2022

Android Development FUNdamentals

CRASH COURSE FOR BEGINNERS

MOHAMMAD FARIZSHAH BIN ISMAIL KAMIL

FARIZGASKIN.COM | farizgaskin@gmail.com | 0162780252 | 49-3, Jalan Tasik Selatan 8, 57000 KL

Table of Contents

1 Basics *4*

1.1 Introduction to Android Development 4

1.2 Android Architecture 8

2 Setup and Configuration *11*

2.1 How to Install and Set up Android Studio on Windows? 11

2.2 How to Run the Android App on a Real Device? 18

3 Android Studio *19*

3.1 Android Studio Main Window 19

3.2 Different Types of Activities in Android Studio 21

3.3 How to Create/Start a New Project in Android Studio? 31

4 Views *34*

4.1 TextView 34

4.2 Copying to Clipboard 37

4.3 How to Add a TextView with Rounded Corner 41

5 EditText *43*

5.1 Android EditText in Kotlin 43

5.2 How to add Mask to an EditText in Android 46

6 ImageView *49*

6.1 ImageView in Android 49

6.2 How to Create Circular ImageView in Android using CardView? 53

7 ListView *55*

7.1 Android ListView in Kotlin 55

8 ScrollView *57*

8.1 HorizontalScrollView in Kotlin 57

9 CardView *60*

9.1 CardView in Android 60

9.2 How to create an Expandable CardView in Android 62

10 GridView *69*

10.1 GridView in Android 69

11 Other Views *74*

11.1 WebView 74

12 Buttons *76*

12.1 Button in Kotlin 76

12.2 RadioButton in Kotlin 78

12.3 CheckBox in Kotlin **86**

13 Intent and Intent Filters ***90***

13.1 Implicit and Explicit Intents **90**

13.1.1 Implicit Intent 90

13.1.2 Explicit Intent 94

13.2 How to send message on Whatsapp **102**

14 Toast ***105***

14.1 Toast in Kotlin **105**

15 RecyclerView ***109***

15.1 RecyclerView in Android **109**

15.2 Pull to Refresh **114**

16 Fragments ***119***

16.1 Introduction to Fragments **119**

16.2 Swipe Navigation **121**

17 Adapters ***126***

17.1 ArrayAdapter **126**

18 Spinner ***128***

18.1 Spinner in Kotlin **128**

20 Notifications ***136***

20.1 Notifications in Android **136**

21 Menu ***140***

22 Date and Time ***144***

22.1 DatePicker in Kotlin **144**

22.2 TimePicker in Kotlin **148**

23 Material Design ***153***

23.1 Responsive UI Design **153**

23.2 Material Design EditText **157**

24 Bars ***163***

24.2 ProgressBar in Kotlin **169**

25 Google Maps ***173***

25.1 How to Generate API Key **173**

25.2 Add Custom Marker **177**

25.3 Add Multiple Markers **180**

26 Chart ***183***

26.1 How to add a Pie Chart **183**

27 Database ***194***

27.1 Room DB – How to perform CRUD 194

28 Storage 211

28.1 Shared Preferences 211

29 JSON 214

29.1 JSON Parsing 214

30 App Publish 218

30.1 How to publish on Google Play Store 218

Conclusion 230

1 Basics

1.1 Introduction to Android Development

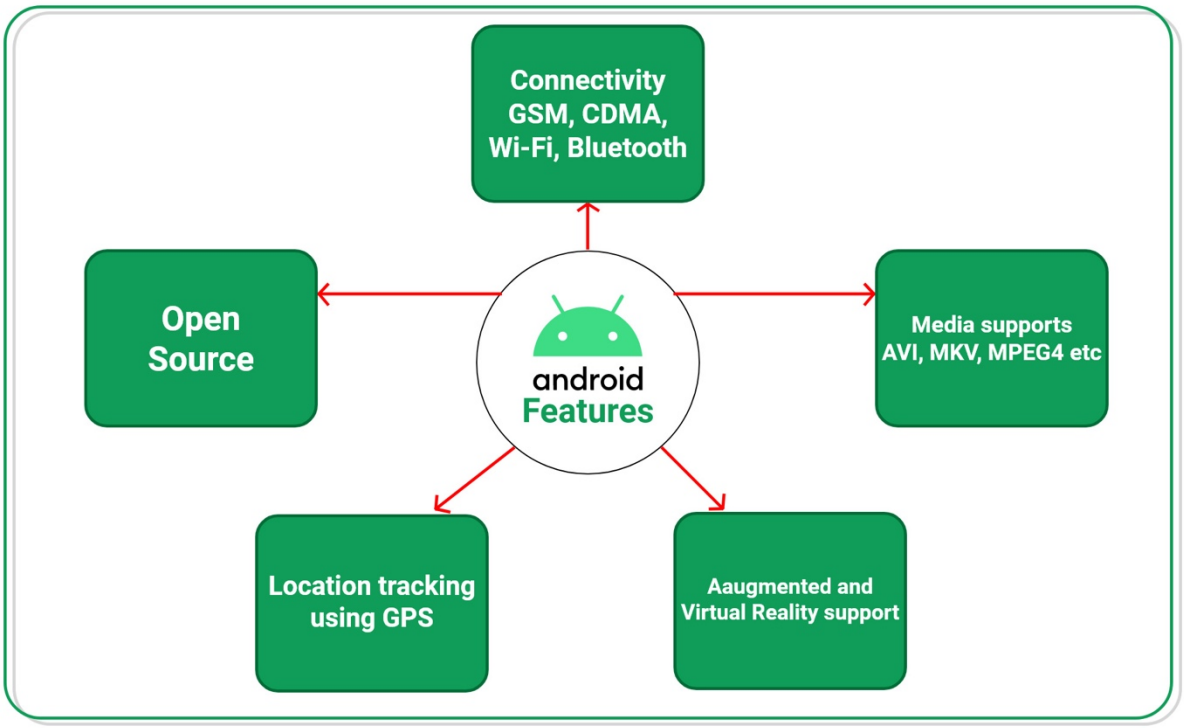
The Android operating system is the largest installed base among various mobile platforms across the globe. Hundreds of millions of mobile devices are powered by Android in more than 190 countries of the world. It conquered around 75% of the global market share by the end of 2020, and this trend is growing bigger every other day. The company named Open Handset Alliance developed Android for the first time that is based on the modified version of the Linux kernel and other open-source software. Google sponsored the project at initial stages and in the year 2005, it acquired the whole company.

In September 2008, the first Android-powered device launched in the market. Android dominates the mobile OS industry because of the long list of features it provides. It's user-friendly, has huge community support, provides a greater extent of customization, and a large number of companies build Android-compatible smartphones. As a result, the market observes a sharp increase in the demand for developing Android mobile applications, and with that companies need smart developers with the right skill set.

At first, the purpose of Android was thought of as a mobile operating system. However, with the advancement of code libraries and its popularity among developers of the divergent domain, Android becomes an absolute set of software for all devices like tablets, wearables, set-top boxes, smart TVs, notebooks, etc.

Features of Android

Android is a powerful open-source operating system that open-source provides immense features and some of these are listed below.



- Android Open Source Project so we can customize the OS based on our requirements.
- Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
- Using wifi technology we can pair with other devices while playing games or using other applications.
- It contains multiple APIs to support location-tracking services such as GPS.
- We can manage all data storage related activities by using the file manager.
- It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
- It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It supports multimedia hardware control to perform playback or recording using a camera and microphone.
- Android has an integrated open-source WebKit layout based web browser to support User Interface like HTML5, CSS3.
- Android supports multi-tasking means we can run multiple applications at a time and can switch in between them.
- It provides support for virtual reality or 2D/3D Graphics

Android Versions

Google launched the first version of the Android platform on Nov 5, 2007. Since then, Google released a lot of android versions such as Apple Pie, Banana Bread, Cupcake, Donut, Éclair, Froyo, Gingerbread, Jellybeans, Kitkat, Lollipop, marshmallow, Nougat, Oreo, etc. with extra functionalities and new features.

The following table shows the version details of android which is released by Google from 2007 to date.

Code Name	Version	API Level	Release Date
Apple Pie	1.0	1	Sep 23, 2008
Banana Bread	1.1	2	Feb 9, 2009
Cupcake	1.5	3	Apr 30, 2009
Donut	1.6	4	Sep 15, 2009
Éclair	2.0 – 2.1	5 – 7	Oct 26, 2009
Froyo	2.2 – 2.2.3	8	May 10, 2010
Gingerbread	2.3 – 2.3.4	9 – 10	Dec 6, 2010
Honeycomb	3.0.x – 3.2.x	11 – 13	Feb 22, 2011
Ice Cream Sandwich	4.0 – 4.0.4	14 – 15	Oct 18, 2011
Jelly Bean	4.1 – 4.1.2	16 – 18	Jul 9, 2012
Kitkat	4.4 – 4.4.4	19	Jul 9, 2012
Lollipop	5.0 – 5.1	21 – 22	Oct 17, 2014
Marshmallow	6.0 – 6.0.1	23	Oct 5, 2015
Nougat	7.0 – 7.1	24 – 25	Aug 22, 2016
Oreo	8.0	26	Aug 21, 2017
Pie	9.0	27	Aug 6, 2018
Android Q	10.0	29	Sep 3, 2019
Android 11	11.0	30	Sep 8, 2020

Programming Languages used in Developing Android Applications

1. Java
2. Kotlin

Developing the Android Application using Kotlin is preferred by Google, as Kotlin is made an official language for Android Development, which is developed and maintained by JetBrains. Previously before the Java is considered the official language for Android Development. Kotlin is made official for Android Development in Google I/O 2017.

Advantages of Android Development

- The Android is an open-source Operating system and hence possesses a vast community for support.
- The design of the Android Application has guidelines from Google, which becomes easier for developers to produce more intuitive user applications.
- Fragmentation gives more power to Android Applications. This means the application can run two activities on a single screen.
- Releasing the Android application in the Google play store is easier when it is compared to other platforms.

Disadvantages of Android Development

Fragmentation provides a very intuitive approach for user experience but it has some drawbacks, where the development team needs time to adjust with the various screen sizes of mobile smartphones that are now available in the market and invoke the particular features in the application.
The Android devices might vary broadly. So the testing of the application becomes more difficult.
As the development and testing consume more time, the cost of the application may increase, depending on the application's complexity and features.

1.2 Android Architecture

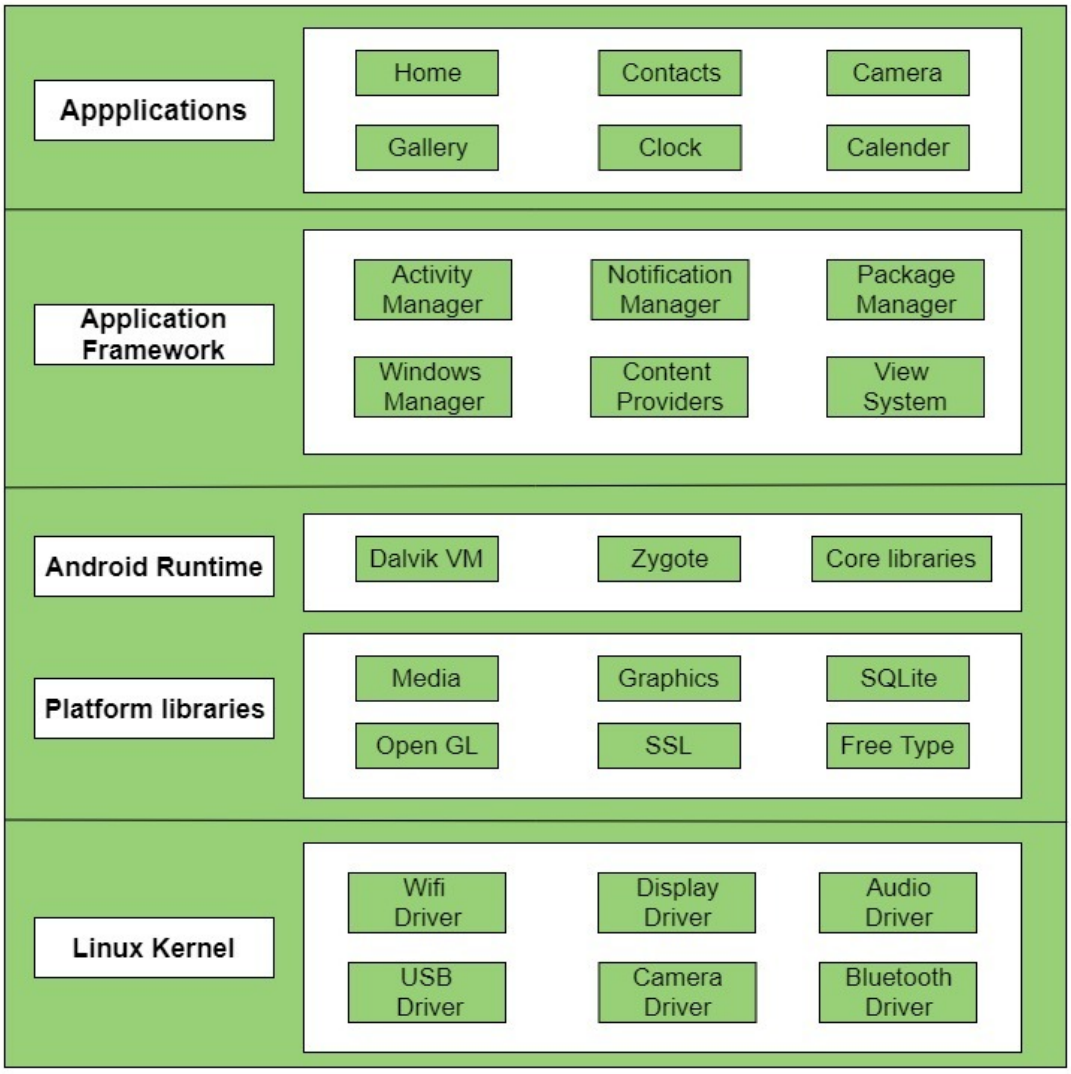
Android architecture contains different number of components to support any android device needs. Android software contains an open-source Linux Kernel having collection of number of C/C++ libraries which are exposed through an application framework services.

Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.

The main components of android architecture are as follows:

- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

Pictorial representation of android architecture with several main components and their sub components –



Applications

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.
It runs within the Android run time with the help of the classes and services provided by the application framework.

Application Framework

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.

It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

Application runtime

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), Dalvik Virtual Machine (DVM) is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

Platform Libraries

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

- Media library provides support to play and record an audio and video formats.
- Surface manager responsible for managing access to the display subsystem.
- SGL and OpenGL both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- SQLite provides database support and FreeType provides font support.
- Web-Kit This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- SSL (Secure Sockets Layer) is security technology to establish an encrypted link between a web server and a web browser.

Linux Kernel

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime.

The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.

The features of Linux kernel are:

- Security: The Linux kernel handles the security between the application and the system.
- Memory Management: It efficiently handles the memory management thereby providing the freedom to develop our apps.
- Process Management: It manages the process well, allocates resources to processes whenever they need them.
- Network Stack: It effectively handles the network communication.
- Driver Model: It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.

2 Setup and Configuration

2.1 How to Install and Set up Android Studio on Windows?

Android Studio is the official IDE (Integrated Development Environment) for Android app development and it is based on JetBrains' IntelliJ IDEA software. Android Studio provides many excellent features that enhance productivity when building Android apps, such as:

- A blended environment where one can develop for all Android devices
- Apply Changes to push code and resource changes to the running app without restarting the app
- A flexible Gradle-based build system
- A fast and feature-rich emulator
- GitHub and Code template integration to assist you to develop common app features and import sample code
- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine, and many more.

System Requirements

Microsoft Windows 7/8/10 (32-bit or 64-bit)
4 GB RAM minimum, 8 GB RAM recommended (plus 1 GB for the Android Emulator)
2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE plus 1.5 GB for Android SDK and emulator system image)
1280 x 800 minimum screen resolution

Installation Guide

Step 1: Head over to https://developer.android.com/studio/#downloads to get the Android Studio executable or zip file.

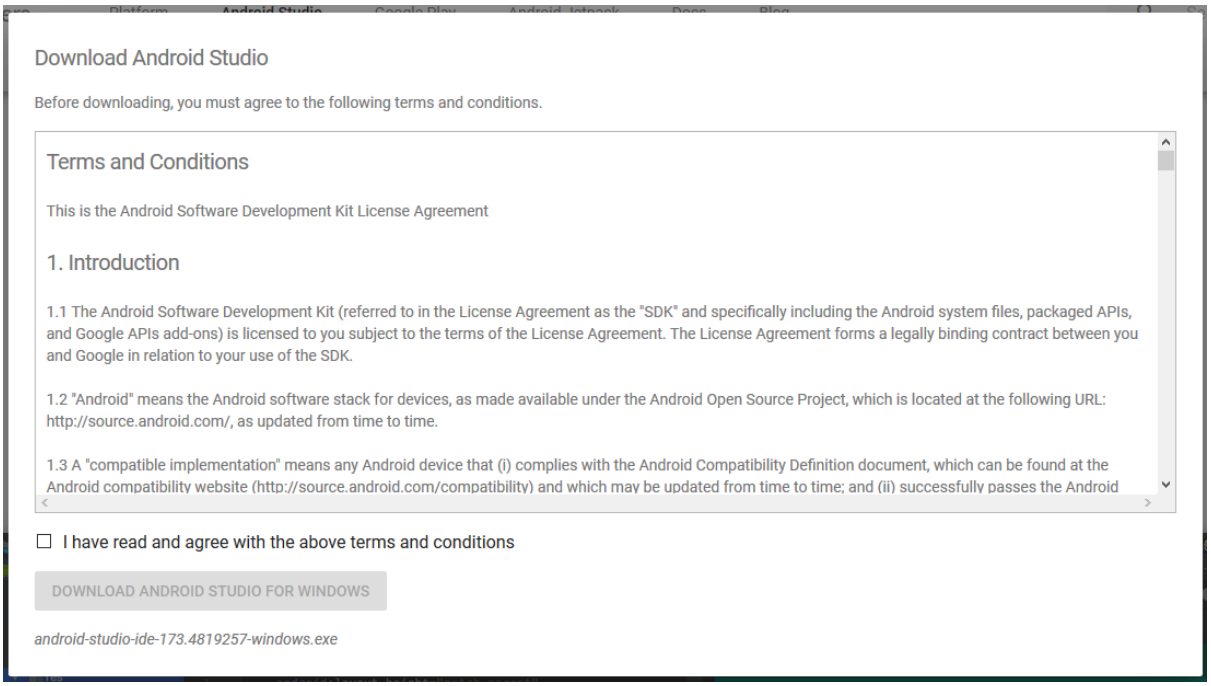
Step 2: Click on the Download Android Studio Button.

Android Studio provides the fastest tools for building apps on every type of Android device.

DOWNLOAD ANDROID STUDIO

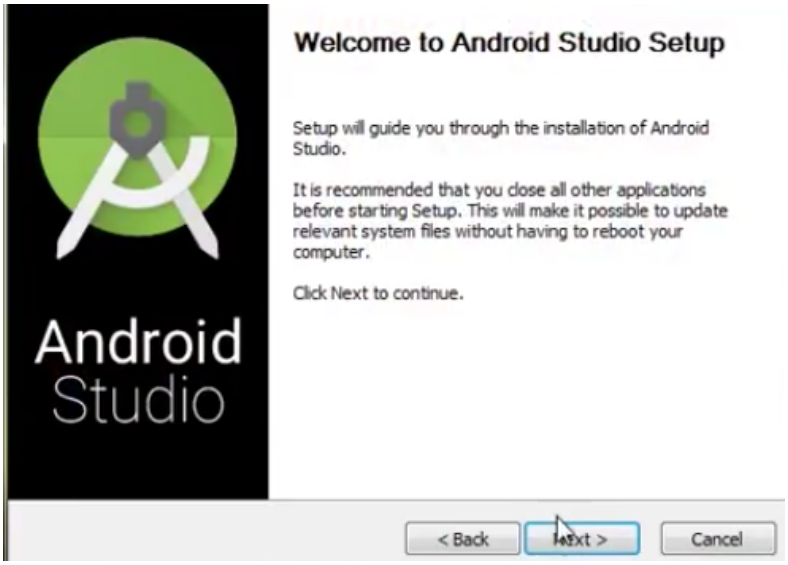
4.1.3 for Windows 64-bit (896 MiB)

Click on the “I have read and agree with the above terms and conditions” checkbox followed by the download button.



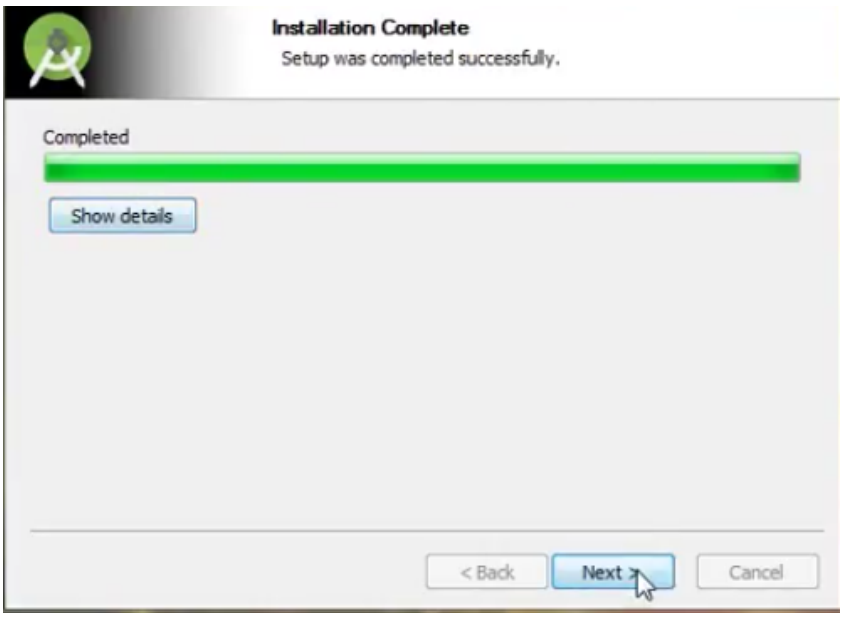
Click on the Save file button in the appeared prompt box and the file will start downloading.

Step 3: After the downloading has finished, open the file from downloads and run it. It will prompt the following dialog box.



Click on next. In the next prompt, it’ll ask for a path for installation. Choose a path and hit next.

Step 4: It will start the installation, and once it is completed, it will be like the image shown below.



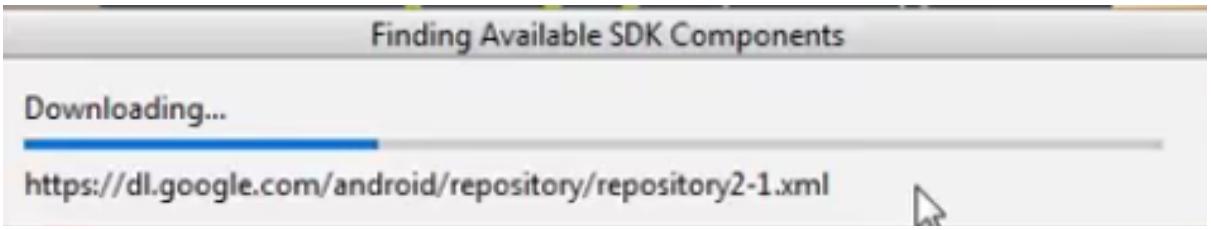
Click on next.

Step 5: Once “Finish” is clicked, it will ask whether the previous settings need to be imported [if the android studio had been installed earlier], or not. It is better to choose the ‘Don’t import Settings option’

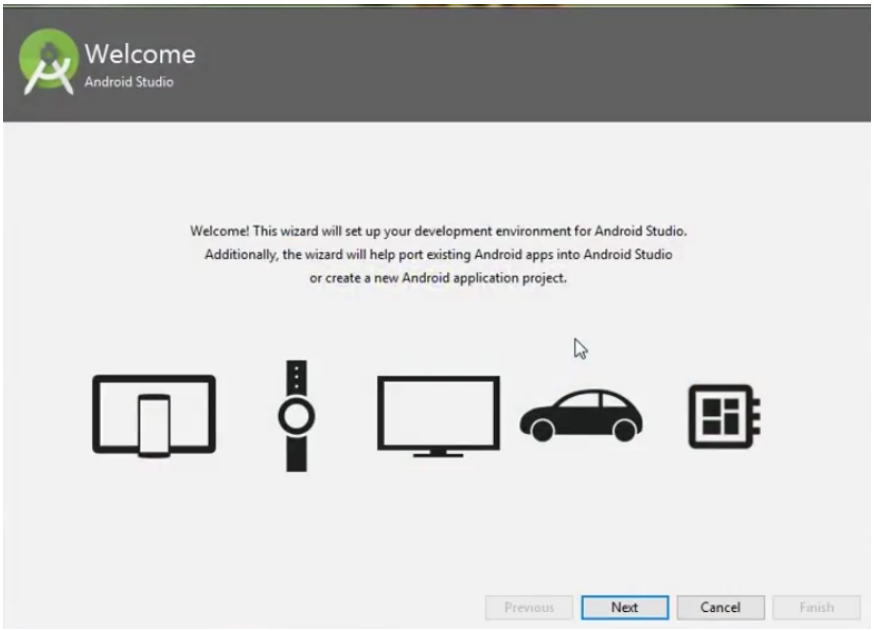
Click the OK button.

Step 6: This will start the Android Studio.

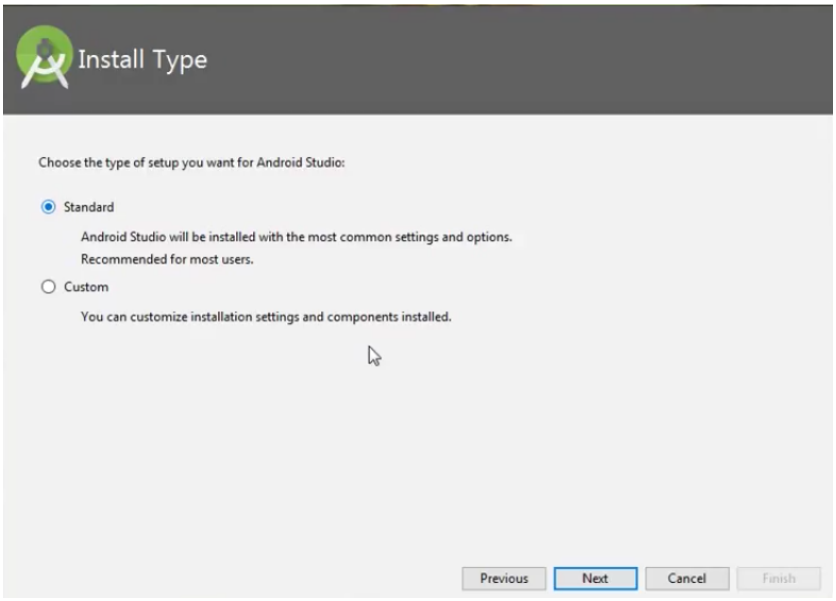
Meanwhile, it will be finding the available SDK components.



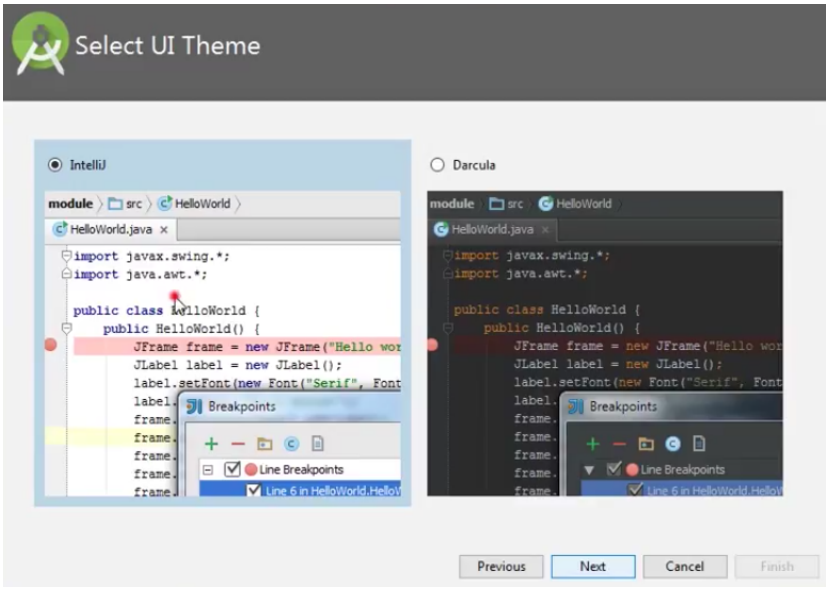
Step 7: After it has found the SDK components, it will redirect to the Welcome dialog box.



Click on Next

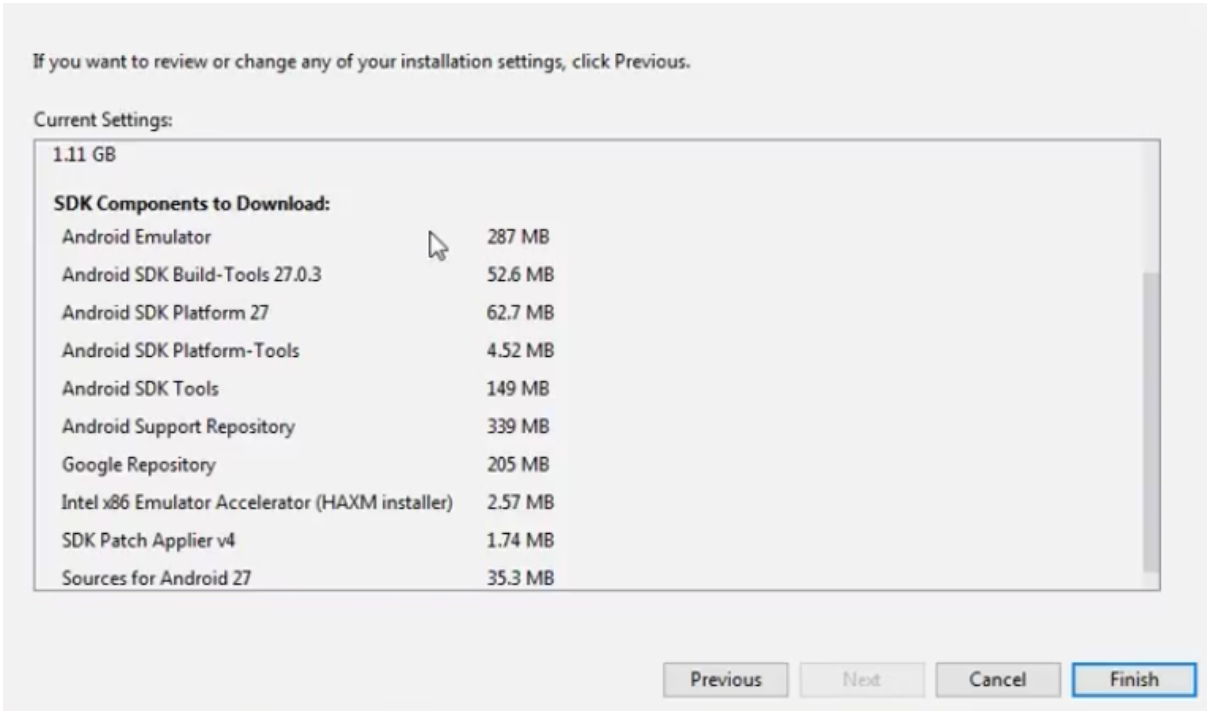


Choose Standard and click on Next. Now choose the theme, whether the Light theme or the Dark one. The light one is called the IntelliJ theme whereas the dark theme is called Darcula. Choose as required.

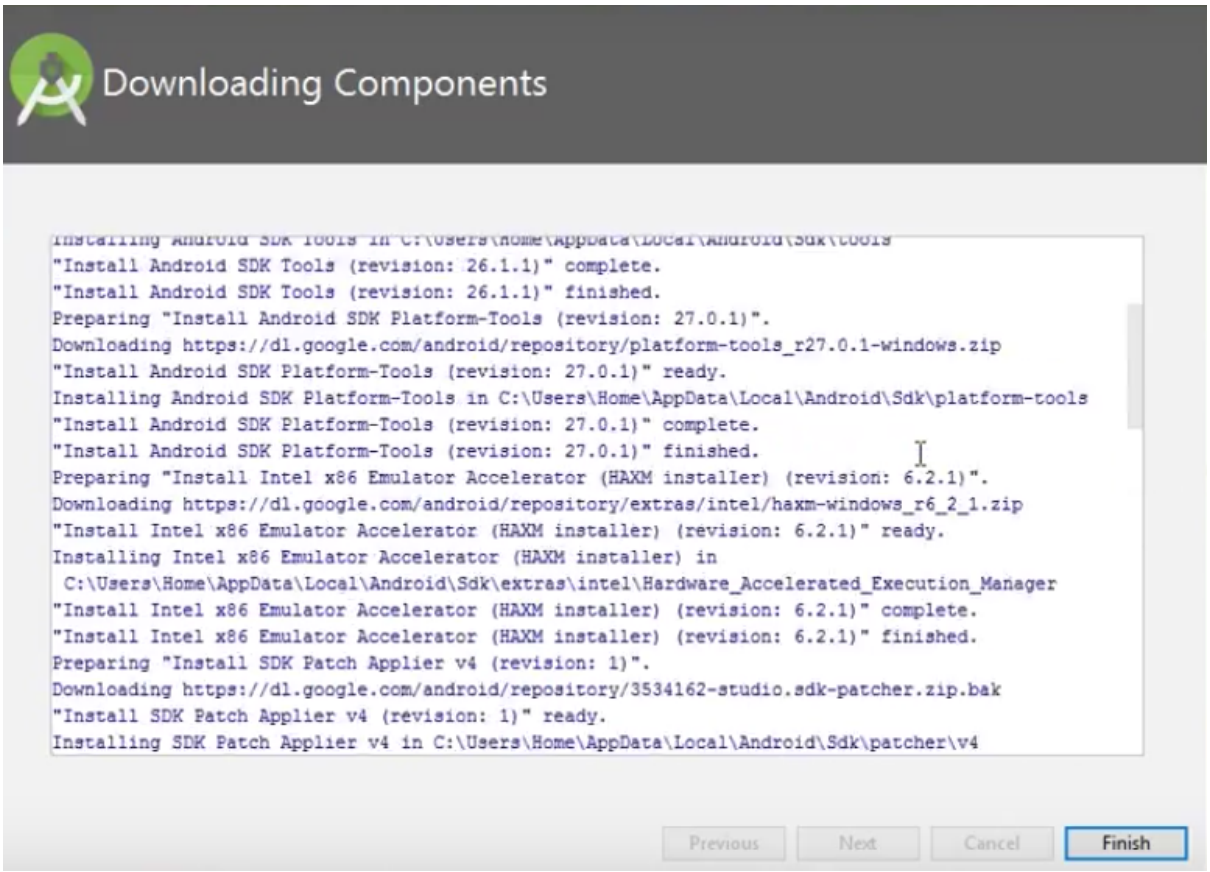


Click on the Next button.

Step 8: Now it is time to download the SDK components.

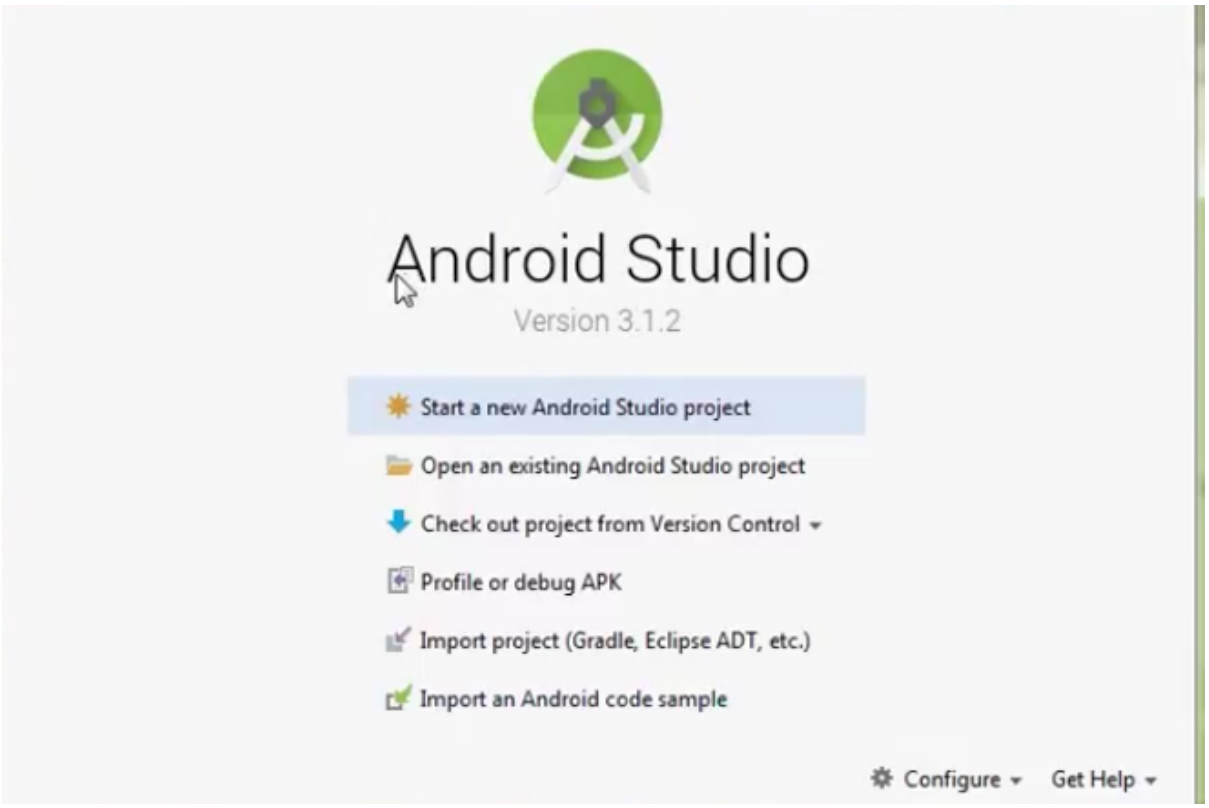


Click on Finish. Components begin to download let it complete.



The Android Studio has been successfully configured. Now it's time to launch and build apps. Click on the Finish button to launch it.

Step 9: Click on Start a new Android Studio project to build a new app.



2.2 How to Run the Android App on a Real Device?

The time comes when the Android Studio project is ready and you want to test that application. One can test the application by running the application which can be done in two ways.

- By running the app on an Android Virtual Device(AVD), and
- By running the app on a real device

So in this article, we are going to discuss how to run the Android app on a real device. So before running the app on a real device we have to set up the real device first. Set up your device as follows:

Steps for Running the Android App on a Real Device

Step 1: In the physical android device goto Settings -> About phone and then find the Build Number Option.

Step 2: Then one needs to tap the Build Number option fast until it shows, “No need, you are already a developer” as a Toast message.

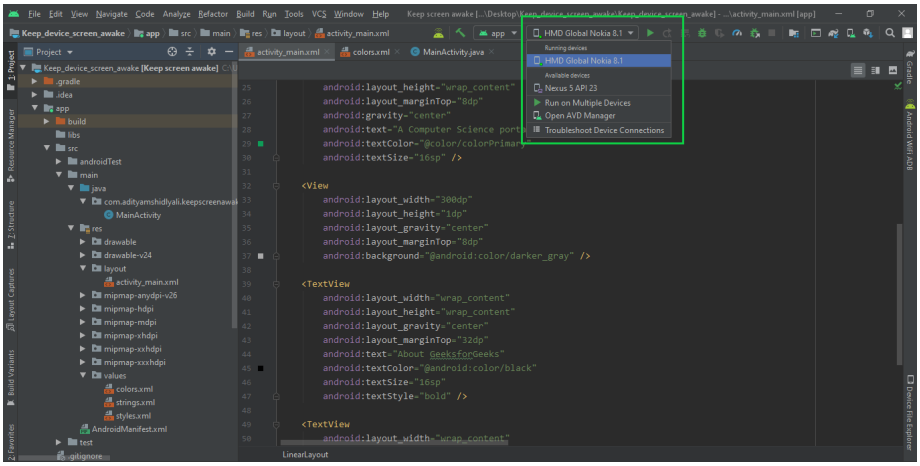
Step 3: Go back to the Settings menu and search for the USB Debugging. And open the USB Debugging option.

Step 4: Then find the USB Debugging option and enable it. One alert dialogue pops up and you need to click “OK” and make sure that the developer option should be kept “on”.

Step 5: Now open the android studio project and then connect the device to the PC using a suitable USB cable and if there are no USB drivers installed in your PC of your android device then you can google search and download and install the USB driver of the specific device model number.

Step 6: As soon as you connect the device, in the physical device, it asks to allow USB debugging for the PC you connected. Then you need to check “Always allow this computer”, and hit “Allow”.

Step 7: Now you can see the device name under the Running devices list in Android Studio IDE. Select your device and then click on the Run Button, and the application you developed in Android Studio will be up and running. Following is the symbol of the Run button.



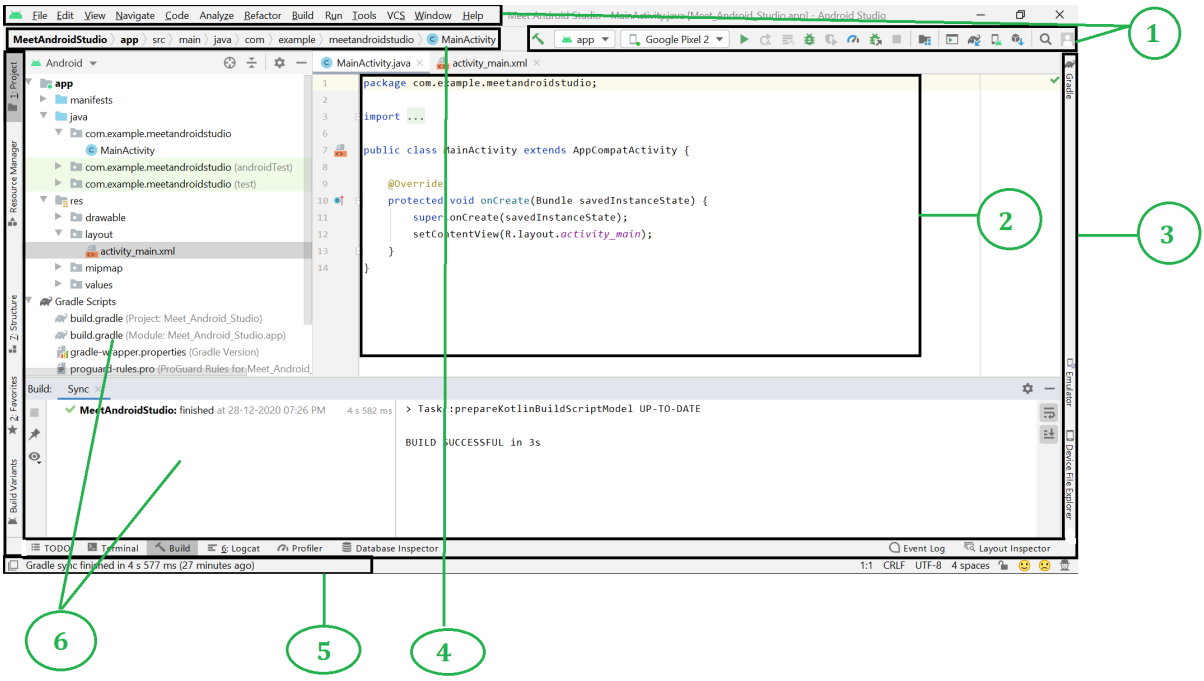
3 Android Studio

3.1 Android Studio Main Window

Android Studio is the official IDE (Integrated Development Environment) for Android app development and it is based on JetBrains' IntelliJ IDEA software. Android Studio provides many excellent features that enhance productivity when building Android apps, such as:

- A blended environment where one can develop for all Android devices
- Apply Changes to push code and resource changes to the running app without restarting the app
- A flexible Gradle-based build system
- A fast and feature-rich emulator
- GitHub and Code template integration to assist you to develop common app features and import sample code
- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine and many more.

Before developing an android application in the android studio the developer must have knowledge of its tool first. So in this article, we are going to discuss the android studio main window. The Android studio main window is consists of several logical areas. Let's discuss each area in detail. Below is the main window of Android Studio 4.1.1.



1. Toolbar

One can find the Toolbar section at the top of the android studio. It contains a wide range of functions including creating a new project which is inside the File Section, Reformat your code which is inside the Code Section, Rebuild your project which is inside the Build section, running your android app which is inside the Run section, and many more. In fact, the Toolbar is the most important part of the android studio.

2. Editor Window

In this window, the users can create, write, and modify their code. This editor window changes depending on the current file type. For example, if you are viewing a layout file(Here activity_main.xml file) then the editor displays the layout editor. Similarly, if you are writing the backend code then the editor displays the Java/Kotlin file (Here MainActivity.java file) depending upon the language you have chosen during the creation of the project.

3. Tool Window Bar

The tool window bar operates around the outside of the IDE window and includes the buttons that allow users to expand or collapse individual tool windows. For example, in the above diagram, we have expanded the Project and the Build button.

4. Navigation Bar

The navigation bar helps the users to navigate throughout the project and open files for editing. It gives a more compressed view of the structure visible in the Project window. For example in the above diagram when we click on the MainActivity.java file in the editor window section then the navigation bar shows us where this file present inside the android studio. (Here app > src > main > java > com > example > meetandroidstudio > MainActivity). Thus it helps us to navigate and modify the required file easily.

5. Status Bar

The status bar displays the current status of the project and the IDE itself, as well as any warnings or messages during the execution of the project.

6. Tool Windows

The tool windows give access to specific tasks like project management, search, version control, and more. It is strongly related to the Tool Window Bar section. When you expand the Tool Window Bar then they are visible inside the Tool Windows section. So the user can also expand and collapse them.

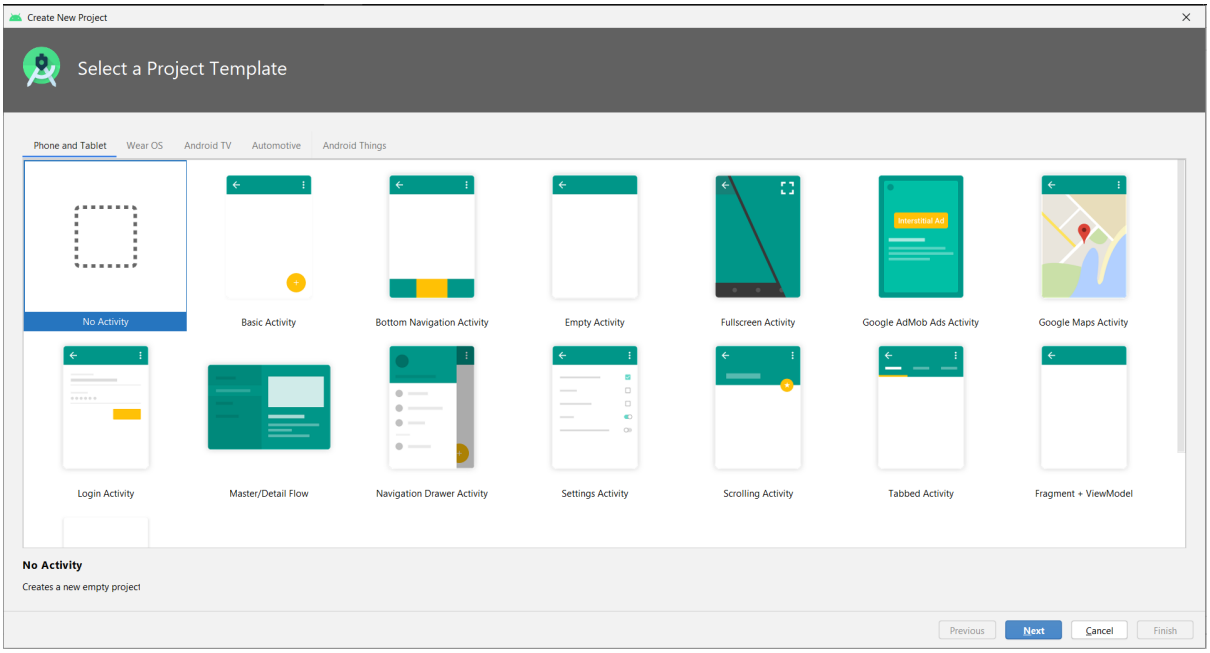
3.2 Different Types of Activities in Android Studio

Android Studio is the official IDE (Integrated Development Environment) for Android application development and it is based on JetBrains' IntelliJ IDEA software. Android Studio provides many excellent features that enhance productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A blended environment where one can develop for all Android devices
- Apply Changes to push code and resource changes to the running app without restarting the app
- GitHub and Code template integration to assist you to develop common app features and import sample code
- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine and many more.

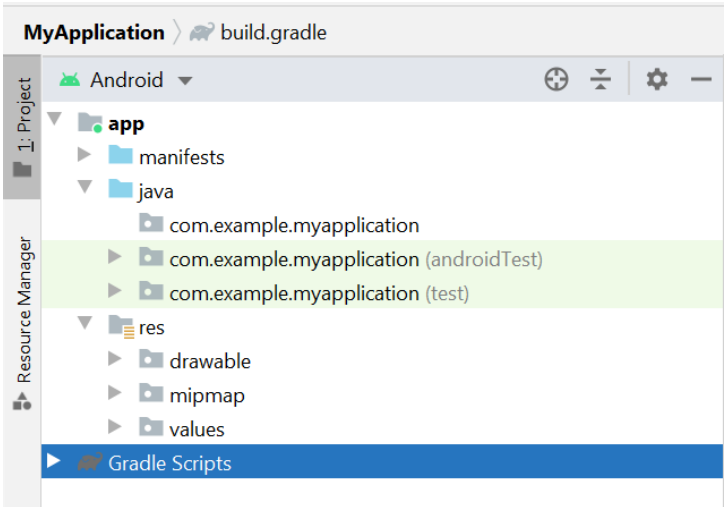
Generally, when a developer wants to create a new project in the android studio he/she needs to select a project template which is consisting of many activities as shown in the below image. (Considering that the developer developing the android app for phone and tablet). So in this article, we are going to discuss what do these activities mean in brief. Here is the list of activities:

1. No Activity
2. Basic Activity
3. Bottom Navigation Activity
4. Empty Activity
5. Fullscreen Activity
6. Google Admob Ads Activity
7. Google Maps Activity
8. Login Activity
9. Master/Detail Flow
10. Navigation Drawer Activity
11. Settings Activity
12. Scrolling Activity
13. Tabbed Activity
14. Fragment + ViewModel
15. Native C++



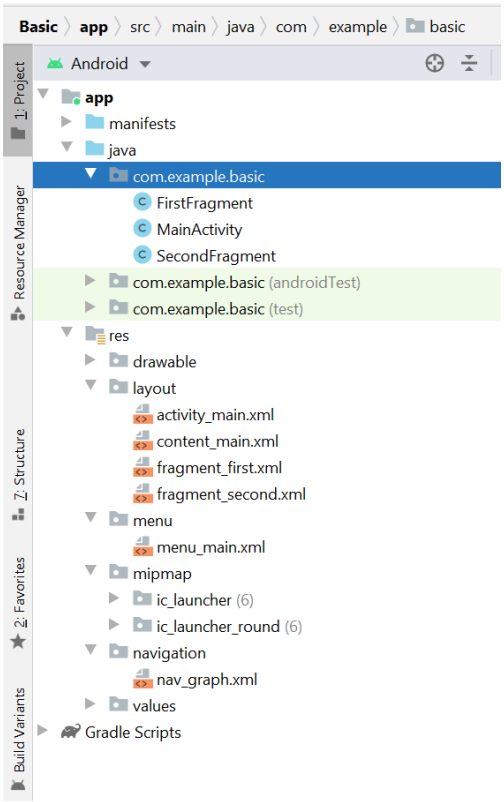
(1) No Activity

As the name suggests No Activity means creating a new empty project. When the developer selects this activity there will be neither an XML file nor a Java/Kotlin file. No files are automatically generated when you select No Activity. The project structure will look like the following:



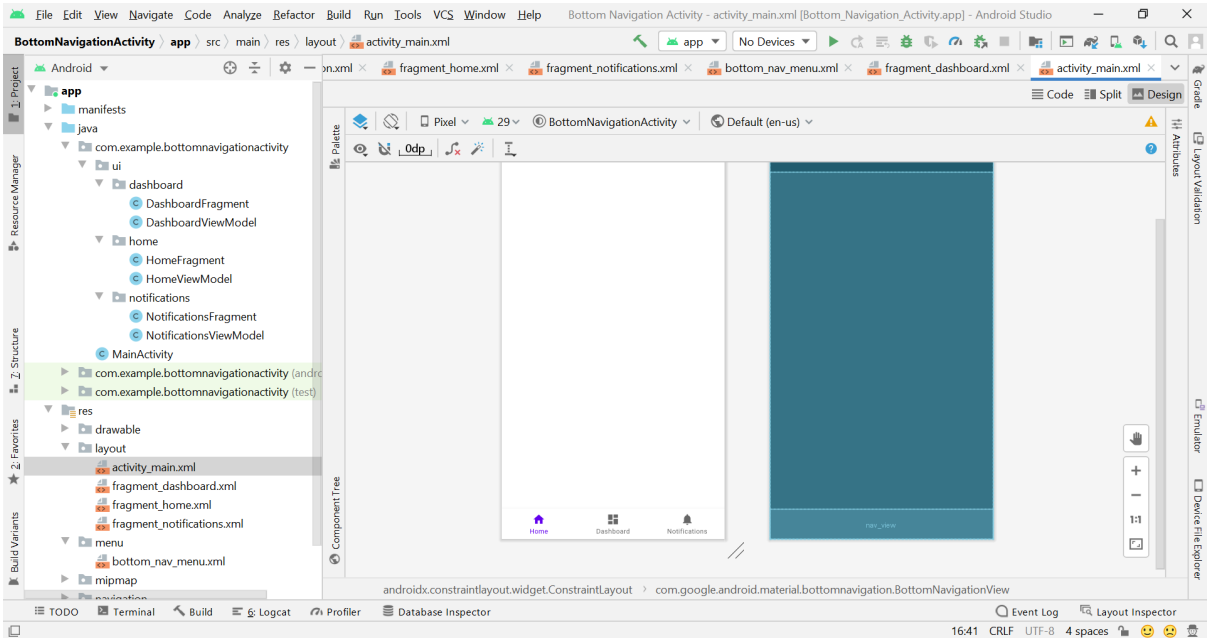
(2) Basic Activity

Basic Activity creates a new basic activity with the navigation component. When the developer selects the basic activity, then you will be getting a menu button, and you will also get a floating action button. These files are automatically created when you select Basic Activity:



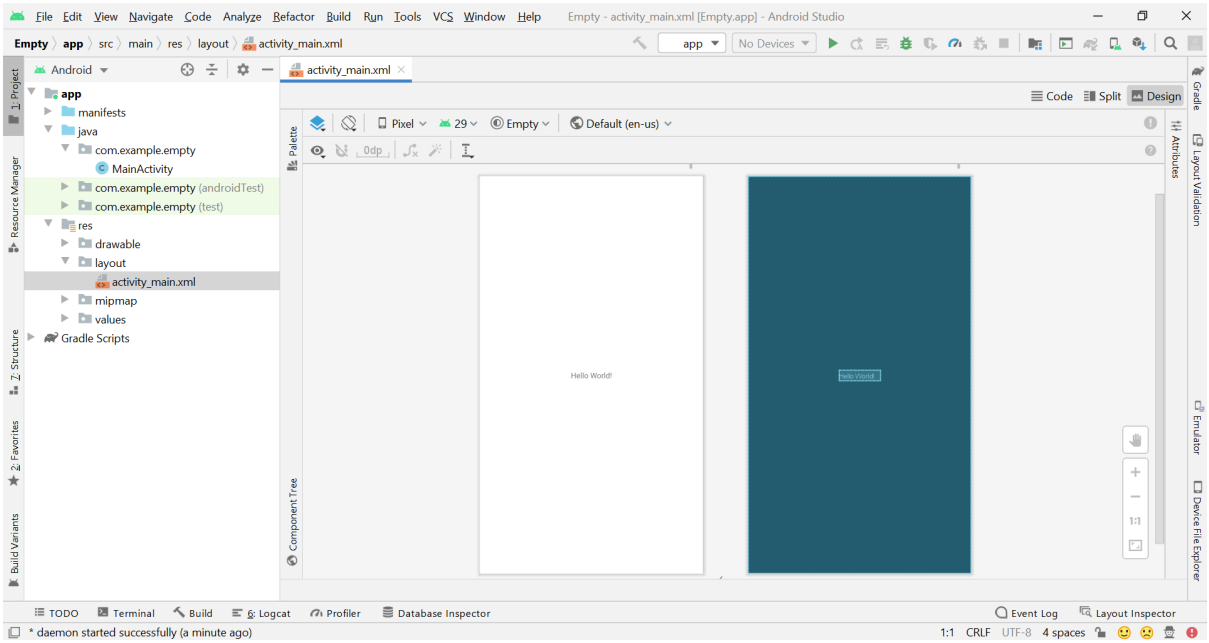
(3) Bottom Navigation Activity

Bottom Navigation Activity creates a new activity with bottom navigations. We all have come across apps that contain a Bottom Navigation Bar. Some popular examples include Instagram, WhatsApp, etc. These files are automatically created when you select Bottom Navigation Activity and the following is the welcome page:



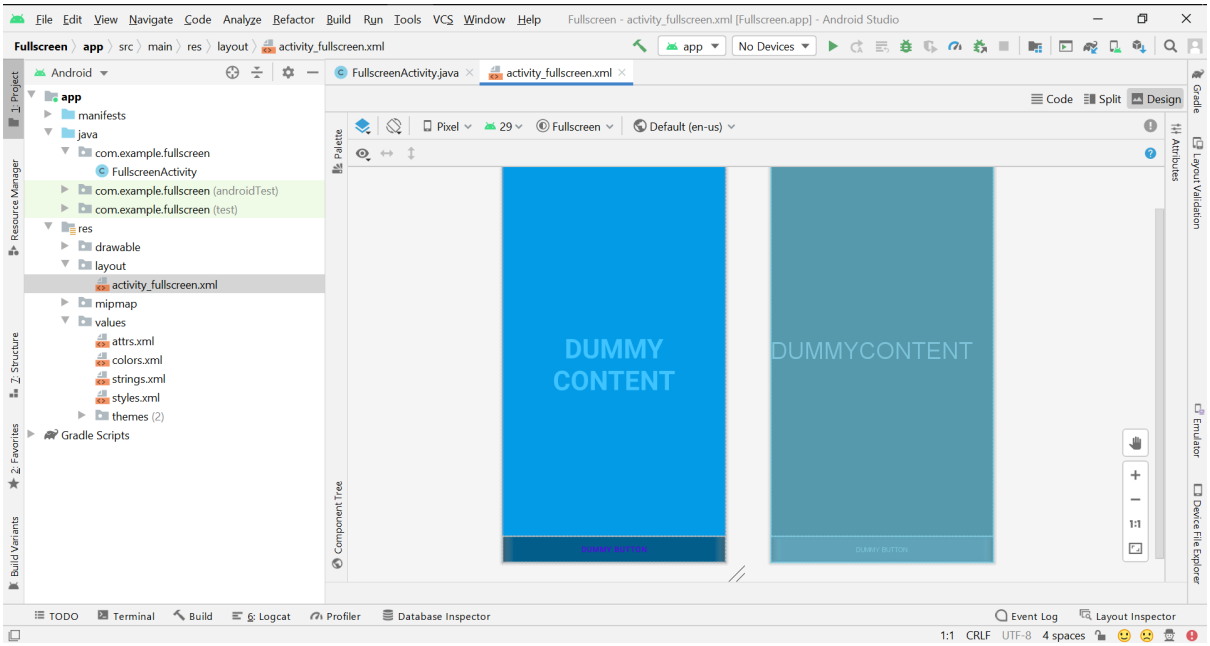
(4) Empty Activity

This is a popular activity and we frequently select this activity when we start developing an android project. It simply creates a new empty activity. These files are automatically created when you select Empty Activity and the following is the welcome page:



(5) Fullscreen Activity

Fullscreen activity creates a new activity that toggles the visibility of the system UI (status and navigation bars) and action bar upon user interaction. Many apps are using Full-Screen Activity to have an attractive screen to show slides etc. These files are automatically created when you select Fullscreen Activity and the following is the welcome page:



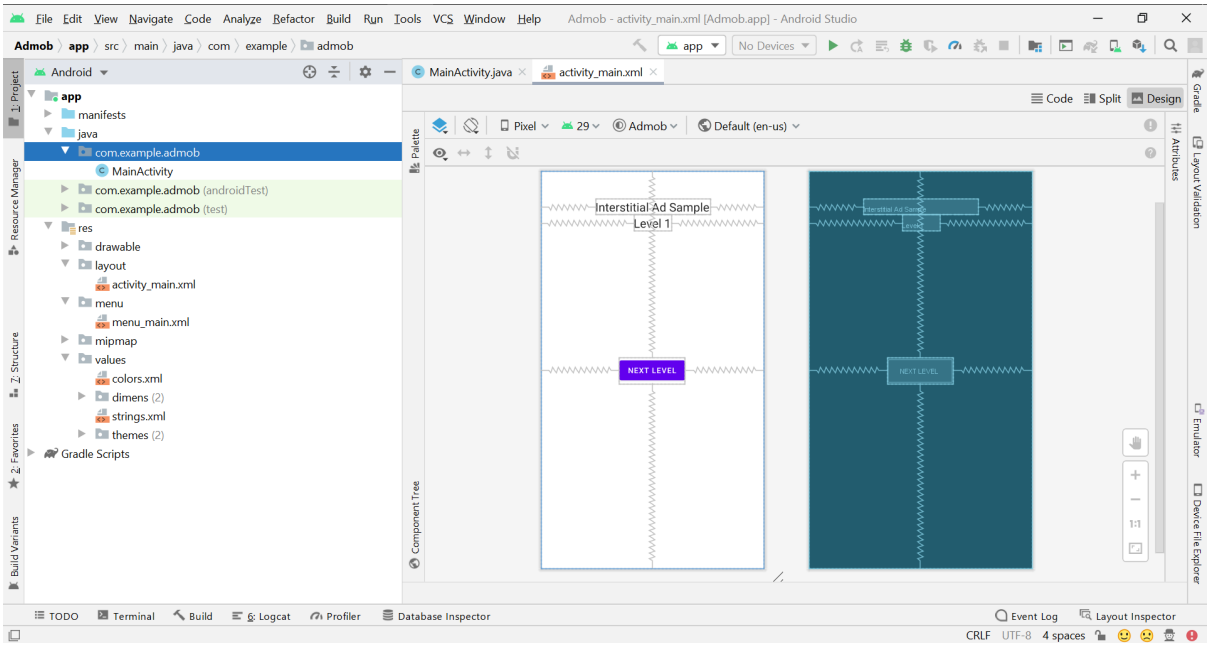
(6) Google Admob Ads Activity

To earn money from the Android app or game, there are many ways such as in-App Purchases, Sponsorship, Advertisements, and many more. But there is another popular method to earn money from the Android app is by integrating an advertisement e.g known as Google AdMob.

Google AdMob is designed with developers in mind, AdMob helps to earn more app revenue, deliver better user experience, and surface actionable insights all with automated tools that do the hard work for you. There are mainly four types of flexible, high-performing format available in Google AdMob

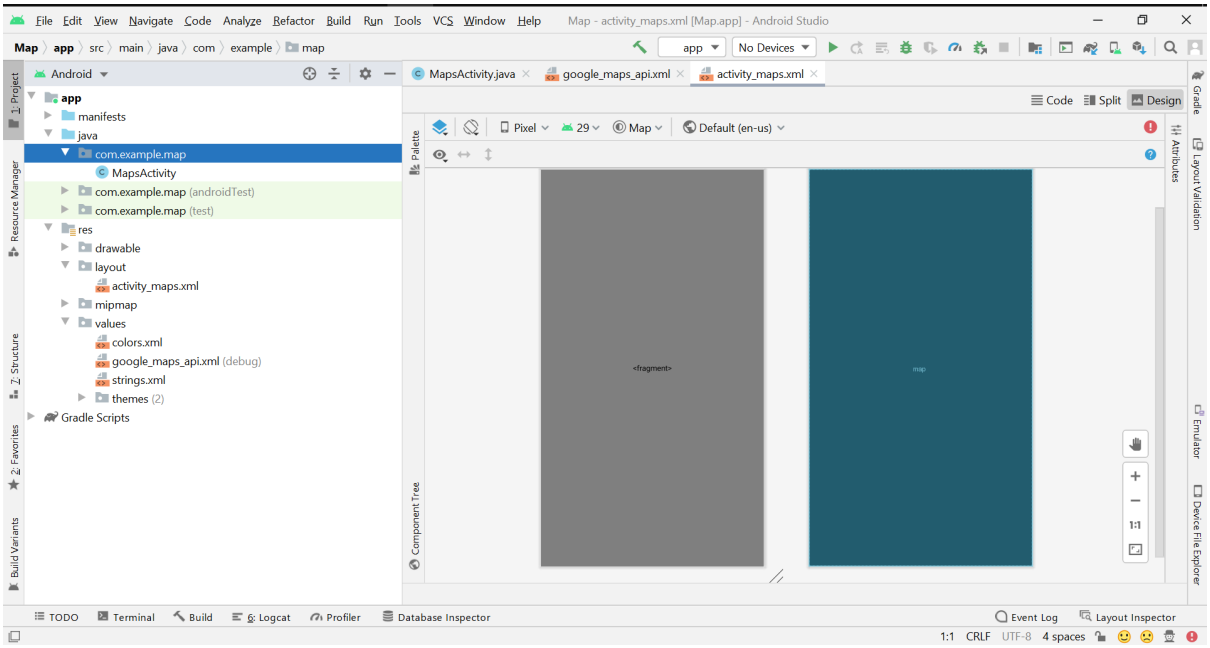
- Native: Ads that you design to fit the app, seamlessly
- Interstitial: Full-screen ads that capture attention and become part of the experience.
- Banner: Traditional formats in a variety of placements.
- Rewarded Video: An immersive, user-initiated video ad that rewards users for watching.

In Android Studio Google Admob Ads Activity creates an activity with AdMob Ad fragment. These files are automatically created when you select Google Admob Ads Activity and the following is the welcome page:



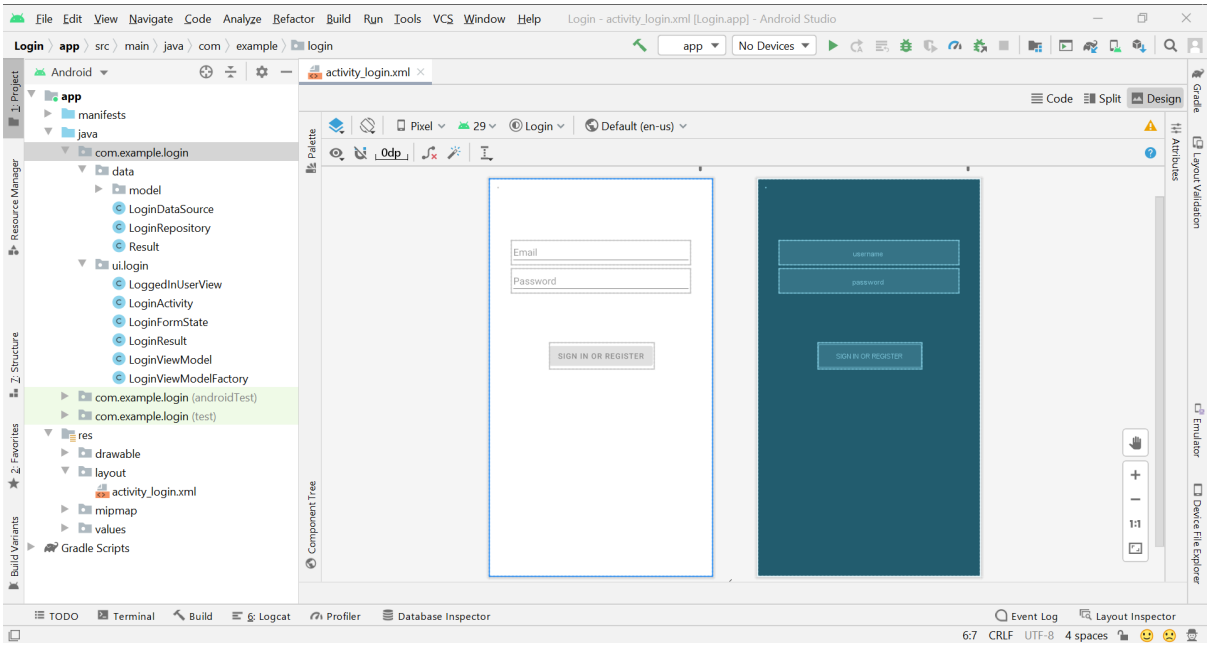
(7) Google Maps Activity

Android permits to integrate google maps in our application. One can show any location on the map or can show various routes on the map etc. One can also customize the map according to the choices. So Google Maps Activity creates a new activity with a Google Map. These files are automatically created when you select Google Maps Activity and the following is the welcome page:



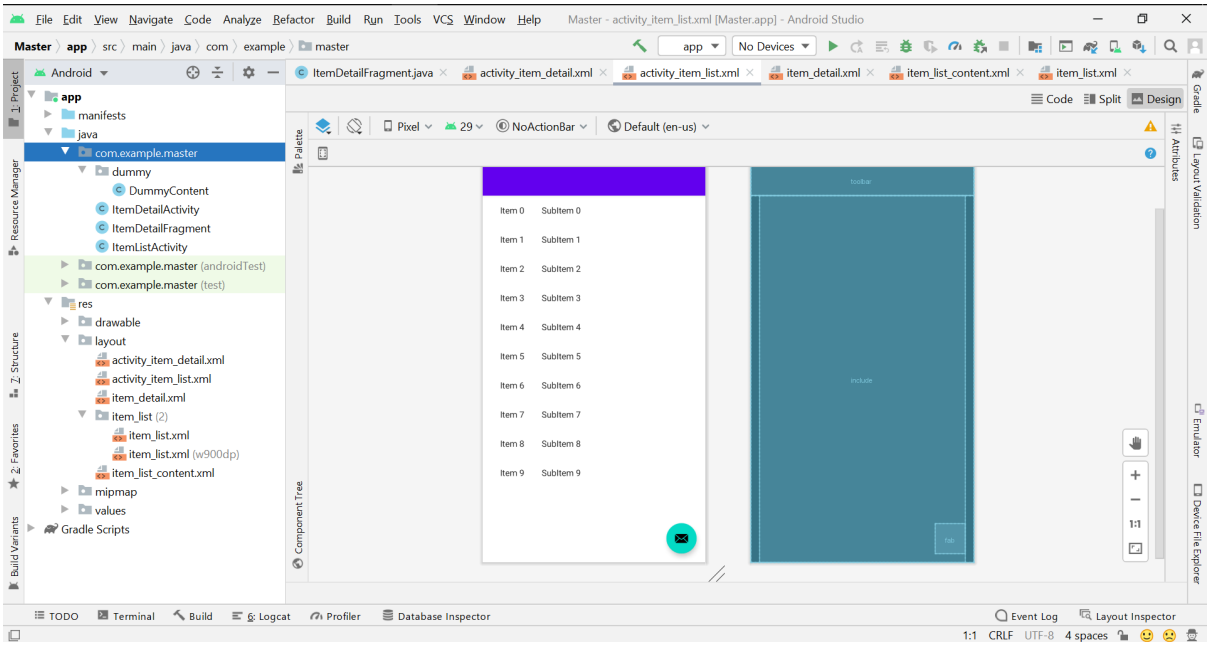
(8) Login Activity

As the name suggests Login Activity creates a new login activity, allowing users to enter an email address and password to log in or to register with the application. Login Activity is one of the most common activities that almost every application contains this activity. These files are automatically created when you select Login Activity and the following is the welcome page:



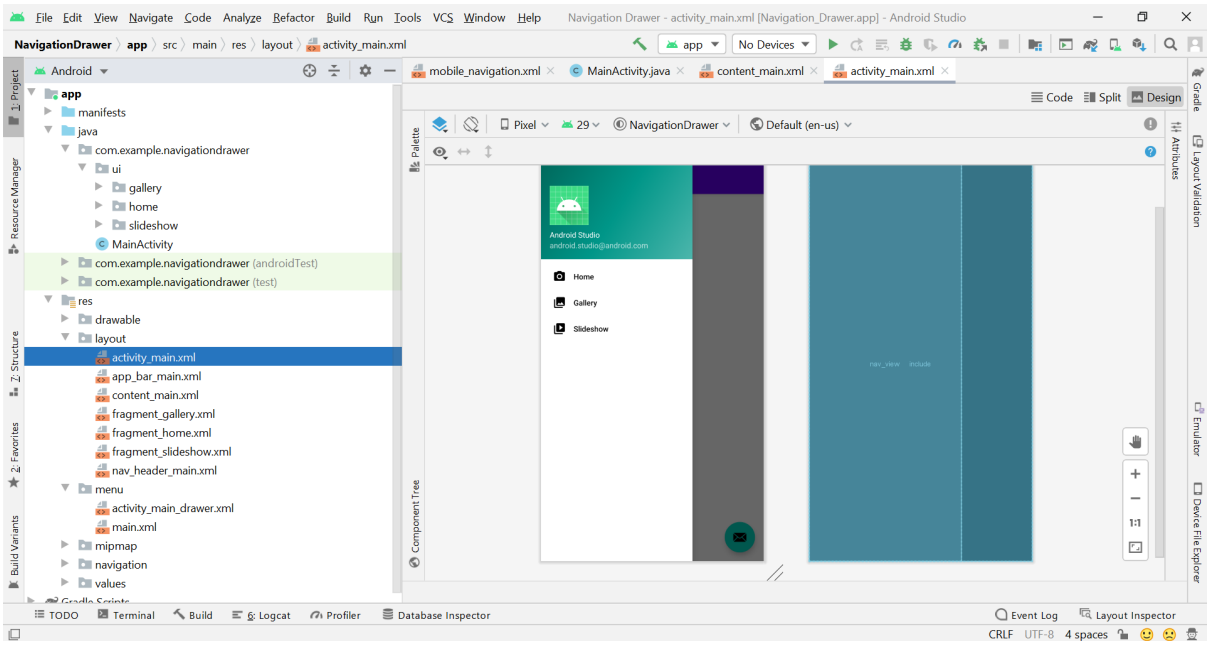
(9) Master/Detail Flow

Master/Detail Flow creates a new master/detail flow, enabling users to view a collection of objects as well as details for each object. This flow is presented using two columns on tablet-sized screens and one column on handsets and smaller screens. This template creates two activities, a master fragment, and a detailed fragment. These files are automatically created when you select Master/Detail Flow and the following is the welcome page:



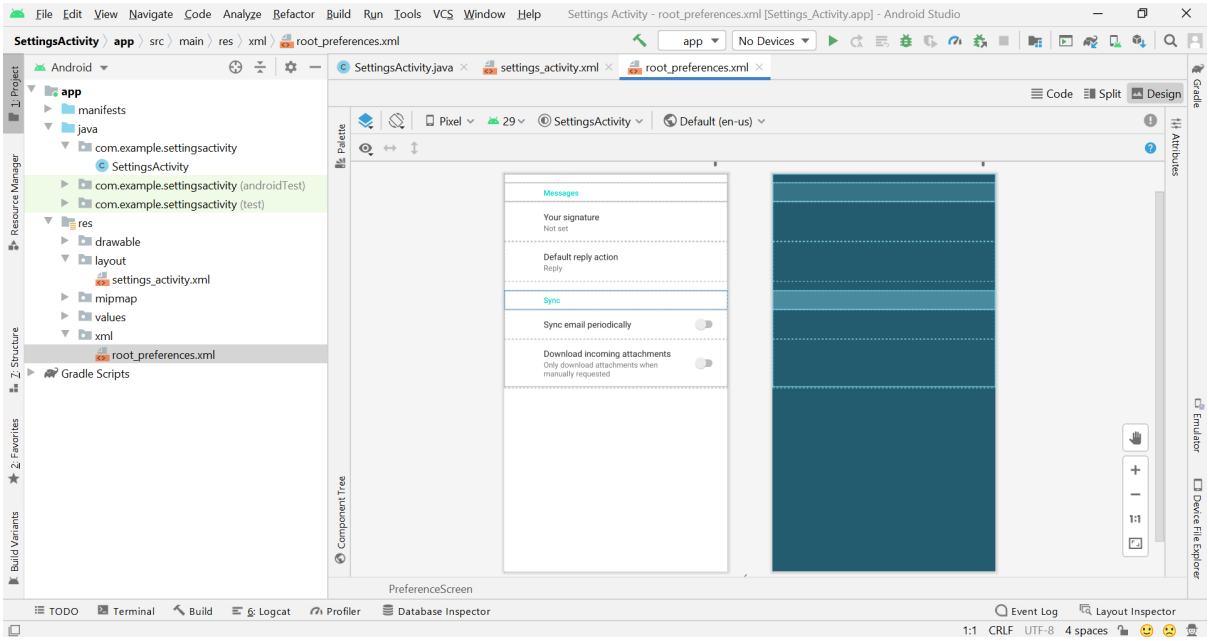
(10) Navigation Drawer Activity

Android Navigation Drawer is a sliding left menu that is used to display the important links in the application. The Navigation drawer makes it easy to navigate to and fro between those links. It's not visible by default and it needs to open either by sliding from left or clicking its icon in the ActionBar. In broader terms, Navigation Drawer is an overlay panel, which is a replacement of an activity screen that was especially dedicated to showing all the options and links in the application. These files are automatically created when you select Navigation Drawer and the following is the welcome page:



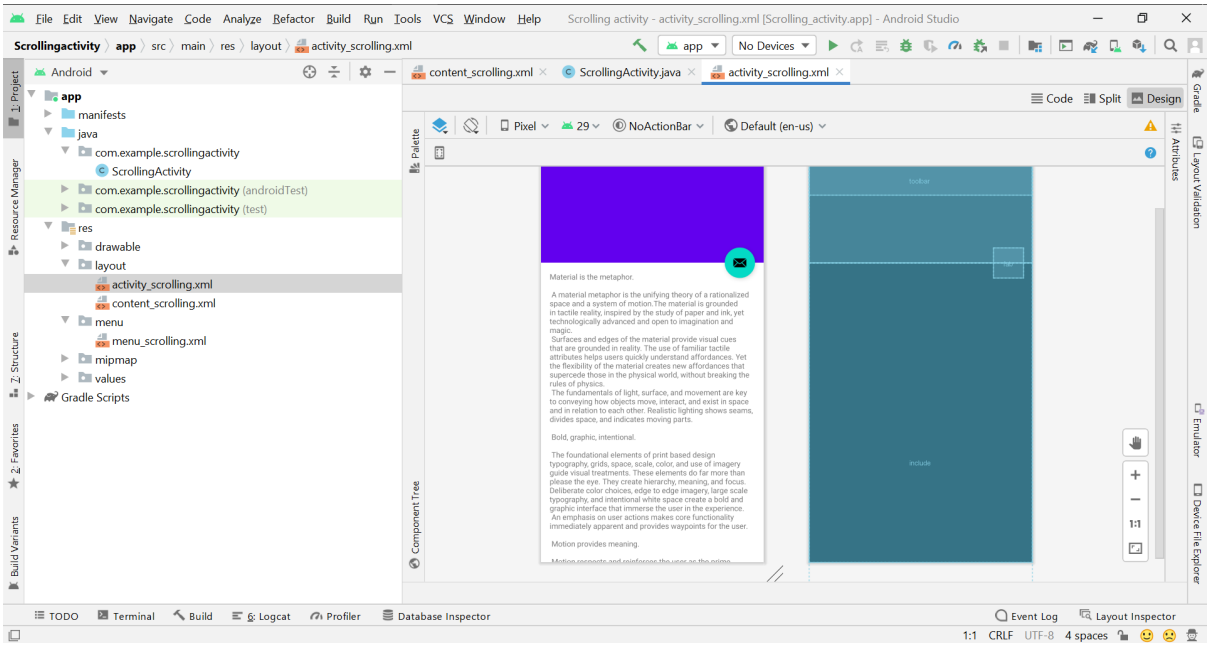
(11) Settings Activity

Setting Activity creates a new activity that allows a user to configure application settings. These files are automatically created when you select Settings Activity and the following is the welcome page:



(12) Scrolling Activity

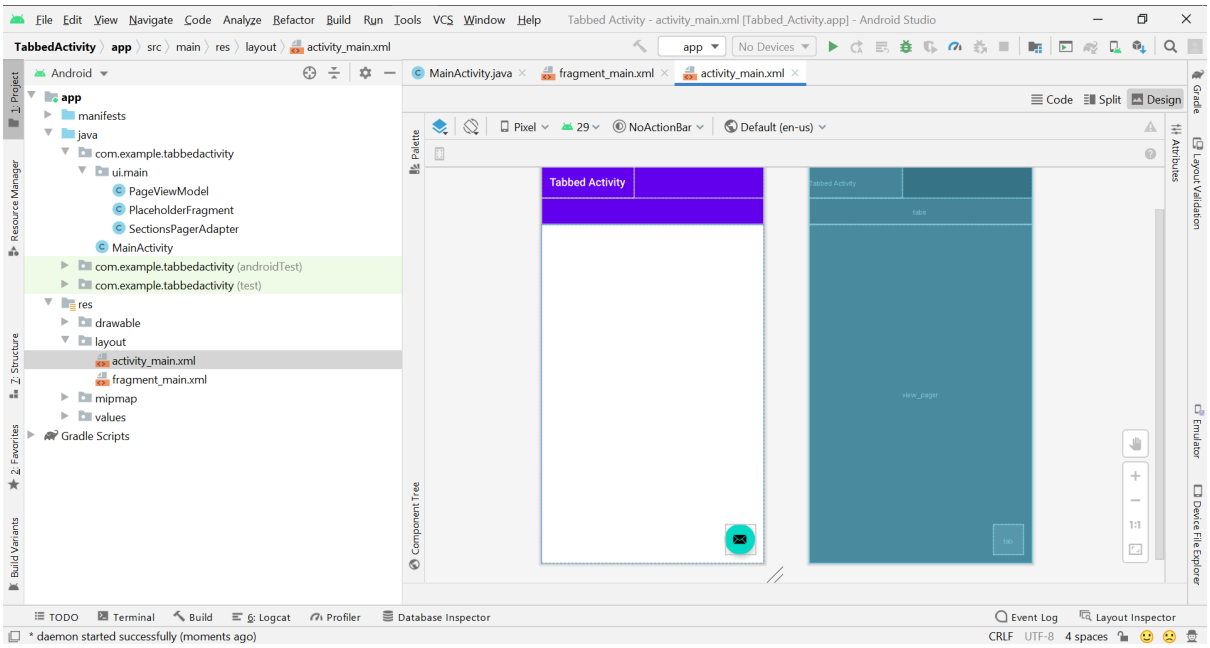
Scrolling activity is an essential activity to have in the app as it provides the users with a perfect view when the layout is long. It creates a new vertical scrolling activity. These files are automatically created when you select Scrolling activity and the following is the welcome page:



(13) Tabbed Activity

In Android, TabLayout gives a horizontal layout to display tabs. If TabLayout is used then along with it, Fragment is also used, because fragments are lightweight and the app can have more functionality on a single screen if more fragments are added. Whenever the user clicks on the tab it will lead to the transaction of one Fragment to another. ViewPager is used to swipe between the tabs. WhatsApp, Facebook, etc. are a perfect example of TabLayout with ViewPager.

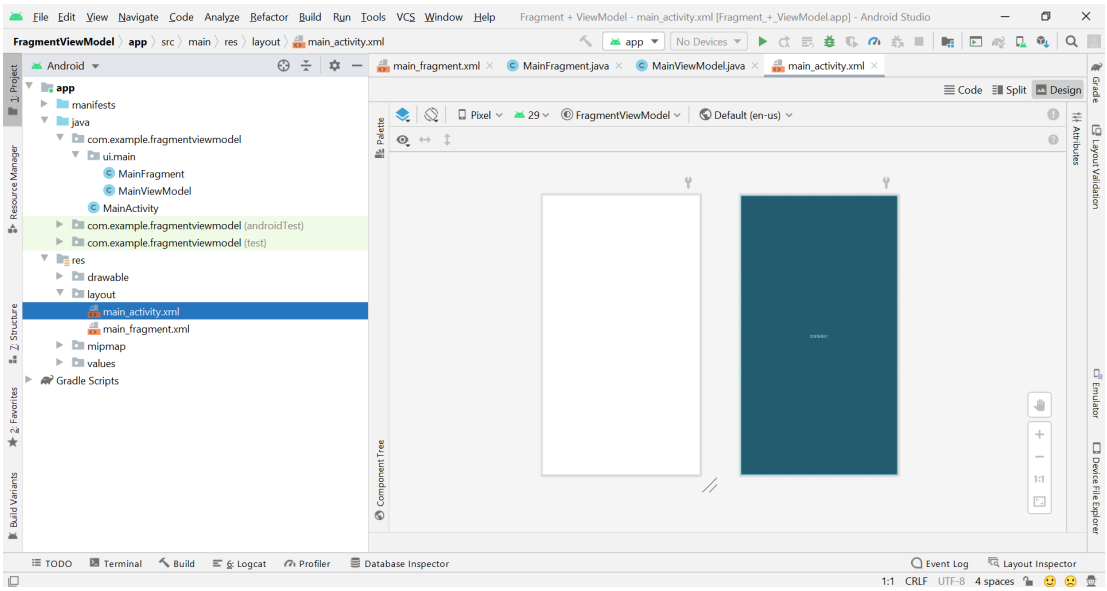
Tabbed Activity creates a new blank activity with tabs. These files are automatically created when you select Tabbed Activity and the following is the welcome page:



(14) Fragment + ViewModel

As the name suggests Fragment + ViewModel creates a new activity and a fragment with the view model.

- Fragment: A Fragment is a piece of an activity which enable more modular activity design. A fragment encapsulates functionality so that it is easier to reuse within activities and layouts.
- ViewModel: It exposes those data streams which are relevant to the View and serves as a link between the Model and the View. Model: This layer is responsible for the abstraction of the data sources. Model and ViewModel work together to get and save the data. View: The purpose of this layer is to inform the ViewModel about the user's action. This layer observes the ViewModel and does not contain any kind of application logic.



(15) Native C++

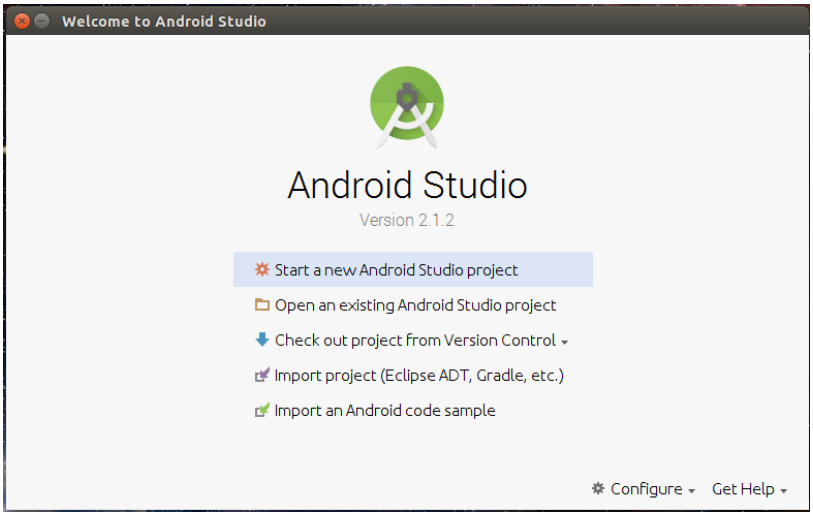
Native C++ creates a new project with an empty activity configured to use JNI. JNI is the Java Native Interface. JNI describes a way for the bytecode that Android compiles from executed code that is written in the Java or Kotlin programming languages to interact with native code that is written in C/C++. JNI is vendor-neutral, has support for loading code from dynamic shared libraries, and while cumbersome at times is efficient.

3.3 How to Create/Start a New Project in Android Studio?

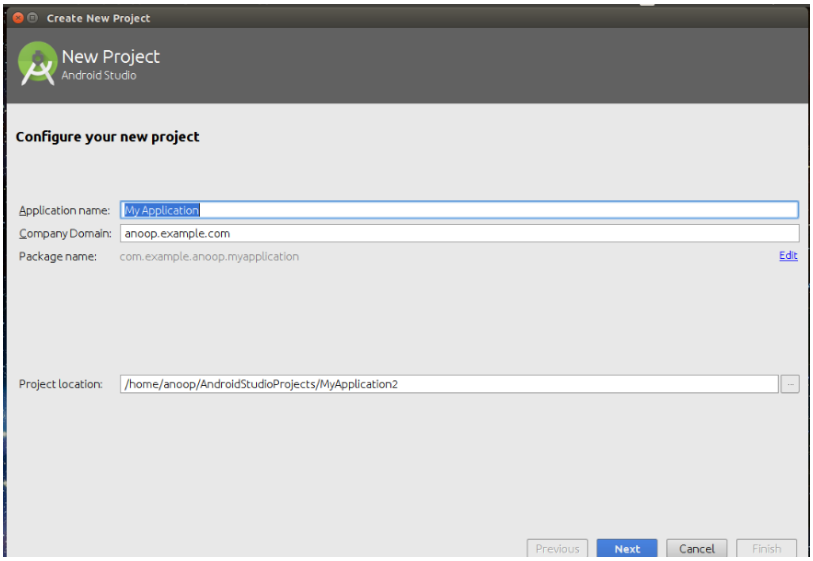
After successfully installing the Android studio and opening it for the first time, a number of options appear as shown in the below images.

Below are the steps to start and set up a new android project in Android Studio.

Select the first option to start a new android project.

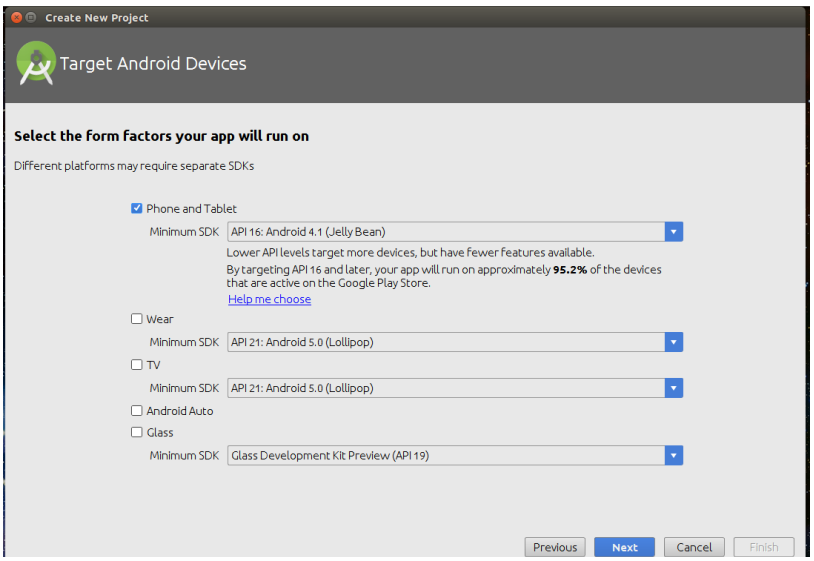


Then, name your application in the 'Application name' Text box.

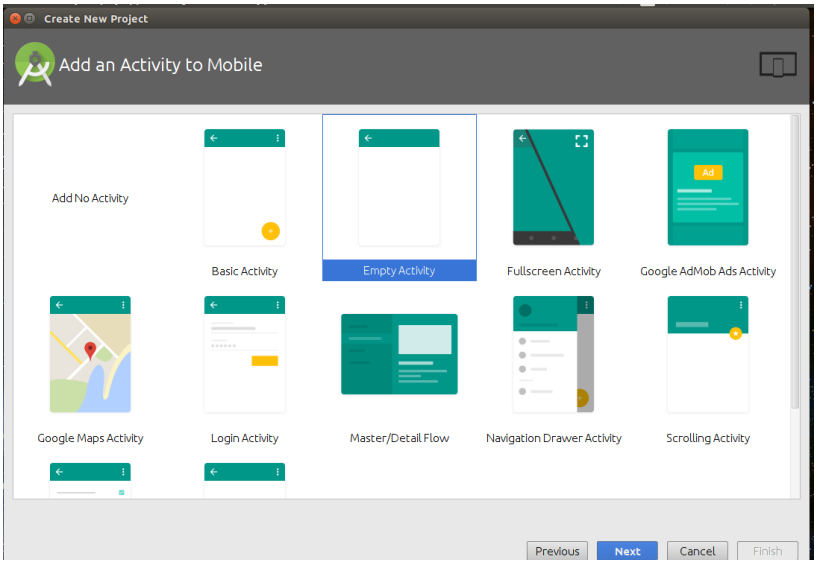


By default, the application name appears as "MyApplication". Click on "Next" after filling the Application name field, leaving the other fields to their default values.

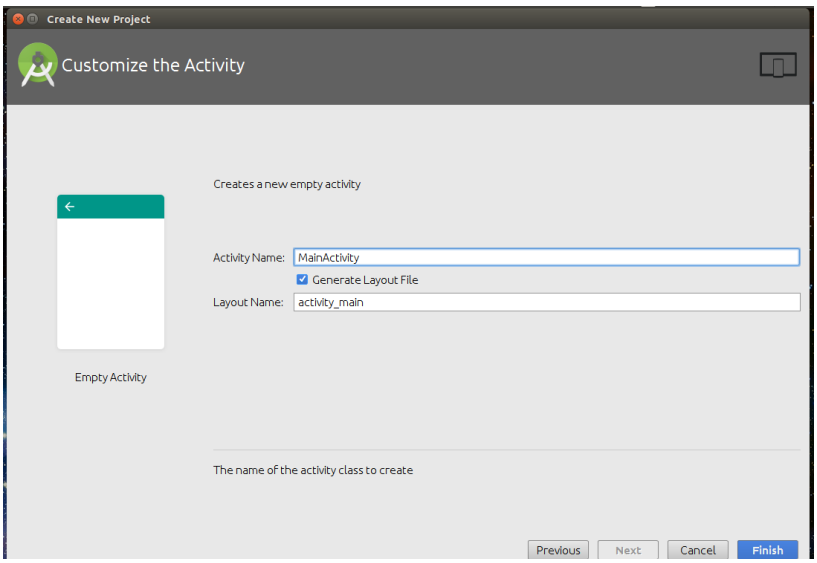
Then select the Minimum SDK to select the operating system which must be least version to run your app, here "Jelly Bean" is made Minimum SDK, and phones and tablets with versions lower to this OS will not be able to run your apps. Click on "Next".



The next step is to choose the Activity to mobile. Activity in Android refers to a single screen with a user interface. For Beginners, “Empty Activity” is recommended.

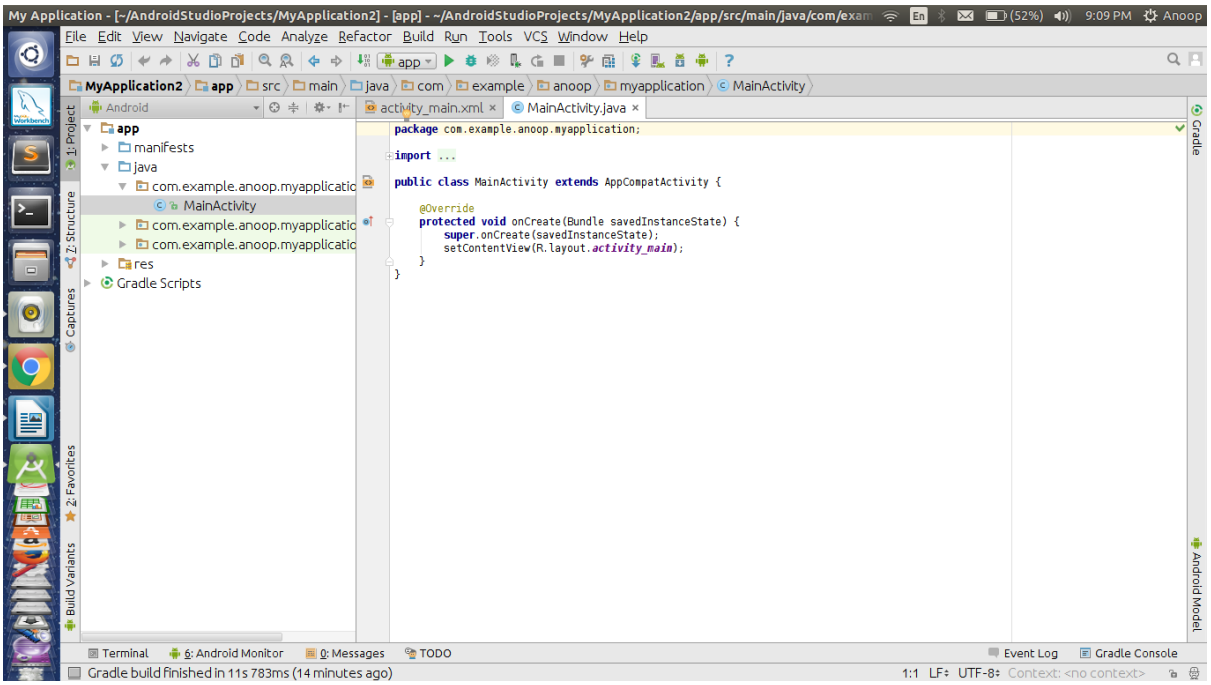


Then, fill the Activity name text field. By default, “MainActivity” is filled in this text box as the name of your activity. In android studio, files are usually named in camel case. An xml file of the activity is created using the first word in the Activity name. An xml file is a file to provide functionalities of user interface for our app.



Click on “Finish”.

Then, a default app is created with all default files. And you can now start writing the application code.



4 Views

4.1 TextView

Android TextView is simply a view that are used to display the text to the user and optionally allow us to modify or edit it. First of all, open Kotlin project in Android Studio.
Following steps are used to create TextView in Kotlin:

1. Add a TextView in activity_main.xml file inside LinearLayout.
2. Add attributes like text, textColor, textSize, textStyle in the activity_main.xml file.
3. Open MainActivity.kt file and set OnClickListener for the textView to show the Toast message.

Modify the strings.xml file

We can add strings in the strings.xml file and use in the other files easily by calling them with their names.

```
<resources>
    <string name="app_name">TextViewInKotlin</string>
    <string name="text_view">FarizGaskin</string>
    <string name="text_on_click">COMPUTER SCIENCE PORTAL</string>
</resources>
```

activity_main.xml file

Open activity_main.xml file and create a TextView using id textView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

<!--EditText with id editText-->

    <TextView
        android:id="@+id/text_view_id"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/text_view"
        android:textColor="#008000"
        android:textSize="40dp"
        android:textStyle="bold"/>
</LinearLayout>
```

Open MainActivity.kt file and get the reference of TextView defined in the layout file.

```
// finding the textView
val textView = findViewById(R.id.text_view_id) as TextView
```

Setting the on click listener to the button

```
textView?.setOnClickListener{ Toast.makeText(this@MainActivity,
                "COMPUTER SCIENCE PORTAL", Toast.LENGTH_LONG).show() }
```

MainActivity.kt file

Open app/src/main/java/yourPackageName/MainActivity.kt to get the reference of TextView.

```
package com.farizgaskin.myfirstkotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.TextView
import android.widget.Toast

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //accessing our textview from layout
        val textView = findViewById<TextView>(R.id.text_view_id) as
TextView
        textView?.setOnClickListener{ Toast.makeText(this@MainActivity,
        R.string.text_on_click, Toast.LENGTH_LONG).show() }
    }
}
```

AndroidManifest.xml file

We are also going to see the code inside main/AndroidManifest.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.farizgaskin.myfirstkotlinapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
```

```
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Run your app on an emulator to see the results

4.2 Copying to Clipboard

Android's Clipboard performs copying and pasting on different data types, such as text strings, images, binary stream data, and other complex data types. Clipboard does the copying and pasting operations within the same application and between multiple applications that have implemented the clipboard framework. Clipboard has a limitation on the number of clip objects it can hold at a time. The clipboard can hold only one object at a time. If an object is put on the clipboard, the previously held object on the clipboard is dropped. The clip object can take in three types of data:

- Text: A string can directly be put into the clip object and then into the clipboard. We can then get the clip object from the clipboard and paste the string into the application's text or storage fields.
- URI: It is used for copying complex data from the content provider. A URI object can be put into a clip object and then loaded onto the clipboard. To perform a paste operation, the clip object must be resolved into the source, such as a content provider.
- Intent: An intent object must be created and put into a clip object and loaded onto the clipboard. Paste action, similar to the text, can be performed.

Step by Step Implementation

To make an application that stores some data into the clipboard and derives the data from it in Android, we follow the following steps:

Example 1: Saving to Clipboard

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Kotlin as the programming language.

Step 2: Working with the activity_main.xml file

Go to the activity_main.xml file which represents the UI of the application. Create an EditText where we shall supply the text to be saved in the clipboard, and a Button to perform the saving action. Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

<!--Text must be entered here-->

    <EditText
        android:id="@+id/txtCopy"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/btnCopy"
        android:layout_centerHorizontal="true"
        android:hint="Type something..." />
```

```
<!--Text entered in the above field gets copied to Clipboard on this
button click-->
    <Button
        android:id="@+id/btnCopy"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Copy to Clipboard" />

</RelativeLayout>
```

Step 3: Working with the MainActivity.kt file

Go to the MainActivity.kt file, and refer to the following code. Below is the code for the MainActivity.kt file. Comments are added inside the code to understand the code in more detail.

```
import android.content.ClipData
import android.content.ClipboardManager
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Declaring the edit text and button from the layout file
        val copyTxt = findViewById<EditText>(R.id.txtCopy)
        val copyBtn = findViewById<Button>(R.id.btnCopy)
        // Initializing the ClipboardManager and Clip data
        val clipboardManager = getSystemService(CLIPBOARD_SERVICE) as
ClipboardManager
        var clipData: ClipData
        // Action when the copy button is clicked
        copyBtn.setOnClickListener {

            // Text from the edit text is stored in a val
            val txtCopy = copyTxt!!.text.toString()
            // clip data is initialized with the text variable declared
above
            clipData = ClipData.newPlainText("text", txtCopy)
            // Clipboard saves this clip object
            clipboardManager.setPrimaryClip(clipData)
            // A toast is shown for user reference that the text is
copied to the clipboard
            Toast.makeText(applicationContext, "Copied to Clipboard",
Toast.LENGTH_SHORT).show()
        }
    }
}
```

Run your app to see the results

Example 2: Pasting from Clipboard

Step 1: Working with the activity_main.xml file

Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!--Text from the clip object will be shown here*-->
    <TextView
        android:id="@+id/txtShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/btnShow"
        android:layout_centerHorizontal="true"
        android:hint="Clipboard Data" />

    <!--*on this button click-->
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Show Clipboard Data" />

</RelativeLayout>
```

Step 2: Working with the MainActivity.kt file

Go to the MainActivity.kt file, and refer to the following code. Below is the code for the MainActivity.kt file. Comments are added inside the code to understand the code in more detail.

```
import android.content.ClipboardManager
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Declare the textview and button from the layout file
        val pasteTxt = findViewById<TextView>(R.id.txtShow)
        val btnPaste = findViewById<Button>(R.id.btnShow)

        // Declaring the clipboard manager
        val clipboardManager = getSystemService(CLIPBOARD_SERVICE) as
ClipboardManager
```

```
        // Action on paste button click
        btnPaste.setOnClickListener {

            // Storing the clip data in a variable
            val pData = clipboardManager.primaryClip

            // Retrieving the items
            val item = pData!!.getItemAt(0)

            // item is converted to string and stored in a variable
            val txtPaste = item.text.toString()

            // Textview is set as txtPaste string
            pasteTxt!!.text = txtPaste

            // Toast for user reference
            Toast.makeText(applicationContext, "Pasted from Clipboard",
Toast.LENGTH_SHORT).show()
        }
    }
}
```

4.3 How to Add a TextView with Rounded Corner

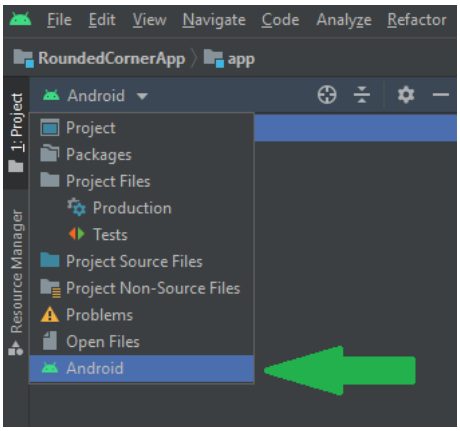
TextView is an essential object of an Android application. It comes in the list of some basic objects of android and used to print text on the screen. In order to create better apps, we should learn that how can we create a TextView which have a background with rounded corners. We can implement this skill in a practical manner for any android application. For creating professional-looking user interfaces, we need to take care of these small things.

In this lesson, You will be learning how to create a TextView with Rounded Corners. First of all, we need to create a drawable resource file, which has a proper definition to make TextView corners rounded, and then we have to add a background attribute to that special TextView object. Let's do it in steps!

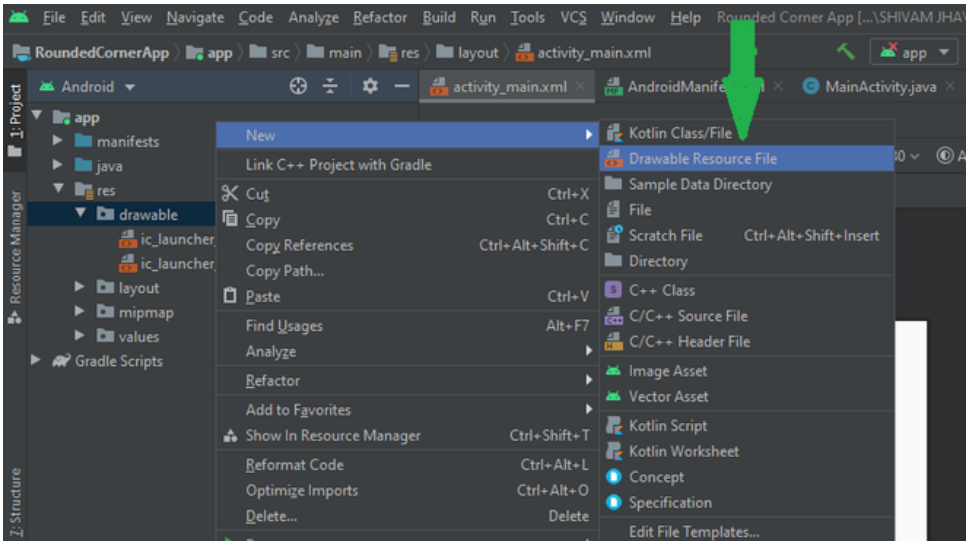
Step by Step Implementation

Step 1: Create a new android studio project, and select an empty activity.

Step 2: Make sure that you have selected the Android option for project structure on the top left corner of the screen, then go to the res/drawable folder.



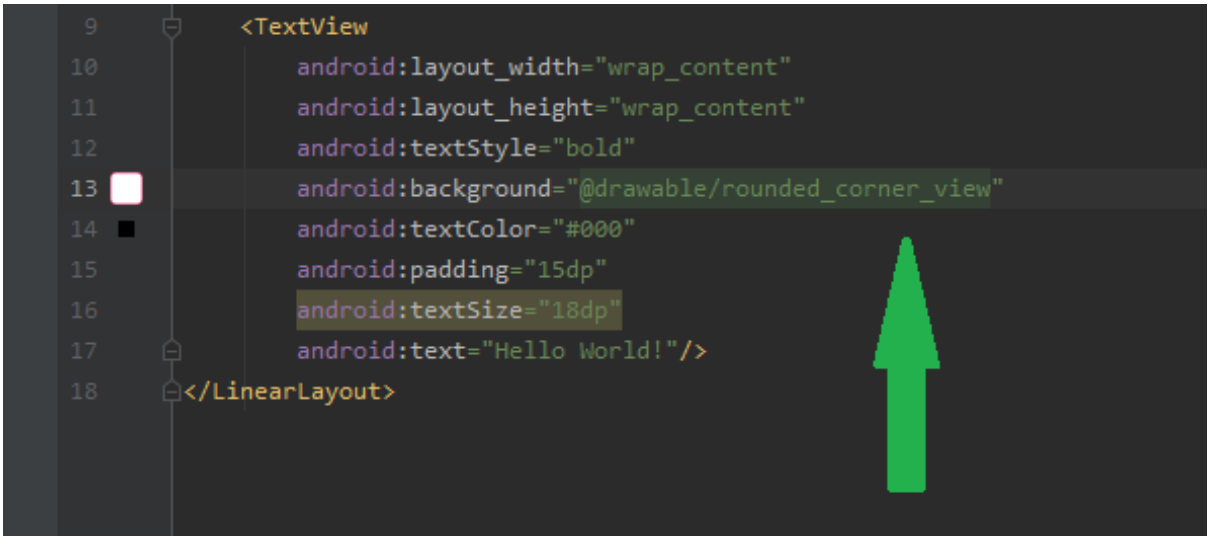
Step 3: Here right-click on the drawable folder and click on new and select drawable resource file. Give it a name of your choice, we are giving it rounded_corner_view.



Step 4: Now in the drawable resource file, which you have just created, remove all the default code and paste the following code.

```
<?xml version="1.0" encoding="utf-8"?>

<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <corners android:radius="10dp" />

      <!-- This is the border color -->
    <stroke android:width="2dp" android:color="#ccc" />

    <!--- This is the background color -->
      <solid android:color="#ccc" />

</shape>
```

Step 5: Now go to the activity_main.xml file and add an attribute to that TextView, for which you want to add rounded corners. The attribute is android: background=”@drawable/rounded_corner_view”.



Run your app to see the results.

5 EditText

5.1 Android EditText in Kotlin

EditText is used to get input from the user. EditText is commonly used in forms and login or registration screens. Following steps are used to create EditText in Kotlin:

1. Add a EditText in activity_main.xml file.
2. Add a Button in activity_main.xml file.
3. Open MainActivity.kt file and set OnClickListner for the button to get the user input from EditText and show the input as Toast message.

activity_main.xml file

Step 1: Open activity_main.xml file and create an EditText using id editText.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

     <!--EditText with id editText-->

    <EditText
            android:id="@+id/editText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="16dp"
            android:hint="Input"
            android:inputType="text"/>

</LinearLayout>
```

Step 2: In activity_main.xml file add code to show a button. Final activity_main.xml file is

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

     <!--EditText with id editText-->

    <EditText
            android:id="@+id/editText"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:layout_margin="16dp"
            android:hint="Input"
            android:inputType="text"/>

    <!--Button with id showInput-->

    <Button
            android:id="@+id/showInput"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="show"
            android:backgroundTint="@color/colorPrimary"
            android:textColor="@android:color/white"
            />

</LinearLayout>
```

Step 3: Open MainActivity.kt file and get the reference of Button and EditText defined in the layout file.

```
// finding the button
val showButton = findViewById<Button>(R.id.showInput)

// finding the edit text
val editText = findViewById<EditText>(R.id.editText)
```

Setting the on click listener to the button

```
showButton.setOnClickListener {

}
```

Getting the text entered by user

```
val text = editText.text
```

MainActivity.kt file

Finally the MainActivity.kt file is

```
package com.farizgaskin.myfirstkotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
```

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // finding the button
        val showButton = findViewById<Button>(R.id.showInput)

        // finding the edit text
        val editText = findViewById<EditText>(R.id.editText)

        // Setting On Click Listener
        showButton.setOnClickListener {

            // Getting the user input
            val text = editText.text

            // Showing the user input
            Toast.makeText(this, text, Toast.LENGTH_SHORT).show()
        }
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.farizgaskin.myfirstkotlinapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Run the app to see the results

5.2 How to add Mask to an EditText in Android

EditText is an android widget. It is a user interface element used for entering and modifying data. It returns data in String format.

Masking refers to the process of putting something in place of something else. Therefore by Masking an EditText, the blank space is replaced with some default text, known as Mask. This mask gets removed as soon as the user enters any character as input, and reappears when the text has been removed from the EditText.

In this lesson, the masking is done with the help of JitPack library, because it can be customized easily according to the need to implement various fields like phone number, date, etc.

Approach:

Add the support Library in your root build.gradle file (not your module build.gradle file). This library jitpack is a novel package repository. It is made for JVM so that any library which is present in github and bigbucket can be directly used in the application.

```
allprojects {
    repositories {
        maven {
            url "https://jitpack.io"
        }
    }
}
```

Add the below dependency in the dependencies section. It is a simple Android edittext with custom mask support. Mask edittext is directly imported and is customized according to the use.

```
dependencies {
    implementation 'ru.egslava:MaskedEditText:1.0.5'
}
```

Now add the following code in the activity_main.xml file. It will create three mask edittexts and one button in activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <br.com.sapereaude.maskedEditText.MaskedEditText
        android:hint="#### #### #### ####"
        android:layout_width="match_parent"
        android:inputType="number"
```

```
        <!-- Set the masked characters -->
        app:mask="#### #### #### ####"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:id="@+id/card"/>

    <br.com.sapereaude.maskedEditText.MaskedEditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:id="@+id/phone"
        android:hint="9876543210"
        android:inputType="phone"
        app:keep_hint="true"

        <!-- Set the masked characters -->
        app:mask="+91 ### ### ####"/>

    <br.com.sapereaude.maskedEditText.MaskedEditText
        android:hint="##:##:####"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:id="@+id/Date"
        android:inputType="date"

        <!-- Set the masked characters -->
        app:mask="##:##:####"/>

    <Button
        android:id="@+id/showButton"
        android:layout_marginTop="40dp"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAllCaps="false"
        android:textSize="18sp"
        android:text="Show"
        />
</LinearLayout>
```

Now add the following code in the MainActivity.java file. All the three mask edittexts and a button are defined. An onClickListener() is added on the button which creates a toast and shows all the data entered in the mask edittexts.

```
package org.farizgaskin.MaskEditText;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import br.com.sapereaude
    .maskedEditText
    .MaskedEditText;

public class MainActivity
```

```
    extends AppCompatActivity {

    MaskedEditText creditCardText,
        phoneNumText,
        dateText;
    Button show;

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

        creditCardText = findViewById(R.id.card);
        phoneNumText = findViewById(R.id.phone);
        dateText = findViewById(R.id.Date);
        show = findViewById(R.id.showButton);

        show.setOnClickListener(
            new View.OnClickListener() {

                @Override
                public void onClick(View v)
                {

                    // Display the information
                    // from the EditText
                    // with help of Taosts
                    Toast.makeText(
                             MainActivity.this,
                             "Credit Card Number "
                                 + creditCardText.getText()
                                 + "\n Phone Number "
                                 + phoneNumText.getText()
                                 + "\n Date "
                                 + dateText.getText(),
                             Toast.LENGTH_LONG)
                         .show();
                }
            });
    }
}
```

Run the app to see the results

6 ImageView

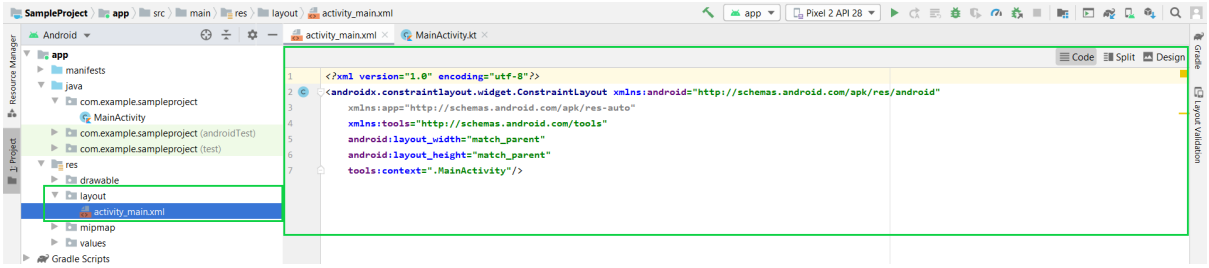
6.1 ImageView in Android

ImageView class is used to display any kind of image resource in the android application either it can be android.graphics.Bitmap or android.graphics.drawable.Drawable (it is a general abstraction for anything that can be drawn in Android). ImageView class or android.widget.ImageView inherits the android.view.View class which is the subclass of Kotlin.Any class. Application of ImageView is also in applying tints to an image in order to reuse a drawable resource and create overlays on background images. Moreover, ImageView is also used to control the size and movement of an image.

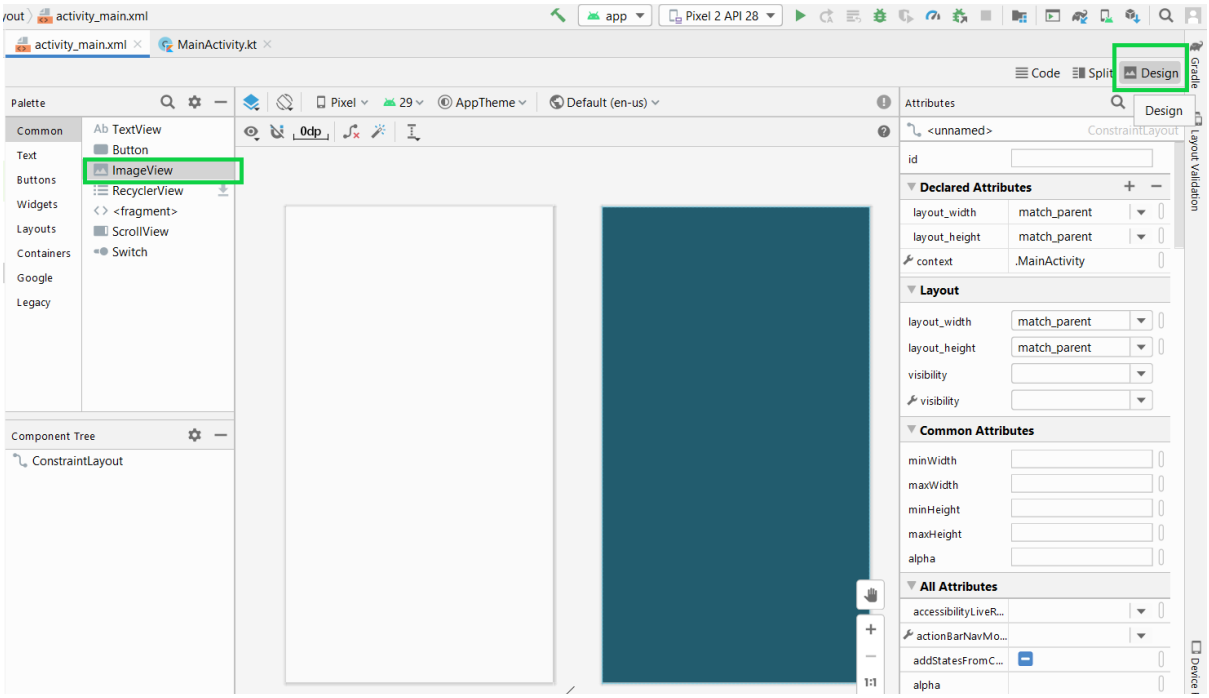
Adding an ImageView to an activity

Whenever ImageView is added to an activity, it means there is a requirement for an image resource. Thus it is oblivious to provide an Image file to that ImageView class. It can be done by adding an image file that is present in the Android Studio itself or we can add our own image file. Android Studio owns a wide range of drawable resources which are very common in the android application layout. The following are the steps to add a drawable resource to the ImageView class.

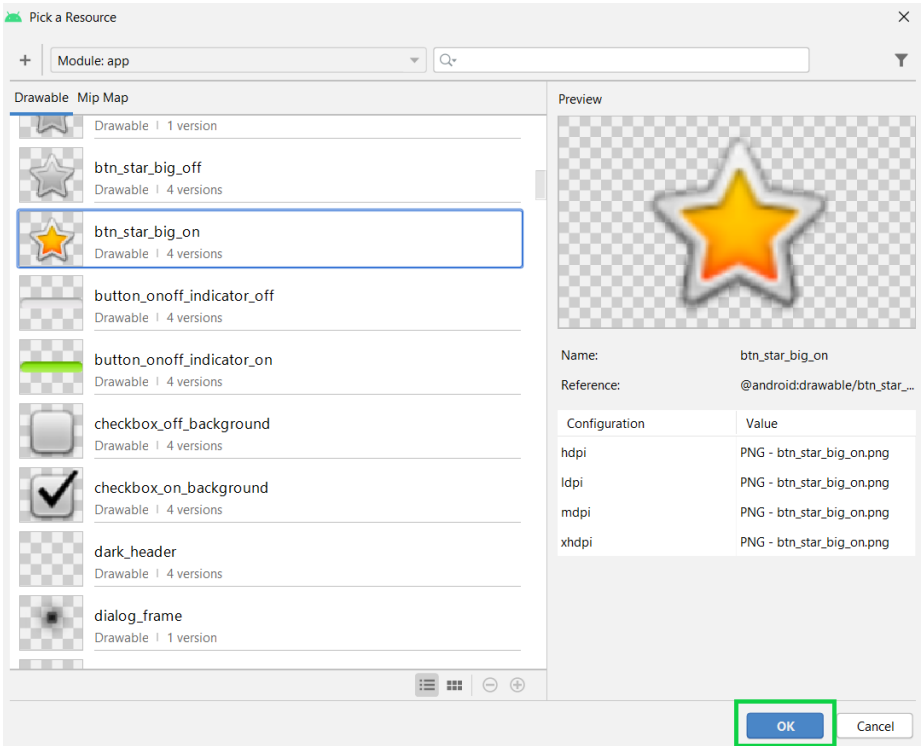
Open the activity_main.xml file in which the image is to be added



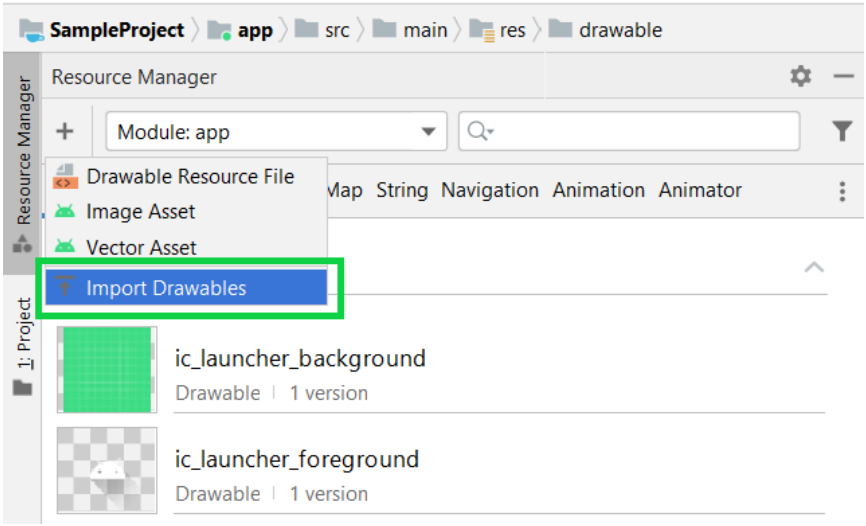
Switch from code view to the design view of the activity_main.xml file.



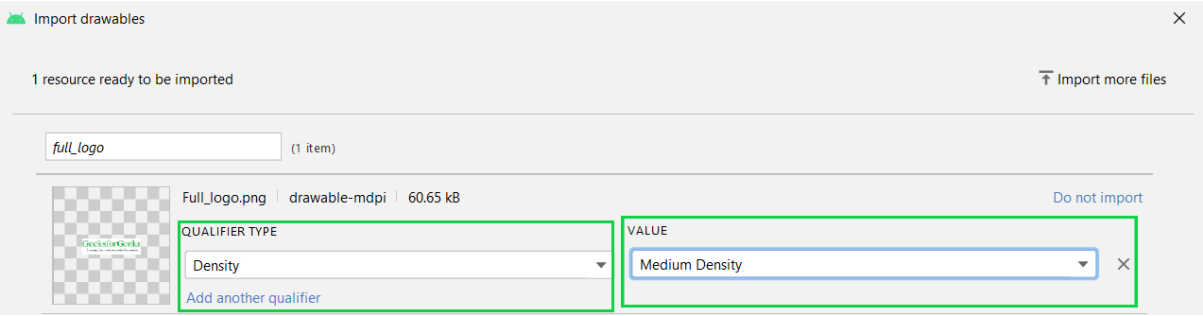
For adding an image from Android Studio Drag the ImageView widget to the activity area of the application, a pop-up dialogue box will open choose from the wide range of drawable resources and click “OK”.



For adding an image file other than Android Studio drawable resources:
Click on the “Resource Manager” tab on the leftmost panel and select the “Import Drawables” option.



Select the path of the image file on your computer and click “OK”. After that set, the “Qualifier type” and “value” of the image file according to your need and click “Next” then “Import”.



Drag the ImageView class in the activity area, a pop-up dialogue box will appear which contain your imported image file. Choose your image file and click “OK”, your image will be added to the activity.

Example

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio.

Step 2: Working with the activity_main.xml file

Navigate to the app > res > layout > activity_main.xml and add the below code to that file. Below is the code for the activity_main.xml file.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/GfG_full_logo"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.078"
        app:srcCompat="@drawable/full_logo" />

    <ImageView
        android:id="@+id/GfG_logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/GfG_full_logo"
        app:srcCompat="@drawable/logo" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 4: Working with the MainActivity file

Go to the MainActivity file and refer to the following code. Below is the code for the MainActivity file. Comments are added inside the code to understand the code in more detail. Since in the activity, only 2 images have been added and nothing else is being done like touching a button, etc. So, the MainActivity file will simply look like the below code.

Kotlin

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
    }
}
```

Run the app to see the results

6.2 How to Create Circular ImageView in Android using CardView?

Displaying an Image in Android is done easily using ImageView., But what if one wants to display a circular image? It is seen that many Android apps use CircularImageView to show the profile images, status, stories, and many other things but doing this with a normal ImageView is a bit difficult. This article will help to create a circular image using CardView. Through cardCornerRadius one can customize the corner of the ImageView.

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language.

Step 2: Add dependency to the build.gradle file

Go to build.gradle file and add this dependency and click on Sync Now button.

```
implementation 'androidx.cardview:cardview:1.0.0'
```

Step 3: Working with the activity_main.xml file

Next, go to the activity_main.xml file, which represents the UI of the project. Below is the code for the activity_main.xml file. Comments are added inside the code to understand the code in more detail.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <!-- Using CardView for CircularImageView -->
    <androidx.cardview.widget.CardView
        android:id="@+id/cardView"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="150dp"
        app:cardCornerRadius="100dp">

        <!-- add a Image image.png in your Drawable folder -->
        <ImageView
            android:id="@+id/imageView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scaleType="centerCrop"
            android:src="@drawable/circular" />
```

```
    </androidx.cardview.widget.CardView>

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/cardView"
        android:layout_marginTop="25dp"
        android:gravity="center"
        android:text="Circular ImageView"
        android:textColor="@color/colorPrimary"
        android:textSize="20sp"
        android:textStyle="bold" />

</RelativeLayout>
```

Step 4: Working with the MainActivity.java file

Finally, go to the MainActivity.kt file, and refer to the following code. Below is the code for the MainActivity.kt file. We have added a Toast message only. It Toasts a message as you click on the Image.

```
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    ImageView imageView;

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

        imageView = (ImageView) findViewById(R.id.imageView);
        imageView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "This is a Circular
ImageView", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Run the app to see the results

7 ListView

7.1 Android ListView in Kotlin

Android ListView is a ViewGroup which is used to display the list of items in multiple rows and contains an adapter which automatically inserts the items into the list.

The main purpose of the adapter is to fetch data from an array or database and insert each item that placed into the list for the desired result. So, it is main source to pull data from strings.xml file which contains all the required strings in Kotlin or xml files.

Android Adapter

Adapter holds the data fetched from an array and iterates through each item in data set and generates the respective views for each item of the list. So, we can say it act as an intermediate between the data sources and adapter views such as ListView, Gridview.

Different Types of Adapter

- ArrayAdapter: It always accepts an Array or List as input. We can store the list items in the strings.xml file also.
- CursorAdapter: It always accepts an instance of cursor as an input means
- SimpleAdapter: It mainly accepts a static data defined in the resources like array or database.
- BaseAdapter: It is a generic implementation for all three adapter types and it can be used in the views according to our requirements.

Now, we going to create an android application named as ListViewInKotlin using an arrayadapter. Open an activity_main.xml file from \res\layout path and write the code like as shown below.

activity_main.xml file

In this file, we declare the LisitView within LinearLayout and set its attributes. Later, we will access the ListView in Kotlin file using the id.

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:orientation="vertical">
    <ListView
        android:id="@+id/userlist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

MainActivity.kt

When we have created layout, we need to load the XML layout resource from our activity onCreate() callback method and access the UI element form the XML using findViewById.

```
import android.widget.ArrayAdapter
import android.widget.ListView
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // use arrayadapter and define an array
        val arrayAdapter: ArrayAdapter<*>
        val users = arrayOf(
            "Virat Kohli", "Rohit Sharma", "Steve Smith",
            "Kane Williamson", "Ross Taylor")
        // access the listView from xml file
        var mListView = findViewById<ListView>(R.id.userlist)
        arrayAdapter = ArrayAdapter(this,
            android.R.layout.simple_list_item_1, users)
        mListView.adapter = arrayAdapter
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:dist="http://schemas.android.com/apk/distribution"
    package="com.farizgaskin.myfirstkotlinapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Run the app to see the results

8 ScrollView

8.1 HorizontalScrollView in Kotlin

In Android ScrollView allows multiple views that are places within the parent view group to be scrolled. Scrolling in the android application can be done in two ways either Vertically or Horizontally.

In this article, we will be discussing how to create a Horizontal ScrollView in Kotlin.

Let's start by first creating a project in Android Studio. To do so, follow these instructions:

First step is to create a new Project in Android Studio. For this follow these steps:

- Click on File, then New and then New Project and give name whatever you like
- Then, select Kotlin language Support and click next button.
- Select minimum SDK, whatever you need
- Select Empty activity and then click finish.

Modify activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<HorizontalScrollView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <LinearLayout
            android:layout_width="200dp"
            android:layout_height="200dp"
            android:layout_marginTop="20dp"
            android:orientation="horizontal">


        <ImageView
                android:id="@+id/image1"
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_marginRight="20dp"
                android:src="@mipmap/image1"/>

        <ImageView
                android:id="@+id/image2"
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_marginRight="20dp"
                android:src="@mipmap/image2"/>

        <ImageView
                android:id="@+id/image3"
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_marginRight="20dp"
                android:src="@mipmap/image3"/>
```

```
        <ImageView
                android:id="@+id/image4"
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_marginRight="20dp"
                android:src="@mipmap/image4"/>

    </LinearLayout>
</HorizontalScrollView>
```

Add Images

We need to add some images which can be used for scrolling purpose. So, we have to copy the images from our local computer path to app/res/mipmap folder.

Note: We have added the images to mipmap folder instead of drawable folder because the size of the images is very large.

Create HorizontalScrollView in MainActivity.kt file

Open app/src/main/java/yourPackageName/MainActivity.kt and do the following changes:

```
package com.farizgaskin.myfirstKotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="i.apps.hscrollview">

    <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category
android:name="android.intent.category.LAUNCHER"/>
```

```
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Run the app to see the results

9 CardView

9.1 CardView in Android

CardView is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation. CardView is the view that can display views on top of each other. The main usage of CardView is that it helps to give a rich feel and look to the UI design. This widget can be easily seen in many different Android Apps. CardView can be used for creating items in listview or inside Recycler View. The best part about CardView is that it extends Framelayout and it can be displayed on all platforms of Android.

Step 1 : Create new Android Studio Project.

For creating new Android Studio Project. Click on File>New>New Project. Make sure to keep your language as JAVA and select Empty Activity.

Step 2 : Add material dependency in build.gradle file.

Navigate to Gradle Scripts then to build.gradle(app) and then add below dependency to it.

```
implementation 'com.google.android.material:material:1.2.1'
```

After adding this dependency you will get to see a popup option at top right corner which displays sync now. Click on that option to sync your project.

Step 3 : Now we will create a simple CardView.

Navigate to app>res>layout>activity_main.xml then create new CardView widget to it.

XML

```
<?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"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".MainActivity">

<androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:cardElevation="10dp"
    app:cardCornerRadius="20dp"
    android:layout_margin="10dp"
    app:cardBackgroundColor="@color/white"
    app:cardMaxElevation="12dp"
    app:cardPreventCornerOverlap="true"
    app:cardUseCompatPadding="true"
    >
```

```
    <!--
    In the above cardview widget
    cardelevation property will give elevation to your card view
    card corner radius will provide radius to your card view
    card background color will give background color to your card view
    card max elevation will give the cardview maximum elevation
    card prevent corner overlap will add padding to CardView on v20 and
before to prevent intersections between the Card content and rounded
corners.
    card use compact padding will add padding in API v21+ as well to have
the same measurements with previous versions.
    below are the two widgets imageview and text view we are displaying
inside our card view.
    -->
    <ImageView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_gravity="center"
        android:src="@drawable/fgimage"
        android:layout_margin="10dp"
        android:id="@+id/img"
        android:contentDescription="@string/app_name" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        android:layout_gravity="bottom|center_horizontal"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"
        android:textSize="20sp"
        android:textStyle="bold"
        />



</androidx.cardview.widget.CardView>

</RelativeLayout>
```

Run your app to see the results

9.2 How to create an Expandable CardView in Android

Expandable Cardview provides to create expansion panels without too much hassle and without writing boilerplate code. An expandable card-view becomes quite useful when it comes to an efficient and systematic presentation of data or information on the screen. It is used in a variety of apps, for example, the Contacts app or the Gallery app. Here, in this tutorial, we'll create a simple Expandable CardView in Android using Java.

Step 1: Create a New Project in Android Studio

1. Click on File, then New => New Project.
2. Choose "Empty Activity" for the project template.
3. Select language as Java.
4. Select the minimum SDK as per your need.

Step 2: Add the CardView Dependency

To be able to use the CardView element, you'll first have to add it's dependency in the project. In the build.gradle (Module: app) file add the following dependency and click on Sync now to synchronize the changes made.

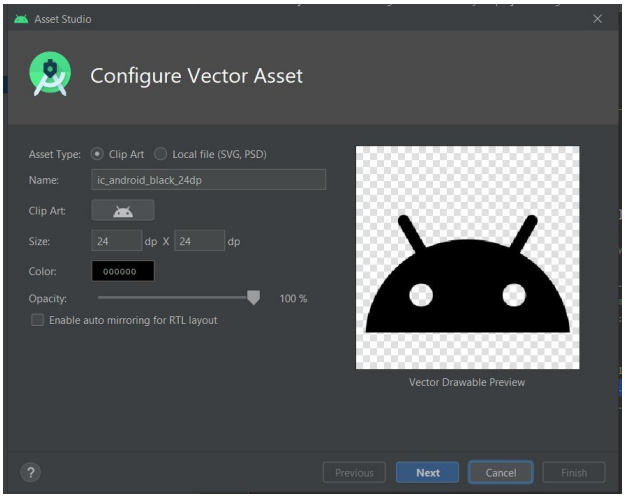
```
implementation("androidx.cardview:cardview:1.0.0")
```

Step 3: Add all the Required Drawable Resources in the Drawable Folder

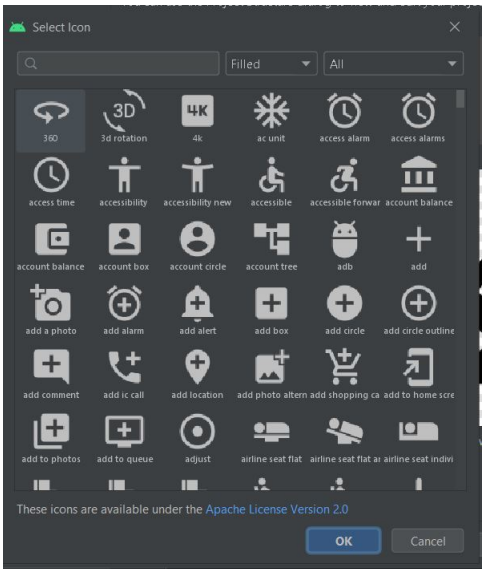
Choose the drawable resources as per the requirement. Here, in the CardView, use two images of the Google icons and 2 other icons to indicate either of the 'expand more' or 'expand less' options.

The expansion icons used here are imported as a vector asset from the Android Studio itself. The steps for the same are as follows:

1. Right-click on the drawable resource folder.
2. Go to new.
3. Click on Vector Asset.
4. The following box pops up. Click on the icon next to Clip Art.



From the variety of icons shown, choose the following two icons:




```
ic_baseline_expand_more_24
ic_baseline_expand_less_24
```

The following files get added to the drawable folder:

Ic_baseline_expand_more_24

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24"
    android:tint="?attr/colorControlNormal">
<path
    android:fillColor="@android:color/white"
    android:pathData="M16.59, 8.59L12, 13.17 7.41,
                      8.59 6, 10l6, 6 6, -6z"/>
</vector>
```

Ic_baseline_expand_less_24

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24"
    android:tint="?attr/colorControlNormal">
<path
    android:fillColor="@android:color/white"
    android:pathData="M12, 8l-6, 6 1.41, 1.41L12,
                      10.83l4.59, 4.58L18, 14z"/>
</vector>
```

Step 4: Modify the XML Layout

In the XML file, create the entire layout along with the portion that you want to be displayed after the CardView is expanded. The basic idea here is to set the visibility of the expandable element to 'gone' or 'visible'.

In the below layout, expanded the CardView to display a list of three subjects. The following is the code for activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#E4E3E3"
    tools:context=".MainActivity">

    <!--Base CardView-->
    <androidx.cardview.widget.CardView
        android:id="@+id/base_cardview"
        style="@style/Base.CardView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.473"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.021">

        <!--This is a ConstraintLayout for the entire CardView
                    including the expandable portion-->
        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_constraintBottom_toBottomOf="@+id/base_cardview"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintVertical_bias="0.511"
            tools:layout_editor_absoluteX="-55dp">

            <!--This is a ConstraintLayout for the fixed portion
                of the CardView. The elements
                that lie within the fixed portion of the CardView
                can be constrained to this layout.-->
            <androidx.constraintlayout.widget.ConstraintLayout
                android:id="@+id/fixed_layout"
                android:layout_width="match_parent"
                android:layout_height="150dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.0"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="0.0">
```

```
                <ImageView
                    android:id="@+id/icon"
                    android:layout_width="150dp"
                    android:layout_height="150dp"
                    android:src="@drawable/icon_one"

app:layout_constraintBottom_toBottomOf="@+id/fixed_layout"
                    app:layout_constraintEnd_toEndOf="@+id/fixed_layout"
                    app:layout_constraintHorizontal_bias="0.0"

app:layout_constraintStart_toStartOf="@+id/fixed_layout"
                    app:layout_constraintTop_toTopOf="@+id/fixed_layout"
                    app:layout_constraintVertical_bias="1.0" />

                <TextView
                    android:id="@+id/heading"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Farizgaskin"
                    android:textColor="#006600"
                    android:textSize="25dp"
                    android:textStyle="bold"

app:layout_constraintBottom_toBottomOf="@+id/fixed_layout"
                    app:layout_constraintEnd_toEndOf="@+id/fixed_layout"
                    app:layout_constraintHorizontal_bias="0.926"

app:layout_constraintStart_toStartOf="@+id/fixed_layout"
                    app:layout_constraintTop_toTopOf="@+id/fixed_layout"
                    app:layout_constraintVertical_bias="0.198" />

                <TextView
                    android:id="@+id/list_of_subjects"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="20dp"
                    android:layout_marginBottom="58dp"
                    android:text="List of subjects"
                    android:textSize="20dp"

app:layout_constraintBottom_toBottomOf="@+id/fixed_layout"
                    app:layout_constraintEnd_toEndOf="parent"
                    app:layout_constraintHorizontal_bias="0.878"
                    app:layout_constraintStart_toStartOf="parent"
                    app:layout_constraintTop_toBottomOf="@+id/heading"
                    app:layout_constraintVertical_bias="0.0" />

                <!--This is ImageButton for the expansion icon.-->
                <ImageButton
                    android:id="@+id/arrow_button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:src="@drawable/ic_baseline_expand_more_24"

app:layout_constraintBottom_toBottomOf="@id/fixed_layout"
                    app:layout_constraintEnd_toEndOf="parent"
                    app:layout_constraintHorizontal_bias="0.802"
                    app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/list_of_subjects"
```

```
                app:layout_constraintVertical_bias="0.0" />

        </androidx.constraintlayout.widget.ConstraintLayout>

        <!--The following is the expandable portion whose
            visibility is initially set to 'gone'.
            The parent LinearLayout contains 3 child LinearLayouts
            that hold a subject name and an icon each.-->
        <LinearLayout
            android:id="@+id/hidden_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:visibility="gone"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/fixed_layout">

            <!--Child LinearLayout 1-->
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">

                <ImageView
                    android:layout_width="50dp"
                    android:layout_height="50dp"
                    android:layout_marginStart="20dp"
                    android:layout_marginTop="10dp"
                    android:layout_marginEnd="10dp"
                    android:layout_marginBottom="10dp"
                    android:src="@drawable/gfg_icon_black" />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginStart="20dp"
                    android:layout_marginTop="20dp"
                    android:text="Database Management"
                    android:textColor="#000000"
                    android:textSize="20dp" />
            </LinearLayout>

            <!--Child LinearLayout 2-->
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">

                <ImageView
                    android:layout_width="50dp"
                    android:layout_height="50dp"
                    android:layout_marginStart="20dp"
                    android:layout_marginTop="10dp"
                    android:layout_marginEnd="10dp"
                    android:layout_marginBottom="10dp"
                    android:src="@drawable/gfg_icon_black" />

                <TextView
                    android:layout_width="wrap_content"
```

```
                        android:layout_height="wrap_content"
                        android:layout_marginStart="20dp"
                        android:layout_marginTop="20dp"
                        android:text="Data Structures"
                        android:textColor="#000000"
                        android:textSize="20dp" />
                </LinearLayout>

                <!--Child LinearLayout 3-->
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:orientation="horizontal">

                    <ImageView
                        android:layout_width="50dp"
                        android:layout_height="50dp"
                        android:layout_marginStart="20dp"
                        android:layout_marginTop="10dp"
                        android:layout_marginEnd="10dp"
                        android:layout_marginBottom="10dp"
                        android:src="@drawable/gfg_icon_black" />

                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_marginStart="20dp"
                        android:layout_marginTop="20dp"
                        android:text="Operating Systems"
                        android:textColor="#000000"
                        android:textSize="20dp" />
                </LinearLayout>
            </LinearLayout>

        </androidx.constraintlayout.widget.ConstraintLayout>
    </androidx.cardview.widget.CardView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 5: Modify the Java File

In the MainActivity.java, using the if-else statements, specify the conditions to manipulate the visibility of the expandable element.

```
package com.example.android.expandable_cardview;

import android.os.Bundle;
import android.transition.AutoTransition;
import android.transition.TransitionManager;
import android.view.View;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import androidx.appcompat.app.AppCompatActivity;
import androidx.cardview.widget.CardView;

public class MainActivity extends AppCompatActivity {

    ImageButton arrow;
```

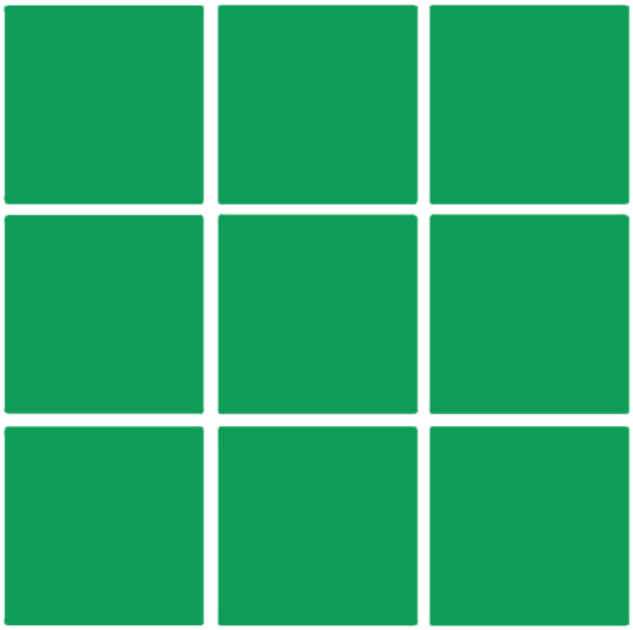
```
    LinearLayout hiddenView;
    CardView cardView;

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

        cardView = findViewById(R.id.base_cardview);
        arrow = findViewById(R.id.arrow_button);
        hiddenView = findViewById(R.id.hidden_view);

        arrow.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                // If the CardView is already expanded, set its
visibility
                //  to gone and change the expand less icon to expand
more.
                if (hiddenView.getVisibility() == View.VISIBLE) {

                    // The transition of the hiddenView is carried out
                    //  by the TransitionManager class.
                    // Here we use an object of the AutoTransition
                    // Class to create a default transition.
                    TransitionManager.beginDelayedTransition(cardView,
                                                    new AutoTransition());
                    hiddenView.setVisibility(View.GONE);

arrow.setImageResource(R.drawable.ic_baseline_expand_more_24);
                }

                // If the CardView is not expanded, set its visibility
                // to visible and change the expand more icon to expand
less.
                else {

                    TransitionManager.beginDelayedTransition(cardView,
                                                  new AutoTransition());
                    hiddenView.setVisibility(View.VISIBLE);

arrow.setImageResource(R.drawable.ic_baseline_expand_less_24);
                }
            }
        });
    }
}
```

Run the app to see the results

10 GridView

10.1 GridView in Android

A GridView is a type of AdapterView that displays items in a two-dimensional scrolling grid. Items are inserted into this grid layout from a database or from an array. The adapter is used for displaying this data, setAdapter() method is used to join the adapter with GridView. The main function of the adapter in GridView is to fetch data from a database or array and insert each piece of data in an appropriate item that will be displayed in GridView. This is how the GridView structure looks like. Note that we are going to implement this project using the Java language.



XML Attributes of GridView

android:numColumns: This attribute of GridView will be used to decide the number of columns that are to be displayed in Grid.
android:horizontalSpacing: This attribute is used to define the spacing between two columns of GridView.
android:verticalSpacing: This attribute is used to specify the spacing between two rows of GridView.

Example

Now let us see an example in which we will implement a simple GridView in Android App. In the GridView, we are now displaying the list of courses available on Farizgaskin.

Step 1: Creating a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language.

Step 2: Add google repository in the build.gradle file of the application project.

```
buildscript {
 repositories {
    google()
    mavenCentral()
}
```

All Jetpack components are available in the Google Maven repository, include them in the build.gradle file

```
allprojects {
 repositories {
    google()
   mavenCentral()
 }
}
```

Step 3: Modify the activity_main.xml file

Add GridView to the activity_main.xml file. Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:numColumns=2 is the number of columns for Grid View
         android:horizontalSpacing is the space between horizontal
         grid items.-->
    <GridView
        android:id="@+id/idGVcourses"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:horizontalSpacing="6dp"
        android:numColumns="2"
        android:verticalSpacing="6dp" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 4: Create an XML layout file for each item of GridView

Create an XML file for each grid item to be displayed in GridView. Click on the app > res > layout > Right-Click > Layout Resource file and then name the file as card_item. Below is the code for the card_item.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<!--XML implementation of Card Layout-->
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:layout_gravity="center"
    android:layout_margin="5dp"
    app:cardCornerRadius="5dp"
    app:cardElevation="5dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/idIVcourse"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_gravity="center"
            android:src="@mipmap/ic_launcher" />

        <TextView
            android:id="@+id/idTVCourse"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/app_name"
            android:textAlignment="center" />

    </LinearLayout>

</androidx.cardview.widget.CardView>
```

Step 5: Create a Modal Class for storing Data

Modal Class is the JAVA Class that handles data to be added in each GridView item of GridView. For Creating Modal Class.

Now click on app > java > apps package name > Right-Click on it.
Then Click on New > Java Class.
Name your Java Class file as CourseModel. Below is the code for the CourseModel.java file.

```
public class CourseModel {

    // string course_name for storing course_name
    // and imgid for storing image id.
    private String course_name;
    private int imgid;

    public CourseModel(String course_name, int imgid) {
        this.course_name = course_name;
        this.imgid = imgid;
    }

    public String getCourse_name() {
        return course_name;
    }

    public void setCourse_name(String course_name) {
```

```
        this.course_name = course_name;
    }

    public int getImgid() {
        return imgid;
    }

    public void setImgid(int imgid) {
        this.imgid = imgid;
    }
}
```

Step 6: Create an Adapter Class

Adapter Class adds the data from Modal Class in each item of GridView which is to be displayed on the screen. For Creating Adapter Class.

Now click on app > java > apps package name > Right-Click on it.
Then Click on New > Java Class.
Name your Java Class file as CourseGVAdapter. Below is the code for the CourseGVAdapter.java file.

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import java.util.ArrayList;

public class CourseGVAdapter extends ArrayAdapter<CourseModel> {
    public CourseGVAdapter(@NonNull Context context,
ArrayList<CourseModel> courseModelArrayList) {
        super(context, 0, courseModelArrayList);
    }
    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView,
@NonNull ViewGroup parent) {
        View listitemView = convertView;
        if (listitemView == null) {
            // Layout Inflater inflates each item to be displayed in
GridView.
            listitemView =
LayoutInflater.from(getContext()).inflate(R.layout.card_item, parent,
false);
        }
        CourseModel courseModel = getItem(position);
        TextView courseTV = listitemView.findViewById(R.id.idTVCourse);
        ImageView courseIV = listitemView.findViewById(R.id.idIVcourse);
        courseTV.setText(courseModel.getCourse_name());
        courseIV.setImageResource(courseModel.getImgid());
        return listitemView;
    }
}
```

Step 7: Modify the MainActivity.java file

Now in this file, we will perform all backend operations that will be adding data to GridView. Below is the code for the MainActivity.java file.

```
import android.os.Bundle;
import android.widget.GridView;
import androidx.appcompat.app.AppCompatActivity;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    GridView coursesGV;

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

        coursesGV = findViewById(R.id.idGVcourses);

        ArrayList<CourseModel> courseModelArrayList = new
ArrayList<CourseModel>();
        courseModelArrayList.add(new CourseModel("DSA",
R.drawable.ic_gfglogo));
        courseModelArrayList.add(new CourseModel("JAVA",
R.drawable.ic_gfglogo));
        courseModelArrayList.add(new CourseModel("C++",
R.drawable.ic_gfglogo));
        courseModelArrayList.add(new CourseModel("Python",
R.drawable.ic_gfglogo));
        courseModelArrayList.add(new CourseModel("Javascript",
R.drawable.ic_gfglogo));
        courseModelArrayList.add(new CourseModel("DSA",
R.drawable.ic_gfglogo));

        CourseGVAdapter adapter = new CourseGVAdapter(this,
courseModelArrayList);
        coursesGV.setAdapter(adapter);
    }
}
```

Run the app to see the results

11 Other Views

11.1 WebView

WebView is a view that display web pages inside the application. It is used to turn the application into a web application. MainActivity.java

MainActivity.java

```
package com.farizgaskin.webview;

import android.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Binding MainActivity.java with
        // activity_main.xml file
        setContentView(R.layout.activity_main);

        // Find the WebView by its unique ID
        WebView w = (WebView) findViewById(R.id.web);

        // loading http://www.google.com url in the the WebView.
        w.loadUrl("http://www.google.com");

        // this will enable the javascript.
        w.getSettings().setJavaScriptEnabled(true);

        // WebViewClient allows you to handle
        // onPageFinished and override Url loading.
        w.setWebViewClient(new WebViewClient());

    }
}
```

activity_main.xml

In the xml file only use of WebView is made inside RelativeLayout.

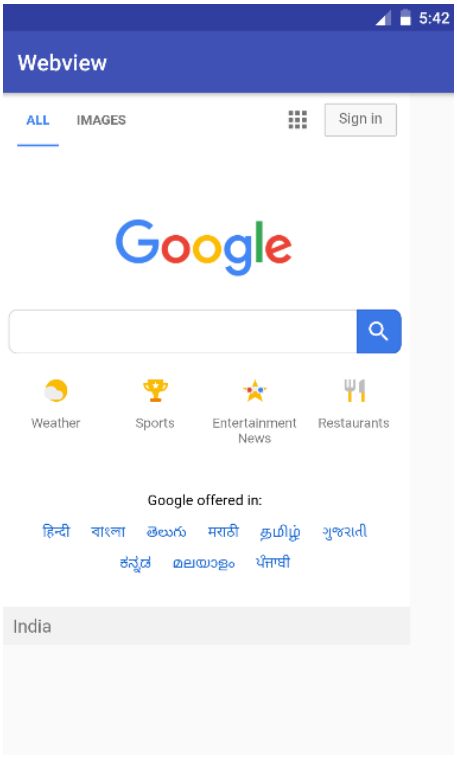
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.hp.webview.MainActivity">
```

```
    <WebView
        <!-- covers 368dp width as required. -->
        android:layout_width="368dp"

        <!-- unique ID of WebView -->
        android:id="@+id/web"

        <!-- covers 495dp height as required. -->
        android:layout_height="495dp"

        tools:layout_editor_absoluteX="8dp"
        tools:layout_editor_absoluteY="8dp" />
</RelativeLayout>
```

In AndroidManifest.xml, one needs to include below permission, in order to access internet:

```
"uses-permission android:name="android.permission.INTERNET"
```

Run the app to see the results



12 Buttons

12.1 Button in Kotlin

In Android applications, a Button is a user interface that is used to perform some action when clicked or tapped. It is a very common widget in Android and developers often use it. This lesson demonstrates how to create a button in Android Studio.

Example

In this example step by step demonstration of creating a Button will be covered. The application will consist of a button that displays a toast message when the user taps on it.

Step 1: Create a new project

1. Click on File, then New => New Project.
2. Choose “Empty Activity” for the project template.
3. Select language as Kotlin.
4. Select the minimum SDK as per your need.

Step 2: Modify the strings.xml file

Navigate to the strings.xml file under the “values” directory of the resource folder. This file will contain all strings that are used in the Application. Below is the appropriate code.

```
<resources>
    <string name="app_name">FG | Button In Kotlin</string>
    <string name="btn">Button</string>
    <string name="message">Hello World!! This is a Button.</string>
</resources>
```

Step 3: Modify the activity_main.xml file

Add a button widget in the layout of the activity. Below is the code of the activity_main.xml file which does the same.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#168BC34A"
    tools:context=".MainActivity">

    <!-- Button added in the activity -->
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:background="#4CAF50"
        android:paddingStart="10dp"
        android:paddingEnd="10dp"
        android:text="@string/btn"
        android:textColor="@android:color/background_light"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 4: Accessing the button in the MainActivity file

Add functionality of button in the MainActivity file. Here describe the operation to display a Toast message when the user taps on the button. Below is the code to carry out this task.

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.Toast

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // storing ID of the button
        // in a variable
        val button = findViewById<Button>(R.id.button)

        // operations to be performed
        // when user tap on the button
        button?.setOnClickListener()
        {
            // displaying a toast message
            Toast.makeText(this@MainActivity, R.string.message,
Toast.LENGTH_LONG).show() }
    }
}
```

Run the app to see the results

12.2 RadioButton in Kotlin

Android Radio Button is bi-state button which can either be checked or unchecked. Also, it's working is same as Checkbox except that radio button cannot allow to be unchecked once it was selected.

Generally, we use RadioButton controls to allow users to select one option from multiple options.

By default, the RadioButton in OFF(Unchecked) state but we can change the default state of RadioButton by using android:checked attribute.

Following steps to create new project-

- Click on File,then New => New Project.
- Then, check Include Kotlin Support and click next button.
- Select minimum SDK, whatever you need.
- Select Empty activity and then click finish.

Modify the strings.xml file

We can write the name of the application as RadioButtonInKotlin and write other strings which can be used.

```
<resources>
    <string name="app_name">RadioButtonInKotlin</string>
    <string name="checked">checked</string>
    <string name="unchecked">unchecked</string>
</resources>
```

Add RadioButtons in activity_main.xml file

In android, we use radio buttons inside RadioGroup to combine set of radio buttons into single group and it will make sure that user can select only button from the group of buttons.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/root_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
    <RadioGroup
        android:id="@+id/radio_group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#dbeceb"
        android:padding="15dp">

        <TextView
            android:id="@+id/title"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Which is your favorite color?"
            android:textStyle="bold"
            android:textSize="20sp"/>

        <RadioButton
            android:id="@+id/red"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="RED"
            android:onClick="radio_button_click"/>

        <RadioButton
            android:id="@+id/green"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="GREEN"
            android:onClick="radio_button_click"/>

        <RadioButton
            android:id="@+id/yellow"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="YELLOW"
            android:onClick="radio_button_click"/>

        <RadioButton
            android:id="@+id/pink"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="PINK"
            android:onClick="radio_button_click"/>
    </RadioGroup>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get Selected Color"/>
</LinearLayout>
```

Here, we are trying to implement a scenario where you need to select your favorite color. So, in activity_main.xml file, we have added 4 radio buttons inside a radio group. Each button represent a color. Now, only one radio button can be selected at a time. *

Now, we will access this widget in kotlin file and show a proper message whenever a radio button is selected.

Now, Open MainActivity.kt file and add below code into it.

```
package com.farizgaskin.myfirstkotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*
import kotlinx.android.synthetic.main.activity_main.*
```

```
import android.widget.RadioGroup

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Get radio group selected item using on checked change listener
        radio_group.setOnCheckedChangeListener(
            RadioGroup.OnCheckedChangeListener { group, checkedId ->
                val radio: RadioButton = findViewById(checkedId)
                Toast.makeText(applicationContext," On checked change :"+
                        " ${radio.text}",
                    Toast.LENGTH_SHORT).show()
            })
        // Get radio group selected status and text using button click
event
        button.setOnClickListener{
            // Get the checked radio button id from radio group
            var id: Int = radio_group.checkedRadioButtonId
            if (id!=-1){ // If any radio button checked from radio group
                // Get the instance of radio button using id
                val radio:RadioButton = findViewById(id)
                Toast.makeText(applicationContext,"On button click :" +
                        " ${radio.text}",
                    Toast.LENGTH_SHORT).show()
            }else{
                // If no radio button checked in this radio group
                Toast.makeText(applicationContext,"On button click :" +
                        " nothing selected",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }
    // Get the selected radio button text using radio button on click
listener
    fun radio_button_click(view: View){
        // Get the clicked radio button instance
        val radio: RadioButton =
findViewById(radio_group.checkedRadioButtonId)
        Toast.makeText(applicationContext,"On click : ${radio.text}",
            Toast.LENGTH_SHORT).show()
    }
}
```

In MainActivity.kt file, we have accessed radio group in which I have added four radio buttons. Then, we have set a listener to display toast message whenever radio button selection changes.

Since AndroidManifest.xml file is very important in any android application, we are also going to mention it here.

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.farizgaskin.myfirstkotlinapp">
```

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Run the app to see the results

12.3 RadioGroup in Kotlin

RadioGroup class of Kotlin programming language is used to create a container which holds multiple RadioButtons. The RadioGroup class is beneficial for placing a set of radio buttons inside it because this class adds multiple-exclusion scope feature to the radio buttons.

This feature assures that the user will be able to check only one of the radio buttons among all which belongs to a RadioGroup class. If the user checks another radio button, the RadioGroup class unchecks the previously checked radio button. This feature is very important when the developer wants to have only one answer to be selected such as asking the gender of a user.

Example

In this example step by step demonstration of creating a RadioGroup will be covered, it consists of 5 RadioButton children which ask the user to choose a course offered by Farizgaskin. When the user checks one of the RadioButtons and clicks the Submit button a toast message appears which displays the selected option.

Create new project

1. Click on File, then New => New Project.
2. Choose “Empty Activity” for the project template.
3. Select language as Kotlin.
4. Select the minimum SDK as per your need.

Open activity_main.xml file

Below is the code for activity_main.xml file to add a RadioGroup and its 5 RadioButton children. A normal submit button is also added to accept and display the response selected by the user.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#168BC34A"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/Heading"
        android:textAlignment="center"
        android:textColor="@android:color/holo_green_dark"
        android:textSize="36sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
```

```
        app:layout_constraintVertical_bias="0.19" />

    <RadioGroup
        android:id="@+id/radioGroup1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="#024CAF50"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.24000001">

        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/RadioButton1"
            android:textSize="24sp" />

        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/radioButton2"
            android:textSize="24sp" />

        <RadioButton
            android:id="@+id/radioButton3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/radioButton3"
            android:textSize="24sp" />

        <RadioButton
            android:id="@+id/radioButton4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/radioButton4"
            android:textSize="24sp" />

        <RadioButton
            android:id="@+id/radioButton5"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/radioButton5"
            android:textSize="24sp" />
    </RadioGroup>

    <Button
        android:id="@+id/submitButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#AB4CAF50"
        android:text="@string/submit_Button"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/radioGroup1"
        app:layout_constraintVertical_bias="0.17000002" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Open MainActivity.kt file

Below is the code for MainActivity.kt file to access RadioGroup widget in kotlin file and show a proper message whenever a radio button is selected by the user.

```
package com.example.sampleproject

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.RadioButton
import android.widget.RadioGroup
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    var radioGroup: RadioGroup? = null
    lateinit var radioButton: RadioButton
    private lateinit var button: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Display name of the Application
        title = "RadioGroup in Kotlin"

        // Assigning id of RadioGroup
        radioGroup = findViewById(R.id.radioGroup1)
        // Assigning id of Submit button
        button = findViewById(R.id.submitButton)

        // Actions to be performed
        // when Submit button is clicked
        button.setOnClickListener {

            // Getting the checked radio button id
            // from the radio group
            val selectedOption: Int = radioGroup!!.checkedRadioButtonId

            // Assigning id of the checked radio button
            radioButton = findViewById(selectedOption)

            // Displaying text of the checked radio button in the form of
toast
            Toast.makeText(baseContext, radioButton.text,
Toast.LENGTH_SHORT).show()
        }
    }
}
```

Modify strings.xml file

All the strings which are used in the activity, from the application name to the option of various RadioButtons are listed in this file.

```
<resources>
    <string name="app_name">RadioGroup in Kotlin</string>
    <string name="Heading">Choose a FG course</string>
    <string name="RadioButton1">Amazon SDE Test Series</string>
    <string name="radioButton2">Placement 100</string>
    <string name="radioButton3">C++ STL</string>
    <string name="radioButton4">CS Core Subjects</string>
    <string name="radioButton5">DSA Self paced</string>
    <string name="submit_Button">Submit</string>
</resources>
```

Run the app to see the results

12.4 CheckBox in Kotlin

A CheckBox is a special kind of button in Android which has two states either checked or unchecked. The Checkbox is a very common widget to be used in Android and a very good example is the "Remember me" option in any kind of Login activity of an app which asks the user to activate or deactivate that service.

There are many other uses of the CheckBox widget like offering a list of options to the user to choose from and the options are mutually exclusive i.e., the user can select more than one option. This feature of the CheckBox makes it a better option to be used in designing multiple-choice questions application or survey application in android.

Example

This example demonstrates the steps involved in designing an activity that consists of 5 CheckBox and an additional submit button to display a toast message that user response has been recorded.

Create new project

1. Click on File, then New => New Project.
2. Choose "Empty Activity" for the project template.
3. Select language as Kotlin.
4. Select the minimum SDK as per your need.

Open activity_main.xml file

Below is the code for activity_main.xml file to add 5 CheckBox. A normal "submit" button is also added to display a toast message that user response has been recorded.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#168BC34A"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/Heading"
        android:textAlignment="center"
        android:textColor="@android:color/holo_green_dark"
        android:textSize="36sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.17000002" />
```

```
    <LinearLayout
        android:id="@+id/checkBox_container"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.18">

        <CheckBox
            android:id="@+id/checkBox"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/checkBox1_text"
            android:textSize="18sp"
            android:padding="7dp"/>

        <CheckBox
            android:id="@+id/checkBox2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/checkBox2_text"
            android:textSize="18sp"
            android:padding="7dp"/>

        <CheckBox
            android:id="@+id/checkBox3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/checkBox3_text"
            android:textSize="18sp"
            android:padding="7dp"/>

        <CheckBox
            android:id="@+id/checkBox4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/checkBox4_text"
            android:textSize="18sp"
            android:padding="7dp"/>

        <CheckBox
            android:id="@+id/checkBox5"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/checkBox5_text"
            android:textSize="18sp"
            android:padding="7dp"/>
    </LinearLayout>

    <Button
        android:id="@+id/submitButton"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:background="#AB4CAF50"
        android:fontFamily="@font/roboto"
        android:text="@string/submitButton"
        android:textSize="18sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/checkBox_container"
        app:layout_constraintVertical_bias="0.23000002" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Open MainActivity.kt file

Below is the code for MainActivity.kt file to access CheckBox widget in kotlin file and show a proper message whenever the submit button is clicked by the user.

```
package com.example.checkboxinkotlin

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.Toast

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Assigning id of the submit button
        val button : Button = findViewById(R.id.submitButton)

        // Actions to be performed
        // when Submit button is clicked
        button.setOnClickListener{

            // Display toast message
            Toast.makeText(applicationContext,
                "Your response has been recorded",
Toast.LENGTH_LONG).show()
        }
    }
}
```

Modify strings.xml file

All the strings which are used in the activity, from the text view to the Checkbox texts are listed in this file.

```
<resources>
    <string name="app_name">CheckBox in Kotlin</string>
    <string name="Heading">Services provided by Fariz Gaskin</string>
    <string name="checkBox1">Coding contests</string>
```

```
    <string name="checkBox2_text">Civil Engineering Courses</string>
    <string name="checkBox1_text">Coding Contests</string>
    <string name="checkBox3_text">Computer Science Courses</string>
    <string name="checkBox4_text">Company specific coding
questions</string>
    <string name="checkBox5_text">Download movies</string>
    <string name="submitButton">SUBMIT</string>
</resources>
```

AndroidManifest.xml file

Below is the code for AndroidManifest.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http:// schemas.android.com/apk/res/android"
    package="com.example.checkboxinkotlin">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Run the app to see the results

13 Intent and Intent Filters

13.1 Implicit and Explicit Intents

Intent is an messaging object which passes between components like services, content providers, activities etc. Normally startActivity() method is used for invoking any activity.

Some of the general functions of intent are:

1. Start service
2. Launch activity
3. Display web page
4. Display contact list
5. Message broadcasting

Intent Classification

There are two types of intents

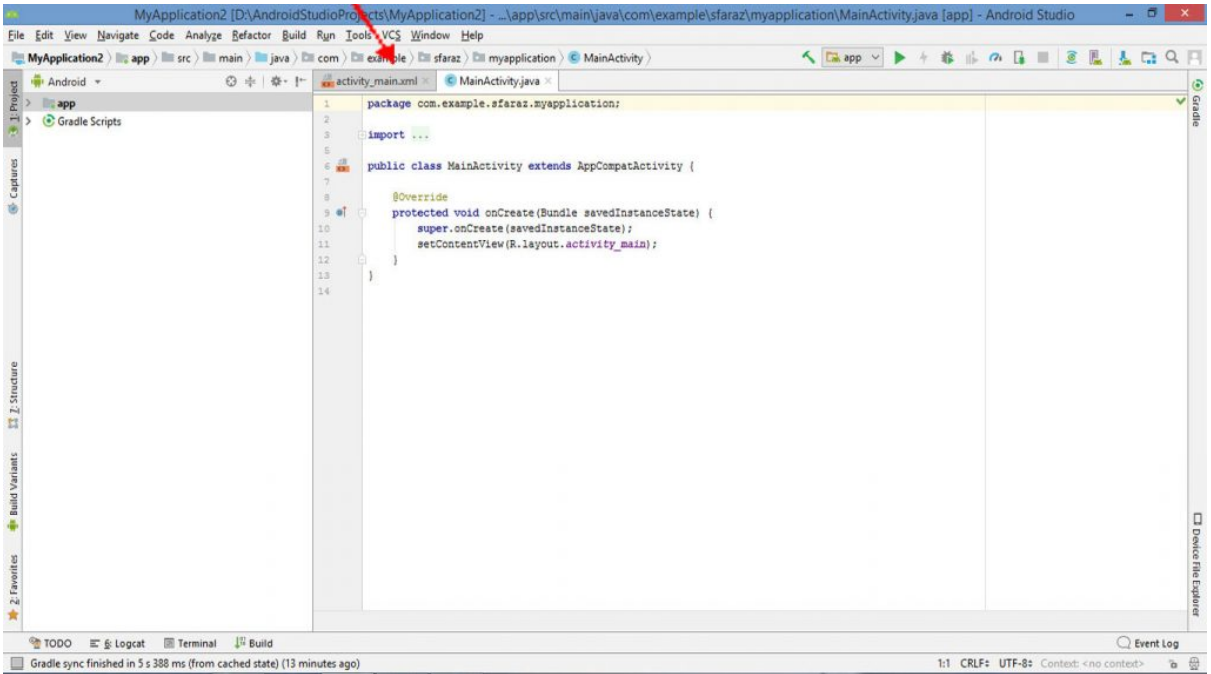
1. Implicit Intent
2. Explicit Intent

13.1.1 Implicit Intent

Using implicit Intent, component can't be specifying. An action to be performed is declared by implicit intent. Then android operating system will filter out component which will response to the action.

How to create an Android App to open a webpage using implicit intent

Step 1: Create XML file and Java File. Please refer the pre-requisites to learn more about this step.



Step 2: Open “activity_main.xml” file and add following widgets in a Constraint Layout.

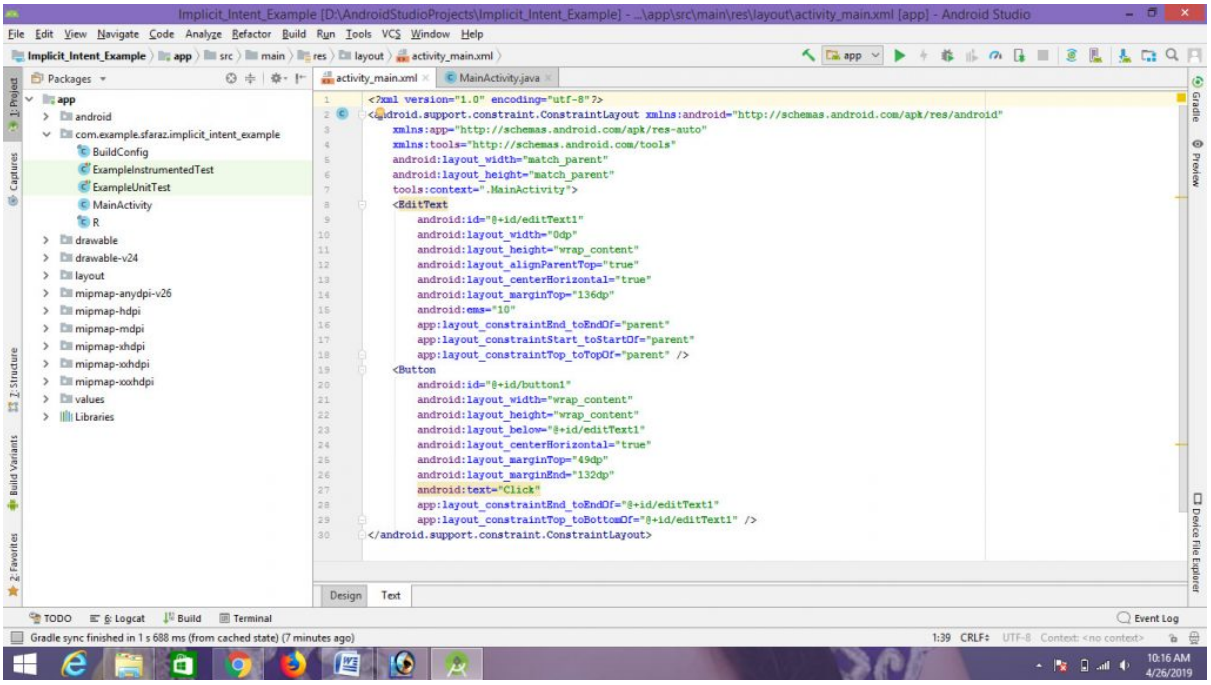
1. An EditText to input text.
2. A Button to open webpage.

Also, Assign IDs for each component as shown in the image and the code below. The assigned ID for a component helps in the identification of component and can be easily use in the Java files.

```
android:id="@+id/id_name"
```

Here the given IDs: editText1, button1

This will make the UI of the Application.



Step 3: Now, after the UI, this step will create the Backend of the App. For this, Open “MainActivity.java” file and instantiate the component (Button) created in the XML file using findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

```
ComponentType object =
(ComponentType)findViewById(R.id.IdOfTheComponent);
```

Syntax for used components (EditText, Button):

```
EditText editText = (EditText)findViewById(R.id.editText1);
Button button = (Button)findViewById(R.id.button1);
```

Step 4: This step involves setting up the operations to display the Toast Message. These operations are as follows:

a). First Add the listener on button and this button will open webpage.

```
button.setOnClickListener(new View.OnClickListener() {});
```

b). Create String type variable to store the value of 'EditText'. Value is accepted and converted to string.

```
String url = editText1.getText().toString();
```

c). Create an Intent object MainActivity.java class to open the webpage.

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
```

d). The startActivity() method starts to call a webpage for opening specified by the intent.

```
startActivity(intent);
```

Therefore the code snippet for Implicit Intent:

```
String url = editText1.getText().toString();
Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse(url));
startActivity(intent);
```

Complete code for activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

<!-- add an edittext to input text -->
    <EditText
        android:id="@+id/editText1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
```

```
        android:layout_centerHorizontal="true"
        android:layout_marginTop="136dp"
        android:ems="10"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<!-- add a button for click -->
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="49dp"
        android:layout_marginEnd="132dp"
        android:text="Click"
        app:layout_constraintEnd_toEndOf="@+id/editText1"
        app:layout_constraintTop_toBottomOf="@+id/editText1" />

</android.support.constraint.ConstraintLayout>
```

Complete code for MainActivity.java

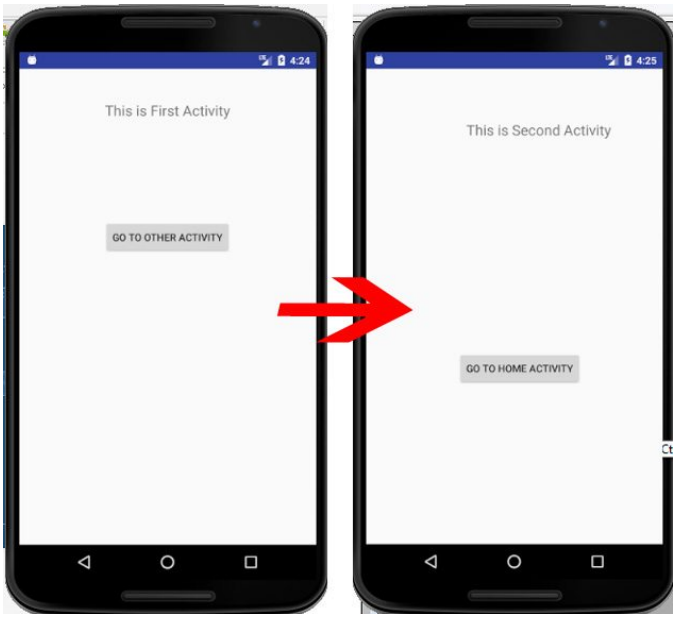
```
package org.farizgaskin.implicitIntent_example;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {

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

        // Bind the components to their respective objects
        // by assigning their IDs
        // with the help of findViewById() method
        final EditText editText1 =
(EditText)findViewById(R.id.editText1);
        Button button = (Button)findViewById(R.id.button1);

        // implementation of onClick event for Implicit Intent
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {

                // performing webpage open action
                String url = editText1.getText().toString();
```

```
                Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse(url));
                startActivity(intent);
            }
        });
    }
}
```

Run the app to see the results

13.1.2 Explicit Intent

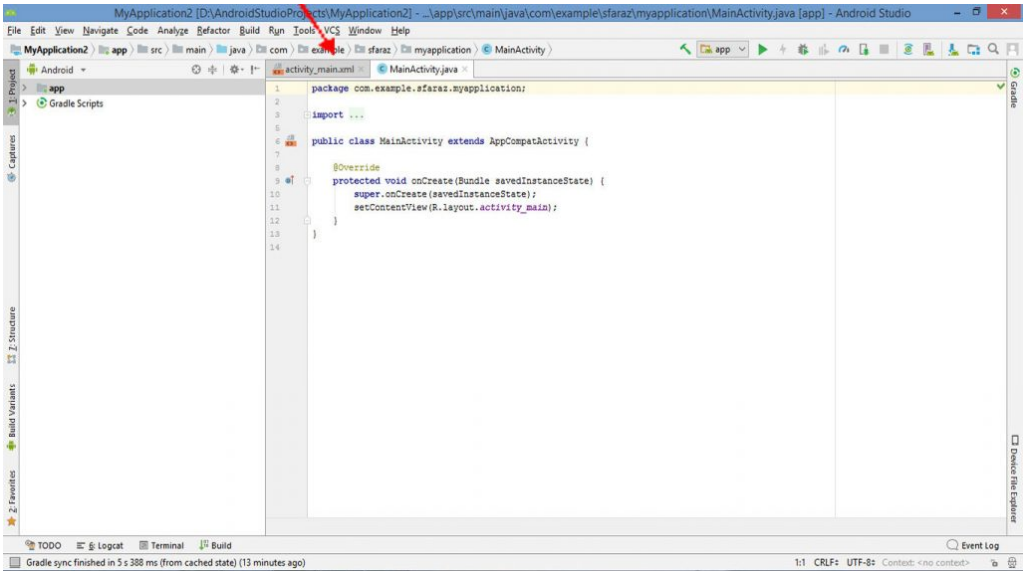
Using explicit intent any other component can be specified. In other words, the targeted component is specified by explicit intent. So only the specified target component will be invoked.



In the above example, There are two activities (FirstActivity, SecondActivity). When you click on 'GO TO OTHER ACTIVITY' button in the first activity, then you move to second activity. When you click on 'GO TO HOME ACTIVITY' button in the second activity, then you move to the first activity. This is getting done through Explicit Intent.

How to create an Android App to move to next activity using Explicit Intent

Step 1: Create XML file and Java File. Please refer the pre-requisites to learn more about this step.



Step 2: Open “activity_main.xml” file and add following widgets in a Constraint Layout.

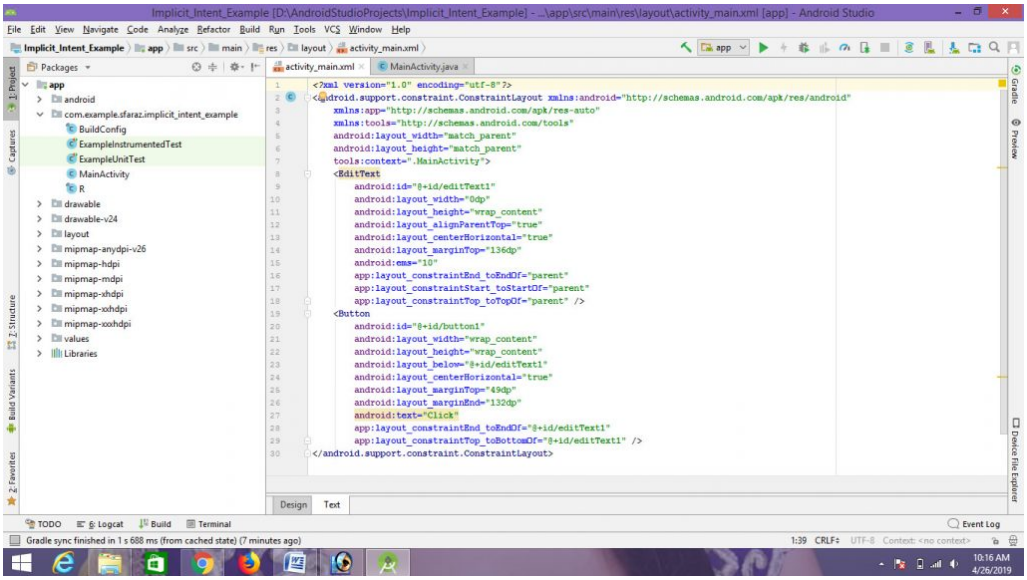
1. A Button for moving to second activity.
2. A TextView for writing some text.

Also, Assign IDs for each component (Button, TextView) as shown in the image and the code below. The assigned ID for a component helps in the identification of component and can be easily use in the Java files.

```
android:id="@+id/id_name"
```

Here the given IDs: Button01, TextView01.

This will make the UI of the Application.



Step 3: Now, after the UI, this step will create the Backend of the App. For this, Open "MainActivity.java" file and instantiate the component (Button, TextView) created in the XML file using findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

```
ComponentType object =
(ComponentType)findViewById(R.id.IdOfTheComponent);
```

Syntax for used components (Button, TextView):

```
Button button = (Button)findViewById(R.id.Button01);
TextView textView = (TextView)findViewById(R.id.TextView01);
```

Step 4: This step involves setting up the operations to create explicit intent. These operations are as follows:

a. First Add the listener on button and using this button you will move to other activity.

```
button1.setOnClickListener(new View.OnClickListener() {});
```

b. Now, Create an Intent.

```
Intent i = new Intent(getApplicationContext(), SecondActivityName.class);
```

Parameters: This is having two parameters:

- getApplicationContext(): Returns the context for the entire application.
- OtherActivityName.class: You should use your activity name here.

Therefore the code to make an explicit intent is:

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
```

c. Now Start the Targeted Activity.

startActivity(i):This starts target activity.

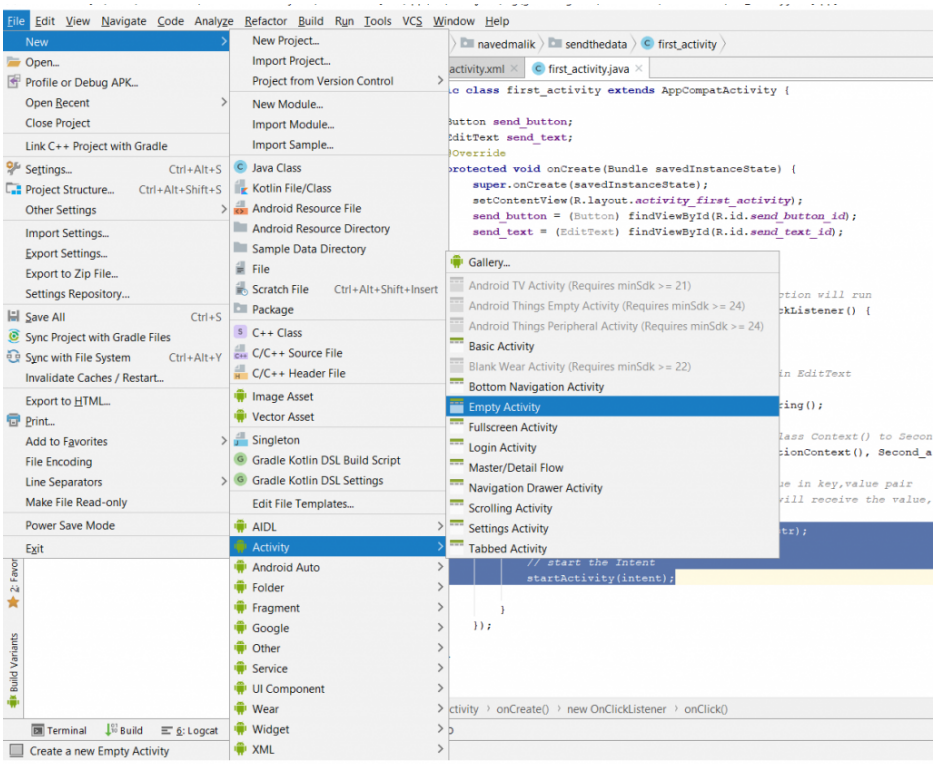
The code snippet for explicit intent:

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

Step 5: Now we have to create a second activity as a destination activity.

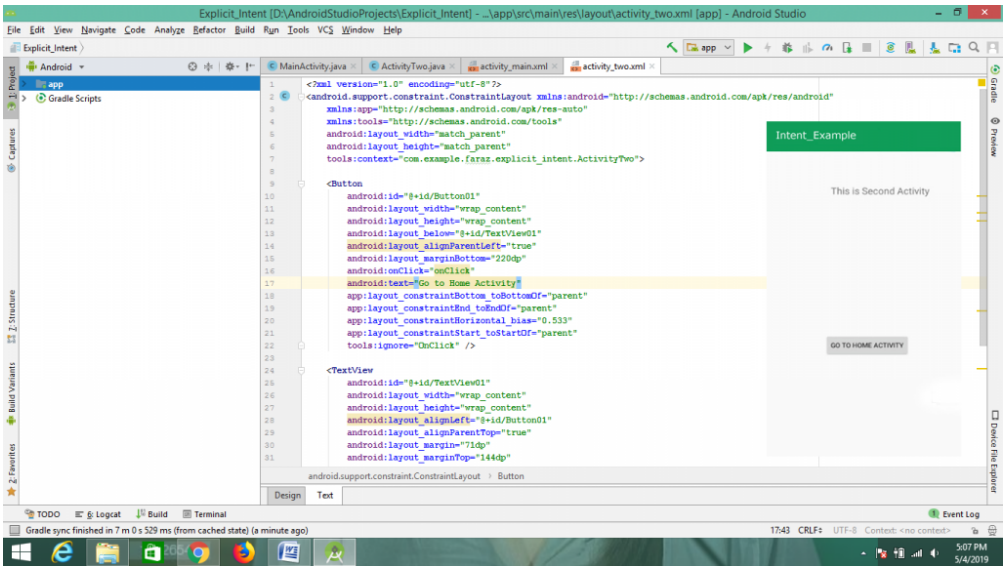
The steps to create the second activity is as follows:

```
android project > File > new > Activity > Empty Activity
```



Step 6: Now open your second xml file.

Add Button and TextView for moving back to home activity and to write some text on activity respectively. Assign ID to Button and Textview. Second Activity is shown below:



Step 7: Now, open up the your second activity java file and perform the following operation.

a. First Add the listener on button and using this button you will move to home activity.

```
button1.setOnClickListener(new View.OnClickListener() {});
```

b. Now, Create an Intent.

```
Intent i = new Intent(getApplicationContext(), OtherActivityName.class);
```

Therefore the code to make an explicit intent is:

```
Intent i = new Intent(getApplicationContext(), MainActivity.class);
```

c. Now Start the Targeted Activity.

startActivity(i):This starts target activity.

The code to show the Toast message:

```
Intent i = new Intent(getApplicationContext(), MainActivity.class);
startActivity(i);
```

Step 8: Now Run the app and operate as follows:

- When the app is opened, it displays a “GO TO OTHER ACTIVITY” button.
- Click the “GO TO OTHER ACTIVITY” button.
- Then you will move to second activity.
- Also, you will get a “GO TO HOME ACTIVITY” button on the second activity, When you will click this, you will move to home activity.

Complete code for activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="org.farizgaskin.explicit_intent.MainActivity">

 <!-- add a button to move to next activity -->
    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/TextView01"
        android:layout_marginTop="209dp"
        android:onClick="onClick"
        android:text="Go To Other Activity"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="OnClick" />

 <!-- add a TextView to write some text on activity -->
    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/Button01"
        android:layout_alignParentTop="true"
        android:layout_marginTop="44dp"
        android:minHeight="60dip"
        android:text="This is First Activity"
        android:textSize="20sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Complete code for MainActivity.java

```
package org.farizgaskin.explicit_intent;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
    // Defining the object for button
    Button button1;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Bind the components to their respective objects
        // by assigning their IDs
        // with the help of findViewById() method
        button1 = (Button)findViewById(R.id.Button01);

        button1.setOnClickListener(new OnClickListener() {
            public void onClick(View view)
            {

                // Creating explicit intent
```

```
                Intent i = new Intent(getApplicationContext(),
                                      ActivityTwo.class);
                startActivity(i);
            }
        });
    }
}
```

Complete code for activity_two.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="org.farizgaskin.explicit_intent.ActivityTwo">

<!-- add a button for moving to home activity -->
    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/TextView01"
        android:layout_alignParentLeft="true"
        android:layout_marginBottom="220dp"
        android:onClick="onClick"
        android:text="Go to Home Activity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.533"
        app:layout_constraintStart_toStartOf="parent"
        tools:ignore="OnClick" />

 <!-- add a TextView to write some text on activity -->
    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/Button01"
        android:layout_alignParentTop="true"
        android:layout_margin="71dp"
        android:layout_marginTop="144dp"
        android:layout_marginEnd="88dp"
        android:layout_marginRight="88dp"
        android:minHeight="60dip"
        android:text="This is Second Activity"
        android:textSize="20sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Complete code for ActivityTwo.java

```
package org.farizgaskin.explicit_intent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class ActivityTwo extends Activity {

    // Defining the object for button
    Button button1;
    @Override
    public void onCreate(Bundle bundle)
    {
        super.onCreate(bundle);
        setContentView(R.layout.activity_two);

        // Bind the components to their respective objects
        // by assigning their IDs
        // with the help of findViewById() method
        button1 = (Button)findViewById(R.id.Button01);

        button1.setOnClickListener(new OnClickListener() {
            public void onClick(View view)
            {

                // Creating explicit intent
                Intent i = new Intent(getApplicationContext(),
                                      MainActivity.class);
                startActivity(i);
            }
        });
    }
}
```

Run the app to see the results

13.2 How to send message on Whatsapp

Whatsapp is the one of most popular messaging App. Many android applications need the functionality to share some messages directly from their app to WhatsApp. For example, if a user wants to share the app or share a message from the app then this functionality comes in use.

Either user can send a text or a predefined text can also be sent through this. This article demonstrates how an android application can send messages on WhatsApp. As a pre-requisite, Whatsapp must be installed on the user's device.

In this lesson, we will try to create an android app that sends message on WhatsApp using Kotlin.

Step 1: Modify activity_main.xml file

Open the activity_main.xml file and add the layout code. activity_main.xml contains a Linear Layout which contains a EditText to input the message and a Button to submit the message.

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:context=".MainActivity"
      android:orientation="vertical">

  <!--EditText to take message input from user-->
  <EditText
      android:id="@+id/message"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="16dp"
      android:hint="Enter you message here"
      android:lines="8"
      android:inputType="textMultiLine"
      android:gravity="left|top"
      />

  <!--Button to send message on Whatsapp-->
  <Button
      android:id="@+id/submit"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal"
      android:text="Submit"
      android:background="@color/colorPrimary"
      android:textColor="@android:color/white"/>
</LinearLayout>
```

Step 2: Working with MainActivity.kt file

Take the reference of EditText and Button in Kotlin file. References are taken using the ids with the help of findViewById() method.

```
// Referencing the Edit Text
val messageEditText = findViewById<EditText>(R.id.message)
```

Similarly taking reference of button

```
// Referencing the button
val submit = findViewById<Button>(R.id.submit)
```

Write a function to send message to WhatsApp. Create an intent with ACTION_SEND and specify the whatsapp package name to this so that it opens directly. com.whatsapp is the package name for official WhatsApp application.

```
fun sendMessage(message:String){

     // Creating intent with action send
     val intent = Intent(Intent.ACTION_SEND)

     // Setting Intent type
     intent.type = "text/plain"

     // Setting whatsapp package name
     intent.setPackage("com.whatsapp")

     // Give your message here
     intent.putExtra(Intent.EXTRA_TEXT, message)

     // Checking whether whatsapp is installed or not
     if (intent.resolveActivity(packageManager) == null) {
         Toast.makeText(this,
                        "Please install whatsapp first.",
                        Toast.LENGTH_SHORT).show()
         return
     }

     // Starting Whatsapp
     startActivity(intent)
 }
```

Set the click listener using setOnClickListener on the button to send the message. Get the text entered by the user and call the function to send message on whatsapp.

```
// Setting on click listener
submit.setOnClickListener {
    val message = messageEditText.text.toString()

// Calling the function
    sendMessage(message);
}
```

Complete MainActivity.kt

```
package com.gfg

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Referencing the Edit Text
        val messageEditText = findViewById<EditText>(R.id.message)
        // Referencing the button
        val submit = findViewById<Button>(R.id.submit)

        // Setting on click listener
        submit.setOnClickListener {
            val message = messageEditText.text.toString()

        // Calling the function
        sendMessage(message);
        }
    }

    fun sendMessage(message: String) {

        // Creating intent with action send
        val intent = Intent(Intent.ACTION_SEND)

        // Setting Intent type
        intent.type = "text/plain"
        // Setting whatsapp package name
        intent.setPackage("com.whatsapp")
        // Give your message here
        intent.putExtra(Intent.EXTRA_TEXT, message)

        // Checking whether whatsapp is installed or not
        if (intent.resolveActivity(packageManager) == null) {
            Toast.makeText(this,
                           "Please install whatsapp first.",
                           Toast.LENGTH_SHORT).show()
            return
        }

        // Starting Whatsapp
        startActivity(intent)
    }
}
```

Run the app to see the results

14 Toast

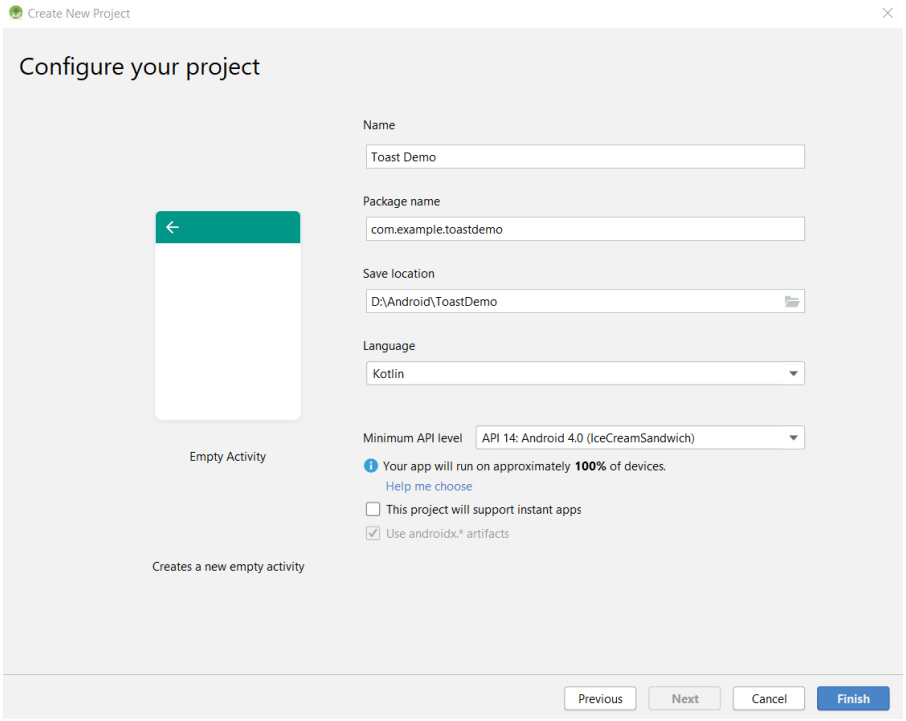
14.1 Toast in Kotlin

A Toast is a short alert message shown on the Android screen for a short interval of time. Android Toast is a short popup notification which is used to display an information when we perform any operation in our app.

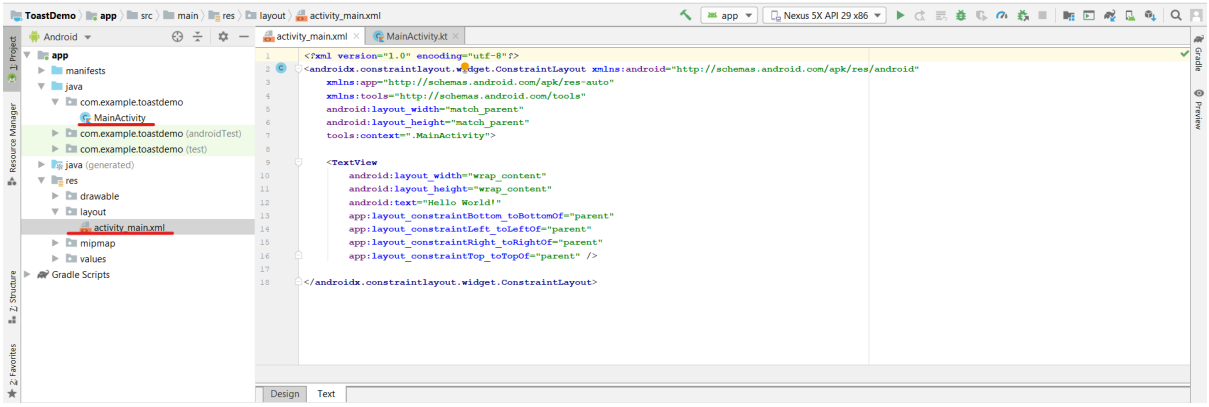
In this tutorial, we shall not just limit us by creating a lame toast but also do some user interactive stuff.

First, we shall create a screen with an Edit Text (Text box to take input) and a Button.

Step 1: Create a project in Android studio with an empty activity and make sure that you have configured the app with Kotlin language.



Step 2: In the project window on the left side, navigate to app > res > layout > activity_main.xml.



Here we are going to add an Edit Text and a button. So, remove the hello word Text View in the XML.

Now your XML should look like this.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 3: Add an Edit text to the layout.

```
<EditText
    android:id="@+id/messageEditText"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="Message to toast"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Step 4: Add a button to the layout.

```
<Button
   android:id="@+id/toastButton"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:text="Toast"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintHorizontal_bias="0.5"
   app:layout_constraintTop_toTopOf="parent" />
```

Step 5: Let's align the edit text and button. To make the components linked in Android we have constraints. We shall keep the Button at the end of Edit Text and Edit Text at the beginning of Button.

Add the following for Button.

```
app:layout_constraintStart_toEndOf="@+id/messageEditText"
```

Add the following for EditText.

```
app:layout_constraintEnd_toStartOf="@+id/toastButton"
```

Save the XML file and check the design tab (you can find it at the bottom left corner of editor tab). You can see the EditText and Button aligned on top of the application.

Step 6: Now, Let's go to the Kotlin class file and make the toast happen.
Open the Kotlin class file in app > java > com.example.toastdemo > MainActivity.kt

Add the following method to the class.

```
fun toastMessage(view: View)
{
    val messageEditText = findViewById<EditText>(R.id.messageEditText)
    val message = messageEditText.text.toString()
    var toast = Toast.makeText(this, message, Toast.LENGTH_LONG)
    toast.show()
}
```

Here, messageEditText is the object for EditText we have created in the Layout file in Step 3. To get the contents of EditText we have used the text property and stored it in a variable message.

Making a Toast – here is the heart of this tutorial,

```
var toast = Toast.makeText(this, message, Toast.LENGTH_LONG)
```

We are storing the Toast in a variable toast.To make the Toast visible we should call toast.show()

Step 7: Now we shall add an on click event listener to the button and make it to call our toastMessage() method when a click/tap was done to the button.
Go back to the activity_main.xml layout file and add this attribute to the Button

```
android : onClick = "toastMessage"
```

And that's all we are done! Now run the App, enter a message in edit text and click the button. It will toast the message.

Hiding the Android Soft Keyboard after clicking the Toast button to see a clear Toast

```
fun hideKeyboard(activity: Activity)
{
    val view = activity.findViewById<View>(android.R.id.content) if (view
!= null)
    {
        val imm = activity.getSystemService(Context.INPUT_METHOD_SERVICE)
```

```
                as InputMethodManager
            imm.hideSoftInputFromWindow(view.windowToken, 0)
    }
}
```

Add the above method to the main activity class and call it before making toast.

Run the app to see the results

15 RecyclerView

15.1 RecyclerView using ListView

RecyclerView is more flexible and advanced version of ListView and GridView. RecyclerView is used for providing a limited window to a large data set, which means it is used to display a large amount of data that can be scrolled very efficiently by maintaining a limited number of Views. In RecyclerView we supply data and define how each item looks, and the RecyclerView library dynamically creates the content when it is needed. RecyclerView was introduced in Material Design in Android 5.0(API level 21.0).

How RecyclerView Works?

- RecyclerView is a ViewGroup that contains Views corresponding to your data. It itself a View so, it is added to the layout file as any other UI element is added.
- ViewHolder Object is used to define each individual element in the list. View holder does not contain anything when it created, RecyclerView binds data to it. ViewHolder is defined by extending RecyclerView.ViewHolder.
- Adapters are used to bind data to the Views. RecyclerView request views, and binds the views to their data, by calling methods in the adapter. The adapter can be defined by extending RecyclerView.Adapter.
- LayoutManager arranges the individual elements in the list. It contains the reference of all views that are filled by the data of the entry.

Example
In this example, we are going to use RecyclerView as ListView. Here, we are going to display the list of courses with their images as a vertical list using RecyclerView.

Step by Step Implementation

Step 1: Create A New Project

Note that select Java as the programming language.

Step 2: Add Dependency

We are going to use RecyclerView as ListView. So, we need to add the dependency for it. For adding the dependency Go to Gradle Scripts > build.gradle(Module: app) and add the following dependency. After adding the dependency click on Sync Now.

```
dependencies {
  implementation 'androidx.recyclerview:recyclerview:1.1.0'
}
```

Before moving further let's add some color attributes in order to enhance the app bar. Go to app > res > values > colors.xml and add the following color attributes.

```
<resources>
    <color name="colorPrimary">#0F9D58</color>
    <color name="colorPrimaryDark">#16E37F</color>
    <color name="colorAccent">#03DAC5</color>
</resources>
```

Step 3: Working with the activity_main.xml file

In this step, we will add a RecyclerView to our activity_main.xml file which is used to display data of listItems. Go to the app > res > layout > activity_main.xml and the following code snippet.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <!--RecyclerView-->
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Step 4: Create a new layout file list_item.xml

In this step, we will create a new layout file for the single list item view. Go to app > res > layout > right-click > New > Layout Resource File and name it as list_item. list_item.xml contains an ImageView and a TextView which is used for populating the RecyclerView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <!--For image src we have used ic_launcher
        and for text "courseName" they are used
        only for reference how it will looks"-->
    <ImageView
        android:id="@+id/courseImg"
        android:layout_width="72dp"
        android:layout_height="72dp"
        android:padding="8dp"
        android:src="@mipmap/ic_launcher_round" />
```

```
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_gravity="center">
        <TextView
            android:id="@+id/courseName"
            android:text="courseName"
            android:textStyle="bold"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    </LinearLayout>

</LinearLayout>
```

Step 5: Create a new Adapter class

Now, we will create an Adapter class that acts as a bridge between the UI Component and the Data Source .i.e., courseImg, courseName, and RecyclerView. Go to the app > java > package > right-click and create a new java class and name it as Adapter. Below is the code snippet is given for it.

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.ArrayList;

// Extends the Adapter class to RecyclerView.Adapter
// and implement the unimplemented methods
public class Adapter extends RecyclerView.Adapter<Adapter.ViewHolder> {
    ArrayList courseImg, courseName;
    Context context;

    // Constructor for initialization
    public Adapter(Context context, ArrayList courseImg, ArrayList
courseName) {
        this.context = context;
        this.courseImg = courseImg;
        this.courseName = courseName;
    }

    @NonNull
    @Override
    public Adapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        // Inflating the Layout(Instantiates list_item.xml
          // layout file into View object)
        View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.list_item,
parent, false);

        // Passing view to ViewHolder
```

```
        Adapter.ViewHolder viewHolder = new Adapter.ViewHolder(view);
        return viewHolder;
    }

    // Binding data to the into specified position
    @Override
    public void onBindViewHolder(@NonNull Adapter.ViewHolder holder, int
position) {
        // TypeCast Object to int type
        int res = (int) courseImg.get(position);
        holder.images.setImageResource(res);
        holder.text.setText((String) courseName.get(position));
    }

    @Override
    public int getItemCount() {
        // Returns number of items
          // currently available in Adapter
        return courseImg.size();
    }

    // Initializing the Views
    public class ViewHolder extends RecyclerView.ViewHolder {
        ImageView images;
        TextView text;

        public ViewHolder(View view) {
            super(view);
            images = (ImageView) view.findViewById(R.id.courseImg);
            text = (TextView) view.findViewById(R.id.courseName);
        }
    }
}
```

Step 6: Working with the MainActivity file

In MainActivity.java class we create two ArrayList for storing courseImg and courseName. These images are placed in the drawable folder(app > res > drawable). You can use any images in place of these. And then we get the reference RecyclerView and set the LayoutManager as LinearLayoutManager and Adapter, to show items in RecyclerView. Below is the code for the MainActivity.java file.

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.recyclerview.widget.StaggeredGridLayoutManager;
import java.util.ArrayList;
import java.util.Arrays;

public class MainActivity extends AppCompatActivity {

    RecyclerView recyclerView;

    // Using ArrayList to store images data
```

```
    ArrayList courseImg = new
ArrayList<>(Arrays.asList(R.drawable.data_structure,
R.drawable.c_plus_plus,

R.drawable.c_hash, R.drawable.java_script,
                                                        R.drawable.java,
R.drawable.c,
                                                        R.drawable.html,
R.drawable.css));
    ArrayList courseName = new ArrayList<>(Arrays.asList("Data
Structure", "C++", "C#", "JavaScript", "Java",
                                                         "C-Language",
"HTML 5", "CSS"));

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

        // Getting reference of recyclerView
        recyclerView = (RecyclerView) findViewById(R.id.recyclerView);

        // Setting the layout as linear
          // layout for vertical orientation
        LinearLayoutManager linearLayoutManager = new
LinearLayoutManager(getApplicationContext());
        recyclerView.setLayoutManager(linearLayoutManager);

        // Sending reference and data to Adapter
        Adapter adapter = new Adapter(MainActivity.this, courseImg,
courseName);

        // Setting Adapter to RecyclerView
        recyclerView.setAdapter(adapter);
    }
}
```

Run the app to see the results

15.2 Pull to Refresh

The SwipeRefreshLayout widget is used for implementing a swipe-to-refresh user interface design pattern. Where the user uses the vertical swipe gesture to refresh the content of the views. The vertical swipe is detected by the SwipeRefreshLayout widget and it displays a distinct progress bar and triggers the callback methods in the app.

In order to use this behavior, we need to use the SwipeRefreshLayout widget as the parent of a ListView or GridView. These Material Design UI Patterns are seen in applications like Gmail, Youtube, Facebook, Instagram, etc. It allows the user to refresh the application manually. SwipeRefreshLayout class contains a listener called OnRefreshListener.

The classes which want to use this listener should implement SwipeRefreshLayout.OnRefreshListener interface. On vertical swipe-down gesture, this listener is triggered and onRefresh() method is called and can be overridden according to the needs.

Example

In this example, we would store data into the ArrayList which is used for populating the RecyclerView. Whenever onRefresh() method is called the ArrayList data gets rearranged. Note that we are going to implement this project using the Java language.

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language.

Step 2: Adding dependencies

We are going to use RecyclerView and SwipeRefreshLayout. So, we need to add dependencies for them. For adding these dependencies Go to Gradle Scripts > build.gradle(Module: app) and add the following dependencies. After adding these dependencies you need to click on Sync Now.

```
dependencies {
   implementation "androidx.recyclerview:recyclerview:1.1.0"
   implementation "androidx.swiperefreshlayout:swiperefreshlayout:1.1.0"
}
```

Before moving further let's add some color attributes in order to enhance the app bar. Go to app > res > values > colors.xml and add the following color attributes.

```
<resources>
    <color name="colorPrimary">#0F9D58</color>
    <color name="colorPrimaryDark">#16E37F</color>
    <color name="colorAccent">#03DAC5</color>
</resources>
```

Step 3: Working with the activity_main.xml file

In this step, we will create SwipeRefreshLayout and Add RecyclerView to it. Go to app > res > layout > activity_main.xml and add the following code snippet.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.swiperefreshlayout.widget.SwipeRefreshLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/swipeRefreshLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</androidx.swiperefreshlayout.widget.SwipeRefreshLayout>
```

Step 4: Create a new layout file list_item.xml for the list items of RecyclerView.

Go to the app > res > layout > right-click > New > Layout Resource File and name it as list_item. list_item.xml layout file contains an ImageView and a TextView which is used for populating the rows of RecyclerView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:orientation="horizontal"
    android:padding="10dp">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="120dp"
        android:layout_height="120dp"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:paddingStart="10dp"
        android:text="Farizgaskin" />

</LinearLayout>
```

Step 5: Creating Adapter class for RecyclerView

Now, we will create an Adapter.java class that will extend the RecyclerView.Adapter with ViewHolder. Go to the app > java > package > right-click and create a new java class and name it as Adapter. In Adapter class we will override the onCreateViewHolder() method which will inflate the list_item.xml layout and pass it to View Holder.

Then onBindViewHolder() method where we set data to the Views with the help of View Holder. Below is the code snippet for Adapter.java class.

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.ArrayList;

// Extends the Adapter class to RecyclerView.Adapter
// and implement the unimplemented methods
public class Adapter extends RecyclerView.Adapter<Adapter.ViewHolder> {
    ArrayList images, text;
    Context context;

    // Constructor for initialization
    public Adapter(Context context, ArrayList images, ArrayList text) {
        this.context = context;
        this.images = images;
        this.text = text;
    }

    @NonNull
    @Override
    public Adapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        // Inflating the Layout(Instantiates list_item.xml layout file
into View object)
        View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.list_item,
parent, false);

        // Passing view to ViewHolder
        Adapter.ViewHolder viewHolder = new Adapter.ViewHolder(view);
        return viewHolder;
    }

    // Binding data to the into specified position
    @Override
    public void onBindViewHolder(@NonNull Adapter.ViewHolder holder, int
position) {
        // TypeCast Object to int type
        int res = (int) images.get(position);
        holder.images.setImageResource(res);
        holder.text.setText((CharSequence) text.get(position));
    }
```

```
    @Override
    public int getItemCount() {
        // Returns number of items currently available in Adapter
        return text.size();
    }

    // Initializing the Views
    public class ViewHolder extends RecyclerView.ViewHolder {
        ImageView images;
        TextView text;

        public ViewHolder(View view) {
            super(view);
            images = (ImageView) view.findViewById(R.id.imageView);
            text = (TextView) view.findViewById(R.id.textView);
        }
    }
}
```

Step 6: Working with MainActivity.java file

In MainActivity.java class we create two ArrayList for storing images and text. These images are placed in the drawable folder(app > res > drawable). You can use any images in place of this. And then we get the reference of SwipeRefreshLayout and RecyclerView and set the LayoutManager and Adapter to show items in RecyclerView and implement onRefreshListener. Below is the code for the MainActivity.java file.

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.swiperefreshlayout.widget.SwipeRefreshLayout;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

public class MainActivity extends AppCompatActivity {
    SwipeRefreshLayout swipeRefreshLayout;
    RecyclerView recyclerView;

    // Using ArrayList to store images and text data
    ArrayList images = new ArrayList<>(Arrays.asList(R.drawable.facebook,
R.drawable.twitter,
                    R.drawable.instagram, R.drawable.linkedin,
R.drawable.youtube, R.drawable.whatsapp));
    ArrayList text = new ArrayList<>(Arrays.asList("Facebook", "Twitter",
"Instagram", "LinkedIn", "Youtube", "Whatsapp"));

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

        // Getting reference of swipeRefreshLayout and recyclerView
        swipeRefreshLayout = (SwipeRefreshLayout)
findViewById(R.id.swipeRefreshLayout);
```

```
        recyclerView = (RecyclerView) findViewById(R.id.recyclerView);

        // Setting the layout as Linear for vertical orientation to have
swipe behavior
        LinearLayoutManager linearLayoutManager = new
LinearLayoutManager(getApplicationContext());
        recyclerView.setLayoutManager(linearLayoutManager);

        // Sending reference and data to Adapter
        Adapter adapter = new Adapter(MainActivity.this, images, text);

        // Setting Adapter to RecyclerView
        recyclerView.setAdapter(adapter);

        // SetOnRefreshListener on SwipeRefreshLayout
        swipeRefreshLayout.setOnRefreshListener(new
SwipeRefreshLayout.OnRefreshListener() {
            @Override
            public void onRefresh() {
                swipeRefreshLayout.setRefreshing(false);
                RearrangeItems();
            }
        });
    }

    public void RearrangeItems() {
        // Shuffling the data of ArrayList using system time
        Collections.shuffle(images, new
Random(System.currentTimeMillis()));
        Collections.shuffle(text, new
Random(System.currentTimeMillis()));
        Adapter adapter = new Adapter(MainActivity.this, images, text);
        recyclerView.setAdapter(adapter);
    }
}
```

Run the app to see the results

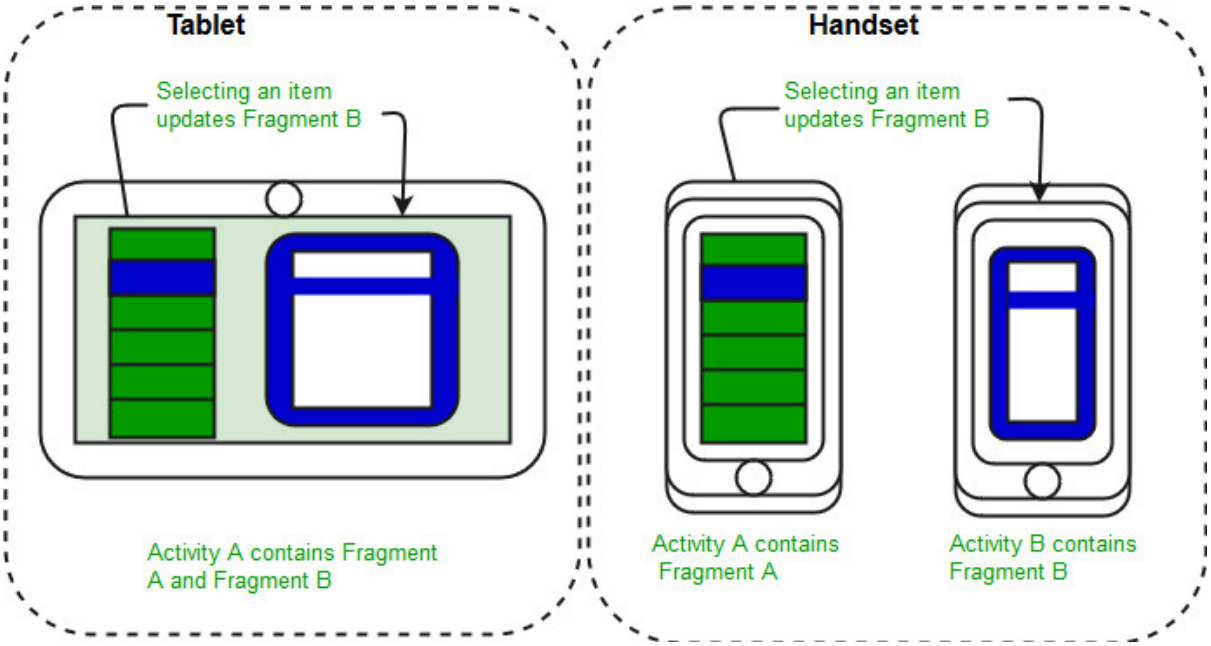
16 Fragments

16.1 Introduction to Fragments

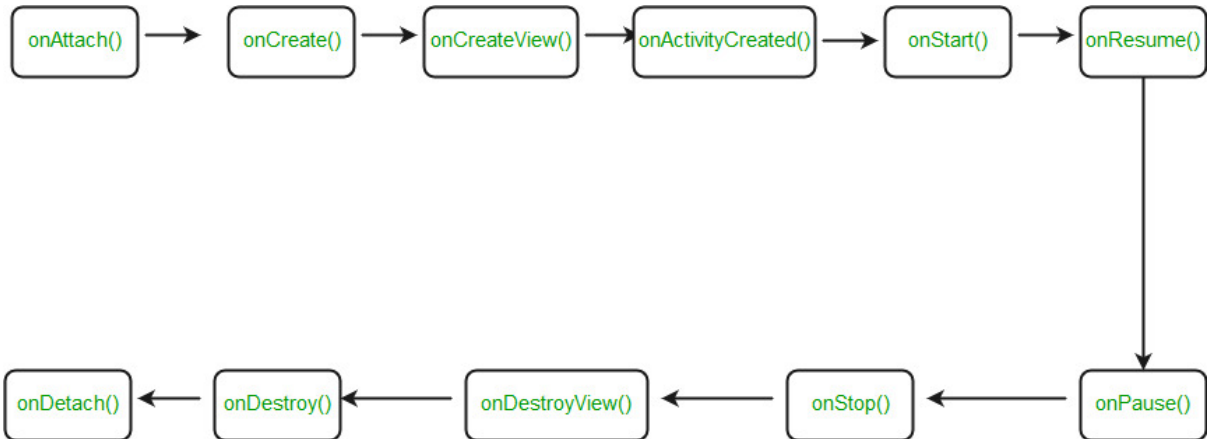
A Fragment is a piece of an activity which enable more modular activity design. A fragment encapsulates functionality so that it is easier to reuse within activities and layouts.

Android devices exists in a variety of screen sizes and densities. Fragments simplify the reuse of components in different layouts and their logic. You can build single-pane layouts for handsets (phones) and multi-pane layouts for tablets. You can also use fragments also to support different layout for landscape and portrait orientation on a smartphone.

The below image shows how two UI modules defined by fragments can be combined into one activity for a tablet design but separated for a handset design.



Android fragments have their own life cycle very similar to an android activity.



Fragment Lifecycle

- onAttach() : The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
- onCreate() : The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- onCreateView() : The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- onActivityCreated() : The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
- onStart() : The onStart() method is called once the fragment gets visible.
- onResume() : Fragment becomes active.
- onPause() : The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
- onStop() : Fragment going to be stopped by calling onStop()
- onDestroyView() : Fragment view will destroy after call this method
- onDestroy() :called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

Types of Fragments

- Single frame fragments : Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
- List fragments : fragments having special list view is called as list fragment
- Fragments transaction : Using with fragment transaction. we can move one fragment to another fragment.
- Handling the Fragment Lifecycle

A Fragment exist in three states :

- Resumed : The fragment is visible in the running activity.
- Paused : Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).
- Stopped : The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive (all state and member information is retained by the system). However, it is no longer visible to the user and will be killed if the activity is killed.

16.2 Swipe Navigation

When talking about Android Apps, the first thing that comes to mind is variety. There are so many varieties of Android apps providing the user with beautiful dynamic UI. One such feature is to navigate in our Android Apps using left and right swipes as opposed to clicking on buttons. Not only does it look more simple and elegant but also provides ease of access to the user.

There are many apps which use this swipe feature to swipe through different activities in the app. For example, the popular chatting app, Snapchat, uses it to swipe through lenses, chats and stories. Here, we will learn how to implement swipe views in our own Android app.

We can implement this by use of two features:

1. Fragments A fragment is just a part of an activity. We can have a fragment that takes up part of a screen or a whole screen. Or we can show multiple fragments at the same time to make up a whole screen. Within an activity, we can also swap out different fragments with each other.
2. ViewPager ViewPager is a class in Java which is used in conjunction with Fragments. It is mostly used for designing the UI of the app.

How does it work?

First we need to set an adapter on the ViewPager using the setAdapter() method. The adapter which we set is called FragmentPagerAdapter which is an abstract class in Java. Hence, we create our own SampleFragmentPagerAdapter which extends from FragmentPagerAdapter and displays our Fragments on the screen.

When we launch the app in our device, the ViewPager asks the SampleFragmentPagerAdapter how many pages are there to swipe through. The getCount() method of the adapter returns this answer to the ViewPager. Then the ViewPager asks for the Fragment which is at the 0th position and the adapter returns that particular fragment which is then displayed by ViewPager on our screen.

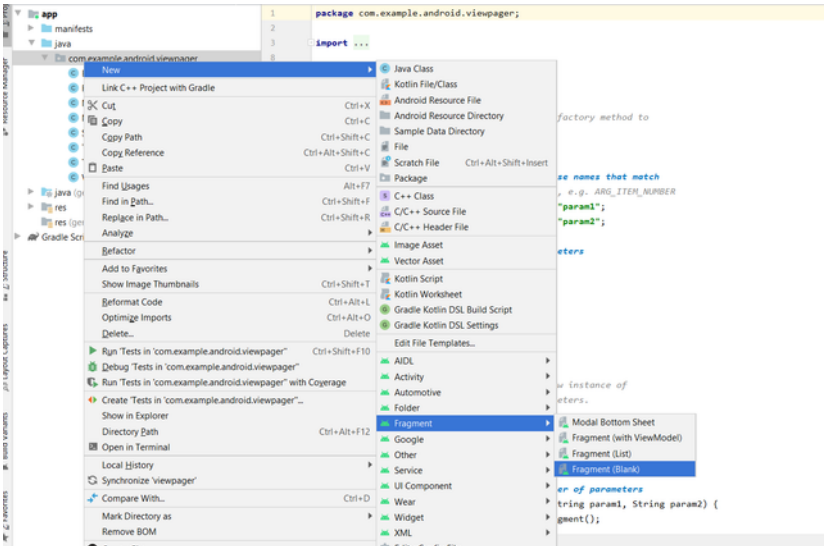
When we swipe left, ViewPager asks adapter for the Fragment at 1st position and similarly, it is displayed on the screen and it continues so on.

Step by step implementation:

We will be creating three fragments that is three screens that the user can swipe through. Then we will add these fragments to our FragmentPagerAdapter and finally set it on ViewPager.

Step1: Creating Fragments:

To create a fragment, click on app > Java > com.example.android(Right Click) > New > Fragment > Fragment(Blank)



We can create as many Fragments as we want but since we will be displaying only three screens, hence we will create three Fragments.

Now we will open our Fragment1.java and copy this code over there:

```
package com.example.android.gfg;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {

    @Override
    public View onCreateView(
        LayoutInflater inflater,
        ViewGroup container,
        Bundle savedInstanceState)
    {
        return inflater
            .inflate(
                R.layout.fragment_1,
                container, false);
    }
}
```

Explanation:

- We are importing v4 Fragment with BuildTools version “23.0.2“ . If we get an error in our imports, we must make sure that our BuildTools and SDK version corresponds to the libraries which we import.
- We name our Fragment as Fragment1.
- Fragment1 displays the layout fragment_1.xml which we will make now.

Step 2: Creating layouts for Fragments:

Each Fragment needs to display a layout. We can design this layout anyhow we want. Here is the code to how we would implement this layout as:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/gfg"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="With the idea of imparting programming knowledge,
Fariz Gaskin, an IT trainer started a dream, FGTech. Whether programming
excites you or you feel stifled, wondering how to prepare for interview
questions or how to ace data structures and algorithms, FG is a one-stop
solution."
        android:textSize="20sp"
        android:textColor="#81c784"/>
</LinearLayout>
```

Run the app to see the current progress

Similarly, we will create two more Fragments and a respective layout for each one of them.

Step 3: Creating our FragmentPageAdapter:

Now that we have all our three Fragments and three layouts are associated with each one of them, we will now build our FragmentPageAdapter and call it SimpleFragmentPageAdapter. This can be done by first creating a new Java Class and naming it as SimpleFragmentPageAdapter. Here's the code it will contain:

```
package com.example.android.gfg;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

public class SimpleFragmentPagerAdapter
    extends FragmentPagerAdapter {

    public SimpleFragmentPagerAdapter(
        FragmentManager fm)
    {
        super(fm);
    }
```

```
    @Override
    public Fragment getItem(int position)
    {
        if (position == 0) {
            return new Fragment1();
        }
        else if (position == 1) {
            return new Fragment2();
        }
        else {
            return new Fragmnet3();
        }
    }

    @Override
    public int getCount()
    {
        return 3;
    }
}
```

Explanation:

- The getItem() method returns the Fragment at specified position. So first will be Fragment1 and upon swiping left, we will get the other Fragments in the same order.
- The getCount() method returns the number of Fragments there are to be displayed, which, in this case is three.

Step 4: Creating MainActivity.Java and activity_main.xml :

Now that we have everything ready, the last step is just to make our MainActivity.Java and activity_main.xml files. They can vary largely depending on the app we are making but in this case, the files are pretty simple and looks like this:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<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"
    android:orientation="vertical"
    tools:context="com.example.android.viewpager.MainActivity">

    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

MainActivity.java

```
package com.example.android.gfg;

import android.os.Bundle;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;

public class MainActivity
    extends AppCompatActivity {

    @Override
    protected void onCreate(
        Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // Set the content of the activity
        // to use the activity_main.xml
        // layout file
        setContentView(R.layout.activity_main);

        // Find the view pager that will
        // allow the user to swipe
        // between fragments
        ViewPager viewPager
            = (ViewPager)findViewById(
                R.id.viewpager);

        // Create an adapter that
        // knows which fragment should
        // be shown on each page
        SimpleFragmentPagerAdapter
            adapter
            = new SimpleFragmentPagerAdapter(
                getSupportFragmentManager());

        // Set the adapter onto
        // the view pager
        viewPager.setAdapter(adapter);
    }
}
```

Run the app to see the results

17 Adapters

17.1 ArrayAdapter

The Adapter acts as a bridge between the UI Component and the Data Source. It converts data from the data sources into view items that can be displayed into the UI Component. Data Source can be Arrays, HashMap, Database, etc. and UI Components can be ListView, GridView, Spinner, etc. ArrayAdapter is the most commonly used adapter in android.

When you have a list of single type items which are stored in an array you can use ArrayAdapter. Likewise, if you have a list of phone numbers, names, or cities. ArrayAdapter has a layout with a single TextView. If you want to have a more complex layout instead of ArrayAdapter use CustomArrayAdapter. The basic syntax for ArrayAdapter is given as:

Example

In this example, the list of courses is displayed using a simple array adapter. Note that we are going to implement this project using the Java language.

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language.

Step 2: Working with the activity_main.xml file

Go to the layout folder and in activity_main.xml file change the ConstraintLayout to RelativeLayout and insert a ListView with id simpleListView. Below is the code for the activity_main.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"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Step 3: Create a new layout file

Go to app > res > layout > right-click > New > Layout Resource File and create a new layout file and name this file as item_view.xml and make the root element as a LinearLayout. This will contain a TextView that is used to display the array objects as output.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/itemTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

</LinearLayout>
```

Step 4: Working with the MainActivity.java file

Now go to the java folder and in MainActivity.java and provide the implementation to the ArrayAdapter. Below is the code for the MainActivity.java file.

```
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    ListView simpleListView;
    // array objects
    String courseList[] = {"C-Programming", "Data Structure", "Database",
"Python", "Java", "Operating System", "Compiler Design", "Android
Development"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        simpleListView = (ListView) findViewById(R.id.simpleListView);
        ArrayAdapter<String> arrayAdapter = new
ArrayAdapter<String>(this,
                R.layout.item_view, R.id.itemTextView, courseList);
        simpleListView.setAdapter(arrayAdapter);
    }
}
```

Run the app to see the results

18 Spinner

18.1 Spinner in Kotlin

Android Spinner is a view similar to dropdown list which is used to select one option from the list of options. It provides an easy way to select one item from the list of items and it shows a dropdown list of all values when we click on it.

Default value of the android spinner will be currently selected value and by using Adapter we can easily bind the items to spinner object. Generally, we populate our Spinner control with list of items by using an ArrayAdapter in our Kotlin file.

First we create a new project by following the below steps:

1. Click on File, then New => New Project.
2. After that include the Kotlin support and click on next.
3. Select the minimum SDK as per convenience and click next button.
4. Then select the Empty activity => next => finish.

Modify activity_main.xml file

In this file, we use the TextView and Spinner widgets and also set their attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/linear_layout"
    android:gravity = "center">

    <TextView
        android:id="@+id/txtView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select language:"
        android:textSize = "20dp" />

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@id/txtView"/>

</LinearLayout>
```

Update strings.xml file

Here, we update the name of the application using the string tag. We also create the list of the items which will be used in the dropdown menu.

```
<resources>
    <string name="app_name">SpinnerInKotlin</string>
    <string name="selected_item">Selected item:</string>

    <string-array name="Languages">
        <item>Java</item>
        <item>Kotlin</item>
        <item>Swift</item>
        <item>Python</item>
        <item>Scala</item>
        <item>Perl</item>
    </string-array>
</resources>
```

Access Spinner in MainActivity.kt file

First, we declare variable languages to access the strings items from the strings.xmnl file.

```
val languages = resources.getStringArray(R.array.Languages)
```

then, we access the spinner and set ArrayAdaptor to control the list of items.

```
val spinner = findViewById(R.id.spinner)
    if (spinner != null) {
        val adapter = ArrayAdapter(this,
            android.R.layout.simple_spinner_item, languages)
        spinner.adapter = adapter
```

Complete code for MainActivity.kt

```
package com.farizgaskin.myfirstkotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // access the items of the list
        val languages = resources.getStringArray(R.array.Languages)

        // access the spinner
```

```
        val spinner = findViewById<Spinner>(R.id.spinner)
        if (spinner != null) {
            val adapter = ArrayAdapter(this,
                android.R.layout.simple_spinner_item, languages)
            spinner.adapter = adapter

         spinner.onItemSelectedListener = object :
           AdapterView.OnItemSelectedListener {
             override fun onItemSelected(parent: AdapterView<*>,
                                         view: View, position: Int, id:
Long) {
                 Toast.makeText(this@MainActivity,
                    getString(R.string.selected_item) + " " +
                         "" + languages[position],
Toast.LENGTH_SHORT).show()
                }

                override fun onNothingSelected(parent: AdapterView<*>) {
                    // write code to perform some action
                }
            }
        }
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.farizgaskin.myfirstkotlinapp">

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Run the app to see the results

Android Development FUNdamentals

19 AlertDialog

19.1 Creating an AlertDialog Box

Alert Dialog shows the Alert message and gives the answer in the form of yes or no. Alert Dialog displays the message to warn you and then according to your response the next step is processed.

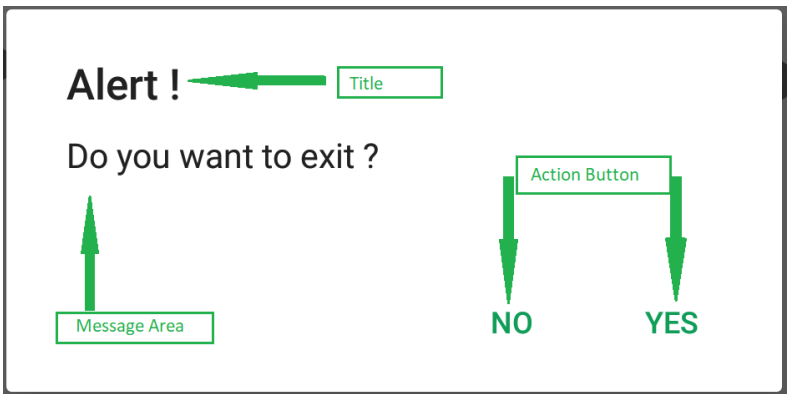
Android Alert Dialog is built with the use of three fields: Title, Message area, Action Button.

Alert Dialog code has three methods:

- setTitle() method for displaying the Alert Dialog box Title
- setMessage() method for displaying the message
- setIcon() method is use to set the icon on Alert dialog box.

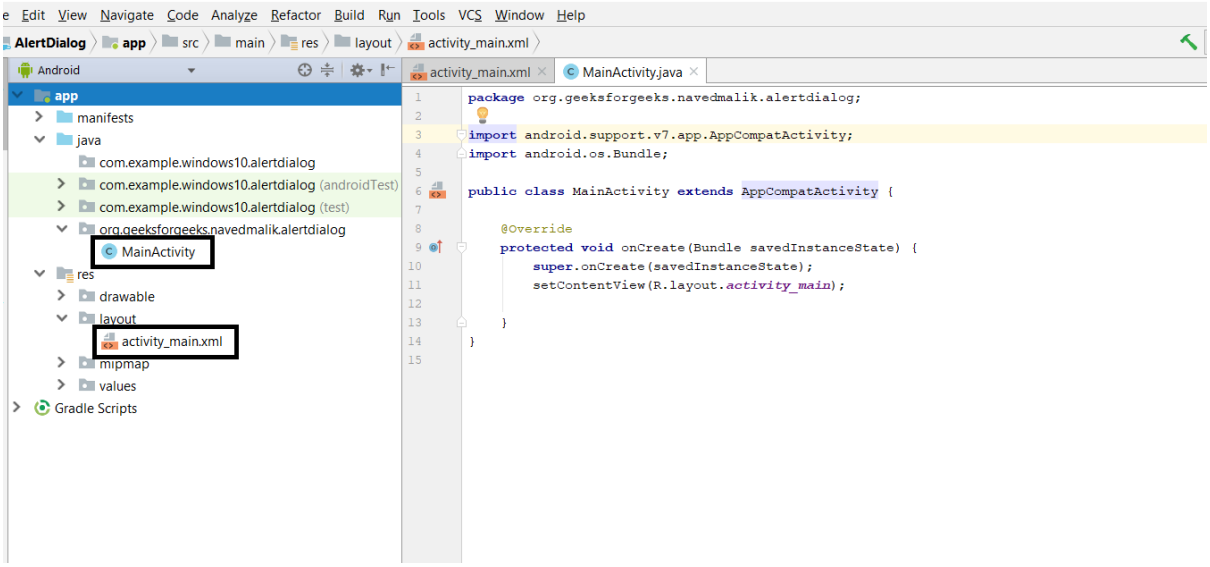
Then we add the two Button, setPositiveButton and setNegativeButton to our Alert Dialog Box as shown below.

Example:

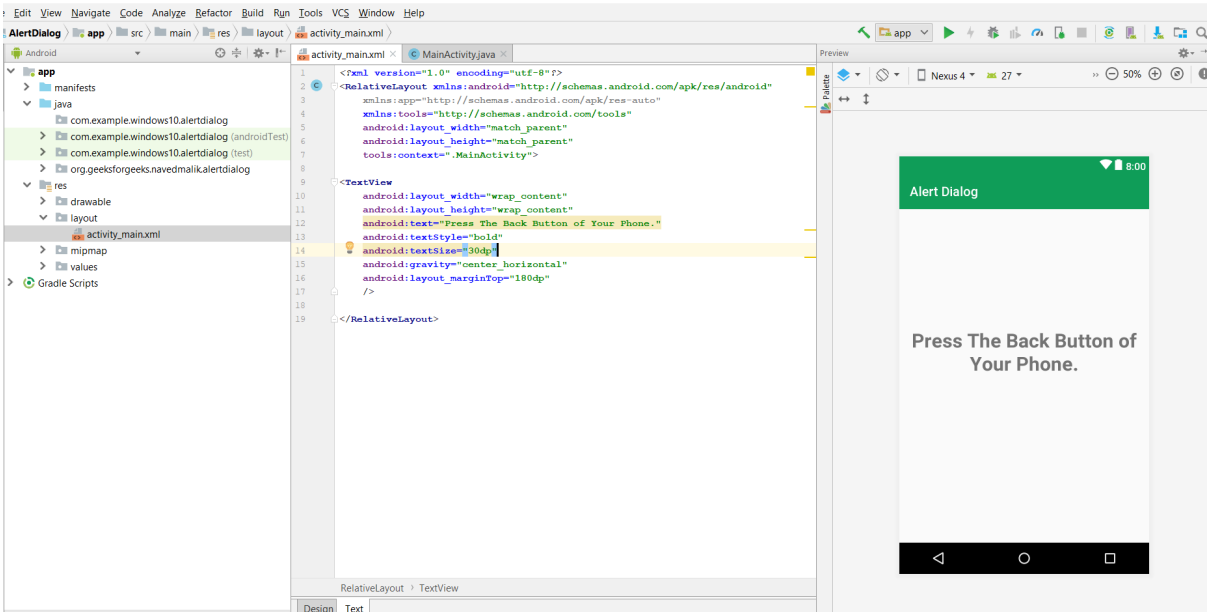


Below are the steps for Creating the Alert Dialog Android Application:

Step 1: Create a new project. After that, you have java and XML file.



Step 2: Open your XML file and then add TextView for message as shown below (you can change it accordingly).



Step 3: Now, open up the activity java file. After, on create method declaration, the onbackpressed method is called when you click the back button of your device.

Step 4: Create the object of Builder class Using AlertDialog.Builder. Now, set the Title, message.

Step 5: In a builder object set the positive Button now gives the button name and add the OnClickListener of DialogInterface. Same as with the negative Button, at last, create the Alert dialog Box with builder object and then show the Alert Dialog.

Step 6: Now if positive button press finish the app goto outside from the app if negative then finish the dialog box

Step 7: Now run it and then press the back button. After that click Yes or No Button.

The complete code of MainActivity.java or activity_main.xml of Alert Dialog is given below:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Press The Back Button of Your Phone."
    android:textStyle="bold"
```

```
    android:textSize="30dp"
    android:gravity="center_horizontal"
    android:layout_marginTop="180dp"
    />

</RelativeLayout>
```

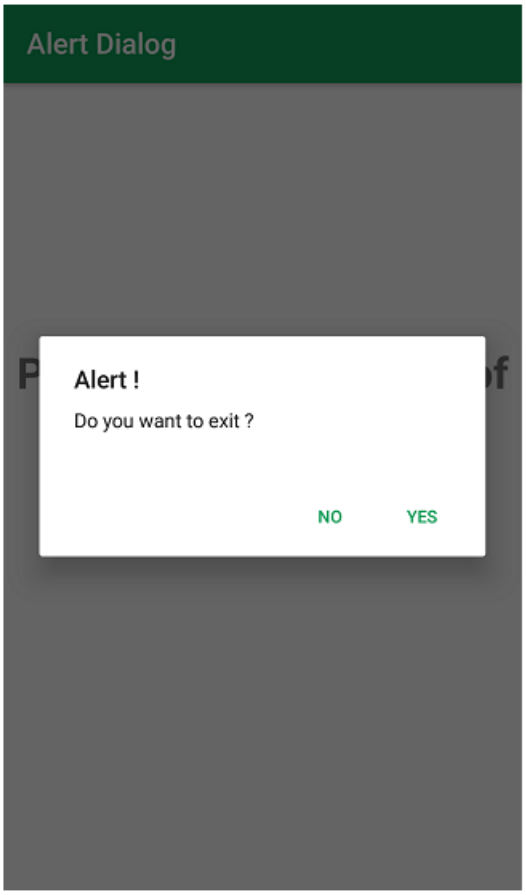
MainActivity.java

```
package org.farizgaskin.alertdialog;

import android.content.DialogInterface;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

    // Declare the onBackPressed method
    // when the back button is pressed
    // this method will call
    @Override
    public void onBackPressed()
    {

        // Create the object of
        // AlertDialog Builder class
        AlertDialog.Builder builder
            = new AlertDialog
                  .Builder(MainActivity.this);

        // Set the message show for the Alert time
        builder.setMessage("Do you want to exit ?");

        // Set Alert Title
        builder.setTitle("Alert !");

        // Set Cancelable false
        // for when the user clicks on the outside
        // the Dialog Box then it will remain show
        builder.setCancelable(false);

        // Set the positive button with yes name
        // OnClickListener method is use of
        // DialogInterface interface.

        builder
            .setPositiveButton(
                "Yes",
                new DialogInterface
                    .OnClickListener() {
```

```
                        @Override
                        public void onClick(DialogInterface dialog,
                                            int which)
                        {

                            // When the user click yes button
                            // then app will close
                            finish();
                        }
                    });

        // Set the Negative button with No name
        // OnClickListener method is use
        // of DialogInterface interface.
        builder
            .setNegativeButton(
                "No",
                new DialogInterface
                    .OnClickListener() {

                        @Override
                        public void onClick(DialogInterface dialog,
                                            int which)
                        {

                            // If user click no
                            // then dialog box is canceled.
                            dialog.cancel();
                        }
                    });

        // Create the Alert dialog
        AlertDialog alertDialog = builder.create();

        // Show the Alert Dialog box
        alertDialog.show();
    }
}
```

Run the app to see the results


Alert Dialog
Press The Back Button of Your Phone.


Alert Dialog
Alert !
Do you want to exit ?
NO
YES

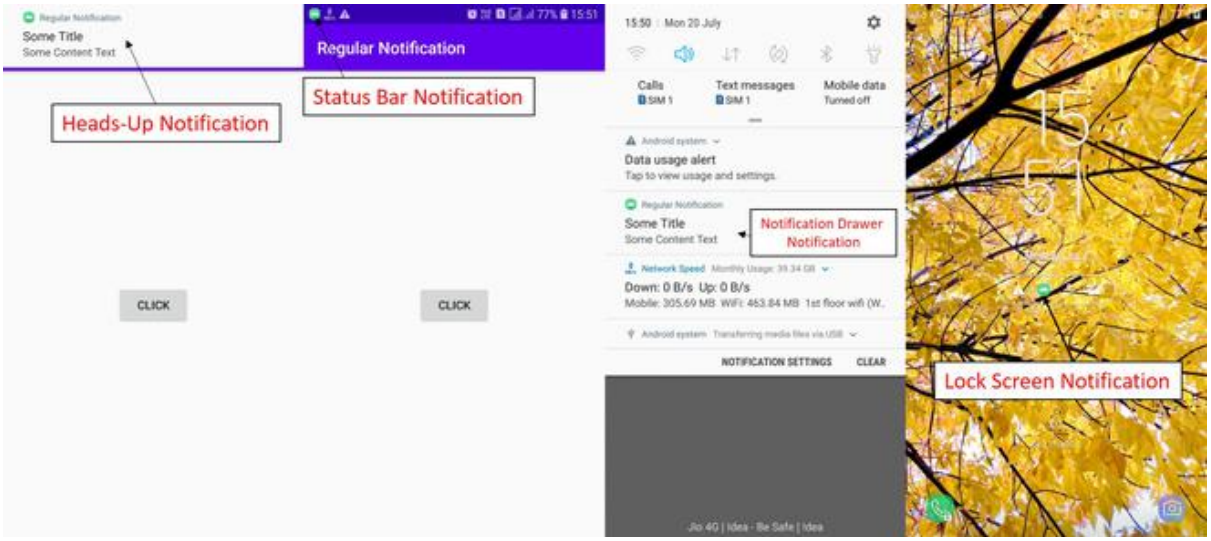
20 Notifications

20.1 Notifications in Android

Notification is a kind of message, alert, or status of an application (probably running in the background) that is visible or available in the Android's UI elements. This application could be running in the background but not in use by the user. The purpose of a notification is to notify the user about a process that was initiated in the application either by the user or the system. This article could help someone who's trying hard to create a notification for developmental purposes.

Notifications could be of various formats and designs depending upon the developer. In General, one must have witnessed these four types of notifications:

1. Status Bar Notification (appears in the same layout as the current time, battery percentage)
2. Notification drawer Notification (appears in the drop-down menu)
3. Heads-Up Notification (appears on the overlay screen, ex: Whatsapp notification, OTP messages)
4. Lock-Screen Notification (I guess you know it)



In this lesson, we will be discussing how to produce notifications in Kotlin.

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Kotlin as the programming language.

Step 2: Working with the activity_main.xml file

Go to the activity_main.xml file and refer to the following code. In this step, we are going to design our layout page. Here, we will use the RelativeLayout to get the Scroll View from the Kotlin file. Below is the code for the activity_main.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"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Send Notification" />

</RelativeLayout>
```

Step 3: Create a new empty activity

Name the activity as afterNotification. When someone clicks on the notification, this activity will open up in our app that is the user will be redirected to this page. Below is the code for the activity_after_notification.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"
    tools:context=".afterNotification">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Welcome To Farizgaskin"
        android:textSize="15sp"
        android:textStyle="bold" />

</RelativeLayout>
```

Step 5: Working with the MainActivity.kt file

Go to the MainActivity.kt file and refer to the following code. Below is the code for the MainActivity.kt file. Comments are added inside the code to understand the code in more detail.

```
import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import android.graphics.BitmapFactory
import android.graphics.Color
import android.os.Build
import android.os.Bundle
import android.widget.Button
import android.widget.RemoteViews
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    // declaring variables
    lateinit var notificationManager: NotificationManager
    lateinit var notificationChannel: NotificationChannel
    lateinit var builder: Notification.Builder
    private val channelId = "i.apps.notifications"
    private val description = "Test notification"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // accessing button
        val btn = findViewById<Button>(R.id.btn)

        // it is a class to notify the user of events that happen.
        // This is how you tell the user that something has happened in
the
        // background.
        notificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        // onClick listener for the button
        btn.setOnClickListener {

            // pendingIntent is an intent for future use i.e after
            // the notification is clicked, this intent will come into
action
            val intent = Intent(this, afterNotification::class.java)

            // FLAG_UPDATE_CURRENT specifies that if a previous
            // PendingIntent already exists, then the current one
            // will update it with the latest intent
            // 0 is the request code, using it later with the
            // same method again will get back the same pending
            // intent for future reference
            // intent passed here is to our afterNotification class
            val pendingIntent = PendingIntent.getActivity(this, 0,
intent, PendingIntent.FLAG_UPDATE_CURRENT)
```

```
            // RemoteViews are used to use the content of
            // some different layout apart from the current activity
layout
            val contentView = RemoteViews(packageName,
R.layout.activity_after_notification)

            // checking if android version is greater than oreo(API 26)
or not
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                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.drawable.ic_launcher_background)

.setLargeIcon(BitmapFactory.decodeResource(this.resources,
R.drawable.ic_launcher_background))
                        .setContentIntent(pendingIntent)
            } else {

                builder = Notification.Builder(this)
                        .setContent(contentView)
                        .setSmallIcon(R.drawable.ic_launcher_background)

.setLargeIcon(BitmapFactory.decodeResource(this.resources,
R.drawable.ic_launcher_background))
                        .setContentIntent(pendingIntent)
            }
            notificationManager.notify(1234, builder.build())
        }
    }
}
```

With this, we have now successfully created a “Notification” for our application. Note that the parameters listed in the above code are required and the absence of any single parameter could result in crashing or not starting the application. The content title, content text, small icon are customizable parameters but are mandatory also. One can change their values according to the requirement.

Run the app to see the results

21 Menu

21.1 Implementing OptionsMenu

In android, there are three types of Menus available to define a set of options and actions in our android applications.

The Menus in android applications are the following:

- Android Options Menu
- Android Context Menu
- Android Popup Menu

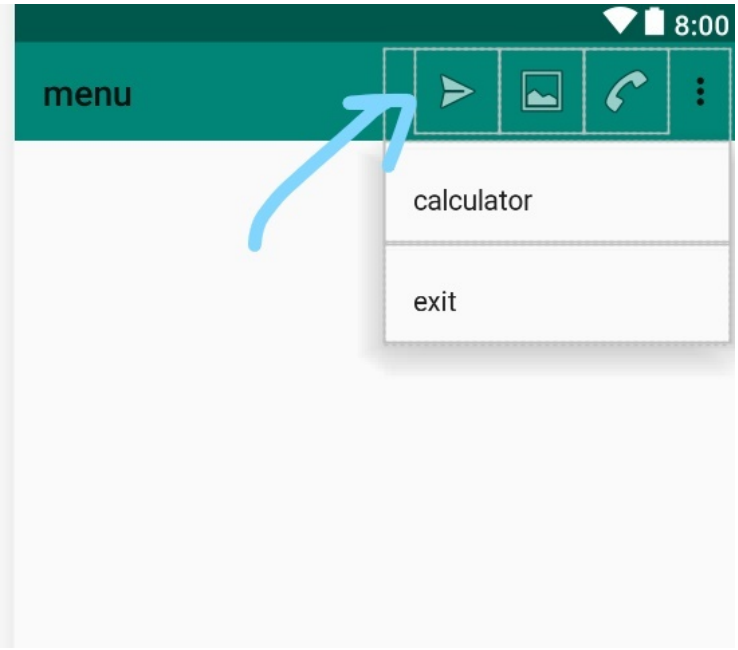
Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc. When and how this item should appear as an action item in the app bar is decided by the Show Action attribute. The values that can be given for the showAsAction attribute:

always: This ensures that the menu will always show in the action bar.

```
app:showAsAction="always"
```

Example

```
<item
    android:id="@+id/message"
    android:icon="@android:drawable/ic_menu_send"
    app:showAsAction="always"
    android:title="message"
/>
```



never: This means that the menu will never show, and therefore will be available through the overflow menu

```
<item
    android:id="@+id/exit"
    app:showAsAction="never"
    android:title="exit"/>
```

Below is the complete code for implementing the Options Menu in Android is given below:

menu_main.xml

```
<menu
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">

    <item
        android:id="@+id/message"
        android:icon="@android:drawable/ic_menu_send"
        app:showAsAction="always"
        android:title="message"/>

    <item
        android:id="@+id/picture"
        android:icon="@android:drawable/ic_menu_gallery"
        app:showAsAction="always|withText"
        android:title="picture"/>

    <item
        android:id="@+id/mode"
        android:icon="@android:drawable/ic_menu_call"
        app:showAsAction="always"
        android:title="mode"/>

    <item
        android:id="@+id/about"
        android:icon="@android:drawable/ic_dialog_info"
        app:showAsAction="never|withText"
        android:title="calculator"/>

    <item
        android:id="@+id/exit"
        app:showAsAction="never"
        android:title="exit"/>
</menu>
```

MainActivity.java

```
package com.example.menu;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

import static android.widget.Toast.LENGTH_LONG;

public class MainActivity extends AppCompatActivity {

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

    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.menu, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId()) {
        case R.id.message:
            Toast
                .makeText(
                    getApplicationContext(),
                    "Shows share icon",
                    Toast.LENGTH_SHORT)
                .show();
            return true;

        case R.id.picture:
            Toast
                .makeText(
                    getApplicationContext(),
                    "Shows image icon",
                    Toast.LENGTH_SHORT)
                .show();
            startActivity(i2);
            return (true);

        case R.id.mode:
            Toast
                .makeText(
                    getApplicationContext(),
                    "Shows call icon",
                    Toast.LENGTH_SHORT)
                .show();
            return (true);

        case R.id.about:
            Toast
```

```
                .makeText(
                    getApplicationContext(),
                    "calculator menu",
                    Toast.LENGTH_SHORT)
                .show();
            return (true);

        case R.id.exit:
            finish();
            return (true);
        }
        return (super.onOptionsItemSelected(item));
    }
}
```

Run the app to see the results

22 Date and Time

22.1 DatePicker in Kotlin

Android DatePicker is a user interface control which is used to select the date by day, month and year in our android application. DatePicker is used to ensure that the users will select a valid date. In android DatePicker having two modes, first one to show the complete calendar and second one shows the dates in spinner view.

We can create a DatePicker control in two ways either manually in XML file or create it in Activity file programmatically.

First we create a new project by following the below steps:

1. Click on File, then New => New Project.
2. After that include the Kotlin support and click on next.
3. Select the minimum SDK as per convenience and click next button.
4. Then select the Empty activity => next => finish.

Android DatePicker with Calendar mode

We can use android:datePickerMode to show only calendar view. In the below example, we are using the DatePicker in Calendar mode.

```
<DatePicker
    android:id="@+id/datePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:datePickerMode="calendar"/>
```

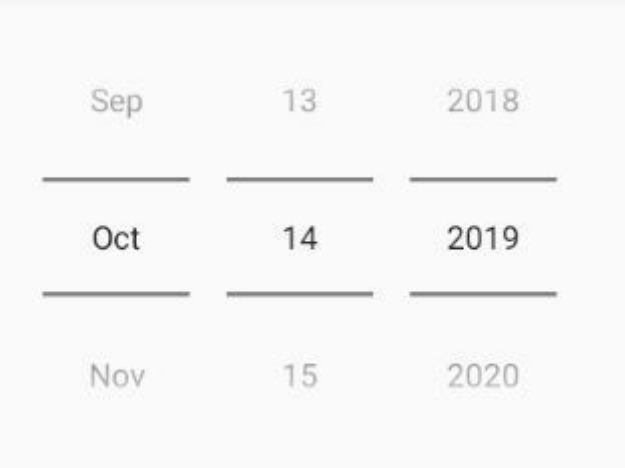
The above code of DatePicker can be seen in android application like this

Android DatePicker with Spinner mode

We can also show the DatePicker in spinner format like selecting the day, month and year separately, by using android:datePickerMode attribute and set android:calendarViewShown=”false”, otherwise both spinner and calendar can be seen simultaneously.

```
<DatePicker
    android:id="@+id/datePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:datePickerMode="spinner"
    android:calendarViewShown="false"/>
```

The above code of DatePicker can be seen in android application like this



To use Calendar DatePicker in activity_main.xml

In this file, we will add the DatePicker and Button widget and set their attributes so that it can be accessed in the kotlin file.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/linear_layout"
    android:gravity = "center">

    <DatePicker
        android:id="@+id/date_Picker"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

To use Spinner DatePicker in activity_main.xml

In this file, we will add the DatePicker and Button widget and set their attributes so that it can be accessed in the kotlin file.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/linear_layout"
    android:gravity = "center">

    <DatePicker
        android:id="@+id/date_Picker"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:datePickerMode = "spinner"
        android:calendarViewShown="false"/>
</LinearLayout>
```

Modify the strings.xml file to add the string-array

Here, we will specify the name of the activity.

```
<resources>
    <string name="app_name">DatePickerInKotlin</string>
</resources>
```

Access the DatePicker in MainActivity.kt file

First of all, we declare a variable datePicker to access the DatePicker widget from the XML layout.

```
val datePicker = findViewById<DatePicker>(R.id.date_Picker)
```

then, we declare another variable today to get the current get like this.

```
val today = Calendar.getInstance()
    datePicker.init(today.get(Calendar.YEAR), today.get(Calendar.MONTH),
    today.get(Calendar.DAY_OF_MONTH)
```

To display the selected date from the calendar we will use

```
{ view, year, month, day ->
    val month = month + 1
    val msg = "You Selected: $day/$month/$year"
    Toast.makeText(this@MainActivity, msg, Toast.LENGTH_SHORT).show()
 }
```

We are familiar with further activities in previous articles like accessing button and set OnClickListener etc.

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.*
import java.util.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val datePicker = findViewById<DatePicker>(R.id.date_Picker)
        val today = Calendar.getInstance()
        datePicker.init(today.get(Calendar.YEAR),
today.get(Calendar.MONTH),
            today.get(Calendar.DAY_OF_MONTH)

        ) { view, year, month, day ->
            val month = month + 1
            val msg = "You Selected: $day/$month/$year"
            Toast.makeText(this@MainActivity, msg,
Toast.LENGTH_SHORT).show()
        }
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.farizgaskin.myfirstkotlinapp">

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

Run the app to see the results

22.2 TimePicker in Kotlin

Android TimePicker is a user interface control for selecting the time in either 24-hour format or AM/PM mode. It is used to ensure that users pick the valid time for the day in our application.

In android, TimePicker is available in two modes first one is clock mode and another one is spinner mode.

We can use TimePicker manually in XML layout or we can create it programmatically in Kotlin file. In this article, we should use TimePicker widget in XML Layout.

First we create a new project by following the below steps:

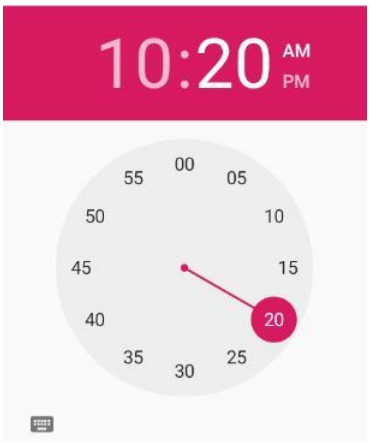
1. Click on File, then New => New Project.
2. After that include the Kotlin support and click on next.
3. Select the minimum SDK as per convenience and click next button.
4. Then select the Empty activity => next => finish.

Android TimePicker with Clock mode

We can use android:timePickerMode to show only clock view. In the below example, we are using the TimePicker in clock mode.

```
<TimePicker
    android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:timePickerMode="clock"/>
```

The above code of TimePicker can be seen in android application like this

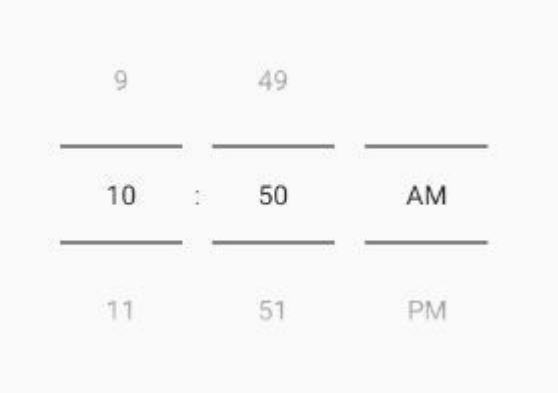


Android TimePicker with Spinner mode

We can also use the TimePicker in spinner format by using android:timePickerMode attribute.

```
<TimePicker
    android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:timePickerMode="spinner"/>
```

The above code of TimePicker can be seen in android application like this



To use Clock TimePicker in activity_main.xml

In this file, we will add the TimePicker and TextView widget and set their attributes so that it can be accessed in the kotlin file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_margin="@dimen/padding"
        android:timePickerMode="clock"/>
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/timePicker"
        android:textSize="18dp"
        android:paddingLeft="80dp" />
</RelativeLayout>
```

To use Spinner TimePicker in activity_main.xml

In this file, we will add the TimePicker and TextView widget and set their attributes so that it can be accessed in the kotlin file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_margin="@dimen/padding"
        android:timePickerMode="spinner"/>

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/timePicker"
        android:textSize="18dp"
        android:paddingLeft="80dp" />

</RelativeLayout>
```

Modify the strings.xml file to add the string-array

Here, we will specify the name of the activity

```
<resources>
    <string name="app_name">TimePickerInKotlin</string>
</resources>
```

Access the TimePicker in MainActivity.kt file

First of all, we define a function OnClickTime() and called from MainActivity.

```
private fun OnClickTime()
```

then, we declare two variables textView and timePicker to access the widgets from the XML layout using their id's.

```
val textView = findViewById(R.id.textView)
val timePicker = findViewById(R.id.timePicker)
```

```
package com.farizgaskin.myfirstkotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.ViewGroup
import android.widget.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        OnClickTime()
    }

    private fun OnClickTime() {
        val textView = findViewById<TextView>(R.id.textView)
        val timePicker = findViewById<TimePicker>(R.id.timePicker)
        timePicker.setOnTimeChangedListener { _, hour, minute -> var hour
= hour
            var am_pm = ""
            // AM_PM decider logic
            when {hour == 0 -> { hour += 12
                am_pm = "AM"
            }
                hour == 12 -> am_pm = "PM"
                hour > 12 -> { hour -= 12
                    am_pm = "PM"
                }
                else -> am_pm = "AM"
            }
            if (textView != null) {
                val hour = if (hour < 10) "0" + hour else hour
                val min = if (minute < 10) "0" + minute else minute
                // display format of time
                val msg = "Time is: $hour : $min $am_pm"
                textView.text = msg
                textView.visibility = ViewGroup.VISIBLE
            }
        }
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.farizgaskin.myfirstkotlinapp">

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
```

```
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Run the app to see the results

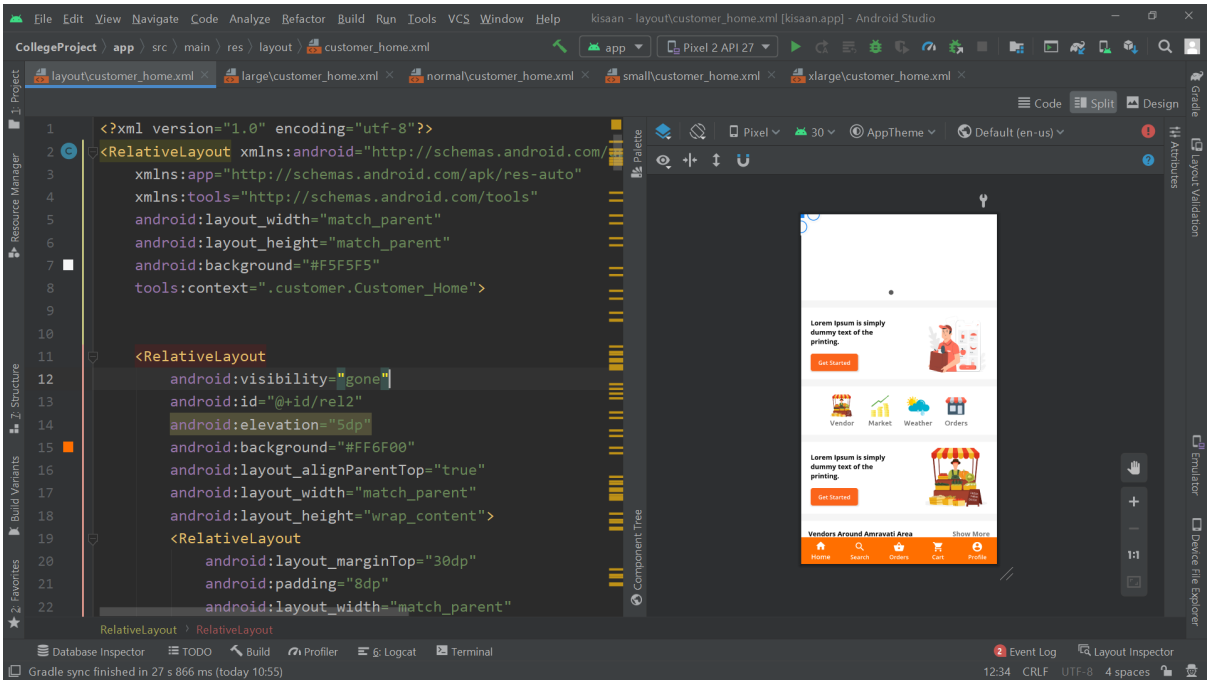
23 Material Design

23.1 Responsive UI Design

Nowadays every Android UI designer is very serious about his UI design. The path of every project goes through UI design. UI design is the face of every project and every website, app & desktop application. UI is the wall between the User & Back-end where users can interact with the server through the UI design.

There is a various aspect where UI designer need to consider while designing any software product, Such as What is the theme of the project? Which is the target audience? either it's 18+ or Old age audience, Does this UI support all types of screens or not?, and many more question arises in UI designer's mind. In this article, we will talk about the future of UI in Android.

In Android, there have various platforms where developers developed their Android application, such as Android Studio, Eclipse, React Native, etc. The Majority of the developer preferred. To make the android app responsive we need to make every individual screen for every type of screen size, Such as Large, X-large, Normal.



To overcome the of issue of redesigning for different screen sizes, Google introduced the Material Component for Android design, to overcome the responsiveness as much as possible. Material Components supports all screen sizes.

Using Material Component in Android Apps

Adding a material component you need to simply add this dependency in your Gradle file

```
implementation 'com.google.android.material:material:1.2.1'
```

After adding this dependency you will able to use the material component in your android project.

```
<com.google.android.material.button.MaterialButton
    android:layout_width="150dp"
    android:layout_height="60dp"
    android:backgroundTint="#5874FF"
    android:layout_below="@id/desc"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    app:cornerRadius="0dp"
    android:textSize="16sp"
    android:padding="10dp"
    android:id="@+id/btn1"
    android:fontFamily="@font/ubuntu_m"
    android:text="Learn More"
    android:textAllCaps="false"/>
```

To make our design more stylish and attractive as well as responsive you need to use Material Card View in your layout.

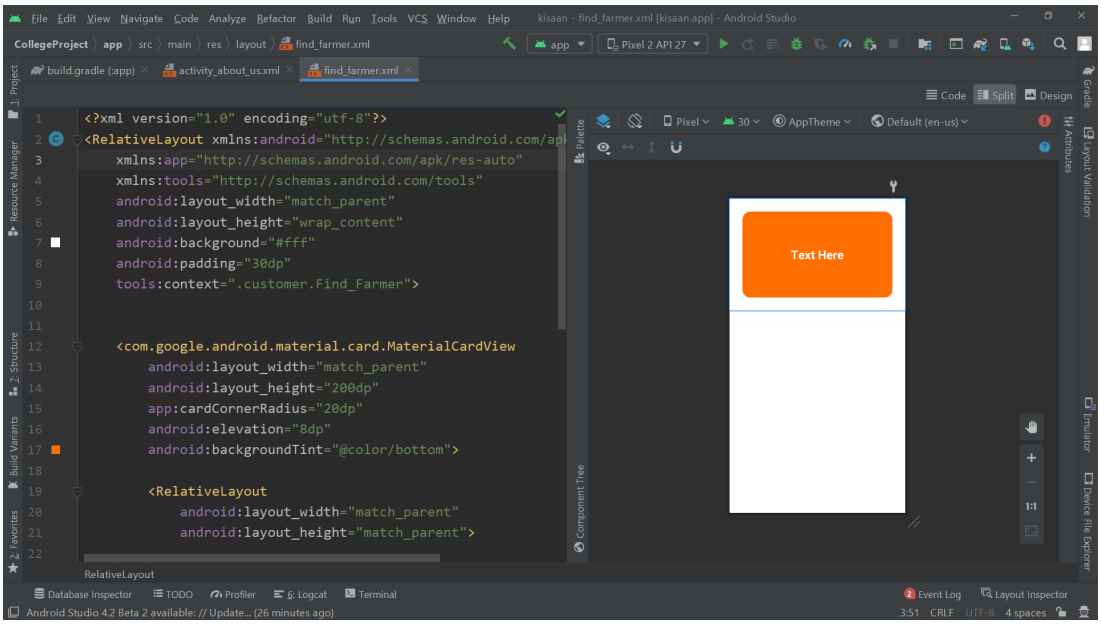
```
<com.google.android.material.card.MaterialCardView
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:id="@+id/card1"
        android:backgroundTint="@color/bottom"/>

    <com.google.android.material.card.MaterialCardView
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:id="@+id/card2"
        android:layout_below="@id/card1"
        android:layout_marginTop="20dp"
        android:backgroundTint="@color/bottom">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textSize="40sp"
                android:layout_centerInParent="true"
                android:textAlignment="center"
                android:textColor="#fff"
                android:textStyle="bold"
                android:text="Text"/>

        </RelativeLayout>

</com.google.android.material.card.MaterialCardView>
```

Material Card View automatically manages the padding and margin as well as sizes sometimes. You can add gradient color inside the card by using a third-party library its looks so awesome. You can add a menu inside the card block, you can add a slider in the card.

Example

Let's Take One Example with android studio. Now we can design an android layout using XML. We can design a material card view component first. Material Card View with parent padding.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#fff"
    android:padding="30dp"
    tools:context=".customer.Find_Farmer">

    <com.google.android.material.card.MaterialCardView
        android:layout_width="match_parent"
        android:layout_height="200dp"
        app:cardCornerRadius="20dp"
        android:elevation="8dp"
        android:backgroundTint="@color/bottom">
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Text Here"
                android:textStyle="bold"
                android:layout_centerInParent="true"
                android:textColor="#fff"
                android:textSize="28sp"/>
        </RelativeLayout>
    </com.google.android.material.card.MaterialCardView>
</RelativeLayout>
```



Material Card View without padding:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#fff"
    tools:context=".customer.Find_Farmer">

    <com.google.android.material.card.MaterialCardView
        android:layout_width="match_parent"
        android:layout_height="200dp"
        app:cardCornerRadius="20dp"
        android:elevation="8dp"
        android:backgroundTint="@color/bottom">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Text Here"
                android:textStyle="bold"
                android:layout_centerInParent="true"
                android:textColor="#fff"
                android:textSize="28sp"/>

        </RelativeLayout>

    </com.google.android.material.card.MaterialCardView>

</RelativeLayout>
```

If you use the material card view without padding then the material component can automatically add default padding and color. This can also be helpful for responsive UI design.

There are many aspects that were material component is easier & flexible than normal components such as shadow, border, etc.

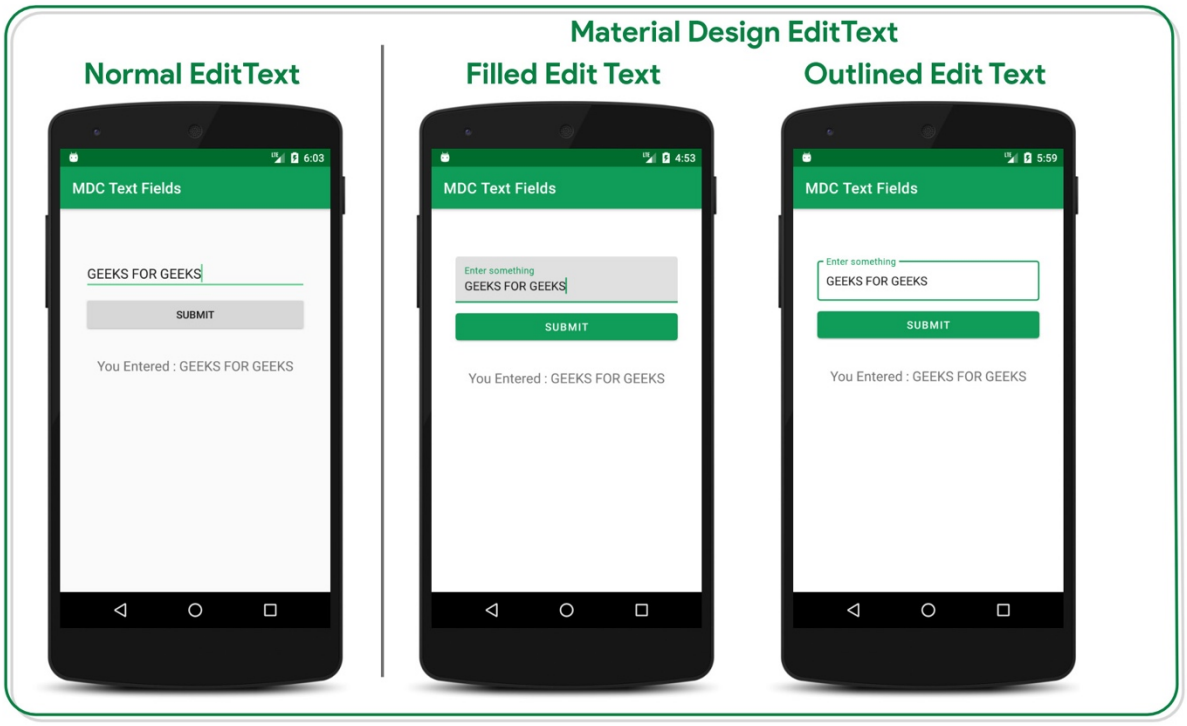
Advantages of Material Component

- Material components are lightweight.
- The material component looks great as compared to others.
- Material component very much as responsive.
- Disadvantages of material component

Sometimes component does not work properly in a real device.

23.2 Material Design EditText

EditText is one of the important UI elements. Edittext refers to the widget that displays an empty text field in which a user can enter the required text and this text is further used inside the application. In this article it's been discussed to implement the special type of text fields, those are called Material Design EditText. Have a look at the normal edit text in android and Material design Text fields in android. The design and the easy to use implementation makes them different from normal EditText fields.



Step by Step Implementation
In this example, we are going to demonstrate two important types of Material Design EditText:

1. Filled EditText
2. Outlined EditText

Step 1: Create a New Project

Select either Java or Kotlin as the programming language.

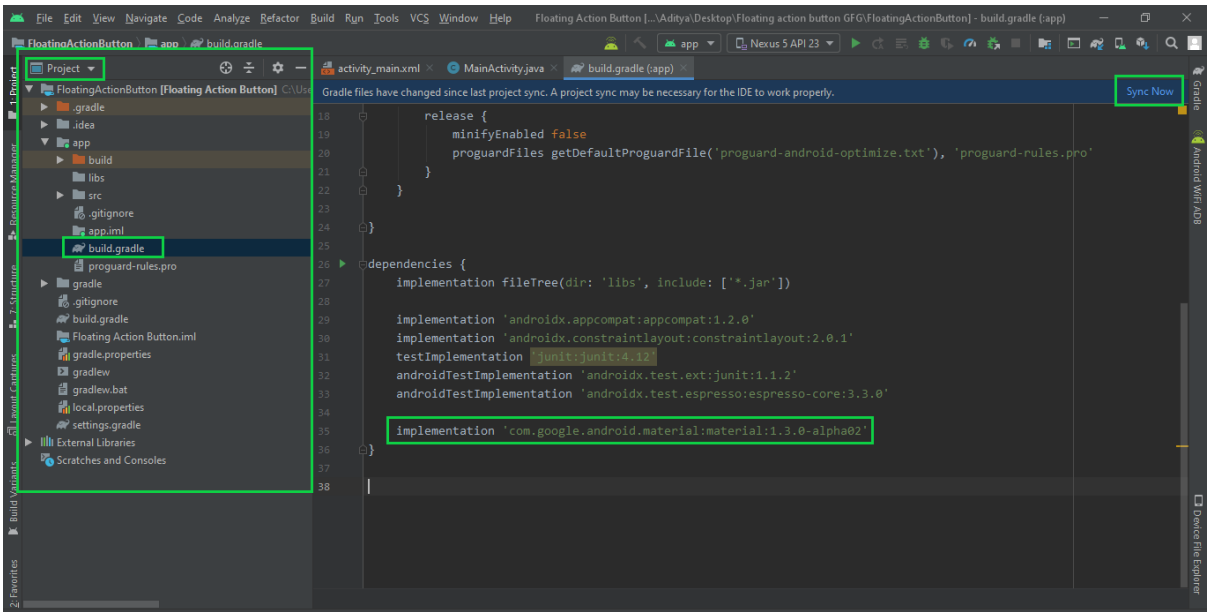
Step 2: Invoke the dependency to the app level gradle file

Invoke the Material Design dependency to app-level gradle file as:

```
implementation 'com.google.android.material:material:1.3.0-alpha03'
```

Get the app level gradle file by going to app > build.gradle file. And click on the "Sync Now" button. And make sure the system should be connected to the network.

Refer to the following image to locate and invoke the dependency in-app level gradle file (Under project hierarchy view).



Step 3: Change the base theme of the application

We need to change the base theme of the application because we are using the material design components. Otherwise, the application crashes immediately after it is launched.

To change the base theme of the application open app > src > main > res > values > styles.xml.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme"
parent="Theme.MaterialComponents.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Implementing the Material Design Filled EditText

Step 4: Working with the activity_main.xml file

1. Invoke the following code to implement the filled EditText.
2. Below is the code for the activity_main.xml file.
3. Comments are added inside the code to understand the code in more detail.

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:orientation="vertical"
    tools:context=".MainActivity"
    tools:ignore="HardcodedText">

    <!--this is the filled layout box for the edit text-->
    <!--this layout must be used to reposition or change
        the height and width of the edit text-->
    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/filledTextField"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="64dp"
        android:layout_marginEnd="32dp"
        android:hint="Enter something">

        <!--this is the actual edit text which takes the input-->
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/edit_text"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </com.google.android.material.textfield.TextInputLayout>

    <!--sample button to submit entered data
        inside from edit text-->
    <Button
        android:id="@+id/submit_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="32dp"
        android:text="Submit" />

    <!--text view which previews the entered data by user-->
    <TextView
        android:id="@+id/text_preview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="32dp"
        android:text="You Entered : "
        android:textSize="18sp" />
```

```
</LinearLayout>
```

In the above code the “com.google.android.material.textfield.TextInputLayout” makes the filled box for the EditText field.

And “com.google.android.material.textfield.TextInputEditText” which is the actual edit text which takes input from the user and this must be used to handle all the inputs in the MainActivity file.

Step 5: Working with the MainActivity file

Now invoke the following java code to handle the material design EditText.
Below is the code for the MainActivity file.

Comments are added inside the code to understand the code in more detail.

```
import android.annotation.SuppressLint
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Register the UI widgets with their appropriate IDs.
        val bSubmit = findViewById<Button>(R.id.submit_button)
        val mEditText = findViewById<EditText>(R.id.edit_text)
        val tvTextPreview = findViewById<TextView>(R.id.text_preview)

        // handle submit button to
          // preview the entered data
        bSubmit.setOnClickListener {
            tvTextPreview.text = "You Entered : " +
mEditText.text.toString()
        }
    }
}
```

Implementing the Material Design Outlined EditText

Step 6: Working with the activity_main.xml file

- Invoke the following code to implement the filled edit text.
- Only difference is the style attribute in the "com.google.android.material.textfield.TextInputLayout" to be invoked.
- Comments are added inside the code to understand the code in more detail.

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:orientation="vertical"
    tools:context=".MainActivity"
    tools:ignore="HardcodedText">

    <!--this is the outlined layout box for the edit text-->
    <!--this layout must be used to reposition or change the
        height and width of the edit text-->
    <!--to get the outlined edit text the style attribute as
        following must be invoked-->
    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/filledTextField"


style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="64dp"
        android:layout_marginEnd="32dp"
        android:hint="Enter something">

        <!--this is the actual edit text which takes the input-->
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/edit_text"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </com.google.android.material.textfield.TextInputLayout>

    <!--sample button to submit entered data inside from edit text-->
    <Button
        android:id="@+id/submit_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="32dp"
        android:text="Submit" />

    <!--text view which previews the entered data by user-->
    <TextView
        android:id="@+id/text_preview"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="32dp"
        android:text="You Entered : "
        android:textSize="18sp" />

</LinearLayout>
```

Run the app to see the results

24 Bars

24.1 ActionBar in Android

In Android applications, ActionBar is the element present at the top of the activity screen. It is a salient feature of a mobile application that has a consistent presence over all its activities. It provides a visual structure to the app and contains some of the frequently used elements for the users.

Android ActionBar was launched by Google in 2013 with the release of Android 3.0(API 11). Before that, the name of this top most visual element was AppBar. AppBar contains only the name of the application or current activity. It was not very much useful for the users and developers also have negligible option to customize it.

All applications that use the default theme provided by the Android(Theme.AppCompat.Light.DarkActionBar), contains an ActionBar by default. However, developers can customize it in several ways depending upon their needs. Components included in the ActionBar are:

- App Icon: Display the branding logo/icon of the application.
- View Controls: Section that displays the name of the application or current activity. Developers can also include spinner or tabbed navigation for switching between views.
- Action Button: Contains some important actions/elements of the app that may be required to the users frequently.
- Action Overflow: Include other actions that will be displayed as a menu.

Designing a Custom ActionBar

The following example demonstrates the steps involved in creating a custom ActionBar for the MainActivity of an application. All important aspects of visual elements like icon, title, subtitle, action buttons, and overflow menu will be covered.

Step 1: Default ActionBar

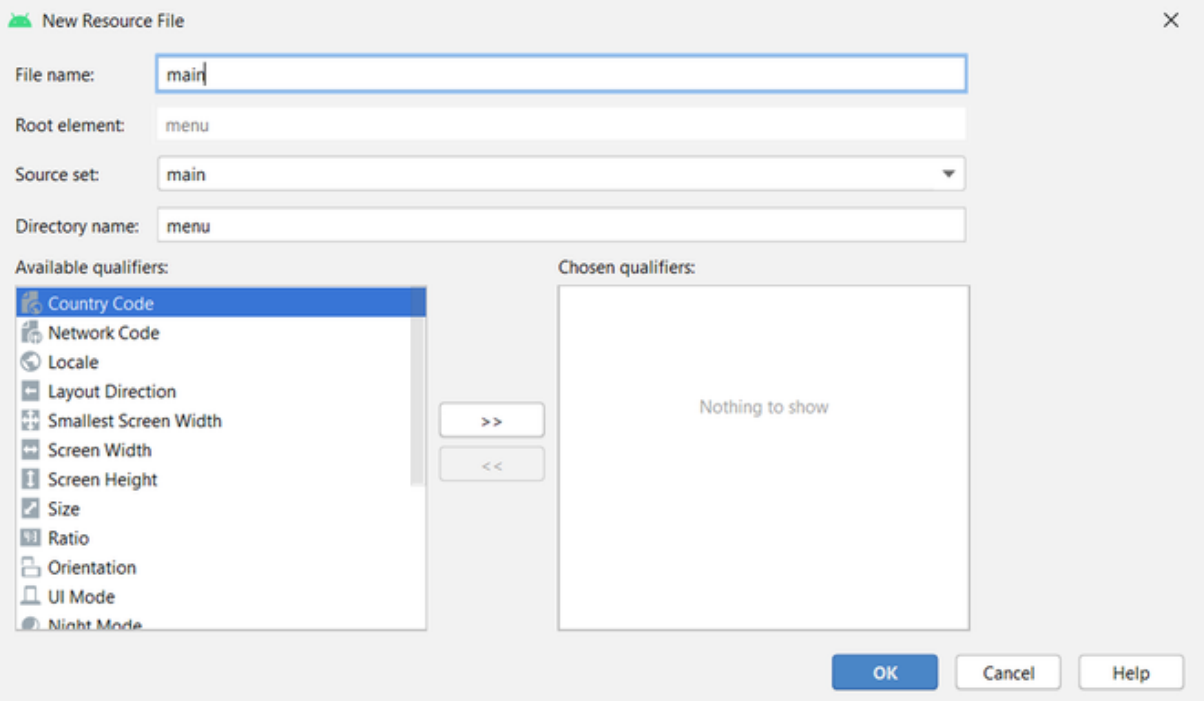
As mentioned earlier, every android app contains an ActionBar by default. This pre-included ActionBar display title for the current activity that is managed by the AncdroidManifest.xml file. The string value of the application's title is provided by @string/app_name resource present under the application nodes.

```
<application
       android:label="@string/app_name">
</application>
```

Step 2: Creating a new directory and design items of ActionBar

To code the elements of ActionBar, create a new directory in the resource folder of the application project files. Right-click on the res folder and selects New -> Directory. Give the name “menu” to the new directory.

Further, create a new Menu Resource File by right click on the menu directory. As the ActionBar is being created for the main Activity, type the name as “main” to the Menu Resource File. With this, a new file named “main.xml” must be created under the menu directory. In this file, one can declare the items which will be displayed as the action buttons of the ActionBar.



For every menu items, the following attributes are needed to be configured:

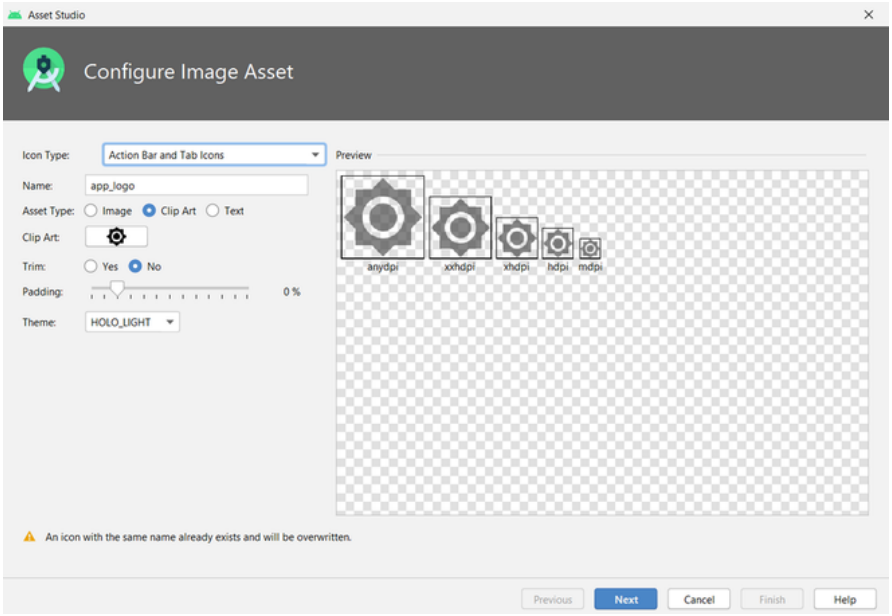
- android:title: Its value contains the title of the menu item that will be displayed when a user clicks and holds that item in the app.
- android:id: A unique ID for the menu item that will be used to access it anywhere in the whole application files.
- android:orderInCategory: The value of this attribute specify the item’s position in the ActionBar. There are two ways to define the position of different menu items. The first one is to provide the same value of this attribute for all items and the position will be defined in the same order as they are declared in the code. The second way is to provide a different numeric value for all items and then the items will position themselves according to ascending order of this attribute’s value.
- app:showAsAction: This attribute defines how the item is going to be present in the action bar. There are four possible flags to choose from:
- always: To display the item in the ActionBar all the time.
- ifRoom: To keep the item if space is available.
- never: With this flag, the item will be not be displayed as an icon in ActionBar, but will be present in the overflow menu.

- withText: To represent an item as both icon and the title, one can append this flag with the always or ifRoom flag(always|withText or ifRoom|withText).
- android:icon: The icon of an item is referenced in the drawable directories through this attribute.

Icon of an ActionBar Item

In order to provide an icon to an item, right-click on the res folder, select new, and then Image Asset. A dialog box will appear, choose the Icon Type as Action Bar and Tab Icons. Choose assets type as "Clip Art" and select an image from the clip art collection.

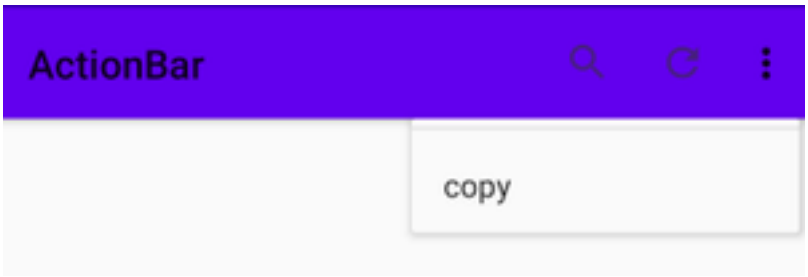
Provide a desired name to the icon. Click on Next, then Finish. This icon will now get loaded in the drawable directory of the res folder. The name provided by the developers to these icons will now be used to reference the item's icon resource.



Below is the code to place a search icon, refresh icon, and an overflow menu in the ActionBar.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- action button for search -->
    <item android:title="search"
        android:id="@+id/search"
        android:orderInCategory="100"
        app:showAsAction="ifRoom"
        android:icon="@drawable/search_icon"/>

    <!-- action button for refresh -->
    <item android:title="refresh"
        android:id="@+id/refresh"
        android:orderInCategory="100"
        app:showAsAction="ifRoom"
        android:icon="@drawable/refresh_icon"/>
```

```
    <!-- action button for copy -->
    <item android:title="copy"
        android:id="@+id/copy"
        android:orderInCategory="100"
        app:showAsAction="never"
        android:icon="@drawable/copy_icon"/>
</menu>
```



Step 3: Working with the Activity File

The items of an ActionBar is designed with a purpose to perform some operations. Those operations/actions of the items are declared in that Activity file for which the ActionBar has been designed. In this example, the target activity is the MainActivity file. Further, the custom title, subtitle, and application logo are also defined in this file. Below is the proper code to design all mentioned items and to display a toast message when a user clicks on the items of ActionBar.

```
import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // calling this activity's function to
        // use ActionBar utility methods
        val actionBar = supportActionBar

        // providing title for the ActionBar
        actionBar!!.title = "  GfG | Action Bar"

        // providing subtitle for the ActionBar
        actionBar.subtitle = "   Design a custom Action Bar"

        // adding icon in the ActionBar
        actionBar.setIcon(R.drawable.app_logo)

        // methods to display the icon in the ActionBar
        actionBar.setDisplayUseLogoEnabled(true)
        actionBar.setDisplayShowHomeEnabled(true)
    }

    // method to inflate the options menu when
    // the user opens the menu for the first time
```

```
    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.main, menu)
        return super.onCreateOptionsMenu(menu)
    }

    // methods to control the operations that will
    // happen when user clicks on the action buttons
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.search -> Toast.makeText(this, "Search Clicked",
Toast.LENGTH_SHORT).show()
            R.id.refresh -> Toast.makeText(this, "Refresh Clicked",
Toast.LENGTH_SHORT).show()
            R.id.copy -> Toast.makeText(this, "Copy Clicked",
Toast.LENGTH_SHORT).show()
        }
        return super.onOptionsItemSelected(item)
    }
}
```

Run the app to see the progress

Step 4: Color of the ActionBar

Head over to style.xml file located in the values directory of the res folder. To change the default color of the ActionBar, one has to change the colorPrimary resource. Below is the code to make the ActionBar color ‘green’.

```
<resources>
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->

    <!-- provided green color to the ActionBar -->
    <item name="colorPrimary">#0F9D58</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>

</resources>
```

Step 5: Working with activity_main.xml file

This file defines the layout of the activity. In this example, the prime focus is on ActionBar, thus the activity will contain only a simple TextView. Below is the code.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#168BC34A"
```

```
    tools:context=".MainActivity">

    <!-- Adding a TextView in the activity -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Geek!!"
        android:textColor="#000000"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Run the app to see the results

Advantages of ActionBar

- Provides a customized area to design the identity of an app
- Specify the location of the user in the app by displaying the title of the current Activity.
- Provides access to important and frequently used actions
- Support tabs and a drop-down list for view switching and navigation.

Disadvantages of ActionBar

- All features of the ActionaBar are not introduced at once but were introduced with the release of different API levels such as API 15, 17, and 19.
- The ActionBar behaves differently when it runs on different API levels.
- The features that were introduced with a particular API does not provide backward compatibility.

24.2 ProgressBar in Kotlin

Android ProgressBar is a user interface control that indicates the progress of an operation. For example, downloading a file, uploading a file on the internet we can see the progress bar to estimate the time remaining in operation. There are two modes of ProgressBar:

1. Determinate ProgressBar
2. Indeterminate ProgressBar

Determinate ProgressBar

In common, we use the Determinate progress mode in ProgressBar because it shows the quantity of progress that has occurred like the (%) percentage of the file downloaded, how much data was uploaded or downloaded on the internet, etc. If we have to use determinate we set the style of the progress bar as below:

```
<ProgressBar
    android:id="@+id/pBar"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Indeterminate ProgressBar

Here, we don't get the idea of the progress of work means how much it has been completed or How long it will take to complete.
We can use indeterminate progressBar like below by setting the indeterminate attribute as true.

```
<ProgressBar
    android:id="@+id/pBar"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:indeterminate="true"/>
```

Add ProgressBar Widget in activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <!-- adding progress bar -->
    <ProgressBar
        android:id="@+id/progress_Bar"
        style="?android:attr/progressBarStyle"
        android:layout_width="200dp"
```

```
        android:layout_height="wrap_content"
        android:layout_marginLeft="70dp"
        android:layout_marginTop="150dp"
        android:indeterminate = "true"
        android:max="100"
        android:minWidth="200dp"
        android:minHeight="50dp"
        android:visibility="invisible"
        android:layout_centerInParent="true"
        android:progress="0"
        android:layout_marginStart="70dp" />

    <!-- adding textview which will show the progress -->
    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/progress_Bar"
        android:layout_centerHorizontal="true" />

     <!-- adding button to start the progress -->
    <Button
        android:id="@+id/show_button"
        android:layout_width="191dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text_view"
        android:layout_marginLeft="70dp"
        android:layout_marginTop="20dp"
        android:text="Progress Bar"
        android:layout_centerHorizontal="true"
        android:layout_marginStart="70dp" />

</RelativeLayout>
```

Access the ProgressBar Widget in MainActivity.kt file

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.ProgressBar
import android.widget.TextView
import android.os.Handler

class MainActivity : AppCompatActivity() {

    private var progressBar: ProgressBar? = null
    private var i = 0
    private var txtView: TextView? = null
    private val handler = Handler()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // finding progressbar by its id
        progressBar = findViewById<ProgressBar>(R.id.progress_Bar) as
ProgressBar
```

```
        // finding textview by its id
        txtView = findViewById<TextView>(R.id.text_view)

        // finding button by its id
        val btn = findViewById<Button>(R.id.show_button)

        // handling click on button
        btn.setOnClickListener {
            // Before clicking the button the progress bar will invisible
            // so we have to change the visibility of the progress bar to
visible
            // setting the progressbar visibility to visible
            progressBar.visibility = View.VISIBLE

            i = progressBar.progress

            Thread(Runnable {
                // this loop will run until the value of i becomes 99
                while (i < 100) {
                    i += 1
                    // Update the progress bar and display the current
value
                    handler.post(Runnable {
                        progressBar.progress = i
                        // setting current progress to the textview
                        txtView!!.text = i.toString() + "/" +
progressBar.max
                    })
                    try {
                        Thread.sleep(100)
                    } catch (e: InterruptedException) {
                        e.printStackTrace()
                    }
                }

                // setting the visibility of the progressbar to invisible
                // or you can use View.GONE instead of invisible
                // View.GONE will remove the progressbar
                progressBar.visibility = View.INVISIBLE

            }).start()
        }
    }
}
```

AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.farizgaskin.myfirstkotlinapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

```
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Run the app to see the results

25 Google Maps

25.1 How to Generate API Key

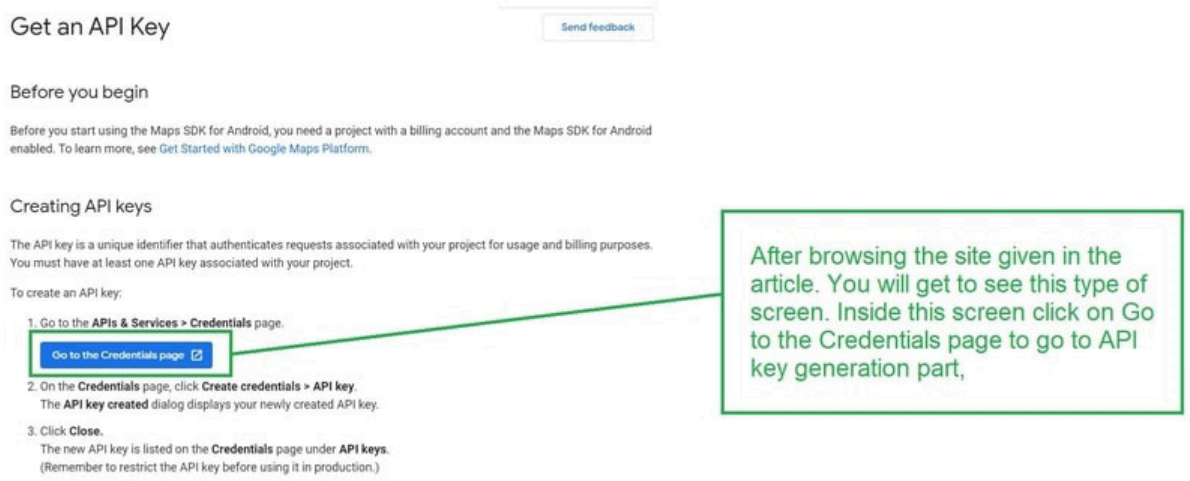
For using any Google services in Android, we have to generate an API key or we have to generate some unique key for using that service provided by Google. So Maps is one of the famous and most used services provided by Google. Maps can be seen in each and every app provided by Google and we also can use that service to add maps to our app. So in this article, we will take a look at the process of generating an API key for using Google Maps.

Step by Step Implementation

Step 1: Browse the below site on your browser

Browse to the link below to go to the Google Maps developer platform. You will get to see the below page. After click on the above link, you will get to see the below screen. Inside this click on the Go to Credentials page to go to the Credentials page for API key generation.

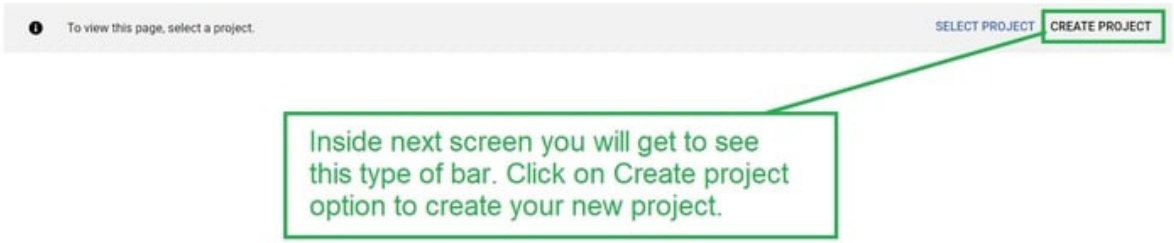
https://developers.google.com/maps/documentation/android-sdk/get-api-key



After clicking on this option a new page will open inside that screen click on create a new project option in the top bar.

Step 2: Creating a new project

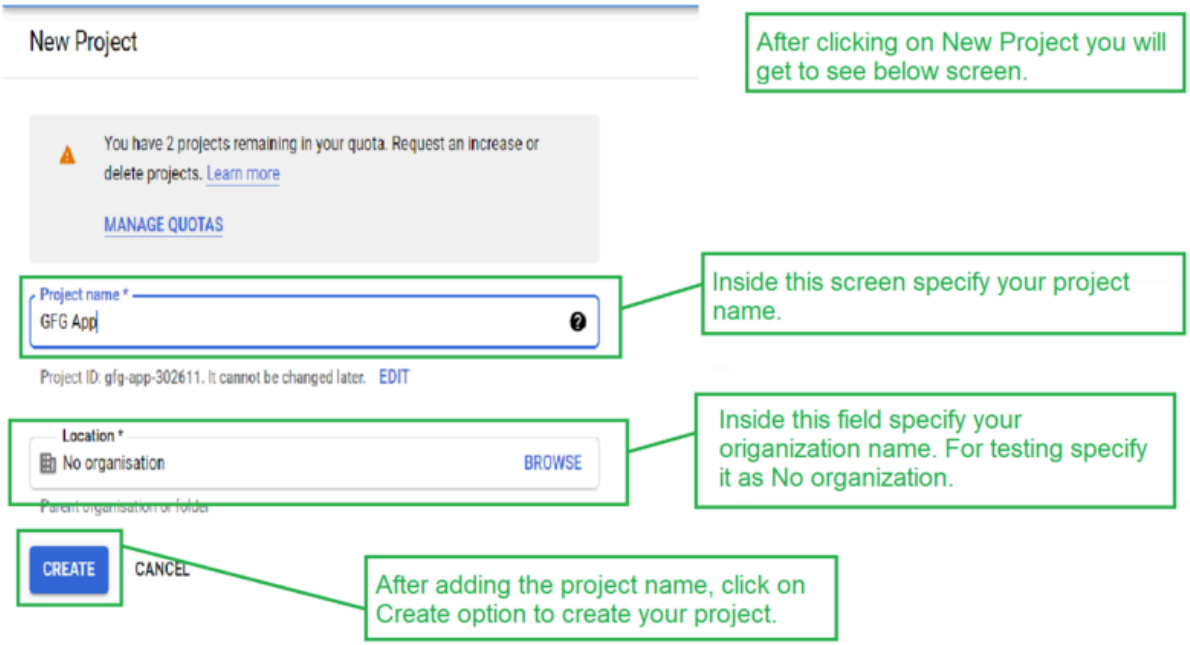
To create a new project click on Create Project option as shown below.



After clicking on create project option. You will get to see the below screen for creating a new project.

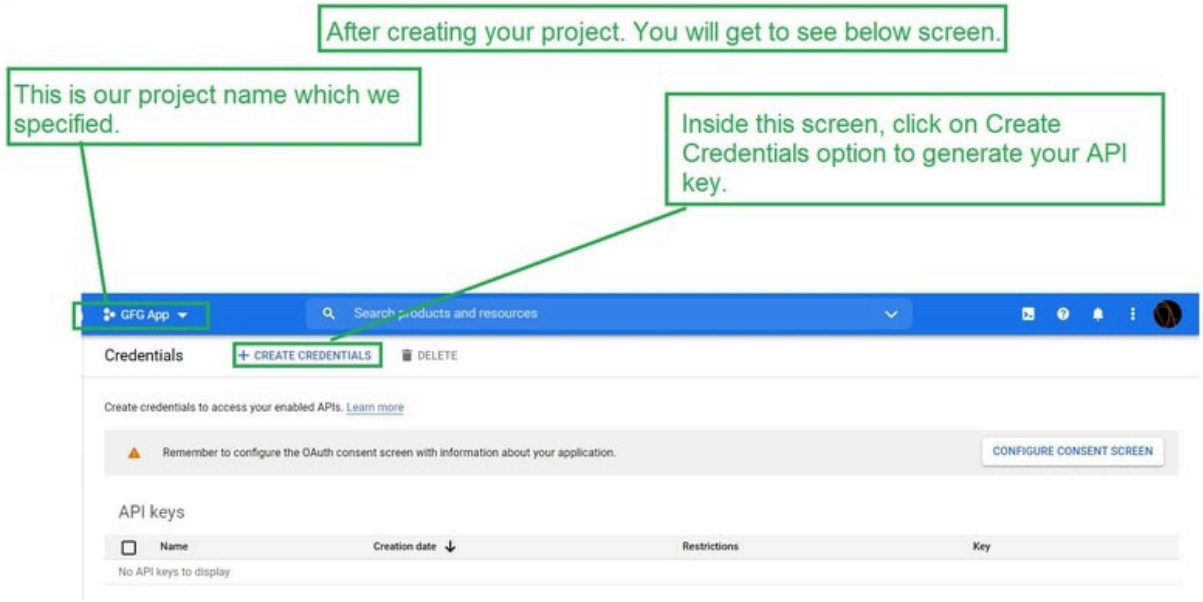
Step 3: Add your name and organization for your project

Add your project name and organization name in the fields present on the screen. For testing, you can use you can provide a no organization option to proceed further.

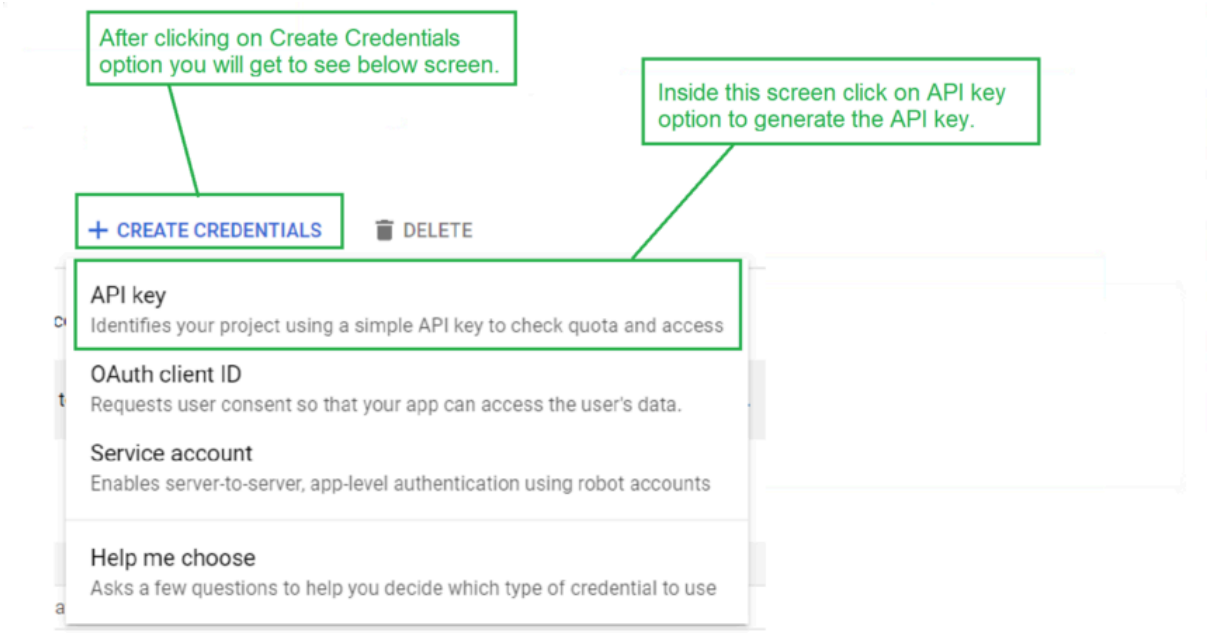


After adding your project name and organization name. Click on Create option to proceed further.

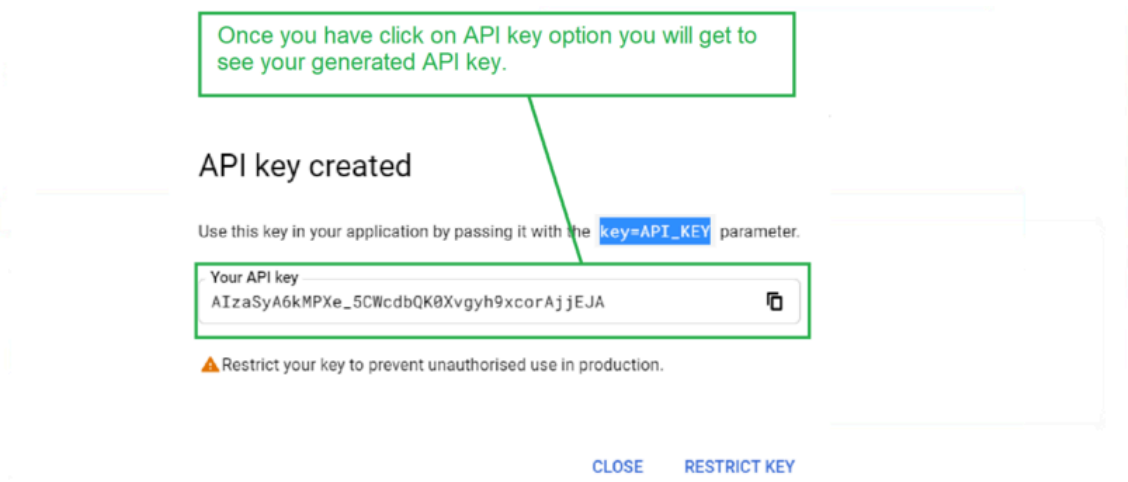
Step 4: Create credentials for your API key



To create an API key for Maps click on Create credentials option and then select the API key option as shown below.



Click on the API key option to generate your API key. After clicking on this option your API key will be generated which is shown below.



Now you can use this API for using maps.

25.2 Add Custom Marker

Google Maps is used in many apps for displaying a location and indicating a specific location on a map. We have seen the use of maps in many apps that provides services such as Ola, Uber, and many more. In these apps, you will get to see How we can add Custom Marker to Google Maps in Android.

What we are going to build in this article?

We will be building a simple application in which we will display a map and on that map, we will be displaying a custom marker on our app. Below is the screenshot in which we will get to see what we are going to do in this project.

Note that we are going to implement this project using the Java language.

Default Marker

Custom Marker

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language. Make sure to select Maps Activity while creating a new Project.

Step 2: Generating an API key for using Google Maps

To generate the API key for Maps you may refer to How to Generate API Key for Using Google Maps in Android. After generating your API key for Google Maps. We have to add this key to our Project. For adding this key in our app navigate to the values folder > google_maps_api.xml file and at line 23 you have to add your API key in the place of YOUR_API_KEY.

After adding this now we are ready for adding the custom marker to our app. After adding the API key, you can run your app and you will get to see a default marker on the Sydney location with the default marker.

Step 3: Adding a custom marker in Google Maps

For adding a custom marker to Google Maps navigate to the app > res > drawable > Right-Click on it > New > Vector Assets and select the icon which we have to show on your Map. You can change the color according to our requirements. After creating this icon now we will move towards adding this marker to our Map.

Step 4: Working with the MapsActivity.java file

Go to the MapsActivity.java file and refer to the following code. Below is the code for the MapsActivity.java file. Comments are added inside the code to understand the code in more detail.

```
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.os.Bundle;

import androidx.core.content.ContextCompat;
import androidx.fragment.app.FragmentActivity;

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.BitmapDescriptor;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified
        // when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move
the camera. In this case,
     * we just add a marker near Sydney, Australia.
```

```
     * If Google Play services is not installed on the device, the user
will be prompted to install
     * it inside the SupportMapFragment. This method will only be
triggered once the user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker
in Sydney")
                // below line is use to add custom marker on our map.
                .icon(BitmapFromVector(getApplicationContext(),
R.drawable.ic_flag)));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }

    private BitmapDescriptor BitmapFromVector(Context context, int
vectorResId) {
        // below line is use to generate a drawable.
        Drawable vectorDrawable = ContextCompat.getDrawable(context,
vectorResId);

        // below line is use to set bounds to our vector drawable.
        vectorDrawable.setBounds(0, 0,
vectorDrawable.getIntrinsicWidth(), vectorDrawable.getIntrinsicHeight());

        // below line is use to create a bitmap for our
        // drawable which we have added.
        Bitmap bitmap =
Bitmap.createBitmap(vectorDrawable.getIntrinsicWidth(),
vectorDrawable.getIntrinsicHeight(), Bitmap.Config.ARGB_8888);

        // below line is use to add bitmap in our canvas.
        Canvas canvas = new Canvas(bitmap);

        // below line is use to draw our
        // vector drawable in canvas.
        vectorDrawable.draw(canvas);

        // after generating our bitmap we are returning our bitmap.
        return BitmapDescriptorFactory.fromBitmap(bitmap);
    }
}
```

Run the app to see the results

25.3 Add Multiple Markers

Google Map is used in most of the apps which are used to represent many locations and markers on Google Maps. We have seen markers on Google maps for multiple locations. In this article, we will take a look at the implementation of Multiple Markers on Google Maps in Android.

What we are going to build in this article?

We will be building a simple application in which we will be displaying multiple markers on maps in different locations. A sample image is given below to get an idea about what we are going to do in this article. Note that we are going to implement this project using the Java language.

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language. Make sure to select Maps Activity while creating a new Project.

Step 2: Generating an API key for using Google Maps

To generate the API key for Maps you may refer to How to Generate API Key for Using Google Maps in Android. After generating your API key for Google Maps. We have to add this key to our Project. For adding this key in our app navigate to the values folder > google_maps_api.xml file and at line 23 you have to add your API key in the place of YOUR_API_KEY.

Step 3: Adding Multiple Markers to our Map

Go to the MapsActivity.java file and refer to the following code. Below is the code for the MapsActivity.java file. Comments are added inside the code to understand the code in more detail.

```
import android.os.Bundle;

import androidx.fragment.app.FragmentActivity;

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 com.google.android.gms.maps.model.MarkerOptions;

import java.util.ArrayList;

public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;

    // below are the latitude and longitude
    // of 4 different locations.
    LatLng sydney = new LatLng(-34, 151);
    LatLng TamWorth = new LatLng(-31.083332, 150.916672);
    LatLng NewCastle = new LatLng(-32.916668, 151.750000);
    LatLng Brisbane = new LatLng(-27.470125, 153.021072);

    // creating array list for adding all our locations.
    private ArrayList<LatLng> locationArrayList;

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

        // Obtain the SupportMapFragment and get notified when the map is
ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        // in below line we are initializing our array list.
        locationArrayList = new ArrayList<>();

        // on below line we are adding our
        // locations in our array list.
        locationArrayList.add(sydney);
        locationArrayList.add(TamWorth);
        locationArrayList.add(NewCastle);
        locationArrayList.add(Brisbane);
    }

    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move
the camera. In this case,
     * we just add a marker near Sydney, Australia.
```

```
     * If Google Play services is not installed on the device, the user
will be prompted to install
     * it inside the SupportMapFragment. This method will only be
triggered once the user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        // inside on map ready method
        // we will be displaying all our markers.
        // for adding markers we are running for loop and
        // inside that we are drawing marker on our map.
        for (int i = 0; i < locationArrayList.size(); i++) {

            // below line is use to add marker to each location of our
array list.
            mMap.addMarker(new
MarkerOptions().position(locationArrayList.get(i)).title("Marker"));

            // below lin is use to zoom our camera on map.
            mMap.animateCamera(CameraUpdateFactory.zoomTo(18.0f));

            // below line is use to move our camera to the specific
location.

mMap.moveCamera(CameraUpdateFactory.newLatLng(locationArrayList.get(i)));
        }
    }
}
```

After adding this code. Now run your app and see the output of the app.

26 Chart

26.1 How to add a Pie Chart

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportions. It depicts a special chart that uses “pie slices”, where each sector shows the relative sizes of data. A circular chart cuts in a form of radii into segments describing relative frequencies or magnitude also known as circle graph. A pie chart represents numbers in percentages, and the total sum of all segments needs to equal 100%.

Step1: Opening a new project

1. Open a new project just click of File option at topmost corner in left.
2. Then click on new and open a new project with whatever name you want.
3. Now we gonna work on Empty Activity with language as Java. Leave all other options as untouched.
4. You can change the name of project as per your choice.
5. By default, there will be two files activity_main.xml and MainActivity.java.

Step 2: Before going to the coding section first you have to do some pre-task.

Go to app->res->values->colors.xml section and set the colors for your app.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#024265</color>
    <color name="colorPrimaryDark">#024265</color>
    <color name="colorAccent">#05af9b</color>

    <color name="color_one">#fb7268</color>
    <color name="color_white">#ededf2</color>
    <color name="color_two">#E3E0E0</color>

    <color name="R">#FFA726</color>
    <color name="Python">#66BB6A</color>
    <color name="CPP">#EF5350</color>
    <color name="Java">#29B6F6</color>

</resources>
```

Go to Gradle Scripts->build.gradle (Module: app) section and import following dependencies and click the “sync Now” on the above pop up.

```
// For Card view
implementation 'androidx.cardview:cardview:1.0.0'

// Chart and graph library
implementation 'com.github.blackfizz:eazegraph:1.2.5l@aar'
implementation 'com.nineoldandroids:library:2.4.0'
```

Step3: Designing the UI

Below is the code for activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/color_white"
    tools:context=".MainActivity">

    <!-- Card view for displaying the --->
    <!-- Pie chart and details of pie chart -->
    <androidx.cardview.widget.CardView
        android:id="@+id/cardViewGraph"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginTop="15dp"
        android:elevation="10dp"
        app:cardCornerRadius="10dp">


        <!--Linear layout to display pie chart --->
        <!-- and details of pie chart-->

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal"
            android:weightSum="2">

            <!--Pie chart to display the data-->

            <org.eazegraph.lib.charts.PieChart
                xmlns:app="http://schemas.android.com/apk/res-auto"
                android:id="@+id/piechart"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:padding="6dp"
                android:layout_weight="1"
                android:layout_marginTop="15dp"
                android:layout_marginLeft="15dp"
                android:layout_marginBottom="15dp"

                />

            <!--Creating another linear layout --->
            <!-- to display pie chart details -->
            <LinearLayout
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:layout_marginLeft="20dp"
```

```
                android:orientation="vertical"
                android:gravity="center_vertical"
                >

                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="15dp"
                    android:layout_gravity="center_vertical">

                    <!--View to display the yellow color icon-->
                    <View
                        android:layout_width="15dp"
                        android:layout_height="match_parent"
                        android:background="@color/R"/>

                    <!--Text view to display R -->
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="R"
                        android:paddingLeft="10dp"/>

                </LinearLayout>

                <!--Linear layout to display Python-->
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="15dp"
                    android:layout_gravity="center_vertical"
                    android:layout_marginTop="5dp">

                    <!--View to display the green color icon-->
                    <View
                        android:layout_width="15dp"
                        android:layout_height="match_parent"
                        android:background="@color/Python"/>

                    <!--Text view to display python text -->
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="Python"
                        android:paddingLeft="10dp"/>

                </LinearLayout>

                <!--Linear layout to display C++-->
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="15dp"
                    android:layout_gravity="center_vertical"
                    android:layout_marginTop="5dp">

                    <!--View to display the red color icon-->
                    <View
                        android:layout_width="15dp"
                        android:layout_height="match_parent"
                        android:background="@color/CPP"/>

                    <!--Text view to display C++ text -->
                    <TextView
```

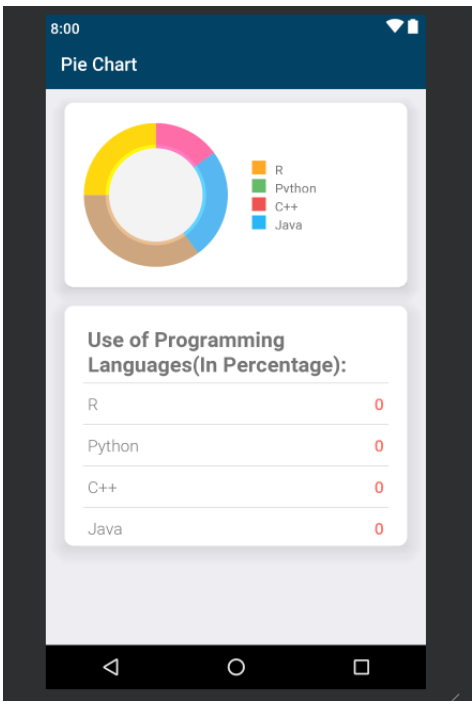
```
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="C++"
                        android:paddingLeft="10dp"/>

                </LinearLayout>

                <!--Linear layout to display Java-->
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="15dp"
                    android:layout_gravity="center_vertical"
                    android:layout_marginTop="5dp">

                    <!--View to display the blue color icon-->
                    <View
                        android:layout_width="15dp"
                        android:layout_height="match_parent"
                        android:background="@color/Java"/>

                    <!--Text view to display Java text -->
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="Java"
                        android:paddingLeft="10dp"/>

                </LinearLayout>

            </LinearLayout>

        </LinearLayout>

    </androidx.cardview.widget.CardView>


    <!-- Another Card view for displaying --->
    <!-- Use of programming languages -->
    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="260dp"
        android:layout_below="@+id/cardViewGraph"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"
        android:elevation="10dp"
        app:cardCornerRadius="10dp"
        android:id="@+id/details">

        <!--Relative layout to display --->
        <!-- use of programming languages -->
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:orientation="vertical">

                    <!--Text view to use of --->
                    <!-- programming languages text-->
                    <TextView
                        android:layout_width="match_parent"
```

```
                        android:layout_height="wrap_content"
                        android:text="Use of Programming Languages(In
Percentage):"
                        android:textSize="23sp"
                        android:textStyle="bold"
                        android:layout_marginLeft="25dp"
                        android:layout_marginTop="20dp"/>

                    <!--View to display the line-->
                    <View
                        android:layout_width="match_parent"
                        android:layout_height="1dp"
                        android:background="@color/color_two"
                        android:layout_marginLeft="20dp"
                        android:layout_marginRight="20dp"
                        android:layout_marginTop="5dp"/>

                    <RelativeLayout
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_marginRight="25dp"
                        android:layout_marginLeft="25dp"
                        android:layout_marginTop="10dp"
                        android:layout_marginBottom="10dp">

                        <!--Text view to display R -->
                        <TextView
                            android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
                            android:fontFamily="sans-serif-light"
                            android:text="R"
                            android:textSize="18sp"/>

                        <!--Text view to display the --->
                        <!-- percentage of programming language --->
                        <!-- used. For now default set to 0-->
                        <TextView
                            android:layout_width="fill_parent"
                            android:layout_height="wrap_content"
                            android:text="0"
                            android:id="@+id/tvR"
                            android:textAlignment="textEnd"
                            android:textSize="18sp"
                            android:textColor="@color/color_one"
                            android:textStyle="bold"
                            android:fontFamily="sans-serif-light"
                            android:layout_alignParentRight="true"/>
                    </RelativeLayout>

                    <!--View to display the line-->
                    <View
                        android:layout_width="match_parent"
                        android:layout_height="1dp"
                        android:background="@color/color_two"
                        android:layout_marginLeft="20dp"
                        android:layout_marginRight="20dp" />

                    <RelativeLayout
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_marginRight="25dp"
```

```
                android:layout_marginLeft="25dp"
                android:layout_marginTop="10dp"
                android:layout_marginBottom="10dp">

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:fontFamily="sans-serif-light"
                    android:text="Python"
                    android:textSize="18sp"/>

                <!--Text view to display the percentage --->
                <!-- of programming language used. --->
                <!-- For now default set to 0-->

                <TextView
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:text="0"
                    android:id="@+id/tvPython"
                    android:textAlignment="textEnd"
                    android:textSize="18sp"
                    android:textColor="@color/color_one"
                    android:textStyle="bold"
                    android:fontFamily="sans-serif-light"
                    android:layout_alignParentRight="true"/>

            </RelativeLayout>
            <View
                android:layout_width="match_parent"
                android:layout_height="1dp"
                android:background="@color/color_two"
                android:layout_marginLeft="20dp"
                android:layout_marginRight="20dp" />

            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginRight="25dp"
                android:layout_marginLeft="25dp"
                android:layout_marginTop="10dp"
                android:layout_marginBottom="10dp">

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:fontFamily="sans-serif-light"
                    android:text="C++"
                    android:textSize="18sp"/>

                <!--Text view to display the percentage --->
                <!-- of programming language used. --->
                <!-- For now default set to 0-->
                <TextView
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:text="0"
                    android:id="@+id/tvCPP"
                    android:textAlignment="textEnd"
                    android:textSize="18sp"
                    android:textColor="@color/color_one"
```

```
                            android:textStyle="bold"
                            android:fontFamily="sans-serif-light"
                            android:layout_alignParentRight="true"/>

                    </RelativeLayout>
                    <View
                        android:layout_width="match_parent"
                        android:layout_height="1dp"
                        android:background="@color/color_two"
                        android:layout_marginLeft="20dp"
                        android:layout_marginRight="20dp" />

                    <RelativeLayout
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_marginRight="25dp"
                        android:layout_marginLeft="25dp"
                        android:layout_marginTop="10dp"
                        android:layout_marginBottom="10dp">

                        <TextView
                            android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
                            android:fontFamily="sans-serif-light"
                            android:text="Java"
                            android:textSize="18sp"/>

                        <!--Text view to display the percentage --->
                        <!-- of programming language used. --->
                        <!-- For now default set to 0-->
                        <TextView
                            android:layout_width="fill_parent"
                            android:layout_height="wrap_content"
                            android:text="0"
                            android:id="@+id/tvJava"
                            android:textAlignment="textEnd"
                            android:textSize="18sp"
                            android:textColor="@color/color_one"
                            android:textStyle="bold"
                            android:fontFamily="sans-serif-light"
                            android:layout_alignParentRight="true"/>

                    </RelativeLayout>

                </LinearLayout>

    </androidx.cardview.widget.CardView>

</RelativeLayout>
```

After using this code in .xml file, the UI will be like:



Step4: Working with Java file

Open the MainActivity.java file there within the class, first of all create the object of TextView class and the pie chart class.

```
// Create the object of TextView and PieChart class
TextView tvR, tvPython, tvCPP, tvJava;
PieChart pieChart;
```

Secondly inside onCreate() method, we have to link those objects with their respective id's that we have given in .XML file.

```
// Link those objects with their respective
// id's that we have given in .XML file
tvR = findViewById(R.id.tvR);
tvPython = findViewById(R.id.tvPython);
tvCPP = findViewById(R.id.tvCPP);
tvJava = findViewById(R.id.tvJava);
pieChart = findViewById(R.id.piechart);
```

Creat a private void setData() method outside onCreate() method and define it.

Inside setData() method the most important task is going to happen that is how we set the data in the text file and as well as on the piechart.

First of all inside setData() method set the percentage of language used in their respective text view.

```
// Set the percentage of language used
tvR.setText(Integer.toString(40));
tvPython.setText(Integer.toString(30));
tvCPP.setText(Integer.toString(5));
tvJava.setText(Integer.toString(25));
```

And then set this data to the pie chart and also set their respective colors using addPieSlice() method.

```
// Set the data and color to the pie chart
pieChart.addPieSlice(
    new PieModel(
        "R",
        Integer.parseInt(tvR.getText().toString()),
        Color.parseColor("#FFA726")));
pieChart.addPieSlice(
    new PieModel(
        "Python",
        Integer.parseInt(tvPython.getText().toString()),
        Color.parseColor("#66BB6A")));
pieChart.addPieSlice(
    new PieModel(
        "C++",
        Integer.parseInt(tvCPP.getText().toString()),
        Color.parseColor("#EF5350")));
pieChart.addPieSlice(
    new PieModel(
        "Java",
        Integer.parseInt(tvJava.getText().toString()),
        Color.parseColor("#29B6F6")));
```

For better look animate the piechart using startAnimation()

```
// To animate the pie chart
pieChart.startAnimation();
```

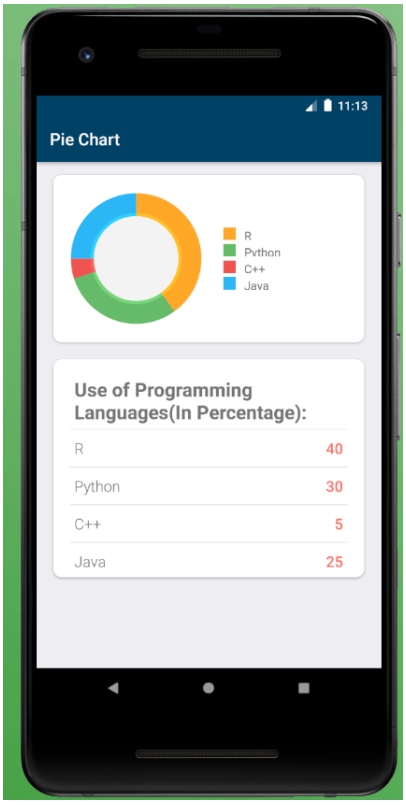
At last invoke the setData() method inside onCreate() method.

Below is the complete code for MainActivity.java file:

```
package com.example.piechart;

// Import the required libraries
import androidx.appcompat.app.AppCompatActivity;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.TextView;
import org.eazegraph.lib.charts.PieChart;
import org.eazegraph.lib.models.PieModel;

public class MainActivity
    extends AppCompatActivity {

    // Create the object of TextView
    // and PieChart class
    TextView tvR, tvPython, tvCPP, tvJava;
    PieChart pieChart;

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

        // Link those objects with their
        // respective id's that
        // we have given in .XML file
        tvR = findViewById(R.id.tvR);
        tvPython = findViewById(R.id.tvPython);
        tvCPP = findViewById(R.id.tvCPP);
        tvJava = findViewById(R.id.tvJava);
        pieChart = findViewById(R.id.piechart);

        // Creating a method setData()
        // to set the text in text view and pie chart
        setData();
    }

    private void setData()
    {

        // Set the percentage of language used
        tvR.setText(Integer.toString(40));
        tvPython.setText(Integer.toString(30));
        tvCPP.setText(Integer.toString(5));
        tvJava.setText(Integer.toString(25));

        // Set the data and color to the pie chart
        pieChart.addPieSlice(
            new PieModel(
                "R",
                Integer.parseInt(tvR.getText().toString()),
                Color.parseColor("#FFA726")));
        pieChart.addPieSlice(
            new PieModel(
                "Python",
                Integer.parseInt(tvPython.getText().toString()),
                Color.parseColor("#66BB6A")));
```

```
        pieChart.addPieSlice(
            new PieModel(
                "C++",
                Integer.parseInt(tvCPP.getText().toString()),
                Color.parseColor("#EF5350")));
        pieChart.addPieSlice(
            new PieModel(
                "Java",
                Integer.parseInt(tvJava.getText().toString()),
                Color.parseColor("#29B6F6")));

        // To animate the pie chart
        pieChart.startAnimation();
    }
}
```

Run the app to see the results



27 Database

27.1 Room DB – How to perform CRUD

Data from the app can be saved on users' devices in different ways. We can store data in the user's device in SQLite tables, shared preferences, and many more ways. In this article, we will take a look at saving data, reading, updating, and deleting data in Room Database on Android. We will perform CRUD operations using Room Database on Android. In this article, we will take a look at performing CRUD operations in Room Database in Android.

What we are going to build in this article?

We will be building a simple application in which we will be adding the different types of courses that are available on FGTech. We will be storing all this data in Room Database and performing CRUD operations on these courses.

What is Room?

Room is a persistence library that provides an abstraction layer over the SQLite database to allow a more robust database. With the help of room, we can easily create the database and perform CRUD operations very easily.

Components of Room

The three main components of the room are Entity, Database, and DAO.

- Entity: Entity is a modal class that is annotated with @Entity. This class is having variables that will be our columns and the class is our table.
- Database: It is an abstract class where we will be storing all our database entries which we can call Entities.
- DAO: The full form of DAO is a Database access object which is an interface class with the help of it we can perform different operations in our database.

Now, let's move towards the implementation of Room Database in Android.

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language.

Step 2: Adding dependency for using Room in build.gradle files

Navigate to the app > Gradle Scripts > build.gradle file and add the below dependencies in the dependencies section.

```
// add below dependency for using room.
implementation 'androidx.room:room-runtime:2.2.5'
annotationProcessor 'androidx.room:room-compiler:2.2.5'

// add below dependency for using lifecycle extensions for room.
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
annotationProcessor 'androidx.lifecycle:lifecycle-compiler:2.2.0'
```

After adding the above dependencies section. Now sync your project and we will move towards our XML file.

Step 3: Working with the activity_main.xml file

Navigate to the app > res > layout > activity_main.xml and add the below code to that file. Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!--recycler view to display our data-->
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/idRVCourses"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!--fab to add new courses-->

<com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/idFABAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="18dp"
        android:layout_marginTop="18dp"
        android:layout_marginEnd="18dp"
        android:layout_marginBottom="18dp"
        android:src="@android:drawable/ic_input_add"
        app:tint="@color/white" />

</RelativeLayout>
```

Step 4: Creating a modal class for storing our data

Navigate to the app > java > your apps package name > Right-click on it > New > Java class and name the class as CourseModal and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import androidx.room.Entity;
import androidx.room.PrimaryKey;

// below line is for setting table name.
@Entity(tableName = "course_table")
public class CourseModal {

    // below line is to auto increment
    // id for each course.
    @PrimaryKey(autoGenerate = true)

    // variable for our id.
    private int id;

    // below line is a variable
    // for course name.
    private String courseName;

    // below line is use for
    // course description.
    private String courseDescription;

    // below line is use
    // for course duration.
    private String courseDuration;

    // below line we are creating constructor class.
    // inside constructor class we are not passing
    // our id because it is incrementing automatically
    public CourseModal(String courseName, String courseDescription,
String courseDuration) {
        this.courseName = courseName;
        this.courseDescription = courseDescription;
        this.courseDuration = courseDuration;
    }

    // on below line we are creating
    // getter and setter methods.
    public String getCourseName() {
        return courseName;
    }

    public void setCourseName(String courseName) {
        this.courseName = courseName;
    }

    public String getCourseDescription() {
        return courseDescription;
    }

    public void setCourseDescription(String courseDescription) {
        this.courseDescription = courseDescription;
    }
```

```
    public String getCourseDuration() {
        return courseDuration;
    }

    public void setCourseDuration(String courseDuration) {
        this.courseDuration = courseDuration;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

Step 5: Creating a Dao interface for our database

Navigate to the app > java > your app's package name > Right-click on it > New > Java class and name as Dao and select Interface. After creating an interface class and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import androidx.lifecycle.LiveData;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

// Adding annotation
// to our Dao class
@androidx.room.Dao
public interface Dao {

    // below method is use to
    // add data to database.
    @Insert
    void insert(CourseModal model);

    // below method is use to update
    // the data in our database.
    @Update
    void update(CourseModal model);

    // below line is use to delete a
    // specific course in our database.
    @Delete
    void delete(CourseModal model);

    // on below line we are making query to
    // delete all courses from our database.
    @Query("DELETE FROM course_table")
    void deleteAllCourses();

    // below line is to read all the courses from our database.
```

```
    // in this we are ordering our courses in ascending order
    // with our course name.
    @Query("SELECT * FROM course_table ORDER BY courseName ASC")
    LiveData<List<CourseModal>> getAllCourses();
}
```

Step 6: Creating a database class

Navigate to the app > java > your app's package name > Right-click on it > New > Java class and name it as CourseDatabase and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import android.content.Context;
import android.os.AsyncTask;

import androidx.annotation.NonNull;
import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import androidx.sqlite.db.SupportSQLiteDatabase;

// adding annotation for our database entities and db version.
@Database(entities = {CourseModal.class}, version = 1)
public abstract class CourseDatabase extends RoomDatabase {

    // below line is to create instance
    // for our database class.
    private static CourseDatabase instance;

    // below line is to create
    // abstract variable for dao.
    public abstract Dao Dao();

    // on below line we are getting instance for our database.
    public static synchronized CourseDatabase getInstance(Context
context) {
        // below line is to check if
        // the instance is null or not.
        if (instance == null) {
            // if the instance is null we
            // are creating a new instance
            instance =
                    // for creating a instance for our database
                    // we are creating a database builder and passing
                    // our database class with our database name.
                    Room.databaseBuilder(context.getApplicationContext(),
                            CourseDatabase.class, "course_database")
                            // below line is use to add fall back to
                            // destructive migration to our database.
                            .fallbackToDestructiveMigration()
                            // below line is to add callback
                            // to our database.
                            .addCallback(roomCallback)
                            // below line is to
                            // build our database.
                            .build();
        }
        // after creating an instance
```

```
        // we are returning our instance
        return instance;
    }

    // below line is to create a callback for our room database.
    private static RoomDatabase.Callback roomCallback = new
RoomDatabase.Callback() {
        @Override
        public void onCreate(@NonNull SupportSQLiteDatabase db) {
            super.onCreate(db);
            // this method is called when database is created
            // and below line is to populate our data.
            new PopulateDbAsyncTask(instance).execute();
        }
    };

    // we are creating an async task class to perform task in background.
    private static class PopulateDbAsyncTask extends AsyncTask<Void,
Void, Void> {
        PopulateDbAsyncTask(CourseDatabase instance) {
            Dao dao = instance.Dao();
        }
        @Override
        protected Void doInBackground(Void... voids) {
            return null;
        }
    }
}
```

Step 7: Create a new java class for our Repository

Navigate to the app > java > your app's package name > Right-click on it > New > Java class and name it as CourseRepository and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import android.app.Application;
import android.os.AsyncTask;

import androidx.lifecycle.LiveData;

import java.util.List;

public class CourseRepository {

    // below line is the create a variable
    // for dao and list for all courses.
    private Dao dao;
    private LiveData<List<CourseModal>> allCourses;

    // creating a constructor for our variables
    // and passing the variables to it.
    public CourseRepository(Application application) {
        CourseDatabase database =
CourseDatabase.getInstance(application);
        dao = database.Dao();
        allCourses = dao.getAllCourses();
    }
```

```
    // creating a method to insert the data to our database.
    public void insert(CourseModal model) {
        new InsertCourseAsyncTask(dao).execute(model);
    }

    // creating a method to update data in database.
    public void update(CourseModal model) {
        new UpdateCourseAsyncTask(dao).execute(model);
    }

    // creating a method to delete the data in our database.
    public void delete(CourseModal model) {
        new DeleteCourseAsyncTask(dao).execute(model);
    }

    // below is the method to delete all the courses.
    public void deleteAllCourses() {
        new DeleteAllCoursesAsyncTask(dao).execute();
    }

    // below method is to read all the courses.
    public LiveData<List<CourseModal>> getAllCourses() {
        return allCourses;
    }

    // we are creating a async task method to insert new course.
    private static class InsertCourseAsyncTask extends
AsyncTask<CourseModal, Void, Void> {
        private Dao dao;

        private InsertCourseAsyncTask(Dao dao) {
            this.dao = dao;
        }

        @Override
        protected Void doInBackground(CourseModal... model) {
            // below line is use to insert our modal in dao.
            dao.insert(model[0]);
            return null;
        }
    }

    // we are creating a async task method to update our course.
    private static class UpdateCourseAsyncTask extends
AsyncTask<CourseModal, Void, Void> {
        private Dao dao;

        private UpdateCourseAsyncTask(Dao dao) {
            this.dao = dao;
        }

        @Override
        protected Void doInBackground(CourseModal... models) {
            // below line is use to update
            // our modal in dao.
            dao.update(models[0]);
            return null;
        }
    }

    // we are creating a async task method to delete course.
```

```
    private static class DeleteCourseAsyncTask extends
AsyncTask<CourseModal, Void, Void> {
        private Dao dao;

        private DeleteCourseAsyncTask(Dao dao) {
            this.dao = dao;
        }

        @Override
        protected Void doInBackground(CourseModal... models) {
            // below line is use to delete
            // our course modal in dao.
            dao.delete(models[0]);
            return null;
        }
    }

    // we are creating a async task method to delete all courses.
    private static class DeleteAllCoursesAsyncTask extends
AsyncTask<Void, Void, Void> {
        private Dao dao;
        private DeleteAllCoursesAsyncTask(Dao dao) {
            this.dao = dao;
        }
        @Override
        protected Void doInBackground(Void... voids) {
            // on below line calling method
            // to delete all courses.
            dao.deleteAllCourses();
            return null;
        }
    }
}
```

Step 8: Creating a class for our Repository

Navigate to the app > java > your app's package name > Right-click on it > New > Java Class and name the class as ViewModal and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import android.app.Application;

import androidx.annotation.NonNull;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;

import java.util.List;

public class ViewModal extends AndroidViewModel {

    // creating a new variable for course repository.
    private CourseRepository repository;

    // below line is to create a variable for live
    // data where all the courses are present.
    private LiveData<List<CourseModal>> allCourses;

    // constructor for our view modal.
```

```
    public ViewModal(@NonNull Application application) {
        super(application);
        repository = new CourseRepository(application);
        allCourses = repository.getAllCourses();
    }

    // below method is use to insert the data to our repository.
    public void insert(CourseModal model) {
        repository.insert(model);
    }

    // below line is to update data in our repository.
    public void update(CourseModal model) {
        repository.update(model);
    }

    // below line is to delete the data in our repository.
    public void delete(CourseModal model) {
        repository.delete(model);
    }

    // below method is to delete all the courses in our list.
    public void deleteAllCourses() {
        repository.deleteAllCourses();
    }

    // below method is to get all the courses in our list.
    public LiveData<List<CourseModal>> getAllCourses() {
        return allCourses;
    }
}
```

Step 9: Creating a layout file for each item of RecyclerView

Navigate to the app > res > layout > Right-click on it > New > layout resource file and name it as course_rv_item and add below code to it. Comments are added in the code to get to know in more detail.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:elevation="8dp"
    app:cardCornerRadius="8dp">

    <LinearLayout
        android:id="@+id/idLLCourse"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:orientation="vertical">

        <!--text view for our course name-->
        <TextView
            android:id="@+id/idTVCourseName"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="8dp"
            android:text="Course Name"
            android:textColor="@color/black"
            android:textSize="15sp" />

        <!--text view for our course duration-->
        <TextView
            android:id="@+id/idTVCourseDuration"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="8dp"
            android:text="Course Duration"
            android:textColor="@color/black"
            android:textSize="15sp" />

        <!--text view for our course description-->
        <TextView
            android:id="@+id/idTVCourseDescription"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="8dp"
            android:text="Course Description"
            android:textColor="@color/black"
            android:textSize="15sp" />
    </LinearLayout>

</androidx.cardview.widget.CardView>
```

Step 10: Creating a RecyclerView Adapter class to set data for each item of RecyclerView

Navigate to the app > java > your app's package name > Right-click on it > New > Java class and name it as CourseRVAdapter and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.DiffUtil;
import androidx.recyclerview.widget.ListAdapter;
import androidx.recyclerview.widget.RecyclerView;

public class CourseRVAdapter extends ListAdapter<CourseModal,
CourseRVAdapter.ViewHolder> {

    // creating a variable for on item click listener.
    private OnItemClickListener listener;

    // creating a constructor class for our adapter class.
    CourseRVAdapter() {
        super(DIFF_CALLBACK);
    }

    // creating a call back for item of recycler view.
```

```
    private static final DiffUtil.ItemCallback<CourseModal> DIFF_CALLBACK
= new DiffUtil.ItemCallback<CourseModal>() {
        @Override
        public boolean areItemsTheSame(CourseModal oldItem, CourseModal
newItem) {
            return oldItem.getId() == newItem.getId();
        }

        @Override
        public boolean areContentsTheSame(CourseModal oldItem,
CourseModal newItem) {
            // below line is to check the course name, description and
course duration.
            return
oldItem.getCourseName().equals(newItem.getCourseName()) &&

oldItem.getCourseDescription().equals(newItem.getCourseDescription()) &&

oldItem.getCourseDuration().equals(newItem.getCourseDuration());
        }
    };

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        // below line is use to inflate our layout
        // file for each item of our recycler view.
        View item = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.course_rv_item, parent, false);
        return new ViewHolder(item);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int
position) {
        // below line of code is use to set data to
        // each item of our recycler view.
        CourseModal model = getCourseAt(position);
        holder.courseNameTV.setText(model.getCourseName());
        holder.courseDescTV.setText(model.getCourseDescription());
        holder.courseDurationTV.setText(model.getCourseDuration());
    }

    // creating a method to get course modal for a specific position.
    public CourseModal getCourseAt(int position) {
        return getItem(position);
    }

    public class ViewHolder extends RecyclerView.ViewHolder {
        // view holder class to create a variable for each view.
        TextView courseNameTV, courseDescTV, courseDurationTV;

        ViewHolder(@NonNull View itemView) {
            super(itemView);
            // initializing each view of our recycler view.
            courseNameTV = itemView.findViewById(R.id.idTVCourseName);
            courseDescTV =
itemView.findViewById(R.id.idTVCourseDescription);
            courseDurationTV =
itemView.findViewById(R.id.idTVCourseDuration);
```

```
            // adding on click listener for each item of recycler view.
            itemView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    // inside on click listener we are passing
                    // position to our item of recycler view.
                    int position = getAdapterPosition();
                    if (listener != null && position !=
RecyclerView.NO_POSITION) {
                        listener.onItemClick(getItem(position));
                    }
                }
            });
        }
    }

    public interface OnItemClickListener {
        void onItemClick(CourseModal model);
    }
    public void setOnItemClickListener(OnItemClickListener listener) {
        this.listener = listener;
    }
}
```

Step 11: Creating a new Activity for Adding and Updating our Course

Navigate to the app > java > your app's package name > Right-click on it > New > Empty Activity and name it as NewCourseActivity and go to XML part and add below code to it. Below is the code for the activity_new_course.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:orientation="vertical"
    tools:context=".NewCourseActivity">

    <!--edit text for our course name-->
    <EditText
        android:id="@+id/idEdtCourseName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Enter Course Name" />

    <!--edit text for our course description-->
    <EditText
        android:id="@+id/idEdtCourseDescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Enter Course Description" />

    <!--edit text for course description-->
    <EditText
```

```
        android:id="@+id/idEdtCourseDuration"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Course Duration" />

    <!--button for saving data to room database-->
    <Button
        android:id="@+id/idBtnSaveCourse"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:padding="5dp"
        android:text="Save your course"
        android:textAllCaps="false" />

</LinearLayout>
```

Step 12: Working with the NewCourseActivity.java file

Navigate to the app > java > your app's package name > NewCourseActivity.java file and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class NewCourseActivity extends AppCompatActivity {

    // creating a variables for our button and edittext.
    private EditText courseNameEdt, courseDescEdt, courseDurationEdt;
    private Button courseBtn;

    // creating a constant string variable for our
    // course name, description and duration.
    public static final String EXTRA_ID =
"com.gtappdevelopers.gfgroomdatabase.EXTRA_ID";
    public static final String EXTRA_COURSE_NAME =
"com.gtappdevelopers.gfgroomdatabase.EXTRA_COURSE_NAME";
    public static final String EXTRA_DESCRIPTION =
"com.gtappdevelopers.gfgroomdatabase.EXTRA_COURSE_DESCRIPTION";
    public static final String EXTRA_DURATION =
"com.gtappdevelopers.gfgroomdatabase.EXTRA_COURSE_DURATION";

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

        // initializing our variables for each view.
        courseNameEdt = findViewById(R.id.idEdtCourseName);
        courseDescEdt = findViewById(R.id.idEdtCourseDescription);
        courseDurationEdt = findViewById(R.id.idEdtCourseDuration);
```

```
        courseBtn = findViewById(R.id.idBtnSaveCourse);

        // below line is to get intent as we
        // are getting data via an intent.
        Intent intent = getIntent();
        if (intent.hasExtra(EXTRA_ID)) {
            // if we get id for our data then we are
            // setting values to our edit text fields.

courseNameEdt.setText(intent.getStringExtra(EXTRA_COURSE_NAME));

courseDescEdt.setText(intent.getStringExtra(EXTRA_DESCRIPTION));

courseDurationEdt.setText(intent.getStringExtra(EXTRA_DURATION));
        }
        // adding on click listener for our save button.
        courseBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // getting text value from edittext and validating if
                // the text fields are empty or not.
                String courseName = courseNameEdt.getText().toString();
                String courseDesc = courseDescEdt.getText().toString();
                String courseDuration =
courseDurationEdt.getText().toString();
                if (courseName.isEmpty() || courseDesc.isEmpty() ||
courseDuration.isEmpty()) {
                    Toast.makeText(NewCourseActivity.this, "Please enter
the valid course details.", Toast.LENGTH_SHORT).show();
                    return;
                }
                // calling a method to save our course.
                saveCourse(courseName, courseDesc, courseDuration);
            }
        });
    }

    private void saveCourse(String courseName, String courseDescription,
String courseDuration) {
        // inside this method we are passing
        // all the data via an intent.
        Intent data = new Intent();

        // in below line we are passing all our course detail.
        data.putExtra(EXTRA_COURSE_NAME, courseName);
        data.putExtra(EXTRA_DESCRIPTION, courseDescription);
        data.putExtra(EXTRA_DURATION, courseDuration);
        int id = getIntent().getIntExtra(EXTRA_ID, -1);
        if (id != -1) {
            // in below line we are passing our id.
            data.putExtra(EXTRA_ID, id);
        }

        // at last we are setting result as data.
        setResult(RESULT_OK, data);
        // displaying a toast message after adding the data
        Toast.makeText(this, "Course has been saved to Room Database. ",
Toast.LENGTH_SHORT).show();
    }
}
```

Step 13: Working with the MainActivity.java file

Navigate to the app > java > your app's package name > MainActivity.java file and add the below code to it. Comments are added inside the code to understand the code in more detail.

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;
import androidx.recyclerview.widget.ItemTouchHelper;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import
com.google.android.material.floatingactionbutton.FloatingActionButton;

import java.util.List;

public class MainActivity extends AppCompatActivity {

    // creating a variables for our recycler view.
    private RecyclerView coursesRV;
    private static final int ADD_COURSE_REQUEST = 1;
    private static final int EDIT_COURSE_REQUEST = 2;
    private ViewModal viewmodal;

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

        // initializing our variable for our recycler view and fab.
        coursesRV = findViewById(R.id.idRVCourses);
        FloatingActionButton fab = findViewById(R.id.idFABAdd);

        // adding on click listener for floating action button.
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // starting a new activity for adding a new course
                // and passing a constant value in it.
                Intent intent = new Intent(MainActivity.this,
NewCourseActivity.class);
                startActivityForResult(intent, ADD_COURSE_REQUEST);
            }
        });

        // setting layout manager to our adapter class.
        coursesRV.setLayoutManager(new LinearLayoutManager(this));
        coursesRV.setHasFixedSize(true);

        // initializing adapter for recycler view.
        final CourseRVAdapter adapter = new CourseRVAdapter();
```

```
        // setting adapter class for recycler view.
        coursesRV.setAdapter(adapter);

        // passing a data from view modal.
        viewmodal = ViewModelProviders.of(this).get(ViewModal.class);

        // below line is use to get all the courses from view modal.
        viewmodal.getAllCourses().observe(this, new
Observer<List<CourseModal>>() {
            @Override
            public void onChanged(List<CourseModal> models) {
                // when the data is changed in our models we are
                // adding that list to our adapter class.
                adapter.submitList(models);
            }
        });
        // below method is use to add swipe to delete method for item of
recycler view.
        new ItemTouchHelper(new ItemTouchHelper.SimpleCallback(0,
ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {
            @Override
            public boolean onMove(@NonNull RecyclerView recyclerView,
@NonNull RecyclerView.ViewHolder viewHolder, @NonNull
RecyclerView.ViewHolder target) {
                return false;
            }

            @Override
            public void onSwiped(@NonNull RecyclerView.ViewHolder
viewHolder, int direction) {
                // on recycler view item swiped then we are deleting the
item of our recycler view.

viewmodal.delete(adapter.getCourseAt(viewHolder.getAdapterPosition()));
                Toast.makeText(MainActivity.this, "Course deleted",
Toast.LENGTH_SHORT).show();
            }
        }).
                // below line is use to attach this to recycler view.
                        attachToRecyclerView(coursesRV);
        // below line is use to set item click listener for our item of
recycler view.
        adapter.setOnItemClickListener(new
CourseRVAdapter.OnItemClickListener() {
            @Override
            public void onItemClick(CourseModal model) {
                // after clicking on item of recycler view
                // we are opening a new activity and passing
                // a data to our activity.
                Intent intent = new Intent(MainActivity.this,
NewCourseActivity.class);
                intent.putExtra(NewCourseActivity.EXTRA_ID,
model.getId());
                intent.putExtra(NewCourseActivity.EXTRA_COURSE_NAME,
model.getCourseName());
                intent.putExtra(NewCourseActivity.EXTRA_DESCRIPTION,
model.getCourseDescription());
                intent.putExtra(NewCourseActivity.EXTRA_DURATION,
model.getCourseDuration());
```

```
                // below line is to start a new activity and
                // adding a edit course constant.
                startActivityForResult(intent, EDIT_COURSE_REQUEST);
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == ADD_COURSE_REQUEST && resultCode == RESULT_OK)
{
            String courseName =
data.getStringExtra(NewCourseActivity.EXTRA_COURSE_NAME);
            String courseDescription =
data.getStringExtra(NewCourseActivity.EXTRA_DESCRIPTION);
            String courseDuration =
data.getStringExtra(NewCourseActivity.EXTRA_DURATION);
            CourseModal model = new CourseModal(courseName,
courseDescription, courseDuration);
            viewmodal.insert(model);
            Toast.makeText(this, "Course saved",
Toast.LENGTH_SHORT).show();
        } else if (requestCode == EDIT_COURSE_REQUEST && resultCode ==
RESULT_OK) {
            int id = data.getIntExtra(NewCourseActivity.EXTRA_ID, -1);
            if (id == -1) {
                Toast.makeText(this, "Course can't be updated",
Toast.LENGTH_SHORT).show();
                return;
            }
            String courseName =
data.getStringExtra(NewCourseActivity.EXTRA_COURSE_NAME);
            String courseDesc =
data.getStringExtra(NewCourseActivity.EXTRA_DESCRIPTION);
            String courseDuration =
data.getStringExtra(NewCourseActivity.EXTRA_DURATION);
            CourseModal model = new CourseModal(courseName, courseDesc,
courseDuration);
            model.setId(id);
            viewmodal.update(model);
            Toast.makeText(this, "Course updated",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "Course not saved",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

Run the app to see the results

28 Storage

28.1 Shared Preferences

One of the most Interesting Data Storage options Android provides its users is Shared Preferences. Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage. Shared Preferences can be thought of as a dictionary or a key/value pair.

For example, you might have a key being “username” and for the value, you might store the user’s username. And then you could retrieve that by its key (here username). You can have a simple shared preference API that you can use to store preferences and pull them back as and when needed. Shared Preferences class provides APIs for reading, writing, and managing this data.

Example to Demonstrate the use of Shared Preferences in Android

Below is the small demo for Shared Preferences. In this particular demo, there are two EditTexts, which save and retain the data entered earlier in them. This type of feature can be seen in applications with forms. Using Shared Preferences, the user will not have to fill in details again and again. Invoke the following code inside the activity_main.xml file to implement the UI:

```
<?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"
    tools:context=".MainActivity"
    tools:ignore="HardcodedText">

    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:text="Shared Preferences Demo"
        android:textColor="@android:color/black"
        android:textSize="24sp" />

    <!--EditText to take the data from the user
        and save the data in SharedPreferences-->
    <EditText
        android:id="@+id/edit1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textview"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:hint="Enter your Name"
        android:padding="10dp" />

    <!--EditText to take the data from the user and
```

```
          save the data in SharedPreferences-->
    <EditText
        android:id="@+id/edit2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/edit1"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:hint="Enter your Age"
        android:padding="10dp"
        android:inputType="number" />

</RelativeLayout>
```

Working with the MainActivity.java file to handle the two of the EditText to save the data entered by the user inside the SharedPreferences. Below is the code for the MainActivity.java file. Comments are added inside the code to understand the code in more detail.

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

import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    private EditText name, age;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = findViewById(R.id.edit1);
        age = findViewById(R.id.edit2);
    }

    // Fetch the stored data in onResume()
    // Because this is what will be called
    // when the app opens again
    @Override
    protected void onResume() {
        super.onResume();

        // Fetching the stored data
        // from the SharedPreference
        SharedPreferences sh = getSharedPreferences("MySharedPref",
MODE_PRIVATE);

        String s1 = sh.getString("name", "");
        int a = sh.getInt("age", 0);

        // Setting the fetched data
        // in the EditTexts
        name.setText(s1);
        age.setText(String.valueOf(a));
    }
```

```
    // Store the data in the SharedPreference
    // in the onPause() method
    // When the user closes the application
    // onPause() will be called
    // and data will be stored
    @Override
    protected void onPause() {
        super.onPause();

        // Creating a shared pref object
        // with a file name "MySharedPref"
        // in private mode
        SharedPreferences sharedPreferences =
getSharedPreferences("MySharedPref", MODE_PRIVATE);
        SharedPreferences.Editor myEdit = sharedPreferences.edit();

        // write all the data entered by the user in SharedPreference and
apply
        myEdit.putString("name", name.getText().toString());
        myEdit.putInt("age", Integer.parseInt(age.getText().toString()));
        myEdit.apply();
    }
}
```

Run the app to see the results

29 JSON

29.1 JSON Parsing

JSON (JavaScript Object Notation) is a straightforward data exchange format to interchange the server's data, and it is a better alternative for XML. This is because JSON is a lightweight and structured language. Android supports all the JSON classes such as JSONStringer, JSONObject, JSONArray, and all other forms to parse the JSON data and fetch the required information by the program.

JSON's main advantage is that it is a language-independent, and the JSON object will contain data like a key/value pair. In general, JSON nodes will start with a square bracket ([) or with a curly bracket ({). The square and curly bracket's primary difference is that the square bracket ([) represents the beginning of a JSONArray node. Whereas, the curly bracket ({) represents a JSONObject.

So one needs to call the appropriate method to get the data. Sometimes JSON data start with [. We then need to use the getJSONArray() method to get the data. Similarly, if it starts with {, then we need to use the getJSONobject() method. The syntax of the JSON file is as following:

```
{
"Name": "FGTech",
"Estd": 2006,
"age": 25,
"address": {
    "buildingAddress": "49-3 Jalan Tasik Selatan 8",
    "city": "Kuala Lumpur",
    "state": "WP",
    "postalCode": "57000"
},
```

In this article, we are going to parse a JSON file in Android. Note that we are going to implement this project using the Kotlin language.

Step by Step Implementation

To parse a JSON file in Android, follow the following steps:

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Kotlin as the programming language.

Step 2: Working with the activity_main.xml file

Go to the activity_main.xml file which represents the UI of the application. Create a ListView as shown. Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <!--This listView will display the list items-->
    <ListView
        android:id="@+id/user_list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:dividerHeight="1dp" />

</LinearLayout>
```

Step 3: Create another layout resource file

Go to app > res > layout > right-click > New > Layout Resource File and create another layout list_row.xml to show the data in the ListView. Below is the code for the list_row.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="5dip">

    <!--TextView to display the name-->
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="17dp"
        android:textStyle="bold" />

    <!--TextView to display the designation-->
    <TextView
        android:id="@+id/designation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_marginTop="7dp"
        android:textColor="#343434"
        android:textSize="14dp" />

    <!--TextView to display the location-->
    <TextView
        android:id="@+id/location"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/designation"
```

```
        android:layout_alignBottom="@+id/designation"
        android:layout_alignParentRight="true"
        android:textColor="#343434"
        android:textSize="14dp" />
</RelativeLayout>
```

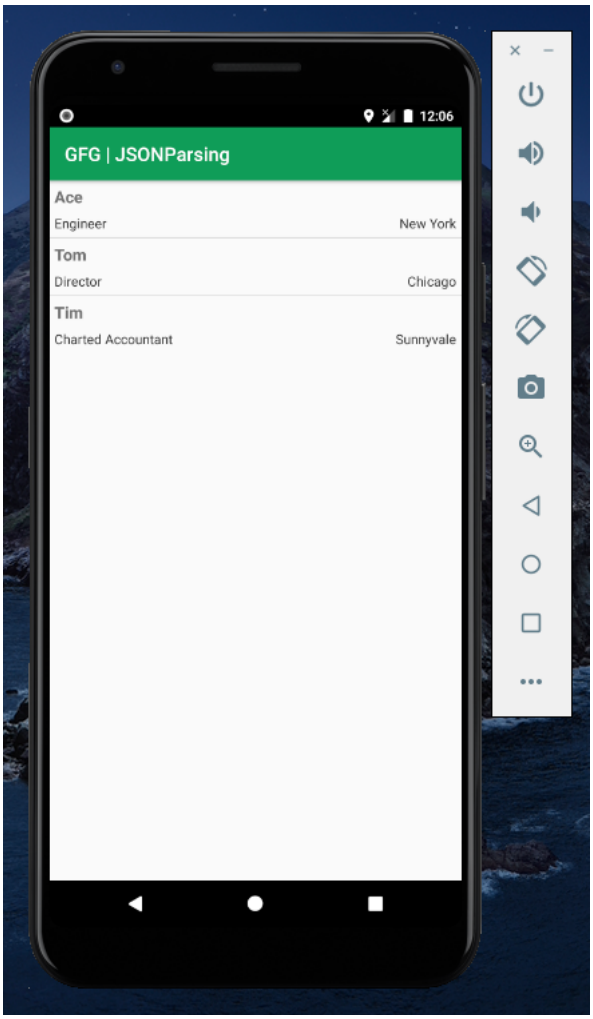
Step 4: Working with the MainActivity.kt file

Go to the MainActivity.kt file, and refer to the following code. Below is the code for the MainActivity.kt file. Comments are added inside the code to understand the code in more detail.

```
import android.os.Bundle
import android.util.Log
import android.widget.ListAdapter
import android.widget.ListView
import android.widget.SimpleAdapter
import androidx.appcompat.app.AppCompatActivity
import org.json.JSONException
import org.json.JSONObject
import java.util.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // private string declare in the latter section of the program
        val jsonStr = listData
        try {

            // Create a userList string hashmap arraylist
            val userList = ArrayList<HashMap<String, String?>>()

            // Declaring the listView from the layout file
            val lv = findViewById<ListView>(R.id.user_list)

            // Initializing the JSON object and extracting the
information
            val jObj = JSONObject(jsonStr)
            val jsonArry = jObj.getJSONArray("users")
            for (i in 0 until jsonArry.length()) {
                val user = HashMap<String, String?>()
                val obj = jsonArry.getJSONObject(i)
                user["name"] = obj.getString("name")
                user["designation"] = obj.getString("designation")
                user["location"] = obj.getString("location")
                userList.add(user)
            }

            // ListAdapter to broadcast the information to the list
elements
            val adapter: ListAdapter = SimpleAdapter(
                this, userList, R.layout.list_row,
                arrayOf("name", "designation", "location"), intArrayOf(
                    R.id.name,
                    R.id.designation, R.id.location
                )
            )
```

```
            lv.adapter = adapter
        } catch (ex: JSONException) {
            Log.e("JsonParser Example", "unexpected JSON exception", ex)
        }
    }


    // JSON object in the form of input stream
    private val listData: String
        get() = ("{ \"users\" :[" +

"{\"name\":\"Ace\",\"designation\":\"Engineer\",\"location\":\"New
York\"}" +

",{\"name\":\"Tom\",\"designation\":\"Director\",\"location\":\"Chicago\"
}" +
                ",{\"name\":\"Tim\",\"designation\":\"Charted
Accountant\",\"location\":\"Sunnyvale\"}] }")
}
```

Run the app to see the results



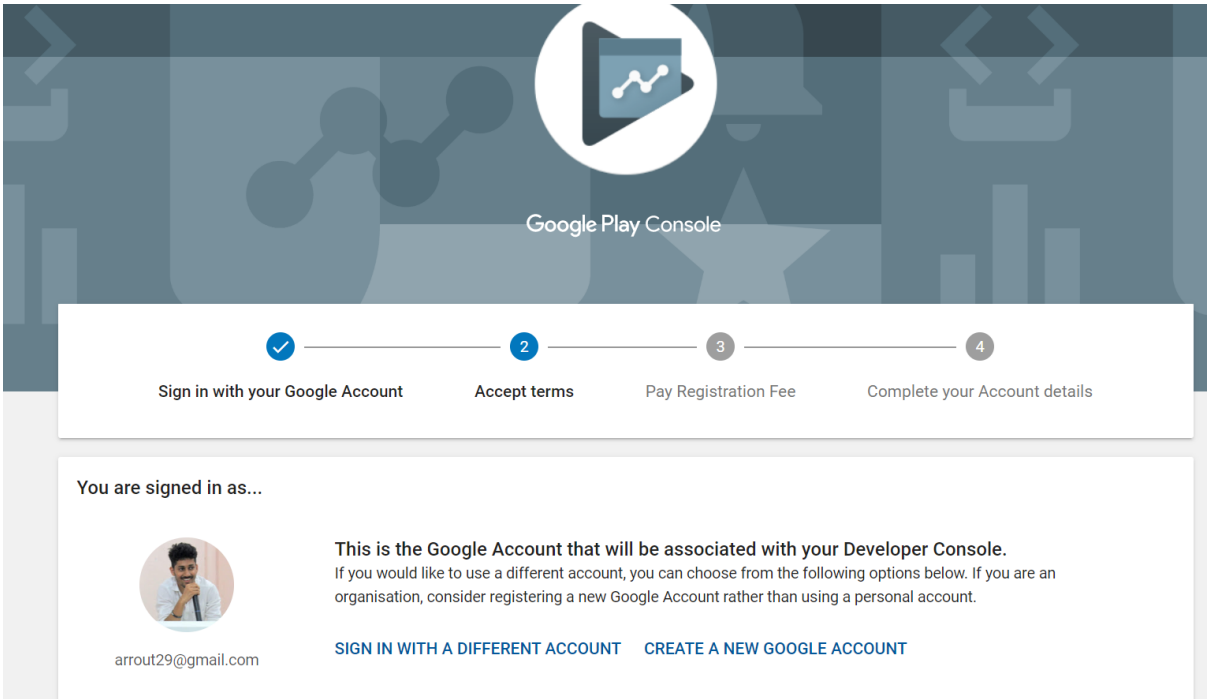
30 App Publish

30.1 How to publish on Google Play Store

Nowadays smartphones are among the most essential gadgets for users. Over 60% of people sleep with their phones by their side and check it first thing in the morning. Almost all businesses, including retail stores, have mobile apps to keep their audience engaged. Various applications like games, music player, camera, etc. are built for these smartphones for running on Android. Google Play store features quite 3.3 million apps.

Step 1: Make a Developer Account

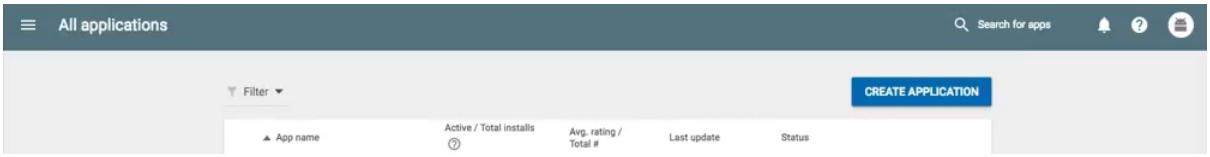
A developer account is must be needed to upload an app on the Google Play Store, and the process is very simple. Just go through Google Play Store and do as instructed.



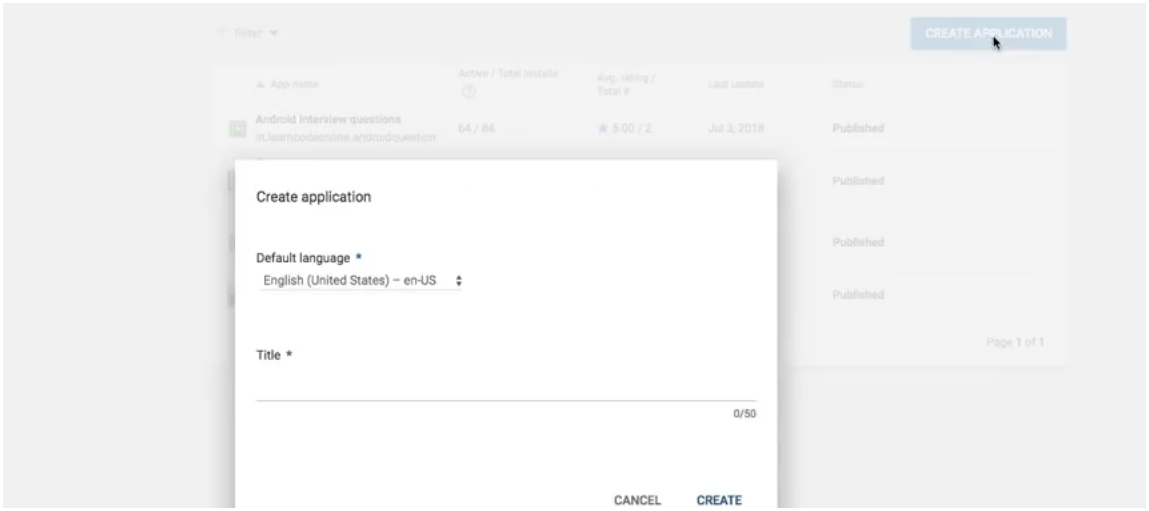
The account can be created in four simple steps:

1. Sign-In with Your Google Account
2. Accept Terms
3. Pay Registration Fee of $25.
4. Complete Your Account Details

Step 2: After you completed step 1 you will be redirected to this page where you have to click on the CREATE APPLICATION button.

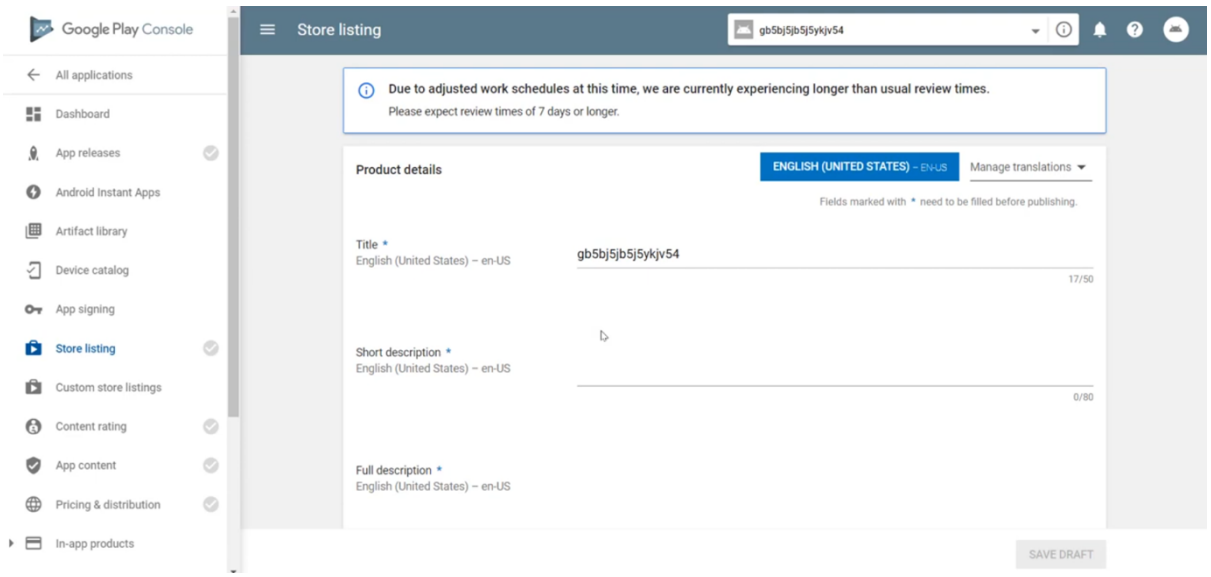


Once you click on it a pop up will be shown like this where you have to choose your Default language and Title of your app. Then click on the CREATE button.

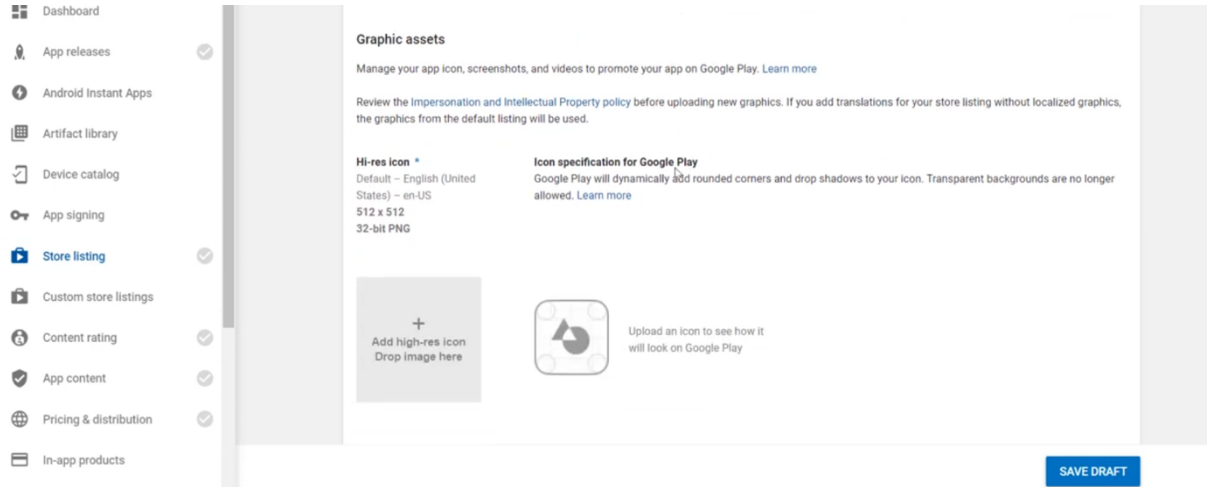


Step 3: Store listing

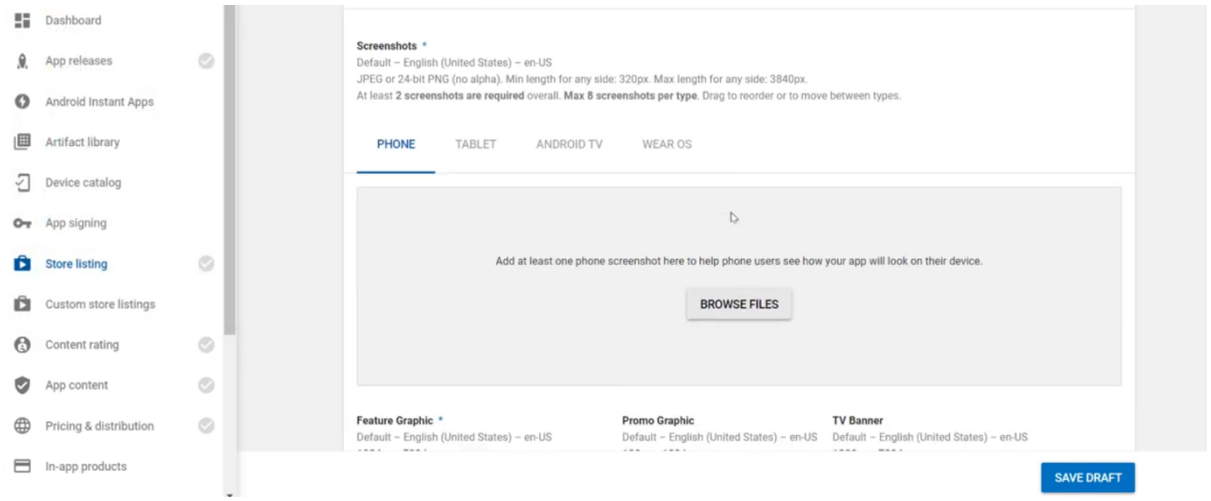
After you completed step 2 you will be redirected to this page where you have to provide the Short description and Full description of your App.



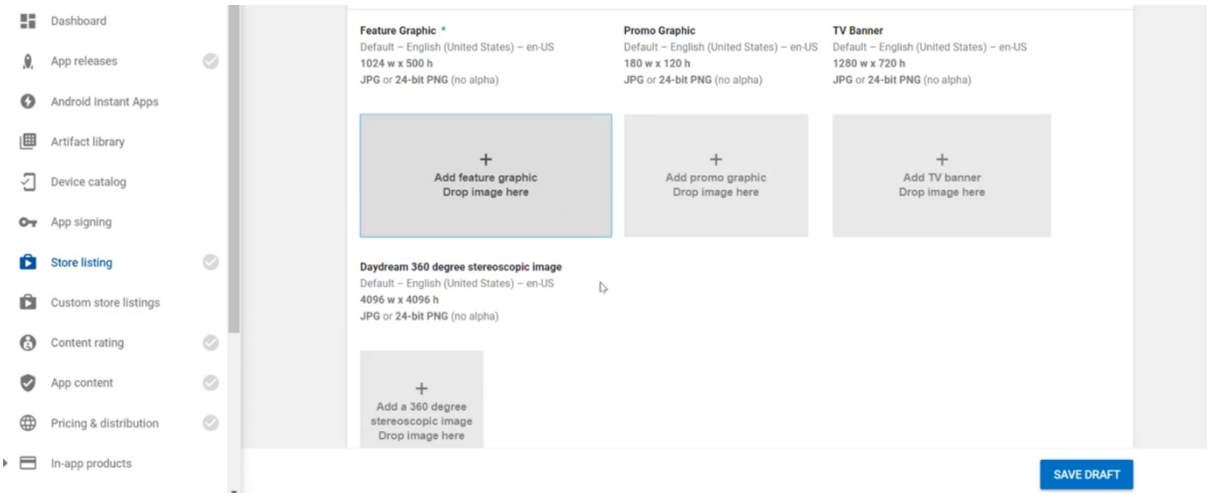
Then you scroll down the page and now you have to add the Hi-res icon of your app.



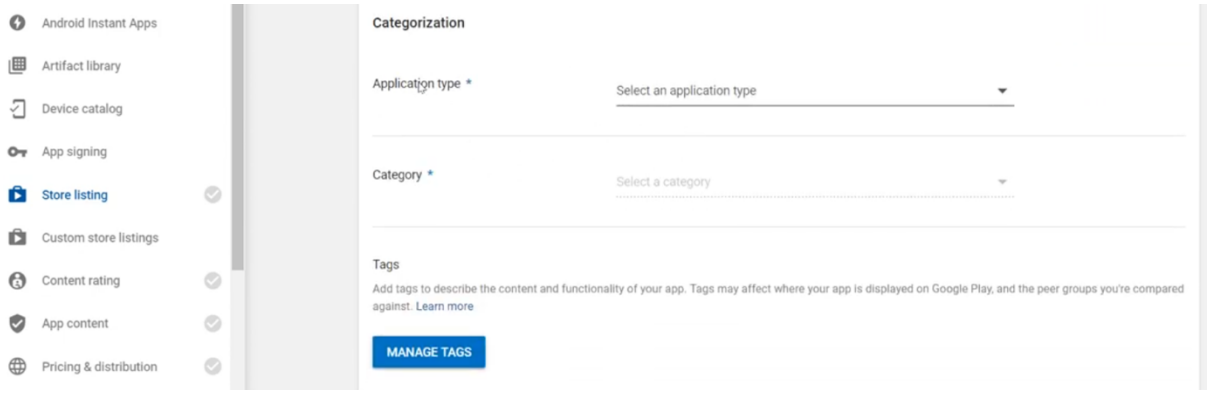
Then you have to provide the Screenshots of your app.



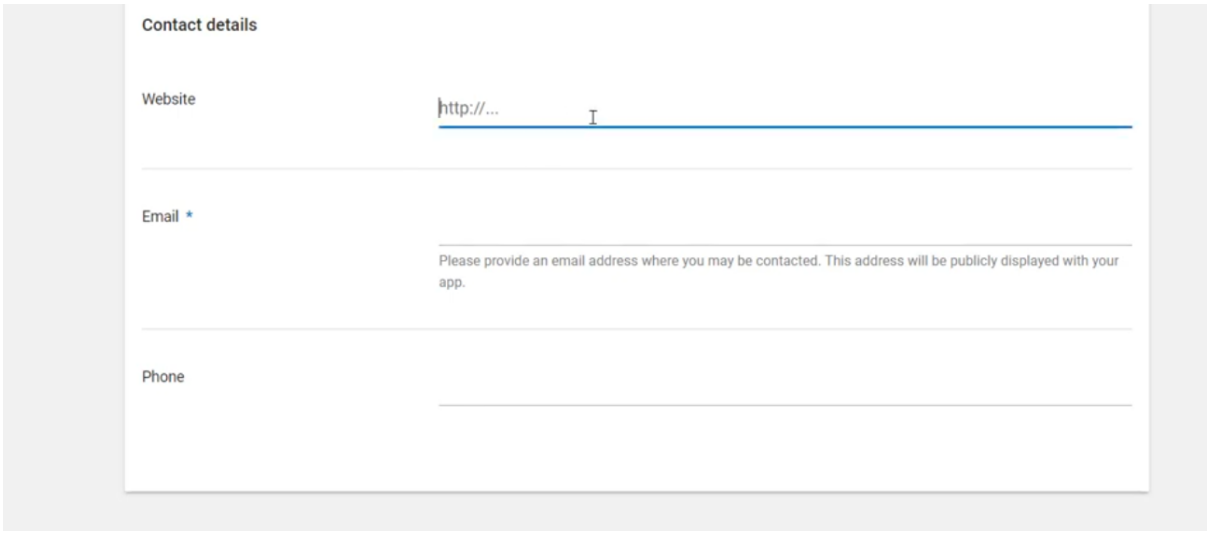
The next thing you have to provide is the Feature Graphic of your app. Note that this graphic is then used everywhere your app is featured on Google Play.



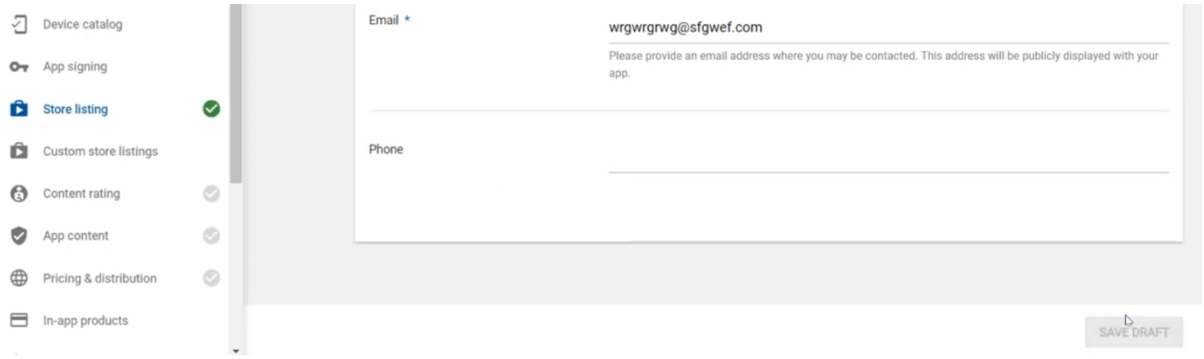
Then come to Categorization part where you have to provide your Application type and Category of your app.



Then come to Contact details part where you have to provide your Website(if any), email, and Phone of yours.

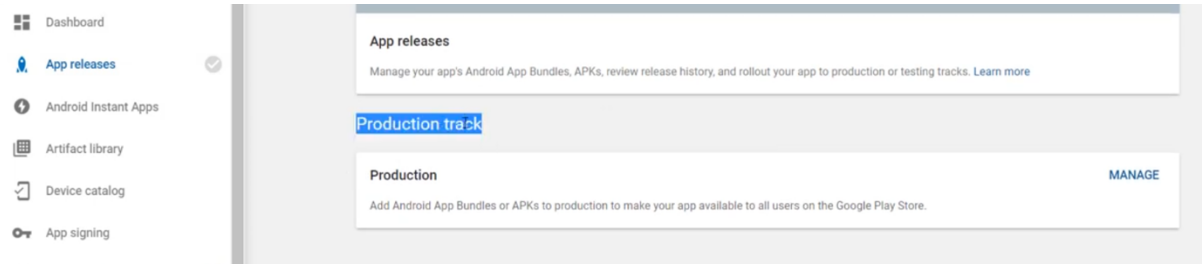


And finally when you click on SAVE DRAFT button you can see that Store listing tab is now become turned to green and you are done for Store listing.

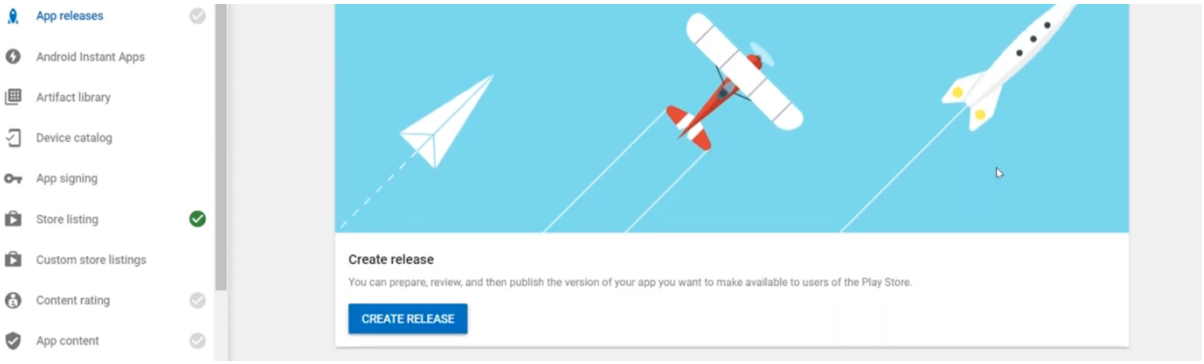


Step 4: App release

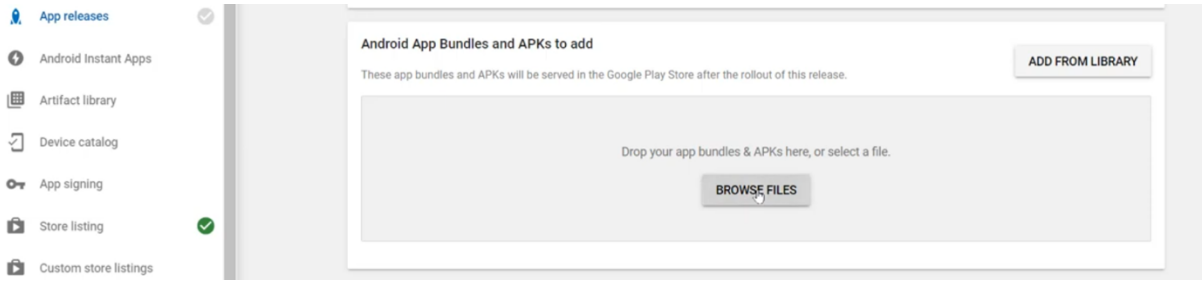
After completing step 3 go to App releases then scroll down to Production track and click on MANAGE button.



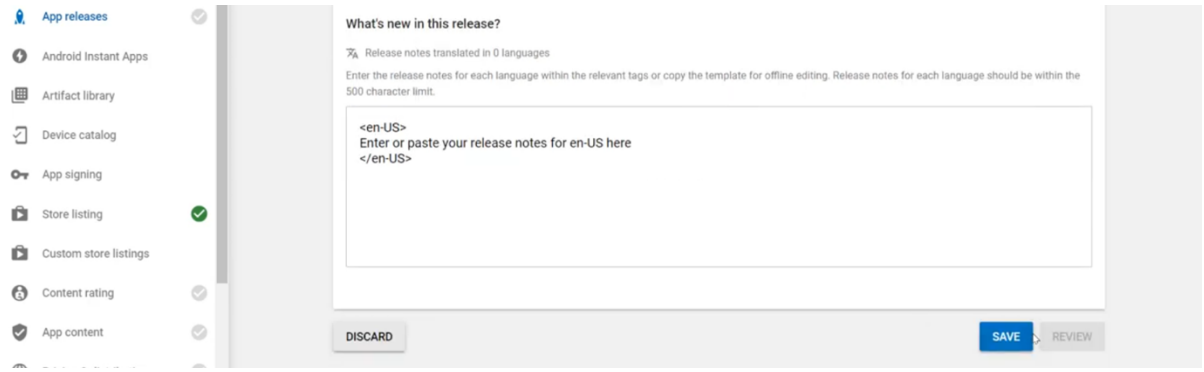
After redirecting to the next page click on the CREATE RELEASE button.



After that on the next page, you have to upload your APK file in Android App Bundles and APKs to add section.

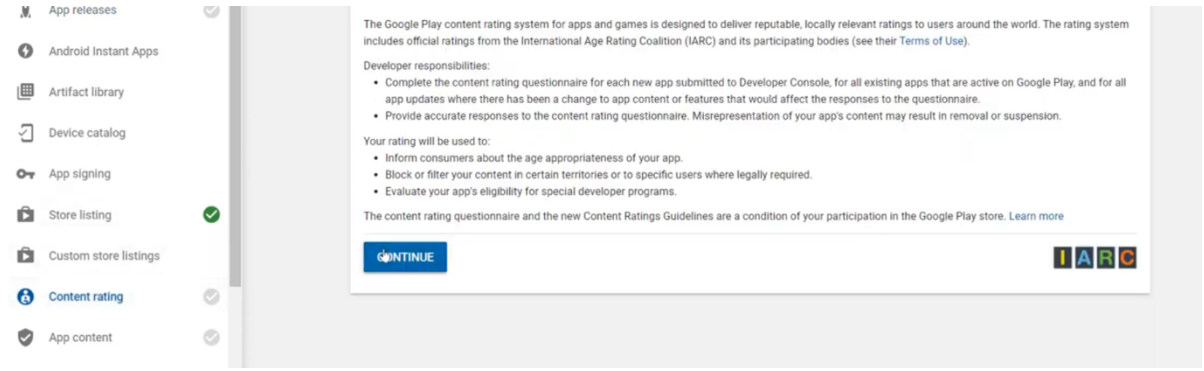


After that simply click on the SAVE button.

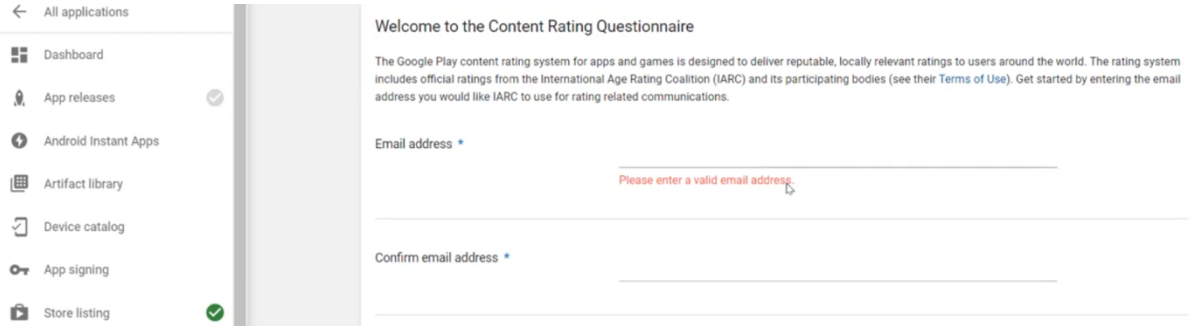


Step 5: Content rating

Now after completing step 4 go to Content rating and click on CONTINUE button.

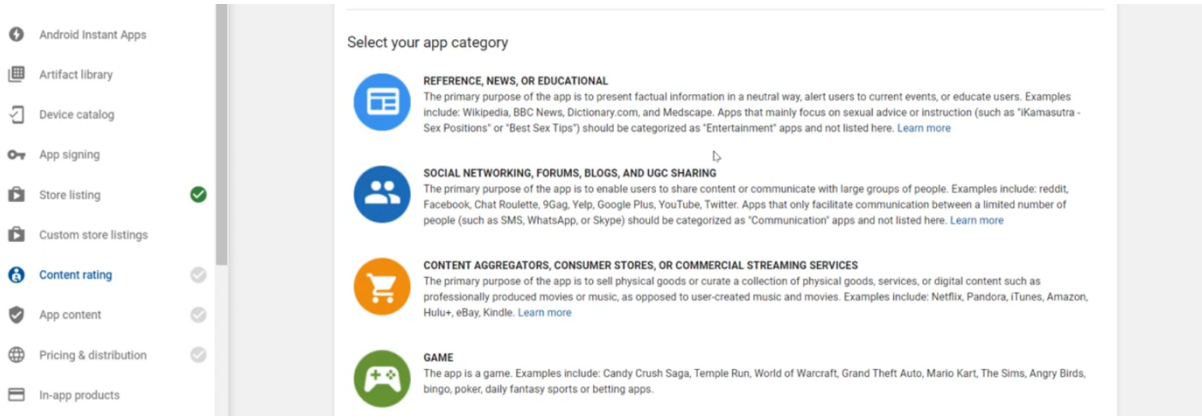


After that fill your email address as well as confirm the email address.

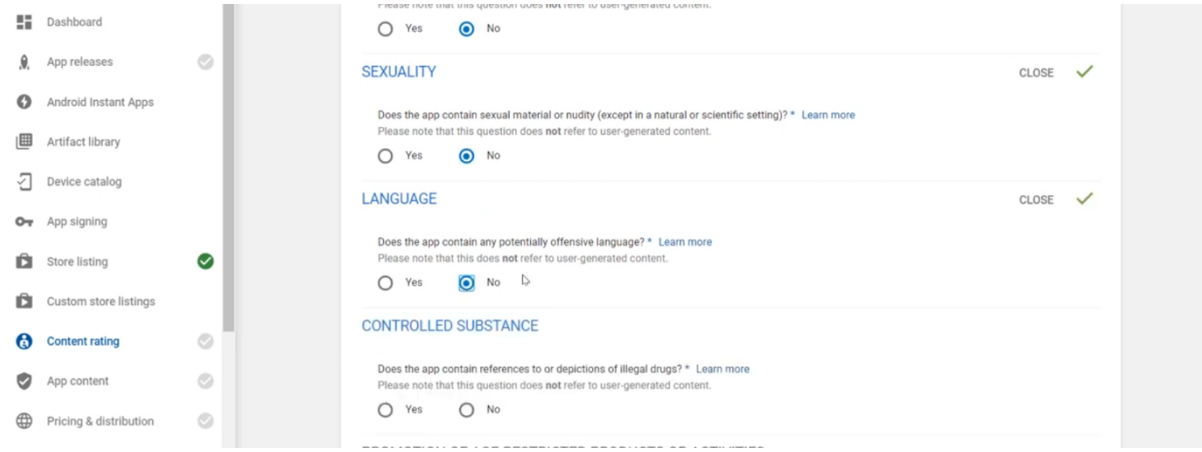


And then Select your app category.

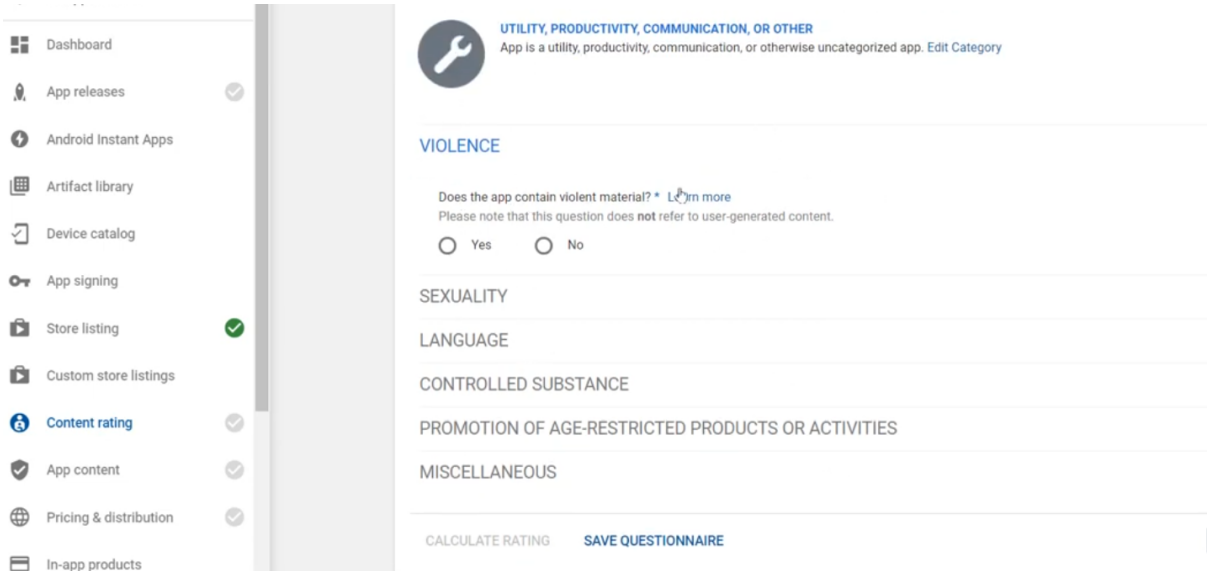
Android Development FUNdamentals



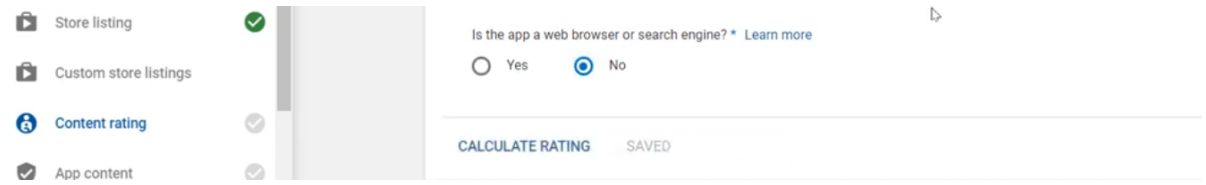
After selecting your app category make sure that you read all of these and answer them correctly.



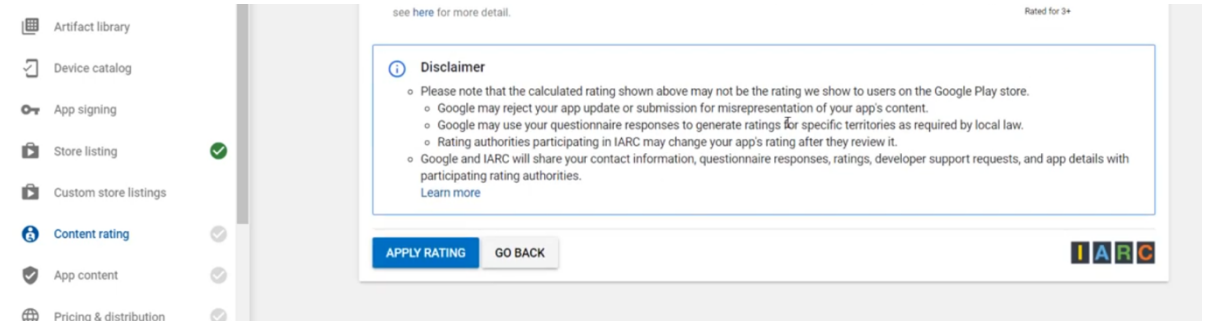
And after answering them correctly don't forget to click on SAVE QUESTIONNAIRE button.



Once you saved all those things then click on CALCULATE RATING button.

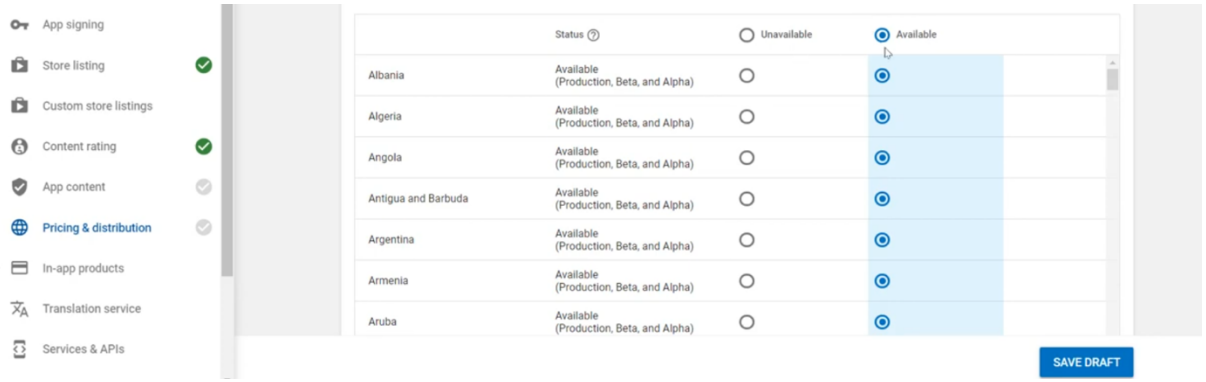


When you redirected to another page scroll down and click on APPLY RATING button. And you are done for Content rating section. Don't forget to notice that Content rating section is now become turned to green.

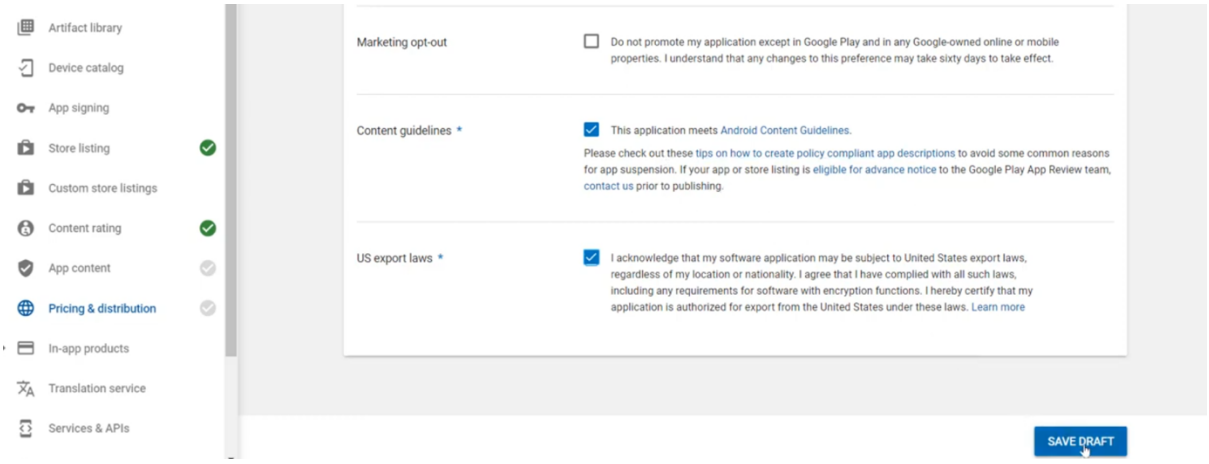


Step 6: Pricing & distribution

Then go to the Pricing & distribution section. Then select the country in which you want to available your app.

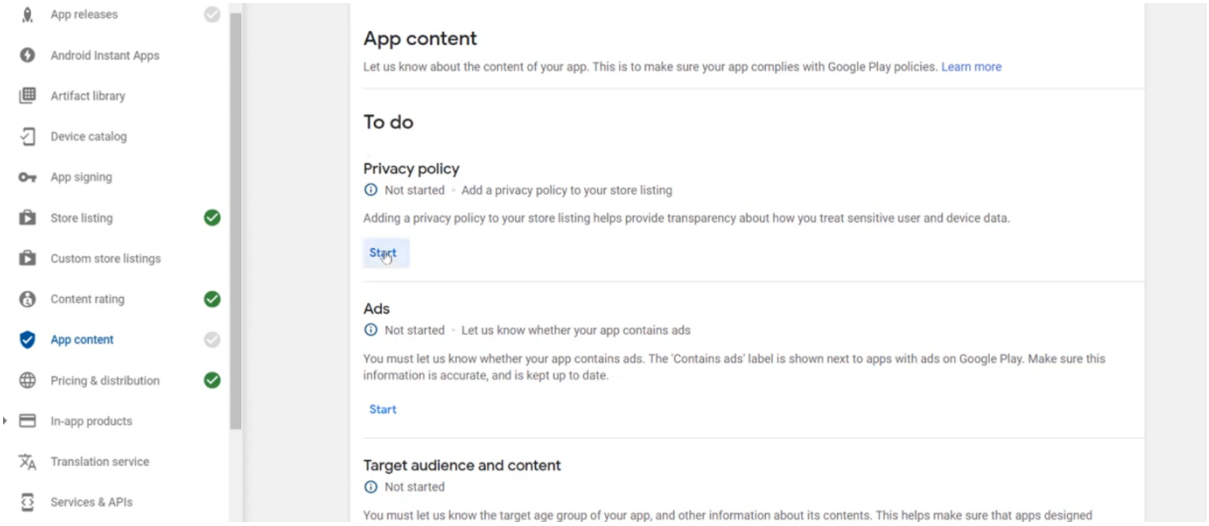


Then go down and down and check out the Content guidelines and US export laws section by marking them tick mark. And click on the SAVE DRAFT button. Don't forget to notice that Pricing & distribution section is now become turned to green tick.

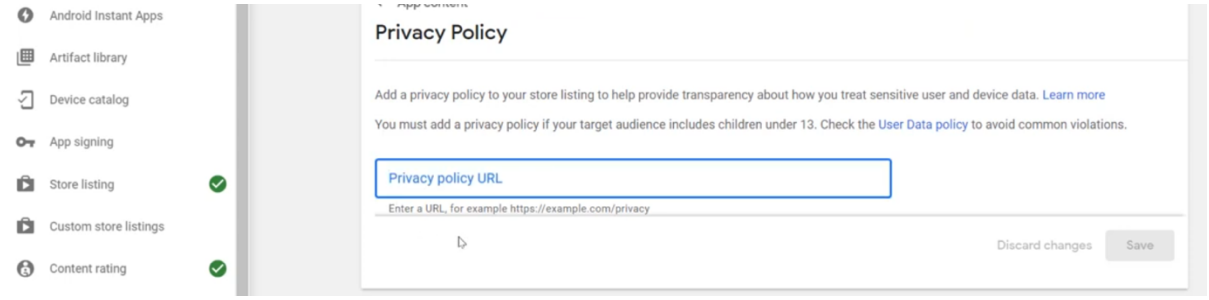


Step 7: App content

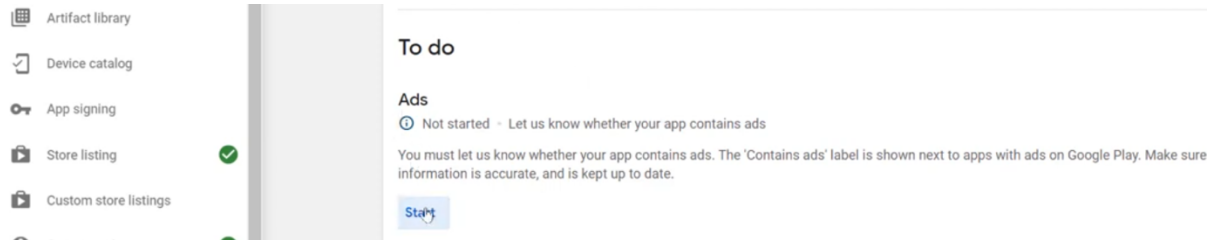
Then come to the App content section. And in the Privacy policy section click on the Start button.



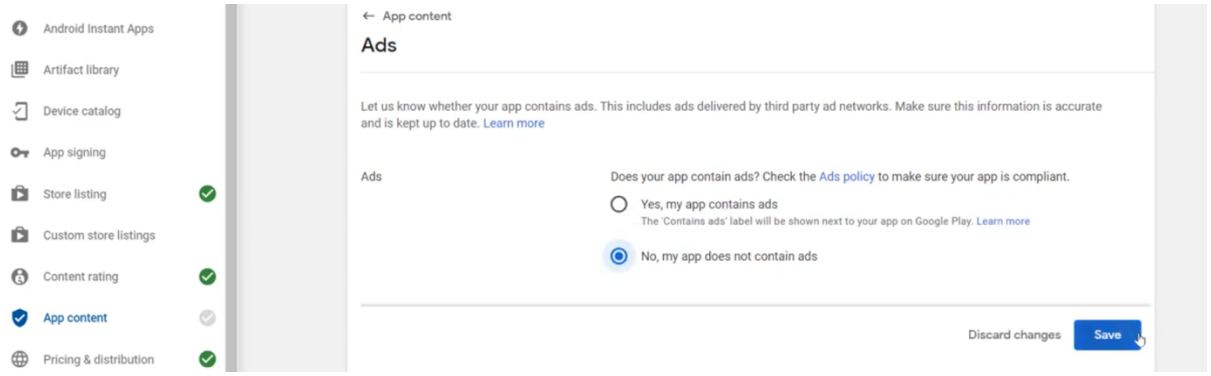
And then provide a valid Privacy policy URL. Note that google will check this.



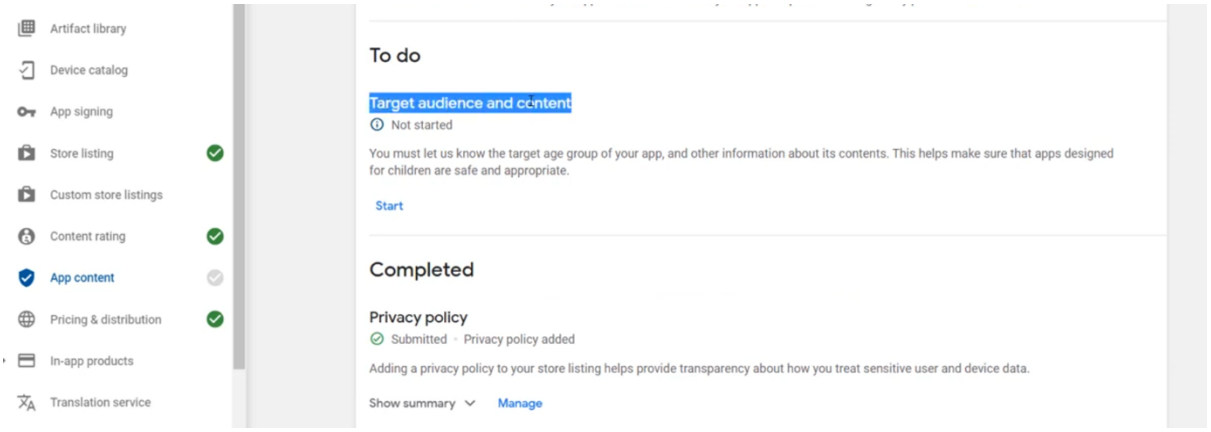
Then go back and continue further steps by clicking start button in Ads section.



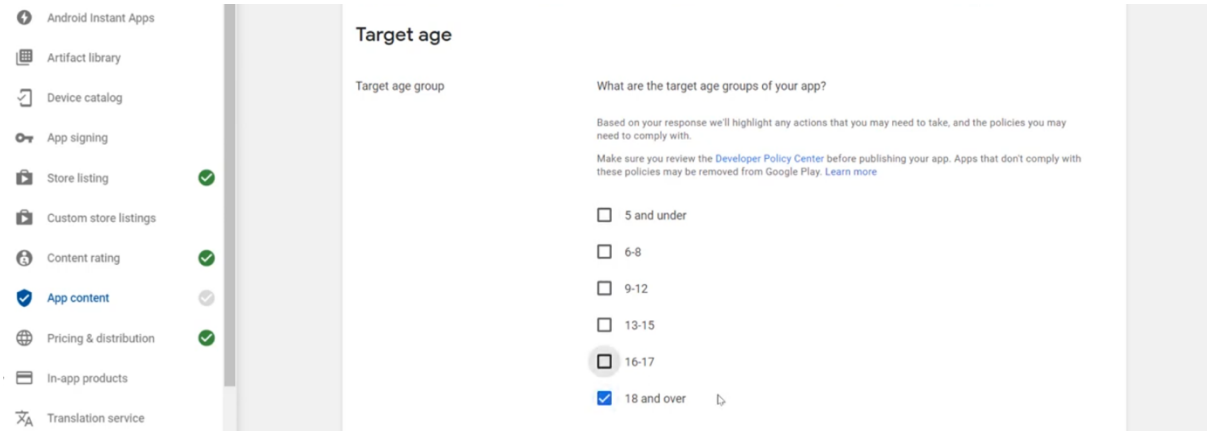
Then select does your app contain ads or not? And click on SAVE button.



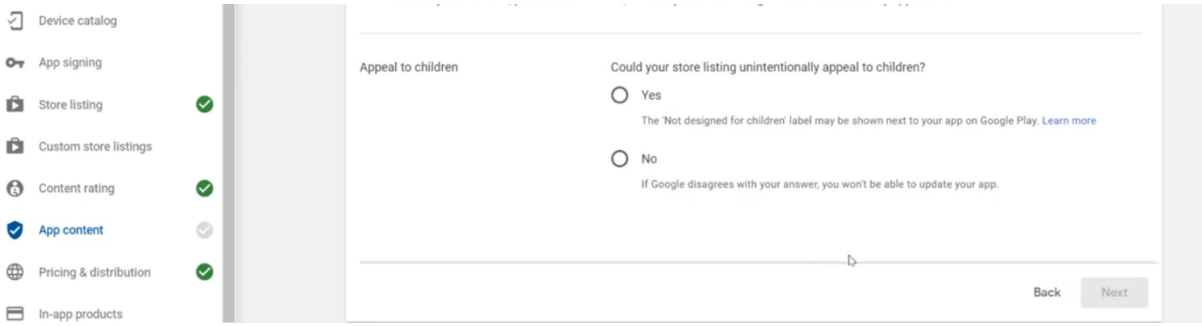
Then again go back and continue further steps by clicking start button in Target audience and content section.



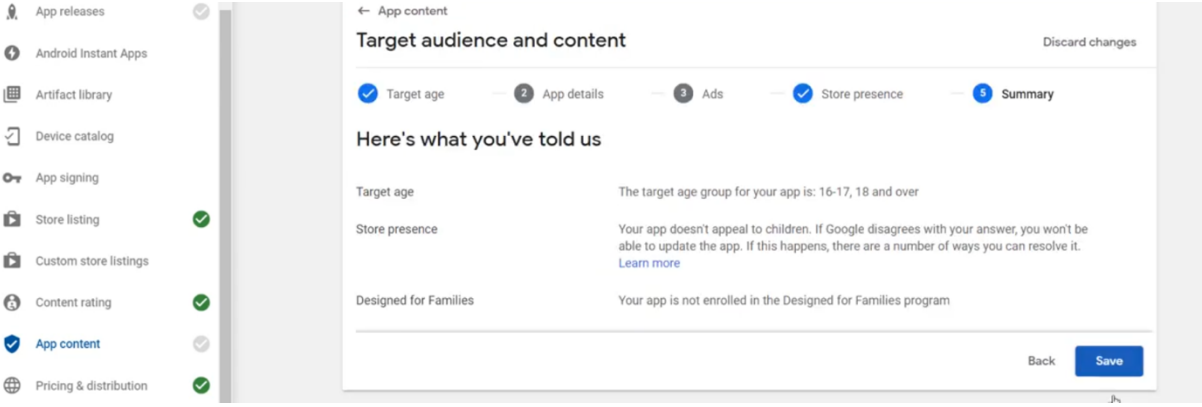
In the next page select the Target age group and scroll down and click on the Next button.



Then check the Appeal to children section. And click on the Next button.

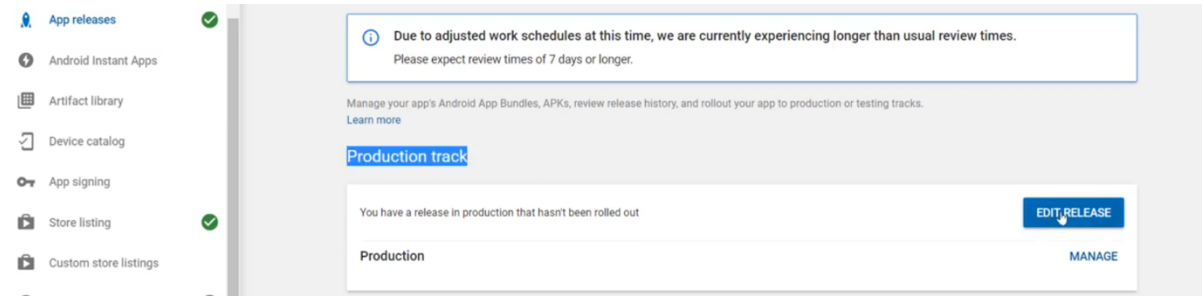


On the next page click on the Save button and you are done for App content section.

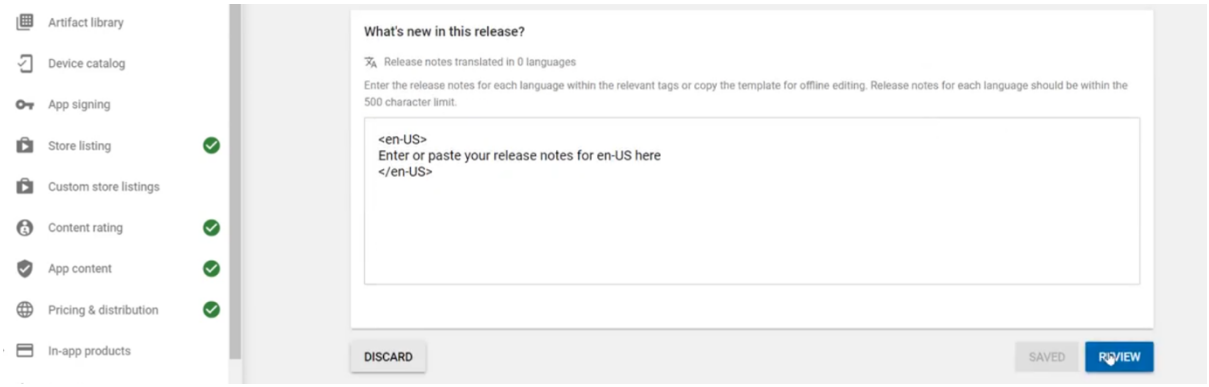


Step 8: App releases

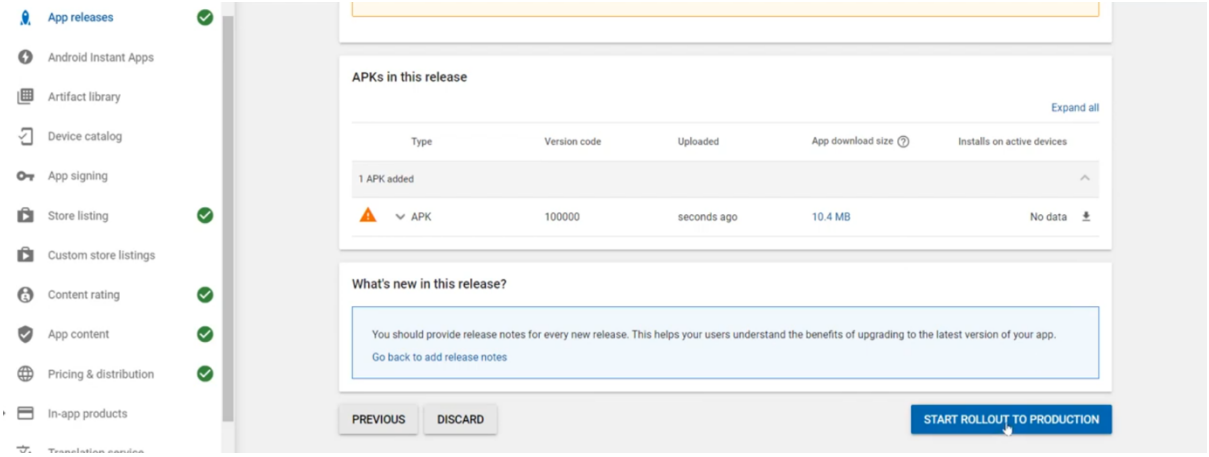
Again go back to the App releases section. And in the Production track click on the EDIT RELEASE button.



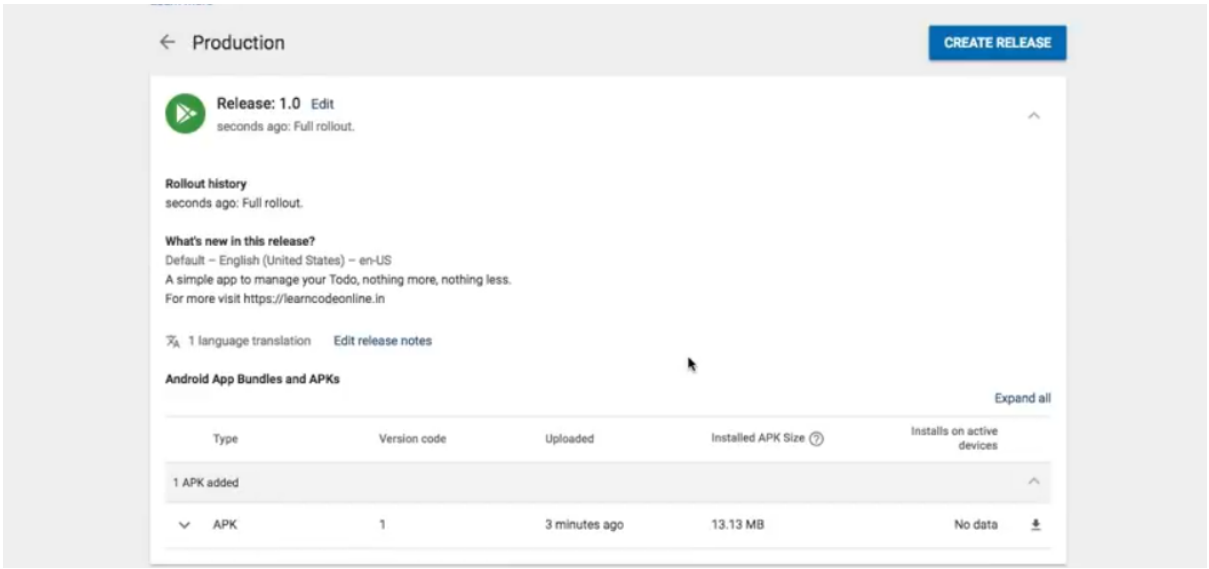
Then on the next page go down and down and click on the REVIEW button.



And finally, on the next page click on the START ROLLOUT TO PRODUCTION button to send your app to review. And you are finally done.



After usually 4 to 5 days they will review your app and let you know to either approve or reject your app.



The End

Conclusion

This book has been created and regularly updated to reflect the current changes in mobile apps development for the Android platform. Should there be any issues relating to the tutorials, feel free to send me an email at farizgaskin@gmail.com