Hochschule Anhalt - Fachbereich Informatik und Sprachen

B. Sc. Angewandte Informatik /
Medieninformatik

Bachelorarbeit

Entwicklung einer Kochbuch-App für Android-Smartphones

Erstellt von:

Tino Schmittat, 4052082
tino.schmittat@student.inf.hs-anhalt.de

Erstgutachter: Prof. Dr. Carôt
Zweigutachter: Prof. Dr. Bade

Eingereicht am: 07.05.2015

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst habe. Ich versichere, dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahren gewesen ist.

(Unterschrift)

Inhaltsverzeichnis

1 Einleitung

Seit Anbeginn ihrer Existenz begleiten verschiedene Philosophien und Lebensweisen die Menschheit. Von der Religion bis zum Wirtschaftsprinzip sind unzählige Gedankenkonstrukte vorhanden. Dies kann bis in unsere heutige Gesellschaft beobachtet werden. Jedoch variieren viele dieser Gedankenkonstrukte in Hinsicht auf Quantität und Qualität, wie auch bei den Vertretern des Veganismus.

Viele dieser versuchen möglichst nach dem Motto: „Was ich nicht will, das du mir tust, tu ich dir nicht!“ zu leben. Nur im Unterschied zu vielen anderen Gedankenkonstrukten, sehen die Vertreter des Veganismus sich nicht als überlegene Lebensform, sondern als eben-würdiges Mitgeschöpf und beziehen andere Lebewesen, wie die Tiere, mit ein in ihre Betrachtungsweisen.

Für diese steht dabei außer Frage, dass ein Unterschied zwischen Mensch und Tier besteht. Doch gibt es viele Tiere, die dem Menschen überlegene Sinnesorgane besitzen, wie Fledermäuse mit ihrem Gehör oder Delphine mit ihrem Sonar.

Eine Grundauffassung ist, dass wir alle ein Bedürfnis teilen, das Bedürfnis, Schmerz zu vermeiden. Und um diesen Schmerz und das damit verbundene Leid zu vermeiden beziehungsweise es möglichst zu minimieren, vermeiden die meisten Veganer Produkte oder Lebensmittel, die aus oder mit Hilfe von Lebewesen produziert wurden.

Aus der Sicht des Veganismus ist jede Tötung eines Lebewesens, aus Profitgründen Mord und jedes festhalten oder ausbeuten eines Lebenswesens gegen deren freien Willen falsch. So ist in dessen Augen ein Stück Fleisch nicht einfach nur ein Stück Fleisch, sondern Teil eines Leichnams der eigens aus Profitgründen erzeugt wurde. Aber auch Milchprodukte sind für dem Veganismus keine positiven Erzeugnisse. Für diese müssen Kühe dauerhaft künstlich schwanger gehalten werden. Meist werden deren Kälber nach dem austragen geschlachtet oder zumindest von der Mutter getrennt und mit minderwertigen Erstsatzprodukten gefüttert.

Es findet nicht nur ein Verzicht oder Boykott von direkt erzeugten tierischen Produkten statt, sondern es wird auch nach Möglichkeit versucht, darauf zu achten, keine Kleidung zu kaufen, die tierische Produkte enthält. Dies setzt sich auch beim Kauf von Pflegeprodukten fort. Dabei versuchen Veganer, es zu vermeiden, Pflegeprodukte zu kaufen, die mit Tierversuchen erprobt wurden.

Bis zum heutigen Tag teilen ca. 1% der deutschen Bevölkerung diese Auffassung [VEB14]. Pro Tag sollen ca. 200 Deutsche dazukommen. Damit stellt der Veganismus eine der am stärksten wachsenden Märkte dar. Dieses Wachstumspotential haben auch schon einige internationale Lebensmittelkonzerne, wie McDonalds feststellen können [STE14].

Doch ist die vegane Lebensweise für viele Menschen noch etwas Unbekanntes oder wird von ihnen als Unsinn abgetan. Dies schlägt sich im Angebot sämtlicher Produkte nieder. Angefangen vom Besuch in einem Restaurant, bis hin zum Schuh- oder Softwarekauf.

2 Problem und Ziel

Wie bei den meisten Produkten ist auch im Bereich Software kein großes Sortiment vorhanden, welches auf die Bedürfnisse vegan lebender Menschen angepasst ist. Wobei zahlreiche Systeme aus dem Bereich „Open Source“ existieren, die einen ersten Schritt in Richtung Unabhängigkeit und Freiheit darstellen. Dies stellt eine Parallele zum Veganismus dar. Deshalb ist es naheliegend, dass sich Betriebssysteme, wie Ubuntu Linux oder auch Web-Browser, wie Mozilla Firefox, einer großen Beliebtheit bei alternativ denkenden Menschen erfreuen.

Aber auch im Bereich Mobilgeräte existieren unabhängige Projekte, so wie Android. Das Nutzen dieser Software stellt jedoch kein „muss“ dar, sondern viel mehr ein „kann“. So gibt es unabhängig von der Plattform oder dem Medium nur wenig auf Veganer zugeschnittene Software. Dies zeigt sich zum Beispiel an dem Mangel von digitalen veganen Kochbüchern. Hingegen existieren für den omnivoren[1] Menschen umfangreiche Rezeptsammlungen in digitaler Form - wie zum Beispiel die Online-Plattform „www.chefkoch.de“. Diese bietet ein umfangreiches Sortiment an Rezepten an. Darunter befinden sich auch vegane Rezepte, die aber leider sehr begrenzt sind und schlecht kategorisiert wurden.

Daneben existieren noch unzählige Blogs und kleinere Webseiten, die jeweils ein paar Rezepte anbieten. Dies gilt nicht nur für den deutschsprachigen Raum sondern auch im internationalen Bereich. Dort existieren ebenfalls unzählige kleinere Plattformen, die im Aufbau wie im Inhalt geradezu identisch sind. Aber es fehlt diesen nicht nur an Umfang sondern auch an Interaktivität. Es ist den Nutzern nicht ohne weiteres möglich, selbstständig Rezepte einzureichen oder eigene Bewertungen abzugeben.

Dieses Problem soll hier gelöst werden. Der Fokus wird dabei auf den mobilen Geräten liegen und Android soll als Plattform dienen - zum einen wegen seiner hohen Verbreitung und zum andern, weil es sich um eine „Open Source“-Plattform handelt.

Die zu entwickelnde Android-Anwendung soll grundlegend über eine Rezeptdatenbank verfügen, deren Informationen sie ausgibt. Zusätzlich dazu soll ein System entwickelt werden, welches Vorschläge, abhängig von der Eingabe, ausgibt. Bei diesem System soll der Nutzer angeben, welche Lebensmittel er zur Verfügung hat und es sucht die möglichst passenden Kochideen aus der Datenbank heraus. Es soll auch über ein Bewertungssystem für die Rezepte verfügen und dazu soll noch die Möglichkeit für den Nutzer bestehen, eigene Rezepte hochzuladen.

1 Allesesser

3 Grundlagen

In diesem Abschnitt soll gezeigt werden, was Android ist und wie es funktioniert. Es wird beschrieben, welche Funktionen Android hat und welche Möglichkeiten in der Entwicklung mit Android bestehen.

3.1 Android

Android ist ein Betriebssystem beziehungsweise eine Plattform, welche speziell für mobile Geräte wie Netbooks, Tablets und Smartphones entwickelt wurde. Unter der Führung des Google Konzerns wurde Android von der Open Handset Alliance veröffentlicht und wird bis heute von ihr weiterentwickelt [TKH12].

Im Jahr 2007 erschien das erste Smartphone mit Android als Betriebssystem, sowie 2011 die ersten Tablets.

Heutzutage findet man Android auch in vielen weiteren Systemen, wie zum Beispiel in Fernsehsticks, Smartwatches oder auch in Google Glass. Außerdem gibt es Anstrengungen Android in Pkws als Standard Infotaimentsystem zu implementieren.

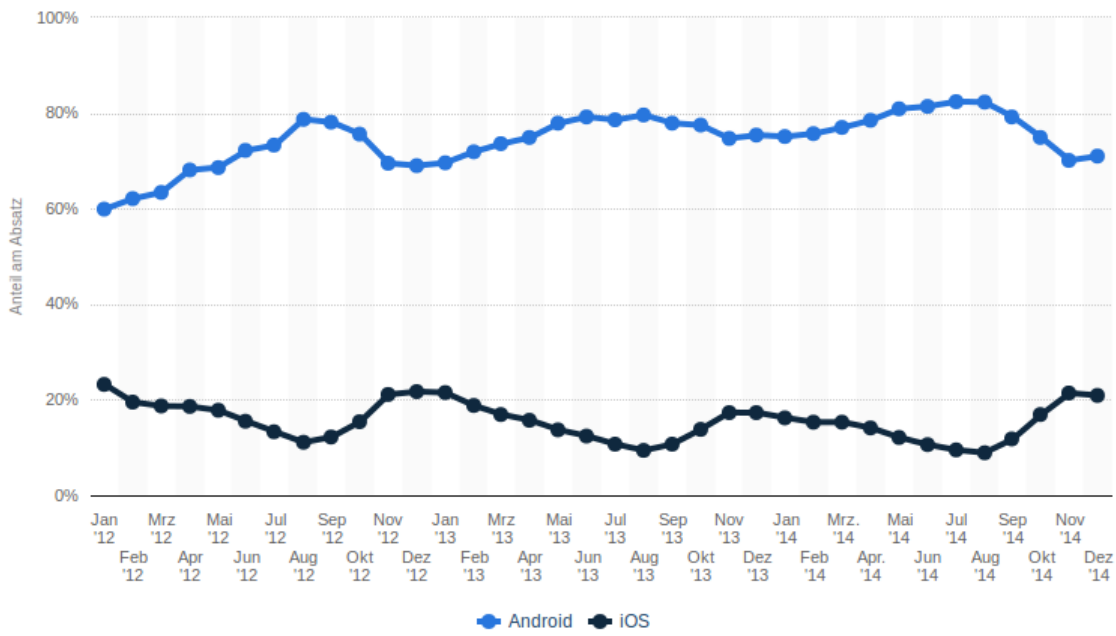


Abbildung 1: Marktanteile Android / iOS bei Smartphones [STA15]

Android ist ein häufig genutztes Betriebssystem, im Bereich von Mobilgeräten sogar das Verbreitetste. Der Marktanteil beträgt 60 bis 80 Prozent des weltweiten Marktes. Laut Google sollen bereits bis zum Jahr 2013 eine Milliarde Geräte aktiviert worden sein [GOL14].

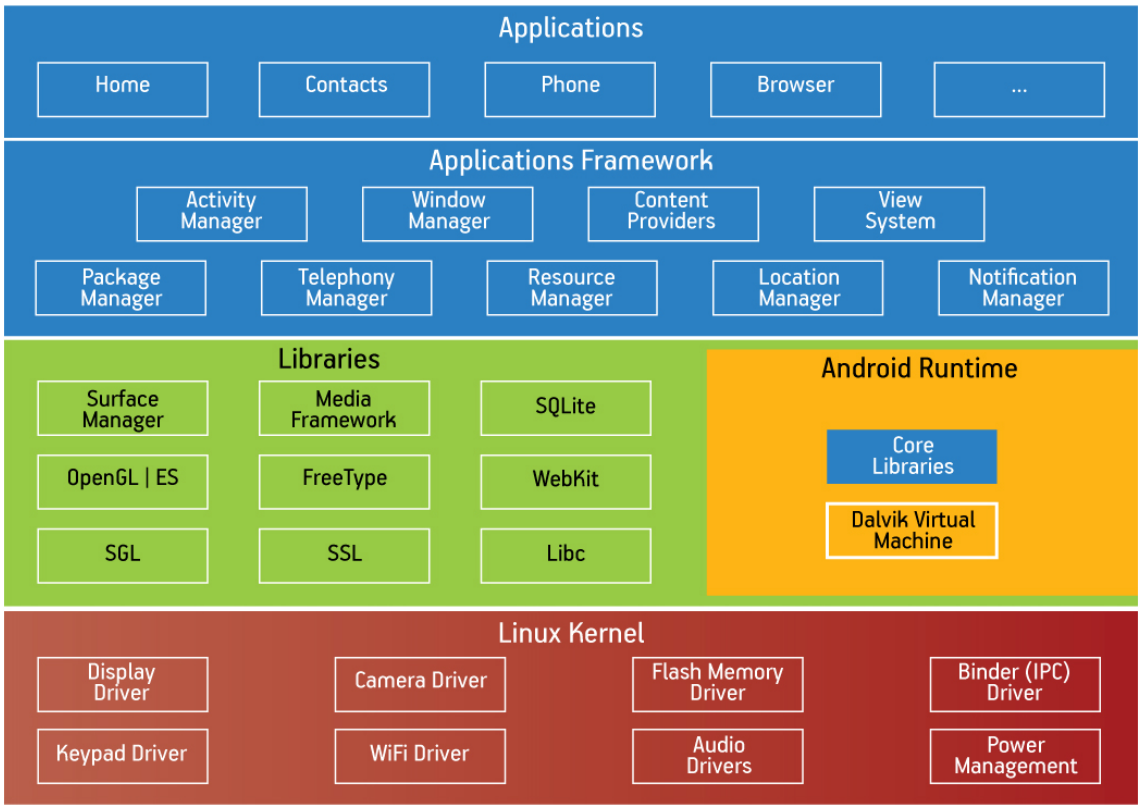


Abbildung 2: Schematischer Aufbau der Android-Plattform [MGO10]

3.1.1 Applications

Die Applications bilden die bei Android von Werk aus mitgelieferten Anwendungen, wie eine SMS-, eine E-Mail-, eine Kontakte und eine Browseranwendung.

3.1.2 Application Framework

Das Application Framework bildet eine Art Rahmen für die Anwendungsarchitektur. Dadurch soll eine einheitlichere Anwendungsentwicklung ermöglicht werden. Diese soll es erlauben, Komponenten wiederzuverwenden und die Struktur für Entwickler zu vereinfachen.

Bei der Entwicklung von Android-Apps werden diverse Dienste und Funktionen vom Application Framework bereitgestellt [TKH12].

- ***Views***, beschreiben alle Arten von Elementen, die es dem Nutzer erlauben mit der Anwendung zu agieren. Die Views in Android beinhalten einfachere Elemente wie Buttons, Listviews und Textviews. Android ermöglicht auch das aus Views komplexere Views gebaut werden können.
- Der ***Content Provider*** erlaubt es auf Daten anderer Anwendungen zuzugreifen. Des Weiteren ist es auch möglich mit Hilfe des Content Providers, anderen Anwendungen, Daten zur Verfügung zu stellen.

- In einer Android-App bietet der ***Resource Manager*** die Möglichkeit, auf nicht fest im Code integrierte Anwendungselemente zuzugreifen. Darunter zählen zum Beispiel Layoutdateien, Grafiken und XML[2]-Dateien.
- Der ***Notification Manager*** bietet den Android-Anwendungen die Möglichkeit, auf die Actionbar oder auf die Android-Statusleiste zuzugreifen.
- Mit Hilfe des ***Activity Managers*** wird der Lebenszyklus der Anwendung verwaltet und gesteuert.

3.1.3 Bibliotheken

Android beinhaltet eine Reihe von C/C++ Bibliotheken, die von verschiedenen Komponenten genutzt werden. Dem Nutzer wird damit die Möglichkeit geboten, indirekt auf diese zuzugreifen. Zu den Bibliotheken gehören:

- Die ***SystemC Library*** ist eine Implementierung der Standard C–Bibliothek von der Berkeley Software Distribution. Diese wurde speziell auf die Anforderungen mobiler Geräte angepasst.
- Die ***Media Libraries*** basieren auf den OpenCORE-Bibliotheken von PacketVideos. Diese bieten die Möglichkeit der Wiedergabe und Aufnahme von Audio-, Video- und Grafikdateien in gängigen Formaten wie MPEG4, MP3, AAC, JPG und PNG.
- Der ***Surface Manager*** verwaltet die Bildschirmzugriffe und fügt die 2D und 3D Ausgaben zu einem Gesamtbild zusammen.
- ***LibWebCore*** ist eine Rendering Engine für Webinhalte. Dieses basiert auf der quelloffenen WebKit Engine.
- ***SGL*** ist die zugrundeliegende 2D-Grafikbibliothek.
- Die ***3D libraries*** basieren auf OpenGL ES und sind eine 3D - Grafikbibliothek. Diese nutzt die Hardwarebeschleunigung, wenn eine solche zur Verfügung stehen sollte.
- Bei ***FreeType*** handelt es sich um eine Rendering Engine für Bitmap- und Vektorschriften.

3.1.4 Android Runtime

Android-Apps werden von einer virtuellen Maschine Namens Dalvik ausgeführt. Dalvik wurde ausschließlich für mobile Geräte entwickelt und unterscheidet sich in einigen Punkten von einer Standard-Java-Laufzeitumgebung. Dalvik zeichnet sich durch ein eigenes Bytecodeformat und einen eigenen Befehlssatz aus. Es ist registerbasiert und .class-Dateien werden mit einem Tool namens dx in Dalvik Executables umgewandelt.

Dalvik wurde jedoch nicht auf geringen Speicherverbrauch optimiert. Es spielte bei der Entwicklung das Ausführen von mehreren virtuellen Maschinen eine große Rolle. Jede

2 Extensible Markup Language

Android-App besitzt ihre eigene Instanz auf der Dalvik virtual maschine. Dies soll Stabilität und Sicherheit erhöhen. Abgesehen von der Dalvik virtual maschine, besitzt das Android Runtime eine zweite Komponente namens Core Libaries. Diese beinhaltet Java Standard-Bibliotheken wie java.io oder java.lang. Dabei verwendet Android einen Teil der freien Java-Implementierung Apache Harmony.

3.1.5 Linux-Kernel

Android basiert auf einem Linux-Kernel, ist aber keine klassische Linux-Distribution. Dieser Kernel ist für die Speicherverwaltung und Prozessverwaltung verantwortlich. Des weiteren stellt dieser eine Schnittstelle für den Bereich Multimedia und Netzwerkkommunikation dar. Und auch die Gerätetreiber werden vom Linux-Kernel bereitgestellt.

3.1.6 Entwicklungswerkzeuge

Um Android-Apps entwickeln zu können, wird eine Entwicklungsumgebung, das Android Software Development Kit und das Java Development Kit benötigt. Hierbei kann mit Android Development Tools in Verbindung mit Eclipse oder dem von Google entwickelten Android Studio gearbeitet werden.

3.2 Hardwareressourcen

Android setzt diverse Hardwarekomponenten voraus, beziehungsweise besitzt die Möglichkeit diese anzusprechen und zu verwenden:

- Die ***Telefondienste*** werden hauptsächlich dafür verwendet, die von Telefonen allgemein bekannten Funktionen zu bieten, wie Telefonate oder Mitteilungen anderer Art zu versenden oder zu empfangen. Diese Dienste sind in Android primär implementiert, da Android typischerweise in Mobiltelefonen Verwendung findet.

- Der ***Speicher*** ist ein primärer Teil eines Android-Telefones, da dort dauerhaft Daten gespeichert werden sollen. Dieser bietet die Möglichkeit, zum Beispiel Datenbanken oder auch Kontakte zu speichern. Abgesehen vom primären Speicher, der meistens relativ klein ausfällt, gibt es noch die optionale Möglichkeit der Verwendung einer Secure Digital Memory Card auf die Android auch unter anderem die Möglichkeit des Zugriffs bietet.

- Im Bereich ***Multimedia*** besitzt Android die Fähigkeit Video, Bild und Ton wiederzugeben oder zu erzeugen. Die Fähigkeit dazu kann individuell nach Geräte abgefragt werden.

- Beim ***Netzwerk*** bietet Android die Möglichkeit auf mehreren Schichten zu kommunizieren. Android-Geräte sind allgemein dafür ausgelegt, internetfähig zu sein.

- Android besitzt die Fähigkeit ***Sensoren, beziehungsweise Ortungsdienste*** zu verwenden. Diese Fähigkeit reicht von der Benutzung des GPS, bis zum Zugriff auf Daten von physikalischen Kräften, die auf das jeweilige Gerät einwirken.

3.3 Komponenten

Android verfügt über eine Vielzahl von individuellen Komponenten. Dabei existieren eigene Programmstrukturen wie Fragments oder Activities.

3.3.1 Activitys

Eine der wichtigsten Komponenten in der Programmierung mit Android sind die Activitys. Diese repräsentieren zumeist eine Aktion wie einen Anruf zu empfangen, eine SMS zu versenden oder das heutige Wetter anzuzeigen. Jeder Activity steht im Normalfall ein Fenster mit Steuerungselementen zur Verfügung, wenn auch nicht zwingend. In Android ist es auch möglich, dass die Activity in einem kleinen Fenster in einer anderen Activity ausgeführt wird.

Eine App beinhaltet meistens mehrere Activitys. Der Benutzer bestimmt dabei zumeist die Reihenfolge den Aufrufs über seine Eingaben. Es besteht jedoch die Möglichkeit, die Reihenfolge in Ausnahmen per Programmlogik vorzugeben.

Jede App besitzt eine Mainactivity. Diese wird beim Programmstart ausgeführt und wird meistens als Hauptnavigationsseite verwendet.

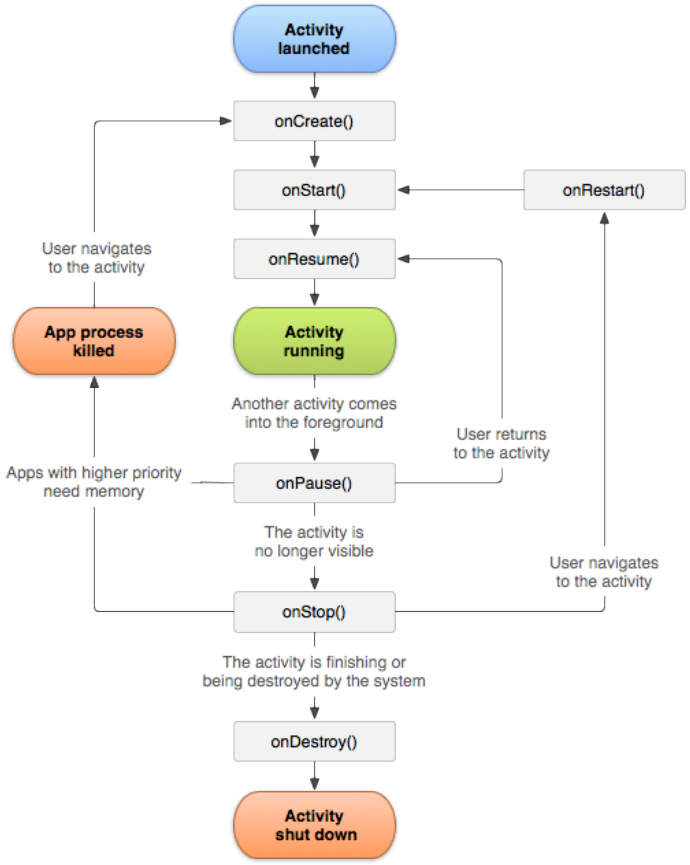


Abbildung 3: Activity Lifecycle [DAA15]

Activitys können 3 Zustände annehmen:

- Activity ist ***aktiv und im Vordergrund***. Dabei befindet sich der Status zwischen *onResume()* und *onPause()*. In diesem Status kann der Nutzer mit der Activity über die Steuerungselemente kommunizieren.
- Die Activity ist ***sichtbar, aber nicht im Vordergrund***. Hierbei befindet sich die Activity im Status *onPause()*. Dabei kann die Activity unter Umständen nur noch einen Teil das Bildschirms ausfüllen oder transparent angezeigt werden.
- Activity ist ***nicht sichtbar und befindet sich im Hintergrund***. Während dessen befindet sich die Activity im Status *onStop()*.

Es bleiben Variablen und Instanzen im Speicher erhalten, wenn sich die Activity im Status *onStop()* oder *onPause()* befindet. Sollte jedoch eine Notsituation im *onPause()* Status auftreten, kann die Activity vom System zerstört werden. Im *onStop()* Status werden die Daten bereits darauf vorbereitet.

3.3.2 Intents

Im Grunde stellen Intents Nachrichtenobjekte dar. Diese sollen hauptsächlich die Kommunikation zwischen Activitys ermöglichen. Es gibt drei Möglichkeiten über einen Intent zu kommunizieren:

- ***Beim starten einer Activity.*** Bei dem Erzeugen einer Instanz einer Activity, wird die Methode *startActivity()* mit der Instanz des Objektes des Intents aufgerufen. Diese Instanz kann dann Daten für die neu erzeugte Activity enthalten und daraus extrahiert beziehungsweise verwendet werden.
- ***Mit dem starten eines Services.*** Diese Komponente führt Hintergrundoperationen ohne Nutzeroberfläche aus. Meist wird diese für einmalige Operationen wie den Download einer Datei verwendet. Analog zur Methode „beim starten einer Activity“, wird hier ein Intent-Objekt erzeugt und verwendet. Jedoch wird hier die Methode *startService()* verwendet.
- ***Beim Empfang durch den Broadcast.*** Dieser ermöglicht es einer App, Nachrichten von anderen Apps zu empfangen. Android liefert verschiedene Broadcasts bei System-Events, wie beim starten des Standby-Modus oder dem beenden des Ladevorgangs. Hierbei werden die Methoden *sendBroadcast()*, *sendOrderedBroadcast()* und *sendStickyBroadcast()* verwendet.

Dabei unterteilen sich Intents in zwei Typen:

- ***Explizite Intents*** zeichnen sich durch die Benennung der Zielkomponenten aus. Diese werden üblicherweise innerhalb einer App verwendet, da alle Informationen über das Ziel, wie zum Beispiel der Klassennamen der Activity, bekannt sind. Dies kann zum Beispiel beim Ausführen einer neuen Activity mit der Übergabe benutzerdefinierter Daten sein.
- ***Implizite Intents*** hingegen besitzen keine spezifischen Komponenten, wie die expliziten Intents. Diese versuchen eine möglichst allgemeine Aktion zu erzeugen, was es für externe Apps leichter machen soll, mit diesen zu arbeiten.

Dies kann zum Beispiel das Laden eines Galeriebildes in den Telefonspeicher sein.

Damit ein impliziter Intent verarbeitet werden kann, muss ein sogenannter Intent-Filter erzeugt werden. Dieser muss wiederum in der Manifestdatei, die näher im Abschnitt 3.3.3 Manifest besprochen wird, eingetragen werden. Wurde kein Intent-Filter vermerkt so kann eine App nur explizite Intents verarbeiten.

3.3.3 Manifest

Jede Android-App benötigt eine Datei mit dem Namen AndroidManifest.xml. Diese Datei repräsentiert die grundsätzlichen Informationen der Android-App. Ohne sie kann eine Android-App nicht ausgeführt werden. Darin wird vermerkt, welches Java-Paket verwendet wurde oder auch welche Activitys aufgerufen werden können. Es wird aber auch festgehalten, auf welche Bibliotheken zugegriffen wird oder auf welche Hardwarekomponenten die Android-App zugreifen darf.

3.3.4 Widgets

Android ermöglicht es, sogenannte Widgets zu implementieren und diese dann über den Home-Bildschirm oder an der Systemleiste anzuzeigen. Dies soll es wiederum dem Nutzer erleichtern, an App-Inhalte zu gelangen. Es handelt sich hierbei um eine Art von Informationstafel, die meist für App bezogene Nachrichten verwendet wird.

3.3.5 Fragmente

Eine weiter wichtige Komponente stellen die Fragmente dar. Diese beschreiben eine Art Verhalten oder einen bestimmten Teil einer Nutzeroberfläche. Sie sind ab Android 3.0 verfügbar. Es können mehrere Fragmente in einer Activity verwendet werden. Diese könnte man als eine Art eigenständigen, modularen Teil einer Activity betrachten. Mit eigenem Lebenszyklus und eigenen Eingaben.

Jedes Fragment muss in eine Activity eingebettet sein. Das Fragment verknüpft seinen Lebenszyklus mit dem Lebenszyklus der verknüpften Activity. Also wenn die Activity gestoppt wird, so werden auch alle verknüpften Fragmente gestoppt.

Um eine höher Dynamik im Umgang mit verschiedenen Displaygrößen zu erreichen, wurden die Fragmente eingeführt. Sie sollen es unter anderem ermöglichen auf größeren Displays mehrere Oberflächen zu kombinieren, die sonst einzeln angezeigt werden müssten.

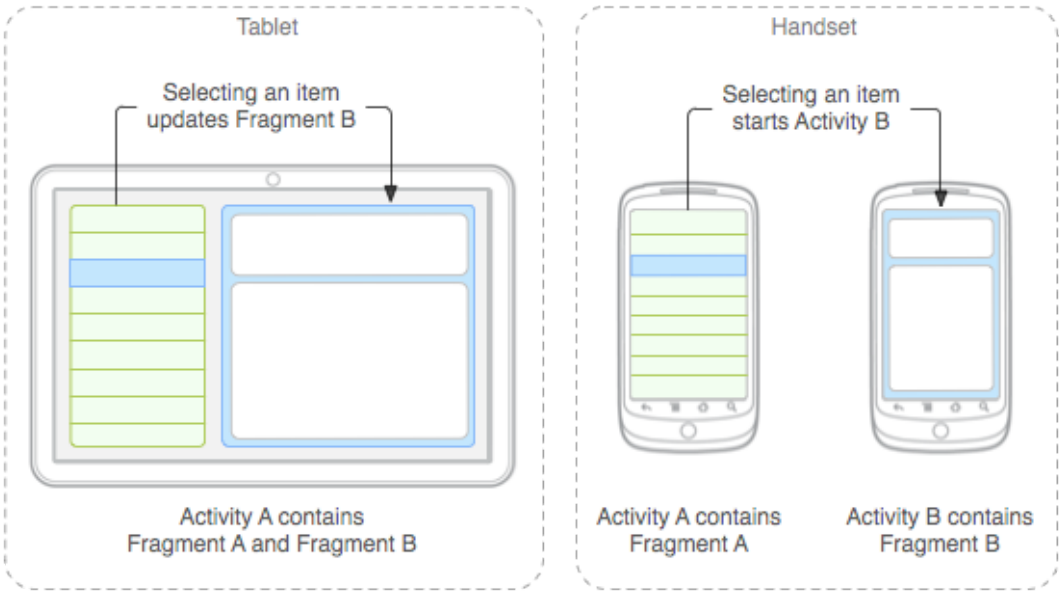


Abbildung 4: Fragmente [DAF15]

Fragmente sollten eine modulare und wieder benutzbare Komponente darstellen. Dabei beschreibt jedes Fragment eine Layout, welches in Abschnitt 3.4.1 Layout und Views näher besprochen wird. Diese haben zusätzlich einen eigenen Lebenszyklus und besitzen darüber hinaus die Möglichkeit in mehreren Activities verwendet zu werden.

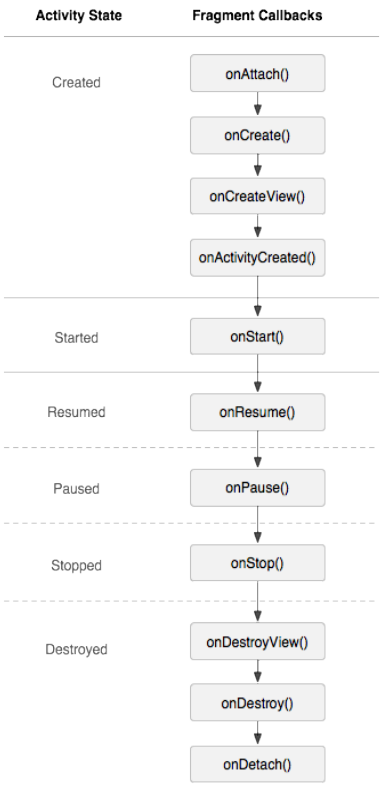


Abbildung 5: Fragment Lebenszyklus [DAF15]

Der Lebenszyklus eines Fragmentes ähnelt sehr dem einer Activity. Wie bei einer Activity existieren drei Zustände:

- Im Status ***Resumed*** ist das Fragment in der Activity sichtbar und fokussiert.
- Ist eine andere Activity im Vordergrund, aber die Activity des Fragmentes noch sichtbar, so ist sie ***Paused***.
- ***Stopped*** ist ein Fragment entweder, wenn die Activity gestoppt wurde oder wenn das Fragment von der Activity entfernt wurde.

3.3.6 Prozesse

Die Basis für Android bildet der Linux-Betriebssystemkern. Dieser bietet die Möglichkeit des Multitaskings. Dabei steuert der Kern die Prozesse so, dass Prozesse, die zu viel Rechenzeit in Anspruch nehmen, in den Hintergrund gedrängt werden und andere Prozesse abgearbeitet werden können.

Android nutzt eine „importance hierarchy“, basierend auf Ausführung und Status des Prozesses. Prozesse mit der niedrigsten „importance“ werden dabei als erstes eliminiert und dann die Nächste mit der niedrigsten „importance“, solange eine Freigabe von neuen Systemressourcen notwendig ist.

Insgesamt umfasst die „importance hierarchy“ fünf Level, der Wichtigkeit nach hierarchisch aufgelistet, von sehr wichtig bis unwichtig:

- Ein ***Vordergrundprozess***, ist ein Prozess den der Nutzer aktuell ausführt.
- ***Sichtbare Prozesse*** zeichnen sich darin aus, dass sie keine Vordergrundprozesse sind, aber das sie trotzdem beeinflussen können, was der Nutzer auf der Oberfläche sieht.
- ***Service Prozesse*** werden nicht direkt vom Nutzer wahrgenommen und übernehmen Aufgaben, die eine eher indirekte Wichtigkeit besitzen, wie das Abspielen von Hintergrundmusik.
- ***Hintergrundprozesse*** haben keinen direkten Einfluss auf die Wahrnehmung des Nutzers, wie eine Activity, die momentan nicht für den Nutzer sichtbar ist. Dabei kann das System diese Prozesse jederzeit „killen“, um benötigten Speicher freizugeben.
- Ein ***leerer Prozess*** beinhaltet keine aktiven Anwendungsdaten. Er dient lediglich dazu das Speichern von Daten vorzubereiten und die Startzeit diverser Komponenten zu beschleunigen.

Es werden alle Android-Apps in einer eigenen Laufzeitumgebung aufgerufen, als eigener Linux-Prozess, so dass eine Fehlfunktion der Android-App das Betriebssystem des Gerätes nicht zum Absturz bringen kann.

Im Rahmen des Multitaskings kann ein Prozess innerhalb einer App auch als eigener Thread verarbeitet werden. Diese sind bereits in der zugrundelegenden Java Klassenbibliothek vorhanden und werden von der Komponente Apache Harmony

erweitert. Apache Harmony ermöglicht es, mehrere Prozesse gleichzeitig nebeneinander auszuführen.

Android unterstützt dabei asynchrone Tasks, welche neue Prozesse im Hintergrund erzeugen und sich auch gut für komplexere Aufgaben eignen. Dabei beinhaltet asynchrone Tasks die Methode *doInBackground(),* in welcher Hintergrundaufgaben ausgeführt werden können. Die Methode *OnPostExecute()* liefert das dazugehörige Ergebnis.

3.3.7 Services

Bei Aufgaben, die im Hintergrund auch ohne aktive Nutzeroberfläche ausgeführt werden sollen, bietet Android die Möglichkeit Services zu nutzen. Mit diesen kann zum Beispiel sichergestellt werden, dass ein Musikplayer auch Musik abspielt, wenn der Nutzer nicht aktiv in der Musikplayer-App aktiv ist. Diese werden in zwei Kategorien unterteilt:

- ***Gestartete Services*** werden mit *startService()* aufgerufen. Diese werden gestoppt, wenn die erzeugende Komponente gestoppt wurde. Normalerweise sollte diese Serviceart für einzelne Operationen genutzt werden.
- Aufgerufen werden ***gebundene Services*** mit *bindService()*. Dieser bietet eine Art von Server-Client-Schnittstelle. Dabei können Anfragen verschickt und Daten abgefragt werden. Der Service wird so lange ausgeführt, wie er an eine Komponente gebunden ist. Er kann aber auch an mehrere Komponenten gleichzeitig gebunden sein.

3.3.8 Content Provider

Die Hauptaufgabe der Content Providers besteht in der Verwaltung von Daten. Er bietet darüber hinaus die Möglichkeit, eine gewisse Datensicherheit bereitzustellen.

Um Zugriff auf die Daten zu erlangen, muss ein ContentResolver-Objekt erzeugt werden. Dieser ermöglicht dann eine indirekte Kommunikation zum Content Provider. Das Objekt ist dann in der Lage die Daten zurückzugeben und zu empfangen.

Dabei nutzt Android den Content Provider beim Verwalten von Audiodateien, Videodateien und Kontaktdaten.

3.3.9 Datenbanken

Um in Android nachhaltig Daten speichern zu können und auf diese nach Beendigung der jeweiligen App weiterhin zugreifen zu können, wird eine Datenbankanwendung benötigt. Hierbei bietet Android den Zugriff auf das bereits in Android eingebettet SQLite. Bei SQLite handelt es sich um eine von Hwaci - Applied Software Research entwickelte Open Source Datenbanklösung. Dabei arbeitet SQLite mit Befehlen auf Basis von SQL[3], wie eine normale relationale Datenbank. SQLite ist dabei Serverlos und schreibt, beziehungsweise liest direkt Daten vom oder in den Speicher.

3 Structured Query Language

Dieses ist in jedem Android-Gerät von Werk aus vorinstalliert. Dabei wird von der SQLite Datenbank weder eine Installationsprozedur oder Datenbankverwaltung benötigt. Android bietet somit eine vorkonfigurierte Version von SQLite, die automatisch von Android konfiguriert wird. Der Nutzer muss lediglich im Kontext der jeweiligen App, die Daten per SQL-Befehl steuern.

3.4 Benutzeroberflächen

Bei den Benutzeroberflächen bietet Android viele Gestaltungsmöglichkeiten. Jedoch nutzt es dafür im Normalfall XML-Dateien, in denen die jeweiligen Strukturen hierarchisch festgelegt werden. Die dort erzeugten Komponenten können aber auch auf Quelltextebene manipuliert und erzeugt werden.

3.4.1 Layouts und Views

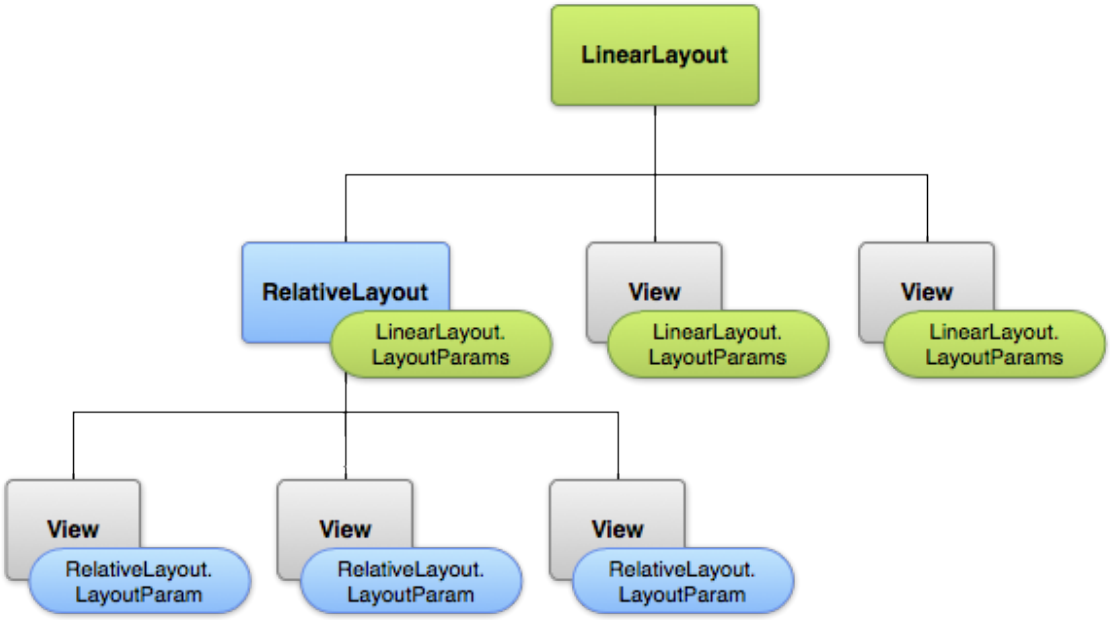


Abbildung 6: Beispielhafte Visualisierung von Layoutparametern [DAL15]

Der Grundbaustein einer jeden Android Benutzeroberfläche sind die Layouts. Diese definieren die visuelle Struktur der Nutzeroberfläche. Innerhalb der Layouts können Elemente erzeugt werden, die in sogenannte Views oder Viewsgroups eingeteilt werden. Dazu gehören zum Beispiel Textfelder oder Buttons. Jede View oder Viewgroup kann individuelle Attribute enthalten. Dazu gehören zum Beispiel Ids. Dabei besitzt die Id die wichtige Aufgabe der Identifikation. Worüber die View beziehungsweise Viewgroup auf Codeebene angesprochen und manipuliert werden können. In der XML-Datei werden die Attribute nach dem Schema „layout_irgendetwas“ definiert. Normalerweise werden alle Views in anderen Views oder Viewsgroups verschachtelt.

Jede Viewgroup muss eine festgelegte Höhe bzw. Breite einhalten und jede View muss individuell definiert werden. Android bietet dabei die komfortable Möglichkeit, Höhe und Breite per *wrap_content*, was die Größe anhand des Inhaltes der View einstellt oder per *match_parent*, was die Größe an das Elternelement anpasst, einzustellen. Android unterstützt keine Größeneinstellungen in einfachen Pixeln, sondern in dp[4].

Bei der Positionierung von Elementen interpretiert Android die Geometrie einer View als Rechteck. Dabei kann sie einfach per zweidimensionaler Beschreibung ausgedrückt werden. Zum Beispiel kann eine Positionierung einfach per *left* oder *bottom* mit Koordinaten ausgedrückt werden. Genauso kann die aktuelle Position per Code mit *getLeft()* oder *getBottom()* abgefragt werden.

Abgesehen von der einfachen Positionierung bietet Android auch die Möglichkeit des Einstellens oder Abfragens von Padding, Margin und der Size.

Android unterscheidet drei grundlegende Layouttypen:

- Das ***lineare Layout*** ist der grundlegendste Typ, hierbei werden alle Kind-Elemente horizontal oder vertikal am Layout ausgerichtet. Dieses Layout erzeugt automatisch eine Scroll-Leiste, wenn die Layoutlänge größer als der Bildschirm wird.
- Bei dem ***relativen Layout*** werden alle Kind-Elemente relativ zu anderen Elementen ausgerichtet. Zum Beispiel wird ein Text unter einem anderen Text positioniert.
- Die ***WebView*** zeichnet darin aus, das sie eine Webseite anzeigt.

Zusätzlich zu den Standard Layouts existieren in Android noch spezielle Layouts, welche auch als Mischung aus Layout und View betrachtet werden können:

- Die ***ListView*** zeigt eine einfache Liste, in der Inhalte zeilenweise dargestellt und gescrollt werden können.
- Eine ***GridView*** bietet hingegen ein quadratisches Raster, welches in Zeilen und Spalten eingeteilt wird.

4 Dichte unabhängiger Pixel

3.4.2 Menüs

In Android sind Menüs eine der Standardkomponenten im Bereich der Interaktion. Google empfiehlt diese zu nutzen, um eine konsistente Nutzerfahrung zu bieten [DAM15]. Traditionelle Android-Menüs bieten lediglich die Möglichkeit höchstens sechs Menüpunkte zur Verfügung zu stellen. Dies war auch ein Grund, warum Google ab Android 3.0 die Action Bar einführte. Auf diese wird im Abschnitt 3.4.4 Action Bar näher eingegangen.

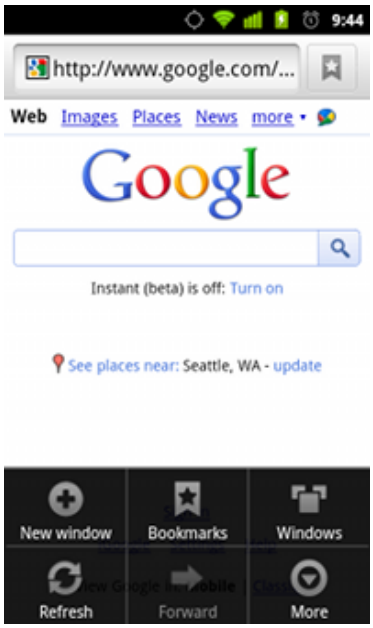


Abbildung 7: Android-Menü [DAM15]

Dabei teilen sich die Menüs in drei Typen ein:

- Das ***Optionsmenü*** stellt das ehemalige primäre Menü dar. Es sollten Menüpunkte bereitgestellt werden, die die aktuelle Activity betreffen. Des Weiteren sind diese durch eine Android bedingte Sanktionierung auf sechs Punkte begrenzt.
- Bei dem ***Kontextmenü*** handelt es sich um ein schwebendes Menü. Es wird durch ein langes Klicken auf ein Oberflächenelement aufgerufen. Diese werden meistens mit Listviews kombiniert.
- ***Popupmenüs*** zeigen für gewöhnlich eine Liste von Elementen an. Sie sollen hauptsächlich eine übersichtliche Möglichkeit darstellen, um Aktionen in Bezug auf den aktuellen Inhalt anzuzeigen.

3.4.3 Navigation Drawer

Bei dem Navigation Drawer handelt es sich um eine Navigationsleiste, die in Android-Apps als eine Art Hauptmenü Verwendung finden soll. Dabei ist diese im Normalfall mit der linken Seite des App-Bildschrimes verknüpft und kann per Knopfdruck oder wischen von links nach rechts geöffnet werden.

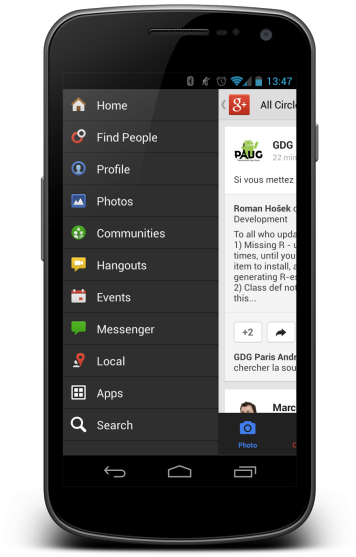


Abbildung 8: Navigation Drawer [AND13]

3.4.4 Action Bar

Android bietet mit seiner Action Bar eine obere Leiste die nicht nur Text anzeigt, sondern auch Funktionen ausführen kann. Diese könnte auch mit der aus anderen Betriebssystemen bekannten Werkzeugleiste verglichen werden.

Abbildung 9: Action Bar [DAB15]

Die mögliche Darstellungsvariation umfasst dabei:

(1) Das Icon der App.

(2) Menüpunkte die mit wichtigen Funktionen belegt sind.

(3) Weitere Menüpunkte mit oft verwendeten Funkionen.

Die Action Bar bietet mehrere Schlüsselfunktionen:

- Ermöglicht eine konsistente Navigation.
- Macht wichtige Aktionen besser zugänglich.
- Kann einer App eine gewisse Identität verleihen.

Bei der Action Bar ist es wichtig zu beachten, das mindestens Android 3.0 verwendet wird.

3.4.5 Settings

Damit der Nutzer die Einstellungen der App verändern kann, bietet Android eine Preference-Programmierschnittstelle.

Settings werden nicht wie normale View-Objekte erstellt, sie werden von einer Unterklasse von Preference deklariert. Jede Preference stellt dabei einen Menüpunkt in einer Liste dar. Dort kann zum Beispiel ein Menüpunkt aus einem Text und einem Kontrollkästchen bestehen.

Preferences unterstützten aber nur folgende Datentypen:

- Boolean
- Float
- Int
- Long
- String

Des Weiteren kann für die Preferences keine normale Activity genutzt werden. Es muss auf die PreferenceActivity zurückgegriffen werden.

Zusätzlich können die Menüpunkte, um die Übersichtlichkeit zu erhöhen, in Gruppen eingeteilt und mit einem Titel versehen werden. Darüber hinaus ist die Nutzung von sogenannten Unterbildschirmen möglich. Dabei wird beim Klicken auf den jeweiligen primären Menüpunkt ein neues Fenster mit den Unterpunkten geöffnet, wobei der Name des primären Menüpunktes den Menütitel einnimmt.

3.4.6 Dialoge

Dialoge sind kleine Fenstern in denen Nutzern Anfragen gestellt werden können beziehungsweise dem Nutzer zusätzliche Informationen zur Verfügung gestellt werden können. Sie bedecken nur einen Teil des Bildschirmes und werden zum Beispiel dafür verwendet Passwörter zu erfragen oder Fehlermeldungen auszugeben.

Android bietet drei Arten von Dialogen:

- Der ***AlertDialog*** zeigt einen Titel, maximal 3 Knöpfe, optional eine Liste beziehungsweise einen Text oder auch ein individuell erstelltes Layout. Der AlertDialog stellt die meist verwendete Dialogart dar.
- Der ***DatePickerDialoge*** wird primär nur dafür verwendet, möglichst einfach ein Datum auszuwählen.
- Der ***TimePickerDialog*** wird analog zum DatePickerDialog zur komfortablen Zeitauswahl verwendet.

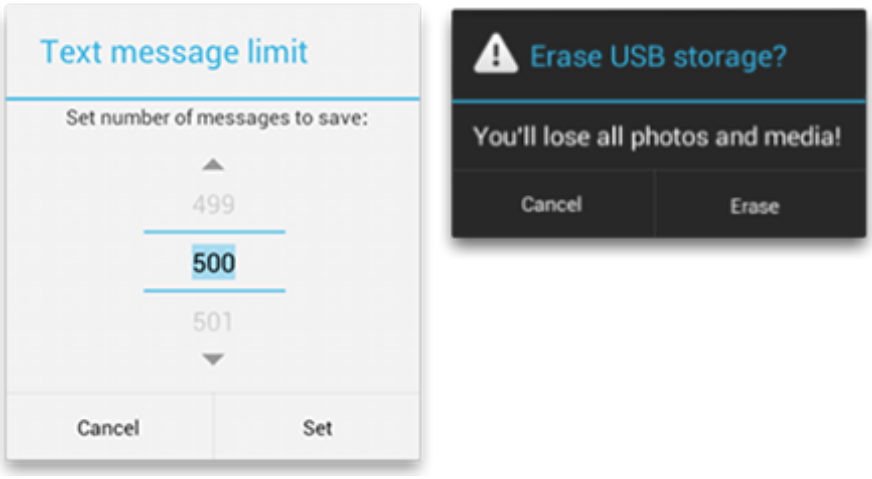


Abbildung 10: AlertDialogs [DAD15]

Dialoge können darüber hinaus auch als Vollbild angezeigt oder in Fragmente eingebettet werden.

3.4.7 Notifications

In Android besteht die Möglichkeit, sogenannte Notifications zu erzeugen. Diese sind Mitteilungen, die dem Nutzer außerhalb der eigentlichen App in der sogenannten Notification Area, angezeigt werden.

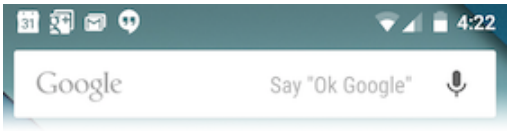


Abbildung 11: Notification Area [DAN15]

Wenn der Nutzer von der Notification Area nach unten wischt, öffnet sich der Notification Drawer.

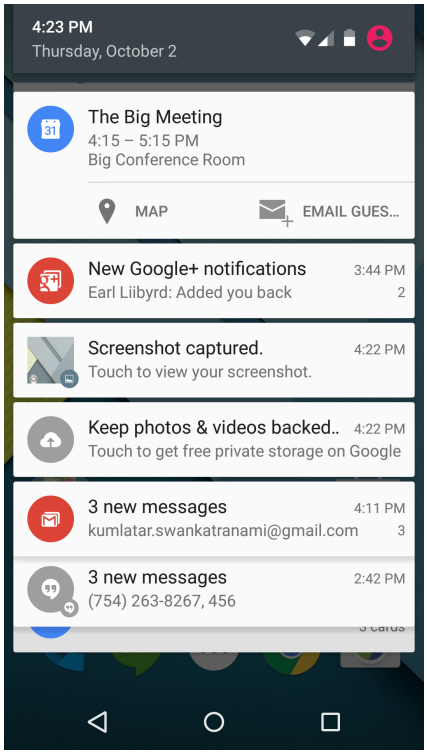


Abbildung 12: Notification Drawer [DAN15]

Eine Notification muss drei Komponenten enthalten:

- Eine kleines ***Icon***.
- Einen ***Titel***.
- Einen ***Detailtext***.

Optional kann noch eine Komponente Namens Action hinzugefügt werden. Diese sollen es erlauben, direkt von der Notification mit der Activity, der jeweiligen Android-App zu interagieren.

Das einfache Klicken auf eine Notification öffnet für gewöhnlich die jeweilige Anwendung. Dabei kann jeder Notification eine Priorität zugeordnet werden, die sich über fünf Stufen von -2 bis +2 erstreckt.

Ab Android 5.0 werden auch Heads-Up Notifications unterstützt. Bei Heads-Up Notifications handelt es sich um kleine schwebende Fenster, die es möglich machen, eine Aktion durchzuführen oder abzubrechen, ohne die aktuelle App zu verlassen.

Abbildung 13: Heads-Up Notification [DAN15]

Als weiteres Feature bietet Android ab der Version 5.0, die Lock Screen Notifications. Diese sind Notifications, die auf dem Sperrbildschirm angezeigt werden. Dabei kann der Nutzer bestimmen, welche und ob überhaupt Notifications auf dem Sperrbildschirm erscheinen sollen.

3.4.8 Toasts

Bei Toasts handelt es sich um kleine Pop-up-Fenster, die möglichst einfach und unkompliziert dem Nutzer Feedback geben sollen.

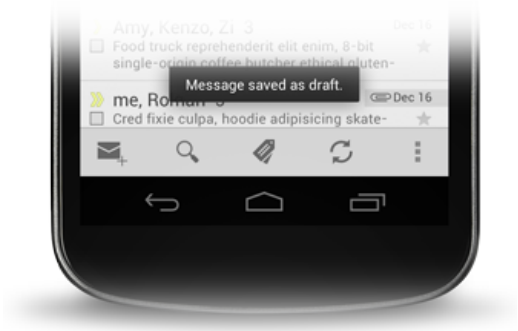


Abbildung 14: Toast [DAT15]

Ein Toast ruft im Normalfall lediglich einen Text auf und verschwindet dann nach einer gewissen Zeitspanne wieder. Die von Android vorgegebenen Zeitspannen betragen 3500 und 2000 Millisekunden.

4 Konzept

Bei der Entwicklung der veganen Kochbuch-App „Vapp“ soll einen Client-Server Verbindung gewährleistet werden, wobei das Android-Smartphone die Rolle das Clients übernimmt. Der Server soll die Daten in Form einer Datenbank für die App zur Verfügung stellen.

Der Datenaustausch bildet dabei die Kernfunktion der „Vapp“. Es sollen dabei je nach Bedarf Daten wie Rezepte oder Bewertungen ausgetauscht werden.

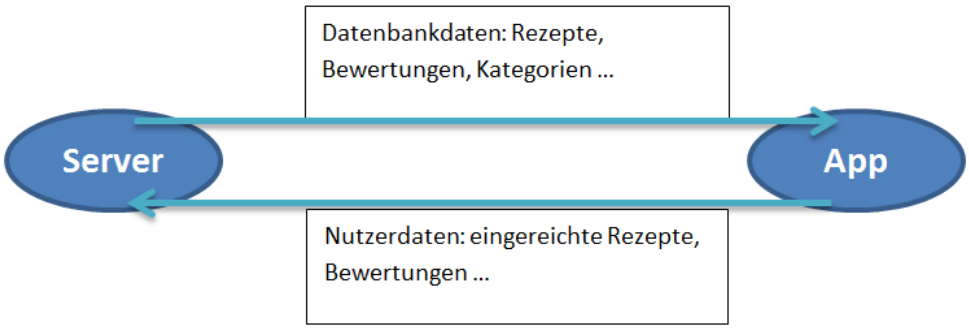


Abbildung 15: Schematischer Aufbau des Datenverkehrs der Vapp

Die Hauptaufgabe des Android-Smartphones soll das Speichern und Visualisieren dieser Daten sein.

Die Visualisierung der Daten beziehungsweise die Funktionalitäten der App werden in vier Bereiche unterteilt: Rezeptsuche, Rezeptdatenbank, Rezept einreichen und Feedback. Die Navigation soll es dem Nutzer ermöglichen jeden Bereich von jedem anderen Bereich aus zu erreichen.

4.1 Rezeptsuche

Der erste Bereich der „Vapp“ ist die Rezeptsuche. In der Rezeptsuche werden dem Nutzer alle verfügbaren Zutaten aufgelistet. Aus diesen Zutaten kann der Nutzer dann wiederum die von ihm gewünschten Zutaten auswählen. Nach der getroffenen Auswahl wird der Nutzer auf eine Ergebnisseite weitergeleitet. Die Ergebnisseite besteht aus einer Liste von Rezepten, in der der Rezeptname, das Rezeptbild, die bisherigen Rezeptbewertungen und die aus der Suche enthaltenen Zutaten abgebildet werden. Sollte sich der Nutzer daraufhin für ein Rezept entscheiden und dieses öffnen, so wird ihm eine Detailansicht mit Rezeptnamen, Rezeptbild, Rezeptbewertungen, Zubereitungsliste und Zutatenliste angezeigt. Optional bietet sich in der Detailansicht für den Nutzer die Möglichkeit, das Rezept über einen Button auf der Menüleiste über Portale, wie Facebook oder Twitter, zu teilen. Das Rezept wird daraufhin in eine Bilddatei umgewandelt und an den jeweiligen Dienst versandt.

4.2 Rezeptdatenbank

Im Bereich der Datenbank werden die Rezepte erst einmal in Kategorien eingeteilt, um eine gewisse Übersichtlichkeit zu gewährleisten. Diese Kategorien werden in Form einer Liste dargestellt. Auf der Liste sind die jeweiligen Kategorien mit einem Kategorienamen und einem Kategoriebild zu sehen. Wird eine Kategorie vom Nutzer

gewählt, so wird der Nutzer auf eine Liste von Rezepten, die unter die jeweilige Kategorie fällt, weitergeleitet. Diese Rezeptliste zeigt dann die jeweiligen Namen, Bewertungen und Bilder der Rezepte an. Sollte daraufhin ein Rezept vom Nutzer ausgewählt werden, so wird es analog zu der Detailansicht in der Rezeptsuche angezeigt.

4.3 Rezept einreichen

Um eine Form der Interaktivität zu erreichen und die Generierung von Content zu steigern, wird die „Vapp“ mit der Möglichkeit des Erstellens von eigenen Rezepten ausgestattet. Dem Nutzer wird dabei ein Formular vorgegeben, mit dessen Hilfe er ein vollwertiges Rezept an den Server schicken kann. Damit ein Rezept erfolgreich versandt werden kann, muss es einen Titel besitzen, eine der vorgegeben Kategorien muss ausgewählt sein, es muss eine Beschreibung vorhanden sein und es müssen die Zutaten, in die dafür vorgesehene Liste, eingetragen werden. Optional besteht noch die Möglichkeit, entweder eine Bild aus der Galerie des Smartphones mit zu versenden, oder über den Knopf „Bild machen“ eines zu machen. Nachdem der Nutzer dann einen zufallsgenerierten Sicherheitscode in eingegeben hat, kann das Rezept versandt werden. Nach einem erfolgreichem Versand, erscheint ein Dialog, der auf den erfolgreichen Versand verweist und mit den Nutzer auf die Mainactivity oder auf den Startbildschirm der „Vapp“ weiterleitet.

4.4 Feedback

Um leicht mit dem Entwickler in Kontakt treten zu können, wird ein Bereich für das Feedback eingerichtet. Dies ermöglicht es, dem Nutzer direktes Feedback zu versenden. Dabei soll ein Textfeld für eventuelle Anmerkungen und ein Bewertungsfeld für einfaches Feedback im zu versendenden Formular enthalten sein. Wenn der Nutzer nach der Eingabe der Daten das Formular versenden möchte, muss er analog zum Vorgang beim „Rezept einreichen“, einen Sicherheitscode angeben. Sollte der Sicherheitscode mit dem vorgegebenen übereinstimmen, so werden die Daten versandt, es erscheint ein Dialog und der Nutzer wird zum Startbildschirm der „Vapp“ weitergeleitet.

5 Realisierung

Bei der Realisierung der „Vapp“ wird in erster Linie darauf eingegangen, wie Android-Komponenten auf Codeebene verwendet werden. Es wird aber auch gezeigt, wie die Zusammenarbeit von Java-Code für die Komponenten mit den XML-Dateien für Oberfläche und Manifest funktioniert.

5.1 Activities und Intents

Als Beispiel für die Realisierung von Activities und Intents, soll die Klasse RezeptSucheActivity dienen.

```
1   package rez_suche_objects;
2
3   import android.content.Intent;
4   import android.os.Bundle;
5   import android.view.View;
6   import android.widget.BaseAdapter;
7   import android.widget.ListView;
8
9   import com.vapp.tii.vapp.DrawerActivity;
10  import com.vapp.tii.vapp.MainActivity;
11  import com.vapp.tii.vapp.R;
12
13  import java.util.ArrayList;
14
15  import Adapter_objects.RezeptSucheAdapter;
16
17
18  public class RezeptSucheActivity extends DrawerActivity {
19
20      public static ArrayList<String> ZutatenListe= new ArrayList<>();
21      private BaseAdapter adapter;
22
23      @Override
24      protected void onCreate(Bundle savedInstanceState) {
25          super.onCreate(savedInstanceState,R.layout.suche_rez_list_layout);
26
27          ListView lv=(ListView)findViewById(R.id.suche_rez_listView);
28          adapter = new RezeptSucheAdapter(this, MainActivity.databaseHandler.getDistinctZutatenListe());
29
30          lv.setAdapter(adapter);
31      }
32
33      public void RezeptSuchen(View view){
34
35          if (!ZutatenListe.isEmpty()) {
36
37              Intent intent = new Intent(this, RezeptSucheErgebnisActivity.class);
38              intent.putStringArrayListExtra("zutaten", ZutatenListe);
39              startActivity(intent);
40          }
41      }
42
43      @Override
44      protected void onPostResume() {
45          super.onPostResume();
46          ZutatenListe.clear();
47          adapter.notifyDataSetChanged();
48
49      }
50  }
```

In den Zeilen 1 bis 15 werden wie aus Java bereits bekannt, die notwendigen Bibliotheken und Klassen importiert. Ab Zeile 18 wird die Activity *RezeptSucheActivity* implementiert. Diese erbt von der Klasse *DrawerActivity* auf die im Abschnitt 5.7

Navigation Drawer, näher eingegangen wird. Wie im Abschnitt 3.3.1 Activities erwähnt, besitzt die Activity *RezeptSucheActivity* die vererbten Methoden *onCreate()* und *onPostResume()*. Die Methode *onCreate()* wird beim Erzeugen der Activity *RezeptSucheActivity* ausgeführt und die Methode *onPostResume()* wird nach dem pausieren und zurückkehren zu dieser Activity, ausgeführt. Von der Zeile 33 bis zur Zeile 41 wird die Methode *RezeptSuche()* ausgeführt, in der ein Intent, wie im Abschnitt 3.3.2 erwähnt, erzeugt und ausgeführt wird, sobald das Arraylist Namens ZutatenListe nicht leer ist. Der Intent muss das aktuelle Klassenobjekt, sowie die Zielklasse übergeben bekommen, da es sich hierbei um einen expliziten Intent handelt. Der Intent bekommt in der Zeile 38 die Arraylist „ZutatenListe“ mit der Bezeichnung „zutaten“ übergeben. Die Bezeichnung ist dabei für das Extrahieren der Daten von Bedeutung. Die Daten können nur über die Bezeichnung von der Activity, der die Daten übergeben wurden, gefunden werden. Ein Auszug aus der *RezeptSucheErgebnisActivity* soll zeigen, wie die von dem Intent versandten Daten wieder extrahiert werden können.

```
17  public class RezeptSucheErgebnisActivity extends DrawerActivity {
18
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState,R.layout.suche_rez_ergebnis_list_layout);
22
23          ArrayList<String> Zutatenlist= this.getIntent().getStringArrayListExtra("zutaten");
24
25          ListView listView = (ListView) findViewById(R.id.suche_rezept_list_listView);
26
27          BaseAdapter adapter = new RezeptErgebnisAdapter(this, MainActivity.databaseHandler
28                  .getRezepteByZutat(ListUtility.makeListToString(Zutatenlist)),Zutatenlist);
29
30          listView.setAdapter(adapter);
31
32      }
```

In der Zeile 23 ist zu sehen, wie über die Activity, die per *this* aufgerufen wird, die Methode *getIntent()* ausgeführt wird und die Arrayliste aus *Strings* mit der Beschreibung „zutaten“ extrahiert wird.

5.2 Manifest

Als Beispiel für die Realisierung eines Manifests soll die Datei AndroidManifest.xml der „Vapp“ dienen.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vapp.tii.vapp" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/icon"
        android:label="Vapp"
        android:theme="@style/AppTheme"
        >
        <activity
            android:name=".MainActivity"
            android:label="Vapp" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

Nachdem in der XML-Datei grundlegende Elemente, wie die Version und die Kodierung angegeben werden können, wird in dem Tag *<manifest>* das Schema des Manifests und das dazugehörige Paket, welches die App darstellt, angegeben. In diesem Fall ist das dazugehörige Paket die „Vapp“. Wie in Abschnitt 3.3.3 Manifest angedeutet, wird mithilfe des Tags *<user-permission>* der App Zugriff auf verschiedene Bereiche des Android-Smartphones gewährt. Hier benötigt die „Vapp“ vollständigen Internetzugriff, sowie Lese- und Schreibrechte für den externen Speicher. Innerhalb des *<application>* Tags können Attribute, wie der Anwendungsname oder auch das verwendete Design, festgelegt werden. Mit dem Tag *<activity>* wird eine Activity im Manifest vermerkt. Innerhalb dieses Tags müssen Daten, wie der Name der Activity, angegeben werden. Mit dem Tag *<intent-filter>* werden implizite Intents vermerkt. In diesem Fall wird eine *<category>* und eine *<action>* integriert. Innerhalb der *<action>* wird eingestellt das die Mainactivity beim ausführen des Intents gestartet wird. In der *<category>* wird zusätzlich vermerkt, dass es sich hierbei um den Launcher der App handelt.

In dem hinteren Teil des Manifests werden in diesem Beispiel alle weiteren Activities angegeben und die Tags *<applications>* und *<manifest>* geschlossen.

5.3 Prozesse

Als Beispiel für einen Prozess wird in der „Vapp“ die AsyncTask genutzt. Wie im Abschnitt 3.3.6 Prozesse bereits bemerkt, zeichnen sich diese durch die Möglichkeit aus, im Hintergrund zu arbeiten, während sie den Vordergrund nicht direkt beeinflussen.

```
1   package helper_objects;
2
3   import android.graphics.Bitmap;
4   import android.graphics.BitmapFactory;
5   import android.os.AsyncTask;
6   import android.util.Log;
7   import android.widget.ImageView;
8
9   import java.net.URL;
10  import java.net.URLConnection;
11
12
13  public class ImageDownloader extends AsyncTask<String, Void, ImageView> {
14
15      private String requestUrl;
16      private ImageView view;
17      private Bitmap bitmap;
18
19      public ImageDownloader(String requestUrl, ImageView view) {
20          this.requestUrl = requestUrl;
21          this.view = view;
22
23      }
24
25      @Override
26      protected ImageView doInBackground(String... params) {
27
28          try {
29                  URL url = new URL(requestUrl.toString());
30
31                  URLConnection conn = url.openConnection();
32                  bitmap = BitmapFactory.decodeStream(conn.getInputStream());
33              } catch (Exception ex) {
34
35
36                  Log.i("Fehler beim herunterladen des Bildes!", ex.toString());
37              }
38
39              return view;
40      }
41      @Override
42      protected void onPostExecute(ImageView view) {
43          view.setImageBitmap(bitmap);
44          super.onPostExecute(view);
45      }
46
47
48  }
```

Hierbei erbt die Klasse *ImageDownloader* von der Klasse *AsyncTask,* um auf ihre Fähigkeiten zurückgreifen zu können. Wie von Java gewohnt, werden in Zeile 1 bis 10 alle notwendigen Pakete importiert beziehungsweise wird auch der *ImageDownloader* in der ersten Zeile dem Paket *helper_objects* zugeordnet. Bei der Klassendefinition in

Zeile 13 müssen beim Erben von AsyncTask drei Typen festgehalten werden. Nach dem Erzeugen einer von *AsyncTask* erbenden Klasse, muss um den darin enthaltenen Code auch auszuführen, die Methode *execute()* ausgeführt werden. Dieser Methode kann ein Parameter übergeben werden. Der Typ des ersten Parameters muss mit dem Typ des übergebenen Wertes in der Methode *execute()* übereinstimmen. Der Typ des zweiten Parameters muss mit dem Typ des Übergabeparameters der Methode *onPreExecute()* übereinstimmen. Doch wird im Beispiel der Klasse *ImageDownloader* die Methode nicht verwendet. Android schreibt dabei vor, dass der zweite Parameter auf den Typ „Void“ gesetzt wird. Der Typ des dritten Parameters beschreibt den Typ der Variable die von der Methode *doInBackground()* zur Methode *onPostExecute()* übertragen wird. Weiter unten in der Klasse *ImageDownloader* werden von der Zeile 15 bis 17, Variablen deklariert. Dabei stellt das ImageView-Element eine Art Rahmen für eine Bitmap dar, welches in Android die Bilddateien repräsentieren. Wenn auch im Normalfall alle Variablen über die *execute()*-Methode an die *doInBackground()*-Methode übergeben werden, werden in diesem Fall alle Daten über den Konstruktor beim Erzeugen eines Objektes mitgeliefert. Den Grund dafür stellt das Problem dar, dass die *execute()*-Methode lediglich einen Datentypen übertragen lässt, während in den Konstruktor zwei oder mehr Datentypen geladen werden können. Weiter unten in den Zeilen 28 bis 40 wird dann versucht, über die integrierte *URLConnenction-Klasse,* die Bilddateien herunterzuladen und diese in das Format einer Bitmap zu kodieren. In Zeile 39 wird daraufhin die ImageView an *onPostExecute()* weitergeleitet. Diese setzt dann in Zeile 43 das Bild der ImageView auf die heruntergeladene Bitmap.

5.4 Datenbanken

Das Kernstück der „Vapp“ ist die Datenbank. Alle rezeptbezogenen Informationen werden in ihr gespeichert beziehungsweise werden beim Start der App die Daten der Datenbank mit den Daten einer zentralen Datenbank auf einem Server abgeglichen.

Bei der Realisierung soll dieses Entity-Relationship-Diagram (ERD) als Basis für die Datenbank des Servers sowie für die Datenbank der App dienen:

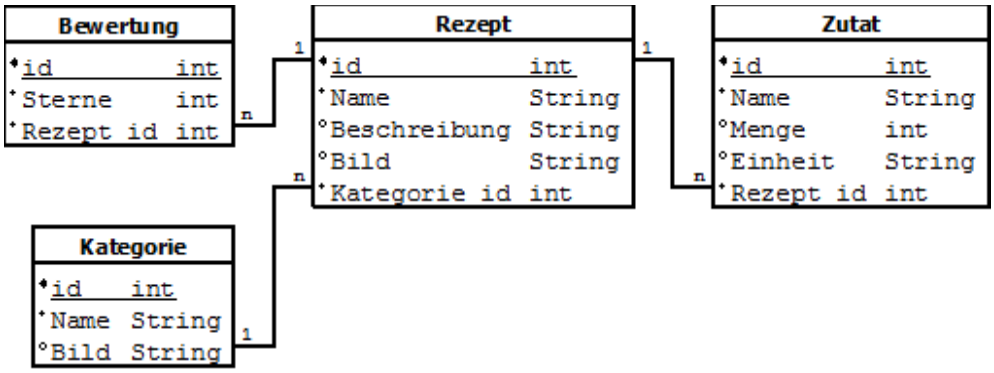


Abbildung 16: ERD der grundlegenden Datenbank der App

Dieses ERD muss je nach Einsatzgebiet individuell angepasst werden. Zum Beispiel sollte auf dem Server extra vermerkt werden, ob das Rezept zum Download freigegeben ist. Um in Android Kommunikation mit einer Datenbank zu ermöglichen ist es notwendig eine Klasse zu erzeugen, die von der abstrakten Klasse *android.database.sqlite.SQLiteOpenHelper* erbt. Dabei ist zu beachten die beiden Methoden *onCreate()* und *onUpgrade()* zu überschreiben.

```java
package handler_objects;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import java.util.ArrayList;
import java.util.List;

import database_objects.Bewertung;
import database_objects.Kategorie;
import database_objects.Rezept;
import database_objects.Zutat;


public class DatabaseHandler extends SQLiteOpenHelper
{

    private static final String DATABASE_NAME = "rezepte";
    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_REZEPT = "CREATE DATABASE IF NOT EXISTS " + DATABASE_NAME;

    private static final String TABLE_REZEPT = "CREATE TABLE IF NOT EXISTS rezept " +
            "( id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "name TEXT, beschreibung TEXT,bild BLOB,kategorie_id integer" +
            ")";

    private static final String TABLE_BEWERTUNG = "CREATE TABLE IF NOT EXISTS bewertung " +
            "( id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "sterne INTEGER,rezept_id INTEGER" +
            ") ";

    private static final String TABLE_ZUTAT = "CREATE TABLE IF NOT EXISTS zutat " +
            "( id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "name TEXT,einheit TEXT,menge INTEGER,rezept_id INTEGER" +
            ")";
    private static final String TABLE_KATEGORIE = "CREATE TABLE IF NOT EXISTS kategorie " +
            "( id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "name TEXT, " +
            "bild BLOB"+
            ")";
    private static final String DROP_DATABASE ="DROP DATABASE "+ DATABASE_NAME;

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL(TABLE_REZEPT);
        db.execSQL(TABLE_BEWERTUNG);
        db.execSQL(TABLE_ZUTAT);
        db.execSQL(TABLE_KATEGORIE);
    }

```

Die *DatabaseHandler-Klasse* erweitert die *SQLiteOpenHelper-Klasse* der bereits in Abschnitt 3.3.9 Datenbanken vorgestellten Datenbanklösung SQLite. In den Zeilen 1 bis 15 werden alle notwendigen Pakete und Objekte importiert. Dazu zählen selbst erstellte Klassen oder auch aus Java bekannte Klassen wie *List*. Während von Zeile 21 bis zur Zeile 45 die für die Datenbank notwendigen Variablen, in Form von SQL-Befehlen, erzeugt werden. Später werden mit diesen String-Variablen die SQL-Befehle darstellen, die einzelnen Komponenten oder auch Tabellen der Datenbank erzeugt. In den Zeilen 47 bis 49 ist zu beachten, dass der Konstruktor der Klasse *DatabaseHandler* implementiert wird. Dieser Konstruktor benötigt dabei den Kontext[5], den er mit dem Namen der Datenbank und der Version der Datenbank an den

Konstruktor der Klasse *SQLiteOpenHelper* übergibt. Beim erzeugen einer Datenbank spielt die Methode *onCreate()* eine große Rolle, ihr wird ein Objekt vom Typen *SQLiteDatabase* intern übergeben und in ihr können mit Hilfe der Methode *execSQL()* SQL-Befehle ausgeführt werden, welche in diesem Fall das erzeugen der in Zeile 26 bis 44 beschriebenen Tabellen ist.

```
59
60      @Override
61      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
62          db.execSQL(DROP_DATABASE);
63          this.onCreate(db);
64      }
65
66
67
68      public void DeleteDB(SQLiteDatabase db){
69          db.execSQL(DROP_DATABASE);
70      }
```

Die in den Zeilen 61 bis 64 abgebildete Methode *onUpgrade()* kann aufgerufen werden, wenn sich die Version der Datenbank geändert hat. Dabei benötigt die Methode *onUpgrade()* ein Objekt vom Typen *SQLiteDatabase*, eine Integer-Variable mit der oldVersion und eine Integer-Variable mit der newVersion. Eine Änderung der Version kann bei der Instanziierung der Klasse *DatabaseHandler* geschehen. Sinnvoll erscheint diese Funktion bei strukturellen Veränderungen der Datenbank, wie dem hinzufügen neuer Spalten. Die Methode *DeleteDB()* wurde zu Testzwecken implementiert, sie sollte Testdaten bei Bedarf löschen. Dies tut sie analog über die Methode *execSQL()* eines Objektes vom Typen *SQLiteDatabase*.

```
72      public Rezept getRezept(int id) {
73
74          SQLiteDatabase database = this.getReadableDatabase();
75
76          Cursor cursor = database.query("rezept", new String[]{"id", "name", "beschreibung", "bild", "kategorie_id"},
77                                         " id = ?", new String[]{String.valueOf(id)}, null, null, null, null);
78
79          if (cursor != null)
80              cursor.moveToFirst();
81
82          Rezept rezept = new Rezept();
83          rezept.setId(cursor.getInt(0));
84          rezept.setName(cursor.getString(1));
85          rezept.setBeschreibung(cursor.getString(2));
86          rezept.setBild(cursor.getString(3));
87          rezept.setKategorie_id(cursor.getInt(4));
88
89
90          return rezept;
91      }
92      public ArrayList<Rezept> getRezepteByKategorie(String id) {
93
94          ArrayList<Rezept> rezepte = new ArrayList<>();
95          SQLiteDatabase database = this.getReadableDatabase();
96
97          Cursor cursor = database.rawQuery("SELECT * FROM rezept WHERE kategorie_id="+id, null);
98          Rezept rezept = null;
99
100         if (cursor.moveToFirst()) {
101             do {
102                 rezept = new Rezept();
103                 rezept.setId(cursor.getInt(0));
104                 rezept.setName(cursor.getString(1));
105                 rezept.setBeschreibung(cursor.getString(2));
106                 rezept.setBild(cursor.getString(3));
107                 rezept.setKategorie_id(cursor.getInt(4));
108
109
110                 rezepte.add(rezept);
111             } while (cursor.moveToNext());
112         }
113         return rezepte;
114
115     }
```

Als Beispiel für das Erzeugen von Objekten auf Basis der Klasse SQLiteOpenHelper sollen die Methoden *getRezept()* in Zeile 74 bis 91 und *getRezepteByKategorie()* in Zeile 92 bis 113 dienen. Das Objekt Rezept, dessen Struktur im Anhang ausführlich abgebildet wird, wird hierbei von der Methode *getRezept()* zurückgegeben. Übergabeparameter ist hierbei die Id des Rezeptes. Als erstes wird ein Objekt vom Typen *SQLiteDatabase* erzeugt. Diesem Objekt wird dann als nächstes die Methode *getReadableDatabase()* zugewiesen. Dabei verfügt die Klasse *DatabaseHandler* noch über viele weitere Methoden, wie auch die Methode *getWriteableDatabase()* mit dessen Hilfe es möglich ist, Informationen in die Datenbank zu schreiben. In der Zeile 76 wird ein Objekt vom Typen Cursor erzeugt. Ein Cursor ist im Kontext von SQLite eine Art Zeigewerkzeug für Datensätze. Ein erzeugter Cursor kann mit der Methode *query()* der Klasse *SQLiteDatabase* mit SQL-Befehlen beziehungsweise notwendigen Extras wie Befehlen zur Sortierung ausgestattet werden. In diesem Fall wird die ganze Tabelle „rezept“ in den Cursor geladen. Der zweite Parameter der Methode *query()* beschreibt die Daten die, dem Cursor zugeordnet werden sollen. Des Weiteren beschäftigen sich Parameter drei und vier mit der Bedingung, der von der Methode übergebenen Id der Datensätze. Nach dem er geladen wurde, wird der Cursor darauf geprüft, ob er einen Inhalt besitzt. Falls der Cursor dann tatsächlich nicht leer sein sollte, wird er mit der Methode *moveToFirst()* auf den ersten Datensatz in sich selbst, ausgerichtet. Nach dem Ausrichten des Cursors wird in der Zeile 82 ein neues Objekt vom Typen Rezept erzeugt und initialisiert. Die jeweiligen Variablen des erzeugten Objektes werden dann mit den Daten des Cursors gefüllt. Dies geschieht mit der Methode *getString(),* falls die Variable des Objektes auch vom Typ *String* ist. Falls sie ein Integer sein sollte, wird die Methode *getInt()* verwendet. Das in den Zeilen 82 bis 87 erzeugte Objekt, wird dann in der Zeile 90 von der Methode *getRezept()* zurückgegeben.

Als zweites Beispiel für den Zugriff auf den *DatabaseHandler* soll die Methode *getRezeptByKategorie()* dienen, diese umfasst Zeile 92 bis 115. Ein nennenswerter Unterschied zur Methode *getRezept()* ist, das hier in der Zeile 97 nicht die Methode *query()* verwendet wird, sondern die Methode *rawQuery()*. Dabei benötigt r*awQuery()* im Gegensatz zur Methode *query()* weniger Informationen und ist leichter zu handhaben. Ihr muss lediglich ein einfacher SQL-Befehl übergeben werden. Die Methode *getRezeptByKategorie()* unterscheidet sich auch durch die Erzeugung von mehr als einem Objekt vom Typen Rezept, dies geschieht in den Zeilen 101 bis 111 in einer Do-While Schleife. Analog zur Methode *getRezept()* wird in Zeile 113 ein Arrayliste von Objekten zurückgegeben.

5.5 Layouts und Views

Wie im Abschnitt 3.4.1 Layouts beschrieben, bestehen jegliche Nutzeroberflächen aus Views oder Layouts. Auch jede Benutzeroberfläche der „Vapp“ nutzt Layouts und Views. Hierbei soll die *activity_feedback.xml* der „Vapp“ als Beispiel für ein Layout dienen.

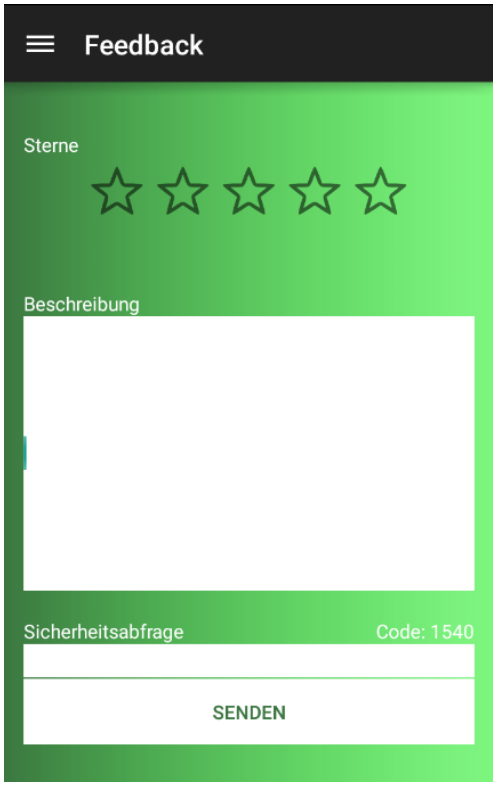


Abbildung 17: Layout des Feedback-Bereichs

```
1  <android.support.v4.widget.DrawerLayout
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:id="@+id/nav_drawer_layout"
6      >
7      <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
8          xmlns:tools="http://schemas.android.com/tools"
9          android:id="@+id/ScrollView01"
10         android:layout_width="fill_parent"
11         android:layout_height="fill_parent"
12         android:background="@drawable/menu_gradient"
13         android:paddingLeft="16dp"
14         android:paddingRight="16dp"
15         android:paddingTop="16dp"
16         android:paddingBottom="16dp"
17         >
18 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
19     xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
20     android:layout_height="match_parent"
21     tools:context="feedback_objects.FeedbackActivity">
22
23     <EditText
24         android:layout_width="wrap_content"
25         android:layout_height="200dp"
26         android:id="@+id/feedback_messagetext"
27         android:layout_below="@+id/feedback_beschreibung_text"
28         android:layout_alignParentLeft="true"
29         android:layout_alignParentStart="true"
30
31         android:layout_alignParentRight="true"
32         android:layout_alignParentEnd="true"
33         android:background="@color/app_color2"
34         />
```

In der Zeile 1 bis 6 befindet sich ein *DrawerLayout,* welches wichtig für die Erstellung eines Navigation Drawers ist und näher im Abschnitt 5.7 Navigation Drawer beschrieben wird. Dem *DrawerLayout* wurden elementare Attribute wie eine *Id*, eine „*width*“ und eine „*height*“ zugeordnet. Innerhalb des *DrawerLayouts* befindet sich in der Zeile 7 bis 17 der Tag einer ScrollView, diese ist notwendig, falls als Kind-Element eine RelativeLayout auftritt, um innerhalb dieses RelativeLayouts scrollen zu können. Innerhalb des Tags des RelativeLayouts befindet sich ein EditText, welcher ein Eingabefeld für Text darstellt. Da der EditText sich innerhalb des RelativeLayouts befindet, muss eine Positionierung relativ zu anderen Elementen vorgenommen werden. In diesem Fall wird der EditText mit der *Id „feedback_messagetext*“ durch das Attribut „*android:layout_below*“ unterhalb einer anderen View mit der *Id „feedback_beschreibung_text“* positioniert. In der Zeile 33 wird dem EditText dann noch mit Hilfe des Attributs *„android:background“* eine Hintergrundfarbe zugeordnet.

```
36      <Button
37          android:layout_width="wrap_content"
38          android:layout_height="wrap_content"
39          android:text="Senden"
40          android:id="@+id/feedback_senden_button"
41          android:textColor="@color/app_color"
42          android:background="@color/app_color2"
43          android:layout_below="@+id/feedback_sicherheitsabfrage_edittext"
44          android:layout_alignParentLeft="true"
45          android:layout_alignParentStart="true"
46          android:layout_alignRight="@+id/feedback_messagetext"
47          android:layout_alignEnd="@+id/feedback_messagetext"
48          android:layout_marginTop="1dp" />
49
50      <RatingBar
51          android:layout_width="wrap_content"
52          android:layout_height="wrap_content"
53          android:id="@+id/feedback_ratingBar"
54          android:layout_below="@+id/feedback_sterne_text"
55          android:layout_centerHorizontal="true"
56          />
57
```

In Zeile 36 bis 48 wird ein Button erzeugt und konfiguriert. Buttons können in Ausnahmen direkt per XML Funktionen zugeordnet werden. Alternativ ist eine Zuordnung auch per Code möglich. Dabei wird ein sogenanntes ClickListener-Objekt erzeugt, welches die *Id* des Buttons zugeordnet bekommt.

```
79      <TextView
80          android:layout_width="match_parent"
81          android:layout_height="wrap_content"
82          android:text="Sicherheitsabfrage"
83          android:id="@+id/feedback_sicherheitsabfrage_text"
84          android:layout_below="@+id/feedback_messagetext"
85          android:layout_marginTop="20dp"
86          android:textColor="@color/app_color2"
87          />
88      <TextView
89          android:layout_width="match_parent"
90          android:layout_height="wrap_content"
91          android:text="Code:"
92          android:id="@+id/feedback_sicherheitsabfrage_wert_text"
93          android:layout_below="@+id/feedback_messagetext"
94          android:gravity="right"
95          android:layout_marginTop="20dp"
96          android:textColor="@color/app_color2"
97          />
98
99      <EditText
100         android:layout_width="match_parent"
101         android:layout_height="wrap_content"
102         android:id="@+id/feedback_sicherheitsabfrage_edittext"
103         android:layout_below="@+id/feedback_sicherheitsabfrage_text"
104         android:background="@color/app_color2"
105
106         />
107
108 </RelativeLayout>
109
110     </ScrollView>
111
112 <include layout="@layout/drawer_include_layout"></include>
113
114
115 </android.support.v4.widget.DrawerLayout>
```

In den Zeilen 108 bis 115 werden alle Tags des Layouts geschlossen und in Zeile 112 wird eine weitere externe XML-Datei per Include-Tag in das Layout integriert. Jenes Layout kommt aber erst in Bezug auf den Navigation Drawer zum tragen.

5.6 Listviews

Android bietet bei den Listviews die Möglichkeit, vorgefertigte Designs zu nutzen, da das Design der „Vapp“ jedoch individuell ist, konnte dies in der Realisierung keine Verwendung finden. Jede Listview der „Vapp“ wurde individuell entworfen. Dabei benötigt eine Listview Komponenten, wie einen Adapter, ein Listenlayout und ein Layout für die einzelnen Punkte der Liste.

```
1  <android.support.v4.widget.DrawerLayout
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:id="@+id/nav_drawer_layout"
6      >
7  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
8      android:layout_width="match_parent" android:layout_height="match_parent"
9      android:background="@drawable/menu_gradient">
10
11     <ListView
12         android:layout_width="wrap_content"
13         android:layout_height="match_parent"
14         android:id="@+id/katalog_rezept_list_listView"
15         android:layout_alignParentTop="true"
16         android:layout_centerHorizontal="true" />
17 </RelativeLayout>
18  <include layout="@layout/drawer_include_layout"></include>
19
20
21 </android.support.v4.widget.DrawerLayout>
```

Eine Listview kann wie jede andere View per Tag in ein Layout integriert werden, wie in Zeile 11 bis 18 der *katalog_rez_list_layout.xml* zu sehen. Es ist dabei notwendig, dass eine *Id* vergeben wird.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"

    >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/rez_titel"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/rez_bild"
        android:layout_toEndOf="@+id/rez_bild"
        android:textColor="@color/app_color2" />

    <ImageView
        android:layout_width="128dp"
        android:layout_height="128dp"
        android:id="@+id/rez_bild"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:src="@drawable/vapp_logo"
       />

    <RatingBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/rez_ratingBar"
        android:layout_below="@+id/rez_bild"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:numStars="5"
        android:rating="5"

        />
</RelativeLayout>
```

In der *katalog_rez_item_layout.xml* werden die einzelnen Elemente der Listview festgelegt. In diesem Fall wird ein Layout erzeugt, welches einen Text, ein Bild und eine Bewertungsleiste enthält.

```java
package Adapter_objects;

import ...

public class RezeptAdapter extends BaseAdapter {

    private final Activity activity;
    private ArrayList<Rezept> rezeptList;

    public RezeptAdapter(ArrayList<Rezept> rezeptList, Activity activity) {
        this.rezeptList = rezeptList;
        this.activity = activity;
    }
    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {

        LayoutInflater inflater = (LayoutInflater) activity
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View view = inflater.inflate(R.layout.katalog_rez_item_layout, parent, false);
        TextView textView = (TextView) view.findViewById(R.id.rez_titel);
        ImageView imageView = (ImageView) view.findViewById(R.id.rez_bild);
        RatingBar ratingBar=(RatingBar)view.findViewById(R.id.rez_ratingBar);

        textView.setText(rezeptList.get(position).getName());
        ratingBar.setRating(Float.valueOf(MainActivity.databaseHandler
                .getBewertung(String.valueOf(rezeptList.get(position).getId()))));
        ratingBar.setIsIndicator(true);

        ImageDownloader imageDownloader = new ImageDownloader(rezeptList.get(position)
                .getBild(),imageView);
        imageDownloader.execute();

        view.setOnClickListener((v) -> {
                Intent intent = new Intent(activity, RezeptDetailActivity.class);
                intent.putExtra("id", String.valueOf(rezeptList.get(position).getId()));
                activity.startActivity(intent);
        });
        return view;
    }

    @Override
    public int getCount() { return rezeptList.size(); }

    @Override
    public Object getItem(int position) { return rezeptList.get(position); }

    @Override
    public long getItemId(int position) { return position; }
}
```

Im Fall der Klasse *RezeptAdapter* wurde, wie in Zeile 23 zu sehen, von der bereitgestellten Klasse vom Typen BaseAdapter geerbt. Android bietet dabei mehrere Grundklassen für Listview-Adapter. Die Klasse BaseAdapter bietet jedoch die einfachste Variante wenn eine Listview individuell gestaltet werden soll. Dabei muss, wie in Zeile 64 bis 74 zu sehen, eine von BaseAdapter erbende Klasse, die Methoden *getCount()*, *getItem()* und *getItemId()* überschreiben. Mit Hilfe dieser Methoden erkennt der Adapter wie groß die Liste ist, wie auf einzelne Elemente zugegriffen werden kann und wie die Elemente positioniert sind. Der Konstruktor der Klasse *RezeptAdapter* in Zeile 28 bis 31 wird mit einer Liste von Objekten mit dem Typen

Rezept und der ausführenden Activity initialisiert. In der Methode *getView()* wird es ermöglicht, einzelne Elemente der Listview zu manipulieren. In Zeile 35 wird ein LayoutInflater erzeugt, der die Aufgabe hat, das Standard-Layout aufzublasen. In der Zeile 38 wird diesem Inflater das Layout für die einzelnen Elemente zugewiesen, da diese manipuliert werden sollen. In den Zeilen 39 bis 41, werden dann den in der Listview vorkommenden Objekte, über ihre *Id* per Code erzeugten Elemente zugewiesen. Darauf folgend werden in den Zeilen 43 bis 46 die Views bearbeitet. In Zeile 48 bis 50 wird das herunterladen eines Bildes über die Klasse ImageDownloader veranlasst. In den Zeilen 52 bis 58 wird ein *OnClickListener* ausgeführt. Dieser horcht, ob die ihm zugewiesene View angeklickt wurde. Beim Anklicken dieser, wird ein Intent ausgelöst, der die *DetailRezeptActivity* mit der dem Element entsprechenden *Id* öffnet.

5.7 Navigation Drawer

Der Aufbau eines bereits in Abschnitt 3.4.3 erwähnten Navigation Drawers ähnelt zu einem großen Teil dem einer Listview. Er verfügt analog zur Listview über einen Adapter, ein Listenlayout und ein Layout für die einzelnen Elemente. Jedoch ist das initialisieren eines Navigation Drawers weitaus komplexer als das einer einfachen Listview. Um einen Navigation Drawer in jeder Activity zu integrieren, wurde eine Klasse implementiert von der jede Activity der „Vapp“ erbt.

```
1   package com.vapp.tii.vapp;
2
3   import ...
14
15  public class DrawerActivity extends ActionBarActivity {
16
17      private DrawerLayout drawerLayout;
18      private ActionBarDrawerToggle toggle;
19      private ListView drawerListview;
20
21      private ArrayList<String> List= new ArrayList<>();
22      private int layoutResourceId;
23
24
25      @Override
26      protected void onPostCreate(Bundle savedInstanceState) {
27          super.onPostCreate(savedInstanceState);
28          toggle.syncState();
29      }
30
31      @Override
32      public void onConfigurationChanged(Configuration newConfig) {
33          super.onConfigurationChanged(newConfig);
34          toggle.onConfigurationChanged(newConfig);
35      }
36      @Override
37      public boolean onOptionsItemSelected(MenuItem item) {
38          if (toggle.onOptionsItemSelected(item)) {
39
40              return true;
41          }
42
43          return super.onOptionsItemSelected(item);
44      }
```

Beim Erzeugen eines Navigation Drawers benötigt eine Activity ein DrawerLayout und einen ActionBarDrawerToggle. Das Layout übernimmt dabei die Rolle der für das Design notwendigen Struktur und der ActionBarDrawerToggle, die des Schalters für das Aufrufen des Navigation Drawers. Mit Hilfe der Methoden *onPostCreate()*, *onConfigurationChanged()* und *onOptionsItemSelected()* wird zum Beispiel das Verhältnis des Drawers nach Neukonfiguration der Bildschirmauflösung ausgeführt.

```
47      protected void onCreate(Bundle savedInstanceState,int layoutResourceId) {
48          super.onCreate(savedInstanceState);
49
50          this.layoutResourceId = layoutResourceId;
51
52
53          setContentView(layoutResourceId);
54
55          List.add("Home");
56          List.add("Suche");
57          List.add("Datenbank");
58          List.add("Einreichen");
59          List.add("Feedback");
60
61          drawerLayout = (DrawerLayout) findViewById(R.id.nav_drawer_layout);
62
63          drawerListview = (ListView) findViewById(R.id.left_drawer);
64
65
66          drawerListview.setAdapter(new DrawerAdapter(List,this,drawerLayout));
67
68          toggle= new ActionBarDrawerToggle(this,drawerLayout, "Öffne...", "Schließe...");
69          drawerLayout.setDrawerListener(toggle);
70
71          getSupportActionBar().setDisplayHomeAsUpEnabled(true);
72          getSupportActionBar().setHomeButtonEnabled(true);
73
74      }
75  }
```

In der Methode *onCreate()* werden die Texte für die Menüpunkte hinzugefügt, welche dem DrawerAdapter in der Zeile 66 mit übergeben werden. In Zeile 68 und 69 wird der Toggler zum Layout zugeordnet. Des Weiteren wird über die Zeilen 71 und 72 sichergestellt, dass der Toggler aktiv ist.

5.8 Dialoge und Toasts

In Android ist es üblich, per Dialog oder Toast mit dem Nutzer zu interagieren. Wo der Dialog eher für Fragestellungen Verwendung findet, soll ein Toast kurze Infos an den Nutzer geben. In der „Vapp“ wurden die meisten Toasts und Dialoge im Kontext des Internetzugriffs verwendet. So auch in diesem Ausschnitt der Mainactivity.

```
1   package com.vapp.tii.vapp;
2
3   import ...
12
13
14  public class MainActivity extends DrawerActivity {
15
16      public static DatabaseHandler databaseHandler;
17      private AlertDialog.Builder builder;
18
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState,R.layout.activity_main);
22
23          databaseHandler=new DatabaseHandler(getApplicationContext());
24
25          builder =new AlertDialog.Builder(MainActivity.this);
26          builder
27          .setTitle("Vapp")
28          .setMessage("Fehlerhafte Internetverbindung!")
29          .setPositiveButton("Beenden", (dialog, id) -> {
31                  MainActivity.this.finish();
32          })
34          .setNegativeButton("Weiter", (dialog, id) -> { dialog.cancel(); });
39
40          AlertDialog dialog = builder.create();
41
42                  if (AsyncHttpClient.isOnline(getApplicationContext())) {
43                      DownloadData();
44                  } else {
45                      dialog.show();
46
47                  }
48              }
49
50              public void löschen() {
51
52                  boolean b = this.deleteDatabase(databaseHandler.getDatabaseName());
53                  Log.d("Alles gelöscht!", "" + b);
54                  Toast.makeText(this,"Alles wurde gelöscht!",Toast.LENGTH_SHORT).show();
55
56              }
```

Der Ausschnitt beinhaltet einen AlertDialog, der in Abschnitt 3.4.6 Dialoge bereits erwähnt wurde. Jener muss über einen Builder konfiguriert werden. Es stehen dabei Optionen, wie der Titel des Fensters, die Nachricht und welche Knöpfe verwendet werden sollen, zur Verfügung. Der Builder erzeugt über die Methode *create(),* wie in der Zeile 40 zu sehen, einen Alertdialog. Dieser muss abschließend mit der *show() Methode* aufgerufen werden. Dem Nutzer erscheint der Dialog dann augenblicklich. In diesem Fall wird ein Dialog ausgelöst, wenn der AsyHttpClient feststellt das keine Internetverbindung vorhanden ist. Im Gegensatz zum Dialog kann ein Toast mit verhältnismäßig geringen Aufwand erzeugt und ausgelöst werden. Üblicherweise

reicht es, das statische Objekt *Toast* mit der Methode *makeText()* aufzurufen und dieser drei notwendige Parameter zu übergeben. Diese Parameter bestehen aus der aktuell aktiven Activity, dem zu zeigenden Text und dem jeweiligen Zeitfenster. Dies erzeugt dann dem Nutzer, wie in Abschnitt 3.4.8 Toasts gezeigt für einen begrenzten Zeitraum, eine kurze Nachricht.

6 Evaluation

Im Rahmen der Evaluation der „Vapp“ bot sich eine Online-Umfrage über die Online-Plattform „soscisurvey.de“ an. Die Webseite „soscisurvey.de“ bietet die Möglichkeit, individuell Online-Umfragen zu gestalten und die daraus resultierenden Testdaten herunterzuladen. Des Weiteren gibt es auf dieser Umfrage-Plattform keine quantitativen Beschränkungen.

Der Test sollte zeigen, in wie weit die „Vapp“ Anklang findet und zu bedienen ist. Die Fragen wurden in 5 Rubriken eingeteilt: Allgemein, Ernährung, Gerät, Rezept und App.

Die Umfrage umfasste insgesamt 15 Datensätze, wovon 10 auswertbar waren. Die fünf nicht auswertbaren Datensätze resultierten aus nicht vollständig ausgefüllten Fragebögen und nicht bestandenen Testfragen.

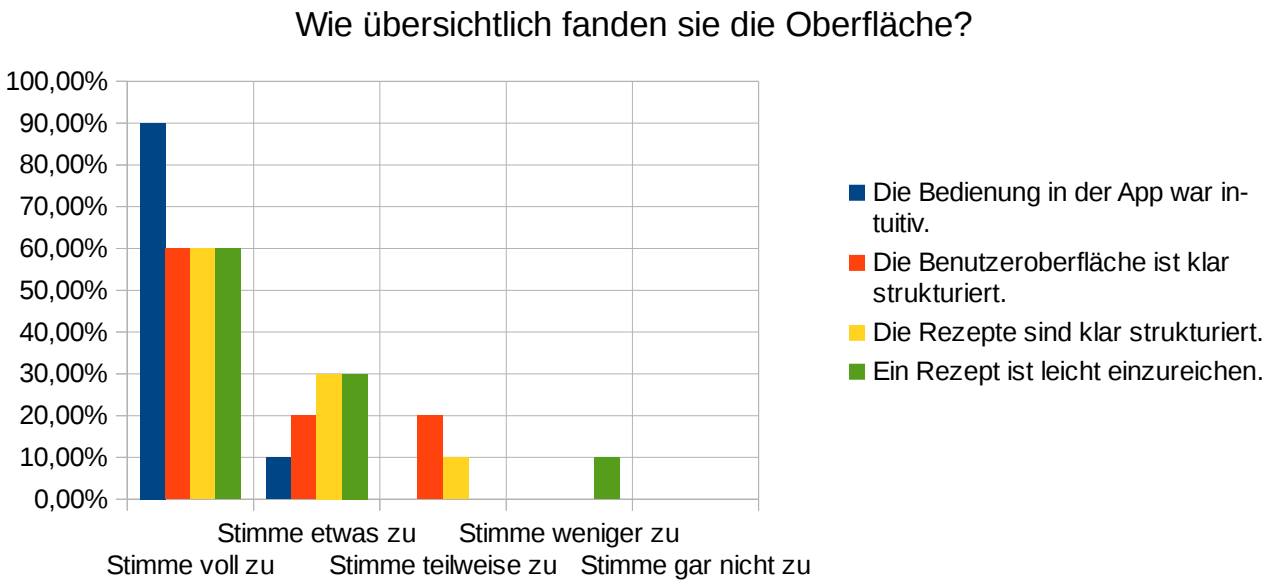
Die Auswertung der Allgemeinen Rubrik ergab, dass das Alter der Probanden von 21 bis 53 Jahre reichte. Das Durchschnittsalter betrug 36 Jahre. Dabei waren 60% der Probanden weiblich und die restlichen 40% männlich.

In der Rubrik Ernährung zeigte sich, dass 90% der Probanden sich vegan ernährten, während 10% sich omnivore ernähren. In der Umfrage wurden auch weitere Antwortmöglichkeiten wie „vegetarisch“ oder sonstiges angeboten, jedoch nahm sie niemand wahr. Auf die Frage „Haben sie Interesse am Veganismus?“ antworteten 90% „Ja“ während 10% die Antwortmöglichkeit „etwas“ wählten. Die Möglichkeit „Nein“ zu wählen, nahm niemand wahr.

Bei der Auswertung der Daten der Rubrik Gerät zeigte sich, dass alle Probanden ein Smartphone besaßen und dieses auch mit Android betrieben wurde. Auf die Frage: „Wie oft benutzen sie das Gerät?“ antworteten 60% „sehr häufig“, während 30% antworteten „häufig“ und die restlichen 10% angaben „regelmäßig“.

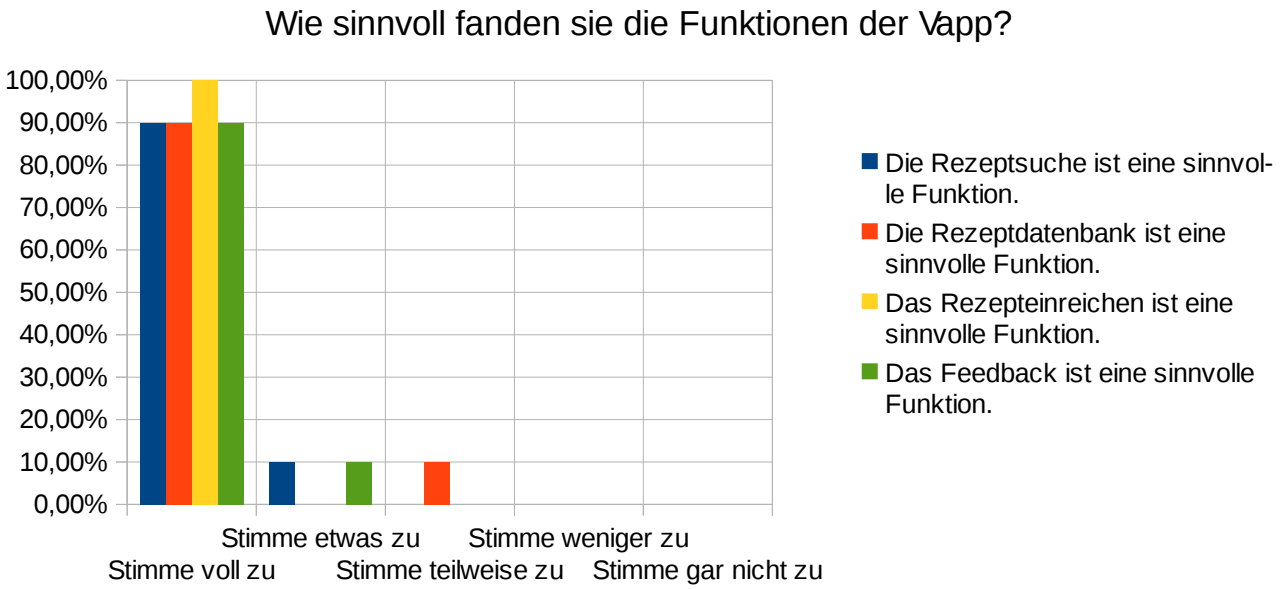
Darauf folgend zeigte sich in der Rubrik Rezepte, dass 40% der Probanden bereits eine Rezeptbuch App auf dem Smartphone installiert hatten, während die restlichen 60% keines auf dem Gerät hatten. Die Antwortmöglichkeit „Weiß nicht“ wurde von niemanden ausgewählt. In der darauf folgenden Frage ob sie diese App regelmäßig nutzen antworteten 25% „sehr häufig“, während 50% angaben diese „regelmäßig“ zu nutzen. Die restlichen 25% sagten aus, die App eher „selten“ zu nutzen.

Die Rubrik App bestand aus zwei einfachen und zwei komplexeren Fragestellungen. Die erste komplexe Fragestellung befasst sich mit der Übersichtlichkeit der Oberfläche.



Dabei ergab sich eine große Zustimmung in Bezug auf Intuitivität und Strukturierung.

Die zweite komplexe Frage befasste sich mit der Sinnhaftigkeit der jeweiligen Funktionen der App.



Bei der Auswertung der zweiten Frage zeigte sich, dass ein Großteil der Probanden alle Funktionen der „Vapp“ als sinnvoll ansah, während die Rezeptsuche eine 100% Zustimmung in Bezug auf die Sinnhaftigkeit erhielt.

Auf die Frage „Was hat ihnen besonders gefallen?“ antworteten Probanden unter anderem:

- einfache Rezepte
- Gruppierung/Auflistung
- Suchfunktion

Bei der Frage „Was bzw. welche neuen Funktion würden sich in Zukunft für die Vapp wünschen?“ wurde angegeben:

- Favoriten-Liste
- Schließen-Knopf
- Kaufinformationen zu den Zutaten
- Kochvideos

Insgesamt ergab die Evaluation ein sehr positives Ergebnis. Die Probanden machten sinnvolle Vorschläge für zukünftige Funktionen der „Vapp“ und ein Großteil der Nutzer war sehr zufrieden mit den Funktionalitäten und der Struktur der „Vapp“.

7 Fazit

Innerhalb dieser Arbeit zeigte Android, dass es nicht nur in Bezug auf die Verbreitung, sondern auch mit Blick auf die einzelnen Komponenten, eine sehr umfangreiche Plattform ist. Android steht damit anderen Plattformen in nichts nach. Es bietet eine hohe Stabilität durch das Ausführen jeder Anwendung in einem eigenen Thread und mit klar strukturierten Komponenten, wie Activities oder Intents, wird dem Entwickler eine zuverlässige Plattform geboten. Der Entwickler erhält durch den Gebrauch bekannter Programmiersprachen, wie Java und bekannter Programmierumgebungen, wie Eclipse, eine konsistente Programmiererfahrung.

Damit ist ohne Weiteres die Entwicklung einer individuellen Anwendung für Mobilgeräte möglich. Unter Betrachtung der jeweiligen Bestandteile, wie den Elementen der Benutzeroberfläche oder den Komponenten auf Codeebene, ermöglicht es die Entwicklung einer vollwertigen Anwendung.

Im Zuge der Evaluation dieser Anwendung, die aus vier Bereichen: Suche, Datenbank, Einreichen und Feedback bestand, zeigte sich, dass sie grundsätzlich den Anforderungen potentieller Nutzer gewachsen ist. Dies schlug sich unter anderem im Kontext der Fragestellung im Bezug auf die Übersichtlichkeit nieder, mit einer zumindest teilweisen Zustimmung von 90% und einer vollkommenen Zustimmung von 60%.

Des Weiteren zeigte sich im Kontext der Fragestellung, wie sinnvoll die einzelnen Bereiche der Anwendung wären, eine hohe Zustimmung von bis zu und über 90%.

Insgesamt ergab sich im Bezug auf die Evaluation, dass ein Großteil der Nutzer zufrieden mit den Funktionen und dem Umfang der entwickelten Anwendung ist.

Es wurden zwar Verbesserungsvorschläge angegeben, aber das Feedback war zum größten Teil sehr positiv.

Abschließend zeigte sich bei der Entwicklung der „Vapp“, dass es möglich war, mit Android ein vollwertiges, interaktives, digitales Kochbuch zu entwickeln.

8 Ausblick

In Hinblick auf Android lässt die „Vapp“ großen Spielraum für mögliche Erweiterungen. So sollen, wenn genügend Nachfrage besteht, viele Verbesserungen und zusätzliche Module hinzukommen.

Im Bereich der internen Verarbeitung könnte eine Verbesserung sein, dass das Herunterladen von Bildern nur einmalig geschieht und die Bilder in der lokalen Datenbank zwischengespeichert werden, statt sie wie momentan implementiert, nach jedem Neustart der App, erneut herunterzuladen. Des Weiteren wäre im Bereich der Kommunikation mit dem Server Verbesserungsspielraum vorhanden, indem Anfragen spezifischer, mit begrenzten Volumen, eingeführt werden würden. Dies würde den von der App verursachten Datenverkehr beträchtlich senken.

Bei den Erweiterungen wäre es weiterhin denkbar, Konten für den Nutzer anzulegen, mit Hilfe derer er mit anderen Nutzern Kontakt aufnehmen kann. Das Einführen einer Kommentarfunktion wäre auch denkbar.

Eine Erweiterung in Hinsicht auf allgemeine Informationen oder Statistiken über eine vegane Lebensweise beziehungsweise ein Stadtführer, in dem vegane Geschäfte oder Restaurants eingetragen werden könnten, wäre ebenfalls möglich.

Man könnte sich auch an bereits existierenden Apps orientieren und einen Barcode-Scanner implementieren, der ein Produkt auf seine Inhaltsstoffe analysiert.

Die bei der Evaluation erbrachten Antworten beinhalteten auch Vorschläge, wie zum Beispiel einen Knopf für das Beenden der App oder mögliche Erweiterungen, wie zusätzliche Kochvideos.

Aber auch außerhalb von Android sind Erweiterungen möglich, wie das Erstellen einer Online-Plattform, von der aus die eingereichten Rezepte ohne viel Arbeit freigegeben werden könnten und ähnliche Funktionen wie die der „Vapp“ geboten werden könnten.

Literaturverzeichnis

[AND13] J. Lehtimaki, Navigation Drawer, In: androiduipatterns.com, Online-Quelle:< http://www.androiduipatterns.com/2013/05/the-new-navigation-drawer-pattern.html> (Stand 14.03.2015)

[DAA15] Google, Activities, In: android.com, Online-Quelle:< http://developer.android.com/guide/components/activities.html> (Stand 14.03.2015)

[DAB15] Google, Action Bar, In: android.com, Online-Quelle:< http://developer.android.com/guide/topics/ui/actionbar.html> (Stand 14.03.2015)

[DAD15] Google, Dialogs, In: android.com, Online-Quelle:< http://developer.android.com/guide/topics/ui/dialogs.html> (Stand 14.03.2015)

[DAF15] Google, Fragments, In: android.com, Online-Quelle:< http://developer.android.com/guide/components/fragments.html> (Stand 14.03.2015)

[DAL15] Google, Layouts, In: android.com, Online-Quelle:< http://developer.android.com/guide/topics/ui/declaring-layout.html> (Stand 14.03.2015)

[DAM15] Google, Menus, In: android.com, Online-Quelle:< http://developer.android.com/guide/topics/ui/menus.html> (Stand 14.03.2015)

[DAN15] Google, Notifications, In: android.com, Online-Quelle:< http://developer.android.com/guide/topics/ui/notifiers/notifications.html> (Stand 14.03.2015)

[DAT15] Google, Toasts, In: android.com, Online-Quelle:< http://developer.android.com/guide/topics/ui/notifiers/toasts.html> (Stand 14.03.2015)

[GOL14] T. Költzsch, Android wird zum Schokoriegel, In: golem.de, Online-Quelle:< http://www.golem.de/news/kitkat-android-wird-zum-schokoriegel-1309-101362.html> (Stand 14.03.2015)

[HAS15] Hwaci - Applied Software Research, About SQLite, In: sqlite.org, Online-Quelle:< http://www.sqlite.org/about.html> (Stand 14.03.2015)

[MGO10] *Mentor Graphics Corporate Office,* Android beyond the handset, In: techdesignforums.com, Online-Quelle:< http://www.techdesignforums.com/practice/technique/android-beyond-the-handset/> (Stand 14.03.2015)

[LAV14] L. Vogel, Android SQLite database and content provider, In: vogella.com, Online-Quelle:< http://www.vogella.com/tutorials/AndroidSQLite/article.html> (Stand 14.03.2015)

[STA15] Statista GmbH, Vergleich der Marktanteile von Android und iOS am Absatz von Smartphones in Deutschland von Januar 2012 bis Januar 2015, In: statista.com, Online-Quelle: < http://de.statista.com/statistik/daten/studie/256790/umfrage/marktanteile-von-android-und-ios-am-smartphone-absatz-in-deutschland/> (Stand 14.03.2015)

[STE14] stern.de GmbH, McDonald's brät jetzt vegane Burger, In: stern.de, Online-Quelle:< http://www.stern.de/wirtschaft/news/fast-food-ohne-fleisch-mcdonalds-braet-jetzt-vegane-burger-2141128.html> (Stand 14.03.2015)

[TKH12] T. Künneth (2012), Android4: Apps entwickeln mit dem Android SDK, Bonn.

[VEB14] Vegetarierbund Deutschland, Anzahl der Vegetarier in Deutschland, In:vebu.de,
Online-Quelle:< https://www.vebu.de/themen/lifestyle/anzahl-der-vegetarierinnen>
(Stand 14.03.2015)

Abbildungsverzeichnis

9 Anhang

```java
package database_objects;


import ...

public class Rezept {

    private int id;
    private String name;
    private String beschreibung;
    private String bild;
    private int kategorie_id;

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getBeschreibung() { return beschreibung; }

    public void setBeschreibung(String beschreibung) { this.beschreibung = beschreibung; }

    public String getBild() { return bild; }

    public void setBild(String bild) { this.bild = bild; }

    public int getKategorie_id() { return kategorie_id; }

    public void setKategorie_id(int kategorie_id) { this.kategorie_id = kategorie_id; }

    public Rezept(int id, String name, String beschreibung, String bild, int kategorie_id) {
        this.id = id;
        this.name = name;
        this.beschreibung = beschreibung;
        this.bild = bild;
        this.kategorie_id = kategorie_id;
    }

    public Rezept() {
    }

    //Requestobjekt für Datenübertragung
    public RequestBuilder getRequest(){
        RequestBuilder requestBuilder=new RequestBuilder();

        requestBuilder.put("name",getName());
        requestBuilder.put("beschreibung",getBeschreibung());
        requestBuilder.put("kategorie_id",String.valueOf(getKategorie_id()));

        return  requestBuilder;
    }
    //Zum prüfen ob Objekt in Liste ist
    public boolean inList(ArrayList<Rezept> rezeptArrayList){
        for (Rezept rezept: rezeptArrayList){
            if (rezept.getId()==this.getId()){
                return true;
            }
        }
        return false;
    }
}
```