

# **Cross-Platform Development**

unter Xamarin.Forms

## **Bachelorarbeit 2**

Angefertigt an der  
Fachhochschule Campus Wien  
Bachelorstudiengang Informationstechnologien und Telekommunikation

Vorgelegt von:  
Maximilian Hans Peter Thomas Peßl  
Personenkennzeichen: c1510475049

Vertiefungsrichtung:  
Telekommunikation

Abgabetermin: 03.06.2018

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:



# Kurzfassung

Apple und Android sind heutzutage die größten Vertreter von mobilen Betriebssystemen 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 mobilen 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 zielbetriebssystemspezifischen Design Elemente zu konzentrieren. Die Arbeit befasst sich ausführlicher mit dem CPF Xamarin.Forms und wie sich die Entwicklung einer mobilen 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 phases, 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

API	Application Programming Interface
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
REST	Representational State Transfer
SA	Shared Asset
SE	Software Engineering
SL	Shared Library
UI	User Interface
XAML	Extensible Application Markup Language

# Schlüsselbegriffe

Cross-Platform  
Cross-Platform Framework  
Integrated Development Environment  
Mobile App Development  
Model View View Model  
Representational State Transfer  
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>Designentwicklung der MCKB Applikation</b>	<b>19</b>
3.1	Applikationsspezifizierung . . . . .	23
3.2	Anpassungen an die Serverumgebung im Vergleich zur Xamarin.Native Version . . . . .	26
<b>4</b>	<b>Ergebnisse</b>	<b>28</b>
<b>5</b>	<b>Fazit</b>	<b>32</b>
<b>A</b>	<b>Anhang/Ergänzende Information</b>	<b>33</b>
	Abbildungsverzeichnis . . . . .	38
	Codeverzeichnis . . . . .	39
	Tabellenverzeichnis . . . . .	39
	Literaturverzeichnis . . . . .	41



# 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 dann 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 (CPF). 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 Long Term Evolution (LTE 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.

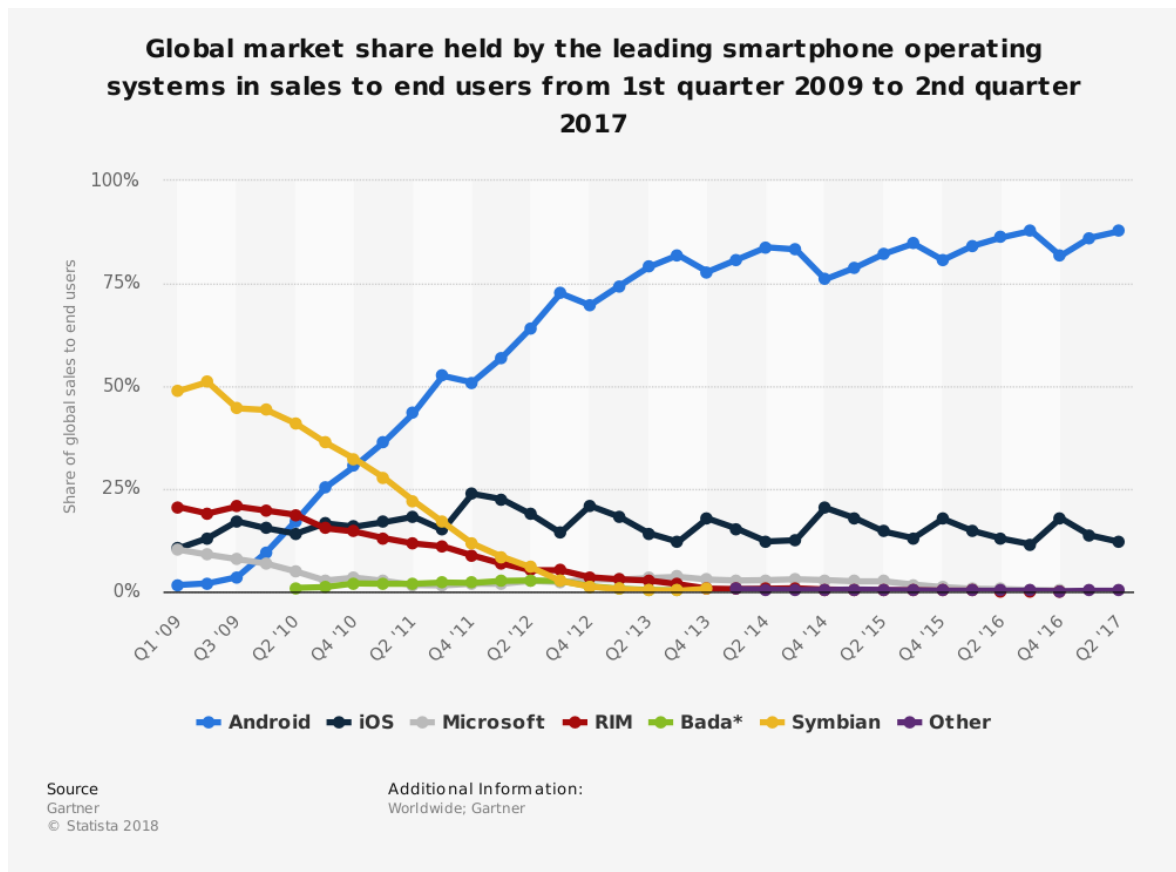


Abbildung 1.1: Weltweiter Marktanteil an mobilen Betriebssystemen von 2007 bis 2017 Stand Q1 2018 (Quelle: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>)

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 zweimal 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 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 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 Integrated Development Environment (IDE) eingebaute Compiler übernimmt die Aufgabe, den Code entsprechend so zu kompilieren, damit dieser auf der Zielpattform 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 Literaturrecherche befassen sich einerseits eine Vielzahl an Artikeln, Bachelor- oder Masterarbeiten mit den Stärken und Schwächen von CPF, 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 Onlinekurse, um das in den Büchern erlernte Wissen praktisch umzusetzen.

Oleksandr Gridin widmet sich in seiner Bachelorarbeit [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 Marc Armgren, der Performance von Xamarin als CPF in Bezug auf UI und Netzwerk Verkehr im Vergleich zu Nativer Applikationsentwicklung. Dabei wurden die Unterschiede zwischen Nativen iOS und Android und deren CP Pendant herangezogen. Die Evaluierung ergab, dass 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 vom 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 Vivek Bhuttoo, Kamlesh Soman and Roopesh Kevin Sungkur behandeln in dem Artikel [BSS17] die Stärken und Schwächen von Xamarin und stellen diese anderen bekannten CPF gegenüber. Dabei bedienten sie sich CPF 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 und Komponenten des NuGET Paket Managers. Ihre Erkenntnisse zeigen, dass es überaus wichtig ist zu wissen welche Technologien CPF verwenden und wie man diese bestmöglich für eine CP App verwenden kann.

Die Arbeit von Ilke Soylemez [MP16] erklärt wie die Entwicklung einer CP Applikation anhand eines Beispiels abläuft. Dabei wird vor allem auf essentielle Bausteine des Xamarin Frameworks eingegangen. Diese Bausteine sind unter anderem die Verwendung einer Portable Class Library (PCL), die für die gemeinsame Geschäftslogik statt einer Shared Library, den Code der CP App implementiert. Weiters wird auf das Model View ViewModel (MVVM) 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 vorhergehenden Arbeit zum Thema CP Development [Pes18] wurde die

Entwicklung einer CP Applikation auf Basis einer Nativen Applikation betrachtet. Dabei ging es vorrangig um Xamarin als CPF und welche Unterschiede oder Schwierigkeiten bei der Entwicklung einer CP 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 eine Xamarin Komponente, beispielsweise für eine Kommunikation mit einer SQL Datenbank, zur Verfügung stand, in die PCL Klasse ausgelagert werden konnte, 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 wurde ursprünglich als Firma Xamarin 2011 von Mono Entwicklern gegründet und 2016 von Microsoft übernommen. Microsoft hat Xamarin weiterentwickelt und stellt es heute als Open-Source CPF für Visual Studio ab Version 2015 und für Mac als Visual Studio for Mac zur Verfügung. Mit diesem Framework kann eine CP Applikation für Android, iOS und Windows 10 (früher Windows Mobile) geschrieben werden.

**Mono** ist eine Software Plattform, die es ermöglicht CP 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 um 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 (JIT) Compiler, welcher bei Xamarin.Android verwendet wird, um die Anwendung für das Android Betriebssystem zu erstellen. Eine iOS Anwendung unterstützt diesen JIT Compiler nicht. Die iOS Anwendung wird in Intermediate Language (IL) kompiliert und anschließend mittels eines Apple Compilers (Ahead-of-Time Compilation) in Native Code übersetzt. Aus diesem Grund wird für die iOS Entwicklung unter Windows ein Mac benötigt.

Bei Xamarin.Forms sieht dieser Prozess etwas anders aus. Es ist die Aufgabe des Xamarin.Forms.Core Assemblers, welcher Klassen und API Schnittstellen definiert, mit den Xamarin.Native Bibliotheken zu interagieren.

**Xamarin** als CPF 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 Phase, jedoch spätestens in der Design Phase geklärt sein.

## 2.1 Xamarin.Forms Überblick

Ziel von Xamarin.Forms ist es, das Maximum an gemeinsamen 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 Applikationscode nativ auf dem Zielbetriebssystem ausgeführt [Ver17] und so gerendert, dass es das typische Design darstellt.

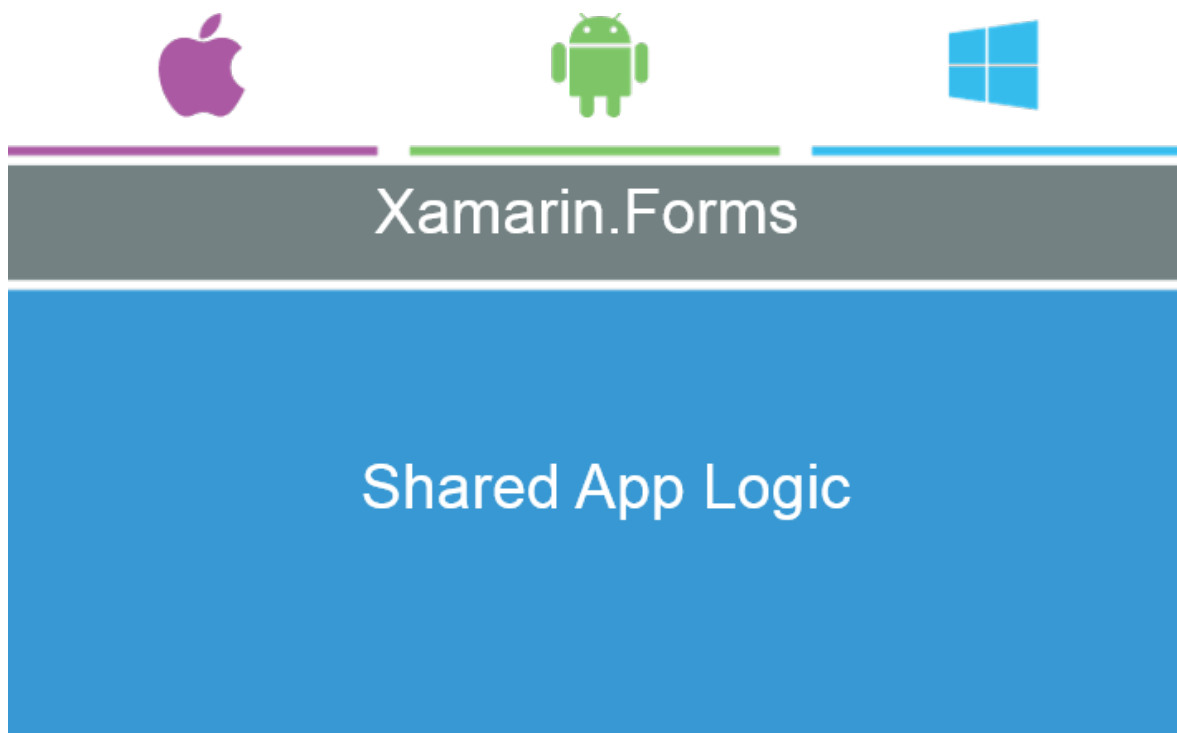


Abbildung 2.1: Xamarin.Forms Architektur (Quelle: <https://blog.goyello.com/>)

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, Stand Q2 2018.

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% zielplattformspezifischen 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.

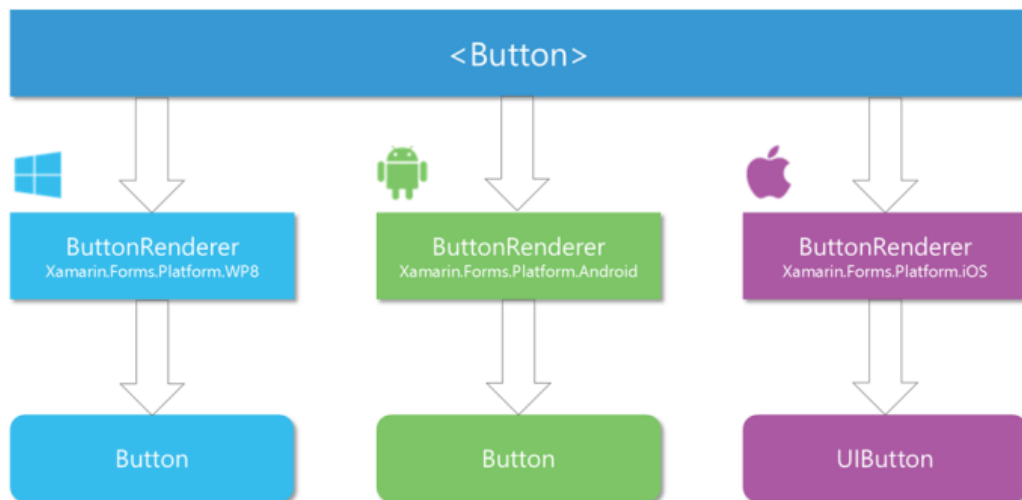


Abbildung 2.2: Wie Xamarin UI Elemente rendert  
(Quelle: [igorelikblog.wordpress.com/2016/07/08/xamarin-form-memory-management](http://igorelikblog.wordpress.com/2016/07/08/xamarin-form-memory-management))

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 das Xamarin.Forms UI Element in Native 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.Forms einzigartig, da im Vergleich mit anderen CP 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.

<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, dass 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 (Funktionen und Variablen Deklarationen). Die Referenzierung erfolgt über die Namespace Deklarationen im XAML File.

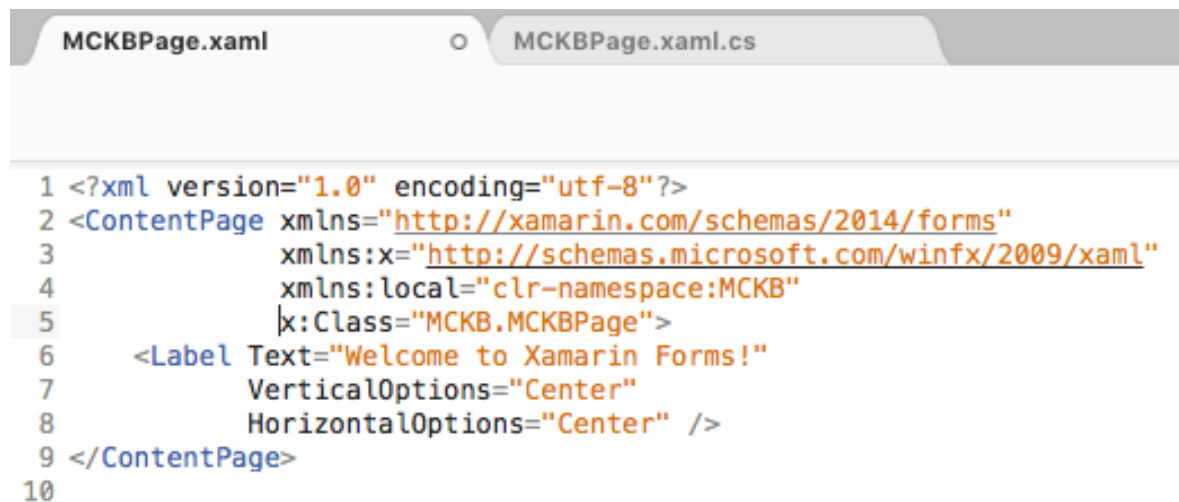


Abbildung 2.3: Aufbau einer XAML Datei für Xamarin.Forms

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 des Projektes, in welcher sich der gemeinsame Code für die CP App befindet.

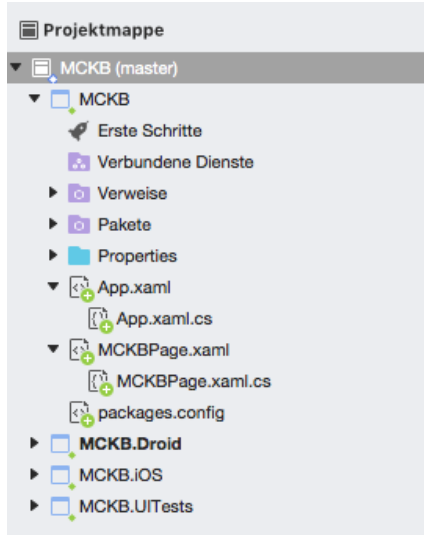


Abbildung 2.4:  
Projekt Struktur einer CP App

In Abbildung 2.4 sind alle relevanten Klassen aufgelistet, die für eine CP Applikation benötigt werden. Die **PCL** Klasse ist im Ordner *MCKB*<sup>a</sup> 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. Der Ordner *Verweise* wird 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.

<sup>a</sup>MCKB - Micro Controller Knowledge Base

## 2.2 Entwicklung unter Xamarin.Forms

Wie schon in Abschnitt 2 kurz angesprochen, werden die Unterschiede der Kompilierung von Xamarin.Forms in diesem Abschnitt erläutert. 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 Projektstruktur die Zielplattform aus und markiert diese als Startprojekt. In Abbildung 2.4 ist dies zum Beispiel *MCKB.Droid*, da dieser Ordner durch Fettschrift hervorgehoben wurde.

Der Prozess der Kompilierung ist in Abbildung 2.5 zu erkennen.

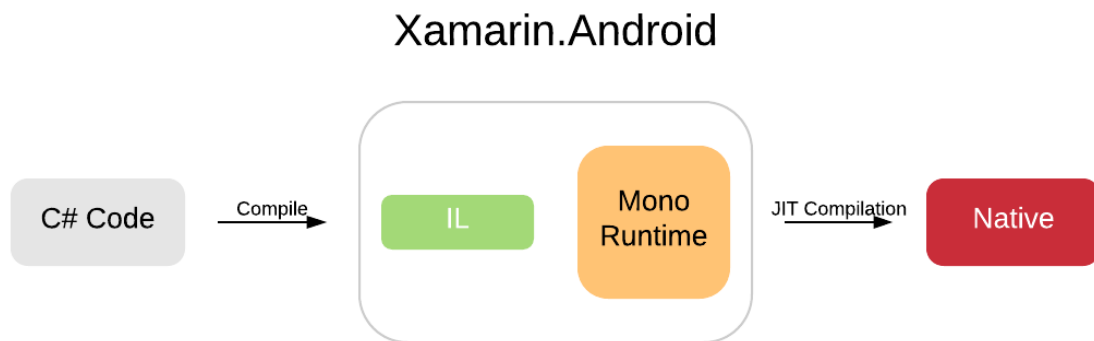


Abbildung 2.5: Kompiliervorgang von Xamarin.Android

Der C# Code wird in die IL umgesetzt und in die Mono Runtime geladen. Anschließend übernimmt der JIT Compiler die Übersetzung in Nativen Code, der anschließend auf dem Zielbetriebssystem ausgeführt wird. Dieser Prozess läuft im Xamarin Framework automatisch ab, sobald die Applikation erstellt und in den virtuellen Emulator oder auf das Hardware Gerät geladen werden soll.

Wählt man in Abbildung 2.4 iOS als Startprojekt (*MCKB.iOS* ist dann in Fettschrift 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: Kompiliervorgang von Xamarin.iOS

Für die Erstellung der iOS App wird, vergleichbar 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 AoT Compiler, der nun die Übersetzung in den Nativen Code übernimmt.

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

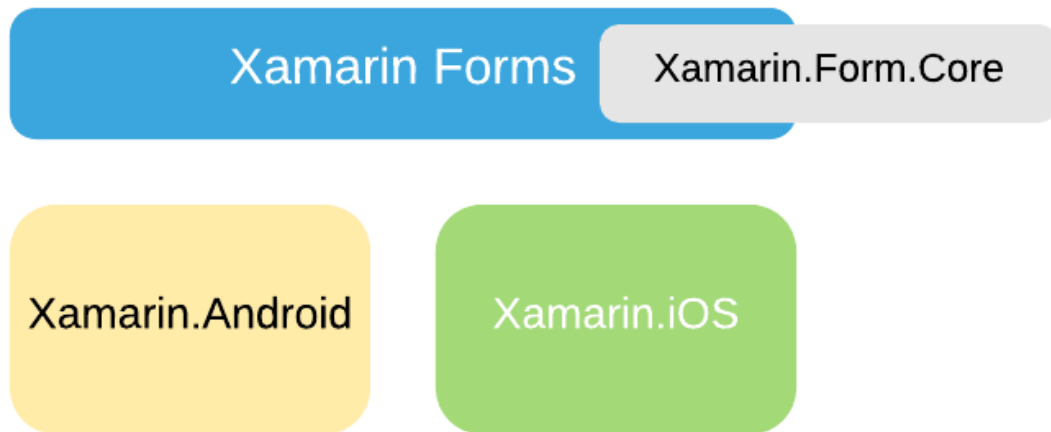


Abbildung 2.7: Xamarin.Forms Architektur

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 Buttons von Xamarin.Forms). Jener Teil, 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 zielplattformspezifische 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 von Applikationen mit Xamarin wird auf das MVVM Pattern, verwiesen da es eine Architekturstruktur für UI 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:** Dies ist das, was der Anwender sieht wenn er die Applikation verwendet. Artikel, der MCKB CP App, 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 GridLayout

**ViewModel:** Dies ist ein Model, welches für einen View entworfen wurde. Es ist eine Klasse mit Eigenschaften, welche den Zustand eines Views sowie Methoden und dessen Logik 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 einem **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 CP Applikation wird keine Unit/Modul Tests implementieren.

## 2.2.2 Projekterstellung mit Visual Studio for Mac

Für die Entwicklung der CP 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 CP Code geteilt werden soll:

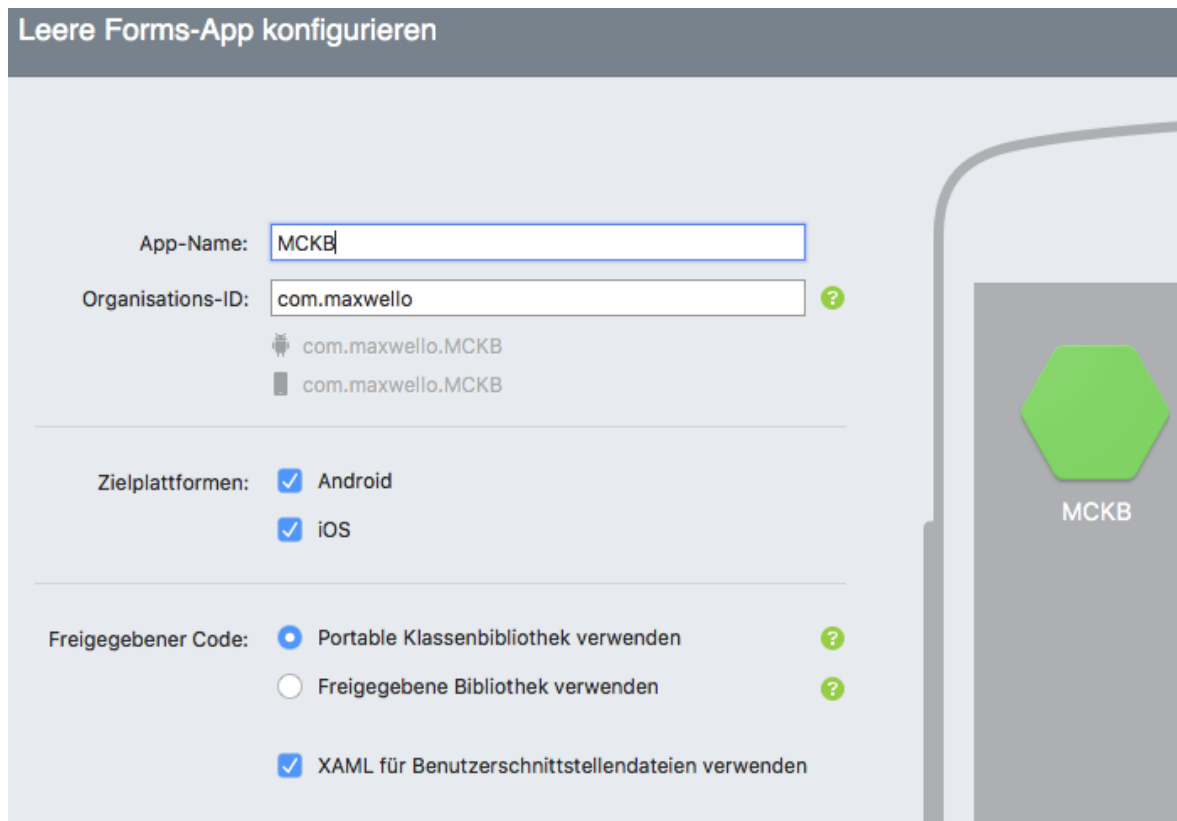


Abbildung 2.8: Erstellung der Xamarin.Forms App - MCKB

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.



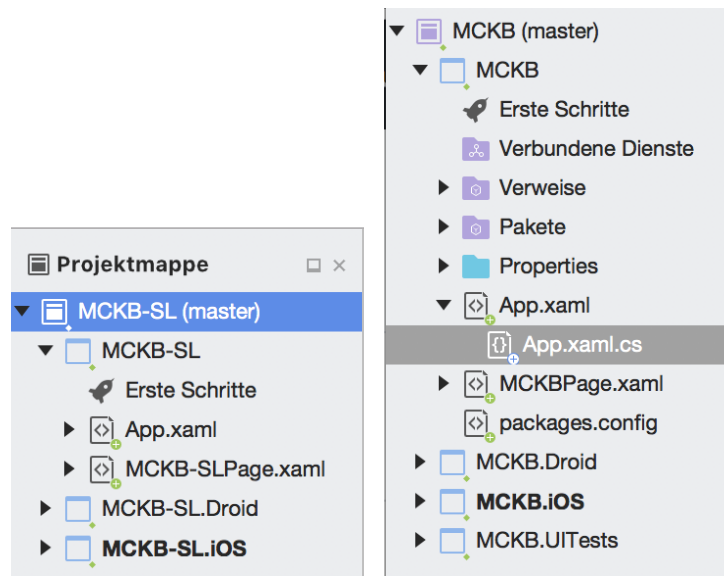


Abbildung 2.9: Unterschied Shared Code - Xamarin.Forms App - MCKB

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 Projekt Files (dependencies) einbinden muss.

Der Unterschied, 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 **PCL** erstellt (in Abbildung 2.8 ersichtlich) alle notwendigen Verweise für eine Android und iOS CP 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 plattformspezifischen Renderer von Android und iOS, damit Elemente in der PCL Klasse in plattformspezifischen 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 CP 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: Ausführung von plattformspezifischen Code in PCL

Die **SL**<sup>4</sup> (in Abbildung 2.8 *Shared Library (SL)/Shared Asset (SA)* ersichtlich) 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: Ausführung von plattformspezifischen Code in SA/SL

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 dieser beschränkt. Jedoch kann dieses Problem durch Implementierung einer Schnittstelle behoben werden, 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

## Designentwicklung der MCKB Applikation

Als Basis für diese Arbeit dienen zwei während des Studiums entwickelte mobile Applikationen, nämlich:

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 CP Variante im 5. Semester an der FH Campus Wien

Die Erstere Applikation wurde im Zuge der **Mobile App Development (MAD)** Vorlesung zur Vertiefung von Software Engineering Wissen 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 die in einer MySQL Datenbank gespeicherten Artikel dar und ermöglicht 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 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 lädt 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 Specification 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

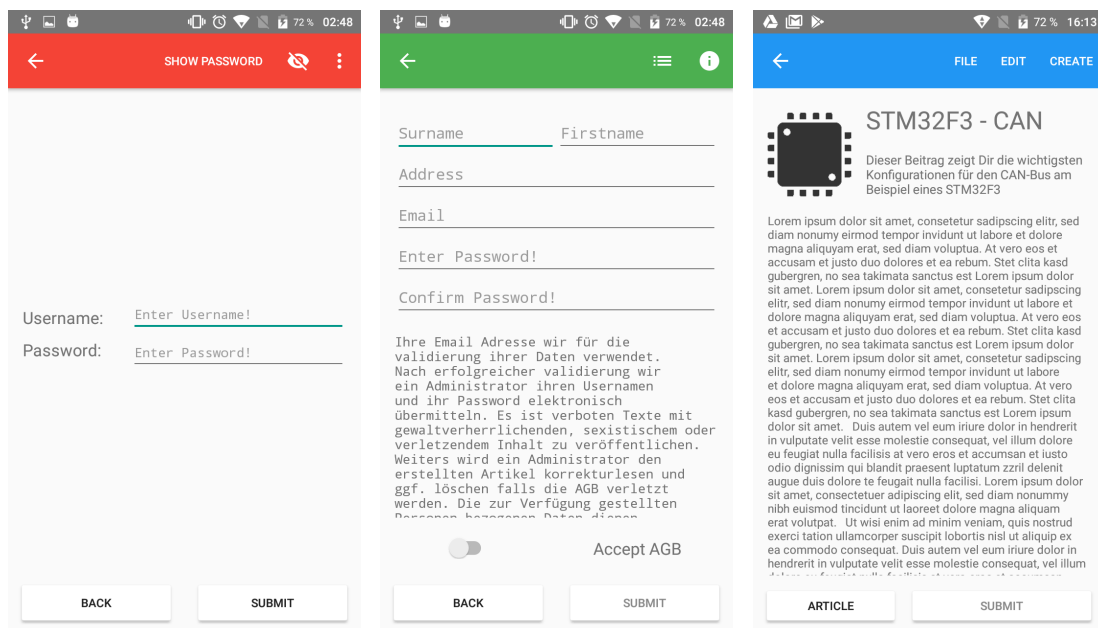


Abbildung 3.1: Design der STM32KB App - v.l.n.r. Login, Registrierung, Lesen

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 jene entsprechenden Taste auf dem Android Gerät.

Für die Entwicklung beider CP Apps wurde eine andere Art der Menüführung herangezogen. Statt einer Navigation über Buttons und Pfeile, sind die funktionalen Anforderungen in Form von *Tabs (Registerkarten)* 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, dass es den typischen Design Merkmalen entspricht.

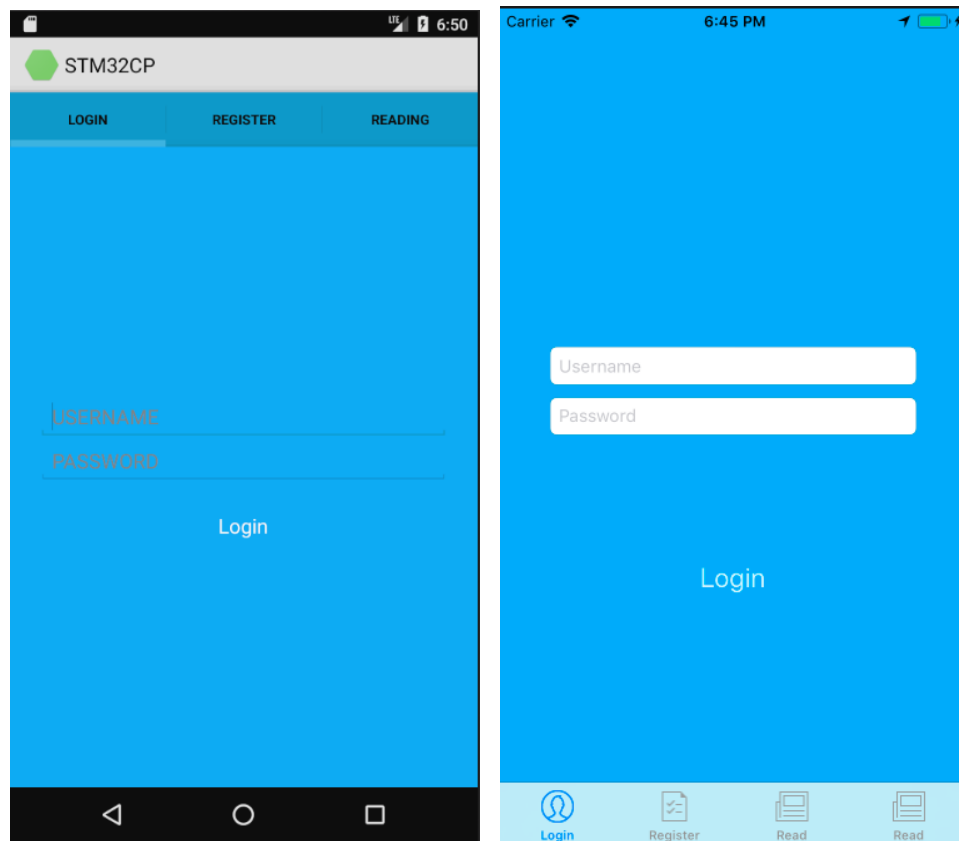


Abbildung 3.2: Design der STM32CP App

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

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 doppeltem 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 auf die XML Dateien der STM32KB App zurückgegriffen werden, sodass mit Copy&Paste bestimmter Elemente das Design rasch entworfen ist.

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

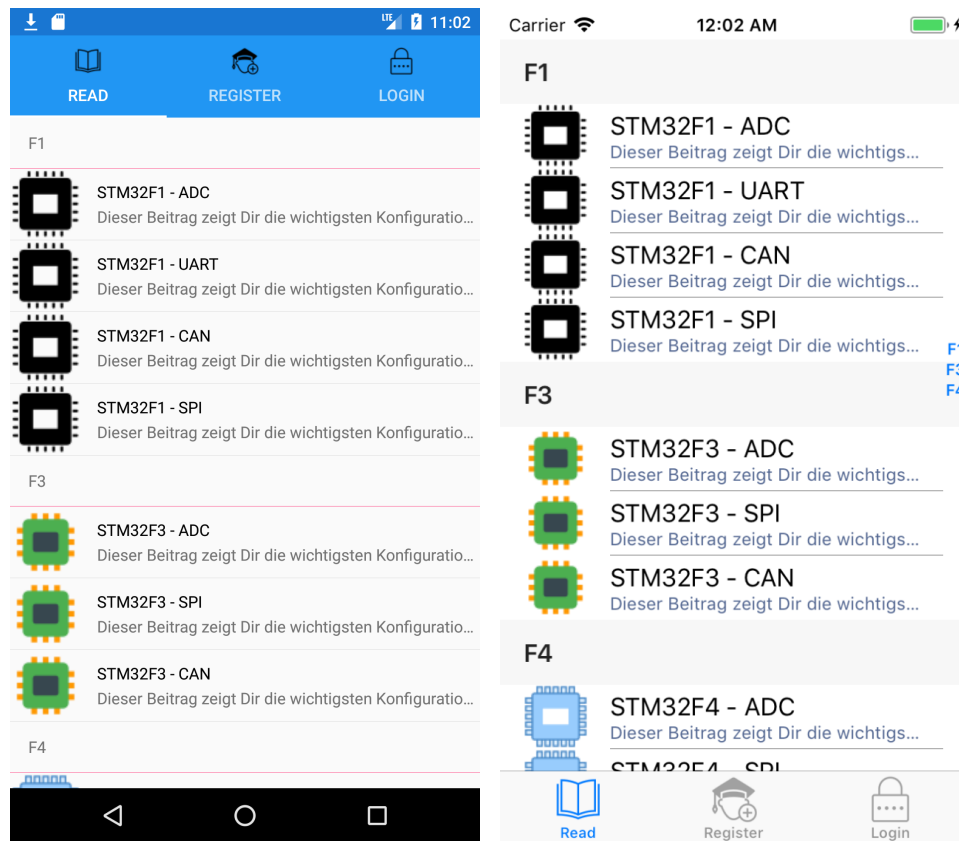


Abbildung 3.3: Designentwurf der MCKB App - Beispiel Artikel lesen

Vergleicht man dies nun mit dem ersten Design Entwurf der MCKB Applikation, ersichtlich in Abbildung 3.3, so ist trotz systemspezifischer Designunterschiede, eine klare Ähnlichkeit zwischen den Zielplattformen zu erkennen. Die *Page* der App ist zur Gänze in der PCL Klasse umgesetzt worden und benötigt einen minimalen zielplattformspezifischen Code für die Darstellung der UI Elemente, jedoch nicht für die Geschäftslogik.

Für Android, siehe Abbildung 3.3 links, ist es klassisch das die *Tabs* für ein *Tabbed Layout*<sup>4</sup> im oberen Bereich des Bildschirms positioniert sind. Bei iOS hingegen sind die *Tabs*, die eine Page beinhalten, im unteren Bereich des Bildschirms positioniert. Das Wechseln zwischen den *Tabs* funktioniert in der Android Version durch Wischen von rechts nach links, bzw. vice versa, oder auswählen der entsprechenden *Tabs* möglich.

<sup>4</sup>In früheren Versionen von Xamarin.Forms wurden keine Icons oder benutzerdefinierte Bilder in den Registerkarten angezeigt. In der Version 2.5.0.121934 ist dies bereits möglich.

## 3.1 Applikationsspezifizierung

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

**Funktionale Anforderungen** sollen jene Funktionalitäten beschreiben, die direkt mit den Anforderungen an die Applikation zusammenhängen. Funktionale Anforderungen können nach dem folgenden Schema<sup>5</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>5</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	Autor 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 Nativen 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	Autor 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>3a. Sofern neue Artikel zur Verfügung stehen können diese durch Bestätigung eines Hinweises herunter geladen werden</li> <li>3b. Alternativ werden Artikel beim Start der App automatisch geladen</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**, sind Anforderungen an die Qualität in der die Funktionalität zu erbringen ist. Folgende nicht funktionale Anforderungen wurden für die MCKB Applikation definiert:

1. Benutzeroberfläche

- Flache Menüebenen bei Artikelauswahl
- Einfache Menüführung
- Harmonische Farben

2. Leistungsanforderungen

- Server Time-Out von maximal 30 Sekunden
- Benutzernachrichten i.e. *Toast* oder *UI Alerts* sollen kurz und informativ sein

3. Entwicklung

- Visual Studio for Mac oder Visual Studio 2015
- Android ab Version 7.0
- iOS ab Version 11.3

## 3.2 Anpassungen an die Serverumgebung im Vergleich zur Xamarin.Native Version

Für die erste CP App konnte über die Xamarin Komponente *MySQL Plugin* über Port 3306 (*MySQL*) auf die Datenbank zugegriffen werden[Pes18]. Auf Kosten der Sicherheit war dies eine einfache Lösung, um Daten aus der Datenbank in die Applikation zu laden. Um diesen Zugriff umsetzen zu können, musste der MySQL Port auf dem Server freigegeben werden, ein *Port-Forwarding* auf dem Router und ein *NAT* auf der Firewall eingerichtet werden.

Am 18. Mai 2018 wurde der Komponenten Store für Xamarin abgeschaltet<sup>6</sup>, um nur noch den **NuGET** Store für Plugins zu verwenden. Dies hatte zur Folge, dass das Hinzufügen des *MySQL Plugins* in der PCL nicht mehr möglich war. Zwar kann man das Plugin für den direkten Datenbank Zugriff dem Android und iOS Projekt einzeln hinzufügen, jedoch muss dann sowohl für das Android, als auch das iOS Projekt die Geschäftslogik für den Zugriff auf die Datenbank zweimal implementiert werden.

Infolgedessen folgt die MCKB App den Vorgaben von Microsoft und verwendet eine *RESTFul API*<sup>7</sup> (REST - Representational State Transfer), um die Daten aus der Datenbank in die App zu laden. Um diesen Zugriff auf dem Webserver über eine Webservice API zu ermöglichen, musste die genannte *RESTFul API* implementiert werden. Die Kommunikation mit dem Server ist in Abbildung 3.4 dargestellt.

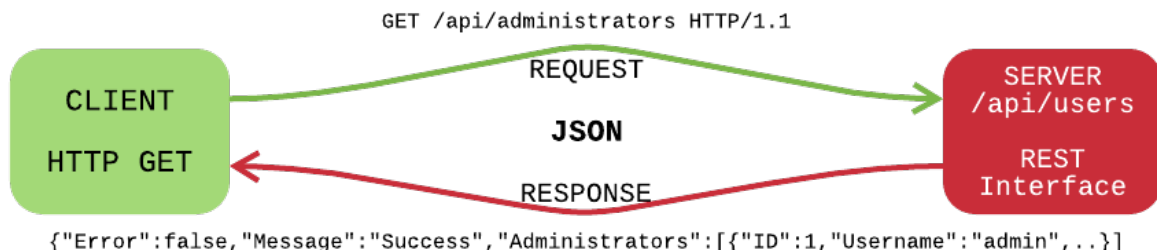


Abbildung 3.4: Client - Server Kommunikation

Die in Abbildung 3.4 dargestellte Kommunikation wird beispielsweise durch einen *GET Request* mit einer spezifischen URL, zum Beispiel *http://ipaddress:8000/api/users* vom Client ausgelöst. Der Server nimmt die Anfrage entgegen, verarbeitet diese und sendet eine *Response* im *JSON (JavaScript Object Notation)* Format an den Client zurück.

Die API baut auf *Node JS* auf und benötigt folgende Dateien:

```
1 drwxrwxr-x 8 server server 4096 Mai 29 13:04 .git
2 drwxrwxr-x 55 server server 4096 Mai 29 12:55 node_modules
3 -rw-rw-r-- 1 server server 249 Mai 29 12:55 package.json
4 -rw-rw-r-- 1 server server 3899 Mai 29 13:04 REST.js
5 -rw-rw-r-- 1 server server 1328 Mai 29 00:44 Server.js
```

Listing 3.1: Webservice Aufbau

<sup>6</sup><https://docs.microsoft.com/en-us/xamarin/cross-platform/troubleshooting/component-nuget?tabs=vswin>

<sup>7</sup><https://docs.microsoft.com/en-us/xamarin/cross-platform/data-cloud/web-services/>

Die Datei *Server.js*, in Zeile sechs des Codeabschnitts 3.1, startet den Webserver und die Schnittstelle */api*, um GET oder POST Requests anzunehmen. In der Datei *REST.js* sind die Schnittstellen zum Aufrufen von Daten konfiguriert. Der Ordner *node\_modules* beinhaltet Pakete, die für das Webservice, beispielsweise *mysql*, *md5* (erzeugt Hashwerte für Passwörter) oder *moment* (erzeugt einen Timestamp), benötigt werden.

Folgende Schnittstellen sind für die App definiert und implementiert<sup>8</sup>:

- GET **/administrators**: Liefert alle in der Datenbank gespeicherten Administratoren
- GET **/administrators/:administrator\_id**: Liefert einen bestimmten Administrator anhand dessen ID zurück
- GET **/articles**: Liefert alle in der Datenbank gespeicherten Artikel
- GET **/articles/:article\_id**: Liefert einen bestimmten Artikel anhand dessen ID zurück
- GET **/article/count**: Liefert die Anzahl an Artikeln zurück
- GET **/users**: Liefert alle in der Datenbank gespeicherten Benutzer
- POST **/users**: ermöglicht das Hinzufügen von neuen Benutzern zur Datenbank

Die Serverantwort wird bei einem der API Aufrufe als *JSON* zurückgeliefert.

```

1 {
2   "Error": false,
3   "Message": "Success",
4   "Articles": 27
5 }
```

Listing 3.2: Serverantwort - **/article/count**: API Call

Um einen neuen Benutzer in der Datenbank anzulegen, wird die URL *http://.../api/users* aufgerufen und die notwendigen Parameter übergeben.

```

1 POST /api/users HTTP/1.1
2 Host: m4xwe11o.ddns.net:8000
3 Content-Type: application/x-www-form-urlencoded
4 Cache-Control: no-cache
5 Postman-Token: b544e89d-a5e0-c3ba-db6a-5efb0122e27a
6
7 surname=Thomas&firstname=Pessl&address=Musterstrasse+18%2F2%2C+1110+Wien
8 &email=maximilian%40pessl.at&password=supersafe&confpassword=supersafe
```

Listing 3.3: Anlegen eines neuen Benutzers durch URL Encoding

In Codeabschnitt 3.3 wird ein HTTP POST auf die URL **/api/users** aufgerufen und in Zeile acht alle notwendigen Parameter als Teil der URL übergeben. Die Geschäftslogik für das Anlegen eines neuen Users ist in Anhang A in Abbildung A.1 zu finden.

<sup>8</sup>Einstiegspunkt ist <http://m4xwe11o.ddns.net:8000/api>.

# 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 regelmäßigen Abständen Updates für die IDEs veröffentlicht. Dabei ist es immer wieder dazu gekommen, dass die Entwicklungsumgebung keine der CP Applikationen erstellen konnte. Um dieses Problem zu umgehen, dürfen nach der Projekterstellung die Paket Aktualisierungen nicht gemacht werden. Wird das Projekt mit einer Xamarin.Forms Version erstellt, kann damit ohne Probleme gearbeitet werden. Ein Aktualisieren der Plugins und Pakete wird dann notwendig, wenn man auf Funktionen einer neueren Xamarin.Forms Version zugreifen will.

Somit musste die MCKB App mit Xamarin.Forms in der Version **2.5.0.121934** entwickelt werden. Wurde Xamarin.Forms auf, die zu diesem Zeitpunkt geschriebene Arbeit, Version **3.0.0.482510** aktualisiert so erhielt man bei Erstellung der iOS oder Android App, die in Abbildung 4.1 dargestellte Fehlermeldung. Der Fehler konnte trotz einer Vielzahl an Workarounds und diverser Anpassungen an den Referenzen in den Verweisen des Projekts nicht behoben werden. Ein Aktualisieren der Pakete und Plugins wurde deshalb vermieden.


!	Zeile	Beschreibung
	<input type="checkbox"/> 44	<b>Xamarin.Forms tasks do not match targets. Please ensure that all projects reference the same version of Xamarin.Forms, and if the error persists, please restart the IDE. (XF002)</b>

Abbildung 4.1: Fehlermeldung nach Aktualisierung aller Projektpakete

Der Einsatz von Webservices, siehe Abschnitt 3.2, ist einerseits auf die Deaktivierung des Xamarin Komponenten Store und andererseits auf die in Codeabschnitt 4.1 dargestellte Fehlermeldung bei hinzufügen des Plugins, **Xamarin.MySQL.Data.1.0.1**, zurückzuführen.

```

1 Install failed. Rolling back...
2 Package 'Xamarin.MySql.Data.1.0.1' does not exist in project 'MCKB'
3 Package 'Xamarin.MySql.Data.1.0.1' does not exist in folder
4 '/Users/maximilianpessl/Projects/MCKB/packages'
5 Executing nuget actions took 46,83 ms
6 Could not install package 'Xamarin.MySql.Data.1.0.1'.
7 You are trying to install this package into a project that targets
8 '.NETPortable,Version=v4.5,Profile=Profile7',
9 but the package does not contain any assembly references or
10 content files that are compatible with that framework.
11 For more information, contact the package author.
```

Listing 4.1: Fehlermeldung - Plugin Installation

Anpassungen an dem Projekt (MCKB PCL), zum Beispiel ändern der PCL Einstellungen welche in Zeile acht in Codeabschnitt 4.1 dargestellt sind, haben diese Fehlermeldung nicht beheben können, wodurch auf *REST* als Webservice API zurückgegriffen werden musste. Das Plugin konnte den Projekten *MCKB.Droid* und *MCKB.iOS* hinzugefügt werden, jedoch bedeutete dies, dass die Client (Smartphone App) Server Kommunikation in beiden Projektordnern implementiert werden musste.

Zur Messung der Kommunikationszeit zwischen dem Senden eines *GET Request* und Empfangen einer *Response* wurden vor dem Senden bzw. nach Empfangen die Systemzeit erhoben und in folgender Tabelle 4.1<sup>1</sup>. Der dafür notwendige Code steht in Anhang A Codeabschnitt A.1 festgehalten. zur Verfügung.

	Xam.Forms.iOS	Xam.Forms.Droid	Xam.Nat.Droid	Xam.Nat.iOS	Nat.Droid
S	14:18:19.3465230	14:33:58.4397800	22:58:54.588	22:48:32.136	22:38:21.135
R	14:18:19.8951310	14:34:01.3333110	22:58:54.717	22:48:32.260	22:38:37.775
S	15:08:08.6368570	15:06:01.3397350	-	-	-
R	15:08:09.3322090	15:06:03.1629580	-	-	-
S	15:08:59.3353410	15:09:50.1086710	-	-	-
R	15:08:59.8671340	15:09:51.9077950	-	-	-

Tabelle 4.1: Ergebnisse der Kommunikationszeit über Webservice - Gegenüberstellung mit früheren Ergebnissen[Pes18] - Smartphone Simulatoren/Emulatoren

<sup>1</sup>Xam.Forms.x - Xamarin.Forms.x App  
Xam.Nat.x - Xamarin.Native.x App  
Nat.Droid - Native Android  
S - Send // R - Receive

Leider war es nicht möglich eine Messung für ein iOS und Android Smartphone durchzuführen, da es nicht möglich war, die MCKB App auf einem iOS Gerät zu installieren. In Tabelle 4.2 sind daher nur die Ergebnisse der Android Version auf einem Alcatel Smartphone festgehalten.

	Xam.Forms.iOS	Xam.Forms.Droid	Xam.Nat.Droid	Xam.Nat.iOS	Nat.Droid
S	-	14:32:04.8747300	-	-	
R	-	14:32:08.8170480	-	-	
S	-	15:12:15.8343150	-	-	
R	-	15:12:19.6612780	-	-	
S	-	15:13:21.6869140	-	-	
R	-	15:13:25.2504780	-	-	

Tabelle 4.2: Ergebnisse der Kommunikationszeit über Webservice - Gegenüberstellung mit früheren Ergebnissen[Pes18] - Smartphone Hardware

Eine Gegenüberstellung der Android App (STM32KB) mit dessen CP Versionen (STM32CP & MCKB) war aufgrund der Änderungen seitens Microsoft nicht möglich. Da der Komponenten Store entfernt wurde, konnte die STM32CP App nicht mehr erstellt werden.

Jedoch ist in beiden Tabellen erkennbar, dass es signifikante Unterschiede zwischen den CP Versionen der STM32KB App gab. Die Ladezeiten der Xamarin.Native App war um einiges geringer als die der Xamarin.Forms Applikationen, wobei die Xamarin.Forms.iOS App annähernd gleich schnell die Daten aus der Datenbank in die Applikation geladen hat.

Die deutlich unterschiedlichen Ladezeiten für die Xamarin.Forms.Droid App könnten dadurch erklärt werden, dass ein Android Emulator nur die notwendigsten Systemprozesse im Hintergrund laufen lässt, während ein echtes Smartphone deutlich ausgelasteter ist.

Bezüglich des Designs und der Programmierung der MCKB App bietet Xamarin.Forms eindeutig einen Zeitgewinn. Der Umstand bis zu 80% oder 90% des Designs und der Geschäftslogik in dem PCL Projekt umzusetzen ist, sofern aufgrund der Spezifizierung möglich, genau das, was CP-Developing so attraktiv macht.

Allerdings gibt es eine Einschränkung bezüglich der MCKB.Droid App. In der Xamarin.Native Version gab es in der *MainActivity* mehrere *Fragments*, um das Layout schnell wechseln zu können. Die Implementierung von *Fragments* ist im PCL Projekt nicht möglich. Weiters impliziert der Name Xamarin.Forms, dass es sich dabei um ein UI Framework mit klaren entscheidungsbasierten oder formularbasierten Abläufen einer App handelt. Erfordert eine CP App viele komplexe UI Elemente, so empfiehlt es sich diese mit Xamarin.Native zu entwickeln. Da die MCKB App keine komplexen UI Elemente benötigt, eignete sich Xamarin.Forms als Framework sehr gut.

Das Design wird, wie schon erwähnt im PCL Projekt für iOS und Android, gemeinsam in den jeweiligen *XAML* Files erstellt. Zwar stellt Visual Studio for Mac eine Vorschau für das Editieren von Design Files zur Verfügung, jedoch gab es hier immer

wieder Probleme mit dem Rendern der UI Elemente. In Abbildung 4.2<sup>2</sup> ist zu erkennen, dass man durch Hinzufügen des *BackgroundColor* Parameters die Ränder des Layout Elements darstellen kann, um so die Grenzen eines *StackLayout*, wie es die Page in Abbildung 4.2 verwendet, darzustellen.

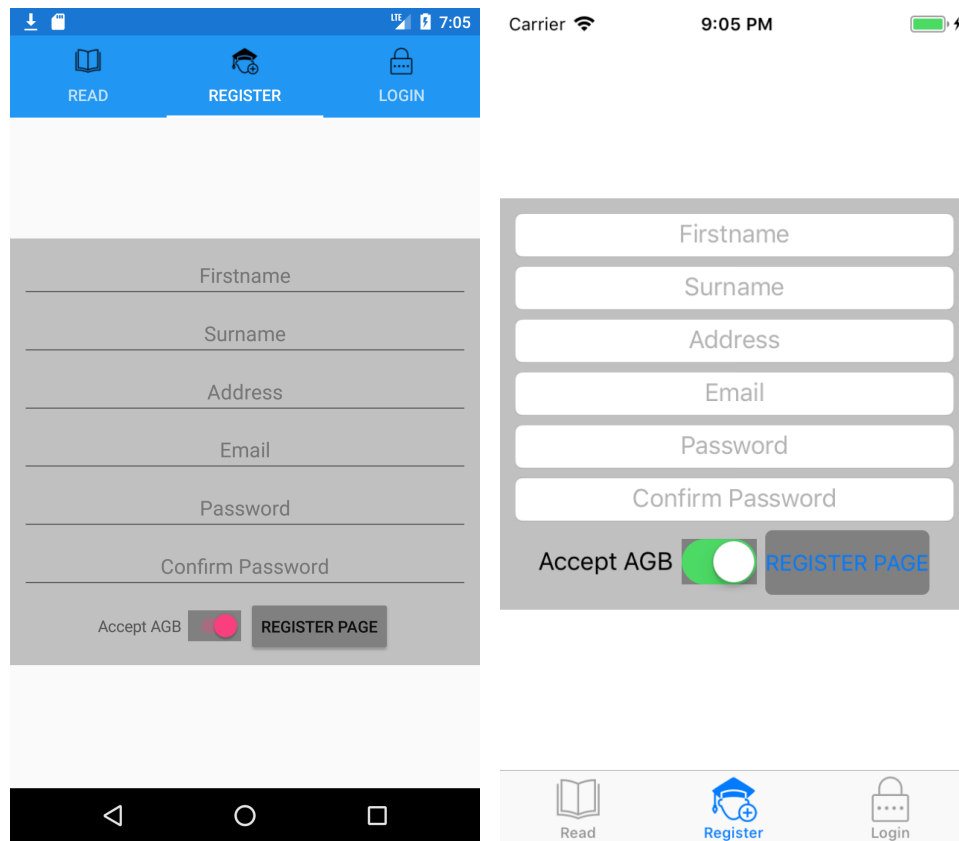


Abbildung 4.2: Darstellung von Layout Grenzen

Der gleiche Ansatz zur Abschätzung der Größe der UI Elemente wurde für die Login Page verwendet.

In Kapitel 3 Abschnitt 3.1 ist die Spezifizierung für den *UseCase* - *Artikel lesen* so definiert, dass durch eine Kategorie an Mikrocontroller die Artikel einer Kategorie durch Auswahl eines Dropdown Menüs angezeigt werden. Im Zuge der Programmierung wurde dies angepasst, ersichtlich in Abbildung 3.3, da mit einer Liste an Kategorien die wiederum eine Liste an Artikeln enthält, ein moderneres Design entworfen werden konnte.

Das endgültige Design der MCKB Applikation ist in Anhang A in Abbildung A.2 dargestellt. Der Code für dieses Design ist in Codeabschnitt A.4 zu finden. Plattformspezifische Designparameter sind durch den **tag** `<OnPlatform></OnPlatform>` definiert.

<sup>2</sup>Der XAML Code befindet sich im Anhang A Codeabschnitt A.3.

# Kapitel 5

## Fazit

Für die Entwicklung von Software auf den vollen Funktionsumfang eines Frameworks zurückgreifen zu können ermöglicht es Probleme gezielt zu lösen. Dabei ist es verständlich, dass ein Framework, wie beispielsweise Xamarin.Forms, nur einen Teil des Funktionsumfanges der Zielpattform zur Verfügung stellt. Kreativität und Erfahrung in der Software/Applikationsentwicklung spielen hierbei eine große Rolle um das Framework so zu verwenden, dass die Software entsprechend der Anforderungen entwickelt wird.

Analyse und Design waren vor Beginn und während der Entwicklung der MCKB App ständig präsent und haben direkten Einfluss eingenommen. Wurde zum Beispiel das iOS Projekt exakt den Vorgaben in dem Design File angezeigt, mussten immer wieder kleine Änderungen an der Android Version vorgenommen werden. Jedoch musste dabei kein Android spezifischer Code geschrieben werden, sondern durch plattformspezifische Parametrierung jene Anpassung vorgenommen werden, um ein ähnlich strukturiertes Design, wie in der iOS Version, zu erzielen. Anfängliche Stolpersteine, die nach wenigen Zeilen Code aufgetreten sind, konnten sowohl durch Recherche zu den Besonderheiten von Xamarin.Forms als auch Kreativität, aus dem Weg geräumt werden.

Der Umstand, dass Xamarin.Forms den großen Vorteil hat, wenig Code doppelt oder dreifach zu schreiben, steht dessen eingeschränkten Funktionsumfang für plattformspezifische Funktionen gegenüber. Ist man es gewohnt Probleme die eventuell nur Android betreffen in dessen Projekt statt der PCL zu beheben, so ist dies zwar möglich, kann im Verlauf jedoch zu ungewollten Nebeneffekten im weiteren Verlauf der Programmierung führen. Speziell bei der Entwicklung der MCKB App kam es immer wieder vor, dass ein spezifisches Problem zuerst im Projekt der Zielpattform behoben werden konnte, jedoch beim Aufkommen eines ähnlichen Problems, das Design zur Funktion die jenes Problem lösen sollte, überdacht werden musste.

Ein direkter Vergleich der Funktionen der Android App mit der CP App zeigt, dass das immerwährende Überdenken des Designs dazu geführt hat, dass die Applikationen laufend evaluiert und validiert worden sind. Jedoch spielte die erste CP App (STM32CP) eine essenzielle Rolle, weil bei der Entwicklung jener App viele Erfahrungen gesammelt werden konnten.



# Anhang A

## Anhang/Ergänzende Information

```
router.post("/users",function(req,res){
    var time = moment();
    var time_format = time.format('YYYY-MM-DD HH:mm:ss');
    var query = "INSERT INTO ??(??,??,??,??,??,??,??,??) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    var table = ["RegisteredUsers","ID","surname","firstname","address","email","password",
    "confpassword","RegistrationDate",null,req.body.surname,req.body.firstname,
    req.body.address,req.body.email,md5(req.body.password),md5(req.body.confpassword),time_format];
    console.warn(table);
    query = mysql.format(query,table);
    connection.query(query,function(err,rows){
        if(err) {
            res.json({"Error" : true, "Message" : "Error executing MySQL query"});
        } else {
            res.json({"Error" : false, "Message" : "User Added !"});
        }
    });
});
```

Abbildung A.1: RESTful - Server Code - User Anlegen

```
1 protected override async void OnAppearing(){
2     // Measing how long a request takes - START
3     DateTime Jan1970 = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
4     TimeSpan javaSpan = DateTime.UtcNow - Jan1970;
5     var time = DateTime.Now.Millisecond.ToString();
6     var time3 = DateTime.UtcNow.ToString();
7     System.Diagnostics.Debug.WriteLine(javaSpan + " Sending Request");
8     var content = await _client.GetStringAsync(Url);
9     var articles = JsonConvert.DeserializeObject<List<Article>>(content);
10    DateTime Jan19702 = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
11    TimeSpan javaSpan2 = DateTime.UtcNow - Jan19702;
12    var time2 = DateTime.Now.Millisecond.ToString();
13    var time4 = DateTime.UtcNow.ToString();
14    System.Diagnostics.Debug.WriteLine(javaSpan2 + " Received Request");
15    // Measing how long a request takes - END
16    ...}
```

Listing A.1: Messung der Kommunikationszeit - MCKB PCL

```

1  ...
2  namespace MCKB{
3      public partial class MainPage : TabbedPage {
4          private const string Url = "http://m4xwe11o.ddns.net:8000/api/articles";
5          private HttpClient _client = new HttpClient();
6          private ObservableCollection<Article> _articles;
7          ...
8          protected override async void OnAppearing(){
9              ...
10             var content = await _client.GetStringAsync(Url);
11             var articles = JsonConvert.DeserializeObject<List<Article>>(content);
12             ...
13             _articles = new ObservableCollection<Article>(articles);
14             ...
15         }
16     }
17 }

```

Listing A.2: MCKB Code-behind - Webservice Call

```

1  <!--Tabbed Page two is for the registration of an user -->
2  <ContentPage Title="Register" Icon="icons8studentregistrationfilled.png">
3      <ContentPage.Padding>
4          <OnPlatform x:TypeArguments="Thickness" iOS="0,20,0,0"
5              Android="0,0,0,0"></OnPlatform>
6      </ContentPage.Padding>
7      <StackLayout HorizontalOptions="FillAndExpand" VerticalOptions="Center"
8          BackgroundColor="Silver" Padding="10,10,10,10">
9          <Entry HorizontalTextAlignment="Center" Placeholder="Firstname"/>
10         <Entry HorizontalTextAlignment="Center" Placeholder="Surname"/>
11         <Entry HorizontalTextAlignment="Center" Placeholder="Address"/>
12         <Entry HorizontalTextAlignment="Center" Placeholder="Email"
13             Keyboard="Email"/>
14         <Entry HorizontalTextAlignment="Center" Placeholder="Password"
15             IsPassword="true"/>
16         <Entry HorizontalTextAlignment="Center" Placeholder="Confirm.Password"
17             IsPassword="true"/>
18         <StackLayout HorizontalOptions="Center" VerticalOptions="CenterAndExpand">
19             <StackLayout Orientation="Horizontal" Padding="0,0,0,0">
20                 <Label Text="Accept AGB" VerticalOptions="Center"/>
21                 <Switch IsToggled="false" x:Name="accept" BackgroundColor="Gray"
22                     VerticalOptions="Center"/>
23                 <Button Text="REGISTER_PAGE" IsEnabled="{Binding _
24                     Source={x:Reference _accept}, _Path=IsToggled}"
25                     BackgroundColor="Gray"/>
26             </StackLayout>
27         </StackLayout>
28     </StackLayout>
29 </ContentPage>

```

Listing A.3: MCKB XAML Code für Registrierung

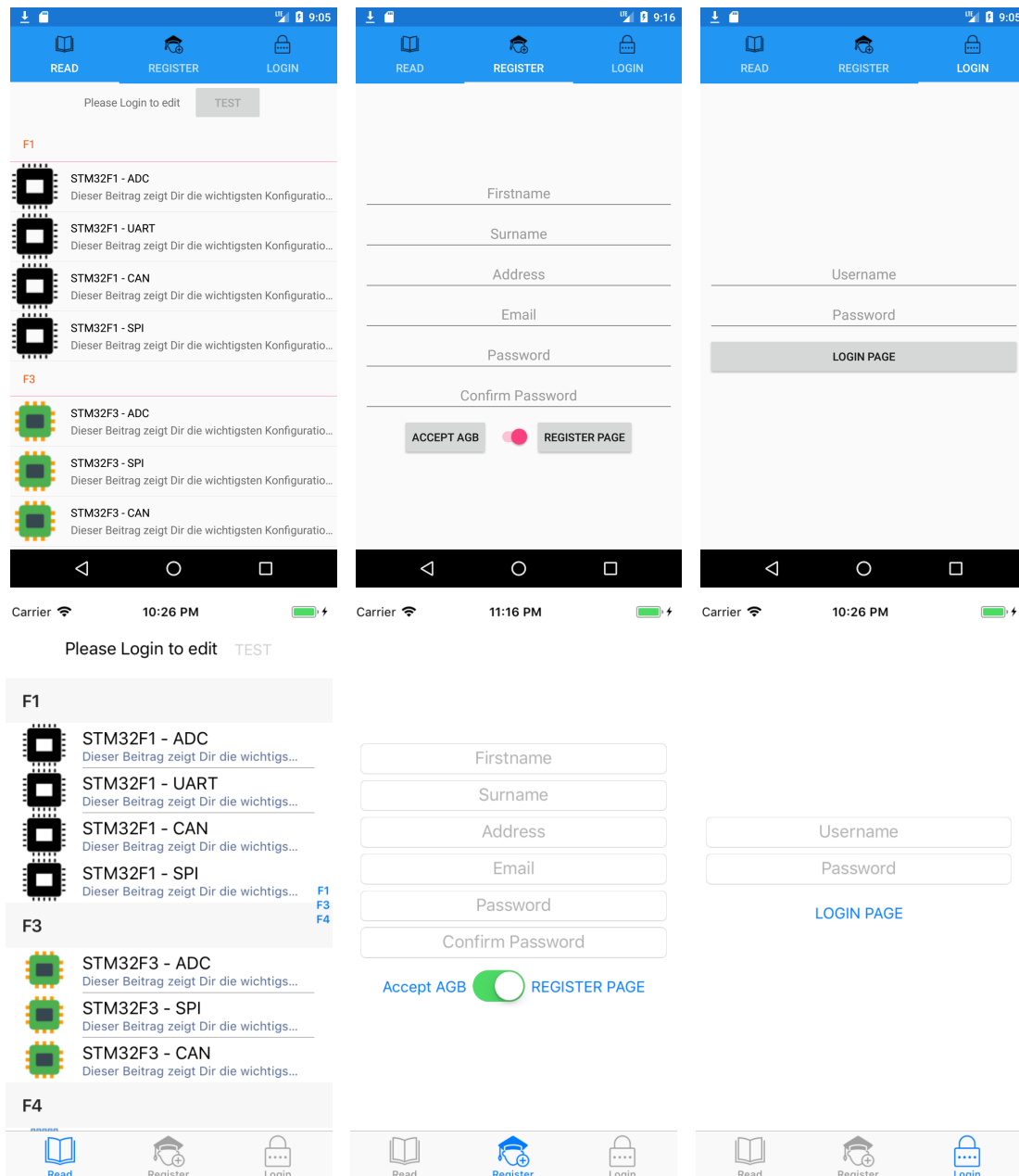


Abbildung A.2: Finales Design der MCKB App - Oben Android / Unten iOS

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="MCKB.MainPage">
3      <!--Tabbed Page one is for the Artiles -->
4      <ContentPage Title="Read" Icon="openbook.png">
5          <ContentPage.Padding>
6              <OnPlatform x:TypeArguments="Thickness" iOS="0,20,0,0"
                Android="0,0,0,0"></OnPlatform>
7          </ContentPage.Padding>
8          <StackLayout>
9              <!--On top of the ListView to provide EDIT function -->
10             <StackLayout>

```

```

11         <StackLayout HorizontalOptions="Center"
12             VerticalOptions="CenterAndExpand" Orientation="Horizontal">
13             <StackLayout.Padding>
14                 <OnPlatform x:TypeArguments="Thickness" iOS="0,5,0,0"
15                     Android="0,0,0,0"></OnPlatform>
16             </StackLayout.Padding>
17             <Label Text="Please Login to edit" VerticalOptions="Center"
18                 HorizontalOptions="StartAndExpand" />
19             <Button Text="Edit" IsEnabled="false" x:Name="editButton"
20                 Margin="10,0,0,0" Clicked="editButtonClicked" />
21         </StackLayout>
22     </StackLayout>
23     <ListView x:Name="articleListView" IsGroupingEnabled="true"
24         GroupDisplayBinding="{Binding Title}">
25         <ListView.ItemTemplate>
26             <DataTemplate>
27                 <ImageCell Text="{Binding Title}" TextColor="Black"
28                     Detail="{Binding Description}" ImageSource="{Binding
29                         ImageUrl}"></ImageCell>
30             </DataTemplate>
31         </ListView.ItemTemplate>
32         <ListView.GroupHeaderTemplate>
33             <DataTemplate>
34                 <TextCell Text="{Binding Title}" Detail="{Binding Description}"
35                     TextColor="#f35e20" DetailColor="#503026" />
36             </DataTemplate>
37         </ListView.GroupHeaderTemplate>
38     </ListView>
39 </StackLayout>
40 </ContentPage>
41
42 <!--Tabbed Page two is for the registration of an user -->
43 <ContentPage Title="Register" Icon="icons8studentregistrationfilled.png">
44     <ContentPage.Padding>
45         <OnPlatform x:TypeArguments="Thickness" iOS="0,20,0,0"
46             Android="0,0,0,0"></OnPlatform>
47     </ContentPage.Padding>
48     <StackLayout HorizontalOptions="FillAndExpand" VerticalOptions="Center"
49         Padding="10,10,10,10">
50         <Entry HorizontalTextAlignment="Center" Placeholder="Firstname"
51             x:Name="regfirstname" />
52         <Entry HorizontalTextAlignment="Center" Placeholder="Surname"
53             x:Name="regsurname" />
54         <Entry HorizontalTextAlignment="Center" Placeholder="Address"
55             x:Name="regaddress" />
56         <Entry HorizontalTextAlignment="Center" Placeholder="Email"
57             x:Name="regemail" Keyboard="Email" />
58         <Entry HorizontalTextAlignment="Center" Placeholder="Password"
59             x:Name="regpassword" IsPassword="true" />
60         <Entry HorizontalTextAlignment="Center" Placeholder="Confirm Password"
61             x:Name="regconfpassword" IsPassword="true" />
62         <StackLayout HorizontalOptions="Center"
63             VerticalOptions="CenterAndExpand">
64             <StackLayout Orientation="Horizontal" Padding="0,0,0,0">
65                 <Button Text="Accept AGB" VerticalOptions="Center"
66                     Clicked="showAgb" />

```

```
49         <Switch IsToggled="false" x:Name="accept" VerticalOptions="Center" />
50         <Button Text="REGISTER_PAGE" IsEnabled="{Binding_
           Source={x:Reference.accept}, Path=IsToggled}"
           Clicked="sendRegistration" />
51     </StackLayout>
52 </StackLayout>
53 </StackLayout>
54 </ContentPage>
55
56 <!--Tabbed Page three is for the login -->
57 <ContentPage Title="Login" Icon="icons8password.png">
58     <ContentPage.Padding>
59         <OnPlatform x:TypeArguments="Thickness" iOS="0,20,0,0"
           Android="0,0,0,0"></OnPlatform>
60     </ContentPage.Padding>
61     <StackLayout HorizontalOptions="FillAndExpand" VerticalOptions="Center"
           Padding="10,10,10,10">
62         <Entry HorizontalTextAlignment="Center" Placeholder="Username"
           Keyboard="Text" x:Name="username" />
63         <Entry HorizontalTextAlignment="Center" Placeholder="Password"
           IsPassword="true" x:Name="password" />
64         <Button Text="LOGIN_PAGE" Clicked="sendLogin" />
65     </StackLayout>
66 </ContentPage>
67 </TabbedPage>
```

Listing A.4: MCKB - Design XAML Code

# 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	Projektstruktur einer CP App . . . . .	11
2.5	Kompilervorgang von Xamarin.Android . . . . .	12
2.6	Kompilervorgang von Xamarin.iOS . . . . .	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, Registrierung, Lesen . . . . .	20
3.2	Design der STM32CP App . . . . .	21
3.3	Designentwurf der MCKB App - Beispiel Artikel lesen . . . . .	22
3.4	Client - Server Kommunikation . . . . .	26
4.1	Fehlermeldung nach Aktualisierung aller Projektpakete . . . . .	28
4.2	Darstellung von Layout Grenzen . . . . .	31
A.1	RESTful - Server Code - User Anlegen . . . . .	33
A.2	Finales Design der MCKB App - Oben Android / Unten iOS . . . . .	35

# Codeverzeichnis

2.1	Beispiel View . . . . .	15
2.2	Beispiel ViewModel für Codeabschnitt 2.1 (View) . . . . .	15
2.3	Ausführung von plattformspezifischen Code in PCL . . . . .	17
2.4	Ausführung von plattformspezifischen Code in SA/SL . . . . .	18
3.1	Webservice Aufbau . . . . .	26
3.2	Serverantwort - <b>/article/count:</b> API Call . . . . .	27
3.3	Anlegen eines neuen Benutzers durch URL Encoding . . . . .	27
4.1	Fehlermeldung - Plugin Installation . . . . .	29
A.1	Messung der Kommunikationszeit - MCKB PCL . . . . .	33
A.2	MCKB Code-behind - Webservice Call . . . . .	33
A.3	MCKB XAML Code für Registrierung . . . . .	34
A.4	MCKB - Design XAML Code . . . . .	35

# Tabellenverzeichnis

3.1	Standard Schema für Funktionale Anforderungen ( <b>Use Cases</b> ) laut SRS	23
3.2	Autoren Login - Funktionale Beschreibung . . . . .	24
3.3	Autor registrieren - Funktionale Beschreibung . . . . .	24
3.4	Artikel lesen - Funktionale Beschreibung . . . . .	25
4.1	Ergebnisse der Kommunikationszeit über Webservice - Gegenüberstellung mit früheren Ergebnissen[Pes18] - Smartphone Simulatoren/Emulatoren	29
4.2	Ergebnisse der Kommunikationszeit über Webservice - Gegenüberstellung mit früheren Ergebnissen[Pes18] - Smartphone Hardware . . . . .	30



# Literaturverzeichnis

- [Arm15] Marc Armgren. Mobile cross-platform development versus native development a look at xamarin platform, 2015. 5
- [BSS17] V. Bhuttoo, 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, 5, 26, 29, 30, 40
- [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