API Setup for Road Trippy Demo App

Road Trippy requires you to provide API configuration by adding two files to the project source code directories. You must configure a Firebase project as well as two Google Maps APIs, and provide the appropriate configuration as detailed below.

In addition to these instructions, you must create your own HTML file in the app source code as `/app/res/raw/legal_language.html` which must be written to accurately describe your own practices on the subject of User Data Privacy and Licensing terms.

Using the instructions below, you will:

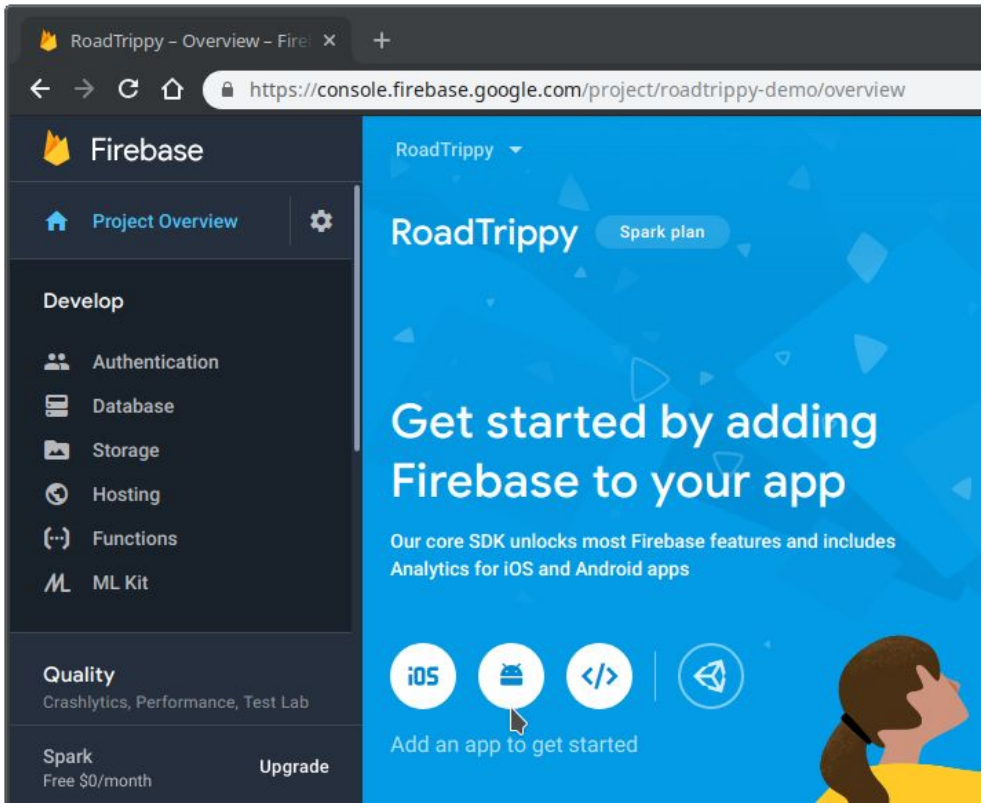
- Create a Firebase project using the free “Spark” service level
- Integrate the Android app with the Firebase project
- Create a Firebase Realtime Database within this Firebase project
- Configure Firebase Authentication within this Firebase project
- Configure Google Maps API access from the Google APIs console

Create & Integrate a Firebase project

Access the Firebase Console at https://console.firebase.google.com and log in if necessary.

Add a new project, using any Project name and Project ID you prefer. Accept the suggested Analytics and Cloud Firestore locations and the default settings, accept the terms and click to Create the project.

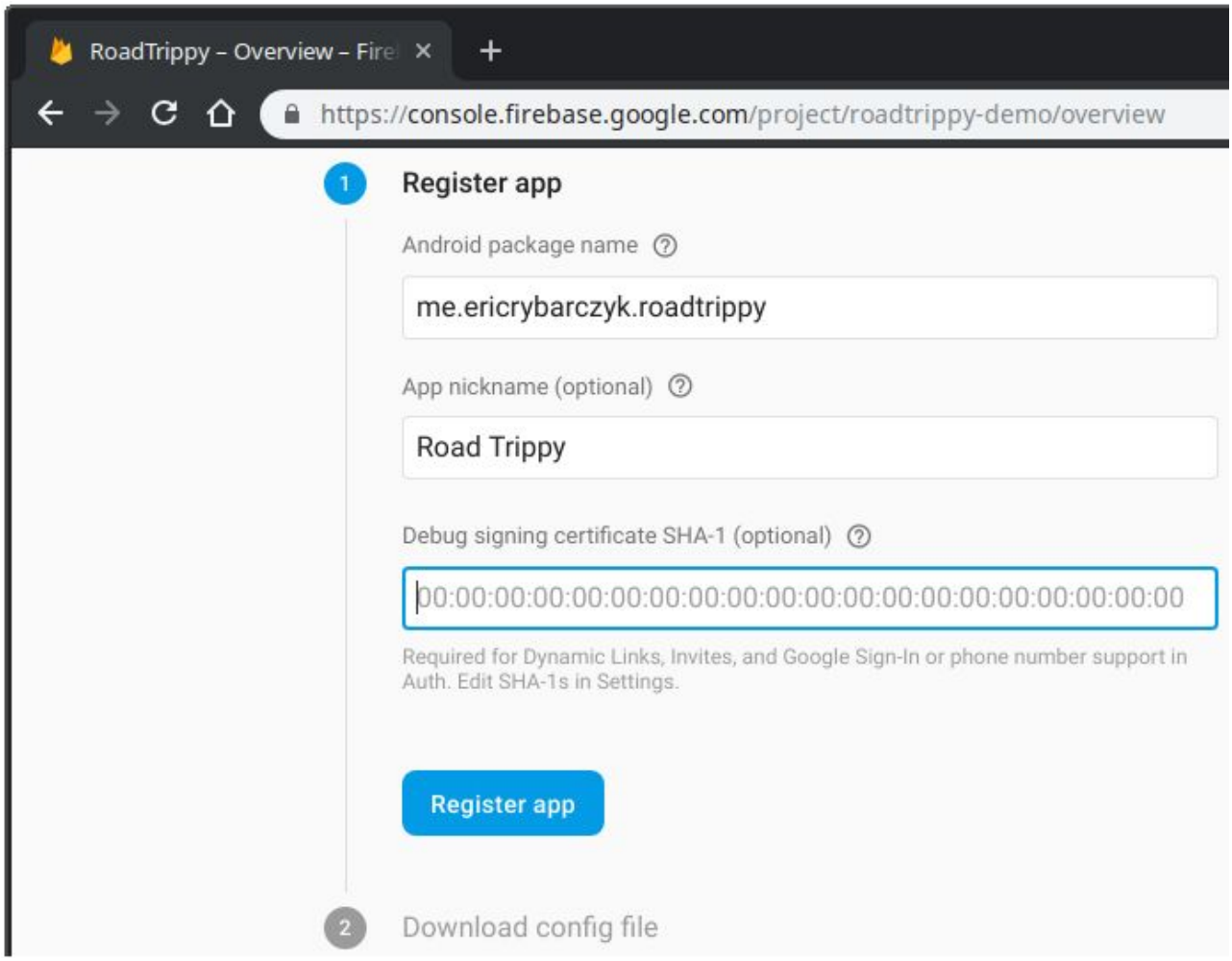
Next, configure the project for Android, by clicking the Android “bugdroid” icon in the main page area shown.



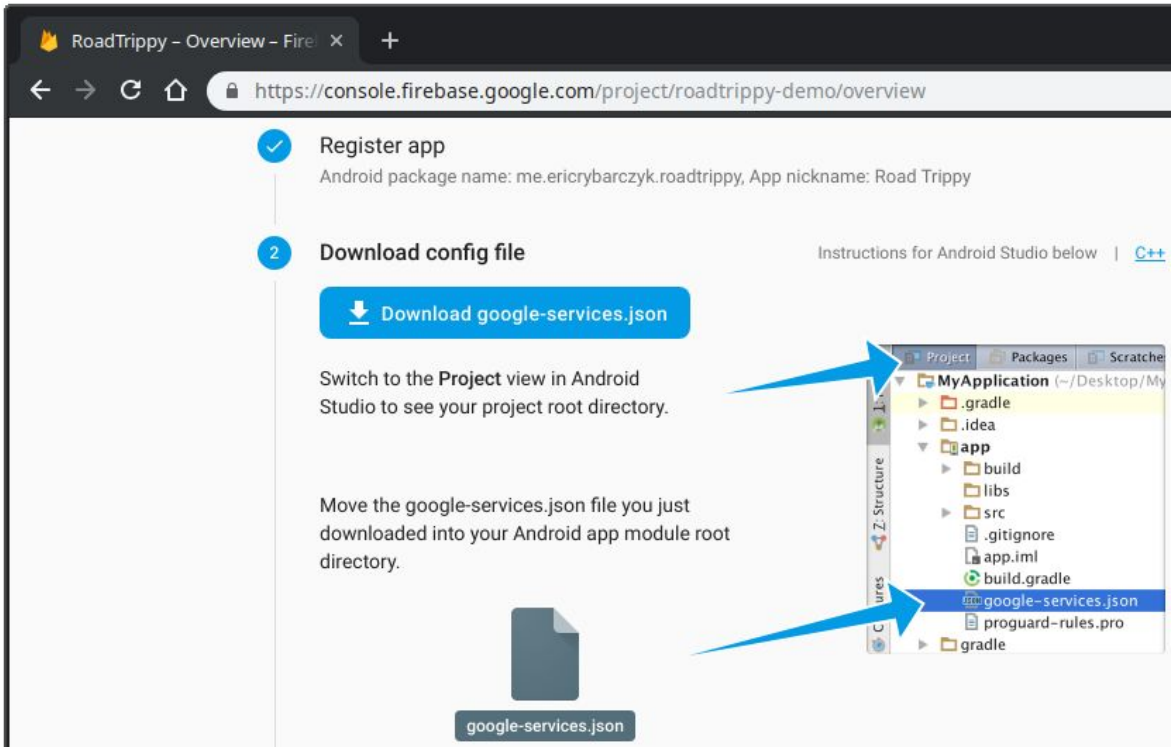
Enter the required app details as shown below. You must include the SHA-1 fingerprint for ***your*** Debug signing certificate because the Road Trippy app includes Google Sign-In functionality.

> **Note**: In my experience, the Java keytool command provided in Google's documentation does not work and returns "`keytool error: java.lang.Exception: Only one command is allowed: both -exportcert and -list were specified.`"
>
> The command below has worked well for me:
>
> ```
> $ keytool -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore
> ```

When prompted to input a password, use "`android`" (no quotes) and then press Enter. The SHA-1 certificate fingerprint will be included in the output. Copy & paste the SHA-1 fingerprint into the field highlighted below.



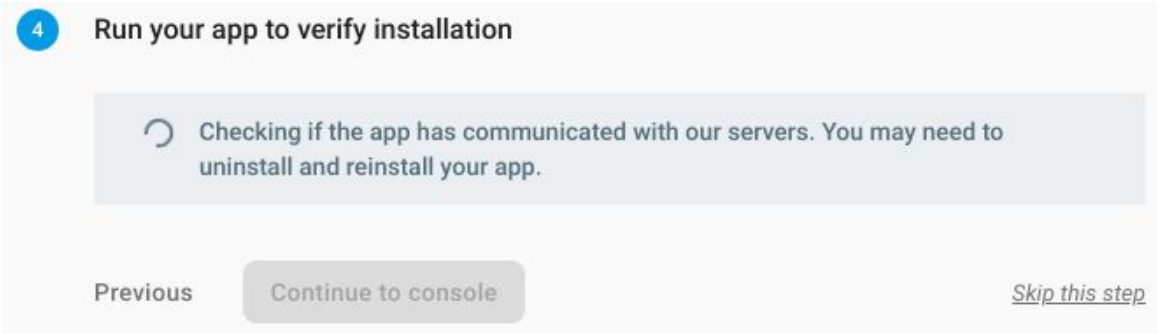
Click "Register app" and then in step 2 you must download the google-services.json file. Add this file to the Android app module root directory: `/app/google-services.json`



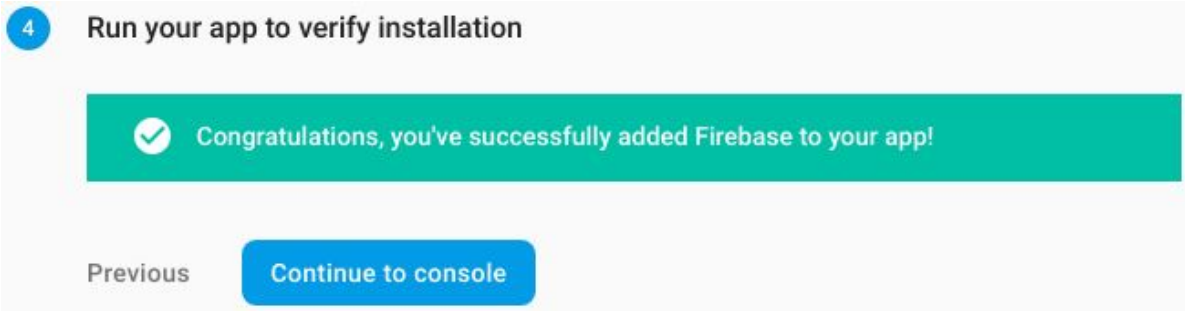
Step 3 can be skipped because the Firebase SDK is already integrated in the app.

Proceed to step 4 and run the app from Android studio using an emulator or physical device. You must be logged in to a Google account in order to run the app, since Google Authentication is a feature of the app.

Note: The app will not be completely functional at this point, but it will make the connection to Firebase.

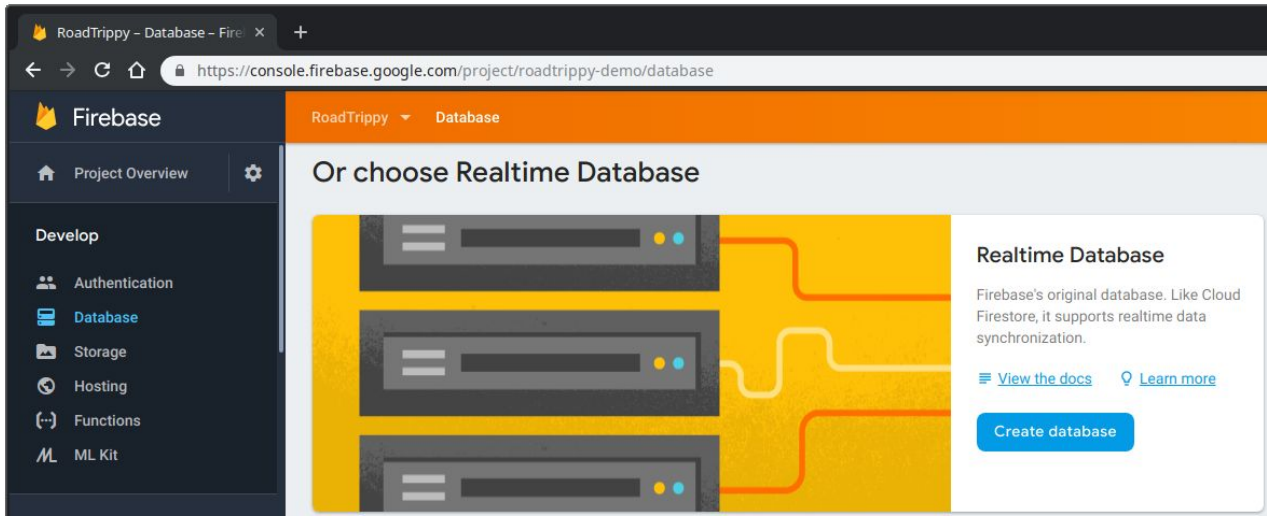


Following the successful launch of the app on your device or emulator you will see the message below.

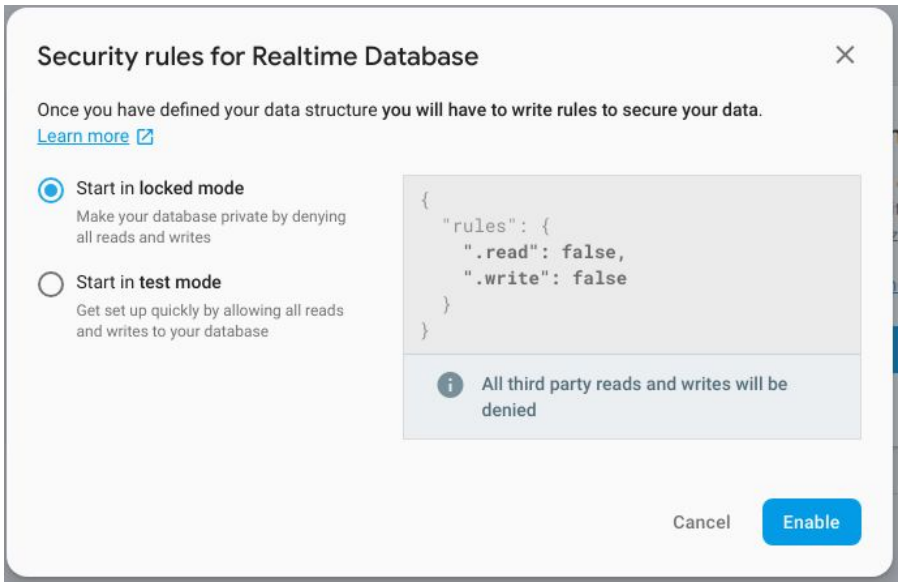


Create a Firebase Realtime Database

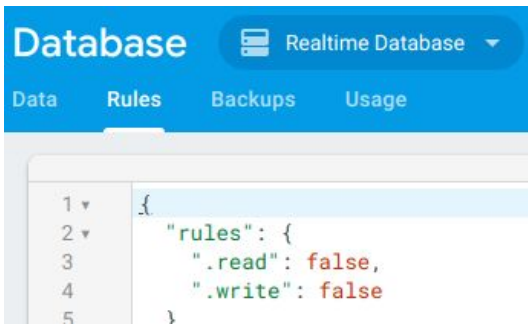
Next, create a Realtime Database. Scroll down to the Realtime Database option as shown below.



Click “Create database” and accept the default Security Rules, which you will modify in a later step. Click “Enable” to initialize the empty database.



Next, click “Rules” and delete the existing default rules content, and replace it with the rules provided below.



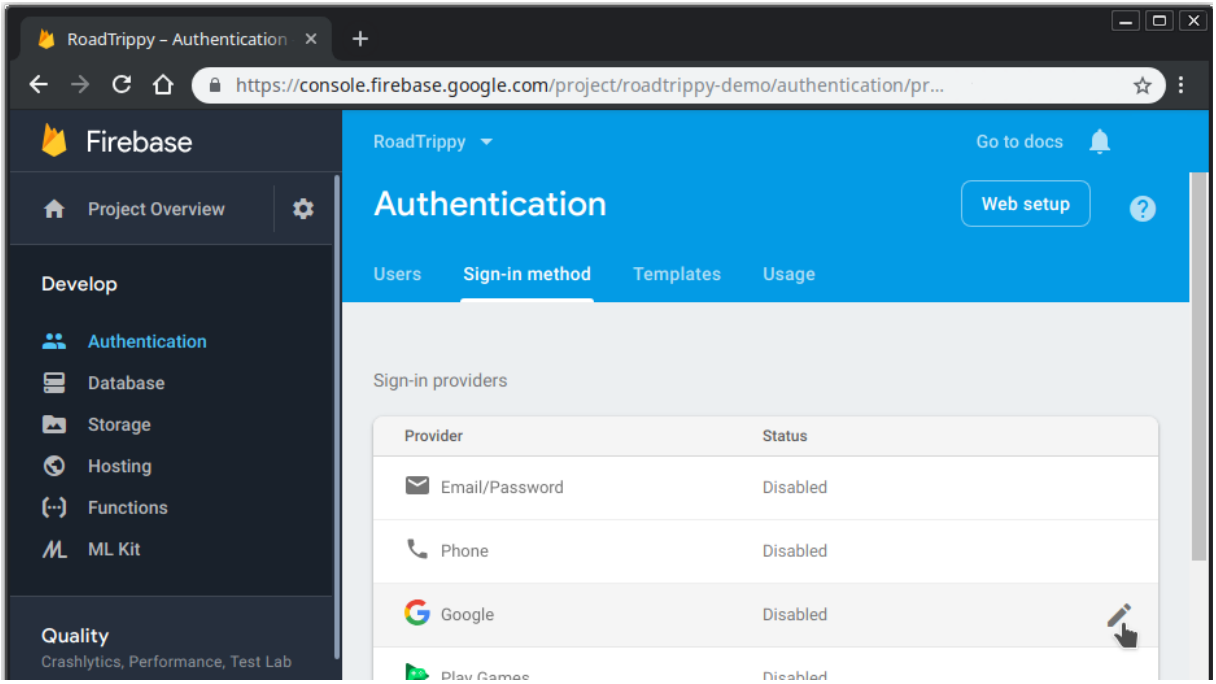
Publish the following rules definition for the Firebase Realtime Database:

```
{
  "rules": {
    "trips": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    },
    "tripdays": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    },
    "tripArchive": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    },
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

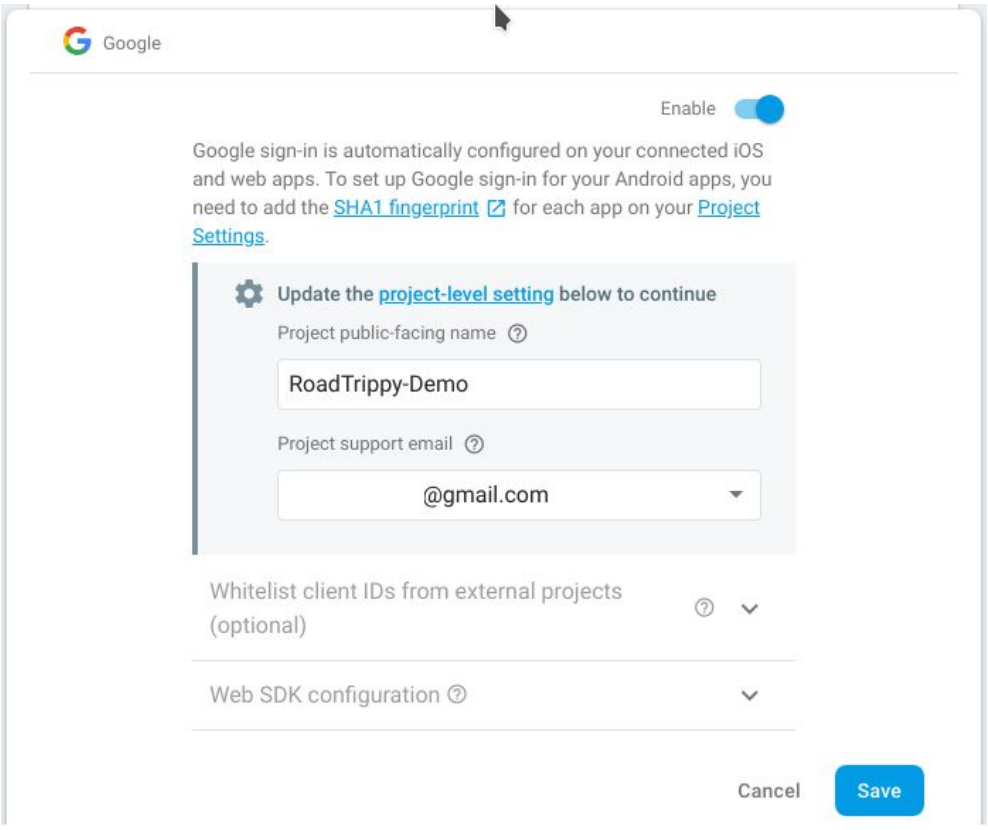
Publish the updated rules by clicking the “Publish” button.

Configure Firebase Authentication

Click the “Authentication” option from the left-side navigation, and then click the “Sign-in method” tab on the Authentication page. Click on the Google option from the list, as shown below.

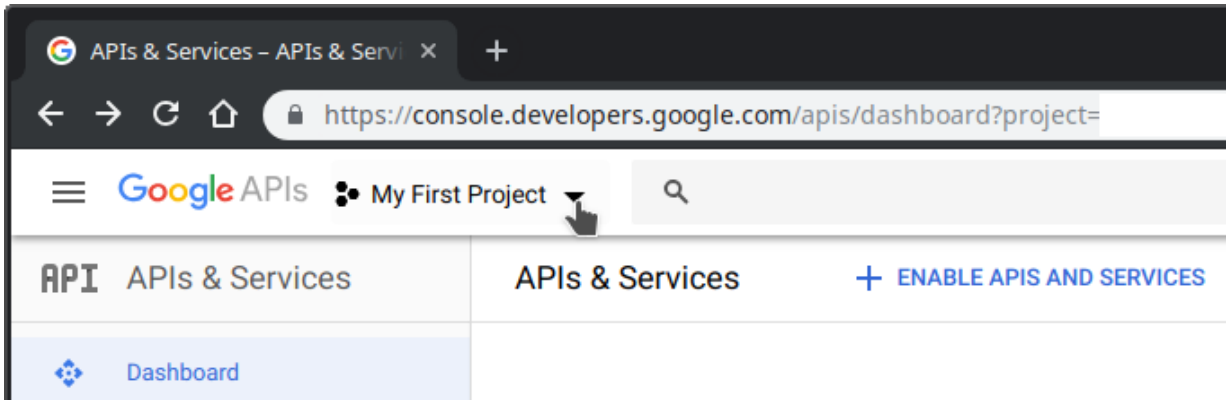


Enter a “Project public-facing name” and a “Project support email” similar to the values shown below. Click “Save” and the Firebase project configuration will be complete.

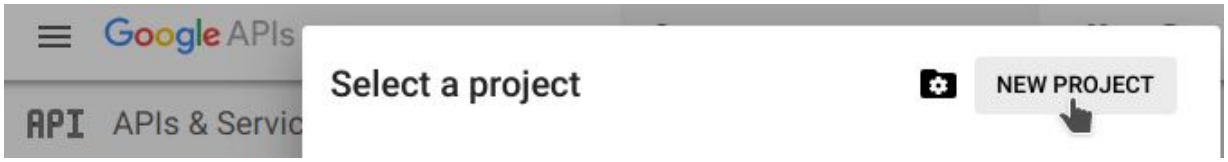


Configure Google Maps API Access

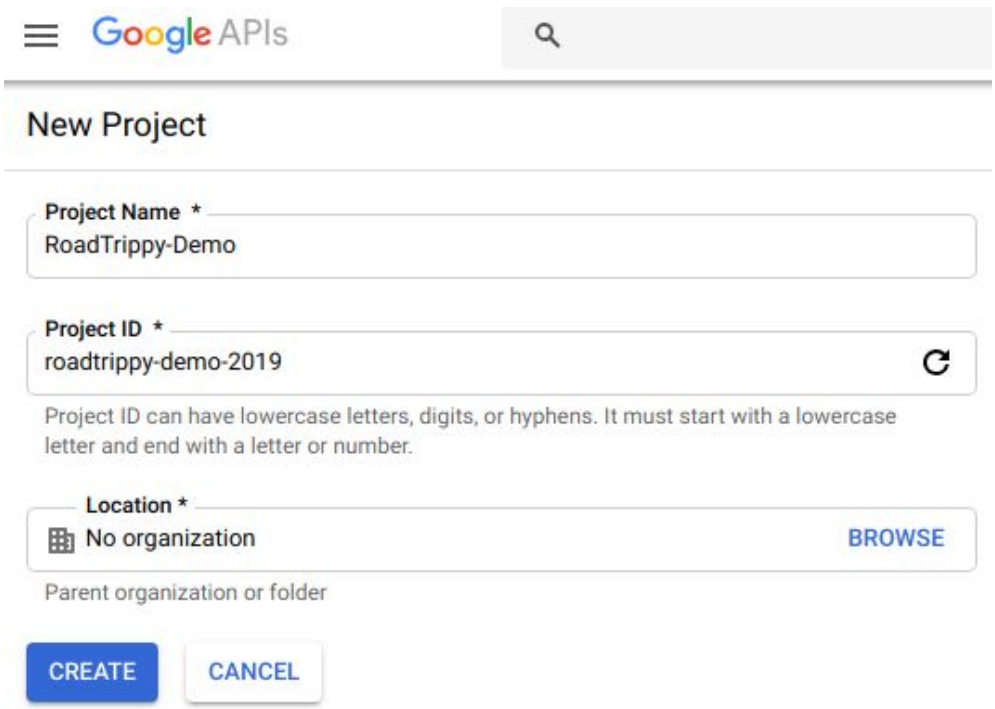
Access the Google Cloud Platform console at https://console.developers.google.com/apis/dashboard and click the drop down menu to select a project, and then create a new project.



Click “NEW PROJECT”

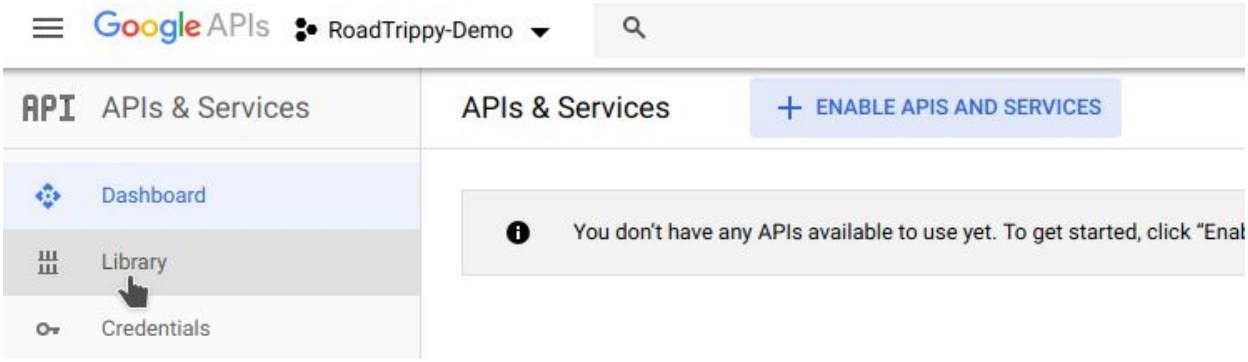


Set the required values. You may accept the defaults if you prefer.

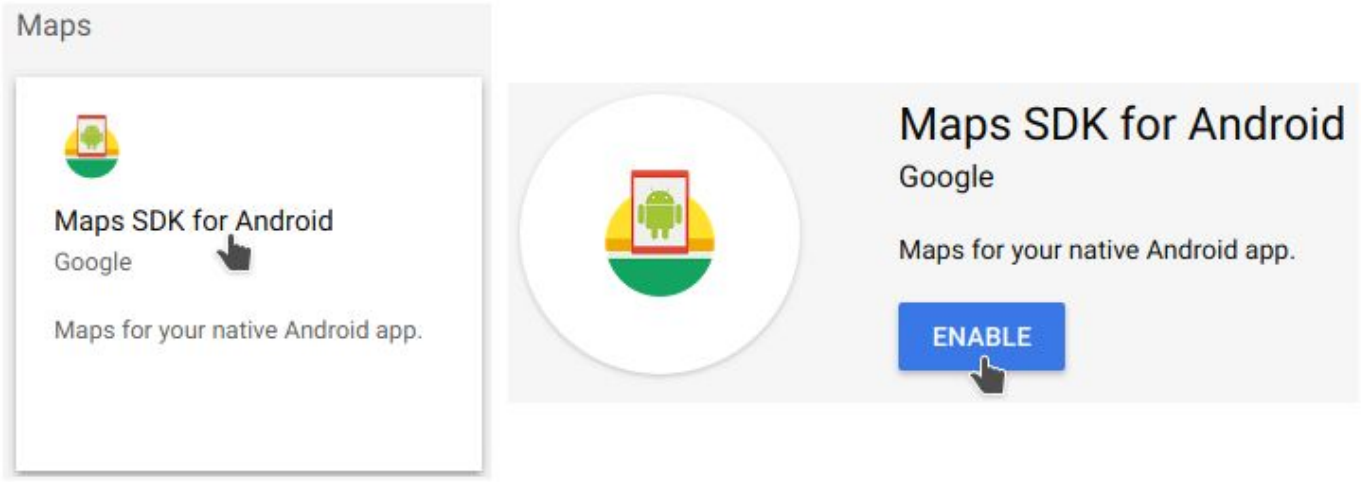


Click “CREATE” to initialize the project.

Click the “Library” option in the left side navigation.

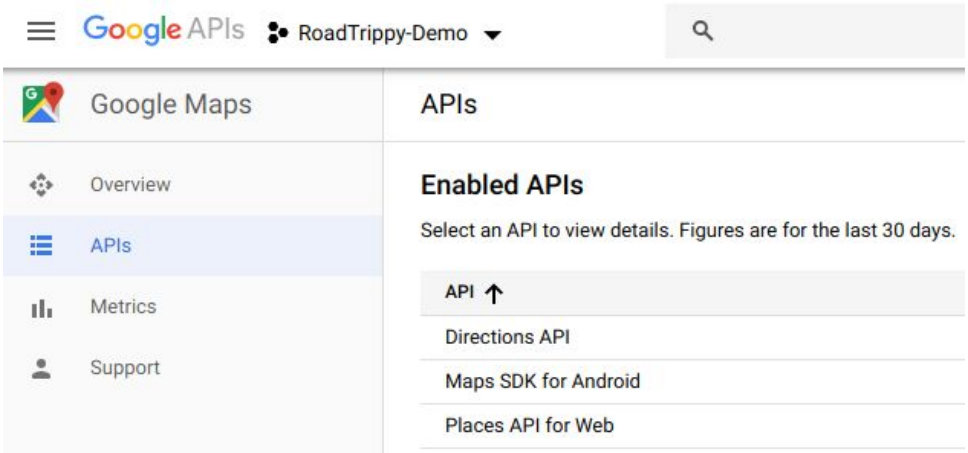


Select “Maps SDK for Android” and then click “ENABLE” on the subsequent screen.



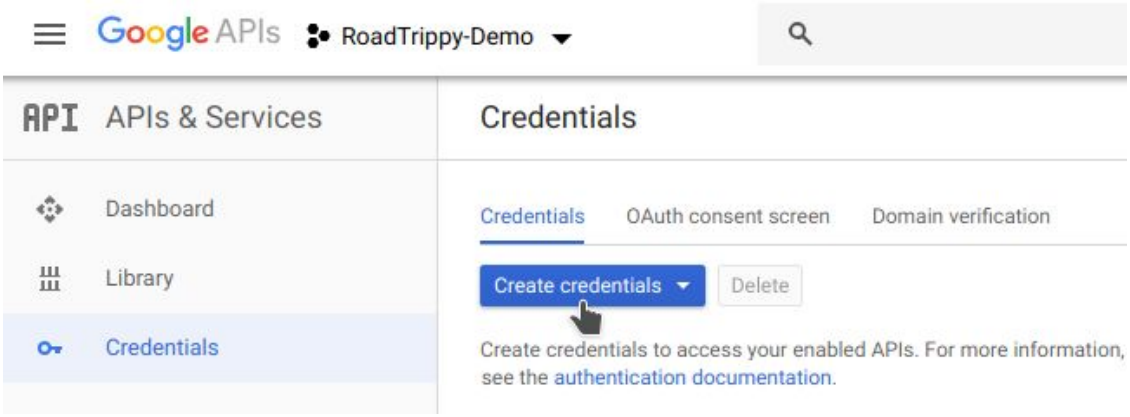
From the APIs navigation option, ensure that you enable each of the following APIs:

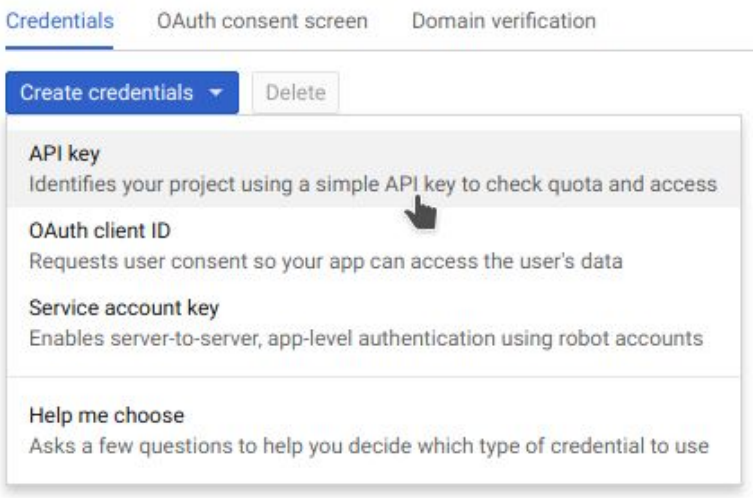
- Directions API
- Maps SDK for Android (enabled in the previous step)
- Places API for Web



API Key Credential

Create a Credential to allow the Android app to access this API project. You will save the generated API Key value in the `/app/res/values/google_maps_api.xml` file in the project source code.





After creating the API Key, copy the value of the key to the `/app/res/values/google_maps_api.xml` file in the project source code, replacing the string value for the item shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="google_maps_key" translatable="false" templateMergeStrategy="preserve">
       your-api-key-here
   </string>
</resources>
```

Configuration of the APIs required by the app is now complete.

The Android app is now functional. You can build the Android app using Android Studio, and deploy it to an emulator or physical device.