UNIT: 5 LOCATION BASED SERVICES (LBS), COMMON ANDROID API, NOTIFICATIONS, SERVICES, DEPLOYMENT OF APPLICATIONS

CS-31 Mobile Application Development in Android using Kotlin

- **GLOBAL POSITIONING SERVICES (GPS)**
- **GEOCODING LOCATIONS**
- **MAPPING LOCATIONS**
- **ANDROID NETWORKING API**
- **ANDROID WEB API**
- **ANDROID TELEPHONY API**
- **ANDROID NOTIFICATION, NOTIFYING THE USER, NOTIFYING WITH THE STATUS BAR**
- **VIBRATING THE PHONE**
- **BLINKING THE LIGHTS**
- **CUSTOMIZING THE NOTIFICATIONS SERVICES**
- **APPLICATION DEVELOPMENT USING JSON IN MYSQL**
- **PUBLISH ANDROID APPLICATION**

-: Assignment 5 :-

- **WHAT IS GPS?**
- **WHAT IS JSON?**
- **WHAT IS LBS?**
- **EXPLAIN GEOCODING LOCATIONS.**
- **EXPLAIN ANDROID NETWORKING API.**
- **EXPLAIN ANDROID WEB API/ WEB VIEW**
- **EXPLAIN ANDROID TELEPHONY API**
- **WRITE STEPS TO IMPLEMENT TELEPHONY API WITH EXAMPLE**
- **WRITE SHORT NOTE ON ANDROID NOTIFICATION**
- **EXPLAIN TYPES OF NOTIFICATION IN ANDROID.**
- **EXPLAIN SERVICE WITH ITS TYPE**
- **EXPLAIN STEPS TO IMPLEMENT SERVICE AND EXPLAIN ITS METHODS**
- **EXPLAIN JSON OBJECT AND JSON ARRAY WITH SUITABLE EXAMPLE**
- **WRITE STEPS TO PUBLISH YOUR ANDROID APPLICATION.**

PREPARED BY: PROF. N.K.PANDYA (PHD*, MCA, BCA, BPP)
KAMANI SCIENCE COLLEGE, AMRELI(BCA/BSC)

GPS

Global Positioning Systems, widely known as GPSs, have a great importance since the days of World War II. Although the initial focus was mainly on military targeting, fleet management, and navigation, commercial usage began finding relevance as the advantages of radiolocation were extended to (but not limited to) tracking down stolen vehicles and guiding civilians to the nearest hospital, gas station, hotel, and so on. A GPS system consists of a network of 24 orbiting satellites, called NAVSTAR (Navigation System with Time and Ranging), and placed in space in six different orbital paths with four satellites in each orbital plane and covering the entire earth under their signal beams.

- The orbital period of these satellites is twelve hours.
- The satellite signals can be received anywhere and at any time in the world.
- The spacing of the satellites is arranged such that a minimum of five satellites are in view from every point on the globe.
- The first GPS satellite was launched in February 1978.
- Each satellite is expected to last approx 7.5 years, and replacements are constantly being built and launched into orbit.
- Each satellite is placed at an altitude of about 10,900 nautical miles and weights about 862 kg.
- The satellites extend to about 5.2m (17ft) in space including the solar panels. Each satellite transmits on three frequencies.
- The GPS is based on well-known concept called the triangulation technique.

Using GPS Features in Your Applications

LBS services and hardware such as a built-in precision GPS are optional features for Android devices. In addition to requiring the appropriate permissions, you can specify which optional features your application requires within the Android Manifest file. You can declare that your application uses or requires specific LBS services using the <usesfeature> tag of the Android Manifest file. Although this tag is not enforced by the Android operating system, it enables popular publication mechanisms such as the Android Market to filter your app and provide it only to users with appropriate devices. If your application will only function well on devices with some sort of method for determining the current location, you could use the following <uses-feature> tag in your application's manifest file:

```
<uses-feature android:name="android.hardware.location" />
```

If your application requires a precise location fix (that is, the device has functional GPS hardware, not just cell tower triangulation or other such mechanisms), you could use the following <uses-feature> tag instead:

```
<uses-feature android:name="android.hardware.location.gps" />
```

New settings for the <uses-feature> tag have been added in recent Android SDK releases. For example, the values we've discussed, such as android .hardware .location, were added in Android 2.2 (API Level 8).

Geocoding Locations

Geocoding is the process of transforming a description of a location—such as a pair of coordinates, an address, or a name of a place—to a location on the earth's surface. You can geocode by entering one location description at a time or by providing many of them at once in a table. The resulting locations are output as geographic features with attributes, which can be used for mapping or spatial analysis.

You can quickly find various kinds of locations through geocoding. The types of locations that you can search for include points of interest or names from a gazetteer, like mountains, bridges, and stores; coordinates based on latitude and longitude or other reference systems, such as the Military Grid Reference System (MGRS) or the U.S. National Grid system; and addresses, which can come in a variety of styles and formats, including street intersections, house numbers with street names, and postal codes.

Mapping Locations

To determine device location, you need to perform a few steps and make some choices. The following list summarizes this process:

- Retrieve an instance of the LocationManager using a call to the getSystemService() method using the LOCATION_SERVICE.
- Add an appropriate permission to the AndroidManifest.xml file, depending on what type of location information the application needs.
- Choose a provider using either the getAllProviders() method or the getBestProvider() method.
- Implement a LocationListener class.
- Call the requestLocationUpdates() method with the chosen provider and the LocationListener object to start receiving location information.

Android networking API

Android contains the standard Java network java.net package which can be used to access network resources. The base class for HTTP network access in the java.net package is the HttpURLConnection class.
To access the Internet your application requires the android.permission.INTERNET permission. To check the network state your application requires the android.permission.ACCESS_NETWORK_STATE permission.
Performing network operations on Android can be cumbersome. You have to open and close a connections, enable caches and ensure to perform the network operation in a background thread.

To simplify these operations several popular Open Source libraries are available. The most popular ones are the following:

- **Volley**
- **OkHttp**

Good practices for network access under Android

Within an Android application you should avoid performing long running operations on the user interface thread. This includes file and network access.
As of Android 3.0 (Honeycomb) the system is configured to crash with a NetworkOnMainThreadException exception, if network is accessed in the user interface thread. A typical setup for performing network access in a productive Android application is using a service. While it is possible to do network access from an activity or a fragment, using a service typical leads to a better overall design because you code in the activity becomes simpler.

Android web API/ Web View

If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using WebView. The WebView class is an extension of Android's View class that allows you to display web pages as a part of your activity layout. It does not include any features of a fully developed web browser, such as navigation controls or an address bar. All that WebView does, by default, is show a web page. A common scenario in which using WebView is helpful is when you want to provide information in your application that you might need to update, such as an end-user agreement or a user guide. Within your Android application, you can create an Activity that contains a WebView, then use that to display your document that's hosted online. Another scenario in which WebView can help is if

your application provides data to the user that always requires an Internet connection to retrieve data, such as email. In this case, you might find that it's easier to build a Web your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout. Instead, you can design a web page that's tailored for Android devices and then implement a WebView in your Android application that loads the web page.
This document shows you how to get started with WebView and how to do some additional things, such as handle page navigation and bind JavaScript from your web page to client-side code in your Android application.

Adding a WebView to your application

To add a WebView to your Application, simply include the <WebView> element in your activity layout. For example, here's a layout file in which the WebView fills the screen.

```
<WebView
 xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/webview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

To load a web page in the WebView, use loadUrl(). For example:

```
val myWebView :WebView =findViewById(R.id.webview)
myWebView.loadUrl("http://www.kscpac.org");
```

Android telephony API

Android Telephony is a package provided by Android to operate and get information about the network interfaces and device details. It has several interfaces and classes that are used to provide various functionalities. Android Telephony Manager is one such class that is present in the android.telephony package. It provides you with several methods to get the network or telephony information and know its current state. Along with this, you also get the subscriber information using the Telephony Manager methods. To access the Telephony Manager in your programming, you can simply import the android.telphony.TelephonyManager class. After that, you need to use the getSystemService() to create the TelephonyManager object, and then you can use it to get system information.

Syntax: val telephonyManager = getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager

How you can implement the telephony and get information about your device and subscribers. Follow the below steps:

1: Run your Android Studio and create a new project.
2: Select Empty Activity, name your project and choose API level 22 and proceed.
3: After the Gradle build is finished, open your AndroidManifest.xml file and add the permissions below.

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

4: Open your activity_main.xml and design UI.
5: Code it into MainActivity.kt file.

```
class MainActivity : AppCompatActivity()
{
   //declaring variables
   lateinit var telephonyResult:TextView
   override fun onCreate(savedInstanceState: Bundle?)
   {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
```

```
        //binding variables with text view
        telephonyResult = findViewById(R.id.telephonyResult)
        //creating a Telephony Manager object
        val telephonyManager = getSystemService(Context.TELEPHONY_SERVICE) as
              TelephonyManager
        var result = "";
        //fetching information of your device using telephony manager
        //and storing them in result
        result = result + "\nCountry ISO = "+ telephonyManager.networkCountryIso
        result = result+ "\nSIM Country ISO = "+ telephonyManager.simCountryIso
        result = result+ "\nRoaming Enabled = "+ telephonyManager.isNetworkRoaming
        //displaying our results
        telephonyResult.text = result
    }
}
```

Android Notification/ Customizing the notifications Services

Notifications are like a message which are used to update users about something. The update can be related to any event, sale, reminder, message, news or warning. Android Notifications occur outside the app's interface. All your notifications are available on your status bar.

Notifications are an essential part of any application because of the following:
1. Updating users timely about any updates.
2. The users can get the updates without even opening their apps.
3. They can decide whether to react to the notification or remove it from the status bar.
4. The users can customize the notifications. For example, they can mute or mark a notification as important.

The above were some of the uses of the notifications. There are many more which we will explore while we proceed.

Types of Android Notifications

User-Generated Notifications

These are the notifications that the users themselves generate. Some of the common examples of this are as follows: text messaging, post liking, etc.

Context-Generated Notifications

The notifications are generated based on the permission sought, for example, Location-based applications.

System-Generated Notifications

The notifications that originate from the system are termed as System-Generated Notifications. For example – System Update Notifications.

User's Action based Notifications

The notifications which come with an option for users to interact are termed User's Action-Based Notifications. For example, you can notice the chance to reply directly from the notification itself in mail or chat applications.

Push Notifications

Push notifications are notifications from the Internet or, more specifically, from the cloud. For example, the manufacturers use this to update users about upcoming sales or new product launches, etc.

Passive Notifications

The notifications that come with some additional data or information are termed passive notifications, such as news and weather feed.

Smart Notifications

A smart notification is a notification that gets delivered to specific devices only. So, for example, if you wished to send some information to all your premium users, you could use smart notifications.

Implementation of Android Notification

Step 1: Launch your Android Studio.

Step 2: Select Create a New Project.

Step 3: Select Empty Activity and proceed.

Step 4: Enter your application name. In my case, it's "NotificationDemo." Next, select Kotlin from the dropdown. For the API level, select API 22 for now.

Step 5: Now go to res —> layout —-> and open activity_main.xml. Now here, you need to add a button using which notification will be generated.

```
<Button
      android:id="@+id/btn_notify"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Show My Notification"
      android:textSize="15sp"
/>
```

Step 6: Now, we need to create a layout for our notification. To do so, go to res—>layout—> and create a new layout XML file.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="64dp"
   android:orientation="horizontal">
   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:padding="10dp"
      android:src="@mipmap/techvidvan" />
   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_gravity="center_vertical"
      android:orientation="vertical">
      <TextView
         android:id="@+id/tv_title"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:textStyle="bold" />
      <TextView
         android:id="@+id/tv_content"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />
   </LinearLayout>
</LinearLayout>
```

Step 7: Now, you just need to edit your MainActivity.kt file as the last step.

```
class MainActivity : AppCompatActivity()
{
   //Variable declarations
```

```
    lateinit var notificationManager : NotificationManager
    lateinit var notificationChannel : NotificationChannel
    lateinit var builder : Notification.Builder
    lateinit var btn_notify:Button
    //keeping channel id
    private val channelId = "com.nkp.notifications"
    private val description = "KSC Demo"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //Defining the Notification Manager Object
        notificationManager        =        getSystemService(Context.NOTIFICATION_SERVICE)        as
NotificationManager
        //Binding the button with our variable
        btn_notify = findViewById(R.id.btn_notify)
        //setting up on click listener
        //whenever the user click the button
        //he gets the notification
        btn_notify.setOnClickListener {
            //setting up the pending intent to the current activity
            val intent = Intent(this,MainActivity::class.java)
            val              pendingIntent              =              PendingIntent.getActivity(this,0,intent,
PendingIntent.FLAG_UPDATE_CURRENT)
            //defining the layout of notification
            val contentView = RemoteViews(packageName,R.layout.my_notification)
            contentView.setTextViewText(R.id.tv_title,"Tech Vidvan's Notification")
            contentView.setTextViewText(R.id.tv_content,"Hope you are enjoying the tutorial.")
            //checking the android version of your device
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
            {
                //setting up the properties for the notification
                //based on android version above oreo
                notificationChannel=
NotificationChannel(channelId,description,NotificationManager.IMPORTANCE_HIGH)
                notificationChannel.enableLights(true)
                notificationChannel.lightColor = Color.GREEN
                notificationChannel.enableVibration(false)
                notificationManager.createNotificationChannel(notificationChannel)
                builder = Notification.Builder(this,channelId)
                    .setContent(contentView)
                    .setSmallIcon(R.mipmap.ic_launcher_round)
                    .setLargeIcon(BitmapFactory.decodeResource(this.resources,R.mipmap.ic_launcher))
                    .setContentIntent(pendingIntent)
            }else
            {
                //setting up the properties for the notification
                //based on android version BELOW oreo
                builder = Notification.Builder(this)
                    .setContent(contentView)
                    .setSmallIcon(R.mipmap.ic_launcher_round)
                .setLargeIcon(BitmapFactory.decodeResource(this.resources,R.mipmap.ic_launcher))
                .setContentIntent(pendingIntent)
            }
            //Finally calling the notify method to show notifications
            //keeping notification id as 9191 for easier reference
            notificationManager.notify(9191,builder.build())
        }
    }
}
```

Vibrating the phone

Vibrating Android device is found effective when you wanted to get notified on the incoming connection in various cases like, user cannot hear a ringtone or he wants to get notified silently. It’s a very basic feature found in all mobile phones.

Step 1: Grant Permission (AndroidManifest.xml)

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

Step 2: Check if the device can vibrate

```
private fun canVibrate() : Boolean {
    val vibrator = getSystemService(Context.VIBRATOR_SERVICE) as Vibrator
    if(vibrator.hasVibrator()) {return true}
    return false
}
```

Step 3: Vibrate the device

```
private fun vibrate(millisecond: Long) {
    val vibrator = getSystemService(Context.VIBRATOR_SERVICE) as Vibrator
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        vibrator.vibrate(VibrationEffect.createOneShot(millisecond,
VibrationEffect.DEFAULT_AMPLITUDE))
    } else {
        vibrator.vibrate(millisecond)
    }
}
```

Vibrate Indefinitely

```
private fun vibrateIndefinitely() {
    val vibrator = getSystemService(Context.VIBRATOR_SERVICE) as Vibrator
    // Start without a delay (0ms)
    // Vibrate duration (100ms)
    // Sleep between vibrations (1000ms)
    val pattern = longArrayOf(0, 100, 1000)
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        vibrator.vibrate(VibrationEffect.createWaveform(pattern, 0))
    } else {
        // 0 = Repeat Indefinitely
        vibrator.vibrate(pattern, 0)
    }
    // To cancel the vibration, in this case ==> Cancel after 60000ms (= 1 minute)
    Timer().schedule(timerTask { vibrator.cancel() }, 60000)
}
```

Vibrate in Patterns

```
private fun vibratePattern() {
    val vibrator = getSystemService(Context.VIBRATOR_SERVICE) as Vibrator
    // First start without a delay, then alternate between vibrate, sleep, vibrate, sleep,...
    val pattern = longArrayOf(0, 250, 500, 250, 500)
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        vibrator.vibrate(VibrationEffect.createWaveform(pattern,
VibrationEffect.DEFAULT_AMPLITUDE))
    } else {
        // -1 indicates to vibrate once, otherwise it will repeat the pattern for x amount of times
        vibrator.vibrate(pattern, -1)
    }
}
```

Blinking the lights

If you have a notification LED flashing, you know that your phone has something to tell you. But you don’t necessarily know what. Light Flow allows you to set different LED colors for different notifications. You could have a yellow light flash when you have a missed call, a green light when you have a text message, a blue light when you have a new email, and a red light when the phone is low on battery and needs to be plugged in. You can control which types of notifications produce a notification light – Light Flow supports notifications from over 550 different apps.

Services

Services in Android is a component that runs in the background without any user interaction or interface. Unlike Activities services does not have any graphical interface. Services run invisibly performing long running tasks - doing Internet look ups, playing music, triggering notifications etc.,
Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them for resource management. The only reason Android will stop a Service prematurely is to provide additional resources for a foreground component usually an Activity. When this happens, your Service can be configured to restart automatically.

Types of Android Services

Foreground Services: Services that notify the user about its ongoing operations are termed as Foreground Services. Users can interact with the service by the notifications provided about the ongoing task. Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.

Background Services: Background services do not require any user intervention. These services do not notify the user about ongoing background tasks and users also cannot access them. The process like schedule syncing of data or storing of data fall under this service.

Bound Services: This type of android service allows the components of the application like activity to bound themselves with it. Bound services perform their task as long as any application component is bound to it. More than one component is allowed to bind themselves with a service at a time. In order to bind an application component with a service bindService() method is used.

Methods of Android Services

Six callback methods are a part of the Service class. We can create, start, stop, destroy a service, or even bind services with application components using these functions. The following are the methods:

onCreate(): This method is called at first whenever a service has to be created. Using this callback method service of an application is created and is ready to perform some task.

onStartCommand(): This method is called whenever startService() method is invoked. This method gets triggered whenever the service starts, and whatever is present inside this method is performed.

onBind(): This method is called whenever bindService() method is invoked. This method helps us to bind the application components with the service created. For the communication between the application component and the service, we need to provide an IBinder Object interface. This method is mandatory and needs to be present in your service class. If you don't need to bind components, then just return null.
onRebind(): This method is required whenever new users get connected to your service. However, this method can only be invoked after the onBind() method.
onUnbind(): This method unbinds all the components bound with the service and disconnects the service's current user.
onDestroy(): This method is used to free up the system resources. Whenever the service has to be destroyed, then the onDestroy() method is called. This method drops the active threads, removes listeners, etc.

Implementation of Android Services
Following are the steps using which you can create your service in Android. We will see you how you can create a music player service that plays your default mobile ringtone for the demonstration.

Step 1: Open your Android Studio.
Step 2: Click on Create New Project.
Step 3: Select Empty Activity and proceed.
Step 4: Enter your application name. In my case, it's "ServiceDemo" Next, select Kotlin from the dropdown. For the API level, select API 22 for now.
Step 5: Now, when your android studio loads, then click on java. Then select the first directory and right-click. Now click on New —> Service. This will open a window asking you the name of your service. Just enter "MyService" and click finish.
Step 6: Now, you can see your service created. By default, you can see the onBind() method is there. Just remove the return statement and keep the return null. Now just see how we coded the MyService.kt file.

```
class MyService : Service()
{
   private lateinit var myPlayer: MediaPlayer;
   override fun onBind(intent: Intent): IBinder?
   {
      //returning null as we are not binding any component to the service
      return null;
   }
   override fun onCreate()
   {
      super.onCreate()
      Toast.makeText(this, "Your Service is created", Toast.LENGTH_LONG).show();
   }
   override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int
   {
      //playing your phone ringtone
      //whenever you start this service
      myPlayer = MediaPlayer.create(this, Settings.System.DEFAULT_RINGTONE_URI)
      // setting the player on loop
      // so that the service keeps running
      // until we stop
      myPlayer.setLooping(true)
```

```
        // starting the player service
        myPlayer.start()
        Toast.makeText(this, "Your Service started", Toast.LENGTH_LONG).show();
        // provides the status of our service
        return START_STICKY
    }
    override fun onDestroy() {
        super.onDestroy()
        //whenever the service is destroyed
        //we need to stop our media player
        myPlayer.stop();
        Toast.makeText(this, "Your Service stopped", Toast.LENGTH_LONG).show();
    }
}
```

Step 7: Now go to the app –> res –> layout –> activity_main.xml and add two buttons one to start the service and another to stop the service. You can notice the activity_main.xml file below.

```
<LinearLayout
    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"
    tools:context=".MainActivity"
    android:orientation="vertical">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="TechVidvan"
        android:fontFamily="sans-serif"
        android:textColor="@color/purple_500"
        android:textStyle="bold"
        android:gravity="center"
        android:textSize="30sp"
        />
    <Button
        android:id="@+id/startServiceButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start your Service"
        android:textSize="20sp"
        android:padding="10dp"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="5dp"
        android:layout_marginTop="100dp"
        />
    <Button
        android:id="@+id/stopServiceButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop your Service"
        android:textSize="20sp"
        android:padding="10dp"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="5dp"
        android:layout_marginTop="50dp"
        />
</LinearLayout>
```

Step 8: After coding the activity_main.xml, now come to the MainActivity.kt file.

```
class MainActivity : AppCompatActivity()
{
    private lateinit var startMusic:Button;
    private lateinit var stopMusic:Button;
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //Binding the start/stop buttons of the layout to our variables
        startMusic = findViewById(R.id.startServiceButton)
        stopMusic = findViewById(R.id.stopServiceButton)
        //Adding listener to each button
        startMusic.setOnClickListener(View.OnClickListener
        {
            //starting the music player service
            //when start service button is pressed
            startService(Intent(this, MyService::class.java))
        })
        stopMusic.setOnClickListener(View.OnClickListener
        {
            //stop the music player service
            //when stop service button is pressed
            stopService(Intent(this, MyService::class.java))
        })
    }
}
```

Application development using JSON

JSON being lightweight and structured, is better than XML and is mainly used these days. Therefore, it becomes essential to understand the steps we need to follow to parse JSON data and get the desired information.

JSON: JSON (JavaScript Object notation) is a lightweight, straightforward, and structured format to interchange data between the application and the server. JSON is used to transfer data (textual information) from server to application or vice versa. Parsing the JSON data helps us in segregating required information from the JSON data.

Structure of JSON: JSON has a structured format that makes developers easily understand and parse it. JSON can start with any of the following brackets:

[ ]: The square brackets are used to define a JSON Array.

{ }: The curly brackets are used to denote a JSON Object.

JSON can have the following structures:

JSON Object: As discussed before, JSON object starts and ends with curly brackets. JSON Objects consist of several key-value pairs, JSON Arrays, and even other JSON Objects.
Below is an example of a JSON Object.

```
{
  "Name": "KSC",
  "Author": “Prof. Nandan Pandya”,
  "Languages": [“C++”, “Java”, “Kotlin”, “Dart”, “Python”, “GO”, “JavaScript”]
  “Frameworks”:
   {
      “Python”: [“Django”, “Flask”],
      “JavaScript”: [“React”, “Ionic”, “Angular”]
   }
}
```

JSON Array: JSON Array starts and ends with square brackets consisting of a list of several values. These values can be of any type String, Integer, Float, or Boolean. Sometimes these values can also be JSON objects.

```
[“CodeChef”,  “CodeForces”,  “AtCoder”,  “TopCoder”, “HackerRank”]
```

Features of JSON
JSON provides several features which make it most widely used among developers. Some of the features of JSON are as follows:

- It is text-based and easy to understand
- It is language-independent, and no matter which language you use, you can use JSON to exchange data between your application and server
- JSON is lightweight and structured and It is scalable
- It allows you to interchange data between application and server
- JSON can carry many types of data

JSON DataTypes

JSON currently allows six types of data which are as follows:

String: Using JSON, you can easily exchange String constants. Strings are a sequence of characters that is usually enclosed within double quotes(“ ”).
Numbers: Using JSON, you can exchange numbers(both integers and floating points).
Boolean: Using JSON, you can exchange Boolean data also, which includes either True or False.
Null: Sometimes, you may need to define a key or object as empty. In such circumstances, you can mark that object as null.
Object: JSON can contain several JSON objects inside it.
Array: JSON can contain several JSON arrays inside it.

Step 1: Start your Android Studio and click on Create New Project. Name it as JSONDemo.
Step 2: Now open your activity_main.xml file and paste the below code there.
Step 3: Lastly, come back to your MainActivity.kt file and paste the below code there.

```
class MainActivity : AppCompatActivity()
{
   lateinit var arrayAdapter: ArrayAdapter<*>
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
```

```
        //binding the ListView with the variable
        val listview: ListView = findViewById(R.id.studentsList)
        //fetching the JSON data in form of string
        val json_stu: String = getRawJSON()
        try {
            var name_list = ArrayList<String>()
            //converting the raw json to JSON Object
            val jsObj = JSONObject(json_stu)
            //fetching the students JSON Array
            val jsArray = jsObj.getJSONArray("students")
            for (i in 0 until jsArray.length()) {
                val obj = jsArray.getJSONObject(i)
                //adding names of the students in the name_list
                name_list.add(obj.getString("name"))
            }
            //creating the array adapter object
            arrayAdapter = ArrayAdapter(this,
                android.R.layout.simple_list_item_1, name_list)
            //providing the array adapter to the list view
            listview.adapter = arrayAdapter
        } catch (ex: JSONException) {
            Log.e("JsonParser ", "Exception", ex)
        }
    }
    private fun getRawJSON(): String {
        return "{\n" +
                "  \"students\": [\n" +
                "    {\n" +
                "      \"name\": \"Student1\"\n" +
                "    },\n" +
                "    {\n" +
                "      \"name\": \"Student2\"\n" +
                "    },\n" +
                "    {\n" +
                "      \"name\": \"Student3\"\n" +
                "    },\n" +
                "    {\n" +
                "      \"name\": \"Student4\"\n" +
                "    },\n" +
                "    {\n" +
                "      \"name\": \" Student5\"\n" +
                "    },\n" +
                "    {\n" +
                "      \"name\": \" Student6\"\n" +
                "    }\n" +
                "  ]\n" +
                "}"
    }
}
```

Publish android application

Step 1: Create a Google Developer account

The creation process includes signing the Google Play Developer distribution agreement, adding some personal information, and paying a one-time registration fee of $25. There is nothing complicated. Just follow the instructions.

Step 2: Add a Merchant Account

If you plan to sell paid apps or in-app purchases, you have to create a Google Merchant Account. There you can manage app sales and your monthly payouts, as well as analyze sales reports.

Step 3: Prepare the Documents

Paperwork always requires much effort, especially when it comes to any kind of legal documents. Based on our experience, we highly recommend starting to prepare the End User License Agreement (EULA) and Privacy Policy in advance.

Step 4: Study Google Developer Policies

Now it's time to make sure that every feature you will implement in the app is aligned with the Google Developer Policies. These documents explain how apps need to be developed, updated, and promoted to support the store's high-quality standards. If Google decides that your product violates some policy chapters, it may be rejected, blocked, or even deleted from the Play Store. Besides, numerous and repetitive violations may lead to the developer account termination.

Step 5: Technical Requirements

You went through the development process, endless testing, and bug fixing, and finally, the "X-day" comes. Before moving on to the upload process, you need to check the following things:

- Unique Bundle ID
- Signed App Release With a Signing Certificate
- The App Size
- The File Format

Step 6: Creating the App on the Google Console

Let's create a new app in your Developer Account:

- Reach to All applications tab in the menu
- Now select Create Application
- Choose the app's default language from the drop-down menu
- Add a brief app description (you can change it later)
- Tap on Create

Step 7: Store Listing

It contains the most important information useful for app store optimization (ASO) and gives the users more details about your app before downloading. The mandatory sections are marked with *.

Step 8: Content Rating

In order not to be marked as an Unrated App (that may lead to app removal), pass a rating questionnaire. You can easily find this section on the left-side menu.The information provided in the questionnaire must be accurate. Any misrepresentation of your app's content might lead to suspension or removal of the Play Store account.

Step 9: Pricing the Application

- Whether your app is free or paid
- Where the app will be available (just choose the countries from the list)
- Whether your app will be available only on the specific devices
- Whether the app has sensitive content and is not suitable for children under the age of 13
- Whether your app contains ads

Step 10: Upload APK and Send for Review

you are ready to upload your app file. That's the most exciting moment ever. The app will be released right after it passes the review. Usually, it takes up to 2 days. Google says the review process could take up to 7 days or even longer. Once the app is reviewed, you'll receive a notification on Google Console Dashboard.