Xamarin CPD

Nico Hohm Nick Johannsen Stefan Köhn

23. Mai 2021

Inhaltsverzeichnis

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...>
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 Verständis, zeigen von Features und zum einarbeiten in Xamarin haben wir eine App geschrieben, die momentan **unter Android** kompiliert wird. 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, entfernt/deaktiviert werden.

(Figure 2)

Klassenbiliothek Die Klassebibliothek ist nachdem Projekt benannt. Hier also Xallary. Wenn Crossplattform-Implementationen 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 Projekt implementiert werden. Außerdem muss bei nur androidspezifischen Hardwarekomponenten diese hier per API Zugriffe implementiert werden.

Zum Teil sind aber schon Implementationen in Xamarin beinhaltet und somit muss dann „nur“ diese in dem Projekt „angemeldet“ werden.

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

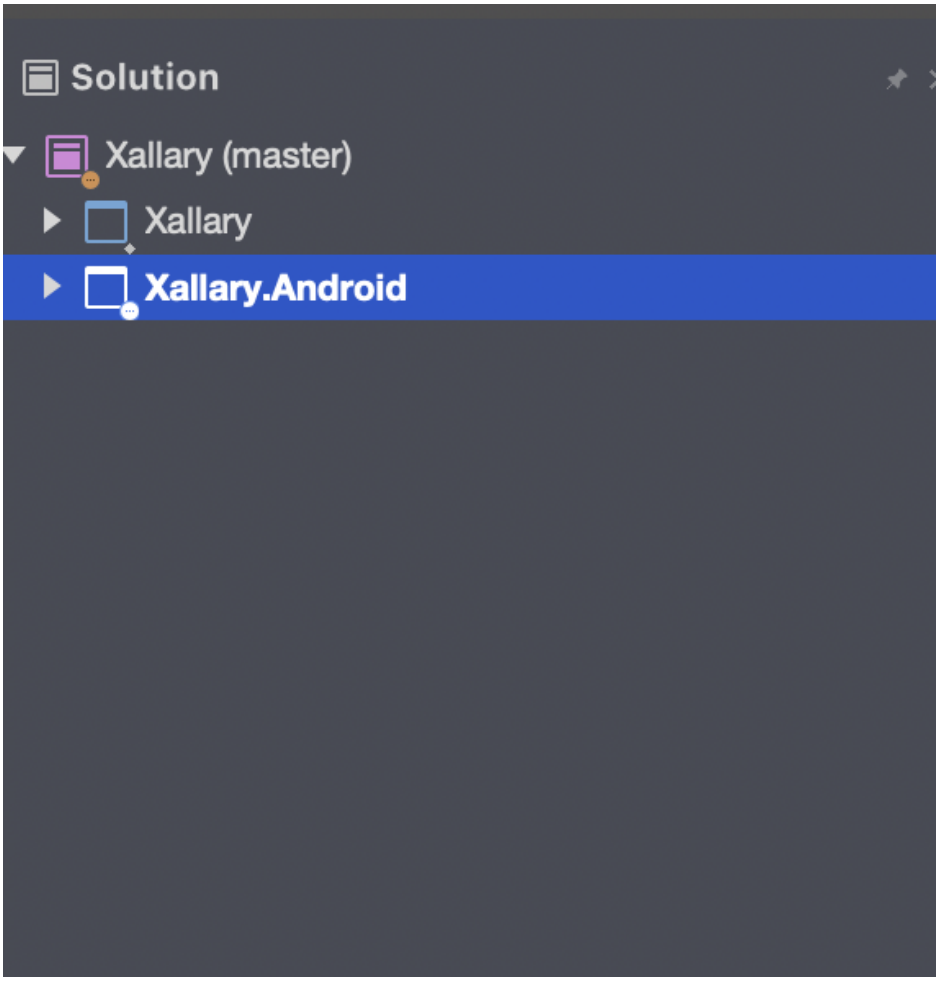


Abbildung 2: Orderstruktur von Xallary

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 Singelton 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 3, 4)

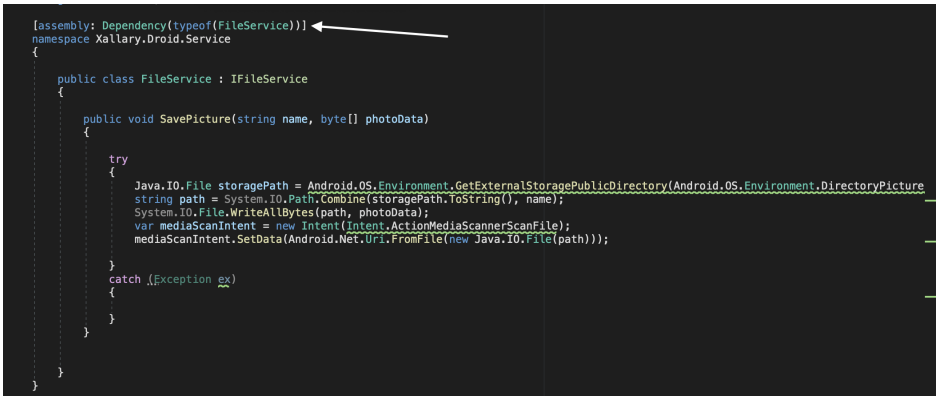


Abbildung 3: Attribute Tag für das exportieren einer Klasse in den Container

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

Als weiteres Pattern wird das *MVVM-Pattern* (Figure 5), also Model-ViewModel-View, verwendet. Dieses Pattern ist ein sehr gängiges in der Welt von C# beziehungsweise von WPF, UWP oder generell Frameworks welche *XAML* als *MarkUp Language* verwenden.

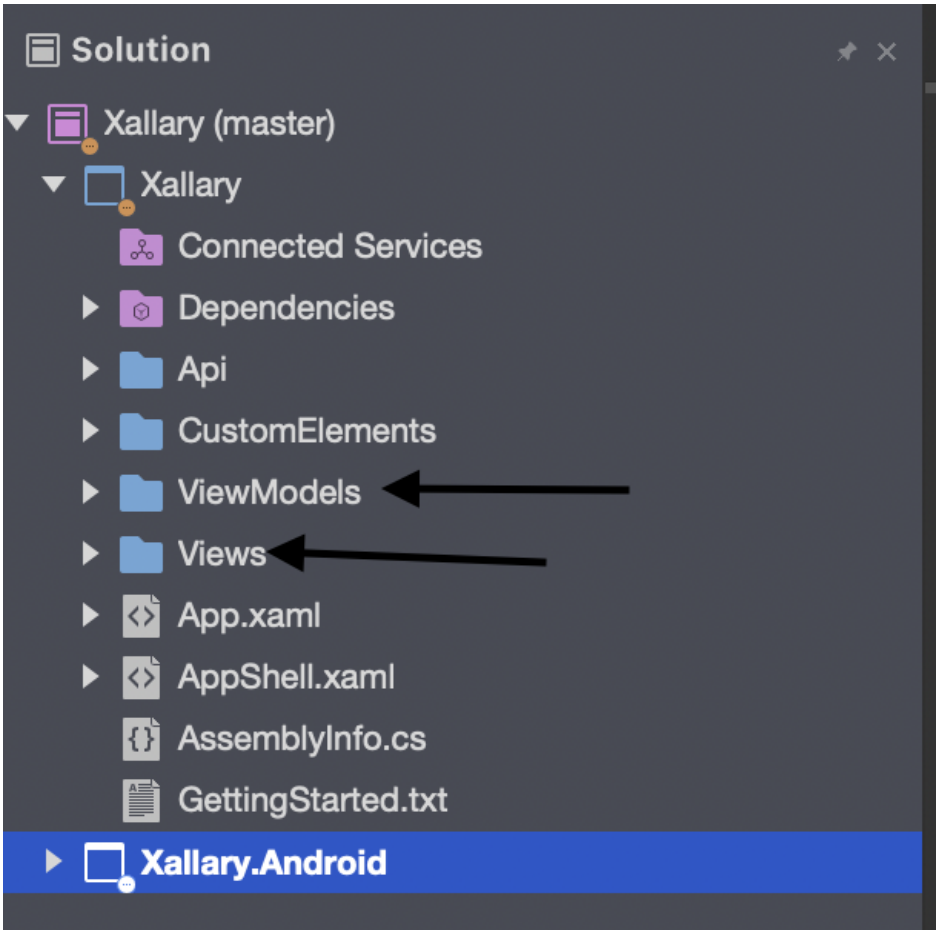


Abbildung 5: Orderstruktur des MVVM-Pattern ohne Model

Außerdem wird das *Command Pattern* verwendet, welches dafür sorgt, dass die *UI* keinen Code in der *Code Behind* besitzt, welcher Klickmethoden auslöst.(Figure 6)

```
<ContentPage.Content>
    <StackLayout>
        <Button Text="Take a Photo" Command="{Binding CapturePhotoCommand}"/>
        <Button  Text="Show a Preview" Command="{Binding PickPhotoCommand}"/>
        <Image VerticalOptions="FillAndExpand" Source="{Binding PhotoPath}" IsVisible="{Binding ShowPhoto}" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>
```

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

2.2 Permissionverwaltung

Wie auch in anderen Sprachen muss eine Berechtigung, also ab hier Persmission, eingeholt werden, damit diverse Ressourcen der Hardware verwendet werden können. Hier werden folgende Persmission (Figure 7) benötigt und verwendet:

- CAMERA
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE

CAMERA Diese Persmission 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 generierende Gallery zu sein. Ohne diese Persmission ist der App untersagt auf die Kamera zuzugreifen

READ_EXTERNAL_STORAGE Da es sich hierbei um eine Gallerie handelt, muss auch eine Persmission gefordert werden, um auf den Internenspeicher 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 Persmission eingeholt werden, die das schreiben auf den Speicher des Gerätes erlaubt, da ansonsten keine Bilder wirklich gespeichert werden können und dann nur im *Cache* des Gerätes zwischen gelagert werden.

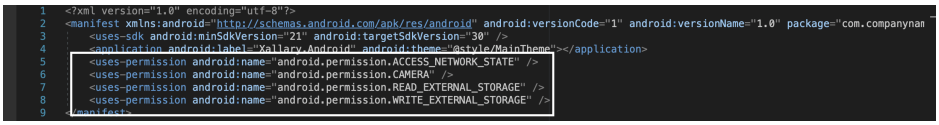


Abbildung 7: Gesetzte Permissions in der AndroidManifest.xml

2.2.1 Benutzer Erfragung für die Permissions

Damit der Nutzer einer App weiß, welche Freigaben er tätigen muss, wird in der Appentwicklung der Nutzer gefragt, ob dieser einverstanden ist, dass

eine App auf beispielsweise die Kamera zugreifen darf. Verneint dieser diese Anfrage wird nicht auf die Kamera zugegriffen bis die Permission vom Benutzer erteilt wurde. Bei *Android* und auch in *Xamarin* muss das Erfragen in der *MainActivity* geschehen. (Figure 8)

```
[Activity(Label = "Xallary", Icon = "@mipmap/icon", Theme = "@style/MainTheme", MainLauncher = true, Confi
public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
{
    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);

        Xamarin.Essentials.Platform.Init(this, savedInstanceState);
        global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
        LoadApplication(new App());
    }
    public override void OnRequestPermissionsResult(int requestCode, string[] permissions,
        [GeneratedEnum] Android.Content.PM.Permission[] grantResults)
    {
        Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode, permissions, grantResults);

        base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

Abbildung 8: RequestPermissionMethod in der MainActivity

2.3 Customelemente

Ein *Customelement* ist wie der Name es sagt, ein individuelles erstelltes Element, welches aus verschiedenen Elementen, wie Button, Grids, Sliders etc. bestehen kann. Für eine App ist es wichtig, dass solche Elemente erstellt werden können, da unterschiedliche Anforderungen nicht immer mit den bereits bereitgestellten Elementen realisiert werden kann. Außerdem können *Custom Elemente* wiederverwendet werden, was einem *Boiler-Code* im UI-Code erspart. Auch brauchen Entwickler nicht 100 mal die selbe Lösung via *Copy and Paste* einfügen und die Wartbarkeit ist hiermit auch eher gewährleistet, als an 100 Stellen den Code zu modifizieren.

Customelemente sind nicht nur ein spezielles Element in *Xamarin* sondern finden auch ihre Verwendung in anderen Sprachen oder Frameworks von C# , wie *WPF* oder *UWP*.

2.3.1 Implementation eines Customelementes

C# In C# werden die Customelemente in einer "nomalen" Klasse geschrieben, diese wiederum von der *View-Klasse* erbt, da die *View* die nötigen *Properties* bestitzt. Außerdem kann dort „leichte" Logik implementiert werden,

die eine View benötigt. Um der selbst erstellten *View* eigene, individuelle *Properties* zu geben, müssen *Bindable Properties* implementiert werden. Diese brauchen bestimmte Parameter. Mehr dazu ist in der Dokumentation zu finden und bei der Abbildung 9.

```
namespace Xallary.CustomElements
{
    /// <summary>
    /// Custom camera preview.
    /// </summary>
    public class CameraPreview : View
    {
        public static readonly BindableProperty CameraProperty = BindableProperty.Create(
        propertyName: "Camera",
        returnType: typeof(CameraOptions),
        declaringType: typeof(CameraPreview),
        defaultValue: CameraOptions.Rear);

        public CameraOptions Camera
        {
            get { return (CameraOptions)GetValue(CameraProperty); }
            set { SetValue(CameraProperty, value); }
        }
    }
}
```

Abbildung 9: Customelement Klasse

Ist das Element korrekt implementiert, kann dieses in jeder *Page/View* verwendet werden, indem das Element in die *View* importiert wird und dann wie jedes „normale“ Element verwendet werden. (Figure 10)

```
    <ContentPage.Content>
        <StackLayout>

            <customElements:CameraPreview   Camera="Rear"
                                            HorizontalOptions="FillAndExpand"
                                            VerticalOptions="FillAndExpand"/>
            <!--<Button Text="Take Photo" Padding="5" Margin="5" HorizontalOptions="Fill"/>-->

        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Abbildung 10: Verwenden eines Customelementes in einer Page/View

Android Die Erstellung eines Customelements exklusiv für Android ist ein wenig aufwendiger. In unserer App haben wir, das Tutorial von *Microsoft* verwendet, welches genau unsere Vorstellungen erfüllt hat. Hier wird außerdem noch ein *Custom Renderer* implementiert, der den C# Code und den erstellten Android Code zusammen „merged“ und dann die Android Rendermaschine verwendet, um das Element korrekt anzuzeigen.

2.4 App im Emulator/Smartphone

Die fertige App sieht wie folgt aus:

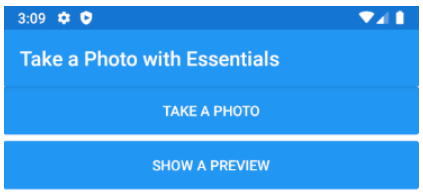
2.4.1 Fullscreen Customcamera-Element

Dieses Element ist das oben beschriebene *Customelement*. Im Bild ist eine Kamera zu sehen, die von dem Emulator erstellt wurde.

Abbildung 11: Customelement, welches die Kamera des Gerätes verwendet

2.4.2 Crossplattform-Implementationen einer funktionierenden Kamera

Ansicht ohne ausgewähltes oder gemachtes Foto Sollte noch kein Foto gemacht worden sein oder ein Foto über das *Filesystem* ausgewählt, dann sieht das Element leer aus. Beide Buttons sind im dazugehörigen *Viewmodel* implementiert. Die Funktion wurde aus dem Beispiel von Xamarin entnommen und für unsere Zwecke angepasst.



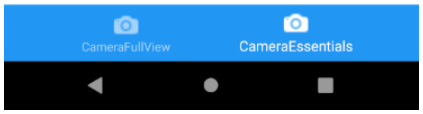


Abbildung 12: Seite mit zwei Buttons und einem Imageview Element, welches noch nicht sichtbar ist

Ansicht mit einem ausgewählten oder gemachten Foto Sollte ein Bild mit der Kamerafunktion gemacht oder mit dem Auswahlbutton ausgewählt worden sein, wird unter den Buttons dieses Bild erscheinen, da es sich bei dem gewählten *Layout* um ein *Stacklayout* handelt und dieses ihre Kindelemente übereinander positioniert.

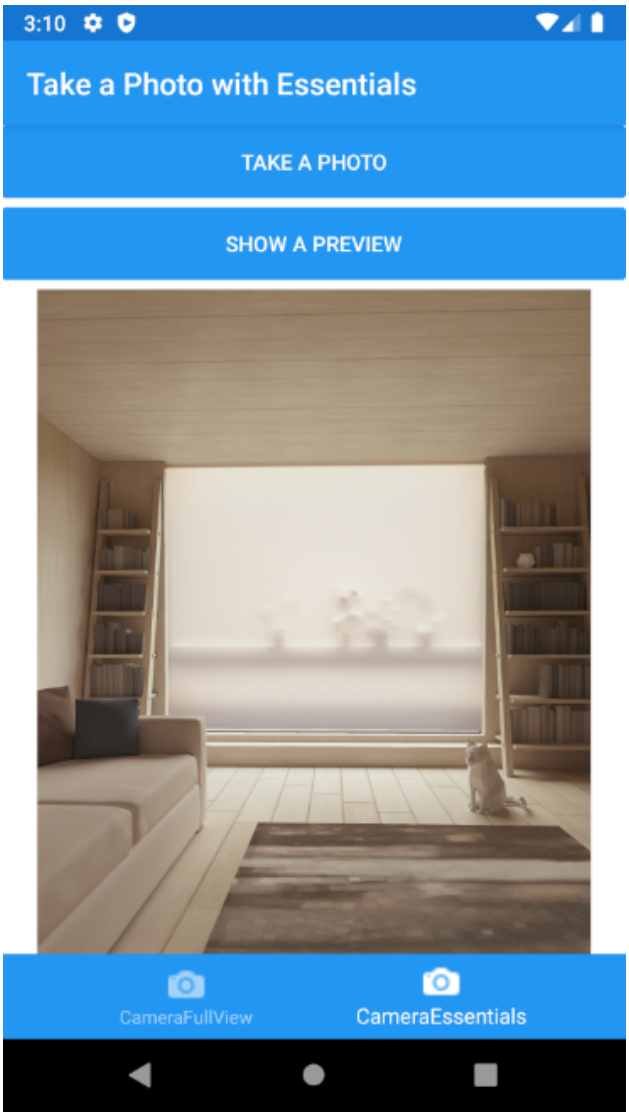


Abbildung 13: Anicht mit Foto

Foto schießen Sobald der Button zum Fotomachen gedrückt wurde, öffnet sich das Betriebssystemkamera-UI, welches jenach Betriebssystem unterscheidlich aussehen kann. Hier kann dann auf das Fotokamerasymbol gedrückt werden und das Foto wird aufgenommen (Figure 14). Nachdem dies gemacht wurde, kommt ein weiterer Dialog, wo der Benutzter entscheiden kann, ob das Bild genommen werden soll und somit gespeichert wird oder, ob es verworfen werden soll. (Figure 14)

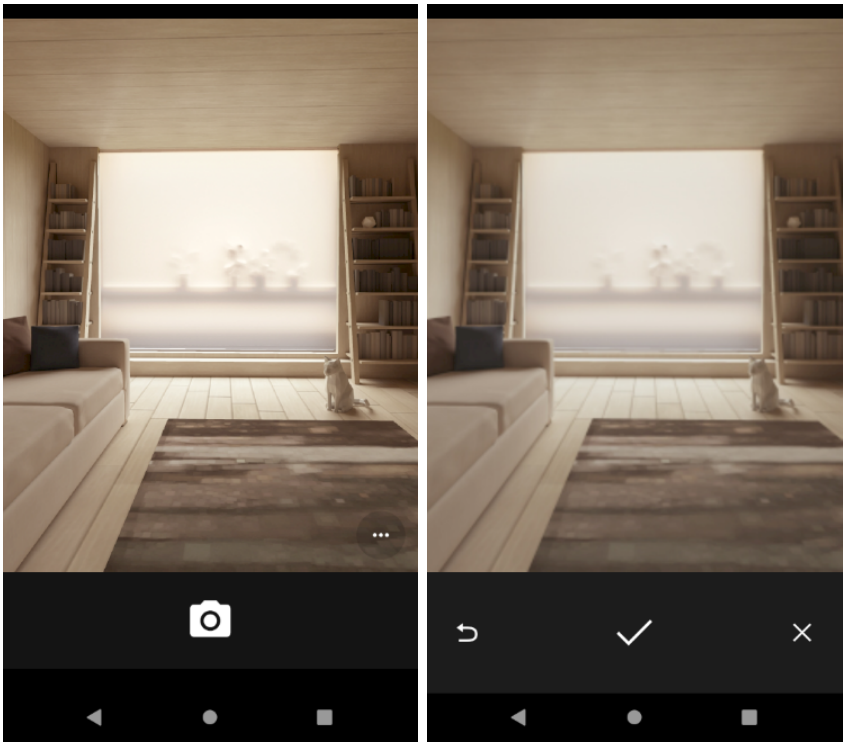


Abbildung 14: Foto schießen und auswählen

Fotos aus dem *Filesystem* auswählen Um die bereits gemachten Fotos anzuzeigen muss dann auf den anderen Button gedrückt werden. Dieser öffnet daraufhin das Filesystem des Betriebssystems. Dort sind dann die Fotos zu sehen, die mit dieser App gemacht wurden. Es ist auch möglich Bilder aus der Gallery hier anzuzeigen. (Figure 15)

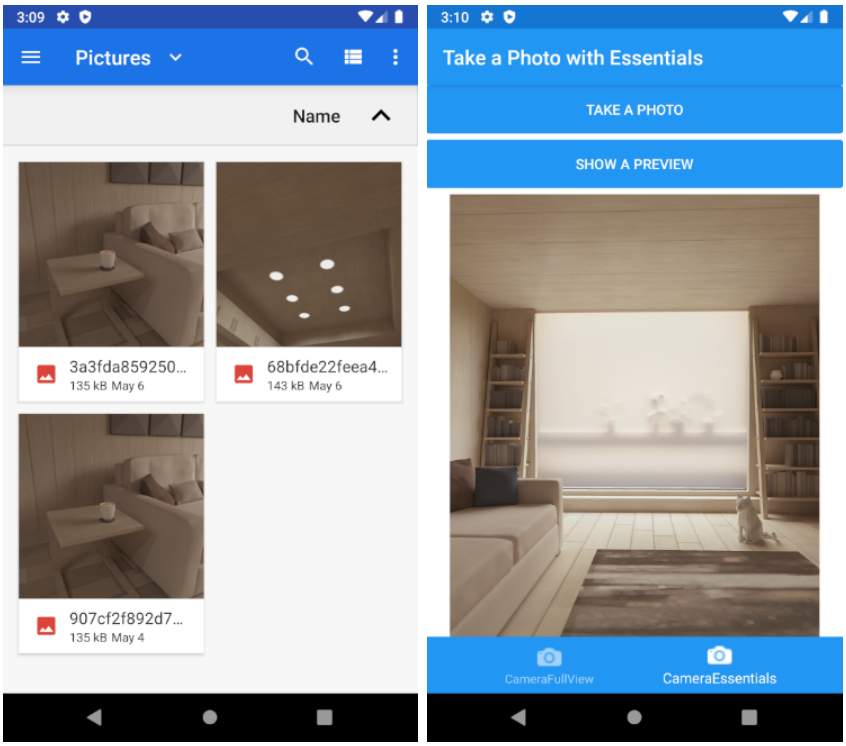


Abbildung 15: Auswählen und anzeigen eines Fotos

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 Platform 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, solang 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 Mobilen 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