UNIT – 5

❖ API in Android App development

In Android app development, an API, which stands for Application Programming Interface, serves as a set of tools, protocols, and definitions that allow different software applications to communicate with each other.

Essentially, it acts as a bridge, enabling your Android app to interact with external services, libraries, or platforms. APIs define the methods and data formats applications can use to request and exchange information.

Here are some common Android APIs categorized by their functionalities:

1. Data Storage & Management APIs

- **Internal Storage API** – Store private app data in the device's internal memory.
- **External Storage API** – Store data on an SD card or external storage.
- **SQLite API** – Manage local databases using SQLite.
- **Room Persistence Library** – A higher-level abstraction over SQLite for easier database management.
- **Shared Preferences API** – Store key-value pairs for lightweight data persistence.
- **Content Provider API** – Share data between apps securely.

2. Networking & Web APIs

- **Android Networking API** – Handle HTTP requests using HttpURLConnection or third-party libraries like Retrofit and Volley.
- **Android Web API** – Integrate web content within an app using WebView.
- **JSON Parsing API** – Parse JSON data using org.json or libraries like Gson and Moshi.
- **Firebase Realtime Database API** – Synchronize app data in real-time with Firebase.

3. Telephony & Messaging APIs

- **Android Telephony API** – Access phone-related features like call logs, IMEI, SIM card details, etc.
- **SMS & MMS API** – Send and receive SMS/MMS messages.
- **Call Management API** – Manage incoming/outgoing calls and detect call state.

4. Location & Mapping APIs

- **Google Maps API** – Embed maps and enable location-based services.
- **GPS & Location API** – Retrieve the user's location using GPS, Wi-Fi, or mobile networks.

- **Geofencing API** – Trigger actions when a device enters or leaves a specific geographical area.
- **Fused Location Provider API** – Efficiently retrieve the user's location with minimal battery drain.

5. Sensors & Connectivity APIs

- **Sensor API** – Access device sensors like accelerometer, gyroscope, magnetometer, etc.
- **Bluetooth API** – Enable Bluetooth communication between devices.
- **Wi-Fi API** – Manage Wi-Fi connections and scan available networks.
- **NFC API** – Enable Near Field Communication (NFC) for contactless data transfer.

6. Multimedia & Camera APIs

- **Camera API (CameraX)** – Capture photos and videos efficiently.
- **Media Player API** – Play audio and video files.
- **Media Recorder API** – Record audio and video.
- **ExoPlayer API** – A customizable media player for streaming media content.

7. UI & Animation APIs

- **RecyclerView API** – Efficiently display lists and grids of data.
- **ConstraintLayout API** – Create complex UI layouts with flexibility.
- **Material Design Components API** – Implement modern UI elements with Material Design.
- **Animation API** – Create smooth UI animations using ObjectAnimator, ViewPropertyAnimator, and MotionLayout.

8. Security & Permissions APIs

- **Biometric API** – Implement fingerprint, face, or iris authentication.
- **Permissions API** – Request and manage runtime permissions.
- **Android Keystore API** – Securely store cryptographic keys.

9. Background Processing APIs

- **WorkManager API** – Manage background tasks efficiently.
- **JobScheduler API** – Schedule background tasks that need execution at specific times.
- **Foreground Service API** – Run persistent tasks in the background.

❖ Internal storage

Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection.

In this chapter we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory.

By default these files are private and are accessed by only your application and get deleted , when user delete your application.

Writing file

In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private , public e.t.c. Its syntax is given below –

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

The method openFileOutput() returns an instance of FileOutputStream. So you receive it in the object of FileInputStream. After that you can call write method to write data on the file. Its syntax is given below –

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Reading file

In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below –

```
FileInputStream fin = openFileInput(file);
```

After that, you can call read method to read one character at a time from the file and then you can print it. Its syntax is given below –

```
int c;
String temp="";
while( (c = fin.read()) != -1){
   temp = temp + Character.toString((char)c);
}
```

```
//string temp contains all the data of the file.
fin.close();
```

Apart from the the methods of write and close, there are other methods provided by the **FileOutputStream** class for better writing files. These methods are listed below –

Sr.No	Method & description
1	**FileOutputStream(File file, boolean append)** This method constructs a new FileOutputStream that writes to file.
2	**getChannel()** This method returns a write-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**write(byte[] buffer, int byteOffset, int byteCount)** This method Writes count bytes from the byte array buffer starting at position offset to this stream

Apart from the the methods of read and close, there are other methods provided by the **FileInputStream** class for better reading files. These methods are listed below –

Sr.No	Method & description
1	**available()** This method returns an estimated number of bytes that can be read or skipped without blocking for more input
2	**getChannel()** This method returns a read-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**read(byte[] buffer, int byteOffset, int byteCount)** This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

❖ Android external storage

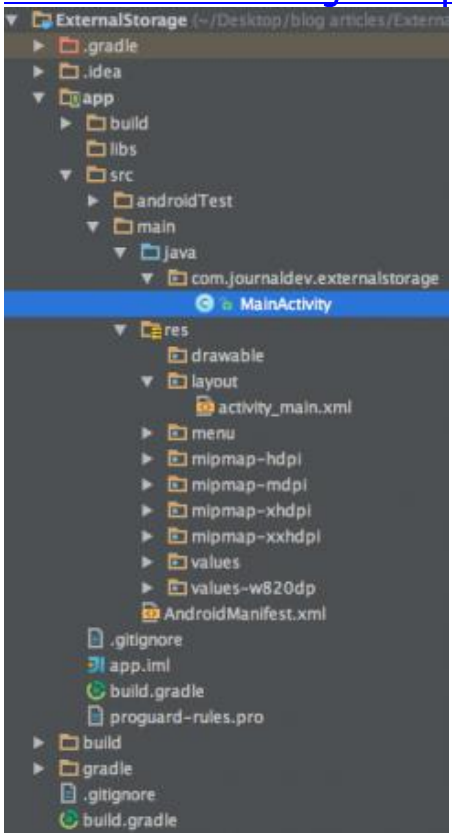
Android external storage can be used to write and save data, read configuration files etc.

External storage such as SD card can also store application data, there's no security enforced upon files you save to the external storage. In general there are two types of External Storage:

- **Primary External Storage**: In built shared storage which is "accessible by the user by plugging in a USB cable and mounting it as a drive on a host computer". Example: When we say Nexus 5 32 GB.
- **Secondary External Storage**: Removable storage. Example: SD Card

All applications can read and write files placed on the external storage and the user can remove them. We need to check if the SD card is available and if we can write to it. Once we've checked that the external storage is available only then we can write to it else the save button would be disabled.

Android External Storage Example Project Structure



Firstly, we need to make sure that the application has permission to read and write data to the users SD card, so lets open up the AndroidManifest.xml and add the following permissions:

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

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

Also, external storage may be tied up by the user having mounted it as a USB storage device. So we need to check if the external storage is available and is not read only.

```
if (!isExternalStorageAvailable() || isExternalStorageReadOnly()) {
   saveButton.setEnabled(false);
  }

private static boolean isExternalStorageReadOnly() {
  String extStorageState = Environment.getExternalStorageState();
  if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState)) {
   return true;
  }
  return false;
 }

 private static boolean isExternalStorageAvailable() {
  String extStorageState = Environment.getExternalStorageState();
  if (Environment.MEDIA_MOUNTED.equals(extStorageState)) {
   return true;
  }
  return false
```

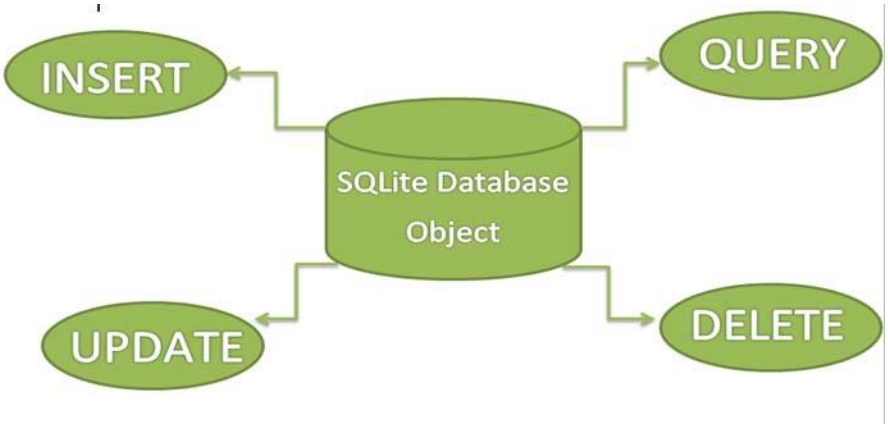
getExternalStorageState() is a static method of Environment to determine if external storage is presently available or not. As you can see if the condition is false we've disabled the save button.

Here apart from the save and read from external storage buttons we display the response of saving/reading to/from an external storage in a textview unlike in the previous tutorial where android toast was displayed.

❖ SQLite Database

SQLite is a Structure query base database, open source, light weight, no network access and standalone database. It support embedded relational database features.

Whenever an application needs to store large amount of data then using sqlite is more preferable than other repository system like SharedPreferences or saving data in files.



SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

The SQLite project was started on 2000-05-09. The future is always hard to predict, but the intent of the developers is to support SQLite through the year 2050. Design decisions are made with that objective in mind.

We the developers hope that you find SQLite useful and we entreat you to use it well: to make good and beautiful products that are fast, reliable, and simple to use. Seek forgiveness for yourself as you forgive others. And just as you have received SQLite for free, so also freely give, paying the debt forward.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

Android has built in SQLite database implementation. It is available locally over the device(mobile & tablet) and contain data in text format. It carry light weight data and suitable with many languages. So, it doesn't required any administration or setup procedure of the database.

❖ Managing data using Sqlite

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.

Its syntax is given below

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

Apart from this , there are other functions available in the database package , that does this job.

Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class.

Its syntax is given below

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS Customer(Username VARCHAR,Password
VARCHAR);");
mydatabase.execSQL("INSERT INTO Customer VALUES('ramu','ram@123');");
```

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from Customer",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

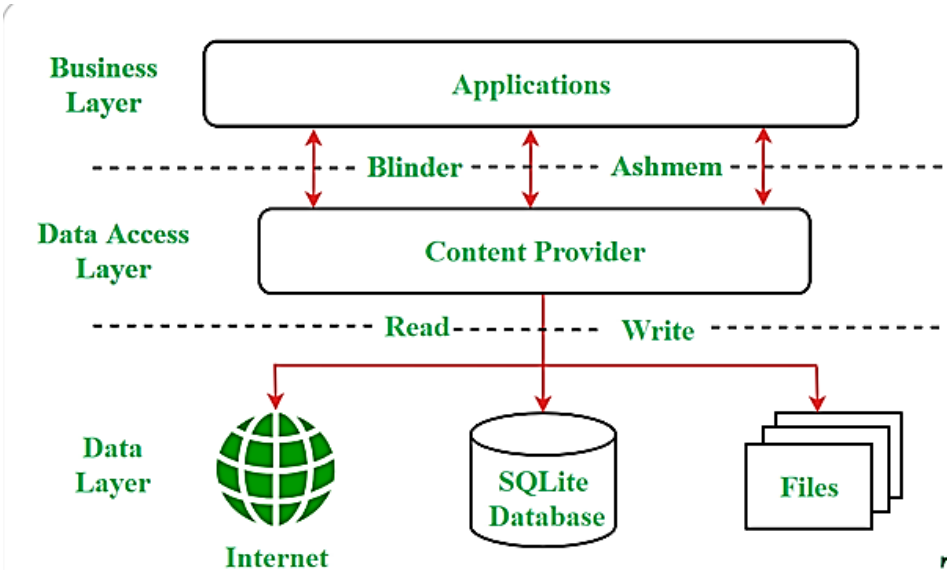
Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
  public DBHelper(){
    super(context,DATABASE_NAME,null,1);
  }
  public void onCreate(SQLiteDatabase db) {}
  public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

❖ Sharing data between Applications with Content Providers

In Android , Content Providers are a very important component that serves the purpose of a relational database to store the data of applications. The role of the content provider in the android system is like a central repository in which data of the applications are stored, and it facilitates other applications to securely access and modifies that data based on the user requirements. Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in **SQLite Database , in files , or even on a network** . In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.



Content URI

Content URI(Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.

Looking to become an expert in Android App Development? Whether you're a student or a professional aiming to advance your career in mobile app development

Structure of a Content URI: *content://authority/optionalPath/optionalID*

Details of different parts of Content URI:

- **content://** – Mandatory part of the URI as it represents that the given URI is a Content URI.
- **authority** – Signifies the name of the content provider like contacts, browser, etc. This part must be unique for every content provider.
- **optionalPath** – Specifies the type of data provided by the content provider. It is essential as this part helps content providers to support different types of data that are not related to each other like audio and video files.
- **optionalID** – It is a numeric value that is used when there is a need to access a particular record.

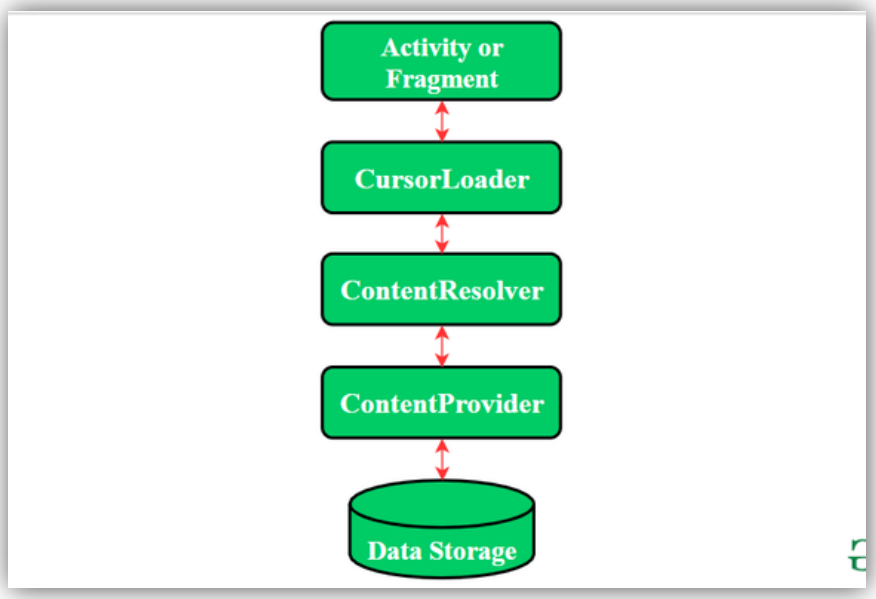
Operations in Content Provider

Four fundamental operations are possible in Content Provider namely **Create** , **Read** , **Update** , and **Delete** . These operations are often termed as **CRUD operations** .

- **Create:** Operation to create data in a content provider.
- **Read:** Used to fetch data from a content provider.
- **Update:** To modify existing data.
- **Delete:** To remove existing data from the storage.

Working of the Content Provider

UI components of android applications like Activity and Fragments use an object **CursorLoader** to send query requests to **ContentResolver.** The ContentResolver object sends requests (like create, read, update, and delete) to the **ContentProvider** as a client. After receiving a request, ContentProvider process it and returns the desired result. Below is a diagram to represent these processes in pictorial form.



Creating a Content Provider

Following are the steps which are essential to follow in order to create a Content Provider:

- Create a class in the same directory where the that **MainActivity** file resides and this class must extend the ContentProvider base class.
- To access the content, define a content provider URI address.
- Create a database to store the application data.
- Implement the **six abstract methods** of ContentProvider class.
- Register the content provider in **AndroidManifest.xml** file using **<provider> tag** .

Following are the six abstract methods and their description which are essential to override as the part of ContenProvider class:

Abstract Method	Description
query()	A method that accepts arguments and fetches the data from the desired table. Data is retired as a cursor object.
insert()	To insert a new row in the database of the content provider. It returns the content URI of the inserted row.

Abstract Method	Description
update()	This method is used to update the fields of an existing row. It returns the number of rows updated.
delete()	This method is used to delete the existing rows. It returns the number of rows deleted.
getType()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
onCreate()	As the content provider is created, the android system calls this method immediately to initialise the provider.

❖ Using Android Networking APIs

Android lets your application connect to the internet or any other local network and allows you to perform network operations.

A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection.

Checking Network Connection

Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides **ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method. Its syntax is given below –

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

Once you instantiate the object of ConnectivityManager class, you can use **getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

The last thing you need to do is to check Connected State of the network. Its syntax is given below –

```
for (int i = 0; i<info.length; i++){
  if (info[i].getState() == NetworkInfo.State.CONNECTED){
    Toast.makeText(context, "Internet is connected
    Toast.LENGTH_SHORT).show();
  }
}
```

Apart from this connected states, there are other states a network can achieve. They are listed below –

Sr.No	State
1	Connecting
2	Disconnected
3	Disconnecting
4	Suspended
5	Unknown

Performing Network Operations

After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.

Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website. Its syntax is as follows –

```
String link = "http://www.google.com";
URL url = new URL(link);
```

After that you need to call **openConnection** method of url class and receive it in a HttpURLConnection object. After that you need to call the **connect** method of HttpURLConnection class.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();
```

And the last thing you need to do is to fetch the HTML from the website. For this you will use **InputStream** and **BufferedReader** class. Its syntax is given below –

```
InputStream is = conn.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
String webPage = "",data="";

while ((data = reader.readLine()) != null){
  webPage += data + "\n";
}
```

Apart from this connect method, there are other methods available in HttpURLConnection class. They are listed below –

Sr.No	Method & description
1	**disconnect()** This method releases this connection so that its resources may be either reused or closed
2	**getRequestMethod()** This method returns the request method which will be used to make the request to the remote HTTP server
3	**getResponseCode()** This method returns response code returned by the remote HTTP server
4	**setRequestMethod(String method)** This method Sets the request command which will be sent to the remote HTTP server
5	**usingProxy()** This method returns whether this connection uses a proxy server or not

❖ web content within your Android app

Android allows you as a developer to build on the power of the web within your apps, so you can benefit from the flexibility and efficiency of being able to display certain types of content.

This lets you seamlessly integrate existing web content into your native Android application, such as to display a news feed, show interactive tutorials, display ads, or even host a mini-game without building everything from scratch. Think of it as a window to the internet, from within your app. There are two ways to embed web content into your app:

- WebView: It displays web content you control inline where you want a high degree of flexibility in customizing or updating the UI.
- Custom Tabs: A full in-app browsing experience powered by the user's default browser (see browser support) for when users click a link and you want to keep them in the app, instead of leaving to an external browser, with much of the browsing experience out-of-the-box.

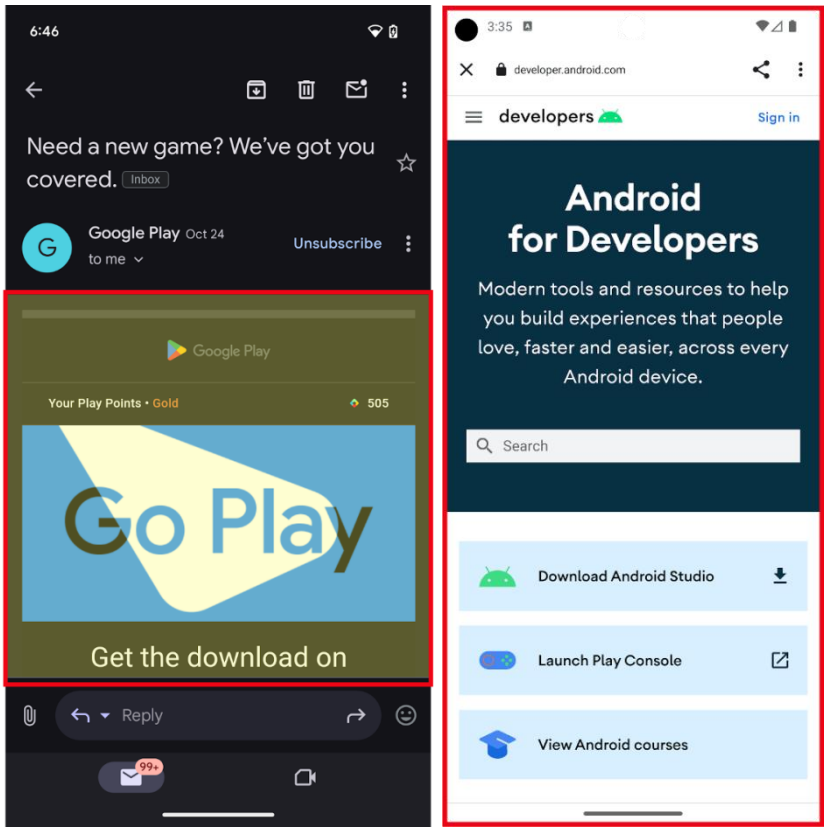


Figure 1. WebView (left) and Custom Tab (right) outlined in red.

Embed web content

- Efficiency: Reuse existing code from your website. Build on existing web technologies and content.

- Integration: Leverage external content from 3P providers, such as Media, Ads, within your app.
- Flexibility: Update content dynamically without being constrained to predefined UIs, or without releasing app updates.

use web content

There are three primary uses cases for using the Web in your Android app:

1. Embedding web content into your app as primary or supporting content: Use WebView

- Display your own web content inline as a primary experience where you want a high degree of flexibility in customizing or updating the UI.
- Display other content such as ads, legal terms and regulations, or other third-party content inline, or as a window within your app experience.

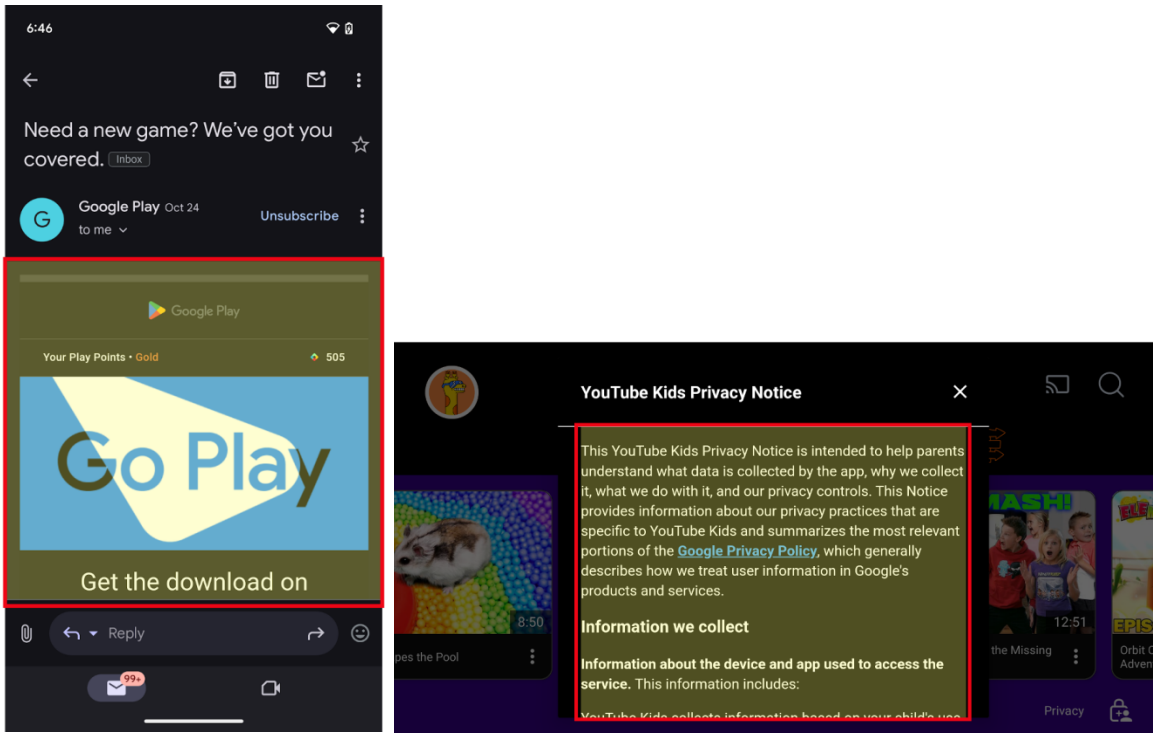


Figure 2. Web content embedded within the app with WebViews as primary (left) and supporting content (right).

2. In-app browsing using Custom Tabs, or WebView for more advanced use cases

- Have a full in-app browsing experience for when users click a link and you want to keep them in the app, instead of leaving to an external browser.
- Note: For large screen devices such as tablets and foldables, there are additional options to help apps take advantage of additional space:

- Apps can open weblinks in split screen using launch an adjacent multi-window experience. This enables users to multitask between your app and a browser at the same time. OR
- Custom Tabs have a side panel option that can open in the same task, but next to your existing app content.
- The Custom Tab is powered by the user's default browser, for browsers which support Custom Tabs.
- While it's possible to use a WebView and provide a highly customizable in-app browsing experience, we recommend Custom tabs for an out-of-the-box browser experience and seamless transition for when a user wants to open a web link in the browser.

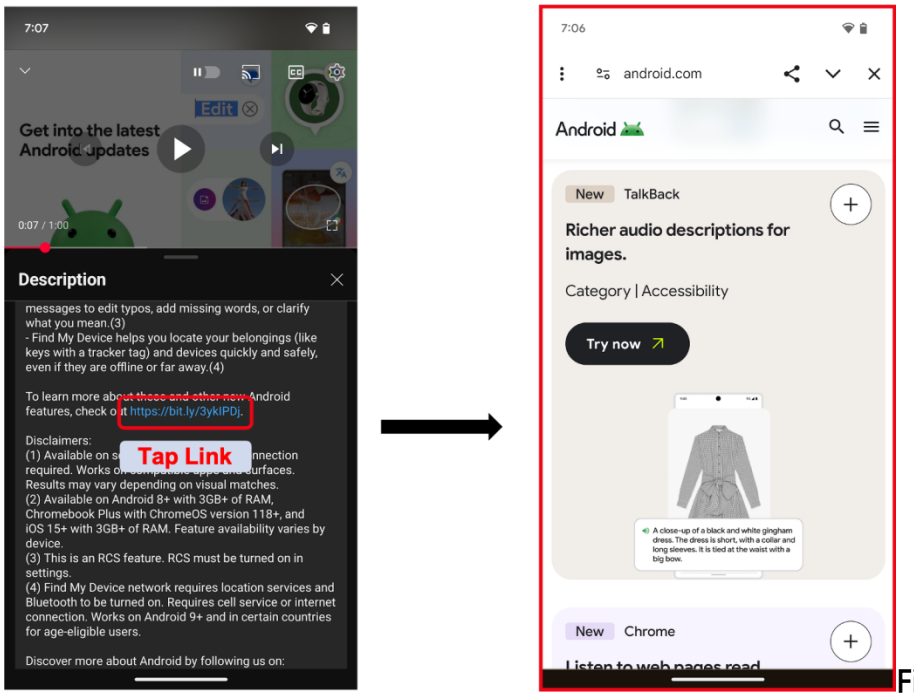


Figure 3. Clicking on an in-app link (left) and opening an in-app browser (right).

3. Login or Authentication flows within your app

Android's suggested approach is to build your login or authentication flows using Credential Manager. If you find you still need to use Embedded Web for these experiences, use the following guidance:

- Some apps use WebViews to provide sign-in flows for their users, including using a username and passkey (or password) specific to your app. This enables developers to unify the authentication flows across platforms.
- When linking out to a third-party identity provider or login experience, such as "Sign in with...", Custom Tabs are the way to go. Launching Custom Tabs ensures the user's credential remains protected and isolated to the 3rd party site.

❖ JSON Parsing

JSON stands for JavaScript Object Notation.It is an independent data exchange format and is the best alternative for XML. This chapter explains how to parse the JSON file and extract necessary information from it.

Android provides four different classes to manipulate JSON data. These classes are **JSONArray,JSONObject,JSONStringer and JSONTokenizer.**

The first step is to identify the fields in the JSON data in which you are interested in. For example. In the JSON given below we interested in getting temperature only.

```
{
   "sys":
   {
      "country":"GB",
      "sunrise":1381107633,
      "sunset":1381149604
   },
   "weather":[
      {
         "id":711,
         "main":"Smoke",
         "description":"smoke",
         "icon":"50n"
      }
   ],

  "main":
   {
      "temp":304.15,
      "pressure":1009,
   }
}
```

❖ Using the Telephony API

The telephony API is used to among other things monitor phone information including the current states of the phone, connections, network etc.

In this example, we want to utilize the telephone manager part of the Application Framework and use phone listeners (PhoneStateListener) to retrieve various phone states.

This is a simple tutorial utilizing the telephony API and its associated listener which can be good before implementing other application functions related to a phone state like connecting the Call.

TelephonyDemo.java

```
package com.telephone;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class TelephonyDemo extends Activity {
  TextView textOut;
  TelephonyManager telephonyManager;
  PhoneStateListener listener;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Get the UI
    textOut = (TextView) findViewById(R.id.textOut);

    // Get the telephony manager
    telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);

    // Create a new PhoneStateListener
    listener = new PhoneStateListener() {
      @Override
      public void onCallStateChanged(int state, String incomingNumber) {
        String stateString = "N/A";
        switch (state) {
        case TelephonyManager.CALL_STATE_IDLE:
          stateString = "Idle";
          break;
        case TelephonyManager.CALL_STATE_OFFHOOK:
          stateString = "Off Hook";
          break;
        case TelephonyManager.CALL_STATE_RINGING:
          stateString = "Ringing";
          break;
        }
```

```
        textOut.append(String.format("\nonCallStateChanged: %s", stateString));
      }
    };

    // Register the listener wit the telephony manager
    telephonyManager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
  }
}
```

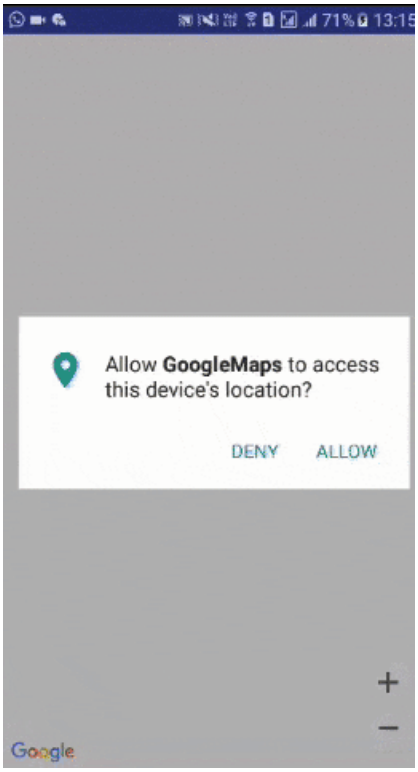
main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="https://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Telephony Demo"
                android:textSize="22sp" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/textOut"
                android:text="Output"></TextView>
</LinearLayout>
```

Output

❖ Google Maps in Android

Android allows us to integrate Google Maps in our application. For this Google provides us a library via Google Play Services for using maps. In order to use the Google Maps API, you must register your application on the **Google Developer Console** and enable the API.

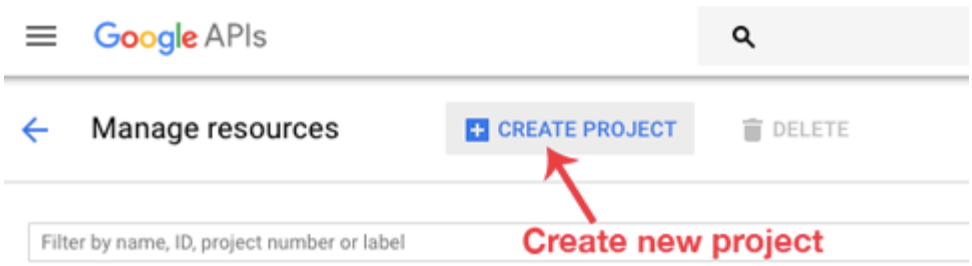


Steps For Getting The Google Maps Api Key:

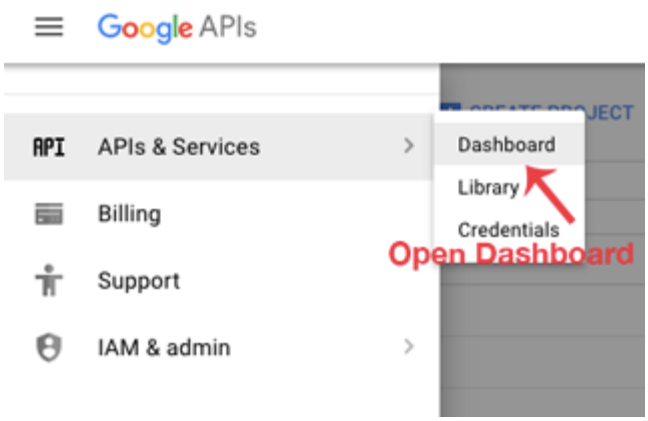
An API key is needed to access the Google Maps servers. This key is free and you can use it with any of your applications. If you haven't created project, you can follow the below steps to get started:

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

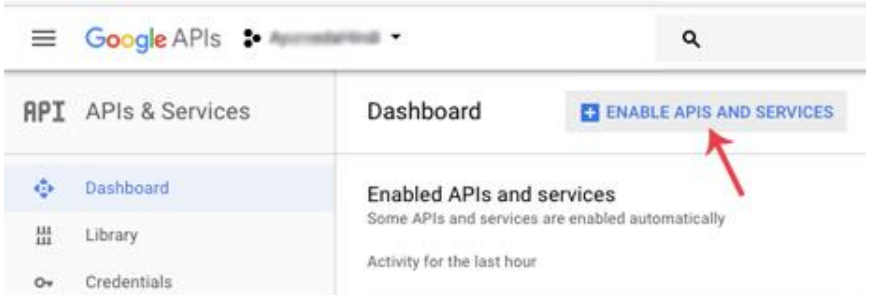
Step 2: Now create new project. You can create new project by clicking on the **Create Project** button and give name to your project.



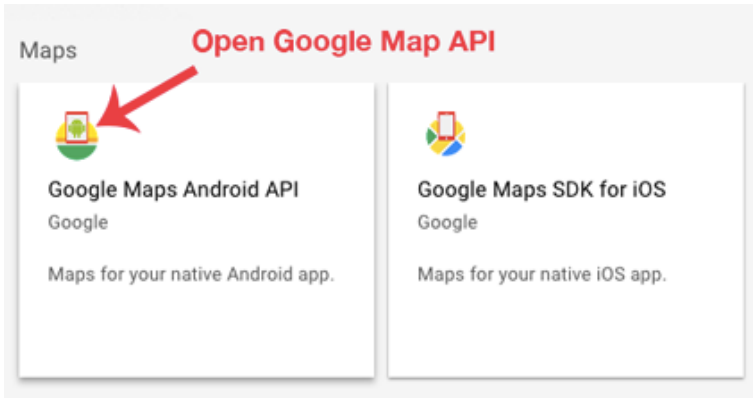
Step 3: Now click on APIs & Services and open **Dashboard** from it.



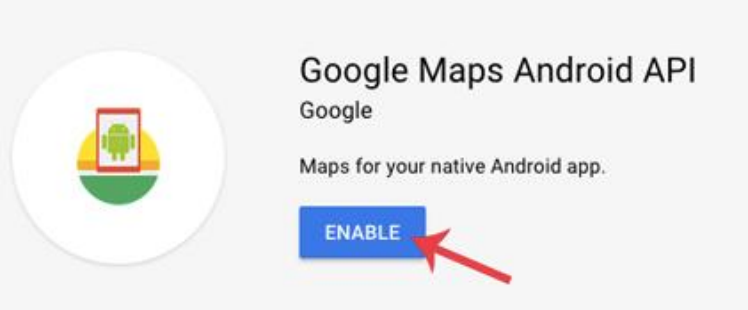
Step 4: In this open **Enable APIS AND SERICES**.



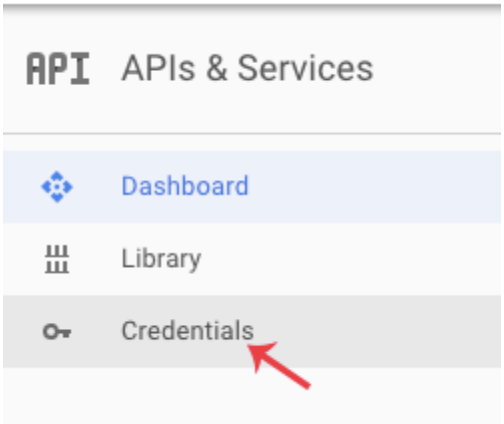
Step 5: Now open Google Map Android API.



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.

Google Maps Example

Step 1: Add Google Maps SDK Dependency

gradle

```
dependencies {
    implementation 'com.google.android.gms:play-services-maps:18.0.2'
}
```

Step 2: Get a Google Maps API Key

1. Visit **Google Cloud Console**.
2. Create a new project and enable **Maps SDK for Android**.
3. Generate an **API Key**.
4. Add the key to AndroidManifest.xml:

```
<meta-data
   android:name="com.google.android.geo.API_KEY"
   android:value="YOUR_API_KEY"/>
```

Step 3: Add a Map Fragment to Your Activity

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

Step 4: Load the Map in Java

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {
   private GoogleMap mMap;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);

      SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
      mapFragment.getMapAsync(this);
   }

   @Override
   public void onMapReady(GoogleMap googleMap) {
      mMap = googleMap;
      LatLng location = new LatLng(-34, 151);
      mMap.addMarker(new MarkerOptions().position(location).title("Marker in Sydney"));
      mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(location, 10));
   }
}
```