HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Rebecca Blischke

Konzeption und Umsetzung eines App - Prototypen für die Erstellung von Immobilien-Exposés

2018

Fakultät Medien

BACHELORARBEIT

Konzeption und Umsetzung eines App - Prototypen für die Erstellung von Immobilien-Exposés

Autor:
Frau Rebecca Blischke

Studiengang:
Mobile Media

Seminargruppe:
MD13w1-B

Erstprüfer:
Prof. Dr. - Ing. Frank Zimmer

Zweitprüfer:
Dipl.- Ing. Stefan Kurzawski

Einreichung:
Mittweida, 27.01.2018

Faculty of Media

BACHELOR THESIS

Conception and implementation of an app-prototyp for the creation of real estate exposés

author:
Ms. Rebecca Blischke

course of studies:
Mobile Media

seminar group:
MD13w1-B

first examiner:
Prof. Dr. - Ing. Frank Zimmer

second examiner:
Dipl.- Ing. Stefan Kurzawski

submission:
Mittweida, 28.01.2018

Bibliografische Angaben:

Nachname, Vorname: Blischke, Rebecca

Thema der Bachelorarbeit

Konzeption und Umsetzung eines App - Prototypen für die Erstellung von Immobilien - Exposés

2018 - 83 Seiten

Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,

Fakultät Medien, Bachelorarbeit, 2018

Abstract

Diese Bachelorarbeit beschreibt den Entwurf und die Implementierung eines App - Prototypen zum Thema Erstellung von Immobilien - Exposés. Die Besonderheit der App liegt im Zusammenspiel mit der Schnittstelle von ImmobilenScout24. In dieser Arbeit soll eine Einführung in das Betriebsystem Android und dessen zugrundeliegenden Technologien und die Schnittstelle gegeben werden. Anschließend wird ein Entwurf der Anwendung erstellt und die Implementierung erläutert. Zum Schluss werden Probleme bei der Entwicklung erörtert sowie Weiterentwicklungsmöglichkeiten aufgezeigt.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface (Programmierschnittstelle)
AOT	ahead – of - time
ARM	Advanced RISC Machines (früher: Acorn RISC Machines)
ART	Android Runtime
CMS	Content Management System-Apps
CSS	Cascading Style Sheets
DalvikVM	Dalvik Virtual Machine
GIF	Graphics Interchange Format (Grafikaustauschformat)
HAL	Hardware Abstraction Layer
HTML	Hypertext Markup Language
HTML5	5. Fassung der Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
iOS	Betriebssystem von Apple für Smartphones und Tablets
JSON	JavaScriptOperationNotion
JAVA	Programmiersprache
JIT	just-in-time
noSQL	Not only SQL
OpenGLES	Open Graphics Library for Embedded Systems
OS	Operating System (Betriebssystem)
SDK	Software Development Kit
SQL	Structured Query Language

UI	User Interface
UTF - 8	8 - Bit Universal Character Set Transformation Format
VM	Virtual Machine
XML	Extenisble Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

1 Einleitung

Die vorliegende Bachelorarbeit wird im Rahmen des Studiums an der Hochschule Mittweida zum Bachelor of Science in Mobile Media als Abschlussbeleg durchgeführt. Den thematischen Schwerpunkt bildet die Konzeption und Umsetzung eines App - Prototypen für die Erstellung von Immobilien - Exposés mit direkter Weiterleitung auf Plattformen wie “ImmobilienScout24.de” , “immowelt.de” und “immonet.de” für die Firma Full Service Agentur Webgalaxie & Systemhaus Krüger GmbH (fortan Webgalaxie genannt). Einleitend wird in den folgenden Kapiteln zunächst das Unternehmen vorgestellt und die Ausgangssituation erläutert. Anschließend wird die Zielsetzung der Bachelorarbeit in Kapitel 1.3 und die Grundlagen für die Entwicklung (Kapitel 2) kurz erklärt und vorgestellt.

1.1 Unternehmensvorstellung

Die Full Service Agentur Webgalaxie ist seit 1992 in dieser Form tätig und betreut über 1100 Unternehmen in ganz Deutschland und Europa. Es handelt sich dabei um mittelständische bis große Unternehmen aus den verschiedensten Branchen.

Die in der Dienstleistungsbranche agierende Agentur bietet die verschiedensten Leistungen im Bereich „Web“. Dabei übernimmt das Unternehmen Aufgaben wie das Webdesign, die Erstellung von Homepages mit oder ohne *CMS*-Systemen, eine Suchmaschinenoptimierung für Webseiten und die Programmierung von Webanwendungen und Softwarelösungen. Auch unterstützt das Unternehmen seine Kunden bei der Erstellung von Online-Shops, Grafik - / Screen - / Logo - Design, Flyern, Prospekten und ähnlichen.

Mit circa 20 Mitarbeitern ist die Full Service Agentur Webgalaxie ein kleines Unternehmen. Intern ist das Unternehmen in die Bereiche der Geschäftsabteilung, Verwaltung, Buchhaltung, den technischer Support, die Grafikabteilung und die Abteilung Entwicklung strukturiert.

1.2 Projektumfeld

Die vorliegende Bachelorarbeit wird in der Abteilung „Webanwendungen & Softwarelösungen“ der Firma Webgalaxie durchgeführt. Zu den wesentlichen Aufgaben dieser Abteilung zählen die Planung, Konzeption und Weiterentwicklung von Webanwendungen und Webseiten. Zu den Projektbeteiligten zählt insbesondere der Auftraggeber Stefan Kurzawski.

1.3 Zielsetzung

Das Ziel dieser Bachelorarbeit ist es eine App zu konzipieren und in Teilen umzusetzen, welche die Erstellung von Immobilien - Exposés mit direkter Weiterleitung auf Plattformen wie “ImmobilienScout24.de” , “immowelt.de” und “immonet.de” für Immobilienmakler erleichtert.

1.4 Kapitelübersicht

Nachdem im Kapitel 1 das Projektumfeld und das Unternehmen vorgestellt wurde, werden im Kapitel 2 die nötigen Grundlagen erläutert, die benötigt werden, um das Ziel dieser Bachelorarbeit zu erreichen. In diesem Kapitel werden zudem Informationen über die Entwicklungsumgebung und Firebase gegeben, bevor im Kapitel 3 der IST- Stand genauer betrachtet wird. Auf diesem IST- Stand baut das folgende Kapitel auf, das sich mit der Konzeption der App beschäftigt. Hierbei wird auf die Idee an sich, die Anforderungsliste sowie diverse Anwendungsszenarien eingegangen. Den Hauptteil dieser Arbeit bildet die Umsetzung des Protoypen. In Kapitel 6 wird zunächst die Umsetzungsstrategie erläutert und danach auf die einzelnen Implementationen der Anwendungsfälle eingegangen. In Kapitel 7 wir auf Probleme bei der Implementierung eingegangen. Da diese Applikation ein Prototyp darstellt wird im nachfolgenden Kapitel (Kapitel 8) auf Weiterentwicklungsmöglichkeiten der App eingegangen. Den Abschluss bildet das Fazit in Kapitel 9.

2 Grundlagen und Hintergründe

In diesem Kapitel werden Grundlagen und Hintergründe der Programmierung erläutert, die für das Verständnis dieser Arbeit beitragen.

2.1 Arten von Apps

In der Erstellung von Apps gibt es drei wesentliche Formen, die zu unterscheiden sind. In den folgenden Kapiteln wird auf diese sowie ihr Vor - und Nachteile kurz eingegangen.

2.1.1 Native Apps

Eine native App wird speziell für ein bestimmtes Betriebssystem entwickelt. Es kommen Strukturen zum Einsatz, die für das jeweilige Betriebssystem typisch sind. Native Apps orientieren sich dabei an einem standardisierten Design des jeweiligen Betriebssystems, dem sogenannten „Look and Feel“ und können dadurch optimal angepasst werden. Native Apps sind sehr performant. Durch die hohe Performance nativer Apps können auch aufwendige 3D - Apps entwickelt und vertrieben werden. Zudem wird auch der volle Zugriff auf Hardwareressourcen ermöglicht.

2.1.2 Web-Apps

Eine Web-App wird im Browser (Google Chrome, Opera, Internet Explorer, Edge, Firefox) ausgeführt und basiert daher auf *HTML5*. Eine Installation auf dem Gerät ist daher nicht notwendig. Schlussfolgernd ist es bei Web - Apps nicht notwendig gerätespezifische Kenntnisse zu besitzen, da diese Apps mittels *HTML*, *CSS* und JavaScript entwickelt werden. Das hat den großen Vorteil, dass Web - Apps im Vergleich zu nativen Apps nur einmal entwickelt werden müssen und auf verschiedenen Betriebssystemen lauffähig sind. Vergleicht man jedoch Web - Apps mit nativen Applikationen, so stellt man fest, dass native Apps performanter sind, um aufwendige Anwendungen zu realisieren. Ein weiterer Nachteil ist die fehlende Anpassung an das jeweilige Betriebssystem. Designs werden einheitlich auf allen Betriebssystemen angezeigt, was das Nutzererlebnis auf einem Smartphone beeinträchtigen kann. Ebenfalls besitzen die

Web-Apps einen limitierten Zugriff auf die Hardwareressourcen des Gerätes. Zudem funktionieren die meisten Funktionen nur mit einer bestehenden Internetverbindung, was einen großen Nachteil darstellt.

2.1.3 Hybride Apps

Eine hybride App kann zwischen einer nativen App und einer Web - App eingeordnet werden. Für die Entwicklung der App kommen wie in der Web - App *HTML*, *CSS* und JavaScript zum Einsatz. Dies macht die Entwicklung für verschiedene Betriebssysteme einfacher. Im Vergleich zur Web - App wird diese App aber auf dem Gerät installiert. Vorteile einer hybriden App liegen darin, dass sie als eigenständige App auf dem Gerät installiert ist und somit wie bei einer nativen App auf die Hardwareressourcen zugegriffen werden kann. Die Umsetzung des Look and Feel von nativen Apps gestaltet sich jedoch schwierig.

2.1.4 Auswahl der Art der Applikation

Für die Umsetzung des Prototypen der App wird eine native App gewählt. Gründe für diese Entscheidung waren:

- Kenntnisse der Programmierung mit Java und dem Android-Betriebssystem
- fehlende Zeit zur Einarbeitung in hybride Apps
- Verwendung des Look and Feels der Android- Anwendungen

2.2 Plattform Android

Android ist ein von Google weiterentwickeltes freies Betriebssystem und Software-Plattform für Smartphones und Tablets.[1] Das 2003 von Andy Rubin gegründete Unternehmen entwickelte die Plattform Android zunächst unter strengster Geheimhaltung, bevor das Unternehmen von dem kalifornischen Unternehmen Google 2005 [2] aufgekauft wurde. Es ist eine Open – Source - Software, die erstmal am

1 vgl. R. Schanze, 31.08.2017: Was ist Android? - Kurz erklärt **[SCHANZE]**

2 vgl. HandyFlash, 15.03.2017: Was ist Android? Das musst du über Android wissen? **[HANDYFLASH]**

21. Oktober 2008[3] vorgestellt wurde. Da diverse Hersteller ihre Geräte auf Android - Basis anbieten, kann sich das Betriebssystem mit ca. 80% Marktanteil (Stand: 2017) neben *iOS*- Geräten (ca. 1% Marktanteil) etablieren[4].

Das Android *OS* kann durch zahlreiche Applikationen, sogenannte Apps, erweitert werden. Diese sind im Google Play Store verfügbar.[5] Zudem handelt es sich hierbei um eine große Anzahl von Apps aus den verschiedensten Kategorien, wie zum Beispiel Spiele, Wetter, Messaging - Apps oder auch Applikationen aus dem Office - Bereich.

Android liefert neben der benötigten Software und dem Betriebssystem auch die nötigen Treiber für mobile Endgeräte (Smartphones), Libraries und Frameworks, sowie fertige Anwendungsprogramme.

Ein weiterer Vorteil von Android ist, dass es sich an verschiedene Screen Größen automatisch anpasst. Dies ist möglich durch das Bereitstellen von Ressourcendateien in verschiedenen Auflösungen und Größen. [6]

Das Android *OS* gibt es mittlerweile in sieben Versionen, die alle den Titel einer Süßspeise neben ihrer Versionsnummer tragen[7]. Zur Zeit wird die Version 8.1 als neuste Version von Android verwendet. Die Version 8.1 zum Beispiel trägt den Namen Oreo. In dieser Version wurden neue Emojis (Smileys), eine Unterstützung von *GIF* - Bildern und eine Vereinheitlichung der App-Icons eingeführt.

Um eine Kompatibilität von Apps über mehrere Versionen zu ermöglichen, verwendet Android sogenannte *API*s. Dieses Application Programming Interface, gibt den Entwicklungsstand einzelner Funktionen des *OS* an. Programmierer können dank der *API* mit dem Android - System kommunizieren und auf dessen Grundlagen aufbauen. Neuere *API* sind immer abwärtskompatibel und fügen nur neue Funktionen hinzu oder verbessern alte Funktionen. Die aktuelle Version ist zur Zeit *API* - Level 27 (Stand: Januar 2018). Das *API* - Level wird somit zu einer tragenden Rolle des Systems, die es dem Nutzer und den Entwicklern ermöglicht, die bestmöglichen Nutzungserlebnisse zu erzielen.[8]

3 vgl. R. Schanze, 31.08.2017: Was ist Android? - Kurz erklärt **[SCHANZE]**

4 R. Schanze, 31.08.2017: Was ist Android? - Kurz erklärt **[SCHANZE]**

5 vgl. HandyFlash, 15.03.2017: Was ist Android? Das musst du über Android wissen? **[HANDYFLASH]**

6 Android Developer: Device Compatibility **[ANDROIDDEV-SCREEN]**

7 vgl. HandyFlash, 15.03.2017: Was ist Android? Das musst du über Android wissen? **[HANDYFLASH]**

8 vgl. Android Developer: Device Compatibility **[ANDROIDDEV-VERSION]**

Das gesamte Android *SDK* lässt sich dank des Open – Source - Status kostenfrei von der Webseite von Android für alle gängigen Smartphones herunterladen.[9] Auf der Homepage von Android finden sich für Entwickler auch die zugehörige Dokumentation, Beispielcodes und Entwicklerwerkzeuge.

2.2.1 Android - Architektur

Android beruht auf dem Betriebssystem Linux, wird aber nicht als eine Linux Distribution angesehen, da einige Funktionen/ Begebenheiten von Linux stark abgeändert oder gar entfernt wurden[10]. Von dem Linux *OS* erbt Android seinen Kernel, das File - System und auch das System, dass bestimmt wie die Berechtigungen vergeben werden müssen. Auch wurden andere Komponenten übernommen, die eigentlich für das Linux - System entwickelt worden. Zusätzlich zu den Linux - Komponenten enthält Android eine weitere Schicht - die Java Schicht, welche aus zwei Prozessen und einer *DalvikVM* besteht. Neben der *VM* sind auch Klassenbibliotheken in den Android - Quellcode implementiert.

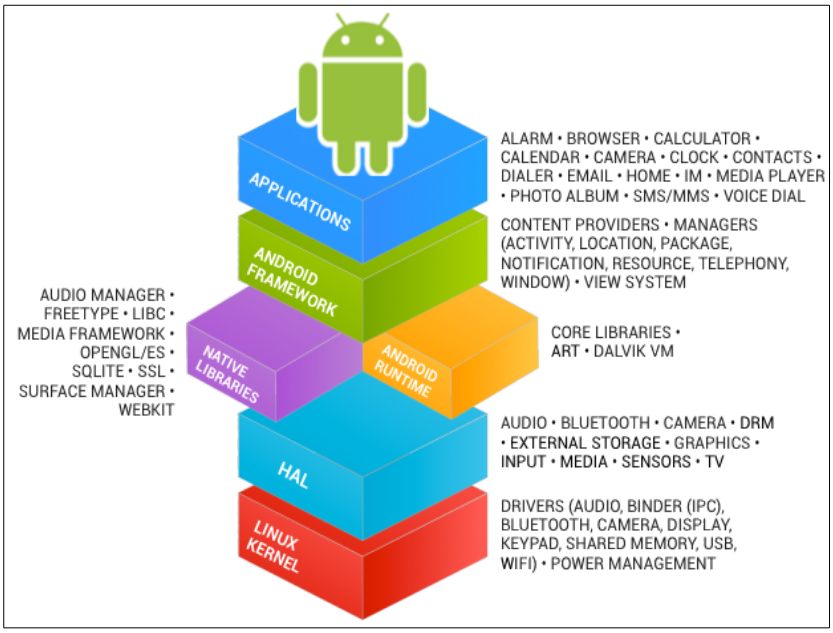

Abbildung 1: Architektur Android

9 vgl. Android Source: About the Android Open Source Project **[ANDROIDSOURCE]**

10 vgl. Christoph H. Hochstätter, 18.05.2011: Android-Architektur: Wieviel Linux steckt in Googles OS? **[HOCH]**

Der Linux Kernel

Der Linux Kernel bildet die Grundlage der Architektur. Er erlaubt dem System von sicherheitstechnischen Funktionalitäten Gebrauch zu machen und vereinfacht es für Hardware - Hersteller Treiber für den bekannten Kernel herzustellen, während er grundständige Treiber bereitstellt.[11]

Der HAL

Der sogenannte *HAL* ist der Hardware Abstraction Layer und im Endeffekt die Software, die es den Apps ermöglicht, Informationen abzurufen. Dabei wird auf die verfügbare Hardware zugegriffen, um Informationen abzurufen. Auch kann durch diesen Layer auf das Anschließen von externen Geräten reagiert werden. Der *HAL* stellt standardisierte Interfaces zur Verfügung, die das *JAVA API* Framework benutzen kann, um verbaute Hardware zu benutzen. Daher besteht der *HAL* auch aus verschiedenen Bibliotheken (sog. Libraries). [12]

Android Runtime

Ab der Version “Lollipop” wurde eine eigens entwickelte Laufzeitumgebung namens “*ART*” (kurz für Android Runtime) in das Android *OS* integriert, die die Aufgaben der *DalvikVM* übernimmt.

Die *ART* kann nach den zwei Prinzipien ahead-of-time (*AOT*) und just-in-time (*JIT*) kompilieren. Zudem kommt als weiterer wichtiger Punkt eine optimierte Speicherverwaltung und ein verbesserter Debugging Support, welches alles Vorteile gegenüber der *DalvikVM* (Dalvik Virtual Machine) bietet.

Neben der Android Runtime wird auch eine Reihe von grundlegenden Core – Runtime - Libraries in Android mitgeliefert. Diese stellen Android die Funktionalitäten des *JAVA* Frameworks zur Verfügung.[13]

Native C / C++ Libraries

11 vgl. Android Developer Guide: Plattform Architecture **[ADG-ARCH]**

12 vgl. Android Developer Guide: Plattform Architecture **[ADG-ARCH]**

13 vgl. Android Developer Guide: Plattform Architecture **[ADG-ARCH]**

Die Komponenten aus dem *ART* bzw. *HAL* basieren auf nativen Code, der meist in C oder C++ geschrieben ist. Dieser native Code ist in Libraries zusammengefasst, welche in Android integriert sind. So können *APIs*, die auf dem Java Framework (wie zum Beispiel *OpenGLES*, *SQL* etc.) basieren, angesprochen und verwendet werden. Entwickler können diese nativen Bibliotheken über das Android *NDK* direkt ansprechen und verwenden. Ebenso können Hersteller ihre eigenen Bibliotheken zu diesem Layer hinzufügen.[14]

Java API Framework

Dieses Framework beinhaltet den vollen Funktionsumfang des Androids Betriebssystems. Es werden alle Komponenten, wie Activities oder auch Benachrichtigungen, als Grundlage für die App Entwicklung geliefert und somit eine Wiederverwendung von Code vereinfacht. Dieser Layer besteht aus einer Vielzahl von Modulen zu denen User – Interface - Elemente, ein Ressourcenmanager oder auch Manager um Inhalte zu verwalten gehören und viele mehr. [15]

Applications

Dieser Layer kann in zwei Komponenten untergliedert werden. Zu einem gibt es die System - Apps die Android von Haus aus mitliefert. Diese sind zum Beispiel der Homescreen, der Browser, Kontakte und auch das Telefon. Außerdem gibt es noch System - Apps die vom jeweiligen Smartphonehersteller zusätzlich zu den grundlegenden System - Apps hinzugefügt werden.

Aus Seite des Nutzers können zudem noch viele andere Apps Android hinzugefügt werden. Diese können selbst entwickelt sein oder einfach aus dem Play Store heruntergeladen werden. Mittlerweile gibt es im Google Play Store über 2,5 Millionen Apps, die sich ein Nutzer, teils kostenfrei teils kostenpflichtig, herunterladen und nutzen kann.[16]

14 vgl. Android Developer Guide: Plattform Architecture **[ADG-ARCH]**

15 vgl. Android Developer Guide: Plattform Architecture **[ADG-ARCH]**

16 vgl. Android Developer Guide: Plattform Architecture **[ADG-ARCH]**

2.2.2 Komponenten einer Android Applikation

Android ist ein Framework das auf verschiedenen Komponenten basiert. Diese Komponenten können individuell aufgerufen werden. So gibt es zum Beispiel Intents und
deren zugehörige Filter, Activities, Fragments, Loaders, Services, Content Providers, Broadcasts, Widgets und Threads. Im Nachfolgenden sollen wichtige Komponenten erklärt werden. Ausführliche Informationen zu den einzelnen Komponenten finden Sie in der Dokumentation von Android.

Activities

Hauptsächlich besteht eine App für Android aus Activities. Eine Activity stellt ein Fenster dar, mit welchem der Nutzer über Berührungen, wie zum Beispiel über einen Touch, Drag oder Pinch – to - Zoom, interagieren kann. Das Aussehen einer Activity wird über das entsprechende Layout - File definiert. Durch das Verwenden von mehreren Activities wird der grundsätzliche Aufbau der App erzeugt. Es ist zu beachten, dass immer nur eine Activity aktiv sein kann. Wird eine Activity gewechselt, so wird die vorangehende pausiert.[17]

Intents

Eine Activity wird über einen Intent aufgerufen, wobei ein Intent ein Objekt ist, dass Informationen über Aktionen einer anderen Komponente des Android Systems beinhaltet. Des Weiteren bilden Intents das Bindeglied zwischen mehreren Komponenten. Intents werden hauptsächlich genutzt um folgende Dinge auszuführen:

- eine Activity zu starten
- einen Service zu starten
- Nachrichten über System Events übermitteln (auch Versenden von Broadcasts)

[18]

17 vgl. IndividuApp, 19.03.2017: Ordnerstruktur eines Android Projekts **[INDIVIDUAPP]**

18 vgl. IndividuApp, 19.03.2017: Ordnerstruktur eines Android Projekts **[INDIVIDUAPP]**

Fragments

Fragmente sind Teile einer Activity und erlauben ein modulares Design der Applikation. Jedes Fragment besitzt dabei sein eigenes Layout, welches in *XML* geschrieben ist.

Es ist möglich Fragmente untereinander auszutauschen während man sich im Lebenszyklus einer Activity befindet. Zudem kann ein Fragment auch von mehreren Activities benutzt werden.

Die Einführung von Fragmenten ermöglicht es eine mehrteilige Benutzeroberfläche zu schaffen, die mit der alleinigen Verwendung von Activities nicht möglich war. Somit ist das Fragment ein wesentlicher Bestandteil des Android Frameworks.

Views, Layouts und andere Ressourcen

Views erzeugen die Ansicht von *UI* Elementen, wie zum Beispiel Buttons, Listen oder auch Formularen.Hinter den Views stehen jedoch die Layouts, welche im *XML* - Format geschrieben werden. Es gibt verschiedene Arten von Layouts, wie zum Beispiel das LinearLayout, das RelativeLayout oder auch das NavigationDrawer - Layout. Jede Layoutart hat ihre Eigenschaften auf die hier jedoch nicht eingegangen werden soll.

Layouts sind aber ein Teil der Ressourcen neben Bildern, Icons, Menüs oder auch Strings.

Content Providers

Diese Komponente ist dafür zuständig Inhalte von anderen Applikationen oder aus dem Dateisystem der App zur Verfügung zu stellen. Die Bereitstellung der Daten erfolgt über Datenmodelle durch das Bereitstellen einer URI. Durch die URI lässt sich ein Datensatz eindeutig identifizieren.[19]

19 vgl. IndividuApp, 19.03.2017: Ordnerstruktur eines Android Projekts **[INDIVIDUAPP]**

Android Manifest

Jede Applikation muss diese in *XML* geschriebene Datei in ihrem Root - Verzeichnis enthalten, da ohne sie ein Starten der Applikation nicht möglich ist. Die Datei muss als `AndroidManifest.xml` bezeichnet sein. Diese Datei enthält alle wichtigen Informationen, die zum Starten der App notwendig sind. Im AndroidManifest sind alle Activities deklariert. Ebenso wird hier festgelegt mit welcher Activity die App starten soll und welche Befugnisse die App benötigt. Zudem kann der Package - Name, das minimale und maximale *API* - Level festgelegt, sowie die Verwendung von Bibliotheken bekannt gegeben werden. [20]

2.2.3 Entwicklungsumgebung -Android Studio

Android Studio (Version 3.0.1) ist die offizielle Entwicklungsumgebung für Android Applikationen und löst die vorherige Umgebung Eclipse ab. Android Studio beinhaltet einen Code Editor und Entwicklertools, die es ermöglichen für alle Android Versionen bzw. Geräte wie Smartphones, Wear oder Android TV, gleichzeitig zu entwickeln. [21] Auch wird ein Emulator oder die Integration von diversen Programmen zur Versionskontrolle zur Verfügung gestellt. Android Studio bündelt damit alle benötigten Funktionen zur Entwicklung und zum Debugging von Apps.

Abbildung 2: Android Studio Logo

20 vgl. IndividuApp, 19.03.2017: Ordnerstruktur eines Android Projekts **[INDIVIDUAPP]**

21 vgl. J. Eason, 26.01.2015: An Update on Eclipse Android Developer Tools **[EASON]**

2.3 XML

Einer der bekanntesten Standards zur Auszeichnung von Datensätzen ist *XML*. Die Abkürzung ist abgeleitet von Extensible Markup Language und bedeutet soviel wie erweiterte Auszeichnungssprache. Mit *XML* können Daten strukturiert gespeichert und plattform - und implementationsunabhängig ausgetauscht werden. *XML* kann für die Darstellung verschiedenster Daten, wie zum Beispiel Dokumente, Bücher, Rechnungen oder auch Immobilien genutzt werden. Somit bietet *XML* eine weitreichende Verbreitung für die Verwendung von *XML* bei dem Austausch von Daten.[22]

XML ist jedoch nicht dafür zuständig die Darstellung der Daten zu übernehmen, das heißt es stellt nur die Struktur der Datensätzen zur Verfügung.

XML ist ein Text-basiertes Format, dass ähnlich wie bei *HTML* sogenannte Tags benutzt, um das in *UTF-8* codierte Dokument übersichtlich zu gestalten. Somit entsteht eine generische Syntax, die von Menschen gelesen und weiterverarbeitet werden können. Mit eigenen Programmen und Codes kann man die Inhalte und Daten innerhalb eines *XML* - Dokumentes verändern. Dabei ist es möglich auf eine Vielzahl freier Bibliotheken zurückzugreifen.

Da *XML* eine Meta – Markup - Sprache ist und als eXtensible (erweiterbar) gekennzeichnet wird, ist es also möglich Elemente zu erfinden, die sich an den Anwendungsbereich anpassen. Aufgrund dieser Erweiterbarkeit der Tags und Elemente lassen sich eine große Menge an verschiedenen Beschreibungen für Datensätze finden, die ein und das selbe Thema beschreiben. Daher haben sich einige Personen und Organisationen zusammengeschlossen und Standards für bestimmte Anwendungsgebiete entwickelt. Hier lässt sich das Beispiel OpenImmo anführen. Dieses wird im folgenden Kapitel näher betrachtet.

22 vgl. Homepage Webhilfe: XML & Co. Einführung **[WEBHILFE]**

2.3.1 XML - Standart “OpenImmo”

OpenImmo ist ein auf *XML* basierender Standard zur Beschreibung eines Datensatzes, der speziell für Immobilien entwickelt wurde. Es ist keine Software oder Schnittstelle. Der von der OpenImmo Initiative 2001 ins Leben gerufene Standard wird mittlerweile von allen gängigen Immobilienportalen und Maklersoftwares unterstützt. [23]

Der OpenImmo - Standard vereinfacht den Austausch von Daten zwischen den verschiedenen Immobilienportalen und Maklersoftwaren. Vor der Standardisierung war es so, dass jedes Portal bzw. jede Software sein eigenes Format genutzt hat, um die Daten von der Software an das Portal zu übergeben. Das heißt im Prinzip, dass eine Maklersoftware in mehrere Standards exportieren können musste, um alle Internet-portale bedienen zu können. Dies ist jedoch umständlich und Usability technisch in der heutigen Zeit nicht mehr denkbar. OpenImmo vereinfacht in diesem Sinne diesen Workflow, in dem es einen Standard bildet, der von allen Softwarelösungen und Internetportalen gelesen und verarbeitet werden kann.[24]

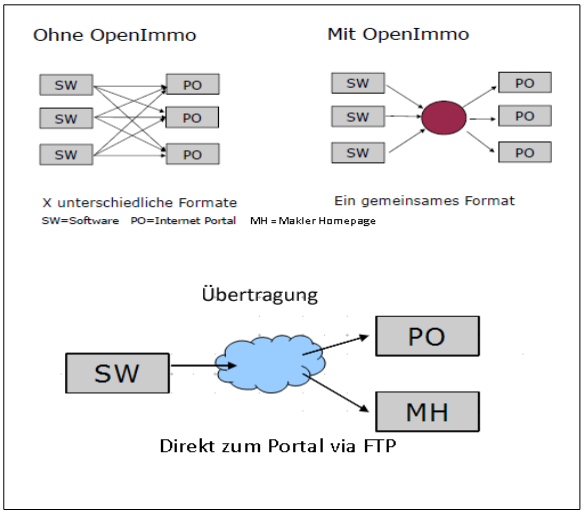

Abbildung 3: Prinzip OpenImmo

23 vgl. **[OPENIMMO1]**

24 vgl. **[OPENIMMO2]**

2.4 JSON

JSON ist eine Alternative zu *XML* und heißt ausgeschrieben JavaScriptObjectNotation. Ebenso wie *XML* ist *JSON* ein Datenformat indem Informationen, wie Objekte, Arrays und sonstige Variablen gespeichert werden.

JSON wird auch als schlank bezeichnet, weil es für den Menschen einfach zu lesen und zu schreiben ist. Aber auch Maschinen können *JSON* einfach lesen bzw. generieren.

Aus der Bezeichnung lässt sich erkennen, dass *JSON* auf einer Untermenge der Programmiersprache JavaScript basiert. Dabei folgt JSON nur den Konventionen der Programmiersprache JavaScript, ebenso wie denen von anderen aus der C - Familie bekannten Sprachen. Daher kann *JSON* in den meisten Programmiersprachen genutzt werden.

JSON kann folgende Datentypen verwenden:

- **Number:** kein Unterschied zwischen integer und float
- **Strings:** Zeichenketten
- **Boolean:** true oder false
- **Arrays:** geordnete Listen von Daten
- **Objekte:** untergeordnete Listen von Key/West-Paaren
- **Null:** Leeren Wert

JSON wird von Firebase benutzt um die Datenbankstruktur darzustellen. Mehr Informationen zu Firebase werden im nächsten Kapitel gegeben.

Abbildung 4: Auszug JSON Datei (Screenshot Firebase RealtimeDatabase)

2.5 Firebase

Firebase ist eine von vielen Möglichkeiten, um das Backend von Apps oder Web - applikationen zu realisieren. Als Plattform stellt Firebase eine breite Palette von Tools und Services zur Verfügung. Diese reichen von Authentifizierungsmöglichkeiten, Datenbanken bis hin zur Performance oder Crashreports.

Geschichtlich gesehen ist Firebase aus dem Startup Evolve im Jahre 2011 entstanden. Damals war es eine *API*, die Entwicklern die Möglichkeit bot, eine Online – Chat - Funktion in ihre Webseite einzubinden. Die Nutzer dieser *API* versendeten mit der Zeit jedoch immer mehr Anwendungsdaten über die *API*, was die Entwickler veranlasste die Chat - Funktion und die Echtzeit - Struktur zu trennen und Firebase zu gründen (April 2012). Kaum zwei Jahre später wurde das Unternehmen von Google

übernommen und entwickelte sich zu einem umfangreichen Service für App - und Webentwicklung.[25]

Firebase unterteilt sich in vier Gruppen aus Diensten:

- Develop (Entwickeln)
- Stability (Stabilität)
- Analytics (Analyse)
- Grow (Wachstum)

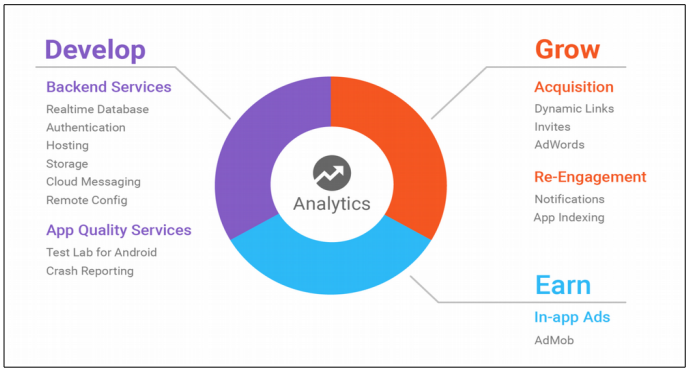

Abbildung 5: Firebase Dienste

2.5.1 FirebaseAuth

Mit Firebase können Nutzerregistrierungen und Loginfunktionalitäten ermöglicht werden. Dabei gibt es vorgefertigte *SDK*s und UserInterface - Bibliotheken, die der Entwickler neben eigenen Authentifizierungsmethoden verwenden kann. Durch die Bereitstellung dieser Funktion spart sich der Entwickler einen großen Anteil an Programmierleistungen, um ein eigenes Authentifizierungssystem einzurichten. Ebenfalls minimiert sich der Verwaltungsaufwand extrem, denn über das System können Konten zusammengeführt oder auch deaktiviert werden und das ohne großen Mehraufwand.

25 vgl. Steven Melendez, 27.05.2014, Sometimes You're Just One Hop From Something Huge **[FASTCOMPANY]**

Firebase bietet von Haus aus folgende Authentifizierungsmöglichkeiten:

- Anonym (Gast)
- E - Mail (Passwort)
- Facebook
- Telefonnummer
- Twitter
- Google
- GitHub

Die Firebase Authentifizierung basiert auf oAuth2.0 und OpenConectID, daher ist eine Einbindung in das eigens entwickelte Backend problemlos möglich[26].

Abbildung 6: Firebase Authentifizierungs - Möglichkeiten

Wie authentifiziert sich ein Nutzer?

Um einen Nutzer zu authentifizieren benötigt der Entwickler die Anmeldedaten von diesem. Diese bestehen meistens aus einer Email - Adresse, einem Passwort oder auch einem oAuth - Token des externen Providers (Facebook, Google etc.). Diese Daten werden dann der Authentifikation *SDK* von Firebase übergeben und dann vom Firebase Backend verifiziert. Ist dies erfolgreich, können Daten von anderen Firebase Komponenten bezogen, bearbeitet oder auch gelöscht werden.[27]

26 vgl. Firebase **[FIREBASEAUTH]**

27 vgl. Firebase **[FIREBASEAUTH]**

2.5.2 Firebase RealtimeDatabase

Die Firebase Datenbank unterscheidet sich von der wohl weit verbreitetsten Variante von Datenbanken wohl darin, dass Firebase eine *NoSQL*-Datenbank ist. Sie wird in einer Cloud gehostet und ermöglicht damit eine Speicherung und Synchronisation von Daten in Echtzeit.

Da Firebase auf der *NoSQL* - Datenbank setzt, werden die Daten *JSON* - Format gespeichert.

Abbildung 7: Screenshot Realtime Database

Die Vorteile der Datenbank liegen daher in den folgenden Punkten:

- **Datenverwaltung in Echtzeit:**

  Wenn ein Datensatz verändert wird, synchronisiert die Datenbank den Datensatz innerhalb von Millisekunden mit allen verbundenen Usern. Dies ist möglich, da Firebase auf *HTTP* - Requests verzichtet.

- **Offline**

  Das Database *SDK* ist so konzipiert, dass Daten auch ohne Verbindung gespeichert werden können. Diese Daten werden im Offline - Modus auf einem Datenträger abgelegt. Sobald es eine aktive Verbindung gibt, werden die Daten synchronisiert und dem Nutzer zur Verfügung gestellt.,

- **Zugänglichkeit**

  Um auf die Datenbanken zuzugreifen, ist kein Anwendungsserver notwendig. Das heißt, auf die Datenbank kann jeder Zeit ein Zugriff über einen Webbrowser oder ein mobiles Gerät geschehen. Dabei ist der Zugriff über eine Definition von Echtzeit - Datenbanksicherheitsregeln geschützt. Diese können individuell angepasst werden und ermöglichen damit die volle Kontrolle über die Datenbank sowie deren Schreib - und Leserechte.

- **Skalierung über mehrere Datenbanken**

  Nutzt man die kostenpflichtige Variante von Firebase kann der Entwickler den Datenbedarf der App reduzieren, indem die Daten in verschiedenen Datenbanken im Firebaseprojekt aufgeteilt werden.

[28]

2.5.3 FirebaseStorage

Natürlich möchte ein User auch Bilder, Videos und andere Dateien in der Applikation hochladen und verwenden. Um dies zu ermöglichen wird FirebaseStorage eingesetzt. Der FirebaseStorage ist ein leistungsfähiger, einfacher und kosteneffektiver Objektspeicherdienst, der für die Google Skalierung entwickelt wurde. Firebase verwendet ein einfaches Verzeichnis - / Dateisystem, um seine Daten zu strukturieren.

Vorteile des FirebaseStorage sind robuste Operationen, starke Sicherheit und eine hohe Skalierbarkeit. Zudem werden Up- und Downloads unabhängig von der

28 vgl. Firebase **[FIREBASEDB]**

Netzwerkqualität durchgeführt. Wird einer dieser Prozesse zwischendurch gestoppt, startet der Prozess bei einem neuen Versuch an der Stelle, wo er unterbrochen wurde. Diese Funktionalität ermöglicht dem Nutzer eine Einsparung von wertvoller Zeit und Bandbreite. [29]

3 IST- Stand

Bevor mit der Konzeption der Applikation begonnen werden konnte, musste eine Betrachtung des Worklfows zur Erstellung von Immobilien - Exposés durchgeführt werden. Ein Immobilienmakler verwaltet seine Objekte (Immobilien) heutzutage in einer Maklersoftware. Es gibt dabei eine Vielzahl von Softwarelösungen, die sich diesem Problem annehmen. Die gängigsten Softwarelösungen sind in Abbildung 7 zu sehen.

Abbildung 8: Übersicht Marklersoftwares

Jede dieser Softwarelösungen hat seine Vor - und Nachteile, die hier jedoch nicht betrachtet werden sollen. Jedoch ist es wichtig zu verstehen, dass diese Software das Hauptarbeitsmittel in einem Maklerbüro ist. Denn mit dieser kann der Immobilienmakler seine Objekte verwalten, das heißt neue Objekte erstellen, sie bearbeiten, sie als vergeben markieren oder auch löschen. Zudem ist es bei den meisten Software - lösungen möglich, die Daten aus der Software direkt in die Portale „ImmobilienScout24“, „immowelt.de“ oder „immowelt.net“ zu exportieren und seine Daten abzugleichen.

29 vgl. Firebase **[FIREBASESTORAGE]**

Der Makler erstellt nun sein Objekt mit allen notwendigen Daten und Bildern in der Software. Er kann sie über die Software seiner Wahl verwalten, bearbeiten und löschen. Ebenso kann er Termine, die das Objekt betreffen, direkt in der Software eintragen und somit verknüpfen.

Bei der Einrichtung der Software kann der Makler diese mit den verschiedensten Immobilienportalen aus dem World Wide Web verknüpfen. Dazu ist ein Nutzerkonto im jeweiligen Portal von Nöten. Mit diesen Nutzerdaten kann der Immobilienmakler dann die Verknüpfung für den Export und Import von Daten herstellen. Ist diese Verknüpfung hergestellt, können nach Erstellung des Exposés die Daten ohne große Mühen direkt in das Portal übertragen werden.

Bei der Erstellung von Exposés trägt der Makler Daten aus den verschiedensten Quellen zusammen. Um dies zu verdeutlichen, soll hier ein Beispiel angebracht werden, auf welches sich auch die folgende Konzeption der Applikation (siehe Kapitel 4) beziehen wird.

Bei Besichtigung einer Immobilie erhält der Makler für den weiteren Verkauf/ Vermietung folgende Dinge:

- Pläne mit Grundrissen in ausgedruckter Form
- Grundsätzliche Eckdaten zur Immobilie in schriftlicher Form

Zusätzlich zu diesen Dingen werden Fotos der Immobilie erstellt, die sich demnach auf der Kamera des Maklers befinden. Ebenso wird sich der Makler Notizen zu entsprechenden Merkmalen/ Besonderheiten der Immobilie machen, um diese dann möglichst gut beschreiben und weiter vermitteln zu können. Dies geschieht alles unter Einsatz verschiedener Medien.

Diese Daten fügt der Makler im Büro anschließend zu einem Exposé zusammen und pflegt sie in die von ihm gewählte Softwarelösung ein.

4 Konzeption

Wie in Kapitel 3 beschrieben, kreist der momentane Workflow eines Immobilienmaklers um die von ihm verwendete Maklersoftware. Der hier entstehende Prototyp der App soll dem Nutzer die Arbeit von unterwegs aus erleichtern.

Es soll dem Nutzer die Möglichkeit gegeben werden alle relevanten Daten zu einer Immobilie direkt über die App in das Portal zu stellen. Ebenfalls soll später eine direkte Veröffentlichung des Exposés auf der Webseite des Nutzers stattfinden. Die Daten sollen sich in allen drei Instanzen (Webseite, Immobilienportal und Applikation) in Echtzeit synchronisieren.

In dieser Bachelorarbeit wird nur ein Teil der Umsetzung dieses Projektes, nämlich die Erstellung eines Prototypen der App, aufgrund der begrenzten Zeit, betrachtet. Daher beziehen sich alle weiteren Kapitel auf die Konzeption bzw. die Erstellung des Prototypen der App.

Dabei ist die App so aufgebaut, dass der Makler, wenn er eine Immobilie besichtigt, gleich alle Daten via App aufnehmen und speichern kann. Ebenso kann er über die Applikation Anhänge, wie Bilder oder auch *PDF* – Dateien einem Exposé hinzufügen. Da es sich um einen Prototypen handelt, sollen in der zu entstehenden Applikation auch nur grundlegende Funktionalitäten implementiert sein. Diese Funktionalitäten sind in Kapitel 5 definiert wurden.

5 SOLL- Stand

5.1 Allgemeine Anforderungsliste

Nr	Anforderung	Beschreibung
1	Umsetzung der App für die Android Plattform	
2	Splashscreen	Für die Benutzerfreundlichkeit und das allgemeine Design wird beim Start der App ein sogenannter Splashscreen verwendet, um den Nutzer in der App willkommen zu heißen.
3	Loginscreen, Registrieren, Passwort vergessen	Der Nutzer kann sich über die Loginseite bei der App anmelden/registrieren und dadurch die Applikation nutzen. Ebenso kann er sich über eine Passwort vergessen – Funktion ein neues Passwort zusenden lassen.
4	Menü	Die Applikation soll ein voll funktionsfähiges ausklappbares Menü besitzen. Die Menü-Struktur soll möglichst einfach und übersichtlich gehalten werden.
5	Exposé	In diesem Bereich können Exposés erstellt, bearbeitet oder auch gelöscht werden. Für die Anzeige soll eine Übersichtseite, sowie Detailansichten für die einzelnen Immobilen entstehen.
6	Mein Account	In dieser Ansicht werden relevante Informationen zu dem Benutzeraccount übersichtlich dargestellt. Der Nutzer hat die Möglichkeit hier sein Profilbild oder ein neues Passwort festzulegen. Ebenso kann er hier auch die Email-Adresse ändern mit der er sich registriert hat.
7	Logout	Der Nutzer erhält die Möglichkeit sich aus seinem Account auszuloggen.

Tabelle 1: Tabelle funktionaler Anforderungen an die App

Nr	Anforderung	Beschreibung
1	Einhaltung der Design Guidelines	Die Designrichtlinien der Android Plattform sollen eingehalten werden.
2	Usability	Die App sollte einfach und verständlich aufgebaut werden.
3	Feedback	Die App sollte dem Nutzer unter bestimmten Bedingungen ein Feedback geben.
4	Selbsterklärbarkeit	Die App sollte selbsterklärend aufgebaut sein. Das heißt der Nutzer sollte keine Probleme beim Verständnis der Funktionen oder Icons haben.
5	Benutzerfreundlichkeit	Die App sollte benutzerfreundlich aufgebaut sein. Das heißt der Nutzer sollte mit der Höflichkeitsform angesprochen werden
6	Verfügbarkeit	Die App sollte in jeder Auflösung darstellbar sein. Zum Beispiel sollte die App auch für Tablets angeboten werden.
7	Robustheit	Die App sollte gegenüber Fehleingaben robust sein.

Tabelle 2: Tabelle nichtfunktionaler Anforderungen

5.2 Anwendungsfälle

Exposé erstellen

Ziel:	Ein neues Exposé ist erstellt
Bedingung:	Benutzer muss eingeloggt sein.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Exposé hinzufügen“ • Formular zum Erstellen eines neuen Exposé-Items öffnet sich • Nutzer gibt alle relevanten Daten in das Formular ein • Nach Klick auf „Speichern“-Symbol speichert das System die Daten in der Datenbank ab
Ergebnis:	Das neue Exposé ist in der App erstellt wurden.

Exposé in der Übersichtseite ansehen

Ziel:	Alle erstellten Exposés werden in der Übersichtsseiten angezeigt
Bedingung:	Benutzer muss eingeloggt sein.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Meine Exposés“ • Alle verfügbaren Exposés werden angezeigt
Ergebnis	Anzeige aller Exposés in einer Listenansicht. Sollten keine Exposés in der Datenbank vorhanden sein wird eine alternative Ansicht angezeigt.

Exposé im Detail ansehen

Ziel:	Ein spezifisches Exposé-Item wird angezeigt.
Bedingung:	Benutzer muss eingeloggt sein. Es sind Exposés in der Datenbank vorhanden.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Mein Exposés“ • Nutzer sucht Exposé • Nutzer „wischt“ von rechts nach links über das Exposé • Nutzer klickt auf „Anzeigen“
Ergebnis	Anzeige des ausgewählten Exposé-Items mit all seinen Details.

Exposé bearbeiten

Ziel:	Ein bestehendes Exposé ist verändert/bearbeitet wurden.
Bedingung:	Benutzer muss eingeloggt sein. Es sind Exposés in der Datenbank vorhanden.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Meine Exposés“ • Suche des abzuändernden Exposés aus der Liste • Nutzer „wischt“ von rechts nach links über das Exposé • Klick „Bearbeiten“ • Nutzer ändert alle relevanten Daten im Formular • Klick auf Speichern
Ergebnis:	Spezifische Daten des Exposés wurden nach Belieben verändert und in der Datenbank gespeichert.

Exposé löschen

Ziel:	Ein spezifisches Exposé-Item wird gelöscht.
Bedingung:	Benutzer muss eingeloggt sein. Es sind Exposés in der Datenbank vorhanden.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Meine Exposés“ • Suche des zu löschenden Exposés aus der Liste • Nutzer „wischt“ von rechts nach links über das Exposé • Nutzer klickt „Löschen“
Ergebnis:	Ausgewähltes Exposé- Item wird in der Datenbank gelöscht und aus der Übersichtseite entfernt.

Anhänge einem bestehenden Exposé hinzufügen

Ziel: Einem bestehenden Exposé wird ein Anhang hinzugefügt.

Bedingung: Benutzer muss eingeloggt sein. Es sind Exposés in der Datenbank vorhanden.

Ablauf:

- Nutzer öffnet das Menü
- Nutzer wählt Punkt „Meine Exposés“
- Suche des Exposés aus der Liste
- Öffnen der Detailansicht durch einen Klick auf das Exposé
- Nutzer klickt „Bilder“ / „PDF“ - Tab
- Nutzer klickt Button „Bild hinzufügen“ / „PDF hinzufügen“
- Nutzer erstellt den Anhang durch Ausfüllen des Formulars
- Nutzer fügt den Anhang über den Button „Bild hinzufügen“/ „PDF hinzufügen“ dem Exposé hinzu

Ergebnis: Einem bestehenden Exposé wird ein Anhang hinzugefügt.

Anhänge aus einem bestehenden Exposé entfernen

Ziel: Einem bestehenden Exposé wird ein Anhang hinzugefügt.

Bedingung: Benutzer muss eingeloggt sein. Es sind Exposés in der Datenbank vorhanden.

Ablauf:

- Nutzer öffnet das Menü
- Nutzer wählt Punkt „Meine Exposés“
- Suche des Exposés aus der Liste
- Öffnen der Detailansicht durch wischen von rechts nach links und Klick auf „Anzeigen“
- Nutzer klickt „Bilder“ / „PDF“ - Tab
- Nutzer sucht zu löschenden Anhang
- Nutzer wischt vom rechten Rand nach Links auf dem gewählten Anhang
- Durch einen Klick auf „Löschen“ wird der Anhang aus dem Exposé entfernt

Ergebnis: Einem bestehenden Exposé wird ein Anhang hinzugefügt.

Profilbild hochladen/ ändern

Ziel:	Ein Profilbild wurde hochgeladen oder geändert.
Bedingung:	Benutzer muss eingeloggt sein.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Mein Account“ • Nutzer gibt klickt auf das alte Profilbild • Nutzer wählt neues Profilbild aus • Nutzer klickt auf „Profilbild ändern“
Ergebnis:	Profilbild ist hochgeladen bzw. geändert wurden.

Email-Adresse ändern

Ziel:	Die Email-Adresse des Benutzeraccounts wurde geändert.
Bedingung:	Benutzer muss eingeloggt sein.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Mein Account“ • Nutzer gibt neue Email-Adresse in Textfeld ein • Nutzer bestätigt Email-Adresse durch klick auf „Email ändern“
Ergebnis:	Die E-Mail-Adresse mit der sich der User einloggt wurde geändert.

Passwort ändern

Ziel:	Das Passwort des Benutzeraccounts wurde geändert.
Bedingung:	Benutzer muss eingeloggt sein.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Mein Account“ • Nutzer gibt neues Passwort in Textfeld ein • Nutzer bestätigt Passwortänderung durch klick auf „Passwort ändern“
Ergebnis:	Das Passwort mit welchem sich der User einloggt wurde geändert.

Benutzeraccount löschen

Ziel:	Der Benutzeraccount wurde aus dem System entfernt.
Bedingung:	Benutzer muss eingeloggt sein.
Ablauf:	• Nutzer öffnet das Menü • Nutzer wählt Punkt „Mein Account“ • Nutzer klickt auf „Account löschen“ • Nutzer wir zur Bestätigungsansicht weitergeleitet • Nutzer klickt auf „Ja, ich möchte meinen Account löschen“
Ergebnis:	Benutzeraccount wurde gelöscht. Der Nutzer kann sich nicht mehr mit seinen alten Daten anmelden.

6 Umsetzung

Im praktischen Teil der vorliegenden Bachelorarbeit soll mit der Umsetzung für die App begonnen werden. Aufgrund der begrenzten Bearbeitungszeit wird der Umfang der umzusetzenden Funktionen jedoch eingeschränkt. Der Ablauf des praktischen Teils soll anhand einer Umsetzungsstrategie geplant und durchgeführt werden.

Die Umsetzung erfolgt für die Plattform Android. Gründe für die Wahl dieses Betriebssystems liegen in dem Vorhandensein von Kenntnissen in diesem Bereich. Es wurde jedoch vorher überlegt, ob die App crossmedial entwickelt werden sollte, jedoch wäre die Einarbeitungszeit in die jeweiligen Frameworks zu aufwendig gewesen, daher die Entscheidung für Android.

Eventuelle Grundlagen für die Entwicklung für Android sind in Kapitel 2 erläutert. Nach der Erläuterung der Strategie werden Informationen zur Implementierung einzelner Funktionen und grundständiger Dinge gegeben.

6.1 Strategie

Die Umsetzungsstrategie basiert auf der Anforderungsliste und den Anwendungsfällen, die im Kapitel 5 definiert wurden.

So sollen im ersten Schritt allgemeine Dinge, wie Layouts, das SplashScreen sowie die Navigation in der App umgesetzt werden. Im Nachgang ist die Einbindung von Firebase ein wichtiger Schritt. Dazu muss in der Firebase - Konsole ein Projekt angelegt und die *SDK*s in das Android Projekt eingebunden werden. Nachdem alle Firebase -
Komponenten eingebunden sind, werden die Funktionalitäten wie die Registrierung, Authentifizierung und der Login Vorgang implementiert. Ist dies erfolgreich abgeschlossen, wird die Erstellung von neuen Exposés in Angriff genommen. Hier wird sich nur auf reine Exposédaten beschränkt. Anhänge bleiben vorerst außen vor. Können Exposés erfolgreich erstellt werden, müssen sie angezeigt werden. Dies erfolgt zu einem in einer Übersichtsseite und zum anderen individuell, das heißt jedes Exposé in einer Detailansicht.

Im nächsten Schritt soll die Funktion des Bearbeitens realisiert werden. Kann der Nutzer Exposés bearbeiten, so muss er diese auch aus dem System löschen können. Ist das Löschen von Exposés eine fest implementierte Funktion, kann sich den Attachments (Anhängen) eines Exposés gewidmet werden. Diese müssen natürlich erst dem Exposé und in den FirebaseStorage sowie in die Datenbank hinzugefügt werden. Danach müssen die Anhänge in den Detailansichten sichtbar gemacht werden. Des Weiteren müssen Anhänge auch gelöscht werden können. Im allerletzten Schritt der Umsetzung dieses Prototypen ist die Optimierung des verwendeten Codes.

6.2 Implementierung

In diesem Kapitel werden genauere Informationen zur Implementierung der einzelnen Anforderungen gegeben. Bei der Implementierung wurde sich an die definierten Anforderungen und die festgelegte Strategie (siehe Kapitel 6.1.) gehalten. Der Code für die nachfolgenden Kapitel befindet sich auf der CD (siehe Anhang 1)

6.2.1 Setup Firebase

Da im Projekt Komponenten der Anwendungsentwicklungsplattform Firebase zum Einsatz kommen, müssen diese vor der Verwendung in das Android - Projekt eingebunden werden.

Um Firebase einem Android-Studio Projekt hinzuzufügen, wird die Firebase - Konsole verwendet. Diese befindet sich online unter der folgenden Webadresse:

`http://console.firebase.google.com`

Link zur Firebase Konsole

Dort wird dann ein neues Projekt für die App angelegt. Im Falle dieser Bachelorarbeit hieß das Projekt „MyImmoExpose“. Beim Anlegen des Projektes bekommt dieses eine ID zugewiesen. Nach dem Klick auf „Projekt erstellen“ öffnet sich das Dashboard. Von da aus muss dem Projekt eine Android-App hinzugefügt werden.

Um dies zu tun, wurde der Android Paketname (hier: *com.webgalaxie.blischke.myimmoexpose*) angegeben. Im nächsten Schritt erzeugt die Konsole eine „google-services.json“ - Datei, die der App hinzugefügt werden muss. Dabei ist wichtig, dass die Datei zuerst heruntergeladen und dann in den *app/* Ordner verschoben wird. Ist dies erfolgt, werden nun die *SDKs* dem Gradle hinzugefügt. Dies passiert auf zwei Ebenen des Gradles.

- **Im Root-Level:**

  Im Root-Level/des Gradels muss die Abhängigkeit der Google Services und auch das Maven Repository deklariert werden.

```
// Top-level build file where you can add configuration options common to all
sub-projects/modules.

buildscript {

    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.0.1'
        // google-services plugin
        classpath 'com.google.gms:google-services:3.1.2'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
allprojects {
    repositories {
        google()
        jcenter()
        maven{
            url "https://maven.google.com" // Google's Maven repository
        }
        maven { url "https://jitpack.io"  //required for UCrop
        }
    }
}
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Code 1: Project Gradle

- **im Modul-Level:**

  Im Gradle auf dem Modul-Level werden dagegen die einzelnen Libaries, die Firebase zur Verfügung stellt, deklariert. Für dieses Projekt müssen die folgenden Bibliotheken im Gradle kompiliert werden. Zusätzlich muss am Ende des Modul Gradles noch folgende Codezeile eingefügt werden, damit die Google-Services zur Anwendung kommen können.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.webgalaxie.blischke.bachelortakesix"
        minSdkVersion 21
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
       testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        ...
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    ...
    //Add Firebase Libaries
    compile 'com.google.firebase:firebase-auth:11.8.0'
    compile 'com.google.firebase:firebase-database:11.8.0'
    compile 'com.google.firebase:firebase-core:11.8.0'
    compile 'com.google.firebase:firebase-storage:11.8.0'
    ...
}
apply plugin: 'com.google.gms.google-services'
```

Code 2: Module Gradle

6.2.2 SplashScreen

Ein SplashScreen ist ein graphischer Platzhalter, der meist beim Start einer App oder Computeranwendung für wenige Sekunden sichtbar ist. Er soll einen „angenehmen“ Start in das Nutzungserlebnis der App bieten.

Abbildung 9: Screenshot SplashScreen

Die Implementierung des SplashScreens erfolgt über die Activity *SplashScreen.java*. In der zugehörigen Layoutdatei *splashscreen.xml* im *res/layout* Ordner des Projektes wird das Layout festgelegt. Dieses besteht aus drei separaten Teilen, die mittels der Activity animiert werden. Für die Animation sind zwei Animations- XMLs erstellt worden (*uptodown.xml* und *downtoup.xml*). Um die Elemente zu animieren müssen in der Activity die Layoutelemente definiert werden. Dafür werden zu einem die Relativlayouts und auch die Animationen als Variablen deklariert.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:duration="800"
        android:fromYDelta="100%p"
        android:toYDelta="0%p" />
</set>
```

Code 3: XML für die Animation uptodown

In der *onCreate()* - Methode werden die Variablen dann nachdem das Layout gesetzt wurde über die Methode *findViewById()* den Elementen in der Layout-XML zugeordnet.

Um die Animationen zu laden wird die Funktion *loadAnimation()* der Klasse *AnimationUtils* verwendet. Folglich erhält der Entwickler ein Animationsobjekt, welches mittels der Methode *setAnimation()* dem Layoutelement zugeordnet werden muss.

Da ein SplashScreen eigentlich nur ein graphischer Platzhalter ist und nur für kurze Zeit beim Start eines Computerprogramms oder in diesem Falle einer App zur Anzeige kommt, wird dieser mittels eines Threads nur für 3 Sekunden angezeigt. Ein Thread ist ein Aktivitätsprozess, der ein Teil des Gesamtprozesses eines Programms ist. Er kann mittels *sleep()* für eine definierte Anzahl an Millisekunden pausiert werden.

Für die Implementierung des SplashScreens wird also ein Thread *splashThread* erstellt. In der *run()* – Methode des Threads wird zuerst der Thread pausiert und danach die folgende Activity (hier *LoginScreen.java)* ausgeführt. Im Anschluss muss der Thread nur noch über die Methode *start()* ausgeführt werden.

```
        // add logic for the splashscreen
        // splash screen will change to LoginActivity after 3000 milliseconds
        Thread splashThread = new Thread(){
           @Override
           public void run() {
             try{
                  // pause the thread for 3 seconds
                  sleep(3000);
                  // finish the current activity
                  finish();
                  // start the new activity
              startActivity(new Intent(getApplicationContext(),LoginScreen.class));
              }catch (InterruptedException e){
                  // if there was an error print it out
                  e.printStackTrace();
              }
           }
      };
```

Code 4: Anzeigen des SplashScreens über einen Thread in der SplashScreen.java

6.2.3 Registrierung

Damit ein Nutzer die App verwenden kann, muss dieser sich registrieren. Dies wird über den Activity Register Screen ermöglicht. Für die Registrierung müssen die folgenden Firebase Komponenten initialisiert werden: *FirebaseAuth* und *FirebaseUser.*

In der *onCreate()* - Methode werden die Viewelemente initialisiert und die onClick()- Events für den Registrieren-Button und den Back-to-login-Button definiert. Bei einem Klick auf den Back-to-Login-Button wird über einen Intent der LoginScreen aufgerufen.

Dagegen wird bei einem Klick auf den Registrieren-Button die *registerUser()* - Methode aufgerufen. In dieser Methode werden die Eingabe aus den Eingabefeldern in Strings umgewandelt. Danach wird geprüft, ob die Felder leer sind. Falls dies der Fall ist, wird ein Fehler angezeigt und der Focus auf das Feld gesetzt.

Ebenfalls wird geprüft, ob die Eingabe dem Email-Format entspricht und das Passwort eine Mindestlänge von 6 Zeichen besitzt. Sind die Eingabefelder nicht leer und erfüllen alle Anforderungen wird ein ProgressDialog angezeigt und die Registrierung durchgeführt.

Um die Registrierungsmethode *createUserWithEmailAndPasswort()* zu verwenden, muss eine Instanz der FirebaseAuthentifizierung erzeugt werden. Der Methodenaufruf erfolgt dann entsprechend den folgenden Codes:

```
1  // FirebaseAuth Instanz erzeugen
2  private FirebaseAuth mAuth;
3
4
5  // Nutzer registrieren
6  mAuth.createUserWithEmailAndPassword(email, password)
7      .addOnCompleteListener(this, new OnCompleteListener < AuthResult > () {
8          @Override
9          public void onComplete(@NonNull Task < AuthResult > task) {
10             if (task.isSuccessful()) {
11                 // Sign in success, update UI with the signed-in user's information
12                 Log.d(TAG, "createUserWithEmail:success");
13                 FirebaseUser user = mAuth.getCurrentUser();
14
15             } else {
16                 // If sign in fails, display a message to the user.
17                 Log.w(TAG, "createUserWithEmail:failure", task.getException());
18                 Toast.makeText(EmailPasswordActivity.this, "Authentication failed.",
19                     Toast.LENGTH_SHORT).show();
20
21             }
22             // ...
23         }
24     });
```

Code 5: Registrierung eines Users

Als Parameter werden dabei die E-Mail und das Passwort übergeben. Der `AddonCompleteListener` wird der Methode angefügt, um das Ergebnis der Registrierung zu verarbeiten.

Als erstes wird der Progress Dialog ausgeblendet. Danach erfolgt eine Überprüfung, ob die Anmeldung erfolgreich war oder nicht. Bei Erfolg wird der Nutzer automatisch eingeloggt. Zusätzlich werden die Daten aus den Textfeldern über die Methode `putDateInDB()` in die RealtimeDatabase unter der User ID geschrieben. Ist die Registrierung nicht erfolgreich, wird ein Fehler ausgegeben.

Um in die Datenbank schreiben zu können, müssen in der Methode `putDataInDB()` zuerst die Eingaben aus den Textfeldern zu Strings umgewandelt werden. Dafür wird eine Datenbank Instanz erzeugt und in einer Referenz der Knotenpunkt „Users“ angelegt. In „Users“ werden nun alle Nutzer unter ihrer UUID gespeichert.

Um dem User, die entsprechenden Wortpaare zuordnen zu können, müssen die Values zu den Childnodes gesetzt werden. Dabei ist der erste Child-Value der sogenannte Key des Werkpaares, während die `setValue()` - Methode den Wert des Paares setzt.

```java
private void putDataInDB() {
        // Daten aus den Texteingabefeldern in Strings schreiben
        String name = nameRegisterInput.getText().toString();
        String email = emailRegisterInput.getText().toString();
        String passwort = passworRegisterInput.getText().toString();

        // Aktuellen Nutzer ermitteln
        user = auth.getCurrentUser();

        // Datenbank Referenz erzeugen
        DatabaseReference database = FirebaseDatabase.getInstance().getReference().child("Users")
        // neue Datenbank Referenz erzeugen, die die UUID als Unterpunkt hat
        DatabaseReference currentUserDB = database.child(auth.getCurrentUser().getUid());

        // Daten aus den Eingabefeldern in die Datenbank schreiben
        currentUserDB.child("name").setValue(name);
        currentUserDB.child("email").setValue(email);
        currentUserDB.child("passwort").setValue(passwort);
        currentUserDB.child("image").setValue(Constants.DEFAULT_USER_PROFIL_PICTURE_URL);
    }
```

Code 6: putDatainDB() - Methode in der Registrierungs- Activity

In dem Fragment „My Account“ können diese Values im Nachhinein verändert werden und ebenso ein Profilbild für den jeweiligen Account gesetzt werden. Mehr dazu in Kapitel 6.2.15.

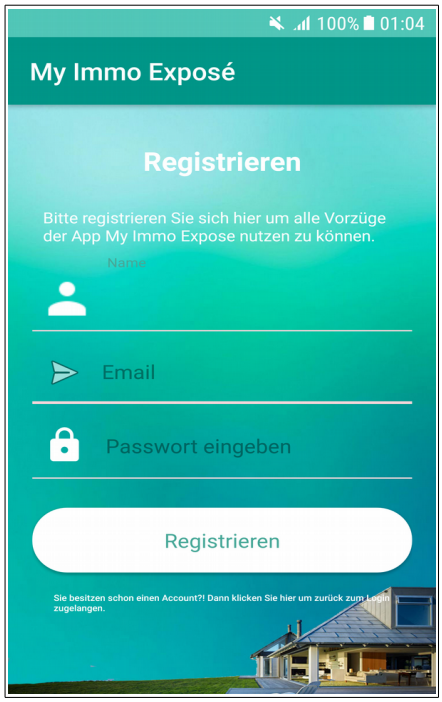

Abbildung 10: Screenshot Registerungsansicht

6.2.4 Login

Da der SplashScreen auf die Login Activity verweist, wurde nach der Programmierung des SplashScreens der LoginScreen implementiert. Dazu wurde zunächst das Layout erstellt. In der zugehörigen Activity wurden alle für die Funktionalität relevanten Elemente referenziert und instanziiert.

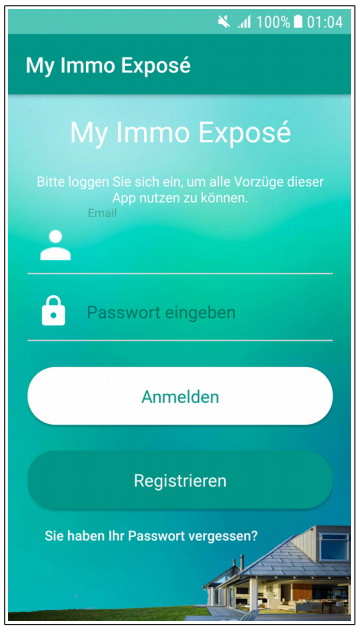

Abbildung 11: Screenshot Loginansicht

Da der Login über die `FirebaseAuth` erfolgt, wurden zusätzlich die Objekte `FirebaseAuth` und `FirebaserUser` deklariert.

In der `onCreate()` - Methode der Activity wird zuerst eine Instanz der Authentifizierung erzeugt. Danach muss überprüft werden, ob der Nutzer schon oder noch angemeldet ist. Falls das der Fall ist wird der Nutzer direkt in das Dashboard der App weiter geleitet.

Um den eigentlichen Loginvorgang zu realisierten, wird auf dem Login-Button ein `onClickListener()` implementiert. In diesem werden die Inhalte aus der Texteingabe in Strings umgewandelt. Ebenso wird überprüft, ob die Eingabefelder leer sind. Falls diese leer sein sollten, wird ein Fehler angezeigt und der Nutzer aufgefordert, diesen zu korrigieren.

Nach dieser Überprüfung folgt die eigentliche Logik des Logins. Da Firebase eine Reihe von Loginmöglichkeiten zur Verfügung stellt (Facebook, Google, Gastlogin, Email-Passwort-Login etc.) gibt es verschiedene Methoden den Login durchzuführen. Der einfachheitshalber wurde sich hier auf den Login mit Email und Passwort beschränkt.

Mit `auth.signInWithEmailAndPasswort(mail, passwort)` wird der User eingeloggt. Um die Weiterleitung zum Dashboard zu ermöglichen, benötigt man einen `onCompleteListener()`.

Dieser überprüft, ob der Login erfolgreich war. Falls nicht wird ein Fehler ausgegeben. Bei einem erfolgreichen Login wird die Weiterleitung auf das Dashboard vorgenommen. All diese Schritte werden über die folgenden Codezeilen realisiert.

Des weiteren erfolgen in dieser Activity noch die Weiterleitung für die Registrierung und die „Passwort vergessen“ Funktion. Diese werden in der `onCreate()` - Methode der Activity deklariert.

```
1  // FirebaseAuth Instanz erzeugen
2  private FirebaseAuth mAuth;
3
4  // Beispiel Logindaten
5  String email ="example@example.com"
6  String passwort ="123456"
7
8  // Nutzer registrieren
9  mAuth.signInWithEmailAndPassword(email, password)
10         .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
11             @Override
12             public void onComplete(@NonNull Task<AuthResult> task) {
13                 if (task.isSuccessful()) {
14                     // Sign in success, update UI with the signed-in user's information
15                     Log.d(TAG, "signInWithEmail:success");
16                     FirebaseUser user = mAuth.getCurrentUser();
17                     updateUI(user);
18                 } else {
19                     // If sign in fails, display a message to the user.
20                     Log.w(TAG, "signInWithEmail:failure", task.getException());
21                     Toast.makeText(EmailPasswordActivity.this, "Authentication failed.",
22                             Toast.LENGTH_SHORT).show();
23                     updateUI(null);
24                 }
25
26                 // ...
27             }
28         });
29
```

Code 7: Login User

6.2.5 Passwort vergessen

Über diese Funktion kann sich der Nutzer ein neues Passwort an seine Email zusenden lassen. Die Funktion kann über das Loginformular durch einen Klick auf „Sie haben ihr Passwort vergessen?“ aufgerufen werden.

Der Nutzer wird danach aufgefordert die E-Mail in ein Feld einzugeben.Diese Texteingabe wird von der Activity *ForgetPasswortScreen.java* weiter verarbeitet. Dies passiert über die Methode *sendPasswortResetEmail()*. Diese Methode erfordert eine Instanz der Firebase Authentifizierung und einen Übergabeparameter in Form der Emailadresse.

```
1  FirebaseAuth auth = FirebaseAuth.getInstance();
2  String emailAddress = "user@example.com";
3
4  auth.sendPasswordResetEmail(emailAddress)
5          .addOnCompleteListener(new OnCompleteListener<Void>() {
6              @Override
7              public void onComplete(@NonNull Task<Void> task) {
8                  if (task.isSuccessful()) {
9                      Log.d(TAG, "Email sent.");
10                 }
11             }
12         });
```

Code 8: Passwort vergessen

Zusätzlich wird in der Activity überprüft, ob das Eingabefeld leer ist. Falls das Zusenden fehlgeschlagen ist, wird eine Fehlermeldung ausgegeben. Auch gibt es eine Möglichkeit wieder zum Loginformular zurückzukehren.

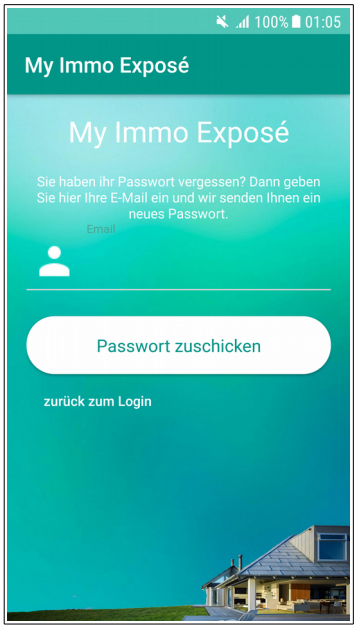

Abbildung 12: Screenshot Passwort vergessen Ansicht

6.2.6 NavigationDrawer und Menü

Um in die App eine aussagekräftige Navigation zu implementieren, wurde das Konzept des NavigationDrawers angewandt.

Der NavigationDrawer ist eines der flexibelsten Navigationselemente die Android zu bieten hat und findet in den meisten Apps seinen Einsatz. Er wird in den meisten Fällen von der linken Bildschirmseite auf den Bildschirm oder über das sogenannte Hamburger - Menü oder auch DataToggle zur Anzeige gebracht.

Für die Implementierung des NavigationDrawers wurde zuerst eine Menü Ressource erstellt. In diesem Menü wurden vier Items für die einzelnen Menüpunkte erstellt. Jedem Menüpunkt wurde für ästhetische Gründe ein passendes Icon zugewiesen.

Im nächsten Schritt erfolgte die Festlegung des generellen Layout des Drawers. Dieses beseht laut den Designgrundlagen des Material Designs zumeist aus einem Header-bereich (deklariert in `res/layout/nav_header_main.xml`) und dem Navigationsmenü. Beide Komponenten werden in der `actvity_main.xml` zu den NavigationDrawer in der NavigationView zusammen gefügt.

Zusätzlich beinhaltet das DrawerLayout noch die AppBar. Diese wird aus übersichtlichen Gründen in einer separaten Layout-Ressource deklariert. In der AppBar ist meist der Titel des aktiven Menüpunktes zusehen. Zudem wird ein zweites Layout über `<include…/>` dem jetzigen Layout hinzugefügt, das `content_main.xml`. In der `content_main.xml` ist meist ein RelativeLayout deklariert, das die ID `content_frame` besitzt.

Diese ID ist wichtig, da diese benötigt wird, um mittels des Codes in `MainActivity.java` die Fragmente gegeneinander entsprechend der Menüpunkte auszutauschen.

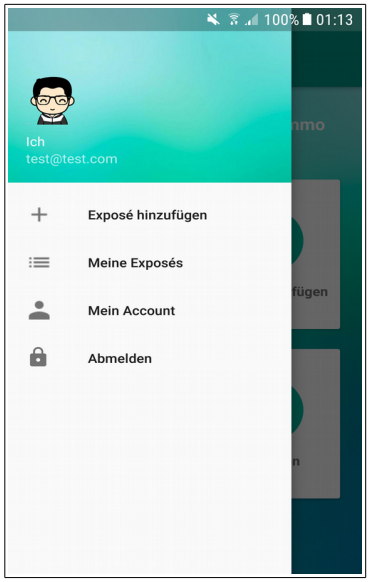

Abbildung 13: Screenshot NavigationDrawer / Menü

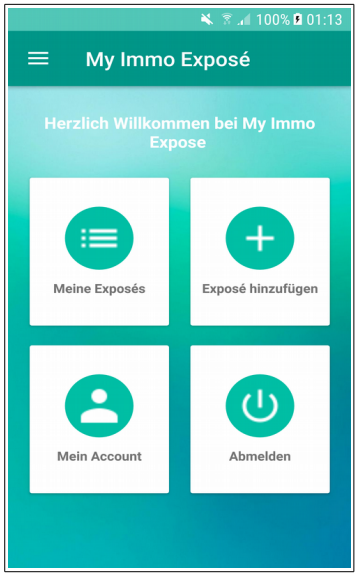

Abbildung 14: Screenshot Dashboard/ Home

Des Weiteren werden in der `content_main.xml` alle Designelemente wie Buttons etc. deklariert, die beim ersten Aufrufen der `MainActivity` gezeigt werden sollen. In der `content_main.xml` ist ein Dashboard erstellt wurden, welches einen Schnellzugriff zu allen wichtigen Funktionen der App bieten soll. Die Logik dazu wird in der `MainActivity.java` implementiert.

Um den NavigationDrawer seine Funktionalität zu implementieren, wird in der `onCreate()` – Methode der `MainActivity` als erstes eine Referenz zum DrawerLayout aus der `activity_main.xml` erzeugt.

Im nächsten Schritt wird das übliche Menü Icon und dessen Funktionalität implementiert. Dafür wird die vorgefertigte Klasse *ActionBarDrawerToogle* verwendet.

Es muss eine neue Instanz dieser Klasse erzeugt und der Klasse der Context, die Toolbar sowie zwei stellvertretende Strings übergeben werden. Diese Strings stehen für das Öffnen bzw. das Schließen des NavigationDrawers. Ist dies implementiert, so muss dem Drawer der sogenannte Toggle hinzugefügt und eine Synchronisation durchgeführt werden.

Damit die Menüpunkte des NavigationDrawer auch klickbar sind, muss eine Referenz zu der NavigationView geschaffen werden. Danach folgt eine Implementierung des *NavigationItemSelectedlListener()*. Dieser wird in der Activity über das Interface *NavigationView.onNavigationItemSelectecListener* implementiert.

Um die Events bei einem Klick auf ein Navigationselement zu verarbeiten, wird nun die Methode *onNavigationItemSelected()* aufgerufen. Damit jedes Navigationselement einzeln angesprochen werden kann, wurde diesen bei der Menüerstellung eine eindeutige ID zugewiesen.

Nachfolgend wird jeder Klick entsprechend der Funktion des Menüelementes behandelt. Beispielsweise wird bei einem Klick auf „Exposé hinzufügen“, welches die ID *nav_add_new_expose* besitzt, das Fragment *addNewExposé* aufgerufen. Dies erfolgt analog für alle anderen Menüpunkte. Nach dem Ausführen des Codes für die Menüpunkte schließt sich der NavigationDrawer automatisch.

Für die Integration des Logout-Prozesses müssen in die *MainActivity* der *AuthStateListener* von FirebaseAuth in die *onCreate()* - Methode implementiert werden. Der *AuthStateListener* überprüft, ob ein User angemeldet ist. Falls dies nicht der Fall ist, wird automatisch das Loginformular aufgerufen. Wichtig ist, dass der *AuthStateListener* in der *onStart()* – Methode der Activity den FirebaseAuth Instanz hinzugefügt und in der *onStop()* - Methode wieder entfernt wird.

Die *MainActivity* beinhaltet zusätzlich auch die Funktionalität des Dashboards. Um die im Layout erstellten *CardViews* klickbar werden zu lassen, wird zuerst eine Referenz für jede einzelne *CardView* erzeugt. Auf diese wird dann ein *onClickListener()* implementiert. Bei einem Klick auf die jeweilige *CardView* wird dann analog zu den Menüpunkten das Standard-Fragment mit einem anderen Fragment ersetzt.

```
1  // Methode zum Verarbeiten der Klickevents im Menü
2  // Ansichten werden ausgetauscht in dem die Fragemente mittels des FragmentManagers ausgetauscht werden
3  @Override
4  public boolean onNavigationItemSelected(MenuItem item) {
5
6      int id = item.getItemId();
7
8      if (id == R.id.nav_all_expose) {
9          ShowAllFragment showAllFragment = new ShowAllFragment();
10         getSupportFragmentManager().beginTransaction()
11             .replace(R.id.content_frame, showAllFragment)
12             .addToBackStack(null)
13             .commit();
14     } else if (id == R.id.nav_add_new_expose) {
15         //Replace content_frame with the AddNewExposeFragment
16
17         AddNewExpose addNewExpose = new AddNewExpose();
18         getSupportFragmentManager().beginTransaction()
19             .replace(R.id.content_frame, addNewExpose)
20             .addToBackStack(null)
21             .commit();
22     } else if (id == R.id.nav_my_account) {
23         //Replace content_frame with the MyAccountFragment
24         MyAccountFragment myAccountFragmentFragment = new MyAccountFragment();
25         getSupportFragmentManager().beginTransaction()
26             .replace(R.id.content_frame, myAccountFragmentFragment)
27             .addToBackStack(null)
28             .commit();
29     } else if (id == R.id.nav_logout) {
30         //Log user out
31         auth.signOut();
32     }
33
34     // Wenn ein Navigationselement geklickt wurde, schließe das Menü
35     DrawerLayout drawer = findViewById(R.id.drawer_layout);
36     drawer.closeDrawer(GravityCompat.START);
37     return true;
38 }
```

Code 9: onNavigationItemSelected() aus der MainActivity

6.2.7 Exposé erstellen

Um mit den Daten in der Firebase Datenbank arbeiten zu können, muss eine Referenz zu diesen aufgebaut werden.Da die hier erstellten Immobilien einem User zugeordnet werden sollen, wird die User ID als Knotenpunkt der Datenbank verwendet.

Damit eine Immobilie/ Exposé der Datenbank hinzugefügt werden kann, wird ein Datenmodell verwendet. Dieses ist in der Klasse *Immobilie.java* definiert. Diese Klasse enthält alle Attribute, die eine Immobilie besitzen kann. So zum Beispiel eine ID,

den Titel, Adressdaten oder auch Angaben über Größe, Preisangaben, Beschreibungstexte zur Lage, Ausstattung oder auch generelle Beschreibungen werden mit allen anderen Attributen als Strings (Zeichenketten) am Model angelegt.

Zusätzlich werden zu jedem Attribut ein Getter definiert. Mit Hilfe der Getter - Methoden kann ein schnellerer Zugriff auf die Attribute einer Immobilie ermöglicht werden. Mit Hilfe dieses Modells ist die Grundlage für das Hinzufügen einer Immobilie geschaffen.

Um die Daten aus den Textfeldern in Verbindung mit der Immobilie zu bringen, wird zuerst jedes Textfeld `findViewByID()` referenziert. Über die Methode `getStringsFromInputs()` werden die Inhalte der Textfelder in Strings geschrieben.

Die Methode `saveExposéToDB()` verwendet dann die Strings aus der Methode `getStringsFromInputs()` zur Erstellung des Exposés. Bevor die Immobilie jedoch erstellt werden kann, wird überprüft, ob zumindest der Titel gegeben ist, bevor ein neues Immobilien - Objekt erzeugt wird.

Über die `saveExposéToDB()` - Methode wird die Immobilie der Datenbank hinzugefügt. Die Methode `saveExposéToDB()` wird dann ausgeführt, wenn der Nutzer auf das Speichern-Symbol in der AppBar des Fragmentes klickt.

Danach wird der Nutzer mit der Methode `changeFragment()` auf eine Übersichtsseite seiner bis jetzt erstellten Exposés umgeleitet.

Bei einem Klick auf das X-Symbol werden alle Eingabefelder geleert und es kann noch ein mal von neuem begonnen werden. Das Leeren der Eingabefelder, sowohl als auch das Speichern eines Exposés sind über ein Optionsmenü realisiert. Dafür muss in der `onCreate()`- Methode des Fragments `setHasOptionsMenu()` auf `true` gesetzt werden. In der Methode `onCreateOptionsMenu()` muss das entsprechende Menü aufgerufen werden. Im Methodenaufruf von `onOptionsItemSelected` werden die Funktionen, der einzelnen Elemente festgelegt.

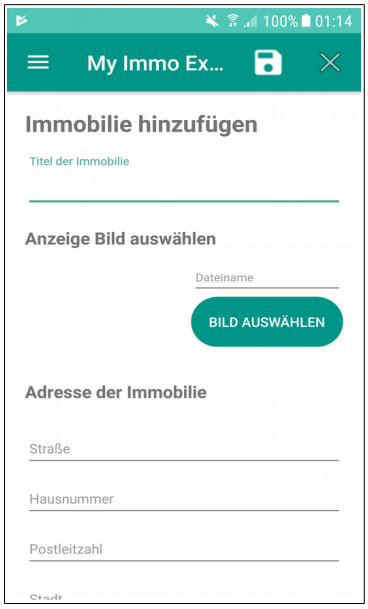

Abbildung 15: Screenshot Exposé erstellen

6.2.8 “Alle Exposé” Übersichtsseite

Um dem Nutzer einen Überblick über alle seiner erstellten Exposés zu ermöglichen, war es notwendig eine individuelle Ansicht für diese in Form einer Liste zu erzeugen.

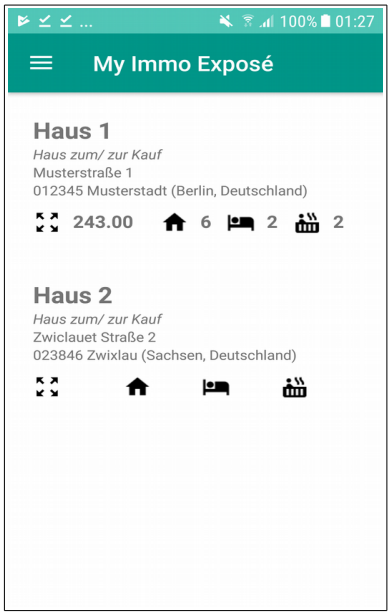

Abbildung 16: Screenshot Exposéübersicht

Da die Liste von Immobilen die wichtigsten Infos zu einer Immobilie auf einen Blick darstellen soll, wurde ein individueller Adapter für die Listenansicht *ImmobilienSwipeRecyclerViewAdapter* erstellt. Dabei stellt der Adapter eine Brücke zwischen den User Interface und den Daten her. Adapter beinhalten somit die Daten und senden diese und die View-Elemente. Somit können über einen Adapter auch eigene Layouts als Listenelemente erzeugt werden, welche für den hier bezweckten Einsatz von Bedeutung sind.

Um das Aussehen der Listenelemente festzulegen wurde im ersten Schritt das Template für die Anzeige erstellt (*res/layout/layout_expose_list.xml*). In diesem Layout sollen alle wichtigen Informationen zu einer Immobilie sichtbar sein, daher wurde für jede Information eine TextView deklariert. Diese können über ihre IDs angesprochen werden. Das Template wird in der *getView()* – Methode festgelegt.

Im nächsten Schritt wurde der Adapter *ImmobilienSwipeRecyclerViewAdapter* definiert. Nachdem die Referenzen zu den Viewelementen im *SimpleViewHolder* erstellt wurden, können über die *setText()* - Methode die Daten übergegeben werden. Dazu werden die Getter - Methoden aus dem Datenmodel für die Immobilie (*Immobilie.java*) verwendet. Dieser individuelle Adapter wird in dem Fragment *ShowAllFragment* nach dem Auslesen der Daten angewandt.

Für die weitere Implementierung werden in dem *ShowExposéFragment* Immobilien in einem ListArray mit dem Namen *immobilien* angelegt. In der *onCreate()* - Methode wird aus diesem ListArrray ein neues Listenobjekt erzeugt. Um eine Referenz zur Datenbank aufzubauen, wird eine Instanz der FirebaseAuth benötigt, da die Daten unter der *UserID* gespeichert sind.

Um die View-Elemente mit Daten zu befüllen, wird in der Methode *onBindViewHolder()* die Immobilie anhand ihrer Position in der *RecyclerView* referenziert. Über die Getter - Methoden, die im Datenmodell *Immobilie.java* definiert wurden, können die Inhalte für eine Immobilie geladen werden. In der *onBindViewHolder()* - Methode wird ebenfalls auch die Funktionalität der Swipeelemente definiert. In dem Fragment *ShowAllFragment* wird dann der Adapter *ImmobilienSwipeRecyclerViewAdapter* in der *onCreate()* - Methode der *RecyclerView* zugewiesen. Ebenfalls wird dort der *ScrollListener* für die *SwipeView* an die *RecyclerView* gebunden.

Zur Verbesserung des Nutzungserlebnisses für den User wurde eine Überprüfung der Datenbank auf Vorhandensein von Daten implementiert. Nur wenn sich Immobilien in der Datenbank befinden, wird eine Liste dieser angezeigt. Falls keine Daten im Datasnapshot enthalten sind, wird eine alternative Ansicht angezeigt. Diese Ansicht beinhaltet eine TextView, die den Nutzer darauf hinweist, dass sich keine Immobilien in der Datenbank befinden und einen Button, der dem Nutzer die Möglichkeit gibt zu der Erstellung einer neuen Immobilie zu springen.

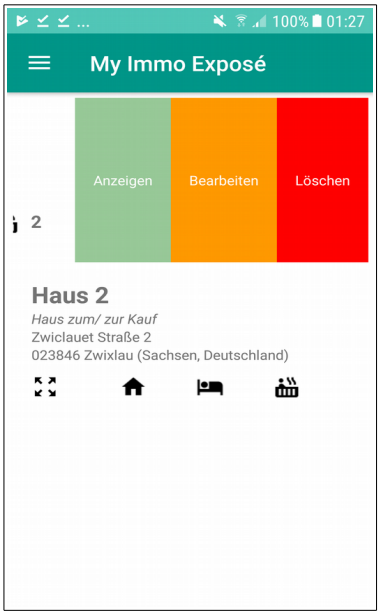

Abbildung 17: Screenshot SwipeOptionen Exposéliste

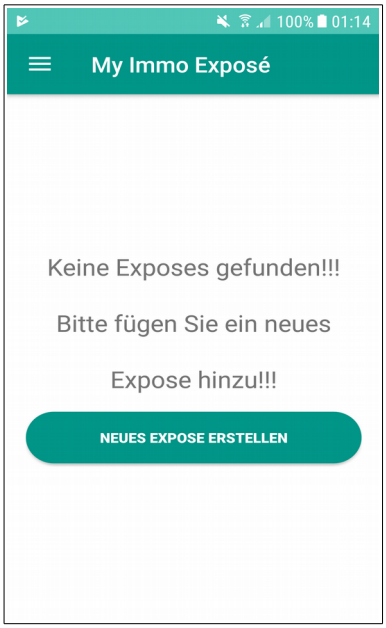

Abbildung 18: Screenshot "Keine Exposé"

6.2.9 Exposé Detailansicht

Um sich ein Exposé im Detail anzusehen wird das Fragment `ShowExposéFragment` durch einen Klick auf eine Immobilie aus der Übersichtsseite aufgerufen. Damit die richtige Immobilie angezeigt wird, muss dem `ShowExposéFragment` über das Bundle die ExposéID übergeben werden. Ein Bundle ermöglicht die Übergabe von Parametern zu einer anderen Activity oder einem Fragment. In der `ShowExposéFragment.java` wird die ExposéID dann ausgelesen und an die entsprechenden Stellen (hier an die Tab-Fragmente) übergeben.

Das generelle Layout der Detailansicht basiert auf einem *TabLayout* (deklariert in *layout/show_expose_layout.xml*). Das Layout besteht aus einem *AppBarLayout*, welches eine *Toolbar* und das *TabLayout* enthält, sowie dem *ViewPager*. Der *ViewPager* ist für die Anzeige der Tabinhalte (Fragmente) zuständig.

Um das TabLayout zu verwirklichen wurde ein *PageAdapter* (adapters/SectionsPageApadter.java) erstellt. Dieser ermöglicht die Initialisierung der Tabs aufgrund von ihrer Positionierung.

Das *TabLayout* besteht grundsätzlich aus vier einzelnen Tabs, um die Daten strukturiert und übersichtlich darzustellen. Für diese vier Tabelemente wurde jeweils ein Fragment erstellt. In den Fragmenten für die Tabs werden die Daten aus der RealtimeDatabase von Firebase ausgelesen und verarbeitet.

Die Fragmente werden über den Code in der *ShowExposéFragment.java* dem *TabLayout* ergänzt. Dazu muss eine Instanz des *SectionsPageAdapters* erzeugt und dem *ViewPager* zugeordnet werden. Dies passiert in der *onCreateView()* - Methode der *ShowExposéFragment.java*.

In der Methode *setupViewPager()* wird die ExposéID aus dem Bundle *oldbundle* ausgelesen und in ein neues Bundle *newBundle* geschrieben. Dieses Bundle wird dann den neuen Instanzen der vier Tabfragmente über *setArguments()* hinzugefügt.

Danach werden die Fragmente dem Adapter *adapter* über die Funktion *addFragment()* hinzugefügt. Zum Schluss wird der Apdater dem *ViewPager* über *setAdapter()* zugewiesen.

Des Weiteren wird in der *onCreateView()* der Titel und der Untertitel des Exposés in der Toolbar gesetzt. Dazu werden die Daten aus den Datenbankreferenzen gelesen und über *setTitle()* und *setSubtitle()* gesetzt.

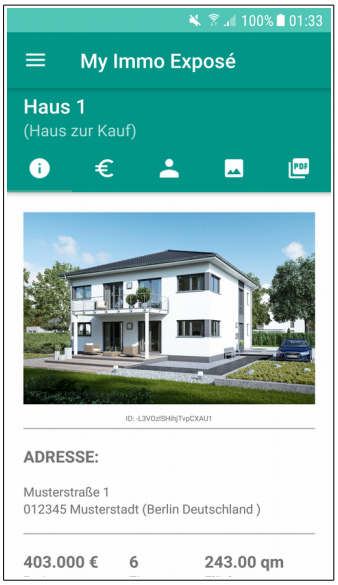

Abbildung 19: Screenshot Infotab

Abbildung 20: Screenshot Kostentab

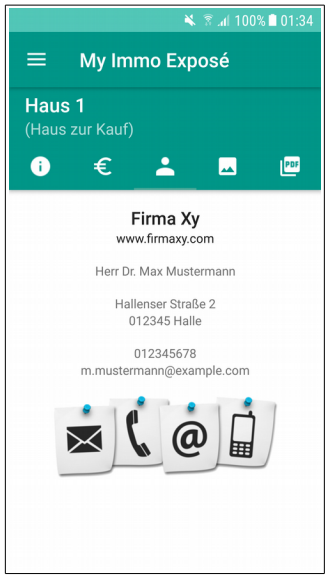

Abbildung 21: Screenshot Kontakttab

6.2.10 Exposé bearbeiten

Dem Nutzer soll natürlich auch die Möglichkeit gegeben werden Immobilien bearbeiten zu können. Um ein Exposé zu editieren wird die Exposéübersicht aufgerufen. Über das „Wischen“ (von rechts nach links) über dem Exposé - Item in der Liste erscheint das Swipemenü aufgerufen. Durch anschließend Klick auf „Bearbeiten“ in dem Menü gelangt der Nutzer zur Ansicht, die es ihm ermöglicht ein Exposé zu bearbeiten.

Nach dem Wechsel des Fragmentes zum `EditExposéFragment` wird der nötige Code zum Editieren der Immobilie deklariert. Dazu werden Referenzen zu den Datenbanken und Viewelementen benötigt. Diese werden in der `onCreate()` - Methode deklariert. Ebenfalls werden in der `onCreate()` - Methode die Daten aus der RealtimeDatabase gelesen und in die Textinputs geschrieben, sodass der Nutzer weiß, welche Daten er bearbeiten muss.

In der Methode `getStringsFromInputs()` werden die Eingaben aus den Textinputs in Strings umgewandelt. Dies ist nötig damit in der `updateExposé()` - Methode die Daten wieder in die Datenbank geschrieben werden können.

Das Updaten der einzelnen Werte erfolgt über die Methode `setValue()`. Dazu wird das jeweilige Wertepaar geupdatet.

6.2.11 Exposé löschen

Um ein Exposé zu löschen muss sich der Nutzer in der Exposéübersicht befinden. Über das SwipeMenü kann dort ein Exposé gelöscht werden. Zum Löschen muss das „Löschen“ - Icon in Form eines Mülleimers geklickt werden.

Das Löschen eines Exposé-Items ist in der `onBindViewHolder()` - Methode des `ImmobilienSwipeRecyclerView` – Adapter deklariert.

Um ein Exposé löschen zu können, werden die dementsprechenden Datenbankreferenzen benötigt. Das Exposé-Item wird zum Löschen über dessen ID angesprochen. Da die Immobilien Daten an drei verschiedenen Stellen (Immobilie, Contact, Attachments) in die Datenbank geschrieben werden, müssen sie auch an allen drei Stellen über die Methode `removeValue()` entfernt werden. Nach dem die Daten gelöscht wurden wird das Fragment ausgetauscht.

```
// setup onClick for the delete button
        viewHolder.delete.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {

                mItemManger.removeShownLayouts(viewHolder.swipeLayout);

                immoDataRef.addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        mItemManger.removeShownLayouts(viewHolder.swipeLayout);

                        immoDataRef.removeValue();
                        contactDataRef.removeValue();
                        attachmentDataRef.removeValue();

                        notifyItemRemoved(position);
                        notifyItemRangeChanged(position, immobilien.size());
                        mItemManger.closeAllItems();

                        Toast.makeText(mContext, "Immobilie wurde gelöscht", Toast.LENGTH_SHORT).show();
                    }

                    @Override
                    public void onCancelled(DatabaseError databaseError) {

                    }
                });

                notifyItemRemoved(position);
                notifyItemRangeChanged(position, immobilien.size());
                mItemManger.closeAllItems();
            }
        });
```

Code 10: Exposé löschen

6.2.12 Anhänge hinzufügen

Um Anhänge einem Exposé hinzuzufügen, muss das Exposé erstellt sein und sich der Nutzer in der Ansicht aller Exposés befinden. Nach Aufruf des Exposés in seiner Detailansicht kann der Nutzer nun in den Tab „Bilder“ (für Bilduploads) oder „PDF“ springen. In diesem Tab befindet sich eine Button „Bild hinzufügen“. Auf diesen muss der User klicken, um einen neuen Anhang dem Exposé hinzuzufügen. Durch den Klick wird der User auf das `AddAttachmentFragment` weitergeleitet. In diesem Fragment kann der Nutzer nun sein Bild auswählen und dem Exposé hinzufügen.

Um dies zu ermöglichen wird eine `StorageReference` und eine `Database-Reference`, sowie zwei Buttons benötigt. Die Referenzen zeigen zu einem auf den physischen Speicherort der Datei im FirebaseStorage und zum anderen wird über die Datenbankreferenz ein `AttachmentUpload` erzeugt der eine Referenzierung des Bildes ermöglicht.

Der `AttachmentUpload` ist hier ein Datenmodel, dass die Attribute eines Anhangs/ Attachments festlegt. Folgende Attribute besitzt ein Attachment bei der Erstellung:

- **Name:** Der Name ist der eigentliche Titel des Bildes. Er wird in der Listenansicht der Attachments angezeigt.
- **Url:** Die URL ist die Referenz auf den Speicherplatz des Bildes im FirebaseStorage. Über sie kann das Bild geladen werden.
- **ID:** Die ID ist das unverkennbar Erkennungsmerkmal eines Attachments. Sie ist für jeden Anhang eineindeutig.
- **ImmoID:** Um einen Anhang einer Immobilie zuordnen zu können, wird die ID der Immobilie in der Datenbank gespeichert.

Die einzelnen Anhänge werden in Form einer Liste dargestellt. Dazu wurde im Layout neben dem Button eine `RecyclerView` implementiert. Diese wird durch in der `onCreateView()` des `AttachmentTabFragmentes` mit Daten aus der Datenbank gefüllt. Besonderheit dieser Listenansicht ist, dass über einen Wisch von rechts nach links weiter Optionen, wie das Löschen eines Anhangs freigeschalten werden.

Lässt der Nutzer sich die Bildanhänge anzeigen, kann durch einen Klick auf das jeweilige Bild eine größere Ansicht dessen erzeugt werden. In der PDF - Anzeige führt ein Klick auf die PDF – Bezeichnung dazu, dass die Datei heruntergeladen wird.

6.2.13 Anhänge löschen

Um einen Anhang zu löschen, muss sich der Nutzer in der jeweiligen Übersicht befinden in der die PDF oder das Bild hinterlegt ist. Über eine Wischgeste von rechts nach Links über dem Anhang wird das SwipeMenü aufgerufen. Dort muss der Nutzer aud „Löschen“ klicken. Der Anhang wird danach gelöscht.

Die Funktionalität des Löschens ist bei PDFs und auch bei Bilder in deren `SwipeRecyclerViewAdapter` deklariert. Um den Anhang zu löschen muss eine Datenbankreferenz, die auf den entsprechenden Anhang zeigt, erzeugt werden. Dieser ist über seine ID eindeutig identifizierbar. Über die Methode `removeValue()` wird der Datensatz aus der Datenbank entfernt.

```
auth = FirebaseAuth.getInstance();
user = auth.getCurrentUser();
user_id = user.getUid();

attachmentID = item.getId();
immo_id = item.getImmo_ID();

pdfDataRef = FirebaseDatabase.getInstance().getReference(Constants.DATABASE_PATH_IMAGE_UPLOADS)
    .child(user_id)
    .child(immo_id)
    .child(Constants.DATABASE_PATH_ATTACHMENTS).child(Constants.DATABASE_PATH_PDFs)
    .child(attachmentID);


viewHolder.Delete.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        mItemManger.removeShownLayouts(viewHolder.swipeLayout);

        pdfDataRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {

                pdfDataRef.removeValue();
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        });

        notifyItemRemoved(position);
        notifyItemRangeChanged(position, attachmentUploads.size());
        mItemManger.closeAllItems();
    }
});
```

Code 11: Beispiel PDF - Anhang löschen

6.2.14 Mein Account

Im Menüpunkt „Mein Account“ kann der Nutzer Details zu seinem Account ändern. Das heißt es werden ihm folgende Dinge ermöglicht:

- sein Profilbild zu ändern
- seine Email-Adresse zu ändern
- Passwort ändern
- oder auch seinen Account zu löschen

Damit der Nutzer all diese Dinge ändern kann, wird das Fragment *MyAccount* implementiert. Für alle Funktionalitäten muss in der *onCreate()* - Methode der Klasse der aktuelle Nutzer ermittelt werden. Der Nutzer wird üblicher Weise über die Methode *getCurrentUser* ermittelt. Jedoch ist es möglich das diese Methode null zurückgeben kann. Dies passiert zum Beispiel, wenn der Nutzer auf einem anderen Gerät gelöscht wurde. In diesem Fall ist der Nutzer, der von der Methode *getCurrentUser* zurückgegeben wurde, gültig, jedoch schlagen alle nachfolgende Aufrufe an Ressourcen fehl. Daher wurde in dem *MyAccountFragment* ein *AuthStateListener* implementiert. Dieser fängt alle Änderungen des Tokenstatuses eines Nutzers ab und kann somit auf Änderungen an Accountdaten reagieren. Daher sind die Funktionalitäten die einen Account betreffen auch in der Methode *onAuthStateChanged()* des *FirebaseAuth.AuthStateListener()* deklariert.

Profilbild ändern

Der aktuell angemeldete Nutzer kann sein Profilbild natürlich selbst festlegen. Dafür muss er den Menüpunkt „Mein Account“ aufrufen. Durch einen Klick auf das bisherige Profilfoto, kann der Nutzer ein neues Profilbild auswählen. Zum Auswählen des Bildes wird die Methode *choosePicture()* auf die *ImageView* angewandt. Über die Methode *onActivityResult()* wird das ausgewählte Bild in der *ImageView* angezeigt.

Um das Profilbild des Nutzers ändern zu können wird auf den in der View erstellten Button ein *onClick()* - Event gesetzt. In der *onClick()* - Methode des Buttons wird dann zuerst eine *StorageReference* und eine *DatabaseReference* erzeugt. Über die *putFile()* - Methode wird die Url *filePath* des Bildes an die *StorageReference* übergeben und das Bild somit in den Storage von Firebase hochgeladen. Im selben Schritt wird die URL des hochgeladenen Bildes *imageDownloadURL* über die Metode *setValue()* in die Datenbank geschrieben.

Email-Adresse ändern

Um die Email-Adresse eines Nutzeraccount zu ändern wird die Methode *updateEmail()* verwendet. An diese Methode muss die neue Email-Adresse *newEmail* übergeben werden. Diese wird aus dem Texteingabefeld ausgelesen. Um die Email-Adresse in der Datenbank zu ändern wird eine Datenbankreferenz erzeugt die auf das Childnode *email* eines Nutzers zeigt. Über die Methode *setValue()* wird die neue Email-Adresse *newEmail* aus dem Texteingabefeld als neuer Wert gesetzt. Um jedoch die Email-Adresse in der FirebaseAuth zu ändern muss sich vergewissert werden, dass der User sich vor kurzem eingeloggt hat. Dafür wird die Methode *reauthentificate()* auf den FirebaseUser *user* angewandt (siehe Code 13). Diese Methode benötigt die Credentials eines des Nutzers – also seine Email-Adresse und sein Passwort. Diese können über die folgende Codezeile aufgerufen werden:

```
// Userdaten aus FirebaseAuth aus lesen
final AuthCredential credential = EmailAuthProvider.getCredental(
                                                user.getEmail(), oldPassword);
// User reauthtifizieren
user.reauthenticate(credential);
```

Code 12: Abruf der Nutzerdaten aus der Authentifizierung

Um die Email-Adresse eines Nutzers nun in der FirebaseAuth wird die Methode *updateEmail()* aufgerufen. An diese Methode wird die neue Email-Adresse *newEmail* aus der Texteingabe übergeben.

Passwort ändern

Das Passwort muss an zwei verschiedenen Stellen geändert werden: der FirebaseAuth und in der RealtimeDatabase. Um das Passwort in der Datenbank zu ändern wird eine Datenbankreferenz erzeugt die auf das Childnode `passwort` eines Nutzers zeigt. Über die Methode `setValue()` wird die neue Passwort `newPasswort` aus dem Texteingabefeld als neuer Wert gesetzt. Um jedoch das Passwort in der FirebaseAuth zu ändern muss sich vergewissert werden, dass der User sich vor kurzem eingeloggt hat. Um das zu umgehen wird die Methode `reauthentificate()` auf den FirebaseUser `user` angewandt (siehe Code 13).

Um das Passwort in der FirebaseAuth wird die Methode `updatePasswort()` aufgerufen. An diese Methode wird das neue Passwort `newPasswort` aus der Texteingabe übergeben.

Account löschen

Um einen Nutzeraccount zu löschen wird der User auf eine neue Activity `UserDeleteActivity` weitergeleitet. Im Layout dieser Activity befinden sich zwei Buttons. Der Button „Nein, ich möchte meinen Account behalten“ leitet den User zurück auf das Dashboard und bricht den Vorgang des Löschen eines Accounts ab. Der andere Button „Ja, ich möchte meinen Account löschen“ führt zum Löschen des Nutzeraccounts. Für die Funktionalität beider Buttons wird ein `onClickListener` implementiert.

Um den Nutzeraccount zu löschen wird als erstes der Nutzer aus der Applikation ausgeloggt. Um einen Nutzer auszuloggen wird die Methode `signOut()` auf das FirebaseAuth – Objekt `mAuth` angewandt. Danach kann der FirebaseUser `user` mittels `user.delete()` gelöscht werden. Danach wird der Nutzer auf die Loginseite der Applikation umgeleitet.

7 Probleme und Lösungen

7.1 Fehlerarten

Während der Implementierung des Prototypen sind die unterschiedlichsten Probleme aufgetreten. Die Ursachen dieser und die Fehlerbehebung war oft schwierig, da die Probleme an den verschiedensten Stellen aufgetreten sind. Eine schnelle Problemlösung war deswegen nicht immer möglich.

Um Fehler zu lokalisieren bietet Android Studio das Tool Logcat. Dieses ist für die Behebung von Fehlern unverzichtbar, da es Exceptions (Fehler) in der Konsole sichtbar macht. Der Debugger von Android Studio lokalisiert zusätzlich die Stelle an der der Fehler auftritt. Eine genaue Lokalisierung erfolgt durch Angabe der Zeile und des Dateinamens. Zusätzlich zu diesen Angaben wird der Fehler kurz beschrieben.

Die folgenden Fehlerarten können bei der Implementierung des Prototypen auftreten:

- Syntaxfehler
- Lexikalische Fehler
- Semantische Fehler
- Logische Fehler
- Laufzeitfehler

In Android Studio ist eine intelligente Fehlererkennungsfunktion integriert, die es ermöglicht, dass Fehler in Echtzeit erkannt werden. Somit können die meisten Fehler während der Entwicklung bereits erkannt und behoben werden. Somit wird das Auftreten von Fehlern minimiert. Jedoch ist es nicht möglich mit dieser Echtzeiterkennung logische oder auch Laufzeitfehler zuerkennen. Diese zwei Arten von Fehlern treten jedoch nur unerwartet und vereinzelt auf. Mit Hilfe des Logcats und des Debuggers können diese Fehler jedoch auch schnell behoben werden, sobald sie lokalisiert wurden.

Wie bereits angesprochen werden mittels des Logcats von Android Studio Exceptions ausgegeben. Diese stellen Ausnahmen bzw. Fehler dar, die erst bei der Laufzeit der Androidapplikation auftreten. Werden Exceptions nicht behandelt/ abgefangen bringen die App zum Absturz.

7.2 NullPointerException

Während der Programmierung des App – Prototypen sind am häufigsten NullPointerException aufgetreten. Diese Exception tritt auf wenn ein Objekt leer ist, aber darauf zugegriffen werden soll. Der Debugger zeigt den Fehler in der Regel an der Stelle an, an der er auftritt. Es ist jedoch möglich, dass Fehler tief verschachtelt sind, wenn beispielsweise über mehrere Klassen und Hilfsklassen hinweg gearbeitet wird. Die Ursachenforschung kann sich in diesem Fall zeitlich in die Länge ziehen, da der Entwickler den Fehler manuell nachverfolgen und beheben muss.

7.3 Daten werden nicht angezeigt

Ein weiterer Fehler der häufig aufgetreten ist, war das die Daten aus der Firebase RealtimeDatabase nicht angezeigt wurden. Dies lag in den meisten Fällen daran, dass die Datenbank Referenzen nicht auf den richtigen Pfad verwiesen. Um dies weitestgehend zu vermeiden wurde die Klasse `Constants.java` erstellt. In dieser Klasse wurden alle Dateipfad - Element instanziiert und auch die URLs für die Defaultbilder festgelegt.

8 Weiterentwicklungsmöglichkeiten & Fazit

Da diese App ein Prototyp ist, kann sie natürlich um eine Vielzahl von Funktionalitäten erweitert werden. In den folgenden Abschnitten soll ein Ausblick auf die Zukunft dies Prototypen gegeben werden.

8.1 Anbindung an die IS24-API

Der jetzige Prototyp basiert auf einer Mischung des Immobilienstandard OpenImmo und Daten, die aus der Dokumentation von ImmobilienScout24 übernommen sind.

Da ImmobilienScout24 eine der marktführenden Onlineplattformen im Immobilienmarkt ist und viele Makler ihre Objekte auf dieser Plattform veröffentlichen, ist es wohl sinnvoller die App an die API von ImmobilienScout24 anzubinden bzw. die Schnittstelle zu integrieren. Es folgt eine kurze Erklärung zu ImmobilienScout24 und der API.

ImmobilienScout24 ist einer der ersten Online Marktplätze der Scout24 AG und damit seit 1999 tätig. Das Unternehmen hat seinen Sitz in Berlin und beschäftigt mittlerweile 500 Mitarbeiter. Mit mehr als 1,5 Millionen Datenbankeinträgen, mehr als 8 Millionen Besuchern und 2 Milliarden Seitenaufrufen im Monat zählt das Unternehmen ImmobilenScout24 zu den marktführenden Onlineplattformen im Immobilienbereich. IS24 bietet privaten sowie gewerblichen Anbietern die Möglichkeit Immobilien jeglicher Art gegen ein Entgelt auf ihre Plattform einzustellen. Neben der eigentlichen Präsentation der Immobilie können virtuelle Rundgänge hinzugefügt werden. Zudem verfügt das Portal über Online Ratgeber und Analyse- Tools, die der Nutzer kostenfrei nutzen kann.

8.1.1 Die Import/Export API

Die Import/Export - Programmierschnittstelle ist die wohl am meisten genutzte *API* von ImmobilienScout24. Sie bietet dem Entwickler die folgenden Möglichkeiten:

- Erstellung von Immobilien
- Bearbeitung von Immobilien
- Löschen von Immobilien

Damit ist diese API ein mächtiger Bestandteil von ImmobilienScout24 (in der *API* werden fünf Komponenten verwendet). Die Schnittstelle baut auf vier grundlegenden Komponenten auf. Die wichtigsten Komponenten sind dabei die RealEstates. Sie sind die Objekte und stellen somit die Immobilie dar. Zusätzlich gibt es noch die Attachments (Anhänge). Mit diesen können Bilder, Videos oder auch Liegepläne und PDFs einem Exposé hinzugefügt werden. Auch können URLs als Attachment deklariert werden. Neben diesen zwei Komponenten gibt es noch die Funktionen des Veröffentlichen (Publish) und des Contact. Die Funktion Contact bietet die Möglichkeit Kontaktinformationen zu einer Immobilie hinzuzufügen.

8.1.2 Vorteile der API

Die Verwendung der API bringt dem Developer natürlich auch Vorteile. So werden erstellte Objekte in Echtzeit online gestellt und auch Fehlercodes in der Applikation angezeigt. Ebenfalls werden Änderungen an Objekten in Echtzeit vorgenommen. Daher sind diese auch sofort in der Online Plattform von ImmobilienScout24 sichtbar.

Da in der heutigen Zeit der Datenschutz großgeschrieben wird, übermittelt die API alle Daten über *HTTPS*. Ebenfalls werden Daten über *HTTPS* abgerufen.

Ein großer Vorteil ist, dass die *API* von Smartphone - Applikation sowie auch von Webseiten oder den gängigen CMS-Systemen genutzt werden kann. Ebenso bietet sie bereits vorgefertigte Datenmodelle für bestimmte Arten von Immobilien und hat somit einen größeren Umfang als die bis jetzt verwendeten Datenmodelle, die stark vereinfacht wurden.

8.2 Erweiterungen von Funktionen

Aufgrund der beschränkten Zeitvorgabe um dieses Projekt zu erfüllen, ist es natürlich möglich diesen Prototypen um folgende Funktionen zu erweitern und damit dem Nutzer ein besseres Nutzungserlebnis und vereinfachten Workflow zu bieten.

So könnte man dem Prototypen eine Art von Dokumenten-Scanner hinzufügen. Dieser soll dem Nutzer ermöglichen Dokumente, wie zum Beispiel den Grundrisse oder den Energieausweis einer Immobilie, direkt über die Kamera einzuscannen und damit dieser den Dokumenten der App hinzugefügt werden kann, ohne auf andere Apps wie “CamScanner” zurückzugreifen.

Das Erstellen von Grundrissen ähnlich der App RoomScan wäre auch denkbar. Dabei müsste die App intern Maße über den Entfernungssensor abgreifen und auch auswerten.

Ein automatischer Upload der in der App erstellten Exposés, ebenso wie Zustände eines Exposé (z.B. Entwurf, veröffentlichtes Exposé) sind natürlich auch weitere Optionen, die in den Prototypen der App implementiert werden können.

8.3 Fazit

Das Ziel des Projektes war es einen App - Prototypen für die Erstellung von Immobilien – Exposés zu entwerfen. Dabei sollten die grundlegenden Funktionen der App nach den Designrichtlinien von Android umgesetzt werden. Zu den grundlegenden Funktionalitäten gehörten zum Beispiel: das Erstellen, Löschen und Bearbeiten von Immobilien – Exposés, sowie das hinzufügen von Anhängen. Ebenso sollten eine Authentifizierung und die damit verbundenen Accountaktivitäten implementiert werden.

Die gestellten Anforderungen konnten zu 90% erfüllt werden. Verbesserungen in der Erstellung und Anzeige von Immobilien, wie zum Beispiel das Anzeigen von nur immobilientypspezifischen Eingabefeldern, können neben der Anbindung an die ImmobilienScout24 *API* vorgenommen werden. Abschließend kann man jedoch sagen dass eine Grundlage für die weitere Entwicklung dieses Projektes geschaffen wurde.

Es wurden alle Kernziele, welche in den Anforderungen (siehe Kapitel 5) definiert wurden, umgesetzt. Damit kann das Projekt als gelungen bezeichnet werden.

Die Umsetzung dieses Projektes hat meinen Wissensstand, was die Android – Entwicklung mit Firebase betrifft, bereichern können. Ebenso konnte ich bereits erworbene Kenntnisse aus dem Studium einbringen und weiter vertiefen.

Zusammenfassend bin ich mit meiner Leistung, sowie dem Gesamtergebnis zufrieden. Ich bin mir zudem sicher, dass mir die gewonnenen Erfahrungen, Kenntnisse und Fähigkeiten im Laufe der weiteren Studien - und Arbeitszeit zu Gute kommen werden. Ich kann mir vorstellen, in Zukunft weitere Projekte im Bereich der Android-Entwicklung zu übernehmen und nach Abschluss des Studiums eine Beschäftigung in diesem Bereich aufzunehmen.

Literaturverzeichnis

[LÖSEL] Sylvia Lösel/ Laimingas : Was ist Android?

Stand: 22.01.2018 um 20.00 Uhr
URL: https://www.it-business.de/was-ist-android-a-673197/

[ANDROIDDEV-VERSION] Android Developer Guide: Device Compatibility – Platform version

Stand 22.01.2018 um 14:00 Uhr
URL: https://developer.android.com/guide/practices/compatibility.html#Versions

[HANDYFLASH] Handyflash: Was ist Android? - Das musst du über Android wissen

Stand 22.01.2018 um 14:00 Uhr
URL: https://www.handyflash.de/ratgeber/ist-android-das-musst-du-ueber-android-wissen/

[ANDROIDDEV-SCREEN] Android Developer Guide: Device Compatibility – Screen Configuration

Stand 22.01.2018 um 14:00 Uhr
URL: https://developer.android.com/guide/practices/compatibility.html#Versions

[SCHANZE] Robert Schnaze: Was ist Android? - Kurz erklärt

Stand 22.01.2018 um 14:00 Uhr
URL: http://www.giga.de/apps/android-os/specials/was-ist-android-eine-erklaerung/

[EASON] Jamal Eason: An update on Eclipse Android Developer Tools

Stand 12.10.2017 um 14:00 Uhr
URL: https://android-developers.googleblog.com/2015/06/an-update-on-eclipse-android-developer.html

[ANDROIDSOURCE] Jamal Eason: An update on Eclipse Android Developer Tools

Stand 12.10.2017 um 14:00 Uhr
URL: https://source.android.com

[HOCH] Christoph H. Hochstädter: Anroid-Architektur – Wie viel Linux steckt in Google OS?

Stand 12.10.2017 um 14:00 Uhr
URL: http://www.zdnet.de/41553061/android-architektur-wieviel-linux-steckt-in-googles-os/

[ADG-ARCH] Android Developer Guide: Android Architecture

Stand 12.10.2017 um 14:00 Uhr
URL: https://developer.android.com/guide/platform/index.html

[INDIVIDUAPP] IndividuApp: Ordnerstruktur eines Android - Projekts

Stand 30.10.2017 um 13:00 Uhr

URL: https://individuapp.com/softwareentwicklung/android-kurs/android-ordnerstruktur/

[WEBHILFE] Homepage – Webhilfe: XML & Co. Einführung

Stand 11.11.2017 um 16:00 Uhr

URL: https://www.homepage-webhilfe.de/XML/

[OPENIMMO1] OpenImmo

Stand 16.11.2017 um 9:00 Uhr

URL: http://www.openimmo.de

[OPENIMMO2] OpenImmo , Prinzip

Stand 16.11.2017 um 9:00 Uhr

URL: http://www.openimmo.de/go.php/p/12/cm_prinzip.htm

[FASTCOMPANY] Steven Melendez, Sometimes You're Just One Hop From Something Huge

Stand 05.12.2017 um 22:00 Uhr

URL: https://www.fastcompany.com/3031109/sometimes-youre-just-one-hop-from-something-huge

[FIREBASEAUTH] Firebase: Firebase Authentification

Stand 23.12.2017 um 11:00 Uhr

URL: https://firebase.google.com/docs/auth/?authuser=0

[FIREBASEDB] Firebase: Firebase Realtime Database

Stand 23.12.2017 um 11:00 Uhr

URL: https://firebase.google.com/docs/database/?authuser=0

[FIREBASESTORAGE] Firebase: Cloud Storage

Stand 23.12.2017 um 11:00 Uhr

URL: https://firebase.google.com/docs/storage/?authuser=0

Bildquellenverzeichnis

Abbildung 1: Architektur Android

Quelle: Android Source: The Android Source Code
Stand: 03.01.2018
URL: https://source.android.com/setup/

Abbildung 2: Android Studio Logo

Quelle: Elite Innovative Solutions Inc, Android Apps
Stand: 03.01.2018
URL: http://www.eliteisinc.com/wp-content/uploads/2015/11/android-studio.png

Abbildung 3: Prinzip OpenImmo

Quelle: OpenImmo
Stand: 03.01.2018
URL: http://www.openimmo.de/common/bdcms/opim/prinzip.png

Abbildung 5: Firebase Dienste

Quelle: LinkedIn, Firebase: Backend As Service (veröffentlicht am 06.07.2017)
Stand: 03.01.2018
URL:
https://media.licdn.com/mpr/mpr/AAEAAQAAAAAAAAz9AAAAJGYxZWQ1ZmVkLTBhMTMtNGM1Ni04NzIzLTZjMDA3Y2M2ZjljMw.png

Abbildung 5: Firebase Authentifizierungsmöglichkeiten

Quelle: Firebase Docs, Firebase Authentification
Stand: 03.01.2018
URL: https://firebase.google.com/docs/auth/images/auth-providers.png

Abbildung 8: Übersicht Marklersoftwares

Quelle: Systemhaus.com, Marklersoftware zur Immoblilenverwaltung im Test & Vergleich im Überblick

Stand: 03.01.2018

URL: https://systemhaus.com/wp-content/uploads/2016/11/maklersoftware-anbieter-neu.jpg

Anlagen

Beiliegendes Medium

- **Projekt:** MyImmoExpose
- **APK:** myimmoexpose.apk
- **README:** Installation APK
- **PDF:** Bachelorarbeit Rebecca Blischke

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Leipzig, den 26.Januar 2018 _____

Ort, den TT. Monat JJJJ Vorname Nachname