

# **Cross-platform development**

unter Xamarin.Forms

## **Bachelorarbeit 2**

Angefertigt an der der  
Fachhochschule Campus Wien  
Bachelorstudiengang Informationstechnologien und Telekommunikation

Vorgelegt von:  
Maximilian Hans Peter Thomas Peßl  
Personenkennzeichen: c1510475049

Vertiefungsrichtung:  
Telekommunikation

Abgabetermin: 01.06.2018

Erklärung:

Ich erkläre, dass die vorliegende Bachelorarbeit von mir selbst verfasst wurde und ich keine anderen als die angeführten Behelfe verwendet bzw. mich auch sonst keiner unerlaubter Hilfe bedient habe.

Ich versichere, dass ich dieses Bachelorarbeitsthema bisher weder im In- noch im Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Weiters versichere ich, dass die von mir eingereichten Exemplare (ausgedruckt und elektronisch) identisch sind.

Datum:

Unterschrift:



# Danksagung

(Falls gewünscht.)

# Kurzfassung

Apple und Android sind heutzutage die größten Vertreter von Mobilien Betriebssysteme und erfreuen sich stetig steigender Beliebtheit. Als Software Entwickler muss nach dem Design einer Applikation und der funktionalen Anforderungen schlussendlich eine Entscheidung über die Zielplattform getroffen werden. Diese Wahl gestaltet sich an sich sehr leicht weil, als größte Vertreter entweder Apples iOS oder Googles Android zur Verfügung stehen. Dies spiegelt den klassischen Software Engineering (SE) Prozess mit den Phasen Analyse, Design, Implementierung, Testen und Wartung wieder. Dabei ist es gegenstandslos ob es sich um die Entwicklung einer Nativen Applikation oder Cross-platform (CP) Anwendung handelt. Vor allem in der Implementierung und Test Phase ist die Entwicklung symmetrisch.

Jedoch gibt es bedeutende Unterschiede in den Phasen Analyse und Design. Bei der Verwendung von Cross-platform Frameworks (CPF) spielt die Analyse der Anforderungen und die darauf anschließende Design Phase eine beträchtliche Rolle. Xamarin.Forms ermöglicht es für die Entwicklung einer Mobilien Applikation die Anwendung aus einer sehr objektiven Betrachtungsweise zu spezifizieren. Dabei ermöglicht es dem Entwickler sich auf die Funktionalitäten der Anwendung und nicht auf die Ziel-Betriebssystem spezifischen Design Elemente zu konzentrieren. Die Arbeit befasst sich ausführlicher mit dem CPF Xamarin.Forms und wie sich die Entwicklung einer Mobilien Applikation durch Anwendung eines solchen Frameworks verändert.

# Abstract

Apple and Android are the largest representatives of mobile operating systems today and are becoming increasingly popular. As a software developer, a decision about the target platform must finally be made after the design of an application and the functional requirements. This choice is very easy in itself because the largest representatives are either Apple's iOS or Google's Android. This reflects the classic Software Engineering (SE) process with the phases analysis, design, implementation, testing and maintenance. It is irrelevant whether it concerns the development of a native application or a cross-platform (CP) application. Especially in the implementation and test phase, the development is symmetrical.

However, there are significant differences in the analysis and design phases. When using cross-platform frameworks (CPF), the analysis of the requirements and the subsequent design phase plays a considerable role. Xamarin.forms makes it possible for the development of a mobile application to specify the application from a very objective point of view. It allows the developer to focus on the functionalities of the application and not on the target operating system specific design elements. The work deals in more detail with the CPF Xamarin.forms and how the development of a mobile application changes through the application of such a framework.

# Abkürzungsverzeichnis

CP	Cross-platform
CPF	Cross-platform Framework
IDE	Integrated Developement Environment
MAD	Mobile App Development
MVC	Model View Controller
MVVM	Model View View Model
PCL	Portable Class Library
SA	Shared Asset
SE	Software Engineering
SL	Shared Library
UI	User Interface

# Schlüsselbegriffe

Cross-platform  
Cross-platform Framework  
Integrated Development Environment  
Mobile App Development  
Model View View Model  
Software Engineering  
Xamarin  
Xamarin.Forms  
Xamarin.Native



# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Related Work . . . . .	5
<b>2</b>	<b>Xamarin als Cross-Platform Framework</b>	<b>7</b>
2.1	Xamarin.Forms Überblick . . . . .	8
2.2	Entwicklung unter Xamarin.Forms . . . . .	12
<b>3</b>	<b>Design Entwicklung der MCKB Applikation</b>	<b>19</b>
3.1	Applikations Spezifizierung . . . . .	22
3.2	Unterschiede zur Xamarin.Native Version . . . . .	24
<b>4</b>	<b>Ergebnisse</b>	<b>25</b>
<b>5</b>	<b>Fazit</b>	<b>26</b>
<b>A</b>	<b>Anhang/Ergänzende Information</b>	<b>27</b>
	Abbildungsverzeichnis . . . . .	28
	Codeverzeichnis . . . . .	29
	Tabellenverzeichnis . . . . .	29
	Literaturverzeichnis . . . . .	31

# Kapitel 1

## Einführung

Mobile App Development (MAD) und Software Engineering (SE) sollen den Entwicklungsprozess einer Software oder Mobilen Applikation insofern erleichtern, dass durch klar definierte Phasen der Entwicklungsprozess dokumentiert und verstanden wird [Was10]. Die Phasen bei SE beginnen mit der Analyse der Anforderungen an die zu entwickelnde Applikation. Durch gezielte Fragen und Diskussionen mit dem Kunden wird versucht ein erstes Bild von dem zu schaffen was nach Abschluss der Entwicklung der breiten Masse an Anwendern zur Verfügung gestellt werden soll. Anschließend wird es in der Design Phase spezifischer. In dieser Phase geht es nicht primär um das Aussehen und die Darstellung der unterschiedlichen User Interface (UI) Elemente sondern auch um die Spezifizierung der funktionalen und nicht funktionalen Anforderungen an die Anwendung [Was10]. Anschließend folgt die Phase der Implementierung gefolgt von einer Test Phase. Die Software geht anschließend an den Kunden bzw. an die Smartphone Anwender wenn es sich um eine Applikation handelt und wird in der Wartungsphase, im normal Fall einer kontinuierlichen Weiterentwicklung unterzogen. In der Wartungsphase geht es um die Behebung von *Bugs* (Anwendungsfehler die einen Absturz der Applikation oder das Verwenden der Applikation erheblich erschweren), welche während der Testphase nicht gefunden wurden.

Diese Arbeit beschäftigt sich hauptsächlich mit den beiden ersten Phasen von SE und der Implementierung unter Xamarin.Forms als Cross-platform Framework. Xamarin und im besonderen Xamarin.Forms werden in Abschnitt 2 genauer erläutert. Abschnitt 2 bildet die Grundlage zu verstehen wie die Entwicklung mit einem CPF funktioniert und welche Überlegungen notwendig sind eine Applikation mit eben jenem Framework zu entwickeln.

Jene Überlegungen die vor der Implementierung oder der ersten Programmierung von Funktionen der Applikation getroffen werden müssen, sind die Basis folgender Fragestellung mit welcher sich die Arbeit im zentralen beschäftigen wird:

*Welche Einschränkungen ergeben sich durch Xamarin.Forms als Cross-platform Framework, im Vergleich zu Nativer Applikationsentwicklung unter Android und Cross-platform Entwicklung unter Xamarin.Native?*

## 1.1 Motivation

Smartphones haben heutzutage einen wichtigen Bestandteil des täglichen Lebens eingenommen. Der kurze Blick auf das Smartphone, auf dem Weg in die Schule oder in die Arbeit, sind Teil einer täglichen Routine geworden. Die Informationsbeschaffung ist durch mobiles Internet wie LTE (Long Term Evolution oder auch 4G) und immer schneller werdender Prozessoren in Smartphones einfacher denn je geworden. Doch nicht nur die Hardware der Anwender befindet sich in einem stetigen Prozess der Weiterentwicklung sondern auch die darauf laufende Software wie das Betriebssystem und diverse Applikationen. Apples iOS befindet sich derzeit in der Version 11.2.6 und wird bald Version 11.3.3 veröffentlichen. Im Vergleich dazu ist Android in Version 8.1.0 auf den aktuellsten Geräten bereits vorinstalliert. Apple begann mit seinem mobilen Betriebssystem iOS im Jahre 2005 und veröffentlichte 2007 iOS in der Version 1.0.x. Google begann 2008 seinen Start mit Android in der Version 1.0, auch bekannt als API 1. Seit der Einführung beider Betriebssysteme hat sich in den Jahren bis 2017 einiges getan. Gab es zu Beginn des Ersten Quartals 2009 noch folgende Vertreter:

- Google - Android
- Apple - iOS
- Microsoft - Windows
- RIM - Blackberry
- Nokia - Symbian

Sind aktuell nur iOS und Android mit weltweiten Marktanteilen von 87,7% für Android und 12,1% für iOS übrig geblieben. Dies ist in Abbildung 1.1 zu erkennen.

Durch kontinuierliche Weiterentwicklung von Android und iOS erreichten diese eine immer größer werdende Beliebtheit wodurch andere Betriebssysteme weitgehend an Bedeutung verloren haben und schlussendlich keinen nennenswerten Anteil laut heutigen Statistiken besitzen.

Jedoch sollte als Ausnahme die Firma RIM mit dem Betriebssystem BlackBerry OS erwähnt werden. Seit BlackBerry OS 10 wurde es möglich Android Anwendungen ausführen zu können. Dies erzielte zwar nicht den erhofften Erfolg, war jedoch eine Besonderheit. Die Möglichkeit Anwendungen eines anderen Betriebssystems, wie zum Beispiel Android, installieren und auf die gleiche Art und Weise verwenden zu können ermöglichte es Anwendern, die Blackberry aufgrund seines außergewöhnlich Sicherheitsorientierten Betriebssystems verwendeten, das Beste aus beiden Welten zu vereinen.



Abbildung 1.1: Weltweiter Marktanteil an Mobilien Betriebssystemen von 2007 bis 2017 Stand Q1 2018 (Quelle: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>)

Android ist aus heutiger Sicht das beliebteste Betriebssystem im weltweiten Vergleich. Doch warum finden sich in beiden App Stores, Google Play für Android und App Store für Apple, weit mehr als 2.000.000 Applikationen, wobei Android mit 3.361.843<sup>1</sup> Anwendungen um mehr als 30% mehr Applikationen als iOS zur Verfügung stellt. Diese Differenz scheint auf den Ersten Blick enorm, jedoch gilt es dabei folgenden Unterschied zu beachten. Applikationen können unter Android einfacher entwickelt und veröffentlicht werden. Apple zieht hier eine Grenze und gibt bestimmte Vorgaben vor und überprüft Apps genauer als Google. Weiters benötigt man einen Apple Developer Zugang der mit jährlichen Kosten verbunden ist.

<sup>1</sup><https://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/>

Die Motivation eine Cross-platform Applikation zu entwickeln ruht meist auf der Überlegung, warum man eine Applikation zwei Mal entwickeln muss, wenn sich eigentlich nur das Aussehen der Applikation verändert, nicht aber die grundlegenden Funktionalitäten [Pes18]. Als Beispiel dient die Facebook App die sowohl für Android als auch iOS mit React Native programmiert wurde. Demnach steht eine Vielzahl an CPF's der unterschiedlichsten Hersteller zur Verfügung.

Folgende sind die populärsten <sup>2</sup>:

- Corona SDK - Spiel und Mobile App Development
- Xamarin
- Appcelerator Titanium
- React Native
- PhoneGap
- Ionic
- Sencha Touch

Alle CPF's bieten auf unterschiedlichste Art und Weise die Möglichkeit Applikationen auf Basis des "Shared Codes", gemeinsame Funktionalitäten zusammenzufassen und nur einmal implementieren zu müssen. Der in der Integrierten Development Environment (IDE) eingebaute Compiler übernimmt die Aufgabe den Code entsprechend so zu kompilieren, damit dieser auf der Zielplattform nativ ausgeführt werden kann.

---

<sup>2</sup><https://medium.com/@MasterOfCodeGlobal/best-10-android-frameworks-for-building-android-apps-d2d0ee48e464>

## 1.2 Related Work

Im Zuge der Literatur Recherche befassen sich einerseits, eine Vielzahl an Artikeln, Bachelor oder Master Arbeiten mit den Stärken und Schwächen von Cross-platform Frameworks, andererseits jedoch mit sehr spezifischen Themen der Applikationsentwicklung wie zum Beispiel Responsive UI Elements oder die Unterschiede wie ein Layout mit dem notwendigen Code dahinter interagiert und verknüpft wird. Dabei wird wiederum im speziellen auf die Kommunikation zwischen Layout und Code geachtet und analysiert.

Die Bücher [Jen15], [Ver17] und [Joh15] dienen als Grundlage um den Prozess der Implementierung einer Applikation mit der IDE zu verstehen und zu üben. Weiters gab es zwei Online Kurse um das in den Büchern erlernte Wissen praktisch umzusetzen.

Oleksandr Gridin widmet sich in seiner Bachelor Arbeit [Gri15], dem Xamarin Framework um dessen Stärken und Schwächen, anhand einer Android und Windows (Universal Windows Platform (UWP)) Applikation zu zeigen. Dabei wird sehr auf den Aufbau des Xamarin Frameworks, unter anderem auch Xamarin.Forms eingegangen. Im Zuge seiner Entwicklung ist es immer wieder zu Schwierigkeiten gekommen weil es keine oder nur wenige Komponenten gab um diverse Funktionalitäten umzusetzen.

In seiner Arbeit [Arm15] *Mobile Cross-platform development versus native development* widmet sich Armgren Marc, der Performance von Xamarin als Cross-platform Framework in Bezug auf UI und Netzwerk Verkehr im Vergleich zu Nativer Applikations Entwicklung. Dabei wurden die Unterschiede zwischen Nativen iOS und Android und deren Cross-platform Pendant herangezogen. Die Evaluierung ergab das erst dann entschieden werden kann ob ein CPF verwendet werden soll wenn die Anforderungen klar definiert sind. Applikationen die eine Vielzahl an schwierigen Berechnungen durchführen müssen, wie zum Beispiel Spiele, profitierten von Nativen Code, da dieser performanter auf dem Zielbetriebssystem interpretiert und ausgeführt werden kann. In Bezug auf die Netzwerk Performance wurden keine Unterschiede festgestellt.

Die Autoren V. Bhutto und K. Soman und R. K. Sungkur behandeln in dem Artikel [BSS17] die Stärken und Schwächen von Xamarin und stellen diese anderen bekannten Cross-platform Frameworks gegenüber. Dabei bedienen sie sich CPF's wie PhoneGap, verwendet HTML/CSS und Javascript um CP Apps zu entwickeln, und Titanium welches nur Javascript verwendet. Weiters beschäftigt sich dieser Artikel mit der Installation von CPF's und Komponenten des NuGET Paket Managers. Ihre Erkenntnisse zeigen das es überaus wichtig ist zu wissen welche Technologien Cross-platform Frameworks verwenden und wie man diese bestmöglich für eine CP App verwenden kann.

Die Arbeit von Ilke Soylemez [MP16] erklärt wie die Entwicklung einer Cross-platform Applikation anhand eines Beispiels abläuft. Dabei wird vor allem auf essentielle Bausteine des Xamarin Frameworks eingegangen. Diese Bausteine sind unter anderem die Verwendung einer PCL (Portable Class Library) die, für die gemeinsame Geschäftslogik statt einer Shared Library, den Code der Cross-platform App implementiert. Weiters wird auf das MVVM (Model View View Model) Design Pattern von Xamarin eingegangen und wie dieses das Layout den notwendigen Code behind implementiert. Allerdings wird sich weitgehend mit Xamarin.Native befasst wobei das

MVVM Design Pattern sowohl bei Xamarin.Native als auch bei Xamarin.Forms zum Einsatz kommt.

In meiner vorigen Arbeit zum Thema Cross-platform Development [Pes18] wurde die Entwicklung einer Cross-platform Applikation auf Basis einer Nativen Applikation betrachtet. Dabei ging es vorrangig um Xamarin als CPF und welche Unterschiede oder Schwierigkeiten bei der Entwicklung einer Cross-platform App im Vergleich zu Nativer Applikationsentwicklung auftreten. Dabei wurde eine in JAVA unter Android Studio entwickelte Applikation erneut spezifiziert und unter Xamarin.Native für Android und iOS implementiert. Im Zuge der erneuten Anforderungsanalyse und Design Phase konnten während der Implementierung einige Zeilen an Code eingespart werden, da Xamarin Komponenten beispielsweise für eine Kommunikation mit einer SQL Datenbank zur Verfügung stellt in die PCL Klasse ausgelagert werden wodurch diese Kommunikation mit einem Server nur einmal implementiert werden musste. Beide Applikationen konnten durch aufrufen der Funktionen den selben Code ausführen. Allerdings verlangte Xamarin.Native das Design der App zweimal zu Entwickeln, was einen enormen Programmieraufwand verlangte.

# Kapitel 2

## Xamarin als Cross-Platform Framework

Xamarin ist ursprünglich als Firma Xamarin 2011 von Mono Entwicklern gegründet und 2016 von Microsoft übernommen worden. Microsoft hat Xamarin weiterentwickelt und stellt es heute als Open-Source Cross-platform Framework für Visual Studio ab Version 2015 und für Mac als Visual Studio for Mac zur Verfügung. Mit diesem Framework kann eine Cross-platform Applikation für Android, iOS und Windows 10 (früher Windows Mobile) geschrieben werden.

**Mono** ist eine Software Plattform die es ermöglicht Cross-platform Applikationen bzw. Software unter Einbindung von Microsofts .NET Framework zu entwickeln. Es basiert auf C# als Programmiersprache und ist eine plattformunabhängige Software. Programme die in .NET geschrieben wurden und auf einem Linux Kernel ausgeführt werden sollen greifen auf Mono zurück um dies zu ermöglichen. Mono bildet somit eine Schnittstelle Microsofts .NET Framework zu verwenden.

**Das .NET Framework** wiederum ist eine, ursprünglich von Microsoft entwickelte Softwareentwicklungsplattform, die in direkter Konkurrenz zu JAVA SE und JAVA EE steht. Es stellt eine Laufzeitumgebung (Common Language Runtime) für auszuführende Programme zur Verfügung und ermöglicht das Einbinden einer Vielzahl von Klassenbibliotheken und Programmierschnittstellen.

Wird eine .NET Anwendung ausgeführt so ist zum Kompilierungszeitpunkt eine Übersetzung in eine Zwischensprache (Common Intermediate Language) notwendig. Anschließend wird das Kompilat von der .NET Laufzeitumgebung in die Maschinsprache des Zielsystems übersetzt und ausgeführt. Diese Übersetzung aus der Common Intermediate Language geschieht durch den sogenannten Just-In-Time Compiler (JIT) - welcher bei Xamarin.Android verwendet wird um die Anwendung für das Android Betriebssystem zu erstellen. Eine iOS Anwendung unterstützt diesen JIT nicht. Die iOS Anwendung wird in Intermediate Language (IL) kompiliert und anschließend mittels eines Apple Compilers (Ahead-of-Time Compilation) in Nativen Code übersetzt. Aus diesem Grund wird für die iOS Entwicklung unter Windows ein Mac benötigt.

Bei Xamarin.Forms sieht dieser Prozess etwas anders aus da Xamarin.Forms eine Ebene über Xamarin.Native ist. Es ist die Aufgabe des Xamarin.Forms.Core Assemblers welcher Klassen und API Schnittstellen definiert um mit den Xamarin.Native



Bibliotheken zu interagieren zu können.

**Xamarin** als Cross-platform Framework kann auf zwei Arten für die CP App Entwicklung verwendet werden:

- Xamarin Platform - auch bekannt als Xamarin.iOS und Xamarin.Android
- Xamarin.Forms

Je nach Art der zu entwickelnden Applikation eignet sich entweder Xamarin.Native oder Xamarin.Forms. Eine Entscheidung für welche Version sich entschieden wird sollte bestenfalls während der Analyse, jedoch spätestens in der Design Phase geklärt werden.

## 2.1 Xamarin.Forms Überblick

Ziel von Xamarin.Forms ist es, das Maximum an gemeinsamem Code und einheitlichen UI Elementen zur Verfügung zu stellen. Wird eine App während der Implementierungsphase für iOS oder Android erstellt um erste Tests durchzuführen wird der Applikations Code nativ auf dem Zielbetriebssystem ausgeführt [Ver17] und so gerendert das es das typische Design darstellt.

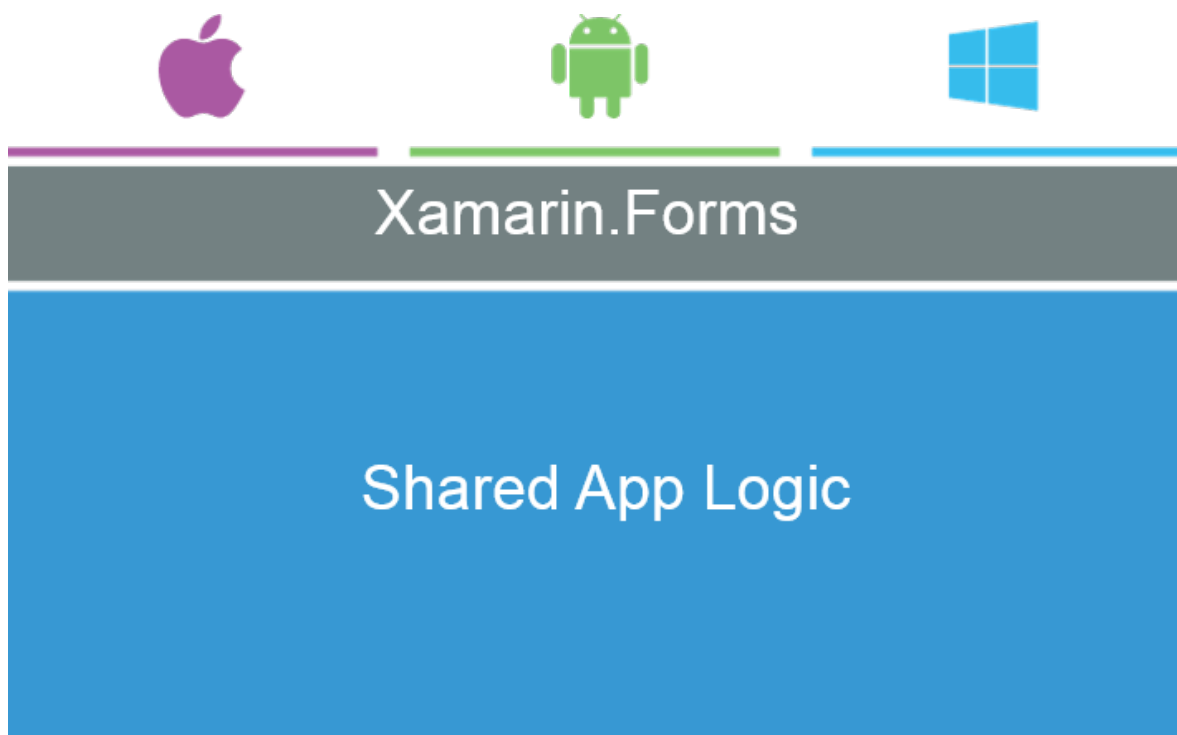


Abbildung 2.1: Xamarin.Forms Architektur (Quelle: <https://blog.goyello.com/>)

Abbildung 2.1 zeigt wie die Xamarin.Forms Architektur aussieht. Dabei werden die Nativen Bibliotheken von Android *Android SDK*, iOS *iOS UIKit* und Windows Phone bzw. Windows Mobile *UWP SDK* eingebunden. Das Einbinden dieser Bibliotheken stellt dem Framework eine Vielzahl an UI Elementen zur Darstellung von Inhalten zur

Verfügung. In der derzeitigen Version von Xamarin.Forms sind es 17 unterschiedliche Elemente.

Jedoch ist zu beachten, dass der in Abbildung 2.1 in grau dargestellte Balken den Shared UI Code repräsentiert. Auf diesem aufbauend muss mit ungefähr 20% Zielplattform spezifischen UI Design gerechnet werden [Her15]. Näheres dazu wird in den folgenden Abschnitten 2.2.1 sowie 3 genauer erläutert.

Wie Xamarin.Forms das rendern eines UI Elements durchführt ist in Abbildung 2.2 zu erkennen.

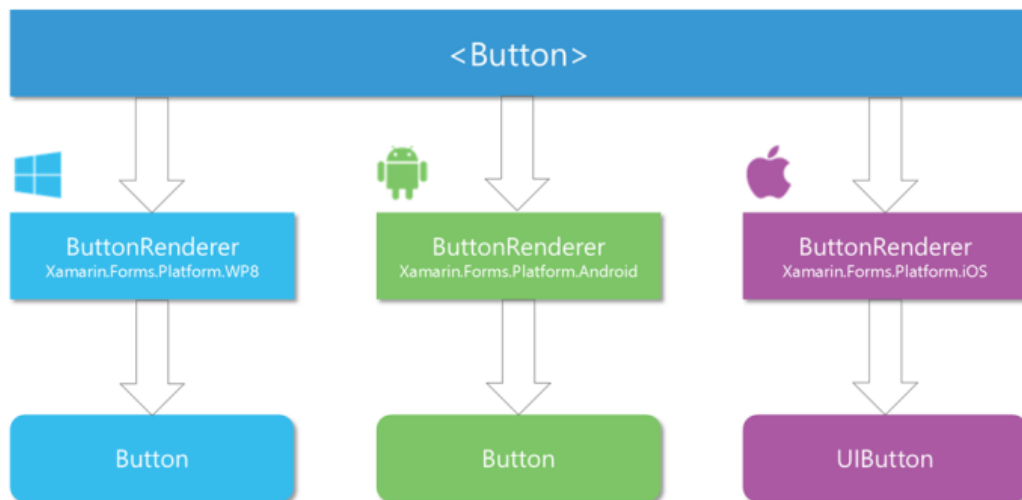


Abbildung 2.2: Wie Xamarin UI Elemente rendert  
(Quelle: [igorelikblog.wordpress.com/2016/07/08/xamarin-form-memory-management](http://igorelikblog.wordpress.com/2016/07/08/xamarin-form-memory-management))

Das Element **<Button>** wird im Layout File welches in XAML (Extensible Application Markup Language) geschrieben wurde, eingebaut. Wird der Code nun kompiliert ist der *ButtonRenderer* jener Plattformspezifische renderer der, dass Xamarin.Forms UI Element in Nativen Code umsetzt. Der **<Button>** unter Xamarin.Forms wird dadurch in einen **UIButton** für iOS oder einen **Android.Button** für Android übersetzt. Dieses rendern macht Xamarin.Form einzigartig, da im Vergleich mit anderen Cross-platform Technologien das Design für die Zielplattformen meist mittels HTML oder CSS vorgenommen werden muss, um die UI Elemente in Native Design Elemente zu transformieren [Her15].

**Rendering** ist der Arbeitsschritt einer Bilderstellung aus Objekten die, wie im Beispiel des Xamarin.Forms Frameworks, aus grafischen Objekten bestehen, welche in XAML definiert wurden<sup>1</sup>.

**XAML** ist die Beschreibungssprache die Xamarin verwendet um die grafische Gestaltung des UI zu ermöglichen. Da XAML nur eine Beschreibungssprache ist kann es keinen Code enthalten. Alle Event Handler, wie zum Beispiel das Klicken eines Button, müssen in einem Code File definiert werden. test

<sup>1</sup><https://www.itwissen.info/Rendering-rendering.html>

Eine Standard XAML Datei ist in Abbildung 2.3 dargestellt. Die Deklarationen in Zeile zwei bis drei definieren das der Namespace von Microsoft verwendet werden soll. In Zeile fünf wird ein Präfix für einen lokalen Namespace definiert um Zuweisungen im XAML File mit dem Code Behind File zu ermöglichen. Wird eine XAML Datei der PCL Klasse hinzugefügt, erzeugt die IDE neben der Layout Datei zusätzlich eine Code behind Datei. Diese ist in Abbildung 2.3 als zweiter Tab zu erkennen. Über die Klasse *MCKBPage.xaml.cs* wird das Verhalten der App implementiert. Die Referenzierung erfolgt über die Namespace Deklarationen im XAML File.



Abbildung 2.3: Aufbau einer XAML Datei für Xamarin.Forms

Für die Entwicklung der CP App wird Visual Studio for Mac verwendet, weil somit eine iOS Anwendung einfacher getestet werden kann. Man kann eine iOS App auch mit Windows erstellen, allerdings benötigt man dazu einen Mac im gleichen WLAN Netzwerk damit der Compiler die App erzeugen und ausführen kann.

**PCL** steht für Portable Class Library und ist jene Klasse in welcher sich der gemeinsame Code für die Cross-platform App befindet.

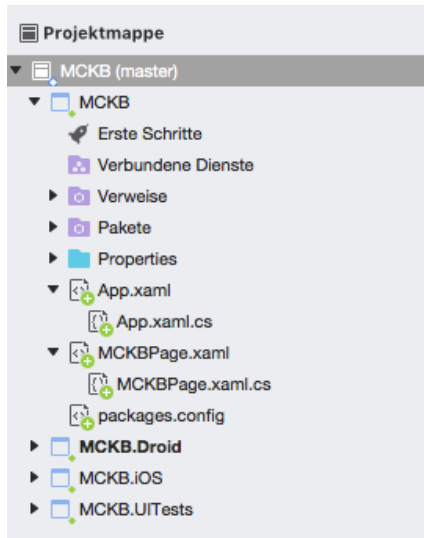


Abbildung 2.4:  
Projekt Struktur einer CP App

In Abbildung 2.4 sind alle relevanten Klassen aufgelistet die für eine Cross-platform Applikation benötigt werden. Die **PCL** Klasse ist im Ordner *MCKB* implementiert. Die weiteren Ordner beginnend mit *MCKB*, gefolgt von *Droid* oder *iOS* können dafür verwendet werden gezielt Plattformspezifischen Code zu implementieren welcher nicht von beiden Zielbetriebssystemen verwendet wird. Weiters ist das XAML File aus Abbildung 2.3 mit dessen Code Behind File zu erkennen. Die Ordner, *Verweise* werden benötigt damit der Compiler die korrekte Zuordnung zum Zielbetriebssystem durchführen kann. Der Ordner *Pakete* ermöglicht es Pakete des **NuGET** Paket Managers einzubinden. Ein solches Paket kann beispielsweise eine MySQL Anbindung an einen Server sein.

Die in Abbildung 2.1 wird im Laufe der Entwicklung um einige XAML und Code Behind Files wachsen. Der Prozess der Programmierung ist in Abschnitt 3 festgehalten.

## 2.2 Entwicklung unter Xamarin.Forms

Wie schon in Abschnitt 2 eingangs kurz angeschnitten wurden die Unterschiede der Kompilierung von Xamarin aufgezeigt. Als Entwickler kann man in C# auf die schon bekannten .NET Klassen, aufgrund von Mono zugreifen und die Applikationslogik implementieren. Ist die App für den ersten Testlauf bereit, wählt man in der Projekt Struktur die Zielplattform aus und markiert diese als Start Projekt. In Abbildung 2.1 ist dies zum Beispiel *MCKB.Droid*, da dieser Ordner durch eine Dicke Schrift hervorgehoben wurde.

Der Prozess der Kompilierung ist in Abbildung 2.5 zu erkennen.

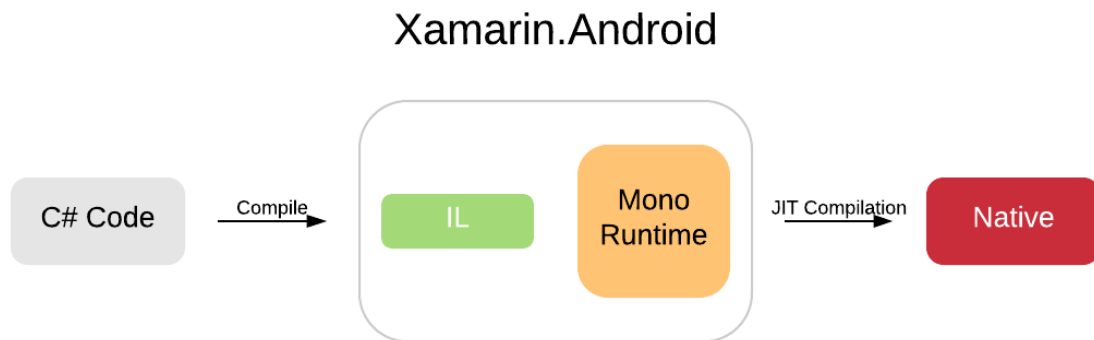


Abbildung 2.5: Wie Xamarin.Android kompiliert wird

Der C# Code wird in die Intermediate Language (IL) umgesetzt und in die Mono Runtime geladen. Anschließend übernimmt der Just-In-Time (JIT) Compiler die Übersetzung in Nativen Code der anschließend auf dem Zielbetriebssystem ausgeführt wird. Dieser Prozess läuft im Xamarin Frameworks automatisch ab sobald die Applikation erstellt und auf das Virtuelle oder Hardware Gerät geladen werden soll.

Wählt man in Abbildung 2.1 iOS als Startprojekt, *MCKB.iOS* ist dann Dick hervorgehoben, so sieht der Erstellungsprozess differenzierter aus, da Apple eine JIT Kompilierung nicht unterstützt. Betrachtet man Abbildung 2.6 genauer erklärt es warum man einen Apple Computer für die Erstellung einer iOS oder MacOS Anwendung benötigt.



Abbildung 2.6: Wie Xamarin.iOS kompiliert wird

Für die Erstellung der iOS App wird gleich wie bei einer Android App der C# Code

in die IL kompiliert. Da nun keine Mono Runtime zur Verfügung steht wird ein Apple Compiler benötigt. Dieser Compiler ist der Ahead-of-Time (AOT) Compiler der nun die Übersetzung in Nativen Code übernimmt.

Xamarin.Forms stellt eine höhere Abstraktions Ebene für die Entwicklung einer Applikation dar. Der Kompilierungsprozess lässt sich folgendermaßen darstellen.

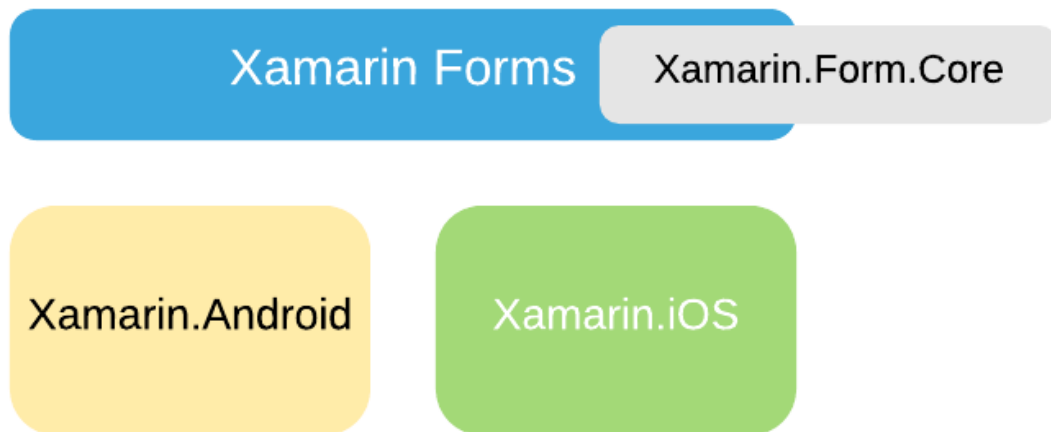


Abbildung 2.7: Xamarin.Forms Architektur

Abbildung 2.7 beschreibt was durch Xamarin.Forms erzielt werden soll. Es baut auf den Bibliotheken für die Übersetzung in Nativen Android und iOS Code auf und ist das fehlende Bindeglied für den Prozess in Abbildung 2.2 (Rendering eines Button von Xamarin.Forms). Jener Part der für die Übersetzung des Codes zuständig ist, ist die *Xamarin.Forms.Core* Bibliothek. Die in dem XAML File festgelegten UI Elemente der PCL Klasse sind Teil dieser *Xamarin.Forms.Core* Bibliothek. Der Compiler übersetzt nun für die UI Elemente für Android, siehe Abbildung 2.5 und iOS, siehe Abbildung 2.6, damit das Zielplattform spezifische Design dargestellt wird.

### 2.2.1 Model View ViewModel - Design/Architectural Pattern

**MVVM** ist eine Variante des *Model View Controller Pattern* und dient dazu eine verbesserte Testbarkeit (Unit-Testing) durch Trennung der Benutzerschnittstelle und der sich dahinter befindlichen Logik der Benutzerschnittstelle zu ermöglichen. In einigen Büchern zur Entwicklung mit Xamarin wird auf das MVVM Pattern verwiesen da es eine Architektur Struktur für User Interfaces implementiert[Ver17]. Dies ist vor allem für komplexe Applikationen von Nutzen da durch eine verbesserte Testbarkeit von Modulen die Benutzerfreundlichkeit erheblich gesteigert werden kann.

**Model:** Repräsentiert ein Business Objekt das die Daten und das Verhalten der Applikation kapselt. Für die MCKB Applikation wären dies:

- Artikel
- Anwender
- Kommentare zu Artikeln

**View:** Ist das was der Anwender sieht wenn er die Applikation verwendet. Artikel werden in *Pages*<sup>2</sup> dargestellt. Kommentare, welche Teil einer Artikel Page sind, die es zu Artikeln gibt werden angezeigt und können hinzugefügt werden. Wie der View definiert wird hängt von der Design Spezifizierung der Page und dessen Layout ab und kann Beispielsweise eines folgender Layouts sein:

- Content Page
- Navigation Page
- Tabbed Page
- Carousel Page
- ...

Innerhalb einer Page kann ein Layout definiert werden um Daten nach einem Schema darzustellen. Folgende *Layouts*<sup>3</sup> kann eine Page enthalten:

- Content View
- Scroll View
- Stack, Absolute, Relative oder Grid Layout

**ViewModel:** Dies ist ein Model welches für einen View entworfen wurde. Es ist eine Klasse mit Eigenschaften welche den Zustand eines View's sowie Methoden und die Logik eines View's implementiert.

---

<sup>2</sup><https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/pages>

<sup>3</sup><https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/layouts>

Ein ViewModel für den View in Codeabschnitt 2.1, welches eine *Page* mit einem *StackLayout* (*horizontale Anordnung von UI Elementen*), einem **Button** und einen **ListView** mit **Items** enthält, hätte beispielsweise ein ViewModel, welches in Codeabschnitt 2.2 abgebildet ist, um Elemente dem ListView hinzuzufügen:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4     xmlns:local="clr-namespace:MCKB" x:Class="MCKB.MCKBPage">
5     <StackLayout>
6         <Button Text="Add"/>
7         <ListView x:Name="ArticleView">
8             <ListView.ItemTemplate>
9                 <DataTemplate>
10                     <TextCell Text="{Binding ArticleName}" />
11                 </DataTemplate>
12             </ListView.ItemTemplate>
13         </ListView>
14     </StackLayout>
15 </ContentPage>

```

Listing 2.1: Beispiel View

```

1 class ViewModel{
2     ObservableCollection Item;
3     void Add(){
4         //Implement function here
5     }
6 }

```

Listing 2.2: Beispiel ViewModel für Codeabschnitt 2.1 (View)

Auf den Ersten Blick sieht das ViewModel in Codeabschnitt 2.2 aus wie eine code-behind Implementierung (der Button wird mit einem Event-Handler verlinkt um die Add Funktion aufzurufen) für das XAML des View in Codeabschnitt 2.1 aus. Jedoch ist ein ViewModel keine code-behind Implementierung.

**Code-behind** ist fest an Xamarin gekoppelt. Dies führt dazu, dass es schwer ist für eine solche code-behind Klasse einen Unit-Test zu implementieren. Um den Code in der code-behind Klasse dennoch testen zu können wird dieser Code in die ViewModel Klasse geschrieben und abgeändert um eine minimale Kopplung mit Xamarin zu haben. Der Code ist nun ein **Plain Old CLR Object (POCO)** Objekt für das Unit-Tests erzeugt werden können.

**Unit/Model Tests** sind Software Tests welche funktionale Einzelteile von Software auf deren korrekte Funktionalität testen. Die für diese Arbeit geschriebene Cross-platform Applikation wird keine Unit/Modul Test implementieren.



## 2.2.2 Projekterstellung mit Visual Studio for Mac

Für die Entwicklung der Cross-Platform Applikation wird **Visual Studio for Mac** verwendet, um sowohl eine Android als auch iOS Version der Applikation entwickeln zu können. Die Projekterstellung unter Windows mit **Visual Studio 2015** oder **2017** verläuft analog im gleichen Schema.

Zu Beginn wird eine Standard Xamarin.Forms Applikation erstellt. Hierbei muss entschieden werden, wie der Cross-Platform Code geteilt werden soll:

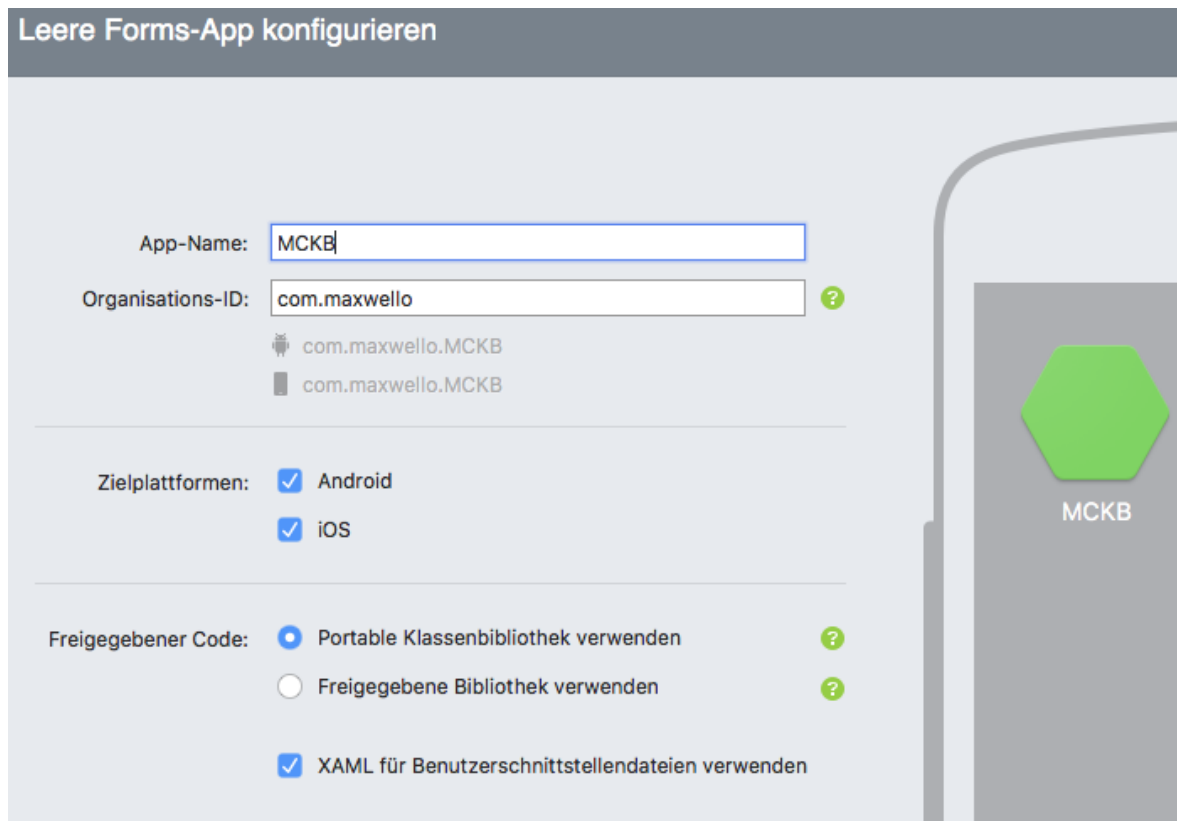


Abbildung 2.8: Erstellung der Xamarin.Forms App - MCKB

In Abbildung 2.8 ist zu erkennen, dass schon bei der Erstellung der Applikation festgelegt werden muss, auf welche Art und Weise der *Shared Code*, der CP-App implementiert werden soll.

In Abbildung 2.9 ist zu erkennen, wie sich **Portable Class Library** von **Shared Library** unterscheiden. Wird PCL für den freigegebenen Code verwendet, siehe Abbildung 2.9 links, so erstellt die IDE alle notwendigen Verweise und verlinkt diese. Anders ist dies bei einem Shared Library (SL) Projekt, bei welchem der Entwickler selbst die notwendigen Verweise zu weiteren dependencies einbinden muss.

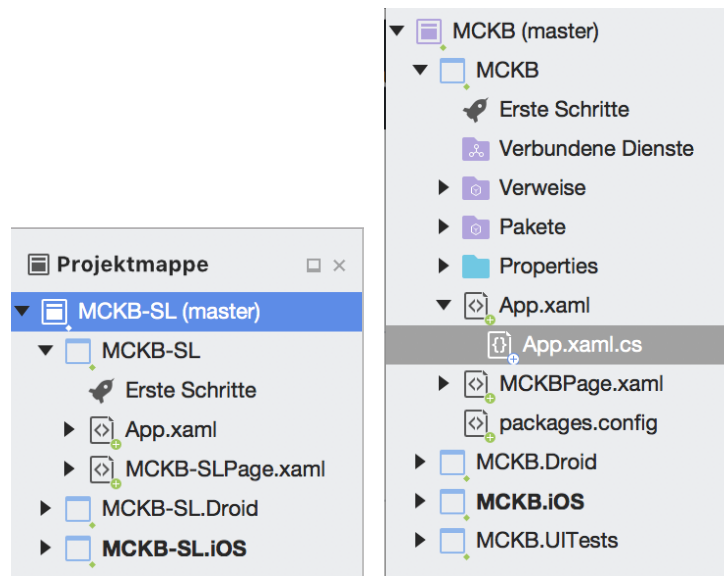


Abbildung 2.9: Unterschied Shared Code - Xamarin.Forms App - MCKB

Der Unterschied bei der Festlegung auf welche Art und Weise der freigegebene Code für die Applikation verwaltet werden soll muss von Beginn an definiert werden. Eine Änderung im Nachhinein ist nicht mehr möglich.

**Eine Portable Klassenbibliothek (PCL)** erstellt wie schon erwähnt alle notwendigen Verweise für eine Android und iOS Cross-Plattform Applikation. Dabei spielen die folgenden Verweise eine wichtige Rolle:

- *Xamarin.Forms.Core* definiert eine allgemeine API (Application Programming Interface) für Klassen wie zum Beispiel Buttons, Labels, ListViews uvm.
- *Xamarin.Forms.Platform* linkt die Plattform spezifischen Renderer von Android und iOS damit Elemente in der PCL Klasse in Plattform spezifischen Code übersetzt werden können.
- *Xamarin.Forms.Xaml*

Der Code in einer PCL wird zur Laufzeit zu einer *dynamically-linked library (DLL)* für die Zielplattform Referenzen zusammengefasst. Ist es für die Cross-platform Applikation notwendig Nativen Code auszuführen, kann dies durch bestimmte **IF Abfragen** in der PCL Klasse umgesetzt werden.

```

1  ...
2  if (Device.OS == TargetPlatform.iOS){
3      //Run iOS Code here..
4  }
5  ...
6  else if (Device.OS == Target.Platform.Android){
7      //Run Android Code here...
8  }
9  ...

```

Listing 2.3: Plattform spezifischen Code in PCL ausführen

**Die Freigegebene Bibliothek (*Shared Library (SL)*)/*Shared Asset (SA)*)**<sup>4</sup> stellt den einfachsten Ansatz für *Code Sharing* dar. Für die Zielplattform wird nur jener Code dementsprechend kompiliert wenn es die dafür notwendigen Präprozessor Anweisungen enthält. Es wird somit der gesamte Inhalt des Projektes in jeden referenzierenden Teil des Projektes kopiert, als wäre es ein Teil davon.

```
1 #if __IOS__
2     //Run iOS Code here..
3 ...
4 #elif __ANDROID__
5     //Run Android Code here...
6 ...
7 #endif
```

Listing 2.4: Plattform spezifischen Code in SA/SL ausführen

Codeabschnitt 2.4 veranschaulicht diese Präprozessor Anweisungen die in den C# Klassen den Plattformspezifischen Code zu finden sind.

Wird eine PCL als Code Sharing Basis verwendet, wird der Zugriff auf mögliche *.NET Klassen* auf eine kleinere Untermenge beschränkt, jedoch kann dieses Problem durch Implementierung einer Schnittstelle, da dadurch auf die Ursprüngliche Menge an *.NET Klassen* zurückgegriffen werden kann.

---

<sup>4</sup><https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/code-sharing>

# Kapitel 3

## Design Entwicklung der MCKB Applikation

Als Basis für diese Arbeit dienen zwei während des Studiums entwickelte mobile Applikationen.

1. **STM32KB**<sup>1</sup> - Mit Android Studio, in JAVA, entwickelte Applikation im 4. Semester an der FH Campus Wien
2. **STM32CP** - Mit Visual Studio for Mac, in C#, entwickelte Cross-Plattform Variante im 5. Semester an der FH Campus Wien

Erstere Applikation wurde im Zuge der **Mobile App Development (MAD)** Vorlesung, zur Vertiefung von Software Engineering Skills unter Android Studio in JAVA entwickelt. Diese App ermöglicht die Darstellung von Artikeln zur Programmierung eines Mikrocontroller  $\mu C$  der Firma STM, um häufige Fragen zur Konfiguration unterschiedlicher Funktionen wie zum Beispiel *UART (Universal Asynchronous Receive and Transmit)*, *ADC (Analogue Digital Converter)* oder *CAN (Controller Area Network)* einfach erklärt nachzulesen. Die App selber stellt nur die in einer MySQL Datenbank gespeicherten Artikel dar und ermöglichte es angemeldeten Benutzern bestehende Artikel zu editieren und zu speichern.

Da die STM32KB App selber keine Daten speichert sondern nur darstellt wurde eine Infrastruktur entworfen um die für die App notwendigen Daten mobil abrufen zu können. Es wurde eine unter anderem eine MySQL Datenbank und ein Webserver auf einem Raspberry PI B+ eingerichtet. Neue Artikel konnten über eine PHP Webseite<sup>2</sup> in die Datenbank geschrieben werden. Die App lud die Artikel aus der MySQL Datenbank über verschiedenste PHP Dateien in eine lokale SQL-Lite Datenbank.

Für die Entwicklung der STM32CP App (geschrieben mit Xamarin.Native) greift auch diese App auf die schon bestehende Infrastruktur an Server und Datenbank zurück um die Spezifizierten funktionalen Anforderungen zu erfüllen.

---

<sup>1</sup>Download via <http://m4xwe11o.ddns.net/MAD-Test/App/stm32kb.apk>

<sup>2</sup><http://m4xwe11o.ddns.net/MAD-Test/webwriter.php>

Entsprechend der Anforderungen an die STM32KB App wurden folgende Funktionale Anforderungen im dazugehörigen *Software Requirements Document (SRS)* festgehalten:

- **Autor login** - Benutzer können sich in der App anmelden und sehen einen EDIT Button bei Artikeln
- **Autor registrieren** - Anwender können sich registrieren um als Autoren freigeschaltet zu werden
- **Artikel lesen** - Benutzer können Artikel lesen

In Abbildung 3.1 sind jene funktionale Anforderungen mit deren Layouts aus der App dargestellt. Die Menüführung erfolgt großteils über Buttons und der Zurück Funktion von Android über den Pfeil links oben oder jener entsprechenden Taste auf dem Android Gerät.

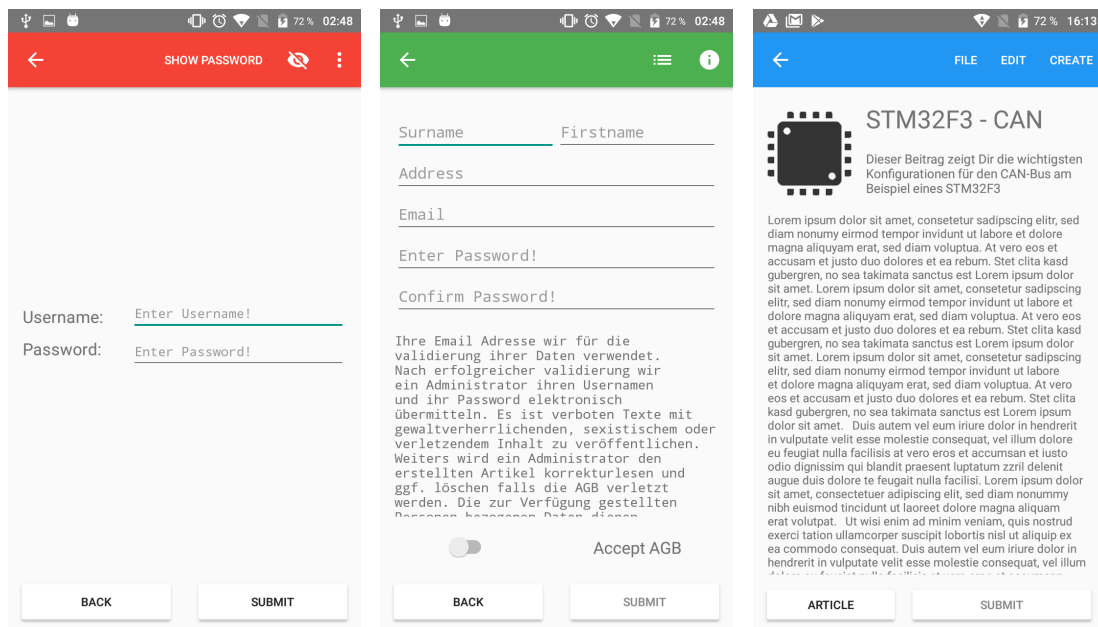


Abbildung 3.1: Design der STM32KB App - v.l.n.r. Login, Register, Lesen

Für die Entwicklung beider CP-Apps wurde eine andere Art der Menüführung herangezogen. Statt einer Navigation über Buttons und Pfeile sind die funktionale Anforderungen in Form von Tabs umgesetzt worden. Ein Tabbed Layout steht sowohl unter Xamarin.Native als auch Xamarin.Forms zur Verfügung und wird von dem jeweiligen Zielbetriebssystem Renderer so gerendert das es dem Typischen Design Merkmalen entspricht.

In Abbildung 3.2 ist das veränderte Menü durch Tabs ersichtlich. Dem Anwender wird durch ein übersichtlicheres Design die Handhabung der Applikation vereinfacht.

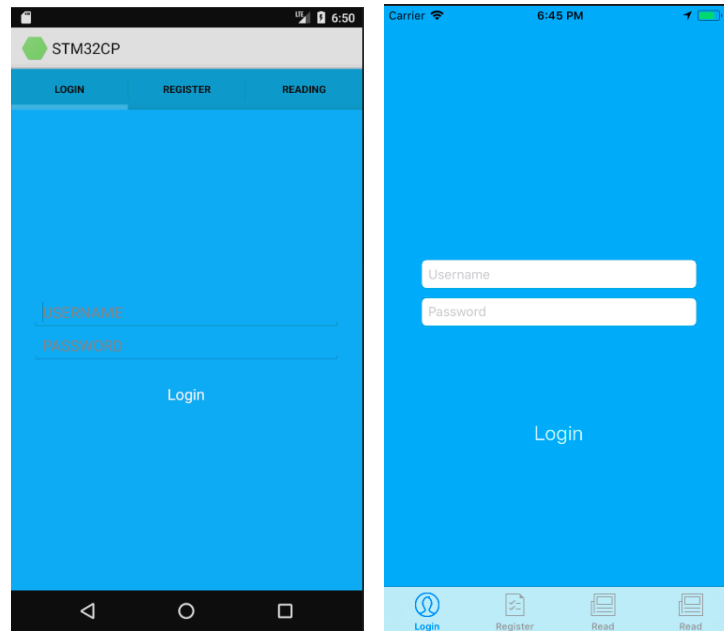


Abbildung 3.2: Design der STM32CP App

Da die STM32CP Applikation mit Xamarin.Native entworfen wurde, musste das Design sowohl für Android s.h. Abbildung 3.2 links als auch iOS, rechts in Abbildung 3.2, getrennt entworfen werden. Zwar klingt dies nun nach doppelten Aufwand für das Design, jedoch hielt sich dies in Grenzen. Das Design einer iOS App ist durch das *Storyboard*, welches für iOS Apps verwendet wird, rasch entworfen. Per Drag&Drop lassen sich die gewünschten UI Elemente im Layout platzieren und müssen durch Anpassung der *Constraints*<sup>3</sup> positioniert werden. Für das Design der Android Version kann auch die *XML* Dateien der STM32KB App zurückgegriffen werden sodass mit Copy&Paste bestimmter Elemente das Design rasch entworfen ist.

Auf Basis der Erfahrungen die, während der Spezifizierung als auch der Programmierung der CP Variante einer Nativen Android Applikation, gesammelt wurden sollen diese dabei helfen die MCKB App mit Xamarin.Forms zu entwickeln.

---

<sup>3</sup>Durch einen Constraint wird eine Beziehung zwischen Rahmen des Layouts und dem UI Element erzwingen, damit dessen relative Position immer gleich ist

## 3.1 Applikations Spezifizierung

Die Spezifizierungen sollen weitestgehend an die vorhergehenden Applikationen anknüpfen. Ein Schlüsselwort wie muss, erzwingt die Umsetzung während sollte, die Möglichkeit der Umsetzung vorschlägt.

Die Spezifizierungen umfassen folgende Bereiche:

- Design
- Funktionale Anforderungen
- Nicht Funktionale Anforderungen

**Design:** Das Design der App muss mit maximal drei Menüebenen bedienbar sein. Sofern Xamarin.Forms ein Tabbed Layout zur Verfügung stellt so sollte dieses für eine bestmögliche Benutzerfreundlichkeit verwendet werden. Für die Auflistung der Artikel ist folgende Hierarchie zu wählen:

- Dropdown für Kategorien
  - Kategorie 1 - i.e. **STM32F1**
    - \* Unterkategorie 1.1 - i.e. **STM32F1 - UART**
    - \* Unterkategorie 1.2 - i.e. **STM32F1 - CAN**

Bezüglich der Farbwahl sollte diese harmonisch sein, demnach blasse Farben die nicht zu sehr in den Vordergrund drängen.

**Funktionale Anforderungen** sollen jene Funktionalitäten beschreiben die direkt mit den Anforderungen an die Applikation zusammen hängen. Funktionale Anforderungen können nach folgenden Schema<sup>4</sup> definiert werden:

Use Case	Name der Anforderung
Kurzbeschreibung	Kurze Definition was passieren soll
Vorbedingung	Was muss vorher gegeben sein
Nachbedingung	Wie ist der Zustand nachher
Fehlersituation	Welcher Fehler kann auftreten
Systemzustand im Fehlerfall	-
Akteure	Wer verwendet die Anforderung
Standardablauf	Systematischer Ablauf für die Anforderung
Alternativabläufe	-
Trigger	Wie wird die Anforderung angestoßen

Tabelle 3.1: Standard Schema für Funktionale Anforderungen (**Use Cases**) laut SRS

<sup>4</sup>Das SRS Dokument wurde in der Vorlesung Software Engineering im 4. Semester vorgestellt

Die in Tabelle 3.2 dargestellte Anforderung musste nicht angepasst werden.

Use Case	Author Login
Kurzbeschreibung	Anwender können sich als Autoren anmelden
Standard Ablauf	<ol style="list-style-type: none"> <li>1. Der Anwender öffnet die App</li> <li>2. Der Tab <b>Login</b> wird ausgewählt</li> <li>3. Der Anwender gibt <i>Username</i> und <i>Password</i> ein</li> <li>4. Die Applikation übermittelt die eingegebenen Daten an die Datenbank und lässt diese überprüfen.</li> <li>5. Bei erfolgreicher Prüfung ist der User für die Dauer der Verwendung der Applikation als Autor angemeldet und kann (sofern die Funktion implementiert ist) Artikel erstellen oder bearbeiten</li> </ol>

Tabelle 3.2: Autoren Login - Funktionale Beschreibung

Die in Tabelle 3.3 beschriebene Anforderung wurde in der Ersten CP Version angepasst. In der Native Version der Applikation wurden nach Eingabe der Daten vier Fragen an den Anwender gestellt um dessen Wissen über Mikrocontroller zu prüfen. Die Ergebnisse der Fragen und die Daten aus dem Registrierungsformular wurden anschließend an einen Administrator der App übermittelt, damit dieser den neuen Benutzer freischalten kann. Der Schritt mit den Fragen zur Wissensüberprüfung wurde nicht implementiert um nur essentielle Funktionale Anforderungen zur Verfügung zu stellen.

Use Case	Author registrieren
Kurzbeschreibung	Anwender können sich registrieren um als Autoren freigeschaltet zu werden
Standard Ablauf	<ol style="list-style-type: none"> <li>1. Der Anwender öffnet die App</li> <li>2. Der Tab <b>Registrieren</b> wird ausgewählt</li> <li>3. Der Anwender befüllt alle Eingabefelder</li> <li>4. Der Anwender muss den Hinweis auf Freischaltung durch einen Administrator zustimmen</li> <li>5. Dem Benutzer wird visuell mitgeteilt das eine Registrierungsanfrage gestellt wurde</li> </ol>

Tabelle 3.3: Autor registrieren - Funktionale Beschreibung



Als letzte Funktionale Anforderung ist in Tabelle 3.4 die Anforderung an das Lesen von Artikeln beschrieben. Diese Anforderung ist ein Zentraler Baustein für die Implementierung des gemeinsamen Codes für die Geschäftslogik.

Use Case	Artikel lesen
Kurzbeschreibung	Benutzer können Artikel lesen
Standard Ablauf	<ol style="list-style-type: none"><li>1. Der Anwender öffnet die App</li><li>2. Der Tab <b>Lesen</b> wird ausgewählt</li><li>3. Sofern neue Artikel zur Verfügung stehen können diese durch Bestätigung eines Hinweises herunter geladen werden</li><li>4. In der Page erscheint ein Drop Down Menü zur Auswahl der Kategorie</li><li>5. Der Anwender wählt eine Kategorie und gelangt in die Unterkategorie</li><li>6. Es werden die Artikel der korrespondierenden Kategorie dargestellt</li></ol>

Tabelle 3.4: Artikel lesen - Funktionale Beschreibung

**Nicht funktionale Anforderungen:**

## 3.2 Unterschiede zur Xamarin.Native Version

# Kapitel 4

## Ergebnisse

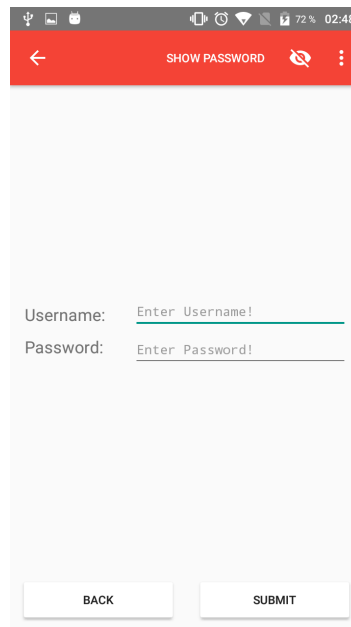
Im Zuge der Spezifizierung und Entwicklung mit Visual Studio for Mac *früher Xamarin Studio* ist es zu einigen Hürden gekommen. Angefangen bei der Projekt Erstellung, da Xamarin.Native und Xamarin.Forms in regelmäßig Updates für die IDE's veröffentlicht.

# Kapitel 5

## Fazit

# Anhang A

## Anhang/Ergänzende Information

A screenshot of a mobile application's login screen. At the top, there is a red header bar with a white back arrow on the left, the text "SHOW PASSWORD" in the center, and a white lock icon and a three-dot menu icon on the right. Below the header, the main area is light gray. It contains two input fields: "Username:" with a placeholder "Enter Username!" and "Password:" with a placeholder "Enter Password!". At the bottom, there are two white buttons with black text: "BACK" on the left and "SUBMIT" on the right. The top of the screen shows a status bar with various icons and the time "02:48".

### EIGENER ANHANG

(Hier können Schaltpläne, Programme usw. eingefügt werden.) Hier werden die Entwürfe zur Design der App gelistet sein.b

# Abbildungsverzeichnis

1.1	Weltweiter Marktanteil an Mobilen Betriebssystemen von 2007 bis 2017 Stand Q1 2018 (Quelle: <a href="https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/">https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/</a> ) . . . . .	3
2.1	Xamarin.Forms Architektur (Quelle: <a href="https://blog.goyello.com/">https://blog.goyello.com/</a> ) . . . . .	8
2.2	Wie Xamarin UI Elemente rendert . . . . .	9
2.3	Aufbau einer XAML Datei für Xamarin.Forms . . . . .	10
2.4	Projekt Struktur einer CP App . . . . .	11
2.5	Wie Xamarin.Android kompiliert wird . . . . .	12
2.6	Wie Xamarin.iOS kompiliert wird . . . . .	12
2.7	Xamarin.Forms Architektur . . . . .	13
2.8	Erstellung der Xamarin.Forms App - MCKB . . . . .	16
2.9	Unterschied Shared Code - Xamarin.Forms App - MCKB . . . . .	17
3.1	Design der STM32KB App - v.l.n.r. Login, Register, Lesen . . . . .	20
3.2	Design der STM32CP App . . . . .	21

# Codeverzeichnis

2.1	Beispiel View . . . . .	15
2.2	Beispiel ViewModel für Codeabschnitt 2.1 (View) . . . . .	15
2.3	Plattform spezifischen Code in PCL ausführen . . . . .	17
2.4	Plattform spezifischen Code in SA/SL ausführen . . . . .	18

# Tabellenverzeichnis

3.1	Standard Schema für Funktionale Anforderungen ( <b>Use Cases</b> ) laut SRS	22
3.2	Autoren Login - Funktionale Beschreibung . . . . .	23
3.3	Autor registrieren - Funktionale Beschreibung . . . . .	23
3.4	Artikel lesen - Funktionale Beschreibung . . . . .	24

# Literaturverzeichnis

- [Arm15] Marc Armgren. Mobile cross-platform development versus native development a look at xamarin platform, 2015. 5
- [BSS17] V. Bhutto, K. Soman, and R. K. Sungkur. Responsive design and content adaptation for e-learning on mobile devices. In *2017 1st International Conference on Next Generation Computing Applications (NextComp)*, pages 163–168, July 2017. 5
- [Gri15] Oleksandr Gridin. *Xamarin as a tool for cross-platform mobile development*. PhD thesis, 2015. 5
- [Her15] Dan Hermes. *Xamarin Mobile Application Development: Cross-Platform C-Sharp and Xamarin.Forms Fundamentals*. Apress, 2015. 9
- [Jen15] Derek Jensen. *Xamarin.Forms Succinctly*. Syncfusion Inc, 2015. 5
- [Joh15] Paul F. Johnson. *Cross-platform UI Development with Xamarin.Forms: Create a fully operating application and deploy it to major mobile platforms using Xamarin.Forms*. Packt Publishing, 2015. 5
- [MP16] Archit Poddar Mukesh Prajapati, Dhananjay Phadake. Study on xamarin cross-platform framework. pages 13–18, July 2016. 5
- [Pes18] Maximilian Pessl. *Cross-platform developement mittels Xamarin.Native*. PhD thesis, 2018. 4, 6
- [Rad16] Amer A. Radi. *Evaluation of Xamarin Forms for MultiPlatform Mobile Application Development*. PhD thesis, 2016.
- [Soy17] Ilke Soylemez. *WAPICE NEWS MOBILE APPLICATION*. PhD thesis, 2017.
- [Ver17] Gerald Versluis. *Xamarin.Forms Essentials: First Steps Toward Cross-Platform Mobile Apps*. Apress, 1 edition, 2017. 5, 8, 14
- [Was10] Anthony Wasserman. Software engineering issues for mobile application development. pages 397–400, 01 2010. 1