Bachelor MKI

Mobile Computing

SS 2020

Prof. Dr. Natividad Martínez Madrid

- Seminar Dokumentation -

Geolocation and Maps Integration

Fanni Tamara Marosi, 764345

Fanni_Tamara.Marosi@Student.Reutlingen-University.De

Ebru Selin Özcelik, 764349

Ebru_Selin.Oezcelik@Student.Reutlingen-University.De

6. Semester

Vorgelegt am: 24.06.2020

Inhaltsverzeichnis

1. Motivation

Diese Ausarbeitung wurde im Rahmen des Moduls „*Mobile Computing*“ geschrieben und beinhaltet die Themengebiete Geolocation und Maps Integration in mobilen Anwendungen. Es werden verschiedene Standortprovider aufgezeigt und miteinander verglichen. Zusätzlich wird eine Variante erläutert, wie Programmierer Karten, welche auf Daten von Google Maps basieren, in ihre Anwendungen integrieren können.

Wir entschieden uns für das Thema Geolocation, da es sich bei unserer App hauptsächlich um eine Anwendung für die Nachbarschaft handelt. In der ersten Version ist geplant, dass der Benutzer bei der Registrierung die Stadt und Postleitzahl als Wohnort angibt. Für eine erweiterte, spätere Version haben wir überlegt, eine Positionsbestimmung über GPS-Daten einzubauen. Falls gewünscht, könnte der Benutzer den eigenen Standort bei der Registrierung über ein Geolocation-Feature bestimmen lassen.

Darüber hinaus ist für eine spätere Version geplant, dass Benutzer bei der Suche nach einer Anzeige von ihrem Wohnort ausgehend einen bestimmten Radius einstellen können, in dem gesucht werden soll. Hierfür würde es sich anbieten, die Positionsbestimmung und Map Integration einzubauen.

2. Grundlagen

In diesem Kapitel werden die Grundlagen von dem Global Positioning System (GPS) vorgestellt. GPS ist ein Satellitensystem, welches dazu dient, die globale Position zu bestimmen [1, S.666].

2.1 GPS

Capderou führt an, dass das amerikanische Verteidigungsministerium ein System entwickelte, welches als *Navigation Satellite Time and Ranging / Global Positioning System* (*Navstar/GPS*) bekannt wurde. Es dient der globalen Positionsbestimmung. Mithilfe dieses Systems sollte Transit, das bereits bestehende System, welches seine Grenzen erreicht hatte, abgelöst werden.
Erste Studien fanden bereits seit den 1960er Jahren statt. Seit 1978 gibt es die ersten funktionsfähigen Satelliten für die Bereiche Forschung und Entwicklung [1, S. 666].

2.2 Funktionsweise

Nach Angaben von Doberstein besteht das GPS-System aus 24 Hauptsatelliten, welche die Erdumlaufbahn umkreisen. Die Satelliten haben eine Entfernung von 20.000 Kilometer zur Erde. Die Entfernung wird vom Meeresspiegel gemessen. Alle Satelliten enthalten eine präzise Uhr, die miteinander synchronisiert sind [2, S.23].

Eine Voraussetzung für das Funktionieren des Systems ist, dass an jedem Ort auf der Erde jederzeit die Daten von vier Satelliten empfangen werden können. Schüttler erläutert, dass dies durch die Konstellation der Satelliten gewährleistet wird [3, S.46]. Die Position des Empfängers wird relativ zu dessen Entfernung zu den Satelliten ermittelt. Anhand der Position von mindestens vier Satelliten lässt sich die Position des Empfängers ableiten, indem man die Entfernung zu den Satelliten ermittelt und durch Überschneidungen die möglichen Orte immer weiter eingrenzt, bis sich die Position auf Meter genau bestimmen lässt. Je genauer die Zeitmessung und die Position der Satelliten bestimmt werden, desto genauer ist die eigene Positionsbestimmung [3, S.2 ff.].

Bodenstationen überwachen die Positionen der Satelliten und sorgen dafür, dass das System wie vorgesehen funktioniert. Hierfür ist es unter anderem notwendig, die Satellitenbahnen genau zu vermessen, die eingebauten Uhren zu überwachen und mögliche Störungen zu erkennen [3, S.44].

3. Mobile

Einige mobile Anwendungen erfordern, dass die Daten über den Standort des Nutzers erfragt werden können. In diesem Kapitel wird aufgezeigt, welche Möglichkeiten es gibt, die Standortdaten eines Benutzers zu erhalten.

3.1 Standortanbieter (Provider)

Es werden drei Möglichkeiten beschrieben, woher Android-Anwendungen Standortdaten eines Benutzers beziehen kann. Intern werden mehrere Technologien verwendet, um den Standort eines Benutzers zu ermitteln. Welche Hardware verwendet wird, hängt vom Typ des *Standortanbieters* ab, der für das Erfassen der Daten ausgewählt wurde. Unter Android werden die drei Standortanbieter GPS, WLAN-Netzwerke und Mobilfunkmasten eingesetzt.

3.1.1 GPS

Android bietet Programmierern die Möglichkeit die Lokalisierung eines Geräts über GPS vorzunehmen. Der große Vorteil besteht in der Genauigkeit. Jedoch lassen sich hier auch einige Nachteile feststellen. GPS funktioniert nicht immer zuverlässig in Gebäuden. Die Antwortzeit nach der ersten Anfrage kann länger dauern und der Stromverbrauch des Geräts ist ziemlich hoch, wenn der GPS-Sensor aktiviert ist [4].

3.1.2 WLAN-Information

Bei der zweiten Variante wird die Position des Geräts über ein Netzwerk oder Funkzellen des Mobilfunknetzes bestimmt. Die Bestimmung der Koordinaten ist jedoch ungenauer als bei der GPS-Variante. Dies ist vor allem in flachen Gebieten, in denen die Funkzellen größer sind, der Fall. Als Vorteil lässt sich jedoch festhalten, dass die Positionsbestimmung über Netzwerke innerhalb von Gebäuden möglich ist. Des Weiteren ist die Antwortzeit kürzer und der Akkuverbrauch nicht so hoch wie bei GPS [4].

3.1.3 Netzwerkanbieter

Dieser Anbieter kombiniert WLAN, Mobilfunkdaten und von Funktürmen erfasste GPS-Daten. Der Netzwerkanbieter verbraucht weniger Strom als der GPS-Anbieter, gibt aber Standortdaten von unterschiedlicher Genauigkeit zurück.[1]

3.2 Geokoordinaten

In diesem Kapitel werden die sexagesimale und dezimale Darstellung von Geodaten vorgestellt. Des Weiteren wird der Leser über Androids Location-Objekt informiert.

3.2.1 Sexagesimale und dezimale Darstellung

Eine geografische Angabe der Position besteht aus einem Längen- und einem Breitengrad. Die Längengrade werden von 0 bis 180 Grad östlich oder westlich von Greenwich in England definiert. Die Breitengrade werden 0 bis 90 Grad nördlich oder südlich ausgehend vom Äquator definiert. Ein Grad besteht aus 60 Winkelminuten, eine Winkelminute wiederrum aus 60 Winkelsekunden. Winkelminuten werden mit einem Apostroph, Winkelsekunden mit zwei Apostrophen abgekürzt.

[1] https://developers.google.com/location-context/fused-location-provider

Für die Darstellung der Position wird im Computer die Dezimalschreibweise genutzt, da diese für den Computer praktischer ist. In dem Location-Objekt, welches in Kapitel 3.2.2 erläutert wird, werden der Längen - und Breitengrad als double-Werte gespeichert.

Louis und Müller zeigen die Umwandlung von der sexagesimalen in die dezimale Schreibweise anhand eines Beispiels mit der Golden Gate Bridge in San Francisco auf:

⇒ Sexagesimale Schreibweise:	Dezimale Schreibweise:
37° 49‘ N	37,816667
122° 29‘ W	-122,48333

Die Gradzahlen werden als Vorkommastellen dargestellt. Die Nachkommastellen sind die Minuten und Sekunden. Anhand des Vorzeichens lässt sich bestimmen, um welche Richtung es sich handelt. Norden und Osten werden durch ein positives, Süden und Westen durch ein negatives Vorzeichen repräsentiert [4].

3.2.2 Das Location-Objekt

Bei dem Location-Objekt handelt es sich um eine Instanz der Klasse Location. Diese Klasse repräsentiert einen geografischen Standort. Das Android SDK stellt Programmierern diese Klasse in dem Paket android.location.Location zur Verfügung. Ein Location-Objekt kann Attribute wie einen Längen- und Breitengrad, Zeitstempel und die Angabe der Höhe besitzen.

Android garantiert, dass alle Location-Objekte, welche von einem LocationManager erzeugt werden, über einen gültigen Breiten- und Längengrad und Zeitstempel verfügen. Der LoactionManager wird in dem nachfolgenden Kapitel 4.1 vorgestellt.

Location-Objekte verfügen über mehrere public Methoden. Einige von ihnen werden in der nachfolgenden Tabelle 1 aufgezeigt. Die Auswahl erfolgt anhand der subjektiven Einschätzung, wie häufig sie genutzt werden.[2]

Tabelle 1: Public Methoden des Location-Objekts. Quelle: Eigene Darstellung auf Basis [2]

Methodenname/-signatur	**Rückgabetyp**	**Beschreibung**
getAltitude()	double	Mithilfe dieser Methode lässt sich die Höhe, falls vorhanden, ermitteln. Gemessen wird sie in Metern in dem Referenzsystem „World Geodetic System 1984“ (WGS 84).
getLatitude()	double	Der Rückgabewert dieser Methode ist der Breitengrad des geografischen Standorts (gemessen in Grad).
getLongitude()	double	getLongitude() liefert den Längengrad des geografischen Standorts zurück. Dieser wird in Grad gemessen.
distaneTo(dest: Location!)	float	Diese Methode ermöglicht es, die ungefähre Entfernung zwischen dem Location-Objekt, auf welchem sie aufgerufen wird, und dem als Parameter übergebenen Location-Objekt zu bestimmen.

[2] https://developer.android.com/reference/kotlin/android/location/Location

getTime()	long	An dieser Stelle wird ein Zeitstempel (UTC) zurückgegeben. Es handelt sich um die Anzahl der Millisekunden, welche seit dem 01.01.1970 vergangen sind.

4. Android's Location API vs Google's Location Services API

Das Arbeiten mit Standorten ist eine der häufigsten Aufgaben im Entwicklungsprozess mobiler Apps. Während der Entwicklung stellen sich daher häufig Fragen, welche Technologie zu verwenden ist, um genauere Ergebnisse, maximale Betriebsstabilität und einfache Implementierung zu erzielen. In diesem Kapitel werden zwei Application Programming Interfaces (APIs), Android Location Manager und Google Fused Location Provider vorgestellt und miteinander verglichen.

4.1 Location Manager - Android's Location API

Der LocationManager ist eine Klasse im Android SDK. Mithilfe des LocationManagers erhält der Programmierer Zugang zu den Geolocation-Services des Systems, auf welchem die Anwendung ausgeführt wird. Die Anwendung kann periodisch den Standort des Geräts abfragen oder eine Benachrichtigung erhalten, wenn sich der Nutzer in der Nähe eines angegebenen Standorts befindet.[3]

4.1.1 Location Manager initialisieren

Es gibt in Android eine Klasse für die Positionsbestimmung des Geräts. Diese Klasse heißt LocationManager und befindet sich im android.hardware-Paket. Ein Objekt dieser Klasse kann vom Programmierer in einer Activity wie folgt initialisiert werden:

LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOACTION_SERVICE) [4].

4.1.2 Anbieter wählen und Daten empfangen

Durch den LocationManager lässt sich festlegen, von welchem Provider die Daten empfangen werden sollen. Hierzu braucht man eine Klasse, die das Interface LoactionListener implementiert. Diese Klasse wird dazu genutzt, um sich beim LocationManager anzumelden und Geodaten zu empfangen.

// Mit dieser Anweisung erhält die Anwendung jede Minute GPS-Updates

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 6000, 0, this);

// Mit dieser Anweisung erhält die Anwendung nach jeder Minute und alle 10 Meter Netzwerk-Updates

locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 6000, 10, this);

Die Methode requestLocationUpdates arbeitet mit vier Parametern: der Providertyp, der zu benachrichtigende Listener (this), minTime und minDistance.

Mit minTime lässt sich bestimmen, wann, also nach wie vielen Millisekunden, Updates der Positionsdaten geschickt werden sollen.

Ein Update soll nur dann abgeschickt werden, wenn sich die Position des Empfängers seit dem vorherigen Update um mindestens die angegebene minDistance verändert hat [4].

[3] https://developer.android.com/reference/kotlin/android/location/LocationManager

4.1.3 Empfänger abmelden

Eine Activity muss mit requestLocationUpdates() für den Empfang von Geolokationsdaten beim LocationManager an- und abgemeldet werden. Sobald kein Bedarf mehr an Geolokationsdaten besteht, sollte sich die Activity mit der Methode removeUpdates(LocationListener) abmelden [4].

4.2 Fused Location Provider API – Google Play

Der Fused Location Provider ist eine Standort-API, welche von Google Play-Diensten angeboten wird. Die API entscheidet automatisch von welchem Anbieter die Geodaten angefordert werden sollen.
Der Fused Location Provider bietet die Möglichkeit, verschiedene Signale von unterschiedlichen Providern, wie GPS und WLAN-Netzwerke, zu kombinieren und die Positionsbestimmung vorzunehmen. So lassen sich zum Beispiel möglichst genaue oder die bestmöglichen Positionsdaten mit möglichst niedrigem Akkuverbrauch bestimmen.[1] Außerdem kann man erweiterte Funktionen wie Geofencing verwenden.**Fehler! Textmarke nicht definiert.** Um die Standortdienste von Google nutzen z u können, muss die App eine Verbindung zum Google Play Services Client herstellen.[1]

4.2.1 FusedLocationProviderClient

Der FusedLocationProviderClient ist der Haupteinstiegspunkt für die Interaktion mit den Fused-Location-Anbietern. Ohne den Client kann man weder Geodaten von Providern anfordern noch Aktualisierungen oder Callbacks starten.[4]

4.2.2 LocationRequest

LocationRequest ist ein Datenobjekt, welches Parameter für Anfragen an die FusedLocationProviderApi enthält und das Interface Parcelable implementiert. Mithilfe von LocationRequest-Objekten lassen sich Standortaktualisierungen von Providern über die FusedLocationProviderApi anfordern.

Benötigt die Anwendung sehr genaue Positionsdaten, kann die Anfrage mit setPriority auf PRIORITY_HIGH_ACCURACY und setInterval auf fünf Sekunden-Intervalle eingestellt werden. Dies eignet sich für Anwendungen, wie zum Beispiel Mapping-Apps, welche den Standort des Benutzers in Echtzeit aktualisieren möchten.

Mit setPriority auf PRIORITY_NOPOWER lassen sich Standortabfragen erzeugen, bei denen die Anwendung selbst keine Updates erhält. Sie bezieht die Standortinformationen des Benutzers von anderen Anwendungen, welche diese erfragen. Die PRIORITY_NOPOWER-Anweisung weist einen niedrigen Akkuverbrauch auf und eignet sich für Anwendungen, die den Standort nicht unbedingt benötigen, aber dennoch davon profitieren.

PRIORITY_BALANCED_POWER_ACCURACY eignet sich für Anwendungen, welche Updates periodisch empfangen und dennoch keinen allzu hohen Akkuverbrauch herbeiführen möchten. Mit setFastestInterval(long) lassen sich kurze Intervalle erzeugen und mit setInterval(long) längere wie zum Beispiel 60 Minuten.[5]

[4]https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient
[6] https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest

4.3 Vergleich

In diesem Kapitel werden Google Location API Services und der Location Manager von Android miteinander verglichen. Wir schauen uns zuerst die beiden Frameworks im Allgemeinen an. Im Anschluss wird dargestellt, wie sich die beiden in Bezug auf verschiedene Standortprovider verhalten.

Tabelle 2: Allgemeiner Vergleich Google Location API Services und Android Location Manager. Quelle: Eigene Darstellung auf Basis [6]

API	**Google Location API Services**	**Android Location Manager**
Framework	Baut auf dem Android-Framework auf	Teil des Android-Frameworks
Leistung und Stabilität	Hoch	Mittel
Batterie	Bietet weniger Batterieentlastung "out of the box".	Die Batterieentladung ist ein Problem, wenn sie nicht ordnungsgemäß verwaltet wird.
Verfügbarkeit	Nur an Geräten mit Google Play Services verfügbar.	In allen Geräten verfügbar
Werte	7 Dezimalwerte	14 Dezimalwerte
Anbieter	Es kombiniert automatisch verschiedene Signale, um die Standortinformationen bereitzustellen und verwendet alle drei Anbieter.	Erfordert eine Logik zum Wechseln der Standortanbieter, wenn das Gerät keinen Standort finden kann und verwendet alle drei Anbieter.
Kontrolle	Weniger genaue Kontrolle über GPS	Feinere Kontrolle

Tabelle 3: Vergleich Google Location API Services und Android Location Manager bezüglich des GPS-Providers. Quelle: Eigene Darstellung auf Basis [6]

API	**Google Location API Services**	**Android Location Manager**
Anbieter - GPS	PRIORITY_HIGH_ACCURACY	GPS_PROVIDER
Erforderliche Berechtigungen	ACCESS_FINE_LOCATION für einen genaueren Standort oder ACCESS_COARSE_LOCATION für einen weniger genauen Standort	ACCESS_FINE_LOCATION
Genauigkeit	10m - 100m	10m - 100m
Leistungsbedarf	Hoch	Hoch
Verfügbarkeit	Dort, wo Google Play-Dienste verfügbar sind (nicht in China).	Weltweit (mit freier Sicht zum Himmel)
Standortaktualisierung	In der Regel einmal alle 5 Sekunden, aber die Werte können geändert werden.	In der Regel einmal pro Sekunde
Bemerkung	Erhält zwar (möglicherweise) genauere Standortaktualisierungen als die anderen Einstellungen, ist aber dennoch anfällig für den "Urban Canyon" - Effekt.	Sollte keine freie Sicht zum Himmel vorhanden sein, kann es passieren, dass GPS-Punkte nicht sehr gut gebündelt werden (Positionspunkte "springen"). Die Genauigkeit kann aufgrund des "Urban Canyon" - Effekts in

[6] https://riptutorial.com/de/android/topic/1837/ort

		bestimmten Bereichen irreführend sein.

Tabelle 4: Vergleich Google Location API Services und Android Location Manager bezüglich des Netzwerk-Providers. Quelle: Eigene Darstellung auf Basis [6]

API	**Google Location API Services**	**Android Location Manager**
Anbieter – Netzwerk (WLAN, Mobilfunk)	PRIORITY_BALANCED_POWER_ACCURACY oder PRIORITY_LOW_POWER	NETWORK_PROVIDER
Erforderliche Berechtigungen	ACCESS_FINE_LOCATION für einen genaueren Standort oder ACCESS_COARSE_LOCATION für einen weniger genauen Standort	ACCESS_COARSE_LOCATION oder ACCESS_FINE_LOCATION
Genauigkeit	100m - 1000m +	100m - 1000m +
Leistungsbedarf	Niedrig - Mittel	Niedrig- Mittel
Verfügbarkeit	Überall, wo Google Play-Dienste verfügbar sind (nicht in China). In Reichweite des Zellenturm- oder WLAN-Signals.	In Reichweite des Zellenturm- oder WLAN-Signals
Standortaktualisierung	Gleicher Wert wie PRIORITY_HIGH_ACCURACY	Wird seltener als GPS ausgeführt
Bemerkung	PRIORITY_BALANCED_POWER_ACCURACY: Es ist zwar unwahrscheinlich, dennoch kann diese Einstellung GPS verwenden, um einen Standort zu erzeugen. PRIORITY_LOW_POWER: Verwendet wahrscheinlich kein GPS-Signal und wird im Allgemeinen zum Erkennen signifikanter Standortänderungen verwendet.	Die Genauigkeit hängt von einer Anzahl verschiedener Faktoren ab. Diese Faktoren sind beispielsweise die Anzahl der WLAN-Signale, die Signalstärke oder der Typ des Zellturms.

Tabelle 5: Vergleich Google Location API Services und Android Location Manager, wenn Standortdaten von einer anderen App erhalten werden. Quelle: Eigene Darstellung auf Basis [6]

API	**Google Location API Services**	**Android Location Manager**
Anbieter – andere App	PRIORITY_NO_POWER	PASSIVE_PROVIDER
Erforderliche Berechtigungen	ACCESS_FINE_LOCATION für einen genaueren Standort oder ACCESS_COARSE_LOCATION für einen weniger genauen Standort	ACCESS_FINE_LOCATION
Genauigkeit	10m - 1000m +	10m - 1000m +

Leistungsbedarf	Keine	Keine
Verfügbarkeit	Überall dort, wo Google Play-Dienste verfügbar sind (nicht in China).	Nur wenn eine andere App einen Standort über GPS oder ein Netzwerk empfängt.
Standortaktualisierung	Es werden keine Standorte zurückgegeben, es sei denn, ein anderer Client hat Standortaktualisierungen durch Google Play-Dienste angefordert. In diesem Fall fungiert diese Anforderung als passiver Listener für diese Standorte.	Es erfolgen keine ständigen Aktualisierungen. Dieser hört passiv auf andere Apps, die Standortanfragen stellen und gibt diese Orte zurück.
Bemerkung		Gibt keine von FusedLocationProviderApi generierten Daten zurück.

5. Maps SDK für Android

Das Maps SDK für Android bietet dem Programmierer die Möglichkeit, der Anwendung Karten (Maps) hinzuzufügen. Die Karten basieren auf den Daten von Google Maps. Mit der API lässt sich der Zugriff auf Google Maps-Server verwalten. Darüber hinaus werden mithilfe der API Daten automatisch heruntergeladen und die Kartendarstellung und die Reaktion auf Gesten zur Kartensteuerung verwirklicht.

API-Aufrufe können dazu genutzt werden, um die Benutzeransicht eines bestimmten Kartenbereiches zu verändern. Dies wird umgesetzt, indem man die Karte zum Beispiel mit Markierungen, Polygonen und Overlays ergänzt. Mithilfe dieser Objekte lassen sich die Standorte mit zusätzlichen Informationen versehen. Auf diese Art und Weise kann auch eine Interaktion mit der Karte integriert werden.

Folgende Grafikelemente können Programmierer den Karten mittels der API hinzufügen:

- ⇒ **Markierungen**: Symbole, welche an Positionen auf der Karte gebunden sind
- ⇒ **Polylinien**: Gruppen von Liniensegmenten
- ⇒ **Polygone**: Umschlossene Segmente
- ⇒ **Boden-Overlays**: Bitmap Grafiken, welche an bestimmte Positionen auf der Karte gebunden sind
- ⇒ **Kachel-Overlays**: Bildgruppen; sie werden oben auf den Kacheln der Basiskarte dargestellt.[7]

Um das Maps SDK für Android benutzen zu können, installiert der Programmierer das Google Play Services SDK. Bei der Erstellung des Projekts ist darauf zu achten, eine Google Maps Activity auszuwählen (siehe Abbildung 1).

[7] https://developers.google.com/maps/documentation/android-sdk/intro?hl=de

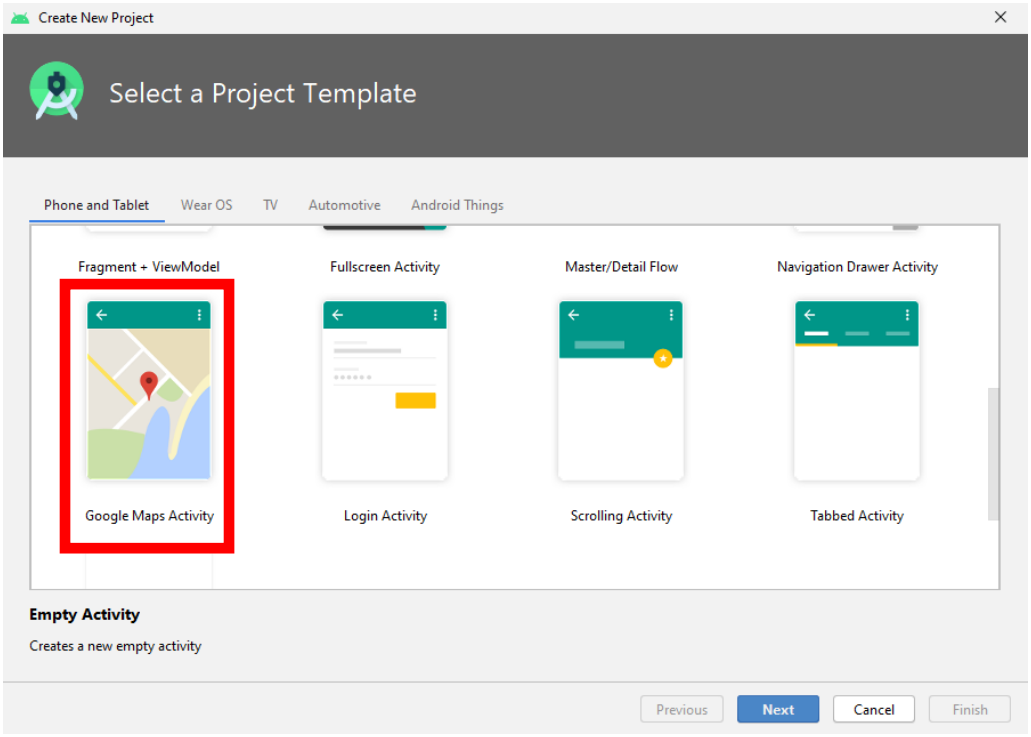


Abbildung 1: Ein neues Projekt mit einer Google Maps Activity in Android Studio erstellen

Des Weiteren wird ein API-Schlüssel mit der Einschränkung für Android Apps benötigt, um die Google Maps Server anfragen zu können. Der API-Schlüssel ist kostenlos und kann mit einigen Schritten generiert werden. Beim Erstellen des Projekts wird automatisch eine Datei namens `google_maps_api.xml` erzeugt. Diese Datei enthält einen Link, welchen man im Browser aufrufen kann, um den API-Schlüssel zu generieren. Nachdem ein API-Key erzeugt wurde, sollte dieser in die `google_maps_api.xml`-Datei eingefügt werden (siehe Abbildung 2).[8]

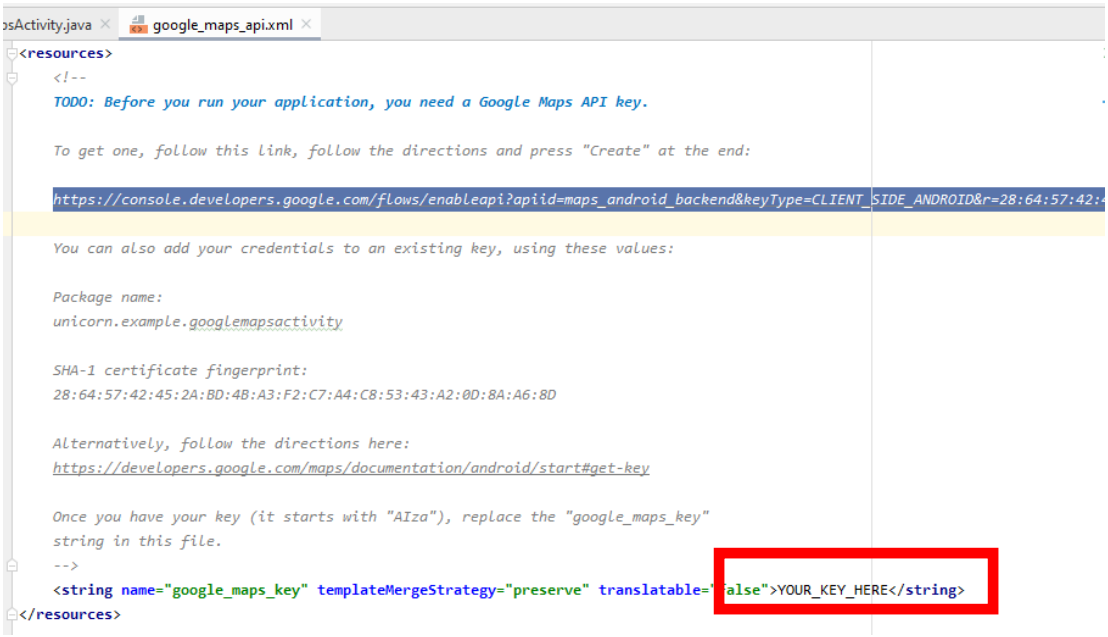


Abbildung 2: google_maps_api.xml

[8] https://developers.google.com/maps/documentation/android-sdk/start

6. Eine GPS-Tracker-App

Im Folgenden stellen wir eine kleine GPS-Tracker-App vor. Bei der Implementierung der App sind wir einem YouTube-Tutorial gefolgt.[9] [10] [11] [12] [13]

Die nachstehenden Screenshots entstammen unserer Implementierung. Wir haben uns überlegt, unseren Kommilitonen den Quellcode zur Verfügung zu stellen. Allerdings werden sie einige Zeilen ergänzen müssen, damit die App funktioniert. Im Weiteren werden die Schritte beschrieben, wie vorangegangen werden soll.

1. Google Play Service öffnen ->
    a. https://developers.google.com/android/guides/setup
    b. -> Google Location and Activity Recognition suchen
    c. -> API zu build.gradle (:app) hinzufügen

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
// API hinzufügen
    implementation 'com.google.android.gms:play-services-location:17.0.0'
```

2. LocationRequest initialisieren

```
public class MainActivity extends AppCompatActivity {

    private static final int PERMISSION_FINE_LOCATION = 99;
    TextView tv_lat, tv_lon, tv_altitude, tv_accuracy, tv_speed, tv_sensor,
tv_updates, tv_address;
    Switch sw_locationsupdates, sw_gps;
//LocationRequest deklarieren ☺
//LocationRequest = locationRequest;

    <…>


    final int DEFAULT_UPDATE_INTERVAL = 30;
    final int FAST_UPDATE_INTERVAL = 5;
```

[9] Android Studio GPS location tracker tutorial 01: https://www.youtube.com/watch?v=V62sxpyxapU
[10] Android Studio GPS location tracker tutorial 02: https://www.youtube.com/watch?v=2ibBng2eJJA&t=336s
[11] Android Studio GPS location tracker tutorial 03: https://www.youtube.com/watch?v=_CdZ3xURK-c
[12] Android Studio GPS location tracker tutorial 04: https://www.youtube.com/watch?v=uTxItxrDVSk
[13] Android Studio GPS location tracker tutorial 04: https://www.youtube.com/watch?v=_0Bqd_Mv2uI

```
sw_gps.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(sw_gps.isChecked()){
//Anbieter GPS ☺
//LocationRequest.PRIORITY_HIGH_ACCURACY

          locationRequest.setPriority(LocationRequest.<…>);
            tv_sensor.setText("Using GPS sensors");
        }else{

//Anbieter Netzwerk ☺
//LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY
locationRequest.setPriority(LocationRequest.<…>);
            tv_sensor.setText("Using Cell Towers or Wifi");
        }
    }
});
```

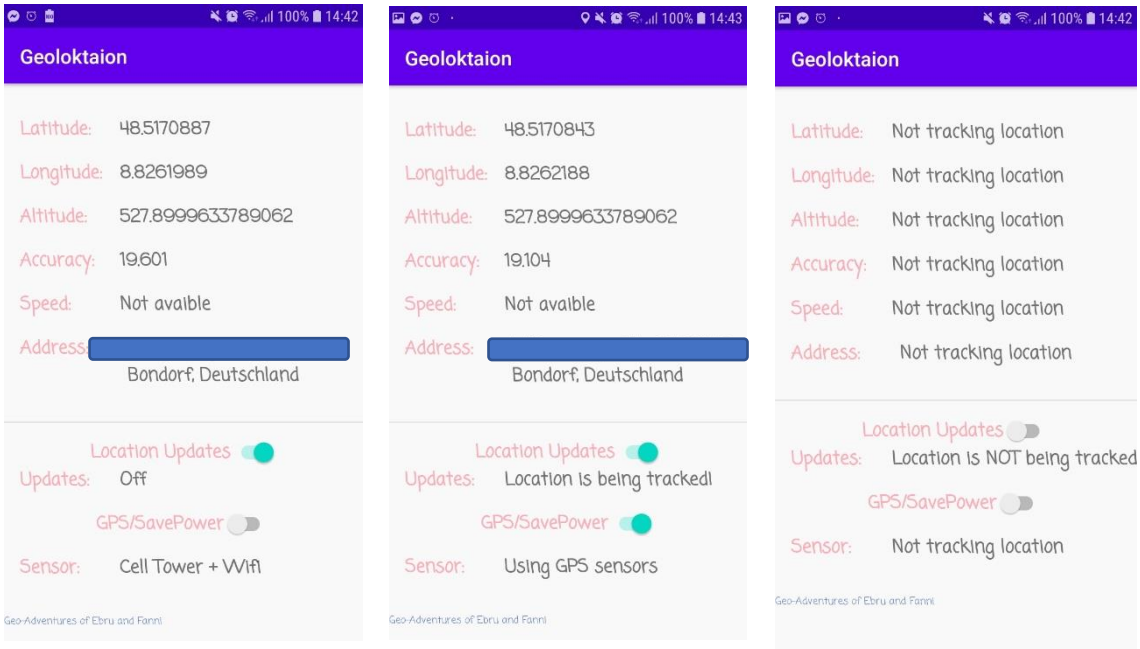
3. FusedLocationProviderClient initialisieren

```
private void updateGPS(){
// FusedLocationProviderClient initialisieren
// = LocationServices.getFusedLocationProviderClient(MainActivity.this);

 fusedLocationProviderClient = <…>

    if(ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED){
            fusedLocationProviderClient.getLastLocation().addOnSuccessListener(this,
new OnSuccessListener<Location>(){
                @Override
                public void onSuccess(Location location) {
                    updateUIValues(location);
                }
            });
    }
    else{
          if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
              requestPermissions(new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, PERMISSION_FINE_LOCATION);
          }
    }}
```

4. So sieht die GUI der App aus:



7. Fazit

Vergleicht man Google Location API Services mit dem LocationManager von Android, lässt sich feststellen, dass die Google Services eine höhere Leistung und Stabilität während der Laufzeit aufweisen.

Werden für die Updates des Standortes Parameter für einen Mindestzeitraum und eine Mindestentfernung festgelegt, erhält der LocationManager ältere Standorte oder Kartenaktualisierungen. Häufig wird auch einer der beiden Parameter ignoriert. Dies ist ein Problem, welches bei den Google Location API Services nicht vorkommt. Darüber hinaus verbrauchen sie weniger Akkuleistung.

Um jedoch Google Location API Services nutzen zu können, muss man Google Play Services installieren. Dies kann in einigen Fällen zu einem Problem führen, denn die Anwendung wird auf Geräten mit üblicher Firmware, die nicht von Google unterstützt werden oder auf denen Google Services nicht verfügbar sind, nicht funktionieren. In China zum Beispiel sind Google Play Services aufgrund der Innenpolitik nicht verfügbar und somit wird ein großer Teil des Marktes ausgegrenzt. Hier muss man auf den LocationManager von Android zurückgreifen.

Zusammenfassend wird empfohlen, die Google Location API Services Android's LocationManager vorzuziehen. Sollten diese jedoch aufgrund der oben aufgeführten Schwierigkeiten nicht verfügbar sein, stellt der LocationManager von Android die einzige Alternative dar.[14]

[14] https://appus.software/blog/difference-between-locationmanager-and-google-location-api-services

Quellenverzeichnis

[1] Capderou, M.: Handbook of Satellite Orbits. From Kepler to GPS. Springer, 2014. DOI 10.1007/978-3-319-03416-4

[2] Doberstein, D.: Fundamentals of GPS Receivers. A Hardware Approach. Springer, 2012. DOI 10.1007/978-1-4614-0409-5

[3] Schüttler, T.: Satellitennavigation. Wie sie funktioniert und wie sie unseren Alltag beeinflusst. Springer, 2014. DOI 10.1007/978-3-642-53887-2

[4] Louis, D., Müller, L.: Android, Kapitel 17. Hanser, 2016. DOI 10.3139/978-3-446-45112-4