# CSC 392/492: Mobile Applications Development for Android II

## Assignment 3 – Walking Tours App *(400 pts)*

Uses: Google Maps Activity, Google Location Services, Location Listener, Geocoding, Geofencing, Full-screen, Custom Fonts

### A) Walking Tours App General Description

- This app is your own personal walking tour guide for an extensive number of Chicago's famous architectural "must-see" buildings such as Aqua Tower, Willis Tower, Chicago Theatre, or Rookery Building. Use this self-guided walking tour to explore the most famous constructions of the Loop District, Chicago.
- The Walking Tours app notifies the user of nearby architectural wonders when they are near the building location. Tapping on a notification displays information such as a photo of the building, the building name, address, and a description of its architectural significance.
- This app uses a Google Maps Activity to display a map showing the tour's path, the user's nearby surroundings, the Geofence ranges, and to track and display their movement as they proceed on their tour.

### B) Behavior Use Cases and Activities:

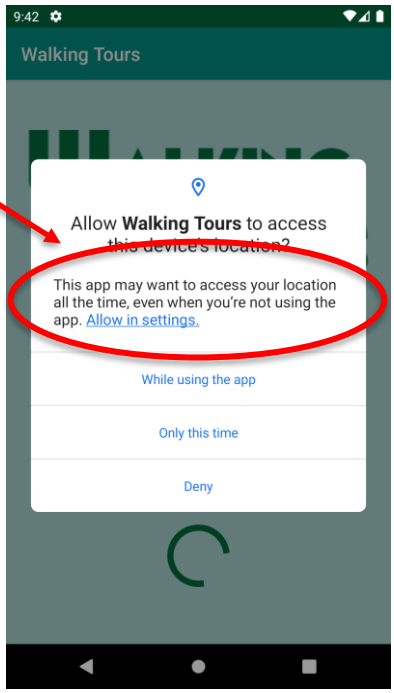
- The user's live location is used to update their map position.
- The Google Maps Activity will make use of location data provided continuously by a Location Listener to determine the user's current location and to plot their location on the map.
- The map continuously centers itself on the user's current location. The current location is shown via an icon that is oriented to show the users direction of travel. Their path on the map is displayed as a solid line that marks where they have been.
- Geofences are loaded from a cloud-based source so they can change and update without requiring app updates (http://www.christopherhield.com/data/WalkingTourContent.json). As the user moves, they are alerted if they enter the range of any of the Geofences.
- The architectural information offered to the user can be viewed by tapping on the Walking Tours notifications that will appear in the notification bar. Multiple notifications may be present if the user crosses several Geofence boundaries.
- Walking Tours notification content includes the name of the building, a photo of the building, the address, and a detailed description of its architectural significance.
- When the app is stopped, the Walking Tours notifications should be removed.
- When the app is started, no history of previously displayed notifications or any previous route path should be displayed.

## C) Application behavior and design

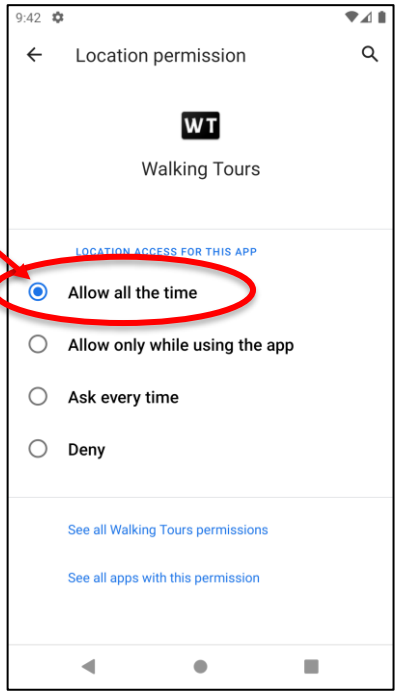
### 1) Startup "splash" activity

- Use the Splash activity to perform initial tasks while the splash logo is displayed:
  - If the app does not have both ACCESS_FINE_LOCATION and ACCESS_BACKGROUND_LOCATION permissions, they must be requested. *Note that ACCESS_BACKGROUND_LOCATION requests offer an "all the time" option – which we want to use (see images).*
    - Upon granting permissions, you need to request high accuracy location services (see below).
    - If the app already has these permissions, you can immediately request high accuracy location services (see below).
    - If the ACCESS_FINE_LOCATION and ACCESS_BACKGROUND_LOCATION permissions are not granted, use an AlertDialog to inform the user that that the app cannot be used without these permissions. Then they tap OK in the dialog, exit the app (`finish()`).
  - Upon success of the high accuracy location services request, you can open the Maps activity (with slide-up transition animation).
  - Upon failure of the high accuracy location services request, you should attempt to resolve the API exception (as we showed in class examples). If that succeeds, you can open the Maps activity (with slide-up transition animation). If not, use an AlertDialog to inform the user that that the app cannot be used without these services. Then they tap OK in the dialog, exit the app (`finish()`).
- Note, as you know, once the app has ACCESS_FINE_LOCATION and ACCESS_BACKGROUND_LOCATION permissions, you do not need to ask the user for these again. However, high accuracy location services requests must be made *every time* the app is started, as they the services may have been disabled since the last time the app was executed.

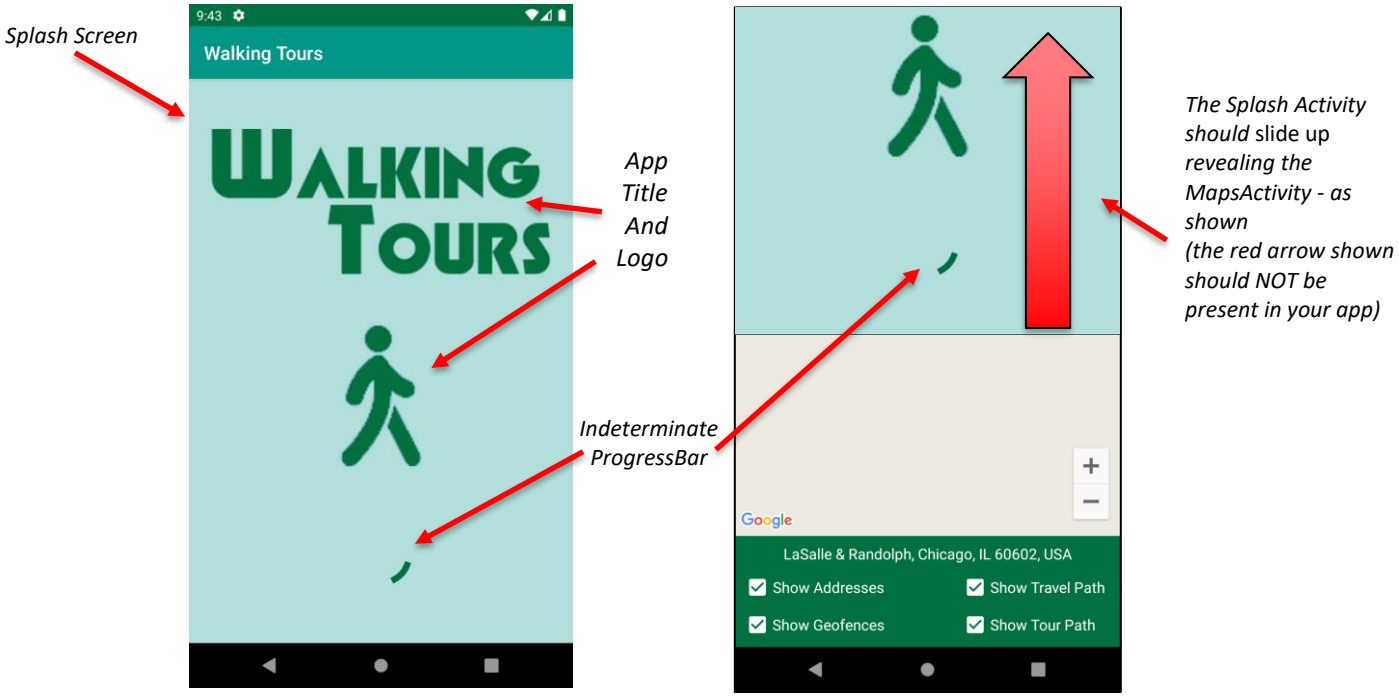
*Background Location Permission Needed*




*Select "Allow all the time", then tap the left-arrow on the upper-left of the screen.*



- Once permissions requests and location services requests have been approved/completed, you can open the Maps activity (with slide-up transition animation).




## 2) Maps Activity (Google Map):

- The Maps Activity should operate in gull-screen "immersive" mode.
- The Maps Activity needs to own a List of LatLng objects – this will be used to store location updates.
- Upon starting the Maps Activity, start the Fence Manager thread (described later) and initialize the map. *Note – remember, you cannot use the map until the onMapReady method is called back once the google map fragment is ready to use.*
- Once the map is ready:
  - Set the UI as follows (feel free to experiment with the various map settings, but the required features and functionality must be present:

```
mMap.animateCamera(CameraUpdateFactory.zoomTo(16));
mMap.getUiSettings().setRotateGesturesEnabled(false);
mMap.setBuildingsEnabled(true);
mMap.getUiSettings().setZoomControlsEnabled(true);
mMap.getUiSettings().setCompassEnabled(true);
```

  - Create a Location Listener and request location updates using the GPS provider, and set the min time to 1000ms and the min distance to 1 meter.
  - Your location listener should send all Location updates back to the Maps Activity

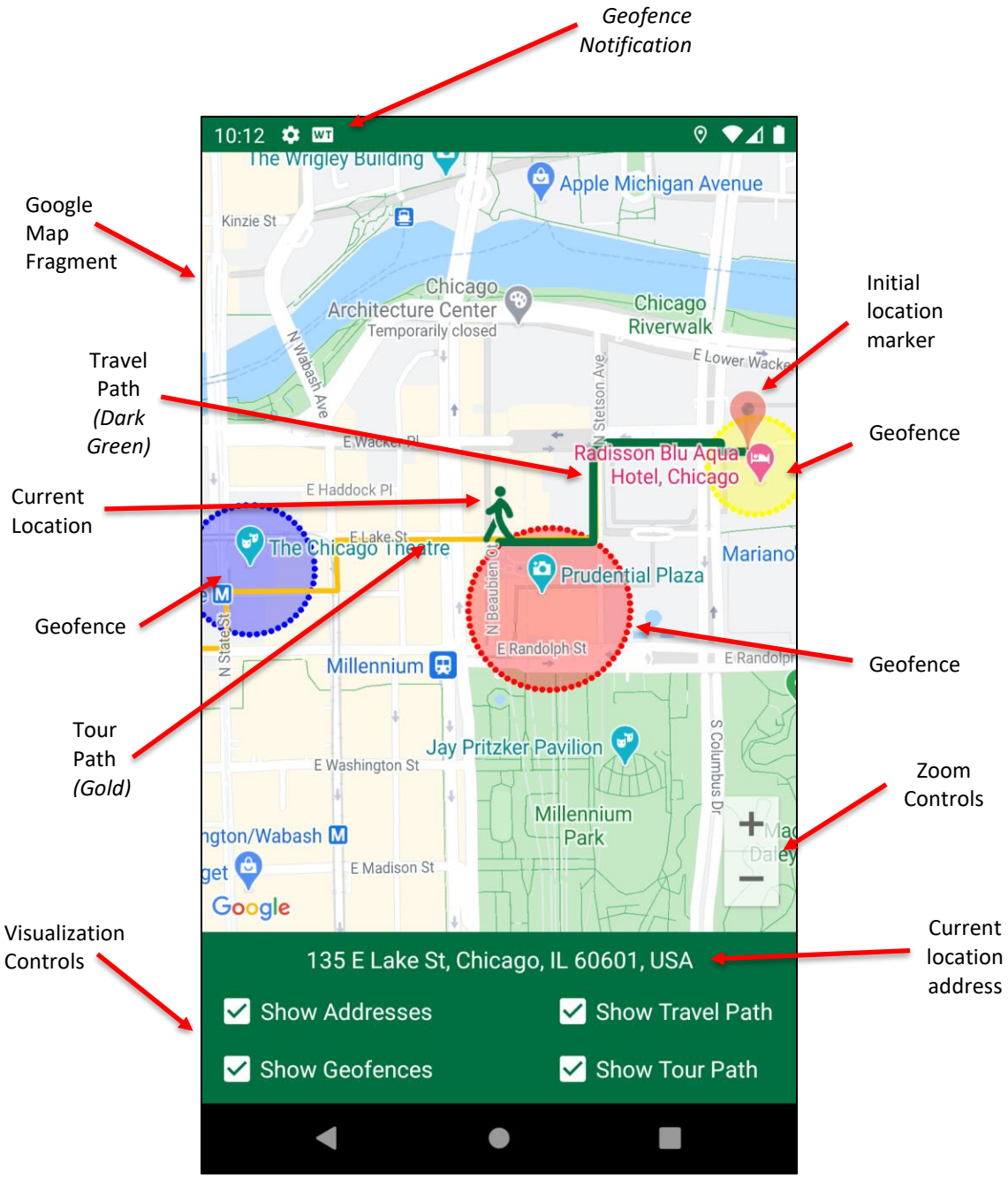
- You need a method to receive the coordinate points that make up the tour path from the Fence Manager (described later) – passed in as a list of LatLng points. The function must create and add a polyline to the map using the list of points passed in.
- You need a method to receive and display a geofence on the map. This method receives an object containing the GeoFence data, and uses that data to display the GeoFence circle on the screen.
- You should remove your location listener from the location manager in the onDestroy method (not in onPause or onStop).
- You will need individual methods to turn on/off the addresses, geofences, tour path, and travel path. These should be called when the corresponding checkbox on the UI is checked/unchecked. These items (Polylines, Circles, etc) can be made visible or invisible via their *setVisible* method.
- Location updates received by the Maps Activity from the location listener should be handled as follows:
  - Create a LatLng object using the latitude and longitude from the location object.
  - Add that LatLng object to the Maps Activity's list of LatLng objects
  - If the LatLng is the first coordinate added to the list, consider that the starting location and add a semi-transparent marker at that point.
  - Use a GeoCoder to get the street address of the LatLng location. Use that street address to update the current address on the screen.
  - If there are now more than one points in the list of LatLng objects, remove any previous travel-path polyline and redraw the travel-path polyline using the Maps Activity's list of LatLng objects
  - Calculate the radius of the current location icon as:

```
float z = mMap.getCameraPosition().zoom;
float factor = (float) ((35.0 / 2.0 * z) - (355.0 / 2.0));
float multiplier = ((7.0f / 7200.0f) * screenWidth) - (1.0f / 20.0f);
float radius = factor * multiplier;
```

  - If the Location object's bearing is less than 180 degrees, use the right-facing walker icon

    else if the bearing is greater than 180 degrees, use the left-facing walker icon
  - Resize the icon using the calculated radius value as the width and height for the icon
  - Create the icon's BitmapDescriptor from the resized icon.
  - Remove any previous current-location marker and add a new marker using the location passed in and the icon's BitmapDescriptor
  - Move the camera to the new LatLng location.

**Maps Activity (Google Map):**



### 3) GeoFence Manager

The Geofence manager class is responsible for loading the *GeoFence* and *tour-path* data. GeoFences will be created and activated and then passed to the MapsActivity for display. A list of LatLng objects is created from the tour path data, that list is passed to the MapsActivity for display.

The FenceManager will be very similar to the FenceMgr class found in our GeoFencing examples with the following exceptions:

- The FenceManager will need to own a HashMap of walking tour building objects, using the building name as the key *(add an accessor that returns a walking tour building object based on the name passed to it).*
- At the end of the FenceManager constructor (after creating the GeofencingClient), initiate the download of the fence & tour-path JSON data file in its own thread. *JSON file format is described later.*
- The Geofence/tour-path JSON download thread:
  - GeoFences: For each geofence found in the JSON file:
    - Create a walking tour building object using the data from the JSON Object
    - Create a Geofence object using the building name (as the request id), and the latitude, longitude, and fence radius to define the circular region. Set the Notification Responsiveness to 1000ms, the Transition Types to GEOFENCE_TRANSITION_ENTER, and the Expiration Duration as NEVER_EXPIRE.
    - Create a GeofencingRequest object using the Geofence object.
    - Build a geofence Pending Intent as was done in class examples.
    - Using the GeofencingClient object, add the new GeoFence (with the previously created pending intent).
    - Save the walking tour building object in the HashMap described earlier, using the building name as the key.
  - Tour Path: For each latitude/longitude pair found in the JSON file:
    - *Note: before loading these, declare a List of LatLng objects to store the values*
    - Create a LatLng object using the latitude/longitude pair.
    - Add the LatLng object to the List of LatLng objects
    - Once all points are loaded in the list, call the method in the Maps Activity you previously created that accepts this List of LatLng objects and pass it the list.
  - When this is all done, for each walking tour site object in the HashMap described earlier, call the Maps Activity method you previously created that accepts the object and draws the geofence on the map.

*Remember to add android:usesCleartextTraffic="true" in your manifest as the class examples have done.*

4) GeofenceBroadcastReceiver

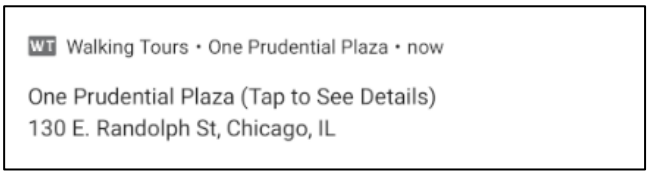
The Geofence Broadcast Receiver receives broadcasts when a geofence area is entered (you should use GEOFENCE_TRANSITION_ENTER only), and from that broadcast generates a notification on the user's device.

When you receive a broadcast via the onReceive method, first get the GeofencingEvent from the intent passed in. Then get the geofence transition-type from the GeofencingEvent. If the transition type is GEOFENCE_TRANSITION_ENTER, do the following (otherwise ignore it):

- Get the list of Geofence objects that have been triggered from the GeofencingEvent:

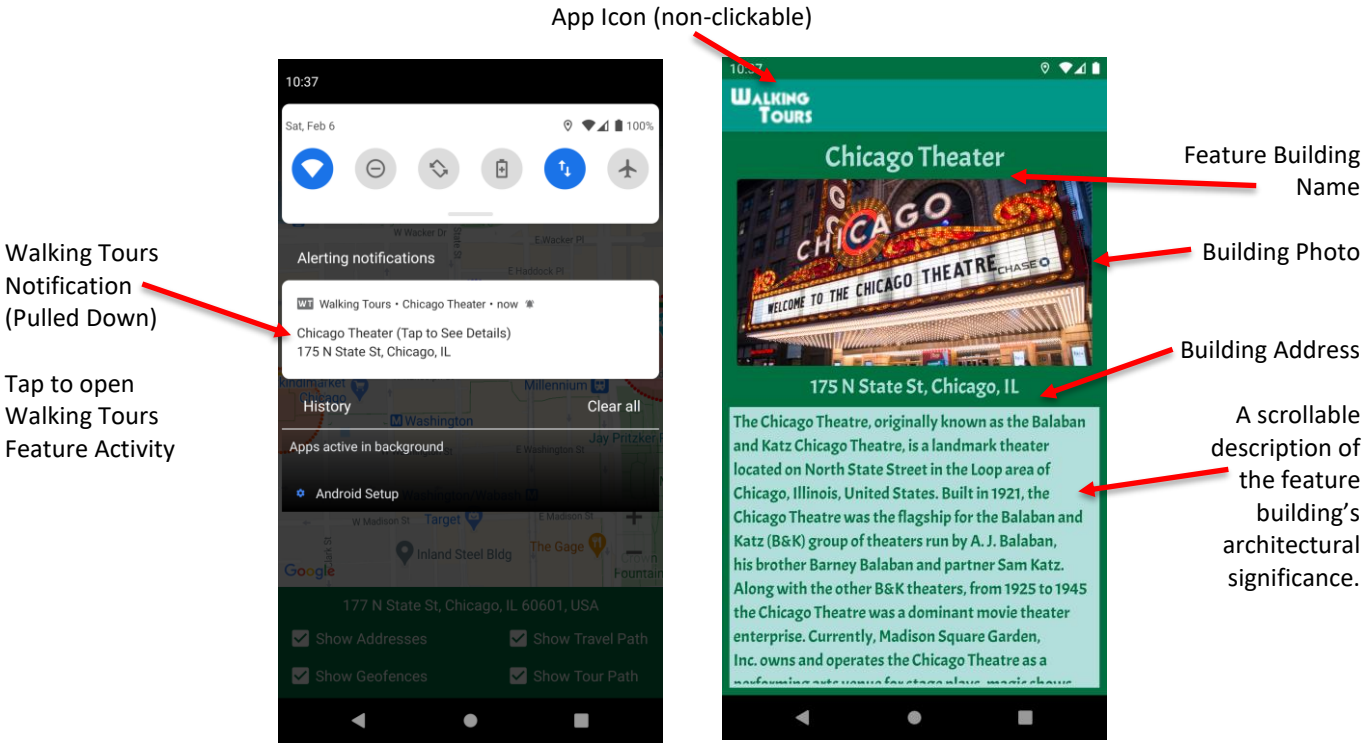
```
List<Geofence> geoFences = geofencingEvent.getTriggeringGeofences();
```

- For each Geofence object in the list, get the walking tour building object from the fence manager using the request id (the site name) found in the Geofence object. Use that walking tour building object to generate a notification.
- Generate a Notification, with the walking tour building object added as an extra to the notification's intent. The notification should have the walking tour building name as the notification *title* with " (Tap to See Details)" appended to the name, the building name as the notification *sub-text*, the walking tour building address as the *content text*. Example:




5) Walking Tours Notifications and Building Activity examples *(Note: There are no functional elements in this activity).*

Walking Tours feature *data is specified in a cloud-based JSON file (described in the next section)*



## D) Geofence Data Download

The Geofences to be used in this app are defined in a JSON file that should be accessed from within your app at: http://www.christopherhield.com/data/WalkingTourContent.json

The fence JSON file contains one outer JSON Object with TWO top-level fields (JSONArrays). One is called “fences”, which refers to a JSON Array containing the fences, and the other is called “path”, which refers to a JSON Array containing the latitude and longitude points that make up the tour path. A sample of that file is shown:

```
{
  "fences": [ ← A JSONArray of JSONObjects (buildings)
    {
      "id": "Aqua Tower",  ← Building Name
      "address": "225 N Columbus Dr, Chicago, IL",  ← Building Address
      "latitude": 41.886542999999996,  ← Building Latitude
      "longitude": -87.61998,  ← Building Longitude
      "radius": 60.0,  ← This is used when creating the Geofence to specify its radius
      "description": "Aqua is an 82-story mixed-use ......",  ← Building Description
      "fenceColor": "#FFFF00", ← Used to set fence color
      "image": "https://s3.amazonaws.com......jpg" ← Building Image
    },
    {
      "id": "One Prudential Plaza",
      "address": "130 E. Randolph St, Chicago, IL",
      "latitude": 41.8849569,
      "longitude": -87.62305239999999,
      "radius": 100.0,
      "description": "When completed in 1955,.......",
      "fenceColor": "#FF0000",
      "image": "https://www.aisc.org/contentassets/......jpeg"
    },
    ...
    {
      "id": "Field Museum of Natural History",
      "address": "1400 S Lake Shore Dr, Chicago, IL",
      "latitude": 41.8661733,
      "longitude": -87.61698620000001,
      "radius": 100.0,
      "description": "The Field Museum of Natural .....",
      "fenceColor": "#22DD22",
      "image": "https://cdn.britannica.com/76/......jpg"
    }
  ],
  "path": [  ← A JSONArray of Strings (latitude-longitude pairs)
    "-87.62, 41.88673",
    "-87.62029, 41.88672",  ← Latitude/Longitude Pairs
    "-87.6203, 41.88672",
    "-87.62042, 41.88671",
    "-87.62052, 41.88671",
    "-87.62052, 41.88675",
    "-87.6205, 41.88682",
    "-87.6206, 41.88682",
    "-87.62061, 41.88679",
    "-87.62074, 41.88679",
    ...
    "-87.61676, 41.86546",
    "-87.617, 41.86545",
    "-87.617, 41.86565"
  ]
}
```

### E) Custom Font

All TextViews, EditTexts, CheckBoxes, etc in this app should use the provided "Acme-Regular" font. The example code below is provided as a refresher/guide on using custom fonts.

*Note, the provided Acme-Regular.ttf font file must be put in the following path:*
*Project ➔ app ➔ src ➔ main ➔ assets ➔ fonts ➔ Acme-Regular.ttf*

To load:

```
Typeface textFont = Typeface.createFromAsset(getAssets(), "fonts/Acme-Regular.ttf");
```

To use:

```
myText.setTypeface(textFont);
```

### F) Provided Images/Files

fence_notif.png:
*Used as notification "small icon"*


home_image.png:
*Used as home indictor icon*


logo.png:
*Used on Splash Activity*


Font: Acme-Regular.ttf:
*Used for all text elements*

walker_left.png:
*Used as map marker icon*

walker_right.png:
*Used as map marker icon and on Splash Activity*

Audio file: notif_sound.mp3:
*Used as notification sound*

**Assignment Assistance**

If you are stuck on an assignment problem that you have exhaustively researched and/or debugged yourself, you can put your project in a file share (like Google Drive), make the link publicly available, and email the link so that I can examine the problem.

All emailed assistance requests must include a detailed description of the problem, the steps and/or data used to create the problem, and the details of what steps you have already taken in trying to determine the source of the problem.

***Note: To make your submission zip file smaller, before zipping your project file you must remove the ".gradle" folder (found in your project's root directory), and remove the "build" folder (found in the "app" folder in your project's root directory).***

**Submissions & Grading**

1) Submissions must consist of your zipped project folder. For submission - before zipping your project file you *must* remove the ".gradle" folder (found in your project's root directory), and remove the "build" folder (found in the "app" folder in your project's root directory). Submissions not following these requirements will be penalized.
2) Submissions should reflect the concepts and practices we cover in class, and the requirements specified in this document.
3) Late submissions will be penalized by 10% per week late. (i.e., from one second late to 1 week late: 10% penalty, from one week late to 2 weeks late: 20% penalty, etc.).
4) Grading will be based upon the presence and proper functionality of all features and behaviors described in this document.

*If you do not understand anything in this handout, please ask.*

*Otherwise the assumption is that you understand the content.*

***Unsure? Ask!***