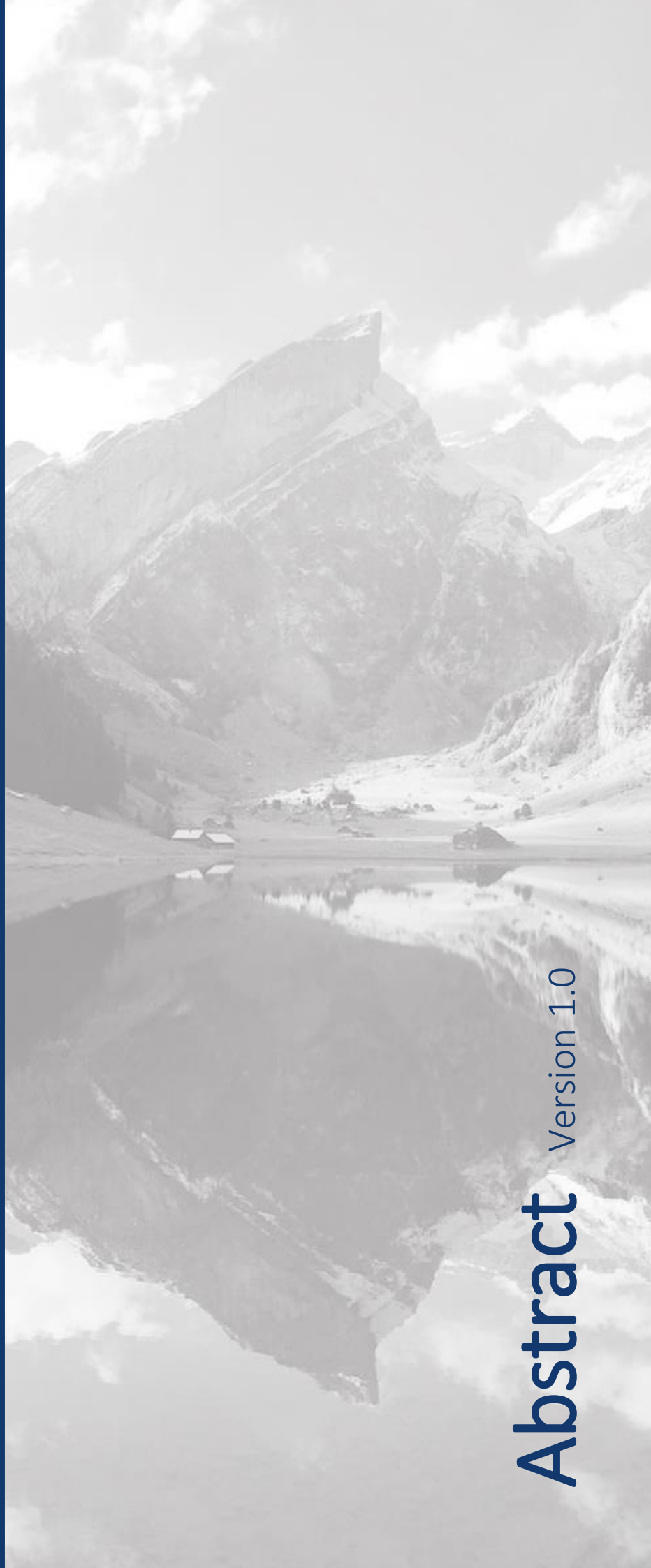
kwix

share your experiences



Abstract

Version 1.0

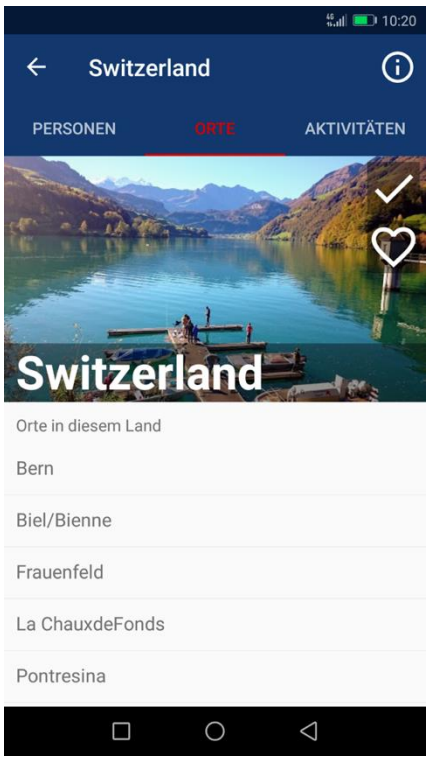
Thomas Egle
Silvano Ferretti
Pascal Kolp

kwix – share your experiences

Jeder hatte bestimmt schon mal das Erlebnis, dass er von den Ferien oder einer Reise zurückkehrte und dann im Gespräch mit Kollegen feststellte, dass der eine oder andere auch bereits dort gewesen ist. Eigentlich hätte man vor der Reise noch nützliche Tipps oder Informationen zum besuchten Ort einholen können, hätte man gewusst, dass der besagte Kollege oder die besagte Kollegin bereits dort war.

Aus einem solchen Erlebnis heraus entstand die Idee zur Masterarbeit *kwix*. Mit der mobilen Applikation *kwix* ist es möglich, all seine besuchten Orte zu erfassen und zu schauen, wo seine Kontakte aus dem Adressbuch bereits waren und diese, falls gewünscht, direkt zu kontaktieren. Auch kann man eine Wunschliste mit all den Orten, welche man noch besuchen möchte, erstellen.

Wir haben uns entschieden, die Applikation mit Xamarin als Cross-Plattform zu entwickeln. Die Umsetzung erfolgte agil nach SCRUM. Die Planung der zweiwöchigen Sprints, das Git Repository und das Zusammentragen von Informationen, haben wir im VSTS (Visual Studio Team Services) gelöst.
Als Grobkonzept für die Applikation haben wir zu Beginn der Masterarbeit ein ausführliches Wireframe erstellt. Daraus haben wir sieben Zwischenziele abgeleitet, die es zu erreichen galt und welche uns auch als Fortschrittskontrolle dienten.
Die Entwicklung der Applikation selbst wurde in C# in Visual Studio realisiert.



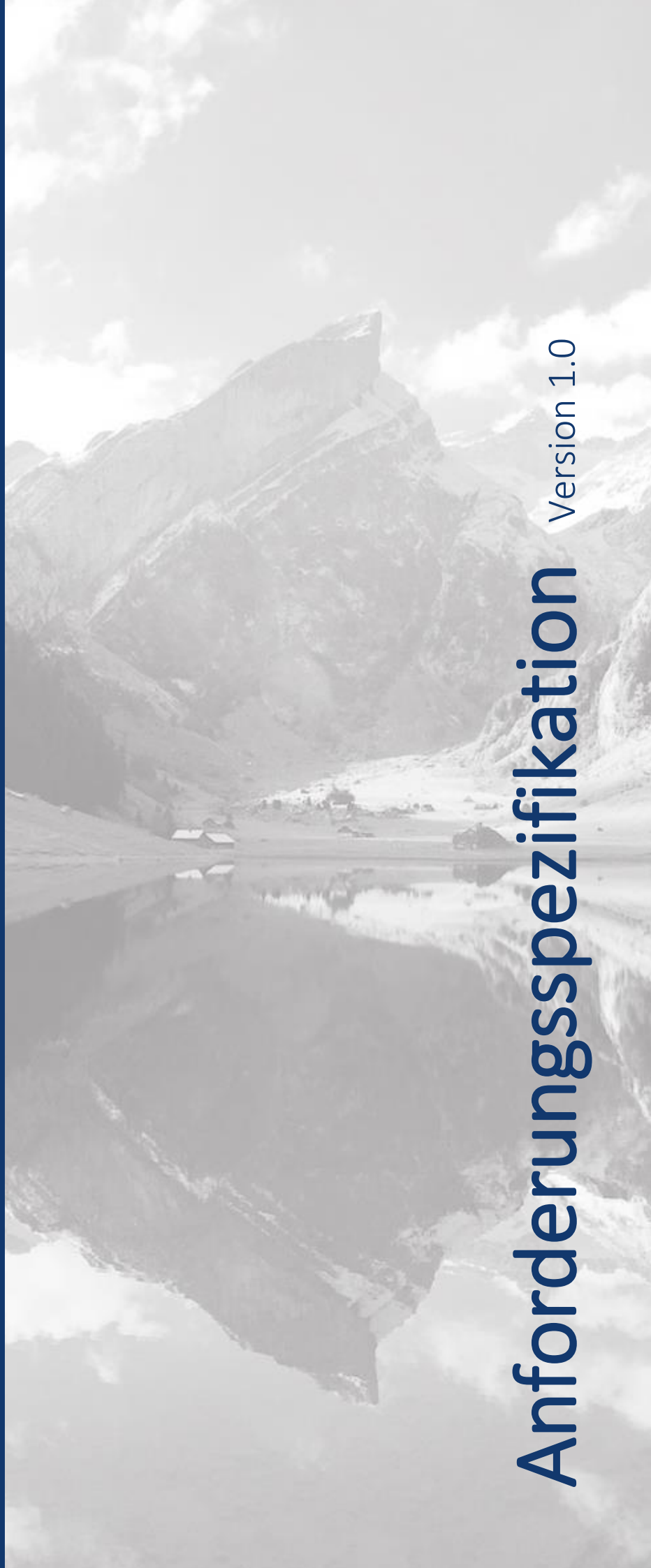
Das Verwalten der Daten erfolgt über eine relationale Datenbank, welche auf Microsoft Azure läuft. Zusätzlich wurden REST Webservices implementiert, die für die Kommunikation mit der Datenbank sowie die Google Places API Abfragen zuständig sind. Die Informationen von Google Places werden für die Ortssuche benötigt.

Das Deployment und die Verteilung der Applikation an Betatester erfolgt über das Microsoft App Center.

Das Endergebnis entspricht unseren Vorstellungen. Alle von uns gesetzten Ziele wurden erreicht und umgesetzt.
Da jede Google Places Suchabfrage kostenpflichtig ist, besteht die grösste Herausforderung darin, *kwix* profitabel betreiben zu können. Deswegen, und weil uns noch zusätzliche Ideen vorschweben, haben wir uns entschieden, das Projekt weiterzuverfolgen.
Es ist geplant, *kwix* im Google Play Store und im Apple AppStore kostenlos zur Verfügung zu stellen.

kwix

share your experiences



Anforderungsspezifikation

Version 1.0

Thomas Egle
Silvano Ferretti
Pascal Kolp

Änderungsgeschichte

Version	Autor	Beschreibung	Datum
0.1	Pascal Kolp	Erstellung	24.04.2018
0.2	Pascal Kolp, Silvano Ferretti, Thomas Egle	Überarbeitung anhand der Review Findings	28.04.2018
0.3	Thomas Egle	Sequenzdiagramm für Autocomplete hinzugefügt	08.05.2018
0.4	Thomas Egle	UseCase überarbeitet Sequenzdiagramme überarbeitet, neue hinzugefügt	05.08.2018
0.5	Pascal Kolp	Beschreibung Autocomplete, Suche eines Place	10.08.2018
0.6	Silvano Ferretti	Systemkontextdiagramm überarbeitet	10.08.2018
0.7	Thomas Egle	Kapitel 6 Sequenzdiagramme überarbeitet	21.08.2018
1.0	Projektteam	Freigabe	26.08.2018

Inhaltsverzeichnis

1 Einführung

1.1 Zweck

Dieses Dokument beschreibt die Funktionsweise anhand von Anwendungsfällen (Use Cases), die Schnittstellen und die nicht-funktionalen Anforderungen des Projekts *kwix - share your experiences*.

1.2 Gültigkeitsbereich

Dieses Dokument ist über die ganze Projektdauer gültig. Änderungen werden fortlaufend ergänzt und in der Änderungsgeschichte festgehalten.

1.3 Referenzen

Nr	Titel	Quelle
[1]	Glossar	…\Dokumente\Glossar_V1.0.pdf
[2]	Android Verbreitung	https://developer.android.com/about/dashboards/
[3]	iOS Verbreitung	https://developer.apple.com/support/app-store/

1.4 Glossar

Alle verwendeten Begriffe aus dem Glossar werden in diesem Dokument *kursiv* geschrieben.

Siehe Glossar [1]

2 Allgemeine Beschreibung

2.1 Produkt Funktion

kwix - share your experiences ist eine Mobile Applikation, welche es ermöglicht besuchte *Places* zu erfassen und zu erfahren, wo die eigenen Kontakte bereits waren.

Das Vorgehen lässt sich am einfachsten anhand eines kleinen Rollenspiels erklären:

- Alice fährt übers Wochenende nach Berlin.
- Sie erfasst Berlin als Ort in der App.
- Bob, ein Freund von Alice, hat einige Zeit später ebenfalls die Idee nach Berlin zu gehen.
- Bob sucht in der App nach dem Ort Berlin.
- Da Alice als Kontakt in seinem Telefonbuch gespeichert ist und auch Alice seine Telefonnummer gespeichert hat, sieht er in den Suchergebnissen, dass Alice bereits schon mal in Berlin war.
- Bob hat nun die Möglichkeit Alice direkt aus der App heraus zu kontaktieren. Dies kann über einen Telefonanruf oder eine Kurznachricht (SMS oder WhatsApp) geschehen.
- So erfährt Bob direkt aus erster Hand, was man in Berlin gesehen haben muss.

Die Idee der App ist es, Informationen über einen *Place* direkt von einer bekannten Person zu erhalten. Bei anonymisierten Kommentaren über einen *Place*, weiss man nicht wie zuverlässig diese sind, da man diese Personen, welche die Kommentare erfasst haben, nicht persönlich kennt.

2.2 System Abgrenzung

In dem Kontextdiagramm sind alle für die folgenden Betrachtungen relevanten Umsysteme aufgezeigt. Die Verbindungslinien zeigen die Kommunikationswege ohne dabei auf Inhalt und Richtung einzugehen.
Der Systemkontext lässt sich auf essenzieller Ebene wie folgt abgrenzen:

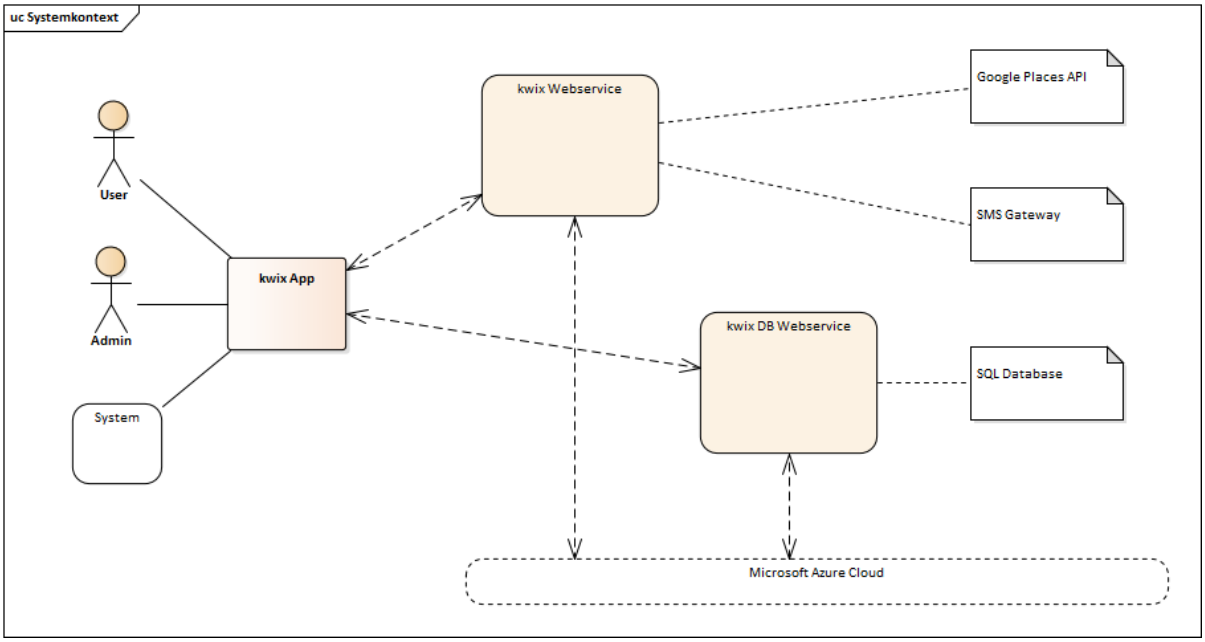


Abbildung 1 - Systemkontext

2.3 Akteure

Ein Akteur kann ein Benutzer, ein Administrator oder das System selbst sein.

2.3.1 Benutzer

Ein Benutzer/Nutzer ist eine natürliche Person, welche das System anwendet.
Ein Benutzer kann wie folgt mit dem System interagieren:

- Use Case 1: *Place* suchen
- Use Case 2: *Place* zu besuchten *Places* hinzufügen
- Use Case 3: *Place* zur Wunschliste hinzufügen
- Use Case 4: *Place* von besuchten *Places* entfernen
- Use Case 5: *Place* von Wunschliste entfernen
- Use Case 6: Kontaktieren
- Use Case 7: Registrieren

2.3.2 System

Das System kann sowohl die Rolle eines Webservice oder die der Applikation einnehmen.
Es führt Hintergrundfunktionen aus, welche weder der Administrator noch ein Benutzer auslösen kann.

- Use Case 8: Kontaktdaten auslesen
- Use Case 9: Benutzer bekannt machen
- Use Case 10: Zuletzt besuchte *Places* auslesen

2.3.3 Administrator

Ein Administrator ist ein Administrator des Systems. Er kann zusätzliche Informationen vom System über die Datenbank abfragen.

- Use Case 11: Anzahl Benutzer anzeigen
- Use Case 12: Erfasste *Places* anzeigen

2.4 Einschränkungen

Da das Projektteam der eigene Auftraggeber ist, bestehen keine Einschränkungen.

2.5 Wireframe

Für die konzeptionelle Erstellung der Applikation wurde zu Beginn der Arbeit ein ausführliches Wireframe erstellt.

2.5.1 Übersicht über die Bedienweise der Applikation

Nach der Installation der Applikation muss sich der Benutzer einmalig registrieren. Beim erneuten Öffnen der Applikation gelangt man direkt auf die Startseite mit den beliebten Ländern. In der Sektion der Länder kann mit einer Geste zwischen Ländern, Orten und Aktivitäten hin und her navigiert werden.
Wird ein beliebtes Land angeklickt kommt man eine Stufe tiefer, wo man sieht welche Kontakte bereits dieses Land besucht haben. Zusätzlich kann man auf dieser Ebene zu den beliebten Orten und Aktivitäten dieses Landes navigieren.

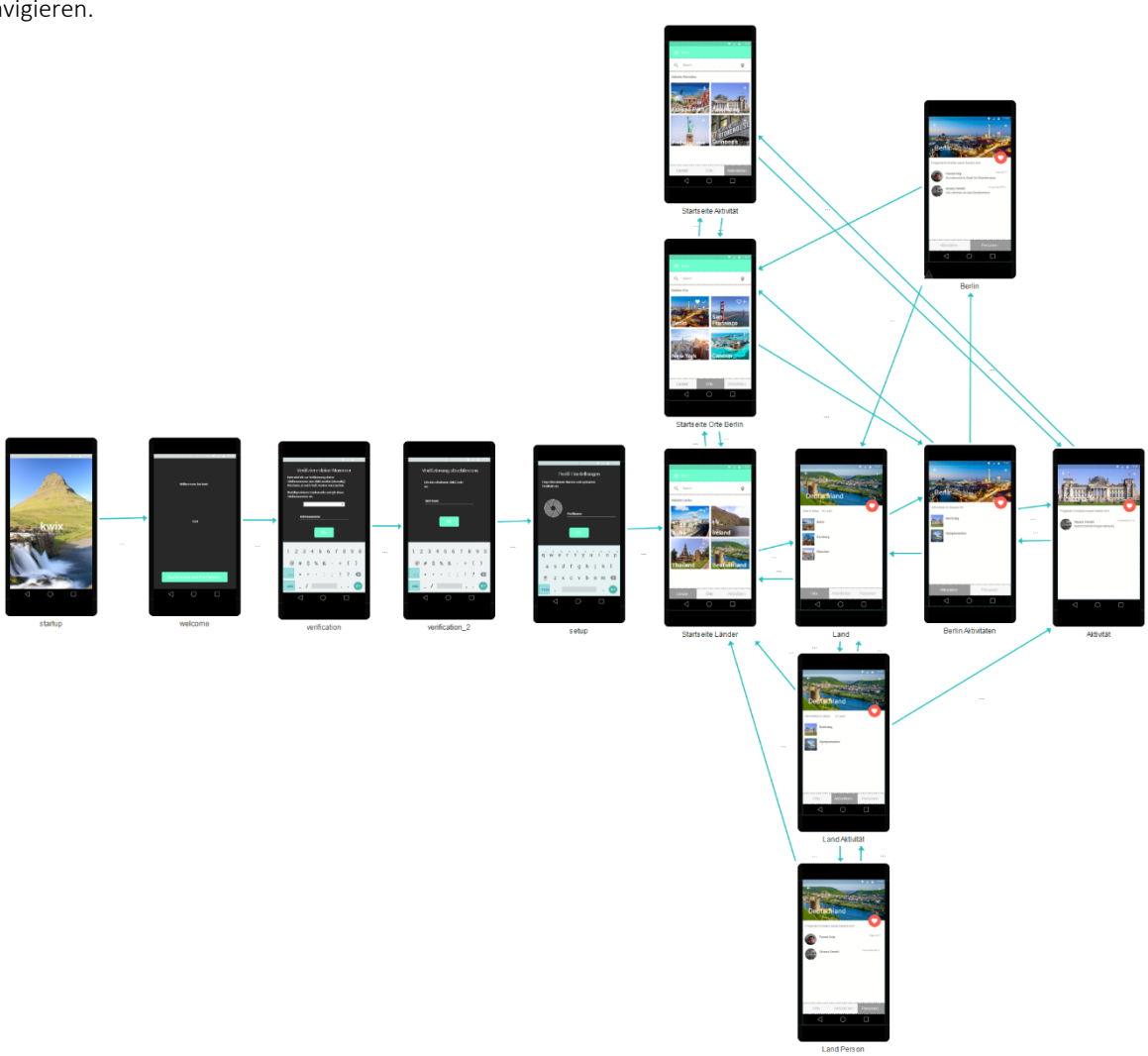


Abbildung 2 - Wireframe: Übersicht

2.5.2 Einen besuchten Ort hinzufügen

Um zum Beispiel einen Ort zu seinen besuchten *Places* hinzuzufügen, muss das Plus-Zeichen angeklickt werden. Dieser Button ist sowohl bei Ländern, Orten und Aktivitäten verfügbar.
Es erscheint ein *Pop-Up* um die Eingabe zu bestätigen oder abzubrechen.
Nach erfolgreichem Hinzufügen erscheint das Land, die Orte oder die Aktivität in der Rubrik Besuchte Orte.

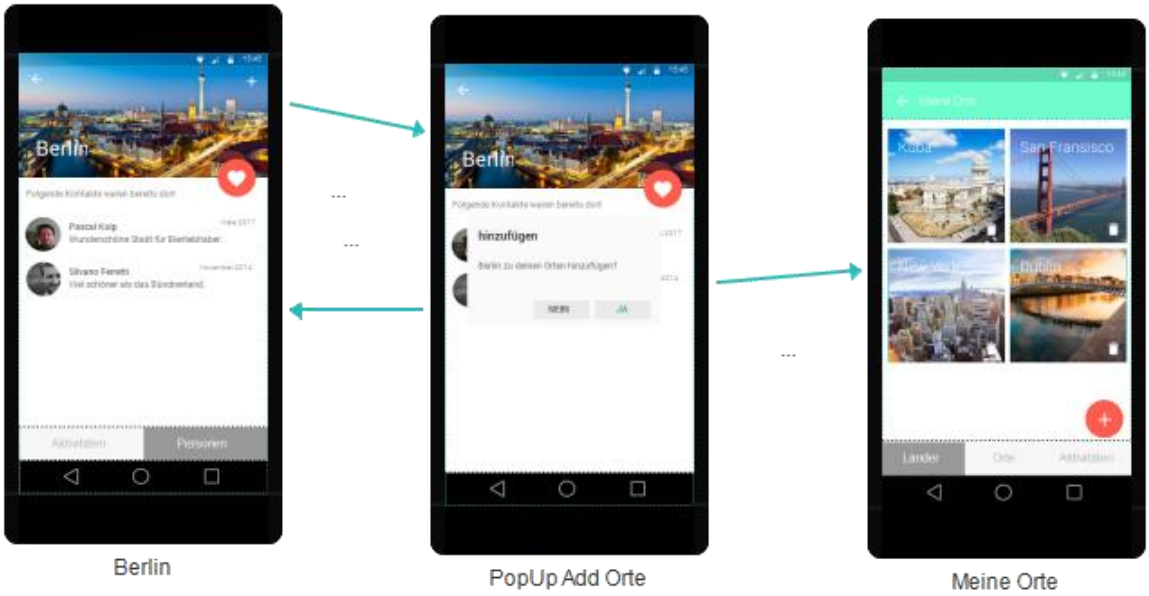


Abbildung 3 - Wireframe: Ort hinzufügen

2.5.3 Einen Ort zur Wunschliste hinzufügen

Um einen Ort seiner Wunschliste hinzuzufügen, muss der "Floating-Button" mit dem Herz Symbol angeklickt werden. Dieser Button ist sowohl bei Ländern, Orten und Aktivitäten verfügbar.
Es erscheint ein *Pop-Up* um die Eingabe zu bestätigen oder abzubrechen.
Nach erfolgreichem Hinzufügen, erscheint das Land, der Ort oder die Aktivität in der Wunschliste.

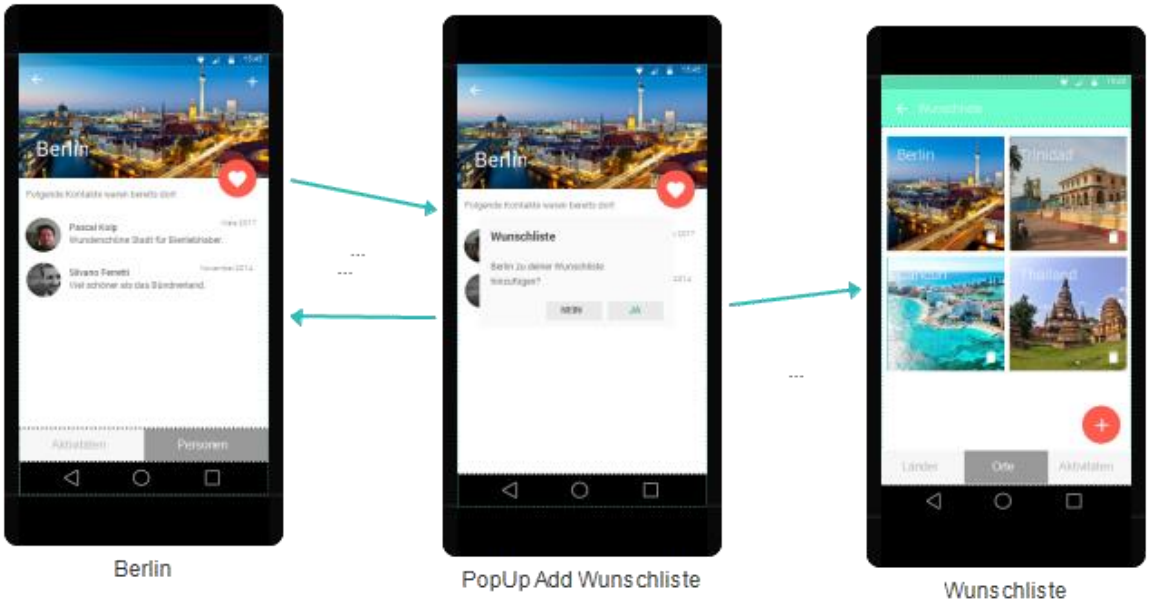


Abbildung 4 - Wireframe: Ort zu Wunschliste hinzufügen

2.5.4 Menü

Von der Startseite aus gelangt man ins Menü. Von dort aus hat man die Möglichkeit, seine bereits besuchten *Places* anzeigen zu lassen, die Wunschliste aufzurufen oder seine Profileinstellungen anzusehen und zu bearbeiten.

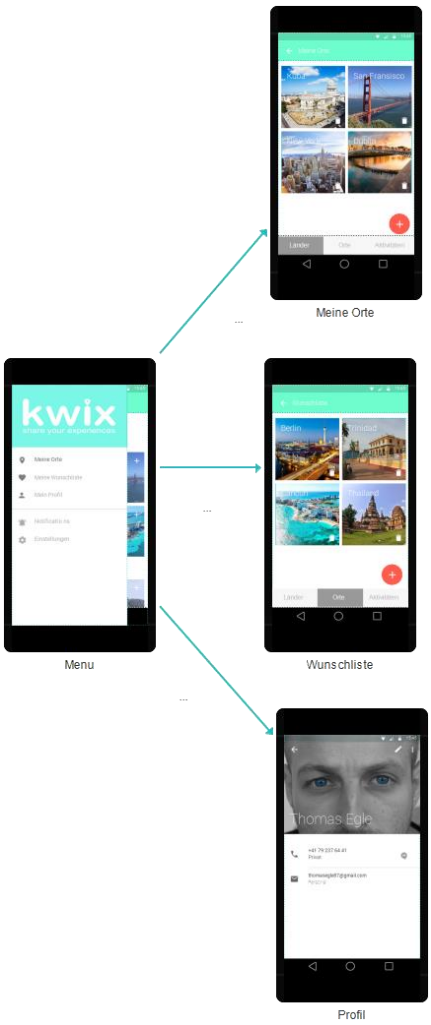


Abbildung 5 - Wireframe: Menu

3 Use Cases

Die zwölf Use Cases umfassen die wichtigsten Kernaufgaben der Applikation.

3.1 Übersicht der Use Cases (UC)

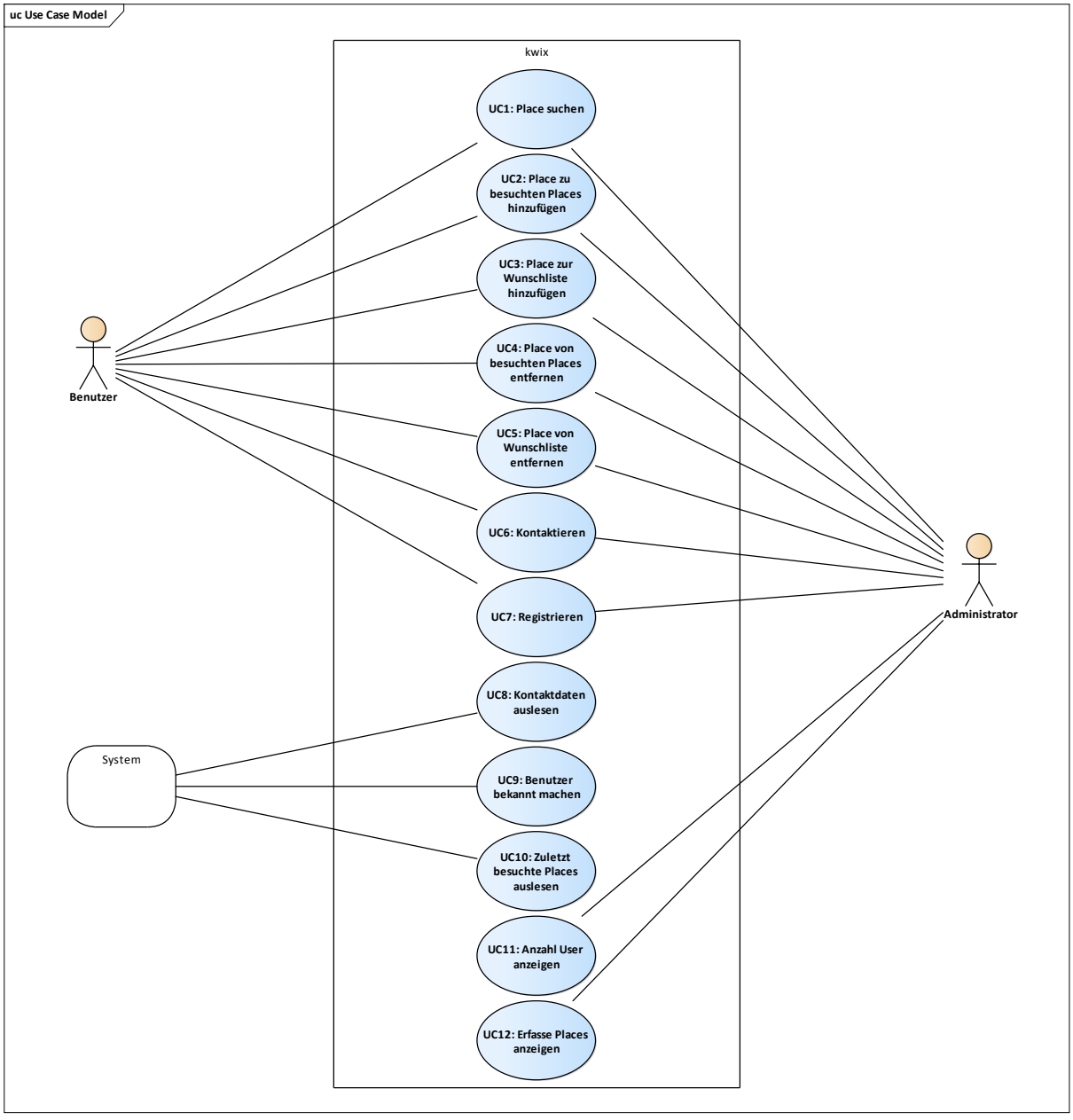


Abbildung 6 - Use Case Übersicht

3.1.1 UC1: Place suchen

ID	UC1
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann nach einem beliebigen *Place* suchen. Ein *Place* kann ein Land, ein Ort oder eine Aktivität sein.
Auslösende Nachricht	*Suche(Place)*
Szenario	• *Place* suchen Siehe Kapitel 6.3 Sequenzdiagramm: GooglePlace Autocomplete

3.1.2 UC2: Place zu besuchten Places hinzufügen

ID	UC2
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann einen *Place* zu seinen besuchten *Places* hinzufügen.
Auslösende Nachricht	*Place zu meinen besuchten Places hinzufügen (Place)*
Szenario	• *Place* suchen oder auswählen • *Place* zu meinen besuchten *Places* hinzufügen • *Place* dem Benutzer in der Datenbank als besucht eintragen • *Place* über Menüpunkt "Besucht" aufrufen Siehe Kapitel 6.4 Sequenzdiagramm: Hinzufügen einer Aktivität

3.1.3 UC3: Place zur Wunschliste hinzufügen

ID	UC3
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann einen *Place* zu seiner Wunschliste hinzufügen.
Auslösende Nachricht	*Place der Wunschliste hinzufügen (Place)*
Szenario	• *Place* suchen oder auswählen • *Place* zu meiner Wunschliste hinzufügen • *Place* dem Benutzer in der Datenbank als Wunsch eintragen • Wunschliste über Menüpunkt "Wunschliste" aufrufen

3.1.4 UC4: Place von besuchten Places entfernen

ID	UC4
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann einen *Place* von seinen besuchten *Places* entfernen.
Auslösende Nachricht	*Place von meinen besuchten Places entfernen (Place)*
Szenario	• *Place* in besuchten *Places* auswählen Android: langes tippen auf den *Place* iOS: *Place* nach links *swipen* • *Place* von meinen besuchten *Places* entfernen • *Place* dem Benutzer in der Datenbank entfernen Siehe Kapitel 6.4 Sequenzdiagramm: Hinzufügen einer Aktivität

3.1.5 UC5: Place von Wunschliste entfernen

ID	UC5
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann einen *Place* von seiner Wunschliste entfernen.
Auslösende Nachricht	*Place von Wunschliste entfernen (Place)*
Szenario	• *Place* in Wunschliste auswählen Android: langes tippen auf den *Place* iOS: *Place* nach links *swipen* • *Place* von Wunschliste entfernen • *Place* dem Benutzer in der Datenbank entfernen

3.1.6 UC6: Kontaktieren

ID	UC6
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann einen anderen Benutzer kontaktieren (Telefonanruf / SMS / WhatsApp), wenn dieser einen gesuchten *Place* bereits besucht hat und diese untereinander bekannt sind (UC9).
Auslösende Nachricht	*Kontaktiere (Benutzer)*
Szenario	• *Place* suchen oder auswählen • Kontakt kontaktieren

3.1.7 UC7: Registrieren

ID	UC7
Auslösender Aktor	Benutzer, Administrator (als Benutzer)
Motivation	Ein Benutzer kann sich mit seiner Telefonnummer im System registrieren.
Auslösende Nachricht	• *Authentifizierungs SMS senden (Telefonnummer)* • *Code überprüfen (Code, Telefonnummer)*
Szenario	Siehe Kapitel 6.1 Sequenzdiagramm: Authentifizierung eines Benutzers

3.1.8 UC8: Kontaktdaten auslesen

ID	UC8
Auslösender Aktor	System
Motivation	Das System kann von einem Benutzer die Kontaktdaten auslesen und diese als Hashwert in der Datenbank abspeichern. Durch das Abspeichern der Kontaktdaten als Hashwert wird der Schutz der persönlichen Daten gewährleistet.
Auslösende Nachricht	*Kontaktdaten auslesen und mit Datenbank abgleichen ()*
Szenario	• Kontaktdaten vom Benutzer auslesen • Kontaktdaten in der Datenbank speichern

3.1.9 UC9: Benutzer bekannt machen

ID	UC9
Auslösender Aktor	System
Motivation	Das System kann Anhand von UC8 die Kontakte bekannt machen.
Auslösende Nachricht	*Verbindung zwischen Benutzern herstellen ()*
Szenario	Siehe Kapitel 6.6 Sequenzdiagramm: Freunde erkennen

3.1.10 UC10: Zuletzt besuchte Places auslesen

ID	UC10
Auslösender Aktor	System
Motivation	Das System kann Anhand von UC9 die zuletzt besuchten *Places* von seinen Kontakten auslesen.
Auslösende Nachricht	*Zuletzt besuchte Places auslesen (Telefonnummer)*
Szenario	• Das System sendet die *gehashte* Telefonnummer vom Benutzer an den DB Webservice. • DB Webservice gibt die sechs zuletzt besuchten Länder, Orte und Aktivitäten der Kontakte vom Benutzer zurück.

3.1.11 UC11: Anzahl Benutzer anzeigen

ID	UC11
Auslösender Aktor	Administrator
Motivation	Ein Administrator kann sich die Anzahl der registrierten Benutzer, welche in der Datenbank angelegt sind, anzeigen lassen.
Auslösende Nachricht	*Zeige Anzahl registrierter Benutzer ()*
Szenario	• Datenbank Abfrage

3.1.12 UC12: Erfasste Places anzeigen

ID	UC12
Auslösender Aktor	Administrator
Motivation	Ein Administrator kann sich eine Liste aller erfassten *Places*, welche in der Datenbank angelegt sind, anzeigen lassen.
Auslösende Nachricht	*Liste aller Places ()*
Szenario	• Datenbank Abfrage

4 Nicht-funktionale Anforderungen (NF)

4.1 NF1: Verfügbarkeit

NF1: Die angebundene Azure SQL Datenbank befindet sich in einer Cloud. Die garantierte Verfügbarkeit zur Datenbank wir von Microsoft mit mindestens 99.99% angegeben.

4.2 NF2: Zuverlässigkeit

NF2: Die angezeigten Daten entsprechen den von den Benutzern hinterlegten Informationen.

4.3 NF3 / NF4: Anmeldevorgang

Der Anmeldevorgang soll einfach und benutzerfreundlich sein:
NF3: Die Anmeldung erfolgt über die Mobilnummer
NF4: Der Anmeldevorgang dauert maximal 1 Minute.

4.4 NF5: Robustheit

NF5: Die Applikation soll stabil laufen und soll nicht mehr als 1mal pro 24h Laufzeit abstürzen.

4.5 NF6: Effizienz

NF6: Es können parallel mindestens 20 Benutzer auf die Applikation mit vollem Funktionsumfang zugreifen.

4.6 NF7: Kunden- und Systemsicherheit

NF7: Die eingegebenen Daten der Benutzer werden so verarbeitet, dass es anhand der gespeicherten Daten nicht möglich sein wird einen Aufschluss auf den jeweiligen Benutzer zu erlangen.

4.7 NF8: Suchabfrage

NF8: Eine Suchabfrage dauert weniger als fünf Sekunden.

5 Domain Model

Untenstehendes Diagramm zeigt die Real World Entitäten als UML Domain Model. Dabei werden folgende drei Hauptanwendungsfälle modelliert:

- Use Case 2: *Place* zu besuchten *Places* hinzufügen
- Use Case 3: *Place* zur Wunschliste hinzufügen
- Use Case 9: Benutzer bekannt machen

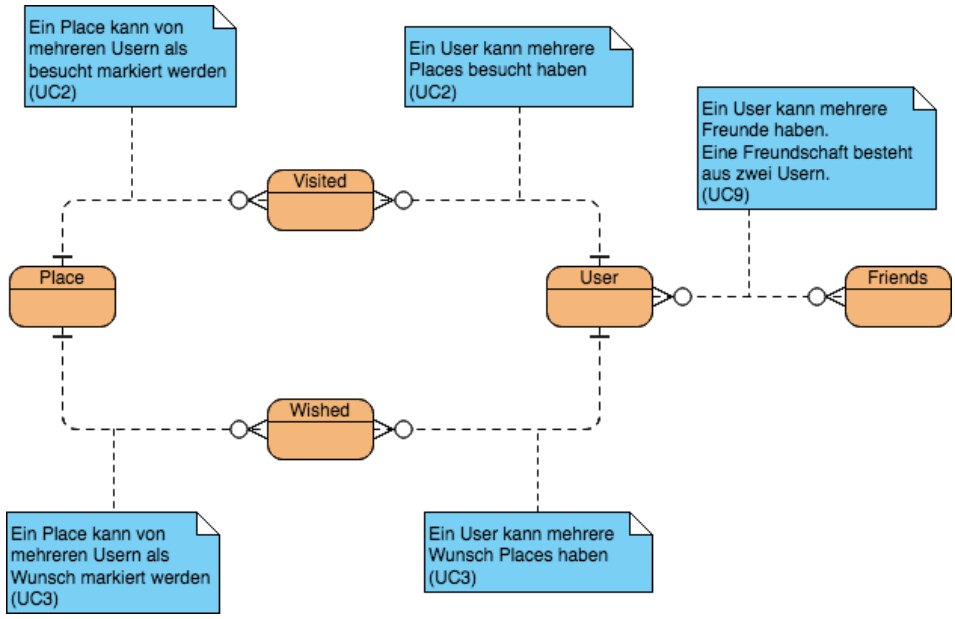


Abbildung 7 - Domain Model

6 System-Interaktion

Die folgenden Sequenzdiagramme zeigen die Interaktionen vom Benutzer mit der App *kwix*, sowie der Systeme untereinander.

6.1 Sequenzdiagramm: Authentifizierung eines Benutzers

Die Authentifizierung eines Benutzers erfolgt mittels Telefonnummer.

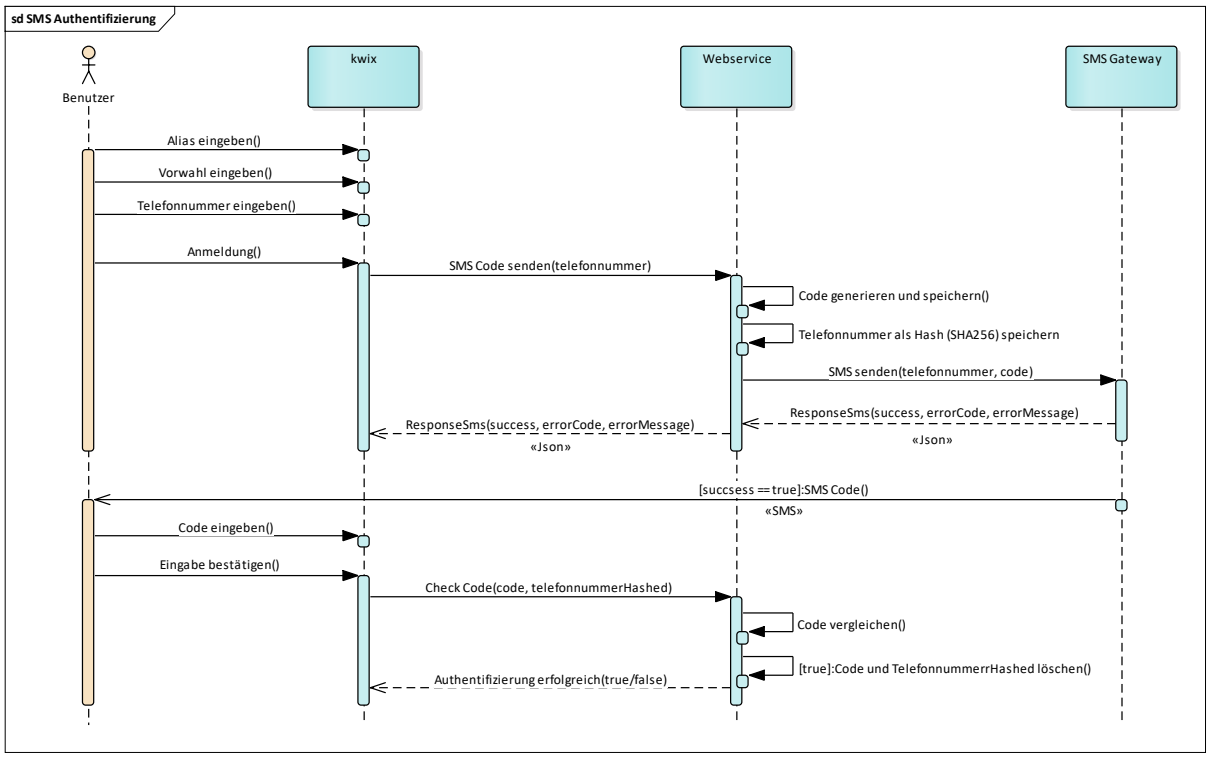


Abbildung 8 - Sequenzdiagramm: SMS Authentifizierung

6.2 Sequenzdiagramm: GooglePlace Autocomplete

Das Sequenzdiagramm ist ohne die Umkreissuche dargestellt. Bei der Autocomplete Abfrage mit Umkreissuche wird von der App der GPS Standort abgefragt und dann an den Webservice als optionaler Parameter übergeben.

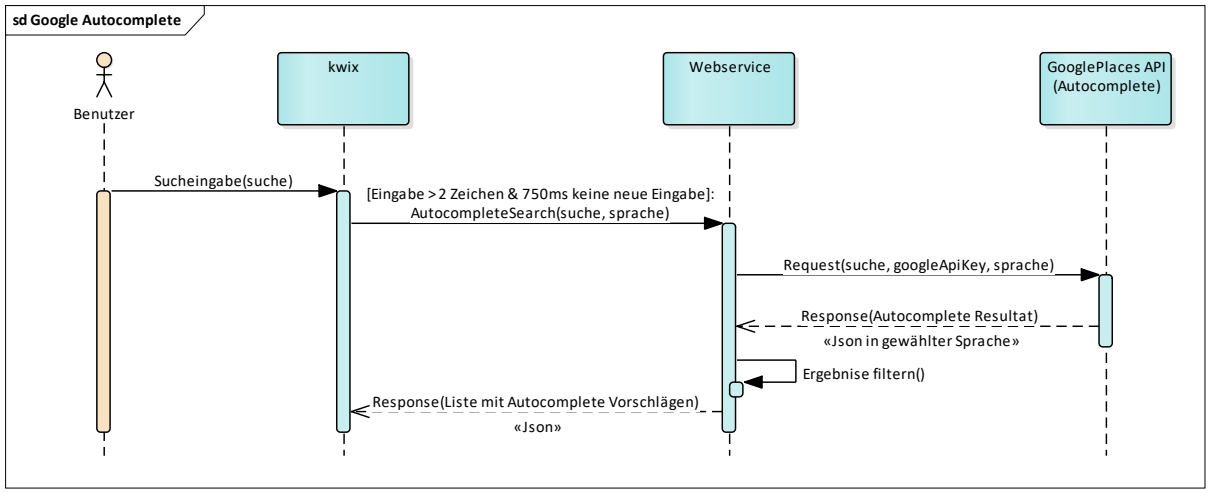


Abbildung 9 - Sequenzdiagramm: Google Autocomplete

6.3 Sequenzdiagramm: Suchen und Anzeigen eines Place

Von der Autocomplete Abfrage (siehe Kapitel 6.2 Sequenzdiagramm: GooglePlace Autocomplete) wird eine sogenannte Google Place ID des gesuchten *Places* übermittelt. Diese ID erlaubt es ein *Place* eindeutig zu identifizieren. So hat zum Beispiel ein Land immer die gleiche ID, egal in welcher Sprache danach gesucht wird.

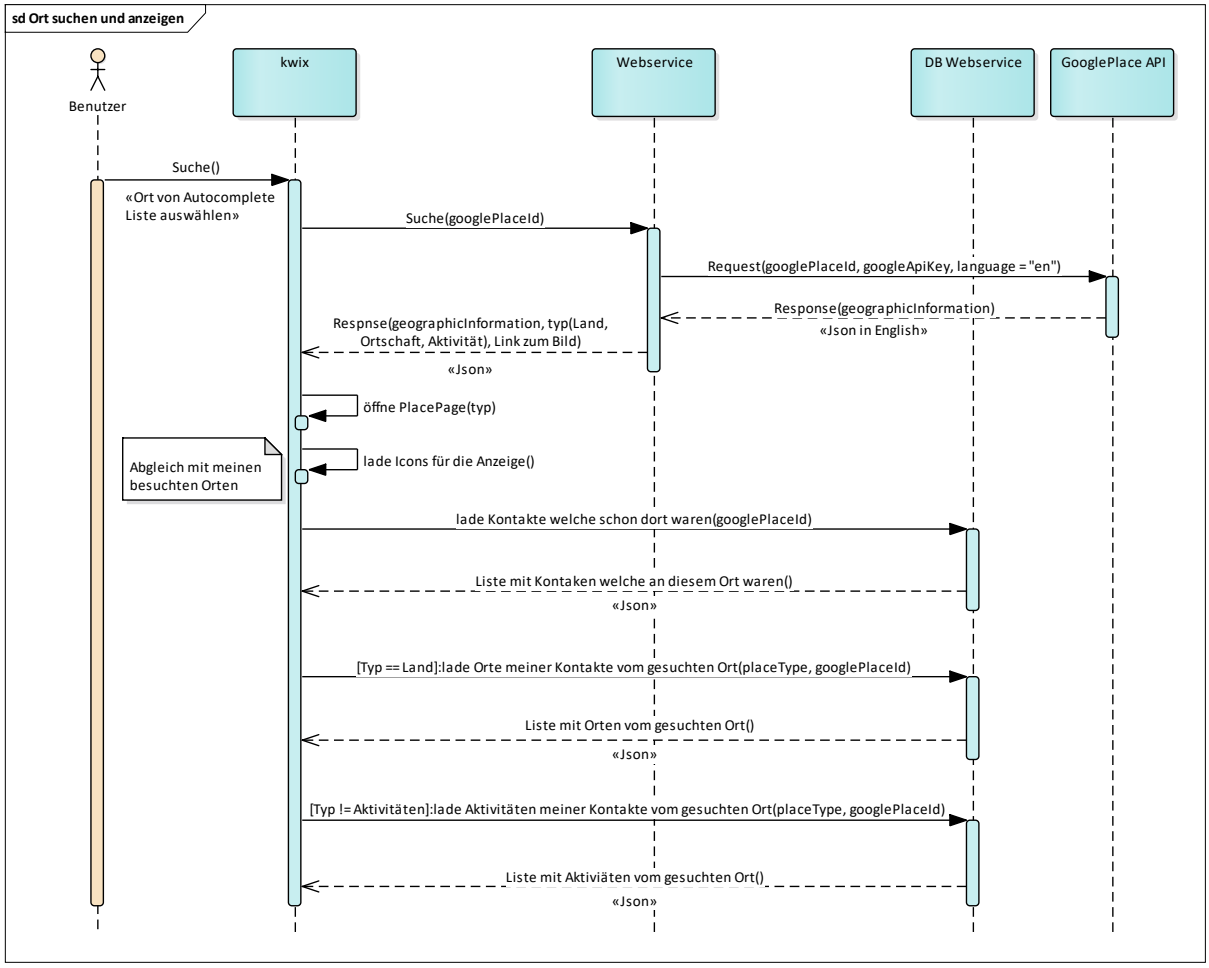


Abbildung 10 - Sequenzdiagramm: Ort suchen und anzeigen

Die folgenden Wireframes zeigen die Darstellung der Suchergebnisse auf der entsprechenden *Place* Seite, welche anhand des Types (*Country*, *Locality*, *Activity*) aufgerufen werden.

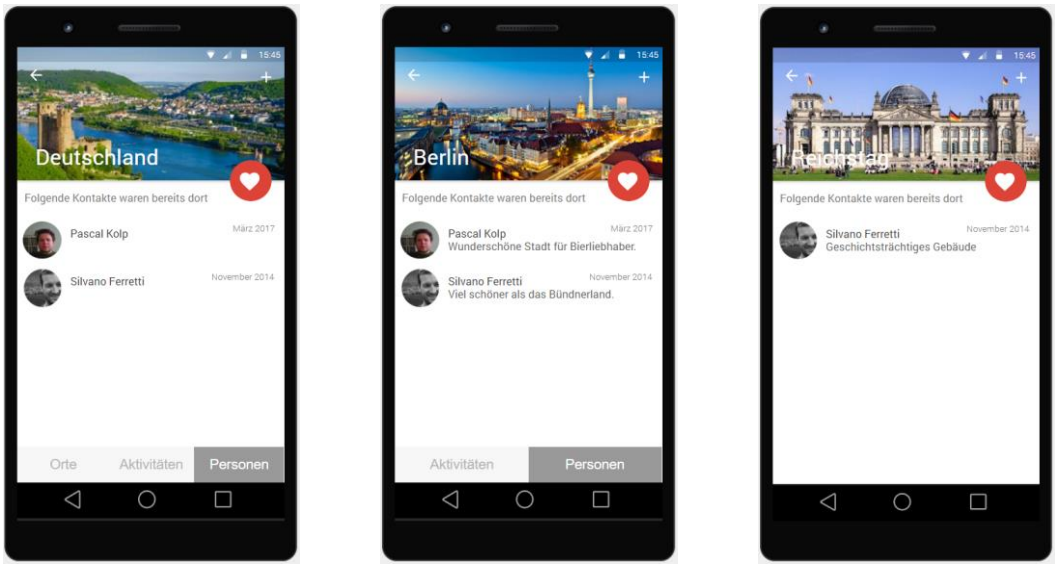


Abbildung 11 - Places Seiten aus Wireframes

6.4 Sequenzdiagramm: Hinzufügen einer Aktivität

Eine Aktivität kann auf der Startseite oder auf der Aktivitäten Seite hinzugefügt werden. Die Ausgangslage für das folgende Sequenzdiagram ist: Das Land, in dem sich die Aktivität befindet, ist bereits in der Datenbank eingetragen und wurde vom Benutzer bereits als besuchtes Land hinzugefügt. Das Hinzufügen einer Aktivität zur Wunschliste erfolgt nach dem gleichen Prinzip.

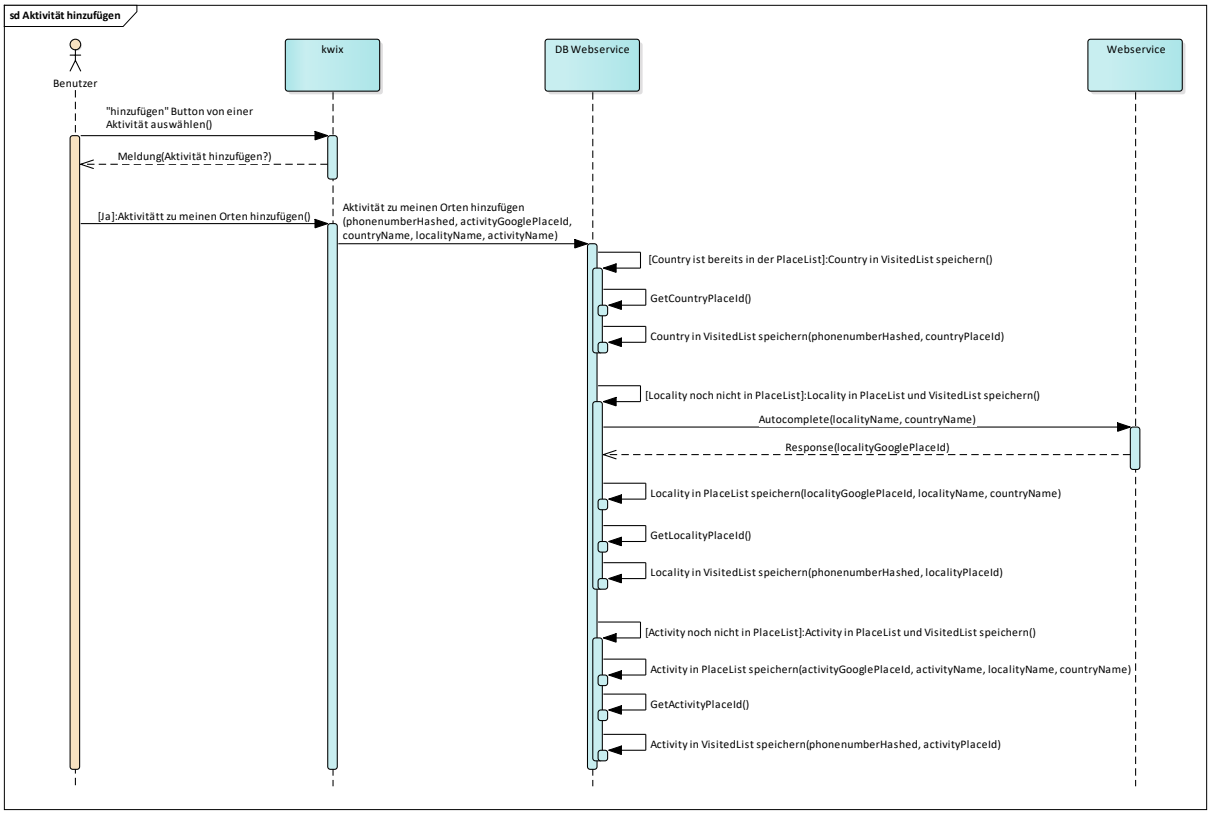


Abbildung 12 - Sequenzdiagramm: Aktivität hinzufügen

6.5 Sequenzdiagramm: Löschen von einem Land

Beim Löschen eines Landes von den besuchten *Places* eines Benutzers, werden auch immer alle Orte und Aktivitäten, welche von diesem Land als besucht gespeichert wurden, mitgelöscht.

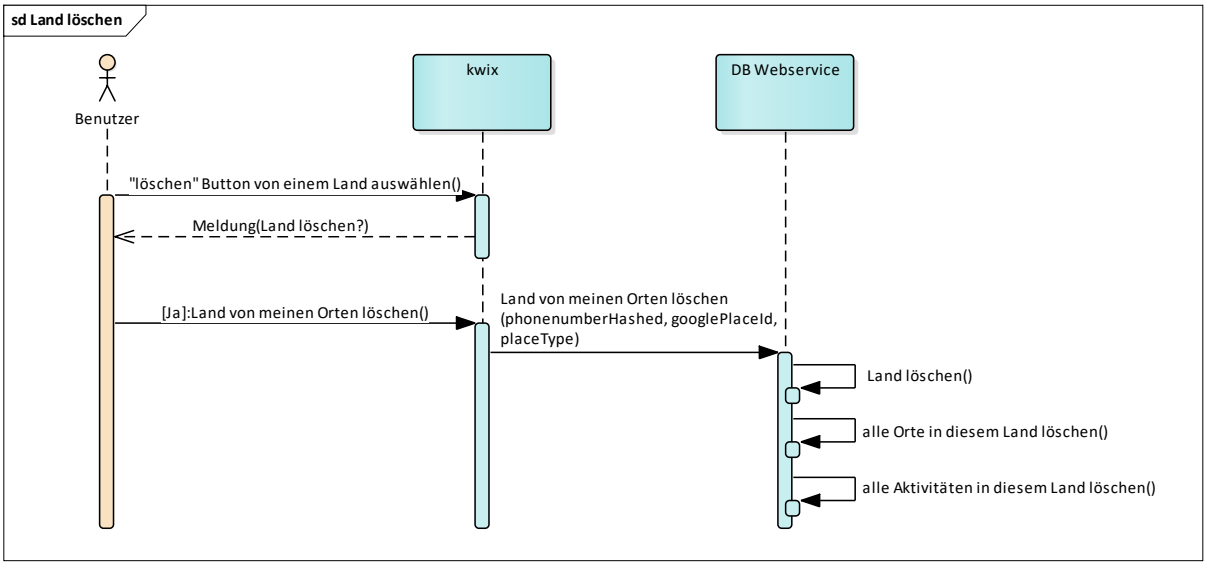


Abbildung 13 - Sequenzdiagramm: Land löschen

6.6 Sequenzdiagramm: Freunde erkennen

Damit jeder Benutzer nur die *Places* der eigenen Kontakte sieht, macht das System bei jedem starten der App einen Abgleich der Telefonnummern aus dem Telefonbuch des Benutzers.

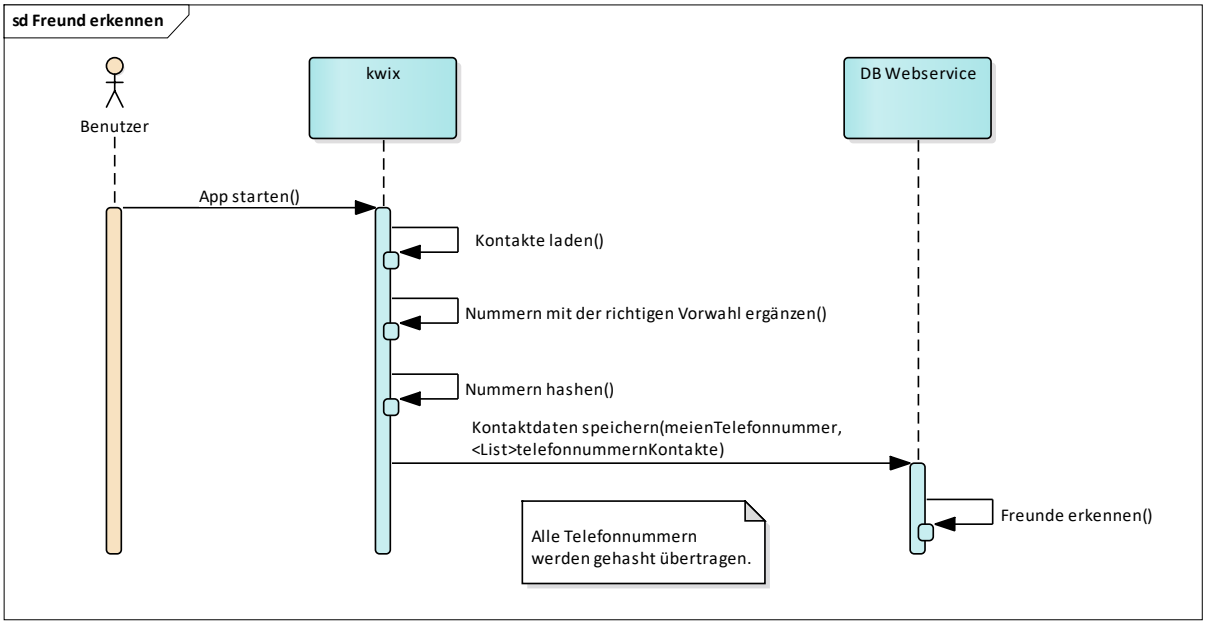


Abbildung 14 - Sequenzdiagramm: Freunde erkennen

7 Schnittstellen

Die Schnittstellen der Applikation bestehen in der Kommunikation mit den verschiedenen Webservices (siehe Kapitel 2.2 System Abgrenzung).

7.1 Datenstruktur der Schnittstellen

Die Datenstruktur der Webservices ist in *JSON*.

7.2 Randbedingungen

Bei den Randbedingungen wurde darauf geachtet, eine grösstmögliche Verfügbarkeit der verschiedenen Betriebssysteme abzudecken.

7.2.1 Android - Kompatibilität

Die Applikation unterstützt Smartphones ab einer Android Version 5.0 (Lollipop, API Level 21) und ist kompatibel bis und mit Android 8.1 (Oreo API Level 27). Rund 87% aller Android Smartphones haben eine Android Version von 5.0 oder höher. [2]

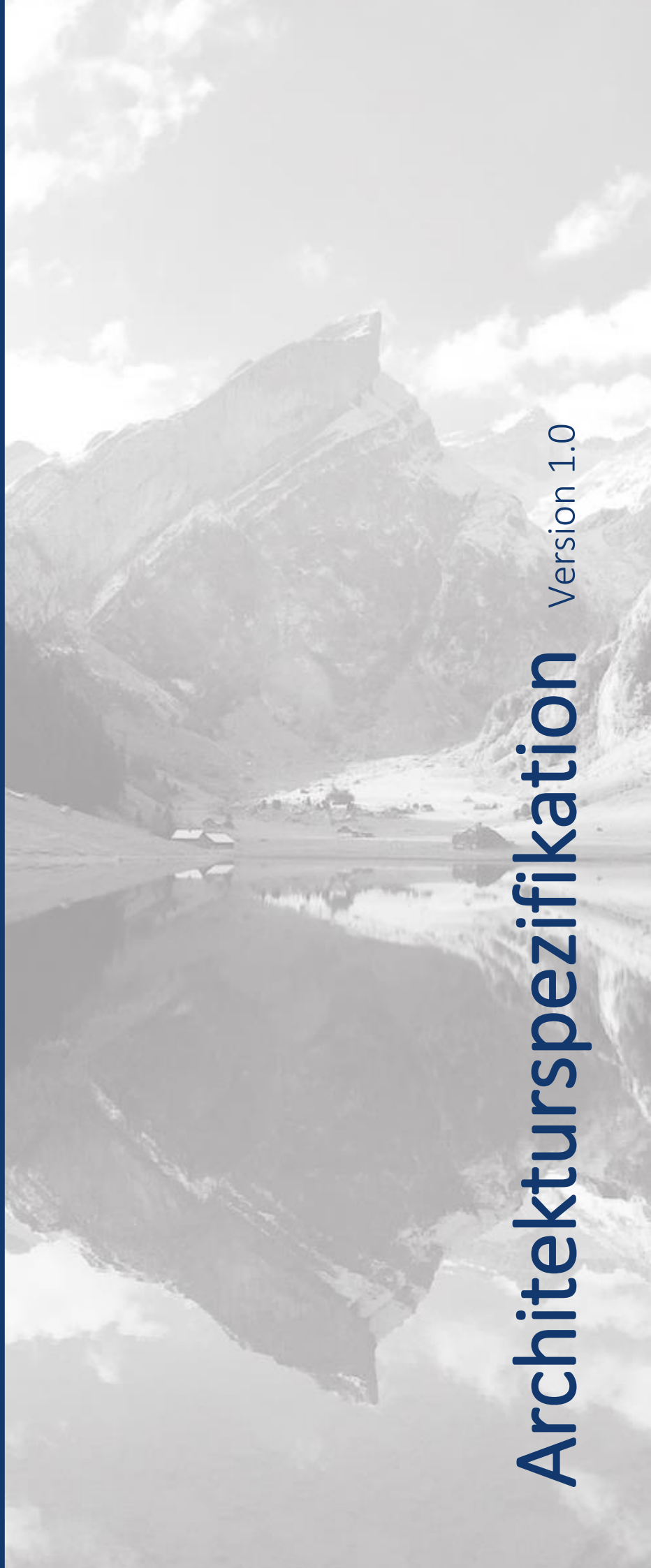
7.2.2 iOS - Kompatibilität

Die Applikation unterstützt Geräte (iPhone) ab iOS 10, was 95% aller iPhones entspricht. [3]

8 Abbildungsverzeichnis

kwix

share your experiences



Architekturspezifikation

Version 1.0

Thomas Egle
Silvano Ferretti
Pascal Kolp

Änderungsgeschichte

Version	Autor	Beschreibung	Datum
0.1	Thomas Egle	Erstellung	23.04.2018
0.2	Thomas Egle	GooglePlace API: Kosten und Logoanforderung hinzugefügt	08.05.2018
0.3	Thomas Egle	App Center hinzugefügt	15.06.2018
0.4	Thomas Egle	Optimierung DB Abfragen hinzugefügt	06.08.2018
0.5	Silvano Ferretti	API Schnittstellen hinzugefügt Klassendiagramme hinzugefügt	13.08.2018
0.6	Pascal Kolp	Architektonische Ziele und Einschränkungen hinzugefügt	13.08.2018
0.7	Pascal Kolp	Datenschutz und Datensicherheit, Archtektur und Designentscheidungen hinzugefügt	17.08.2018
0.8	Thomas Egle	Systemübersicht, Deplyoment und Kosten hinzugefügt	17.08.2018
0.9	Silvano Ferretti	Grössen und Leistungen hinzugefügt	20.08.2018
1.0	Projektteam	Freigabe	26.08.2018

Inhaltsverzeichnis

1 Einführung

1.1 Zweck

Dieses Dokument beschreibt das Software-Design, die technische Umsetzung sowie die Architektur- und Designentscheidungen, die getroffen wurden, um alle Anforderungen des Projekts *kwix - share your experiences* zu erfüllen.

1.2 Gültigkeitsbereich

Dieses Dokument ist über die ganze Projektdauer gültig. Änderungen werden fortlaufend ergänzt und in der Änderungsgeschichte festgehalten.

1.3 Referenzen

Nr	Titel	Quelle
[1]	Glossar	…\Dokumente\Glossar_V1.0.pdf
[2]	Foursquare	https://de.foursquare.com/
[3]	Country Code	http://www.countryareacode.net/de http://www.countryareacode.net/de/was-ist-eine-telefonvorwahl#IDD
[4]	SMS Gateway	https://www.moreify.com/en
[5]	Restcountries	https://restcountries.eu/
[6]	Google Place Autocomplete	https://developers.google.com/places/web-service/autocomplete
[7]	Google Logo Anforderung	https://developers.google.com/places/web-service/policies?hl=de
[8]	Google Place Details	https://developers.google.com/places/web-service/details
[9]	Google Place Photos	https://developers.google.com/places/web-service/photos
[10]	WhatsApp API	https://api.whatsapp.com/send?phone=&text= https://faq.whatsapp.com/en/android/26000030/?category=5245251
[11]	ReactiveUI	https://reactiveui.net/
[12]	App Center	https://appcenter.ms https://appcenter.ms/orgs/kwix https://docs.microsoft.com/en-us/appcenter/sdk/getting-started/xamarin
[13]	Datenschutzverordnung	https://www.kmu.admin.ch/kmu/de/home/praktisches-wissen/kmu-betreiben/e-commerce/eu-regelung-zum-datenschutz.html
[14]	Projektplan	…\Dokumente\Projektplan_V1.0.pdf
[15]	Anforderungsspezifikation	…\Dokumente\Anforderungsspezifikation_V1.0.pdf
[16]	Azure	https://azure.microsoft.com/de-de/pricing/ https://azure.microsoft.com/de-de/free/students-starter-faq/
[17]	Google Kosten	https://developers.google.com/maps/premium/usage-limits
[18]	Google Rechnung	…\Dokumente\Rechnung_Google.pdf
[19]	SMS Gateway Kosten	https://members.moreify.com/en/pricing

1.4 Glossar

Alle verwendeten Begriffe aus dem Glossar werden in diesem Dokument *kursiv* geschrieben.

Siehe Glossar [1]

2 Systemübersicht / Architekturübersicht

Die mobile Applikation *kwix* ist mit Xamarin in C# im Visual Studio entwickelt. Im Visual Studio ist Microsoft Azure, welches die Webservices (C#), Datenbank (SQL) und Visual Studio Team Service (*VSTS*) enthält, direkt verknüpft.
Die komplette Projektplanung, die drei *Git Repositories* sowie ein Wiki (Notizen, Links, Protokolle) sind im *VSTS* enthalten. Das App Center, welches die Applikation auf Android und iOS *buildet* und verteilt, ist mit dem *kwix Git Repository* vom *VSTS* verknüpft.

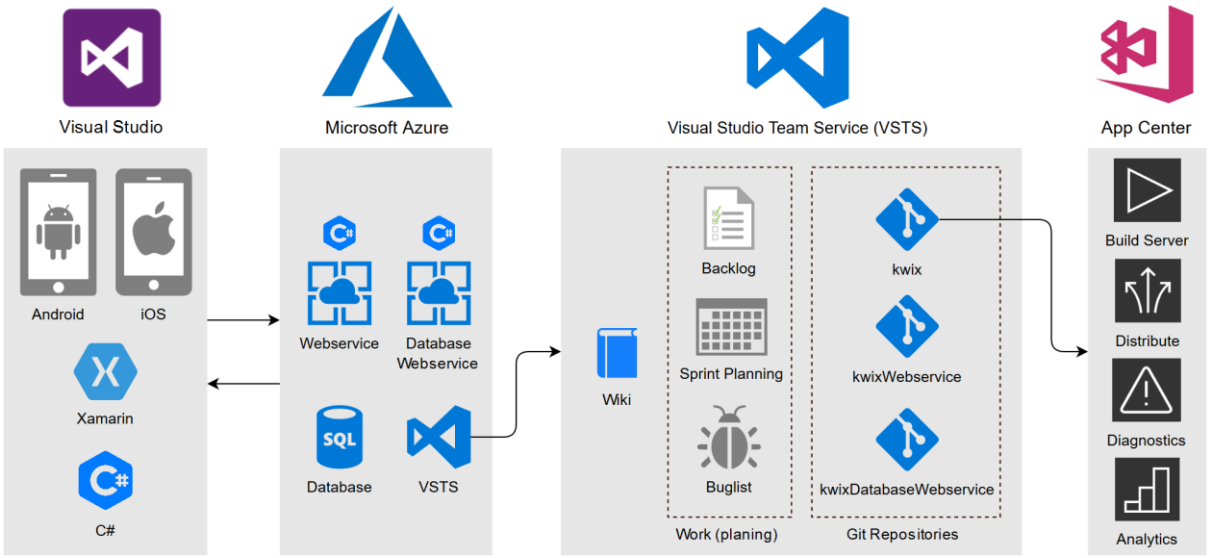


Abbildung 1 - Architekturübersicht

2.1 Anwendungsbereich

Die App *kwix – share your experiences* richtet sich an Android und iOS Nutzer, welche gerne Reisen und die selber bereisten *Places* seinen Kontakten mitteilen möchte. Mit *kwix* muss man sich nicht mehr auf irgendwelche anonymen Bewertungen aus dem Internet verlassen, da man die Personen, welche bereits an dem gewünschten *Place* waren, persönlich kennt.

2.2 Deployment View

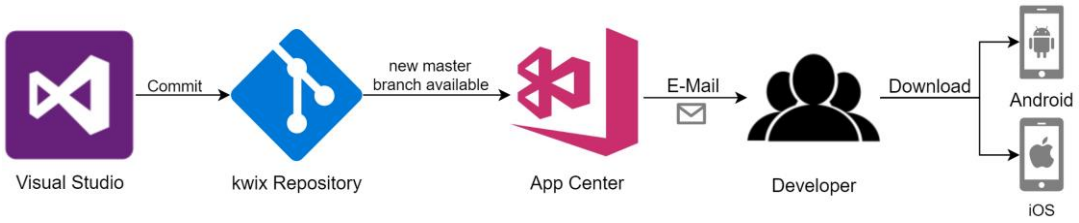


Abbildung 2 - Deployment Model

Das Deployment Model in der Abbildung 2 stellt die Verteilung der Android und iOS Applikation dar. Momentan wird *kwix* noch nicht über einen App Store (Google Play Store oder iOS App Store) verteilt. Wie die Abbildung 3 zeigt, kann dies aber im App Center einfach implementiert werden.

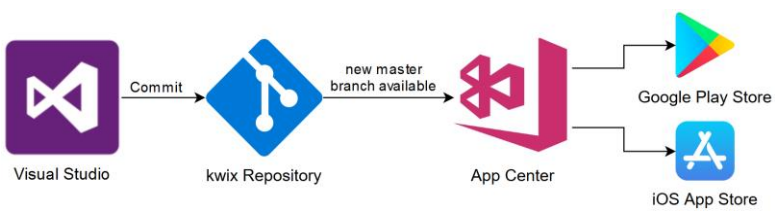


Abbildung 3 - Deployment Model mit App Stores

3 Architektonische Ziele und Einschränkungen

Um eine grösstmögliche Flexibilität zu erreichen, wurde die SQL Datenbank und die Webservices in der Azure Cloud ausgelagert, damit eine nachträgliche Skalierung jederzeit möglich ist.

3.1 Ziele

Die Applikation soll nur als User Interface dienen, welches dem Benutzer die gewünschten Inhalte zur Verfügung stellt. Zusätzlich zu der Applikation wurden zwei getrennte Webservices erstellt. Ein Webservice für alle Datenbankzugriffe und ein Webservice für die verschiedenen Services (siehe Kapitel 4.2.1 Allgemeiner Webservice). Die Webservices sind beliebig anpassbar, ohne dass ein Update der App notwendig ist.
Die Datenbank soll erweiterbar und skalierbar sein, damit auf ändernde Benutzerzahlen schnell reagiert werden kann.

Ein weiterer Grund, um die Services in einen Webservice auszulagern, ist die Austauschbarkeit der Services, wie der Google Place API oder des SMS Gateways. Auch kann bei einer Anpassung der *API* durch den Anbieter schnell reagiert werden.
Die Google Place API stellt für die Applikation ein sehr wichtiger Teil dar. Wenn aus strategischen oder wirtschaftlichen Gründen diese *API* nicht mehr verwendet werden kann, ist es möglich den Anbieter zu wechseln. Eine Alternative zur Google Places API ist die *API* von Foursquare [2].

3.2 Einschränkungen

Bei einer möglichen Umstellung der Google Places API müssen zwei grundsätzliche Anpassungen vorgenommen werden, welche jedoch die einzigen Einschränkungen bei einer Umstellung der API darstellen würden:

1.) Anpassung der *API* Antworten welche im *JSON* Format zur Verfügung gestellt werden. Diese sind bei Google und Foursquare unterschiedlich.
2.) Portierung des *Place* Keys welcher ein *Place* eindeutig identifiziert.

4 Logische Architektur

Da die Applikation auf dem Xamarin.Forms Framework aufbaut und Xamarin.Forms Data Binding unterstützt, wird das populäre Model View ViewModel (*MVVM*) Pattern verwendet. Für den Presentation Layer wird XAML verwendet. Alle Seiten sind mit XAML aufgebaut und der jeweilige Code Behind ist, wo möglich, schlank gehalten und es wird direkt über Data Binding zwischen ViewModel und View interagiert. Die hauptsächliche Logik steckt in den ViewModels. In den Models werden die benötigten Datenstrukturen zur Verfügung gestellt.
Es wurde darauf geachtet, so viel wie möglich als plattformunabhängiger Code, welcher auf der .NET Standard Library aufbaut, zu verwenden. Teilweise konnte plattformspezifischer Code nicht umgangen werden, da Android und iOS nicht dieselben Implementierungsvarianten unterstützen. So gibt es auf der Android Plattform zum Beispiel eine Snackbar Notifikation, welche auf iOS nicht verfügbar ist. Das Auslesen der Telefonnummern aus dem Telefonbuch wird ebenfalls plattformspezifisch abgehandelt.
Die Anbindung an die Webservices erfolgt durch verschiedene Services, welche von den ViewModels aufgerufen werden.

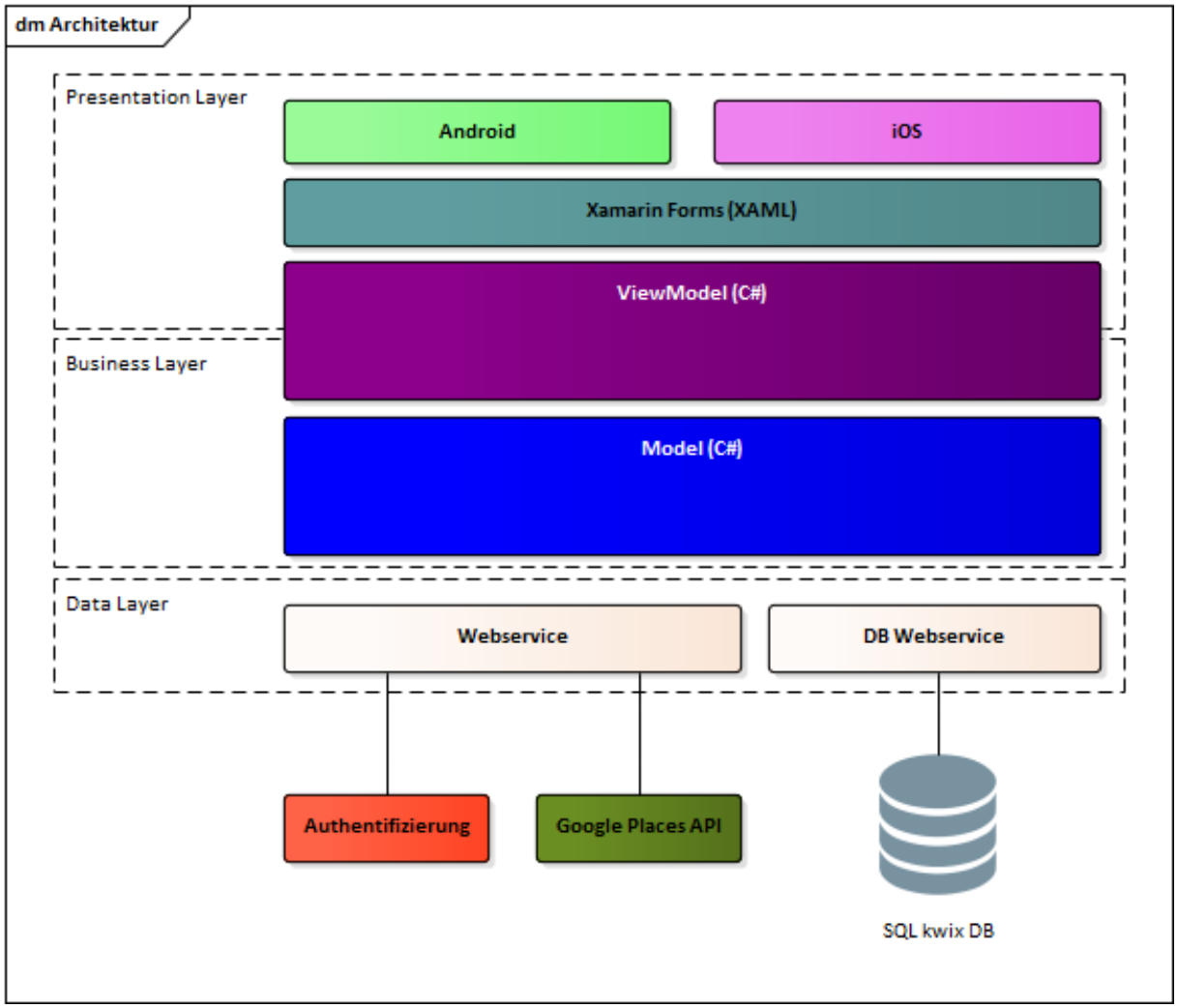


Abbildung 4 - Logische Architektur

4.1 Software Komponenten

Die Applikation baut auf der Model View ViewModel Architektur auf. Dabei stellt die View die grafische Bedienoberfläche dar. Dieser Teil wurde mittels XAML umgesetzt. Die komplette Logik der Applikation befindet sich in den ViewModels. Die ViewModels enthalten einerseits die Daten, welche über Data Binding oder Commands mit der View verbunden sind, andererseits die meisten Methoden, welche für die Anzeige oder die Auswertung von Benutzereingaben verarbeitet werden müssen. Weitere Methoden wie z.B. die Kommunikation zu unseren Webservices sind in das Package Services ausgelagert. Das Model stellt dem ViewModel die benötigten Daten und Klassen zur Verfügung.
Zwischen View und ViewModel besteht eine lose Kopplung wo hingegen eine feste Verbindung zwischen ViewModel und Model besteht. Die Views können so einfach angepasst werden ohne dass die Logik geändert werden muss.

Im Kapitel 4.4 Klassendiagramm wird die selbe Diagrammstruktur verwendet. So befinden sich auf der linken Seite immer die Views, in der Mitte die ViewModels und wenn vorhanden auf der rechten Seite die zugehörenden Models.

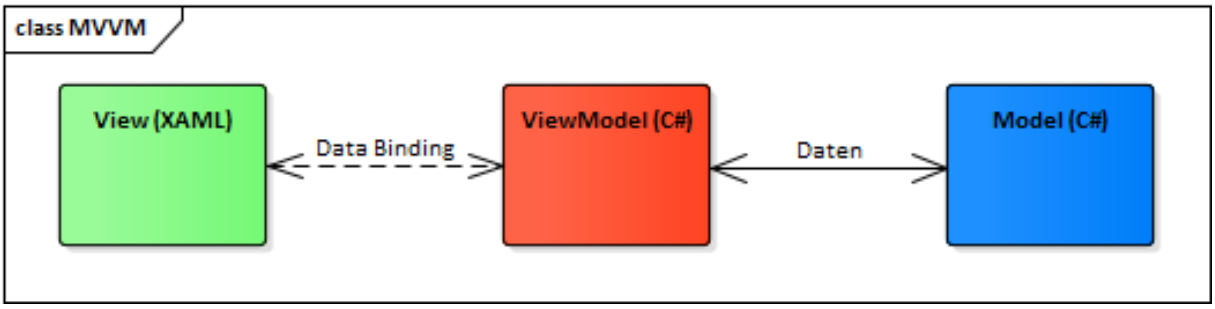


Abbildung 5 - MVVM Struktur

4.2 Schnittstellen

Nachfolgend werden die Schnittstellen zu den Webservices erläutert. Dabei wird auf die zwei verwendeten Webservices eingegangen. Die Datenstruktur der Rückgabewerte von den Webservices ist in *JSON*.

4.2.1 Allgemeiner Webservice

Dieser Webservice basiert auf *REST*. Die Basis *URI* zu diesem Webservice lautet www.api.kwix.ch.
Die für den allgemeinen Webservice verwendeten Models haben folgenden Aufbau:

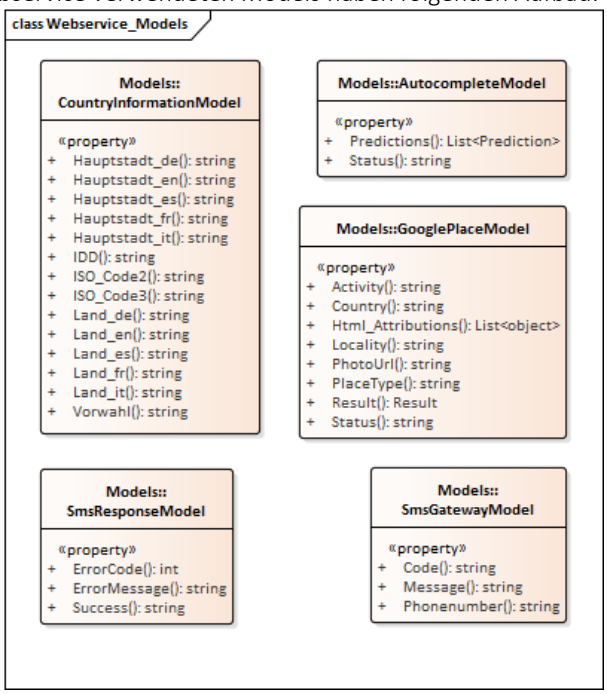


Abbildung 6 - Models Webservice

Das GooglePlaceModel beinhaltet folgende Komponenten:

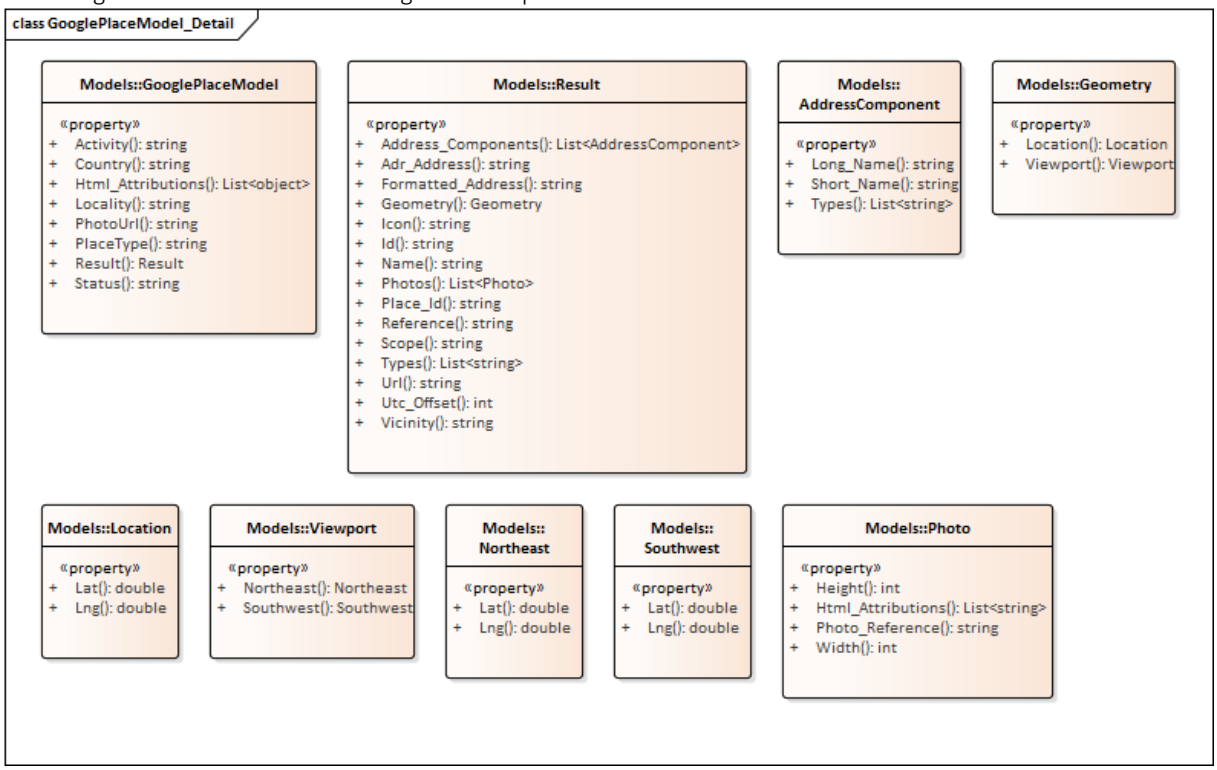


Abbildung 7 - GooglePlace Model

4.2.1.1 SmsGateway

Sendet dem Benutzer ein *SMS* über einen SMS Gateway [4] mit dem Authentifizierungscode.

Aufruf

Methode	URI
GET	/api/SmsGateway

Parameter

Name	Typ	Beschreibung	Optional
phonenumber	String	Telefonnummer des Benutzers	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	SmsResponseModel

4.2.1.2 Authentication

Nachdem man den per *SMS* erhaltenen Code eingegeben hat, wird die Authentication Methode aufgerufen um den Code zu verifizieren.

Aufruf

Methode	URI
GET	/api/Authentication

Parameter

Name	Typ	Beschreibung	Optional
phonenumber	String	Telefonnummer des Benutzers als *Hash* Wert	Nein
code	String	Authentifizierungscode, welcher der Benutzer per *SMS* erhalten hat	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	true/false

4.2.1.3 CountryInformation

Diese Methode liefert anhand des ISO-Ländercodes Informationen zu einem Land. Darin enthalten ist zum Beispiel die Telefonvorwahl, welche für die standardisierte Formatierung der Telefonnummer benötigt wird.

Über die Methode CountryInformation kann die Ländervorwahl des entsprechenden Landes abgerufen werden. Dies wird benötigt, um bei Telefonnummern, welche ohne Vorwahl auf dem Smartphone vom Benutzer abgespeichert wurden, die Vorwahl zu ergänzen. Anhand der Vorwahl (Area Code), welche der Benutzer bei der Registrierung angegeben hat, wird der Wohnort (Land) des Benutzers angenommen. Anhand dieser Information werden alle Telefonnummern, die der Benutzer ohne Vorwahl gespeichert hat, mit der entsprechenden Vorwahl inklusive dem IDD (International direct dialing) ergänzt.

Es gibt bereits *API's* wie z.B. Restcountries [5], welche anhand eines zweistelligen ISO Country Codes die entsprechende Ländervorwahl zurückgeben. Jedoch erhält man da die IDD für die Vorwahl nicht, welche das "+" Zeichen ersetzt (in der Schweiz wird das "+" Zeichen durch die IDD "00" ersetzt). Deshalb wurde eine eigene API implementiert, welche selber verwaltet und erweitert werden kann.

Aufruf

Methode	URI
GET	/api/CountryInformation

Parameter

Name	Typ	Beschreibung	Optional
countryInformation	String	Vorwahl ("+" durch "00" ersetzt)	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	CountryInformationModel

4.2.1.4 CountryInformationAll

Liefert eine Liste aller vorhandenen Informationen zu den hinterlegten Ländern.

Aufruf

Methode	URI
GET	/api/CountryInformationAll

Parameter

Name	Typ	Beschreibung	Optional
-	-	-	-

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	List<CountryInformationModel>

4.2.1.5 Autocomplete

Diese Methode wird bei der Autovervollständigung benötigt. Wird nach einem *Place* gesucht und mindestens drei Zeichen eingegeben wurden, wird diese Methode aufgerufen, um dem Benutzer Vorschläge von *Places* zu liefern.
Diese Methode liefert die Google Places ID, welche für weitere *Places* Abfragen benötigt wird. [6]

Aufruf

Methode	URI
GET	/api/Autocomplete

Parameter

Name	Typ	Beschreibung	Optional
search	String	Zeichenfolge der Suche	Nein
language	String	Sprache in der die Ergebnisse zurück geliefert werden. Wird dieser Parameter weggelassen erhält man das Resultat auf Englisch	Ja
latitude	String	Breitengrad der aktuellen Position. Wird nur bei aktivierter Umkreissuche verwendet	Ja
longitude	String	Längengrad der aktuellen Position. Wird nur bei aktivierter Umkreissuche verwendet.	Ja

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	AutocompleteModel

4.2.1.5.1 Logoanforderung

Google schreibt vor, dass wenn man einen Dienst der Google Place API verwendet und die Daten nicht zusammen mit einer Google Karte anzeigt, muss ein "powered by Google" Logo angezeigt werden. Für das Logo stehen zwei verschiedene Varianten zur Verfügung. Eines für die Verwendung auf einem weissen Hintergrund und eines für die Verwendung auf einem nicht weissen Hintergrund. [7]

Abbildung 8 - Google Logo weisser Hintergrund

Abbildung 9 - Google Logo nicht weisser Hintergrund

4.2.1.6 GooglePlaces

Diese Methode liefert die Google Place Details Information [8] zu einem *Place* und dessen Identifikation. Ebenfalls ist ein Link zu einem Bild von Google Place Photos [9] enthalten, welches in der Applikation angezeigt wird.

Aufruf

Methode	URI
GET	/api/GooglePlaces

Parameter

Name	Typ	Beschreibung	Optional
placeId	String	Google Place ID des gesuchten *Places*	Nein
language	String	Ländercode für Sprache in welcher das Resultat geliefert wird. Wird nichts angegeben erfolgt die Rückgabe in Englisch	Ja

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	GooglePlaceModel

4.2.1.7 WhatsApp

Diese Methode liefert die WhatsApp API [10] *URI*, um einen Benutzer direkt per WhatsApp zu kontaktieren. Sie wurde erstellt, falls sich die WhatsApp *URI* ändern sollte.

Aufruf

Methode	URI
GET	/api/WhatsApp

Parameter

Name	Typ	Beschreibung	Optional
-	-	-	-

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	https://api.whatsapp.com/send?phone=phonenumber&text=

4.2.2 Datenbank Webservice

Dieser Webservice beinhaltet alle Methoden, die im Zusammenhang mit der Datenbank stehen. Alles was in die Datenbank eingetragen und ausgelesen wird, erfolgt über die folgenden Methoden. Dieser Webservice baut ebenfalls auf der REST Architektur auf und basiert auf dem Microsoft Framework ASP.NET. Die Basis *URI* zu diesem Webservice lautet www.api.db.kwix.ch.

Die für diesen Webservice verwendeten Models haben folgenden Aufbau:

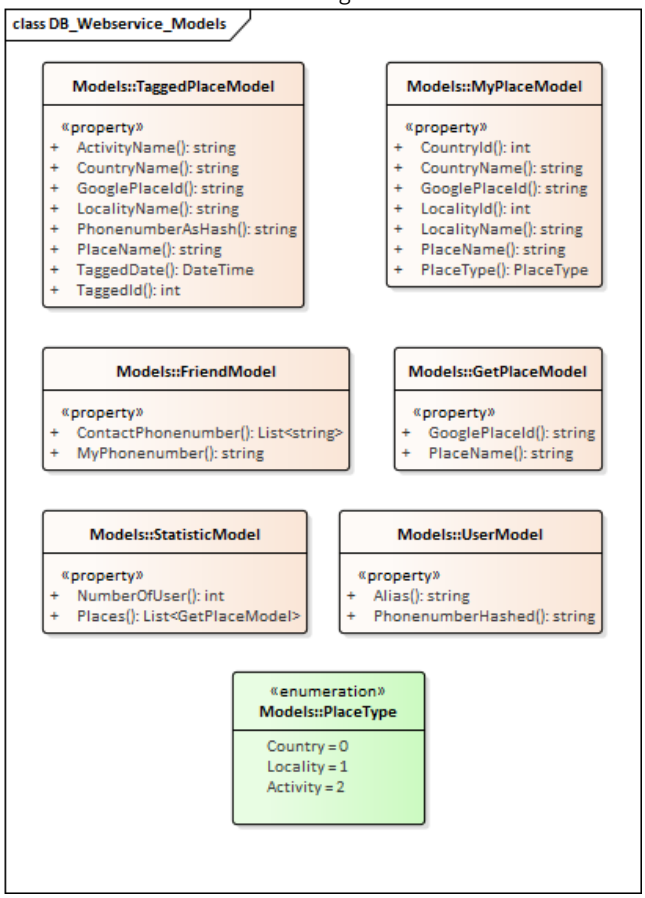


Abbildung 10 - Models Datenbank Webservice

4.2.2.1 DeleteVisitedPlace

Diese Methode wird verwendet um einen besuchten *Place* eines Benutzers zu entfernen.

Aufruf

Methode	URI
DELETE	/api/dp/deleteVisitedPlaceFromUser

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein
googlePlaceId	String	Google Place ID des zu löschenden *Places* aus der Liste der besuchten *Places*	Nein
placeType	PlaceType	Typ des *Places*. Wird z.B. ein Land gelöscht werden auch alle Orte und Aktivitäten von diesem Land aus der Liste der besuchten *Places* entfernt.	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.2 RegUser

Diese Methode fügt einen neu registrierten Benutzer in die Tabelle User ein.

Aufruf

Methode	URI
POST	/api/dp/reguser

Body Parameter: UserModel

Name	Typ	Beschreibung
alias	String	Aliasname, welcher der Benutzer gewählt hat
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.3 GetActivityFromLocality

Diese Methode liefert als Resultat alle Aktivitäten, die zu einem Ort hinterlegt sind.

Aufruf

Methode	URI
GET	/api/dp/getActivityFromLocality

Parameter

Name	Typ	Beschreibung	Optional
googlePlaceId	String	Google Place ID des Ortes zu welchem die Aktivitäten aufgezeigt werden sollen	Nein
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	GetPlaceModel

4.2.2.4 AddUserToPlace

Diese Methode wird verwendet, wenn ein Benutzer einen *Place* als besucht hinzufügt.

Aufruf

Methode	URI
POST	/api/dp/addUserToPlace

Body Parameter: TaggedPlaceModel

Name	Typ	Beschreibung
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert
googlePlaceId	String	Google Place ID des *Places* welcher als besucht hinzugefügt werden soll
countryName	String	Ländername des *Place*
localityName	String	Ortsname des *Place* wenn der *Place* ein Ort oder eine Aktivität ist
activityName	String	Aktivitätsname des *Place* wenn der *Place* eine Aktivität ist
taggedDate	Date	Datum wann der *Place* hinzugefügt wurde

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.5 DeleteUser

Löscht ein *kwix* Benutzer seinen Account, wird mit dieser Methode der Benutzer aus der Datenbank gelöscht. Ebenfalls werden all seine besuchten *Places* und seine Kontakte gelöscht.

Aufruf

Methode	URI
DELETE	/api/dp/deleteUser

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.6 GetPlaceFromCountry

Liefert alle Orte oder Aktivitäten zu einem Land.

Aufruf

Methode	URI
GET	/api/dp/getPlaceFromCountry

Parameter

Name	Typ	Beschreibung	Optional
googlePlaceId	String	Google Place ID von einem Land	Nein
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein
placeType	PlaceType	Angabe ob Orte oder Aktivitäten des Landes zurückgegeben werden sollen.	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	GetPlaceModel

4.2.2.7 GetStatistic

Liefert die Anzahl an *kwix* Benutzer sowie alle *Places*, welche mit *kwix* bereits einmal als besucht oder zur Wunschliste hinzugefügt wurden.

Aufruf

Methode	URI
GET	/api/dp/getStatistic

Parameter

Name	Typ	Beschreibung	Optional
-	-	-	-

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	StatisticModel

4.2.2.8 AddContactToFriend

Überträgt alle Telefonnummern der Kontakte vom Benutzer als *Hash* Wert in die Datenbank. Dies ist notwendig um herauszufinden, welche Benutzer sich gegenseitig sehen können. Es können sich dann nur Benutzer sehen, welche beide voneinander die Telefonnummer gespeichert haben.

Aufruf

Methode	URI
POST	/api/dp/addContactToFriend

Body Parameter: FriendModel

Name	Typ	Beschreibung
myPhonenumber	String	Telefonnummer des Benutzers als *Hash* Wert
contactPhonenumber	Collection of string	Telefonnummer der gespeicherten Kontakte als *Hash* Wert

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.9 AddUserToWishPlace

Diese Methode wird verwendet, wenn ein Benutzer einen *Place* zu seiner Wunschliste hinzufügen will.

Aufruf

Methode	URI
POST	/api/dp/addUserToWishPlace

Body Parameter: TaggedPlaceModel

Name	Typ	Beschreibung
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert
googlePlaceId	String	Google Place ID des *Places* welcher zur Wunschliste hinzugefügt werden soll
countryName	String	Ländername des *Place*
localityName	String	Ortsname des *Place* wenn der *Place* ein Ort oder eine Aktivität ist
activityName	String	Aktivitätsname des *Place* wenn der *Place* eine Aktivität ist
taggedDate	Date	Datum wann der *Place* hinzugefügt wurde

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.10 UpdateUser

Wenn ein Benutzer sein Alias ändert, wird mit dieser Methode die Information in der Datenbank aktualisiert.

Aufruf

Methode	URI
PUT	/api/dp/updateUser

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash*-Wert	Nein
alias	String	Gewählter Alias des Benutzers	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.11 DeleteWishedPlace

Wenn ein *Place* zur Wunschliste hinzugefügt wurde und dieser *Place* aus diese Liste entfernt werden soll, dann wird diese Methode aufgerufen.

Aufruf

Methode	URI
DELETE	/api/dp/deleteWishedPlaceFromUser

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein
googlePlaceId	String	Google Place ID des *Places*, welcher aus der Wunschliste entfernt werden soll.	Nein
placeType	PlaceType	Typ des *Places*. Wird z.B. ein Land gelöscht werden auch alle Orte und Aktivitäten von diesem Land aus der Wunschliste entfernt	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
204	Bearbeitung erfolgreich	Kein Inhalt

4.2.2.12 GetWishedPlaces

Liefert alle *Places*, welche von einem Benutzer zur Wunschliste hinzugefügt wurden.

Aufruf

Methode	URI
GET	/api/dp/getWishedPlaces

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	MyPlaceModel

4.2.2.13 GetPlaces

Liefert alle *Places*, welche von einem Benutzer bereits als besucht hinzugefügt wurden.

Aufruf

Methode	URI
GET	/api/dp/getPlaces

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	MyPlaceModel

4.2.2.14 GetLastVisitedPlace

Auf der Startseite werden die sechs zuletzt besuchten Länder, Orte und Aktivitäten angezeigt. Mit dieser Methode werden diese *Places* ermittelt und zurückgegeben.

Aufruf

Methode	URI
GET	/api/dp/getLastVisitedPlace

Parameter

Name	Typ	Beschreibung	Optional
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein
placeType	PlaceType	Angabe ob die sechs zuletzt besuchten Länder, Orte oder Aktivitäten zurückgegeben werden sollen	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	GetPlaceModel

4.2.2.15 GetFriendFromPlace

Liefert die Kontakte vom Benutzer, welche einen *Place* besucht haben. Wenn bei dem Parameter "googlePlaceId" der String "onlyGetFriends" statt einer Google Place Id übergeben wird, dann wird eine Liste mit allen *kwix* Kontakten vom Benutzer zurückgegeben.

Aufruf

Methode	URI
GET	/api/dp/getFriendFromPlace

Parameter

Name	Typ	Beschreibung	Optional
googlePlaceId	String	Google Place ID eines gesuchten *Places* oder "onlyGetFriends" wenn alle *kwix* Kontakte abgefragt werden sollen	Nein
phonenumberAsHash	String	Telefonnummer des Benutzers als *Hash* Wert	Nein

Rückgabewert

HTTP Code	Beschreibung	Schema
200	Bearbeitung erfolgreich	UserModel

4.3 Wichtige Abläufe

In den folgenden Kapiteln sind zwei wichtige Abläufe dargestellt. Im Kapitel 4.3.1 wird auf das Formatieren der Telefonnummern eingegangen, da es wichtig ist, dass alle gespeicherten Telefonnummern im selben Format hinterlegt werden. In der Applikation können sich nur Personen sehen, welche voneinander die Telefonnummer gespeichert haben, weshalb ein Algorithmus entwickelt wurde, der die Nummern miteinander vergleicht.

Im Kapitel 0 wird aufgezeigt, wie die *Places* in die Kategorien Länder, Orte und Aktivitäten unterteilt werden. Dabei sind die vorgefertigten Informationen der Google Places API [8] von zentraler Bedeutung. Google unterteilt die *Places* ebenfalls bereits in Kategorien, doch dies sind viel mehr als für *kwix* benötigt. So werden einige Google Kategorien zu einer *kwix* Kategorie zusammengefasst.

4.3.1 Format der Telefonnummer

Es ist zwingend notwendig, dass alle in der Datenbank hinterlegten Telefonnummern demselben Format entsprechen. Dies aus dem Grund, damit die Freundschaft zwischen zwei Personen hergestellt werden kann. Die eigene Nummer eines *kwix* Benutzers wird ebenfalls in diesem Format hinterlegt. Das von *kwix* verwendete Format sieht folgendermassen aus:

Abbildung 11 - Format der Telefonnummer

Hat eine gespeicherte Telefonnummer keine Vorwahl (*IDD* und Landesvorwahl), so wird diese zu der Telefonnummer hinzugefügt. In diesem Vorgang wird das "+" Zeichen durch die *IDD* ersetzt. Es werden auch Formatierungszeichen wie Klammern oder Leerzeichen entfernt. Nachfolgend einige Beispiele:

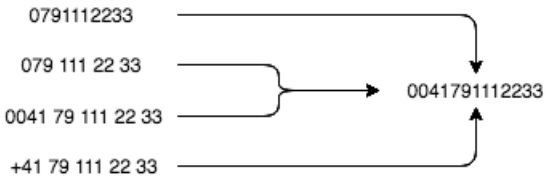


Abbildung 12 - Beispiele Telefonnummern

Vor dem übertragen in die Datenbank wird die Nummer durch eine *Hashfunktion* (SHA-256) verschlüsselt. So wird sichergestellt, dass die hinterlegten Nummern nicht durch einen Angriff missbraucht werden können.

Wird aus der Applikation heraus ein Freund kontaktiert, wird die im Telefon hinterlegte Nummer verwendet. Dadurch ist es bei einer Kontaktierung möglich einen z.B. schon vorhandenen WhatsApp Chat mit dieser Person weiterzuführen.

4.3.2 Einteilung der *Places*

Für die Suche von *Places* wird die Google Place API [8] verwendet. Von dieser Datenquelle werden verschiedene Informationen über den gesuchten *Place* zur Verfügung gestellt. Anhand dieser Informationen wird ermittelt, ob es sich um ein Land, einen Ort oder eine Aktivität handelt. Suchergebnisse welche diese Kriterien nicht erfüllen werden herausgefiltert und nicht dargestellt.
Google Places kategorisiert die Suchabfragen ebenfalls. Die Google Kategorien werden als Array von "types[]" von der Google Places API zurückgegeben.
Anhand dieses "types" kann unterschieden werden, um was für ein "Place Typ" es sich handelt damit die richtige *Place* Seite angezeigt werden kann.

types[] enthält	kwix Kategorie
country	Land (*Country*)
locality	Ort (*Locality*)
restliche types ausschliesslich der Ausnahmen (siehe Beschreibung unten)	Aktivität (*Activity*)

Die restlichen *Places*, die gefunden werden und oben genannten types nicht enthalten, werden der *kwix*-Kategorie Aktivitäten zugewiesen. Dabei gibt es noch folgende Ausnahmen:

- Enthält types das Schlüsselwort "route", "transit_station" oder "colloquial_area" wird das Ergebnis nicht in den Suchresultaten erscheinen.
- Enthält types das Schlüsselwort "administrative_area_level_1" oder "political" wird es ebenfalls ausgeschlossen, da es sich dabei um ein Bundesland, Bundesstaat oder einen Kanton handelt.

Beispiel einer Country API Antwort (Schweiz):


```
  {
    "description": "Schweiz",
    "id": "70fc199988a0f4fd1b474a8a7f74d66c6249f182",
    "matched_substrings": [
      {
        "length": 7,
        "offset": 0
      }
    ],
    "place_id": "ChIJYW1Zb-9kjEcRFXvLDxG1Vlw",
    "reference": "ClRJAAAAkYB3DqvoO8JvrcY7S6upp4zQKIJ39Xll07jQch5i1rulyQL4LR3-Lcvv04SQjrnfVRrZTv9XZMHjoInpLam05K5Q
                  0rgl_bTcO8OkCCPd57ASEFlTcwdJKXZIQraGKAPdHXEaFIlCOCj7g-_sMwC2ty6JT1UmUmvt",
    "structured_formatting": {
      "main_text": "Schweiz",
      "main_text_matched_substrings": [
        {
          "length": 7,
          "offset": 0
        }
      ],
      "secondary_text": null
    },
    "terms": [
      {
        "offset": 0,
        "value": "Schweiz"
      }
    ],
    "types": [
      "country",
      "political",
      "geocode"
    ]
  },
```


Erkennung des "types" anhand der übermittelten Parameter

4.4 Klassendiagramm

Dieser Abschnitt geht hauptsächlich auf die Klassen der ViewModels ein. Es wird aufgezeigt mit welchen Views die ViewModels in Verbindung stehen und welche Models dazu verwendet werden.

4.4.1 BaseViewModel

Alle verwendeten ViewModels ausser das SearchRxViewModel erben vom BaseViewModel. Dieses beinhaltet allgemeine Komponenten, wie zum Beispiel die Anzeige, ob eine Ansicht für den Benutzer gesperrt werden muss. Ebenfalls in dieser Klasse ist der PropertyChangedEventHandler implementiert, welcher dafür zuständig ist, dass die ViewModels eine Eingabe vom Benutzer mitbekommen und die Views aktualisiert werden.

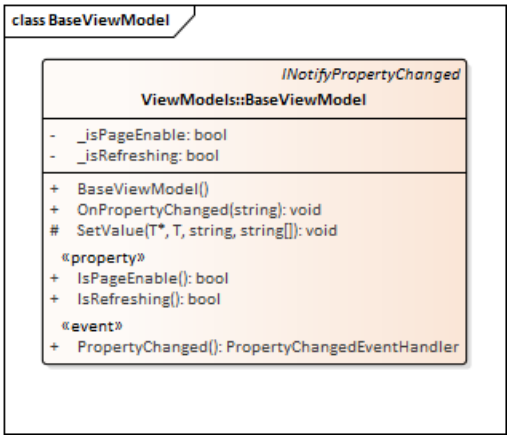


Abbildung 13 - Base ViewModel

4.4.2 About Page

Auf der About Page werden Informationen zu der Applikation angezeigt. Die Page beinhaltet folgende Informationen:

- Mail an info@kwix.ch um bei Fragen oder Problemen die Entwickler zu kontaktieren
- kwix.ch Webauftrit von *kwix*
- Nutzungsbedingungen von *kwix*
- Verwendete Opensource Lizenzen

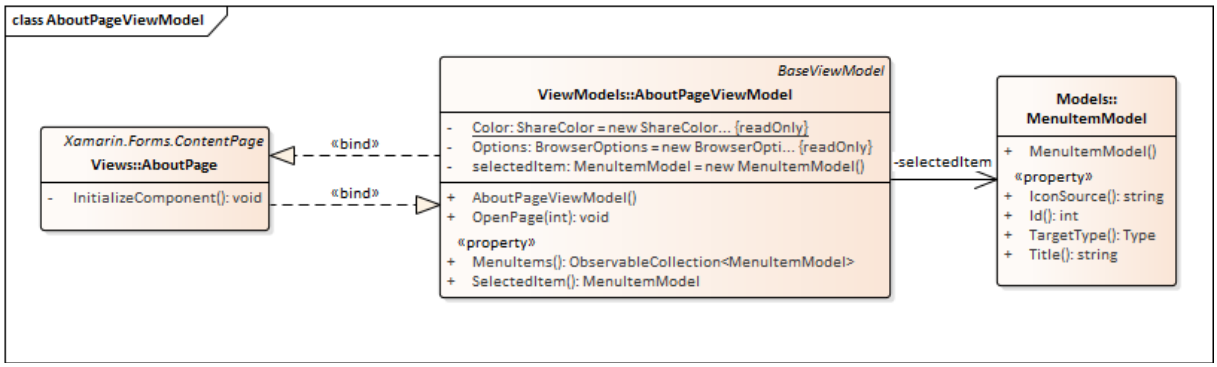


Abbildung 14 - About Page ViewModel

4.4.3 Admin Page

Die Admin Page ist nur im Debug Modus für die Entwickler sichtbar. Darauf können Nutzungsinformationen über die App ausgelesen werden. So wird auf dieser Seite angezeigt, wie viele Nutzer momentan bei *kwix* registriert sind und wie viele *Places* bereits in der Datenbank erfasst wurden. Ebenfalls werden alle erfassten *Places* in einer Liste dargestellt.
Um die erfassten *Places* und die Anzahl an Benutzer darzustellen wird der DbServiceGet verwendet.

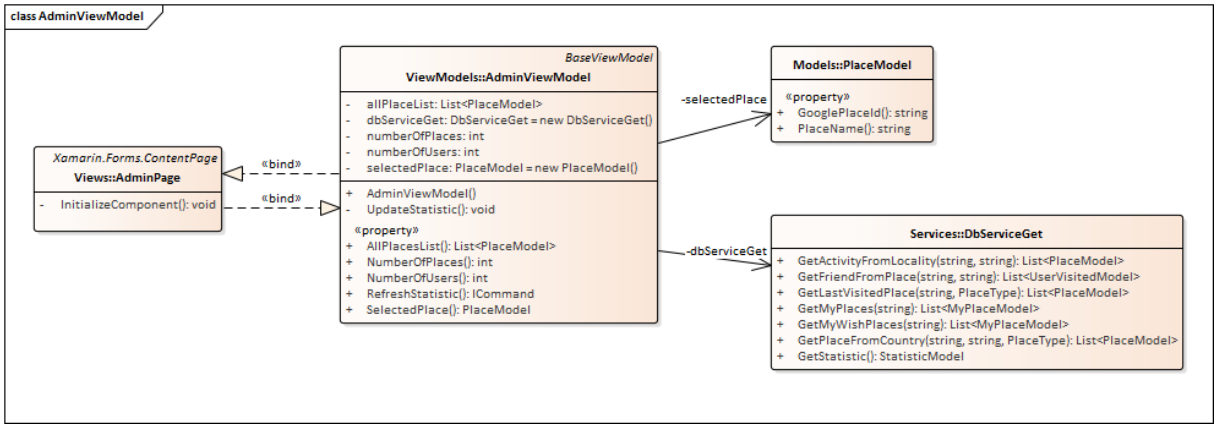


Abbildung 15 - Admin Page ViewModel

4.4.4 Contact Page

In der Kontakt Seite werden alle *kwix* Kontakte des Benutzers angezeigt. Im ViewModel sind die Methoden zu finden, welche benötigt werden um alle Nummern auf dasselbe Format zu formatieren. Ebenfalls wird darin die Methode aufgerufen um die Verbindung zwischen Freunden herzustellen. Diese Methode wird bei jedem Neustart der Applikation angestossen. Es kann sei, dass in der Zwischenzeit ein neuer Kontakt hinzugekommen oder gelöscht wurde.

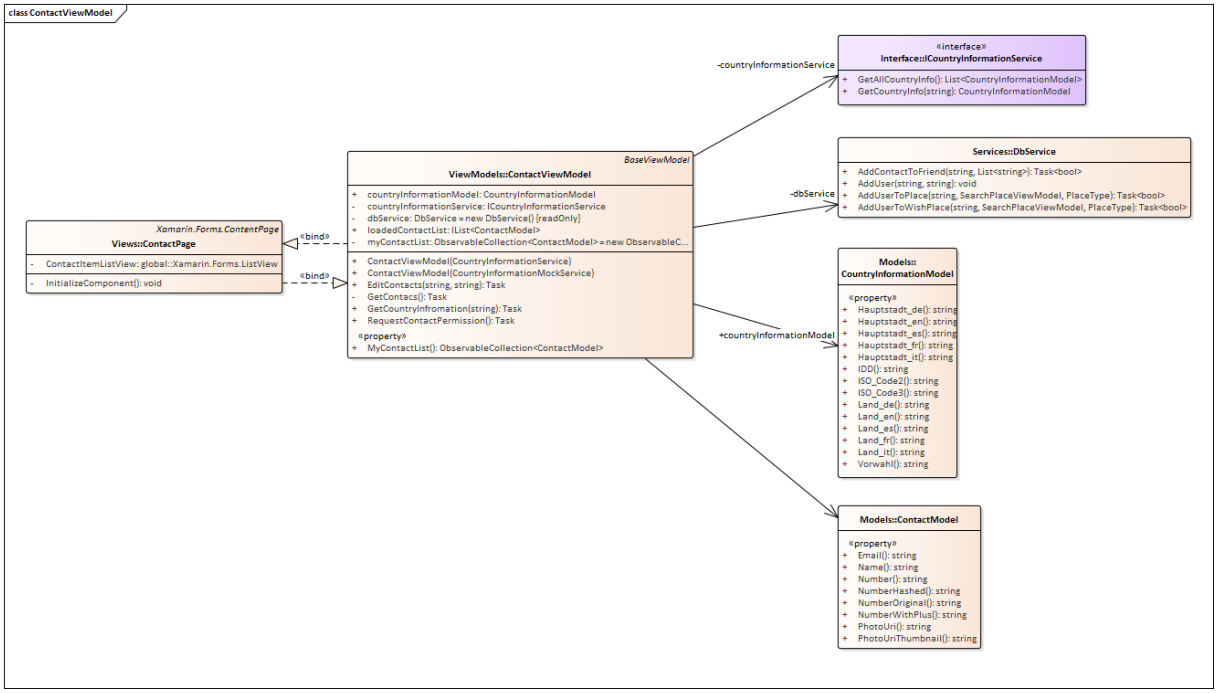


Abbildung 16 - ViewModel Contact Page

4.4.5 Seiten Besucht / Wunschliste

Die beiden Seiten Besucht und Wunschliste sind fast identisch. Der Unterschied besteht darin, dass eine Seite *Places* enthaltet, welche vom Benutzer bereits besucht wurden und die Wunschliste nur *Places*, welche vom Benutzer mit dem Herz, sprich als Wunschdestination vermerkt wurden.
Die Seiten sind in drei Tabs unterteilt, welche wiederum Länder, Orte und Aktivitäten entsprechen. Die Listen werden alphabetisch sortiert und nach Ländern gruppiert. Bei den Aktivitäten wird ebenfalls noch der Ort angezeigt, in welchem sich die Aktivität befindet.
Das ViewModel beinhaltet die einzelnen Datenbankabfragen zu der Visited Tabelle sowie zu der Wished Tabelle. In diesen Tabellen sind die Informationen hinterlegt, welche Benutzer welche *Places* besucht oder als Wunsch eingetragen haben.
Über diese Listen ist es auch möglich besuchte *Places* und Wunschdestinationen wieder zu löschen.
Da die meisten Methoden für Besucht und Wunsch identisch sind, wird den Methoden einen Enumeratorwert mitgegeben um zu unterscheiden ob es sich um ein Besucht- oder Wunschplace handelt. So konnte einiges an Code gespart werden und das DRY-Prinzip (*don't repeat yourself*) konnte auch eingehalten werden.

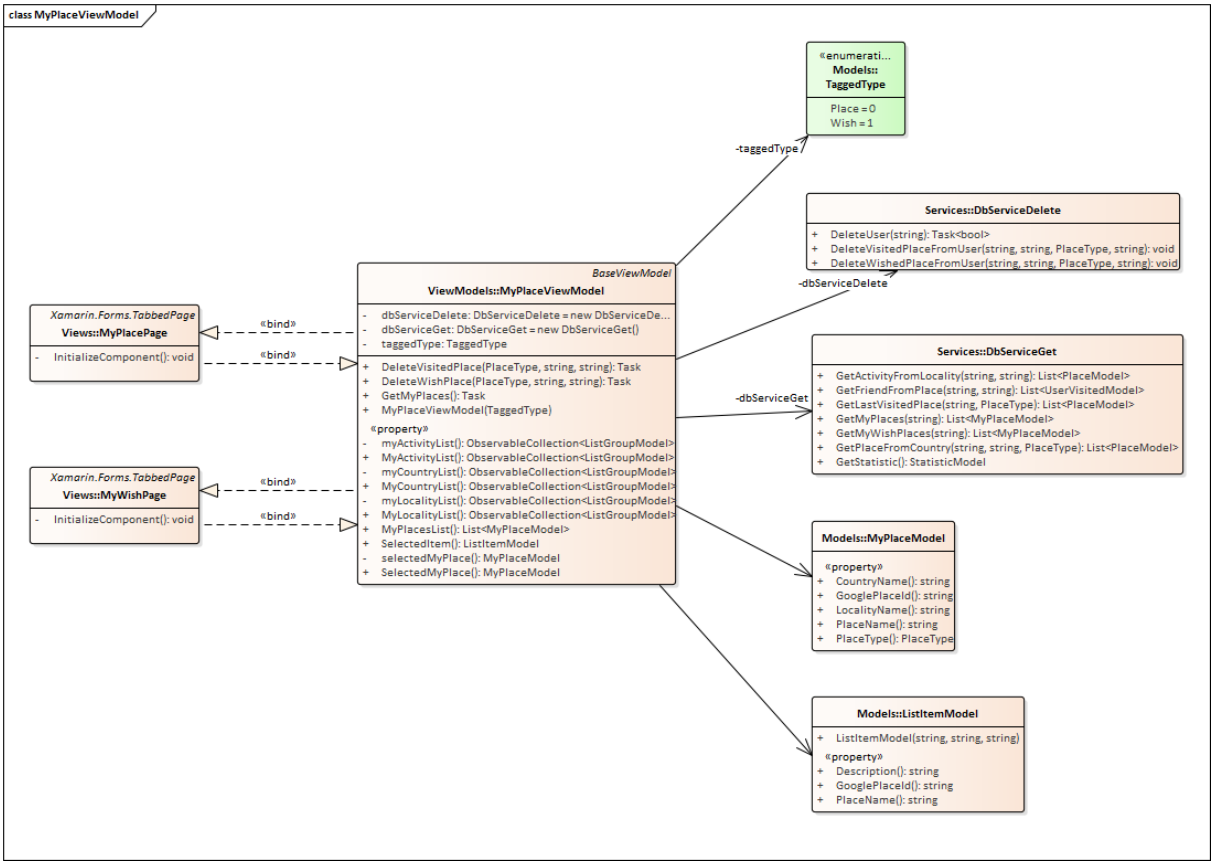


Abbildung 17 - ViewModel MyPlace Page

4.4.6 SearchPlaceViewModel

Das SearchPlaceViewModel ist das mächtigste ViewModel. Es ist für die meisten Seiten und Inhalte zuständig. Im SearchPlaceViewModel sind alle Seiten beinhaltet, welche Listen von *Places* darstellen.

So sind die Seiten CountryPage, LocalityPage und ActivityPage mit dem SearchPlaceViewModel *gebindet*. Auf diese Seiten gelangt man, wenn über die Startseiten ein zuletzt besuchten *Place* angewählt wird oder wenn nach einem *Place* über die SearchPage gesucht wird. Der Unterschied dieser Seiten zeigt sich in der Anzahl an Tabs. Die CountryPage beinhaltet die Tabs Personen, Orte und Aktivitäten. Die LocalityPage die Tabs Personen und Aktivitäten und die LocalityPage nur Personen.

Über diese Seiten können direkt die Freunde kontaktiert werden. Ebenfalls beinhalten diese Seiten einen Infobutton über welchen unter anderem eine Google Suche zum besagten *Place* gestartet werden kann. Beim betätigen des Infobuttons öffnet sich ein *Pop-Up* Fenster, welches als InfoPopUp View separat implementiert wurde. Wird bei einer Aktivität der Infobutton betätigt, wird ebenfalls das Land und der Ort angezeigt in welchem sich diese Aktivität befindet. Auch wird ein langer Name einer Aktivität hier komplett dargestellt. Auf den *Places* Seiten werden zu lange Namen gekürzt.

Die StartPage erhält die Informationen ebenfalls vom SearchPlaceViewModel und beinhaltet drei Tabs. Auf diesen drei Tabs sind die sechs zuletzt besuchten Länder, Orte und Aktivitäten zu finden. Es kann direkt über die StartPage ein *Place* als besucht hinzugefügt oder direkt in die Wunschliste eingetragen werden.

Das SeachPlaceViewModel ist mit den Datenbank Services (DbServices und DbServicesGet) verbunden. Durch diese Services werden die Informationen zu den *Places* aus der Datenbank herausgelesen. Zum Beispiel wenn nach einem Land gesucht wird, erhält der Benutzer ebenfalls eine Liste aller Orte und Aktivitäten welche ebenfalls zu diesem Land gehören.

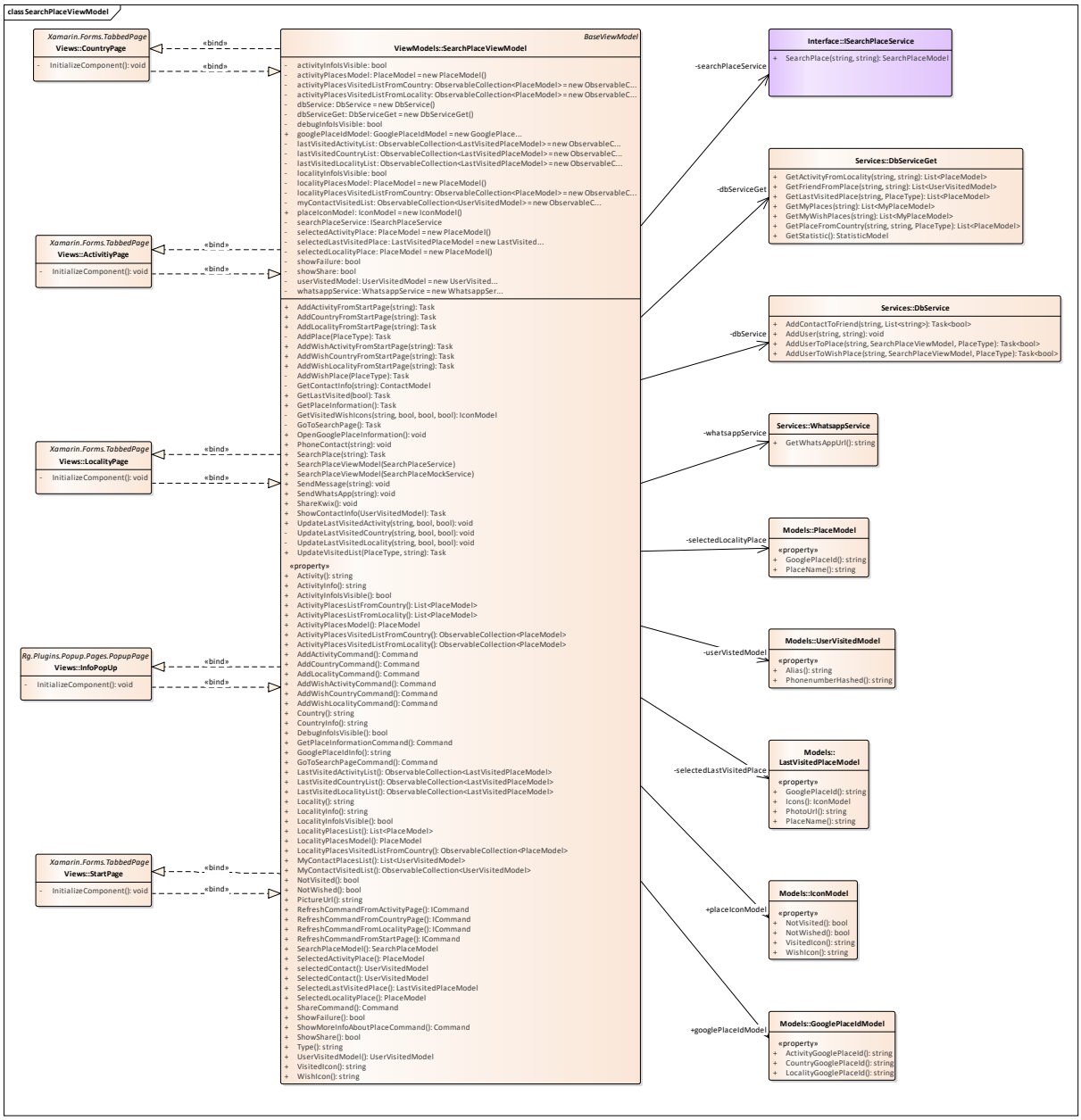


Abbildung 18 - SearchPlace ViewModel

4.4.7 Search Page

Das SearchRxViewModel ist das Einzige, welches nicht vom BaseViewModel ableitet. Für dieses ViewModel wurde das ReactiveUI Framework [11] verwendet. Das aus dem Grund, um die Eingabe der Suche zu verzögern. Wird nach einem *Place* gesucht, wird die Eingabe um 750ms verzögert und dann an den Autocomplete Service (siehe Kapitel 4.2.1.5 Autocomplete) gesendet. Der Webservice wird aber erst ab einer Eingabe von drei Zeichen aufgerufen. So können die Abfragen auf die Google Places API reduziert werden. Die Zeit von 750ms ist so hoch gewählt, da die Eingabe über eine Smartphone Tastatur eher länger dauert und dadurch nicht bei jeder Eingabe von einem Zeichen eine API Abfrage ausgeführt wird.
Weil diese Klasse nicht von BaseViewModel ableitet, mussten ebenfalls die Informationen ob eine Seite gesperrt oder der Inhalt neu geladen wird nochmals implementiert werden.
Auf der SearchPage ist das Eingabefeld für eine *Places* Suche zu finden. Die Suchergebnisse werden dann unterhalb in einer Liste dargestellt. Auch kann über diese Seite die Umkreissuche aktiviert werden. Bei aktiver Umkreissuche werden dem Benutzer zuerst die Ergebnisse in der Nähe vom aktuellen Standort angezeigt.

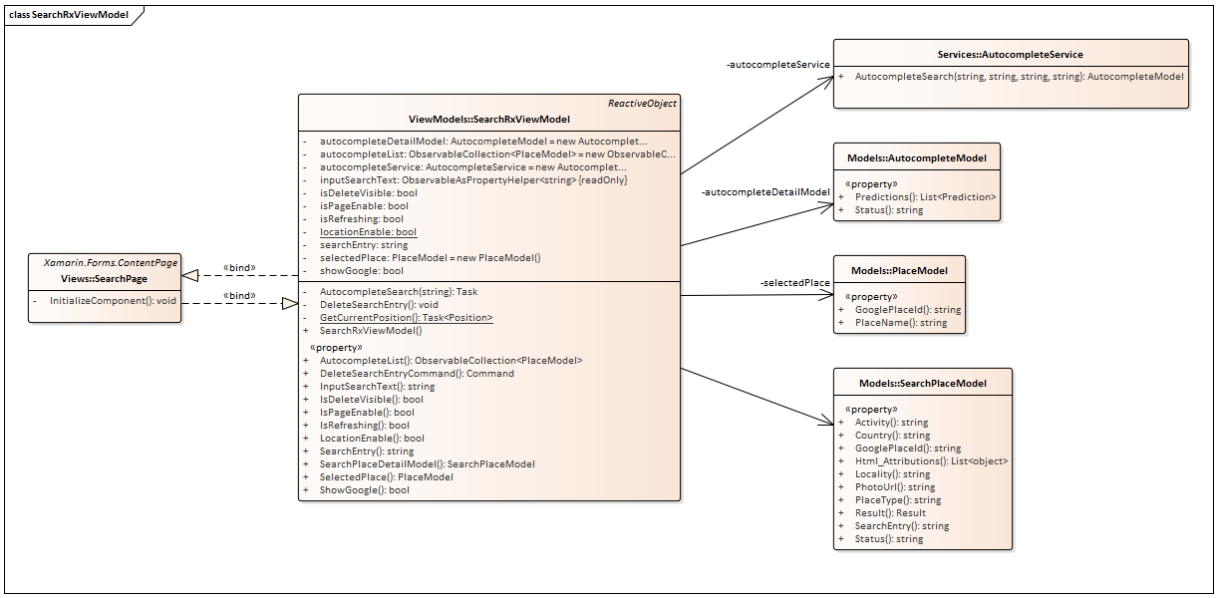


Abbildung 19 - Search Page View Model

4.4.8 SplashScreen

Der SplashScreen dient dazu, dem Benutzer anzuzeigen was beim Aufstarten der App im Hintergrund geladen wird. So kann dem Benutzer signalisiert werden, dass etwas im Gange ist und die App nicht einfach wieder beendet wird, bevor alle Inhalte geladen wurden.
Es werden die Kontakte mit der Datenbank abgeglichen, eventuell neu hinzugekommene Kontakte in die Datenbank übertragen oder gelöschte entfernt. Danach werden die zuletzt besuchten *Places* abgefragt und für die Darstellung vorbereitet.
Das SplashScreenViewModel benötigt für diese Anzeige kein Model im Hintergrund.

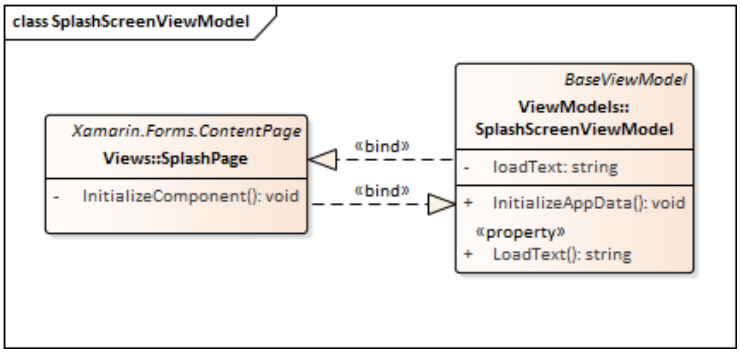


Abbildung 20 - SplashScreen ViewModel

4.4.9 UserViewModel

Das UserViewModel ist für verschiedene Seiten zuständig. Die Seite um die Benutzerinformationen festzulegen, sowie die Seite um den erhaltenen *SMS* Code einzugeben, werden darüber gesteuert.

Im Menü kann die Seite Profil aufgerufen werden, wo die eigene hinterlegte Telefonnummer angezeigt wird. Auch kann hier der gewählte Benutzername angepasst werden. Der Benutzername wird für die Anzeige auf den *Places* Seiten verwendet. Auf diesen Seiten wird nur der *kwix* Benutzername angezeigt. Wird jedoch auf einen Benutzer geklickt erscheint der Name und die Telefonnummer so wie sie im Telefonbuch hinterlegt sind. Auf der Profil Seite besteht die Möglichkeit den Benutzeraccount zu löschen. Dabei werden alle Daten des Benutzers aus den verschiedenen Datenbanktabellen entfernt.

Die Seite Einstellungen ist nur im Debug Modus verfügbar. Auf dieser Seite wurden während der Implementation diverse Sachen ausprobiert und diente mehr als Spielwiese. Momentan wird hier nur noch die eingestellte Sprache vom Gerät angezeigt.

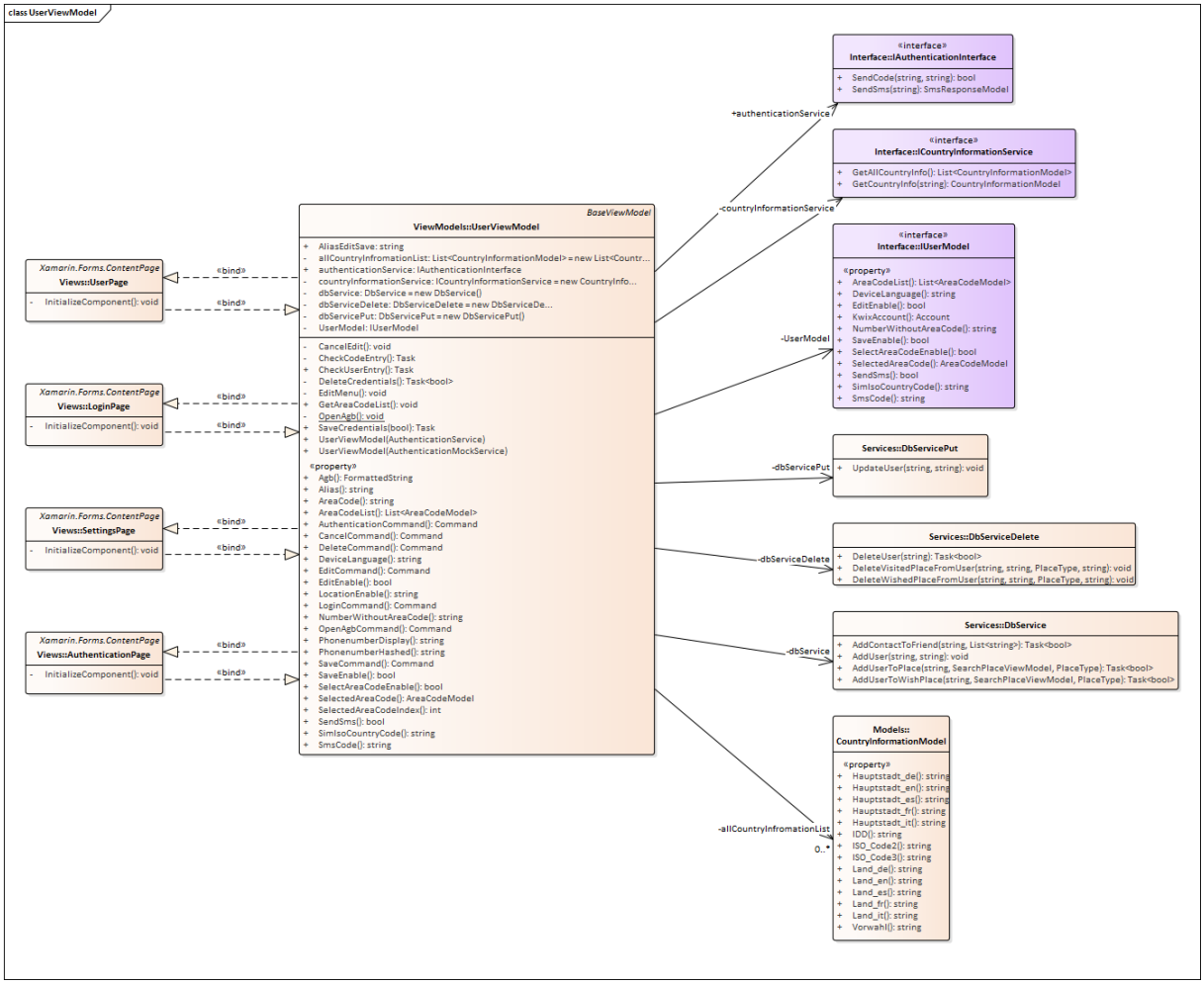


Abbildung 21 - User ViewModel

4.4.10 Datenbank

Die Datenbank, welche für *kwix* verwendet wird, besteht aus fünf Tabellen.

Die Tabelle Place enthält alle *Places*, welche über *kwix* als besucht oder in die Wunschliste eingetragen wurden. Wurde ein *Place* nur von einem Benutzer besucht und dieser Benutzer löscht diesen Eintrag wieder, bleibt der *Place* jedoch weiterhin in der Tabelle Place bestehen. Die Tabelle enthält die Spalten PlaceID (Primary Key), PlaceType, CountryID, LocalityID, PlaceName und die Google Place ID. Wird eine Aktivität hinzugefügt wird ebenfalls der Ort und das Land, in dem sich die Aktivität befindet, hinzugefügt. Bei einem Land bleiben die Spalten CountryID und LocalityID immer leer. Bei einem Ort wird in der Spalte CountryID die PlaceID eingetragen von dem Land in dem sich der Ort befindet. Die Spalte LocalityID bleibt bei einem Ort leer. Bei einer Aktivität wird in der Spalte CountryID die PlaceID des Landes und bei LocalityID die PlaceID des Ortes eingetragen. Zu jedem *Place* wird die GooglePlaceID eingetragen. So können bei einer späteren Abfrage Google Credits (siehe Kapitel 10.2 Google API) gespart werden, da die GooglePlaceID nicht nochmals abgefragt werden muss.
Die Tabelle User enthält die Daten zu einem *kwix* Benutzer. Darin hinterlegt sind die Telefonnummer und der gewählte Alias des Benutzers. Die Telefonnummer ist dabei als *Hashwert* hinterlegt um vor Angriffen geschützt zu sein.

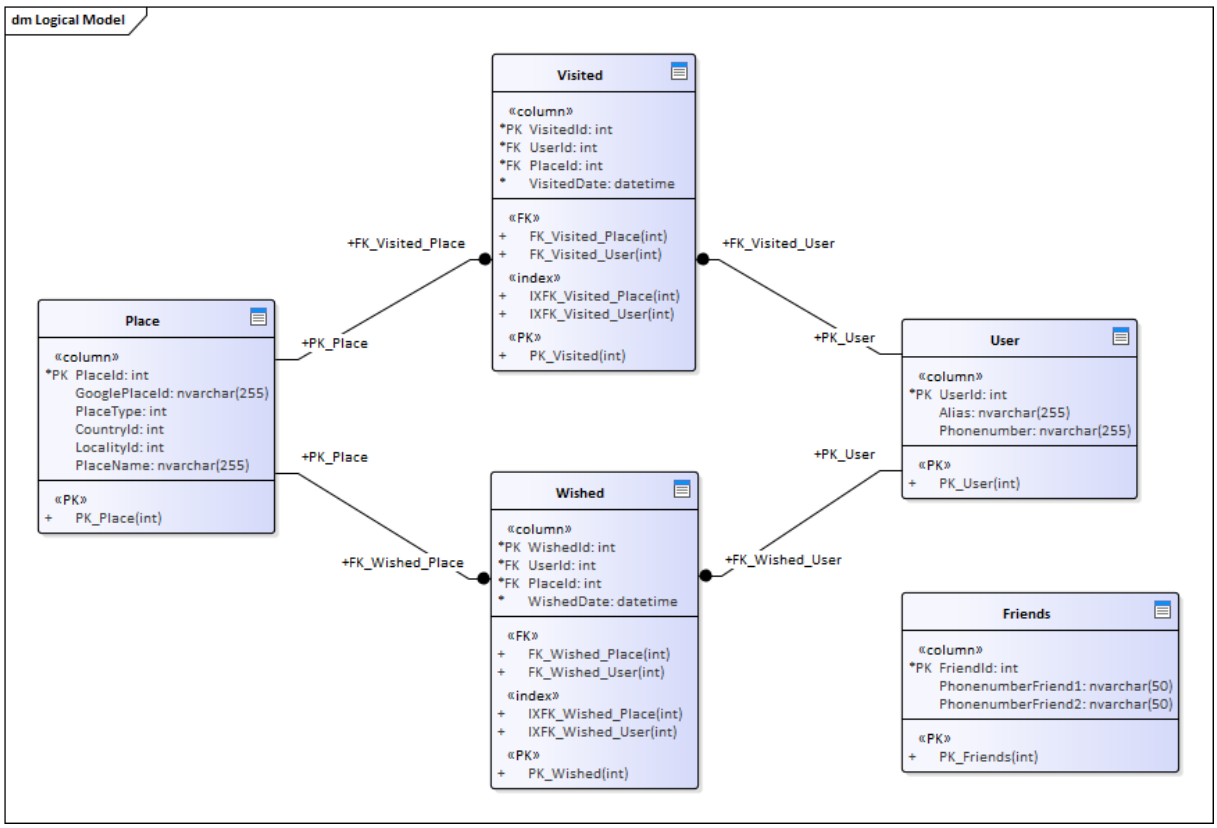
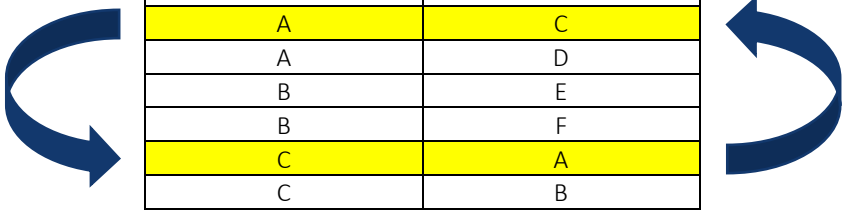


Abbildung 22 - Übersicht Datenbank Tabellen

Die Tabellen Visited und Wished sind identisch aufgebaut. In der Visited Tabelle werden die besuchten *Places* der Benutzer eingetragen. In der Wished Tabelle die *Places*, welche von den Benutzern auf die Wunschliste gesetzt wurden. Beide Tabellen haben eine Fremdschlüsselbeziehung zu der Place Tabelle sowie zu der User Tabelle. Wird ein *Place* von einem Benutzer besucht wird hier die PlaceID sowie die UserID eingetragen. Ebenfalls wird das Datum und die Uhrzeit erfasst zu dem Zeitpunkt als der *Place* hinzugefügt wurde.

Die Friends Tabelle enthält die Telefonnummern der Benutzer sowie die Nummern deren Kontakte. Alle Nummern sind dabei als *Hashwert* hinterlegt um die Daten zu schützen. Diese Tabelle wird benötigt um die Freundschaften der einzelnen Benutzer zu identifizieren. Dabei muss der Inhalt einer Kombination von PhonenumberFriend1 und PhonenumberFriend2 demselben Inhalt entsprechen wie PhonenumberFriend2 und PhonenumberFriend1. Ist das der Fall haben beide Benutzer gegenseitig die Nummer voneinander und sie können sehen, wo der andere Benutzer bereits war.

Im folgenden Beispiel sind nur die Kontakte A und C befreundet, da nur diese gegenseitig voneinander die Nummer haben. A hat zwar die Nummer von B gespeichert aber B nicht die Nummer von A. Daher sieht A keine Einträge B.



PhonenumberFriend1	PhonenumberFriend2
A	B
A	C
A	D
B	E
B	F
C	A
C	B

Abbildung 23 - Darstellung Friend Erkennung

Die Tabelle Friends wird bei jedem Aufstarten der App aktualisiert, sofern neue Nummern dazugekommen oder Nummern gelöscht wurden.
Wird in der App eine Abfrage zu einem *Place* ausgeführt, wird direkt in dieser Tabelle nachgeschlagen, welche Kontakte dem Benutzer angezeigt werden.

5 Prozesse und Threads

Um eine höchstmögliche Performance der Applikation zu erreichen werden parallele Prozesse benötigt. In C# gibt es hierzu die Parallel Invokes welche diese Aufgabe übernehmen. Als Beispiel einer solchen Anwendung wird das Laden der zuletzt besuchten Places aufgezeigt.

```
Parallel.Invoke(() => lastVisitedCountryDbList = dbServiceGet.GetLastVisitedPlace(App.userViewModel.PhonenumberHashed, PlaceType.Country),
                () => lastVisitedLocalityDbList = dbServiceGet.GetLastVisitedPlace(App.userViewModel.PhonenumberHashed, PlaceType.Locality),
                () => lastVisitedActivityDbList = dbServiceGet.GetLastVisitedPlace(App.userViewModel.PhonenumberHashed, PlaceType.Activity));
```

Abbildung 24 - Beispiel Parellel Invokes

Um die Applikation möglichst fliessend und benutzerfreundlich zu gestalten, wurde darauf geachtet, wo möglich, asynchrone Tasks zu verwenden. Da es in der App zu repetitiven Funktionsaufrufen kommen kann (mehrfaches Hinzufügen von *Places*) dürfen diese Aufrufe nicht «geschützt» werden.

6 Deployment und verwendete Technologien

Das Deployment wurde zu Beginn der Arbeit komplett im *VSTS* umgesetzt. Mit der Zeit hat sich das App Center für das *Builden* und *Publishen* durchgesetzt. Das App Center hat noch weitere Möglichkeiten, welche im separaten Kapitel 6.2 App Center beschrieben sind.
Die Webservices werden momentan nicht automatisch *deployed* und müssen bei einer Anpassung manuell *published* werden.
Da Xamarin von Microsoft übernommen wurde, wurde bei der Auswahl der verwendeten Technologien auf Microsoft Produkte gesetzt. So wurde *VSTS* und Azure verwendet, da diese optimal mit Visual Studio interagieren.

6.1.1 Deployment Android

Jeder Master Branch, welcher auf dem *VSTS gepusht* wird, wird vom App Center als *Release* inklusive der Signatur (Keystore file) *gebuildet*. Als eindeutige Buildnummer wird das Datum in Sekunden (Unixtime) verwendet. Nach erfolgreichem *Build* wird ein Downloadlink zur App automatisch per Mail an die Entwickler von *kwix* verteilt, wo die App heruntergeladen und auf einem Smartphone installiert werden kann.

6.1.2 Deployment iOS

Für iOS geschieht das Deployment nach dem gleichen Prinzip über das App Center wie für Android. Bei der Signatur für iOS wird ein sogenanntes Provisioning Profile sowie ein P12 Zertifikat benötigt. Für diese beiden Signaturen wird ein Developer Account von Apple benötigt, den *kwix* besitzt.

6.1.3 Tests

Die ViewModels werden, wo es sinnvoll ist, mit UnitTest getestet. Die beiden Webservice werden bei jeder Änderung manuell durch den Entwickler getestet und nicht mit UnitTests, da sonst kostenpflichtige API Abfragen mitgetestet werden.

Jeder Entwickler hat ein Android Smartphone, mit dem er während der Entwicklung die Android App debuggen und testen kann. Im Visual Studio steht auch ein Android Emulator zur Verfügung, jedoch ist das debuggen auf einem echten Android Gerät um einiges schneller.
Zwei Entwickler besitzen zudem einen MAC auf dem sie per iPhone Emulator die iOS App debuggen und testen können. Ebenfalls wurde die App auf einem iPhone getestet.

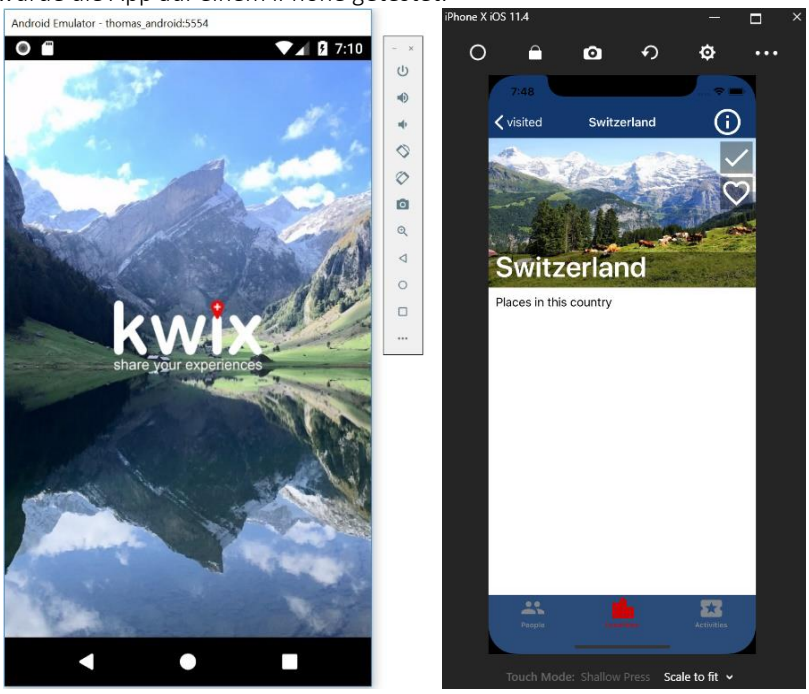


Abbildung 25 - Android und iPhone Emulatoren

6.2 App Center

Das App Center [12] entstand aus der Übernahme von HockeyApp durch Microsoft. Mit dem App Center ist es möglich eine App auf echten Geräten zu *builden*, zu testen und den anschliessenden *Release* zu *deployen*. Das Verteilen geschieht entweder per Mail an eine Gruppe oder direkt über einen Store (Google Play Store oder iOS App Store). Zudem bietet das App Center die Möglichkeit eine App zu überwachen und zu analysieren.

Bei *kwix* werden folgende Funktionen vom App Center für Diagnose- und Analyse-Zwecke verwendet:

Crashes
Alle Crashes, welche die *kwix* App hat, werden an das App Center übermittelt.

Exceptions
Jede Exception, die in *kwix* generiert wird, wird im App Center aufgelistet.

Beispiel für die Verwendung in einer Exception:

```
try{ }
catch (Exception ex)
{
   Crashes.TrackError(ex);
}
```

Events
Mit den Events werden Fehler, welche nicht durch Exceptions behandelt werden, aufgezeichnet. Es können bei einem Event bis zu 20 "Properties" mitgegeben werden. Dies dient der besseren Übersicht der Events.
Bei *kwix* werden bei jedem Event mindestens die beiden Properties "Category" und "Message" verwendet.
Die bei *kwix* verwendeten Kategorien werden im Kapitel 6.2.2 kwix Kategorien beschrieben. Als Message dient eine kurze Beschreibung des Events. Die Events, welche bei *kwix* verwendet werden, sind im Kapitel 6.2.1 kwix Events beschrieben.

Beispiel eines Events, wie er bei *kwix* verwendet wird:

```
Analytics.TrackEvent("Note", new Dictionary<string, string>
{
   { "Category", "UserAction" },
   { "Message", "User do not allow access to contacts." }
});
```

6.2.1 kwix Events

Folgende Events werden in der *kwix* App zum Tracken verwendet. Die Daten sind anonymisiert und somit nicht nachverfolgbar.

Note	Hinweise, welche zur Beeinträchtigung der App führen, jedoch die App nicht zum Absturz bringen. z.B. Zugriff auf Kontakte verweigern
Error	Fehler, die bei einer Abfrage von einem Webservice zustande gekommen sind
UserAction	Hinweise, wie der Nutzer die App verwendet bzw. durch die App navigiert

6.2.2 kwix Kategorien

Folgende Kategorien werden in der *kwix* App verwendet:

UserEntry	Eingaben vom Benutzer
DeviceInfo	Infos über das verwendete Smartphone
Webservice	Fehler bei einer Webservice Abfrage
DbWebservice	Fehler bei einer DB-Webservice Abfrage

7 Datenspeicherung

Die Vor- und Nachteile einer relationalen Datenbank im Vergleich zu einer Dokumenten orientierten Datenbank in Bezug auf die Applikation waren für uns sehr schwierig einzuschätzen da hier die Erfahrungswerte fehlten. Während des Studiums wurde ausschliesslich mit relationalen Datenbanken gearbeitet und diese daher vertraut waren, haben wir uns für eine relationale Datenbank entschieden.

7.1 Aufbau Azure Cloud Service

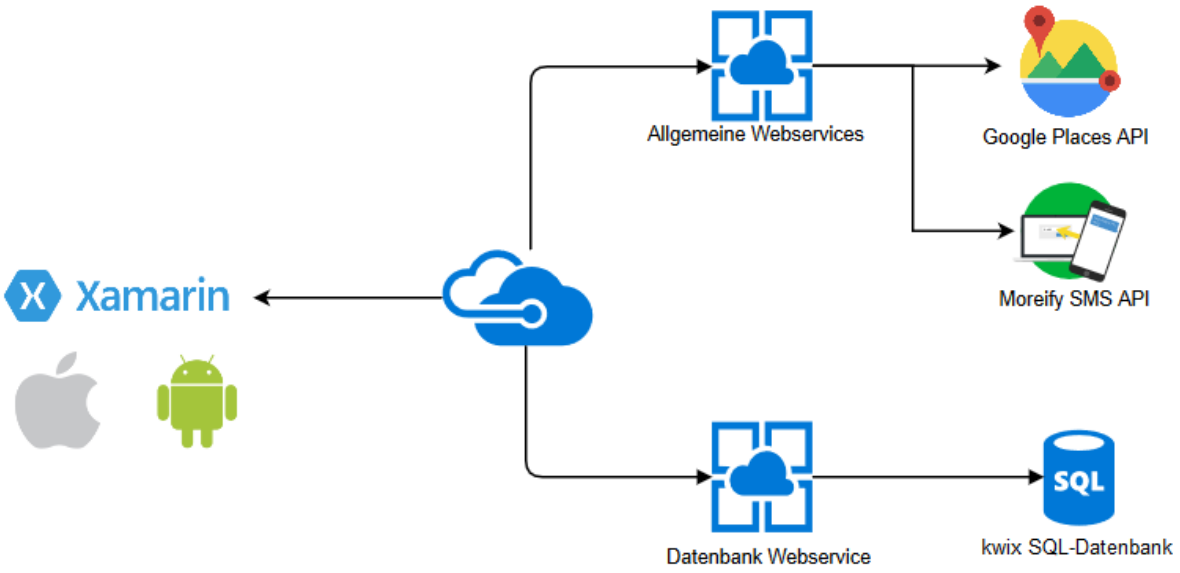


Abbildung 26 - Aufbau Azure Cloud Service

7.2 Datenschutz

Seit dem 25. Mai 2018 gibt es in der Europäischen Union eine neue Datenschutzverordnung [13]. Diese gilt auch für Unternehmen in der Schweiz, welche personenbezogene Daten von natürlichen Personen verarbeiten, welche sich in der Europäischen Union befinden. Da *kwix* in diesen Bereich gelangt wurde von Anfang an darauf geachtet und unter www.kwix.ch/agb eine entsprechende Nutzungsbedingung der Applikation erstellt. Bei der Erstregistrierung von *kwix* muss man diesen Nutzungsbedingungen einmalig zustimmen um *kwix* nutzen zu können.

7.3 Datensicherheit

Der sichere Umgang mit sensiblen Daten hat bei *kwix* oberste Priorität. Daher wird die von den Nutzern eingegebene Telefonnummer in der Applikation mit einem *Hashverfahren* (SHA256) verschlüsselt. Auch die Kontaktinformationen welche benötigt werden um festzustellen, wer mit wem befreundet ist, wird nur verschlüsselt verarbeitet. Somit hat niemand Einsicht in sensible Daten.

8 Grössen und Leistungen

Zu Beginn der Masterarbeit wurde ein ausführliches Wireframe erstellt. Die Idee war, dass alle drei Entwickler einmal ihre eigene Vorstellung von der Applikation aufzeigen sollten. Es wurde festgestellt, dass die Meinungen der drei Entwickler gar nicht so weit auseinander lagen. Die einzelnen Ideen wurden dann zusammengetragen und ein Wireframe erstellt. Daraus wurden dann sieben Zwischenziele abgeleitet, welche es zu erreichen gab.

Die Zwischenziele sind im Dokument Projektplan [14] als Meilensteine definiert. Im Dokument Anforderungsspezifikation [15] wird näher darauf eingegangen. Hier werden übersichtshalber nochmals die Meilensteine kurz aufgelistet.

Meilenstein	Ergebnis
Der Nutzer muss einen besuchten *Place* erfassen können	Es soll möglich sein über die Applikation einen *Place* zu erfassen und diesen in der Datenbank einzutragen.
Der Nutzer kann nach Länder, Orten und Aktivitäten suchen	Es soll möglich sein über die Applikation nach *Places* zu suchen. Die Suchergebnisse werden in der Applikation angezeigt.
Der Nutzer soll sich mit seiner Telefonnummer registrieren können.	Es soll möglich sein, sich mit seiner Telefonnummer zu registrieren. Die Daten werden in der Datenbank erfasst.
Der Nutzer sieht, welche seiner Kontakte bereits an von ihm gesuchten *Place* waren	Es kann nach einem *Place* gesucht werden und als Ergebnis wird nur angezeigt, welche Kontakte bereits dort waren.
Der Nutzer sieht nur Einträge von Kontakten, welche sich gegenseitig kennen und beide die Applikation verwenden	Bei einer Suche werden als Resultat nur Kontakte angezeigt, welche sich gegenseitig kennen. Das heisst, dass beide jeweils die Nummer des Kontakts im Telefonbuch gespeichert haben.
Der Nutzer kann Kontakte direkt kontaktieren	Es soll möglich sein, eine Person direkt aus der Applikation heraus zu kontaktieren.
Der Nutzer kann *Places* als Wunschdestination markieren	Es soll möglich sein, *Places* als Wunschdestination zu markieren und in einer Liste anzuzeigen.

Die Überprüfung, ob ein Ziel erreicht wurde, wurde direkt in der App geprüft. So hatten alle drei Entwickler stetig die Aufgabe diese Zwischenziele im Auge zu behalten und beim Erscheinen einer neuen Version immer zu prüfen, ob das formulierte Ziel auch erreicht wurde.

8.1 Wireframe

Es wird der Vorgang zwischen Wireframe und endgültiger Umsetzung aufgezeigt. Die Interpretation ob die App gemäss den Vorgaben umgesetzt wurde bleibt dem Leser überlassen.

8.1.1 Startseite

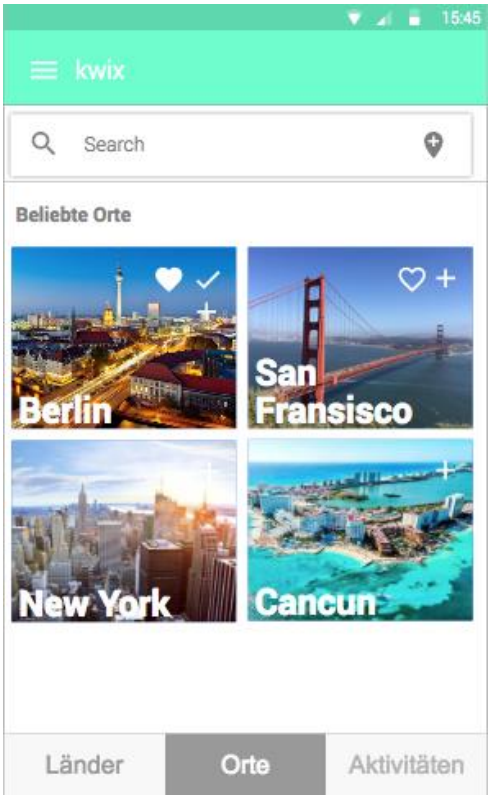


Abbildung 27 - Wireframe: Startseite

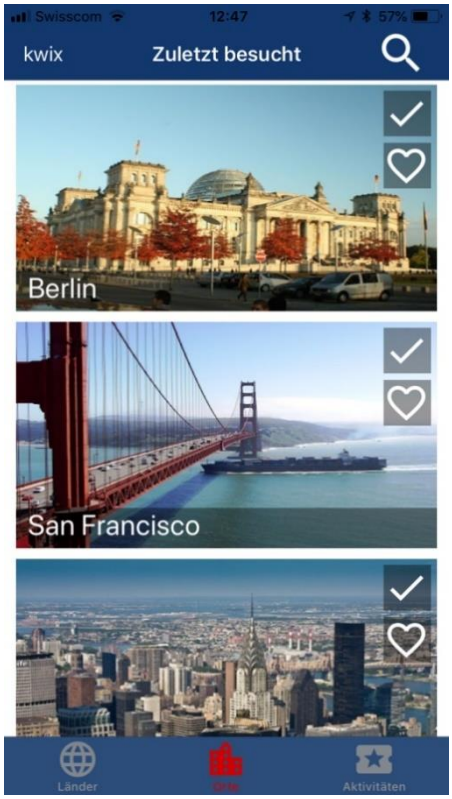


Abbildung 28 - Finale Version: Startseite

Auf der Startseite war zuerst eine Kachelansicht von beliebten Ländern, Orten und Aktivitäten geplant. Wird ein *Place* aber überdurchschnittlich oft besucht bleibt er ständig auf der Startseite. Um dem Benutzer mehr Abwechslung zu bieten werden nun die zuletzt besuchten Länder, Orte und Aktivitäten angezeigt. Das Kachellayout ist in der finalen Version durch eine Liste mit Bildern ersetzt worden. Dies aus dem Grund, dass durch die neue Anordnung mehr Platz für den Text bleibt. Ebenso können die Symbole fürs Hinzufügen und als Wunsch markieren mehr Platz einnehmen und sind so auf einem Smartphone leichter anzuwählen.
Die Idee, dass von der Startseite direkt ein *Place* als besucht markiert werden kann oder dass ein *Place* in die Wunschliste eingetragen wird, wurde aus dem Wireframe übernommen.
Bei der entgültigen Version wurde sowohl beim Text als auch bei den Symbolen einen schwarzen transparenten Hintergrund hinzugefügt, um die Lesbarkeit zu verbessern.

8.1.2 Suche nach einem Land

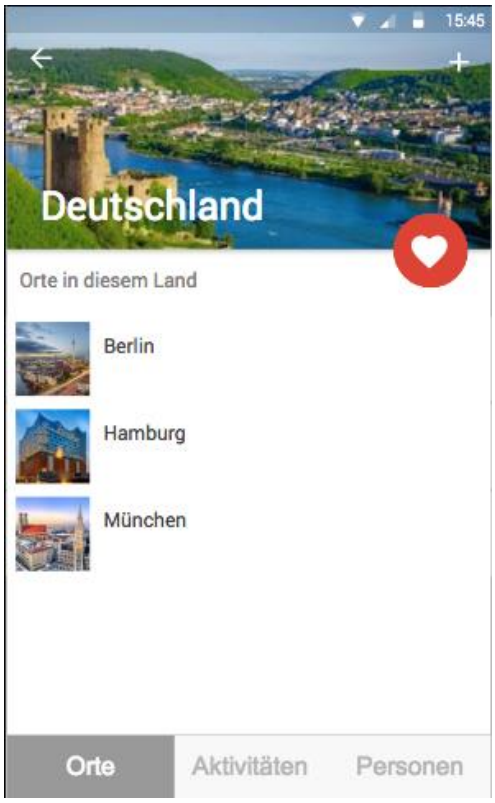


Abbildung 29 - Wireframe: Suche nach einem Land

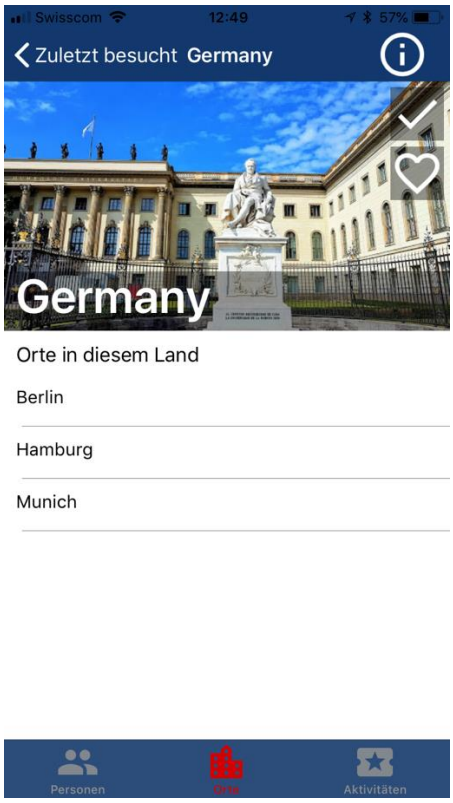


Abbildung 30 - Finale Version: Suche nach einem Land

Wird nach einem *Place* gesucht wird dieser mit einem Bild und der Möglichkeit als besucht hinzuzufügen oder in die Wunschliste einzutragen, dargestellt.
Die Anordnung der Tabs wurde ein wenig umgestellt. So befindet sich in der fertigen Version der Tab mit den Personen, welche den *Place* bereits besucht haben auf der ersten Seite. Dies aus dem Grund, da die Kontaktierfunktion der Hauptanwendungsfall der App darstellt.
Der Android Floating Action Button wurde in der finalen Version durch ein einfaches Icon mit Herz ersetzt. So passt er zum Design des Plus Buttons und beide Buttons besitzen denselben Stellenwert.

8.1.3 Personen welche einen Ort besucht haben

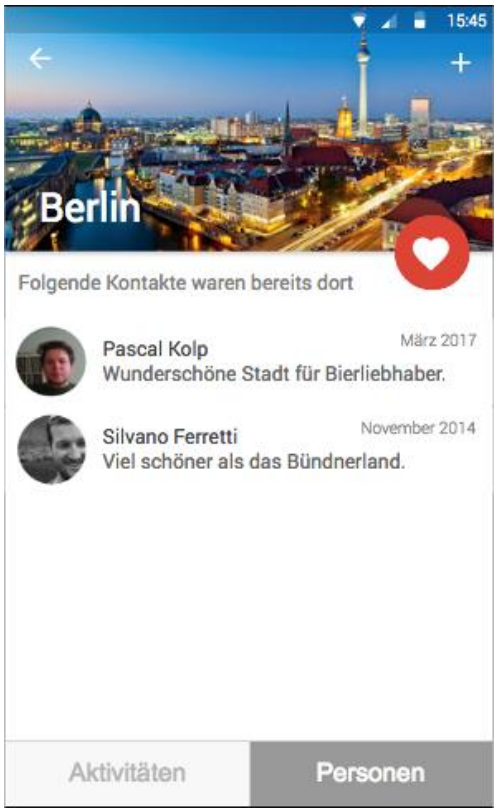


Abbildung 31 - Wireframe: Personen welche einen Ort besucht haben

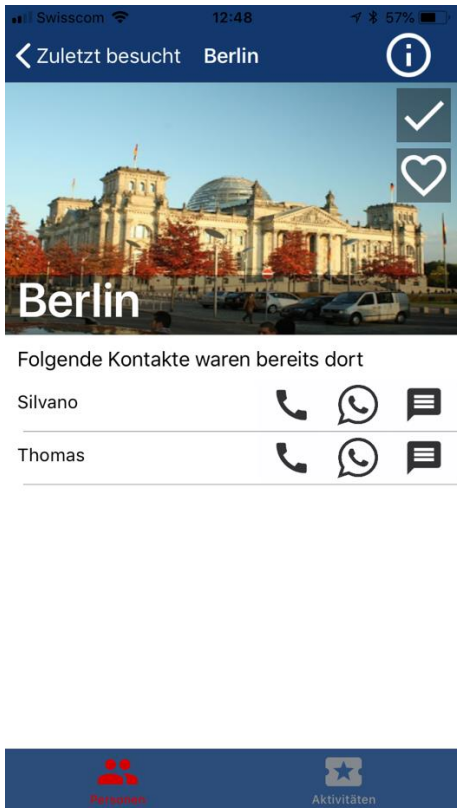


Abbildung 32 - Finale Version: Personen welche einen Ort besucht haben

Auf der Seite der Personen wird nun direkt die Funktion angeboten die Freunde per Telefonanruf, WhatsApp oder SMS zu kontaktieren. Im Wireframe war geplant, dass die Person zuerst angeklickt werden muss, bevor eine Person kontaktiert werden kann. Mit der neuen Variante braucht der Benutzer nun nicht mehr so viele Klicks bis er jemanden kontaktieren kann.
Das Profilbild des Benutzers wurde vorläufig auch noch weggelassen. Neu hinzugekommen ist der Infobutton. Beim Klicken auf diesen Button öffnet sich ein *Pop-Up* Fenster in welchem nochmals die Informationen zum *Place* angezeigt werden. So besteht die Möglichkeit für den Benutzer, der nach einer Aktivität sucht, direkt zu kontrollieren ob sich die gesuchte Aktivität auch am korrekten Ort befindet. Ebenfalls kann über diesen Button direkt eine Google Suche zum *Place* aufgerufen werden.

8.1.4 Menü

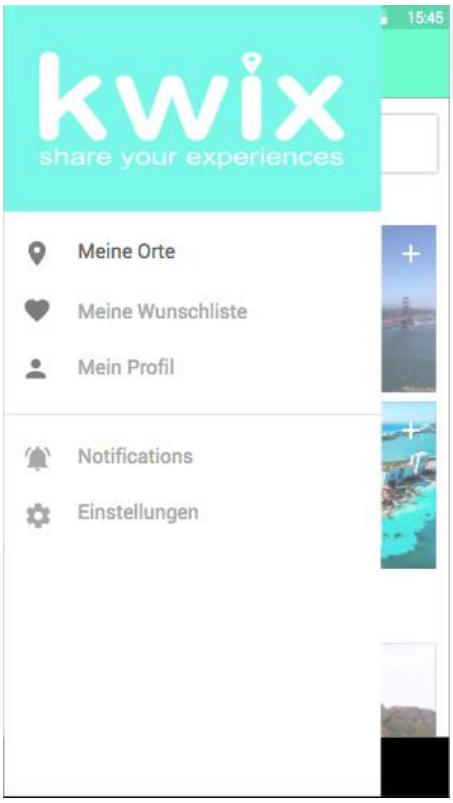


Abbildung 33 - Wireframe: Menü

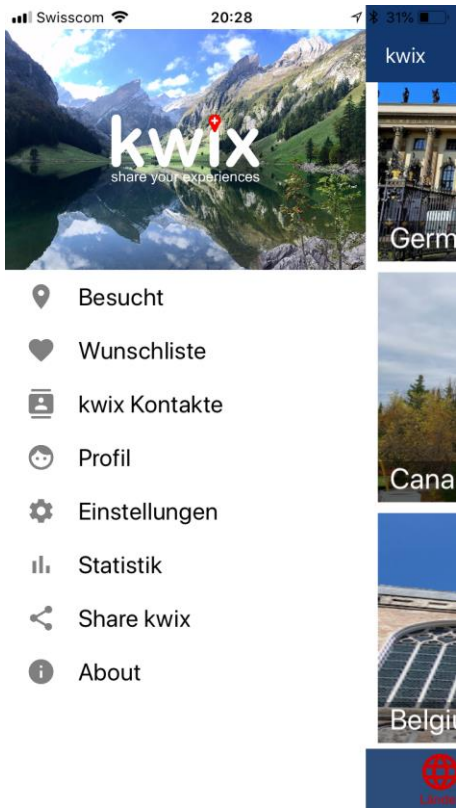


Abbildung 34 - Finale Version: Menü

Das Menü ist wie im Wireframe durch einen *Swipe* nach rechts von der Startseite aus erreichbar. Auf dem Screenshot der finalen Version sind noch einige Menüpunkte zu sehen, welche nur im Debug Modus verfügbar sind und dem Benutzer nicht angezeigt werden. So wird einem Benutzer nur Besucht, Wunschliste, kwix Kontakte, Profil, Share kwix und About angezeigt.

9 Architektur- und Designentscheidungen

9.1 Trennung der Webservices

Zu Beginn des Projekts haben wir uns dazu entschieden alle Schnittstellen über einen Webservice zu implementieren. Dies hat den grossen Vorteil, dass wir flexibel und ohne grösseren Aufwand auf Veränderungen der Gegenseite reagieren können. Als wir zu Beginn des Projekts die Datenbank aufbauten und erste Versuche machten, stellten wir fest, dass es jedoch grosse Vorteile hat, wenn wir diesen Webservice getrennt vom allgemeinen Webservice betreiben. Zum Beispiel ist es uns so möglich getrennt an beiden Webservices zu arbeiten und diese auch getrennt voneinander zu veröffentlichen (*publishen*).

9.2 Model View ViewModel (MVVM)

Für *kwix* haben wir das Entwurfsmuster Model View ViewModel (*MVVM*), welches von Xamarin unterstützt wird, angewendet. Es dient zur Trennung von Darstellung und Logik der Applikation. Der Vorteil gegenüber dem MVC (Model View Controller) ist, dass keine separaten Controller Instanzen erforderlich sind und durch das Data Binding der Implementierungsaufwand erheblich reduziert werden kann.

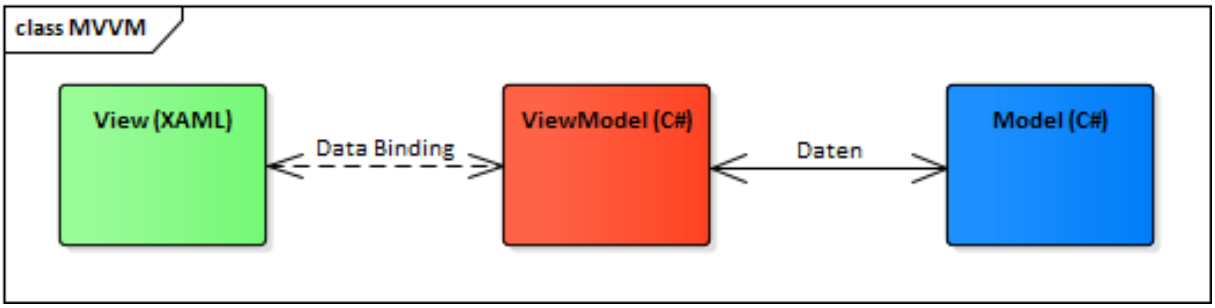


Abbildung 35 - MVVM Struktur

9.3 Integrierte Entwicklungsumgebung

Da wir uns entschieden haben *kwix* mit Xamarin auf Visual Studio zu entwickeln, wollten wir eine grösstmögliche Kompatibilität der verwendeten Tools erreichen. Somit lag die Entscheidung nahe, dass wir wo möglich Microsoft Produkte eingesetzt haben.

9.4 Google Places API

Mit dem Projektstart haben wir uns einen Überblick über die Anbieter von verschiedenen Daten *API's* in der *Places* Suche beschafft. Auf dem Markt gibt es hier zwei grosse Player zum einen ist dies Google [8] und zum anderen Foursquare [2]. Wir haben uns für Google entschieden, da Google eine grössere Datenmenge zur Verfügung stellen können. Im Nachhinein würden wir mit Sicherheit unsere Entscheidung nochmals überdenken, da Google im Juni 2018 das Kontingent von Gratisabfragen massiv gekürzt hat (siehe Kapitel 10.2 Google API) und somit auch unsere Möglichkeiten eingeschränkt hat.

9.5 Optimierung der Datenbankabfrage der zuletzt besuchten *Places*

Für die Anzeige der zuletzt besuchten *Places* der eigenen Kontakte stellten wir fest, dass die Datenbankabfrage für diese Funktion mit über 9.5 Sekunden sehr lange gedauert hat. Dies hat nach einem Refactoring des Codes zu einer Optimierung von fast Faktor zehn geführt und die Abfrage auf unter eine Sekunde gebracht.

Die Dauer der Abfragen wurde mit einer Test-Methode ermittelt, welche die benötigte Zeit für zehn Abfragen ermittelte.

9.5.1 Abfrage vor Optimierung (Dauer für 10 Abfragen: 95'724 ms)

```
listLastVisitedCountries =
    (from c in ctx.Places
     from v in ctx.Visiteds
     from f in ctx.Friends
     from f2 in ctx.Friends
     where c.PlaceType ==   PlaceType.Country && c.PlaceId == v.PlaceId &&
                            f.PhonenumberFriend1 == phonenumberAsHash &&
                            f.PhonenumberFriend2 == v.User.Phonenumber &&
                            f2.PhonenumberFriend1 == v.User.Phonenumber &&
                            f.PhonenumberFriend1 == f2.PhonenumberFriend2
     orderby v.VisitedDate descending
     select new LastVisitedPlaceModel
     { PlaceName = c.PlaceName, GooglePlaceId = c.GooglePlaceId })
     .DistinctBy(x => x.PlaceName).Take(6).ToList<LastVisitedPlaceModel>();
```

9.5.2 Abfrage nach Optimierung (Dauer für 10 Abfragen: 8'573 ms)

Verwendung von Join und Anpassung der Reihenfolge der Abfragen.

```
listLastVisitedCountries =
    (from f in ctx.Friends
     where f.PhonenumberFriend1 == phonenumberAsHash
     join v in ctx.Visiteds on f.PhonenumberFriend2
     equals v.User.Phonenumber
     join c in ctx.Places on v.PlaceId equals c.PlaceId
     where c.PlaceType == 0
     from f2 in ctx.Friends
     where f2.PhonenumberFriend1 == v.User.Phonenumber
     where f.PhonenumberFriend1 == f2.PhonenumberFriend2
     orderby v.VisitedDate descending
     select new LastVisitedPlaceModel
     {PlaceName = c.PlaceName, GooglePlaceId = c.GooglePlaceId})
    .DistinctBy(x => x.PlaceName).Take(6).ToList<LastVisitedPlaceModel>();
```

9.6 Freunde erkennen

Diese Funktion war für uns eine der grössten Herausforderungen und stellt das Herzstück von *kwix* dar.
Um uns mit unserer App abgrenzen zu können wollten wir diese Funktionalität unbedingt erreichen und haben diese auch als ein Meilenstein im Projektantrag definiert. Der Grundgedanke liegt darin, dass wenn jemand eine Nummer einer fremden Person speichert, diese Person aber die Nummer von der anderen Person nicht gespeichert hat, die besuchten *Places* dieser Person nicht angezeigt werden. Bei WhatsApp zum Beispiel kann man eine fremde Nummer speichern und sieht ob diese Person WhatsApp verwendet, sowie deren freigegeben Informationen. Dies brachte uns auf die Idee, dass wir wirklich nur befreundeten Personen, das heisst beide haben voneinander die Telefonnummer auf dem Smartphone gespeichert, anzeigen wer welchen *Place* besucht hat.

9.6.1 Ausgangslage

Anhand der folgenden Vorgaben wird die Freundeserkennung erläutert:

1.) Bob darf nur sehen, wo Alice bereits war, wenn Bob die Nummer von Alice und Alice die Nummer von Bob gespeichert hat.
2.) Hat Bob nur die Nummer von Alice, Alice aber nicht die Nummer von Bob gespeichert, dann darf Bob nicht sehen, wo Alice bereits war.
3.) Hat Alice die Nummer von Bob, Bob aber nicht die Nummer von Alice, kann Bob nicht sehen, wo Alice bereits war

9.6.2 Tabellen in der kwix-Datenbank

9.6.2.1 User

Tabelle mit ID, Username und gehashter Telefonnummer.

ID	Username	Telefonnummer

9.6.2.2 Friends

Tabelle mit ID, PhonenumberFriend1 und PhonenumberFriend2. Es werden alle gespeicherten Kontakte in die Tabelle Friends übertragen.

ID	PhonenumberFriend1	PhonenumberFriend2

9.6.3 Szenario

1.) Bob meldet sich bei *kwix* an. Die Telefonnummer von Bob wird verschlüsselt gespeichert

ID	Username	Telefonnummer
1	Bob	9c167ff7d51b6e6fe7900fe8676352d2

2.) Alle Kontakte von Bob werden ausgelesen und in die *kwix* Datenbank als *Hashwert* übertragen

ID	PhonenumberFriend1	PhonenumberFriend2
1	9c167ff7d51b6e6fe7900fe8676352d2	db505b5efff35fc000aab5279b725e9e
2	9c167ff7d51b6e6fe7900fe8676352d2	b6fc7dbc2dc6ec5da8f7c64f87cb2e42
3	9c167ff7d51b6e6fe7900fe8676352d2	2422e8e4a3010c32fa09dba68dc84fa8

3.) Alice meldet sich bei *kwix* an. Die Telefonnummer von Alice wird ebenfalls verschlüsselt in der *kwix* Datenbank gespeichert.

ID	Username	Telefonnummer
1	Bob	9c167ff7d51b6e6fe7900fe8676352d2
2	Alice	db505b5efff35fc000aab5279b725e9e

4.) Die Kontakte von Alice werden in die Friends Tabelle als *Hashwert* übertragen.

ID	PhonenumberFriend1	PhonenumberFriend2
1	9c167ff7d51b6e6fe7900fe8676352d2	db505b5efff35fc000aab5279b725e9e
2	9c167ff7d51b6e6fe7900fe8676352d2	b6fc7dbc2dc6ec5da8f7c64f87cb2e42
3	9c167ff7d51b6e6fe7900fe8676352d2	2422e8e4a3010c32fa09dba68dc84fa8
4	db505b5efff35fc000aab5279b725e9e	9c167ff7d51b6e6fe7900fe8676352d2
5	db505b5efff35fc000aab5279b725e9e	7c12022d813c5d84164c063e7ecbd4a2
6	db505b5efff35fc000aab5279b725e9e	53e1b4c29928a66a354f9fb323cd69d1

5.) Bei einer Suche nach einem *Place* oder auch um die zuletzt besuchten *Places* anzuzeigen, wird jeweils immer in der Friends Tabelle nachgeschaut, ob beide die Telefonnummer voneinander haben.
Sucht Bob nach einem *Place* wird in der Friends Tabelle nach seiner *gehashten* Telefonnummer (9c167ff7d51b6e6fe7900fe8676352d2) in der Spalte PhonenumberFriend2 gesucht. Wird der *Hashwert* gefunden, wird die zugehörende PhonenumberFriend1 ausgelesen. Im Beispiel wäre das bei ID 4 der Fall und der *Hashwert* von PhonenumberFriend1 lautet db505b5efff35fc000aab5279b725e9e. Mit diesem *Hashwert* wird wieder eine Suche gestartet und dabei muss dieser in der Spalte PhonenumberFriend2 vorkommen. Ebenfalls muss bei dieser Suche in der Spalte PhonenumberFriend1 der *Hashwert* von Bob's Telefonnummer enthalten sein. Das ist im Beispiel bei ID 1 der Fall. Die PhonenumberFriend2 von ID 1 wird nun verwendet um eine Suche in der User Tabelle zu starten, um an den Benutzernamen zu gelangen.
Wenn Alice nun den von Bob gesuchten *Place* schon besucht hat, wird Alice bei den Personen aufgelistet und kann von Bob kontaktiert werden.

Bob muss warten, bis sich Alice angemeldet hat, damit er ihre Einträge sehen kann. Alice sieht bei ihrer ersten Anmeldung bereits Bob’s Einträge, da beide die Telefonnummer voneinander haben.

Sollte es vorkommen, dass eine Telefonnummer von einem Freund mehrmals im Telefonbuch gespeichert ist, wird das von der App geprüft und die Duplikate werden entfernt.

10 Kosten

Da die Kosten für diese Masterarbeit vollumfänglich von den drei Entwicklern getragen werden, wurde bei der Auswahl der verwendeten Software und Dienste auf die Kosten geachtet.

10.1 Microsoft Azure

Für Studenten sind bei Microsoft Azure die meisten Service kostenlos [16]. Es gibt jedoch gewisse Einschränkungen bezüglich Speicher und Geschwindigkeit. Bei den von *kwix* verwendeten Diensten (*VSTS*, Webservice und Datenbank) gab es weder Einschränkungen noch zusätzlichen Kosten.

10.2 Google API

Google verlangt für jede *API* Abfrage Ihrer Dienste sogenannte Credits [17]. Für die von *kwix* verwendeten Dienste "Place Detail" und "Place Photos" werden pro Abfrage 2 Credits fällig und für jede "Place Autocomplete" Abfrage 0.1 Credits. Bis Mitte Juni 2018 bekam jeder bei Google registrierte Entwickler pro Monat 150'000 gratis Credits zur Verfügung. Ab Mitte Juni 2018 hat Google eine neue Preispolitik eingeführt, bei der jeder Entwickler pro Monat 200 USD an gratis Abfragen bekommt. Dies entspricht etwa 50'000 Credits (Google legt die Credits in der Abrechnung [18] nicht klar offen), was nur noch einem Drittel an gratis Abfragen entspricht.
Die Kosten für die Google API müssen vor dem Verteilen über die App Stores massiv gesenkt werden, oder es muss auf eine Alternative gewechselt werden.

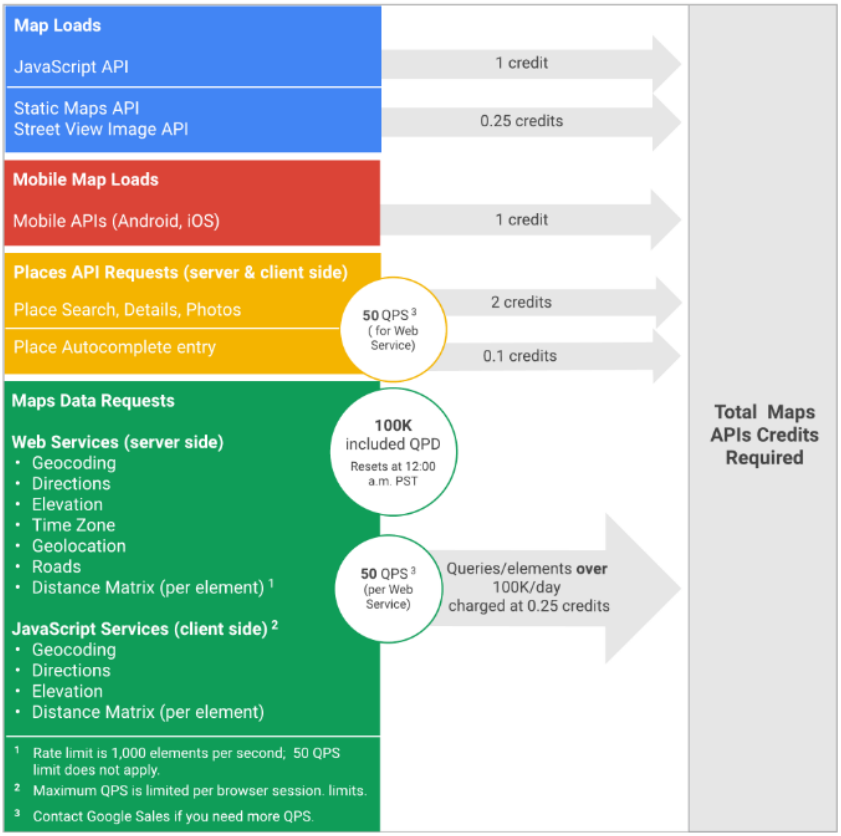


Abbildung 36 - Google Credits

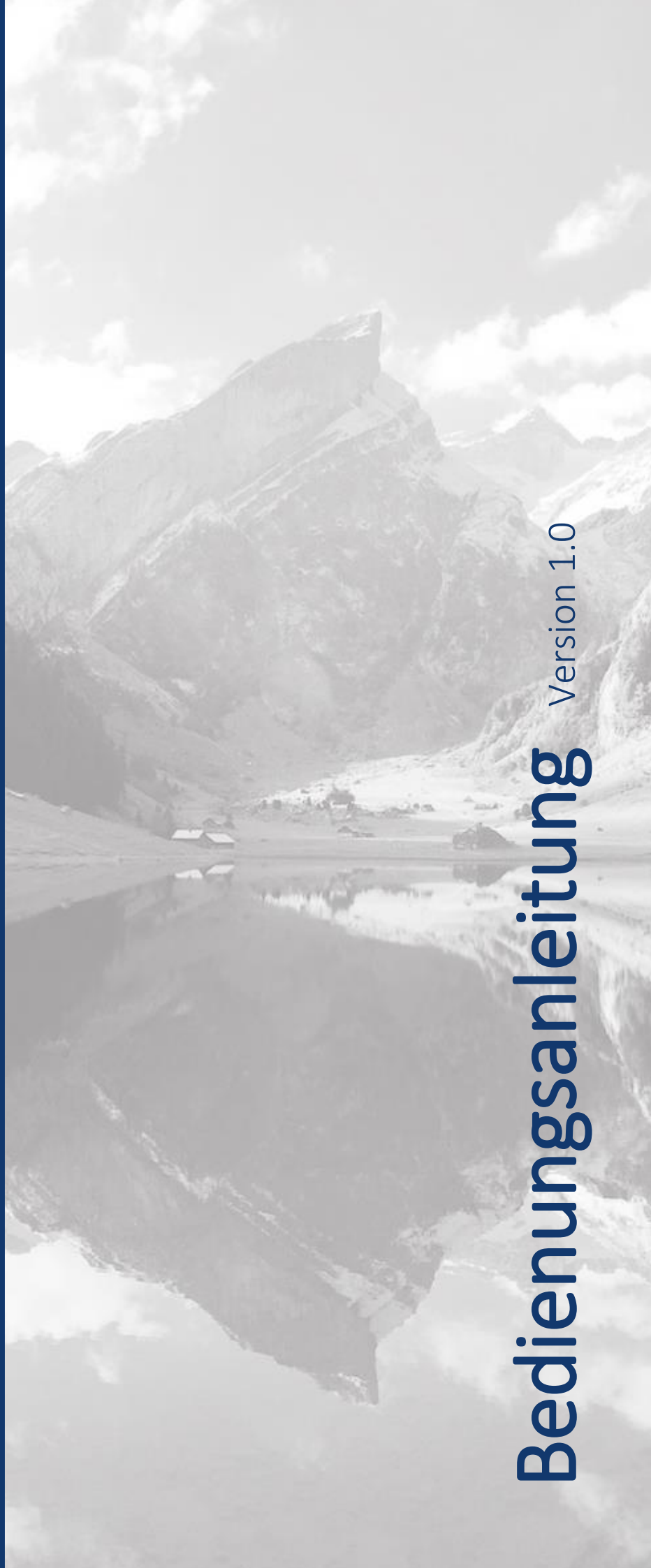
10.3 SMS für Authentifizierung

Der verwendete SMS Dienst "Moreify" [19] kostet *kwix* pro versendete SMS zu einem Benutzer mit einer Schweizer Telefonnummer 0.07€. Für die Entwicklung und Verbreitung im Rahmen dieser Masterarbeit ist dies vertretbar. Bei einer allfälligen grösseren Nutzerzahl muss dieser Dienst sicherlich nochmals hinterfragt werden, da sich die dadurch anfallenden Kosten summieren könnten.

11 Abbildungsverzeichnis

kwix

share your experiences



Bedienungsanleitung

Version 1.0

Thomas Egle
Silvano Ferretti
Pascal Kolp

Änderungsgeschichte

Version	Autor	Beschreibung	Datum
0.1	Silvano Ferretti	Erstellung	27.08.2018
1.0	Projektteam	Freigabe	27.08.2018

Inhaltsverzeichnis

1 Einführung

1.1 Zweck

Dieses Dokument beschreibt die Funktionsweise der App *kwix*.

1.2 Gültigkeitsbereich

Dieses Dokument ist über die ganze Projektdauer gültig. Änderungen werden fortlaufend ergänzt und in der Änderungsgeschichte festgehalten.

1.3 Referenzen

Nr	Titel	Quelle
[1]	Glossar	…\Dokumente\Glossar_V1.0.pdf

1.4 Glossar

Alle verwendeten Begriffe aus dem Glossar werden in diesem Dokument *kursiv* geschrieben.

Siehe Glossar [1]

2 Anmeldung

Beim erstmaligen Anmelden bei *kwix* muss ein Benutzername, die eigene Telefonnummer sowie deren Vorwahl eingegeben werden.

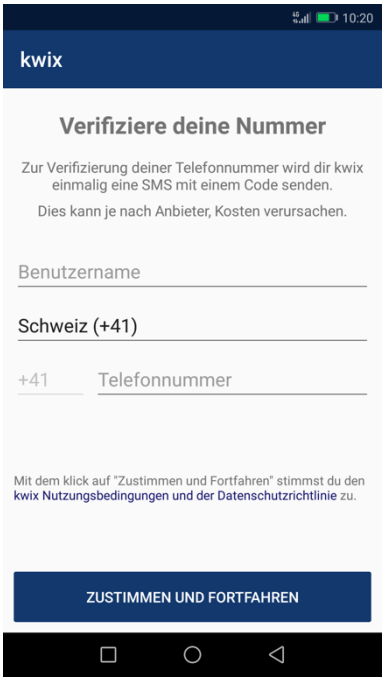


Abbildung 1 - Login Seite

Durch klicken auf "Zustimmen und Fortfahren" werden die Nutzungsbedingungen und die Datenschutzrichtlinien von *kwix* akzeptiert und ein *SMS* mit dem Authentifizierungscode wird an den Benutzer gesendet. Der erhaltene, sechsstellige Code muss auf der nächsten Seite eingegeben werden.

Abbildung 2 - Authentifizierungs Seite

Durch klicken des Buttons "Abschliessen" gelangt man auf die Startseite.

3 Startseite

Auf der Startseite werden die zuletzt besuchten Länder, Orte und Aktivitäten angezeigt. Zwischen diesen drei Tabs kann mittels *swipen* (Android) oder Auswahl (iOS) gewechselt werden.

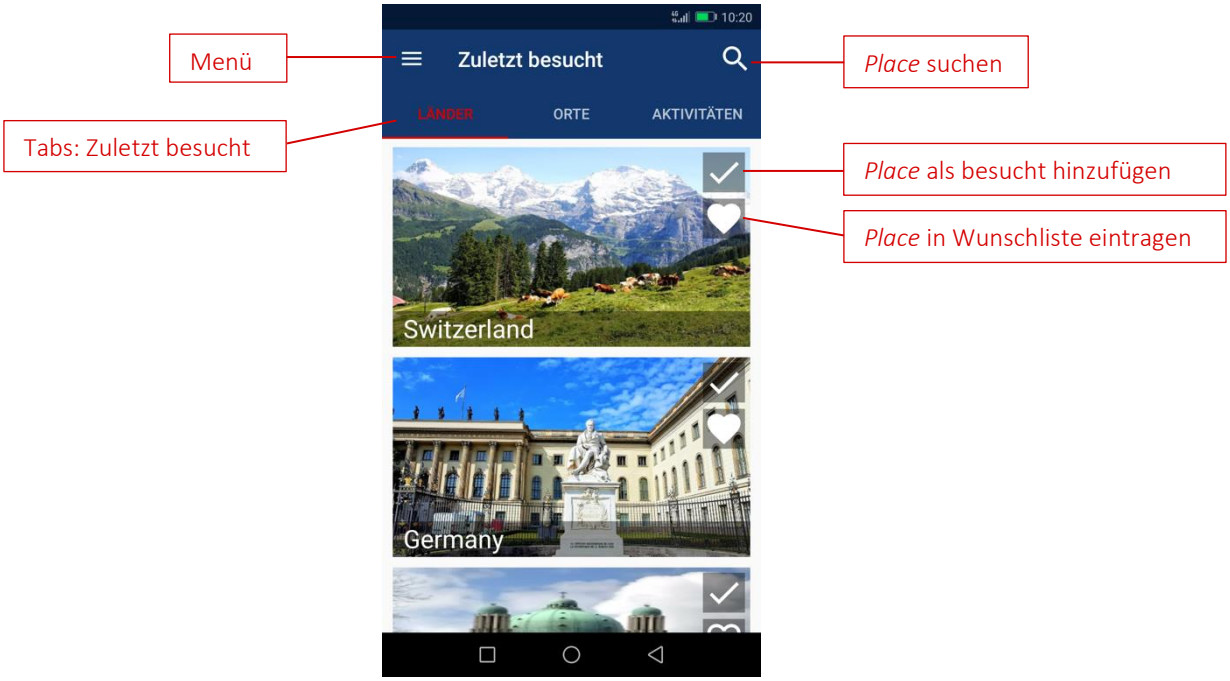


Abbildung 3 - Startseite

Die zuletzt besuchten *Places* können direkt als besucht oder in die Wunschliste eingetragen werden.
Beim Klick auf das Suchen Symbol gelangt man auf die Seite um nach einem *Place* zu suchen.

Symbol	Bedeutung
+	*Place* wurde noch nicht besucht
✓	*Place* wurde bereits besucht
♡	*Place* ist nicht in Wunschliste eingetragen
♥	*Place* ist in Wunschliste eingetragen

4 Suchseite

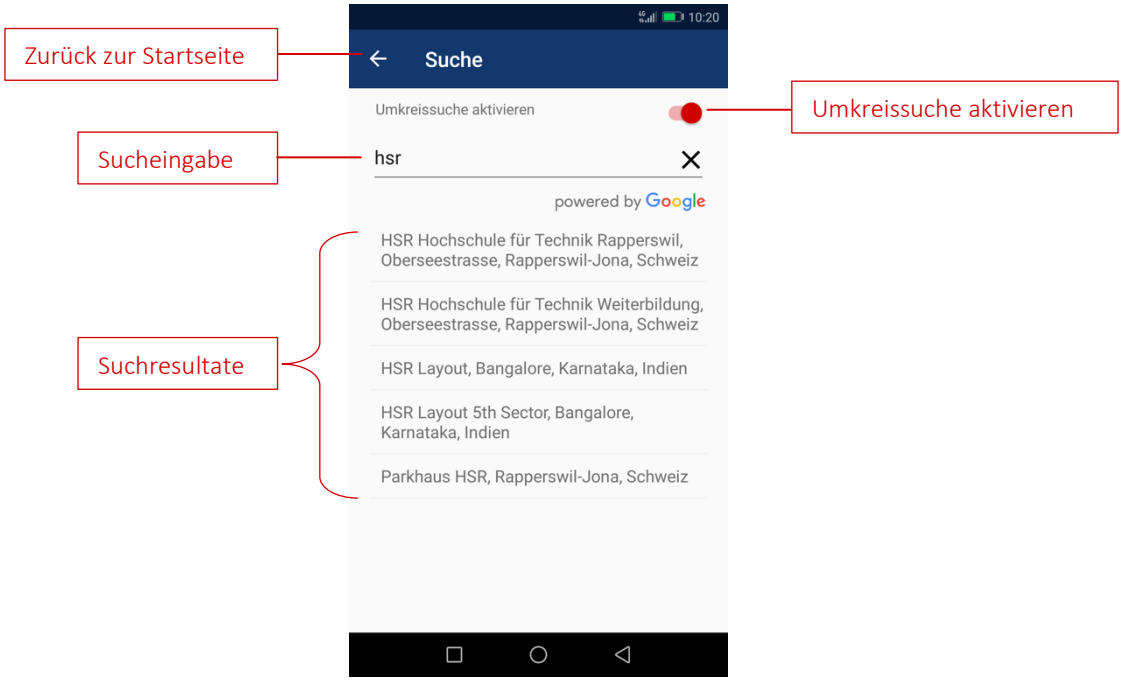


Abbildung 4 - Suchseite

Auf der Suchseite kann die Umkreissuche eingestellt werden.
Die Suchergebnisse erscheinen, sobald ein Text von mehr als zwei Zeichen in das Suchfeld eingegeben wurde.
Die gefundenen *Places* können dann ausgewählt werden.

5 Menü

Mit einem *Swipe* nach rechts oder durch Klicken auf das Menu Icon gelangt man ins Menü.
Im Menü können über "Besucht" alle besuchten *Places* angezeigt werden. Unter "Wunschliste" sieht man die *Places*, die als Wunsch hinzugefügt wurden. Bei "kwix Kontakte" werden alle Kontakte angezeigt, welche im Telefonbuch gespeichert sind und ebenfalls *kwix* nutzen.

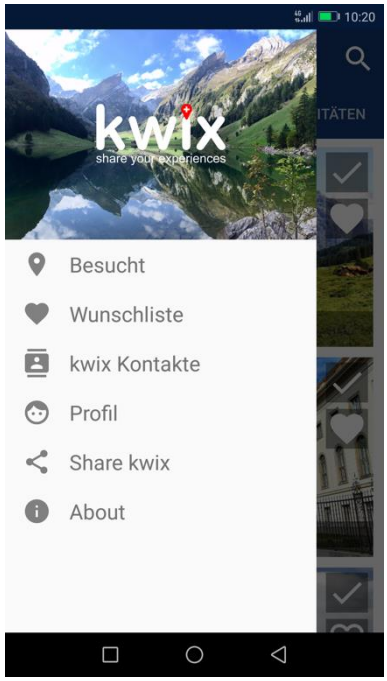


Abbildung 5 - Menü

Über "Profil" kann der eigene Benutzername angepasst werden. Ebenfalls kann über diese Seite der *kwix*-Account gelöscht werden.
Bei "Share kwix" können Freunde auf *kwix* aufmerksam gemacht werden.
Auf der About Seite können unter anderem die Nutzungsbedingungen und die Datenschutzrichtlinien eingesehen werden.

6 *Place* Seite

Wird über die Suchseite nach einem *Place* gesucht oder auf der Startseite einen zuletzt besuchten *Place* angewählt, wird die entsprechende *Place* Seite aufgerufen.

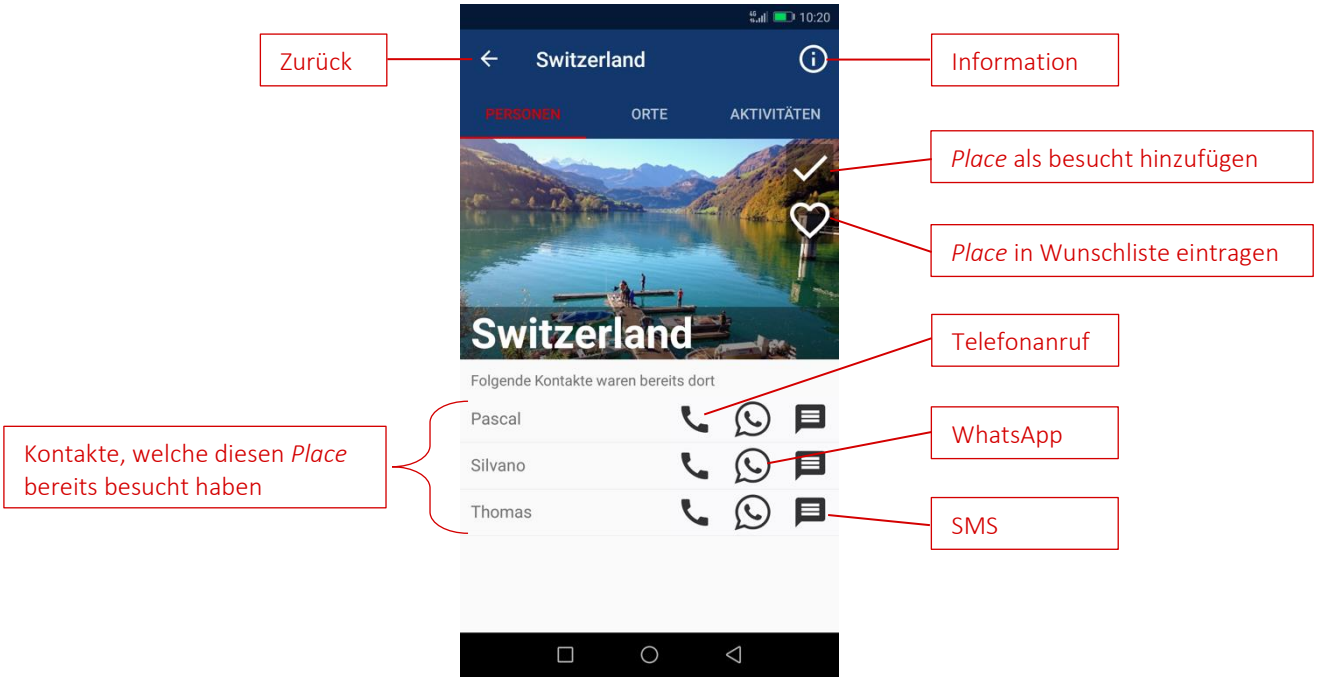


Abbildung 6 - Place Seite

Die Tabs sind je nach gesuchtem *Place* Type unterschiedlich:

Suche nach …	Vorhandene Tabs
Land	Personen Orte Aktivitäten
Ort	Personen Aktivitäten
Aktivität	Personen

Über die *Places* Seite können die *kwix* Kontakte kontaktiert werden. Es steht Telefonanruf, WhatsApp oder *SMS* zur Auswahl. Durch einen Klick auf eine Person, erscheint ein *Pop-Up*, in welchem die Telefonnummer sowie der Name, wie er im Telefonbuch hinterlegt ist, angezeigt wird.

7 Abbildungsverzeichnis