Inhalt

Abhängigkeiten

Die App basiert auf dem Android Framework und zieht die folgenden Abhängigkeiten an:

- **Cameraview**: Für die Einbindung der Kamera
- **Google Support Libs** (appcompat, design): Zur Erstellung der Grafischen Oberfläche
- **Butterknife**: Für das Annotation basierte zugreifen auf UI Elemente
- **Dexter**: Einfache Abfrage von Berechtigungen für Android 6 und neuer.

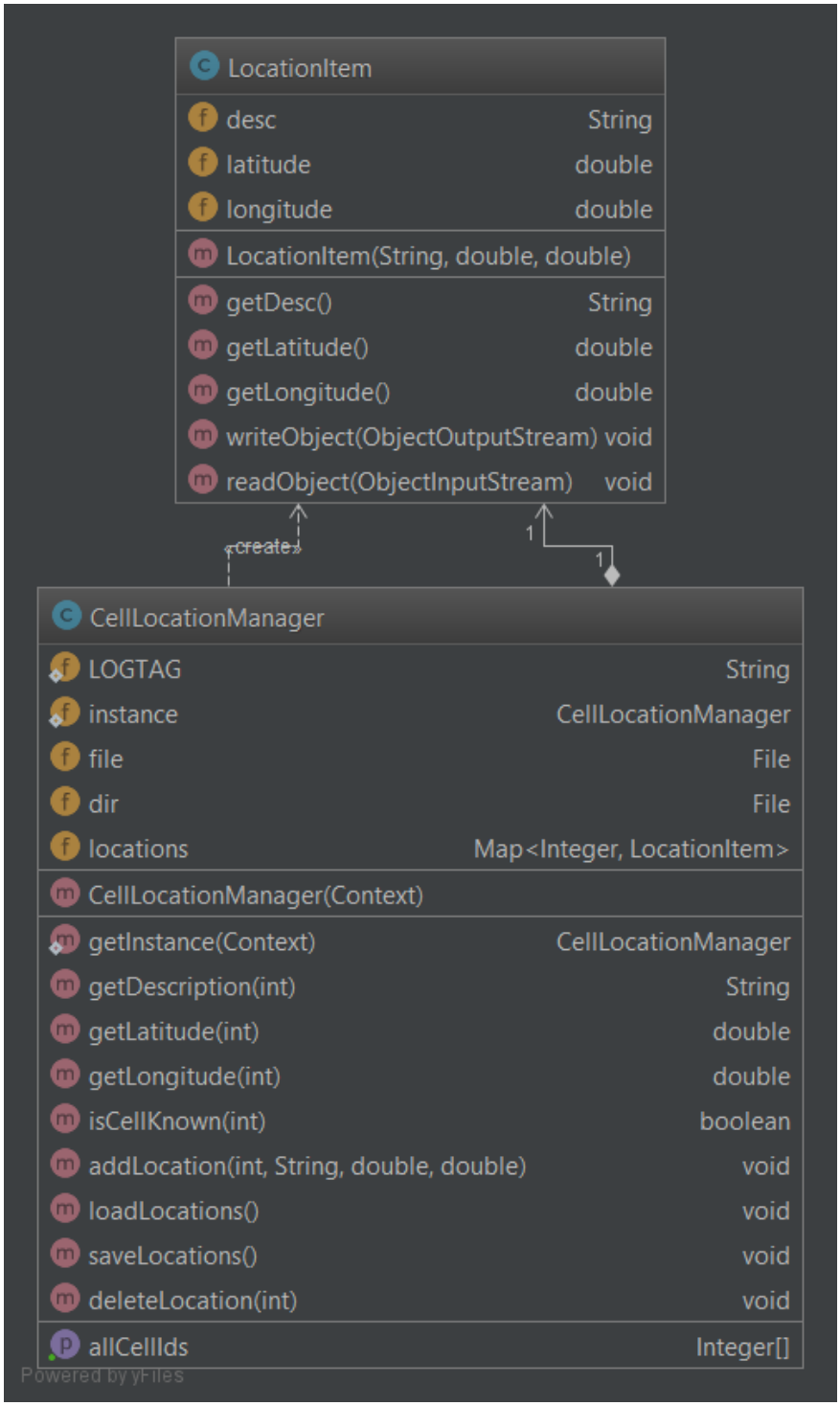
SW-Struktur

Die App ist in drei Packages untergliedert. Das UI Package hat wiederrum drei Unterpakete. Um es nach Features zu gliedern.

- **Model**: Datenhaltung der Lokalisierungen
- **Service**: Hintergrunddienst zur Erfassung von Zellwechsel
- **Ui**: Grafische Oberfläche
    - **Camera**: Kamera Oberfläche und Speicherung der Bilder
    - **Locations**: Liste der Lokalisierungen und das bearbeiten einer Lokalität
    - **Permissions**: Berechtigungsabfrage für Android 6

Model

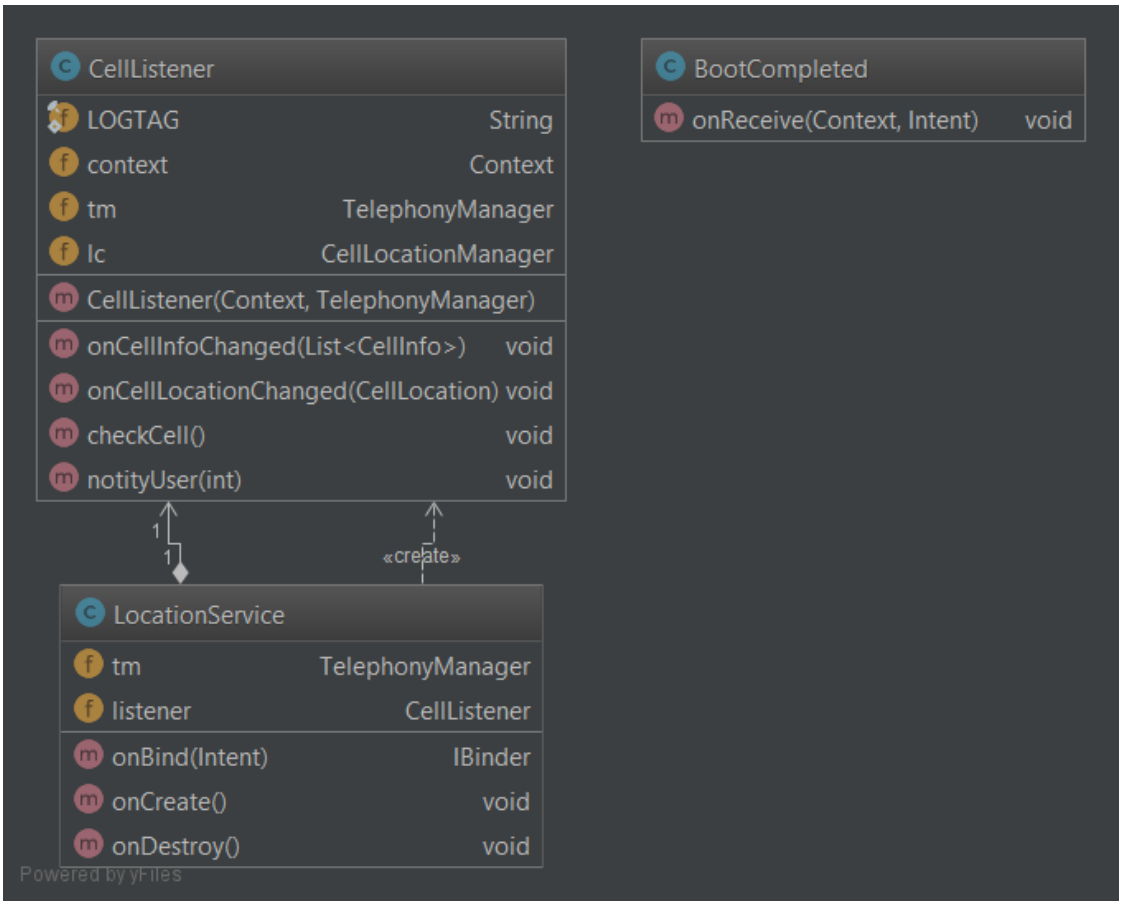
Alle anderen Pakete verwenden das „Model“ Paket um die Daten zu lesen und zu schreiben. In dem Model wird einerseits das Datenmodell und andererseits auch die Persistierung implementiert. Somit muss der Aufrufende Code sich keine Gedanken machen wie oder wann Daten gespeichert werden müssen.



Service

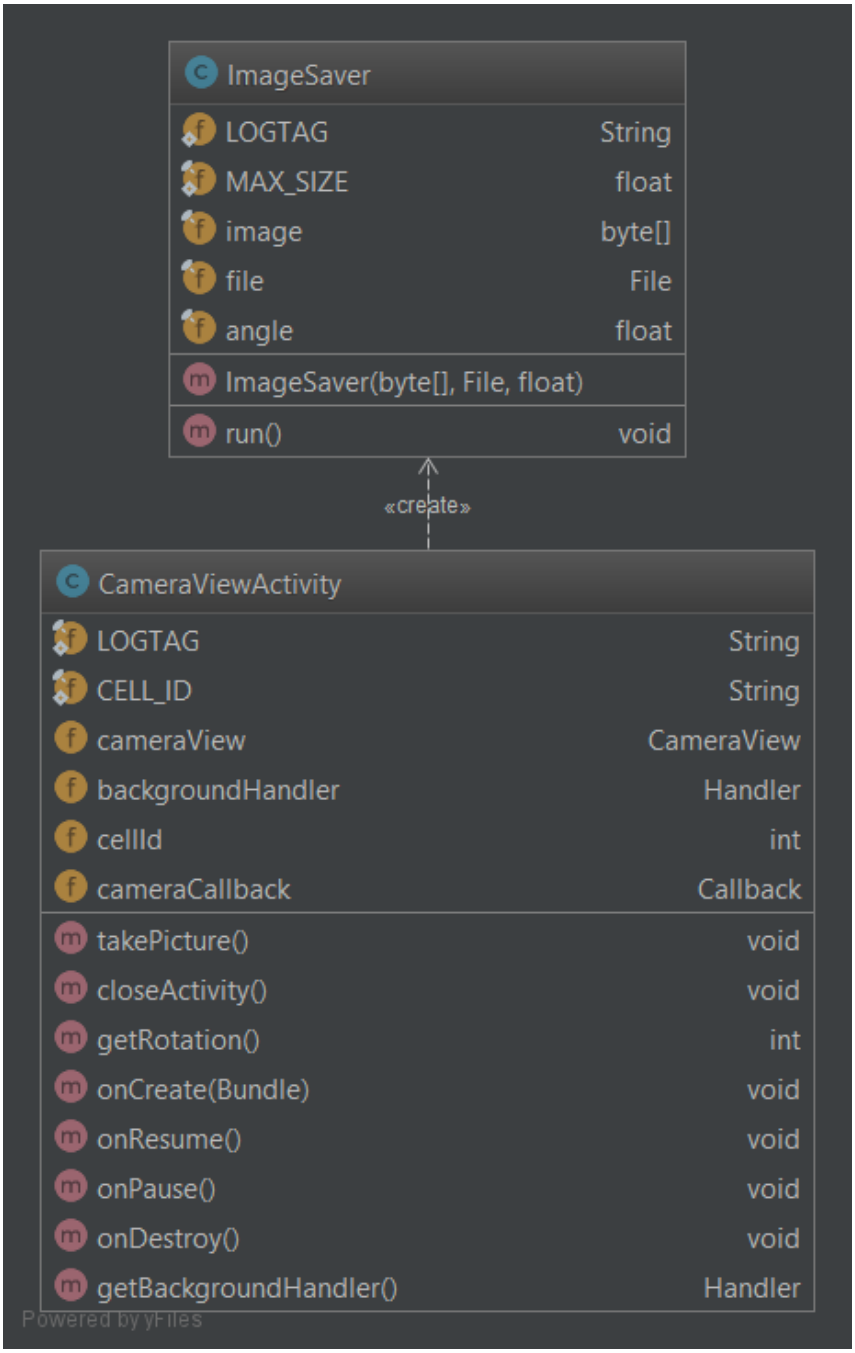
Der Service dient dazu auf Zellwechsel zu reagieren und zu überprüfen ob es sich um eine unbekannte Zelle handelt. Falls dies der Fall ist wird eine Benachrichtigung für den Benutzer erstellt.

Der Event „BootCompleted“ dient zum starten des Services wenn das Gerät neugestartet wird.



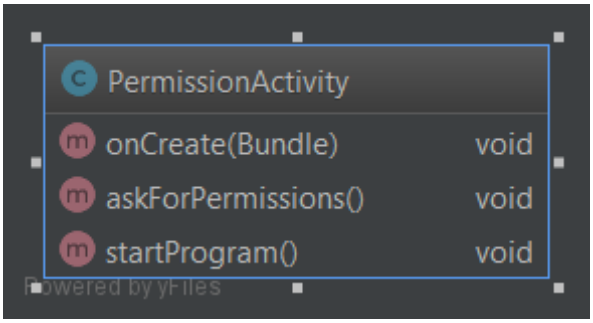
UI – Kamera

Das Kamera Paket besteht aus zwei Klassen. Dem „ImageSaver“ welcher die Daten in einem thread bearbeitet und abspeichert. Des Weiteren beinhaltet das Paket die CameraActivity. Diese stellt das live Vorschaubild der Kamera dar um ein Foto zu machen und übergibt die Daten an den „ImageSaver“.



UI – Permissions

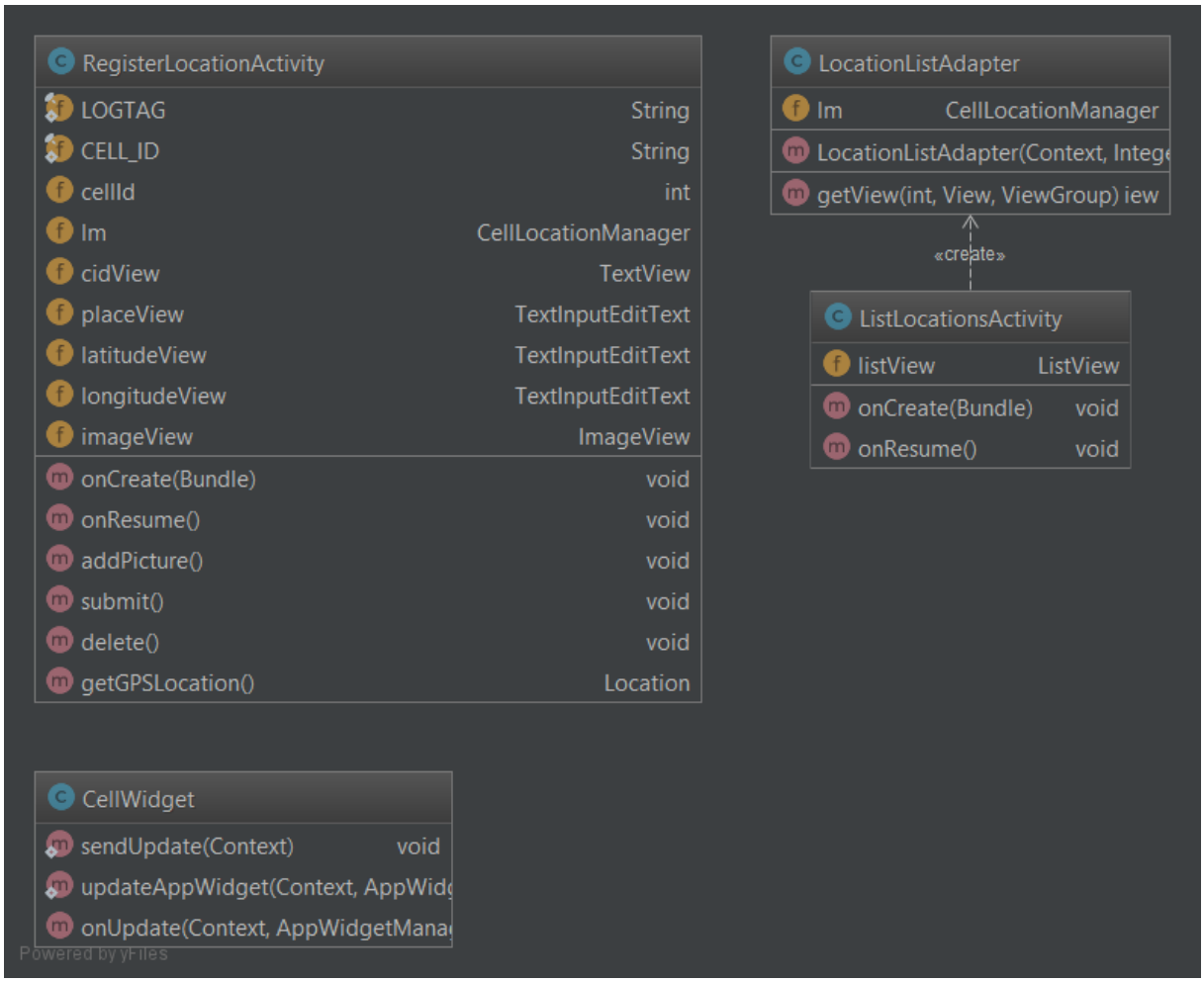
Die Activity „PermissionActivity“ ist nur für Android 6 oder neuer nötig, da sie dazu dient die Berechtigungen Anzufragen die benötigt werden. Diese Activity ist somit keine Voraussetzung für die Prüfung, da das Zielgerät Android 5 nutzt. Es wurde aber dennoch integriert um die Anwendung auch auf aktuellen Geräten ausführen zu können.



UI – Locations

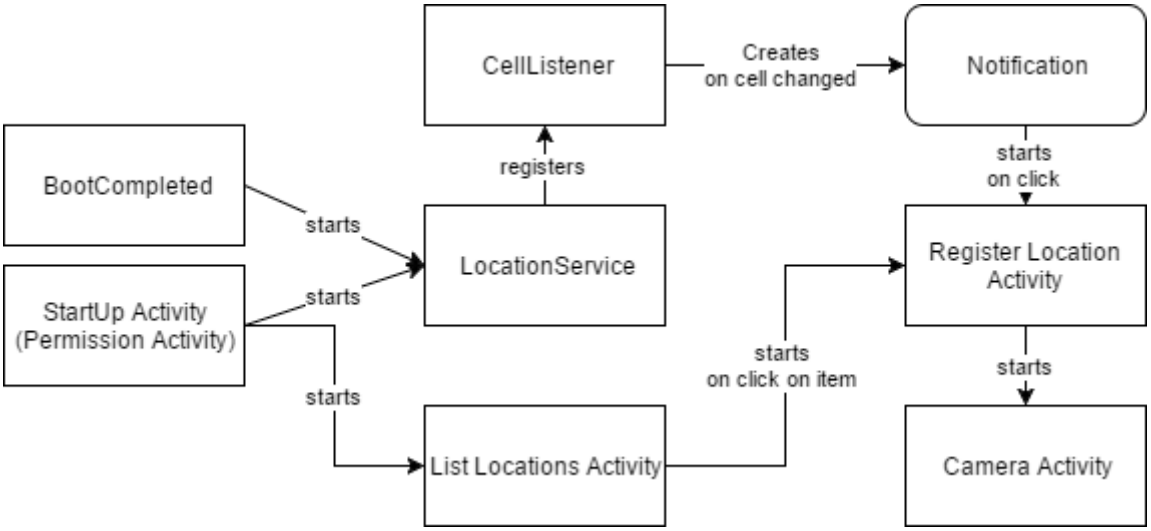
Das Pakete „Locations“ dient zur Anzeige und Verwaltung der Orte. Die „ListLocationActivity“ zeigt mithilfe des Adapters die Liste an Orten an. Die Activity „RegisterLocationActivity“ dient zum Bearbeiten der Ortsinformationen.

Das „CellWidget“ ist das Widget für den Homescreen.



SW-Interaktionen

Der Cell Listener wird entweder beim booten des Geräts oder beim Öffnen der Activity gestartet, weil es nicht möglich ist diesen nach der Installation direkt zu starten. Dieser Service erstellt eine Benachrichtigung sobald eine neue unbekannte Zelle sichtbar ist. Über die Benachrichtigung kann man eine neue Zelle abspeichern. Die gleiche Activity wird auch genutzt um die Zellen später wieder zu bearbeiten. Von der Activity aus kann in den Kamera View gewechselt werden, um ein Foto für die Zelle zu machen.



Sensorabfrage

Zur Einbindung der Kamera kommt die Library „cameraview“[1] zum Einsatz. Die Bibliothek steht unter der Apache License 2.0 zur Verfügung und kann somit frei verwendet werden. Für die Prüfung ist es auch erlaubt externe Bibliotheken einzubinden, daher ist es schlüssig nicht alles selbst zu entwickeln um die Produktivität zu erhöhen.
Ein weiterer Vorteil einer solchen Bibliothek ist es, dass sie auf neuen Plattformen die Camera2 Schnittstelle verwendet und auf älteren die Camera1. Die Einbindung der Kamera ist in dem Fall sehr einfach und funktioniert auf allen Geräten ab API Level 9.
Man benötigt ein Layout in dem man den View einbindet:

```
<com.google.android.cameraview.CameraView
    android:id="@+id/camera"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:adjustViewBounds="true" />
```

Diesen View kann man dank Butterknife einfach in der Activity einbinden:

```
@BindView(R.id.camera)
CameraView cameraView;
```

Um den Preview zu starten muss nur die Kamera beim Starten der Activity gestartet werden.

```
@Override
protected void onResume() {
    super.onResume();
    cameraView.start();
}
```

Um den Life-cycle vollständig abzubilden muss die Kamera beim Beenden auch beendet werden.

```
@Override
protected void onPause() {
    cameraView.stop();
    super.onPause();
}
```

Um Bilder aufnehmen zu können muss noch ein Callback registriert werden. Dieser wird mit den Bilddaten aufgerufen. Dieser macht nichts weiter als die Daten an den Thread zum Speichern zu übergeben, was in eine extra Klasse ausgelagert ist.

```
private CameraView.Callback cameraCallback = new
CameraView.Callback() {
    @Override
    public void onPictureTaken(CameraView cameraView, final byte[]
data) {
        Log.d(LOGTAG, "take picture: " + cellId);
        getBackgroundHandler().post(new ImageSaver(data, new
File(getFilesDir(), cellId + ".jpg"), getRotation()));
        closeActivity();
    }
};
```

[1] https://github.com/google/cameraview

Der ImageSaver thread erhält die Bilddaten als Byte Array. Um das Bild skalieren und rotieren zu können muss es zunächst geladen werden.

```
Bitmap image = BitmapFactory.decodeByteArray(this.image, 0,
this.image.length);
```

Anschließend wird berechnet wie groß das Bild sein kann wenn eine Seite maximal 128 Pixel sein darf.

```
int oWidth = image.getWidth();
int oHeight = image.getHeight();
float scale;

if (oWidth > oHeight) {
    scale = MAX_SIZE / oWidth;
} else {
    scale = MAX_SIZE / oHeight;
}
Bitmap scaledBitmap = Bitmap.createScaledBitmap(image, (int) (oWidth
* scale), (int) (oHeight * scale), true);
```

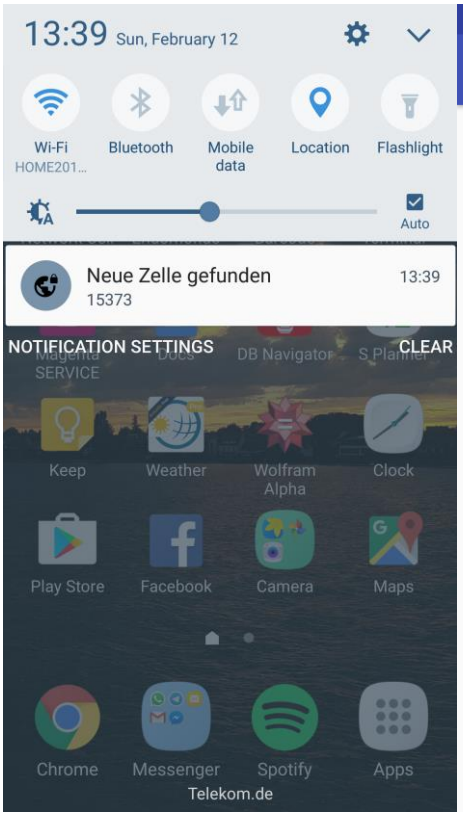
Die Bilddaten werden von der Kamera direkt an den Thread übertragen und sind somit unrotiert. Das heißt je nachdem wie man das Handy gehalten hat ist es verkehrt herum. Um dies auszugleichen wird das Foto noch rotiert.

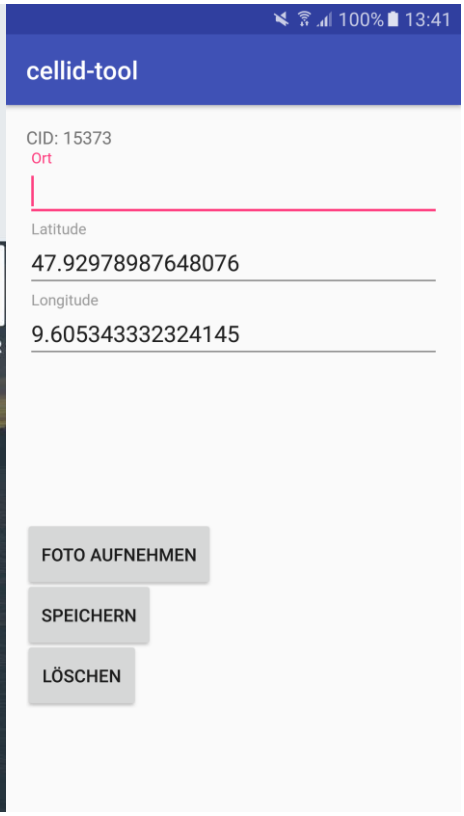
```
Matrix matrix = new Matrix();
matrix.postRotate(angle);
Bitmap rotatedBitmap = Bitmap.createBitmap(scaledBitmap, 0, 0,
scaledBitmap.getWidth(), scaledBitmap.getHeight(), matrix, true);
```

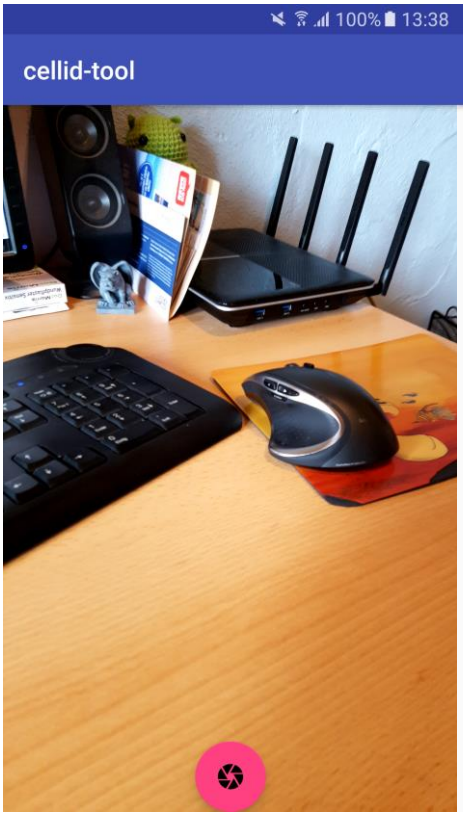
Zum Abspeichern des Fotos wird es mit JPEG komprimiert.

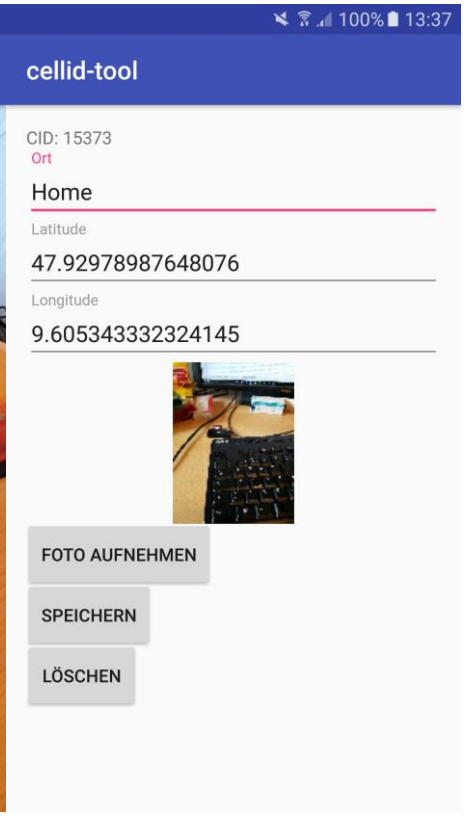
```
output = new FileOutputStream(file);
rotatedBitmap.compress(Bitmap.CompressFormat.JPEG, 99, output);
```

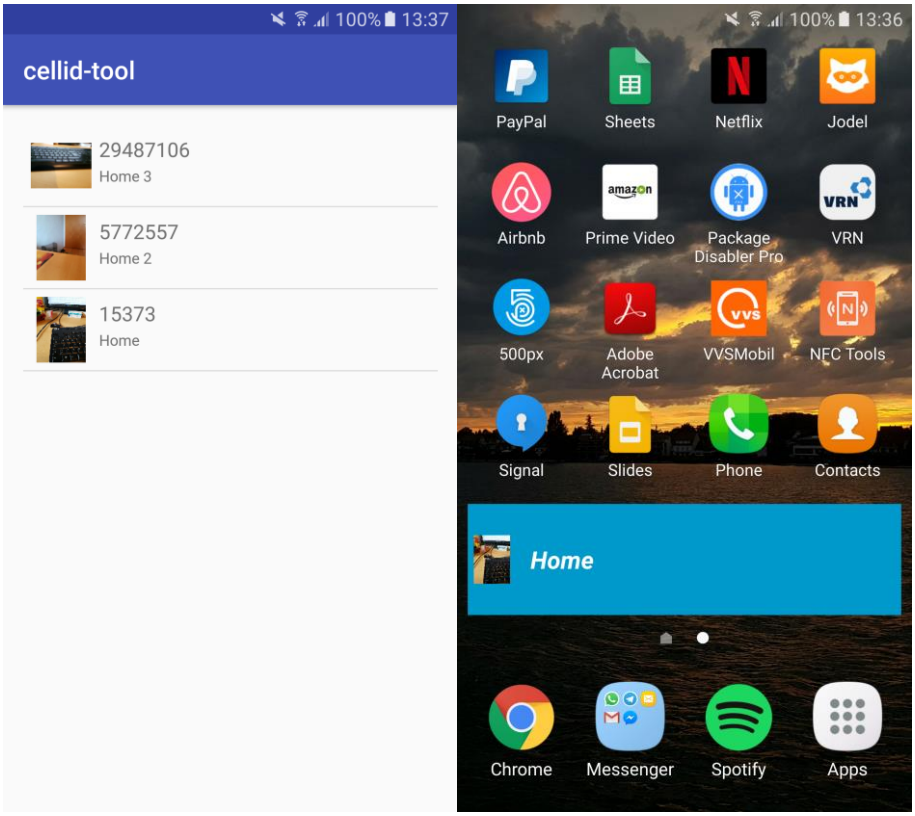
Screenshots










100% 13:37
cellid-tool
29487106
Home 3
5772557
Home 2
15373
Home
100% 13:36
PayPal
Sheets
Netflix
Jodel
Airbnb
Prime Video
Package Disabler Pro
VRN
500px
Adobe Acrobat
VVSMobil
NFC Tools
Signal
Slides
Phone
Contacts
Home
Chrome
Messenger
Spotify
Apps