Lab: MapRuler (10 pts.)

Lab Objectives:

- Learn how to create a project with Google Maps
- Learn how to find the current location with FusedLocationProvider
- Learn how to use Geocoder

What to Turn in:

- Demonstrate your working code to a TA.

Goal

The goal of this lab is to create an app that displays a map with a custom theme (2 pts). The app should place a marker on the map at the user's current location (3 pts). The app should print the distance between the user's current location and another location (3 pts). The user should be able to specify that second location as a string (2 pts).

Steps

1. Start Android Studio and create a new project.
    a. When selecting an Activity to start with, **choose No Activity**.
    b. Call the project MapRuler (or whatever you want).
    c. Make sure to use Java (unless you're purposefully using Kotlin).

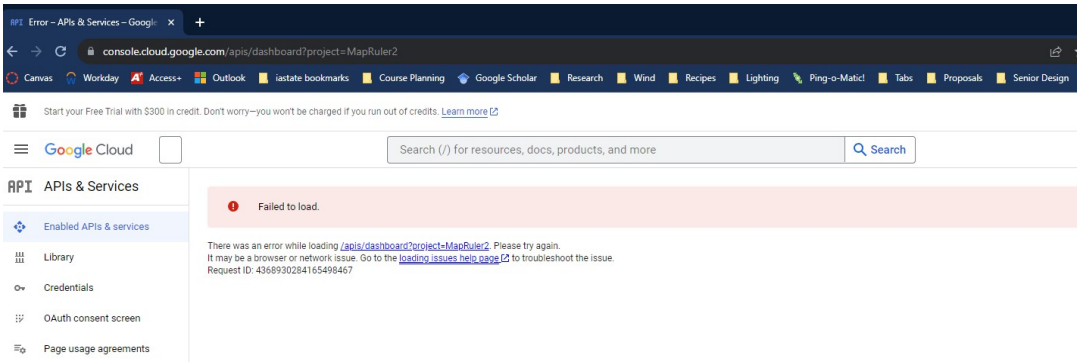
2. After the initial Gradle sync is done, right-click on the "app" directory in the project explorer and choose New->Google->Google Maps Views Activity. You can use the default settings.

3. You should now have an Android project with MapsActivity.java and activity_maps.xml. Run the app. We get the error "Default activity not found." What does that mean? Troubleshoot and fix this problem. *Hint: what did we do differently when we created this project, compared to prior projects? What file might you need to edit to solve a problem of this kind? After making the correct edits, you may need to manually run a Gradle sync (elephant button).*

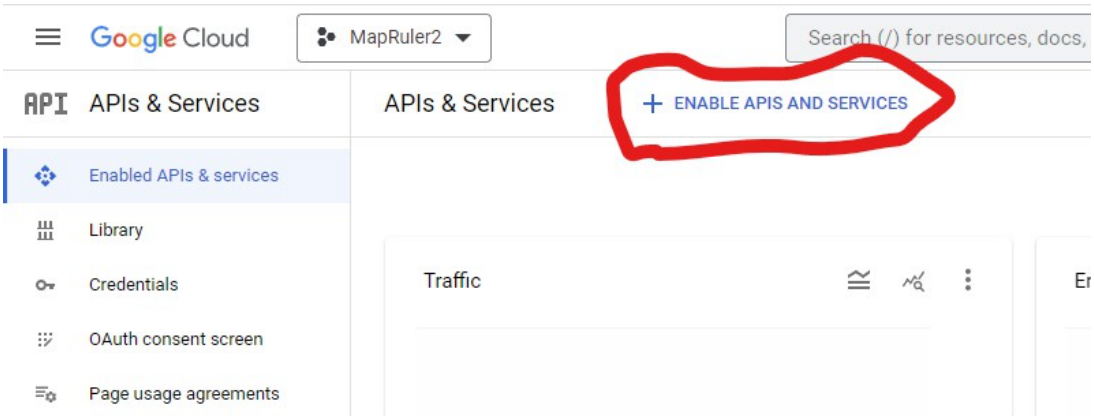
4. Launch the app again. You should get a "blank" map due to an inability to connect to the Google Maps Platform on Google Cloud.

5. Open AndroidManifest.xml and find the section discussing a Google Maps API key.
    a. Open the URL to the documentation on Creating API keys.

b. Follow the steps to access the Credentials page on the Google Maps Platform in the Google Cloud console.
c. Log in if needed (you can use the same Google account you used for the Firebase lab).
d. Create a new Google Cloud project. Name it MapRuler or whatever you would like.
e. **You may be prompted to input payment information.** You can either do this, or you should be able to browse to https://console.cloud.google.com/apis/dashboard?project=your-project-name, replacing "your-project-name" with the name of the project you just created, and skip the payment info.

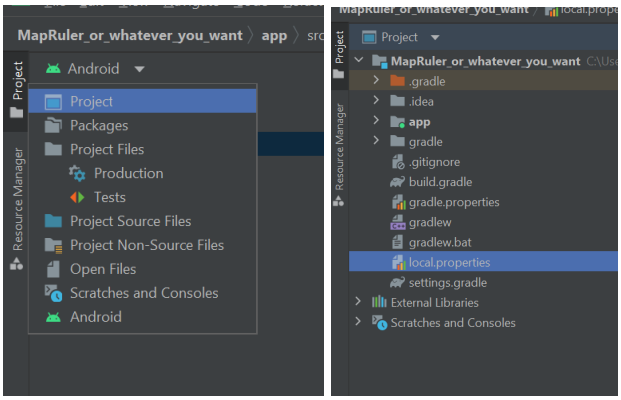


f. In the left menu, click Enabled APIs & services and enable the Maps SDK for Android:

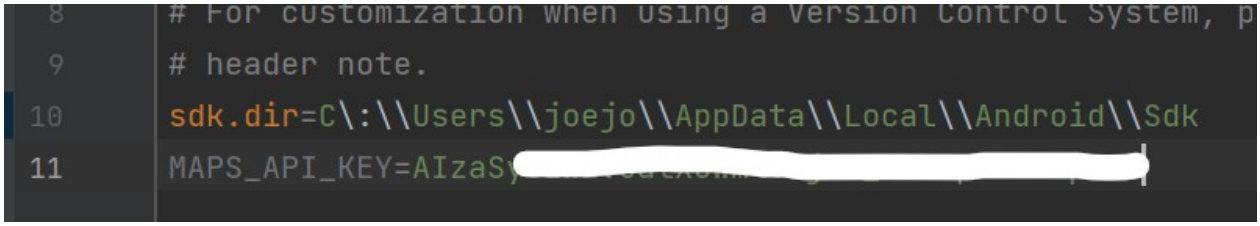


If you get kicked back to the payment info screen, return to the link above and you should find that the API was enabled anyway.

g. Then follow the remainder of the documentation to create a Google Maps API key. This key allows your app (the client) to communicate with the Google Maps API server.
h. Open your local.properties file (the path can be seen in the image below).



In local.properties, create a variable called MAPS_API_KEY and paste your Maps API key as the value:



Then, update the value of the maps meta-data element in AndroidManifest.xml to read “android:value="${MAPS_API_KEY}" />”.

Why not just put the API key directly in the manifest? Consider if you were storing this project in a Git repository. You would almost certainly want to commit your manifest file to the repo. But **you should never commit an API key to a repository**. If someone else discovered this key, they would be able to use the API and you would get charged for it!

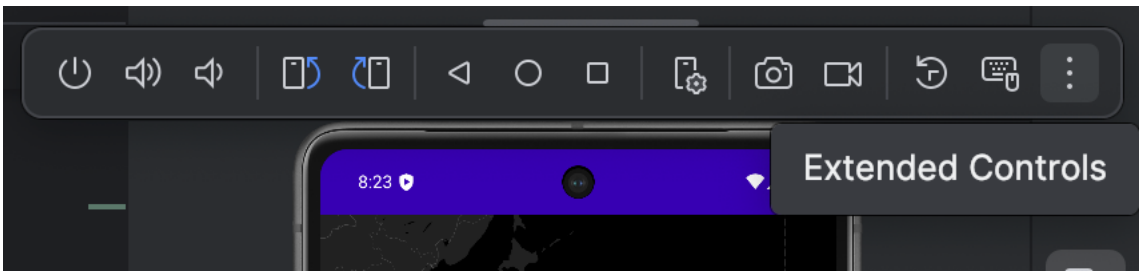
6. Run the application. Now the map should load and probably start you out somewhere in Australia. Let's explore the rest of the code. Answer the below questions for yourself.
    a. Open activity_maps.xml. What's in the layout?
    b. Open MapsActivity.java. What happens in onCreate()? How is the map loaded? Is it loaded synchronously or asynchronously? Where does the marker on the map come from?

7. In the res directory, create a new Android Resource Directory named "raw" for resource type of "raw."
    a. Once you’ve created the raw directory, add a new file to the directory called mapstyle_mine.json
    b. Now create a custom styling file at https://mapstyle.withgoogle.com and insert the generated json into your mapstyle_mine.json file. If prompted to try the

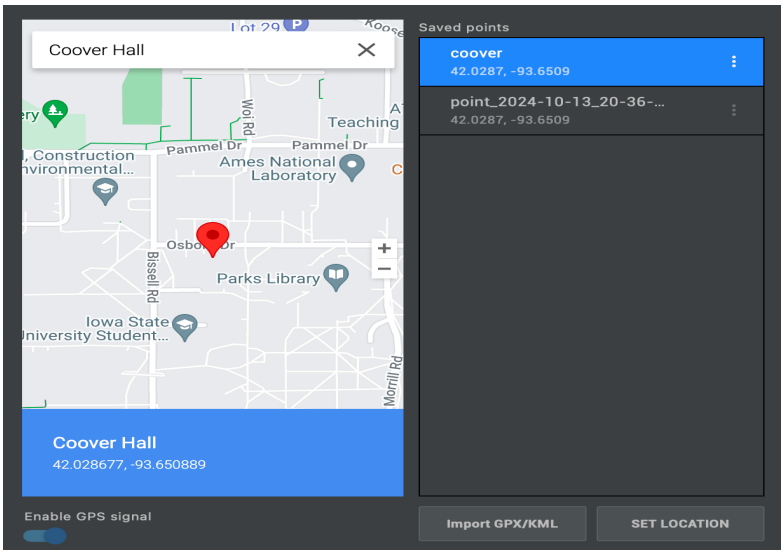
cloud-based styling tools, choose "Use the legacy JSON styling wizard."

8. To apply your style in the app, in onMapReady() of MapsActivity, call setMapStyle() on the GoogleMap object. The parameter should be a MapStyleOptions, which you can obtain by calling the [loadRawResourceStyle() static method of MapStyleOptions](#).

9. The emulator does not support GPS by itself.
   Since you will be working on this within the emulator, go to **extended controls > location**



   Now, search for a location to add and click on **save point**



   Now, click on your saved point and set location
   This will set your GPS location as the point you just selected.

10. Change the code so that the map marker is placed at the user's current location. To do so, follow the steps at https://developer.android.com/training/location/retrieve-current to use FusedLocationProvider.

    You may need to add this into your app-level dependencies:
    implementation("com.google.android.gms:play-services-location:21.0.1")

    Note that you may need to install Google Play services from the SDK Tools tab of the SDK Manager (**Tools** > **SDK Manager** > **SDK Tools** > **Google Play services)**. As discussed in the documentation, you will need to request location permissions (in the manifest and at run-time). If the Location obtained through getLastLocation() is continually null

or not updating, try opening Google Maps on the device to make Google Play services update the location.

11. Now we want to be able to identify a second location (latitude and longitude) in order to measure the distance between there and your current location. But, most humans are much more comfortable using street addresses than latitude and longitude. The process of finding latitude and longitude for a street address is called geocoding.
    a. Use Geocoder, specifically the getFromLocationName method, to get an Address object corresponding to a place of your choosing.
    b. From the Address object, you can get the latitude and longitude.
    c. Use Location.distanceBetween to find the distance between the user's current location and the chosen place.
    d. Convert this distance to miles or kilometers, and display it to the user, e.g. in a Toast.

12. One final step: add an EditText to the bottom of the layout (beneath the Fragment). This EditText should allow the user to type in a location. The app should then display the distance between the user's current location and the location the user entered, as in the previous step.

    Example screenshot: