



Entwicklung einer Foto-App für Android

Studienarbeit

im Studiengang Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Tobias Dorra und Philipp Pütz

November 2015

Namen

Tobias Dorra, Philipp Pütz

Matrikelnummern

6139224, 5408001

Bearbeitungszeitraum

4 Wochen

Kurs

TINF14-AIBC

Semester

3. Semester

Inhaltsverzeichnis

Abbildungsverzeichnis	i
1 Einführung	1
1.1 Funktionen	1
1.2 Projektumfang	2
2 Architektur	6
2.1 Model	8
2.2 Controller	9
2.3 View	10
3 Benutzeroberfläche	11
3.1 Beschreibung der Benutzeroberfläche	11
3.2 Detailansicht eines Bildes	14
3.3 Der Floating Action Button	15
3.4 Verwendete Libraries	17
4 Implementierung	19

Abbildungsverzeichnis

2.1 Übersicht über das Zusammenspiel der wichtigsten Komponenten	7
3.1 App-Startansicht	11
3.2 Schematische Darstellung	11
3.3 Navigation Drawer	12
3.4 Schematische Darstellung	12
3.5 Streamansicht	13
3.6 Schematische Darstellung	13
3.7 Kontextmenü - CardView	14
3.8 Detailansicht eines Bildes	14
3.9 Detailansicht mit Action Bar	15
3.10 Share-Provider	15
3.11 Floating Action Menu	16
3.12 Kamera-Intent	16
3.13 Bildtitel festlegen	16
3.14 Content-Provider zum Auswählen eines Bildes	17

1 Einführung

Bei gemeinsamen Freizeitaktivitäten mit Freunden entstehen oft viele Fotos. Diese werden dann entweder gar nicht oder erst Wochen später miteinander geteilt.

Unser Ziel ist es, die Benutzererfahrung beim Teilen der Bilder zu verbessern. Das soll mit einer eigenen Foto-App passieren. Anstatt die gesammelten Bilder erst nach der Veranstaltung auszuwählen und dann an die entsprechenden Kontakte zu schicken, sollen die geschossenen Bilder bereits während der Veranstaltung automatisch mit den anderen anwesenden Freunden geteilt werden.

1.1 Funktionen

1.1.1 Organisation der Fotos

Die Fotos werden in sogenannten „Streams“ gruppiert. Jeder Stream repräsentiert dabei ein gemeinsames Erlebnis mit Freunden. Streams können umbenannt oder gelöscht werden. Beim Löschen eines Streams werden alle enthaltenen Bilder ebenfalls gelöscht.

Um einen Stream mit Fotos zu befüllen, kann entweder die Gerätekamera genutzt werden, oder es werden Fotos aus der Android-Galerie importiert. Die Fotos werden mit Name und Vorschaubild im Stream angezeigt. Außerdem existiert eine Detailansicht, in der man das Foto vergrößert betrachten kann. Fotos können umbenannt oder gelöscht werden. Ausgewählte Fotos können wieder zurück in die Android-Galerie exportiert, oder über die Teilen-Funktion anderen Apps zugänglich gemacht werden.

1.1.2 Teilen von Fotos

Zu einem Stream können mehrere Benutzer hinzugefügt werden. Die Fotos in einem Stream werden live zwischen allen Teilnehmern synchronisiert.

Alle Teilnehmer in einem Stream sind gleichberechtigt. Es gibt also keinen „Streamadministrator“ mit besonderen Rechten. Jeder Teilnehmer eines Streams kann Bilder hinzufügen, Bilder löschen und die Teilnehmerliste verwalten. Insbesondere ist jeder Teilnehmer auch dazu berechtigt, weitere Teilnehmer hinzuzufügen.

Entfernt ein Benutzer einen Stream, so bleibt der Stream bei den anderen Teilnehmern noch vorhanden. Bei diesen wird lediglich der Benutzer aus der Teilnehmerliste entfernt.

Wird ein Benutzer nachträglich zu einem Stream hinzugefügt, so erhält er nicht nur die neuen Bilder, sondern bekommt auch automatisch Zugriff auf alle Bilder, die bereits vor seinem Eintritt im Stream vorhanden waren.

1.1.3 Nutzung alternativer Methoden zur Datenübertragung

Wenn man gezwungen ist, das Mobilfunknetz zu benutzen, ist die Internetverbindung oft schlecht. Außerdem haben viele Smartphone-Nutzer ein begrenztes Datenvolumen. Daher sollen die Daten nicht über das Internet, sondern über alternative Verbindungen übertragen werden. Diese beinhalten:

1. Bluetooth
2. Bluetooth Low Energy (eventuell, falls technisch machbar)
3. Wi-Fi Direct

1.2 Projektumfang

1.2.1 Umgesetzte Funktionalitäten

Die Anwendung, die im Rahmen dieses Projektes entwickelt wurde, stellt lediglich einen Prototyp dar. Es werden also nicht alle beschriebenen Funktionen umgesetzt. Bei der Architektur wurde jedoch darauf geachtet, dass die Anwendung um die beschriebenen Funktionalitäten erweiterbar ist. Die folgenden Funktionalitäten wurden im Prototyp umgesetzt:

ID	Beschreibung
F-10	Bilder können zur App hinzufügt werden
F-10.1	Bilder können über die Kamera aufgenommen werden

ID	Beschreibung
F-10.2	Bilder können aus der Galerie geladen werden
F-20	Bilder können verwaltet werden
F-20.1	Bilder können gelöscht und umbenannt werden
F-20.2	Bilder haben verschiedene Attribute (Bildtitel, Aufnahmefinformationen)
F-30	„Streams“ (Ordner) gruppieren Bilder
F-30.1	Streams können erstellt, gelöscht und umbenannt werden
F-40	Bilder werden in einer listenähnlichen Darstellung mit Vorschaubild und Bildinformationen angezeigt
F-50	Bilder können in einer Vollbildansicht angezeigt werden
F-50.1	Aus der Vollbildansicht können Bilder in die Galerie exportiert und per E-Mail versendet werden
F-60	Die Architektur ist erweiterbar
D-10	Alle Appdaten (Streams, Bilder, Datenbanken) werden in einen speziellen Verzeichnis gespeichert
D-10	Gespeicherte Bilder erhalten als Zusatz das aktuelle Datum
D-20	Alle Bild- und Streaminformationen werden in einer Datenbank gespeichert
UI-10	Es existiert eine Hauptansicht
UI-10.1	Die Hauptansicht zeigt Vorschaubilder von aufgenommenen und aus der Galerie geladenen Bildern an
UI-10.2	Zu jedem Bild werden in der Hauptansicht der Bildtitel und Aufnahmefinformationen angezeigt
UI-10.3	Durch antippen der Bilder startet eine neue Activity die das Bild in einer Vollbildansicht anzeigt
UI-10.4	Durch einen „long tap“ auf ein Bild wird ein Kontextmenü angezeigt
UI-10.4.1	Im Kontext Menü stehen folgende Optionen zur Verfügung: Bild umbenennen, Bild löschen, Bild in Galerie exportieren

ID	Beschreibung
UI-10.5	Ein Floating-Action Button Menü beinhaltet die wichtigsten Aktionen
UI-10.5.1	Über das Menü können Bilder mit der Kamera aufgenommen werden
UI-10.5.2	Über das Menü können Bilder aus der Galerie geladen werden
UI-10.6	Ein „Navigation Drawer“ bietet die Möglichkeit neue „Streams“ anzulegen, die ähnlich wie Ordner, Bilder gruppieren
UI-10.6.1	Die Hauptansicht zeigt den Inhalt eines Streams an
UI-10.7	Über das Hauptmenü können Streams umbenannt werden
UI-10.8	Über das Hauptmenü können Streams und alle zugehörigen Bilder gelöscht werden
UI-10.9	Eine Standardansicht animiert den Nutzer zum Anlegen eines Streams, wenn kein Stream angelegt ist
UI-20	Es existiert eine Vollbildansicht für Bilder
UI-20.1	Die Vollbildansicht wird durch antippen eines Bildes in der Hauptansicht geöffnet
UI-20.2	Es ist möglich ein Bild aus der Vollbildansicht in die Galerie zu exportieren
UI-20.3	Es ist möglich ein Bild aus der Vollbildansicht zu teilen (Versand per E-Mail falls E-Mail Client vorhanden)

1.2.2 Offene Funktionalitäten

Die im folgenden aufgelisteten Funktionalitäten wurden bisher nicht implementiert und werden für weitere Versionen vorgemerkt.

ID	Beschreibung
F-70	Es können Benutzer zu Streams hinzugefügt werden

ID	Beschreibung
F-70.1	Neue Benutzer erhalten Zugriff auf vorhandene und zukünftige Bilder im Stream
F-80	Bilder werden zwischen den verschiedenen Nutzern live synchronisiert
F-90	Es existiert ein Art Synchronisationslog der die unterschiedlichen Aktivitäten der Benutzer untereinander abgleicht
F-100	Benutzer können aus Streams austreten
F-100.1	Tritt ein Benutzer aus einem Stream aus, so werden lokal alle Bilder des Streams und der Stream selbst gelöscht
F-100.2	Ausgetretene Benutzer werden bei anderen Benutzern aus der Streamteilnehmerliste entfernt
F-110	Es stehen verschiedene Synchronisationsmöglichkeiten zur Verfügung:
F-110.1	Bluetooth, Bluetooth Low Energy (falls verfügbar), Wi-Fi Direct
F-120	Alle Streamteilnehmer können über eine Teilnehmerliste eingesehen werden
F-130	In der Vollbildansicht kann an Bilder herangezoomt werden
F-130	Durch Gesten kann in der Vollbildansicht zum nächsten oder vorherigen Bild gewechselt werden
F-140	Das gesamte UI ist für verschiedene Endgeräte (Tablets, Android TVs) optimiert
D-10	Der Synchronisationlog und die Teilnehmerlisten werden in der Datenbank gespeichert
UI-30	In der Hauptansicht eines Streams ermöglicht ein Menübutton die Teilnehmerliste einzusehen und Teilnehmer hinzuzufügen
UI-40	Der Menüpunkt „Einstellungen“ im Navigation Drawer ermöglicht allgemeine Synchronisationseinstellungen festzulegen

2 Architektur

Um die Anwendung modular und damit auch erweiterbar zu halten, haben wir uns bei der Umsetzung des Projektes am MVC-Prinzip orientiert. Deshalb lässt sich die gesamte Anwendung in drei Komponenten aufteilen: Das Model ist für die Repräsentation und Speicherung der Daten zuständig. Diese Daten werden im View angezeigt. Er beinhaltet in der Hauptsache die Layouts der verschiedenen Activities sowie die zugehörigen Code-Behind-Klassen. Der Controller ist für die Verarbeitung der Daten zuständig und stellt somit das Bindeglied zwischen Model und View dar.

Eine grafische Übersicht über die hier vorgestellte Architektur ist in Abbildung 2.1 zu sehen.

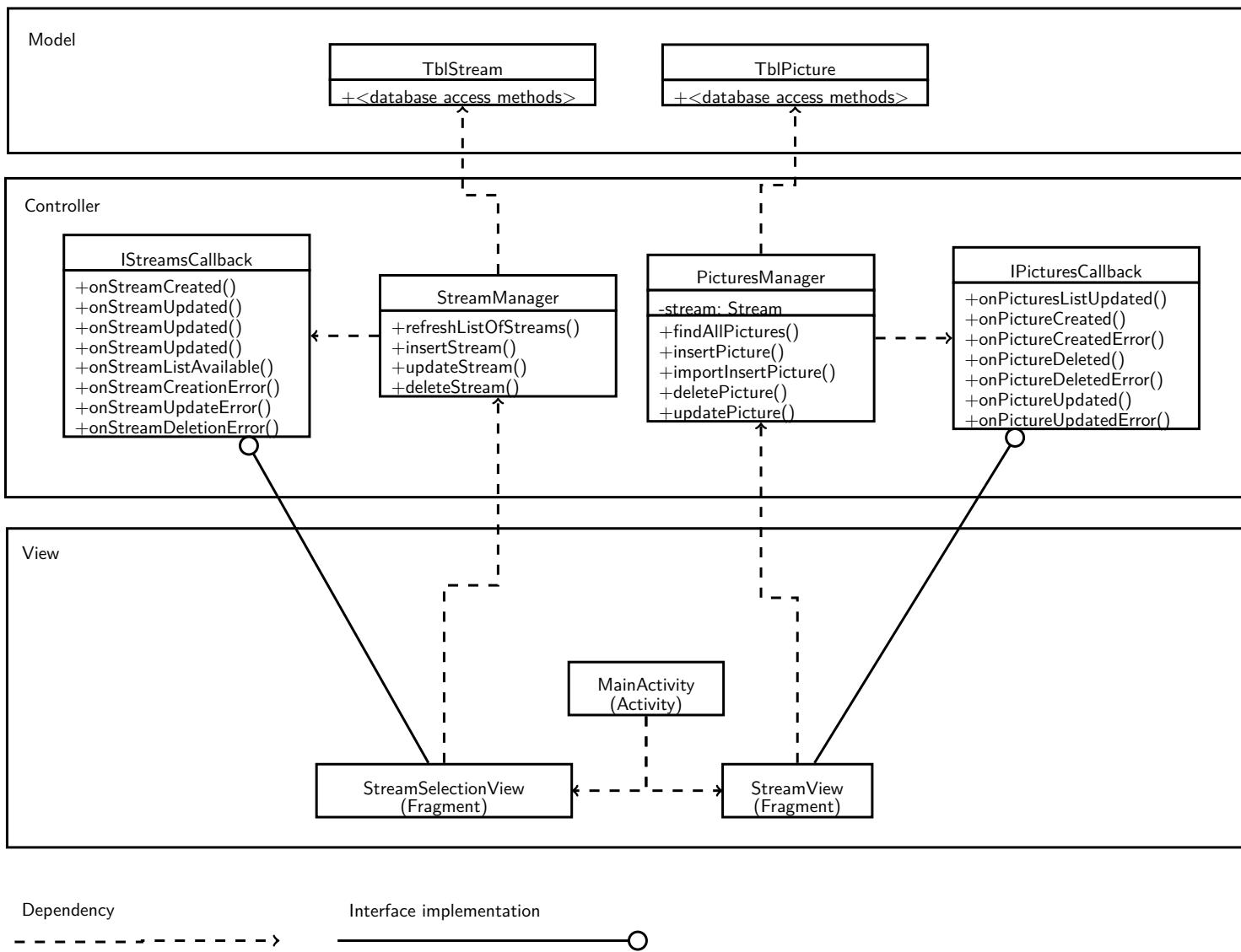


Abbildung 2.1: Übersicht über das Zusammenspiel der wichtigsten Komponenten

2.1 Model

Während die Bilder selber auf der SD-Karte des Gerätes gespeichert werden, haben wir uns dazu entschieden, die Metadaten in einer SQLite-Datenbank zu speichern. So sind auch komplexere Datenabfragen möglich, was uns später bei der Synchronisation der Daten zwischen den verschiedenen Streamteilnehmern zugute kommen wird.

2.1.1 Datenbanklayout

Die Datenbank besteht aus zwei Tabellen: „Picture“ und „Stream“.

Jeder Datensatz in der Tabelle „Stream“ repräsentiert einen Stream in der App. Die Tabelle enthält die folgenden Spalten:

1. **streamid**: Ordnet jedem Stream eine eindeutige ID zu.
2. **streamname**: Speichert den Name eines Streams.
3. **created**: Enthält einen Unix-Timestamp, der den Zeitpunkt der Erstellung eines Streams wiederspiegelt.

Jeder Datensatz in der Tabelle „Picture“ repräsentiert ein Bild in der App. Die Tabelle enthält die folgenden Spalten:

1. **pictureid**: Ordnet jedem Bild eine eindeutige ID zu.
2. **picturename**: Speichert den Titel eines Bildes.
3. **created**: Enthält einen Unix-Timestamp, der angibt, wann ein Bild in die App importiert wurde.
4. **filename**: Speichert den Dateinamen, an dem die eigentliche Bilddatei zu finden ist.
5. **streamid**: Enthält die ID des Streames, in dem sich das Bild befindet.

2.1.2 Datenbankzugriff

Um auf die Datenbank zuzugreifen, gibt es pro Tabelle eine Klasse („TblStream“ und „TblPicture“). Diese enthält für jede Datenbankabfrage eine statische Methode. Diese Methoden können vom Controller benutzt werden, um Daten abzufragen oder zu verändern.

2.2 Controller

Teil des Controllers sind die beiden Klassen „PicturesManager“ und „StreamManager“. Sie stellen dem View alle Funktionalitäten zur Verfügung, die dieser zum Anzeigen und Editieren von Bildern und Streams benötigt.

- StreamManager

1. **refreshListOfStreams**: Abfragen der Liste aller Streams.
2. **insertStream**: Erzeugen eines Streams.
3. **updateStream**: Umbenennen eines Streams.
4. **deleteStream**: Löschen eines Streams, inklusive aller enthaltener Bilder.

- PicturesManager

Der PicturesManager arbeitet immer auf allen Bildern eines speziellen Streams. Der Stream, der vom PicturesManager verwendet wird, wird diesem im Konstruktor übergeben.

1. **findAllPictures**: Abfragen einer Liste aller Bilder des Streams.
2. **insertPicture**: Erzeugen eines Bildes mithilfe einer Bilddatei, die sich bereits im richtigen Ordner befindet. Diese Methode wird zum importieren von Kameraaufnahmen verwendet.
3. **importInsertPicture**: Importiert ein Bild von einer beliebigen Uri. Diese Methode wird zum Importieren von Bildern von der Galerie verwendet.
4. **deletePicture**: Löscht ein Bild aus der Datenbank sowie die zugehörige Bilddatei.
5. **updatePicture**: Ändert den Titel eines Bildes.

2.2.1 Multitasking

Alle Aktionen, die vom „PicturesManager“ oder „StreamManager“ ausgeführt werden, greifen über das Model indirekt auf die Datenbank zu. Desweiteren muss der „PicturesManager“ Bilder kopieren und verkleinern. Dies sind alles Aktionen, die potenziell lange dauern. Daher dürfen diese nicht im UI-Thread einer App ausgeführt werden. Das würde nämlich zur Folge haben, dass sich das Benutzerinterface aufhängt. Um das zu vermeiden, führen wir alle Aktionen in einem separaten Thread aus. Dafür kommt die Klasse „AsyncTask“ von Android zum Einsatz.

2.2.2 Kommunikation mit dem View

Nachdem Daten geändert wurden, muss dies dem View mitgeteilt werden, damit dieser die Darstellung aktualisieren kann.

Dafür existieren die Interfaces „IPicturesCallback“ und „IStreamsCallback“. Sie definieren Callback-funktionen, die die Code-Behind-Klassen aus dem View über Änderungen der Daten informieren. Dafür müssen diese das jeweils passende Interface implementieren und sich beim „StreamManager“ oder „PicturesManager“ als Callback-Objekt anmelden. Die beiden Controller-Klassen können dann in den jeweils passenden Situationen die passende Callback-Funktion des registrierten Callback-Objektes aufrufen. So wird der View über die Änderung der Daten informiert.

2.3 View

Der View wird durch die XML-Layoutdateien sowie durch die zugehörigen Code-Behind-Klassen gebildet. Um das Benutzerinterface möglichst modular zu halten, wurde bei der Umsetzung massiver Gebrauch von Fragments gemacht.

Fragments sind wiederverwendbare Teile des Benutzerinterfaces, die ihre eigene XML-Layoutdatei und ihre eigene Code-Behind-Klasse haben. Activities müssen dann nur noch aus den passenden Fragments zusammengesetzt werden. Dabei kann eine Activity Fragments auch dynamisch laden und austauschen.

Auf diese Weise kann die App später einfach um die Unterstützung von Tablets erweitert werden.

3 Benutzeroberfläche

3.1 Beschreibung der Benutzeroberfläche

3.1.1 Appstart

Nach dem erstmaligen Start der App durch anklicken des App Icons gelangt der Benutzer zur in Abbildung 3.1 gezeigten Benutzeroberfläche.

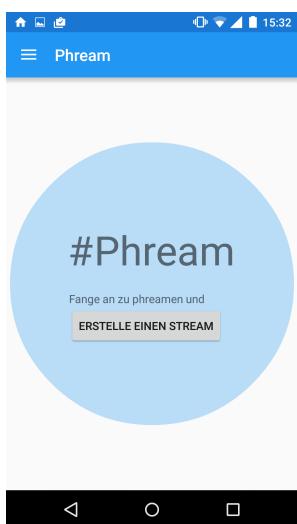


Abbildung 3.1: App-Startansicht



Abbildung 3.2: Schematische Darstellung

Die in Abbildung 3.2 eingezeichnete Statusbar ist standardmäßig in jeder Ansicht der App zusehen und zeigt allgemeine Statusinformationen zum Androidgerät an. Die Action Bar wird von der App selbst gestaltet und beinhaltet verschiedene Optionen, wie beispielsweise Menüs. Den größten Teil der App nimmt das Fragment in Anspruch. In diesem werden die verschiedenen Informationen der App dargestellt. Das Fragment ist dabei eine Art selbständige „Sub-Activity“, die einen eigenen Lifecycle, inklusive eigener Action Listener, besitzt. Das Fragment wird dynamisch durch einen Manager bei Benutzeraktionen ausgetauscht.

In der aktuell angezeigten Startansicht (Abbildung 3.1), die beim erstmaligen Start oder wenn Stream (Ordner) angelegt ist, angezeigt wird, beinhaltet das Fragment den dargestellten Text, den Button zum Anlegen eines Streams und den im Hintergrund platzierten Kreis. Durch anklicken des Buttons öffnet sich ein Eingabefenster, in dem ein neuer Streamname eingegeben werden kann. Nach Bestätigung des Streamnamens wird der Stream angelegt und erscheint daraufhin im Navigation Drawer.

3.1.2 Navigation Drawer

Der Navigation Drawer (Abbildung 3.3) kann durch klicken auf das Menüicon oben links neben dem Appnamen oder durch einen Swipe vom linken Bildschirmrand geöffnet werden. In der daraufhin eingeblendeten Ansicht hat der Benutzer die Möglichkeit zwischen verschiedenen Streams zu wechseln und neue Streams anzulegen. Durch anklicken eines Streamnamens wird dieser aktiv und wird daraufhin in der Hauptansicht angezeigt. Durch einen swipe zum linken Bildschirmrand oder durch anklicken des Pfeils links neben dem Appnamen kann der Navigation Drawer geschlossen werden.

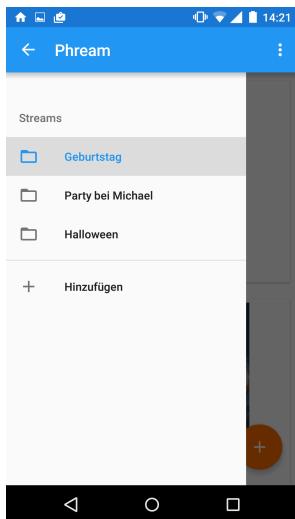


Abbildung 3.3: Navigation Drawer

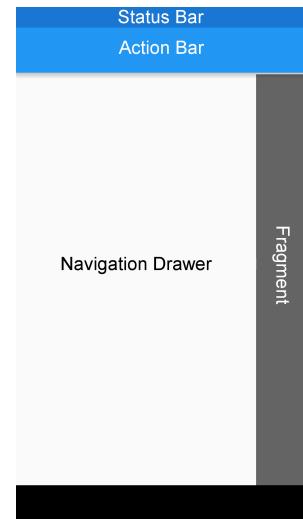


Abbildung 3.4: Schematische Darstellung

In der schematischen Darstellung (Abbildung 3.4) ist zu sehen, dass der Navigation Drawer das aktuelle Fragment „überdeckt“ und den restlichen Teil des Fragments abdunkelt. Der Navigation Drawer selbst wird dabei wieder aus einem Fragment zusammengesetzt. In der Action Bar wird dabei der Button zum Öffnen des Navigation Drawer zu einem Pfeil.

3.1.3 Streamansicht

Nach dem Auswählen eines Streams im Navigation Drawer oder beim Appstart (wenn ein Stream angelegt ist) wird dieser in der Hauptansicht angezeigt. Die Streamansicht (Abbildung 3.5) zeigt alle Bilder an, die dem entsprechenden Stream zugeordnet wurden. Das Fragment wird dabei mit CardViews gefüllt, die eine ImageView mit einem Vorschaubild, dem Bildtitel, sowie Aufnahmemeinformationen zum entsprechenden Bild beinhaltet. Der Benutzer kann durch Scroll-Gesten durch die Ansicht navigieren. Das oben rechts in der Action Bar angezeigte „Options-Symbol“ beinhaltet die Optionen, den aktuellen Stream umzubenennen und zu löschen. Beim Löschvorgang eines Streams werden alle Daten zum Stream und dessen Inhalt aus der Datenbank und alle Bilder von der SD-Karte gelöscht.

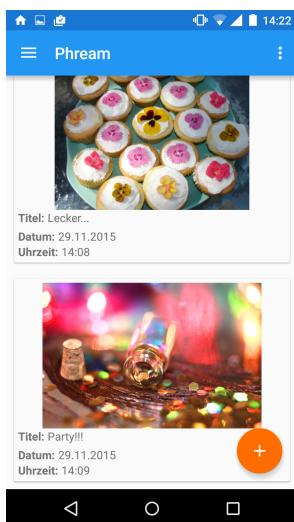


Abbildung 3.5: Streamansicht

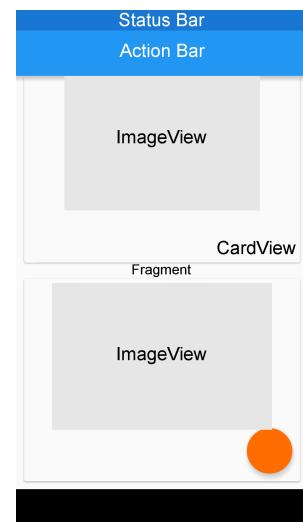


Abbildung 3.6: Schematische Darstellung

Kontextmenü des CardView

Durch einen „long tap“ auf eine CardView in der Hauptansicht öffnet sich ein Kontextmenü zum aktuellen Bild. Das Kontextmenü bietet verschiedene Optionen, wie das Umbenennen des Bildes, das Löschen des Bildes und einer Export Möglichkeit in die Galerie. Durch antippen der gewünschten Option erscheinen zusätzliche Abfragen je nach Menüeintrag. Es wird beispielsweise beim Umbenennen ein AlertDialog erzeugt, der die Eingabe eines neuen Bildtitels ermöglicht. Beim Löschen oder Exportieren des Bildes sollen eingeblendete AlertDialogs versehentliche Nutzeraktionen abfangen und eine Bestätigung der Aktion vom Benutzer erfragen.

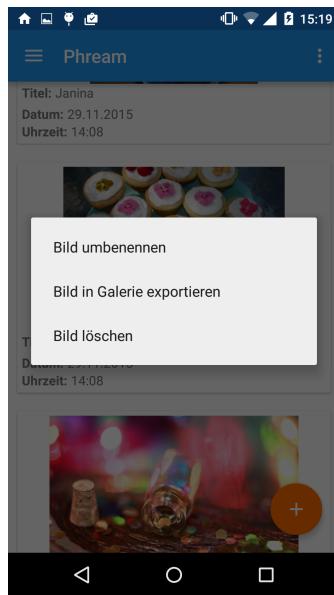


Abbildung 3.7: Kontextmenü - CardView

3.2 Detailansicht eines Bildes

Durch anklicken einer Card in der Hauptansicht, öffnet sich eine zweite Activity die eine Detailansicht (Abbildung 3.8) des Bildes im Vollbild ermöglicht. Die Detailansicht besteht dabei aus einer Fullscreenactivity mit einer ImageView, die die gesamte Activity ausfüllt.



Abbildung 3.8: Detailansicht eines Bildes

Durch einen „short tap“ wird die Status Bar und die Action Bar wieder eingeblendet und bietet dem Benutzer verschiedene Interaktionsmöglichkeiten (Abbildung 3.9).

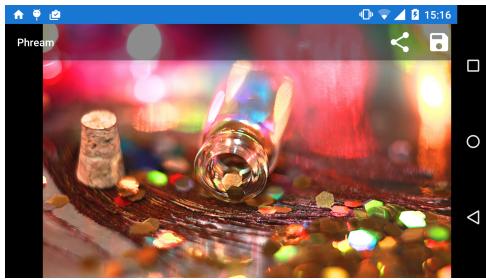


Abbildung 3.9: Detailansicht mit Action Bar

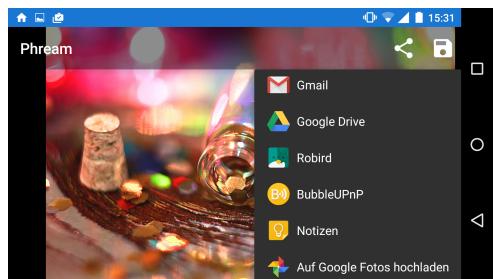


Abbildung 3.10: Share-Provider

Durch anklicken des „Speichern-Symbols“ oben rechts in der Action Bar kann der Benutzer das aktuelle Bild in die Android Galerie exportieren. Dieses wird daraufhin mit aktuellen Datum und Uhrzeit als erstes in der Android Galerie angezeigt. Durch antippen des „Share-Symbols“ öffnet sich der Share Provider (Abbildung 3.10) von Android. Der Share-Provider ist eine allgemeines Framework, welches die Option bietet verschiedene Dateien mit anderen Apps zu teilen oder ihnen zur Verfügung zu stellen. Eine Option ist beispielsweise, sofern ein E-Mail Client vorhanden ist, das aktuelle Bild per E-Mail zu versenden.

3.3 Der Floating Action Button

Der Floating Action Button in der Hauptansicht dient der Gruppierung zweier wichtiger Funktionen der App. Dabei bietet der Floating Action Button verschiedene Transformationsverhalten und öffnet durch antippen ein Floating Action Menu mit weiteren Optionen (Abbildung 3.11). Weiterhin ist der Floating Action Button leicht durch eine Daumengeste erreichbar und gruppiert die Hauptfunktionalitäten der aktuellen Activity in einem Punkt. Je nach angetippten Menüeintrag startet der Galerieimport oder ein Kamera-Intent zum Aufnehmen eines Fotos.

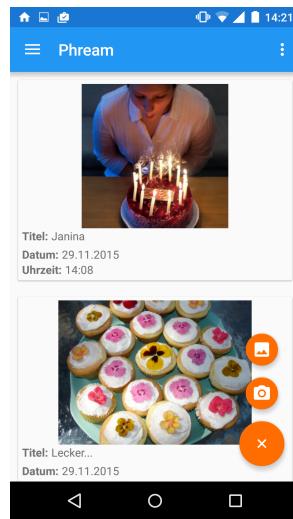


Abbildung 3.11: Floating Action Menu

3.3.1 Aufnehmen eines Fotos

Nach dem Auswählen des Menü Eintrags im Floating Action Menu startet ein Kamera-Intent (Abbildung 3.12), über das der Benutzer ein Foto aufnehmen kann und auch alle weiteren Kameraoptionen zur Verfügung hat. Nach dem Bestätigen des aufgenommenen Fotos kann der Benutzer einen Bildtitel (Abbildung 3.13) festlegen.



Abbildung 3.12: Kamera-Intent

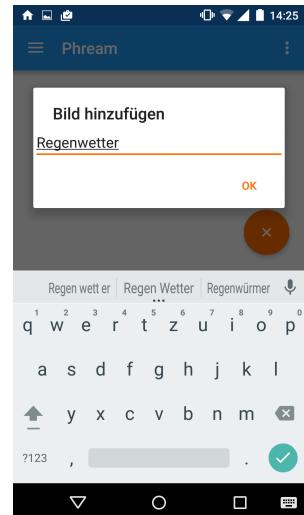


Abbildung 3.13: Bildtitel festlegen

3.3.2 Importieren eines Fotos

Nach dem Auswählen der Importierfunktion im Floating Action Menu startet der Content-Provider (Abbildung 3.14), der dem Benutzer die Option bietet Dateien (in diesem Fall Bilder) auszuwählen. Dabei beschränkt sich die Dateiauswahl nicht nur auf die eigene Galerie, sondern zeigt auch falls vorhanden, Bilder aus Google Drive oder Dropbox an. Dabei werden die Bilder erst aus der Cloud geladen und dann in die App importiert. Nach dem Importieren des Bildes kann der Benutzer wieder einen Bildtitel festlegen.

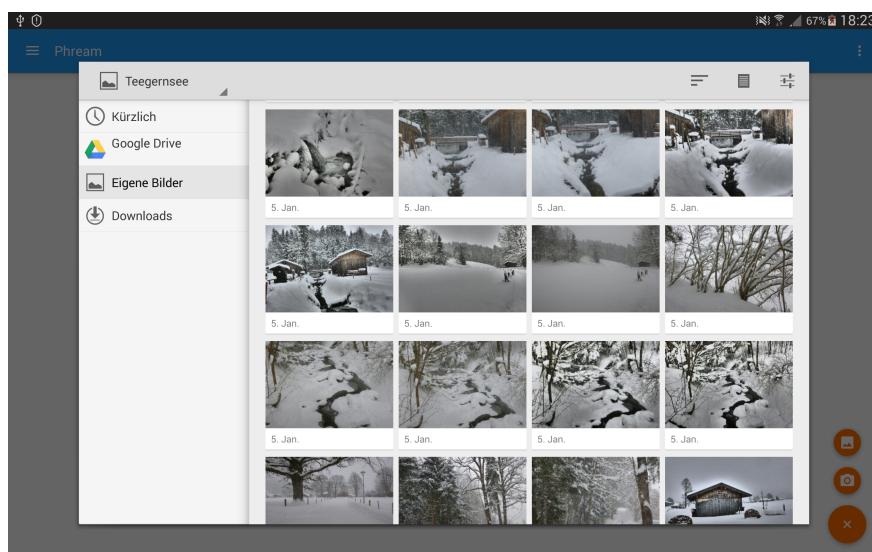


Abbildung 3.14: Content-Provider zum Auswählen eines Bildes

3.4 Verwendete Libraries

Um verschiedene Benutzeroberflächenelemente auch auf früheren Android Versionen verfügbar zu machen, wurde die AppCompat Support Library¹ von Google verwendet. Sie ermöglicht erst in späteren Android Versionen eingefügte Features auch unter älteren Anroid Geräten verfügbar zu machen. Weiterhin wurde zur Gestaltung des Floating Action Menu die AppCompat-Extension-Library² verwendet, um einfach und schnell Floating Action Menu's zu gestalten. Beide Libraries ermöglichen, dass die App von API-Level 16 bis API-Level 22 verfügbar ist.

¹ <http://developer.android.com/tools/support-library/features.html>

² <https://github.com/Tr4Android/AppCompat-Extension-Library>

Aufgrund eines Bugs in der Standardexportfunktion von Android zum exportieren von Fotos in die Galerie, wurden exportierte Bilder immer am Ende der Galerie mit Datum 1.1.1970 angezeigt. Daher wurde die Klasse „CapturePhotoUtils“¹ von samuelkirton zum Exportieren von Bildern verwendet.

¹ <https://gist.github.com/samkirton/0242ba81d7ca00b475b9>

4 Implementierung

Viele implementieren Features beinhalten nur das einfache darstellen von Dialogen oder das Starten von Intents. Daher wird in diesem Abschnitt nur auf dem RecyclerView und dessen Verwendung eingegangen, da dieser eine tiefgründigere Dokumentation erfordert.

Allgemein ist der RecyclerView eine Verbesserung des ListView's und dient der Darstellung von Informationen. Weiterhin bietet der RecyclerView eine bessere Performance gegenüber dem älteren ListView und ist freier gestaltbar. Zudem können einzelne Inhalte des RecyclerView's dynamisch ausgetauscht werden, ohne die komplette Ansicht neu zu laden. Um Inhalt darzustellen benötigt der RecyclerView einen LayoutManager der den Inhalt verwaltet, einen RecyclerViewAdapter, indem der darzustellende Inhalt gespeichert wird und eine Activity / ein Fragment indem der Inhalt angezeigt wird. In diesem Projekt wurde als LayoutManager der Standardlayoutmanager „LinearLayoutManager“ verwendet. Der RecyclerViewAdapter wurde selbst implementiert, um diesem auf die gegebenen Daten anzupassen. Der RecyclerViewAdapter beinhaltet die Klasse ViewHolder, von der für jedes angezeigte Objekt eine Instanz erzeugt wird. Im ViewHolder werden zum einem Referenzen auf das Layout des anzuzeigenden Inhalts gespeichert und zum anderen auf dem gespeicherten Layout, Listener für Klick-Events definiert. In der Methode „onCreateViewHolder()“ wird im RecyclerViewAdapter ein neuer ViewHolder angelegt und diesem das definierte CardLayout zugewiesen. Wenn das entsprechende Objekt auf dem Bildschirm des Nutzers aktiv ist (sprich angezeigt wird), ruft der LayoutManager die Methode „onBindViewHolder()“ auf. In dieser Methode werden die anzuzeigenden Daten der Card geladen und beispielsweise auch das Vorschaubild durch einen „AsyncTask“ erzeugt. Wird eine Card inaktiv (sprich wird nicht mehr angezeigt) wird das ViewHolder Objekt auf einen Stack gelegt. Sollte dieses Objekt wieder in den sichtbaren Bereich des Bildschirm kommen, versucht der RecyclerView, das alte Objekt wieder zu verwenden (zu recyclen). Dadurch steigt die Performance der App.

Um dynamisch den Inhalt des RecylcerView austauschen bietet dieser verschiedene Methoden. Die einfachste Methode um den Inhalt zu tauschen, wenn man nicht weiß welche Inhalte geändert wurden, ist die Methode „mRecyclerView.swapAdapter(mAdapter, false)“ mit einem neuen RecyclerViewAdapter aufzurufen. Dabei tauscht der RecyclerView nur neue / geänderte Elemente aus.

Um alle Benutzer Events auf dem RecyclerView Inhalt (den Cards) kümmert sich der der ViewHolder. Einfache Intent-Starts sind damit zu realisieren. Komplexer wird dies, wenn die App wissen muss welche

Card angetippt wurde, oder beispielsweise „long taps“ registriert werden müssen, da der ViewHolder nicht von „View“ erbt, sondern von „RecyclerView.ViewHolder“. Dieser stellt keine Implementierungen der Methoden „onCreateContextMenu“ und „onContextItemSelected“ bereit und zudem existiert zu den einzelnen Cards keine Code-Behind-Klasse. Daher wurde ein eigener RecyclerView implementiert, der den Standard RecyclerView um einen ContextMenuHolder erweitert und Informationen über das angeklickte Objekt speichert und somit eine Implementierung eines Kontextmenüs ermöglicht.