DHBW CAS

Mobile Computing

Android App Development

Michael Moebius
28.2.2017

Inhalt

Aufgabenstellung

Mit diesem Projekt als Teil der Prüfungsleistung Ihres Studienfachs "Mobile Computing" können Sie etwas zur offenen Technologie-Kommunity beitragen. Wie einer Ihrer Kommilitonen herausgefunden hatte, bildet "opencellid.org" einen Verzeichnisdienst zur Auflösung von Netzwerkzelladressen in geographische Positionen.

Konstruieren Sie eine Smartphone-App für Android 5.0, die solche Zellauflösungsinformation lokal sammelt und gleichzeitig die passende semantische Auflösung (Ortsnamen bzw. -bezeichnung) als Texteingabe vom Benutzer erfragt. Zu jeder neuen Zuordnung soll nach der Texteingabe mit der Smartphone-App ein Bild (z.B. Umgebungsaufnahme, typisches Bauwerk/Monument an der erfassten Stelle) aufgenommen werden, das dann mit reduzierter Auflösung (max. 128x128 Pixel) in die lokale Datensammlung übernommen wird. Der bis hierhin beschriebene Mechanismus muss ohne Internetverbindung oder sonstige Kommunikation funktionieren. Die aufgenommenen Daten sollen über einen Browsing-Mechanismus vom Benutzer inspiziert, editiert und gelöscht werden können. Das UI der App ist ergonomisch und selbsterklärend zu gestalten. Wie genau die lokale Datensammlung/archivierung erfolgt, bleibt dem Entwickler überlassen. Die bisherige Beschreibung stellt zusammen mit der u. g. Anforderungsliste die Basisaufgabe dar. Bei perfekter Erfüllung ist damit bereits die Note 1,0 erreichbar.

Mit folgenden Features verbessern Sie zusätzlich Ihr Ergebnis automatisch um jeweils eine halbe Notenstufe, sofern diese voll funktional sind (Akkumulation ist möglich):

- Upload der gesammelten Daten an opencellid auf Anforderung über UI.
- Aufteilung der App in einen Hintergrund- und Vordergrunddienst. Letzterer wird nur aktiviert zur Bedienung der App übers UI (z.B. Start, Beenden, Dateninspektion) und automatisch, wenn eine neue, bisher unbekannte Zellzuordnungsinformation verfügbar wird.
- Widgetmechanismus zur Anzeige der aktuellen Lokation (ermittelt über Zellen-ID, nicht über GPS) mit dem zugeordneten Bild auf dem Homescreen. Dazu ist natürlich ebenfalls ein Hintergrunddienst erforderlich.

Kernaufgaben für die Prüfungsleistung

- Entwickeln Sie ein SW-System für Android 5.0 Smartphones, das den o. g. Leistungsumfang vollständig erfüllt.
- Verwenden Sie diese SW selbst, um mindestens drei verschiedene Lokationen mit verschiedener Funkzellen-IDs zu erfassen.
- Erstellen Sie eine Dokumentation als einzelne PDF-Datei dazu.

Inhalt der schriftlichen Dokumentation

- Beschreibung der SW-Struktur der App bzw. des App-Bundles (Klassen/Interface-Hierarchie).
- Beschreibung, wie die verschiedenen SW-Instanzen in Ihrem Projekt zusammenwirken.
- Beschreibung, wie Sie die sensorischen Abfragen realisiert haben (kurze Codesequenzen dazu im Dokument zur Erläuterung listen).
- Screenshots/Bildaufnahmen aller UI-Arrangements der App.
- Fotografie des Browsing-Screens der erfassten (minimal drei) Lokationen als Beweis, dass Ihr System nutzbar ist und auch benutzt wurde.
- Optional: Fotografie des Homescreens mit Beispiel der Widget-Anzeige.
- Optional: Upload-Nachweis Ihrer erfassten und beigetragenen Verzeichniseinträge an opencellid.

Analyse der Anforderungen

Funktionale Anforderungen	**Zustand**
Ortsname zu Zelle eingebbar	Erfüllt
Foto Aufnahme bei Zell Eingabe	Erfüllt
Auflösung auf maximal 128x128 reduzieren	Erfüllt Bild wird bei Beachtung des Aspect-Ratio herunterskaliert, so dass eine Seite 128 Pixel ist
Browsing der Zellen	Erfüllt
Editieren der Zellen	Erfüllt
Löschen der Zellen	Erfüllt
Upload an opencellid	Nicht erfüllt (Optional)
Widget für Homescreen	Erfüllt (Optional)

Nicht funktionale Anforderungen	**Zustand**
Muss auf Android 5.0 funktionieren	Erfüllt Getestet mit Android Emulator, siehe Screenshot
Daten müssen lokal gehalten werden	Erfüllt Im RAM bzw. einer Serialisierten Hash-Map
Ergonomische UI	Erfüllt
Selbsterklärende UI	Erfüllt
Verwendung eines Hintergrunddiensts	Erfüllt (Optional)

Abhängigkeiten

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

- **Cameraview**: Für die Einbindung der Kamera
  https://github.com/google/cameraview
- **Google Support Libs** (appcompat, design): Zur Erstellung der Grafischen Oberfläche
  https://developer.android.com/topic/libraries/support-library/index.html
- **Butterknife**: Für das Annotation basierte zugreifen auf UI Elemente
  http://jakewharton.github.io/butterknife/
- **Dexter**: Einfache Abfrage von Berechtigungen für Android 6 und neuer
  https://github.com/Karumi/Dexter

Die Abhängigkeiten sind in dem Gradle Script definiert:

```
compile 'com.android.support:appcompat-v7:25.1.0'
compile 'com.android.support:design:25.1.0'
compile 'com.jakewharton:butterknife:8.4.0'
annotationProcessor 'com.jakewharton:butterknife-compiler:8.4.0'

compile 'com.github.google:cameraview:a9d1eb9bad93dd8da981d380e37f58e35f847af5'
compile 'com.karumi:dexter:3.0.2'
```

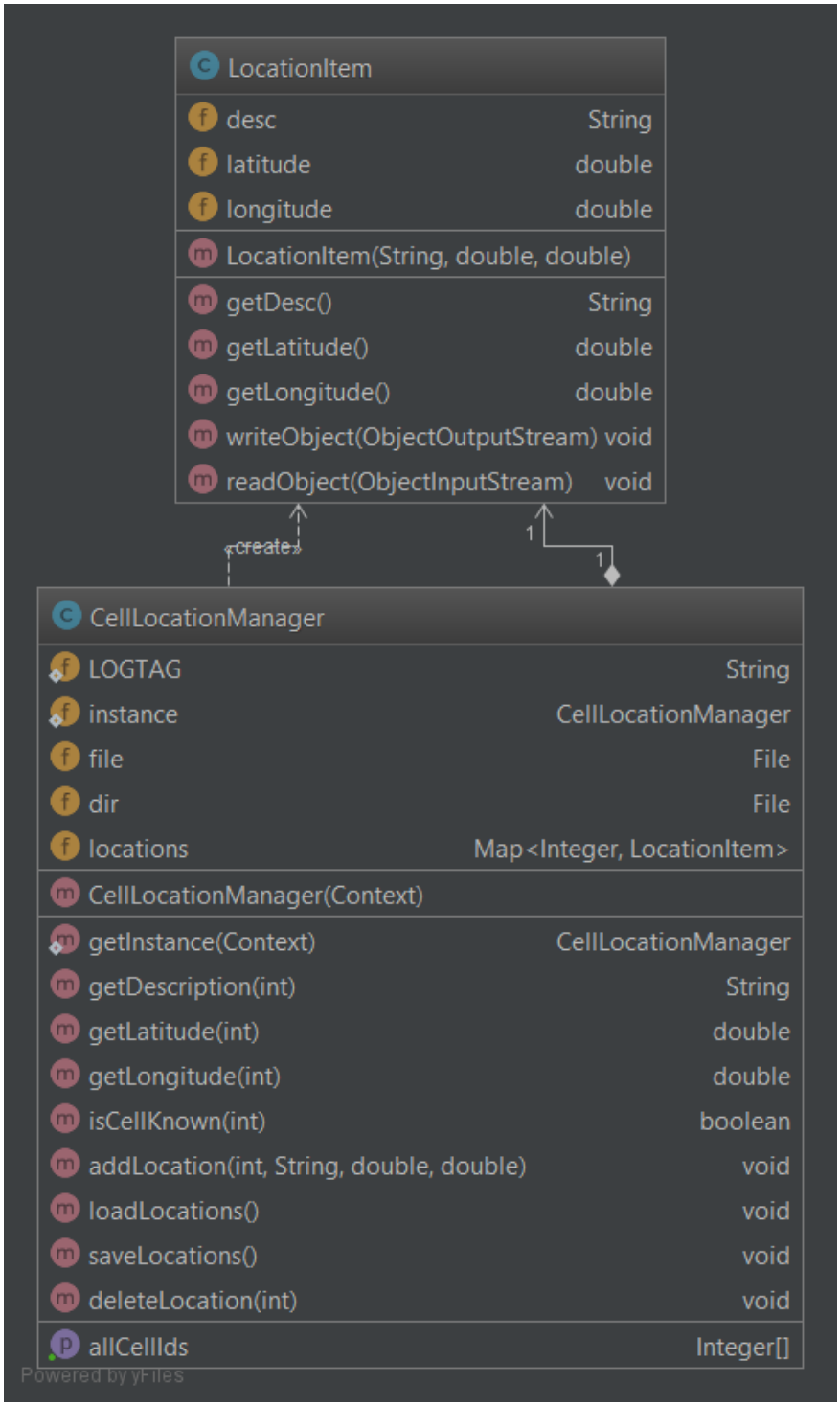
SW-Struktur

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

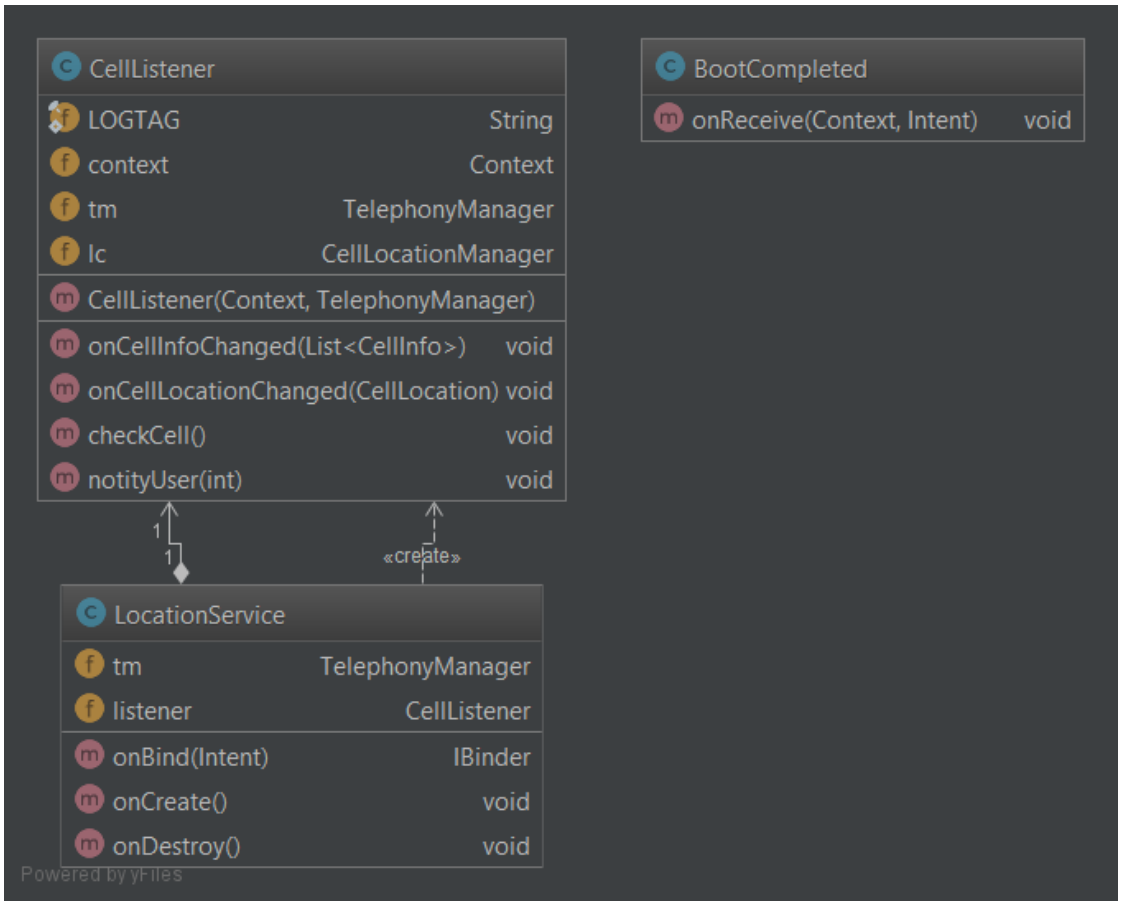
Alle anderen Packages verwenden das „Model“ Package um die Lokalisierungsdaten zu lesen und zu schreiben. In dem Model wird einerseits das Datenmodell und andererseits auch die Speicherung implementiert. Somit muss der aufrufende Code sich keine Gedanken machen wie oder wann Daten gespeichert werden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



Service

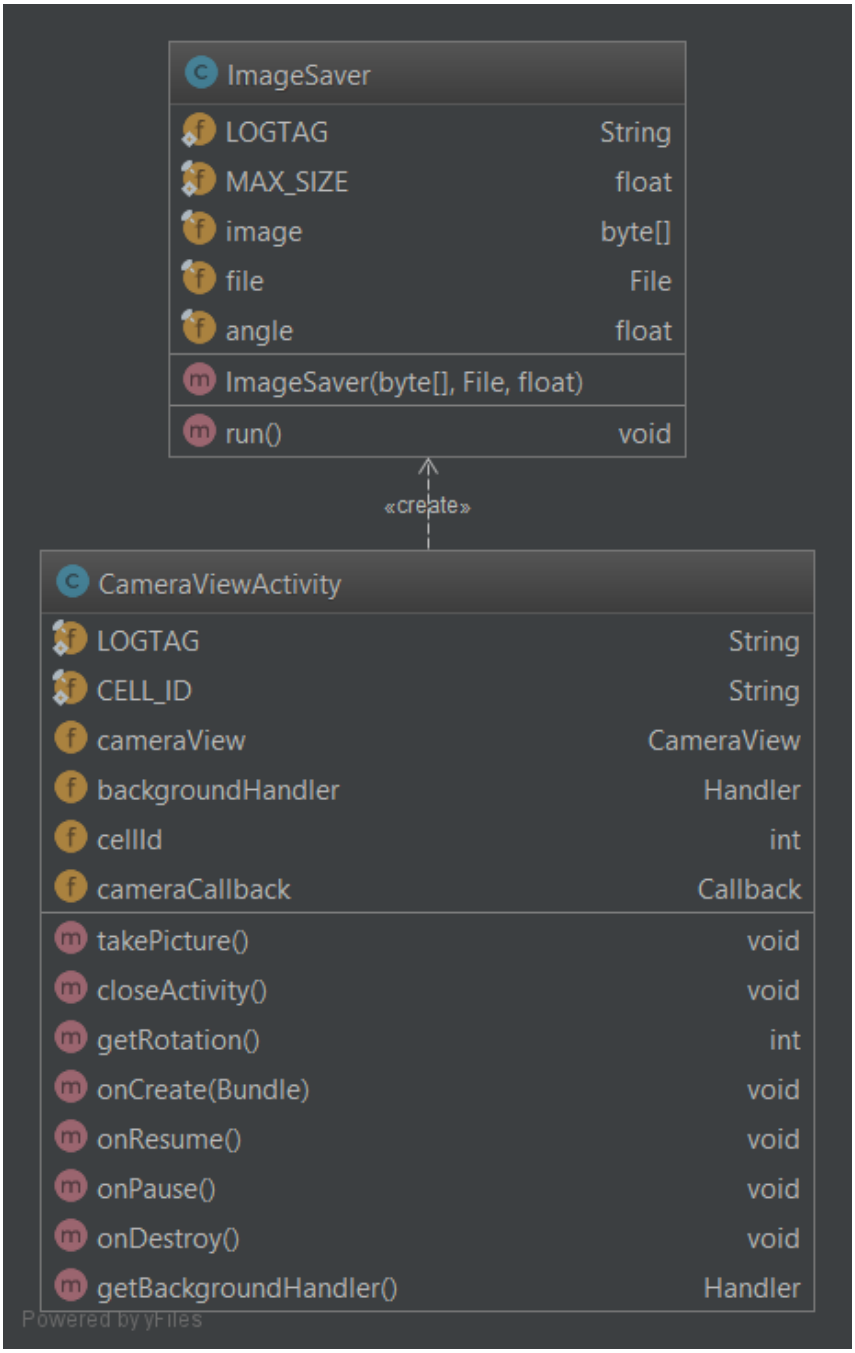
Der Service dient dazu auf Zellwechsel zu reagieren und überprüft ob es sich um eine unbekannte Zelle handelt. Falls dies der Fall ist wird eine Benachrichtigung für den Benutzer erstellt.

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



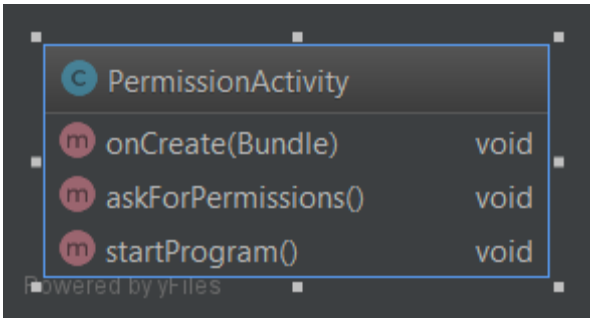
UI – Kamera

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



UI – Permissions

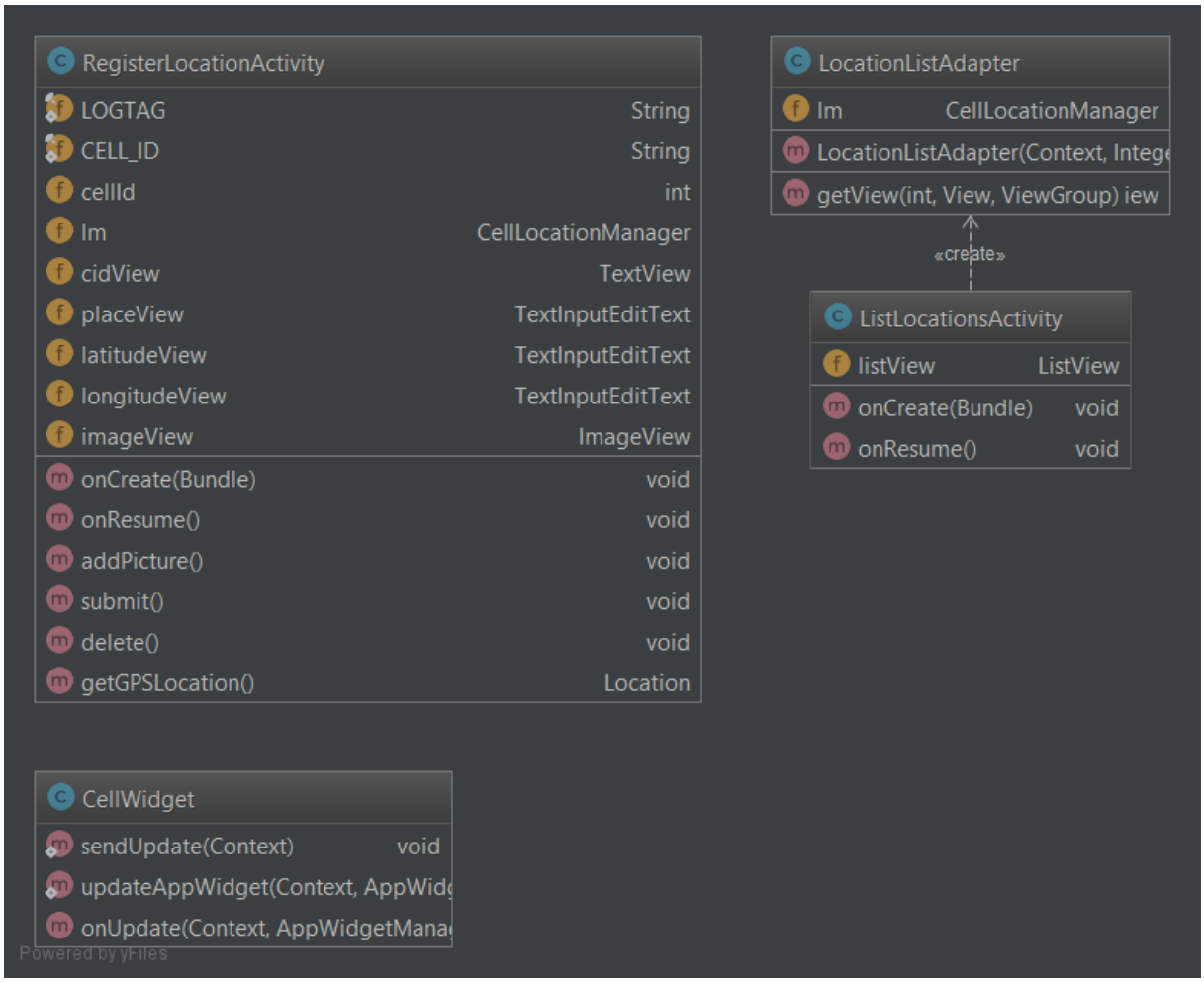
Die Activity „PermissionActivity“ ist nur für Android 6 oder neuer nötig, da sie dazu dient die Berechtigungen Anzufragen die benötigt werden. Diese Activity ist somit keine Voraussetzung für die Prüfung, da das Zielgerät Android 5 nutzt. Es wurde aber dennoch integriert um die Anwendung auch auf aktuellen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



UI – Locations

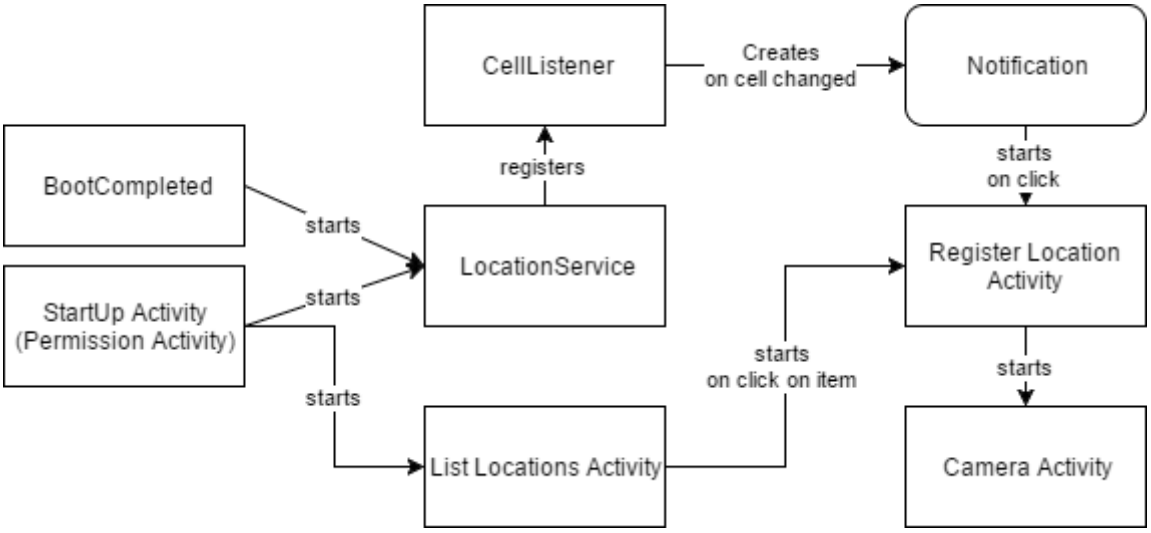
Das Package „Locations“ dient zur Anzeige und Verwaltung der Orte. Die „ListLocationActivity“ zeigt mithilfe des Adapters die Liste an Orten an. Die Activity „RegisterLocationActivity“ dient zum Bearbeiten der Ortsinformationen.

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 der Activity 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);
```

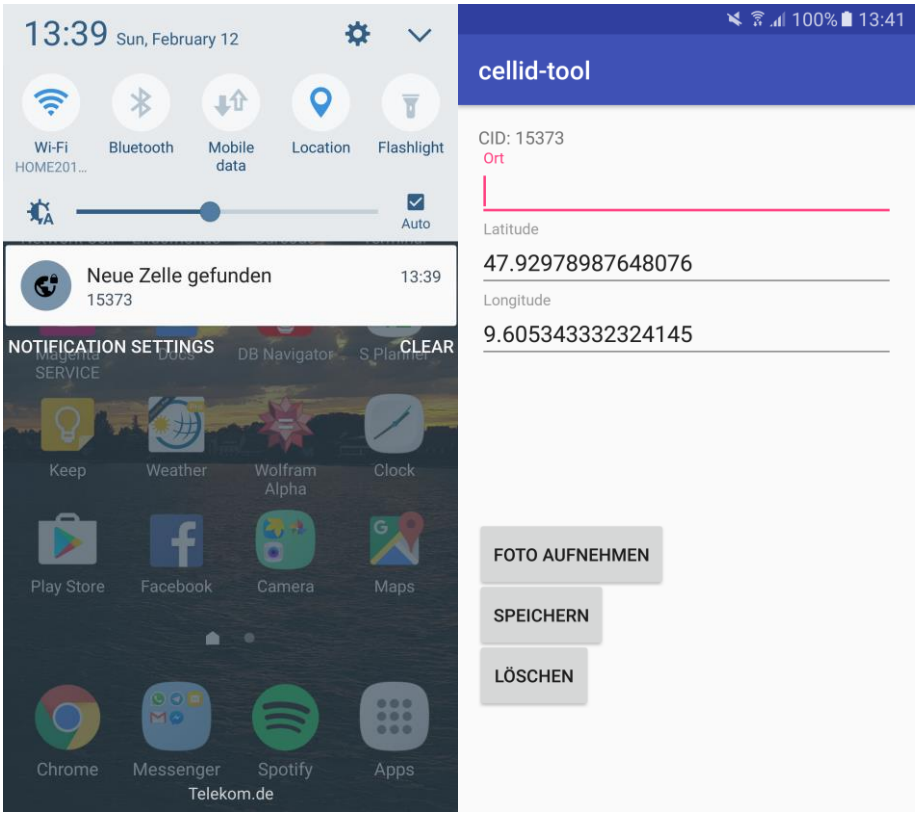
Zum Abspeichern des Fotos wird es mit JPEG komprimiert.

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

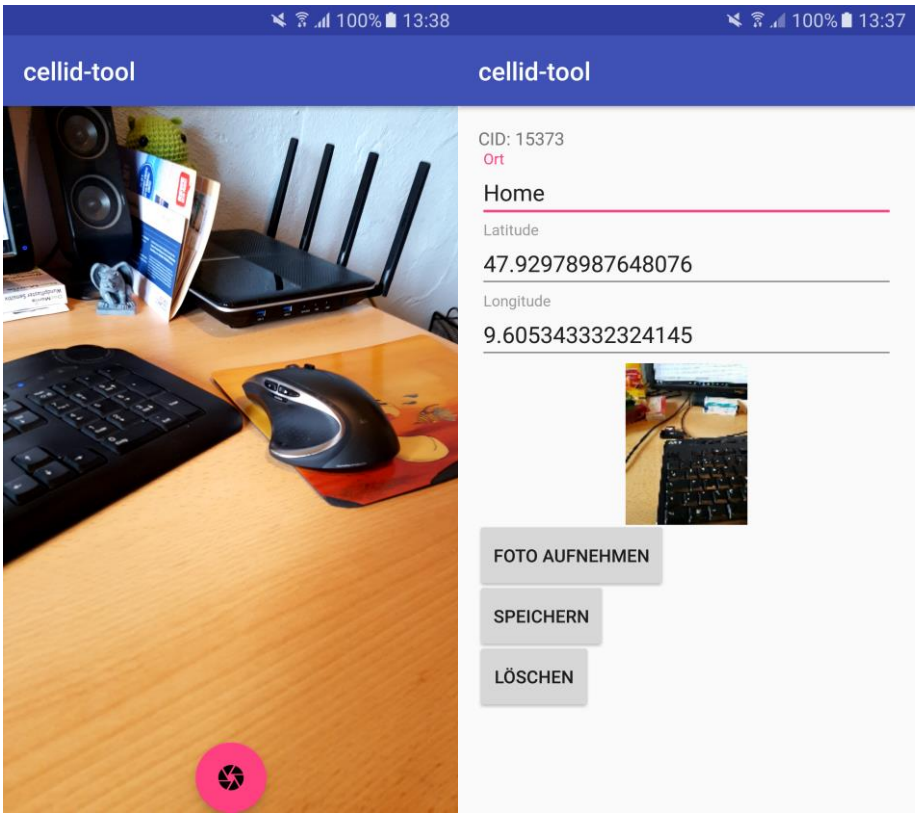
Screenshots

Die Anwendung wurde auf einem Android Emulator mit Android 5.0 getestet und hauptsächlich auf einem Samsung Galaxy S7 mit Android 6.0.1 ausgeführt. Hiervon stammen auch die Screenshots.

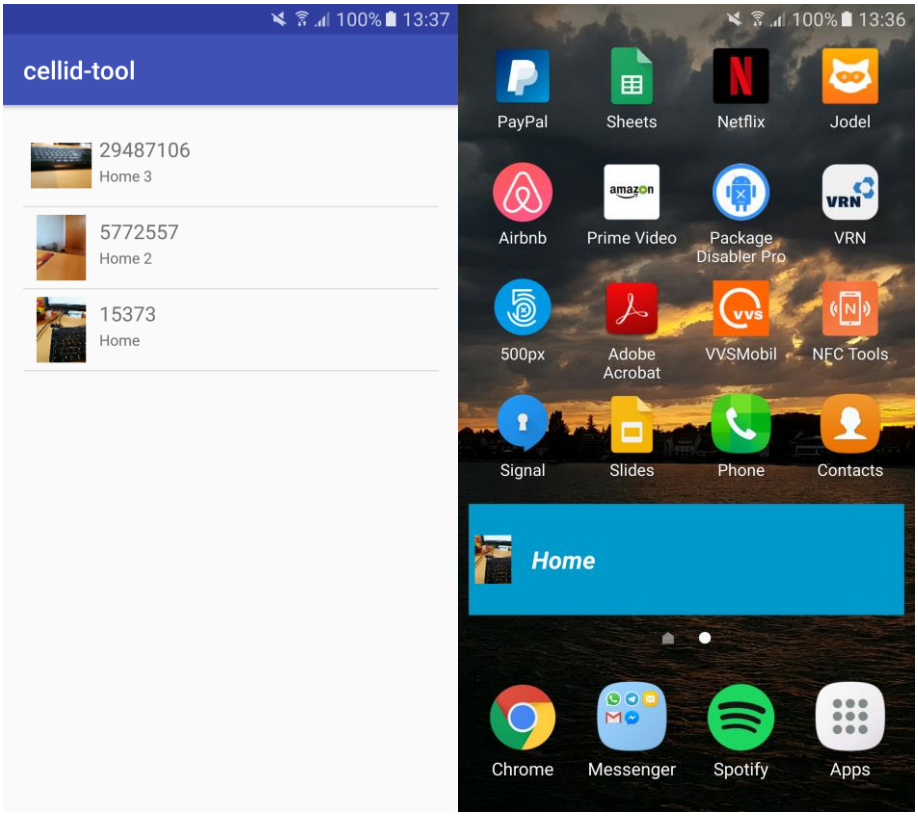
Auf dem Ersten Screenshot sieht man die Benachrichtigung, wenn eine neue Zelle gefunden wurde. Sobald man auf diese Benachrichtigung klickt kommt man zum zweiten Screenshot, der die Activity zur Registrierung der Zelle zeigt.



Der nächste Screenshot zeigt die Kamera Activity zum Aufnehmen eines Fotos. Nach dem Aufnehmen gelangt man zurück zu der Registrier-Activity.



Wenn die App vom Homescreen aus gestartet wird sieht man die Übersichtsliste der registrierten Zellen. Beim Click auf eine gelangt man wieder zu der Detailansicht. Der letzte Screenshot zeigt das Widget auf dem Homescreen.



Android 5.0 Test

Es wird die Cell id „0“ angezeigt, da der Emulator Genymotion, keine Emulation der Cell id ermöglicht. Die App ist aber dennoch vollständig lauffähig.