

Xamarin CPD

Nico Hohm

Nick Johannsen

Stefan Köhn

23. Mai 2021



Inhaltsverzeichnis

1	Überblick	3
1.1	Grundidee und Ziel	3
1.2	Funktionsweise	3
1.3	Funktionsumfang	3
1.3.1	GUI	3
1.3.2	Geräte/Hardware spezifisch	4
1.3.3	Requirements	4
2	Xallary: Laborbeispielapp	6
2.1	Aufbau	6
2.1.1	Verwendetes Designpattern	7
2.2	Permissionverwaltung	10
2.3	Customelemente	11
2.3.1	Was sind <i>Customelemente</i>	11
2.3.2	Implementation eines Customelementes für Android	11
2.3.3	App im Emulator/Smartphone	11
3	Bewertung	12
3.1	Integrierbarkeit in den Entwicklungsprozess	12
3.2	Zukunftssicherheit	12
4	Kosten/Lizenzen	13
5	Fazit	13
6	Literatur / Kommentierte Links	14

1 Überblick

1.1 Grundidee und Ziel

Xamarin ist eine Open Source Plattform von Microsoft. Das Ziel von Xamarin ist es die Cross Plattform Entwicklung mobilen Applikationen zu erleichtern. Hierfür werden C# als Programmier-, XAML als Markup Sprache und .NET bzw. .Net Core als Framework verwendet.

Das Prinzip hinter Xamarin besteht darin eine Codebasis zu entwickeln die zum großen Anteil für die Android, iOS und Microsoft Geräte wiederverwendet werden, wobei native Funktionen separat hinzugefügt werden können um so die Cross Plattform App-Entwicklung gegenüber der herkömmlichen Entwicklung erheblich zu vereinfachen.

Zusätzlich bietet Xamarin alle Vorteile die durch das Verwenden der modernen Programmiersprache C#, die ohne Verlust der Funktionen in die jeweilige Sprache des Mobilen Gerätes übersetzt wird.

1.2 Funktionsweise

Je nach Betriebssystem wird die C# und XAML Dateien in eine native Version kompiliert, die das Betriebssystems ausführen kann. Somit werden hierbei die Vorzüge eines Compilers genutzt im Gegenteil zu Interpretierten Sprachen, wie JavaScript.

1.3 Funktionsumfang

1.3.1 GUI

XAML ist die Markup-Language (siehe Abb. 1) von Microsoft. Diese wird unter anderem bei anderen Frameworks, wie *WPF*, *UWP* oder *Winforms* verwendet. Die Funktionalität ist im zu Webbasierenden Markup-Language mächtiger, da diese von Haus aus eigene Funktionen mitbringt, die in anderen Sprachen via Skripte möglich sind. Ein Beispiel wäre hier die Button Klick Funktion, die bereits implementiert ist und in der Code-Behind, also die C#-Klassendatei hinter der XAML, verwendet werden kann. Ein anderes wichtiges Beispiel wäre hier die Funktionalität der Converter. Das sind besondere Klassen, die User Input nimmt und dann die Werte auf Beispielsweise

Datentypen konvertiert.

```
2  <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4      x:Class="Xallary.Views.AboutPage"
5      xmlns:vm="clr-namespace:Xallary.ViewModels"
6      Title="{Binding Title}">
7
8      <ContentPage.BindingContext>
9          <vm:AboutViewModel />
10     </ContentPage.BindingContext>
11
12     <ContentPage.Resources>
13         <ResourceDictionary>
14             <Color x:Key="Accent">■#96d1ff</Color>
15         </ResourceDictionary>
16     </ContentPage.Resources>
17
18     <Grid...>
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52 </ContentPage>
```

Abbildung 1: Beispiel für XAML

1.3.2 Geräte/Hardware spezifisch

Je nach Gerät oder Betriebssystem können hier die Komponenten unterschiedlich sein, da Xamarin auf die API des Systems zugreift. Je nachdem wie Spezifisch diese Zugriffe sind, wie Beispielsweise das zugreifen auf Internen Speicher oder Funktionen wie *ApplePay*, müssen Plattformspezifisch implementiert werden.

1.3.3 Requirements

Betriebssysteme Da Microsoft mit dotnet einen Schritt in die *Crossplattformwelt* gewagt hat, kann *Xamarin* auf jedes Betriebssystem, *UNIX* oder *Windows*, verwendet werden und auch dort mit Entwickelt. Leider gibt es eine Einschränkungen bei den Apple Betriebssystemen. Hier muss das System welches *XCodes* ausführen kann verwendet werden, damit der *Xamarin/C#* Code kompiliert wird und dann auf die Geräte deployed werden kann.

Eine gängige Methode bei der Xamarin Entwicklung ist es Visual Studio auf einem Mac zu benutzen um Android und iOS Apps zu entwickeln. Sollte außerdem eine version für Windows Geräte benötigt werden wird häufig

zusätzlich Windows auf dem Mac emuliert um den Xamarin Code so zu kompilieren.

Benötigte(s) Tool/IDE Zur Xamarin Entwicklung eignen sich prinzipiell eine Vielzahl von IDEs, wobei einige Funktionen mit Kommandozeilen Befehlen ergänzt werden müssten.

Für eine einfachere Erfahrung werden Microsofts eigene IDEs Visual Studio bzw. Visual Studio Code empfohlen. Beide verfügen über eine kostenfreie Community Version.

Visual Studio bietet zusätzlich einen einfachen Emulator von mobilen Geräten, wodurch eine Entwicklung ohne das jeweilige Gerät mit einer großen Auswahl an Gerätetypen und API Leveln möglich ist.

Während unserer Entwicklung und auch für die Aufgaben haben wir uns deshalb für Visual Studio entschieden. Außerdem bietet der Visual Studio Installer eine einfaches und verständliches Interface zum Downloaden und Anpassen der benötigten Frameworks.

2 Xallary: Laborbeispielapp

Für das bessere Verständnis, zeigen von Features und zum einarbeiten in Xamarin haben wir eine App geschrieben, die momentan **unter Android** kompiliert wird, geschrieben. In dieser App werden die Zugriffe auf die Kamera, auf den Lokalen- und Externenspeicher gezeigt und wie ein Custom Element geschrieben wird, welches eine feste Android Implementation ist, da hierbei auf Hardware- und API-Komponenten zugegriffen werden, die Android spezifisch sind.

2.1 Aufbau

Eine Xamarin Applikation hat **immer** eine feste Grundstruktur, die aus drei Grundprojekten besteht. Es können natürlich auch weitere Projekte hinzugefügt werden, aber es müssen immer folgende Projekte vorhanden sein:

- Klassenbibliothek
- Android, falls nur IOS dann wird das nicht benötigt
- IOS, falls nur Android wird dieses nicht benötigt
- UWP, dieses Projekt wird benötigt, wenn für Windowsgeräte entwickelt wird. Wie auch bei IOS und Android kann es, falls nicht benötigt, entferntn/deaktiviert werden.

(Figure 2.1)

Klassenbibliothek Die Klassebibliothek ist nachdem Projekt benannt. Hier also Xallary. Wenn Crossplattformimplementationen getätigt werden, was die Stärke von Xamarin ist, muss das hier geschehen, da alle Projekte dieses als *Dependency* haben. Das einzige, was hier nicht rein sollte und darf, sind Geräte/Betriebssystemabhängige Implementationen, da diese in den anderen Projekten erfolgen müssen.

Android Sollen Android spezifische Implementationen vorgenommen werden, wie beispielsweise *Snackbars* oder *Toasts*, dann müssen diese in diesem Projekte implementiert werden. Außerdem muss bei nur androidspezifischen Hardwarekomponenten diese hier per API Zugriffe implementiert werden.

Zumteil sind aber schon Implementation in Xamarin beinhaltet und somit muss dann "nur" diese in dem Projekt angemeldet werden.

IOS Wie oben bei Android beschrieben, gilt auch hier dass spezifische *OS* Implementation hier vollzogen werden müssen.

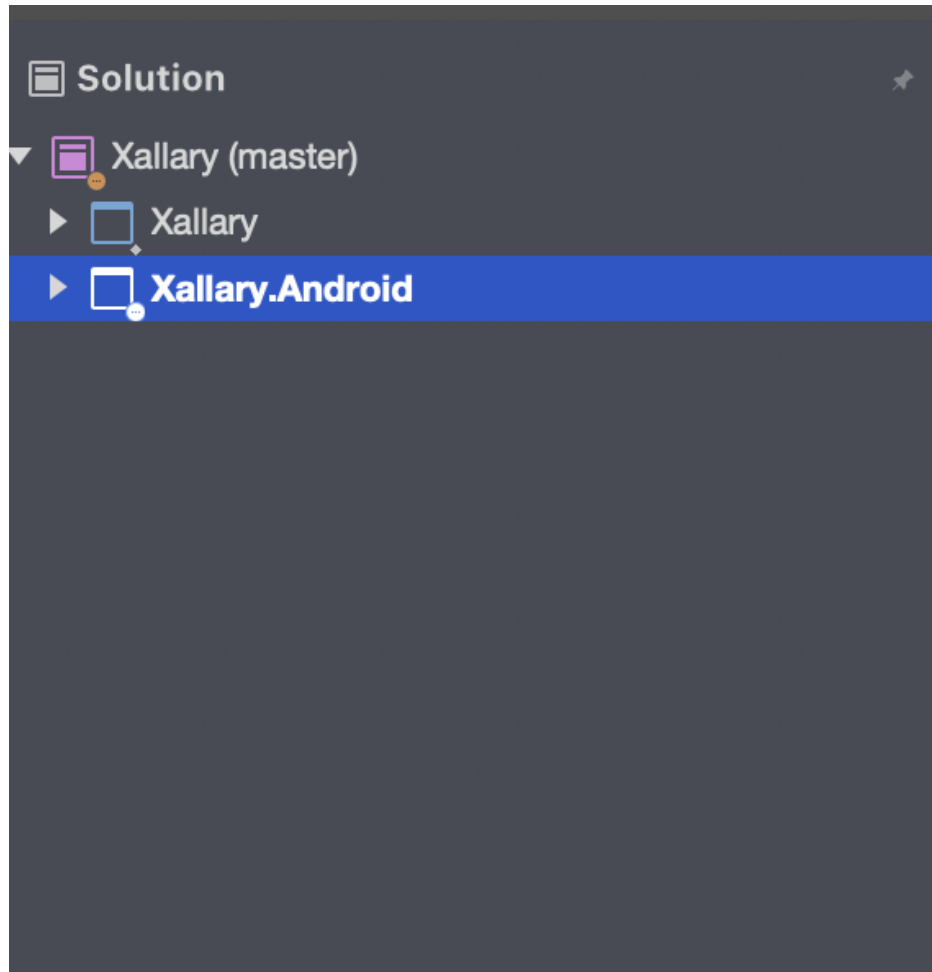
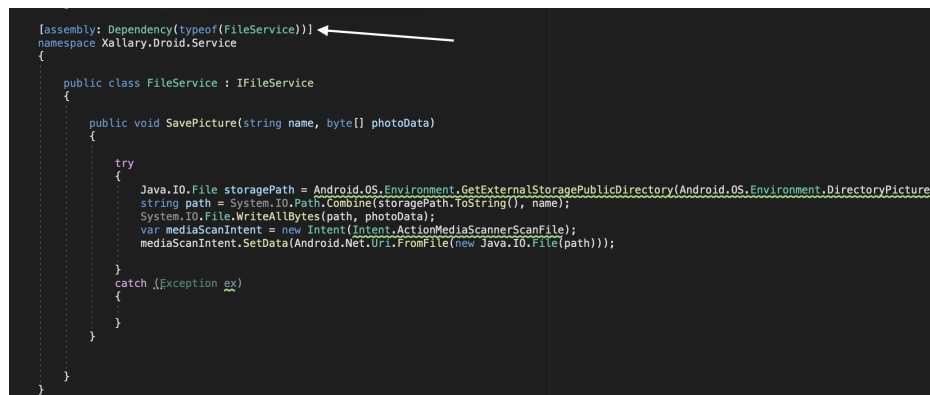


Abbildung 2: Orderstruktur von Xallary

2.1.1 Verwendetes Designpattern

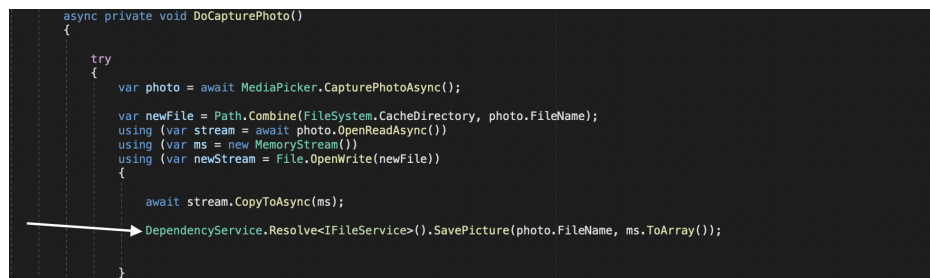
Bei den Designpattern werden verschiedene in diesem Projekt verwendet.

Zum einen wird das *Dependency Injection Pattern* von *Xamarin* verwendet, welches sich um die Verfügbarkeit von Klassen innerhalb der Unterschiedlichen Projekten kümmert. Das Pattern übernimmt die Verwaltung der Klassen, ob diese bswp. als Singleton implementiert werden soll oder die Initialisierung und Instanziierung. Dabei werden *Container* erstellt, wo die Klassen hinein “exportiert” werden, die dann wieder durch einen *import* in einer anderen Klasse importiert werden.(Figure 2.1.1, 2.1.1)



```
[assembly: Dependency(typeof(FileService))]
namespace Xallary.Droid.Service
{
    public class FileService : IFileService
    {
        public void SavePicture(string name, byte[] photoData)
        {
            try
            {
                Java.IO.File storagePath = Android.OS.Environment.GetExternalStoragePublicDirectory(Android.OS.Environment.DirectoryPicture);
                string path = System.IO.Path.Combine(storagePath.ToString(), name);
                System.IO.File.WriteAllBytes(path, photoData);
                var mediaScanIntent = new Intent(Intent.ActionMediaScannerScanFile);
                mediaScanIntent.SetData(Android.Net.Uri.FromFile(new Java.IO.File(path)));
            }
            catch (Exception ex)
            {
            }
        }
    }
}
```

Abbildung 3: Attribute Tag für das exportieren einer Klasse in den Container



```
async private void DoCapturePhoto()
{
    try
    {
        var photo = await MediaPicker.CapturePhotoAsync();

        var newFile = Path.Combine(FileSystem.CacheDirectory, photo.FileName);
        using (var stream = await photo.OpenReadAsync())
        using (var ms = new MemoryStream())
        using (var newStream = File.OpenWrite(newFile))
        {
            await stream.CopyToAsync(ms);
        }
        DependencyService.Resolve<IFileService>().SavePicture(photo.FileName, ms.ToArray());
    }
}
```

Abbildung 4: Verwendung und import der exportierenden Klasse durch ein *Get*

Als weiteres Pattern wird das *MVVM-Pattern* (Figure 2.1.1), also Model-View-ViewModel-View, verwendet. Dieses Pattern ist ein sehr gängiges in der Welt von C# beziehungsweise von WPF, UWP oder generell Frameworks welches *Xaml* als *MarkupLanguage* verwenden.

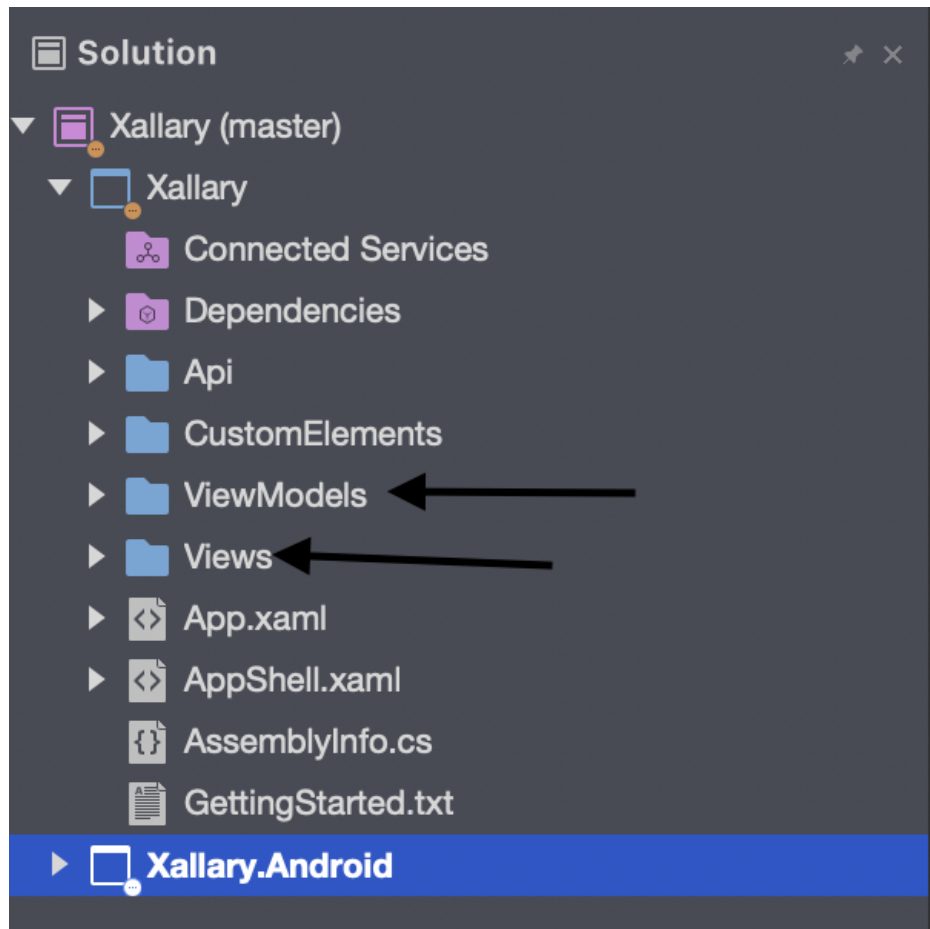


Abbildung 5: Orderstruktur des MVVM-Pattern ohne Model

Außerdem wird das *Command Pattern* verwendet, welches dafür sorgt, dass die *UI* keinen Code in der *Code Behind* besitzt, welcher Klickmethoden auslöst. (Figure 2.1.1)

```

2      <ContentPage.Content>
3      <StackLayout>
4          <Button Text="Take a Photo" Command="{Binding CapturePhotoCommand}" />
5          <Button Text="Show a Preview" Command="{Binding PickPhotoCommand}" />
6          <Image VerticalOptions="FillAndExpand" Source="{Binding PhotoPath}" IsVisible="{Binding ShowPhoto}" />
7      </StackLayout>
8  </ContentPage.Content>
9  </ContentPage>

```

Abbildung 6: Verwendung des *Command Property* mit Binding an die Methode im ViewModel

2.2 Permissionverwaltung

Wie auch in anderen Sprachen muss einer Berechtigung, also ab hier Permission, eingeholt werden, damit diverse Ressourcen der Hardware verwendet werden können. Hier werden folgende Permission (Figure 2.2) benötigt und verwendet:

- CAMERA
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE

CAMERA Diese Permission wird benötigt, damit die App auf die Kamera zugreifen kann und darf. Bei Xallary ist es wichtig da das Ziel der Applikation ist, eine funktionierende Foto machene Gallery zu sein. Ohne diese Permission ist der App untersagt auf die Kamera zuzugreifen

READ_EXTERNAL_STORAGE Da es sich hierbei um eine Gallerie handelt, muss auch eine Permission gefordert werden, um auf den Internen-Speicher zuzugreifen, um dort die persitierten Bilder zu laden und dann in der App anzuzeigen.

WRITE_EXTERNAL_STORAGE Wenn Bilder mit der Kamera aufgenommen werden, sollten diese am besten auch auf dem Gerät oder einem Speichermedium gespeichert werden. Hierbei muss eine Permission eingeholt werden, die das schreiben auf den Speicher des Gerätes erlaubt, da ansonsten keine Bilder wirklich gespeichert werden können und dann nur im *cached*es Gerätes zwischen gelagert werden.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.companyname"
3 <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="30" />
4 <application android:label="Xallary.Android" android:theme="@style/MainTheme"></application>
5 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
6 <uses-permission android:name="android.permission.CAMERA" />
7 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
8 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
9 </manifest>
```

Abbildung 7: Gesetzte Permissions in der AndroidManifest.xml

2.3 Customelemente

2.3.1 Was sind *Customelemente*

2.3.2 Implementation eines Customelementes für Android

2.3.3 App im Emulator/Smartphone

3 Bewertung

3.1 Integrierbarkeit in den Entwicklungsprozess

Mit Xamarin lassen sich einfach Cross Platform Apps entwickeln. Dank der Struktur, von globalen Klassen und Plattform abhängigen Klassen, können Plattform spezifische Funktionen jeweils für z.B. iOS oder Android implementiert werden.

Bei der Cross Plattform Entwicklung mit Xamarin wird eine App entwickelt und mit jeweiligen nativen Funktionen ergänzt statt funktionsgleiche aber programmiertechnisch völlig unterschiedliche Apps in ihrer herkömmlichen Entwicklungsumgebung zu entwickeln, wie es ohne Cross Plattform der Fall wäre.

Eine einfache App mit ohne spezielle Funktionen kann schnell erstellt werden und funktioniert für die beiden Plattformen ohne ergänzt werden zu müssen. Eine App, die viel spezifische Funktionen von der jeweiligen Plattform verwendet braucht zwar mehr Aufmerksamkeit für jede spezifische Plattform, allerdings bietet Xamarin dennoch ein erhebliches Ersparnis gegenüber der herkömmlichen App-Entwicklung für die einzelnen Betriebssysteme.

Außerdem werden viele Entwickler die vergleichsweise modernen Technologien von C# und Visual Studio gegenüber herkömmlichen Methoden bevorzugen. Funktionalitäten wie Lambdas oder Generics können mit Xamarin in C# wie gewohnt verwendet werden und werden ohne Funktionalitätsverlust in die jeweilige Sprache übersetzt, auch wenn beispielsweise Android Java selbst nicht über diese verfügt.

3.2 Zukunftssicherheit

Xamarin verbindet die populären Sprachen C# und XAML mit der App-Entwicklung für die größten Hersteller von mobilen Geräten, Xamarin füllt eine Nische, die auch langfristig erhalten bleiben sollte, vorausgesetzt es erhält genügend Support.

Da Xamarin von Microsoft betrieben wird kann davon ausgegangen werden, dass es weitergeführt wird, solange ein ausreichendes Interesse besteht. Erst kürzlich wurde die Version 5 von Xamarin.Forms veröffentlicht.

Microsoft hat mit .Net maui bereits einen Nachfolger für Xamarin in Arbeit,

allerdings basiert dieser weiterhin stark auf Xamarin. Sich als Entwickler mit Xamarin auseinanderzusetzen ist also weiterhin sinnig, da sich das gesammelte Wissen leicht auf Maui übertragen lassen wird.

Aus diesem Grund kann Xamarin weiterhin als Zukunftssicher betrachtet werden, obwohl ein Nachfolger angekündigt wurde.

4 Kosten/Lizenzen

Xamarin ist ein Open Source Projekt unter der MIT-Lizenz, die den freien und kostenlosen Umgang mit der Software gewährleistet.

Lediglich bei der IDE könnten potentielle Kosten entstehen, sollte sich ein Entwickler oder Betrieb für einen Premium Service entscheiden.

Die von Microsoft empfohlene IDE Visual Studio bietet beispielsweise Professional und Enterprise Versionen für 45\$ bzw. 250\$ im Monat.

Allerdings sind auch kostenfreie Optionen wie Visual Studio Community verfügbar und bieten einen ausreichenden Funktionsumfang für die Xamarin Entwicklung.

5 Fazit

Xamarin bietet eine benutzerfreundliche Cross Plattform Entwicklung, mit der Entwickler von mobilen Applikationen viel Zeit und Mühe einsparen können, ohne auf Funktionalität verzichten zu müssen. Mit C# und Microsofts Visual Studio ermöglicht Xamarin App Entwicklung mit moderner Programmiersprache und IDE die viele Vorteile gegenüber der herkömmlichen Entwicklungsumgebung wie z.B. Android Java enthält und so vielen Entwicklern eine attraktive Alternative zur nativen App-Entwicklung bietet. Allgemein bietet Xamarin eine verständliche und praktische Möglichkeit für Cross Plattform Entwicklung von Mobilien Applikationen und sollte daher generell von Entwicklern in Betracht gezogen werden, besonders wenn diese bereits über Kenntnisse von C# und XAML verfügen und eine Applikation für mehr als eine Plattform entwickelt werden muss.

6 Literatur / Kommentierte Links

- Xamarin GitHub
- Docs Microsoft Xamarin
- ZDNET Online Artikel