Unit-VI Security and Application Deployment

Course Outcome:

Publish Android applications.

Unit Outcomes:

6a. Explain the given location based service.
6b. Write the steps to customize the given permissions for users.
6c. Explain features of the given android security service.
6d. Write the steps to publish the given android App.

Contents:

6.1 SMS Telephony
6.2 Location Based Services: Creating the project, Getting the maps API key, Displaying the map, Displaying the zoom control, Navigating to a specific location, Adding markers, Getting location, Geocoding and reverse Geocoding, Getting Location data, Monitoring Location.
6.3 Android Security Model, Declaring and Using Permissions, Using Custom Permission.
6.4 Application Deployment: Creating Small Application, Signing of application, Deploying app on Google Play Store, Become a Publisher, Developer Console

---

6.1 SMS Telephony

In android, we can send SMS from our android application in two ways either by using SMSManager API or Intents based on our requirements.

If we use SMSManager API, it will directly send SMS from our application. In case if we use Intent with proper action (ACTION_VIEW), it will invoke a built-in SMS app to send SMS from our application.

Classes of SMS Telephony

1. Telephony.Sms.Conversations
   Contains a view of SMS conversations (also referred to as threads).

2. Telephony.Sms.Draft
   Contains all draft text-based SMS messages in the SMS app.

3. Telephony.Sms.Inbox
   Contains all text-based SMS messages in the SMS app inbox.

4. Telephony.Sms.Intents
   Contains constants for SMS related Intents that are broadcast.

5. Telephony.Sms.Outbox
   Contains all pending outgoing text-based SMS messages.

6. Telephony.Sms.Sent
   Contains all sent text-based SMS messages in the SMS app.

Android Send SMS using SMSManager API

In android, to send SMS using SMSManager API we need to write the code like as shown below.

```
SmsManager smgr = SmsManager.getDefault();

smgr.sendTextMessage(MobileNumber,null,Message,null,null);
```

SMSManager API required SEND_SMS permission in our android manifest to send SMS. Following is the code snippet to set SEND_SMS permissions in manifest file.

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Android Send SMS using Intent

In android, Intent is a messaging object which is used to request an action from another app component such as activities, services, broadcast receivers, and content providers. To know more about an Intent object in android check this Android Intents with Examples.

To send SMS using the Intent object, we need to write the code like as shown below.

```
Intent sInt = new Intent(Intent.ACTION_VIEW);

sInt.putExtra("address", new String[]{txtMobile.getText().toString()});

sInt.putExtra("sms_body",txtMessage.getText().toString());

sInt.setType("vnd.android-dir/mms-sms");
```

Even for Intent, it required a SEND_SMS permission in our android manifest to send SMS. Following is the code snippet to set SEND_SMS permissions in manifest file.

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Now we will see how to send SMS in android application using SMSManager API with examples.

Android Send SMS Example

Create a new android application using android studio and give names as SendSMSExample. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:id="@+id/fstTxt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:layout_marginTop="150dp"
    android:text="Mobile No" />
  <EditText
    android:id="@+id/mblTxt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:ems="10"/>
  <TextView
    android:id="@+id/secTxt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Message"
    android:layout_marginLeft="100dp" />
  <EditText
```

```
    android:id="@+id/msgTxt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:ems="10" />
  <Button
    android:id="@+id/btnSend"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:text="Send SMS" />
</LinearLayout>
```

Now open our main activity file MainActivity.java from \src\main\java\com.tutlane.sendsmsexample path and write the code like as shown below

MainActivity.java

```
package com.tutlane.sendsmsexample;
import android.content.Intent;
import android.net.Uri;
import android.provider.Telephony;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity
{
   private EditText txtMobile;
   private EditText txtMessage;
   private Button btnSms;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtMobile = (EditText)findViewById(R.id.mblTxt);
      txtMessage = (EditText)findViewById(R.id.msgTxt);
      btnSms = (Button)findViewById(R.id.btnSend);
      btnSms.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            try{
               SmsManager smgr = SmsManager.getDefault();

smgr.sendTextMessage(txtMobile.getText().toString(),null,txtMessage.getText().toString(
),null,null);
               Toast.makeText(MainActivity.this, "SMS Sent Successfully",
Toast.LENGTH_SHORT).show();
            }
            catch (Exception e){
               Toast.makeText(MainActivity.this, "SMS Failed to Send, Please try again",
Toast.LENGTH_SHORT).show();
            }
         }
      });
```

```
    }
}
```

If you observe above code, we are sending SMS using SMSManager api on button click. As discussed, we need to add a SEND_SMS permission in our android manifest.

Now open android manifest file (AndroidManifest.xml) and write the code like as shown below

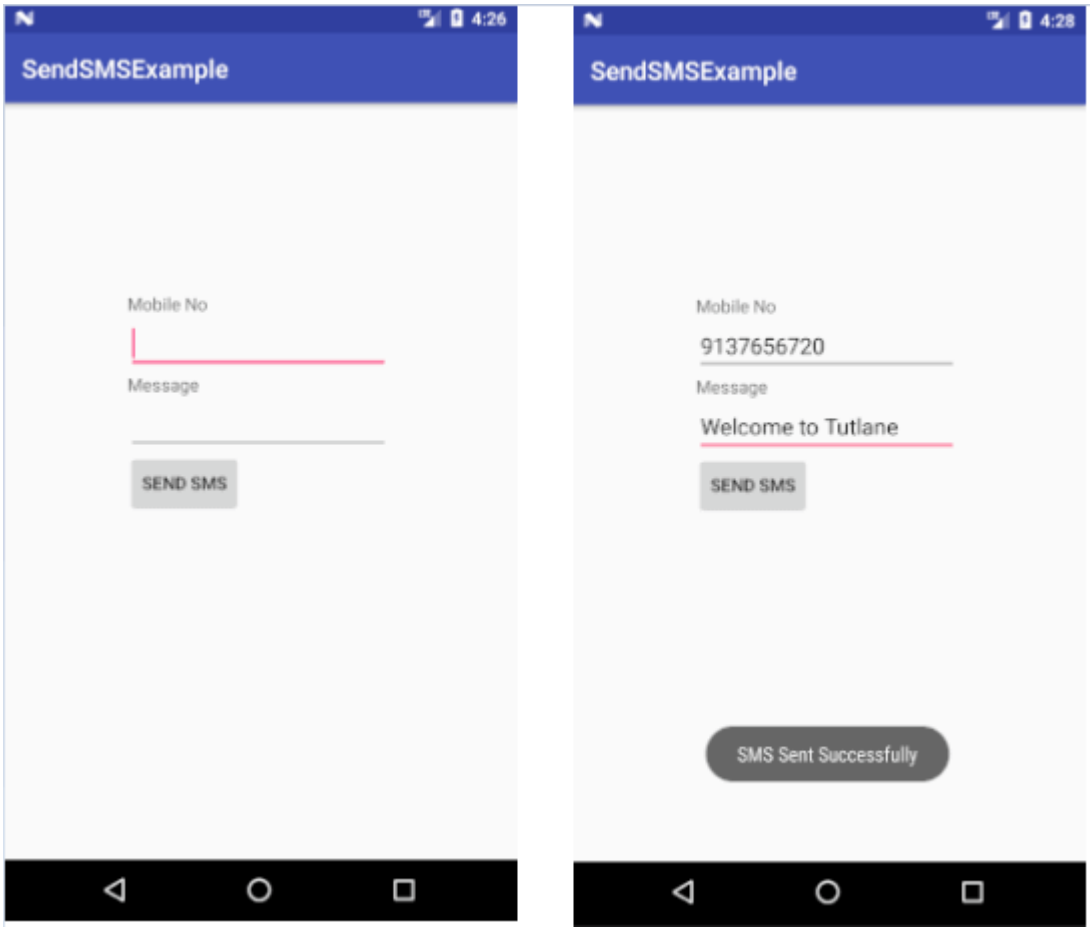
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tutlane.sendsmsexample">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

If you observe above AndroidManifest.xml file, we added a SEND_SMS permissions in manifest file.

Output of Android Send SMS Example

When we run above program in android studio we will get the result like as shown below.



Once you enter all details and click on Send SMS button it will send SMS and show the alert message like as mentioned in above image. The above example we implemented using SMSManager API. In case if we want to use Intents to send SMS to replace button click code like as shown below.

```
btnSms.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      try{
         Intent i = new Intent(Intent.ACTION_VIEW);
         i.setData(Uri.parse("smsto:"));
         i.setType("vnd.android-dir/mms-sms");
```

```
        i.putExtra("address", new String(txtMobile.getText().toString()));
        i.putExtra("sms_body",txtMessage.getText().toString());
        startActivity(Intent.createChooser(i, "Send sms via:"));
      }
      catch(Exception e){
        Toast.makeText(MainActivity.this, "SMS Failed to Send, Please try again", Toast.LENGTH_SHORT).show();
      }
    }
});
```

This is how we can send SMS using either SMSManager API or Intent objects in android applications based on our requirements.

6.2 Location Based Services

Location Based Services are provided by Android through its location framework. The framework provides a location API which consists of certain classes and interface. These classes and interface are the key components which allow us to develop Location Based Application in Android.

Classes and Interfaces of Location Based Services:

LocationManager – This class helps to get access to the location service of the system.
LocationListener – This interface acts as the listener which receives notification from the location manager when the location changes or the location provider is disabled or enabled.
Location – This is the class which represents the geographic location returned at a particular time.

Android Google Map

Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc. We can also customize Google map according to our requirement.

Types of Google Maps

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

1. **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
2. **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
3. **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
4. **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
5. **None:** This type displays an empty grid with no tiles loaded.

Syntax of different types of map

1. googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
2. googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
3. googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
4. googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

Methods of Google map

Google map API provides several methods that help to customize Google map. These methods are as following:

Methods	Description
addCircle(CircleOptions options)	This method add circle to map.
addPolygon(PolygonOptions options)	This method add polygon to map.
addTileOverlay(TileOverlayOptions options)	This method add tile overlay to the map.
animateCamera(CameraUpdate update)	This method moves the map according to the update with an animation.
clear()	This method removes everything from the map.
getMyLocation()	This method returns the currently displayed user location.
moveCamera(CameraUpdate update)	This method reposition the camera according to the instructions defined in the update.
setTrafficEnabled(boolean enabled)	This method set the traffic layer on or off.
snapshot(GoogleMap.SnapshotReadyCallback callback)	This method takes a snapshot of the map.
stopAnimation()	This method stops the camera animation if there is any progress.

Google Map - Layout file

Now you have to add the map fragment into xml layout file. Its syntax is given below –

```
<fragment
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.MapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"/>
```

Google Map - AndroidManifest file

The next thing you need to do is to add some permissions along with the Google Map API key in the AndroidManifest.XML file. Its syntax is given below –

```
<!--Permissions-->

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.
  READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!--Google MAP API key-->

<meta-data
  android:name="com.google.android.maps.v2.API_KEY"
  android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />
```

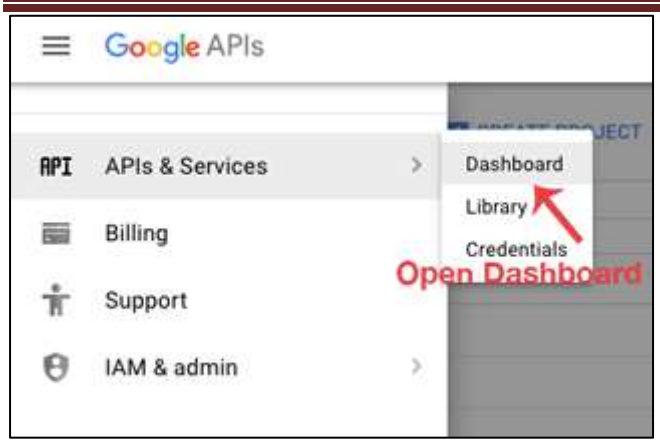
Getting the maps API key

An API key is needed to access the Google Maps servers. This key is free and you can use it with any of your applications. If you haven't created project, you can follow the below steps to get started:

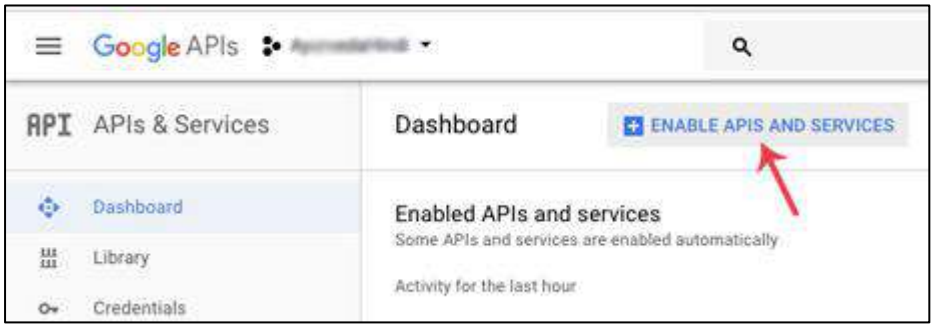
Step 1: Open Google developer console and signin with your gmail account: https://console.developers.google.com/project

Step 2: Now create new project. You can create new project by clicking on the **Create Project** button and give name to your project.

Step 3: Now click on APIs & Services and open **Dashboard** from it.

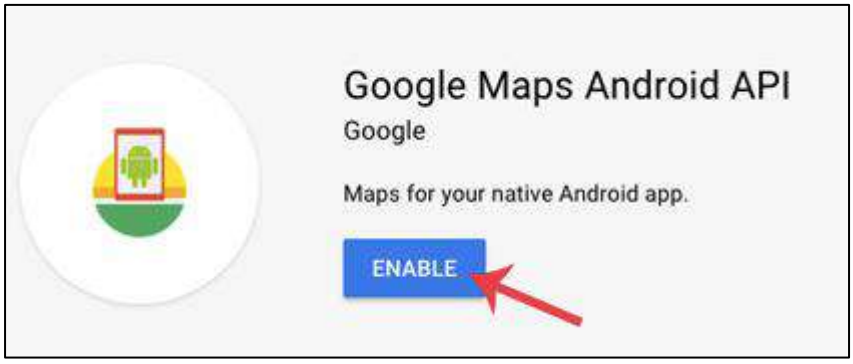


Step 4: In this open **Enable APIS AND SERICES**.

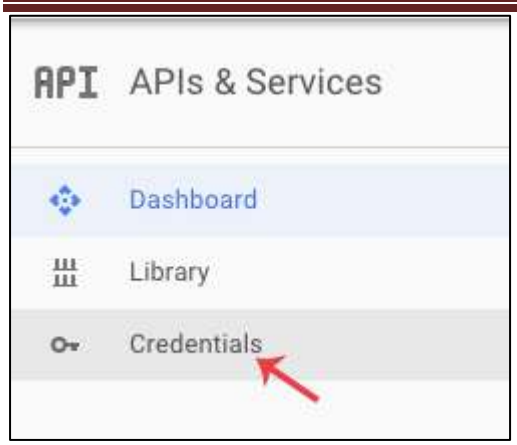


Step 5: Now open Google Map Android API.

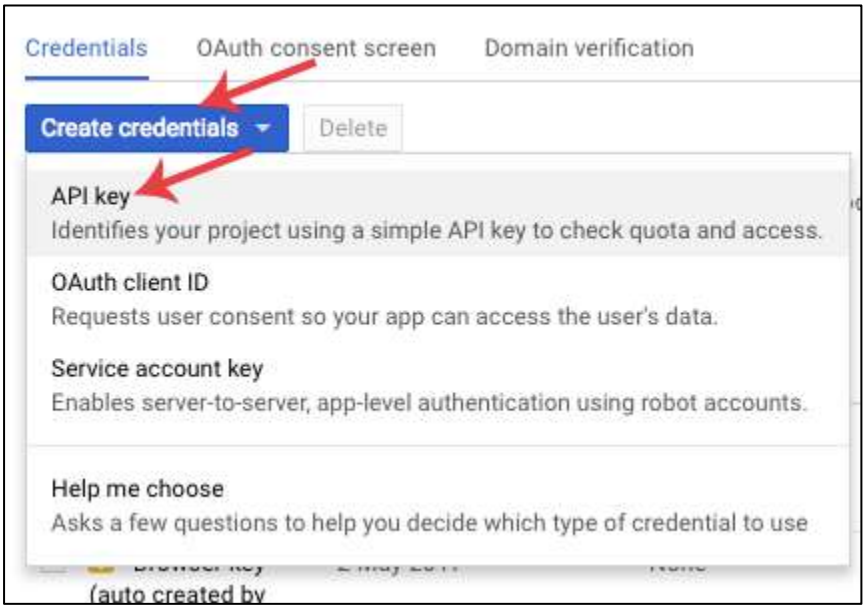
Step 6: Now enable the Google Maps Android API.



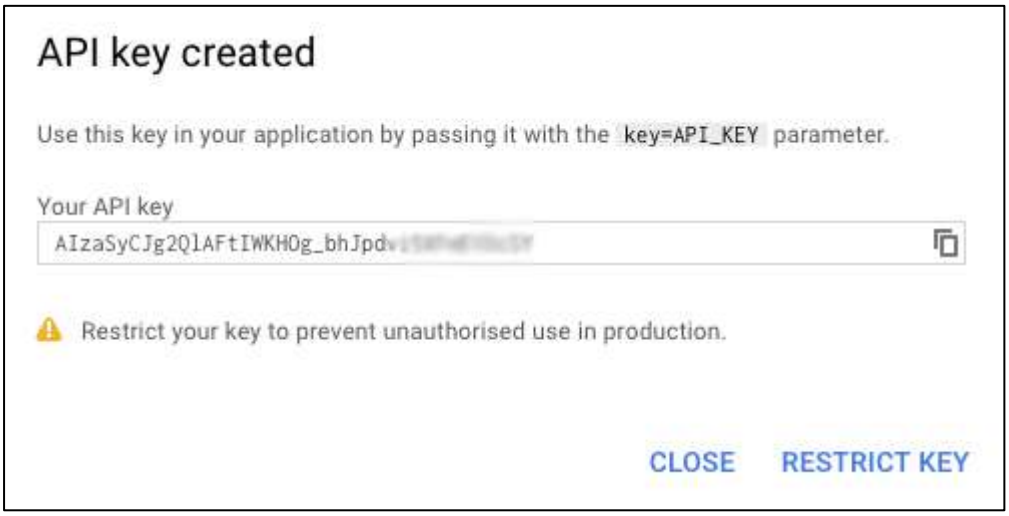
Step 6: Now go to **Credentials**



Step 7: Here click on Create credentials and choose API key



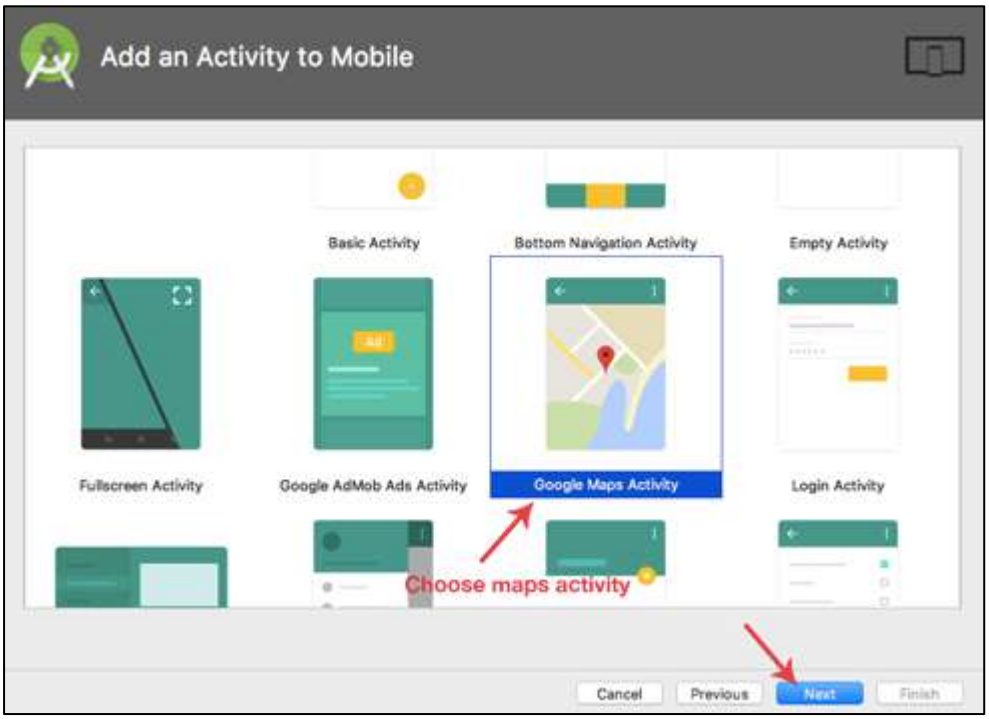
Step 8: Now API your API key will be generated. Copy it and save it somewhere as we will need it when implementing Google Map in our Android project.



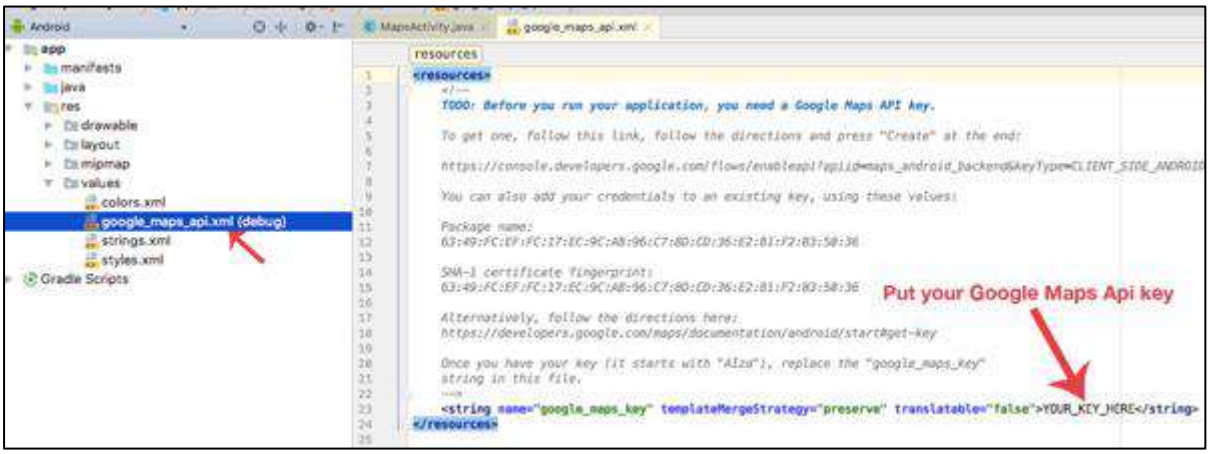
Creating the project, displaying the map, displaying the zoom control

Step 1: Create a New Android Project and name it GoogleMaps.

Step 2: Now select Google Maps Activity and then click Next and finish.



Step 3: Now open **google_maps_api.xml** (debug) in values folder



Step 4: Here enter your Google Maps API key in place of YOUR_KEY_HERE.

Step 5: Now open build.gradle and add compile 'com.google.android.gms:play-services:8.4.0' in dependencies

build.gradle code

```
apply plugin: 'com.android.application'

android {
   compileSdkVersion 26
   buildToolsVersion "26.0.2"
   defaultConfig {
```

```
        applicationId "com.abhiandroid.GoogleMaps.googlemaps"
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:26.+'
    compile 'com.google.android.gms:play-services:8.4.0'
    testCompile 'junit:junit:4.12'
}
```

Step 6: Now open activity_maps.xml and add a fragment code in it

Here add a fragment element to the activity's layout file to define a Fragment object. In this element, set the android:name attribute to "com.google.android.gms.maps.MapFragment". This automatically attaches a MapFragment to the activity. The following layout file contains a fragment element:

activity_maps.xml code

```
<fragment android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  xmlns:map="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="com.abhiandroid.GoogleMaps.googlemaps.MapsActivity"/>
```

Step 6: Now define internet and location permissions in Android Manifest

INTERNET – To determine if we are connected to Internet or not. **ACCESS_FINE_LOCATION** – To determine user's location using GPS. It will give us precise location.

AndroidManifest.xml code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="com.abhiandroid.GoogleMaps.googlemaps"
  xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="@string/google_maps_key"/>
    <activity
      android:name="com.abhiandroid.GoogleMaps.googlemaps.MapsActivity"
      android:label="@string/title_activity_maps">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
```

```
</manifest>
```

Step 7: Now we will code MapsActivity.java file for inserting callbacks in Google Maps:

-OnMapReadyCallback: This callback is called when the map is ready to be used

```
@Override
public void onMapReady(GoogleMap googleMap) {}
```

-GoogleApiClient.ConnectionCallbacks: This callback is called whenever device is connected and disconnected and implement onConnected() and onConnectionSuspended() functions.

```
//When the connect request has successfully completed
@Override
public void onConnected(Bundle bundle) {}
//Called when the client is temporarily in a disconnected state.
@Override
public void onConnectionSuspended(int i) {
}
```

-GoogleApiClient.OnConnectionFailedListener: Provides callbacks for scenarios that result in a failed attempt to connect the client to the service. Whenever connection is failed onConnectionFailed() will be called.

```
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
}
```

-LocationListener: This callback have function onLocationChanged() that will be called whenever there is change in location of device.

```
@Override
public void onLocationChanged(Location location) {}
```

-onMapReady(): This function is called when the map is ready to be used.

-buildGoogleApiClient(): This method is used to initialize the Google Play Services.

```
@Override
public void onMapReady(GoogleMap googleMap) {
   mMap = googleMap;
   mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

```
    mMap.getUiSettings().setZoomControlsEnabled(true);
    mMap.getUiSettings().setZoomGesturesEnabled(true);
    mMap.getUiSettings().setCompassEnabled(true);

    //Initialize Google Play Services
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            buildGoogleApiClient();
            mMap.setMyLocationEnabled(true);
        }
    } else {
        buildGoogleApiClient();
        mMap.setMyLocationEnabled(true);
    }
}
```

-addConnectionCallbacks(): You need to call registers a listener to receive connection events from this GoogleApiClient.

-addOnConnectionFailedListener(): This methods adds a listener to register to receive connection failed events from this GoogleApiClient.

-GoogleApiClient.Builder: Builder is used to help construct the GoogleApiClient object and addApi () specify which Apis are requested by your app.

-mGoogleApiClient.connect(): A client must be connected before executing any operation.

```
protected synchronized void buildGoogleApiClient() {
 mGoogleApiClient = new GoogleApiClient.Builder(this)
 .addConnectionCallbacks(this)
 .addOnConnectionFailedListener(this)
 .addApi(LocationServices.API)
 .build();
 mGoogleApiClient.connect();
}
```

-Zoom Controls: The Maps API provides built-in zoom controls that appear in the bottom right hand corner of the map. These can be enabled by calling:

```
mMap.getUiSettings().setZoomControlsEnabled(true);
```

-Zoom Gestures:

ZoomIn: Double tap to increase the zoom level by 1.

Zoom Out: Two finger tap to decrease the zoom level by 1.

```
mMap.getUiSettings().setZoomGesturesEnabled(true);
```

-Compass: You can set compass by calling below method:

```
mMap.getUiSettings().setCompassEnabled(true);
```

-Changing the Map Type:

The Android Maps API provides normal, satellite, terrain and hybrid map types to help you out:

```
mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

MAP_TYPE_NORMAL : Represents a typical road map with street names and labels.

MAP_TYPE_SATELLITE: Represents a Satellite View Area without street names and labels.

MAP_TYPE_TERRAIN: Topographic data. The map includes colors, contour lines and labels, and perspective shading. Some roads and labels are also visible.

MAP_TYPE_HYBRID : Combines a satellite View Area and Normal mode displaying satellite images of an area with all labels.

Map_TYPE_NONE : No tiles. It is similar to a normal map, but doesn't display any labels or coloration for the type of environment in an area.

Add the following inside setUpMap() just below the setMyLocationEnabled() call:

The location of the user is updated at the regular intervals. We have used FusedLocationProvider. We have used requestLocationUpdates() method to get regular updates about a device's location. Do this in the onConnected() callback provided by Google API Client, which is called when the client is ready.

LocationRequest mLocationRequest is used to get quality of service for location updates from the FusedLocationProviderApi using requestLocationUpdates.

```
@Override
```

```
public void onConnected(Bundle bundle) {
   mLocationRequest = new LocationRequest();
   mLocationRequest.setInterval(1000);
   mLocationRequest.setFastestInterval(1000);
   mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCUR
ACY);
   if (ContextCompat.checkSelfPermission(this,
         Manifest.permission.ACCESS_FINE_LOCATION)
         == PackageManager.PERMISSION_GRANTED) {
    LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
            mLocationRequest, this);
   }
}
```

Whenever user's location is changed. For that Google has predefined function onLocationChanged that will be called as soon as user's location change. Here we are getting the coordinates of current location using getLatitude() and getLongitude() and we are also adding Marker.

Complete code of MapsActivity.java class:

```
package com.abhiandroid.GoogleMaps.googlemaps;
import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Address;
import android.location.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationManager;
import android.os.Build;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.support.v4.content.ContextCompat;
import android.widget.Toast;
import com.google.android.gms.common.ConnectionResult;
```

```
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.abhiandroid.GoogleMaps.googlemaps.R;

import java.io.IOException;
import java.util.List;
import java.util.Locale;
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,
      GoogleApiClient.ConnectionCallbacks,
      GoogleApiClient.OnConnectionFailedListener,
      LocationListener {
   public static final int MY_PERMISSIONS_REQUEST_LOCATION = 99;
   GoogleApiClient mGoogleApiClient;
   Location mLastLocation;
   Marker mCurrLocationMarker;
   LocationRequest mLocationRequest;
   private GoogleMap mMap;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);

      if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
```

```
        checkLocationPermission();
    }
    SupportMapFragment mapFragment = (SupportMapFragment)
          getSupportFragmentManager()
                .findFragmentById(R.id.map);

    mapFragment.getMapAsync(this);
  }
  @Override
  public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    mMap.getUiSettings().setZoomControlsEnabled(true);
    mMap.getUiSettings().setZoomGesturesEnabled(true);
    mMap.getUiSettings().setCompassEnabled(true);
    //Initialize Google Play Services
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
      if (ContextCompat.checkSelfPermission(this,
          Manifest.permission.ACCESS_FINE_LOCATION)
          == PackageManager.PERMISSION_GRANTED) {
        buildGoogleApiClient();
        mMap.setMyLocationEnabled(true);
      }
    } else {
      buildGoogleApiClient();
      mMap.setMyLocationEnabled(true);
    }
  }
  protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
```

```
            .build();
        mGoogleApiClient.connect();
    }
    @Override
    public void onConnected(Bundle bundle) {
        mLocationRequest = new LocationRequest();
        mLocationRequest.setInterval(1000);
        mLocationRequest.setFastestInterval(1000);
        mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCU
RACY);
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
                    mLocationRequest, this);
        }
    }
    @Override
    public void onConnectionSuspended(int i) {
    }
    @Override
    public void onLocationChanged(Location location) {
        mLastLocation = location;
        if (mCurrLocationMarker != null) {
            mCurrLocationMarker.remove();
        }
//Showing Current Location Marker on Map
        LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(latLng);
        LocationManager locationManager = (LocationManager)
                getSystemService(Context.LOCATION_SERVICE);
        String provider = locationManager.getBestProvider(new Criteria(), true);
```

```
if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_
GRANTED &&
    ActivityCompat.checkSelfPermission(this,  Manifest.permission.ACCESS_COARSE_
LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
  return;
}
Location locations = locationManager.getLastKnownLocation(provider);
List<String> providerList = locationManager.getAllProviders();
if (null != locations && null != providerList && providerList.size() > 0) {
  double longitude = locations.getLongitude();
  double latitude = locations.getLatitude();
  Geocoder geocoder = new Geocoder(getApplicationContext(),
      Locale.getDefault());
  try {
    List<Address> listAddresses = geocoder.getFromLocation(latitude,
        longitude, 1);
    if (null != listAddresses && listAddresses.size() > 0) {
      String state = listAddresses.get(0).getAdminArea();
      String country = listAddresses.get(0).getCountryName();
      String subLocality = listAddresses.get(0).getSubLocality();
      markerOptions.title("" + latLng + "," + subLocality + "," + state
          + "," + country);
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFacto
ry.HUE_BLUE));
mCurrLocationMarker = mMap.addMarker(markerOptions);
mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
mMap.animateCamera(CameraUpdateFactory.zoomTo(11));
```

```
    if (mGoogleApiClient != null) {
        LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient,
                this);
    }
}
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
}
public boolean checkLocationPermission() {
    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {

        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.ACCESS_FINE_LOCATION)) {
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION);
        } else {
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION);
        }
        return false;
    } else {
        return true;
    }
}
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_LOCATION: {
```

```
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
              if (ContextCompat.checkSelfPermission(this,
                  Manifest.permission.ACCESS_FINE_LOCATION)
                  == PackageManager.PERMISSION_GRANTED) {
                if (mGoogleApiClient == null) {
                  buildGoogleApiClient();
                }
                mMap.setMyLocationEnabled(true);
              }
            } else {
              Toast.makeText(this, "permission denied",
                  Toast.LENGTH_LONG).show();
            }
            return;
          }
        }
      }
    }
```

Output:

Now run the App. If you are connected to internet and provide access to your location then in Map you will see your current location.

Navigating to a specific location,

Navigating to a destination is done using a NavController, an object that manages app navigation within a NavHost. Each NavHost has its own corresponding NavController. NavController provides a few different ways to navigate to a destination, which are further described in the sections below.

To retrieve the NavController for a fragment, activity, or view, use one of the following methods:

- NavHostFragment.findNavController(Fragment)
- Navigation.findNavController(Activity, @IdRes int viewId)
- Navigation.findNavController(View)

After you've retrieved a NavController, you can call one of the overloads of navigate() to navigate between destinations. Each overload provides support for various navigation scenarios, as described in the following sections.

Adding markers,

Customizing Google Map

You can easily customize google map from its default view, and change it according to your demand.

Adding Marker

You can place a maker with some text over it displaying your location on the map. It can be done by via **addMarker()** method. Its syntax is given below –

```
final LatLng TutorialsPoint = new LatLng(21 , 57);
Marker TP = googleMap.addMarker(new MarkerOptions()
  .position(TutorialsPoint).title("TutorialsPoint"));
```

Changing Map Type

You can also change the type of the MAP. There are four different types of map and each give a different view of the map. These types are Normal,Hybrid,Satellite and terrain. You can use them as below

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Enable/Disable zoom

You can also enable or disable the zoom gestures in the map by calling the **setZoomControlsEnabled(boolean)** method. Its syntax is given below –

```
googleMap.getUiSettings().setZoomGesturesEnabled(true);
```

Apart from these customization, there are other methods available in the GoogleMap class , that helps you more customize the map. They are listed below –

Sr.No	Method & description
1	**addCircle(CircleOptions options)** This method add a circle to the map
2	**addPolygon(PolygonOptions options)** This method add a polygon to the map
3	**addTileOverlay(TileOverlayOptions options)** This method add tile overlay to the map
4	**animateCamera(CameraUpdate update)** This method Moves the map according to the update with an animation
5	**clear()** This method removes everything from the map.

6	**getMyLocation()** This method returns the currently displayed user location.
7	**moveCamera(CameraUpdate update)** This method repositions the camera according to the instructions defined in the update
8	**setTrafficEnabled(boolean enabled)** This method Toggles the traffic layer on or off.
9	**snapshot(GoogleMap.SnapshotReadyCallback callback)** This method Takes a snapshot of the map
10	**stopAnimation()** This method stops the camera animation if there is one in progress

Getting location

Appropriate use of location information can be beneficial to users of your app. For example, if your app helps the user find their way while walking or driving, or if your app tracks the location of assets, it needs to get the location of the device at regular intervals. As well as the geographical location (latitude and longitude), you may want to give the user further information such as the bearing (horizontal direction of travel), altitude, or velocity of the device. This information, and more, is available in the Location object that your app can retrieve from the fused location provider. In response, the API updates your app periodically with the best available location, based on the currently-available location providers such as WiFi and GPS (Global Positioning System). The accuracy of the location is determined by the providers, the location permissions you've requested, and the options you set in the location request.

Declare permissions

Apps that use location services must request location permissions. In most cases, you can request the coarse location permission and still get reasonably accurate location information from the available location providers.

The following snippet demonstrates how to request the coarse location permission:

```
<manifest ... >
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
</manifest>
```

Geocoding and reverse Geocoding

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

Geocoding is the process of finding the geographical coordinates (latitude and longitude) of a given address or location. Reverse Geocoding is the opposite of geocoding where a pair of latitude and longitude is converted into an address or location.

For achieving Geocode or Reverse Geocode you must first import the proper package.

```
import android.location.Geocoder;
```

The geocoding or reverse geocoding operation needs to be done on a separate thread and should never be used on the UI thread as it will cause the system to display an Application Not Responding (ANR) dialog to the user.

To Achieve Geocode, use the below code

```
Geocoder gc = new Geocoder(context);
if(gc.isPresent()){
List<Address> list = gc.getFromLocationName("155 Park Theater, Palo Alto, CA", 1);
Address address = list.get(0);
double lat = address.getLatitude();
double lng = address.getLongitude();
}
```

To Achieve Reverse Geocode, use the below code

```
Geocoder gc = new Geocoder(context);
if(gc.isPresent()){
List<address> list = gc.getFromLocation(37.42279, -122.08506,1);
Address address = list.get(0);
StringBuffer str = new StringBuffer();
str.append("Name: " + address.getLocality() + "\n");
str.append("Sub-Admin Ares: " + address.getSubAdminArea() + "\n");
str.append("Admin Area: " + address.getAdminArea() + "\n");
str.append("Country: " + address.getCountryName() + "\n");
str.append("Country Code: " + address.getCountryCode() + "\n");
String strAddress = str.toString();
}
```

Getting Location data

There are two types of location providers,

1. GPS Location Provider
2. Network Location Provider

Any one of the above providers is enough to get current location of the user or user's device. But, it is recommended to use both providers as they both have different advantages. Because, GPS provider will take time to get location at indoor area. And, the Network Location Provider will not get location when the network connectivity is poor.

Network Location Provider vs GPS Location Provider

- Network Location provider is comparatively faster than the GPS provider in providing the location co-ordinates.
- GPS provider may be very very slow in in-door locations and will drain the mobile battery.
- Network location provider depends on the cell tower and will return our nearest tower location.
- GPS location provider, will give our location accurately.

Steps to get location in Android

1. Provide permissions in manifest file for receiving location update
2. Create LocationManager instance as reference to the location service
3. Request location from LocationManager
4. Receive location update from LocationListener on change of location

Provide permissions for receiving location update

To access current location information through location providers, we need to set permissions with android manifest file.

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

ACCESS_COARSE_LOCATION is used when we use network location provider for our Android app. But, ACCESS_FINE_LOCATION is providing permission for both providers. INTERNET permission is must for the use of network provider.

Create LocationManager instance as reference to the location service

For any background Android Service, we need to get reference for using it. Similarly, location service reference will be created using getSystemService() method. This reference will be added with the newly created LocationManager instance as follows.

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

Request current location from LocationManager

After creating the location service reference, location updates are requested using requestLocationUpdates() method of LocationManager. For this function, we need to send the type of location provider, number of seconds, distance and the LocationListener object over which the location to be updated.

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

Receive location update from LocationListener on change of location

LocationListener will be notified based on the distance interval specified or the number seconds.

Monitoring Location.

6.3 Android Security Model

The Android security model is primarily based on a sandbox and permission mechanism. Each application is running in a specific Dalvik virtual machine with a unique user ID assigned to it, which means the application code runs in isolation from the code of all others applications. As a consequence, one application has not granted access to other applications' files.

Android application has been signed with a certificate with a private key Know the owner of the application is unique. This allows the author of The application will be identified if needed. When an application is installed in The phone is assigned a user ID, thus avoiding it from affecting it Other applications by creating a sandbox for it. This user ID is permanent on which devices and applications with the same user ID are allowed to run in a single process. This is a way to ensure that a malicious application has Can not access / compromise the data of the genuine application.

It is mandatory for an application to list all the resources it will Access during installation. Terms are required of an application, in The installation process should be user-based or interactive Check with the signature of the application.

Declaring and Using Permissions

The purpose of a permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request. (2)

Permissions are divided into several protection levels. The protection level affects whether runtime permission requests are required. There are three protection levels that affect third-party apps: *normal*, *signature*, and *dangerous* permissions.

Normal permissions cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps. For example, permission to set the time zone is a normal permission. If an app declares in its manifest that it needs a normal permission, the system automatically grants the app that permission at install time. The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

Signature permissions: The system grants these app permissions at install time, but only when the app that attempts to use permission is signed by the same certificate as the app that defines the permission.

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user approves the permission, your app cannot provide functionality that depends on that permission. To use a dangerous permission, your app must prompt the user to grant permission at runtime. For more details about how the user is prompted, see Request prompt for dangerous permission.

Using Custom Permission

Apps can define their own custom permissions and request custom permissions from other apps by defining `<uses-permission>` elements. To enforce your own permissions, you must first declare them in your `AndroidManifest.xml` using one or more `<permission>` elements.

For example, an app that wants to control who can start one of its activities could declare a permission for this operation as follows:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp" >

    <permission
      android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
```

```
      android:description="@string/permdesc_deadlyActivity"
      android:permissionGroup="android.permission-group.COST_MONEY"
      android:protectionLevel="dangerous" />
    ...
</manifest>
```

The `protectionLevel` attribute is required, telling the system how the user is to be informed of apps requiring the permission, or who is allowed to hold that permission, as described in the linked documentation.

The `android:permissionGroup` attribute is optional, and only used to help the system display permissions to the user.

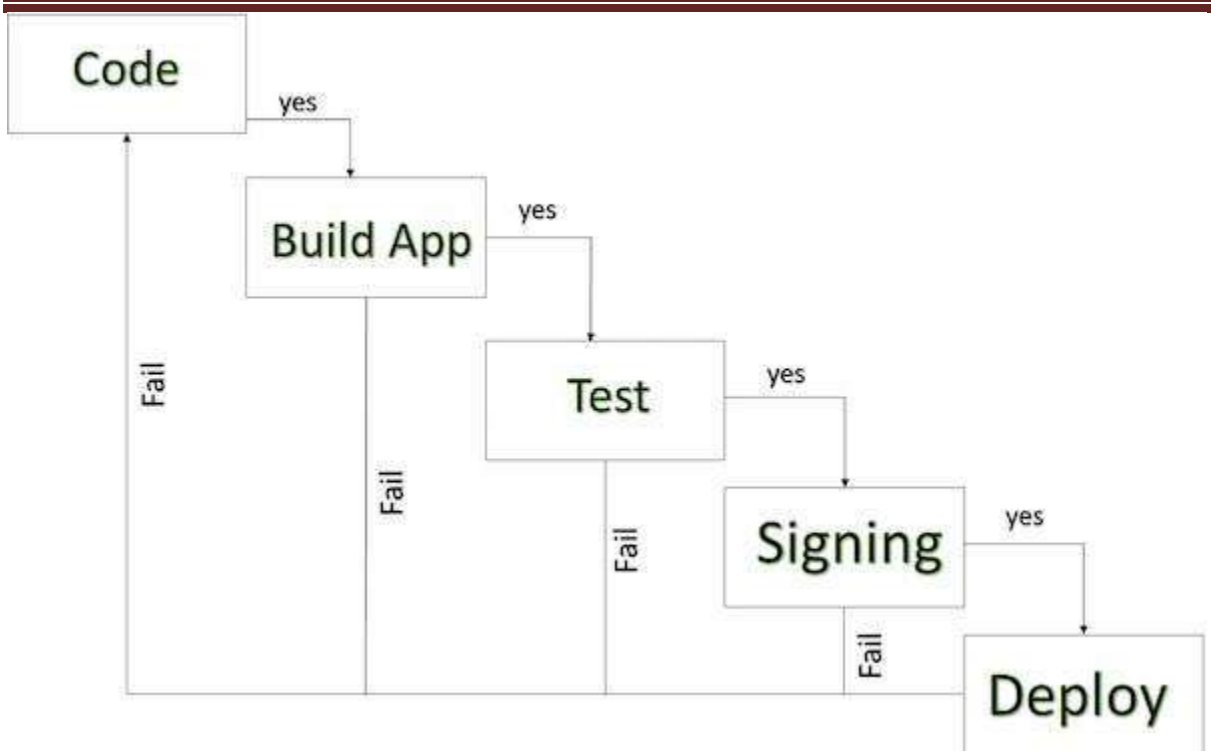
You need to supply both a label and description for the permission. These are string resources that the user can see when they are viewing a list of permissions (`android:label`) or details on a single permission (`android:description`). The label should be short; a few words describing the key piece of functionality the permission is protecting. The description should be a couple of sentences describing what the permission allows a holder to do. Our convention is a two-sentence description: the first sentence describes the permission, and the second sentence warns the user of the type of things that can go wrong if an app is granted the permission.

Here is an example of a label and description for the `CALL_PHONE` permission:

```
<string name="permlab_callPhone">directly call phone numbers</string>
<string name="permdesc_callPhone">Allows the app to call
    phone numbers without your intervention. Malicious apps may
    cause unexpected calls on your phone bill. Note that this does not
    allow the app to call emergency numbers.</string>
```

6.4 Application Deployment:

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



Android development life cycle

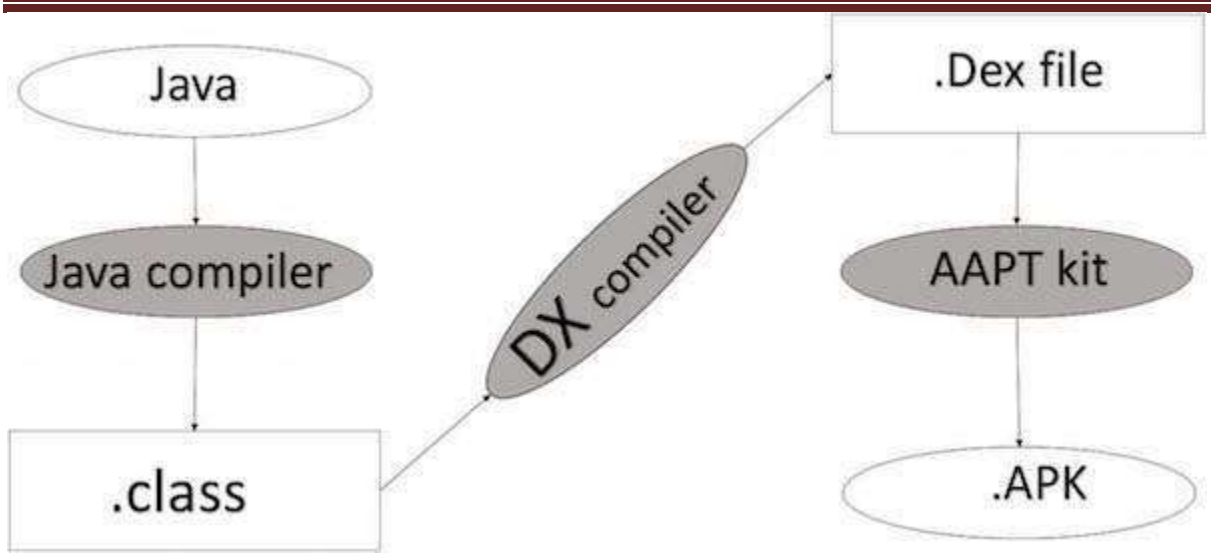
Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application –

Step	Activity
1	**Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	**Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	**Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone,

	localization or any other specific requirement as per the targeted region.
4	**Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.
5	**SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.
6	**Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.
7	**Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	**Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: Preparing for Release.
9	**Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

Export Android Application Process



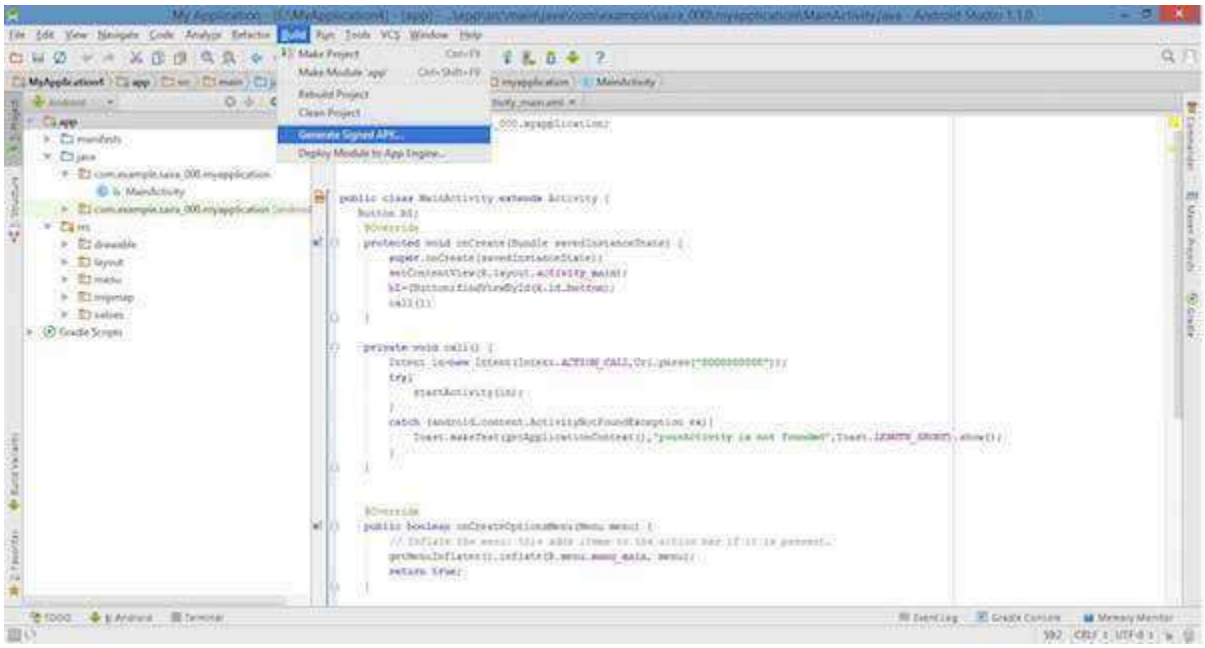
Apk development process

Before exporting the apps, you must some of tools

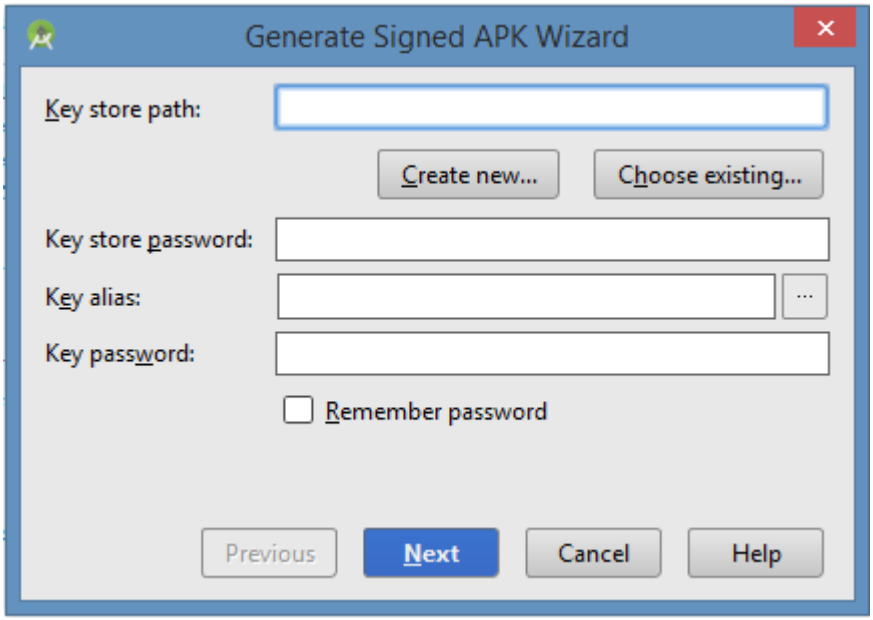
- **Dx tools**(Dalvik executable tools): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

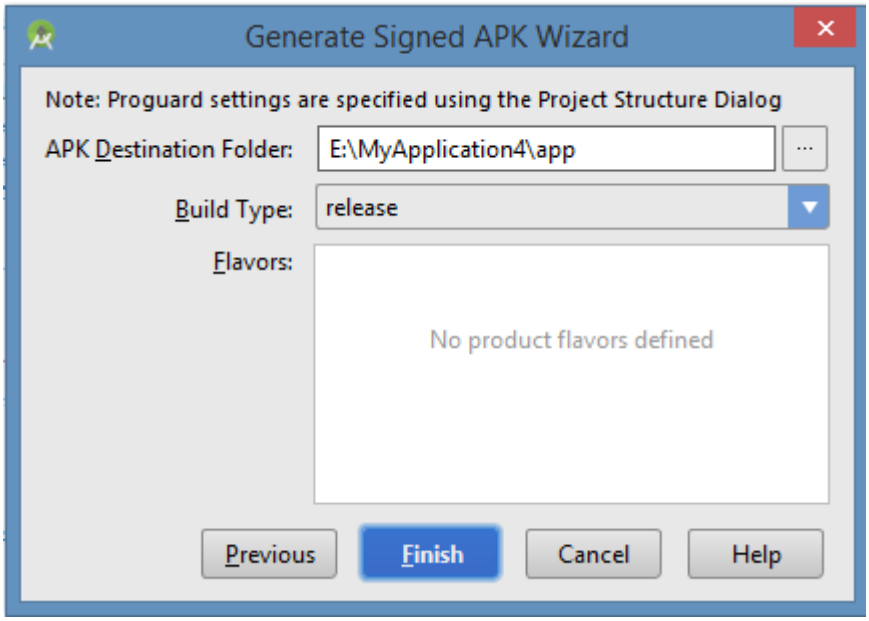
To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application –



Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.



Enter your key store path,key store password,key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application –

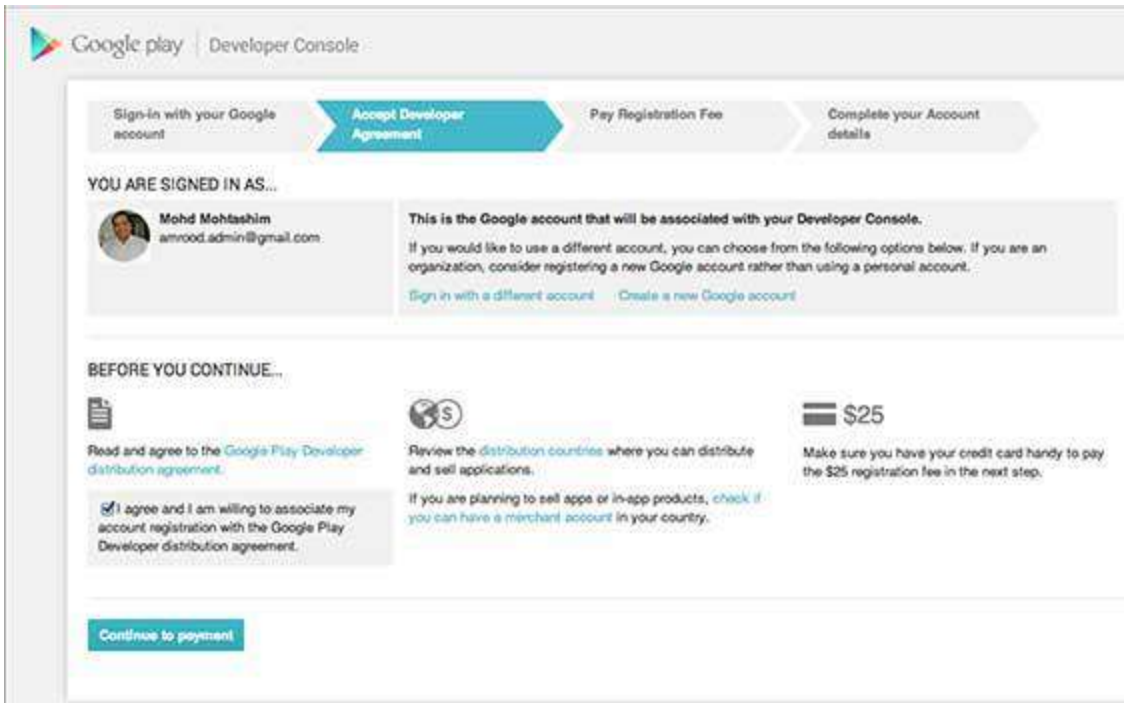


Once you filled up all the information,like app destination,build type and flavours click **finish** button While creating an application it will show as below

Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

Google Play Registration

The most important step is to register with Google Play using Google Play Marketplace. You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace. You will have following screen to accept terms and condition.



You can use **Continue to payment** button to proceed to make a payment of $25 as a registration fee and finally to complete your account detail.

Once you are a registered user at Google Play, you can upload **release-ready APK** for your application and finally you will complete application detail using application detail page as mentioned in step 9 of the above mentioned checklist.

Signing Your App Manually

You do not need Android Studio to sign your app. You can sign your app from the command line using standard tools from the Android SDK and the JDK. To sign an app in release mode from the command line –

- Generate a private key using keytool

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

- Compile your app in release mode to obtain an unsigned APK
- Sign your app with your private key using jarsigner

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore my-release-key.keystore my_application.apk alias_name
```

- Verify that your APK is signed. For example –

```
$ jarsigner -verify -verbose -certs my_application.apk
```

- Align the final APK package using zipalign.

```
$ zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

Steps steps to publish the Android application.

Step 1: Sign up. Sign up for an account on the **Android** Developer Console. ...

Step 2: Create a new **application**. ...

Step 3: Prepare multimedia. ...

Step 4: Prepare code for **release**. ...

Step 5: Build a **release**-ready APK. ...

Step 6: Upload APK. ...

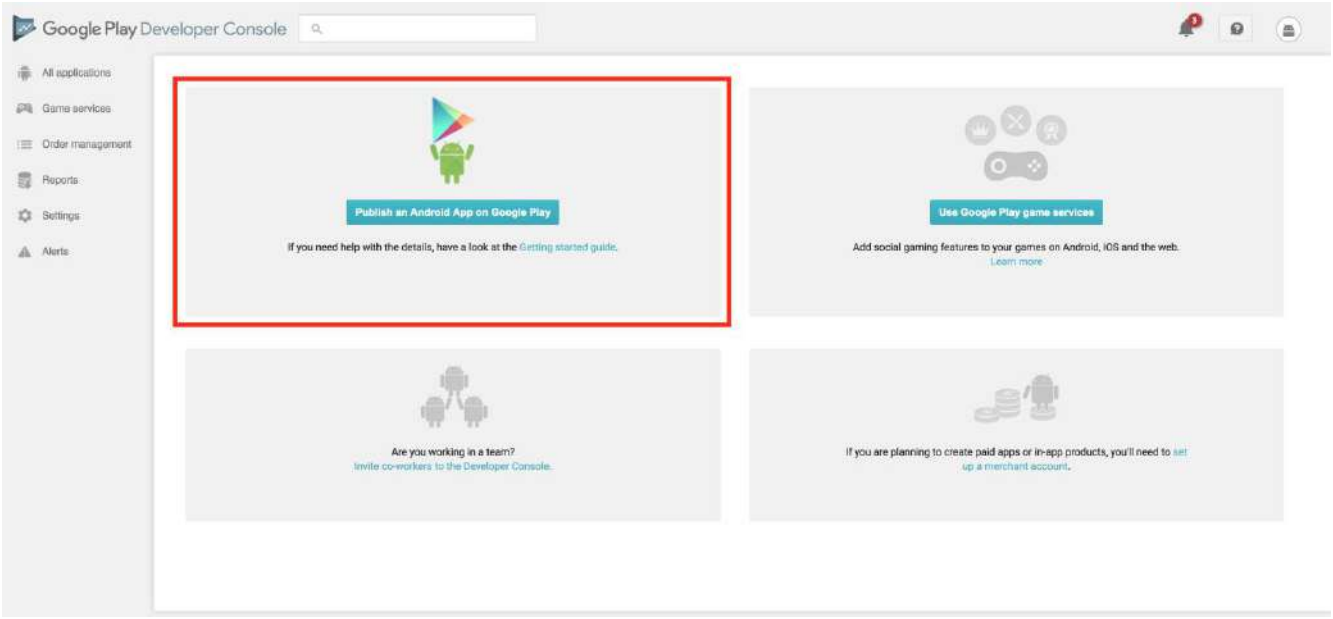
Complete the checklist on the left until all the items have a green checkmark.

Step 1: Sign up

Sign up for an account on the Android Developer Console. Creating an account costs $25.

Step 2: Create a new application

- On the Developer Console select the *Publish an Android Application* option.
- Fill out the details: Title, Short Description, Full Description.



Step 3: Prepare multimedia

- Screenshots: I used the android emulator to take screenshots of my app.
- Hi-res icon: I used the launcher icon. It was an SVG file, so I converted it to PNG using GIMP.

- Feature graphic: This is an image that shows up on the top of the app download page in Google Play on mobile phones.

Step 4: Prepare code for release

- Remove ***log*** statements.
- Remove the ***android:debuggable*** attribute from your manifest file. I didn't have to do this because Android Studio automatically sets this attribute based on the kind of APK its building. Neat!
- Set the ***android:versionCode*** attribute in the manifest tag in manifest.xml. Two important notes: (1) This must be an integer that increases with each release. (2) This number is not displayed to users.
  I chose "1".
- Set the ***android:versionName*** attribute in the manifest tag in manifest.xml. This string is shown to users and has no other purpose.
  I chose "1.0".

Step 5: Build a release-ready APK

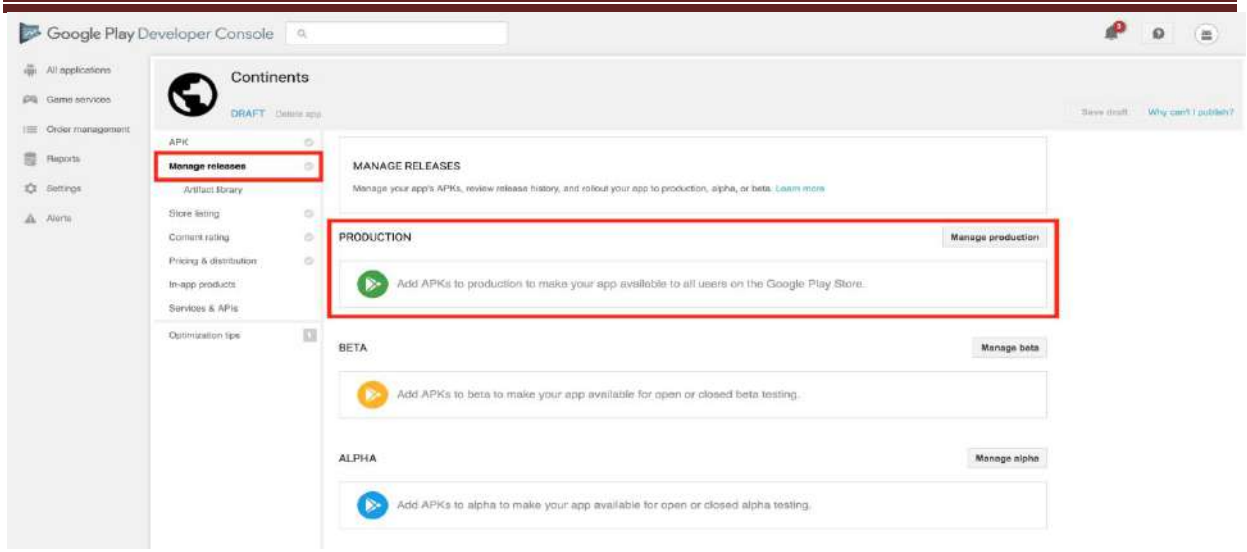
The release-ready APK is different from the debug APK in that it is signed with certificate that is owned by the developer. This is done to ensure that updates to the app come from a verified source, i.e. a developer with access to the private key.

I recommend you follow the instructions here to create a signed APK.

- Android Studio -> Build -> Generate Signed APK
- A Java Keystore (JKS) is a repository of public-private key pairs.
- You must sign all APKs with the same key pair.
- Losing a key-pair consequences that you will not be able to push updates to your app.

Step 6: Upload APK

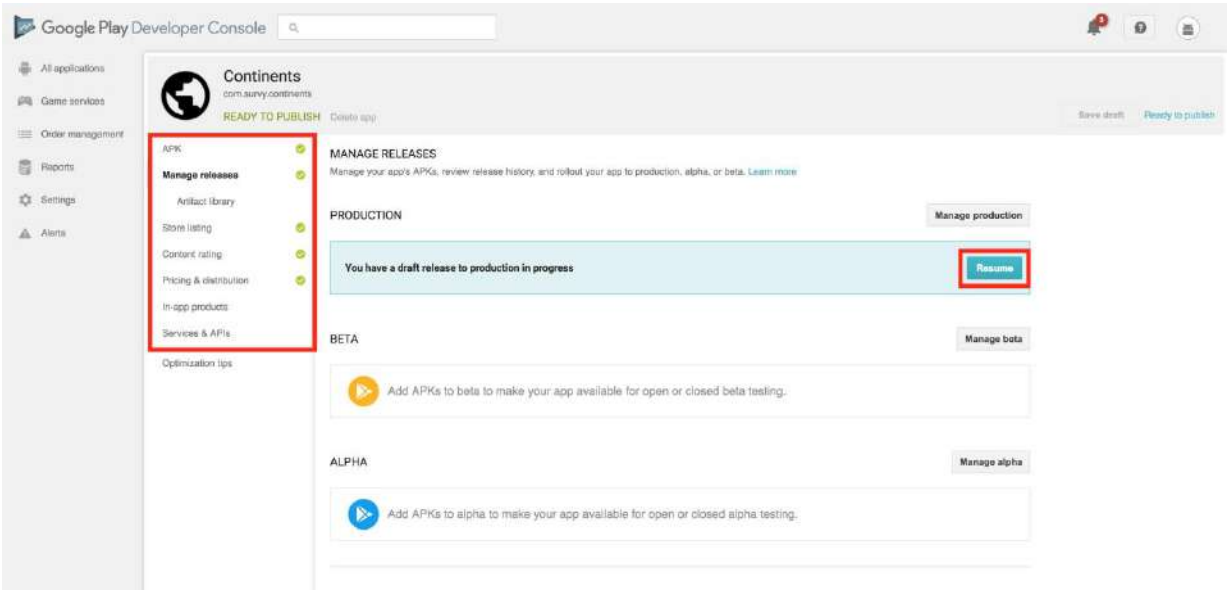
Go back to the Developer Console and click on *Manage Releases*. Then create a *Production Release* and upload your signed APK.



Google will perform a check on the APK. My app was using an SVG for the launcher icon, which is no-bueno. I had to change it to PNG and recreate the signed APK.

Step 7:

Complete the checklist on the left until all the items have a green checkmark. The console re-evaluates the checklist every time you click *Save Draft* in the top right.



You are now ready to publish

Developer Console

The Android Things Console provides easy and secure deployment of updates to your connected devices. Google provides the infrastructure to host and deliver system and app updates with the developer in final control.

Using the console, developers and device makers can:

- Download and install the latest Android Things system image
- Build factory images that contain OEM applications along with the system image
- Push over-the-air (OTA) seamless updates, including OEM applications and the system image, to devices
- Manage and share OEM applications across products and owners
- Monitor informative analytics to understand how well products are performing

Add developer account users & manage permissions

There are three different access levels on the Play Console: account owner, admins, and users. Your access type determines what actions you can take and what information you can view on the Play Console.

Account access levels

Type	Description
Account owner	• First registered the account on the Play Console • Has full access to the Play Console • Can add users, manage individual permissions, and remove user access • Is the only person who can have a linked payments profile to sell paid apps • Is the only person who can edit information on the Payments Settings page in the Play Console • Is the only person who can edit Developer Profile information in the Play Console
Admin	• Has the "Manage user permissions" permission • Can be given access to all or specific apps • Can add users, manage individual permissions, and remove user access
User	• Can have different levels of access to the Play Console • Can be given access to all or specific apps • Can't invite new users or edit user permissions • Doesn't need to pay the $25 registration fee

Give users access

Step 1: Decide whether your user needs global or per-app access

Before you set up permissions, you need to decide if your user needs global or per-app access.

- Global: Global access applies to all apps in your developer account.
- Per-app: Per-app access only applies to the selected app.

For details on how global and per-app access impacts a specific permission, select the permission below under "Permission definition & uses."

Step 2: Add users & turn permissions on or off

If you're an account owner or admin, you can add users to your Play Console account and manage permissions across all apps or for specific apps.

1. Sign in to your Play Console.
2. Click Settings > Users & permissions.
    - To add a user, select Invite new user and follow the onscreen instructions.
    - To update permissions for an existing user, hover over their email address and select the pencil icon .
    - Note: Users can only sign in to the Play Console with a Google account using the same email address that you invite.
3. Use the "Role" selector to choose a pre-defined role or use the checkboxes for individual permissions.
4. Choose whether each permission applies to all apps in your developer account ("Global") or specific apps.
    - To add an app to the permissions table, use the down arrow next to "Add an app."
    - To see details for each permission, review the permission definitions section.
5. Click Send Invitation.