Christoph-Scheiner-Gymnasium Ingolstadt

Seminararbeit

aus dem wissenschaftspropädeutischen Seminar

„Programmieren von Android-Apps“

im Fach Informatik

Erstellung einer Android-App zum Lernen von Namen mit besonderem Augenmerk auf Datenaustausch mit anderen Systemen

angefertigt von	Leander Dreier
Reifeprüfungsjahrgang	2017
Kursleiter	OStR Pabst

Ingolstadt, den 07. November 2016

Inhaltsverzeichnis

1. Einleitung

Ich wurde letztens gefragt, was das Thema meiner Seminararbeit sei, worauf ich antwortete: „Ich programmiere eine Namenlern-App“. Der Fragende schien damit nicht wirklich etwas anfangen zu können, es schien ihm nicht klar zu sein, wozu man solch eine App brauchen könnte, er könne sich die Namen seiner Klassenkameraden schließlich auch ohne Hilfe eines Computers merken.

Es gibt aber durchaus Anwendungsgebiete für mein Programm: Es kann von Lehrern verwendet werden, um sich die Namen ihrer Schüler leichter zu merken, da dies aufgrund der vielen und häufig wechselnden Schüler oft ein Problem ist. Ein Lehrer hat bereits angefragt, diese App verwenden zu dürfen. Aber natürlich beschränkt sich meine Zielgruppe nicht nur auf Lehrer. Alle Arbeitnehmer und -geber, die mit viel Personal arbeiten und alle anderen, die Schwierigkeiten haben, sich Namen zu merken, können sich ihr Leben mit diesem Programm ein Stück leichter machen.

Das Grundprinzip der Anwendung ist es, je ein Bild mit einem Namen zu verknüpfen und den Benutzer abzuprüfen, ob er den Namen zu den Bildern weiß.

Dass es aber nicht nur möglich ist, mit dieser Software Namen und Bilder zu verknüpfen, zeigt eine Anfrage seitens der Klinik für Kinder- und Jugendpsychiatrie und Psychotherapie der Kliniken St. Elisabeth in Neuburg an der Donau, die App in der Diagnostik und Therapie von Autismuspatienten einzusetzen. Dabei werden Emotionen mit Gesichtsausdrücken verknüpft, so zum Beispiel ein lachender Mund mit Freude.

Was es nun genau mit dieser Anwendung auf sich hat, werde ich Ihnen im folgenden darlegen.

2. Grundfunktionen des Programms

Wie oben beschrieben, geht es in NameMemo um die Zuordnung von Namen zu Bildern und das Lernen dieser Verknüpfungen. (In der weiteren Arbeit wird von einer Standardzuordnung Name-Bild ausgegangen.)

Dies wird folgendermaßen umgesetzt: Zuerst sieht der Nutzer ein Bild und einen Button mit der Beschriftung „Name anzeigen“[1]. Jetzt ist der Nutzer angehalten, sich den zu dem gezeigten Bild gehörigen Namen zu überlegen. Hat er das getan, drückt er den Button, worauf nun der tatsächliche Name und zwei weitere Buttons – „Ja“ und „Nein“ – erscheinen[2]. Hat der Nutzer den richtigen Namen gewusst, drückt er auf „Ja“, andernfalls auf „Nein“ und ein neues Bild erscheint, bei dem das Prozedere von vorne beginnt. Das Ergebnis wird in einer Datenbank abgespeichert und kann unter anderem beeinflussen, wie häufig das Bild von nun an erscheinen wird.

Zudem hat der Nutzer, ab einer gewissen Anzahl aufeinanderfolgender, positiver Bestätigungen (Standard: 3), die Möglichkeit, den Eintrag aus der Datenbank zu löschen, um ihn somit als „gelernt“ abzuhaken. Nach den Standardeinstellungen wird das Bild umso seltener zur Abfrage aufgerufen, je näher man dieser Hürde kommt. Wurden alle Einträge entfernt, erscheint eine Meldung sowie ein Platzhalterbild.

Die Einträge in dieser Datenbank können zusätzlich auf ein anderes Gerät übertragen werden, um somit mehrere Instanzen der App synchron zu halten.

3. Das Laden der Bilder

In einer früheren Version wurden die anzuzeigenden Bilder in der ImageView durch *Bitmaps* erzeugt. Macht man allerdings ein neues Foto mit der von Android bereitgestellten Activity, so muss dieses Bild oft erst noch gedreht werden. Dies ist notwendig, da die Ausrichtung des Gerätes bei dem Erstellen des Bildes nur in Form von Exif-Daten (einer Form von Metadaten) gespeichert wird. Erzeugt man nun die Bitmap des gewünschten Bildes, muss man diese Daten erst auslesen und das Bild manuell drehen, bevor sie in die ImageView geladen werden kann. Dies benötigt viel Speicher, weshalb dieser Vorgang bei einigen Geräten zu Abstürzen des Programms führt.

Aus diesem Grund verwendet NameMemo nun das Framework *Glide*, die auf der dazugehörigen GitHub-Seite wie folgt beschrieben wird: „Glide is a fast and efficient open

1 Vgl. Abb. 1.1 (Anhang)
2 Vgl. Abb. 1.2 (Anhang)

source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface.“[3] Glide kümmert sich nicht nur automatisch um die korrekte Rotation des Bildes, sondern sorgt auch für deren einfache Größenmanipulation. Das bedeutet, dass jedes Bild, welches in eine ImageView geladen wird, in seiner Größe angepasst wird.

Hierfür werden beim Start der App Werte für Höhe und Breite generiert und gespeichert. Die Breite entspricht dabei der Breite jener Fläche des Displays, das die Anwendung enthält.[4] Die Höhe allerdings entspricht nur 75% der Höhe dieser Fläche, um dem Platzanspruch der Actionbar gerecht zu werden. Für die „normale“ ImageView werden diese Werte als minimale und maximale Größe gesetzt und die Bilder werden auch in dieser Größe von Glide geladen. Für die Bildvorschau beim Hinzufügen von Dateien in Listenform werden die Werte vor Weiterverwendung noch halbiert.

4. Die Datenbank

Wie oben bereits erwähnt, wird zum Speichern sowohl der „Bilderdaten“, als auch der Einstellungen, eine Datenbank verwendet, um die Daten bei einem Schließen der Anwendung nicht zu verlieren. Zu diesem Zweck wird die Engine *SQLite* eingesetzt, da sie bereits auf jedem Androidgerät vorhanden ist. Die beiden Tabellen *settings* und *pictures* werden von je einer DAO-Klasse (DAO = Data Access Object) verwaltet. Nur darauf wird im sonstigen Code zugegriffen, wodurch sie für die gesamte Interaktion mit der Datenbank zuständig sind.

4.1 Einstellungen in der Datenbank

Die Tabelle *settings* ist, wie es der Name vermuten lässt, für die Einstellungen zuständig. So wird hier gespeichert, wie oft der Benutzer bei einem Bild nacheinander auf „Ja“ drücken muss, bevor er das Angebot bekommt, es zu löschen. Weiter wird gespeichert, ob die Bilder nacheinander oder in einer zufälligen Reihenfolge nach dem in Punkt 2 beschriebenen Algorithmus erscheinen sollen (Standard: zufällig), ob alle Bilddateien in

3 BUMP
4 Vgl. ANDROID, *Display*

einem Ordner gesammelt werden sollen – ist diese Option aktiviert, werden Bilder, die vom Speicher hinzugefügt werden, in diesen Ordner kopiert (Standardwert: an), und ob man eben bei diesem Hinzufügen von Bildern aus dem Speicher des Gerätes eine Android-übliche Activity oder einen einfacher gehaltenen Dialog in Listenform verwenden möchte (Standard: Activity).

Die Standardwerte werden bei dem Starten des Programms gesetzt, sofern die Einträge nicht bereits in der Datenbank vorhanden sind, also im Normalfall nur bei dem ersten Start der App oder nach einem Reset. In der Anwendung kann der Nutzer, wenn er sich in der MainActivity befindet, alle oben beschriebenen Einstellungen in einem dafür vorgesehenen Dialog ändern. In diesem Dialog findet man auch die Möglichkeit alle Daten aus der Datenbank, also aus beiden Tabellen, zu löschen, was auch zu einem Reset der Einstellungen führt.

Auf technischer Ebene besteht jede Einstellung aus einem String („Kennung“) und einer Ganzzahl („Wert“). Die Kennung ist hierbei der Primärschlüssel, um Überschneidungen zu verhindern. Soll eine Änderung vorgenommen werden, wird in der Tabelle nach der Kennung gesucht und der dazugehörige Wert geändert. Zum Zwischenspeichern werden die Werte, wieder anhand ihrer Kennung, in *onStart()* und bei einer Änderung aus der Datenbank ausgelesen.

4.2 Bilder in der Datenbank

In der anderen Tabelle – *pictures* – werden die verschiedenen benötigten Daten zu den jeweiligen Bildern gesichert. Zum einen wird der Pfad zu der entsprechenden Bilddatei gespeichert. Es wird darauf verzichtet, die Datei aus der Datenbank rekonstruieren zu können, um Speicherplatz und Ressourcen zu sparen. Weiter werden der vom Nutzer zugeordnete Name, die Anzahl an Gesamtaufrufen und die Anzahl an positiven Rückmeldungen des Nutzers (= Drücken von „Ja“) in Folge gesichert. Letzterer Wert wird auf 0 zurückgesetzt, wenn nach Erscheinen des Bildes auf „Nein“ gedrückt wird.

Der Pfad der Bilder ist dabei der Primärschlüssel der Tabelle, weswegen man eine Bilddatei nicht mehrmals verwenden kann. Dadurch kann man eine gesuchte Zeile durch eben diesen Pfad genau identifizieren, um sie dann zu manipulieren. Für eine einfachere Handhabung zur Laufzeit werden Daten zu Bildern nicht direkt aus der Datenbank

abgerufen. Stattdessen wird eine Liste aller Einträge der Tabelle abgefragt und als Array gespeichert. Dieses wird dann unter anderem dafür verwendet, das nächste anzuzeigende Bild zu bestimmen und die Informationen dazu auszulesen.

4.3 Hinzufügen von Wertepaaren

In dem Programm sind drei Beispielbilder vorhanden. Der Zweck der App ist aber natürlich, eigene Bilder hinzuzufügen, um dann den Namen der jeweils zu sehenden Person zu lernen. Dafür stehen dem Nutzer lokal zwei Möglichkeiten zur Verfügung: die Aufnahme eines neuen Fotos mit der geräteinternen Kamera und das Auswählen eines bereits vorhandenen Bildes aus dem Speicher des Handys.

4.3.1 Aufnahme eines neues Fotos

Möchte man ein neues Foto schießen, so gelangt man in die Activity der Kamera-App des Gerätes, wo man wie gewohnt ein Bild aufnehmen kann. Hat man dies erfolgreich abgeschlossen, kann man dem Bild in dem erscheinenden Dialog einen Namen zuweisen. Nachdem man auch dies getan hat, ist die Verknüpfung in der Datenbank gespeichert und wird von nun an auftauchen. Die Bilddateien werden in einem eigenen Ordner gespeichert. Er befindet sich im Gerätespeicher unter *Pictures/NameMemo* und wird vom Programm angelegt, falls er noch nicht existiert.

Dieses Feature benötigt keine zusätzlichen Berechtigungen, da es die Kamera nicht direkt, sondern über die standardisierte Activity verwendet.

4.3.2 Auswahl eines bereits vorhandenen Bildes

Eine weitere Möglichkeit ist es, Dateien zu verwenden, welche bereits im Speicher vorhanden sind. Dafür gibt es nun wiederum zwei Varianten. Welche verwendet wird, kann in den Einstellungen ausgewählt werden.

Zum einen kann man seine Bilddatei in gewohnter Weise mithilfe von Android-Hausmitteln finden. Ist diese Option aktiv, erscheint zuerst ein Dialog zum Auswählen der Anwendung, mit der man das Bild finden möchte (zum Beispiel „Galerie“ oder „Fotos“). Das weitere Vorgehen hängt dann von der jeweiligen Anwendung ab.

Zum anderen ist es auch möglich, die Datei direkt in der App mithilfe einer einfachen Liste

in einem Dialog auszuwählen. In diesem werden alle JPG- und PNG-Dateien, alle Ordner im aktuellen Verzeichnis, sowie ganz oben ein Eintrag `..` angezeigt. Wählt man `..` aus, gelangt man im Dateisystem eine Ebene nach oben. Wählt man einen Ordner aus, gelangt man in diesen. Gestartet wird in dem App-eigenen Verzeichnis für Bilder, wie oben beschrieben.

Bei beiden Varianten wird man, sobald man seine Auswahl getroffen hat, aufgefordert, einen Namen zu dem Bild einzugeben. Nach der Eingabe wird der Eintrag in der Datenbank abgespeichert.

4.3.3 Lesen einer externen Datei

Zum Schluss bietet sich noch die Möglichkeit, die Datenbankdatei einer anderen Instanz der App, welche auf einem anderen Gerät läuft, zugesendet zu bekommen, um dann auf verschiedene Weisen weiter damit zu arbeiten. Die Handlungsmöglichkeiten durch den Nutzer werden dabei in einem Dialog dargestellt, welcher in der abstrakten Klasse *net.Net* implementiert ist. Diese benutzt dann, je nach Auswahl, verschiedene Methoden der Helferklassen *database.ImportNewDb* oder *main.Helper*. Die Auswahlmöglichkeiten sind folgende[5]:

- Ersetzen der aktuellen Datenbankdatei mit der neuen.
- Erzeugen einer Liste mit in der momentanen Datenbank noch nicht vorhandenen Einträgen. Der Nutzer kann die gewünschten Datenpaare dann einzeln übernehmen.
- Erzeugen einer Liste wie oben, aber auch mit aktualisierten Einträgen. Das Vorgehen ist entsprechend.
- Löschen der neuen Datei. Dieser Schritt wird bei jedem Abschluss automatisch ausgeführt und entspricht hier einem Abbruch.

5 Vgl. Abb. 1.11 (Anhang)

5. Datenaustausch mit anderen Systemen

Soviel zu der lokalen Funktionsweise des Programms. Kommen wir nun zu dem Kernaspekt der App: dem Datenaustausch mit anderen Systemen. Mit der Anwendung ist es möglich, die Tabelle *pictures* der Datenbank mit anderen Systemen zu teilen und seinen Fortschritt somit zu synchronisieren. Im Folgenden werden drei Varianten vorgestellt: die Übertragung mithilfe von Near Field Communication (NFC), via Bluetooth und mithilfe eines festen Servers im Internet. Implementiert sind allerdings nur die ersten beiden. Warum, wird in Punkt 5.4 genauer erläutert. Die beiden implementierten Ausführungen werden jeweils in einer separaten Activity umgesetzt.

5.1 Near Field Communication (NFC)

Near Field Communication wurde 2002 gemeinsam von *Sony* und *NXP Semiconductors* entwickelt und ermöglicht es, eine drahtlose Verbindung zwischen zwei Geräten oder zwischen einem Gerät und einem NFC-Tag aufzubauen, indem diese nahe aneinander gehalten werden. Die Reichweite beträgt dabei wenige Zentimeter. Sobald sich zwei Partner nahe genug kommen, wird die Verbindung hergestellt. NFC will durch diese Limitierung Lauschangriffe erschweren. Deshalb wird NFC in verschiedensten Gebieten eingesetzt, wie beispielsweise, um Zahlungen zu tätigen, Schlüssel für andere Verbindungen (zum Beispiel Wi-Fi oder Bluetooth) zu übermitteln oder auch kleinere Dateien schnell und einfach zu übertragen.[6]

Android-Geräte, die NFC unterstützen, besitzen drei verschiedene Betriebsarten:

1. Lese-/Schreibmodus: Ermöglicht es dem Gerät, NFC-Tags oder Sticker zu lesen und/ oder zu schreiben.

2. P2P-Modus: Ermöglicht es dem Gerät, Daten mit anderen gleichstehenden Geräten auszutauschen. Dieser Modus benutzt Android Beam.

3. Kartenemulationsmodus: Ermöglicht es dem Gerät, selbst als NFC-Karte zu agieren. Die emulierte Karte kann dann von anderen externen NFC-Lesern gelesen werden.[7]

---

6 Vgl. CURRAN 2012, 371
7 Vgl. ANDROID, *Near Field Communication*

In NameMemo hat man die Möglichkeit, die Datenbankdatei der App auf ein anderes Gerät, auf dem die Anwendung läuft, zu übertragen, um dann auf dem Zielgerät weitere Operationen damit zu starten. Somit wird hier Betriebsart 2 benutzt, die Datei wird also mithilfe der Android Beam Dateiübertragungs-API übertragen. Um diese Funktion nutzen zu können, werden zwei NFC-fähige Geräte mit Android 4.1 oder höher benötigt. Diese beiden Kriterien werden daher im *onCreate()* der Activity geprüft, die für die Datenübertragung via NFC zuständig und für beide Partner gleich ist. Danach wird in *onStart()* die Aktivierung von NFC und Android Beam durch den Nutzer erzwungen. Da es nicht möglich scheint, ein Feedback über den aktuellen Aktivitätszustand zu bekommen, wird der Dialog zur Aktivierung solange wieder erscheinen, bis der Nutzer die notwendigen Features aktiviert oder die App beendet. Sind dann beide Geräte erfolgreich in der Activity angekommen, wird *NfcAdapter.setBeamPushUris(datenbankdatei, this)* aufgerufen. Dies führt dazu, dass nun eben die Datenbankdatei bei einer Übertragung gesendet wird. Um jetzt eine Übertragung zu beginnen, müssen die beiden Geräte „Rücken an Rücken“ gebracht werden und auf dem gewünschten Sender muss der Bildschirm berührt werden. Nachdem die Datei übertragen wurde, können die Geräte voneinander entfernt werden. Jetzt muss nur noch, da es auch hier kein Feedback gibt, bei dem Empfänger auf den Button gedrückt werden[8], um den in Punkt 4.3.3 beschriebenen Vorgang zu starten.

5.2 Bluetooth

Der Kern der Bluetooth-Spezifikation definiert die Technologie als Baustein für Entwickler, die damit interoperable Geräte produzieren und somit zu einem florierenden Bluetooth-Ökosystem beitragen können. Sie wird von der Bluetooth Special Interest Group (SIG) überwacht und von den Bluetooth SIG Working Groups regelmäßig aktualisiert und verbessert, um sich dem sich entwickelnden Markt anzupassen. Man unterscheidet grundlegend zwischen zwei Varianten von Bluetooth: *Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR)*, verabschiedet als Version 2.0/2.1 und *Bluetooth with low energy (LE)*, verabschiedet als Version 4.0/4.1/4.2. Die beiden Implementationen sind für verschiedene Anwendungsgebiete optimiert und benutzen unterschiedliche Chipsets: Bluetooth BR/EDR erzeugt eine anhaltende Verbindung über kurze Distanz, Bluetooth LE

8 Vgl. Abb. 1.10 (Anhang)

ermöglicht kurze Stöße von Funkverbindungen über längere Distanz und schont somit eine eventuell vorhandene Batterie. Geräte wie Smartphones oder Tablets arbeiten im *Dual Mode*, können sich also sowohl mit BR/EDR- als auch mit LE-Geräten verbinden.[9]

Somit wird Bluetooth auch von Android unterstützt, wobei der Zugriff auf die Funktionen durch die *Android Bluetooth APIs* geregelt wird. Mit diesen APIs kann eine Anwendung auf Android unter anderem nach anderen Bluetooth-Geräten suchen, gekoppelte Geräte mithilfe des Bluetooth-Adapters anzeigen, sich mit anderen Geräten über die *Network Service Discovery* verbinden und mit verbundenen Geräten Daten austauschen. Bluetooth LE wird dabei erst ab Android 4.3 von der API unterstützt.[10]

In NameMemo gelangt man, nach entsprechender Auswahl, in die *BluetoothActivity*, wo in *onCreate()* geprüft wird, ob das Gerät Bluetooth unterstützt. Ist dies der Fall, wird in *onStart()* kontrolliert, ob Bluetooth auch aktiviert ist und der Nutzer gegebenenfalls durch einen Dialog dazu aufgefordert, dies nachzuholen. Wird die Aktivierung abgelehnt, kehrt man wieder zur MainActivity zurück. Ist man allerdings erfolgreich in der BluetoothActivity angekommen, wird ein neuer „Server“ gestartet und es wird gleichzeitig eine Liste mit gekoppelten Geräten angezeigt. Alternativ hat man auch die Möglichkeit, nach neuen Geräten zu suchen, wozu man sie erst sichtbar schalten muss. Wird ein Eintrag aus der Liste ausgewählt, wird der eigene Server geschlossen und es wird ein „Client“ erstellt, welcher dann versucht, mit dem Server der anderen Partei eine Verbindung herzustellen.

Da eine Verbindung via Bluetooth keine Server-Client-, sondern eine Peer-to-Peer-Verbindung ist, werden in dem Sinne keine echten Server oder Clients erstellt. Die verwendeten Begriffe dienen nur zur Veranschaulichung der eingenommenen Rolle. So wird im *onStart()* der Activity ein Thread gestartet, welcher nach einer eingehenden Verbindung lauscht (*AcceptThread*). Tippt der Nutzer aber auf einen Eintrag in der Liste der nahen Geräte, wird dieser Thread geschlossen und es wird stattdessen einer gestartet, dessen Aufgabe es ist, mit dem AcceptThread einer anderen Instanz Kontakt aufzunehmen und eine Verbindung aufzubauen (*ConnectThread*). Diese Verbindung besteht dann in einem weiteren Thread, welcher auf beiden Verbindungspartnern läuft und den weiteren

9 Vgl. BLUETOOTH
10 Vgl. ANDROID, *Bluetooth*

Datenaustausch ermöglicht (*HandleThread*).

Wurde eine Verbindung hergestellt und der HandleThread gestartet, so erscheint statt der oben genannten Liste nun auf beiden Geräten ein Button zum Senden, welcher nun von dem entsprechenden Part gedrückt werden soll[11]. Ist dies geschehen, wird in der sendenden Instanz die Datenbankdatei zu einem Byte-Array umgewandelt und anschließend in Paketen der Größe 8192 Bytes, also 8 Kilobytes, an das Gegenüber gesendet. Diese Aufteilung erfolgt, da es zu Komplikationen kommen kann, falls zu große Datenpakete auf einmal über Bluetooth verschickt werden. Der empfangende Part liest den ankommenden Stream währenddessen aus und erstellt eine Datei, die der ursprünglich gesendeten entspricht. Ist der Vorgang abgeschlossen, wird die Activity des Senders beendet und der Empfänger verfährt wie in Punkt 4.3.3 beschrieben.

5.3 Server-Client

Eine weitere Möglichkeit zum Verschicken der Daten wäre es, einen Server zu hosten, mit welchem sich zwei Clients verbinden und Daten austauschen können. Dies wurde in NameMemo nicht umgesetzt; im Folgenden wird also nur eine mögliche Vorgehensweise beschrieben.

Auf dem Server läuft eine Software, welche auf HTTP-Anfragen reagieren kann. Vergleichbar ist dies mit einem Webserver, der eine Internetseite, wie zum Beispiel http://www.google.com, hostet. Nun wird hier allerdings keine, mit einem Webbrowser aufrufbare, graphische Seite, sondern lediglich eine einfache, textbasierte Möglichkeit mit dem Server zu kommunizieren, benötigt. Dies ist in Java beispielsweise mithilfe von *Jetty* möglich. Jetty ist ein auf Java basierter Webserver, welcher in einen Server eingebettet werden kann.[12]

Um die Übertragung zu instanziieren, senden beide Android-Geräte einen *Request* an den Server, damit dieser erkennt, dass sie zum Datenaustausch bereit sind. Als *Response* schickt der Server dann eine Liste der anderen aktiven Geräte zurück, wo sich der Benutzer nun das gewünschte Ziel aussuchen kann. Wird der Wunsch abgesetzt, einem bestimmten Gerät Daten zu schicken oder von ihm zu empfangen, muss dies erst durch den

11 Vgl. Abb. 1.9 (Anhang)
12 Vgl. PABST

ausgewählten Client bestätigt werden, um unerwünschten Datenaustausch zu vermeiden. Haben sich nun zwei Partner gefunden, können sie mithilfe weiterer Requests und Antworten darauf Dateien und andere Informationen verschicken. Ein Request ist dabei eine Zeichenkette, in welche die Datei umgewandelt werden und aus welcher die Datei am Ziel wieder erstellt werden muss. Das kann wie oben durch ein Byte-Array oder durch eine Umwandlung der Daten in *XML* erfolgen.

5.4 Vergleich und Fazit

Kommen wir nun zu einem Vergleich der drei oben genannten Varianten und damit zu einem Fazit, beziehungsweise dem Grund der Auswahl der ersten beiden für das Projekt.

NFC. NFC ist die für den Benutzer und für den Programmierer einfachste Möglichkeit, die Datenbankdatei zu verschicken. Es müssen lediglich die beiden Geräte aneinandergehalten und der Bildschirm des Senders berührt werden. Dadurch besteht allerdings die Einschränkung der Reichweite auf circa vier Zentimeter, wodurch wiederum ein sehr hohes Sicherheitsniveau gewährt wird. Eine weitere Einschränkung ist die, dass nicht alle Geräte NFC und Android Beam unterstützen; dieses Problem betrifft vor allem Geräte mit einer Android-Version kleiner als 4.1. Durch *Android Beam* ist darüber hinaus die Größe der Datei, welche verschickt werden kann, limitiert und für *S-Beam*, welches für größere Dateien gedacht ist, steht keine API bereit. Dies ist kein Problem für die Datenbankdatei von NameMemo, wohl aber für das zusätzliche, automatische Verschicken von Bildern, das in der App nicht umgesetzt wurde.

Bluetooth. Bluetooth ist im Vergleich zu NFC etwas komplexer: Der gewünschte Verbindungspartner muss erst gesucht werden, wozu er sich sichtbar schalten muss. Dabei kann es problematisch sein, wenn sich mehrere sichtbare Geräte mit gleichem oder ähnlichen Anzeigenamen in der Umgebung befinden. Die Reichweite ist mit zehn bis 100 Metern[13] deutlich höher als bei NFC, was es aber auch potentiellen Angreifern deutlich leichter macht, die Verbindung zu belauschen oder zu manipulieren. Mithilfe von Software wie *Bloover*, *BackTrack* oder *BTCrack* lassen sich leicht eventuell vorhandene Lücken in der Bluetooth-Architektur und anderem Code auf dem Handy ausnutzen.[14] Bluetooth ist auf Handys schon länger verbreitet als Android, weshalb man heutzutage so gut wie immer

13 Vgl. KARBACHER
14 Vgl. BECKER, 9 f. und 22

davon ausgehen kann, dass es auch auf den zwei Geräten unterstützt wird, die synchronisiert werden sollen. Auch gibt es keinerlei Einschränkungen bezüglich einer Dateigröße, da Dateien sowieso nicht im Ganzen übertragen werden (siehe oben).

Server-Client. Diese Herangehensweise ist, verglichen mit den anderen beiden, bei weitem die komplexeste, da man für die Umsetzung als erstes einen Server hosten und warten muss. Zudem müssen sich beide Clients mit dem Server verbinden können, sei dies über ein lokales Netzwerk oder über das Internet. Letzteres kann darüber hinaus zusätzliche Kosten für die Clients verursachen oder kann bei Geräten wie Tablets gar nicht erst möglich sein. Des weiteren ist eine „Standard“-Verbindung über HTTP nicht gegen Mithören gesichert und sollte daher abgesichert werden, zum Beispiel via HTTPS. Ein Vorteil der Server-Client-Kommunikation ist die Reichweite, welche bei einem Server im Internet oder Nutzung eines VPNs unbegrenzt hoch sein kann, gegeben den Fall, dass man sich mit dem Netz verbinden kann. Zwar gibt es bei dieser Methode technisch kaum eine Datenbeschränkung, jedoch ist die Wartezeit zu beachten, welche mit dem Down- beziehungsweise Upload von großen Datenpaketen einhergeht.

Fazit. Der Datenaustausch von NameMemo ist nicht über weitere Distanzen vorgesehen. Zudem soll die Anwendung ohne einen Server auskommen und somit eine potentiell längere Lebensdauer haben. Außerdem richtet sich die App an Benutzer mit unterschiedlicher technischer Erfahrung. Aus diesem Grund wurde die Idee eines Server-Client-Modells als nicht zielführend betrachtet und verworfen. Mit NFC wird eine besonders einfache und sichere und mit Bluetooth eine besonders kompatible Möglichkeit, mehrere Geräte synchron zu halten, implementiert. Dem Nutzer wird empfohlen, nach Möglichkeit NFC zu verwenden und ansonsten auf Bluetooth auszuweichen. Der Gedanke, Bilder automatisch zu synchronisieren, wurde auf Grund der Komplexität und dem mangelndem Nutzen ebenfalls nicht umgesetzt, da das Senden von Bilddateien auch mit „Android-Hausmitteln“ einfach zu bewerkstelligen ist.

6. Entscheidungen während des Projekts

Wie bei jedem Projekt, wurden auch bei diesem verschiedene Entscheidungen getroffen, welche im weiteren Verlauf teilweise auch wieder verworfen oder geändert wurden. Im Folgenden werden deshalb die wichtigsten Entschlüsse dargelegt.

6.1 Strukturierung in Activities und Dialoge

NameMemo besteht aus sechs Activities und dreizehn Dialogen. Dieses Verhältnis ist nicht zufällig, sondern wurde aus verschiedenen Gründen bewusst gewählt:

Ein Anlass für diese Gestaltung ist die *Vorbeugung von Orientierungslosigkeit* beim Navigieren durch die App, da man bei den Dialogen stets die aktuelle Activity im Hintergrund sieht und einfach zu dieser zurückkehren kann, indem man diese antippt. Ein weiterer Grund ist die Simplizität vieler Dialoge, wodurch es sich beispielsweise für den Dialog zum Anzeigen der Verbindungsmöglichkeiten nicht lohnen würde, eine neue Activity zu starten. Darüber hinaus entsteht so eine Gliederung der Bedienungsmöglichkeiten, wie man an dem Diagramm[15] gut erkennen kann: Alles, was mit Datenaustausch mit anderen Systemen zu tun hat, ist klar von dem lokalen Part abgegrenzt; auch das Hinzufügen von Bildern auf lokaler Ebene ist separiert. Dies fördert die Übersichtlichkeit, sowohl des Codes, als auch für den Nutzer. Zudem sind einige Activities und Dialoge von Android vorgegeben und können somit nicht in ihrer Darstellungsform manipuliert werden. Alle diese Punkte führen also zu der in NameMemo gegebenen Struktur.

6.2 Gestaltung der Actionbar

Die Actionbar, auch Appbar genannt, ist die Leiste oben auf dem Bildschirm. Sie ist bei NameMemo speziell gestaltet. Normalerweise könnte man den Namen der App, beziehungsweise der Activity, das Icon der App und ein bis zwei Symbole bei einem normal großen Handy anordnen. Bei der hier verwendeten Anwendung jedoch wird in der MainActivity auf ein Appicon und eine Beschriftung verzichtet. Damit wurde Platz für die vier wichtigsten Menüsymbole geschaffen. Die Icons hierfür sind zum Teil von Android und zum Teil nachbearbeitete Symbole aus dem Internet. Es wird ein grau-auf-schwarzes

15 Abb. 2.3 (Anlage)

Design für die Actionbar umgesetzt. Ein Grund für das Legen der Menüelemente in die Actionbar ist die einfachere Erreichbarkeit als im *Action Overflow*, ein anderer ist eine Verbesserung des Designs. Um die vier Einträge alle anzeigen zu können, muss in der *menubar.xml* für die entsprechenden Items der Eintrag *app:showAsAction* auf *always* gesetzt werden, da Android bei einer Verwendung von *ifRoom* mehr Einträge in den Overflow verschiebt als nötig.

6.3 Fixierung der Orientierung

Eine Android-App kann allgemein verschiedene Ausrichtungen beziehungsweise Orientierungen haben. Einige Möglichkeiten sind dabei *landscape* („auf der Seite liegend“), *portrait* („aufrecht stehend“), *reverseLandscape* (landscape „von der anderen Seite“) oder *reversePortrait* (portrait „von der anderen Seite“).[16] Ändert sich die Ausrichtung einer App, wenn man zum Beispiel das Gerät dreht, wird die aktuelle Activity, an die neuen Gegebenheiten angepasst, neugestartet. Dadurch gehen lokale Änderungen an dieser verloren, wenn sie nicht durch spezielle Maßnahmen gesichert wurden.

So kann der aktuelle Zustand einer Activity oder App mithilfe eines *Statecontrollers* gespeichert werden, welcher jede Zustandsänderung aufzeichnet und bei Bedarf (Neustarten der Activity) den aktuellen Zustand zurückgibt. Weitere Möglichkeiten sind das Benutzen von *SharedPreferences*, einem Interface, das Zugriff auf und Modifikation von Einstellungsdaten erlaubt[17], oder das Speichern von wichtigen Werten in einer Datenbank, wie es in NameMemo auch zum Teil umgesetzt wird.

Allerdings ist die Orientierung von NameMemo auf *portrait* begrenzt, da eine Unterstützung der anderen Modi nicht sinnvoll erscheint. Dies liegt vor allem an der Tatsache, dass die App vor allem für Bilder von einzelnen Personen gedacht ist, welche meistens im portrait-Mode aufgenommen werden. Zusätzlich würde das ständige Updaten des Zustands unter Verwendung eines Statecontrollers die Komplexität des Codes aufgrund der vielen Dialoge, deutlich erhöhen. In einer früheren Version der Anwendung wurde dieser Weg bereits getestet, dann aber aufgrund der oben genannten Punkte verworfen und die Orientierung in *AndroidManifest.xml* auf *portrait* festgesetzt.

16 Vgl. ANDROID, <*activity*>
17 Vgl. ANDROID, *SharedPreferences*

7. Ausblick

Wie bereits in der Einleitung erwähnt, kann die App in leicht veränderter Form auch zu anderen Zwecken eingesetzt werden. Dazu gehört die Diagnostik und Therapie von Autismus.

„Autistische Störungen gehören zu den schwersten psychischen Problemen des Kindesalters.“[18] Kinder mit autistischen Symptomen wurden erstmals vor etwas über 100 Jahren von dem Pädagogen und Leiter der *Erziehungsanstalt für geistig abnorme und nervöse Kinder in Wien,* Theodor Heller, beschrieben.[19] Während Autismus lange als eine sehr seltene Erkrankung galt[20], werden autistische Störungen in den letzten Jahrzehnten immer häufiger diagnostiziert. Aktuell geht man davon aus, dass etwa 1% der Bevölkerung an einer autistischen Störung leidet.[21] Aktuelle wissenschaftliche Forschungen haben ergeben, dass Autisten typische Probleme mit der sozialen Kognition haben. Soziale Kognition umfasst dabei eine Reihe von Informationsverarbeitungsprozessen, die im Gehirn in spezifischen neuronalen Netzwerken – man spricht von *social brain* – bei zwischenmenschlichen Kontakten stattfinden, wie beispielsweise die Aufmerksamkeitszuwendung und die Emotionserkennung.[22]

Die App kann dabei zum einen im Rahmen einer Autismus-Diagnostik eingesetzt werden, um Hinweise zu bekommen, ob ein Kind Probleme mit der Erkennung von Emotionen hat. Zum anderen kann damit in der Therapie das Erkennen von Emotionen getestet und trainiert werden, womit bei Erfolg der Alltag für den Patienten deutlich erleichtert wird.

Dies wird erreicht, indem Fotos von Gesichtern aufgenommen werden, bei welchen der Patient anschließend die mimisch ausgedrückten Emotionen erkennen soll. Hierbei ist die Funktion, mithilfe der App Fotos aufzunehmen und direkt in die Datenbank zu übernehmen, besonders hilfreich. Auch die Möglichkeit, mehrere Geräte auf demselben Stand zu halten, ist von Vorteil, da somit das Handy des Nutzers mit dem Gerät des behandelnden Therapeuten synchronisiert werden kann.

Es ändern sich allerdings auch einige Anforderungen an das Programm: Zu Beginn der

18 POUSTKA 2008, Einleitung
19 Vgl. HELLER
20 Vgl. NOTERDAEME 2010, 31
21 Vgl. AWMF, 22
22 Vgl. SCHWENCK 2014, 5 ff.

Arbeit mit den Patienten ist es sinnvoll, eine für alle gleichbleibende Grundlage an Bildern gegeben zu haben, um Ergebnisse einfacher miteinander vergleichen zu können. Somit werden also beim ersten Start der App bereits einige Datensätze in die Datenbank eingetragen sein (vergleichbar mit den aktuellen Beispielbildern). Diese Bilder enthalten dann besonders einfach zu erkennende Darstellungen, wie beispielsweise Emojis oder Emoticons, wodurch auch die Größe der Anwendung nicht zu sehr zunimmt. Im Laufe der Therapie können dann, wie jetzt auch, individuell Datensätze hinzugefügt werden.

Beim Abarbeiten der Einträge bieten sich ebenfalls neue Vorgehensweisen an: Der *Glückwunschdialog*, welcher erscheint, sobald man in diesem Fall eine Emotion korrekt betiteln konnte[23], soll zur besseren Nachverfolgung nicht die Möglichkeit bieten, den entsprechenden Eintrag zu löschen, sondern lediglich einen motivierenden Highscore anzeigen. Zu diesem Zweck ist auch das Speichern zusätzlicher Daten in der Datenbank vonnöten, wie die höchste bisherige Kette an richtigen Zuordnungen. Darüber hinaus ist das Sichern der Gesamtzahl der bisherigen korrekten Zuordnungen eines Eintrages neben der Zahl der Aufrufe insgesamt hilfreich, um eine Angabe über die relative Häufigkeit richtiger und falscher Zuordnungen produzieren zu können. Solche Werte können dann, in einer Statistik aufbereitet, ausgegeben werden.

Je nachdem, wie verlässlich der Patient ist, kann es sinnvoll sein, Nebenfunktionen der Anwendung, wie das Ändern von Einstellungen, nur der behandelnden Kraft zugänglich zu machen. Dies wird durch einen zusätzlichen Schalter in den Einstellungen erreicht, welcher mit einem *PIN-Code* gesichert wird. Um möglichst aussagekräftige Ergebnisse zu erzielen, kann es darüber hinaus sinnvoll sein, dass der Nutzer zu jedem Bild die zugehörige Emotion in ein Textfeld eingeben muss und nicht einfach mithilfe von Buttons zum nächsten Bild zu gelangt. Dieser Schritt schützt nicht nur vor Manipulationen des Patienten: man kann somit auch die inkorrekten Eingaben abspeichern und später auswerten.

Um die Schwierigkeit zusätzlich zu erhöhen, bietet es sich noch an, nur die obere oder untere Gesichtshälfte und damit nur Augen- oder Mundpartie zu zeigen.

23 Vgl. Abb. 1.3 (Anhang)

8. Schluss

Wie man nun sehen konnte, bietet meine App durchaus viele verschiedene Anwendungsmöglichkeiten, die auch bereits nachgefragt werden: Im Laufe der Arbeit an diesem Programm haben sich weitere Personen gemeldet, welche sich das Lernen von Namen mit dieser Anwendung einfacher machen wollen. Doch nicht nur deshalb wird NameMemo in der nächsten Zeit nicht in Vergessenheit geraten. Es ist bereits fest geplant, einen Ableger davon – *NEmo (Neuburger Emotionserkennungstraining)* – zu entwickeln und einzusetzen. Falls Sie nun auch Interesse an dem Programm gefunden haben, können Sie mich jederzeit unter der Mailadresse im Infodialog anschreiben und um eine Kopie der kostenlosen Version des Programms bitten.

9. Danksagungen

Danken möchte ich Dominik Okwieka für seine kompetente fachliche Unterstützung und sein Lektorat. Danken möchte ich weiter Simon Stolz und den Usern von StackOverflow für das Bereitstellen vieler hilfreicher Tipps. Besonderer Dank gilt meinem Vater, Dr. med. Jürgen Dreier, Leiter der Spezialambulanz für Familien mit autistischen Kindern und Jugendlichen an der Klinik für Kinder- und Jugendpsychiatrie und Psychotherapie der Kliniken St. Elisabeth in Neuburg an der Donau, für die Unterstützung in Sachen Autismus und meiner Mutter Roswitha Dreier, Diplompsychologin, die mich erst zu einer Ausformulierung dieser Danksagung bewegt hat.

10. Anhang 1: Screenshots

Abb. 1.1: Hauptbildschirm 1

Abb. 1.2: Hauptbildschirm 2

Abb. 1.3: Glückwunschdialog

Abb. 1.4: Bild hinzufügen

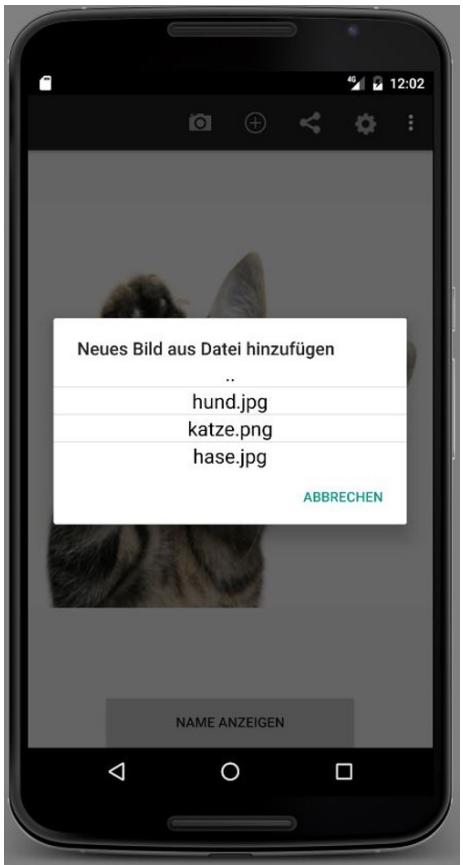


Abb. 1.5: Einstellungen

Abb. 1.6: Appinfo

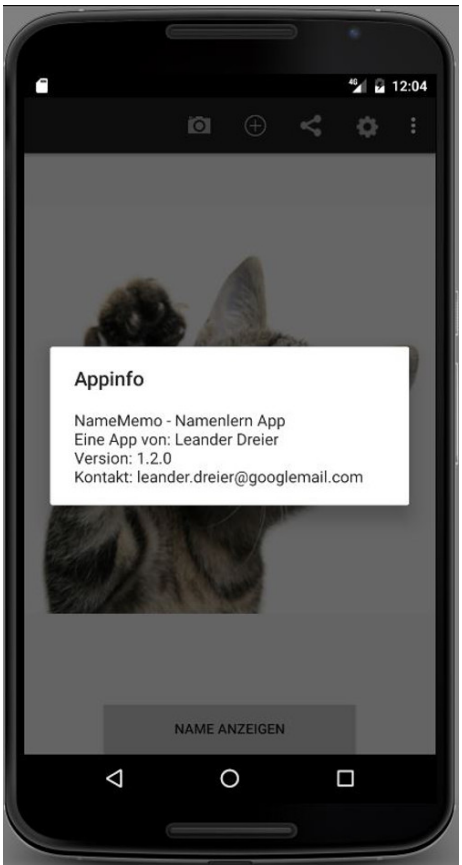


Abb. 1.7: Datenübertragung

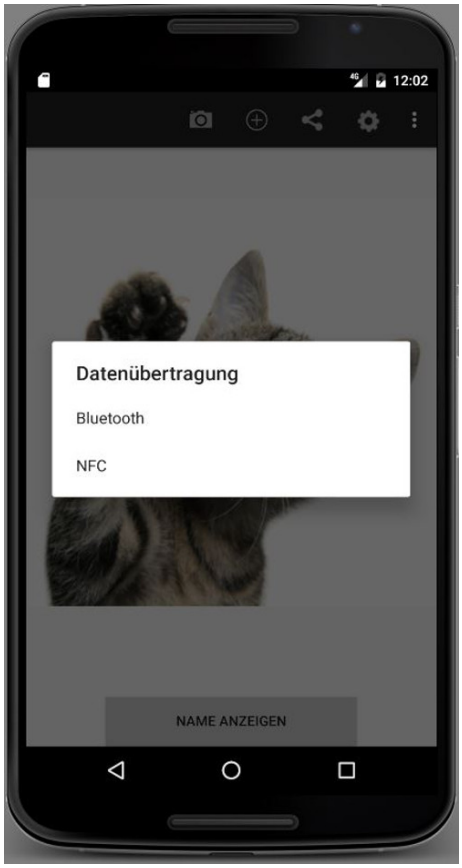


Abb. 1.8: Bluetooth 1

Abb. 1.9: Bluetooth 2

Abb. 1.10: NFC

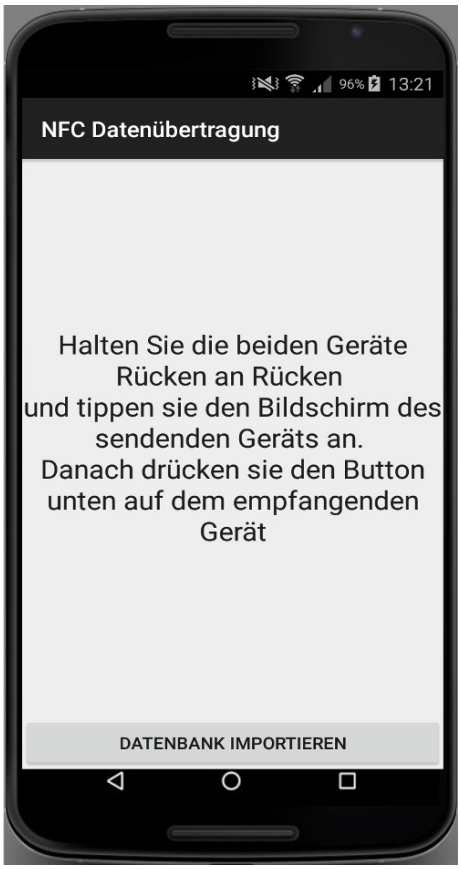


Abb. 1.11: Import 1

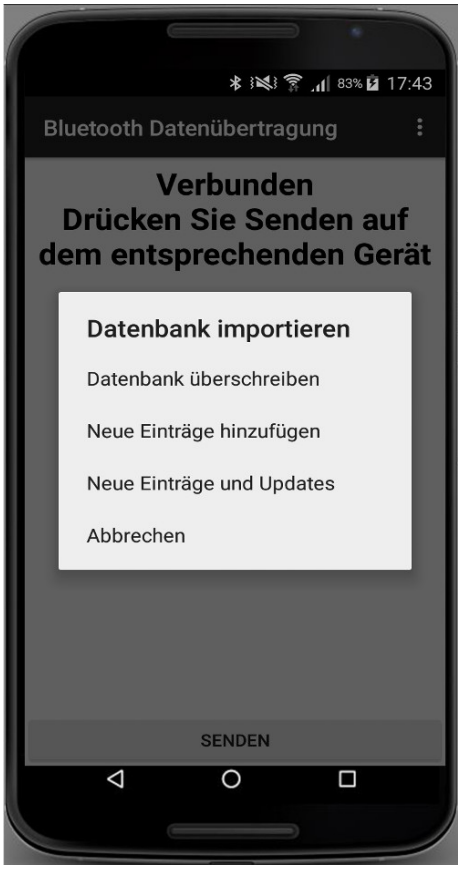
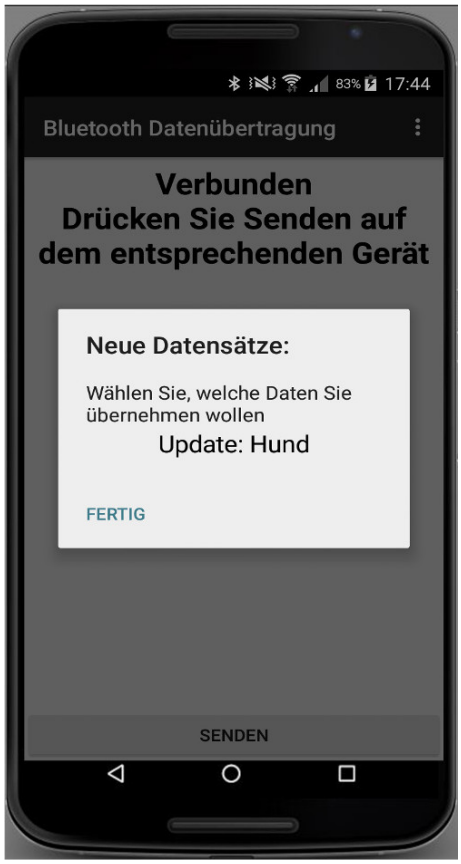


Abb. 1.12: Import 2



11. Anhang 2: Diagramme

Abb. 2.1: Strukturübersicht

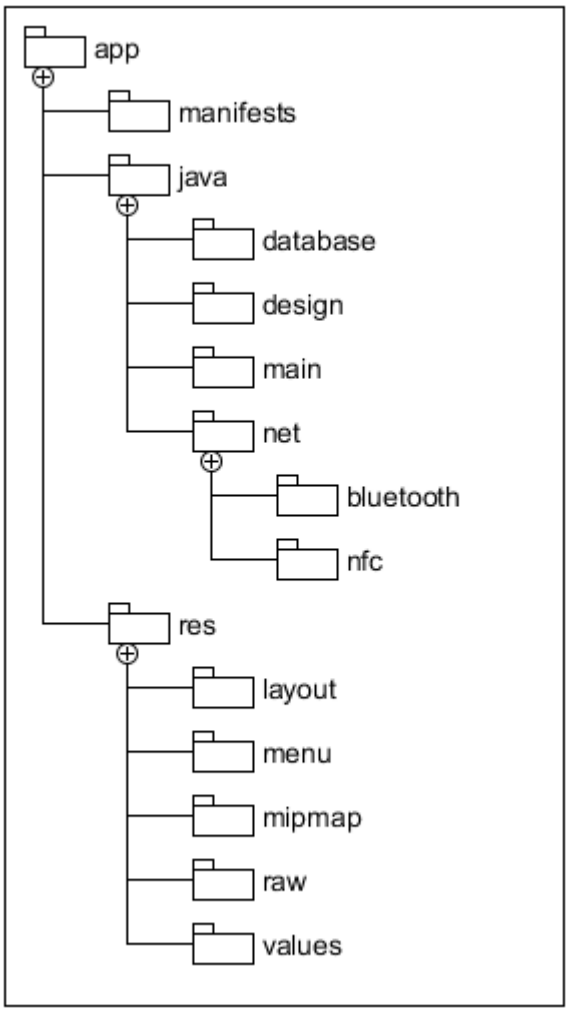
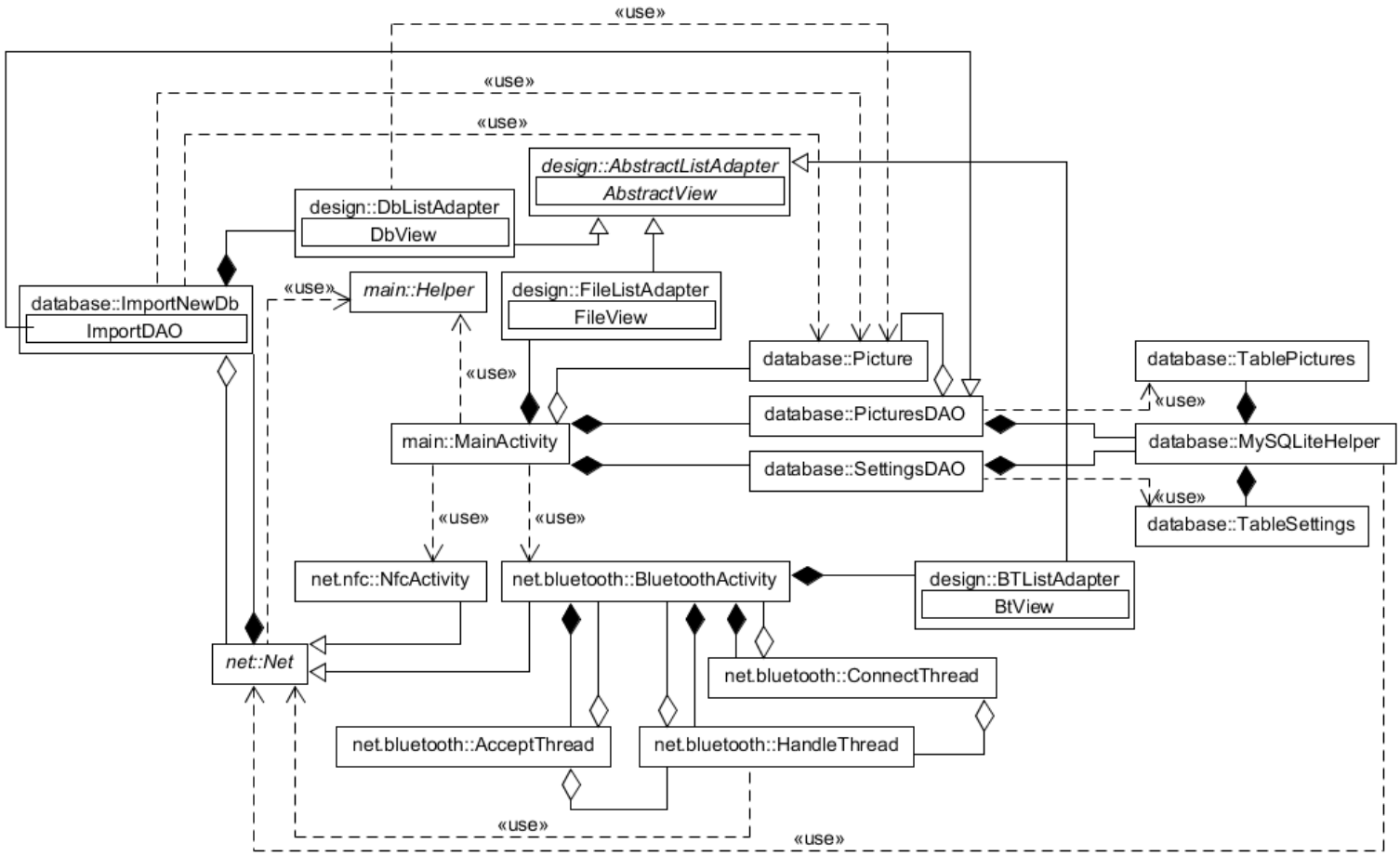


Abb. 2.2: Klassendiagramm



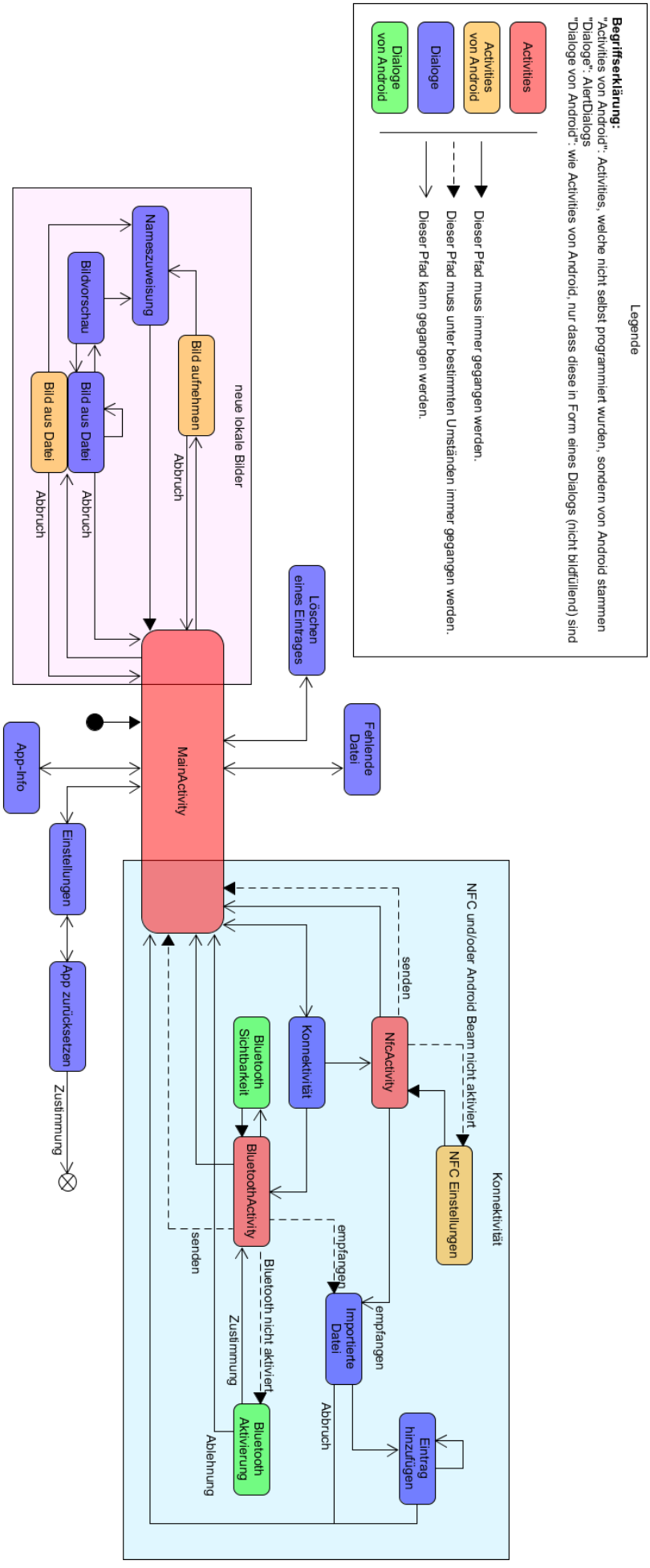


Abb. 2.3: Activities und Dialoge

12. Literaturverzeichnis

ANDROID Open Source Project: *<activity>*, https://developer.android.com/guide/topics/manifest/activity-element.html, aufgerufen am 23.10.2016

ANDROID Open Source Project: *Action Bar*, https://developer.android.com/design/patterns/actionbar.html, aufgerufen am 18.10.2016

ANDROID Open Source Project: *Bluetooth*, https://developer.android.com/guide/topics/connectivity/bluetooth.html, aufgerufen am 07.10.2016

ANDROID Open Source Project: *Display*, https://developer.android.com/reference/android/view/Display.html, aufgerufen am 20.09.2016

ANDROID Open Source Project: *Near Field Communication*, https://developer.android.com/guide/topics/connectivity/nfc/index.html, aufgerufen am 01.10.2016

ANDROID Open Source Project: *SharedPreferences*, https://developer.android.com/reference/android/content/SharedPreferences.html, aufgerufen am 23.10.2016

AWMF: *Autismus-Spektrum-Störungen im Kindes-, Jugend- und Erwachsenenalter – Teil 1: Diagnostik*, http://www.awmf.org/uploads/tx_szleitlinien/028-018l_S3_Autismus-Spektrum-Stoerungen_ASS-Diagnostik_2016-05.pdf, aufgerufen am 01.11.2016

BECKER, Andreas: *Bluetooth Security & Hacks*, https://gsyc.urjc.es/~anto/ubicuos2/bluetooth_security_and_hacks.pdf, vom16.08.2007, aufgerufen am 09.10.2016

BLUETOOTH SIG, Inc.: *Bluetooth Core Specification*, https://www.bluetooth.com/specifications/bluetooth-core-specification, aufgerufen am 07.10.2016

BUMP Technologies: *Glide*, https://github.com/bumptech/glide/blob/master/README.md vom 13.07.2016, aufgerufen am 16.09.2016

CUNO, Andrea: *Near Field Communication*, https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2010-08-2/NET-2010-08-2_01.pdf, aufgerufen am 29.09.2016

CURRAN, Kevin; Millar, Amanda; Garvey, Conor Mc.: *International Journal of Electrical and Computer Engineering,* Vol. 2 No. 3, vom Juni 2012, Seite 371 (http://search.proquest.com/openview/822cf597b6e8dab49e0eb29493af13e9/1)

ECLIPSE Foundation: *jetty://*, http://www.eclipse.org/jetty/, aufgerufen am 08.10. 2016

ECMA International: *Near Field Communication - Interface and Protocol (NFCIP-1),* 3. Auflage, Juni 2013 (http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-340.pdf)

HELLER, Theodor: *Über dementia infantilis*, In: Z Erforsch Behandl Jugndl Schwachsinns 2, S. 17–28, zitiert nach der S3-Leitlinie „Autismus-Spektrum-Störungen im Kindes-, Jugend- und Erwachsenenalter“ der Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften AWMF, 2016, S. 10

KARBACHER: *Bluetooth Sendeleistung und Reichweite*, http://www.karbacher.org/lexikon/bluetooth-sendeleistung-und-reichweite/, aufgerufen am 09.10.2016

NOTERDAEME, Michele: *Autismus-Spektrum-Störungen (ASS). Ein integratives Lehrbuch für die Praxis*, Kohlhammer 2010

PABST, Martin: *Cient-Server-Kommunikation von Android (Client) zu Java (Server) via http*, http://www.pabst-software.de/doku.php?id=programmieren:java:android:httpclient:start, aufgerufen am 08.10.2016, zuletzt geändert am 05.02.2015 um 15:06

POUSTKA, Fritz; Bölte, Sven; Feineis-Metthews, Sabine; Schmötzer, Gabriele: *Leitfaden Kinder- und Jugendpsychiatrie, Band 5: Autistische Störungen*, aktualisierte Auflage, Hogrefe 2008

SCHWENCK, Christina; Ciaramidaro, Angela: *Soziale Kognition bei Autismus-Spektrum-Störungen und Störungen des Sozialverhaltens*, in: Kindheit und Entwicklung, 23 (1), von 2014, S. 5-12

13. Verwendete Programme

Android Studio und SDK Tools

Eclipse

GIMP - GNU Image Manipulation Program

LibreOffice

Pandoc

UMLet

14. Eidesstattliche Erklärung

Ich erkläre, dass ich die Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis aufgeführten Quellen und Hilfsmittel verwendet habe.

Ingolstadt, den 07. November 2016

_____

(Leander Dreier)