Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

Web Application Integration Techniques

Study Guide

Mr. Anurag Kewat
Lectuere
CSE, PIT.
Parul University

Contents

6 Web Application Integration Techniques

6.1 Introduction to Async Task

AsyncTask is an abstract class in Android that offers us the freedom to execute demanding tasks in the background while keeping the UI thread light and the application responsive. When launched, an Android application operates in a single thread. Due to this single-thread approach, tasks that take a long time to fetch a response may cause the program to become unresponsive. We use Android AsyncTask to perform these heavy tasks in the background on a separate thread and return the results back to the UI thread in order to prevent this. As a result, the UI thread is always responsive when AsyncTask is used in an Android application.

The purpose of AsyncTask was to make it possible to use the UI thread correctly and conveniently. The most frequent use case, however, was UI integration, which led to Context leaks, missed callbacks, or crashes when settings changed. Additionally, it behaves differently depending on the platform version, swallows exceptions from doInBackground, and offers little benefit over using Executors directly. AsyncTask is not intended to be a general-purpose threading system; rather, it is intended to be a helper class for Thread and Handler. AsyncTasks are best used for brief operations (a few seconds at the most.) It is strongly advised that you use the various APIs offered by the java.util.concurrent package, such as Executor, ThreadPoolExecutor, and FutureTask, if you need to keep threads running for extended periods of time.

Asynchronous tasks are divided into three generic types: **Params, Progress, & Result** and four steps: **onPreExecute, doInBackground, onProgressUpdate, & onPostExecute**. The following lists the three generic types utilized in an Android AsyncTask class:

- **Params:** Parameters sent to the task upon execution
- **Progress:** Progress units shared during the background computation
- **Result:** Result obtained from the background computation

The following definitions outline the fundamental methods used in an Android AsyncTask class:

- **doInBackground():** The code that has to be run in the background is contained in the doInBackground() method. The publishProgress() method in this method allows us to repeatedly deliver results to the UI thread. We only need to use the return statements to signal that the background processing has been finished.
- **onPreExecute():** The code that runs prior to the beginning of the background processing is contained in this function.
- **onPostExecute():** After the doInBackground method has finished processing, the onPostExecute() method is called. This method receives the output of the doInBackground method as its input.
- **onProgressUpdate():** This method can use the progress updates it receives from the publishProgress method, which publishes progress updates from the doInBackground function, to update the UI thread.

AsyncTask Example

The following code snippet must be present in the MainActivity class in order to launch an AsyncTask.

```
MyTesk mtTask = new MyTesk();
myTask.execute();
```

The execute method is used to start the background thread in the excerpt above, where we've used a sample class name that extends AsyncTask.

```
public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);


         ExampleAsync task = new ExampleAsync;

         task.execute(10);

    }


    private static class ExampleAsync extends AsyncTask<Integer, Integer, String> {

        @Override

        protected void onPreExecute() {

            super.onPreExecute();

            // ...

        }


        @Override

        protected String doInBackground(Integer... integers) {

            // ...

            return "Finished!";

        }


        @Override

        protected void onProgressUpdate(Integer... values) {
            super.onProgressUpdate(values);
            // ...
        }

        @Override
        protected void onPostExecute(String string) {
            super.onPostExecute(string);
            // ...
        }
    }
}
```

6.2 Communication with Web API

Web APIs allow applications to communicate with servers using standard internet protocols, usually HTTP or HTTPS. Communication typically follows a request–response mechanism, where the client sends a request (GET, POST, PUT, DELETE) to an API endpoint and receives structured data in return. APIs make it possible for applications to fetch remote content, authenticate users, send data to cloud storage, and integrate with external services like payment gateways, maps, and social media platforms. Communication involves defining endpoints, setting headers, sending parameters, and processing responses. Tools such as **Postman**, **cURL**, and built-in browser DevTools help developers test and understand APIs. On the application side, libraries like **Retrofit, Volley, Axios, and Fetch** simplify API integration by handling networking, threading, and error responses.
APIs provide scalability and flexibility, enabling modular development where the app and server evolve independently. Proper error-handling, secure token-based authentication, and managing network delays are essential for robust API communication.

What is Web API?

- API stands for **A**pplication **P**rogramming **I**nterface.
- A Web API is an application programming interface for the Web.
- A Browser API can extend the functionality of a web browser.
- A Server API can extend the functionality of a web server.

Browser APIs

All browsers have a set of built-in Web APIs to support complex operations, and to help accessing data. For example, the Geolocation API can return the coordinates of where the browser is located.

Example

Get the latitude and longitude of the user's position:

```
const myElement = document.getElementById("demo");

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    myElement.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  myElement.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
```

Third Party APIs

Third party APIs are not built into your browser.
To use these APIs, you will have to download the code from the Web.

Examples:

- YouTube API - Allows you to display videos on a web site.
- Twitter API - Allows you to display Tweets on a web site.
- Facebook API - Allows you to display Facebook info on a web site.

6.3 Introduction to JSON Data

JSON (JavaScript Object Notation) is a lightweight and widely-used data interchange format used for transmitting structured data across the web. It is easy for humans to read and easy for machines to parse, making it a universal format in modern web and mobile development. JSON represents data as key–value pairs, arrays, objects, booleans, and numbers.
JSON's popularity comes from its simplicity, minimal size, and compatibility with almost all programming languages. Web APIs commonly return JSON responses because of their efficiency and speed in network communication. It is more compact and readable compared to XML, making it preferred for RESTful APIs and modern data-driven applications.
Typical JSON objects include user profiles, product lists, app configuration settings, and real-time data like weather or notifications. Developers work with JSON using built-in libraries provided by languages

such as JavaScript, Java, Kotlin, Python, and Swift. Understanding JSON is essential for API integration, data manipulation, and database connectivity in web and mobile development.

Why is JSON?

JSON stands for **J**ava**S**cript **O**bject **N**otation. It is a format for structuring data. This format is used by different web applications to communicate with each other. JSON is the replacement of the XML data exchange format in JSON. It is easy to struct the data compare to XML.

It supports data structures like arrays and objects and the JSON documents that are rapidly executed on the server. It is also a Language-Independent format that is derived from JavaScript. The official media type for the JSON is application/json and to save those file **.json** extension.

Key Features of JSON

- **Human-readable**: JSON is easy to understand and write, making it suitable for developers and non-developers alike.
- **Compact format**: It is a lightweight data format that reduces bandwidth and enhances performance in data exchanges.
- **Cross-platform support**: JSON is supported by virtually all modern programming languages and platforms, including browsers, mobile apps, and servers.
- **Flexible data types**: JSON supports strings, numbers, arrays, objects, booleans, and null values, making it highly versatile for various use cases.
- **Text-based format**: JSON is a text-based format, making it simple to transmit over HTTP protocols.

JSON Syntax and Structure

JSON follows a simple structure, which consists of two primary data types: objects and arrays. Data is in name/value pairs and they are separated by commas. It uses curly brackets to hold the objects and square brackets to hold the arrays.

Objects in JSON

A **JSON object** is a collection of key-value pairs enclosed in curly braces {}. The key is a string, and the value can be a variety of data types such as strings, numbers, arrays, or even other objects.

```
{
   "name": "John Doe",
   "age": 30,
   "isStudent": false
}
```

Arrays in JSON

A **JSON array** is an ordered list of values enclosed in square brackets []. The values inside an array can be of any data type, including strings, numbers, and objects.

```
{

   "fruits": ["apple", "banana", "cherry"]
}
```

Example:

```
{
  "Courses": [
    {
      "Name" : "Java Foundation",
      "Created by" : "Geeksforgeeks",
      "Content" : [ "Java Core", "JSP",
              "Servlets", "Collections" ]
    },

    {
      "Name" : "Data Structures",
      "also known as" : "Interview Preparation Course",
      "Topics" : [ "Trees", "Graphs", "Maps" ]
    }
  ]
}
```

Advantages of JSON:

- JSON stores all the data in an array so data transfer makes easier. That's why JSON is the best for sharing data of any size even audio, video, etc.
- Its syntax is very easy to use. Its syntax is very small and light-weighted that's the reason that it executes and response in a faster way.
- JSON has a wide range for the browser support compatibility with the operating systems, it doesn't require much effort to make it all browser compatible.
- On the server-side parsing the most important part that developers want, if the parsing will be fast on the server side then the user can get the fast response, so in this case JSON server-side parsing is the strong point compare tot others.

Disadvantages of JSON:

- The main disadvantage for JSON is that there is no error handling in JSON, if there was a slight mistake in the JSON script then you will not get the structured data.
- JSON becomes quite dangerous when you used it with some unauthorized browsers. Like JSON service return a JSON file wrapped in a function call that has to be executed by the browsers if the browsers are unauthorized then your data can be hacked.
- JSON has limited supported tools that we can use during JSON development.

JSON vs XML: Which is Better?

Feature	**JSON**	**XML**

Readability	Easy to read and write	More verbose and complex
Size	Smaller, lightweight	Larger due to additional tags
Parsing Speed	Faster parsing	Slower parsing
Data Representation	Based on key-value pairs	Based on tags and attributes
Support	Broadly supported	Supported, but less efficient for data exchange

JSON is generally preferred over XML due to its simpler syntax, smaller file sizes, and faster parsing speed.

Conclusion
JSON has become the standard format for data interchange in web applications, APIs, and cloud-based systems due to its lightweight nature, ease of use, and versatility. Whether we're working on front-end or back-end development, understanding how to work with JSON is crucial for efficiently handling data. With its simple syntax, broad compatibility, and speed advantages, JSON is the go-to format for modern **web development**. Although it has a few limitations, like the lack of **error handling** and potential security risks, these can be mitigated through best practices.

6.4 Introduction to JSON Data

JSON parsing refers to converting JSON data into native objects that an application can use. When an app receives a JSON response from a server, the data must be parsed to extract values like strings, numbers, arrays, or objects. Parsing ensures that the application can understand and process API responses, display information to users, or store it in local databases.

Most programming languages provide JSON parsers—for example, Gson, Moshi, Jackson (Java/Kotlin), JSON.parse() (JavaScript), and Codable (Swift). The parsing process generally involves identifying keys and mapping them to variables or model classes. Developers must also handle formatting errors, missing fields, or unexpected data types, which are common in real-world API responses.

There are two major parsing techniques: manual parsing (reading data key-by-key) and automatic parsing (mapping JSON to objects using model classes). JSON parsing helps build dynamic applications capable of processing live data such as news feeds, social media content, product updates, and weather forecasts.

The JSON.parse() method is used to convert a JSON string into a JavaScript object. It’s become important when dealing with data in JSON format, interacting with APIs, or storing data in the browser.

- It converts a JSON string into a JavaScript object.
- Throws a SyntaxError if the input string is not valid JSON.
- Accepts an optional reviver function to transform the parsed data.

```
const s = '{"name": "Rahul", "age": 25, "city": "Delhi"}';
const obj = JSON.parse(s);
console.log(obj);
```

Output

```
{ name: 'Rahul', age: 25, city: 'Delhi' }
```

- s is a valid JSON string.
- JSON.parse() converts it into an object obj.
- Now you can access the object's properties like obj.name, obj.age, etc.

Syntax:

JSON.parse(text[, reviver]);

- **text**: The JSON string to parse.
- **reviver (optional):** A function to transform the parsed values before returning.

Using the Reviver Function

The reviver function allows you to modify the parsed values before they are returned. You can use it to manipulate data as needed during the parsing process.

```
const s = '{"name": "Rahul", "age": 25, "city": "Delhi"}';
const obj = JSON.parse(s, (key, value) => {
  if (key === 'age') {
    return value + 1;
  }
  return value;
});
console.log(obj);
```

Output

```
{ name: 'Rahul', age: 26, city: 'Delhi' }
```

- The reviver function adds 1 to the age value during the parsing process.
- The modified object is returned with age updated to 26.

Common Use Cases of JSON.parse()

1. Parsing API Responses

When you receive a response from an API in JSON format, you need to convert the string to an object to work with it in JavaScript.

```
fetch('https://api.example.com/user')

   // Fetches the data and converts it to JSON
```

```
    .then(res => res.json())
    .then(s => {
        // Convert JSON string to object
        const obj = JSON.parse(s);
        // Access properties like obj.name
        console.log(obj.name);
    });
```

2. Storing and Retrieving Data from localStorage
You can store objects as JSON strings in localStorage. When you retrieve them, you need to parse the string back into an object.

```
// Saving an object
const a = { name: 'Rahul', age: 25 };
localStorage.setItem('user', JSON.stringify(a));

// Retrieving and parsing the object
const s = localStorage.getItem('user');
const obj = JSON.parse(s);
console.log(obj.name);  // Output: Rahul
```

3. Working with Configuration Files
You can load configuration settings stored in a JSON file, then parse it into a usable JavaScript object.

```
const s = '{"theme": "dark", "language": "en"}';
const obj = JSON.parse(s);
console.log(obj.theme);
```

Handling Common Errors with JSON.parse()

1. Invalid JSON Format: If the JSON string is malformed, JSON.parse() will throw a SyntaxError.

```
// Invalid JSON (keys must be in double quotes)
const s = "{name: 'Rahul', age: 25}";
try {
    const obj = JSON.parse(s);  // Throws SyntaxError
} catch (e) {
   console.log("Error:", e.message);
}
```

Output

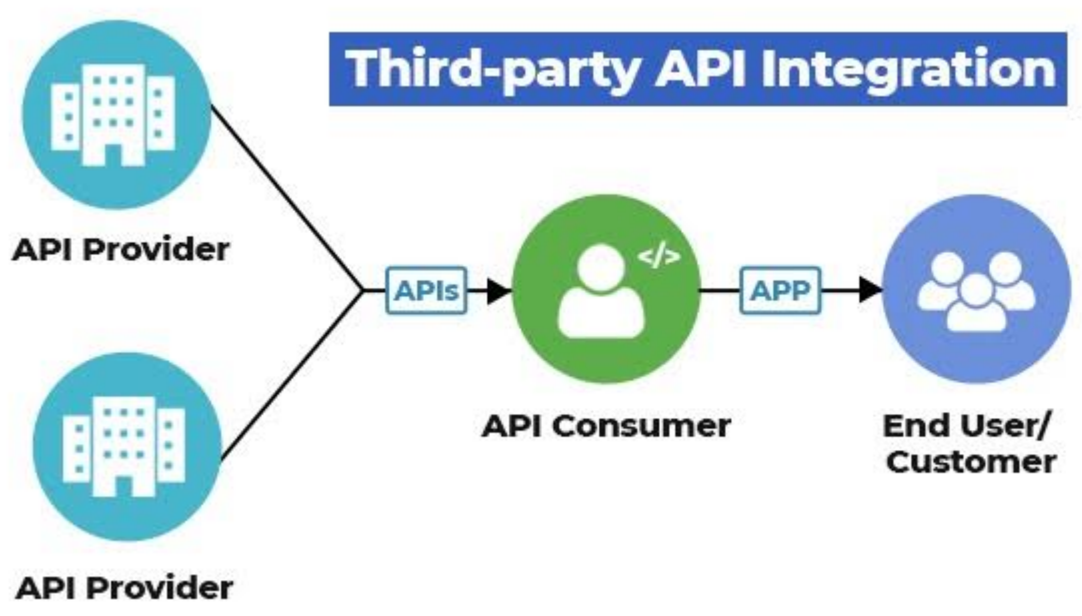
```
Error: Unexpected token n in JSON at position 1
```

2. Non-String Input: JSON.parse() only accepts strings. If you try to parse a number or an array, it will throw an error.

```
const a = 12345;
try {
   const obj = JSON.parse(a);
} catch (e) {
   console.log("Error:", e.message);
}
```

6.5 Implementation of Third-Party Library to Fetch Network Data

Third-party libraries simplify network operations by providing optimized, ready-made components to handle API calls, threading, caching, and error-handling. Instead of writing complex networking code manually, developers use libraries like **Retrofit**, **Volley**, **OkHttp**, **Alamofire**, and **Axios** to fetch and process network data efficiently.
These libraries support asynchronous execution, automatic JSON parsing, retry mechanisms, logging, and secure communication. For example, **Retrofit** works seamlessly with converters like Gson or Moshi to parse JSON directly into model classes. **Volley** is excellent for handling multiple network operations efficiently, while **OkHttp** provides low-level control for custom networking requirements.
Using third-party libraries enhances code readability, reduces development time, and ensures robustness. They also offer built-in features like response caching, timeout controls, token-based authentication, and connection pooling. Modern application development heavily depends on these libraries for scalable and maintainable networking solutions.



Benefits of Using Third-Party API

Integrating third-party APIs into your applications offers several significant benefits.

Time and Cost Savings

Building certain features and functionalities from scratch can be time-consuming and costly. Third-party APIs provide pre-built solutions that can be quickly integrated, saving development time and resources.

Enhanced Functionality

Third-party APIs can supercharge your applications by providing access to services and data that would be challenging or impossible to create on your own. Whether it's geolocation services, social media integration, or payment processing, these APIs extend your application's capabilities.

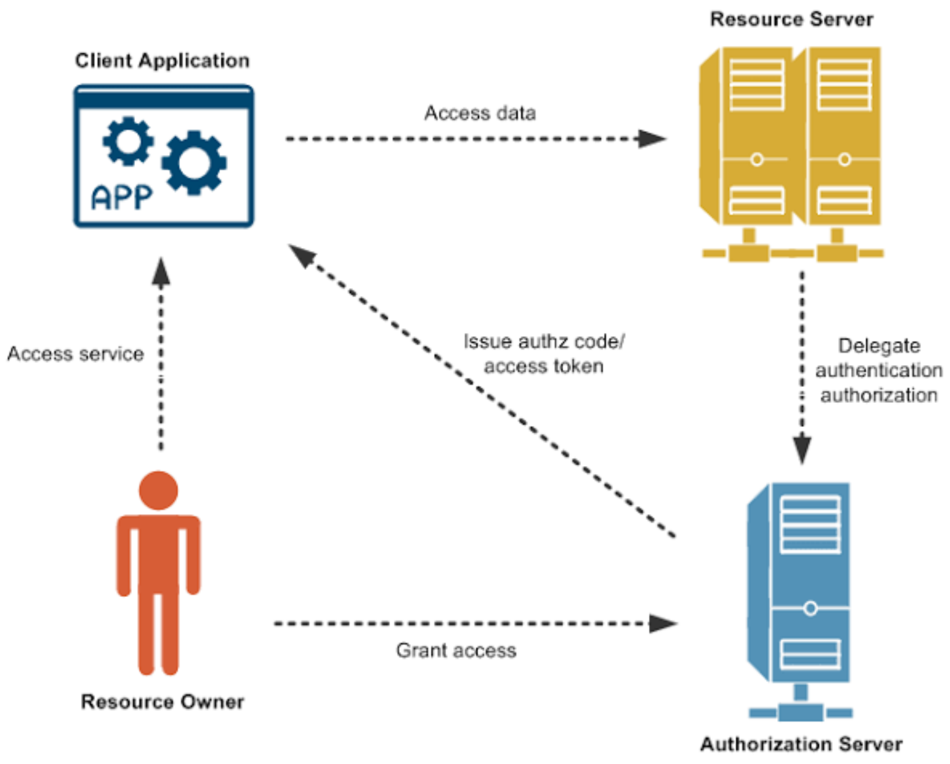
API Authentication

As we now understand the importance of third-party APIs, let us get into the first technical aspect: API authentication. Authentication is the process of verifying the identity of the requester. For security reasons, most third-party APIs require some form of authentication before they grant access.

Common Authentication Methods

There are several authentication methods commonly used with APIs.

API Keys



API keys are unique tokens provided by the API provider. These keys are included in API requests to identify the application and grant access. While they are simple to use, API keys can be less secure if not handled properly.

Here’s an example of how you might include an API key in an HTTP request using Python and the popular requests library.

```
python
import requests
api_key = “your_api_key_here”
url = “https://api.example.com/data”
response = requests.get(url, headers={“Authorization”: f”Bearer {api_key}”})
# Handle the API response here
```

OAuth

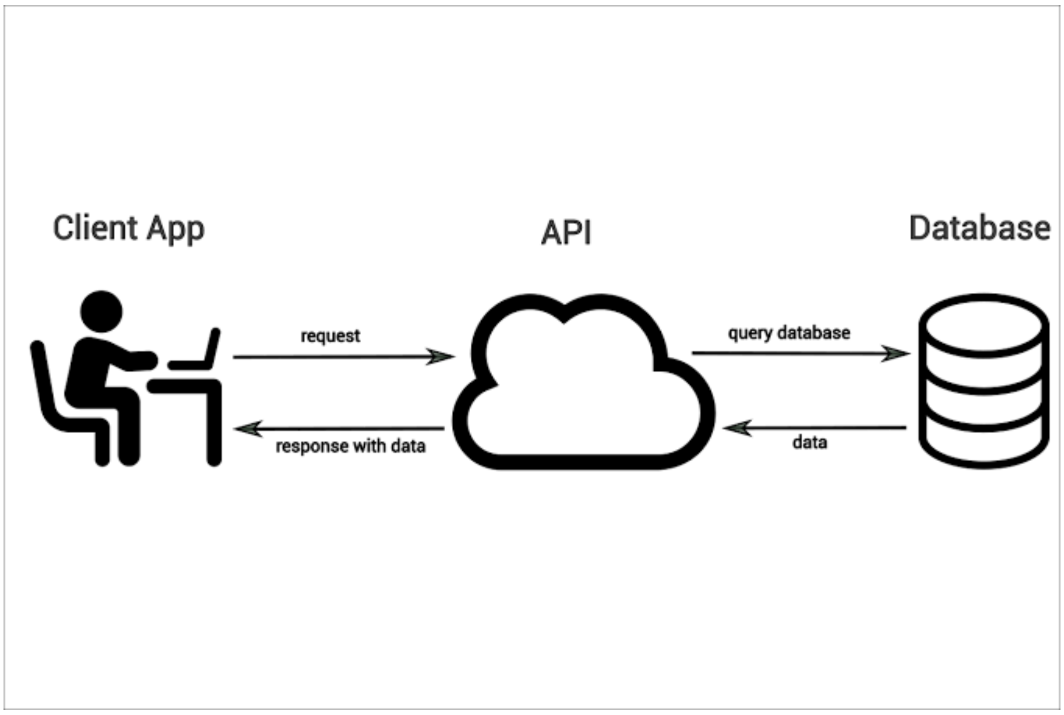
OAuth (Open Authorization) is a more complex but secure authentication method often used by social media platforms and other web services. It involves obtaining access tokens that represent the user’s authorization to access their data on a third-party service.

To implement OAuth, you will typically need to follow a series of steps, including redirecting users to the service’s authentication page, obtaining an access token, and using that token to make authenticated requests.

The specific implementation of OAuth can vary greatly depending on the service you are integrating. Popular libraries and frameworks often provide tools to streamline the OAuth process, so you do not have to build it from scratch.

```
javascript
// Example OAuth flow in JavaScript using a library like Passport.js
const passport = require('passport');
const TwitterStrategy = require('passport-twitter').Strategy;
// Configure the Twitter OAuth strategy
passport.use(new TwitterStrategy({
    consumerKey: 'your_consumer_key',
    consumerSecret: 'your_consumer_secret',
    callbackURL: 'your_callback_url'
  },
  (token, tokenSecret, profile, done) => {
    // Handle user authentication here
  }
));
```

Making API Requests



Now that you understand the basics of API authentication, it's time to explore how to make requests to third-party APIs. The HTTP protocol plays a central role in this process, as it serves as the foundation for communication between your application and the API.

HTTP Methods

HTTP supports several methods for making requests, but the most commonly used ones in API integration are:

- **GET:** Used to retrieve data from the API. It's ideal for fetching information without making any changes on the server.
- **POST:** Used for sending data to the API, often to create new resources or update existing ones.
- **PUT:** Similar to POST but typically used for updating existing resources, and it should be idempotent, meaning repeated requests produce the same result as a single request.
- **DELETE:** As the name suggests, this method is used to delete resources on the server.

Constructing API Requests

To make an API request, you'll typically use a programming language-specific library or module that simplifies the process. One popular choice is Axios for JavaScript, or the built-in requests library in Python.

Here's a simplified example of making a GET request to retrieve data from a hypothetical API:

```
javascript
const axios = require('axios');
const apiUrl = 'https://api.example.com/data';
axios.get(apiUrl)
  .then((response) => {
    // Handle the API response here
    console.log(response.data);
  })
  .catch((error) => {
    // Handle any errors that occur during the request
    console.error(error);
  });
```

In this example, we use Axios to send a GET request to the API endpoint, and we handle the response and potential errors accordingly.

6.6 Notifications

Notifications are a key feature of modern mobile and web applications, allowing apps to communicate important information to users even when the app is not active. Notifications can deliver updates, reminders, alerts, messages, or promotional content. In Android, notifications are managed through the **NotificationManager** and built using the **NotificationCompat** builder.
There are two types: **local notifications** (generated within the app) and **push notifications** (sent from a remote server using services like Firebase Cloud Messaging). Notifications may include text, images, action buttons, sound, and vibrations. They enhance user engagement and improve app retention when used responsibly.
Developers must follow UI/UX guidelines to avoid disturbing the user with unnecessary alerts. Notification channels, introduced in newer Android versions, allow users to customize notification preferences. Proper implementation ensures secure delivery, timely updates, and better interaction with the application.

Appearances on a device

Notifications automatically appear to users in different locations and formats. A notification appears as an icon in the status bar, a more detailed entry in the notification drawer, and a badge on the app's icon. Notifications also appear on paired wearables.

Status bar and notification drawer

When you issue a notification, it first appears as an icon in the status bar

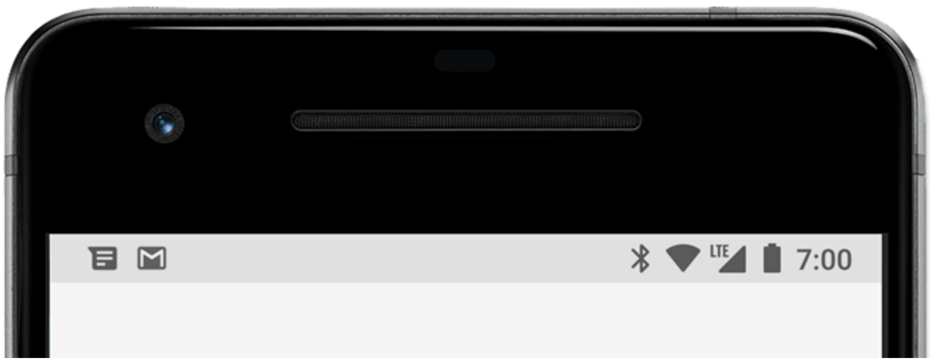
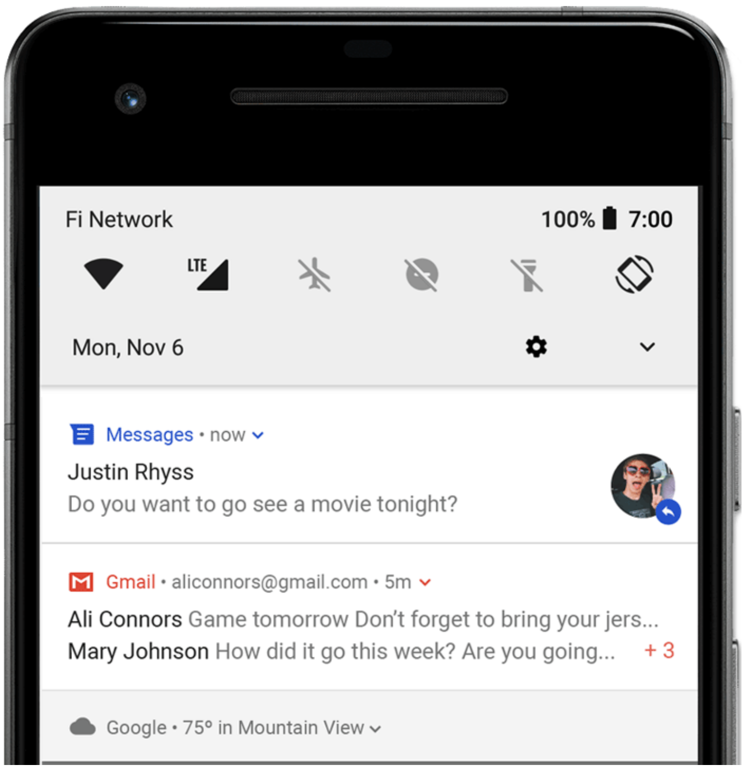


Figure 1. Notification icons appear on the left side of the status bar.

Users can swipe down on the status bar to open the notification drawer, where they can view more details and take actions with the notification.



Fi Network 100% 7:00

Mon, Nov 6

Messages • now
Justin Rhyss
Do you want to go see a movie tonight?

Gmail • aliconnors@gmail.com • 5m
Ali Connors Game tomorrow Don't forget to bring your jers...
Mary Johnson How did it go this week? Are you going... + 3

Google • 75° in Mountain View

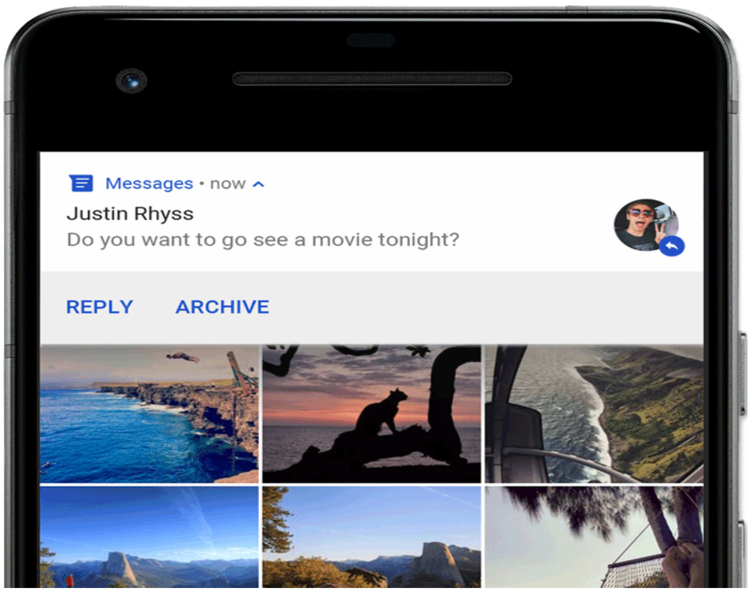
Figure 2. Notifications in the notification drawer.

Users can drag down on a notification in the drawer to reveal the expanded view, which shows additional content and action buttons, if provided. Starting in Android 13, this expanded view includes a button that lets users stop an app that has ongoing foreground services.

A notification remains visible in the notification drawer until it's dismissed by the app or user.

Heads-up notification

Beginning with Android 5.0, notifications can briefly appear in a floating window called a *heads-up notification*. This behavior is normally for important notifications that the user needs to know about immediately, and it only appears if the device is unlocked.



Messages • now
Justin Rhyss
Do you want to go see a movie tonight?

Figure 3. A heads-up notification appears in front of the foreground app.

The heads-up notification appears when your app issues the notification. It disappears after a moment, but it remains visible in the notification drawer as usual.

Conditions that might trigger heads-up notifications include the following:

- The user's activity is in fullscreen mode, such as when the app uses fullScreenIntent.
- The notification has high priority and uses ringtones or vibrations on devices running Android 7.1 (API level 25) and lower.
- The notification channel has high importance on devices running Android 8.0 (API level 26) and higher.

Lock screen

Beginning with Android 5.0, notifications can appear on the lock screen.

You can programmatically set whether notifications posted by your app show on a secure lock screen and, if so, the level of detail visible.

Users can use the system settings to choose the level of detail visible in lock screen notifications or to disable all lock screen notifications. Starting with Android 8.0, users can disable or enable lock screen notifications for each notification channel.

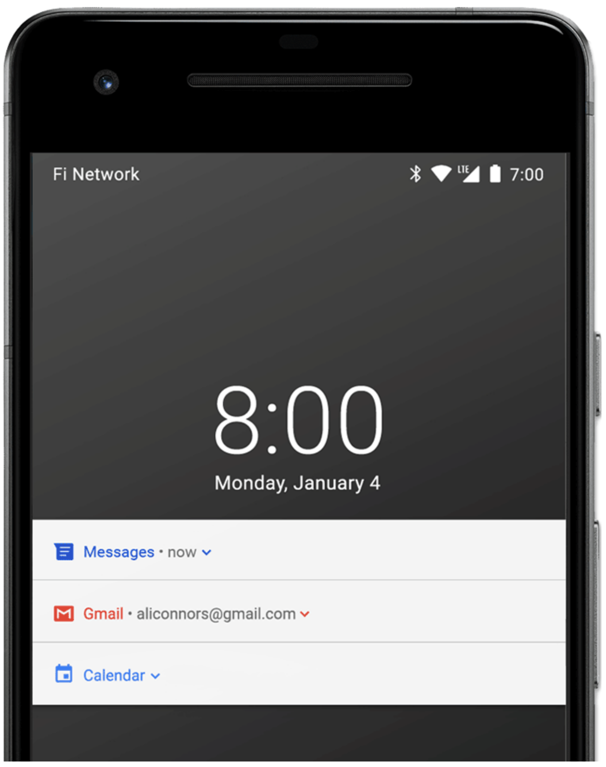


Figure 4. Notifications on the lock screen with sensitive content hidden.

Notification anatomy

The design of a notification is determined by system templates, and your app defines the contents for each portion of the template. Some details of the notification appear only in the expanded view.

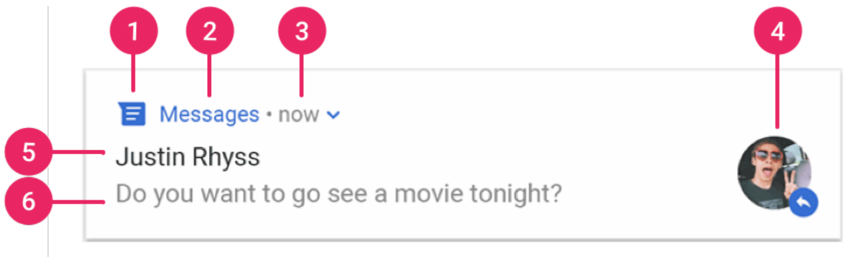


Figure 7. A notification with basic details.

The most common parts of a notification are indicated in figure 7, as follows:

- Small icon: required; set using setSmallIcon().
- App name: provided by the system.
- Time stamp: provided by the system, but you can override it using setWhen() or hide it using setShowWhen(false).
- Large icon: optional; usually used only for contact photos. Don't use it for your app icon. Set using setLargeIcon().
- Title: optional; set using setContentTitle().
- Text: optional; set using setContentText().

We strongly recommend using system templates for proper design compatibility on all devices. If necessary, you can create a custom notification layout.

6.7 Telephony API and Google API

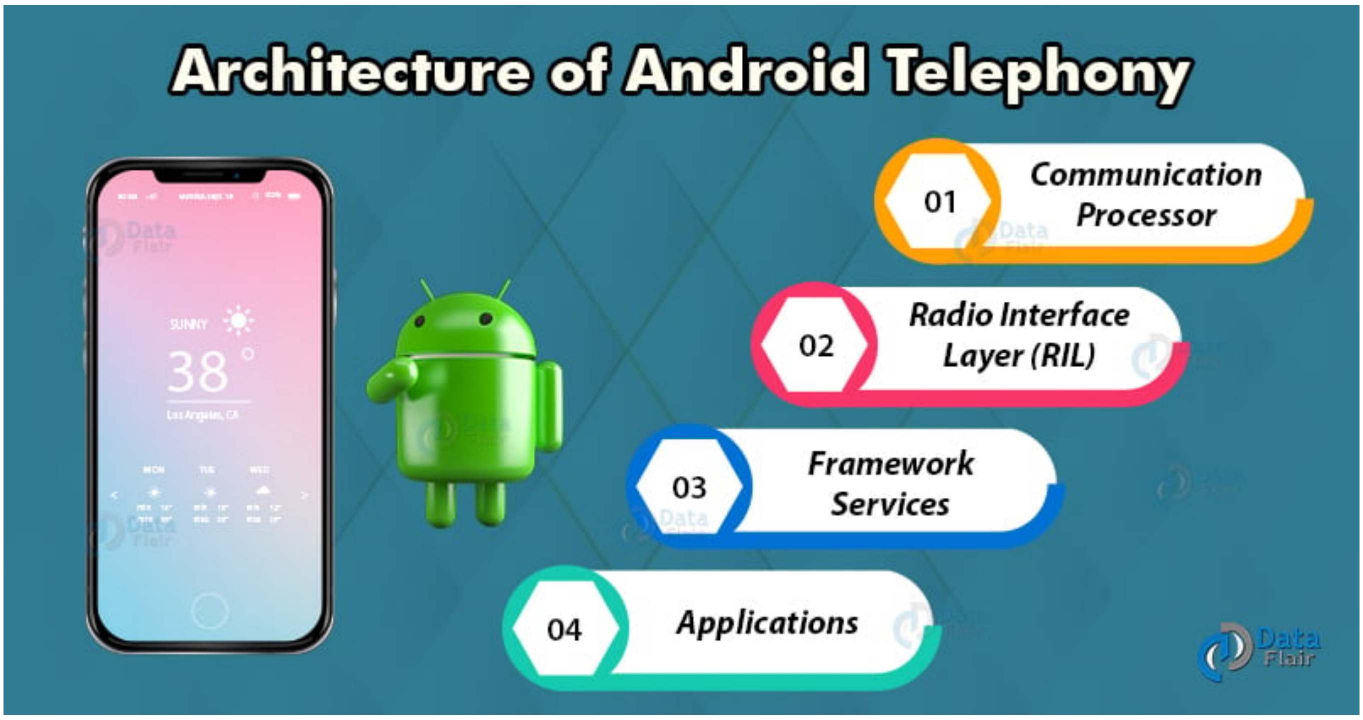
The Telephony API provides access to a device's telecommunication services, allowing applications to interact with calling features, SIM information, network status, and device identifiers. Using this API, developers can detect the network type, read the phone number (if permitted), monitor signal strength, and manage incoming call states.
The Telephony API is commonly used in dialer apps, call-blocking apps, OTP-based login systems, and telecom service applications. It offers features such as making calls programmatically using intents, listening to call states, retrieving IMEI (older versions), and identifying the mobile network provider. However, telephony features require strict permissions (like READ_PHONE_STATE or CALL_PHONE) due to privacy and security concerns. Many sensitive functions are now restricted to system apps. Developers must ensure responsible use of the API while maintaining user trust and complying with operating system policies.

What is Android Telephony?

Android Telephony framework provides us the functionalities of the mobile. It gives us information about functionalities like calls, SMS, MMS, network, data services, IMEI number, and so on.

For better understanding, you can consider Dialer, Browser, Sim App toolkit, Broadcast receivers, and so on.

Architecture of Android Telephony


Architecture of Android Telephony
01 Communication Processor
02 Radio Interface Layer (RIL)
03 Framework Services
04 Applications
Data Flair

Android Telephony architecture works in 4 layers that are :

1. **Communication Processor**
2. **Radio Interface Layer (RIL)**
3. **Framework Services**
4. **Applications**

Let us try to understand them briefly one by one :

1. Communication Processor
It is an input/output processor to distribute and collect data from a number of remote terminals. It is a specialized processor designed to communicate with the data communication network.

2. Radio Interface Layer
It is a bridge between the hardware and Android phone framework services. Rather we say, it is a protocol stack for Telephone. It has two main components that are:

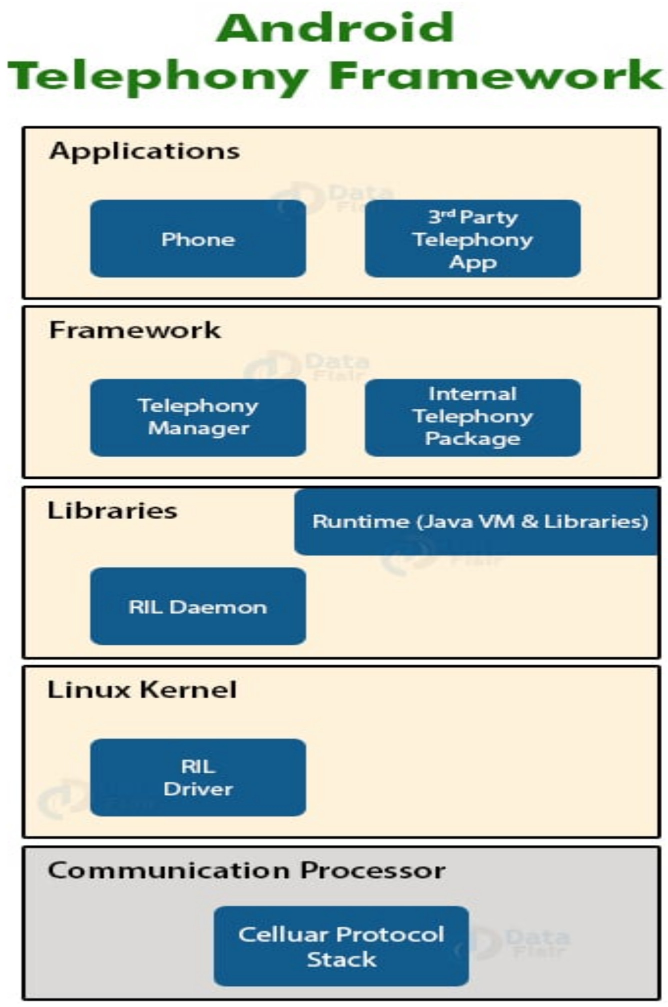
- **RIL Daemon**– It starts when the android system starts. It reads the system properties to find a library that is to be used for Vendor RIL.
- **Vendor RIL**– It is also known as RIL Driver. It can be understood as a library that is specific to each modem.

3. Framework Services
The telephony Framework starts and initializes along with the system. All the queries by Application API are directed to RIL using these services.

4. Application
These are the Application UI related to telephony such as Dialer, SMS, MMS, Call tracker, etc. These applications start with the android system boot up. These are tied with framework services of telephony.

Android Telephony Framework consists of two types of packages that are:

1. Internal Telephony Packages: This is generally the used for default telephony app.apk.
2. Open Technology Packages: This is for third-party apps.
These are 4 layers of Android Telephony



Implementation of Android Telephony

We will now implement it in **Android Studio** using the following steps:

1. At first, create a new project and name it.

2. Now, open the layout file, and define the following:

Open the **main_activity-xml** file –

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:label="Telephony"
android:paddingLeft="1dp"
android:paddingTop="1dp"
android:paddingRight="1dp"
android:paddingBottom="1dp"
tools:context=".MainActivity">
<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
android:layout_marginLeft="30dp"
android:layout_marginTop="150dp"
android:fontFamily="adamina"
android:text="Mobile Details:"
android:textSize="20dp" />
</RelativeLayout>
```

3. Now open the **MainActivity.java** file and write the following code there:

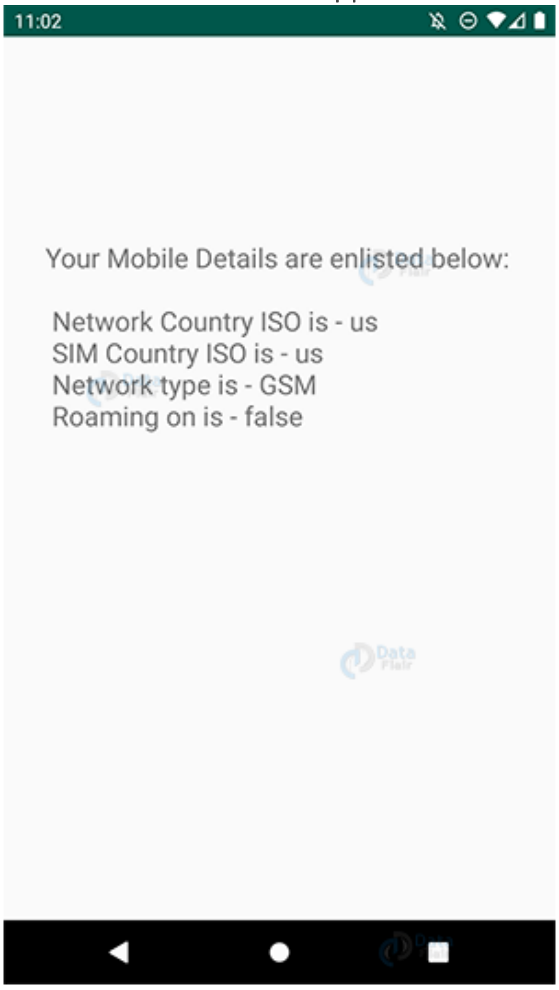
```
package com.DataFlair.androidtelephony;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;
public class MainActivity extends Activity {
package com.DataFlair.androidtelephony;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;
public class MainActivity extends Activity {
TextView tv;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
tv = findViewById(R.id.textView);
//instance of TelephonyManager
TelephonyManager tele_man = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
String nwcountryISO = tele_man.getNetworkCountryIso();
String SIMCountryISO = tele_man.getSimCountryIso();
String PhoneType = ""; // it'll hold the type of phone i.e CDMA / GSM/ None
int phoneType = tele_man.getPhoneType();
switch (phoneType) {
case (TelephonyManager.PHONE_TYPE_CDMA):
PhoneType = "CDMA";
break;
case (TelephonyManager.PHONE_TYPE_GSM):
PhoneType = "GSM";
break;
case (TelephonyManager.PHONE_TYPE_NONE):
PhoneType = "NONE";
break;
}
// true or false for roaming or not
boolean checkRoaming = tele_man.isNetworkRoaming();
String data = "Your Mobile Details are enlisted below: \n";
data += "\n Network Country ISO is - " + nwcountryISO;
data += "\n SIM Country ISO is - " + SIMCountryISO;
data += "\n Network type is - " + PhoneType;
data += "\n Roaming on is - " + checkRoaming;
//Now we'll display the information
tv.setText(data);
}
}
```

4. Now we will write the following code in the **Manifest.xml file**.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.DataFlair.androidtelephony"
android:versionCode="1"
android:versionName="1.0">
<uses-sdk android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher_background"
```

```
android:label="@string/app_name"
android:paddingLeft="1dp"
android:paddingTop="1dp"
android:paddingRight="1dp"
android:paddingBottom="1dp"
android:theme="@style/AppTheme">
<activity
<b> </b>android:name="com.DataFlair.androidtelephony.MainActivity"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<meta-data
android:name="preloaded_fonts"
android:resource="@array/preloaded_fonts" />
</application>
</manifest>
```

5. Now we will RUN the application and following would be the Output.



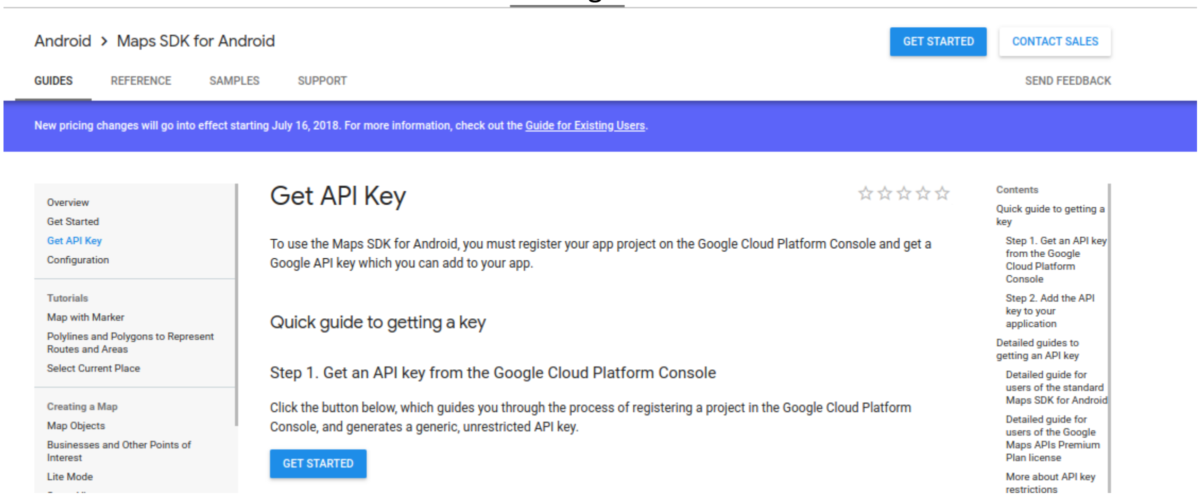
```
11:02

Your Mobile Details are enlisted below:

 Network Country ISO is - us
 SIM Country ISO is - us
 Network type is - GSM
 Roaming on is - false
```

Google API

Google APIs enable integration of powerful Google services into applications, including Maps, Location Services, YouTube, Firebase, Google Sign-In, Drive, and more. These APIs allow apps to access real-time data, cloud storage, authentication systems, machine learning features, and location-based services.
For example, **Google Maps API** allows embedding interactive maps, markers, routes, and geolocation.
Firebase APIs support authentication, real-time databases, cloud messaging, analytics, and hosting.
Google Sign-In simplifies user authentication with secure OAuth 2.0 protocols.
Google APIs usually require enabling services in the Google Cloud Console and using an API key. They provide detailed documentation, SDKs, and libraries that simplify integration. The APIs support various platforms including Android, iOS, Web, and backend systems.
Integrating Google APIs significantly enhances app functionality, improves scalability, and provides enterprise-level reliability. They enable developers to build smarter, more interactive, and data-driven applications with minimal effort.

Maps are of great use and it increases the productivity of an app. Google Maps API allows Android developers to integrate Google Maps in their app. Below is the step-by-step process to integrate Google Maps into Android applications:

1. Goto https://developers.google.com/maps/documentation/android-sdk/get-api-key and click on *"GET STARTED"* button as shown in the figure:


Android > Maps SDK for Android
GET STARTED CONTACT SALES
GUIDES REFERENCE SAMPLES SUPPORT SEND FEEDBACK

New pricing changes will go into effect starting July 16, 2018. For more information, check out the Guide for Existing Users.

Overview
Get Started
Get API Key
Configuration
Tutorials
Map with Marker
Polylines and Polygons to Represent Routes and Areas
Select Current Place
Creating a Map
Map Objects
Businesses and Other Points of Interest
Lite Mode

Get API Key

To use the Maps SDK for Android, you must register your app project on the Google Cloud Platform Console and get a Google API key which you can add to your app.

Quick guide to getting a key

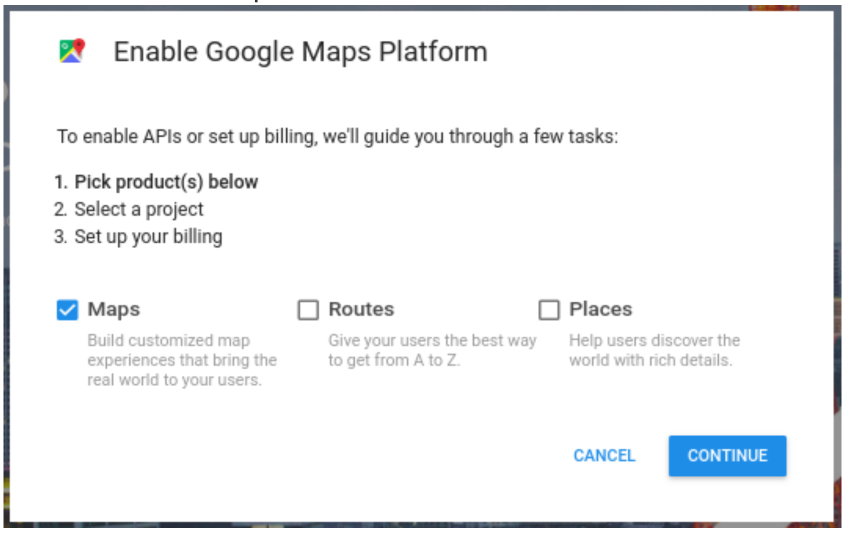
Step 1. Get an API key from the Google Cloud Platform Console

Click the button below, which guides you through the process of registering a project in the Google Cloud Platform Console, and generates a generic, unrestricted API key.

GET STARTED

Contents
Quick guide to getting a key
Step 1. Get an API key from the Google Cloud Platform Console
Step 2. Add the API key to your application
Detailed guides to getting an API key
Detailed guide for users of the standard Maps SDK for Android
Detailed guide for users of the Google Maps APIs Premium Plan license
More about API key restrictions

2. Now select the Map checkbox and click on the Continue button as shown


Enable Google Maps Platform

To enable APIs or set up billing, we'll guide you through a few tasks:

1. Pick product(s) below
2. Select a project
3. Set up your billing

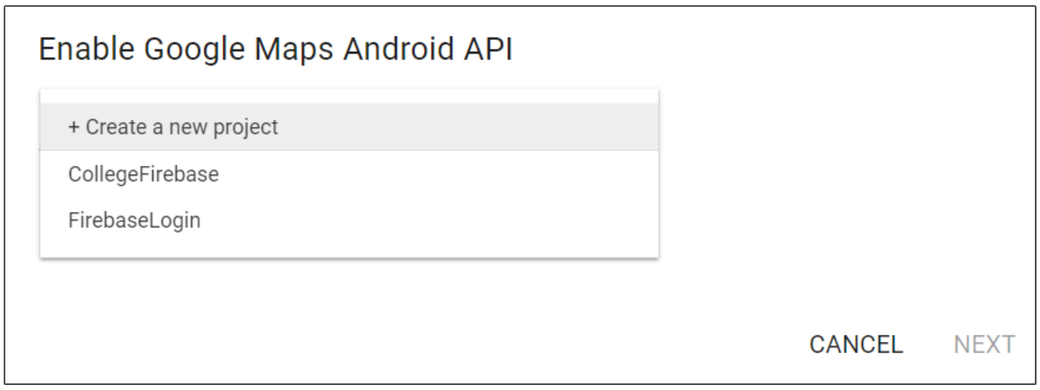
[✓] Maps
Build customized map experiences that bring the real world to your users.

[ ] Routes
Give your users the best way to get from A to Z.

[ ] Places
Help users discover the world with rich details.

CANCEL CONTINUE

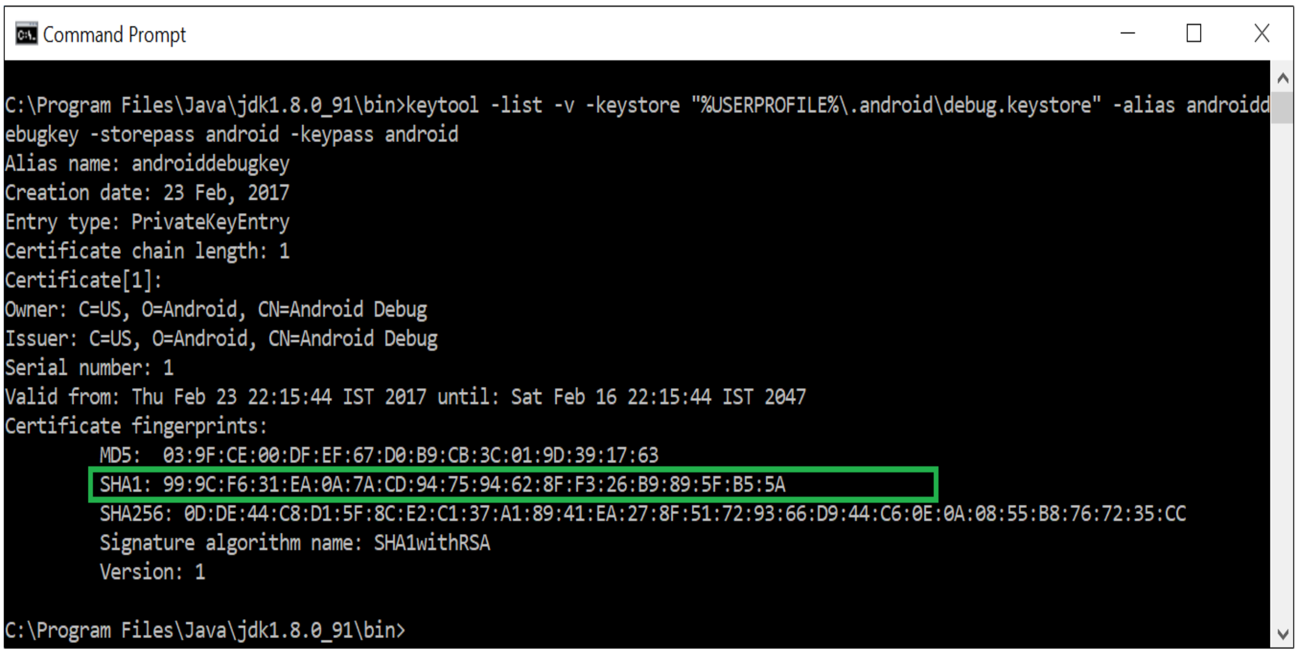
3. Select a project in which you want to enable Google Map API, and click on Next. A new key will be generated for the chosen project.



4. Skip the Billing Process
5. For integrating Google Map API, your machine's SHA1 certificate is needed. So to find SHA1 certificate, follow below steps:
   - Open Command Prompt and go to your Java bin Folder

     ```
     cd C:\Program Files\Java\jdk1.8.0_91\bin
     ```

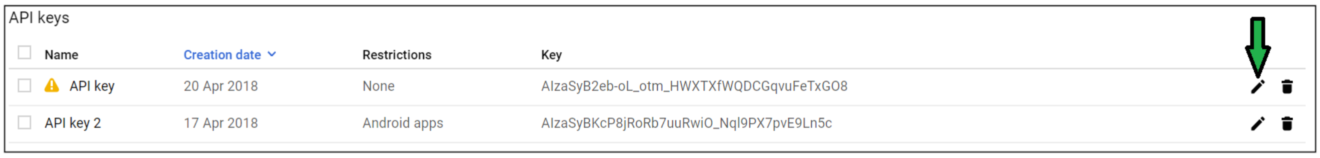
   - Give the following CMD command for getting Certificate Footprints:
     keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android



```
C:\Program Files\Java\jdk1.8.0_91\bin>keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androidd
ebugkey -storepass android -keypass android
Alias name: androiddebugkey
Creation date: 23 Feb, 2017
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: C=US, O=Android, CN=Android Debug
Issuer: C=US, O=Android, CN=Android Debug
Serial number: 1
Valid from: Thu Feb 23 22:15:44 IST 2017 until: Sat Feb 16 22:15:44 IST 2047
Certificate fingerprints:
         MD5:  03:9F:CE:00:DF:EF:67:D0:B9:CB:3C:01:9D:39:17:63
         SHA1: 99:9C:F6:31:EA:0A:7A:CD:94:75:94:62:8F:F3:26:B9:89:5F:B5:5A
         SHA256: 0D:DE:44:C8:D1:5F:8C:E2:C1:37:A1:89:41:EA:27:8F:51:72:93:66:D9:44:C6:0E:0A:08:55:B8:76:72:35:CC
         Signature algorithm name: SHA1withRSA
         Version: 1

C:\Program Files\Java\jdk1.8.0_91\bin>
```

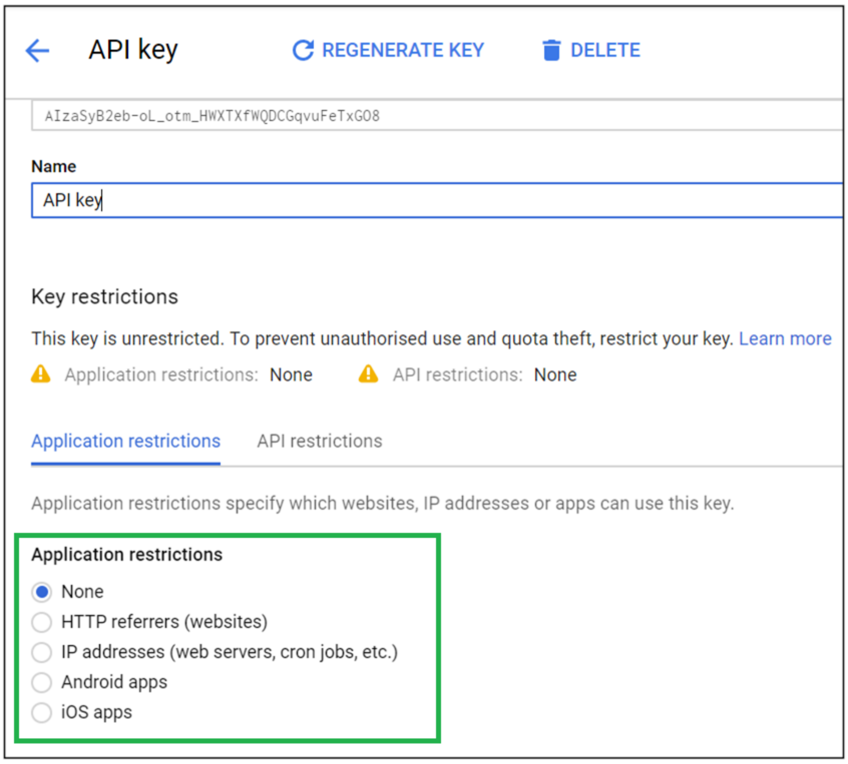
6. Go to https://accounts.google.com/v3/signin/identifier?continue=https%3A%2F%2Fconsole.cloud.google.com%2Fapis%2Fcredentials&followup=https%3A%2F%2Fconsole.cloud.google.com%2Fapis%2Fcredentials&ifkv=AdBytiMAoknixS1AYi5QAHFJ68Hz1jVVHrl5gYrWayMf-v1sZ0LK3hmQUcxb1ONzf09_PM2It5s38Q&osid=1&passive=1209600&service=cloudconsole&flowName=WebLiteSignIn&flowEntry=ServiceLogin&dsh=S-1389637297%3A1752216380674114

7. In the API keys section, click on Pencil button made on the right of API key that you want to select, for attaching your app with.


API keys

Name	Creation date	Restrictions	Key
API key	20 Apr 2018	None	AIzaSyB2eb-oL_otm_HWXTXfWQDCGqvuFeTxGO8
API key 2	17 Apr 2018	Android apps	AIzaSyBKcP8jRoRb7uuRwiO_Nql9PX7pvE9Ln5c

8. In Application Restrictions, select Android apps



AIzaSyB2eb-oL_otm_HWXTXfWQDCGqvuFeTxGO8

Name
API key

Key restrictions
This key is unrestricted. To prevent unauthorised use and quota theft, restrict your key. Learn more
Application restrictions: None API restrictions: None

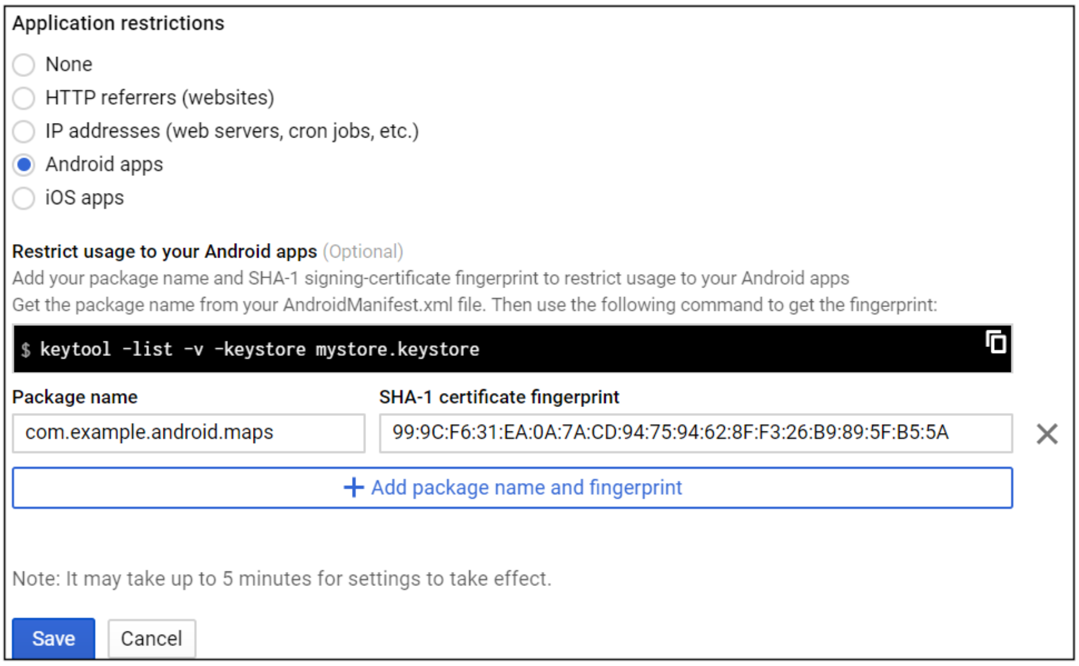
Application restrictions API restrictions

Application restrictions specify which websites, IP addresses or apps can use this key.

Application restrictions
- [✓ None]
- HTTP referrers (websites)
- IP addresses (web servers, cron jobs, etc.)
- Android apps
- iOS apps

9. Click on Add package name and fingerprint

10. Enter your app's package name and the fingerprint which was found in above steps and click Save button.


Application restrictions
- None
- HTTP referrers (websites)
- IP addresses (web servers, cron jobs, etc.)
- [✓ Android apps]
- iOS apps

Restrict usage to your Android apps (Optional)
Add your package name and SHA-1 signing-certificate fingerprint to restrict usage to your Android apps
Get the package name from your AndroidManifest.xml file. Then use the following command to get the fingerprint:

```
$ keytool -list -v -keystore mystore.keystore
```

Package name: com.example.android.maps
SHA-1 certificate fingerprint: 99:9C:F6:31:EA:0A:7A:CD:94:75:94:62:8F:F3:26:B9:89:5F:B5:5A

Note: It may take up to 5 minutes for settings to take effect.

11. Insert the following in Project ->app ->src ->build.gradle ->dependencies

```
compile 'com.google.android.gms:play-services:11.6.0'
```

- Add the following declaration within the element of AndroidManifest.xml

```
<meta-data
   android:name="com.google.android.gms.version"
   android:value="@integer/google_play_services_version" />
<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value="ENTER API_KEY GENERATED BY YOU IN ABOVE STEPS" />
```

- Add the following permissions in Manifest.xml

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

- Specify following specifications in Manifest.xml

```
<uses-feature
      android:glEsVersion="0x00020000"
      android:required="true"/>
```

- Add the following fragment code in ActivityMain.xml for adding Google map to your activity.

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

- Add the following code in MainActivity.java

```
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.LatLng;
import androidx.appcompat.app.AppCompatActivity;

public class MapsMarkerActivity extends AppCompatActivity implements
OnMapReadyCallback {
  // onCreate method is called when the activity is first created
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Retrieve the content view that renders the map.
    setContentView(R.layout.ActivityMain);

    // Get the SupportMapFragment and request notification
    // when the map is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment)
        getSupportFragmentManager().findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
  }

  // This method is called when the map is ready to be used.
  @Override
  public void onMapReady(GoogleMap googleMap) {
    // Add a marker in Sydney, Australia,
    // and move the map's camera to the same location.
    LatLng myPos = new LatLng(Location.getLatitude(), Location.getLongitude());
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(myPos));
  }
}
```

Parul® University
Vadodara, Gujarat
NAAC GRADE A++

https://paruluniversity.ac.in/