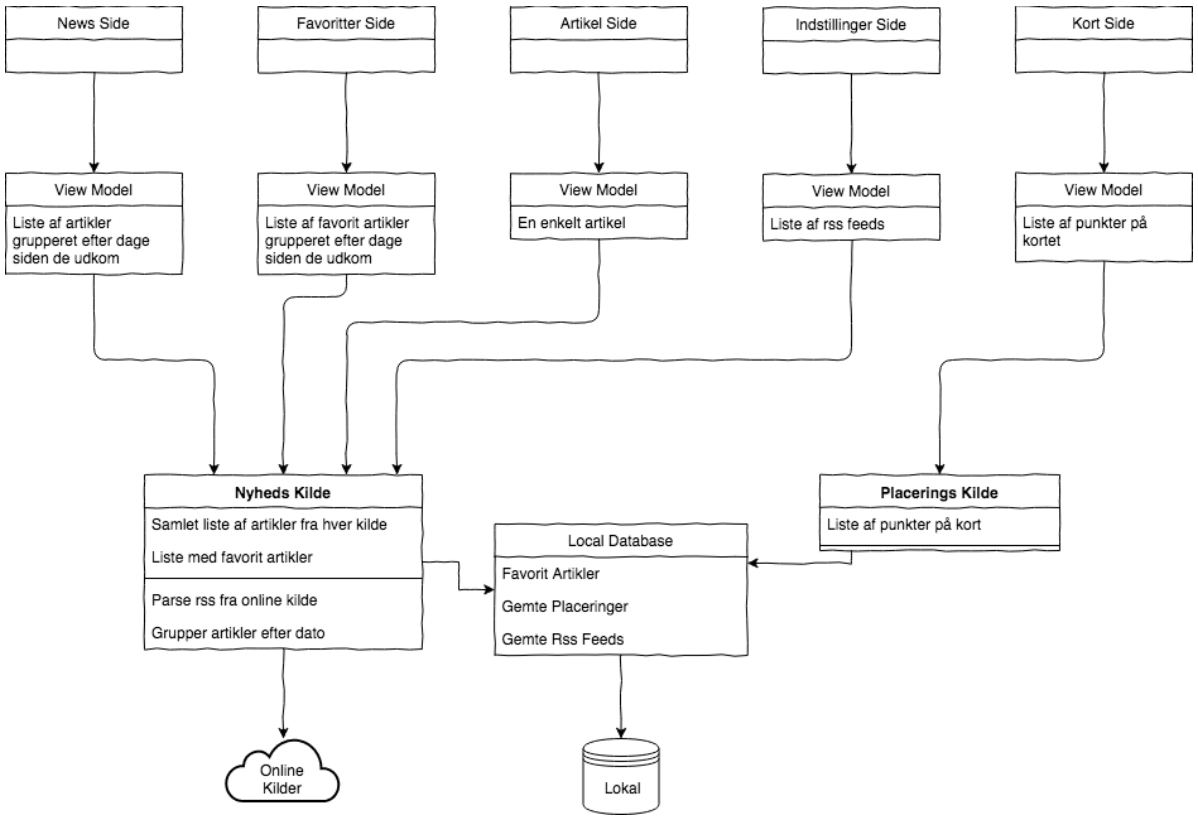
4.2 Anden udviklingsfase

Status

I denne fase var målet at planlægge implementationen af appen, samt at gå i gang med at udvikle features. Vi planlagde at implementere en RSS feed læser, for at teste adgang til data fra eksterne kilder og hvad de krævede at vise denne data på en overskuelig måde.

Udvikling

I den anden udviklingsfase udarbejdede vi modeller der viste opbygningen af applikationen. Vores plan var at teste adgang til native features og hvor godt applikationens performance var, derfor planlagde vi at have en RSS feed viser, for at vise en stor mængde data og et kort. Vi ville hente RSS-filerne fra nettet og gemme enkelte artikler i en lokal database, når brugeren favoriserede dem.

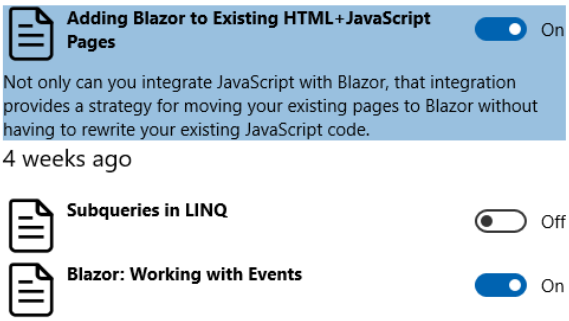


Figur 4.2: En sketch af hvordan vi forestillede os at app'en skulle opbygges.

Figur 4.2 viser vores plan for hvordan vi forestillede os vores app skulle opbygges. Vi konstruerede den med inspiration fra MVVM pattern, da det er det som Xamarin er bygget med henblik på. Pilene viser en idé om hvordan disse dele af programmet burde snakke

med hinanden; Kassen ved roden af pilen snakker til kassen ved spidsen. Det var også i denne fase at vi lavede et mockup af hvordan app'en visuelt skulle opbygges, som kan ses på figur 3.1.

Den første funktionalitet vi fokuserede på, var at gøre app'en i stand til at hente en RSS-fil, omdanne den til en liste af "*Article"* objekter og vise dem i en liste; se figur 4.2 for hvordan det så ud.

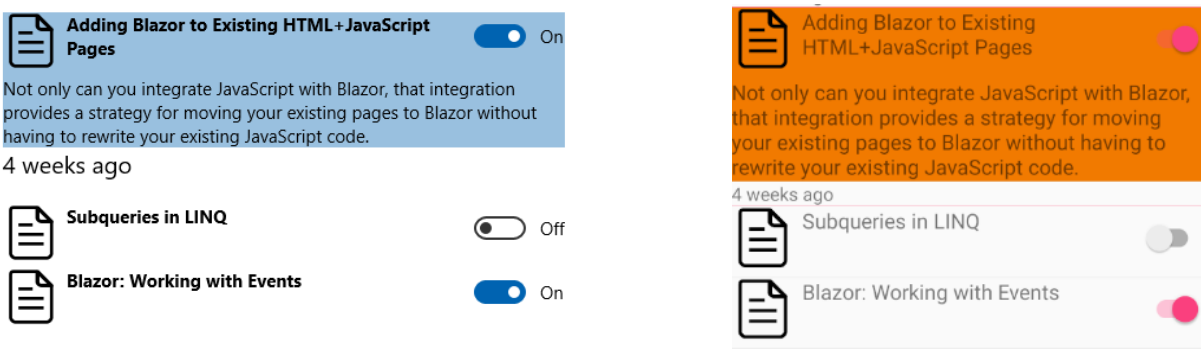


Figur 4.3: Viser hvordan hver artikel ser ud i en liste, med en switch der viser favorit status, til/fra, og hvordan beskrivelsen kun vises når elementet er valgt (illustreret ved farvet baggrund).

Der var to hovedpunkter vi skulle have styr på; hvordan henter og filtrerer vi et RSS-feed og hvordan sætter vi denne data ind i et view. Den første del viste sig at være ret ligetil, da der var et indbygget bibliotek, system.Xml.Linq, til at hente og parse generel XML, som RSS er baseret på. Men der var dog den hage at dette bibliotek ikke var til stede på iOS. Mens system.Xml var tilgængeligt på iOS, så var dette betydeligt mere low level, og ville kræve at vi skrev mere kode for at parse vores fil ordentlig på iOS systemet.

Det andet hovedpunkt var også relativt nemt, men der var dog en erfaring vi stødte på undervejs. For at indsætte data i en liste, brugte vi et ListView. Dette er et element i Xamarin der gør det nemt at vise en liste af elementer, ved at forsyne det med en kilde til data (vores liste af artikler) og en skabelon for hvordan hvert element skal vises. Så langt så godt. Et problem vi stødte ind i var at vi gerne ville gruppere vores data efter udgivelsesdato, som det kan ses på figur 4.3. Det kunne vi godt og ListView understøtter også fint at vise data grupperet på denne måde. Problemet er at vi laver en ny grupperet liste, som ikke opdateres i det den originale liste får ny data. Derfor bliver UI elementerne heller ikke opdateret, da de er forbundet med den grupperede liste. Dette er ikke et uoverkommeligt problem, der findes faktisk mange løsninger til det. På UWP kunne vi bruge en CollectionViewSource, som understøtter filtrering og sortering i sig, mens den holder sig opdateret med den originale liste. På android findes ArrayAdaptor som fungerer på lignende vis. Men i Xamarin er disse utilgængelige. Vi endte med at løse problemet ved manuelt at opdatere ListView'ets datakilde, når der var ændringer. Ved at udvikle cross-platform, var det ikke muligt for os at anvende en simplere løsning der eksisterer i det native framework.

Da vi lavede vores liste af artikler lagde vi mærke til at farverne var forskellige på tværs af platformene; på UWP bliver farverne påvirket af brugeren, i det den farve de har valgt til Windows' bliver reflekteret i app'en, hvor på Android er farverne styret af udvikleren, på figur 4.4 ses de prædefinerede farver for Android, der i vores øjne ikke er optimale til visuel præsentation og letlæselig tekst.



UWP Android

Figur 4.4: Forskellige standard farver på henholdsvis UWP og Android.

Vi stødte ind i et mindre problem med ikonet som ses på figur 4.4, til venstre for hver af artiklerne. Vi havde problemer med at gemme billedet i den delte del af projekterne, og blev i stedet nødt til at gemme det i hvert af de platform specifikke projekter. Med et enkelt billede er der ikke noget stort besvær, men vi hvis vi havde flere kunne skabe problemer hvis vi ville ændre ikonet senere, da vi skulle huske at ændre det tre forskellige steder.

Takeaways

Parsing af RSS

For at kunne vise artiklerne fra en RSS-fil i vores app, blev vi nødt til at parse filen. Vi valgte først at bruge system.Xml.Linq.xDocument klassen til at parse filen, da den er simpel og ligetil. Men det viste sig dog at denne klasse er ikke tilgængelig på iOS.

Filtrering af data

Vi ville gerne bruge den funktionalitet et CollectionViewSource giver os til at binde favorit artiklerne, da det ville gøre det nemmere at holde listerne af artikler opdateret. Problemet er at den kun er tilgængelig i UWP. Derfor er det nødvendigt manuelt at opdatere listerne, når artiklernes favorit boolean ændres. På Android ville løsningen være at bruge en ArrayAdaptor, eller at implementere et Filterable interface.

Forskellige standardindstillinger

De forskellige views har signifikante forskelle i deres udseende fra platform til platform, med hensyn til farve. Dette er forskellen mellem Android og UWP hvor der ikke er gjort noget for at vælge farven. En del af dette ser ud til at være skyldet at på UWP er accent farven styret af brugeren på tværs af apps, hvor på android bliver farven valgt af designeren til det pågældende app. Desuden passer standard farverne nødvendigvis ikke overens med de normale konventioner for den specifikke platform, og dette er noget man selv skal søge.

Lokale billedfiler

Det bliver lettere omstændigt at skulle gemme et billede i appen, da der er forskellige platform konventioner for hvordan denne type ressourcer skal håndteres. Xamarin.Forms kan godt beskrive hvilket billede der skal hentes, uden der er behov for at skrive platform specifik kode, men det samme billede skal duplikeres på tværs af projekter for at det fungerer.

4.3 Tredje udviklingsfase

Status
I denne tredje fase ville vi implementere persistence, i form af en SQLite database. Vi gjorde dette for at teste adgang til native features.

Udvikling
Vores app havde nu en liste af artikler og favoritter, og et kort. Men status for om artiklerne var favorit eller ej blev kun gemt i hukommelsen, så det næste trin var at gemme dem lokalt og derved gøre det muligt for app'en at huske dem fra gang til gang. Måden vi valgte at gøre dette var med en SQLite database. Dette fungerede fint, uden nogen platform specifikke problemer. Vi tilføjede også ikoner ved hvert af artiklerne i listen, for at gøre det nemmere at se hvilken kilde de kommer fra.

Figur 4.5 - Billede af RSS-feed med ikoner af ressource samt et valgt element der viser en kort beskrivelse.

Det næste vi gjorde var at tilføje et link efter artiklens beskrivelse, ses på figur 4.5, som åbnede en ny side der indeholdte et WebView, som åbnede og viste artiklen inde i app'en. Den første gang vi skrev koden til at håndtere når brugeren trykkede på linket, lavede vi en fejl med hvordan metodens signatur var udformet. Resultatet af dette var at app'en ikke korrekt skiftede over til siden med artiklen, men måden den fejlede på var helt forskellig fra UWP til Android; på UWP brød app'en sammen, mens den på Android skiftede til artikel siden, men var ude af stand til at vise artiklen.

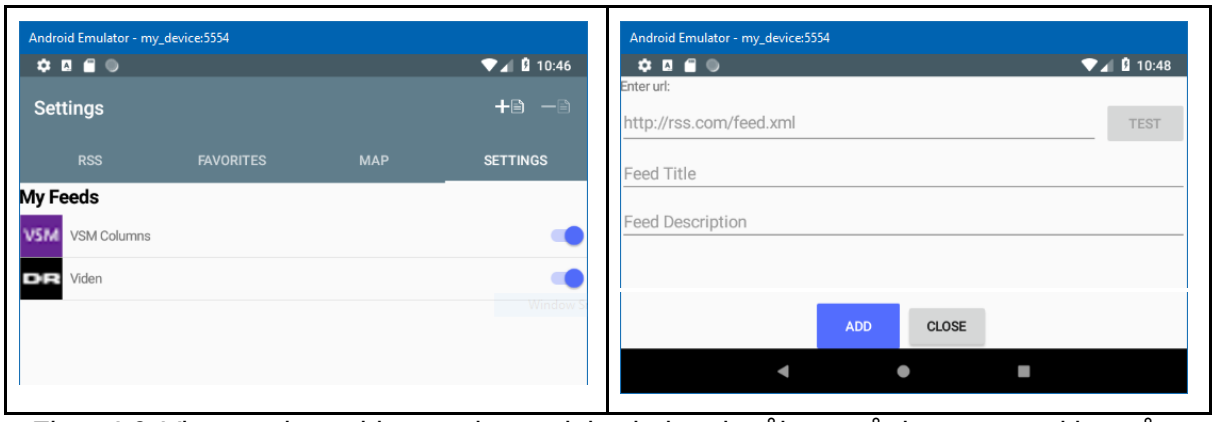
Takeaways
Samme fejl kan have forskellige symptomer på forskellige platforme.
Vi oplevede at en eventhandler som ikke var udformet korrekt, ledte til at UWP udgaven brød helt sammen, mens Android halvvejs udførte opgaven. Dette kan besværliggøre udviklingsprocessen, da udviklerne kan få tvetydige beskeder om hvad forårsager fejl.

4.4 Fjerde udviklingsfase

Status
På dette tidspunkt i projektforløbet anså vi vores Xamarin.Forms app for at være mere eller mindre færdig. Vi manglede at tilføje en *settings* side, hvori brugeren kan give input til eksempelvis hvilke RSS-feeds, der skal anvendes. Desuden tilførte vi et par manglende features til Maps-siden.

Sidste trin var at bygge en tilsvarende app i Android. Dette gjorde vi for senere at danne et sammenligningsgrundlag med et nativt framework især med hensyn til *performance* og forskellen på standard UI konventioner.

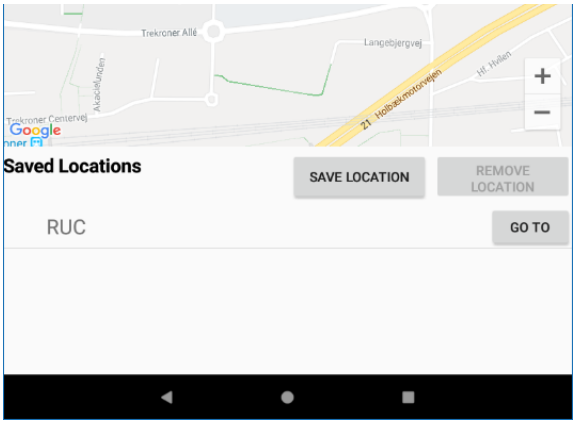
Udvikling
Videreudvikling af vores Xamarin app bestod af en funktionalitet der kunne tilføje og fjerne feeds, samt at slå dem til og fra. Målet med dette var at have en feature der kræver mere input fra brugeren og derved, gøre det muligt for os at teste hvor meget nemt eller svært Xamarin gør det, at bede om input fra brugeren. Vi startede med at implementere en liste der viser feeds, brugeren har gemt i appen, samt en kontakt der indikerer om feed'et er aktivt eller ej. Vi tilføjede knapper i sidens ToolBar, som tillader brugeren at tilføje og slette feeds fra listen. For at tillade brugeren at tilføje feeds, havde vi brug for input, omkring hvilket feed de ville tilføje. Dette implementerede vi via en modal side, se figur 4.6, som kan modtage URL'en på et RSS feed, teste den, og give brugeren et preview af hvad den indeholder.



Figur 4.6: Viser settings siden og det modale vindue der åbnes når brugeren trykker på knappen til at tilføje et nyt feed.

Vi implementerede desuden en funktionalitet til at tilføje markører til kortet på vores kort side. Den består af en liste af placeringer, med et navn. Disse kan tilføjes ved at trykke på knappen "Save Location", som så sætter en markør i midten af kortet. Markørerne kan fjernes ved at vælge placering på listen og derefter trykke på "Remove Location". Ved at trykke på "Go To" knappen ved en location på listen, vil kortet flytte sig over til den placering.

Vi ville gerne kunne placere markører ved at trykke på kortet, men den implementation af kortet der kommer med Xamarin.Forms.Maps ser ikke ud til at inkludere funktionalitet der lader udvikleren vide hvor på kortet brugeren har trykket.



Figur 4.7: Billede af interface hvor brugeren kan tilføje markører til kortet og derved gemme lokationen. De gemte lokationer kan finde i listen og tilgå via "Go To" knappen.

Takeaway

Begrænset Map Funktionalitet

Mapview'et der følger med Xamarin.Forms.Map kan ikke, så vidt vi kunne finde ud af, fortælle hvor på kortet brugeren har trykket, hvilket gør det umuligt at tillade brugeren at sætte markører ved at trykke på kortet.

Små beskeder

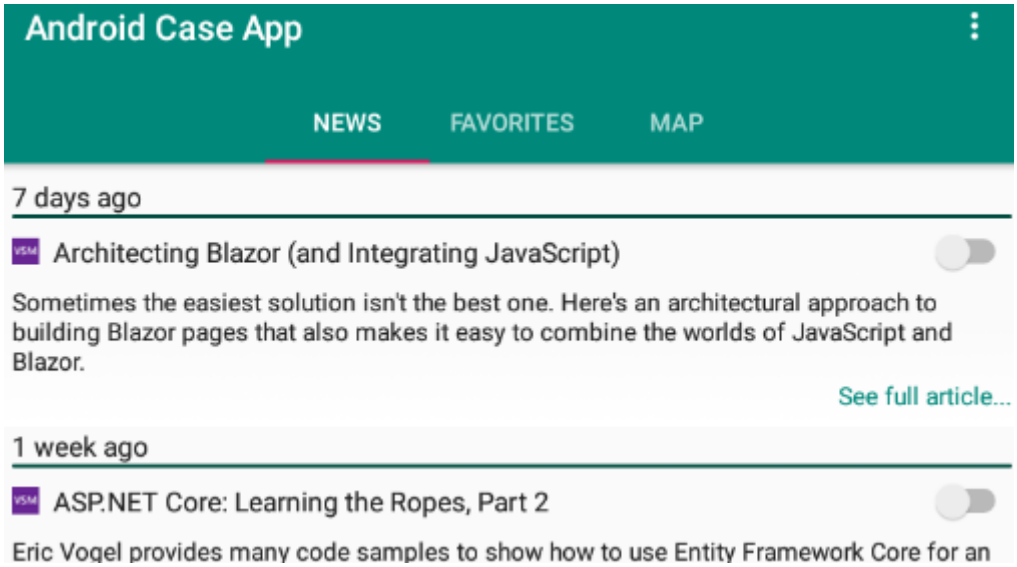
Den indbyggede funktionalitet til at vise små beskeder til brugeren er begrænset. Da vi prøvede at lave en popup der spurgte efter tekst input, fandt vi ud af at det ikke var en mulighed, vi var nødt til at gå til en helt ny side i appen, eller integrerer input felterne i samme side. Vi fandt også ud af at funktionaliteten med at sende Toasts, som kendes fra Android, ikke var tilgængelig, men vi fandt et tredjeparts bibliotek der kunne fylde samme rolle.

Android Nativ Udvikling

Det basale layout, flere faner med hver deres side, var let at duplikere da der var en skabelon der konstruerede det. At implementere kortet var lidt mere kompliceret, grundet to omstændigheder. Der er flere måder at vise et kort i Android end i Xamarin.Forms.Maps og dokumentationen var ikke lige så nem at følge; vi endte med at finde svaret hos en tredjepart.

Implementationen af listen af artikler viste sig at være mere besværlig end forventet. Det indbyggede view til at vise data, som lister, er RecyclerView. Dette view modtager sine data igennem en Adaptor. Problemet med dette er at der ikke er indbygget funktionalitet til at vise grupperet data.

Selve performance testen blev også udført i denne fase, resultatet af denne test vil gennemgås i afsnit 5.1.



Figur 4.8: Billede af RSS feed på Android applikationen.

Takeaway

Nogle funktionaliteter er mere begrænsede i Android end i Xamarin.Forms

Da vi skulle implementere listen med artikler, som er grupperet efter relativ dato, fandt vi at den native implementation af RecyclerView, som er en mere moderne ListView, ikke understøtter grupperet data. Det var derfor nødvendigt enten at selv at implementere en RecyclerView.Adapter som kan håndtere grupperet data, eller gøre brug af et tredjeparts bibliotek. Vi valgte at bruge SectionedRecyclerViewAdapter.

4.5 Usability optimering

Den sidste del af vores vores produktudvikling drejede sig om at optimere usability i applikationen ud fra usability principperne fremlagt af Nielsen vi gennemgik i afsnit 3.2.2. Vi gennemgik de forskellige views og funktioner i vores applikation, for at vurdere hvor vi kunne forbedre usability og generel opbygning af appen.
Den udvikling vi udførte er opsummeret i listen nedenunder og ligeledes har vi angivet hvilken usability kvalitet vi mener ændringen kan have effekt på.

Funktionalitet	**Forventet effekt**
Expand/collapse funktion på artikler i newsfeed Artikler giver en teaser tekst og viser linket "gå til artikel", efter at være klikket på.	Formålet med dette er at **minimere mængden af information brugeren skal holde styr på**. Brugeren bliver kun udsat for titlen på hver artikel, og skal selv vælge at vide mere.
Swipe ned refresh (udover den eksisterende "Refresh" knap) En typisk anvendt metode til at opdatere lister og sider, er ved at swipe ned fra toppen af siden.	Skal forbedre **memorability** og **learnability** i og med at brugerne kan drage på deres erfaring fra andre apps på samme platform hvor denne feature anvendes i samme tilfælde.
Nyt favorit ikon I stedet for en on/off knap skal favorit funktionen i stedet laves til det generiske stjerne ikon.	Dette skal søge at overholde **normale konventioner** for favorit-knapper, samt tillade brugere at **anvende deres forforståelse** af stjerne-ikoner i en applikation
Sende 'Toasts' En typisk feature i Android er at give brugeren små beskeder når de har foretaget sig en handling, som bekræfter at den har taget kraft.	Formålet er at give brugeren **feedback på handlinger** hvor det kunne være nødvendigt, for eksempel når de sletter et element, sendes en bekræftelse på at det er slettet.
Standardfarver Standardfarven der bliver brugt i Android udgaven til markerede elementer i ListView er en meget kraftig orange, der gør den sorte tekst svær at læse. Den skal være mere neutral.	Dette skal søge at overholde **normale konventioner** for markering af valgte elementer med hensyn til farve, og forbedre læsbarheden af listens elementer.
Bekræftelse dialog Efter at have trykket på knappen til at slette et objekt, skal der komme en boks op og bede brugeren om at bekræfte handlingen.	Hjælper med at skabe **Simpel og naturlig dialog** og **forebygge fejl** fra brugeren, da de kan komme til at trykke forkert og derfor er det vigtigt at bede om bekræftelse før en ireversibel handling.

5. Evaluering

Vi vil, i dette afsnit, gennemføre en performance test og en usability inspektion, her vil vi undersøge hvorvidt vi har udviklet en applikation der er let og hurtig at bruge, og i hvor høj grad applikationen føles *nativ*. Vi vil derefter benytte vores udvalgte evalueringsmetoder, til at bedømme hvorvidt cross-platform udvikling og Xamarin, er en god fremgangsmåde til at udvikle mobilapplikationer til flere platforme.

5.1 Performance test

Vi sammenlignede på tre punkter: Hvor lang tid tager det for appen at starte, hvor meget plads bruger appen i hukommelsen og hvor meget RAM bruger appen. Vi testede først tiden det tog for appen at åbne. Dette gjorde vi ved hjælp af en indbygget funktion i Android, som fortæller os hvor mange millisekunder der går fra appens process starter, til den første Activity er åbnet og UI'et er klar til at modtage input (Android Developer[2], 2018).

Vi testede warm og cold boot. Vi startede en warm boot ved at lukke app'en med tilbage-knappen, og derefter åbner den igen med det samme ved at trykke på app ikonet på startskærmen. Cold boot startede vi ved at lukke appen i joblisten (listen af app's der kommer frem når man trykker på app switcher knappen), og så starte den igen ved at trykke på ikonet på startskærmen. Vi testede både på emulator og en fysisk telefon. Tallene som ses i tabellen herunder er et gennemsnit af 14 målinger.

Test af opstartshastigheder ***(Android)***					
Emulator	Xamarin .app	Android native app	**Samsung Galaxy S8**	Xamarin .app	Android native app
Warm	1883,5 ms	458,4 ms	Warm	570,2 ms	167,2 ms
Cold	8200,0 ms	1831,4 ms	Cold	3326,5 ms	371,0 ms

Tabel 5.1: Tabel der viser gennemsnitlig opstartstid af vores test apps på Android.

På emulator fandt vi at Xamarin.Forms app'en tog ~4.5 gange så lang tid at starte ved cold boot og ~4.1 gange så lang tid at starte ved warm boot. På den fysiske enhed tog den ~9 gange så lang tid at starte ved cold boot og ~3.4 gange så lang tid at starte ved warm boot.

Derefter målte vi hvor meget hukommelse app'en brugte, både i RAM og lagerplads. Vi fandt dataen inden i indstillinger for App'en i Android systemet.

Hukommelse ***(Android)***		
Type	Xamarin.app	Android native app
App	17,13 MB	7,85 MB
Gns. forbrug af RAM	15,00 MB	1,60 MB

Tabel 5.2: Tabel der viser pladsforbrug i flash hukommelsen og gennemsnitlig forbrug af RAM af vores test apps på Android.

Vi ser her at Xamarin.Forms app'en fyldte 10 MB mere end Android app'en, og i gennemsnit brugte 13.4 MB mere RAM end den native udgave. Det skal dog bemærkes at en del af dette ekstra forbrug ikke skalerer med app'ens størrelsen (Microsoft [2], 2018), men snarere kan ses som en fast omkostning ved at bruge Xamarin.Forms.

Xamarin.Forms er tungere end native Android
App'en udviklet i Xamarin.Forms tager betydeligt længere tid at starte, end en tilsvarende app på Android. Den fylder også lidt mere, men ikke så meget at det vil være et problem på moderne telefoner.

5.2 Usability inspektion

Evalueringen af de usability tiltag vi har udviklet i vores applikation, baserer vi på en usability inspektion. Inspektionen blev udført med os selv som evaluatorer, og vores ekspertise var udviklingen af applikationen. Yderligere blev en ekstra evaluator anvendt, hvis kendskab til applikationen var begrænset.

Inspektionen foregik ved at hver evaluatorer blev præsenteret for en række opgaver de skulle prøve at udføre, samtidig med at de vurderede elementernes usability ud fra udvalgte principper. I afsnit 3.2.2 præsenterede vi ti usability principper og ud af disse har vi udvalgt seks, der skal indgå i vurderingen under inspektionen.

- **Simpel og naturlig dialog**
- **Anvend brugernes sprog**
- **Minimér mængden af information brugerne skal huske.**
- *Konsistens*
- **Feedback**
- **Klart definerede udveje**
- *Genveje*
- *Præcise og konstruktive fejlbeskeder*
- **Forhindre fejl**
- *Hjælp og dokumentation*

Vi har udvalgt disse baseret på de tiltag vi har taget for at optimere usability i vores app, samt de beslutninger vi tog for at teste usability-principperne i Xamarin.

Evaluatorerne gennemgik opgaverne opstillet i tabellen nedenunder. Hver opgave blev givet en vurdering af letheden ved udførelse. Skalaen for vurdering løber fra 1-5; 1 = meget svær, 5 = meget let. Yderligere kunne opgaven gives betegnelsen “U” skulle det være umuligt for evaluatoren at løse den.

Nr.	Opgavebeskrivelse
1	Find en artikel i RSS feedet og tilgå den. Derefter gå tilbage til RSS feed.
2	Angiv 5 favorit artikler fra forskellige feed udbydere (DR, Google osv.)
3	Find og fjern favorit-markering fra favorit artikler.
4	Gør dit feed personligt ved at tilføje nye RSS feeds.
5	Slå feeds til og fra efter ønske, derefter slet et af de eksisterende feeds.
6	Gem to lokationer på kortet. Centrer på første lokation og slet den anden gemte lokation.
7	Find din hjemby på kortet.

Inspektionen med os, udviklerne, som evaluatorer blev udført ved at vi, isoleret fra hinanden gennemgik applikationen og udførte de nævnte opgaver, mens vi noterede vores oplevelse, samt overordnede dom.

Den sidste inspektion, blev ligeledes udført ved at evaluatoren skulle udføre de samme opgaver, men denne gang var vi tilstede og noterede evaluatorens færden i applikationen.

5.2.1 Inspektion resultater

Resultatet af usability inspektionen ses på tabel 5.3. Scorene evaluatorerne har givet er blevet farvekodet fra rød til gul til grøn. Rød er dårligt, gul er neutralt og grøn er godt. De fulde skemaer af noter og scoringer fra hver evaluator kan findes i bilag 1.1-1.4.

Ud fra farvekoderne i tabellen er det tydeligt at se at der er steder flere steder hvor applikationen kunne forbedres mht. lethed ved anvendelse. Inspektionen udført af hver evaluator påpeger også vigtigheden i at inkludere et flertal af testpersoner, da man oftest lægger mærke til og vægt på forskellige aspekter af et brugerinterface. Ved gennemgang af noterne fra inspektionen samt de angivne score, har vi opsummeret de største problematikker og forsøgt at komme med forslag til hvordan disse problematikker kunne forbedres.

Nr.	Opgavebeskrivelse	EV1	EV2	EV3	EV4
1	Find en artikel i RSS feed og tilgå den. Sæt artiklen som favorit. Derefter gå tilbage til RSS feed.	4	5	4	5
2	Angiv 5 favorit artikler fra forskellige feed udbydere (DR, Google osv.)	4	5	U	5
3	Find og fjern favorit-markering fra favorit artikler.	2	5	5	4
4	Gør dit feed personligt ved at tilføje nye RSS feeds.	1	3	4	1
5	Slå feeds til og fra efter ønske, derefter slet et af de eksisterende feeds.	5	5	5	4
6	Gem to lokationer på kortet. Centrer på første lokation og slet den anden gemte lokation.	3	4	2	3
7	Find din hjemby på kortet.	4	4	3	3

Tabel 5.3 - Opsamling og farvekode af inspektions score fra de fire evaluatorer.

De overordnede problematikker vi har fundet ved inspektionen er som følger:

Problematik 1 (mangel på feedback)
Når brugeren har angivet nye favorit artikler og derefter tilgår favoritlisten, skal brugeren manuelt opdatere siden for at de nye favoritter bliver synlige. Dette kan skabe frustration for brugeren, hvis de ikke forstår hvorfor der mangler favoritter på deres favoritliste. Det er ikke intuitivt at man selv skal opdatere en liste man lige er skiftet til (view til view).

> **Løsning**
> Favoritlisten skal opdateres automatisk når brugeren skifter til favorit-viewet.

Problematik 2 (uklar kommunikation til brugeren)
Listen af news indhenter artikler fra alle de news feeds brugeren har valgt. Det der visuelt differentierer artiklernes oprindelse fra hinanden er udelukkende udbyderens billede. Hvis brugeren har valgt at få artikler fra to forskellige news feeds fra samme udbyder (eks. DR viden og DR Sport), kan der ikke skelnes mellem hvor artiklen oprinder fra.

> **Løsning**
> Navnet på feed skal inkorporeres i visningen af artikler. Navnet på feedet artiklen stammer fra, skal indgå i informationen om hver artikel. Dette kunne gøres i form af små “tags” eller ved at have feednavnet over titlen på artiklen.

Problematik 3 (mangel på feedback, udenfor brugerens sprog og forståelse, mangel på fejlbeskeder)
Vores løsning til at lade brugeren tilføje feeds er ikke optimal. Først og fremmest er de simple tjek vi har lavet for at sikre det er en korrekt URL, der bliver indtastet, ikke tilstrækkelig for at undgå fejl. Yderligere blev der fundet problemer ved flowet i, at tjekke hvorvidt den indtastede URL leder til et RSS feed. Hvis det er et korrekt RSS feed som brugeren har indtastet, vil "TEST" knappen fremvise et preview af de første artikler i feedet. Hvis URL'en ikke er valid, får brugeren ingen feedback.

Løsning 1
Dette problem kan afhjælpes ved at sørge for et mere dybdegående tjek for fejl i forbindelse med den indtastede URL. Ydermere skal der kommunikeres klart til brugeren om hvorvidt den indtastede URL kan bruges til at tilføje et feed. Tilføj knappen bør være inaktiv indtil at appen har vurderet URL'en som valid, hvilket også vil give klar besked til brugeren om at de nu kan tilføje feedet.

Løsning 2
Hvis vi ser kritisk på denne funktion, er den desværre præget af at man skal være meget teknisk anlagt for at forstå hvordan man skal anvende den. Ligeledes har den almene bruger sjældent kendskab til URL adresser der kan anvendes til feeds og hvem der overhovedet udbyder dem. Hvis målgruppen for appen er den almene bruger, kunne man indsamle en generel liste af de mest populære feeds og tillade brugeren at vælge blandt dem. Dette ville indebære at vi selv sørger for at URL'erne er korrekte og at brugeren kun skulle tage stilling til navnet på feedet og at vælge det ønskede feed.

Problematik 4 (ikke konsistent med brugerens forforståelse, uklar information)
På map-viewet var det ikke intuitivt at oprette lokationer. I og med at vi anvender de native maps fra hver platform, trækker folk fra deres eksisterende erfaring, der fortæller dem at de burde være i stand til at sætte markører ved at klikke på kortet. Samme problemstilling gjorde sig synlig når en af evaluatorerne skulle slette lokationer de havde oprettet. En evaluator prøvede at markere de pins der var synlige på mappet, men dette gjorde ikke "Remove" knappen aktiverbar. Yderligere kan det påpeges at de gemte lokationer ikke kan skelnes fra hinanden i ListView, da de alle får samme navn.

Løsning
Det ville være optimalt og forbinde funktionaliteten på kortet med den i listen. Hvis brugeren klikker på kortet kan de oprette en pin, og hvis de vælger en markør på kortet, bliver den samme markør markeret i listen nedenunder.

Mht. til skelnen mellem markørerne i listen kunne brugeren selv ændre navn på markøren. Ligeledes kunne der vises ekstra viden om markøren, f.eks længde- og breddegrader.

Problematik 5 (mangel på fejlbeskeder)
Ved et for højt antal markører på kortet, er det ikke muligt at tilføje flere markører på trods af at systemet giver besked om at markørerne bliver oprettet.

Løsning
En mulighed er at have en begrænsning på antallet af markører og hvor denne begrænsning skal kommunikeres til brugeren. Skulle brugeren prøve at angive en markør på trods af at de er ved det maksimale antal, kunne en dialog præsenteres for brugeren hvor de kan vælge fremgangsmåde for at komme videre i processen.

Problematik 6 (mangel på forventet funktionalitet)
Manglen på søgefunktion på kortet påpegede flere evaluatorer da de nåede til opgaven "Find din hjemby på kortet". Navigation rundt på kortet virkede som forventet og efter konventionerne, men opgaven var ikke nødvendigvis hurtig at løse.

Løsning
Det kan diskuteres om hvorvidt det er meningen at man selv skal navigere rundt på kortet, men en søgefunktion kunne optimere brugerens tilfredshed ved anvendelse af applikationen.

Ved at benytte usability inspektion forventer vi at have fundet de mest centrale og fremtrædende problemstillinger. I en videreudvikling af applikationen ville de ovenstående problematikker være i fokus. Siden at evalueringen er foretaget af personer med kendskab til applikationen erkender vi at evalueringen, i en vis grad, er overfladisk. Vi mener dog, at testen passer overens med hvor langt vi er nået i udviklingsprocessen, da det ønskede formål med metoden er at finde åbenlyse fejl og mangler, frem for at tilpasse til en målgruppes præferencer. Hvis vi befandt os i en senere fase af udviklingen, ville en mere bruger-centreret usability evaluering være passende.

5.3 Evalueringskriterier

Vi benytter, som nævnt i afsnit 3.3.1 og 3.3.2, Heitkötters evalueringskriterier (Heitkötter et. al, 2013). Vi vil her kort beskrive vores erfaring med frameworket Xamarin, ud fra punkterne Heitkötter fremhæver. Vi forsøger at være korte og konkrete når vi beskriver evalueringskriterierne, således at et overblik kan dannes og ud fra dette en konklusion. Vi diskuterer og beskriver mere dybdegående disse kriterier i sammenhæng med takeaways i afsnit 6.

5.3.1 Infrastruktur perspektiv

1 **Licens og pris**
Xamarin er open source og derfor frit tilgængeligt. Det er dog påkrævet at anvende Visual Studio IDE for at arbejde med Xamarin. Visual Studio udbydes som gratis licens, men denne kommer med restriktioner. Man kan købe abonnementer for at undgå disse restriktioner. Prissætningen varierer fra $500 - $6000 /årligt (Visual Studio [2], 2018). I forhold til sin 3.generations modpart ReactNative og det native Android Studio, der begge er open-source og fuldt ud gratis, er det en markant omkostning der skal overvejes. Visual Studio og Xamarin tilbyder forskellige support pakker, som enten er med i abonnementet eller kan tilkøbes.

2 **Understøttede platforme**
Xamarin understøtter de to markedsdominerende platforme Android og iOS, samt den mindre Windows (phone).

3 Adgang til platform specifikke features
Xamarin.Android og Xamarin.iOS, tillader udvikleren at skrive nativt til platformene, og har adgang til næsten alle platform specifikke features. Xamarin.Forms er Xamarins cross-platform orienterede produkt. Xamarin.Forms har adgang til de fleste platform specifikke features, men i lavere grad end de native frameworks. Udviklere kan dog benytte de native og cross-platform produkter i Xamarin sammen. Dette resulterer i et meget solidt og alsidigt cross-platform framework.

4 Langsigtet gennemførlighed
Xamarin er ejet af Microsoft og er et framework til det, allerede populære IDE, Visual Studio. Frameworket bliver bl.a. af denne grund jævnligt opdateret (Xamarin Developer [2], 2018). Xamarins popularitet er stigende, ifølge en undersøgelse af hjemmesiden StackOverflow, ligger frameworket på en 9. plads af mest de populære frameworks (StackOverflow, 2018). Frameworket har et godt udvikler community og det er let at finde svar på eventuelle udviklingsbarrierer man støder på.

5 Udseende og følelse
Det er native UI-elementer der præsenteres for brugeren, ved anvendelsen af Xamarin. Opbygningen af brugerinterface følger de samme standarder som der er på de native platforme. De specifikke platformes UI-elementer kan opbygges separat i Xamarin.Android/iOS og derved opnå komplet nativt udseende og funktionalitet, eller man kan anvende Xamarin.Forms og derved få 95% delt kode. Det er en vurderingssag i hvor stor en grad man vægter cross-platform udvikling eller nativt interface (relateret til usability).

6 Performance
Applikationer fylder mere ved brug af Xamarin, dette skyldes at der medfølger nogle obligatoriske biblioteker, som skal sikre at applikationen virker og er sikre på den pågældende platform (Microsoft [2], 2018). Hvad angår hastighed, ser vi at Xamarin ikke lever op til nativ standard, opstartstider er betydeligt langsommere end native applikationer.

7 Distribuering
Distribuering foregår via henholdsvis App Store, Play Store eller Microsoft Store, og har derfor de givne restriktioner for hver platform.

5.3.2 Udvikling perspektiv

1 Udviklingsmiljø
Udviklingsmiljøet er Visual Studio med frameworket Xamarin. I og med at begge er udviklet og vedligeholdt af Microsoft kan en vis kvalitetsgrad forventes. IDE'et er komplet med hensyn til features. Der er adgang til Emulator både UWP og Android, debugger, autocompletion osv. Det er muligt at betale for cloud-løsninger og ekstra debugging værktøjer.

2 GUI design
Kompleksiteten af UI udvikling i Xamarin var på niveau med native miljøer såsom Android Studio. Xamarin indeholder en brugerflade-editor som gør det muligt at *drag n' drop* UI-elementer. At have brugerflade-editoren, gør det nemmere at lave små ændringer i UI'ets opbygning, da man ikke er nødsaget til at bygge og overføre applikationen til emulatoren eller telefonen, for at se resultatet af ens arbejde.

3 Lethed af udvikling
Siden udvikling foregår i C# og Visual Studio, kan der findes store mængder dokumentation på de fleste problemstillinger som en udvikler kunne støde på omkring disse aspekter. Hvad angår den mere Xamarin specifikke udvikling såsom XAML, viewmodels osv., er der ingen grund til bekymring, der findes rigelig information på Xamarins forum og sider som StackOverflow, på trods af redskabet stadig er relativt nyt. Microsoft har lagt et stort arbejde i at gøre deres Xamarin dokumentation detaljeret og brugervenlig.

4 Vedligeholdelsesevne
Umiddelbart er der markant flere elementer og linjer kode i Xamarin i forhold til native udviklingsmiljøer. Selv den mindste Xamarin app vil have en del linjer kode og ressourcer til at starte med. Dette er b.a. grundet at de mange projekter der som udgangspunkt er tilstede ved cross-platform udvikling (Xamarin.Forms, Xamarin.iOS, Xamarin.Android osv). Vi mener dog at Xamarins projektstruktur er god, og derfor skaber de mange klasser og biblioteker, der følger med Xamarin ikke uoverskuelighed. En anden fordel ved vedligeholdelse i Xamarin, er at en stor del af koden kan holdes i den samlede kodebase, og derfor behøves det ikke vedligeholdes til specifikke platforme.

5 Skalérbarhed
Det er muligt at opdele projektet i mindre dele og have større og flere hold af udviklere arbejde sideløbende. Dette gør sig især gældende hvis man sørger for at vælge et godt design til ens app struktur (f.eks MVVM). Visual Studio understøtter forskellige git-løsninger, der kan hjælpe til samarbejde og version kontrol i et større team.

6 Mulighed for videreudvikling
Skulle man ende i et problem hvor man ikke kan komme videre, ved brug af Xamarin frameworket, kan man forvente en lav kode genbrugelighed. Ingen native miljøer understøtter C#, og ligeledes heller ingen cross-platformværktøjer som vi har kendskab til.

7 Hastighed og pris for udvikling
Udvikling i Xamarin foregår hurtigt og effektivt, især hvis udviklingsteamet er erfarne i C# før påbegyndelse. I sammenligning med native miljøer kan cross-platform tilgange generelt være omkostningseffektiv og her er hurtig og Xamarin er ingen undtagelse.

6. Diskussion

I tidligere studier om cross-platform var den generelle konklusion, at de daværende cross-platformframeworks var et holdbart valg til udviklingen af mobilapplikationer. Ved anvendelsen af cross-platform frameworks blev det dog gjort tydeligt, at det betød kompromisser skulle tages andetsteds. Ofte var konklusionen at man ikke kan opnå samme UX i cross-platform miljøer som man kan i det respektive native miljø. Yderligere blev ringere performance også anset som et kompromis der skulle tages, for at få en samlet kodebase. Her henviser vi til artiklen af Angulo & Ferre (2014), samt artiklen af Xanthopoulos & Xinogalos (2013).

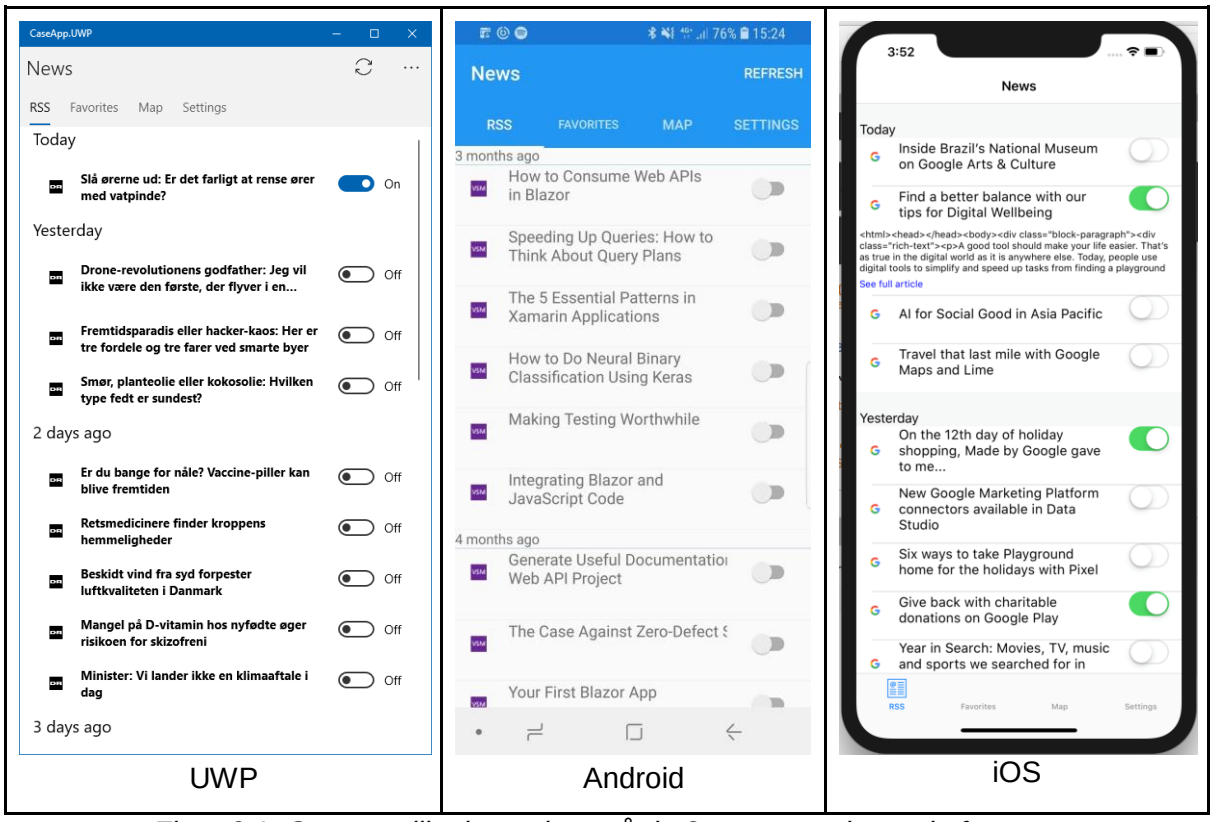
I dette projekt har vi undersøgt de samme problematikker der er blevet fremlagt i tidligere undersøgelser, for at se om de stadig gør sig gældende for cross-platform udvikling i dag. I de frameworks, vi anser som værende den seneste generation af frameworks inden for cross-platform udvikling, er den generelle opfattelse at de ikke bærer de samme problematikker som den forrige generation. Undersøgelsen er foregået ved at udvikle en app/case som netop tager hånd om de problemstillinger der er blevet fremhævet ved tidligere undersøgelser. På denne måde har vi fundet frem til hvorvidt tidligere problemer ved cross-platform udvikling er løst, samt hvilke problemstillinger der kan findes ved nuværende løsninger til cross-platform tilgangen. Vi har fremhævet disse problemstillinger i afsnit 4.1 som 'takeaways'.

Vi gør opmærksomme på at omfanget af vores case er begrænset, vi kan derfor ikke konkludere endegyldigt hvorvidt Xamarin kan bruges til alle projektformer eller formål. Vi kan dog vurdere at i vores tilfælde (ved udvikling af vores case), at Xamarin er et effektivt framework som, på trods af nogle få udfordringer, sagtens kan benyttes til at udvikle en mobil applikation med casens formål. Vi vurderer også at Xamarin har klare fordele hvad angår omkostninger og tidsbesparelser i forhold til at udvikle i native frameworks, hvis applikationen skal udgives til flere platforme. Derimod hvis applikationen skal udgives til én platform, kan det være fordelagtigt at udvikle i et nativt miljø såsom Android Studio, især hvis der lægges fokus på *performance.* En grund til at vælge Xamarin selvom der kun udvikles til én platform, kunne være de mange features og udviklingsredskaber, der følger med Xamarin og Visual Studio. I flere tilfælde har vi stødt på features som ikke findes i de native miljøer, men som kan benyttes i Xamarin, eksempelvis grupperede ListView.

Hvad angår usability ser vi også klare forbedringer i forhold til tidligere generationer af cross-platform udvikling. Mobilapplikationer udviklet i 2. generations cross-platform udviklingsredskaber varierede signifikant i forhold til udseende og *følelse*. Typisk var navigation og placering af UI elementer anderledes i forhold til native applikationer. Dette skyldes bl.a. at udviklingsmiljøerne ikke indeholdte funktionalitet der automatiserede eller hjalp udviklerne til at overholde platform specifikke konventioner. Disse punkter er blevet pointeret i tidligere studier, som bl.a. er blev gennemgået i afsnit 2.1.1. Vi mener at problemstillingerne ved dette forholder sig tæt til usability egenskaberne *learnability* og *memorability,* da brugerne ikke, i samme grad, kan drage på erfaringer fra brugen af andre applikationer og på denne måde, have svært ved at benytte deres pre-eksisterende viden. Desuden kan et rodet, uigenkendeligt interface gøre det sværere for brugerne at huske hvordan navigationen foregår i den pågældende applikation. Vi ser, ved 3. generations

cross-platform værktøjer, at der bliver lagt større fokus på disse usability egenskaber og frameworks såsom Xamarin, indeholder mekanismer der sørger for at konventioner på tværs af platforme bliver overholdt.

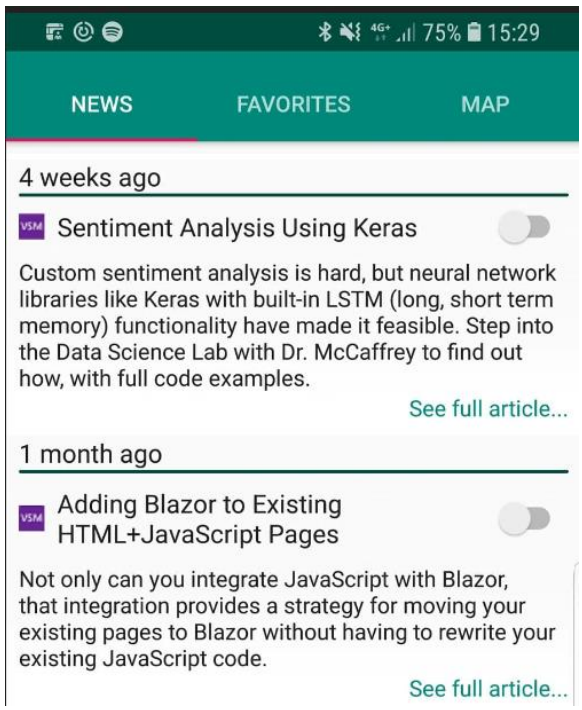
Vores usability inspektion resulterede i en markant liste af usability egenskaber der kunne forbedres i vores applikation. Hoveddelen af disse var dog ikke prægede af at Xamarin framework ikke tillod at udvikle løsningen, men derimod var det os udviklere der havde overset problematikkerne. Inspektionen viste sig at være værdifuld for vores projekt og gav den ønskede form for evaluering af produktet. Vi påpeger dog at der i inspektionen med os selv som evaluatorer, var en klar fornemmelse af vores eksisterende viden om applikationen og dens funktioner ikke hjalp med at finde alle problematikker, der kunne være til stede. I en situation, som den vi satte os selv i ved inspektionen, er det meget svært at forholde sig uvidende til systemet og agere efter sin normale intuition. Metoden er værdifuld at inkorporere i udviklingsprocessen, for at udbedre de største problemer, før en reel user-testing skulle foregå, men den kan ikke anvendes enkeltstående i et større og længere projekt.



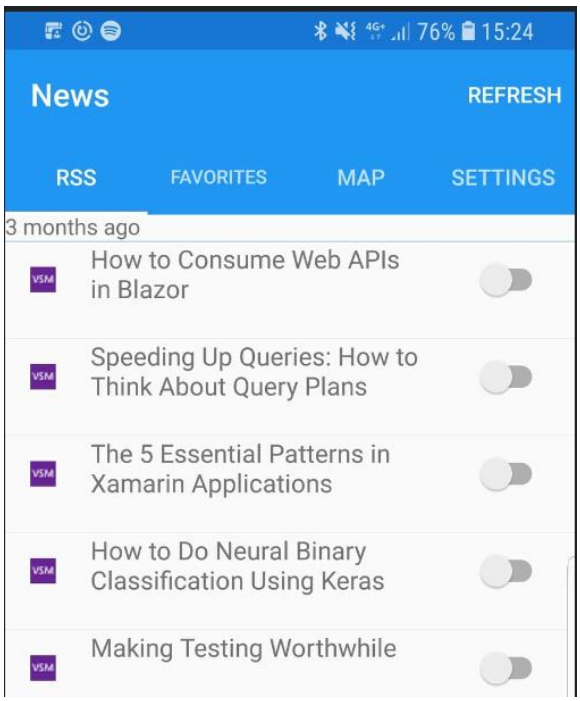
Figur 6.1: Case applikationen kørt på de 3 mest populære platforme

Vi ser, på figur 6.1, at der er forskel på udseende af UI-elementer, farvetema osv. Dette er grundet brugen af native elementer da kode skrevet i Xamarin bliver *interpreted* til nativ kode, og på denne måde kan gøre brug af nativ UI. Der er også mindre forskelle i funktionaliteterne på tværs af platformene, et eksempel kan findes hos ListViewet. UWP indeholder en feature der sikrer at man altid kan se grupperings værdien og fordi koden bliver oversat kan featuren stadig benyttes på trods den ikke er tilgængelig ved alle platforme. På denne måde bliver der, i mange tilfælde, ikke gået på kompromis med native features. Med hensyn til farvetemaer og platform specifikke konventioner, så er det stadig

nødvendigt at tilpasse UI elementer til den specifikke platform, for at skabe en mere nøjagtigt nativ oplevelse.



Nativ Android



Xamarin.Android

6.2: Nativ Android og tilsvarende applikation i Xamarin.Android

Vi kan se at farvetemaet mellem nativ Android og Xamarin.Android varierer fra hinanden. Det er naturligvis muligt at tilpasse til den specifikke platform i Xamarin, men dette åbner for en ny problemstilling. Der er, på nuværende tidspunkt, to muligheder for platform specifik UI tilpasning, enten kan det skrives platform specifikt i det native projekt (Xamarin.Android, Xamarin.iOS), eller det kan skrives i den samlede kodebase hvis der bliver specificeret hvilken platform den aktuelle kode skal køres i. Der er problemer ved begge løsninger. Problemet ved at skrive platform specifikt er at nedsætte formålet ved cross-platform udvikling. Problemet ved at skrive i den samlede kodebase, er at mængden af kode drastisk bliver forøget. Dette kan have effekt på overskueligheden og ydeevnen af applikationen.

Det er sandsynligt, men ikke sikkert, at ved udviklingen af mobilapplikationer vha. Xamarin at man vil støde ind i cross-platform relaterede problemstillinger. Hvor ofte, er afhængig af flere aspekter såsom, i hvor høj grad applikationen skal føles *nativ,* mængden af native features osv. Desuden ser vi også inkonsistente fejlmeddelelser og opførsel på tværs af platforme, hvilket bidrager til en mere kompleks og tidskrævende udviklingsproces. Vi mener dog stadig at fordelene opvejer ulemperne, og at cross-platform udvikling kan tilbyde markante tidsbesparelse med *minimale* kompromiser, som bl.a. afhænger af valget af cross-platform framework.

7. Konklusion

Det er tydeligt at se, hvor der tidligere har været problemer med cross-platform udvikling. Flere studier viser at der var problemer i forhold til *performance* og usability-egenskaber, vi betegner som *learnability* eller *memorability.* Dette skyldtes at applikationerne udviklet i et cross-platform redskab, så ud og *føltes* anderledes end nativt udviklede mobilapplikationer. Vi ser specielt disse problemer ved, hvad vi kategoriserer som 2. generation af cross-platform udviklingsredskaber, hvor fokus primært var at samle kodebasen mellem flere platforme, på trods af en række usability kompromiser. Vi har set en stor udvikling af cross-platform redskaber siden disse studier blev udført, og vi har nu adgang til, hvad vi betegner som, 3. generations værktøjer såsom Xamarin. Med de nyere redskaber har vi, i højere grad, adgang til native features, bedre performance og større fokus på usability.

I denne undersøgelse har vi udviklet en mobil applikation i cross-platform frameworket Xamarin. Fokus har været at teste, hvorvidt de mest centrale problemstillinger fra 1. og 2. generations udviklingsredskaber er blevet forbedret. Resultatet af undersøgelsen viser at der er sket store fremskridt inden for alle undersøgte områder, men der stadig er plads til yderligere forbedring. At få adgang til native features har været nemt og ligetil, med få undtagelser, såsom adgang til Maps hvor eksterne pakker skulle importeres og installeres. Hvad angår usability, har Xamarin inkluderet flere mekanismer der sørger for, at applikationen overholder de platform specifikke konventioner. Dette kan have en positiv effekt på applikationens learnability og memorability. Vi demonstrerede bl.a. dette ved usability inspektionen; evaluatorerne vidste intuitivt hvordan navigationen fungerede og kunne genkende flere funktionaliteter fra andre applikationer. En usability problematik der stadig virker problematisk er tilpasning af visse UI-elementer, såsom ListView. Dette er grundet at implementationen på hver platform vælger UI farve, og derfor skal det specificeres platform specifikt i Xamarin. Med hensyn til ydeevnen af cross-platform applikationer, ser vi stadig at native applikationer klarer sig bedre. Det tager længere tid at starte cross-platform applikationer, og cross-platform applikationer fylder (lidt) mere på telefonens harddisk. Vi mener dog at applikationerne, både nativ og cross-platform føles flydende og responsive efter opstart. Derfor vurderer vi at kompromisser i forhold til applikationens *efficiency* eller performance ikke er signifikante. Vi er dog endnu usikre på, hvor stor effekt ydeevnen har på applikationer med større omfang.

Med alt taget i betragtning, ser vi cross-platform tilgangen som en god mulighed, der kan spare private og virksomheder for meget tid og mange ressourcer. Der bliver stadig indgået kompromisser som i tidligere cross-platform generationer, men kompromiserne er blevet mindre signifikante, og frameworks som Xamarin kan tilbyde fordele såsom ekstra features og *forbedringer* i forhold til projektstruktur, dette kan dog være subjektivt.

8. Litteraturliste

Android Developer [2]; *App startup time* [Web-side] (2018) [https://developer.android.com/topic/performance/vitals/launch-time#av] (besøgt d. 7/12-2018)

Android Developer; *Design for Android* [Web-side] (2018) [https://developer.android.com/design/] (besøgt d. 8/10-2018)

Angulo, E., Ferre, X.; *A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX* [Artikel] (2014)

Apple Developer; *Human Interface Guidelines* [Web-side] (2018) [https://developer.apple.com/design/human-interface-guidelines/] (besøgt d. 8/10-2018)

Chauhan, S.; *Xamarin Apps vs. Native Apps vs. Hybrid Apps* [Web-artikel] (2018) [https://www.dotnettricks.com/learn/xamarin/xamarin-apps-vs-native-apps-vs-hybrid-apps] (besøgt d. 28/10-2018)

Heitkötter, H., Hanschke, S., Majchrzak, T. A.; *Evaluating Cross-Platform Development Approaches for Mobile Applications* [Artikel] (2013)

Martinez, M.; *Two Datasets of Questions and Answers for Studying the Development of Cross-platform Mobile Applications using Xamarin Framework* (2018)

Microsoft [1]; *Xamarin forms* (2018) [https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/tabbed-page] (besøgt den 29/11-2018)

Microsoft [2], *Xamarin Application Package Size* (2018) [https://docs.microsoft.com/da-dk/xamarin/android/deploy-test/app-package-size] (besøgt den 10/12-2018)

Mono Project; *About Mono* (2018) [https://www.mono-project.com/docs/about-mono/] (besøgt den. 28/10-2018)

Nielsen, J.; *Usability 101: Introduction to Usability* [Web-artikel] (2012) [https://www.nngroup.com/articles/usability-101-introduction-to-usability/] (besøgt d. 28/11-2018)

Peppers, J.; *Xamarin Cross-platform Application Development* (2014) [bog]

Preece, J., Rogers, Y., Sharp, H.; *Interaction Design: beyond human-computer interaction* [Bog] (2002)

Robier, J.; *UX Redefined - Winning and Keeping Customers with Enhanced Usability and User Experience* [Bog] (2016)

Sommer, A., Krusche, S.; *Evaluation of cross-platform frameworks for mobile applications* [Artikel] (2013)

StackOverflow; *Developer Survey Results* (2018) [Web-artikel] [https://insights.stackoverflow.com/survey/2018/#technology-frameworks-libraries-and-tools] (besøgt den 25-11-2018)

Visual Studio [1]; *Visual Studio Tools for Xamarin* [Web-artikel] (2018) [https://visualstudio.microsoft.com/xamarin/] (besøgt d. 25/10-2018)

Visual Studio [2*]; Pricing of Visual Studio* [Web-side] (2018) [https://visualstudio.microsoft.com/vs/pricing/] (besøgt d. 7/12-2018)

Xamarin Developer [1]; *Xamarin Documentation* (2018) [Web-side] [https://developer.xamarin.com/releases/android/xamarin.android_8/xamarin.android_8.3/] (besøgt den. 28/10-2018)

Xamarin Developer [2]; *Releases* [Web-side] (2018) [https://developer.xamarin.com/releases/] (besøgt d. 7/12-2018)

Xamarin; *Xamarin App Development with Visual Studio* [Web-side] (2018) [https://visualstudio.microsoft.com/xamarin/] (besøgt d. 8/11-2018)

Xanthopoulos, S., Xinogalos, S.; *A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications* [Artikel] (2013)

Bilag

1. Usability inspektions resultatskemaer

1.1 Evaluator 1

Nr.	Opgavebeskrivelse	Scr.	Noter
1	Find en artikel i RSS feed og tilgå den. Sæt artiklen som favorit. Derefter gå tilbage til RSS feed.	**4**	Det loader lidt langsomt inde i artiklen, men til gengæld er det lige fra kilden.
2	Angiv 5 favorit artikler fra forskellige feed udbydere (DR, Google osv.)	**4**	
3	Find og fjern favorit-markering fra favorit artikler.	**2**	Favorit feedet skal opdateres manuelt af brugeren før de nye favoritter kommer frem. Igen er knapperne ikke gode.
4	Gør dit feed personligt ved at tilføje nye RSS feeds.	**1**	Load wheel i settings er der for evigt til at starte med. Crashede første gang fordi jeg vist angav en forkert URL. Det er ikke klart hvornår ens adresse er korrekt eller ej. Hvis man tilføjer en URL der ikke viser noget ved ”TEST” crasher appen. ”TEST” burde sige om URL’en kan bruges eller ej.
5	Slå feeds til og fra efter ønske, derefter slet et af de eksisterende feeds.	**5**	Meget nemt. Bør jeg skulle bekræfte at jeg gerne vil slette feedet? Knapperne er tæt og jeg kunne jo trykke forkert.
6	Gem to lokationer på kortet. Centrer på første lokation og slet den anden gemte lokation.	**3**	Gemte lokationer hedder bare ”You are here”. Ikke god kommunikation.
7	Find din hjemby på kortet.	**4**	

1.2 Evaluator 2

Nr.	Opgavebeskrivelse	Scr.	Noter
1	Find en artikel i RSS feed og tilgå den. Sæt artiklen som favorit. Derefter gå tilbage til RSS feed.	5	
2	Angiv 5 favorit artikler fra forskellige feed udbydere (DR, Google osv.)	5	Knapperne i feeden blev mindre når jeg trykkede på dem
3	Find og fjern favorit-markering fra favorit artikler.	5	
4	Gør dit feed personligt ved at tilføje nye RSS feeds.	3	Appen crasher når man går ud af den for at finde url på en feed
5	Slå feeds til og fra efter ønske, derefter slet et af de eksisterende feeds.	5	
6	Gem to lokationer på kortet. Centrer på første lokation og slet den anden gemte lokation.	4	
7	Find din hjemby på kortet.	4	Kan ikke søge på addresse

1.3 Evaluator 3

Nr.	Opgavebeskrivelse	Scr.	Noter
1	Find en artikel i RSS feed og tilgå den. Sæt artiklen som favorit. Derefter gå tilbage til RSS feed.	4	Favoritterne opdaterede ikke automatisk, da jeg ville tjekke om den var gemt, ellers føltes det nemt og naturligt.
2	Angiv 5 favorit artikler fra forskellige feed udbydere (DR, Google osv.)	U	Jeg kunne ikke kende forskel på udbydere, da deres billeder var de samme, og der var ikke navn på.

3	Find og fjern favorit-markering fra favorit artikler.	**5**	Ingen bemærkninger, favorit stjernen ser ikke godt ud dog.
4	Gør dit feed personligt ved at tilføje nye RSS feeds.	**4**	Det var nemt at finde siden hvor der kan tilføjes RSS, men besværligt at selv skulle finde RSS links på nettet.
5	Slå feeds til og fra efter ønske, derefter slet et af de eksisterende feeds.	**5**	Ingen bemærkninger
6	Gem to lokationer på kortet. Centrer på første lokation og slet den anden gemte lokation.	**2**	Besværligt, knapperne ser ikke naturlige ud, og vidste ikke at jeg kunne trykke på knapperne under mappet.
7	Find din hjemby på kortet.	**3**	Fungerer som forventet, men ingen søgefunktion gør det sværerer.

1.4 Evaluator 4

Nr.	Opgavebeskrivelse	Scr.	Noter
1	Find en artikel i RSS feed og tilgå den. Sæt artiklen som favorit. Derefter gå tilbage til RSS feed.	5	Tilbgaeknap virker, intuitivt og overskueligt. Godt med stor stjerne.
2	Angiv 5 favorit artikler fra forskellige feed udbydere (DR, Google osv.)	5	Fandt udbyder via iconer. Rigtig god UX. Stjernerne er nemmere.
3	Find og fjern favorit-markering fra favorit artik3ler.	(3)-4	Favoritmarkering var ikke i oversigt bag efter . RSS historik ændret?refresh virker

4	Gør dit feed personligt ved at tilføje nye RSS fee3ds.	1	Gik ned på fejlagtig url. Manglede personlig rss kendskab. Ide: oversigt. Test virkede Add crashede
5	Slå feeds til og fra efter ønske, derefter slet et af de eksisterende feeds.	4-5	Følger standard i setting for tænd/sluk Tilfreds Ingen feedback på test Feedback p rigtig url Add mgl opdatering Min nyy feed kom ikke p men de hører til sp 4.
6	Gem to lokationer på kortet. Centrer på første lokation og slet den anden gemte lokation.	3-4	Knapper er helt i skoven /søen Fint med mange wp. Fingeren zoomer, men jeg kan ikke sætte waypoint i åen!!! Save gir.ok.mgl feedback. Der er noget med at slette location og række følge. Mgl feedback
7	Find din hjemby på kortet.	3-4	Efter navigation virkede save location ikke… er listen for kort (maks antal wp) Ingen søgning (let skuffet der) men navigation med zoom virker. Følger standard.

2. Kildekode til mobilapplikation

I det nedenstående link findes kildekoden til vores Xamarin applikation.
Link til github projekt:
https://github.com/malaue4/3.-Semester-Kandidat