HTL - Perg

Höhere Abteilung für Informatik

Diplomarbeit

app:center App

Implementierung einer Android App für den Gastronomie-Bereich

Projektteam: Sebastian Ganser

Julian Lindinger-Pesendorfer

Projektbetreuer: Prof. Dipl.-Ing. Dr. Michael Buchberger

Auftraggeber: Marcus Holzleitner

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Ort, Datum	Sebastian Ganser
Ort, Datum	Julian Lindinger-Pesendorfer

Danksagung

Wir bedanken uns bei all jenen, die uns bei unserer Diplomarbeit in jeglicher Hinsicht unterstützt haben.

Vor allem bei unserem Betreuungslehrer Dipl.-Ing. Dr. Michael Buchberger möchten wir uns für die fachliche Unterstützung bedanken. Durch seine große Erfahrung in der Software Entwicklung und bei der Verfassung wissenschaftlicher Arbeiten konnte er uns wertvolle Informationen für unsere Diplomarbeit vermitteln.

Weiters bedanken wir uns bei unserem Auftraggeber Marucs Holzleitner und unserer Kontaktperson Ing. David Andlinger. Bei offenen Fragen konnten uns beide Kontaktpersonen mit ihrem Wissen aus ihren selbstständigen Arbeiten in der Software Entwicklung unterstützen.

Herzlichen Dank!

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Kurzbeschreibung

Die Diplomarbeit app:center App wurde im Rahmen der Matura 2015 an der HTL Perg für Informatik implementiert.

app:center App ist eine Android-Applikation, welche als Teil des Produktes „GastroApp“ dient. Die GastroApp ermöglicht Gastronomie-Betrieben Inhalte zu ihrem Betrieb, wie Speisekarten, Neuigkeiten, Galerien und Veranstaltungen auf einer Smartphone-App für die Kunden zur Verfügung zu stellen. Mehr als 70 Prozent der Österreicher verwenden ein Smartphone mit Internetzugang [26]. Jedoch stellen die meisten Betriebe nur eine Webseite zur Verfügung, welche oft nicht einfach über ein Smartphone zu bedienen ist. Die GastroApp ist die perfekte Marketing-Alternative zu klassischen Webseiten. Ein weiterer Vorteil der GastroApp ist die einfache Erweiterbarkeit auf Grund des Aufbaus in verschiedenen Modulen, die der Kunde individuell auf sein Unternehmen abstimmen kann.

Die app:center App dient dem Gastronom als Verwaltungsprogramm für seine GastroApp. Der Gastronom kann damit direkt über sein eigenes Smartphone sämtliche Informationen, Beiträge, Bilder und weitere Module der GastroApp komfortabel, schnell und flexibel bearbeiten. Die Änderungen werden sofort für alle Kunden ersichtlich. Bei speziellen Angeboten ermöglicht die app:center App dem Gastronom eine direkte Nachricht (engl. Push-Notification) an seine Kunden zu senden. Der Gastronom kann mit seinem Smartphone Fotos aufnehmen und diese direkt in eine Galerie hochladen. Bei Veranstaltungen oder Turnieren ist der Zugang über das eigene Smartphone besonders vorteilhaft, da der Gastronom Neuigkeiten und Bilder für seine Kunden bequem über sein eigenes Smartphone sofort hochladen kann, ohne den Ort des Geschehens zu verlassen.

1.2 Abstract

The diploma app:center App was developed at the Higher Technical Institute Perg for informatics in 2014/2015.

app:center App is an Android application which acts as a part of the product „GastroApp“. The product GastroApp enables gastronomic businesses to provide informations like the current menu, news, galleries, events and other features on an app which can be downloaded by the customers. About 70 percent of all Austrians use their smartphone to gather information

over the internet [26]. The GastroApp is the perfect alternative to the classic homepage to increase the marketing. One advantage of the GastroApp is the ease of extension and modification.

The gastronome can use the app:center App as the administration program for his GastroApp. He is able to update all informations shown in the GastroApp directly with his own smartphone. The gastronome is able to send push-notifications directly to his customers.

1.3 Auftraggeber

Auftraggeber dieser Diplomarbeit ist der Unternehmer Marcus Holzleitner, welcher mit dem Unternehmer David Andlinger die GastroApp entwickelt hat. Die GastroApp wurde bereits während der Entwicklungsphase an einige Betriebe sowie an die Kabarettistin und Schauspielerin Andrea Händler verkauft.

1.4 Anforderungen

Die Auftraggeber stellten für die Entwicklung der app:center App Anforderungen, um die perfekte Einbindung in das Produkt GastroApp zu ermöglichen. Die Anforderungen waren eine standardkonforme Android Applikation, die die Design-Richtlinien von Google einhält und einfach sowie intuitiv zu bedienen ist. Die Applikation soll die vollständige Administration von folgenden Modulen gewährleisten:

- Dashboard: Überblick über die wichtigsten Daten der eigenen GastroApp
- Neuigkeiten: Modul, welches das Bearbeiten der Neuigkeiten ermöglicht
- Push-Nachrichten: ermöglicht das Senden von Push-Nachrichten an die Benutzer der GastroApp
- Bilder: ermöglicht das Anlegen und Bearbeiten von Bilder-Galerien
- Speisekarte: ermöglicht es Speisekarten mit Gerichten und Getränken anzulegen
- Wochenkarte: ermöglicht es für bestimmte Wochen eine Wochenkarte anzulegen
- Produkte: gibt Auskunft über gewisse Produkte des Betriebes, die besonders hervorgehoben werden sollen
- Produkt des Monats: der Gastronom kann ein besonderes Produkt mit diesem Modul hervorheben
- Veranstaltungen: der Gastronom kann mit diesem Modul Veranstaltungen anlegen
- Turniere: der Gastronom kann mit diesem Modul Turniere anlegen
- Feedback: mit diesem Modul kann der Gastronom Feedback von seinen Kunden sehen und seinen Kunden antworten
- Kontakt: mit dem Kontakt-Modul kann der Gastronom seine Anschrift, grundlegende Informationen und seine Links zu sozialen Medien speichern
- Statistik: im Statistik-Modul kann der Gastronom Nutzer- und Downloadstatistiken seiner GastroApp überwachen

- Über uns: mit diesem Modul kann der Gastronom Informationen über seinen Betrieb und seine Mitarbeiter für seine Kunden zur Verfügung stellen

Dadurch, dass die GastroApp ständig weiterentwickelt wird, kamen während der Implementierung der app:center App häufig Änderungen auf uns zu. So lernten wir die Applikation so aufzubauen, dass Änderungen keine großen Auswirkungen hatten.

1.5 Motivation

Wir wählten diese Diplomarbeit, da mobile Applikationen immer wichtiger werden und wir unser Wissen in diesem Bereich steigern wollten. Außerdem stand eine hohe Benutzerfreundlichkeit an erster Stelle, was unsere Motivation, eine durchdachte und innovative Applikation zu entwickeln, weiter erhöhte.

1.6 Ergebnisse

Alle Anforderungen des Auftraggebers konnten erfüllt werden und unser Wissen im Bereich mobile Applikation steigerte sich enorm.

Unsere Applikation ermöglicht dem Kunden seine GastroApp optimal zu verwalten. Die Oberfläche wurde intuitiv gestaltet und ermöglicht dem Benutzer seine Informationen für den Kunden schnell und einfach zu bearbeiten. Wenn der Gastronom eine Idee hat, welche er über die GastroApp realisieren möchte, muss er nur die app:center App auf seinem Smartphone öffnen und hat innerhalb einiger Handgriffe bereits die Änderung für seine Kunden zur Verfügung gestellt. Falls der Gastronom Fotos zur Verfügung stellen möchte, kann er direkt mit seinem Smartphone die Fotos machen und über die app:center App hochladen.

Mit der app:center App erhält der Gastronom eine durchdachte Android-Applikation, bei der die Benutzerfreundlichkeit an erster Stelle steht. Bereits einfache Kenntnisse mit Smartphones ermöglichen das flüssige Arbeiten mit der app:center App.

1.7 Überblick der Diplomarbeit

Diese Diplomarbeit ist in die Kapitel Einleitung, Grundlagen, Benutzung der Applikation, Programmstruktur, Implementierung und Beurteilung gegliedert. Die Einleitung umfasst die Kurzbeschreibung, Aufgabenstellung der Diplomarbeit und die erhaltenen Ergebnisse. Im Kapitel Grundlagen werden die verwendeten Technologien und Architekturen beschrieben. Im Kapitel Benutzung der Applikation ist beschrieben, wie die Applikation zu verwenden ist und welche typischen Programmabläufe sowie Ergebnisse auftreten. In den Kapiteln Programmstruktur und Implementierung wird die Architektur, Aufbau der Applikation sowie die Umsetzung beschrieben. Außerdem werden gewisse Programmabläufe, die bei mobilen Applikation häufig vorkommen und schwierig zu implementieren sind, beschrieben. Anschließend wird im Kapitel Beurteilung die eigene Arbeit kritisch beurteilt.

Kapitel 2

Grundlagen

2.1 Schichtenarchitektur

Die Schichtenarchitektur ist eine Client-Server Architektur, welche unterschiedliche Aspekte einer Software-Anwendung auf verschiedene Schichten verteilt. Eine Ausprägung der Schichtenarchitektur ist die Drei-Schichten-Architektur (engl. three-tier architecutre). Die Präsentation, Geschäftslogik und Datenspeicherung werden in verschiedenen Schichten implementiert.

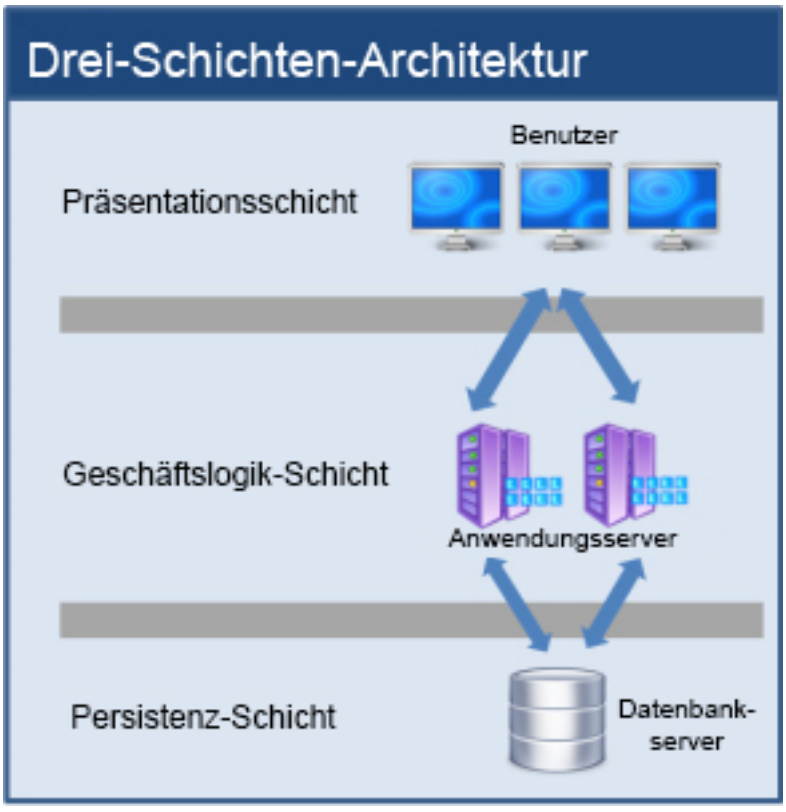


Abbildung 2.1: Drei-Schichten-Architektur [50]

2.1.1 Präsentationsschicht

Wie in Abbildung 2.1 gezeigt, stellt die Präsentationsschicht die oberste Schicht im Drei-Schichten-Modell dar. Die Aufgabe der Präsentationsschicht ist die Darstellung der Daten und Funktionen in einer benutzerfreundlichen Form. Dem Benutzer sollen die für ihn relevanten Daten übersichtlich aufbereitet werden. Beispiele für die Präsentationsschicht sind Web-Anwendungen (z.B. Amazon, Youtube) und Mobile Applikationen.

2.1.2 Logik-Schicht

Die mittlere Schicht umfasst die Geschäftslogik, also die logische Verarbeitung der Daten und Befehle. Sie verarbeitet Kommandos, übernimmt Berechnungen und tauscht Daten zwischen der Präsentations- und Datenschicht aus. Die Logik-Schicht stellt Daten aus der Datenbank für den Benutzer zur Verfügung und verwaltet welche Daten er sehen, erzeugen, ändern oder löschen darf.

2.1.3 Persistenz-Schicht

Die Persistenz-Schicht enthält zumeist eine Datenbank und die physische Speicherung der Daten. Hier werden alle Informationen einer Applikation gespeichert und der Geschäftslogik über eine geeignete Schnittstelle zur Verfügung gestellt.

2.1.4 Datentausch

Der Zugriff auf Daten und Befehle erfolgt immer von Schicht zu Schicht. Die Präsentationsschicht kann beispielsweise nicht direkt auf die Datenbank der Persistenz-Schicht zugreifen, sondern sie gibt Anfragen über die Geschäftslogik ab. Die Geschäftslogik-Schicht übernimmt dann den Zugriff auf die Datenbank, verarbeitet die Daten und stellt sie der Präsentationsschicht zur Verfügung.

2.1.5 Vor- und Nachteile

Durch das Drei-Schichten-Modell werden Abhängigkeiten innerhalb eines Systems klar definiert und die Komplexität verringert. Somit ist die Architektur einfacher zu verstehen und besser zu warten. Änderungen oder ein Austausch einer Schicht sind einfach realisierbar und betreffen andere Schichten nur minimal. Vor allem bei Anwendungen, bei denen mehrere verteilte Benutzersysteme auf dieselben Daten zugreifen wollen, spürt man die Vorteile der Schichtenarchitektur in der Entwicklung. Sobald die Serverseite inklusive Geschäftslogik fertig implementiert ist, können beliebige Benutzeroberflächen auf verschiedenen Betriebssystemen und für verschiedene Anwendungsbereiche einfach programmiert werden. Dabei muss die Serverseite nicht geändert werden und auch später sind Erweiterungen leichter möglich. Aufgrund der häufigen Weiterleitungen und erneuten Verarbeitungen der Daten leidet die Verarbeitungsgeschwindigkeit. In Systemen in denen die Antwortzeit oberste Priorität hat, wird meist auf die Vorteile einer Schichtenarchitektur verzichtet.

2.2 Schichtenarchitektur in der Praxis

2.2.1 Persistenz-Schicht

Server

Der Server beinhaltet die Datenbank und die Geschäftslogik. Am Server werden alle Daten gespeichert und verfügbar gemacht. Es kann Linux oder auch Windows als Betriebssystem verwen-

det werden, wobei die Entscheidung des Betriebssystems auch sämtliche weiteren Entscheidungen über die eingesetzten Technologien beeinflusst. Der Vorteil von Linux als Serverbetriebssystem (z.B. Debian) ist der kostenlose Betrieb und eine große Anzahl von Linux-Benutzern, welche die meisten Fragen zur Konfiguration oder Problemen bereits im Internet geklärt haben. Die Hardware des Servers kann entweder selber gekauft und verwaltet werden, aber auch von einem Anbieter gemietet werden.

Datenbank

Die Wahl der Datenbank sollte gut überlegt sein, denn auch hier entscheidet sich welche Technologien in weiterer Folge verwendet werden können. MySQL hat sich als eine der meistverbreiteten Open-Source Datenbanken in diesem Sektor bewährt.

2.2.2 Geschäftslogik-Schicht

Die Geschäftslogik wird von einem Webserver zur Verfügung gestellt. Der Zugriff auf die Geschäftslogik, und somit auf die Daten, erfolgt über eine Benutzerauthentifizierung. Je nach Anwendungsfall dürfen alle oder nur bestimmte Benutzer zugreifen.

Object Relational Mapping (ORM)

Eine Hauptaufgabe der Geschäftslogik ist die Abbildung der relationalen Datenbank in einer objekt-relationalen Form zur Verfügung zu stellen.
Das erleichtert die Programmierung mit einer objektorientierten Programmiersprache stark, da das Programm weiterhin mit Objekten arbeiten kann. Die Daten aus der Datenbank werden durch den ORM als Objekte dargestellt.
Die meisten Datenbanken speichern ihre Daten relational, also in Tabellen, aber objektorientierte Programmiersprachen speichern die Daten in Objekten. Darum wird ein ORM benötigt. Diese Objekte haben eine eigene Identität, einen Zustand und ein Verhalten. Objekte schützen Daten durch Zugriffsmethoden. Diesen Schutz nennt man Kapselung. In relationalen Datenbanken existieren solche Schutzmechanismen nicht. Dieser Widerspruch von relationalen Datenbanken und objektorientierten Programmiersprachen wird als object-relational impedance mismatch bezeichnet.[31]
Es gab viele verschiedene Lösungsvorschläge für dieses Problem. Einer davon waren objektorientierte Datenbanken, welche aber im Vergleich zu relationalen Datenbanken langsamer sind. Der Vorteil eines ORM ist, dass die Vorteile einer relationalen Datenbank, wie etwa die schnelle Datenabfrage, mit einer objektorientierten Programmiersprache trotzdem genutzt werden können. Ein ORM übernimmt normalerweise eine Klassenbibliothek, wie beispielsweise ADO.NET Entity Framework von Microsoft oder die Referenzimplementierung EclipseLink der JPA (Java Persistence API).

Schnittstellen

Objekte, welche die Daten für den Benutzer darstellen, müssen nun für die Präsentationsschicht über geeignete Schnittstellen zur Verfügung gestellt werden. Das geschieht im Allgemeinen über sogenannte Webservices. Ein Webservice ermöglicht die Kommunikation zwischen zwei Geräten über das Netzwerk. Ein Webservice besitzt einen URI (Uniform Resource Identifier) für die Identifikation im WWW. Außerdem stellt ein Webservice eine Schnittstellenbeschreibung zur Verfügung, welche darüber Auskunft gibt wie mit dem Service zu kommunizieren ist.
Die Übertragung erfolgt meistens über das http-Protokoll und das Format der Daten kann beliebig gewählt werden. Am häufigsten werden die Datenbeschreibungssprachen XML oder

JSON verwendet. Die Extensible Markup Language (XML) ist eine Sprache zur Darstellung von Daten in einer Hierarchie. Die JavaScript Object Notation (JSON) beschreibt ein Datenformat, in der die Daten als Objekte dargestellt werden. Der Vorteil von JSON gegenüber XML in Bezug zum reinen Datenaustausch ist eine einfachere Lesbarkeit und eine geringere Größe der JSON-Objekte im Vergleich zu XML-Objekten. [39]
Webservices können mit verschiedenen Technologien, wie REST und SOAP, umgesetzt werden.

REST (Representational State Transfer)

REST basiert auf den Prinzipien, die bereits im World Wide Web eingesetzt werden. Der Kern von REST sind Ressourcen. Ressourcen sind alles, was eindeutig identifiziert werden kann. Beispiele für Ressourcen sind die Entitäten eines Datenmodells (Bsp.: Benutzer, Produkte, ...). Neben den Entitäten können auch spezifische Daten der Entitäten durch Ressourcen dargestellt werden (Bsp.: Benutzer, die erst seit einem Jahr registriert sind). Alle Ressourcen benötigen einen einzigartigen URI [54]. Diese Einteilung von Ressourcen hat viele Vorteile, denn so kann man gewisse Ressourcen frei publizieren und andere nur gewissen Benutzern zugänglich machen. Der Zugriff auf eine Ressource erfolgt standardmäßig mit http über den URI. Daraufhin erhält der Benutzer die gewünschten Daten, formatiert mit JSON oder XML. Für jede Ressource stehen die http-Methoden GET (lesen), PUT (Ressource anlegen und bearbeiten), POST (Ressource anlegen und bearbeiten) und DELETE (Ressource löschen) zur Verfügung. Mit diesen Methoden lassen sich alle Anwendungsfälle abdecken. [44]

Beispiele für REST-Zugriffe

Als Beispiel dient ein Onlineshop, der Produkte über das Internet an seine Kunden verkauft. Der Kunde besitzt einen Warenkorb, der die ausgewählten Produkte bis zum Kauf speichert. Außerdem werden alle bereits gekauften Produkte des Kunden gespeichert. Alle Entitäten besitzen eine eindeutige Nummer, die die Ressource definiert (ID). Der REST Service ist mit der URL des Anbieters und dem Pfad, auf dem der REST-Service zur Verfügung gestellt wird, erreichbar. Als Datenformat wird JSON benutzt. Im Folgenden befinden sich die Zugriffe, um die gewünschten Ressourcen zu erhalten.
Beispielhafter Zugriff, um alle Kunden des Webshops zu erhalten:

```
GET www.webshop.at/api/kunden
```

Beispielhafte Repräsentation, der Liste, aller Kunden:

Listing 2.1: JSON Repräsentation aller Kunden

```
[
   {
      "id": 1,
      "active": true,
      "firstname": "Max",
      "lastname": "Mustermann",
      "created": "2014-06-27",
      "username": "max_m",
      ...
   },
   {
```

```
12        "id": 2,
13        "active": true,
14        "firstname": "Maria",
15        "lastname": "Musterfrau",
16        "created": "2014-08-27",
17        "username": "maria_m",
18        ...
19     },
20     {
21        ...
22     },
23     ...
24  ]
```

Möchte man jedoch nur auf einen bestimmten Kunden zugreifen, so könnte der Aufruf so aufgebaut sein:

```
1  GET /api/kunden/1
```

Es wird genau auf den Kunden mit der ID 1 zugegriffen und dieser wird mit JSON formatiert zurück gesendet.

Der Zugriff auf alle Produkte, die der Kunde bereits gekauft hat, kann mit einem zusätzliche Parameter realisiert werden. Dem Service wird über den Parameter „kunde“ die ID des gewünschten Kunden übergeben. [7]

```
1  GET /api/produkte?kunde=1
```

SOAP (Simple Object Access Protocol)

SOAP ist ein standardisierter Mechanismus, welcher den Austausch von strukturierten Daten in XML mit dem HTTP-Protokoll zwischen mehreren Systemen ermöglicht. Bei SOAP Webservices werden Regeln für Parameter und Rückgabetypen in einer WSDL-Datei definiert, die bestimmen wie die Daten in der Nachricht abgebildet werden sollen und wie der Zugriff auf die Daten erfolgt [55].

Aufbau von SOAP-Nachrichten

SOAP-Nachrichten bestehen aus einem Umschlag (engl. Envelope), der die Daten und einen optionalen Kopf (engl. Header) enthält (siehe Abbildung 2.2). Der Kopf dient für Meta-Informationen, die zum Beispiel zur Verschlüsselung dienen. [43]

Unterschied zu REST

REST und SOAP unterscheidet unter anderem der Adressraum. REST adressiert jede einzelne Ressource und SOAP besitzt einen zentralen Verteiler (Router), welcher alle Anfragen verwaltet. Der Benutzer kann bei REST direkt auf eine Ressource zugreifen, bei SOAP sendet er seine Anfrage an den Verteiler, welcher seine Anfrage bearbeitet und ihm die passenden Informationen sendet. [44]

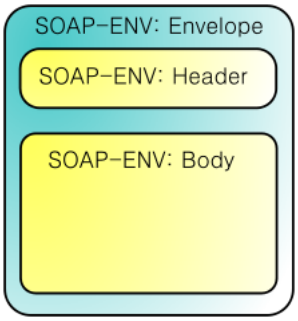


Abbildung 2.2: Aufbau einer SOAP-Nachricht [43]

2.2.3 Präsentationsschicht

Die Präsentationsschicht umfasst alle Programme für den Endbenutzer, sowie alle Programme die zur Administration der Anwendung dienen. Der Administrator soll vollen Zugriff auf alle Daten haben und die Anwendung über eine Oberfläche verwalten können. Dies umfasst beispielsweise die Verwaltung von Produkten, Berechtigungen oder den Zugang zu Statistiken. Der Kunde bzw. Anwender des Dienstes darf nur auf die Daten, die für ihn notwendig sind, Zugriff haben. Zum Beispiel darf er bei einem Online-Shop sämtliche Produkte ansehen, in seinen Warenkorb legen und bestellen. Der Administrator darf auch Produkte anlegen, ändern und löschen. Diese Zugriffskontrolle wird jedoch nicht von der Präsentationsschicht, sondern von der Geschäftslogik-Schicht abgewickelt. Die Präsentationsschicht sendet Anfragen an die Webservices, erhält daraufhin eine Antwort, verarbeitet diese Daten und stellt sie in einer geeigneten Form für den Benutzer dar.

2.3 Mobile App

Als Mobile App, meist auch nur App genannt, wird eine Anwendungssoftware für Mobilgeräte bzw. mobile Betriebssysteme bezeichnet. Meist handelt es sich um Anwendungen für Smartphones und Tablet-PCs. Es sind viele verschiedene Betriebssysteme für Mobilgeräte vorhanden. Im Grunde unterscheidet man zwischen Android, BlackBerryOS, Apple iOS und Windows Phone. Mobile Apps, welche auf dem jeweiligen Betriebssystem installiert werden müssen, können über einen in das Betriebssystem integrierten Onlineshop bezogen und danach direkt auf dem Mobilgerät installiert werden.

2.4 Native Mobile App

Native mobile Apps sind speziell auf ein bestimmtes Betriebssystem angepasst. So ist es beispielsweise nicht möglich eine native mobile App, die für das Betriebssystem Android entwickelt wurde, auf einem mobilen Gerät, welches Apple iOS verwendet, zu installieren und auszuführen. Der Programmierer einer nativen App, muss diese Anwendung für jedes mobile Endgerät einzeln entwickeln und anpassen. Für jedes mobile Betriebssystem existiert ein eigener SDK (Software Development Kit), welcher von Entwicklern genutzt werden kann. Die bevorzugten Programmiersprachen für die größten Betriebssysteme sind C#, C und C++ für Microsoft Windows Phone, Objective-C für Apple iOS und Java für Android. Die nativen mobilen Apps werden als Programm auf dem jeweiligen Betriebssystems ausgeführt und haben daher Zugriff auf Ressourcen des Gerätes, wie etwa Netzwerke, Datenträger, Dateien, Dokumente, GPS, Audio und verschiedene andere Sensoren.

2.4.1 Android

Native Android Apps werden mit der Programmiersprache Java entwickelt. Für die Programmierung benötigt man den JDK (Java Development Kit) und den Android SDK. Diese beiden Baukästen werden im Internet auf diversen Seiten kostenfrei als Download angeboten. Android Apps können kostenlos, ohne jegliche Registrierung, auf jedem Android Gerät getestet werden. Um Android Apps im Google Play Store verkaufen zu können, muss man ein Entwicklerkonto erstellen, bei dem einmalige Registrierungsgebühren anfallen.

Versionen

Das Android-Betriebssystem wurde oft überabeitet und daher liegen viele verschiedene Versionen des Betriebssystems vor. Die Versionen besitzen unterschiedliche Namen und auch unterschiedliche Programmierschnittstellen (APIs - Application Programming Interfaces). Eine API ist ein Programmteil, welcher von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. Zur Bereitstellung von Programmierschnittstellen gehört eine detaillierte Dokumentation der Schnittstellen-Funktionen mit ihren Parametern. Wenn die API Stufe steigt, steigt auch die Funktionalität. Es ist jedoch auch möglich, dass gewisse Funktionen durch höhere API Stufen verloren gehen. [42]
Die unterschiedlichen API Stufen führen zu Problemen bei der Entwicklung von Android Apps. Am Markt sind verschiedene Geräte mit unterschiedlichen Android Versionen und daher auch unterschiedlichen API Stufen erhältlich. Geringere API Stufen unterstützen gewisse Funktionen nicht, welche neuere API Stufen schon unterstützen. Darum ist es als Entwickler schwierig, eine geeignete API Stufe für die App zu finden, damit die App auf allen Geräten funktioniert. Wenn man beispielsweise die App für API Stufe 14 entwickelt, kann die App später nicht auf Geräten mit API Stufe 13 und geringer laufen. Je geringer die API Stufe ist, desto mehr Geräte können die App ausführen. Jedoch stehen dadurch auch weniger Funktionen zur Programmierung zur Verfügung.
Tabelle 2.1 zeigt einige Android Versionen mit ihren Namen, den API Stufen und auf wie viel Prozent aller Android Geräte die App laufen würde, wenn man diese API Stufe als Minimum SDK ansetzen würde. Der Minimum SDK gibt die minimale API Stufe an, die ein Android-Gerät besitzen muss, damit die App auf diesem Gerät funktioniert.

Android Version	API Stufe	Lauffähigkeit auf Android Geräten, falls API Stufe als Minimum SDK angesetzt wird
Android 1.0 (Base)	1	100%
Android 2.0 (Eclair)	5	100%
Android 2.3 (Gingerbread)	9	99,3%
Android 3.0 (Honeycomb)	11	87,9%
Android 4.0 (Ice Cream Sandwich)	14	87,9%
Android 4.1 (Jelly Bean)	16	78,3%
Android 4.4 (KitKat)	19	24,5%
Android 5.0 (Lollipop)	21	00,0%

Tabelle 2.1: Android Versionen mit ihrer Lauffähigkeit

Um die Probleme der unterschiedlichen API Stufen zu umgehen, wurden sogenannte Support Libraries entwickelt. Das Android Support Library Paket ist eine Sammlung von Quelltext Bibliotheken, die abwärtskompatible Versionen von Android APIs bereitstellen. Wenn man beispielsweise eine App für Android API Stufe 14 entwickelt, wäre diese App ohne Support Libraries auf Geräten mit API Stufe 13 und geringer nicht ausführbar. Support Libraries stellen die benötigten Funktionalitäten der API Stufe 14 bereit und somit ist es möglich, diese App auf Geräten ab API Stufe 4 uneingeschränkt auszuführen. Jede Support Library zielt auf eine bestimmte Android API Stufe ab und stellt eine unterschiedliche Gruppe an Funktionen bereit. Jedem Entwickler muss bewusst sein, welche Funktionen man in seiner App verwenden möchte und welche Funktionen von den unterschiedlichen Support Libraries tatsächlich unterstützt werden. Im Grunde werden von Google die Support Libraries der Version 4 und der Version 7 empfohlen. Support Library v4 ist entwickelt worden, um Android Geräte mit der API Stufe 4 und höher zu unterstützen. Die Hauptfunktionen, welche durch diese Support Library unterstützt werden, sind Layouts und Benutzeranzeigen. Support Library v7 ist entwickelt worden, um Android Geräte mit der API Stufe 7 und höher zu unterstützen. Die Hauptfunktionen, welche durch diese Support Library unterstützt werden, sind Leisten zum Anzeigen von Benutzerfunktionen, sogenannte ActionBars. Diese Funktionen sind für die Anzeige von Navigationsleisten notwendig. ActionBars wurden in der API Stufe 11 und höher eingeführt. Um diese mit einer API Stufe 11 und geringer verwenden zu können, muss die Support Library v7 verwendet werden. [29]
Da durch die API Stufe 9 bereits eine Lauffähigkeit von 99,3% aller Geräte erreicht werden kann (siehe Tabelle 2.1), sollte für die Erstellung einer App die API Stufe 9 gewählt werden. Außerdem können durch die Support Library v4 und Support Library v7 die wichtigsten Funktionen abgedeckt werden, die ansonsten erst mit höheren API Stufen erreicht werden könnten.

Entwicklungsumgebung

Die zwei bekanntesten Entwicklungsumgebungen für Android sind Eclipse und Android Studio. Für die Entwicklung der gesamten Arbeit wurde Android Studio verwendet. Android Studio ist eine freie Entwicklungsumgebung von Google. Android Studio bietet folgende Funktionen [4]:

- Unterstützung für die Entwicklung von Android Wear und Android TV Applikationen
- Smart Editing
  - Sobald getippt wird, erscheinen Vorschläge für den Quelltext
- Statische Code Analyse
  - Android Studio überprüft den Quelltext automatisch und liefert eine Liste von Verbesserungsmöglichkeiten
- Unterstützung eines Grafischen Layout Editors
- Open-Source-Entwicklung
  - Der Quelltext von Android Studio ist frei verfügbar
- Auf Gradle basierte Build-Tools

Gradle

Gradle ist ein auf Java basierendes Build-Tool. Build-Tools sind Werkzeuge, die ein vorübergehendes, anwendbares Programm automatisch erzeugen. Gradle wurde für Builds von Softwaresystemen entworfen, welche aus einer Vielzahl von Projekten bestehen. Builds umfangreicher Projekte können viel Zeit in Anspruch nehmen. Darum unterstützt Gradle sowohl inkrementelles Builden, als auch Builden der Software durch parallel ablaufende Build-Prozesse. Inkrementelles Builden bedeutet, dass nur jene Teile der Software erzeugt werden, welche verändert wurden, oder auf veränderten Teilen beruhen. Parallel ablaufende Build-Prozesse ermöglichen es, dass bestimmte Tasks beim Builden (z.B. Tests) parallel auf verschiedenen CPUs oder Rechnern laufen. Damit lässt sich eine wesentlich höhere Geschwindigkeit des Erstellprozesses erreichen. Gradle wird von einigen bekannten Frameworks für deren Builds eingesetzt. Darunter sind zum Beispiel Hibernate, Grails, Spring Integration und Spring Security. Im Jahr 2013 ist das Android-System hinzugekommen. [34]
Falls in Android Studio ein neues Projekt erstellt wird, werden automatisch folgende Gradle-Dateien erzeugt:

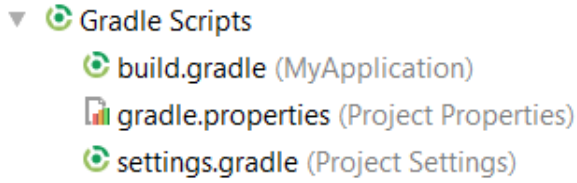


Abbildung 2.3: Gradle Scripts Verzeichnisstruktur

In der Datei build.gradle ist die Definition des Builds mit allen Tasks und Abhängigkeiten eines Projekts festgelegt. Ein Multiprojekt (besteht aus mehreren verschiedenen Projekten) hat pro Projekt eine solche Build-Datei. Die Datei gradle.properties enthält eine Liste von Eigenschaften, welche für die projektspezifische Gradle-Initialisierung eines Builds gültig sind. In der Datei settings.gradle werden für ein Multiprojekt, also ein Projekt, das aus mehreren abhängigen Projekten besteht, die teilnehmenden Unterprojekte festgelegt.

Aufbau

Auf Grund der Vielfalt an Programmiersprachen und Frameworks gibt es auch viele Möglichkeiten, um grafische Benutzerschnittstellen (GUI – Graphical User Interface) zu entwickeln. In Java ist es möglich mit Swing-Klassen zu arbeiten und somit Container-Klassen zu definieren. In diese Container kann man Steuerelemente einfügen. Auf diese Weise entsteht eine hierarchische Struktur. Im .NET-Framework ist es mithilfe von XML-Dokumenten möglich, Steuerelemente hierarchisch zu strukturieren. Die eigentliche Logik, welche die Interaktion mit dem Anwender steuert, wird dann sowohl in Java, als auch im .NET-Framework im zusätzlichen Programmcode definiert. Auf diese Weise erhält man eine Trennung zwischen Präsentation und Anwendungslogik. Bei Android Apps werden nicht Swing-Klassen aus dem Java-SDK verwendet, sondern es werden die GUIs nach einem ähnlichen Konzept wie im .NET-Framework definiert. Die Darstellung der GUI wird in XML-Layout Dateien durchgeführt. In einer Android-App kann auf die GUI in Form von Objekten des Typs View zugegriffen werden. Diese View-Objekte reflektieren die Struktur und die Namensgebung, wie sie in der XML-Datei hinterlegt sind. Die eigentliche Anwendungslogik wird in Objekten vom Typ Activity hinterlegt.[56]

Eine Activity ist ein Objekt, das direkt an der Benutzerfront arbeitet:

- Sie ist mit einer XML-Layout Datei verbunden und kann die Buttons, Eingabefelder, Checkboxen, etc. der GUI steuern.
- Wenn ein Anwender sich entschließt mit der GUI zu interagieren, werden Events ausgelöst, auf welche die Activity reagieren kann.

Lebenszyklus einer Activity

Activities werden auf einem Stack (Stapel) verwaltet. Wenn eine neue Activity gestartet wird, wird sie oben auf den Stapel gelegt. Wenn die Zurück-Taste gedrückt wird, wird die zuvor aktive Activity wieder aktiv. Im Falle von Ressourcenknappheit (geringer Arbeitsspeicher) werden inaktive Activities vom Android Betriebssystem beendet. Activities verfügen über Methoden, die in den unterschiedlichen Phasen des Lebenszyklus aufgerufen werden.

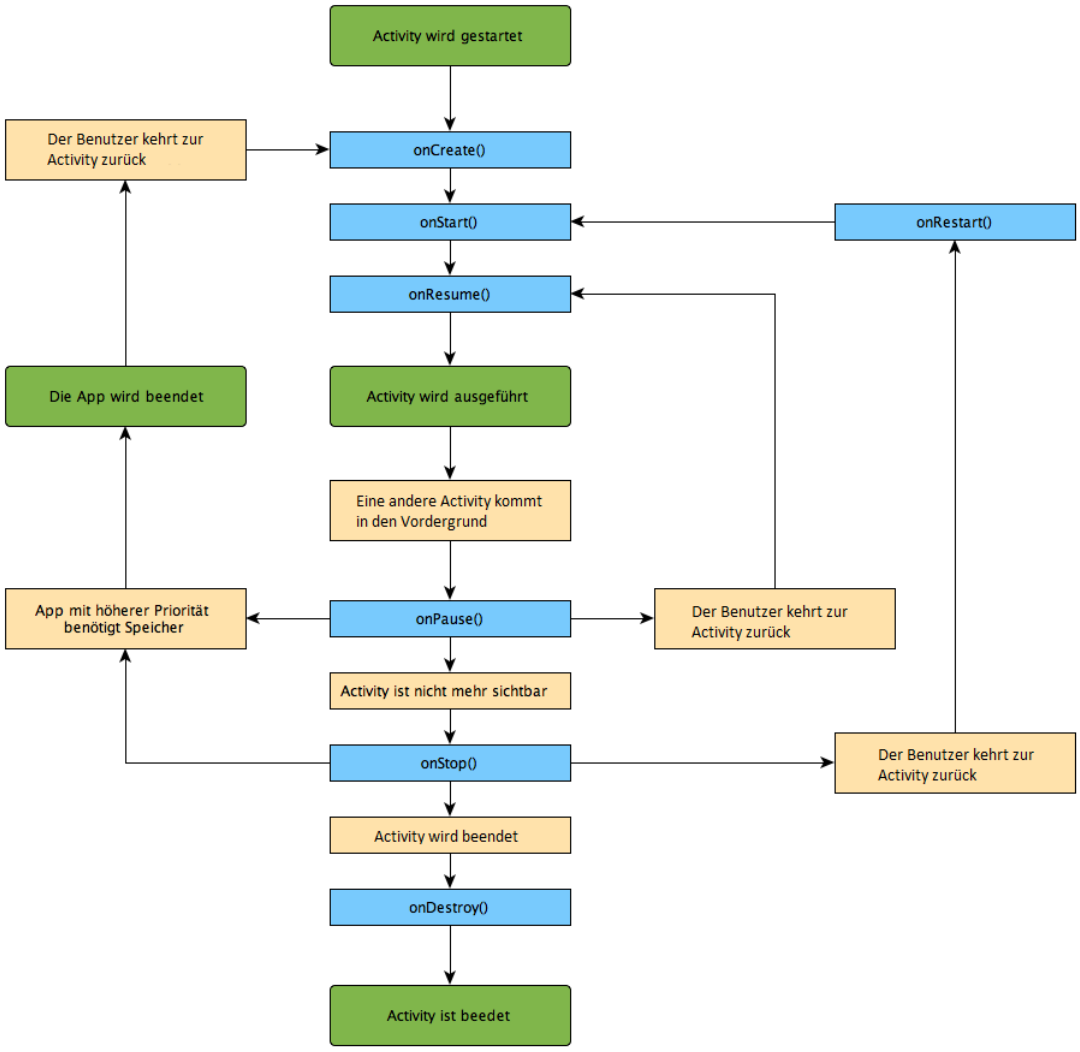


Abbildung 2.4: Lebenszyklus einer Android Activity [51]

Jede Activity enthält die sogenannte onCreate Methode. Wie man in der Abbildung 2.4 sehen kann, wird diese Methode jedes Mal aufgerufen, wenn eine Activity neu erschaffen wird, also wenn ein neues Objekt der Klasse MyActivity erstellt wird. Falls eine bestehende Activity, die sich im Hintergrund befindet, erneut aufgerufen wird, wird die onRestart- bzw. onStart-Methode aufgerufen. Durch die Methode onDestroy wird eine Activity beendet.
Eine Activity Klasse kann in Android wie folgt aussehen:

Listing 2.2: Activity Klasse in Android

```
1 public class MainActivity extends Activity {
2    @Override
3    public void onCreate(Bundle savedInstanceState) {
4       super.onCreate(savedInstanceState);
5       setContentView(R.layout.main);
6    }
7 }
```

In der onCreate Methode sollte die Zuweisung eines Layouts zur Activity erfolgen. Dies wird durch die Methode setContentView erreicht. Dieser Methode wird als Parameter eine Layout-Ressource mitgegeben. In diesem Fall handelt es sich um die XML-Layout Datei main.xml. Diese Datei befindet sich im Applikations-Ressourcen Ordner.

Die XML-Layout Datei kann wie folgt aussehen:

Listing 2.3: XML-Layout Datei in Android

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6
7      <TextView
8          android:layout_width="wrap_content"
9          android:layout_height="wrap_content"
10         android:text="@string/hello"/>
11
12 </LinearLayout>
```

Diese XML-Datei beschreibt, wie die Benutzeroberfläche einer Activity aufgebaut ist. Es wird ein Grundlayout benötigt, um weitere Steuerelemente einfügen zu können. In diesem Fall wird ein Linear Layout verwendet. Es gibt jedoch verschiedene Arten von Layouts. Sie unterscheiden sich in der Anordnung der unterschiedlichen Steuerelemente. So können sie beispielsweise in Form von Tabellen oder in Listen angeordnet werden. Für jeden Anwendungsfall kann ein anderes Layout von Vorteil sein. Das Layout kann konfiguriert werden. Durch die Attribute android:layout_width und android:layout_heigth wird festgelegt, wie breit und hoch das jeweilige Layout ist.

Man kann zwischen folgenden Attributen für Höhe und Breite unterscheiden [21]:

- fill_parent
  - Bedeutet, dass das Layout so groß ist, wie das übergeordnete View-Element. In diesem

Fall ist das Layout so groß wie der Bildschirm des Android-Geräts, auf dem die App läuft. Dieses Attribut ist eher veraltet und wurde ab der API Stufe 8 durch match_parent abgelöst.

- match_parent
  - Bedeutet, dass das Layout so groß ist, wie das übergeordnete View-Element abzüglich des Paddings des übergeordneten View-Elements. Padding bedeutet, dass sich rund um das View-Element ein Rahmen befindet. Das View-Element an sich wird durch das Padding nicht größer.
- wrap_content
  - Bedeutet, dass das Layout nur so groß ist, dass der eigene Inhalt, unter Berücksichtigung des Paddings, Platz hat.

Im Layout können Steuerelemente eingefügt werden. In diesem Fall wird eine Text View eingefügt. Damit die Text View weiß, welchen Text sie am Bildschirm darstellen soll, gibt es zwei verschiedene Möglichkeiten:

- Man kann dem Attribut android:text den anzuzeigenden Text direkt übergeben.
- Man kann dem Attribut android:text eine Referenz auf einen Eintrag in einer Applikations-Ressourcen Datei übergeben.

In diesem Fall erhält die Text View eine Referenz auf den String namens „hello“. Dieser String wurde in der Datei strings.xml definiert. Möchte man den Text direkt übergeben, kann das wie folgt aussehen:

```
1 android:text="Hello World!"
```

Die strings.xml Datei kann wie folgt aussehen:

Listing 2.4: strings.xml Datei in Android

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="app_name">My Application</string>
4     <string name="hello">Hello world!</string>
5 </resources>
```

Zwischen den resources-Tags können verschiedene Strings definiert werden. Diese Strings besitzen einen bestimmten Namen und können durch @string/name aufgerufen werden. Statt „name“ kann der Name des Strings eingetragen werden. Beispielsweise ist der String mit dem Namen „app_name“ jener String, der den Namen der App beinhaltet.

Android Manifest

Jede Android App muss eine Datei namens AndroidManifest.xml beinhalten. Diese Datei muss sich im Stammverzeichnis der App befinden. Das Manifest enthält Informationen über die App, die das Android-System benötigt, um sie ausführen zu können. Es werden folgende Informationen beschrieben [3]:

- Bestimmt die Activity, die beim Start der Applikation ausgeführt wird.

- Die Berechtigungen, die eine App benötigt.
  - Beispiel:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

    * Berechtigung zur Verwendung des Internets eines Android-Gerätes.

- Welche Voraussetzungen zum Betrieb einer App benötigt werden.
  - Beispiel:

```
<uses-feature android:name="android.hardware.camera"/>
```

    * Das Android-Gerät muss eine Kamera besitzen, damit die App in Betrieb genommen werden kann.

- Welche API Stufe mindestens erforderlich ist, um die App in Betrieb nehmen zu können.
  - Beispiel:

```
<uses-sdk android:minSdkVersion="5"
        android:targetSdkVersion="14"
        android:maxSdkVersion="19"/>
```

    * MinSdkVersion gibt die minimale API Stufe an, die ein Android-Gerät besitzen muss, damit die App auf diesem Gerät funktioniert. Falls das Gerät eine geringere Stufe als 5 besitzt, lässt das Android System die Installation dieser App nicht zu.
    * TargetSdkVersion gibt die API Stufe an, unter der die App getestet wurde und unter der die App einwandfrei funktioniert. Falls dieses Attribut nicht angegeben wird, ist die targetSdkVersion automatisch die minSdkVersion.
    * MaxSdkVersion gibt die API Stufe an, die ein Android Gerät maximal besitzen darf, damit die App ausführbar ist. Falls das Gerät eine API Stufe von 20 hat, kann diese App auf diesem Gerät nicht ausgeführt und installiert werden.

- Festlegung, wie die App heißt und in welchem Ordner das App Symbol liegt.
  - Beispiel:

```
<application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
</application>
```

    * Das App Symbol befindet sich im Ordner drawable und hat den Namen ic_launcher.
    * Der Name der App ist in der string.xml Datei unter dem Eintrag app_name definiert.
    * Sowohl der Ordner drawable als auch die Datei string.xml befinden sich im Applikations-Ressourcen Ordner.

Applikations-Ressourcen

Neben den Java-Klassendateien, den Layout Dateien und dem Android Manifest werden oft andere Dateien, wie etwa Bild-, Audio- und String-Dateien verwendet. Diese Ressourcen werden in den res Ordnern hinterlegt, und sind von da aus an jeder Stelle der App erreichbar. Dateien, die hier immer zu finden sind, sind die oben genannten XML-Layouts.

Fragments

Fragments sind Teile einer Activity, welche unabhängig voneinander und unabhängig vom Layout geladen und mit Funktionalität bestückt werden können. Sie existieren innerhalb von Activities und bilden somit die Grundlage für ein dynamisches Layout. Es können mehrere Fragments zu einer Activity hinzugefügt werden. Fragments können als modulare Abschnitte einer Activity angesehen werden. Jedes Fragment hat seinen eigenen Lebenszyklus, der direkt mit dem Lebenszyklus der Activity in Verbindung steht. Falls eine Activity beendet wird, werden auch alle darin befindlichen Fragments beendet. Android hat Fragments in der API Stufe 11 eingeführt. Der Grund dafür war, dass dynamischere und flexiblere Layouts für große Bildschirme (Tablet PCs) erzeugt werden mussten. Fragments sollten modular und wiederverwendbar entwickelt werden. Da jedes Fragment ein eigenes Layout, ein eigenes Verhalten und einen eigenen Lebenszyklus enthält, ist es möglich, ein Fragment in verschiedenen Activities zu integrieren. Fragments sollten daher nicht untereinander abhängig sein. Ein modular aufgebautes Fragment ermöglicht einen Einsatz auf verschiedenen Bildschirmgrößen. Wenn eine App entwickelt wird, um auf Tablet PCs und Smartphones (Handset) zu funktionieren, können die Fragments durch unterschiedliche Layout Einstellungen an die verfügbare Bildschirmgröße angepasst werden.

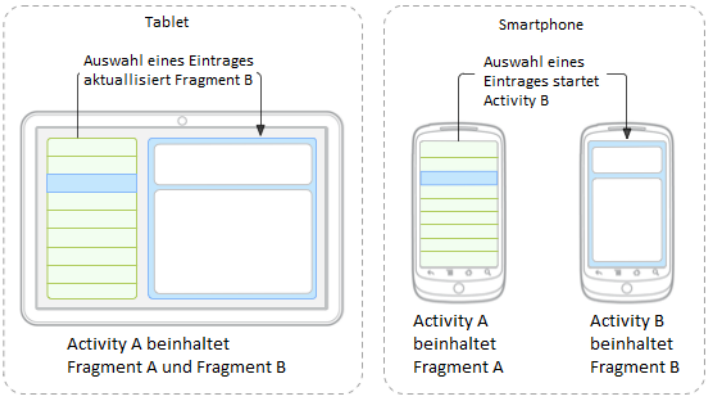


Abbildung 2.5: unterschiedlicher Einsatz von Fragments bei Tablets und Smartphones [15]

Abbildung 2.5 zeigt, dass es möglich ist, auf einem Tablet-PC sowohl Fragment A als auch Fragment B in die Activity A einzubinden. Es kann beides nebeneinander angezeigt werden, da die Bildschirmgröße ausreichend ist. Bei Smartphones muss jedoch für Fragment A eine eigene Activity A erstellt werden und für Fragment B eine eigene Activity B. Ein Beispiel dafür wäre eine Nachrichten-App: Auf einem Tablet-PC beinhaltet eine Activity zwei Fragments, welche nebeneinander angeordnet sind. Im linken Teil befindet sich eine Liste aller Nachrichteneinträge und im rechten Teil ist der Nachrichtenartikel ersichtlich. Jedes dieser beiden Fragments hat seinen eigenen Lebenszyklus und kann seine eigenen Events abarbeiten. So ist es möglich, in ein und derselben Activity auf der einen Seite die Nachrichteneinträge durchzusehen und auf

der anderen Seite den ausgewählten Artikel zu lesen. Auf einem Smartphone ist nicht genügend Platz, um beide Fragments nebeneinander anzeigen zu können. Darum beinhaltet Activity A nur das Fragment für die Liste der Nachrichteneinträge. Sobald ein Benutzer einen Eintrag aufruft, wird Activity B gestartet. Diese beinhaltet Fragment B, welches den Artikel anzeigt. Somit unterstützt die App durch Wiederverwendung von Fragments sowohl Tablet-PCs als auch Smartphones. Um diese Unterstützung muss sich der Entwickler bei der Programmierung der App nicht kümmern, da dies automatisch durch die Fragments möglich gemacht wird.[15]

Neues Projekt erstellen

Wenn ein neues Android-App-Projekt in einer Entwicklungsumgebung erstellt wird, müssen Voreinstellungen vorgenommen werden. Diese Projekt Erstellung bezieht sich auf die Entwicklungsumgebung Android Studio.

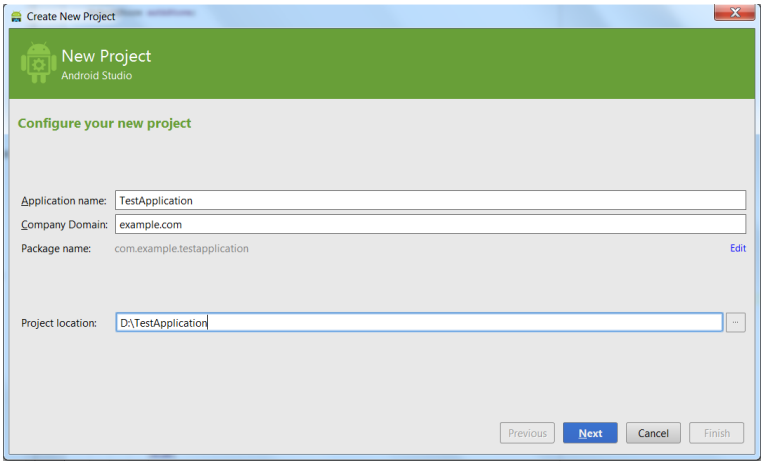


Abbildung 2.6: neues Projekt erstellen Teil 1

Als erstes müssen ein Projektname und ein Projekt Verzeichnis ausgewählt werden (siehe Abbildung 2.6). In diesem Fall lautet der Projektname TestApplication und der App Ordner befindet sich im Verzeichnis D:\TestApplication. Aus der Company Domain wird ein Package Name generiert. Alle Java-Klassen werden in dieses gemeinsame Package zusammengefasst.

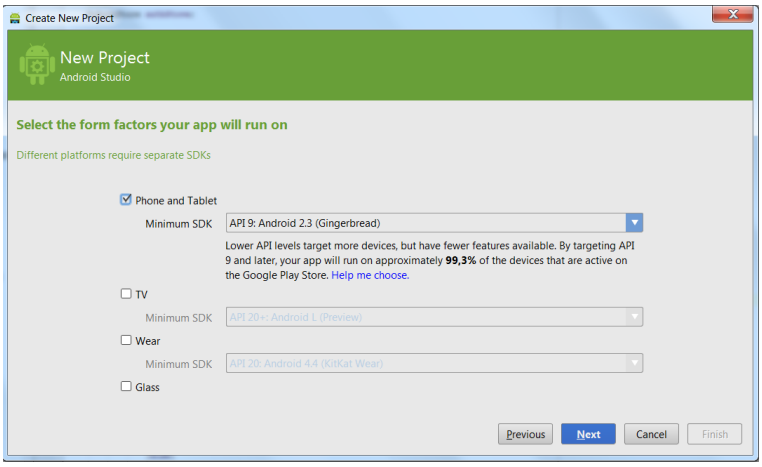


Abbildung 2.7: neues Projekt erstellen Teil 2

Als nächstes muss das Anwendungsgebiet der App ausgewählt werden. Man kann unterscheiden zwischen Smartphone und Tablet, Android TV, Android Wear und Google Glass. Danach muss der minimale SDK ausgewählt werden. In diesem Fall wird als API Stufe die Stufe 9 gewählt.

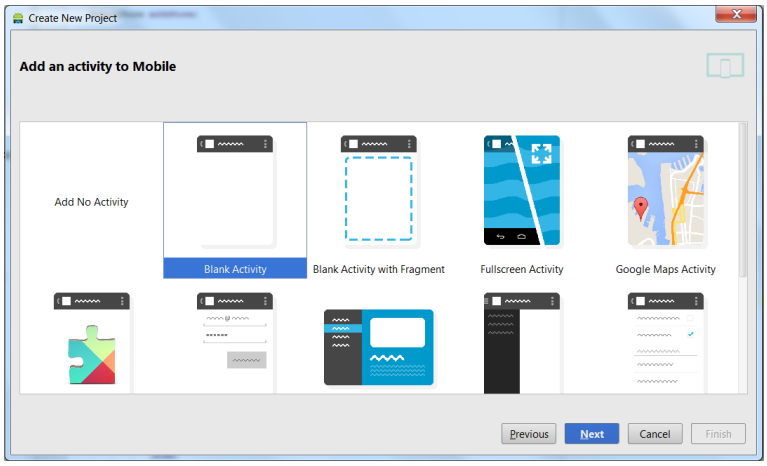


Abbildung 2.8: neues Projekt erstellen Teil 3

Optional kann eine Start Activity erstellt werden, welche beim App Start als erstes aufgerufen wird. Falls keine Activity erstellt werden soll, kann das Projekt nun erstellt werden.

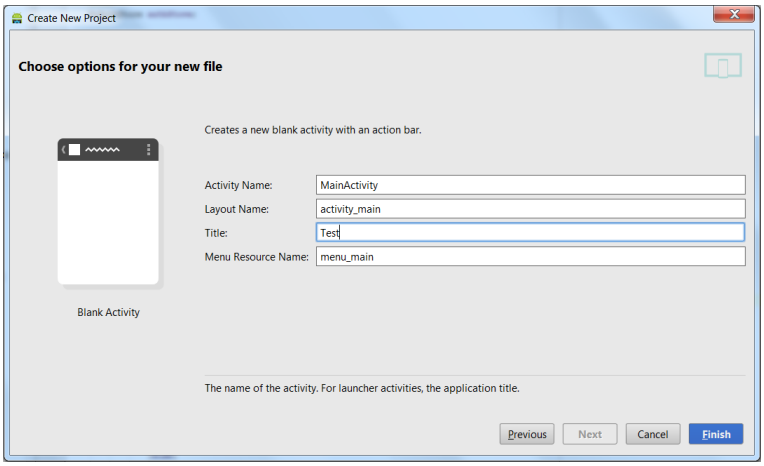


Abbildung 2.9: neues Projekt erstellen Teil 4

Falls eine Start Activity erstellt werden soll, muss der Activity ein Name gegeben werden. Dieser entspricht dem Java Klassennamen der Activity. Der Layout Name und der Menu Resource Name wird aufgrund des Activity Namens automatisch erstellt. Durch Eingabe des Layout Namens wird eine neue XML Datei mit diesem Namen erzeugt. Es muss ein Titel der Activity angegeben werden, der in der App erscheinen soll, sobald diese Activity aufgerufen wird.

Nach Eingabe dieser Voreinstellungen ist ein neues Android App Projekt in Android Studio erzeugt worden. Nun liegt folgende Ordnerstruktur vor:

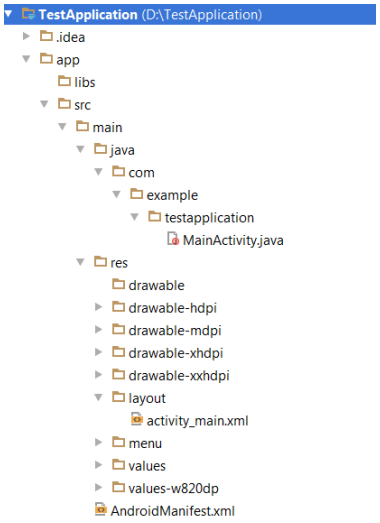


Abbildung 2.10: Ordnerstruktur eines neu erstellten Android Projekts

Im Ordner app/src/main/java/com/example/testapplication befindet sich die Java Klasse der erstellten MainActivity. Im Ordner app/src/main/res befindet sich der Applikations-Ressourcen Ordner, in dem sich alle zusätzlichen Dateien befinden. Im Unterordner layout befinden sich alle XML Layout Dateien. Im Ornder app/src/main befindet sich das Android Manifest, in welchem die Berechtigungen, Voraussetzungen und andere App Eigenschaften definiert sind. Im Ordner libs können Zusatzbibliotheken eingebunden werden, welche für die Entwicklung der App notwendig sein können.

2.4.2 iOS App

Um Apps für das Betriebssystem iOS der Firma Apple programmieren zu können, benötigt man einen Rechner mit dem Betriebssystem iOS und den SDK. Dieses kann man auf der Entwicklerseite von Apple herunterladen und es enthält die XCode-Tools, die für die App Entwicklung nötig sind. XCode ist die Entwicklungsumgebung, mit der iOS Apps entwickelt werden können. Die App Entwicklung erfolgt mit der objektorientierten Programmiersprache Objective-C. Die Software kann nur auf einem virtuellen Simulator getestet werden. Wenn man die App auf dem iPhone, iPad oder iPod selbst ausführen möchte, benötigt man die Entwicklerlizenz von Apple. Diese verursacht jährliche Kosten. Eine fertige App muss bei Apple eingereicht werden. Apple Mitarbeiter überprüfen diese App auf Stabilität und Inhalt und falls Apple die App akzeptiert, wird sie auf dem App Store verfügbar gemacht. Falls nicht, bekommt der Entwickler eine Ablehnung und eine Aufforderung, eine Korrekturversion zu liefern.

2.4.3 Windows Phone App

Windows Phone Apps sind Apps, die auf Windows Phone Geräten ausgeführt werden und über den Windows Phone Store erhältlich sind. Windows Phone Apps können in verschiedenen Programmiersprachen entwickelt werden. Darunter sind zum Beispiel JavaScript, C#, Visual Basic oder C++. Windows Phone Apps können die Windows-Runtime nutzen. Dies ist eine systemeigene, integrierte API. Diese API ist in C++ entwickelt worden und wird unterstützt in JavaScript,

C#, Visual Basic und C++, als wäre sie Teil der jeweiligen Sprache. Für die Entwicklung von Windows Phone Apps wird die aktuelle Windows Version benötigt. Die Entwicklungsumgebung Visual Studio ist im Internet erhältlich und wird für die App Entwicklung benötigt. Nach der Installation von Visual Studio wird noch ein virtueller Windows Phone Simulator zum Testen benötigt. Dieser ist auch im Internet erhältlich. Um eine Windows Phone App auf einem Windows Phone Gerät testen zu können, muss man das Gerät für die Entwicklung registrieren. Zur Entwicklung von Windows Phone Apps wird eine kostenlose Entwicklerlizenz benötigt. Zum Verkauf von Windows Phone Apps ist ein kostenpflichtiges Entwicklerkonto notwendig.[32]

2.5 Hybride Mobile App

Der größte Vorteil von hybriden mobilen Apps ist, dass durch die Entwicklung einer App alle mobilen Betriebssysteme abgedeckt werden können. Es ist keine parallele Softwareentwicklung für die verschiedenen Betriebssysteme in den verschiedenen Programmiersprachen notwendig. Der Entwicklungsaufwand auf Seiten des Softwareherstellers sinkt daher erheblich. Es können innerhalb eines nativen "App-Rahmens" Web-Inhalte angezeigt werden. Daher kann man die App durch HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) und JavaScript Code entwickeln und diese in eine Native-Browser-App einbetten. Für den Benutzer ist es nicht erkennbar, dass es sich um eine Webanwendung handelt, denn er kann die App wie eine native mobile App öffnen. Durch Frameworks, wie etwa PhoneGap, ist es möglich, auf Hardware und Software Komponenten des mobilen Endgerätes zuzugreifen. Die neue Spezifikation der Hypertext Markup Language in Version 5 ist ein großer Fortschritt in der Entwicklung von hybriden mobilen Apps. Daher werden hybride mobile Apps umgangssprachlich auch HTML5 Apps genannt.[35]

2.6 Native Apps verglichen mit hybriden Apps

Der größte Unterschied zwischen nativen und hybriden Apps ist, wie vorhin beschrieben, dass native Apps speziell für ein Betriebssystem entwickelt werden und hybride Apps für mehrere Systeme. In diesem Kapitel möchten wir jedoch genauer auf die Thematik native App gegen hybride App eingehen, da dieses Thema momentan sehr stark diskutiert wird und auch den späteren Entwicklungsverlauf einer Anwendung maßgeblich entscheidet. Durch die vielen verschiedenen Geräte, die heutzutage im Alltag genutzt werden, existieren verschiedenste Hard- und Software Lösungen im mobilen Bereich. Der Grundgedanke von hybriden Apps ist diese Differenz der unterschiedlichen Geräte in der Entwicklung zu umgehen. Die Vorteile von hybriden Apps sind sofort ersichtlich. Es wird ein geringerer Programmieraufwand, geringerer Wartungsaufwand und gleichzeitig eine größere Abdeckung des mobilen Marktes erwartet. Nun stellt sich die Frage wieso überhaupt noch native Apps programmiert werden? Die Antwort ist, dass die Vorteile von hybriden Apps nur bei gewissen Einsatzbereichen wirklich zutreffen. Im Folgenden werden die jeweiligen Vor- sowie Nachteile und die Einsatzbereiche der beiden Technologien beschrieben.

2.6.1 Native App

Der größte Vorteil von nativen Apps ist die Geschwindigkeit und Leistungsfähigkeit. Außerdem können sämtliche Eigenschaften und Funktionen des Betriebssystems vollständig und meist einfacher wie bei hybriden Apps benutzt werden.

Beispiele für Funktionen des Betriebssystems [18]

- Multi-Touch-Display
  - Multi-Touch-Displays erlauben dem Benutzer mit seinem Gerät über Gesten zu interagieren. So kann er nicht nur tippen, sondern auch mit mehreren Fingern gleichzeitig Gesten ausführen. Ein Beispiel für eine solche Geste ist die Zoom-Funktion, bei der zwei Finger zusammen oder auseinander gezogen werden.
- Grafik API
  - Die Betriebssysteme wie Android stellen APIs für Grafikprogrammierung zur Verfügung. Grafisch anspruchsvolle Programme benötigen die maximale Leistung des Gerätes um eine flüssige Darstellung zu ermöglichen.
- Animationen
  - Animationen laufen auf nativen Apps flüssiger, da besserer Zugang zur Hardware besteht. Animationen spielen in guten Benutzeroberflächen mit einer großen Benutzerfreundlichkeit eine wichtige Rolle. Beispiele für solche Animationen sieht man bei vielen „Wisch“-Gesten (z.B. Navigation auf der linken Seite öffnen und schließen).
- Hardware
  - Mit nativen Applikationen ist es viel einfacher auf Hardware, wie zum Beispiel die Kamera, den Speicher, Kontaktdaten oder das Mikrofon zuzugreifen. Bei hybriden Apps wird eine eigene API benötigt, um auf gewisse Hardware zugreifen zu können.
- Spezifische Oberflächen-Elemente
  - Bei der Programmierung einer nativen App kann der Entwickler auf eine Fülle von Benutzeroberflächen-Elemente zurückgreifen. Diese Elemente sind so gestaltet, wie die Benutzer es gewohnt sind. Wenn sich der Entwickler an die Empfehlungen zur Benutzeroberflächengestaltung von Android, Apple oder Windows hält, fühlt sich der Benutzer gleich beim ersten Start der Applikation wohl, da er mit dem Aufbau der Applikation bereits von anderen Applikationen vertraut ist. Ein Nutzer will sofort beim ersten Start der App wissen, wie sie funktioniert. Oft wollen Nutzer sich nicht in eine neue App einarbeiten und deinstallieren diese deshalb darauf und suchen eine Alternative.

Die sinnvolle Kombination dieser Funktionen resultiert in einer Applikation mit sehr hoher Benutzerfreundlichkeit. Die Nachteile von nativen Applikation sind die höheren Entwicklungskosten. Diese Kosten entstehen dadurch, dass die meisten Apps für mehrere Plattformen zur Verfügung gestellt werden sollen, wobei die Apps für jede Plattform erneut implementiert werden muss.

2.6.2 Hybride App

Der große Vorteil von hybriden Apps ist im Vergleich zu nativen Apps eine geringere Entwicklungszeit, wenn die Applikation für mehrere Betriebssysteme entwickelt werden soll. Dieser Vorteil tritt jedoch nur ein, wenn die Anforderungen der Applikation für hybride Apps geeignet sind. Deshalb werden nun die Vorteile von hybriden Apps aufgelistet und anschließend die Nachteile, woraus sich ein klares Einsatzgebiet für hybride Apps ergibt.

Vorteile von hybriden Apps

- Einfach zu implementieren
  - Bereits HTML, Javascript und CSS Kenntnisse reichen aus um eine Anwendung für alle Geräte und Bildschirmgrößen zu entwickeln.
- Geringere Kosten im Vergleich zu nativen Apps
  - Da die Applikation nur einmal programmiert werden muss, sinken die Kosten im Vergleich zu nativen Apps enorm, da hier die Applikation öfter und für verschiedene Systeme programmiert werden muss. Jedoch ist der Entwicklungsaufwand bei sehr umfangreichen Applikationen für hybride Apps oft höher, als mit nativen Applikationen.
- Einfachere Wartung
  - Wenn Fehler auftauchen müssen diese nur in einer Version ausgebessert werden. Bei nativen Apps müsste man den Fehler im schlimmsten Fall für alle Plattformen ausbessern. Auch bei Erweiterungen muss der zusätzliche Programmcode nur einmal entwickelt werden.

Nachteile von hybriden Apps

- Langsamere Performance
  - Da bei hybriden Apps der Code meistens in einem WebView ausgeführt wird, kann der Code nicht direkt an die CPU geschickt werden, sondern benötigt einige Zwischenschritte um ausgeführt werden zu können. Außerdem fehlen wichtige Funktionen wie Garbage-Collection, Threading und eine geeignete Anbindung an die native Grafik API. [49]
- Benutzeroberfläche
  - Da die Benutzeroberfläche nur einmal entwickelt wird, kann schwer auf die einzelnen Oberflächen-Komponenten der einzelnen Betriebssysteme zurückgegriffen werden. Es ist zwar möglich Android-Komponenten, wie zum Beispiel eine Navigationsleiste, auch hybrid umzusetzen, jedoch werden diese nur emuliert und können nur schwer die Benutzerfreundlichkeit einer nativen App erreichen.
- Hardware Zugriff
  - Der Zugriff auf die Hardware des Gerätes ist zwar möglich, jedoch weit schwieriger umzusetzen wie mit nativen Applikationen.

2.6.3 Fazit

Zusammenfassend ergibt sich ein klares Anwendungsprofil für hybride Apps. Wenn eine App möglichst kostengünstig sein soll, jedoch auf allen Plattformen laufen soll, ist eine hybride App die beste Wahl. Jedoch kann eine hybride App nur sehr schwer die Benutzerfreundlichkeit einer nativen App überbieten.
Die meisten Applikationen besitzen Anwendungsprofile für eine native App. Die Benutzerfreundlichkeit steht im Vordergrund, denn die App soll dem Benutzer helfen und nichts komplizierter machen. Das wird nur mit schnellen Ladezeiten, flüssigen Animationen und gewohntem Aufbau

der Oberfläche erreicht. Es wäre zwar möglich eine hybride App mit diesen Anforderungen zu entwickeln, jedoch wäre der Entwicklungsaufwand weit höher, als würde man gleich nativ entwickeln.
Bei unserer Diplomarbeit steht die Benutzerfreundlichkeit im Vordergrund, denn der Anwender soll mit unserer App Freude haben und gerne damit arbeiten. Er soll seine GastroApp damit einfacher und effektiver verwalten können, als wie mit der Weboberfläche. Außerdem werden Hardware-Zugriffe (Kamera, Speicher) und bestimmte Android-Funktionalitäten (Push-Nachrichten) benötigt. Deshalb wurde die Entwicklung einer nativen App gewählt.

2.7 Fachgebiet Gastronomie

Das Produkt „GastroApp“ befasst sich mit dem Fachgebiet der Gastronomie und deren Vermarktung. Die Gastronomie bietet viele Bereiche, welche für den Kunden digital bereitgestellt werden können, um die bestehende Marketingstruktur zu erweitern und die Reichweite zu erhöhen. Das Marketingkonzept des Gastronoms sollte sich von Mitbewerbern abheben, um eine zusätzliche Nachfrage zu erzeugen und neue Kunden anzuwerben. Mit der GastroApp erhält der Gastronom eine neue Art sein Unternehmen und seine Produkte zu bewerben. Über die GastroApp wird das Leistungsangebot und die Philosophie des Unternehmens für den Kunden dargestellt. Um ein Software-Produkt zu erstellen, welches sowohl dem Kunden als auch dem Gastronom einen Mehrwert bieten soll, müssen einige Anforderungen erfüllt werden. Funktionen die dem Kunden einen Mehrwert bieten sind vor allem eine aktuelle Speise- und Wochenkarte, die der Kunde direkt auf seinem Smartphone jederzeit sehen kann. Im Neuigkeiten-Bereich erfährt er von neuen Gerichten, Veranstaltungen, Turnieren oder Aktionen. In einer Galerie kann er Bilder von den Gerichten und dem Ambiente sehen. Über die Kontaktinformationen gelangt der Kunde an die Anschrift des Betriebes und erhält alle Verknüpfungen zu sozialen Medien. Wenn der Kunde einen Tisch reservieren möchte, kann er direkt über die App eine Anfrage stellen. Falls er diesen Betrieb bereits besucht hat, kann er mit der App eine Kritik versenden. Mit Gutscheinpässen und Aktionsprodukten kann der Gastronom neue Kunden gewinnen und bestehende Kunden wieder ermutigen erneut zu kommen. Durch die Anbindung von Benutzerstatistiken kann der Gastronom verfolgen, wie viele Menschen seine GastroApp benutzen.

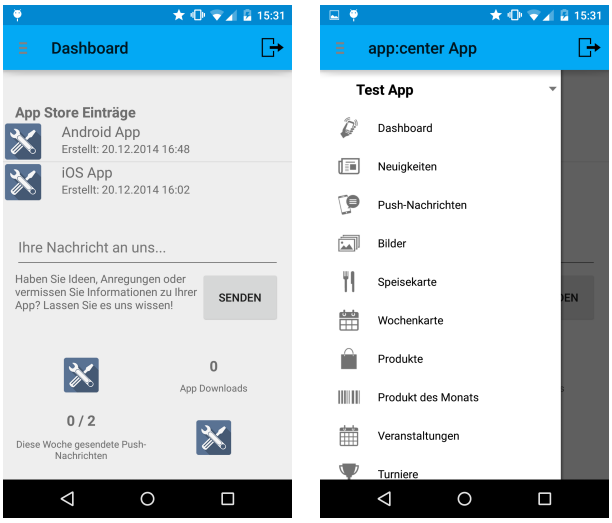
Kapitel 3

Benutzung der Applikation

In diesem Kapitel wird beschrieben, wie die Applikation benutzt werden kann und wie die Module funktionieren.

3.1 Wie benutze ich die Applikation?

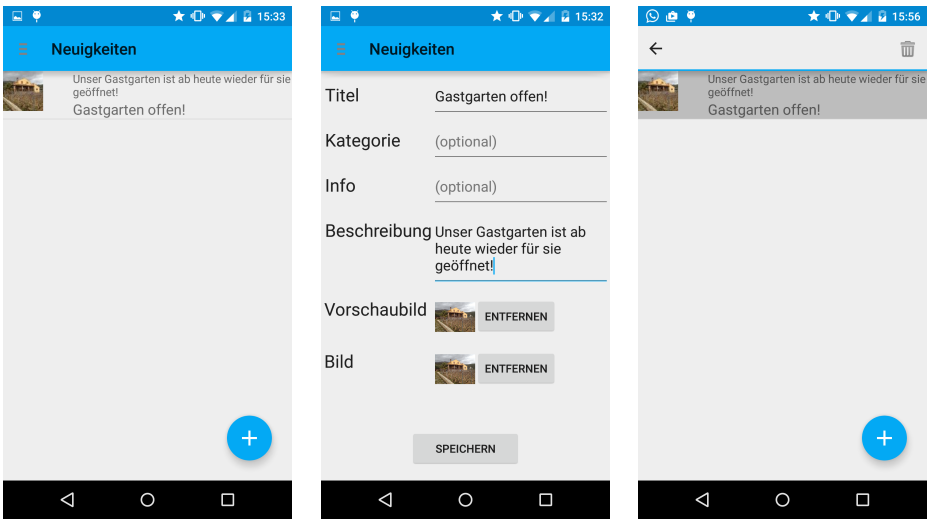
Die Applikation lässt sich wie jede andere Android-Applikation über einen Klick auf das App-Logo im Android Menü starten. Bei dem ersten Fenster muss sich der Benutzer mit seinen GastroApp-Zugangsdaten einloggen. Die Zugangsdaten werden ab diesem Zeitpunkt gespeichert, damit der Kunde bei erneutem Einstieg die Zugangsdaten nicht mehr angeben muss. Über den Ausloggen-Button, welcher sich am Dashboard befindet, werden seine Zugangsdaten wieder gelöscht.
Sobald sich der Benutzer erfolgreich eingeloggt hat wird das Dashboard-Modul geöffnet. Am Dashboard, wie in Abbildung 3.1 (a) gezeigt wird, erhält der Benutzer einen Überblick über die wichtigsten Daten seiner GastroApp.
Die Navigation in der Applikation funktioniert über den so genannten „Navigation Drawer“. Diese Navigationsleiste lässt sich entweder mit einem Klick auf die drei Striche am Bildschirmrand links oben oder mit einer Wisch-Geste vom linken Bildschirmrand nach rechts öffnen.



(a) Dashboard (b) Navigation

Abbildung 3.1: Dashboard und Navigation

Die Navigationsleiste, siehe Abbildung 3.1 (b), zeigt alle verfügbaren Module und mit einem Klick auf den Modulnamen öffnet sich das gewünschte Modul. Alle Module sind nach dem gleichen Schema aufgebaut und lassen sich je nach Funktionalität fast gleich bedienen. Als Beispiel für ein Modul wird das Neuigkeiten-Modul benutzt.



(a) Neuigkeiten (b) Eintrag bearbeiten (c) Eintrag löschen

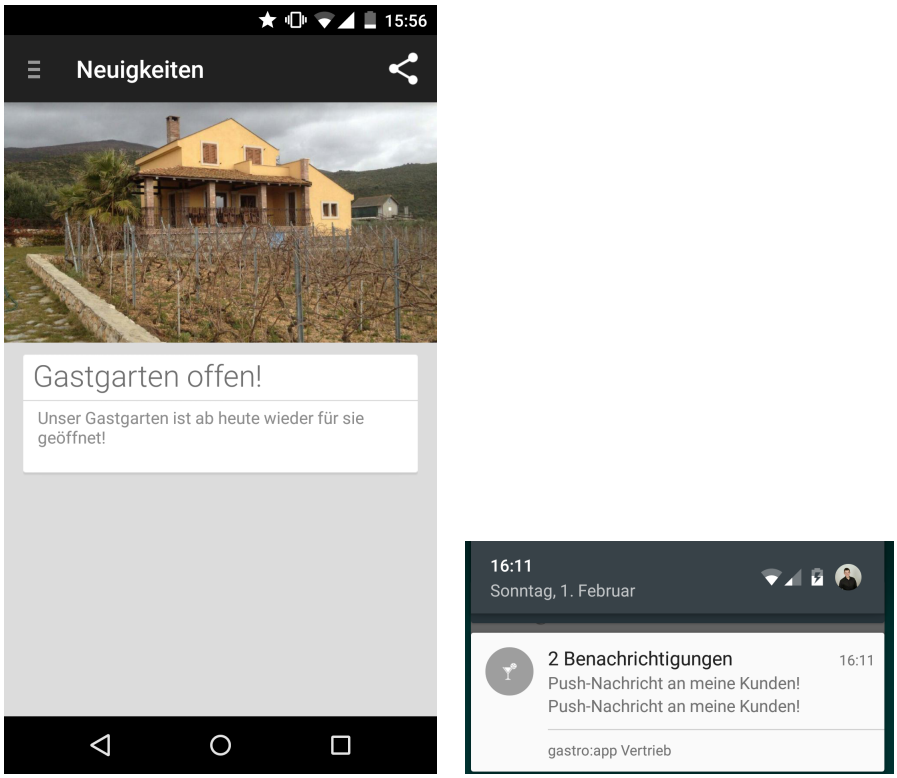
Abbildung 3.2: Möglichkeiten zur Administration von Einträgen

Wie in Abbildung 3.2 (a) zu sehen ist, werden sämtliche Einträge in einer Liste dargestellt. Mit dem Plus-Button am rechten, unteren Bildschirmrand kann ein neuer Eintrag hinzugefügt werden. Mit einem Klick auf einen Eintrag kann dieser Beitrag bearbeitet werden, wie in Abbildung 3.2 (b) dargestellt wird. Mit einem langen, haltenden Klick auf einen Eintrag kann dieser gelöscht werden (siehe Abbildung 3.2 (c)).

Über das Modul Push-Nachrichten kann der Gastronom eine Nachricht direkt an seine Kunden senden. Diese Nachricht erhalten alle Kunden, die die GastroApp des Gastronoms auf ihrem Smartphone installiert haben.

3.2 Ergebnisse

Die Ergebnisse sind sofort in sämtlichen GastroApps der Kunden sichtbar. Wird ein neuer Eintrag hinzugefügt, ein Eintrag bearbeitet oder ein Eintrag gelöscht erhält der Kunde die aktualisierten Daten (siehe Abbildung 3.3 (a)) beim nächsten Start der Applikation. Wird eine Push-Nachricht an die Kunden gesendet, erhält jeder Kunde eine Benachrichtigung, wie in Abbildung 3.3 (b) gezeigt wird, mit dem Inhalt dieser Nachricht.



(a) Neuigkeiten (b) erhaltene Push-Nachricht

Abbildung 3.3: Ergebnisse in der GastroApp

Kapitel 4

Programmstruktur

4.1 Architektur

In diesem Kapitel wird die Systemarchitektur des Produkts „GastroApp“ beschrieben. Das System ist in Form einer Multi-Tier-Architektur aufgebaut. Es besteht aus der Präsentationsschicht, der Logik-Schicht und der Persistenz-Schicht. Die Zusammenhänge der verschiedenen Teilbereiche des Produkts, sowie die unterschiedlichen Schichten, werden in Abbildung 4.1 veranschaulicht.

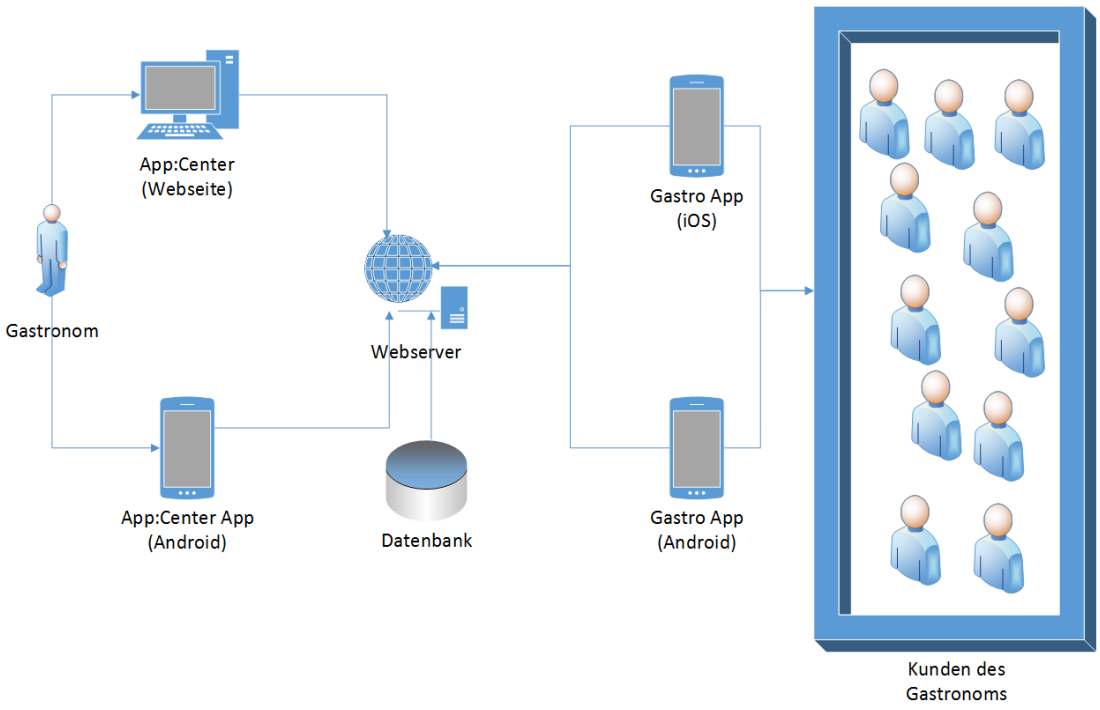


Abbildung 4.1: Systemarchitektur des Produkts „GastroApp“

Wie in Abbildung 4.1 zu erkennen ist, kann der Kunde des Produkts „GastroApp“, also der Gastronom, über das App:Center oder über die App:Center App seine Einträge (Speisekarte, Veranstaltungen, Bilder, etc.) für seinen Gastronomiebetrieb verwalten. Das App:Center ist eine Webseite, mit deren Hilfe die gleichen Einträge bearbeitet, gelöscht und hinzugefügt

werden können, wie mithilfe der App:Center App. Die App:Center App ist eine Android App und das Ergebnis dieser Diplomarbeit. Diese beiden Teilsysteme sind jeweils Bestandteil der Präsentationsschicht. Sie greifen über eine Internetverbindung auf einen RESTful Web Service zu. Dieser Web Service befindet sich auf einem Webserver. Der RESTful Web Service wird im Kapitel 4.1.3 RESTful Web Service näher beschrieben. Dieser Web Service ist die Logik-Schicht des Systems. Die Logik-Schicht greift auf eine Datenbank (die Persistenz-Schicht) zu. Wie das Datenmodell aufgebaut ist und welche Datenbank verwendet wurde, wird im Kapitel 4.1.1 näher erklärt. Die Kunden des Gastronoms können sich über den App Store bzw. über den Google Play Store die GastroApp für iOS bzw. für Android herunterladen. Die GastroApp zeigt die Einträge, die ein Gastronom mithilfe der App:Center App oder des App:Centers mit seinen Kunden teilen möchte. Der Gastronom hat auch die Möglichkeit eine oder mehrere Push Nachrichten an seine Kunden zu versenden. Ein Gastronom kann eine oder mehrere GastroApps besitzen.
Der Austausch der Daten zwischen den Teilsystemen App:Center, App:Center App und den Gastro Apps (jeweils für Android und iOS) erfolgt über eine REST Schnittstelle. Das Datenformat, das zur Übertragung der Daten verwendet wird, ist JSON. Näheres dazu befindet sich im Kapitel 4.1.3.

4.1.1 Datenbank

Die Datenbank stellt die Persistenz-Schicht des Produkts „Gastro App“ dar. Es wird eine MySQL Datenbank verwendet. Dies ist eines der weltweit meist verbreitetsten relationalen Datenbankverwaltungssysteme. MySQL ist als Open-Source-Software verfügbar.[41]

Das Datenmodell, welches in Abbildung 4.2 zu sehen ist, ist ein kleiner Ausschnitt des gesamten Datenmodells. Die grundlegenden Tabellen mit ihren Beziehungen werden jedoch dargestellt.

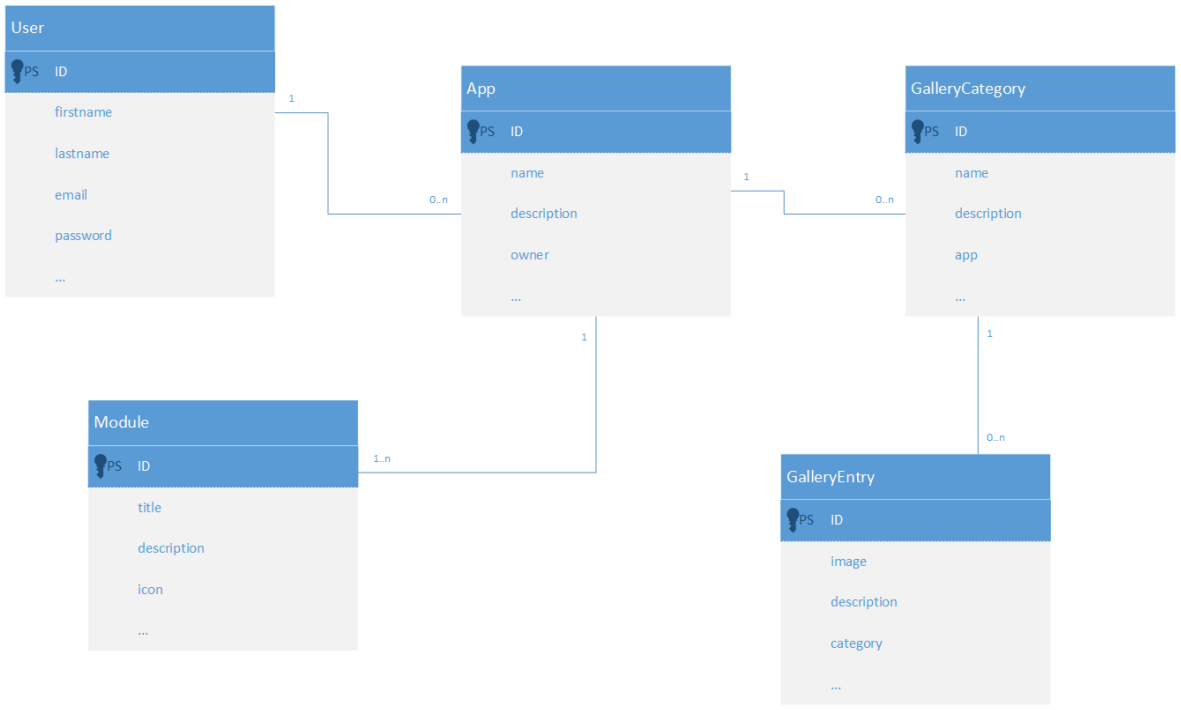


Abbildung 4.2: Datenmodell des Produkts „GastroApp“

Die Tabelle User speichert allgemeine Informationen, wie zum Beispiel Vorname, Nachname, E-Mail Adresse, etc. über den Benutzer der Gastro App (Gastronom). Ein Benutzer kann keine oder mehrere Apps besitzen. Jede App hat eine einzigartige Nummer (ID), einen Namen, eine Beschreibung und einen Benutzer gespeichert. Jede App besitzt ein Modul oder mehrere Module (Dashboard, Neuigkeiten, Turniere, ...). Die Tabelle Module speichert den Titel, die Beschreibung, ein Icon des Moduls und einige andere relevante Informationen. Eine App kann keine oder mehrere Galerie Kategorien besitzen. Eine Galerie Kategorie hat einen Namen, eine Beschreibung und eine App. Außerdem besitzt jede Galerie Kategorie keine oder mehrere Galerie Einträge. Diese Galerie Einträge besitzen ein Bild, eine Beschreibung und eine bestimmte Galerie Kategorie, zur welcher sie gehören. Ähnlich wie die Tabellen GalleryCategory und GalleryEntry mit der Tabelle App in Verbindung stehen, stehen auch die anderen Tabellen, wie zum Beispiel Event, Tournament, NewsEntry, etc. mit der Tabelle App in Verbindung. Um das Datenmodell einfach zu halten, wurden nicht alle Tabellen in der Abbildung 4.2 dargestellt. In den nachfolgenden Kapiteln wird die Präsentationsschicht und die Logik-Schicht mit den Tabellen/Entitäten GalleryCategory und GalleryEntry erklärt.

4.1.2 ORM

Eine der Hauptaufgaben der Logik-Schicht ist es, die relationale Datenbank in einer objektrelationalen Abbildung zur Verfügung zu stellen. Die Daten aus der Datenbank werden mithilfe des Object Relational Mappings (ORM) als Objekte dargestellt. Die Grundlagen des ORMs wurden bereits im Kapitel 2.2.2 erklärt. Das Object Relational Mapping findet beim Produkt „Gastro App“ am Webserver statt und wird vom Framework EclipseLink übernommen.

EclipseLink

EclipseLink ist ein Open-Source Persistenz- und ORM-Framework. Es ermöglicht die Interaktion mit verschiedenen Datensystemen, Datenbanken, Web-Diensten, etc. Es ist die Referenzimplementierung für die Java Persistence API (JPA).[33]
Die Java Persistence API ist eine Schnittstelle für Java-Anwendungen, die die Zuordnung und die Übertragung von Objekten zu Datenbankeinträgen vereinfacht. Sie ermöglicht eine Speicherung von Laufzeit-Objekten, die über eine einzelne Sitzung hinaus verfügbar sein sollen. Bei dieser Speicherung können relationale Datenbanken eingesetzt werden, die grundsätzlich nicht für objektorientierte Datenstrukturen vorgesehen sind.[38]
Neben der API, welche im Package javax.persistence definiert ist, besteht die Java Persistence aus folgenden Komponenten [38]:

- Persistence Entity
  - Eine Persistence Entity ist ein sogenanntes POJO (Plain Old Java Object), das üblicherweise auf eine einzelne Tabelle in der relationalen Datenbank abgebildet wird. Instanzen einer POJO Klasse entsprechen den Zeilen einer Tabelle in der Datenbank. Persistence Entities können je nach Designvorgabe als einfache Datenhaltungsklassen oder als Business-Objekte inklusive Business-Logik realisiert werden. Im Code-Beispiel 4.1 ist das POJO GalleryEntry zu sehen.
- Objektrelationale Metadaten
  - Die Beziehungen zwischen den einzelnen Tabellen der Datenbank werden über objektrelationale Metadaten ausgedrückt. Diese sind entweder als Java-Annotationen

angelegt und/oder in einer separaten XML-Datei abgelegt. Im Code-Beispiel 4.1 sind die Java-Annotationen zu sehen.

- Java Persistence Query Language (JPQL)
  - Dadurch ist es möglich, Abfragen bezüglich der in der Datenbank gespeicherten Entitäten durchzuführen. Diese Abfragen ähneln syntaktisch SQL-Abfragen, beziehen sich aber auf Entitäten statt auf Datenbanktabellen. EclipseLink und andere JPA-Implementierungen überführen die in JPQL formulierten Abfragen zur Laufzeit in ein SQL-Statement, das vom Zieldatenbanksystem verstanden wird.

Listing 4.1: Plain Old Java Object GalleryEntry

```
1  @Entity
2  public class GalleryEntry {
3
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7
8     @OneToOne(cascade = CascadeType.ALL)
9     private Image image;
10
11    private String description;
12
13    private Integer position;
14
15    @ManyToOne
16    private GalleryCategory category;
17
18    ...
19
20 }
```

Im Code-Beispiel 4.1 ist das POJO GalleryEntry zu sehen. Die Annotation @Entity, welche sich über dem Klassennamen befindet, gibt zum Ausdruck, dass diese Klasse von der JPA persistiert werden soll. Die Annotation @Id kennzeichnet den Primärschlüssel und die Annotation @GeneratedValue gibt an, wie der Primärschlüssel generiert werden soll. Durch den Zusatz (strategy=GenerationType.IDENTITY) wird festgelegt, dass während eines commits für jede neue Entität ein neuer Primärschlüssel automatisch erstellt wird. Die Annotation @OneToOne, welche sich auf die Klassenvariable image bezieht, gibt an, dass eine GalleryEntry Entität genau ein Bild besitzt. Die Annotation stellt also die Beziehung zwischen den Tabellen GalleryEntry und Image dar. Der Zusatz (cascade=CascadeType.ALL) bedeutet, dass alle JPA Operationen (Persist, Merge, Remove, Refresh und Detach) sowohl auf die Entität GalleryEntry als auch auf die Entität Image angewendet werden.[52]
Wenn zum Beispiel ein GalleryEntry gelöscht wird, wird auch das dazugehörende Image aus der Datenbank gelöscht. Die Annotation @ManyToOne, welche sich auf das Objekt category des Typs GalleryCategory bezieht, gibt an, dass eine Galerie-Kategorie (GalleryCategory) mehrere Galerie-Einträge (GalleryEntry) haben kann. Es stellt also wieder die Beziehung zwischen den Datenbanktabellen dar. Das POJO GalleryEntry besitzt Zugriffsmethoden (getter- und setter-Methoden) für alle Klassenvariablen.

Die gesamten POJOs (GalleryEntry, NewsEntry, Event, etc.) sind das Ergebnis des ORM. Die Datenbanktabellen können nach dem ORM als Objekte in der Java-Anwendung verwendet werden und mithilfe der RESTful Web Services der Präsentationsschicht zur Verfügung gestellt werden.

4.1.3 RESTful Web Service

Die RESTful Web Services, die vom Webserver zur Verfügung gestellt werden, gehören neben dem ORM zur Logik-Schicht der Multi-Tier-Architektur. Die Logik-Schicht verbindet die Persistenz-Schicht (Datenbank) mit der Präsentationsschicht (App:Center, App:Center App und Gastro Apps). Dazu werden RESTful Web Services eingesetzt. Diese Webservices stellen die POJOs für die Präsentationsschicht bereit. Die Daten werden mit dem Datenformat JSON über eine REST (Representational State Transfer) Schnittstelle übertragen. Die Grundlagen von Webservices, REST und JSON wurden bereits im Kapitel 2.2.2 erklärt.

In Java kann für die Verwendung von Webservices, welche die REST Technologie unterstützen, die JAX-RS (Java API for RESTful Web Services) herangezogen werden. Die Referenzimplementierung von JAX-RS stellt das Open-Source-Projekt Jersey dar.[37]
Für das Produkt „GastroApp“ wird das Jersey Framework eingesetzt. Im Code-Beispiel 4.2 ist eine einfache Verwendung des Jersey Frameworks zu sehen. Das Framework arbeitet mit Java-Annotationen.

Listing 4.2: Einfache Verwendung des Jersey Frameworks[19]

```
1 @Path("myresource")
2 public class MyResource {
3
4    @GET
5    @Produces(MediaType.TEXT_PLAIN)
6    public String getIt() {
7       return "Test!";
8    }
9 }
```

Wie im Code-Beispiel 4.2 zu sehen ist, befindet sich über dem Klassennamen die Annotation @Path. Diese Annotation stellt die URI dar, über welche die Ressource erreichbar ist. Die Ressource MyResource ist über die URI „/myresource“ erreichbar. Die Annotation @GET sagt aus, dass bei einem GET-Aufruf der URI „/myresource“ die Methode getIt aufgerufen wird. Bei einem GET-Aufruf wird der Text „Test!“ als Content-Type „text/plain“ übertragen.[19]

Durch den Content-Type, auch MIME-Type (Multipurpose Internet Mail Extensions) genannt, wird dem Programm, welches den GET-Aufruf ausführt, mitgeteilt, welche Daten der Webserver versendet. Es kann sich zum Beispiel um Klartext, ein HTML-Dokument oder ein PNG-Bild handeln. Das Programm, welches die Daten empfängt, kann dadurch je nach MIME-Type unterschiedlich reagieren.[36]
In diesem Fall wird der Text „Test!“ als Klartext übertragen.

Im Produkt „GastroApp“ sind die Ressourcen ähnlich aufgebaut. Für jede Ressource können die Aktionen POST, GET und DELETE angewendet werden. Um alle Galerien (alle Gallery-Categories) zu erhalten, muss ein GET-Aufruf ausgeführt werden.

Dieser Aufruf kann wie folgt aussehen:

```
GET http://center.gastroapp.at/api/internal/gallerycategories?app=2
```

Durch diesen Aufruf wird eine GET-Anfrage an den Webserver, auf welchem sich die Webservices befinden, geschickt. Die Klasse, welche alle GalleryCategories verwaltet, muss die Annotation @Path(”gallerycategories”) besitzen. Zusätzlich muss eine Methode mit der Annotation @GET vorhanden sein. Dies ist die Methode getAllGalleryCategories(), welche alle Galerie-Kategorien im JSON-Format zurückgibt. Zusätzlich muss noch angegeben werden, aus welcher App des Gastronoms diese Einträge abgerufen werden sollen. In diesem Fall handelt es sich um die App mit der Nummer 2.

Die Antwort des Webservices kann wie folgt aussehen:

Listing 4.3: Antwort des Webservice

```
[
   {
      "id":32,
      "name":"Siegerehrung Fotos"
   },
   {
      "id":35,
      "name":"Unser Restaurant"
   },
   {
      ...
   },
   ...
]
```

Möchte man nun eine Galerie Kategorie löschen oder aktualisieren, muss ein DELETE bzw. POST Aufruf ausgeführt werden. Der Aufruf zum Löschen kann wie folgt aussehen:

```
DELETE http://center.gastroapp.at/api/internal/gallerycategories/35
```

In diesem Fall wird die Galerie Kategorie mit der Nummer 35 gelöscht. Die Nummer der App ist hier nicht mehr notwendig, da durch die Nummer der Galerie Kategorie eindeutig definiert ist, welche Galerie Kategorie gelöscht werden soll. Bei diesem Aufruf wird die Methode deleteGalleryCategory() aufgerufen, welche die Annotation @DELETE besitzt. In dieser Methode wird die Galerie Kategorie gelöscht.

Die Mobilen Apps und die Webseite, welche die Präsentationsschicht des Produkts „Gastro-App“ darstellen, erhalten über Aufrufe der verschiedenen Webservices die Daten, welche zur Anzeige notwendig sind. Dadurch ist das Löschen und Bearbeiten von Einträgen möglich und neue Einträge können hinzugefügt werden.

4.2 app:center App

In diesem Kapitel werden die Software-Architektur, die Frameworks, die Schnittstellen und das Oberflächen Design der Android App beschrieben.

4.2.1 Software-Architektur

Grundlegende Architektur

In diesem Kapitel werden die grundlegenden Klassen, die in Abbildung 4.3 gezeigt werden, beschrieben. Diese Klassen werden zum Starten, zum Navigieren und zum Bedienen der App benötigt.

Beim Start der Android App wird die Activity MainActivity.java gestartet. Die Layout-Datei zur MainActivity befindet sich im Ordner res/layout unter dem Namen activity_main.xml. Die MainActivity stellt den Login-Bereich für den Kunden zur Verfügung. Hier muss sich der Kunde mit seinem Benutzernamen und seinem Passwort anmelden, um auf die App Zugriff zu haben. Die MainActivity übernimmt den Anmeldeprozess, indem sie die Anmeldedaten des Benutzers über eine REST-Schnittstelle mit dem Server abgleicht. Der Zugriff zu den REST-Schnittstellen wird in der Klasse RestClient.java verwaltet. Diese Klasse stellt ein Bean zur Verfügung, welches die Anmeldedaten des Benutzers und die Schnittstellen speichert. Diese Klasse wurde als Singleton-Bean deklariert, da so die Instanz des Beans nur einmal initialisiert werden muss, um dann während der Laufzeit über die komplette Applikation verfügbar zu sein [53]. Der Anmeldeprozess wird im Kapitel Implementierung noch genauer beschrieben. War die Anmeldung des Benutzers erfolgreich, so öffnet sich die NavigationActivity und der Benutzer erhält Zugriff auf die Applikation. Die NavigationActivity enthält die Navigationsleiste (NavigationDrawer), mit der zu den einzelnen Modulen navigiert werden kann. Außerdem kann mit der Navigationsleiste die gastro:app gewechselt werden, falls der Kunde mehrere Apps besitzt. Die NavigationActivity enthält eine Liste von gastro:apps, die aktuell gewählte App und die Liste der aktuell verfügbaren Module.

Die Daten zur GastroApp und zu den Modulen werden durch sogenannte POJOs (Plain Old Java Object) dargestellt. Diese POJOs stellen die Klassen, die vom REST-Service zur Verfügung gestellt werden, dar. So werden sämtliche Einträge, wie Neuigkeiten, Speisekarten oder Turniere abgebildet. Der REST-Client übernimmt dann die Konvertierung von JSON-Objekten, die vom Server gesendet werden, zu POJOs.

Modularer Aufbau

Ein wichtiger Teil der app:center App ist die Modularität. Das Produkt gastro:app ist, wie bereits beschrieben, mit mehreren Modulen aufgebaut. Der Kunde kann also verschiedene Module in seiner gastro:app besitzen und deshalb ist das Menü der app:center App für jede gastro:app unterschiedlich aufgebaut. Diese Modularität wird dadurch gewährt, dass das Menü (NavigationDrawer) erst beim Start der App aufgebaut wird. Sämtliche Module sind bereits implementiert und über die Abfrage am Server, welche Module der Benutzer besitzt, wird dass Menü zusammengebaut. Ein weiterer Punkt ist die Möglichkeit, auch mehrere Apps zu besitzen. Beim Start wird geprüft, wie viele Apps der Kunde besitzt und falls er mehr als eine App besitzt, wird die Navigationsleiste für den Gebrauch von mehreren Apps angepasst. In Abbildung 4.3 ist zu sehen, dass die NavigationActivity eine Referenz auf den REST-Client besitzt und über diesen dynamisch alle gastro:apps und die zugehörigen Module vom Server lädt. Erst sobald diese Informationen vom Server erfolgreich angefordert wurden, wird die Navigationsleiste dynamisch

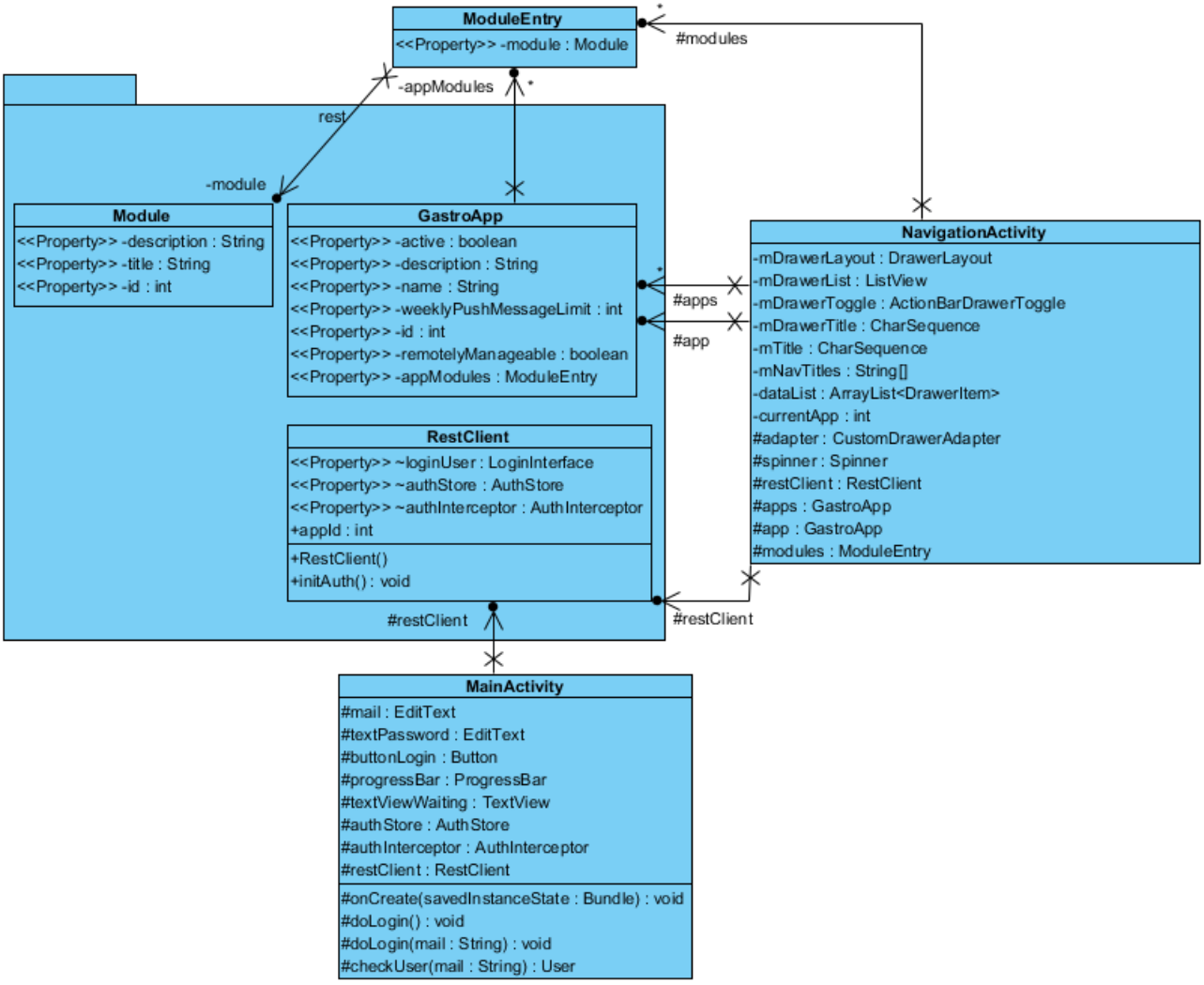


Abbildung 4.3: Grundlegende Architektur der Android App

erstellt. Die konkrete Umsetzung des NavigationDrawer wird im Kapitel Implementierung beschrieben.

4.2.2 Module

Sämtliche Module besitzen ein oder mehrere POJOs. So besitzt das Modul Neuigkeiten das POJO News, welches einen Neuigkeiten-Eintrag darstellt. Außerdem besitzt das POJO News eine Referenz auf das POJO Image, welches dazu dient, Fotos zu speichern (siehe Abbildung 4.4). Da die meisten POJOs sehr ähnlich aufgebaut sind, wird im Folgenden nur das News und Image POJO gezeigt. Die POJOs enthalten eine Reihe von Attributen, wobei manche Pflichtfelder sind. Ist ein Attribut ein Pflichtfeld, so muss dieses beim Erstellen oder Bearbeiten eines Eintrags angegeben sein. Ist ein Pflichtfeld nicht ausgefüllt, so wird dem Benutzer eine Fehlermeldung angezeigt. Beim POJO News sind, wie in Abbildung 4.4 dargestellt, neben der bereits erwähnten Referenz auf das Image folgende Attribute gegeben:

- created vom Typ Kalendar: Dieser Wert wird automatisch bei der Erzeugung eines Neuigkeiten-Eintrags vom Server generiert und stellt das Erstelldatum des Eintrags dar.

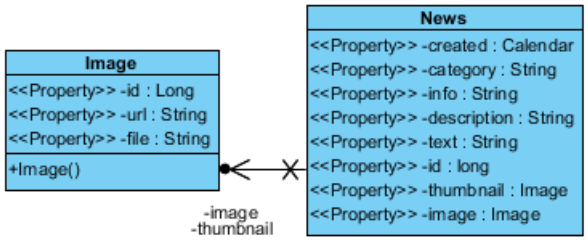


Abbildung 4.4: News und Image POJO

- category vom Typ String: optionale Kategorie, die bei jedem Neuigkeiten-Eintrag angegeben werden kann
- info vom Typ String: kurzer, optionaler Info-Text
- description vom Typ String: verpflichtende Beschreibung des Neuigkeiten-Eintrags
- text vom Typ String: verpflichtender Titel des Neuigkeiten-Eintrags
- id vom Typ long: die ID des Neuigkeiten-Eintrags, welche beim Bearbeiten oder Löschen eines Eintrags benötigt wird. Diese ID wird vom Server bei der Erstellung eines News-Eintrag generiert.
- thumbnail vom Typ Image: Referenz auf ein Image, welches das Vorschaubild des Neuigkeiten-Eintrags festlegt. Das Vorschaubild wird vom Server komprimiert und wird bei der Gastro-App nur klein neben dem Titel angezeigt. Dieses Bild muss verpflichtend gewählt werden.
- image vom Typ Image: Hauptbild des Neuigkeiten-Eintrags, welches auch verpflichtend gewählt werden muss. Dieses Bild erscheint bei der GastroApp mit einem Klick auf den Neuigkeiten-Eintrag.

Das POJO Image enthält wie das POJO News auch eine ID, welche zum Bearbeiten und Löschen benötigt wird, eine URL vom Typ String, welche den kompletten Pfad des Bildes am Server speichert und das Attribut File, welches das Bild als BASE64 kodierten String speichert.

4.2.3 Verwaltungsoberflächen

Der wichtigste Teil der app:center App sind die Verwaltungsoberflächen zu den einzelnen Modulen. Jedes Modul besitzt eine Verwaltungsoberfläche, welche die Anzeige und das Bearbeiten von Einträgen ermöglicht. Die Verwaltungsoberflächen werden in Android als Fragments realisiert und werden durch einen Klick auf den Eintrag in der Navigationsleiste geöffnet. Der Layout-Teil des Fragments wird im Ordner res/layout unter dem Namen fragment_fragmentname.xml gespeichert.

Der Aufbau der Verwaltungsoberflächen gliedert sich meistens in ein Fragment und in ein Detail-Fragment. Viele Module sind ähnlich aufgebaut und deshalb wird das News- sowie NewsDetail-Fragment (siehe Abbildung 4.6) genauer erklärt. Im Falle des NewsFragment zeigt dies eine Liste aller News-Einträge. Diese Einträge werden durch eine Liste von News POJOs dargestellt. Im NewsFragment werden diese in einer ListView (Android Oberflächen Komponente für Listen) dargestellt. Um diesen Einträgen in der Liste ein ansprechendes Aussehen zu verleihen, werden diese Einträge mit Adapter überarbeitet. So kann ein Eintrag der Liste ein Foto (im Falle eines

Neuigkeiten-Eintrags das Thumbnail-Bild), den Titel und eine kurze Beschreibung enthalten (siehe Abbildung 4.5).

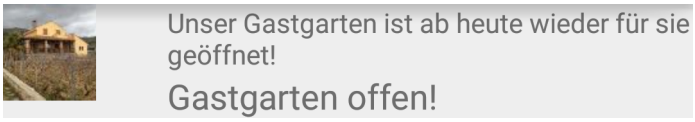


Abbildung 4.5: Anzeige zu Neuigkeiten-Einträgen mit Adapter

Neben der Anzeige aller Neuigkeiten-Einträge dient das NewsFragment auch zum Löschen der Einträge. Bei einem langen Klick auf einen Einträg öffnet sich eine Option um diesen Eintrag zu löschen. Bei einem normalen Klick auf einen Eintrag wird dieser im NewsDetailFragment geöffnet, wo der Eintrag und seine Attribute bearbeitet werden können. Außerdem dient das NewsDetailFragment auch zum Anlegen von neuen Einträgen. Beim Bearbeiten wird die ID des zu bearbeitenden News POJO übergeben. Die Oberfläche ist beim Hinzufügen eines neuen Eintrags und beim Bearbeiten eines bereits vorhandenen Eintrags fast identisch. Beim Bearbeiten werden jedoch die Textfelder mit den aktuellen Werten des News POJO befüllt.

Das NewsFragment besitzt wie jedes andere Fragment eine Layout-Datei (siehe Code-Ausschnitt 4.4). Diese Layout-Datei besitzt den Namen fragment_news.xml und ist im Ordner layout/res zu finden. Dieser Ausschnitt der Layout-Datei zeigt den Aufbau der Oberfläche. Die Hauptkomponente ist die ListView, welche die Einträge anzeigt. Außerdem befindet sich ein FloatingActionButton, welcher das Ereignis zum Öffnen des NewsDetailFragment für das Erstellen eines neuen Eintrags auslöst, eine ProgressBar, welche den Status beim Laden der Einträge anzeigt, und eine TextView, welche beim fehlerhaften Laden von Einträgen eine Fehlermeldung anzeigt, im NewsFragment.

Listing 4.4: Layout-Datei von NewsFragment

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:fab="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listViewNews"/>

    <com.melnykov.fab.FloatingActionButton
        android:id="@+id/fabNewsAddEntry"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
     <ProgressBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/news_progress_bar"
        android:visibility="invisible"
     />

```

```
23      <TextView
24          android:layout_width="wrap_content"
25          android:layout_height="wrap_content"
26          android:textAppearance="?android:attr/textAppearanceLarge"
27          android:text="@string/no_entries_found_error_message"
28          android:id="@+id/news_error_textview"
29       />
30  </RelativeLayout>
```

Die Attribute des NewsFragments, wie in Abbildung 4.6 dargestellt, bestehen aus den Referenzen zu den Layout-Komponenten (newsProgressBar, newsErrorTextView, fab, listViewNews, ...), aus einer Referenz zu dem Rest-Client Bean, einer Liste von News-POJOs und aus einer Referenz zum ActionMode. Der ActionMode ist eine Android-Komponente, welche Funktionalität auf der ActionBar zur Verfügung stellt. Mit dem ActionMode wird der Button zum Löschen und auch zum Bearbeiten angezeigt. Die Implementierung und genauere Funktionsweise wird noch im Kapitel Implementierung gezeigt.

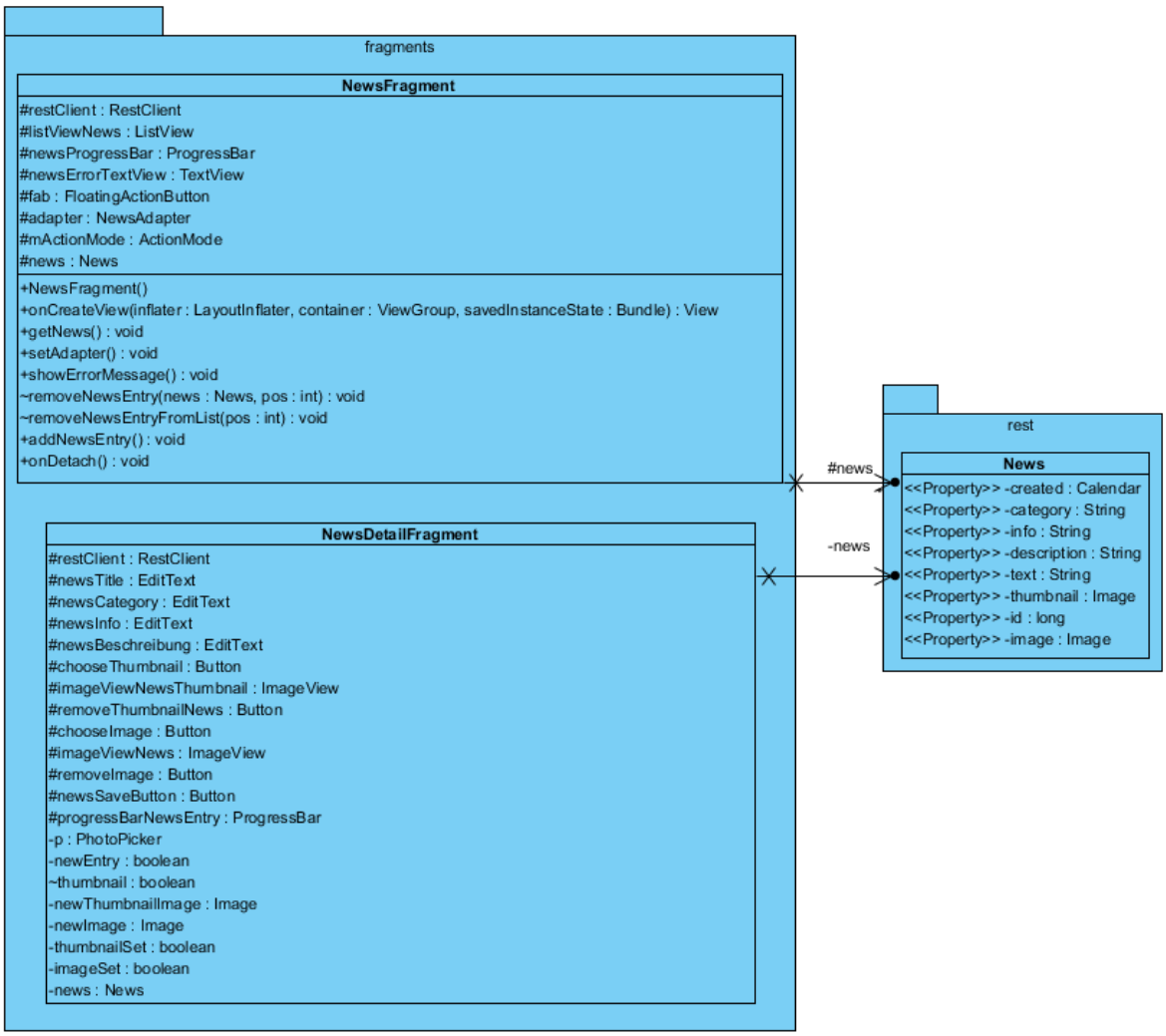


Abbildung 4.6: NewsFragment und NewsDetailFragment

Die Klasse NewsFragment enthält drei wichtige Android-Methoden, nämlich den standardmäßig leeren Konstruktor für Fragments, die Methode onCreateView, welche die Oberfläche mit dem Layout initialisiert und die Methode onDetach, welche für den Lebenszyklus von Fragments benötigt wird. Die Methode onDetach wird aufgerufen, wenn das Fragment geschlossen wird [14].

Neben den Android-Methoden kommen unsere Methoden getNews, setAdapter, showErrorMessage, removeNewsEntry und addNewsEntry vor. Diese Methoden dienen zum Laden sämtlicher Einträge, zum Setzen des Adapters für die ListView, zur Anzeige von Fehlermeldungen und zum Hinzufügen sowie Löschen von Einträgen.

Das NewsDetailFragment enthält Oberflächenkomponenten wie Textfelder zur Bearbeitung von sämtlichen Attributen zu einem News POJO. In einem Detail-Fragment wird immer nur ein Eintrag bearbeitet oder hinzugefügt. Außerdem enthält das DetailFragment die Layout-Komponenten zum Hinzufügen oder Ändern eines Fotos, welches später im Kapitel Implementierung noch genauer erklärt wird.

4.2.4 Frameworks

Wir benutzten für unsere Android App einige Frameworks, um den Entwicklungsprozess zu vereinfachen. Das wichtigste Framework war AndroidAnnotations.

AndroidAnnotations

AndroidAnnotations ist ein Open Source Framework, welches die Entwicklung von Android Apps durch den Einsatz von sogenannten Annotationen stark vereinfacht [5]. Android Annotationen vereinfacht den Programmcode, indem überflüssiger Android Code durch Annotationen ersetzt wird. Annotationen sind eine Form von Metadaten, die Daten über das Programm zur Verfügung stellen. Mit Annotationen kann man Informationen bereitstellen, die zum Beispiel von einem Framework wie AndroidAnnotations benötigt werden um Code zu generieren.[22]

Ein Beispiel für überflüssigen Android-Code sind zum Beispiel Aufrufe um auf Views zuzugreifen. Außerdem liefert AndroidAnnotations folgende Funktionen [5]:

- Threads: Methoden können mit @Background oder @UiThread annotiert werden, um diese Methode entweder als Hintergrundprozess oder im regulären Thread laufen zu lassen.
- Events: Annotierte Methoden können als Event-Listener arbeiten. So sind keine anonymen Event-Listener Klassen mehr nötig. Ein Beispiel für eine solche Event-Listener Annotation ist zum Beispiel @Click, welche bei einem Klick auf einen Button ausgelöst wird.
- REST Client: Einer der größten Vorteile von AndroidAnnotations ist die vorhandene Implementierung eines REST Clients.

Durch den Einsatz von Android Annotations wird die Implementierung und Wartung von Android Code immens erleichtert. Folgendes Code-Beispiel zeigt die Verwendung von Android Annotations im Vergleich zu normalen Android Code. Auf der linken Seite befindet sich normaler Android-Code und auf der rechten Seite derselbe Code mit Verwendung von AndroidAnnotations:

Listing 4.5: Ohne AndroidAnnotations

```
1  public class MainActivity extends
       Activity{
2    ListView list;
3
4    EditText text;
5
6    Button button;
7
8    @Override
9   protected void onCreate(Bundle
       savedInstanceState) {
10    super.onCreate(savedInstanceState);
11      setContentView(R.layout.main);
12      list = (ListView)
            findViewById(R.id.list_view);
13      text = (EditText)
            findViewById(R.id.text_view);
14      button = (Button)
            findViewById(R.id.button);
15
16      button.setOnClickListener(new
            OnClickListener() {
17      @Override
18      public void onClick(View v) {
19        update();
20      }
21    });
22   }
23
24   private void update(){
25      text.setText("Updated!");
26   }
27 }
```

Listing 4.6: Mit AndroidAnnotations

```
1  @EActivity(R.layout.main)
2  public class MainActivity extends
       Activity{
3    @ViewById(R.id.list_view)
4    ListView list;
5
6    @ViewById(R.id.text_view)
7    EditText text;
8
9    @ViewById(R.id.button)
10   Button button;
11
12   @Click(R.id.button)
13   protected void update(){
14      text.setText("Updated");
15   }
16 }
```

AndroidAnnotations übernimmt in diesem Beispiel sämtliche Layout Initialisierungen und Zuweisungen. Außerdem stellt AndroidAnnotations den Event-Listener für einen Button zur Verfügung. Die Verwendung des REST-Clients und anderer AndroidAnnotations Funktionalität wird im Kapitel Implementierung noch genauer gezeigt.

Picasso

Picasso ist ein Framework, welches Bilder komfortabel herunterlädt und in einer ImageView darstellt. Der Vorteil von Picasso ist der komfortable Aufruf, der meistens nur eine Zeile beträgt. Außerdem kann Picasso komplexe Bild-Transformationen und Caching übernehmen. Picasso ist Open Source und frei verfügbar. [27]

Bei der app:center App wird Picasso bei sämtlichen Bildern, die vom Server geladen werden, benutzt. So ein Aufruf sieht unter anderem wie folgt aus:

Listing 4.7: Verwendung von Picasso

```
1 Picasso.with(context).load(item.getThumbnail().getUrl())
2 .placeholder(R.drawable.loading_placeholder_small)
3 .error(R.drawable.error_placeholder_small)
4 .into(holder.imageView);
```

Dieser Aufruf lädt ein Bild eines Neuigkeiten Eintrags mit der URL vom Server herunter und zeigt dieses in einer ImageView an. Außerdem wird während der Ladezeit ein Ladebild angezeigt und falls das Bild nicht heruntergeladen werden konnte, wird ein Fehlerbild angezeigt.

AppCompat v7

Die AppCompat v7 Bibliothek ermöglicht den Einsatz der ActionBar, welche erst mit Android Version 3.0 veröffentlicht wurde, bereits ab Android Version 2.1 [30]. Die ActionBar dient für die Navigation, also zum Öffnen der Navigationsleiste, zeigt den Namen des aktuellen Fragments an und ist die Grundlage für den ActionMode.

FloatingActionButton

Unser Oberflächen-Design beruht auf den aktuellen Design-Richtlinien von Google. Auf das Oberflächen Design wird später noch genauer eingegangen. Die neue Design-Richtlinie von Google heißt Material Design. Diese Design Richtlinie verwendet auch den Floating Action Button. Dieser Button soll unter anderem für das Hinzufügen von neuen Einträgen in eine Liste benutzt werden. Für die einfache Implementation und für Animationen benutzten wir ein Framework (https://github.com/makovkastar/FloatingActionButton). Wir verwendeten den Floating Action Button bei sämtlichen Modulen, die es ermöglichen einen neuen Eintrag anzulegen (siehe Abbildung 4.7).

4.2.5 Schnittstellen

Für die Anforderung der Daten vom Server wird eine REST-Schnittstelle benötigt. Diese REST-Schnittstelle wird mit Hilfe von AndroidAnnotations realisiert. Die Klassen, die für die Umsetzung benötigt werden, werden in Abbildung 4.8 gezeigt. Die Klasse RestClient wird, wie bereits beschrieben, als Singleton Bean für die Anwendung zur Verfügung gestellt. Dieses Bean verwaltet die Anmeldedaten des Benutzers in der Klasse AuthStore. Die Klasse AuthInterceptor implementiert die Klasse ClientHttpRequestInterceptor, welche vom Spring Framework zur Verfügung gestellt wird. Spring liefert die REST Funktionalitäten für AndroidAnnotations, wie zum Beispiel die Klasse RestTemplate, welche sämtliche REST-Methoden (GET, PUT, POST, DELETE) implementiert [9]. Der ClientHttpRequestInterceptor wird benötigt, um HTTP-Anfragen vom Client abzufangen und um diese Anfrage um zusätzliche Header und Body-Daten zu erweitern [17]. Unser AuthInterceptor fügt einen Autorisierungs-Header hinzu, welcher die Anmeldedaten und den Typ der Autorisierung enthält [45]. Dieser Header wird vom Server benötigt, um Zugriff auf die Daten zu erhalten. Der Typ der Authentifizierung ist HTTP-Basic. Das Basic-Authentication Schema schreibt dem Client vor, sich für jeden Bereich mit einem Benutzernamen und einem Passwort zu authentifizieren. Die Anmeldedaten werden durch einen Doppelpunkt getrennt (Benutzername:Passwort) und sind BASE64 kodiert [46]. Das Passwort

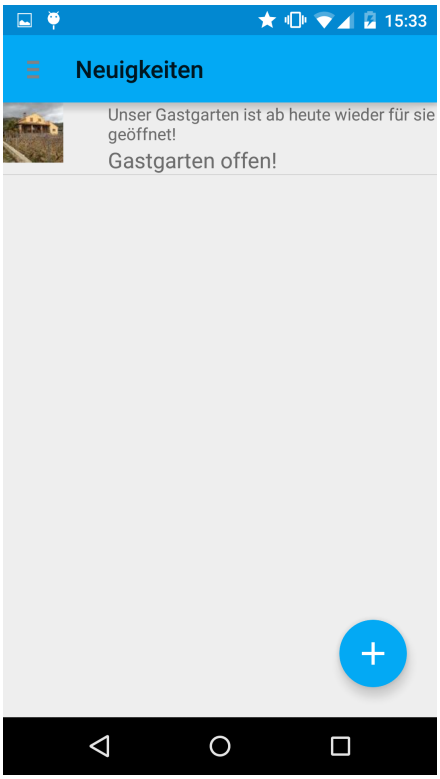


Abbildung 4.7: Floating Action Button

wird zudem vorher mit SHA-256 gehasht, um zusätzliche Sicherheit zu gewährleisten. Der AuthStore und der AuthInterceptor werden dem REST-Client hinzugefügt. In der Methode initAuth der Klasse RestClient wird dann das RestTemplate von der Klasse LoginInterface erstellt, mit dem AuthInterceptor der Autorisierungs-Header hinzugefügt und anschließend dem RestTemplate übergeben. Nun kann über die Klasse LoginInterface auf sämtliche REST-Ressourcen des Servers zugegriffen werden. Die Klasse LoginInterface wurde in Abbildung 4.8 aus Platzgründen verkürzt dargestellt. Mit der REST-Ressource User wird als erstes überprüft, ob der Anmeldevorgang erfolgreich war. Gibt der REST-Client erfolgreich die Daten des angemeldeten Benutzers zurück, so war die Anmeldung erfolgreich. Gibt der REST-Client allerdings eine Fehlermeldung vom Typ 401 Unauthorized zurück, so war die Anmeldung fehlerhaft und die Anmeldedaten des Benutzers falsch. Wenn die Anmeldung erfolgreich war, werden die Anmeldedaten gespeichert, denn diese werden bei jedem Aufruf zu einer REST-Ressource benötigt. Diese Lösung wurde vom Auftraggeber gewählt. Eine andere Möglichkeit für den Zugriff zu REST-Ressouren wäre ein Authentifizierungs-Token der bei jeder Anfrage mitgesendet wird. Die Anmeldedaten werden im AuthStore Bean gespeichert.

Wird ein Eintrag bearbeitet, so wird auf diese Ressource mit dem REST-Client über das Interface LoginInterface zugegriffen. Dieses Interface speichert die URL des RESTful Webservice und sämtliche Methoden zum Anzeigen, Hinzufügen, Bearbeiten oder Löschen der REST-Ressourcen. So eine Methode wird mit einer Annotation angelegt, welche bestimmt ob die Methode eine GET, PUT, POST oder DELETE Anfrage an den Server sendet. Außerdem wird in dieser Annotation, wie im Code-Beispiel 4.8 gezeigt wird, der genaue URI der REST-Ressource angegeben. Die Methode selbst definiert den Rückgabewert, der im Beispiel eine Liste von GastroApp POJOs ist.

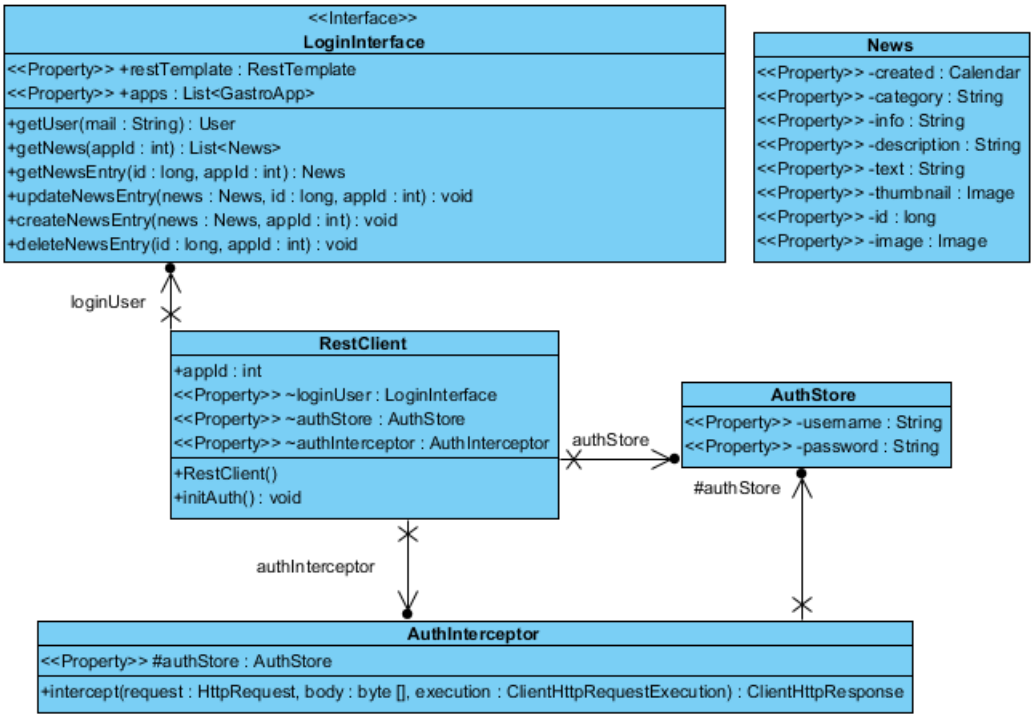


Abbildung 4.8: REST-Schnittstelle

Außerdem können auch Parameter, die der Methode übergeben werden, definiert werden. Diese werden für das Bearbeiten, Hinzufügen oder Löschen einer Ressource benötigt. Um auf diese Methoden Zugriff zu haben muss das RestClient Bean verwendet werden, welches eine Referenz auf das LoginInterface speichert.

Listing 4.8: GET Aufruf

```
1 @Get("api/internal/apps")
2 List<GastroApp> getApps();
```

4.2.6 Oberflächen Design

Der Benutzer muss die Applikation auf einem relativ kleinen Bildschirm problemlos steuern können. Deshalb sollte die Oberfläche der Applikation möglichst intuitiv aufgebaut sein. Für die beste Benutzererfahrung liefert Android Design-Grundlagen. Diese Grundlagen sollten befolgt werden, um den aktuellen Anforderungen einer benutzerfreundlichen Android App zu entsprechen.

Material Design

Mit der Veröffentlichung von Android 5.0 (API Stufe 21) stellte Android die Design-Grundlagen zum sogenannten Material Design vor. Material Design ist eine Spezifikation, um Innovation und Technik mit einem guten Design zu verbinden [48]. Das Material Design kann für sämtliche Bildschirmgrößen und Eingabemethoden umgesetzt werden. So ist zum Beispiele auch die neue

docs.google.com Oberfläche im Material Design. Auch sämtliche Symbole (engl. Icons) wurden überarbeitet und sind für das Material Design erhältlich.

Durch gezielt eingesetzte Farben, Oberflächen und Symbole soll der Benutzer sofort wissen, mit welchen Komponenten er interagieren kann und welche Funktionalität diese verbergen. Material Design nutzt die Möglichkeiten der Technik durch Animationen, Schatten und vielen anderen Design Komponenten um dem Benutzer eine einfachere und übersichtlichere Bedienung seiner Applikation zu ermöglichen.

Für das Material Design sollten Farbpaletten angelegt werden. Diese Farbpaletten bestehen aus einer dunklen und hellen Hauptfarbe, einer Akzentfarbe, einer Textfarbe und aus einer Farbe für Icons. Diese Farben sollten gezielt verwendet werden, um gewisse Komponenten für den Benutzer hervorzuheben und ansprechender zu gestalten. In Android können diese Farben in der colors.xml Datei verwaltet werden.

Komponenten wie die ActionBar spielen beim Material Design eine große Rolle. Die wichtigsten Informationen befinden sich in der ActionBar und auch die Navigationsleiste lässt sich über die ActionBar öffnen. Zu den restlichen Komponenten stellt Google Material Design Richtlinien zur Verfügung, die zeigen wie Komponenten verwendet werden sollen.

Wir erstellten unsere Layouts nach den Material Design Richtlinien. Wie in Abbildung 4.9 zu sehen ist, verwenden wir die ActionBar und den Floating Action Button. Diese sind beide in den Farben unserer Material Design Farbpalette. Durch den Einsatz von Symbolen und Hervorhebungen durch Schatten weiß der Benutzer sofort mit welchen Komponenten er interagieren kann.

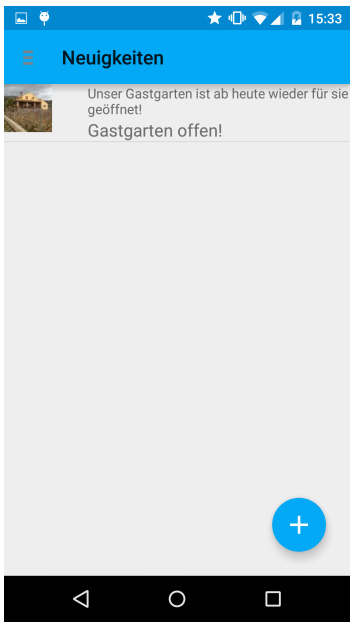


Abbildung 4.9: Material Design Oberfläche

Kapitel 5

Implementierung

5.1 REST-Client

In diesem Kapitel wird die konkrete Umsetzung des REST-Client gezeigt. Die Architektur des REST-Client wurde bereits im Kapitel Programmstruktur erklärt.

5.1.1 Android Annotations

Zu erst muss Android Annotations richtig eingebunden werden, um mit Hilfe von diesem Framework den REST-Client umzusetzen. Es bieten sich auch alternative Frameworks an, jedoch empfiehlt sich besonders für den Einsatz von REST-Clients auf Android Systemen das Framework AndroidAnnotations, da es generell die Entwicklung erleichtert. Mit dem Einsatz von Android Studio und dem Gradle Build System ist die Einbindung von AndroidAnnotations sehr einfach. In der Datei build.gradle können Frameworks aus einem sogenannten Maven Repository geladen werden. Natürlich könnte AndroidAnnotations auch über eine .jar-Datei hinzugefügt werden, jedoch bietet die Einbindung über ein externes Maven-Repository viele Vorteile. So ist die Verwaltung und Wartung der Applikation viel einfacher. Die Einbindung erfolgt wie im Code-Beispiel 5.1 gezeigt wird.

Listing 5.1: Einbindung von AndroidAnnotations in build.gradle

```
1  buildscript {
2      repositories {
3          mavenCentral()
4      }
5      dependencies {
6          classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
7      }
8  }
9
10 apply plugin: 'android-apt'
11
12
13 apt {
14     arguments {
15         androidManifestFile variant.outputs[0].processResources.manifestFile
16         resourcePackageName "at.tripwire.appcenter.app.appcenterapp"
17     }
18 }
19
```

```
20 dependencies {
21     apt "org.androidannotations:androidannotations:3.1"
22     compile 'org.androidannotations:androidannotations-api:3.1'
23     compile
           'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
24     compile 'org.codehaus.jackson:jackson-mapper-asl:1.9.13'
25 }
```

Anfangs wird das android-apt Plugin über ein Maven Repository heruntergeladen und mit apply plugin zum Projekt hinzugefügt. Dann muss das apt-plugin noch konfiguriert werden, in dem der Package Name ergänzt wird. Das apt-plugin wird von AndroidAnnotations benötigt, da es das Verarbeiten von Annotationen (welche von AndroidAnnotations verwendet werden) mit Android Studio und Gradle ermöglicht [2]. Zum Schluss wird noch der Pfad vom AndroidAnnotations Repository mit dem apt Befehl angegeben. Außerdem wird mit dem compile Befehl, welcher Frameworks von Maven Repositories definiert, die zum Kompilieren des Projekts notwendig sind [12], die AndroidAnnotations API und die benötigten Spring und Jackson Frameworks geladen. Diese werden von AndroidAnnotations REST-Client benötigt. Der apt Befehl sagt aus, dass das angegebene Maven Repository nur zum Kompilieren benötigt wird [2].

Um AndroidAnnotations in einer Activity oder in einem Fragment zu verwenden, muss die enstprechende Annotation angegeben werden. AndroidAnnotations erstellt automatische Subklassen von annotierten Activities, welche denselben Namen und einen Unterstrich am Schluss besitzen [13]. Deshalb muss auch in der AndroidManifest.xml der Pfad der MainActivity auf MainActivity_ geändert werden. Wenn eine neue Activity oder ein neues Fragment in Android gestartet wird, so muss dieses auch mit dem Namen und einem Unterstrich gestartet werden. Um AndroidAnnotations korrekt verwenden zu können, sollte jede verwendete Activity und jedes Fragment die Annotation für eine AndroidAnnotations Klasse besitzen.

5.1.2 REST-Ressourcen

Sämtliche REST-Ressourcen, die von der Applikation benötigt werden, werden im Interface LoginInterface.java deklariert. Dieses Interface muss die Annotation @Rest besitzen, welche den Einstiegspunkt der REST-Schnittstelle definiert. Diese Annotation enthält außerdem die Wurzel-URL des RESTful Webservice und einen Konverter, welcher von AndroidAnnotations vorgeschrieben wird. Als Konverter wird die Klasse MappingJacksonHttpMessageConverter vom Spring Framework benutzt. Dieser Konverter kann JSON lesen und schreiben und in Java Objekte konvertieren [8]. Außerdem haben wir noch die @Accept Annotation benutzt, welche das Rückgabeformat der REST-Antworten bestimmt [28]. Die @Accept Annotation ist nicht verpflichtend. Im Code-Beispiel 5.2 wird die Verwendung der Klasse LoginInterface gezeigt. Wir haben den Konverter MappingJacksonHttpMessageConverter benutzt, da wir ausschließlich mit JSON arbeiten. Deshalb haben wir auch als @Accept MediaType JSON gewählt. Zudem haben wir eine Methode getRestTemplate() definiert, welche das RestTemplate liefert. Dieses wird später für die Authentifzierung benötigt.

Listing 5.2: Klasse LoginInterface des REST-Client

```
1 @Rest(rootUrl = "http://center.gastroapp.at:8080/AppCenter/", converters = {
      MappingJacksonHttpMessageConverter.class})
2 @Accept(MediaType.APPLICATION_JSON)
3 public interface LoginInterface{
4     RestTemplate getRestTemplate();
```

```
5
6       @Get("api/internal/users/{mail}")
7       User getUser(String mail);
8
9       @Get("api/internal/apps")
10      List<GastroApp> getApps();
11
12      @Get("api/internal/news?app={appId}")
13      List<News> getNews(int appId);
14
15        @Post("api/internal/news/{id}?app={appId}")
16      void updateNewsEntry(News news, long id, int appId);
17
18      @Post("api/internal/news?app={appId}")
19      void createNewsEntry(News news, int appId);
20
21      @Delete("api/internal/news/{id}?app={appId}")
22      void deleteNewsEntry(long id, int appId);
23  }
```

Mit der @Get Annotation wird eine Methode definiert, welche eine HTTP-Anfrage vom Typ GET an den Webserver sendet [28]. Außerdem muss die genaue URL angegeben werden, die auf die REST-Ressource verweist. Mit den Platzhaltern { und } können zusätzliche Paramter in die URL eingefügt werden. Diese werden zum Beispiel bei der Methode getUser(String mail) und bei der Methode getNews(int appId) benötigt. Beim Aufruf einer dieser Methoden sendet der REST-Client eine HTTP-Anfrage an den Server im JSON Format und bekommt bei einer erfolgreichen Anfrage eine HTTP-Antwort in JSON zurück. Der Konverter übernimmt dann die Konvertierung in Java Objekte (POJOs).

Das Bearbeiten, Löschen und Hinzufügen von neuen Einträgen werden mit der @Post und @Delete Annotation gelöst. Die Aufrufe dazu werden im Code-Beispiel 5.2 gezeigt.

5.1.3 Authentifizierung

Die Klasse RestClient.java dient der Verwaltung von REST-Ressourcen und der Authentifizierung. Diese Klasse wird als Singleton-Bean zur Verfügung gestellt, da es über die komplette Applikation benötigt wird und nur einmal existieren darf. Die Schnittstellen werden mit der Annotation @RestService als globale Variable eingebunden. Dafür wird das Interface LoginInterface benutzt. Außerdem werden die Beans AuthStore und AuthInterceptor eingebunden. Das AuthStore Bean dient zur Verwaltung der Anmeldedaten und das AuthInterceptor Bean stellt den Interceptor für die Authentifzierung zur Verfügung.

Listing 5.3: Klasse AuthInterceptor des REST-Client

```
1  @EBean(scope = EBean.Scope.Singleton)
2  public class AuthInterceptor implements ClientHttpRequestInterceptor{
3
4      @Bean
5      protected AuthStore authStore;
6
7      public ClientHttpResponse intercept(HttpRequest request, byte[] body,
           ClientHttpRequestExecution execution) throws IOException {
8          HttpHeaders headers = request.getHeaders();
```

```
9           HttpAuthentication auth = new
                HttpBasicAuthentication(authStore.getUsername(),
                authStore.getPassword());
10          headers.setAuthorization(auth);
11          headers.set("Connection", "Keep-Alive");
12          return execution.execute(request, body);
13      }
14  }
```

Wie im Code-Beispiel 5.3 gezeigt wird, implementiert die Klasse AuthInterceptor die Klasse ClientHttpRequestInterceptor. Die Funktion dieser Klasse wurde bereits im Kapitel Programmstruktur erklärt. Außerdem besteht eine Referenz auf das AuthStore Bean und eine Methode intercept. Diese Methode fängt die HTTP-Anfrage ab, erstellt ein Objekt vom Typ HttpBasicAuthentication mit den Benutzerdaten und fügt dieses Objekt den HTTP-Header der Anfrage hinzu. Diese HTTP-Anfrage wird dann ausgeführt.

Der Aufruf dieser intercept-Methode kommt von der Klasse RestClient. Die Methode initAuth() beschafft sich zuerst das RestTemplate von der Variable loginUser des Typ LoginInterface. Dann wird eine Liste vom Typ ClientHttpRequestInterceptor erstellt, welcher der AuthInterceptor hinzugefügt wird. Dem RestTemplate wird dann diese Liste mit der Methode setInterceptors() übergeben.

Listing 5.4: initAuth() Methode der Klasse RestClient

```
1   public void initAuth() {
2           RestTemplate template = loginUser.getRestTemplate();
3           List<ClientHttpRequestInterceptor> interceptors = new
                ArrayList<ClientHttpRequestInterceptor>();
4           interceptors.add(authInterceptor);
5           template.setInterceptors(interceptors);
6   }
```

Um nun mit diesem REST-Client zu arbeiten, muss der Benutzer seine Benutzerdaten zuerst in eine Android Komponente eingeben. Das Passwort wird dann mit SHA-256 kodiert und wird dann mit dem Benutzernamen dem AuthStore Bean übergeben. Dann wird dieses Bean dem AuthInterceptor übergeben und wird schließlich dem RestClient Bean übergeben. Dann wird die Methode initAuth() des RestClient Bean aufgerufen und schließlich kann eine REST-Ressource der Klasse LoginInterface aufgerufen werden. Dieser Aufruf muss in einem anderen Thread aufgerufen werden, da Netzwerk Anfragen immer in einem anderen Thread als dem User-Interface Thread ausgeführt werden müssen [10]. Dafür stellt AndroidAnnotations die Annotation @Background zur Verfügung, welche deklariert, dass eine Methode in einem Hintergrund-Thread laufen soll. In dieser Methode kann dann über das RestClient Bean die Methode getLoginUser() aufgerufen werden, welche das LoginInterface liefert. Über dieses kann dann eine HTTP-Anfrage an den RESTful-Webservice abgesetzt werden. Falls diese HTTP-Anfrage nicht erfolgreich war, wird eine Exception geworfen. So kann kontrolliert werden ob der Benutzer korrekte Anmeldedaten angegeben hat. In der Applikation wird als erstes die REST-Ressource User angefordert, da zuerst überprüft werden muss ob der Benutzer nicht gesperrt ist und welche GastroApp der Benutzer besitzt.

5.1.4 Fazit

Der REST-Client von AndroidAnnotations funktioniert sehr gut und gliedert sich in die restliche Funktionalität ein. Durch die Annotationen, die das Erstellen von Beans oder Hintergrund-Threads erleichtern, funktionierte die Erstellung und Bedienung des REST-Clients sehr einfach. Wir können die Verwendung von AndroidAnnotations sehr empfehlen. Möchte man jedoch AndroidAnnotations nicht verwenden, so gibt es auch gute Alternativen für REST-Clients für Android wie das Spring for Android Framework.

5.2 Android Adapter

Ein wichtiger Teil der app:center App Oberflächen besteht aus Listen, deren Elemente mit Adapter gestaltet wurden. Der Adapter ermöglicht in der Softwareentwicklung Funktionalität von Klassen zu verwenden, deren Schnittstellen inkompatibel sind [47]. In Android verwendeten wir den Adapter, um in einem ListView-Eintrag auch Komponenten wie Bilder und verschiedene Texte anzeigen zu können. Mit dem Adapter gestalteten wir unsere ListView-Einträge, wie in Abbildung 5.1 zu sehen ist.

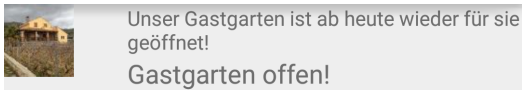


Abbildung 5.1: Anzeige zu Neuigkeiten-Einträgen mit Adapter

Um für ListView-Einträge einen Adapter zu erstellen, wird eine Java Klasse, die von BaseAdapter erbt, und eine Layout-Datei zu dem Adapter benötigt. Die Layout-Datei bestimmt die verwendeten Komponenten und deren Gestaltung. So werden, wie in Abbildung 5.1 gezeigt, bei den Neuigkeiten-Einträgen ein Bild und zwei Texte angezeigt.

Listing 5.5: News Adapter Layout Datei

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      >
3      <ImageView
4          android:id="@+id/product_imageview"/>
5
6      <TextView
7          android:id="@+id/product_textview_small"
8           />
9
10     <TextView
11         android:id="@+id/product_textview_big"
12          />
13 </RelativeLayout>
```

Die Layout-Datei ist eine Android Layout Datei, welche sämtliche Android Komponenten verwenden kann. Unser Adapter-Layout besteht, wie im Code-Beispiel 5.5 gezeigt wird, aus einem RelativeLayout, einer ImageView und zwei TextViews. Die genauen Attribute, die die Position und Größe bestimmen wurden aus Platzgründen ausgespart, verhalten sich jedoch nicht anders wie bei allen anderen Android Layouts.

Zu der Android-Layout Datei wird nun eine Java Klasse benötigt, die vom Typ BaseAdapter

erbt. Die abstrakte Klasse BaseAdapter ist von Android gegeben und dient für die Implementierung eines Adapters zu ListViews [6].

Listing 5.6: News Adapter Java Klasse

```
1  public class NewsAdapter extends BaseAdapter{
2      private Context context;
3      public List<News> news;
4
5      private class ViewHolder{
6          private ImageView imageView;
7          private TextView textViewBig;
8          private TextView textViewSmall;
9      }
10
11     @Override
12     public Object getItem(int i) {return news.get(i);}
13
14     @Override
15     public View getView(int i, View view, ViewGroup viewGroup) {
16         ViewHolder holder = null;
17
18         LayoutInflater mInflater = (LayoutInflater)
19                 context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
20         if (view == null) {
21             view = mInflater.inflate(R.layout.news_adapter, null);
22             holder = new ViewHolder();
23             holder.textViewSmall = (TextView)
                   view.findViewById(R.id.product_textview_small);
24             holder.textViewBig = (TextView)
                   view.findViewById(R.id.product_textview_big);
25             holder.imageView = (ImageView)
                   view.findViewById(R.id.product_imageview);
26             view.setTag(holder);
27         }
28         else {
29             holder = (ViewHolder) view.getTag();
30         }
31         News item = (News)getItem(i);
32         holder.textViewSmall.setText(item.getDescription());
33         holder.textViewBig.setText(item.getText());
34         Picasso.with(context).load(item.getThumbnail().getUrl())
35             .into(holder.imageView);
36         return view;
37     }
38 }
```

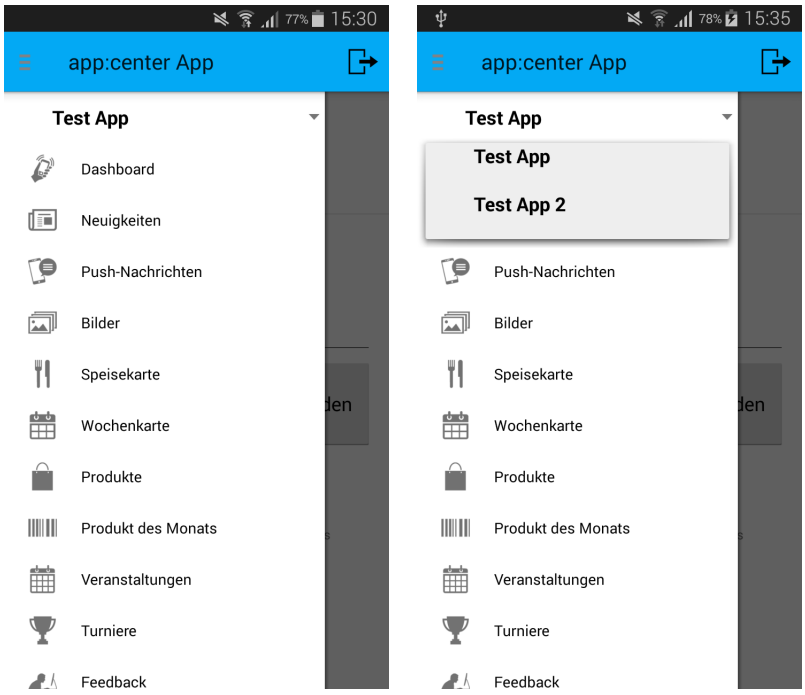
Die Klasse NewsAdapter.java besitzt, wie im Code-Beispiel 5.6 gezeigt wird, eine Referenz auf den Android-Kontext und auf eine Liste von News-Einträgen. Außerdem ist eine private Klasse ViewHolder enthalten, welche die Referenzen auf die Android Layout Komponenten speichern. In der überschriebenen Methode getView wird zuerst überprüft ob das übergebene View-Objekt bereits instanziert wurde. Wenn nicht, wird mit dem LayoutInflater über eine Layout-Datei das View-Objekt instanziert [20].

Mit diesem View-Objekt gelangt man nun zu den einzelnen Android Komponenten und kann diese den Variablen im ViewHolder zuweisen. Falls das View-Objekt bereits instanziert wurde, kann über view.getTag() der ViewHolder geholt werden. Dann wird mit der überschriebenen Methode getItem(i) und dem übergebenen Index der aktuelle News-Eintrag geholt. Dann werden die Attribute des News-Eintrag der View zugewiesen. Das Bild wird mit dem bereits beschriebenen Framework Picasso der ImageView zugewiesen.

Nun kann einer ListView der NewsAdapter zugewiesen werden. Dazu erstellt man ein neues Objekt vom NewsAdapter und übergibt den Kontext, sowie eine Liste von News-Einträgen. Dann wird der ListView mit der Methode setAdapter der Adapter zugewiesen.

5.3 Navigationsleiste

Die Navigation der App funktioniert über die Navigationsleiste (dem sogenannten Navigation-Drawer), welche sich durch eine Wisch-Geste vom linken Bildschirmrand nach rechts öffnet. Die Navigationsleiste kann auch mit einem Klick auf die drei Striche in der rechten oberen Bildschirmecke, die in Abbildung 5.2 zu sehen sind, geöffnet werden. Ein Gastronom kann mehrere Gastro Apps haben, die über eine app:center App verwaltet werden können. Dazu kann innerhalb der Navigationsleiste zwischen mehreren Gastro Apps gewechselt werden (siehe Abbildung 5.2b). Bei der Auswahl eines Eintrages (z.B. Neuigkeiten) wird das entsprechende Fragment geöffnet und die Navigationsleiste wird wieder ausgeblendet.



(a) geöffnete Navigationsleiste (b) Auswahl der Apps

Abbildung 5.2: Navigationsleiste

5.3.1 Layout in XML

Um einen NavigationDrawer anzeigen zu können, muss ein Layout in XML definiert werden. Dazu muss die Haupt-View des Layouts ein DrawerLayout sein. Innerhalb dieses DrawerLayouts muss ein Layout existieren, welches angezeigt wird, sobald der NavigationDrawer ausgeblendet wird, und ein weiteres Layout, dass den Inhalt des NavigationDrawers darstellt.[11]

Wie im Code-Beispiel 5.7 gezeigt wird, wird ein DrawerLayout definiert, welches ein FrameLayout und eine ListView enthält. Das FrameLayout ist zur Anzeige des Hauptinhaltes (der Fragments) notwendig. Die ListView enthält die Einträge, welche beim Öffnen des NavigationDrawers angezeigt werden.

Listing 5.7: NavigationDrawer Layout Datei

```
<android.support.v4.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/drawer_layout"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <FrameLayout
      android:id="@+id/content_frame"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>

   <ListView
      android:id="@+id/left_drawer"
      android:layout_width="300dp"
      android:layout_height="match_parent"
      android:layout_gravity="start"
      android:choiceMode="singleChoice"
      android:divider="@null"
      android:dividerHeight="0dp"
      android:background="#FFFFFF"/>
</android.support.v4.widget.DrawerLayout>
```

Aus dem Code-Beispiel 5.7 gehen einige wichtige Layout Eigenschaften hervor [11]:

- Die View zur Darstellung des Hauptinhaltes (das FrameLayout) muss das erste Element im DrawerLayout sein.
- Die Breite und die Höhe der View zur Darstellung des Hauptinhaltes muss auf den Wert match_parent gesetzt werden. Dies ist zur Ausfüllung des gesamten Bildschirms notwendig, falls der NavigationDrawer ausgeblendet ist.
- Die ListView stellt die Breite in dp Einheiten dar und die Höhe ist auf match_parent gesetzt. Die Breite sollte nicht mehr als 320dp betragen, damit der Benutzer noch einen Teil des Hauptinhaltes sehen kann.

5.3.2 Initialisierung der ListView

In der Aktivität muss die ListView, die zur Anzeige des NavigationDrawers notwendig ist, initialisiert werden.

Listing 5.8: Initialisierung der ListView

```
protected List<ModuleEntry> modules;
protected List<GastroApp> apps;
private ListView mDrawerList;
private ArrayList<DrawerItem> dataList;
protected CustomDrawerAdapter adapter;

...

@Override
protected void onCreate(Bundle savedInstanceState) {

   ...

   dataList=getApp();
   adapter=new CustomDrawerAdapter(this, R.layout.custom_drawer_item, dataList, 0);
   mDrawerList.setAdapter(adapter);
   mDrawerList.setOnItemClickListener(new DrawerItemClickListener());

   ...
}

private ArrayList<DrawerItem> getDataList(int pos){
   ArrayList<DrawerItem> dataList = new ArrayList<DrawerItem>();

   ...

   if(apps.size()>1){
      ArrayList<String> names = new ArrayList<String>();

      for(GastroApp app: apps){
         names.add(app.getName());
      }

      dataList.add(new DrawerItem(true, names));

      ...

      modules = apps.get(pos).getAppModules();
   }else if(appsForRemoteUse.size()==1){

      ...

      modules = apps.get(0).getAppModules();
   }

   if(modules != null){
      for(int i = 0; i < modules.size(); i++){
         Module module = modules.get(i).getModule();
         String item = module.getTitle();

         if(item.equals("Neuigkeiten")){
```

```
52              dataList.add(new DrawerItem(item, R.drawable.drawer_news));
53          }
54
55              ...
56          }
57
58      }else{
59          dataList.add(new DrawerItem("Keine Module gefunden!"));
60      }
61      return dataList;
62  }
```

Wie im Code-Beispiel 5.8 zu sehen ist, wird die ArrayList dataList durch die Methode getApp() initialisiert. Die Methode getApp() befüllt die Liste apps mit GastroApp Objekten. Mithilfe dieser Liste ist es möglich, zwischen mehreren GastroApps zu wechseln. In der Methode getApp() wird die Methode getDataList() aufgerufen. Dieser Methode wird die Position der ausgewählten GastroApp mitgegeben. Standardmäßig ist diese Position 0. In der Methode getDataList() wird eine neue ArrayList dataList von DrawerItems erstellt. Die Klasse DrawerItem repräsentiert einen Eintrag der Navigationsleiste. Wenn es mehr als eine GastroApp gibt, wird der ArrayList dataList eine Liste der GastroApp Namen hinzugefügt. Die Klassenvariable modules enthält nun alle Module der aktuellen GastroApp. Falls nur eine GastroApp vorhanden ist, wird der ArrayList dataList keine Liste mit GastroApp Namen hinzugefügt. Falls Module vorhanden sind, wird je nach Titel des Moduls der ArrayList dataList ein neuer Eintrag mit dem Titel des Moduls und einem Symbol für das jeweilige Modul hinzugefügt. Sind keine Module vorhanden, wird der ArrayList ein neuer Eintrag mit „Keine Module gefunden!“ hinzugefügt. Somit ist es möglich, den NavigationDrawer dynamisch mit den Modulen, die eine GastroApp besitzt, zu befüllen. Zum Schluss wird die ArrayList dataList zurückgegeben.

In der onCreate() Methode erhält die Klassenvariable dataList also eine Referenz auf die ArrayList dataList, welche in der Methode getDataList() initialisiert und befüllt wurde. Nach Aufruf der Methode getApp() wird der ListView ein Adapter der Klasse CustomDrawerAdapter gesetzt, welcher die Liste der DrawerItems enthält und anzeigt. Für die Verwendung von Adapter siehe Kapitel 5.2 Adapter. Danach wird der ListView noch ein ClickListener hinzugefügt, damit auf einen Klick auf eines der Elemente des NavigationDrawers reagiert werden kann. Das Reagieren von Klicks wird im nachfolgenden Kapitel und im Code-Beispiel 5.9 gezeigt.

5.3.3 Öffnen eines Elements

Wenn der Benutzer ein Element des NavigationDrawers der ListView auswählt, sollte darauf reagiert werden. Das ist durch den Aufruf mDrawerList.setOnItemClickListener(new DrawerItemClickListener()); in der Methode onCreate() möglich. Der Methode setOnItemClickListener() muss ein Objekt der Klasse DrawerItemClickListener mitgegeben werden. Diese private Klasse implementiert das Interface ListView.OnItemClickListener (siehe Code-Beispiel 5.9). Durch einen Klick auf ein Element der ListView wird die Methode selectItem() aufgerufen, welcher die Position des ausgewählten Elements mitgegeben wird.

Listing 5.9: Öffnen eines Elements des NavigationDrawers

```
1  private class DrawerItemClickListener implements ListView.OnItemClickListener {
2      @Override
```

```
3     public void onItemClick(AdapterView<?> parent, View view, int position, long
          id) {
4        selectItem(position);
5     }
6  }
7
8  private void selectItem(int position) {
9     DrawerItem selectedItem = (DrawerItem)mDrawerList.getItemAtPosition(position);
10    String item = selectedItem.getItemName();
11    Fragment fragment = null;
12
13    if(item.equals("Neuigkeiten")){
14       fragment = new NewsFragment_();
15       setTitle("Neuigkeiten");
16    }
17
18    ...
19
20    FragmentManager fragmentManager = getSupportFragmentManager();
21    fragmentManager.beginTransaction().replace(R.id.content_frame,
          fragment).commit();
22
23    mDrawerLayout.closeDrawer(mDrawerList);
24 }
25
26 @Override
27 public void setTitle(CharSequence title) {
28    mTitle = title;
29    getSupportActionBar().setTitle(mTitle);
30 }
```

In der Methode selectItem() wird der Name des ausgewählten Elements des NavigationDrawers in der Variable item gespeichert. Danach wird überprüft, welches Element ausgewählt wurde, und ein Objekt des entsprechenden Fragments erzeugt. Dieses Fragment wird in der Variable fragment gespeichert. Außerdem wird der Name des richtigen Elements gesetzt. Dies geschieht mit der Methode setTitle(). In dieser Methode wird der Klassenvariable mTitle der richtige Titel des Fragments gesetzt und in der ActionBar angezeigt. Sobald bekannt ist, welches Element ausgewählt wurde, wird ein Objekt der Klasse FragmentManager erstellt. Mithilfe dieses Objekts ist es möglich, ein anderes Fragment anzuzeigen. Nach der Erzeugung des Objekts wird der aktuelle Inhalt des FrameLayouts content_frame mit dem ausgewählten Fragment ersetzt. Zum Schluss wird das DrawerLayout mDrawerLayout durch den Aufruf der Methode closeDrawer() ausgeblendet. Wie der NavigationDrawer ein- bzw. ausgeblendet werden kann, wird im nachfolgenden Kapitel und im Code-Beispiel 5.10 dargestellt.

5.3.4 NavigationDrawer öffnen und schließen

Um den NavigationDrawer öffnen bzw. schließen zu können, muss die Methode setDrawerListener() des DrawerLayouts aufgerufen werden. Dieser Methode muss eine Implementierung des Interfaces DrawerLayout.DrawerListener mitgegeben werden. Dieses Interface umfasst Methoden, wie z.B. onDrawerOpened() oder onDrawerClosed(), um auf Änderungen des NavigationDrawers eingehen zu können. Wenn jedoch die Aktivität die ActionBar enthält, kann die ActionBarDrawerToggle Klasse erweitert werden. Diese Klasse implementiert das DrawerLay-

out.DrawerListener Interface und somit ist es möglich die Methoden zum ein- bzw. ausblenden des NavigationDrawers zu überschreiben.[11]

Das Code-Beispiel 5.10 zeigt, wie der Titel des ausgewählten NavigationDrawer Elements geändert werden kann. Dazu wird eine Instanz der Klasse ActionBarDrawerToggle verwendet.

Listing 5.10: NavigationDrawer ein- bzw. ausblenden

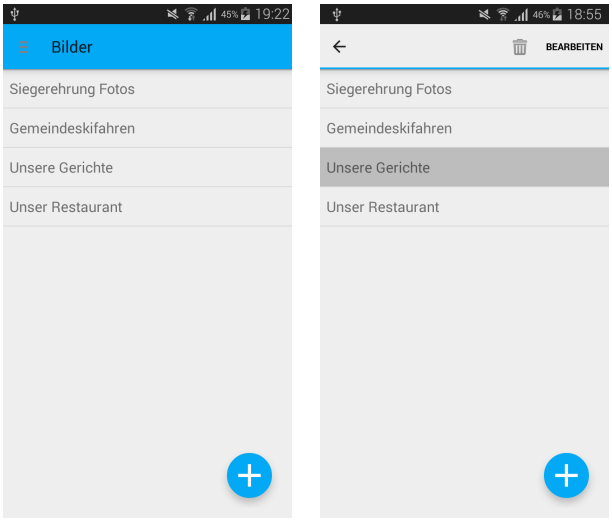
```
1  private ActionBarDrawerToggle mDrawerToggle;
2  private DrawerLayout mDrawerLayout;
3
4  ...
5
6  @Override
7  protected void onCreate(Bundle savedInstanceState) {
8
9     ...
10
11    mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
12    mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
          R.drawable.ic_drawer, R.string.drawer_open, R.string.drawer_close) {
13
14       public void onDrawerClosed(View view) {
15          getSupportActionBar().setTitle(mTitle);
16       }
17
18       public void onDrawerOpened(View drawerView) {
19          getSupportActionBar().setTitle(getTitle());
20       }
21    };
22
23    mDrawerLayout.setDrawerListener(mDrawerToggle);
24
25    ...
26 }
```

In der Methode onCreate() wird das Objekt mDrawerToggle initialisiert. Dazu werden die Methoden onDrawerClosed() und onDrawerOpened() implementiert. Die Methode onDrawerClosed() wird aufgerufen, sobald der NavigationDrawer komplett ausgeblendet ist. In dieser Methode wird der Titel, welcher in der ActionBar angezeigt wird, auf den Titel mTitle gesetzt. Dieses Objekt wird in der Methode setTitle(), welche im Code-Beispiel 5.9 zu sehen ist, bei jedem Aufruf eines neuen Fragments gesetzt. Die Methode onDrawerOpened() wird aufgerufen, sobald der NavigationDrawer komplett eingeblendet ist. In dieser Methode wird der Titel auf den Namen der App geändert. Zum Schluss wird das ActionBarDrawerToggle Objekt mDrawerToggle als DrawerListener dem DrawerLayout gesetzt.[11]
Öffnet nun ein Benutzer die Navigationsleiste, wird der Name der App in der ActionBar angezeigt. Schließt er die Navigationsleiste wieder, wird der Titel des ausgewählten Fragments angezeigt.

5.4 Nutzung von Hardware und Android Funktionen

5.4.1 ActionMode

Das Kontextmenü ActionMode ist ein Steuerelement, das dem Benutzer zu einem bestimmten Kontext verschiedene Aktionen zur Auswahl anbietet. Das Kontextmenü enthält nur Menüpunkte, die zum jeweiligen Zeitpunkt für das ausgewählte Element sinnvoll sind. [40]
Der kontextabhängige ActionMode ist ein fließendes Menü, d.h. es wird durch einen längeren Klick auf ein Element angezeigt und wird nach der Auswahl eines Menüpunktes oder nach Beenden des ActionModes wieder ausgeblendet. Das Menü wird in der sogenannten ActionBar am oberen Rand des Bildschirms angezeigt [24].



(a) Bildgalerien (b) ActionMode

Abbildung 5.3: ActionMode bei den Bildgalerien

Die Abbildung 5.3 zeigt, dass nach einem längeren Klick auf den Eintrag „Unsere Gerichte“ am oberen Bildschirmrand die ActionBar mit dem ActionMode erscheint. Es ist möglich, die Bildgalerie zu löschen oder zu bearbeiten. Mit dem Pfeil links oben wird der ActionMode beendet und die ActionBar wieder ausgeblendet.

Definition eines Menüs in XML

Damit das Menü nicht im Java Code der jeweiligen Aktivität implementiert werden muss, ist es möglich, das Menü mit allen Menüpunkten in einer XML Menü Ressource zu definieren. Dazu bietet Android ein Standard XML Format an.

Damit eine XML Menü Ressource erstellt werden kann, muss im Applikations-Ressourcen Ordner „res“ ein Unterordner mit dem Namen „menu“ erstellt werden. In diesem Verzeichnis können nun Menü Ressourcen erstellt werden.[24]

Die XML Menü Ressource aus Abbildung 5.3 kann wie folgt aussehen:

Listing 5.11: ActionMode XML-Layout

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/delete"
        android:icon="@drawable/trash_can"
        android:title="@string/dialog_delete"
        android:showAsAction="always" />

    <item
        android:id="@+id/edit"
        android:title="@string/edit"
        android:showAsAction="always" />
</menu>
```

Das <menu>Tag aus der ersten Zeile definiert einen Container für Menü Elemente und muss immer das erste Tag einer XML Menü Ressource Datei sein. Es können ein oder mehrere <item>und <group>Tags enthalten sein. Das <item>Tag definiert ein einzelnes Menü Element. Das <group>Tag wurde in diesem Beispiel nicht verwendet, es ist damit allerdings möglich, Menü Elemente zu kategorisieren und zu Gruppen zusammenzufassen.[24]

Ein <item>Tag unterstützt verschiedene Attribute, mit denen das Verhalten und Aussehen eines Menü Elements definiert werden kann [24]:

- android:id
  - Eine ID, die einzigartig ist und sich auf das Menü Element bezieht. Dadurch kann die Applikation auf das Element reagieren, falls der Benutzer dieses Menü Element auswählt.
- android:icon
  - Eine Referenz auf ein Bild, welches als Symbol für das Menü Element verwendet wird.
- android:title
  - Eine Referenz auf einen String, welcher als Titel für das Menü Element verwendet wird.
- android:showAsAction
  - Legt fest, wann und wie das Element im Balken des ActionModes dargestellt wird.

Verwendung des kontextabhängigen ActionModes

Um den ActionMode verwenden zu können, muss auf folgendes geachtet werden:

- Das ActionMode.Callback Interface muss implementiert werden. In den vorgegebenen Methoden des Interfaces kann auf Aktionen des Menü Balkens reagiert werden.
- Der ActionMode wird in der ActionBar durch Aufruf der Methode startActionMode() angezeigt.

Der ActionMode, wie er in Abbildung 5.3 zu sehen ist, wird auf eine ListView angewendet. Die Auswahl und Ausführung von Aktionen mehrerer Elemente einer ListView kann im Actionode implementiert werden.

Die Implementierung des ActionMode.Callback Interfaces kann wie folgt aussehen:

Listing 5.12: ActionMode Implementierung

```
1  private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {
2     @Override
3     public boolean onCreateActionMode(ActionMode mode, Menu menu) {
4        MenuInflater inflater = mode.getMenuInflater();
5        inflater.inflate(R.menu.menu_category, menu);
6        return true;
7     }
8
9     @Override
10    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
11       return false;
12    }
13
14    @Override
15    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
16       final int position = (Integer)mode.getTag();
17       switch (item.getItemId()) {
18          case R.id.delete:
19             AlertDialog.Builder adb=new AlertDialog.Builder(getActivity());
20             adb.setTitle("Löschen");
21             adb.setMessage(position+". Element löschen!");
22             adb.show();
23             mode.finish();
24             return true;
25          case R.id.edit:
26             AlertDialog.Builder adb=new AlertDialog.Builder(getActivity());
27             adb.setTitle("Bearbeiten");
28             adb.setMessage(position+". Element bearbeiten!");
29             adb.show();
30
31             mode.finish();
32             return true;
33          default:
34             return false;
35       }
36    }
37
38    @Override
39    public void onDestroyActionMode(ActionMode mode) {
40       mActionMode = null;
41    }
42 };
```

Die Methode onCreateActionMode() des ActionMode.Callback Interfaces wird aufgerufen, sobald ein ActionMode Objekt erstellt wird (bei Aufruf der Methode startActionMode()). In dieser

Methode wird eine XML Menü Ressource ausgewählt, die in der ActionBar angezeigt werden soll. In diesem Fall handelt es sich um die XML Menü Ressource menu_category. Die Methode onPrepareActionMode() wird jedes mal, wenn der ActionMode angezeigt wird, aufgerufen. In dieser Methode ist es möglich, den ActionMode vorzubereiten und Änderungen vorzunehmen. Falls nichts geändert wird, wird false zurückgegeben. In der Methode onActionItemClicked() werden die Aktionen, die durch eine Auswahl eines Menü Elements des Benutzers ausgelöst werden, verarbeitet. Es sind die Aktionen Löschen und Bearbeiten vorhanden. Durch den Aufruf mode.getTag() wird jener Integer Wert ausgelesen, der beim Aufruf des ActionMode übergeben wurde. Falls der Benutzer das Menü Element „Löschen“ oder „Bearbeiten“ auswählt, wird ein AlertDialog erstellt, der eine Meldung mit der aktuellen Position des ausgewählten ListView Elements anzeigt. Zusätzlich wird je nach Aufruf „löschen“ bzw. „bearbeiten“ ausgegeben. Die Methode onDestroyActionMode() wird aufgerufen, sobald der ActionMode beendet wird. Darin wird der ActionMode auf null gesetzt und das Menü wird in der ActionBar ausgeblendet.

Hier wird gezeigt, wie der ActionMode erstellt wird, wenn der Benutzer länger auf ein Element der ListView klickt:

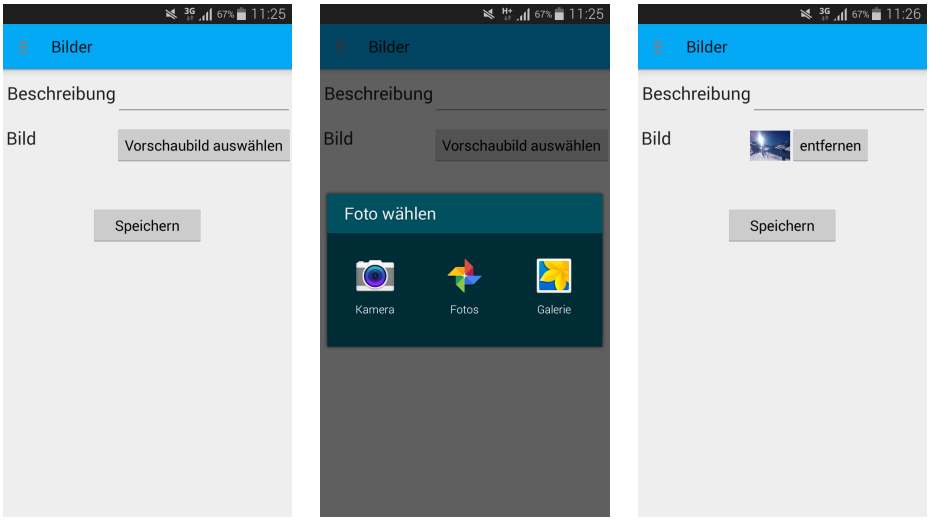
Listing 5.13: Erstellung des ActionMode

```
1  listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
2     public boolean onItemLongClick(AdapterView<?> parent, View v, int position,
          long id) {
3        if (mActionMode != null) {
4           return false;
5        }
6
7        mActionMode = ((NavigationActivity)getActivity())
8                .startSupportActionMode(mActionModeCallback);
9        mActionMode.setTag(position);
10       return true;
11    }
12 });
```

Dem ListView Objekt listView wird ein ItemLongClickListener hinzugefügt. Dadurch ist es möglich, auf einen längeren Klick auf eines der Elemente der ListView zu reagieren. Man erhält automatisch die Position des ausgewählten Elements der ListView. Falls der ActionMode bereits angezeigt wird, ist dieser ungleich null, es wird false zurückgegeben und nichts geschieht.
Die Variable mActionMode ist eine Klassenvariable des Typs ActionMode. Mithilfe der Klassenvariable ist es möglich, das Menü in der ActionBar auszublenden, falls andere Aktionen, wie zum Beispiel das Öffnen einer neuen Aktivität oder eines neuen Fragments, ausgeführt werden. Dabei muss man die Klassenvariable auf null setzen. Die Variable mActionMode wird durch den Aufruf der Methode startSupportActionMode() initialisiert. Dieser Methode wird das ActionMode.Callback Objekt mActionModeCallback übergeben. Dem ActionMode Objekt wird die Position des ausgewählten ListView Elements mithilfe der Methode setTag() übergeben. Dadurch ist es möglich, im ActionMode, wie vorhin beschrieben, auf das ausgewählte Element die Aktionen Bearbeiten und Löschen anzuwenden.

5.4.2 PhotoPicker

Mithilfe des PhotoPickers ist es möglich ein Foto aus dem File System des Android Geräts auszuwählen, um dieses in der App verwenden zu können. Es ist auch möglich ein Foto direkt mit der Kamera aufzunehmen.



(a) Bilder Fragment (b) PhotoPicker öffnen (c) Bild ausgewählt

Abbildung 5.4: Einsatz des PhotoPicker bei den Bildern

Durch Drücken des in Abbildung 5.4a dargestellten Buttons „Vorschaubild auswählen“ wird der PhotoPicker geöffnet. Der geöffnete PhotoPicker ist in Abbildung 5.4b abgebildet. Es ist nun möglich ein Foto direkt aufzunehmen, indem man die Kamera App öffnet. Man kann auch ein Bild über die Apps „Galerie“ oder „Fotos“ auswählen. Nach der Auswahl bzw. Aufnahme des Fotos kehrt man aus der fremden App (Kamera, Galerie, Fotos) zur eigenen App zurück. In Abbildung 5.4c ist das ausgewählte Foto zu sehen. Mit dem Button „entfernen“ ist es möglich, das Foto wieder zu löschen, um ein anderes Bild auszuwählen.

PhotoPicker Klasse

Die PhotoPicker Klasse sieht wie folgt aus:

Listing 5.14: PhotoPicker Implementierung

```
1  private Uri outputFileUri;
2  private Bitmap bp;
3
4  public void openIntents() {
5     File root=new File(Environment.getExternalStorageDirectory()+"/MyDir/");
6     root.mkdirs();
7     String fname=System.currentTimeMillis()+".jpg";
8     outputFileUri=Uri.fromFile(new File(root, fname));
9
10    Intent captureIntent=new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
11    PackageManager packageManager=activity.getActivity().getPackageManager();
12    ResolveInfo res=packageManager.resolveActivity(captureIntent, 0);
13
```

```
14    captureIntent.setComponent(new ComponentName(res.activityInfo.packageName,
          res.activityInfo.name));
15    captureIntent.setPackage(res.activityInfo.packageName);
16    captureIntent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
17
18     List<Intent> cameraIntents = new ArrayList<>();
19     cameraIntents.add(intent);
20
21     Intent galleryIntent=new Intent(Intent.ACTION_GET_CONTENT);
22    galleryIntent.setType("image/*");
23
24     Intent chooserIntent=Intent.createChooser(galleryIntent, "Foto wählen");
25     chooserIntent.putExtra(
26        Intent.EXTRA_INITIAL_INTENTS, cameraIntents.toArray(new Parcelable[]{}));
27
28    fragment.startActivityForResult(chooserIntent, 1);
29  }
30
31  public void onActivityResult(int requestCode, int resultCode, Intent data) {
32     if(resultCode==Activity.RESULT_OK && requestCode==1){
33        Uri selectedImageUri=null;
34        boolean isCamera;
35
36        if(data==null){
37           isCamera=true;
38        }else{
39           String action=data.getAction();
40           if(action==null){
41              isCamera=false;
42              selectedImageUri=data.getData();
43           }else{
44              isCamera=action.equals(MediaStore.ACTION_IMAGE_CAPTURE);
45           }
46      }
47      if(isCamera){
48           selectedImageUri=outputFileUri;
49      }
50
51      try{
52           bp=Media.getBitmap(
53              fragment.getActivity().getContentResolver(), selectedImageUri);
54      }catch(IOException e){
55           e.printStackTrace();
56      }
57     }
58  }
59
60  public Bitmap getBp() { return bp; }
```

Dem Konstruktor der PhotoPicker Klasse muss ein Objekt der Klasse Fragment mitgegeben werden. Dieses Fragment Objekt wird als private Klassenvariable fragment gespeichert. Zusätzlich wird noch ein privates Uri Objekt outputFileUri und ein privates Bitmap Objekt bp gespeichert. Die Klassenvariable bp wird im Konstruktor auf null gesetzt. Das Bitmap Objekt enthält nach

Beenden des PhotoPickers das ausgewählte Foto. In der Methode openIntents wird in den ersten vier Zeilen das Uri Objekt outputFileUri initialisiert. Das Uri Objekt definiert eine bestimmte Datei, welche sich, sofern mit dem PhotoPicker ein neues Foto aufgenommen wurde, im Verzeichnis „MyDir“ befindet. Danach wird ein neuer Intent captureIntent erzeugt. Ein Intent ist ein Aufruf, etwas zu tun. Das Android Betriebssystem, andere Apps oder andere Activities reagieren auf den Intent, der für sie bestimmt ist. Dabei ist es möglich zwischen Apps und Activities zu wechseln.[16]

Diesem Intent wird der Aufruf zugeteilt, die Kamera App zu öffnen, um ein Bild aufzunehmen und zurückzugeben [23]. Das Objekt packageManager der Klasse PackageManager beinhaltet Informationen über alle Pakete/Apps, die auf einem Android Gerät installiert sind [25].
Das Objekt res der Klasse ResolveInfo beinhaltet Informationen über die am besten geeignete App/Activity für den Intent captureIntent. Das Objekt captureIntent wird mit Zusatzinformationen und dem outputFileUri Objekt bestückt. Der Kamera Intent benötigt das outputFileUri Objekt, um das aufgenommene Foto im richtigen Verzeichnis speichern zu können. Der Liste cameraIntents wird nun der Kamera Intent hinzugefügt. Falls mehrere Kamera Apps auf dem Android Gerät installiert sind, könnte man nun dieser Liste noch weitere Kamera Intents hinzufügen. Der Intent galleryIntent ist der Aufruf um Daten auszuwählen. Die Daten, welche ausgewählt werden können, werden durch setType(ïmage/*”) auf Bilder eingeschränkt.[16]

Das Objekt chooserIntent ist ein Intent, der eine Auswahlmöglichkeit anbietet. Dieser Intent enthält den Intent galleryIntent und die cameraIntents Liste. Die Meldung, die beim Öffnen dieses Intents angezeigt wird, lautet „Foto wählen“. Zum Schluss der Methode openIntents wird die Methode startActivityForResult der Klassenvariable fragment aufgerufen. Mit dieser Methode wird eine Aktivität gestartet, von der ein Ergebnis erwartet wird. Die fremde Aktivität, die gestartet werden soll, ist der Intent chooserIntent. Der Integer Wert 1, welcher der Methode übergeben wird, wird in der Methode onActivityResult zurückgegeben, sobald wieder zur eigenen Aktivität zurückgekehrt wird. Für diesen Integer Wert kann ein beliebiger Wert über Null angenommen werden. Durch diesen Wert kann festgestellt werden, ob genau von dem Intent, welcher der Methode startActivityForResult mitgegeben wurde, zurückgekehrt wird.[1]

In der Methode onActivityResult wird geprüft, ob erfolgreich von dem Intent mit dem requestCode 1 zurückgekehrt wurde. Falls erfolgreich zurückgekehrt wurde, werden zwei neue Variablen (selectedImageUri, isCamera) deklariert. Falls der Intent data null ist, wird die Variable isCamera auf true gesetzt. Es wurde also kein vorhandenes Bild aus der Galerie ausgewählt, sondern ein neues mit der Kamera aufgenommen. Falls der Intent data ungleich null ist und der String action null ist, wird die Variable isCamera auf false gesetzt und das Uri Objekt selctedImageUri initialisiert. Falls der String action ungleich null ist, wird geprüft, ob ein neues Foto aufgenommen wurde oder ein bereits vorhandenes ausgewählt wurde. Falls ein neues Foto aufgenommen wurde (isCamera=true), wird das Uri Objekt selctedImageUri auf den Wert der Klassenvariable outputFileUri gesetzt. Danach wird der Bitmap bp eine Bitmap zur entsprechenden Uri gesetzt. Mit der Methode getBp ist die Bitmap von außen erreichbar.

Layout in XML

Um einen PhotoPicker so anwenden zu können, wie es in Abbildung 5.4 zu sehen ist, muss ein gewisses Layout in XML erstellt werden. Auf das Layout zur Eingabe einer Beschreibung und auf den Button „Speichern“ wird im Folgenden nicht näher eingegangen.

Listing 5.15: PhotoPicker XML-Layout

```
<Button
   android:id="@+id/chooseImageGallery"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Vorschaubild auswählen"/>

<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <ImageView
      android:id="@+id/imageViewGallery"
      android:layout_width="50px"
      android:layout_height="50px"
      android:layout_gravity="left"/>

   <Button
      android:id="@+id/removeImageGallery"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:text="entfernen"
      android:layout_gravity="right"
      android:visibility="invisible"/>

</LinearLayout>
```

Der Button „Vorschaubild auswählen“ ist anfangs sichtbar. Neben dem Button befindet sich ein LinearLayout. In diesem LinearLayout befinden sich eine ImageView und ein Button. Die ImageView hat eine Breite und eine Höhe von jeweils 50 Pixel und ist zur Anzeige des ausgewählten Fotos notwendig. Die ImageView befindet sich auf der linken Seite (android:layout_gravity=“left“) des LinearLayouts. Der Button „entfernen“ ist anfangs unsichtbar (android:visibility=“invisible“). Dieser Button befindet sich auf der rechten Seite (android:layout_gravity=“right“) des LinearLayouts.

Verwendung des PhotoPicker

Der PhotoPicker kann wie folgt verwendet werden:

Listing 5.16: Verwendung des PhotoPicker

```
protected PhotoPicker p;
protected Image selectedImage;
protected ImageView imageViewGallery;
protected Button removeImageGallery;
protected Button chooseImageGallery;

@Click
public void chooseImageGallery(){
   p=new PhotoPicker(this);
   p.openIntents();
}

```

```
13  @Override
14  public void onActivityResult(int requestCode, int resultCode, Intent data){
15     p.onActivityResult(requestCode, resultCode, data);
16     Bitmap bp = p.getBp();
17
18     if(bp != null){
19        ByteArrayOutputStream baos = new ByteArrayOutputStream();
20        bp.compress(Bitmap.CompressFormat.JPEG, 70, baos);
21        byte[] b = baos.toByteArray();
22
23        Image image = new Image();
24        image.setFile(""+ Base64.encodeToString(b,
              Base64.DEFAULT));
25
26        selectedImage = image;
27
28        imageViewGallery.setImageBitmap(bp);
29        chooseImageGallery.setVisibility(View.INVISIBLE);
30        removeImageGallery.setVisibility(View.VISIBLE);
31     }
32  }
33  @Click
34  public void removeImageGallery(){
35     imageViewGallery.setImageBitmap(null);
36     selectedImage=null;
37     chooseImageGallery.setVisibility(View.VISIBLE);
38     removeImageGallery.setVisibility(View.INVISIBLE);
39  }
```

Durch einem Klick auf den Button „Vorschaubild auswählen“ wird ein neues Objekt der Klasse PhotoPicker erzeugt. Danach wird die Methode openIntents() des PhotoPickers p aufgerufen (siehe Kapitel PhotoPicker Klasse). Die Methode onActivityResult() muss überschrieben werden. Diese Methode erhält das Ergebnis der startActivityForResult() Methode, welche im PhotoPicker in der Methode openIntents() aufgerufen wurde.[14]

In der Methode onActivityResult() wird die Methode onActivityResult() des PhotoPicker Objekts p aufgerufen. Danach wird die Bitmap des ausgewählten oder aufgenommenen Fotos in dem Bitmap Objekt bp gespeichert. Falls das Bitmap Objekt ungleich null ist, wird die Bitmap auf 70 Prozent der aktuellen Größe komprimiert und in ein Byte Array b gespeichert. Dem Image Objekt image wird nun das Byte Array, welches Base64 kodiert wird, gesetzt. Danach wird die Klassenvariable selectedImage auf das Objekt image gesetzt. Die ImageView zeigt das Bild an, der Button „Vorschaubild auswählen“ wird unsichtbar und der Button „entfernen“ wird eingeblendet. Durch einen Klick auf den Button „entfernen“ wird das Foto aus der ImageView entfernt, das Objekt selectedImage auf null gesetzt und die beiden Buttons werden sichtbar bzw. unsichtbar gesetzt.

Kapitel 6

Beurteilung

Durch die Implementierung und Verfassung unserer Diplomarbeit konnten wir einige wertvolle Erfahrungen gewinnen. Wir konnten unser Wissen bei der Implementierung von mobilen Apps und bei der Verfassung von wissenschaftlichen Arbeiten sehr vergrößern.

Wir haben bereits sehr früh mit der intensiven Planung unserer Diplomarbeit begonnen. Dies war ein großer Vorteil, da wir nie unter Zeitdruck geraten sind. Wir konnten für sämtliche Probleme immer die beste Lösung finden, da wir uns ausreichend Zeit genommen haben, um diese Probleme zu analysieren und Lösungsvorschläge zu finden. So haben wir gesehen, dass die Planung nicht unterschätzt werden darf und eine ausreichende Planung bei der späteren Implementierung sehr hilft.

Da diese Arbeit auf einem bereits bestehenden Produkt aufbaut, waren wir ständig in Kontakt mit dem Auftraggeber Marcus Holzleitner und hielten viele gemeinsame Treffen ab. Wir haben die fertigen Teilbereiche unserer Implementierung immer dem Auftraggeber vorgelegt und konnten so wertvolle Rückmeldungen gewinnen.

Bei der Verfassung der Diplomarbeit holten wir uns ständig Rückmeldungen von unserem Projektbetreuer Dipl.-Ing. Dr. Michael Buchberger und konnten so viele Fehler bereits im Vorhinein vermeiden.

Abschließend möchten wir uns nochmal bei unserem Auftraggeber Marcus Holzleitner, unserer Kontaktperson Ing. David Andlinger und unserem Projektbetreuer Dipl.-Ing. Dr. Michael Buchberger für die Hilfe bei der Implementierung und Verfassung dieser Arbeit bedanken.

Literaturverzeichnis

[1] *Activity*. http://developer.android.com/reference/android/app/Activity.html, zugegriffen am 18.03.2015.

[2] Android-apt. https://bitbucket.org/hvisser/android-apt, zugegriffen am 10.4.2015.

[3] *Android Manifest*. http://developer.android.com/guide/topics/manifest/uses-sdk-element.html, zugegriffen am 14.02.2015.

[4] *Android Studio*. https://developer.android.com/tools/studio/index.html, zugegriffen am 14.02.2015.

[5] *AndroidAnnotations*. http://androidannotations.org/, zugegriffen am 10.4.2015.

[6] *BaseAdapter*. http://developer.android.com/reference/android/widget/BaseAdapter.html, zugegriffen am 10.4.2015.

[7] *Building a RESTful Web Service*. https://spring.io/guides/gs/rest-service/, zugegriffen am 13.2.2015.

[8] *Class MappingJacksonHttpMessageConverter*. http://docs.spring.io/spring/docs/4.0.9.RELEASE/javadoc-api/org/springframework/http/converter/json/MappingJacksonHttpMessageConverter.html, zugegriffen am 10.4.2015.

[9] *Class RestTemplate*. http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html, zugegriffen am 10.4.2015.

[10] *Connecting to the Network*. http://developer.android.com/training/basics/network-ops/connecting.html, zugegriffen am 10.4.2015.

[11] *Creating a Navigation Drawer*. https://developer.android.com/training/implementing-navigation/nav-drawer.html, zugegriffen am 21.03.2015.

[12] *Dependency Management Basics*. http://gradle.org/docs/current/userguide/artifact_dependencies_tutorial.html, zugegriffen am 10.4.2015.

[13] Firstactivity. https://github.com/excilys/androidannotations/wiki/FirstActivity, zugegriffen am 10.4.2015.

[14] *Fragment*. http://developer.android.com/reference/android/app/Fragment.html, zugegriffen am 10.4.2015.

[15] *Fragments*. http://developer.android.com/guide/components/fragments.html, zugegriffen am 14.02.2015, Grafik aus dem Englischen übersetzt.

[16] *Intent*. http://developer.android.com/reference/android/content/Intent.html, zugegriffen am 17.03.2015.

[17] *Interface ClientHttpRequestInterceptor*. http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/client/ClientHttpRequestInterceptor.html, zugegriffen am 10.4.2015.

[18] *Introduction to Android*. http://developer.android.com/guide/index.html, zugegriffen am 28.4.2015.

[19] *Jersey Getting Started*. https://jersey.java.net/documentation/latest/getting-started.html, zugegriffen am 31.03.2015.

[20] *LayoutInflater*. http://developer.android.com/reference/android/widget/BaseAdapter.html, zugegriffen am 10.4.2015.

[21] *Layouts*. http://developer.android.com/guide/topics/ui/declaring-layout.html, zugegriffen am 28.4.2015.

[22] *Lesson: Annotations*. https://docs.oracle.com/javase/tutorial/java/annotations/, zugegriffen am 10.4.2015.

[23] *MediaStore*. http://developer.android.com/reference/android/provider/MediaStore.html, zugegriffen am 17.03.2015.

[24] *Menus*. http://developer.android.com/guide/topics/ui/menus.html, zugegriffen am 10.03.2015.

[25] *PackageManager*. http://developer.android.com/reference/android/content/pm/PackageManager.html, zugegriffen am 17.03.2015.

[26] *Personen mit Nutzung tragbarer Geräte für den mobilen Internetzugang ausserhalb des Haushalts oder der Arbeit*. http://www.statistik.at/web_de/statistiken/informationsgesellschaft/ikt-einsatz_in_haushalten/022210.html, zugegriffen am 30.1.2015, Stand 21.10.2014, Altersgruppe 16 bis 74 Jahre.

[27] *Picasso*. http://square.github.io/picasso/, zugegriffen am 10.4.2015.

[28] Rest api. https://github.com/excilys/androidannotations/wiki/Rest-API, zugegriffen am 10.4.2015.

[29] *Support Library*. http://developer.android.com/tools/support-library/index.html, zugegriffen am 13.02.2015.

[30] *Support Library Features*. https://developer.android.com/tools/support-library/features.html, zugegriffen am 10.4.2015.

[31] *What is Object/Relational Mapping?* http://hibernate.org/orm/what-is-an-orm/, zugegriffen am 13.2.2015.

[32] *What's a Windows Runtime app?* https://msdn.microsoft.com/library/windows/apps/dn726767.aspx, zugegriffen am 14.02.2015.

[33] Wikipedia Autoren. *EclipseLink*. http://de.wikipedia.org/wiki/EclipseLink, zugegriffen am 31.03.2015.

[34] Wikipedia Autoren. *Gradle*. `http://de.wikipedia.org/wiki/Gradle`, zugegriffen am 13.02.2015.

[35] Wikipedia Autoren. *Hybrid-App*. `http://de.wikipedia.org/wiki/Hybrid-App`, zugegriffen am 14.02.2015.

[36] Wikipedia Autoren. *Internet Media Type*. `http://de.wikipedia.org/wiki/Internet_Media_Type`, zugegriffen am 31.03.2015.

[37] Wikipedia Autoren. *Java API for RESTful Web Services*. `http://de.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services`, zugegriffen am 31.03.2015.

[38] Wikipedia Autoren. *Java Persistence API*. `http://de.wikipedia.org/wiki/Java_Persistence_API`, zugegriffen am 31.03.2015.

[39] Wikipedia Autoren. *JavaScript Object Notation*. `http://de.wikipedia.org/wiki/JavaScript_Object_Notation`, zugegriffen am 14.01.2015.

[40] Wikipedia Autoren. *Kontextmenue*. `http://de.wikipedia.org/wiki/Kontextmen%C3%BC`, zugegriffen am 10.03.2015.

[41] Wikipedia Autoren. *MySQL*. `http://de.wikipedia.org/wiki/MySQL`, zugegriffen am 31.03.2015.

[42] Wikipedia Autoren. *Programmierschnittstelle*. `http://de.wikipedia.org/wiki/Programmierschnittstelle`, zugegriffen am 13.02.2015.

[43] Wikipedia Autoren. *SOAP*. `http://de.wikipedia.org/wiki/SOAP`, zugegriffen am 14.01.2015.

[44] Thomas Bayer and Dirk M Sohn. Rest web services. 2007. `http://www.thomas-bayer.com/resources/rest/rest_webservices.pdf`, zugegriffen am 30.1.2015.

[45] Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`, zugegriffen am 10.4.2015.

[46] J. Franks et al. *HTTP Authentication: Basic and Digest Access Authentication*, 1999. `http://www.ietf.org/rfc/rfc2617.txt`, zugegriffen am 10.4.2015.

[47] E Gamma, R Helm, R Johnson, and J Vlissides. *Design patterns: elements of reusable object-oriented software*. 1994.

[48] Google. *Material Design*. `http://www.google.com/design/spec/material-design/introduction.html`, zugegriffen am 10.4.2015.

[49] Christoph Henkelmann. *Mobile Apps - Nativ vs. Hybrid*. `https://www.seosweet.de/blog/2013/08/01/mobile-apps-nativ-vs-hybrid-oder-wer-billig-kauft-kauft-zwei-mal`, zugegriffen am 10.4.2015.

[50] Carsten Howitz. *What is Three Tier Architecture and Why Do You Need It*. `http://blog.simcrest.com/what-is-3-tier-architecture-and-why-do-you-need-it/`, zugegriffen am 14.1.2015; übersetzt aus dem Englischen.

[51] individuapp.com. *Lebenszyklus einer Activity.* http://individuapp.com/de/android-kurs/die-klassen-activity-und-intent, zugegriffen am 14.02.2015.

[52] Oracle. *Enum CascadeType.* https://docs.oracle.com/javaee/6/api/javax/persistence/CascadeType.html, zugegriffen am 31.03.2015.

[53] Oracle. *What Is a Session Bean?* http://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html, zugegriffen am 10.4.2015.

[54] Stefan Tilkov. Rest - der bessere web service? 2009. http://www.thomas-bayer.com/resources/rest/rest_webservices.pdf, zugegriffen am 14.2.2015.

[55] Christian Ullenboom. *Java 7 - Mehr als eine Insel.* http://openbook.rheinwerk-verlag.de/java7/1507_13_001.html, zugegriffen am 14.2.2015.

[56] Wikibooks-Bearbeiter. *Googles AndroidActivities.* Wikibooks, Die freie Bibliothek. http://de.wikibooks.org/wiki/Googles_Android/_Activities, zugegriffen am 14.02.2015.

Abbildungsverzeichnis