

SEMINARARBEIT

im Studiengang Telekommunikation- und Internettechnologien
Lehrveranstaltung Projektarbeit

Xamarin Framework und Microsoft Azure – Photo Sharing App

Ausgeführt von: Christian Pipp

Begutachter: Dipl. Ing. Dr. Thomas Polzer

Wien, 01.02.2017



Inhaltsverzeichnis

Einführung.....	3
1 Xamarin.Forms	3
1.1 Architektur von Xamarin.....	4
1.2 Aufbau eines Xamarin.Forms Projekts.....	5
1.3 Designelementegruppen	7
2 Aufbau der der Photos Applikation	8
2.1 Modelklassen	8
2.2 Views.....	8
2.2.1 CarouselView	8
3 Azure 10	
3.1 Architektur von Azure.....	11
3.2 App Services anlegen	12
3.3 Erstellen von Datenbanktabellen	15
Literaturverzeichnis.....	17
Abbildungsverzeichnis	18
Tabellenverzeichnis	19

Einführung

Xamarin bietet die Möglichkeit mit Xamarin.Forms Plattformübergreifend zu entwickeln. Dabei werden die bekanntesten Plattformen wie Android, iOS oder WindowsPhone unterstützt. Dieses Framework wird von Microsoft unterstützt und es gibt dazu zahlreiche Dokumentation welche frei von Microsoft zur Verfügung gestellt wird. Zu Xamarin gehören auch die zahlreichen Möglichkeiten der verfügbaren Schnittstellen, die man benutzen kann. Eine davon ist die Unterstützung von Azure. Damit wird es ermöglicht Daten der App persistent in Azure zu speichern.

In diesem Projekt war es Ziel einen Photosharing App zu entwickeln die es ermöglicht über verschiedene Accounts eigene Fotos zu verwalten bzw. diese auch für andere User freizuschalten. Eine zusätzliche Kommentarfunktion solle eine soziale Komponente in das Projekt einführen. Zusätzlich war es wichtig, dass diese App zumindest über Android und iOS laufen sollte.

1 Xamarin.Forms

Für unsere Applikation wollen wir Xamarin.Forms verwenden um mit einer Codebasis die App auf allen Plattformen laufen zu lassen. Für das Projekt wurde „*Visual Studio for Mac*“ verwendet damit wird es ermöglicht Apps für iOS und Android zu entwickeln. Da für die Entwicklung der unterschiedlichen OS das darunterliegende betriebssystemspezifische Framework vorhanden sein muss, gibt es keine Windows Version dieser App. So muss zur Entwicklung von iOS zum Beispiel XCode installiert werden um dieses in Xamarin.Forms zu verwenden. Um wirklich für alle drei Plattformen entwickeln zu können braucht man daher zwei Rechner (bzw. eine VM).

Der Vorteil von Xamarin.Forms liegt auf der Hand: Es ermöglicht das Entwerfen einer GUI, die dann wieder von allen OS verwendet werden kann. Die Programmierung dieser GUI kann über C# erfolgen oder aber auch über die eigene Beschreibungssprache XAML die an XML angelehnt ist und dem Beschreibungscode in der Android Entwicklung ähnelt. Leider gibt es keinen XAML Designer in der derzeitigen Visual Studio Version und somit das Design etwas erschwert.

Zusätzlich kann nativer Code implementiert werden um damit spezifische Funktionalität der Zielplattform implementiert werden. Das wird zum Beispiel benötigt um die jeweiligen Fotobibliotheken aufzurufen. Zusätzlich dazu besteht ein Kommunikationskanal zurück zum Core Xamarin.Forms um Werte von den nativen Plattformen weiterverwenden zu können. Im Fall der Fotobibliothek wäre das ein binärer Stream mit dem man auf Xamarin.Forms Seite einen ImageSource initialisieren kann.

1.1 Architektur von Xamarin

Die prinzipielle Architektur von Xamarin.Forms baut darauf auf, dass sich Xamarin.Forms on top der Applikation setzt und als Compiler für diese Betriebssysteme fungiert. Das heißt, es wird tatsächlicher ausführbarer Code für die jeweilige Applikation generiert und Xamarin.Forms agiert nicht als Interpreter. Am besten veranschaulicht untenstehende Abbildung den Aufbau. Xamarin.Forms wird immer dann verwendet, wenn man möglichst wenig nativen Code zu programmieren, da hier deutlich die GUI-Entwicklung vereinfacht.

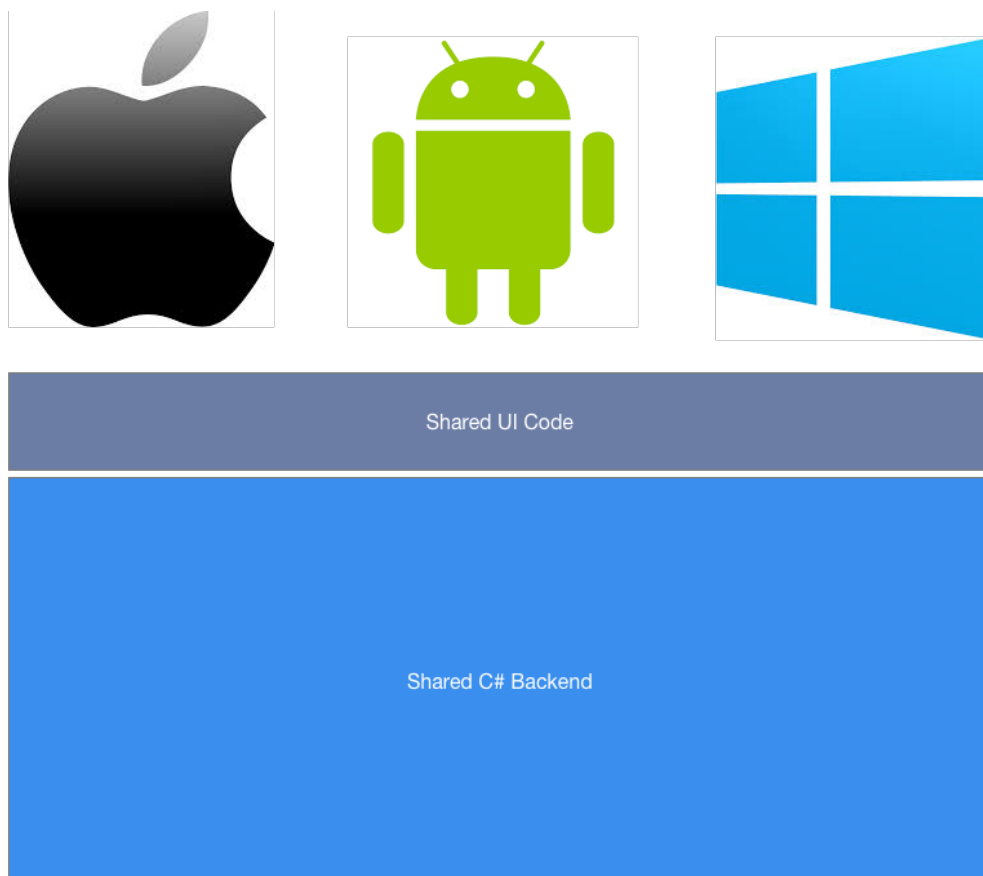


Abbildung 1: Architektur Xamarin.Forms

In früheren Versionen von Xamarin war es noch nicht möglich, einen Shared Code für die UI zu nutzen; mittlerweile ist das über Xamarin.Forms kein Problem. Nach wie vor kann man natürlich für jede Plattform seinen eigenen UI Code entwickeln. Die Xamarin.Forms-Anwendung ist wie jede plattformübergreifende Anwendung aufgebaut, dabei wird der Code in eine portable Klassenbibliothek (Portable Class Libraries) platziert und die jeweiligen nativen Anwendungen nutzen diese dann. Untenstehende Abbildung sollte dieses Prinzip verdeutlichen [1].

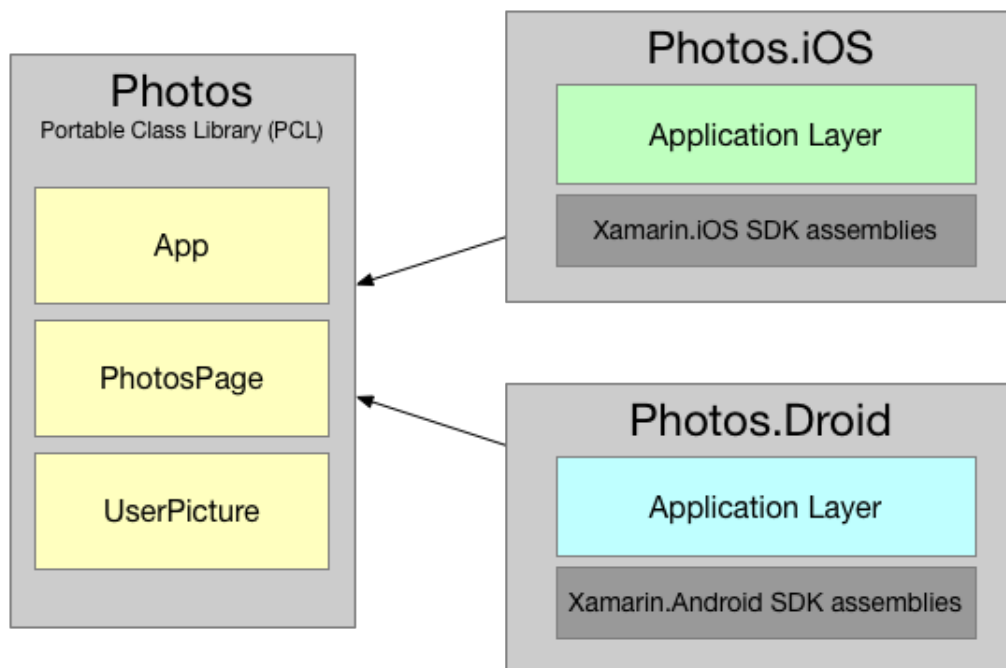


Abbildung 2: Beziehung der Portable Class Libraries

Um das Umzusetzen gibt es in einem Xamarin.Forms drei verschiedene Hauptbereiche die relevant für die Entwicklung sind.

1.2 Aufbau eines Xamarin.Forms Projekts

Wenn man sich die Photos Applikation ansieht gibt es drei Bereiche. Diese Bereiche sind immer zwingend in einem Xamarin.Forms Projekt enthalten.

- **Photos:** Dieser Teil enthält den Code der für die PCL benötigt wird
- **Photos.Droid:** Hier befindet sich spezifischer Android Code und bildet auch den Einstiegscode für Android Anwendungen
- **Photos.iOS:** Hier befindet sich spezifischer iOS Code und bildet auch den Einstiegscode für iOS Anwendungen

Unter diesen einzelnen Teilen befindet sich Unterverzeichnisse die unter anderen Referenzen die zum Ausführen der Anwendung erforderlich sind aber auch Pakete, sogenannten NuGet Pakete, die es ermöglichen Bibliotheken von Drittanbietern in das Projekt einzubinden.

Beim Exekutieren einer Applikation wird zum Beispiel bei iOS die *Main.cs* Datei aufgerufen, diese wiederum ruft nur die *AppDelegate.cs* Datei auf, die den Code auf dem iOS generiert. *AppDelegate* Klasse wird jeder nativer iOS Entwickler kennen, bei Android heisst die *Main.cs*

dann *MainActivity.cs*, die den nativen Android Code enthält. Untenstehen Abbildung verdeutlicht diese Beziehungen untereinander.

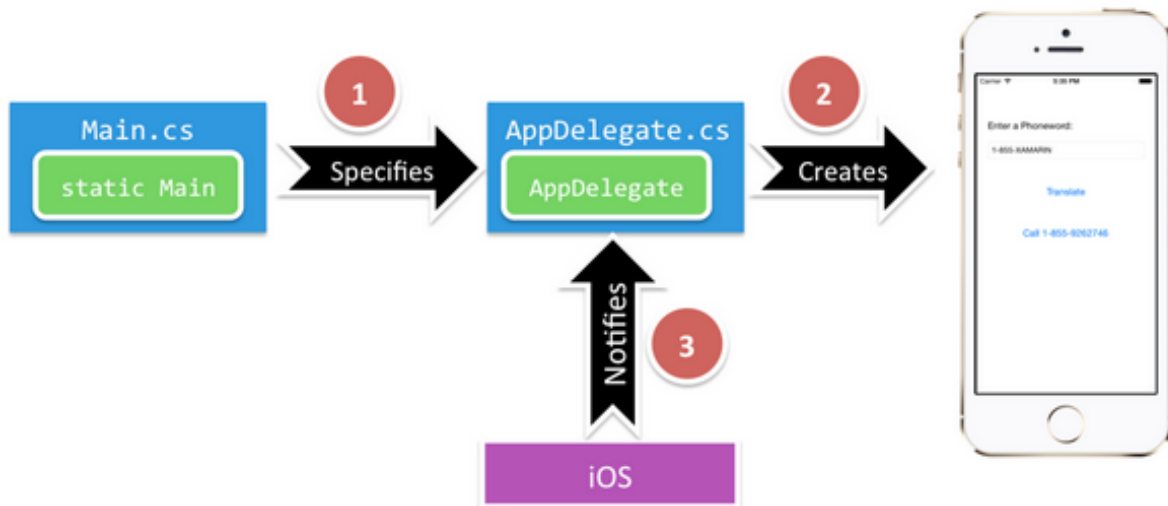


Abbildung 3: Beziehung Main.cs und AppDelegate.cs (Quelle: [1])

Bei der oberen Abbildung gibt es auch den Schritt 3 der eine Kommunikation von iOS zu Xamarin ermöglicht. Diese wird im konkreten Fall auch benötigt, da wir die Fotos die in der iOS Photo Library ausgewählt werden im Shared Code Teil der Applikation verwenden wollen.

Die simple *Main.cs* schaut folgendermaßen aus:

```
namespace Photos.iOS
{
    public class Application
    {
        // This is the main entry point of the application.
        static void Main(string[] args)
        {
            // if you want to use a different Application Delegate class from "AppDelegate"
            // you can specify it here.
            UIApplication.Main(args, null, "AppDelegate");
        }
    }
}
```

Diese Klasse macht nun wiederum nichts anderes und ruft die AppDelegate Klasse auf:

```
namespace Photos.iOS
{
    [Register("AppDelegate")]
    public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplication
    Delegate
    {
        public override bool FinishedLaunching(UIApplication app, NSDictionary options)
        {
            global::Xamarin.Forms.Forms.Init();
            Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init();
            LoadApplication(new App());
            return base.FinishedLaunching(app, options);
        }
    }
}
```

Damit werden alle notwendigen Schritte unternommen um die Anwendung auf der iOS Plattform zu starten. Ein spezifischer Aufruf wurde bei diesen Projekt hinzugefügt und ist oben im Code Abschnitt rot markiert. Dieser ist notwendig um die Dienste für Azure zu initialisieren. Allerdings ist das nur im iOS Code notwendig und Android läuft ohne Anpassungen mit Azure.

Es gibt eine Klasse die sich im Shared Code des Projekts befindet die das Ganze auslöst und zwar die App.xaml.cs. Diese definiert auch welche Page beim Applikation Start aufgerufen wird. Wie bei vielen Klassen in Xamarin.Forms sind die meisten (außer die reinen Model oder Service Klassen) eine Kombination aus einem C# und XAML File. Dabei definiert das XAML File das Design der jeweiligen Page, diese Elemente könnten dann über das C# File angesprochen werden, bzw. es werden Events der einzelnen GUI Elemente behandelt. Prinzipiell ist es auch möglich die gesamte GUI über das C# festzulegen. Dieser Aspekt führt uns auch zu den verschiedenen Designelementen von Xamarin.Forms.

1.3 Designelementegruppen

In Xamarin.Forms gibt es prinzipiell folgende Arten von Designgruppen.

- **Pages:** Damit werden verschiedenen
- **Layout:** Damit wird die Positionierung der einzelnen Elemente festgelegt und es gibt verschiedene Typen dazu wie StackLayout, GridLayout usw.

- **Views:**

Zusätzlich gibt es jede Menge Elemente, die man standardmässig von iOS oder Android kennt, wie ListView, Buttons, Labels, TextViews,

@TODO: tbc....(noch genauer beschreiben)

2 Aufbau der der Photos Applikation

Nach der kurzen Einführung in Xamarin.Forms und deren wichtigsten Komponenten werden hier die Elemente des Projekts beschrieben. Dabei wurde bewusst der Azure Teil rausgenommen um nur die Xamarin.Forms relevanten Teile zu beschreiben, da Azure in einem extra Kapitel beschrieben wird.

2.1 Modelklassen

Um die Daten in der Applikation abzubilden wurde verschieden Klassen angelegt und diese mit entsprechenden Getter- und Setter Methoden ausgestattet. Vorrauschauend auf die Verwendung von Azure wurde auch immer eine eigene ID mit aufgenommen, damit die Datenbankeinträge einzigartig (unique) sind.

- UserPicture : enthält alle Bilder des Users
- PictureComment: enthält alle Kommentare zum jeweiligen Bild

Diese beiden Klassen werden dazu verwendet um die entsprechenden *ListView* in der Applikation durch sogenanntes Data Binding mit entsprechenden Daten zu versorgen.

@TODO: tbc

2.2 Views

Es gibt zwei Views in der Applikation, einer der die Fotos der anzeigen solle und ein andere mit des es möglich ist Kommentare zu den jeweiligen Fotos abzugeben. Zum Anzeigen der Fotos wurde ein CarouselView gewählt den man mit NuGet in seine Applikation einbinden muss um in auch verwenden zu können dazu sind folgende Schritte notwendig.

2.2.1 CarouselView

Der CarouselView wurde schon in Xamarin.Forms Version 2.2.0 fix aufgenommen, aber auf Grund von diversen Problemen in der Version 2.3.0 wieder herausgenommen, deswegen muss man sich diesen View über NuGet in die projekteigenen Packages installiert werden [2]. Wichtig dabei ist das man in den XAML Dateien den entsprechenden Namespace

inkludiert, aber auch in den jeweiligen Start Methoden der nativen Plattformen den View initialisiert. Im konkreten Fall musste zum Beispielt im iOS Projekt folgende Zeile in die AppDelegate hinzugefügt werden:

```
global::CarouselView.FormsPlugin.iOS.CarouselViewRenderer.Init();
```

Damit wurde sichergestellt, dass auch alle CarouselView Methoden innerhalb der C# Klassen verwendet werden konnten. Um den CarouselView in einem XAML File zu verwenden war es notwendig den korrekten Namespace in ContenPage Tag einzupflegen

```
xmlns:cv="clr-  
namespace:CarouselView.FormsPlugin.Abstractions;assembly=CarouselView.FormsPlugin.A  
bstractions"
```

Dadurch wurde es ermöglicht mit den CarouselView Bilder anzuzeigen.

```
<ContentView Grid.Row="1" Grid.Column="0">  
  <cv:CarouselViewControl AnimateTransition="true" ShowArrows="true" ShowIndicators=  
"true" Orientation="Horizontal" x:Name="CarouselPics" >  
    <cv:CarouselViewControl.ItemTemplate>  
      <DataTemplate>  
        <Image Source="{Binding Picture}" />  
      </DataTemplate>  
    </cv:CarouselViewControl.ItemTemplate>  
  </cv:CarouselViewControl>  
</ContentView>
```

Zusätzlich musste im C# Code noch mit ItemSource die korrekte Modelklasse hinzugefügt werden und der CarouselView wurde nach jedem Update der Daten automatisch gerendert.

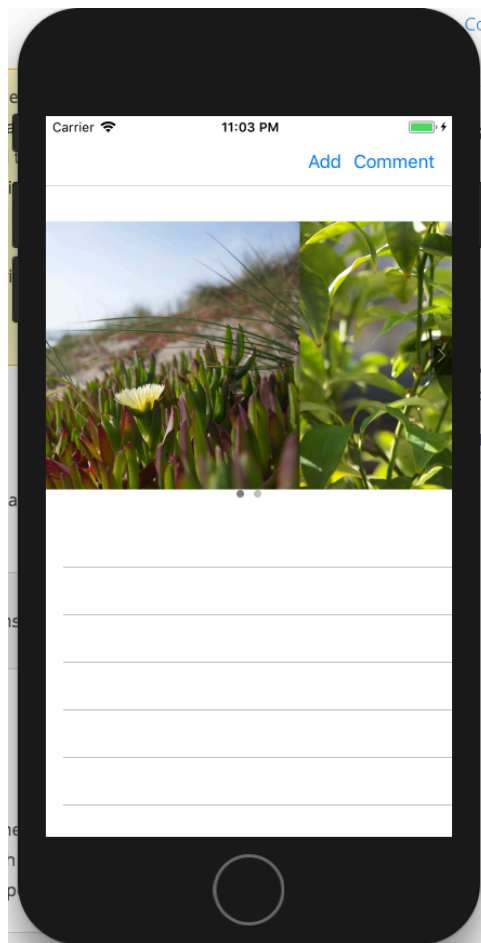


Abbildung 4: CarouselView

Im unteren Teil der Applikation sollten die Kommentare zu den jeweiligen Bildern angezeigt werden.

3 Azure

Zum persistenten Speichern der Daten wurde Azure gewählt und dazu ein Studenten Account über Imagine Microsoft erstellt um dort entsprechende SQL Datenbanken anzulegen und ein sogenanntes App Service zu hosten. Damit konnte man die Datenbankzugriffe direkt in das Xamarin.Forms Projekt einbinden. Diese erfolgte ein paar Schritten direkt im Portal von Azure um alle notwendigen serverseitigen Einstellungen vorzunehmen wichtig dabei war das sich alle Applikationen in der selben Ressourcengruppe befinden müssen. Folgender Abschnitt gibt einen Überblick über die Architektur von Azure und zeigt die notwendigen Schritte um Azure in Xamarin.Forms einzubinden.

3.1 Architektur von Azure

Azure bietet ein mobiles Backend über das die verschiedenen nativen Clients zugreifen können. Dazu kann Xamarin benutzt werden oder man kann auch native Client SDKs verwenden. Wenn kurzfristig keine Internetverbindung besteht kann man die Daten per Offlinesynchronisierung behalten. Als Sicherheitsfeature wird eine Authentifizierung angeboten mit der man die unterschiedlichen Clients identifizieren kann.

Es gibt zwei Speichermöglichkeiten einerseits über den Blob Storage aber auch über eine MSSQL Datenbank [3]. Untenstehende Abbildung gibt eine gute Übersicht über die verfügbaren Komponenten.

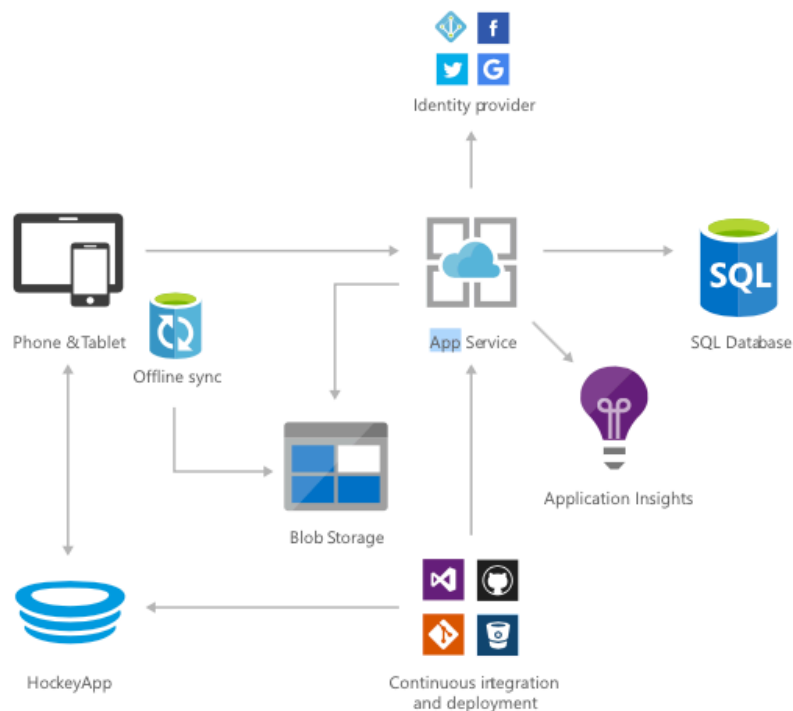


Abbildung 5: Architektur Azure

Im Mittelpunkt steht dabei das App Service, dass alle Komponenten miteinander verbindet und stellt damit die Verbindung zwischen Clients und den Backend, dabei gibt es zwei mögliche Ansätze die verwendet werden können:

- Backend mit Nodes.js
- Backend mit C#

Die Application Insights dienen als Diagnosetool um eventuelle Abstürze oder Probleme zu analysieren. Zusätzlich wird auch eine App (Hockey) zur Verfügung gestellt mit der man die App-Benutzung nachverfolgen kann.

3.2 App Services anlegen

Im ersten Schritt wird ein App Service angelegt um damit die Verbindung von der Clientsoftware – die Xamarin.Forms App – zum Server abzubilden. Dies funktioniert relativ einfach und man muss sich dazu mit einem gültigen Microsoft Account in das Azure Portal einloggen. Dort wählt man den im linken Menü den Punkt *App Services* aus wie in der unteren Abbildung gezeigt.

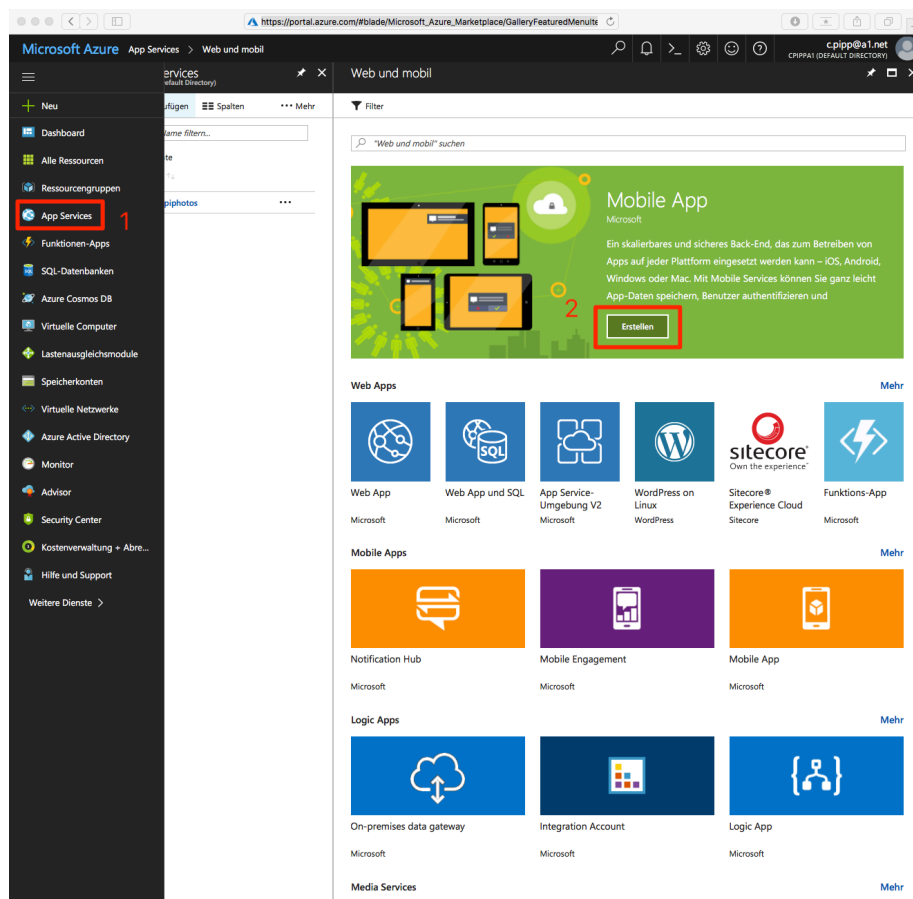


Abbildung 6: Einrichten eines App Services

Im nächsten Schritt wird man gefragt nach dem Namen des App Services gefragt und in welcher Ressourcengruppe das Service angelegt werden soll. Dabei ist es wichtig dass diese Ressourcengruppe gleich mit der Gruppe des Windowsservers sein muss sonst kommt

es später zu Problemen, da sich Elemente unterschiedlicher Gruppen nicht austauschen können.

Abbildung 7: Benennen des App Services

Gleich anschließend wird das gewünschte Service erstellt, dies kann auch einige Minuten dauern. Nach Abschluss sollte man das Service in der Übersicht wieder finden:

Hinzufügen

Spalten

Aktualisieren

Tags zuweisen

Start

Neu starten

Beenden

Löschen

Abonnements: Microsoft Imagine

Nach Name filtern...

Alle Ressourcengruppen

Alle Standorte

Keine Gruppierung

2 Elemente



<input type="checkbox"/>	NAME	STATUS	APP-TYP	APP SERVICE-PL...	STANDORT	ABONNEMENT
<input type="checkbox"/>	 cpiphotos	Running	Mobile App	ServicePlan655d7642...	USA Mitte	Microsoft Imagine
<input type="checkbox"/>	 test500	Running	Mobile App	ServicePlana6aa04a8...	USA Mitte	Microsoft Imagin...

Abbildung 8: Übersicht App Services

Danach geht es weiter mit der Anlage einer MSSQL Datenbank. Dazu muss ein Microsoft Server zur Verfügung stehen auf welchen die Datenbank laufen soll. Zur Anlage dieser Datenbank verwendet man am besten den Punkt Schnellstart im App Service Menü, der unter dem Punkt Bereitstellung zu finden ist. Da wir unser Applikation für iOS und Android lauffähig machen wollen wählen wir auch dementsprechend Xamarin.Forms aus

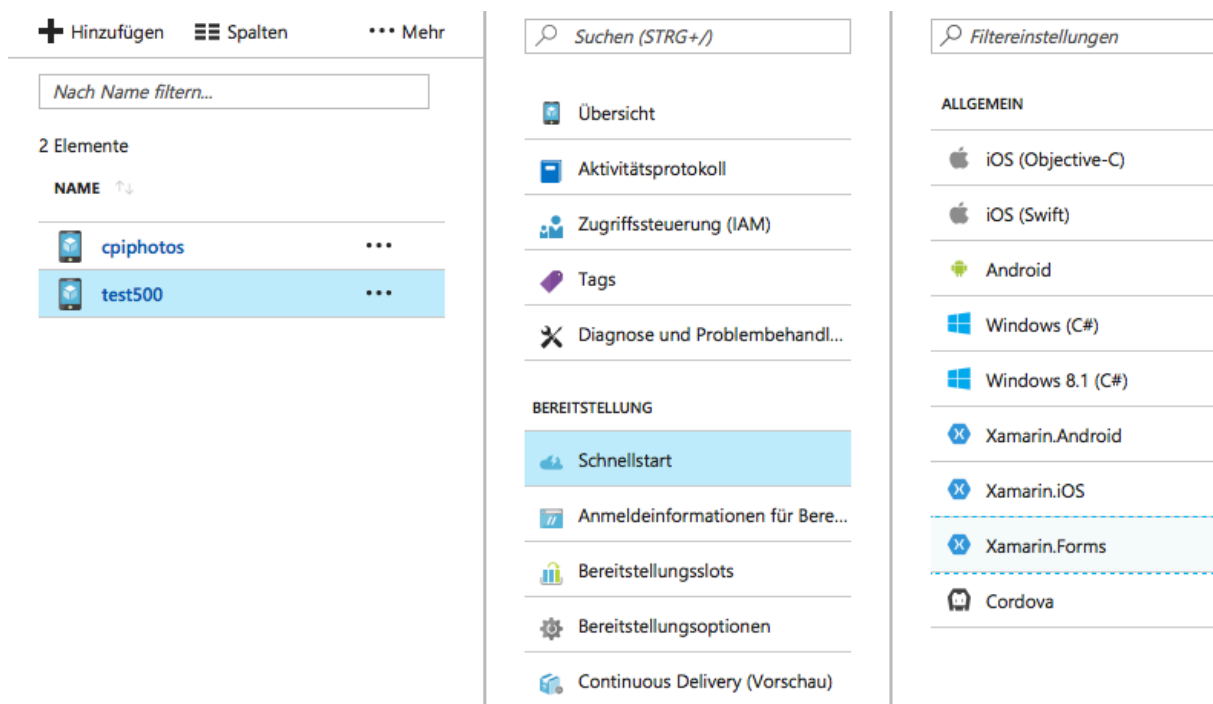


Abbildung 9: Einrichten Datenbank Azure


Anschließend kann man die Datenbank festlegen, welche Backendsprache man verwenden will und sich damit ein Beispielprojekt generieren lassen (an Hand einer einfachen Todo Liste). Man kann dann allerdings auch noch später die gewünschten Datenbanktabellen anlegen. Dies macht man zu Beginn am besten über das Azure Portal, da auch gleichzeitig alle notwendigen serverseitigen Skripts angelegt werden. Dies ist notwendig da in der Applikation nicht direkt mit dem MSSQL Server kommuniziert wird sondern das Backend des Server aufgerufen wird und alle Abfragen handelt. Als Backend Sprache wird *Node.js* benutzt.

Xamarin.Forms

Schnellstart

1


Datenbank verbinden



Sie benötigen eine Datenbank, um diesen Schnellstart durchzuführen. Klicken Sie hier, um eine Datenbank zu erstellen.

2

Tabellen-API erstellen



Um Daten in Ihrem Back-End zu speichern, benötigen Sie eine Tabelle. Wählen Sie unten eine Back-End-Sprache aus, und erstellen Sie eine TodoItem-Tabellen-API.

Back-End-Sprache:

Node.js

☐ Ich bestätige, dass durch diese Aktion alle Websiteinhalte überschrieben werden.

TodoItem-Tabelle erstellen

Um später zusätzliche Tabellen zu erstellen, navigieren Sie zu den Einstellungen für "Einfache Tabellen".

3

Clientanwendung konfigurieren

NEUE APP ERSTELLEN

VORHANDENE APP VERBINDEN

Auf einem Windows-PC: [Visual Studio Community 2015 installieren](#)

Auf einem Mac oder Windows-PC: [Xamarin für Windows installieren](#)

Laden Sie Ihr personalisiertes Xamarin-Projekt herunter, extrahieren Sie es, und öffnen Sie es in Visual Studio oder Xamarin Studio. Die App ist so vorkonfiguriert, dass sie mit Ihrem gehosteten mobilen Back-End funktioniert.

Herunterladen

Führen Sie das Xamarin-Projekt aus, um mit den Daten in Ihrem mobilen Back-End zu arbeiten.

Abbildung 10: Grundkonfiguration App Services

Nachdem die Datenbank verknüpft ist, ist am Server (bis auf die Datenbank Tabellen) einmal alles soweit eingerichtet. Man könnte sich testweise auch eine TodoItem Tabelle erstellen und dann damit eine Testprojekt generieren. Als nächsten Schritt werden nun Datenbanktabellen, die für die Applikation benötigt wird angelegt.

3.3 Erstellen von Datenbanktabellen

Wir wollen das Anlegen der Datenbanktabellen im Azure Portal machen, da damit auch alle notwendigen Backendskripts generiert werden. Diese kann man im Punkt *Mobil* -> *Einfache Tabellen* anlegen [4]. Je nachdem ob man die TodoItem Tabellen generieren hat lassen oder nicht gibt es dort schon Einträge.

Literaturverzeichnis

- [1] „Ausführliche Erläuterungen zu Xamarin.Forms - Xamarin“. [Online]. Verfügbar unter: <https://developer.xamarin.com/de-de/guides/xamarin-forms/getting-started/hello-xamarin-forms/deepdive/>. [Zugegriffen: 17-Feb-2018].
- [2] M. M. | X. M. | X. C. D. | E. M. F. | X. F. D. | Melbourne und Australia, „Carousel View in Xamarin Forms“, *Xamarin Help*, 18-Apr-2016. [Online]. Verfügbar unter: <https://xamarinhelp.com/carousel-view-xamarin-forms/>. [Zugegriffen: 19-Feb-2018].
- [3] „Aufgabenbasierte mobile App für Kunden“. [Online]. Verfügbar unter: <https://azure.microsoft.com/de-de/solutions/architecture/mobile-app-consumer/>. [Zugegriffen: 21-Feb-2018].
- [4] „Getting Started with Azure Mobile Apps' Easy Tables |“, *Xamarin Blog*, 28-Jan-2016. .
- [5] Martin Sauer, *Grundkurs Mobile Kommunikationssysteme*, 5. Aufl. Wiesbaden: Springer Vieweg, 2013.

Abbildungsverzeichnis

Abbildung 1: Architektur Xamarin.Forms	4
Abbildung 2: Beziehung der Portable Class Libraries	5
Abbildung 3: Beziehung Main.cs und AppDelegate.cs (Quelle: [1])	6
Abbildung 4: CarouselView	10
Abbildung 5: Architektur Azure	11
Abbildung 6: Einrichten eines App Services	12
Abbildung 7: Benennen des App Services	13
Abbildung 8: Übersicht App Services	13
Abbildung 9: Einrichten Datenbank Azure	14
Abbildung 10: Grundkonfiguration App Services	15

Tabellenverzeichnis

No table of figures entries found.

In your document, select the words to include in the table of contents, and then on the Home tab, under Styles, click a heading style. Repeat for each heading that you want to include, and then insert the table of contents in your document. To manually create a table of contents, on the Document Elements tab, under Table of Contents, point to a style and then click the down arrow button. Click one of the styles under Manual Table of Contents, and then type the entries manually.