Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

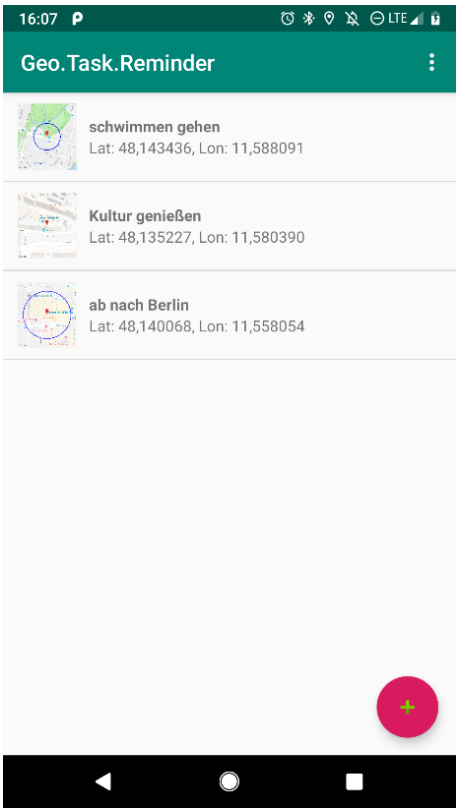
mobile and distributed systems group

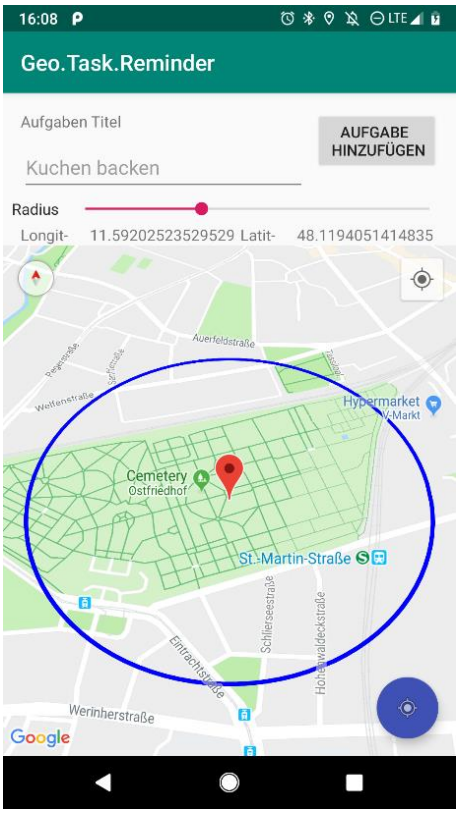
Praktikum Mobile und Verteilte Systeme
Sommersemester 2019
Übungsblatt 3

Ziel der dritten Praxisveranstaltung ist die Einrichtung einer Location-Based Service (LBS) Anwendung wie sie in der Theorie Übung vorgestellt wurden. Hierbei sollen die Verwertung von Standortinformationen und die Einbindung von Kartendaten im Vordergrund stehen.

Aufgabe 3: Geo Task Reminder

In dieser Übung sollen Sie den Einsatz der für die Lokalisierung des Nutzers vorgesehen Android Werkzeugen lernen. Ein zentraler Punkt hierbei ist das Erarbeiten und Ausführen einer „Lokalisierungs-Strategie“ die einerseits die verfügbaren Ressourcen schont, jedoch andererseits immer einen hinreichend genauen Standort liefert. Entwerfen und programmieren Sie hierzu eine Android-App durch die ein Benutzer in Abhängigkeit eines selbstbestimmten Ortes an eine eingetragene Aufgabe erinnert wird. Dies kann z.B. ein intelligenter Einkaufszettel oder Ähnliches sein, der einen Nutzer informiert, wenn er einen bestimmten Bereich betritt, verlässt oder sich in diesem aufhält. Die von Ihnen zu erarbeitende Anwendung soll also eine ortsabhängige Aufgabenerinnerung sein.





a) Vorraussetzungen:

Damit Sie die geforderte Funktionalität in ihre Android App implementieren können, setzen Sie sich bitte mit den folgenden Konzepten auseinander.

1. Sorgen sie dafür, dass ihr Android SDK API mindestens in Level-26 installiert ist.
2. Zur Darstellung von Kartendaten innerhalb von *GoogleMaps* benötigen Sie einen *GoogleMaps-ApiKey*
   Eine detaillierte Anleitung zur Einrichtung der GoogleMaps Api auf Android finden Sie unter: https://developers.google.com/maps/documentation/android-sdk/start
3. Setzen Sie sich mit einer Positionierungsstrategie auseinander. Wann soll ihre Anwendung mit welcher Genauigkeit im aktuellen Kontext der App einen Standort erfragen?
4. Machen Sie sich mit der Benutzung des Android *FusedLocationClients*, des Android *GeofencingClients* und der Verwaltung von *Notifications* vertraut.
5. Sollten Sie noch keinen Kontakt mit dem Wechsel zwischen *Activities* und der Übergabe von Inhalten in gekapselten *Intent*s gehabt haben, sollten Sie sich spätestens jetzt damit auseinandersetzen.
6. Wählen Sie zur Umsetzung der Persistenz ihrer Anwendungsdaten eine entsprechende *Lösung*.

b) Funktionalität

Die von Ihnen zu konzipierende Anwendung sollte die folgenden funktionalen Anforderungen erfüllen:

1. Eingabe von Aufgaben-Objekten mit den Attributen {Name / Aufgabe, Latitude, Longitude, Radius (, Ausführungszeitraum)}
2. Präsentation der eingetragenen Aufgaben in einer *List-* / *GridView* mit einem Vorschaubild aus *GoogleMaps* als Thumbnail, das die Position der Aufgabe zeigt.
3. Sorgen Sie für die Darstellung der Aufgaben als Markierungen auf einer *GoogleMap*. Diese Marker besitzen als *Label* den Namen der Aufgabe.
4. Sorgen Sie für Persistenz der Anwendungs-Daten über das Schließen der Anwendung heraus.
5. Betritt der Benutzer einen kreisrunden Bereich um die gegebene Position, soll eine *Notification* ausgelöst werden, die den Benutzer an seine Aufgabe erinnern soll.
6. Bei Nutzer Interaktion mit der *Notification* soll die Aufgabe aus Ihrer Anwendung entfernt werden.
7. Erweitern Sie diese Funktionen nach Belieben. Bauen Sie z.B. eine weitere Abhängigkeit wie die aktuelle Zeit ein (nicht jedes Geschäft hat 24/7 geöffnet).

c) Stolpersteine und Lösungen:

Während der Konzeption des Übungsblattes sind uns die folgenden Hürden in der Umsetzung aufgefallen. Die Hinweise in dieser Sektion sollen Ihnen als Hilfestellung dienen.

1. Dependecies:

   Neben der Android Bibliotheken, die Android Studio für Sie importiert, müssen Sie die verfügbaren Pakete, aus denen diese entnommen werden selbstständig bereitstellen. Für die Umsetzung unserer Lösung verwendeten wir die folgenden Einträge neben den schon vorhandenen:

```
dependencies {
    // compile 'com.google.code.gson:gson:2.8.2'
    implementation 'com.google.android.gms:play-services-location:16.0.0'
    implementation 'com.google.android.gms:play-services-maps:16.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:design:28.0.0'
    implementation "com.fasterxml.jackson.module:jackson-module-kotlin:2.9.6"
}
```

2. Permissions:

   Damit Sie auf verschiedene Komponenten des Android Frameworks zugreifen können, sollten Sie von Ihrem Anwender die folgenden Berechtigungen einfordern, die sie in Ihrem Manifest angeben:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. ListView Updates:

   Achten Sie im Umgang mit ListViews \ Gridviews darauf, dass Sie Ihrem Adapter darüber informieren, nachdem es zu einer Änderung der darunterliegenden Datensammlung kam, dass Stichwort heißt:

```
listViewAdapter.notifyDataSetChanged()
```

4. Timing:

   Wann soll was passieren? Das Timing ist essentiell, wenn Sie mit dem Android FusedLocationClient arbeiten. Unsere Lösung sieht vor, Standortdaten erst zu verwenden wenn diese auch wirklich vorliegen:

```
try {
    val fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
    fusedLocationClient.flushLocations()
    if (fusedLocationClient.locationAvailability.isComplete){
        result = fusedLocationClient.lastLocation.result
    } else {
        //result =  null
    }
} catch (Exception: SecurityException){
    Log.e(objectTAG, "I have no permission.")
```

```
    askForPermission()
    if (locationController.checkAllPermissions() && root){
        getPosition(root = false)
    }
    //result =  null
}
```

5. Map Objekt Referenzen:

   Aufgrund einer schwammigen Dokumentation, hier ein Hinweis. Referenzen auf Map-Objekte wie Marker oder Kreise lassen sich beim Hinzufügen zur Karte speichern:

```
mMarker = mMap.addMarker(MarkerOptions().position(gpsPos).draggable(true))
mFence = mMap.addCircle(CircleOptions().center(gpsPos).radius(geoFenceRadius))
mMap.setOnMarkerDragListener(MarkerDragListener())
mMap.setOnMapClickListener(ClickListener())
```

6. Geofences entfernen:

   GeoFences lassen sich auch aufgrund ihrer Identifikation (String) wieder entfernen:

```
val geoFencingClient = GeofencingClient(mContext)
geoFencingClient.removeGeofences(listOf(taskName)).run {
    addOnSuccessListener {
        // Geofences removed
        updateToPrefs(taskName)
        Log.d(objectTAG, "GeoFence Removed!")
    }
    addOnFailureListener {
        // Failed to remove geofences
        Log.d(objectTAG, "Cannot remove GeoFence!")
    }
}
```

7. Persistenz:

   Anwendungsdaten können auf sehr verschiedene Arten für die Zukunft aufbewahrt werden. Eine sehr einfache Variante, die für unseren Anwendungsfall mehr als ausreichend ist, stellen die *Shared Preferences* dar. Hier sehen Sie wie man serialisierbare Objekte dort ablegt.

```
private fun writeListToPrefs(list: MutableList<NewTask>): Boolean{
    prefs = mContext.getSharedPreferences(PREFSNAME, 0)
    val editor = prefs.edit()

    val encodedList = mObjectMapper.writeValueAsString(list)
    editor.putString(FENCELIST, encodedList)
    editor.apply()
    return true
}
```

8. Notification:

   Legen Sie doch einen Service mit dem Versprechen an, in Zukunft ausgeführt zu werden.

```
class GeoFenceTransitionsIntentService: IntentService("GeoFenceTransitionsIntent")
{
    private val objectTAG: String = TAG + "_(ServiceIntent): "
    private lateinit var notificationManager: NotificationManager
    private lateinit var geoFenceController: GeoFenceController

    override fun onCreate() {
        super.onCreate()
        notificationManager = getSystemService(NOTIFICATION_SERVICE) as
NotificationManager}
```