BCA's

mobile application development

Android studio

"make changes in something established, especially by introducing new methods, ideas, or products."

@P V V Durgaprasad

ADITYA DEGREE COLLEGE FOR WOMEN

Kakinada

SEMESTER-IV

COURSE 11: MOBILE APPLICATION DEVELOPMENT USING ANDROID

UNIT-I

Introduction to Android: - Overview, History, Features of Android, The Android Platform, Understanding the Android Software Stack – Android Application Architecture –The Android Application Life Cycle – The Activity Life Cycle, Creating Android Activity -Views- Layout Android SDK, Android Installation, Building you First Android application, Understanding

Anatomy of Android Application, Android Manifest file.

UNIT-II

Android Application Design Essentials: Anatomy of an Android applications, Android terminologies, Creating User Interfaces with basic views- Application Context, Activities, Services, Intents, linking activities with Intents,, Receiving and Broadcasting Intents, Android Manifest File and its common settings, Using Intent Filter, Permissions.

UNIT-III

Android User Interface Design Essentials: User Interface Screen elements, Designing User Interfaces with Layouts, Drawing and Working with Animation. Layouts, Recycler View, List View, Grid View and Web view

Input Controls: Buttons, Checkboxes, Radio Buttons, Toggle Buttons, Spinners, Input Events, Menus, Toast, Dialogs, Styles and Themes, Creating lists, and Custom lists

UNIT-IV

Testing Android applications: Publishing Android application, Using Android preferences, Managing Application resources in a hierarchy, working with different types of resources.

UNIT-V

Using Common Android APIs: Internal Storage, External Storage , SQLite Databases , Managing data using Sqlite, Sharing Data between Applications with Content Providers, Using Android Networking APIs, Using Android Web APIs, JSON Parsing, Using Android Telephony APIs, Deploying Android Application to the World. Google maps, Using GPS to find current location, Sensors, bluetooth/Wi-Fi Connectivity.

UNIT - I

❖ Introduction to Android Application and History

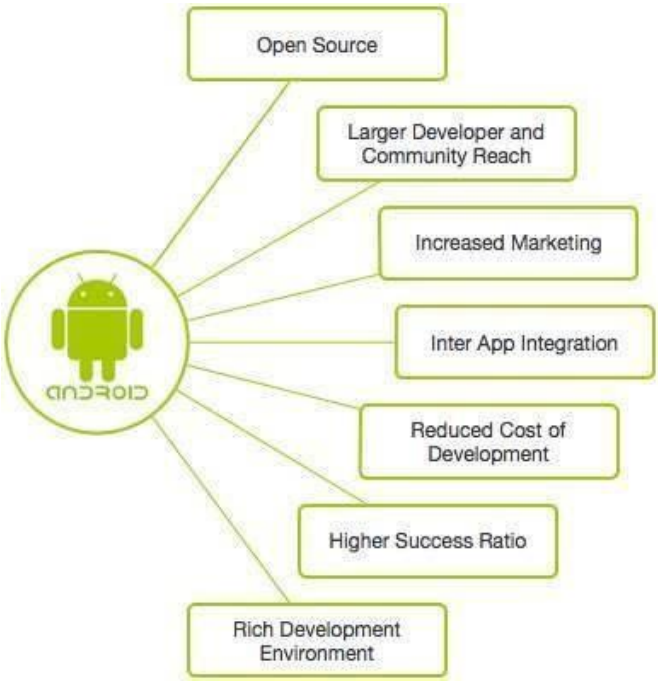
Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.



History of Android

The history of Android is a fascinating journey that spans several decades, characterized by innovation, collaboration, and evolution. From its humble beginnings, when it was competing with Nokia's mobile OS Symbian, the Windows Phone OS, and Blackberry OS, to its dominant position in the mobile operating system market, Android has significantly influenced how we interact with technology.

Here's an in-depth look at the history of Android:

1. Early origins and development (2003-2007)

The story of Android began in 2003 when Andy Rubin, Rich Miner, Nick Sears, and Chris White founded Android Inc. in Palo Alto, California. Their initial goal was to develop an advanced operating system for digital cameras. However, recognizing the potential of their project, they shifted their focus to creating an operating system for mobile devices.

2. Acquisition by Google (2005)

In 2005, Google, led by then-CEO Eric Schmidt, acquired Android Inc., laying the groundwork for what would become one of the most significant developments in the mobile industry. Google's acquisition of Android signaled its entry into the rapidly growing smartphone market and set the stage for developing a new mobile operating system.

3. Open Handset Alliance and the launch of Android (2007)

On November 5, 2007, the Open Handset Alliance (OHA) was unveiled. It comprised several prominent technology companies, including Google, HTC, Samsung, Motorola, and others. The OHA aimed to develop open standards for mobile devices and promote innovation in the mobile industry. Shortly after, on November 5, 2007, Google announced the first beta version of the Android operating system.

4. Android 1.0 and the first Android device (2008)

The first commercial version of Android, Android 1.0, was released on September 23, 2008. The HTC Dream, also known as T-Mobile G1, was the first smartphone to run on the Android operating system. The HTC Dream featured a touchscreen interface, a physical keyboard, and access to Google services such as Gmail, Maps, and YouTube.

Since its initial release, Android has undergone significant evolution with regular updates and new versions introduced to the market, as we will discuss later. The developer preview of Android 15 has been launched in 2024.

5. Growth and dominance in the mobile market

Over the years, Android has experienced tremendous growth, rapidly becoming the world's most popular mobile operating system. According to Statcounter, as of January 2022, Android holds over 72% of the global mobile operating system market share, far surpassing its competitors.

❖ Key Features of Android

Android is an open-source operating system, based on the Linux kernel and used in mobile devices like smartphones, tablets, etc. Further, it was developed for smartwatches and Android TV. Each of them has a specialized interface. Android has been one of the best-selling OS for smartphones. Android OS was developed by Android Inc. which Google bought in 2005.

Android offers a wide array of functionalities catering to both users and developers. Some of the key features of Android include:

- Open-source platform: Android is built on an open-source Linux kernel, allowing developers to access the source code, modify it, and contribute to its development. This openness fosters innovation and collaboration within the Android ecosystem.
- Customizable user interface: Android provides users with the ability to customize their device's user interface, including wallpapers, themes, widgets, and launchers. Users can personalize their devices to suit their preferences and style. This feature sets it apart from its closest competitor, iOS.
- Multitasking: Android supports multitasking, allowing users to run multiple apps simultaneously, switch between them seamlessly, and perform various tasks simultaneously. Users can also use split-screen mode to view two apps side by side.
- Google Play Store: Android users can access the Google Play Store, which offers a vast catalog of apps, games, movies, music, books, and more. The Play Store provides users a centralized platform to discover, download, and install content for their devices.

- Google Assistant: Android devices come with Google Assistant, a virtual assistant powered by artificial intelligence. Google Assistant can perform various tasks, answer questions, provide recommendations, and control smart home devices using voice commands.
- Security features: Android incorporates various security features to protect users' data and privacy. These features include app sandboxing, secure boot, encrypted file systems, Google Play Protect, and regular security updates from device manufacturers.
- Accessibility: Android includes a wide range of accessibility features to accommodate users with disabilities or special needs. These features include screen readers, magnification gestures, color inversion, text-to-speech, and more.
- Google Services integration: Android devices seamlessly integrate with Google services such as Gmail, Google Maps, Google Drive, Google Photos, and others. This integration provides users access to a suite of productivity tools, communication services, and cloud storage options.
- Development tools and support: Android provides developers with comprehensive development tools, including Android Studio, the official integrated development environment (IDE) for Android app development. Developers can also access extensive documentation, APIs, libraries, and resources to build high-quality apps for Android devices.

❖ Different Apps in android

Some Categories of the Android Applications:

1. E-Commerce Apps: E-commerce apps are an example of a B2B model. It helps to people to sell and borrow different items and it saves time and money. In e-commerce applications, we can do trading of commercial goods on online marketplaces. To buy specific items and goods, you simply need to make electronic transactions like UPI, Phonepe, etc. through your smartphone or computer. Flipkart, Amazon, OLX, and, Quiker are examples of e-commerce applications.
2. Educational Apps: Educational apps are too much used to improve knowledge and peoples get productivity. Apps for education can make people more interactive, more engaged, and perform better. Keeping teaching methods good is integral to getting students engaged in their studies and learning apps are a

fantastic way of achieving this. For example, Google Classroom, SoloLearn, edX, Duolingo, etc.

3. Social Media Apps: Social media apps give the opportunity to the peoples connect and communicate together. These apps are mainly used for sharing purposes and making fun. Many peoples use social media applications for influence, marketing/ business, entrepreneurship, etc. Instagram, Facebook, WhatsApp, YouTube, LinkedIn, etc. are examples of social media applications.
4. Productivity Apps: Productivity apps typically organize and complete complex tasks for you, anything from sending an email to figuring out a tip. The easy-to-use Google Drive app gives users access to all of the files saved to the cloud-based storage service across multiple devices. Productivity applications arise in many different forms and they often take a different approach to improving your workflow. For example, Hive, Todoist, Google Docs, etc.
5. Entertainment Apps: Entertainment apps are widely used apps worldwide. It contains OTT platforms and novels and other content. These platforms entertain people and give them much more knowledge about different things. Everyone is watching OTT platforms and those are trending these days, and their development is also in demand all over the world. Hotstar, Netflix, and Amazon prime video are the best examples of this entertainments applications.

❖ Android Versions

Android has evolved through various versions since its inception. Each version brings new features, enhancements, and optimizations to the platform. Here is a list of the major Android versions released to date:

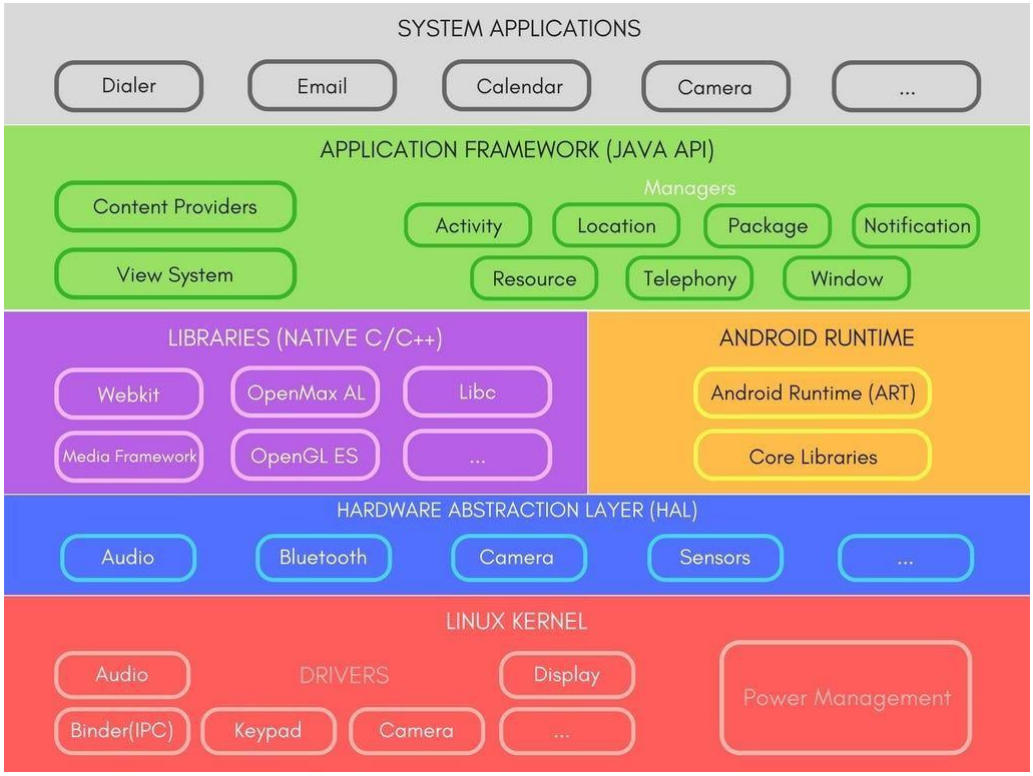
- Android 1.0 (Astro): The initial version of Android was released on September 23, 2008. It introduced basic functionalities such as web browsing, camera support, and access to Google services like Gmail and Google Maps.
- Android 1.1 (Bender): Released on February 9, 2009, Android 1.1 included minor updates and bug fixes to improve system stability and performance.
- Android 1.5 (Cupcake): Introduced on April 27, 2009, Android 1.5 brought significant improvements, such as an on-screen keyboard, support for third-party widgets, and video recording capabilities.
- Android 1.6 (Donut): Released on September 15, 2009, Android 1.6 featured updates to the user interface, improved search functionality, and support for CDMA networks.

- Android 2.0/2.1 (Eclair): Android 2.0 and 2.1, known collectively as Eclair, were released on October 26, 2009. Eclair introduced features such as multiple account support, Bluetooth 2.1, and an updated web browser.
- Android 2.2 (Froyo): Released on May 20, 2010, Android 2.2 (Froyo) introduced significant performance improvements, support for Adobe Flash Player, and the ability to install apps on external storage.
- Android 2.3 (Gingerbread): Introduced on December 6, 2010, Android 2.3 (Gingerbread) focused on refining the user interface, improving gaming performance, and adding support for near field communication (NFC).
- Android 3.0/3.1/3.2 (Honeycomb): Android 3.0 (Honeycomb) was released on February 22, 2011, and was specifically designed for tablets. It featured a redesigned user interface, support for multicore processors, and improved multitasking capabilities.
- Android 4.0 (Ice Cream Sandwich): Released on October 18, 2011, Android 4.0 (Ice Cream Sandwich) merged the tablet and smartphone versions of Android. It introduced features such as a new user interface, enhanced multitasking, and support for facial recognition.
- Android 4.1/4.2/4.3 (Jelly Bean): Android 4.1 (Jelly Bean) was released on July 9, 2012, followed by subsequent updates 4.2 and 4.3. Jelly Bean introduced features such as improved performance, enhanced notifications, and support for multiple user accounts on tablets.
- Android 4.4 (KitKat): Released on October 31, 2013, Android 4.4 (KitKat) focused on optimizing the operating system for low-end devices. It introduced features such as improved memory management, a new dialer app, and support for cloud printing.
- Android 5.0/5.1 (Lollipop): Android 5.0 (Lollipop) was released on November 12, 2014, followed by updates to 5.1. Lollipop introduced the Material Design language, improved performance, enhanced security features, and support for 64-bit processors.
- Android 6.0 (Marshmallow): Released on October 5, 2015, Android 6.0 (Marshmallow) introduced features such as app permissions, Google Now on Tap, and a new battery-saving feature called Doze.
- Android 7.0/7.1 (Nougat): Android 7.0 (Nougat) was released on August 22, 2016, followed by updates to 7.1. Nougat introduced features such as split-screen multitasking, enhanced notifications, and support for Daydream VR.

- Android 8.0/8.1 (Oreo): Android 8.0 (Oreo) was released on August 21, 2017, followed by updates to 8.1. Oreo introduced features, such as picture-in-picture mode, notification dots, and improved battery life, through background app limitations.
- Android 9 (Pie): Released on August 6, 2018, Android 9 (Pie) introduced features such as gesture-based navigation, adaptive battery, and digital wellbeing tools to help users monitor their smartphone usage.
- Android 10: Released on September 3, 2019, Android 10 introduced features such as a system-wide dark mode, improved privacy controls, and support for foldable smartphones.
- Android 11: Released on September 8, 2020, Android 11 focused on enhancing communication, privacy, and control with features like chat bubbles, one-time permissions, and improved media controls.
- Android 12: Released on October 4, 2021, Android 12 introduced a major visual overhaul with Material You design language, enhanced privacy features, and performance improvements.
- Android 13: Android 13 focused on user privacy with features like a photo picker and notification permission settings. Building on Android 12's tablet optimizations, Android 13 enhances system UI, multitasking, and compatibility modes.
- Android 14: Released on October 4, 2023, Android 14 enhances accessibility with features like 200% font scaling and customizable lock screens. Additionally, it introduces support for lossless audio formats and an improved magnifier for low-vision users.
- Android 15: It is the upcoming iteration of the Android operating system, slated for release in early 2025. It introduces advanced encryption features for secure data storage and transmission, among other features.

❖ Android Architecture

The Android architecture is a layered structure that defines the components and interactions within the Android operating system.



1. Linux kernel layer

At the core of the Android architecture lies the Linux kernel, which provides essential hardware abstraction, memory management, process management, security, and device drivers. The Linux kernel serves as the foundation upon which the Android operating system is built, offering low-level functionalities that interact directly with the underlying hardware components of the device.

Key features of the Linux kernel layer in the Android architecture include:

- Hardware abstraction: The Linux kernel abstracts hardware functionalities, allowing the upper layers of the Android stack to interact with hardware components through standardized interfaces.
- Process management: The Linux kernel manages processes and threads, allocating system resources such as CPU time, memory, and input/output operations.
- Memory management: The Linux kernel handles memory allocation, virtual memory management, and memory protection to ensure efficient utilization of system resources.
- Security mechanisms: The Linux kernel enforces security policies through access control mechanisms, permissions, and secure execution environments.

- Device drivers: The Linux kernel provides device drivers to facilitate communication between the operating system and hardware peripherals, such as display drivers, camera drivers, input/output drivers, and network drivers.

2. Hardware abstraction layer (HAL)

Above the Linux kernel layer resides the HAL, which abstracts hardware-specific functionalities and provides standardized interfaces for device drivers and hardware components. The HAL enables device manufacturers to develop drivers for specific hardware configurations while ensuring compatibility with the Android framework.

Key components of the hardware abstraction layer include:

- HAL modules: HAL modules encapsulate hardware-specific functionalities, such as camera, audio, display, sensors, and input devices, into standardized interfaces accessible to higher-level software layers.
- Interface definitions: The HAL defines standardized interfaces, known as Hardware Abstraction Interfaces (HAIs), which specify the methods and parameters for interacting with hardware components.
- Vendor-specific implementations: Device manufacturers like Samsung or Huawei provide vendor-specific implementations of HAL modules tailored to their hardware configurations, ensuring seamless integration with the Android framework.

3. Native libraries layer

The native libraries layer consists of libraries written in C and C++ programming languages that provide core system functionalities and support for native code execution within Android applications. These libraries augment the capabilities of the Java-based Android framework and enable developers to access low-level system resources and hardware features.

Key native libraries in the Android architecture include:

- libc: The C standard library provides fundamental programming utilities and functions for memory management, string manipulation, input/output operations, and system calls.
- libm: The Math library contains mathematical functions and operations, including arithmetic, trigonometric, exponential, and logarithmic functions.
- libz: The Zlib library implements data compression and decompression algorithms, facilitating file compression and decompression operations within Android applications.

- libjpeg/libpng: These libraries provide support for image processing and manipulation, including JPEG and PNG image format decoding and encoding.
- OpenGL/OpenGL ES: The OpenGL (for desktop) and OpenGL ES (for embedded systems) libraries enable hardware-accelerated graphics rendering and 3D rendering within Android applications.

4. Android runtime layer

The Android runtime (ART) layer is responsible for executing and managing Android applications bytecode compiled from Java or Kotlin source code. ART employs ahead-of-time (AOT) compilation to convert bytecode into native machine code, enhancing runtime performance and reducing memory overheads.

Key components of the Android Runtime layer include:

- ART compiler: The ART compiler translates bytecode into native machine code during the installation or upgrade of Android applications, improving runtime performance and efficiency.
- Dalvik virtual machine (legacy): In earlier versions of Android, the Dalvik virtual machine executed bytecode in the form of Dalvik Executable (DEX) files. Dalvik employed just-in-time (JIT) compilation to convert bytecode into native code at runtime.

5. Java API framework layer

The Java application programming interface (API) framework layer comprises a comprehensive set of libraries, APIs, and runtime environments that facilitate the development of Android applications using Java or Kotlin programming languages.

The Java API Framework exposes high-level functionalities and system services to developers, enabling them to create rich, interactive, and feature-rich applications. Key components of the Java API Framework layer include:

- Android software development kit (SDK): The Android SDK provides a collection of development tools, libraries, sample code, and documentation for building Android applications. It includes the Android Debug Bridge (ADB), Android Studio IDE, Android Emulator, and various command-line utilities.
- Core libraries: The core libraries contain essential classes and packages for application development, including data structures, utilities, I/O operations, networking, graphics, and user interface components.
- Android framework APIs: The Android framework APIs expose system-level functionalities and services, such as activity management, resource handling,

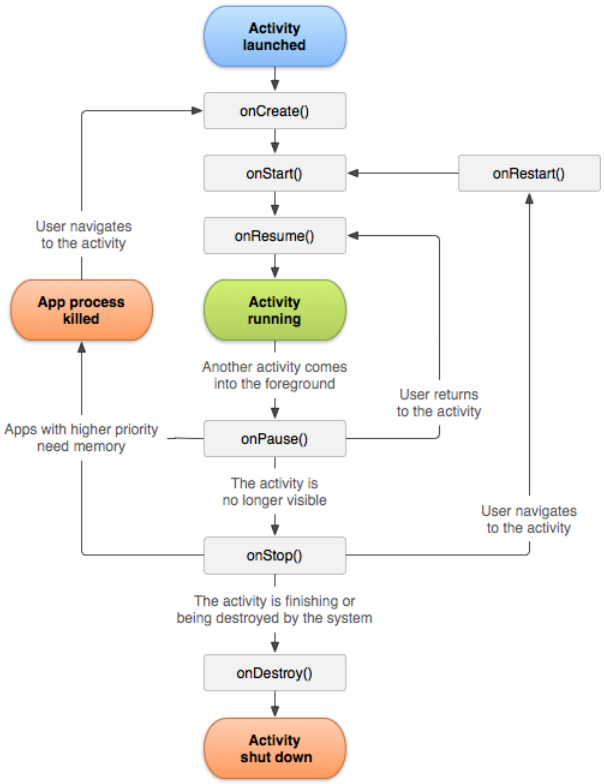
content providers, intents, services, and user interface components (views, layouts, widgets).

6. Application layer

At the topmost layer of the Android architecture is the application layer, which consists of user-installed applications, system applications, and system services running on the Android platform. This layer encompasses a diverse range of applications, including productivity tools, multimedia players, games, social networking apps, communication apps, and more.

❖ Android Activity Lifecycle

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.
Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.



Android activities have a specific lifecycle that they go through as the user interacts with them. Understanding the activity lifecycle is crucial for developing robust and responsive Android apps. In this post, we will go over the various states that an activity can be in and the corresponding methods that are called during those states.

The activity lifecycle is composed of the following states:

1. onCreate(): This method is called when the activity is first created. It is responsible for initializing the activity's UI, such as inflating the layout and finding the views.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //initialize views and other UI related tasks
}
```

2. onStart(): This method is called when the activity becomes visible to the user. It is a good place to start animations and other visual changes.

```
@Override
protected void onStart() {
    super.onStart();
    //start animations and other visual changes
}
```

3. onResume(): This method is called when the activity becomes the foreground activity. It isthe state in which the user can interact with the activity. This is where you should register any listeners or start any services that need to be running while the activity is in the foreground.

```
@Override
protected void onResume() {
    super.onResume();
```

```
    //register listeners and start services
}
```

4. onPause(): This method is called when the activity is no longer the foreground activity. It is a good place to unregister listeners, save any data that needs to be saved, and stop any services that don't need to be running in the background.

```
@Override
protected void onPause() {
    super.onPause();
    //unregister listeners, save data and stop services
}
```

5. onStop(): This method is called when the activity is no longer visible to the user. It is a good place to stop animations and other visual changes.

```
@Override
protected void onStop() {
    super.onStop();
    //stop animations and other visual changes
}
```

6. onDestroy(): This method is called when the activity is about to be destroyed. It is a good place to release any resources and clean up any remaining data.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    //release resources and clean up data
}
```

It's important to note that an activity can also be in a "stopped" state if it is temporarily obscured by another activity, but it will still retain its state and not be destroyed. In this case, the onStop() method will be called, but not onDestroy(). The onRestart() method will be called when the activity becomes visible again.

```
@Override
protected void onRestart() {
    super.onRestart();
    //refresh data or perform any other necessary tasks
}
```

In conclusion, the activity lifecycle is an important aspect of Android development and understanding the various states and methods that are called during those states can help you create more robust and responsive apps. It's important to remember that the activity can be in different states depending on the actions of the user and it's crucial to handle the data and resources accordingly.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast toast = Toast.makeText(getApplicationContext(), "onCreate
Called", Toast.LENGTH_LONG).show();
    }

    protected void onStart() {
        super.onStart();
        Toast toast = Toast.makeText(getApplicationContext(), "onStart
Called", Toast.LENGTH_LONG).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast toast = Toast.makeText(getApplicationContext(), "onRestart
Called", Toast.LENGTH_LONG).show();
    }
```

```
    protected void onPause() {
        super.onPause();
        Toast toast = Toast.makeText(getApplicationContext(), "onPause
Called", Toast.LENGTH_LONG).show();
    }

    protected void onResume() {
        super.onResume();
        Toast toast = Toast.makeText(getApplicationContext(), "onResume
Called", Toast.LENGTH_LONG).show();
    }

    protected void onStop() {
        super.onStop();
        Toast toast = Toast.makeText(getApplicationContext(), "onStop Called",
Toast.LENGTH_LONG).show();
    }

    protected void onDestroy() {
        super.onDestroy();
        Toast toast = Toast.makeText(getApplicationContext(), "onDestroy
Called", Toast.LENGTH_LONG).show();
    }
}
```

❖ Install Android Studio on Windows

Android Studio System Requirements for Windows

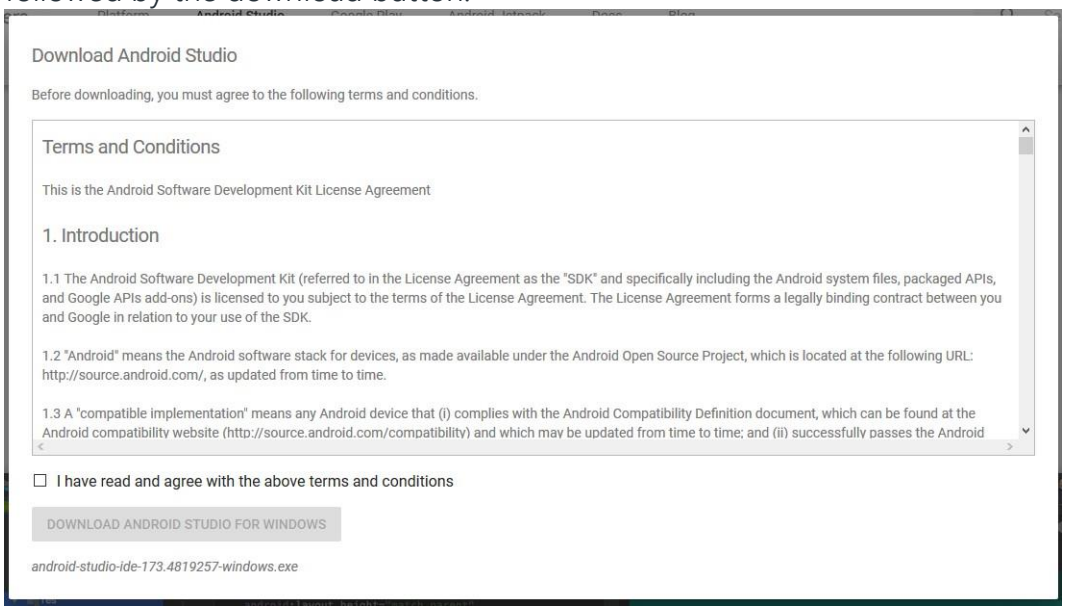
- Microsoft Windows 7/8/10 (32-bit or 64-bit)
- 4 GB RAM minimum, 8 GB RAM recommended (plus 1 GB for the Android Emulator)
- 2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE plus 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Steps to Install Android Studio on Windows

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

Step 2: Click on the Download Android Studio Button.

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

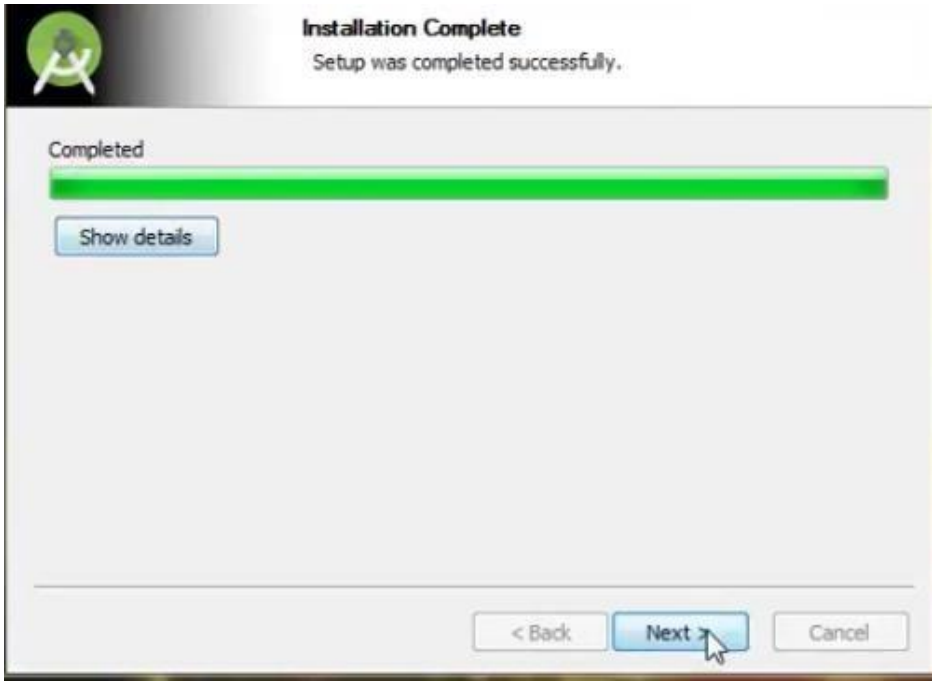


Click on the Save file button in the appeared prompt box and the file will start downloading.

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

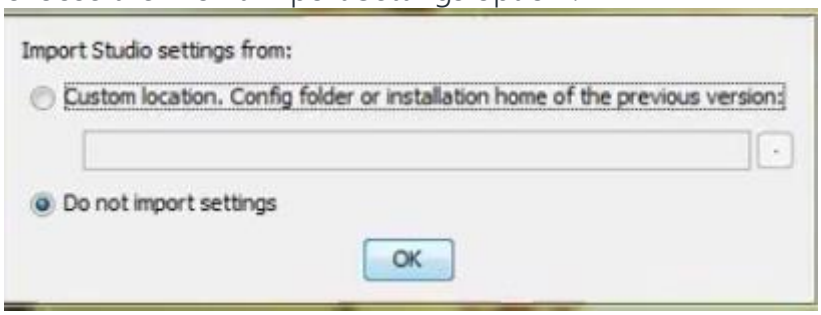
Click on next. In the next prompt, it'll ask for a path for installation. Choose a path and hit next.

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



Click on next.

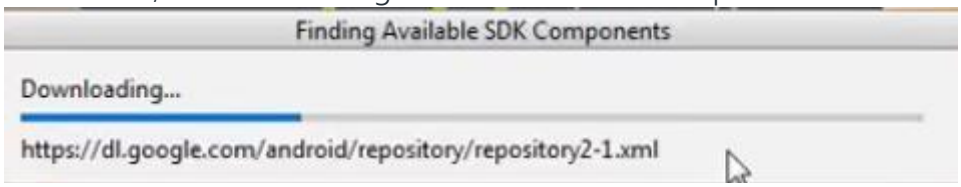
Step 5: Once " Finish " is clicked, it will ask whether the previous settings need to be imported [if the android studio had been installed earlier], or not. It is better to choose the 'Don't import Settings option'.



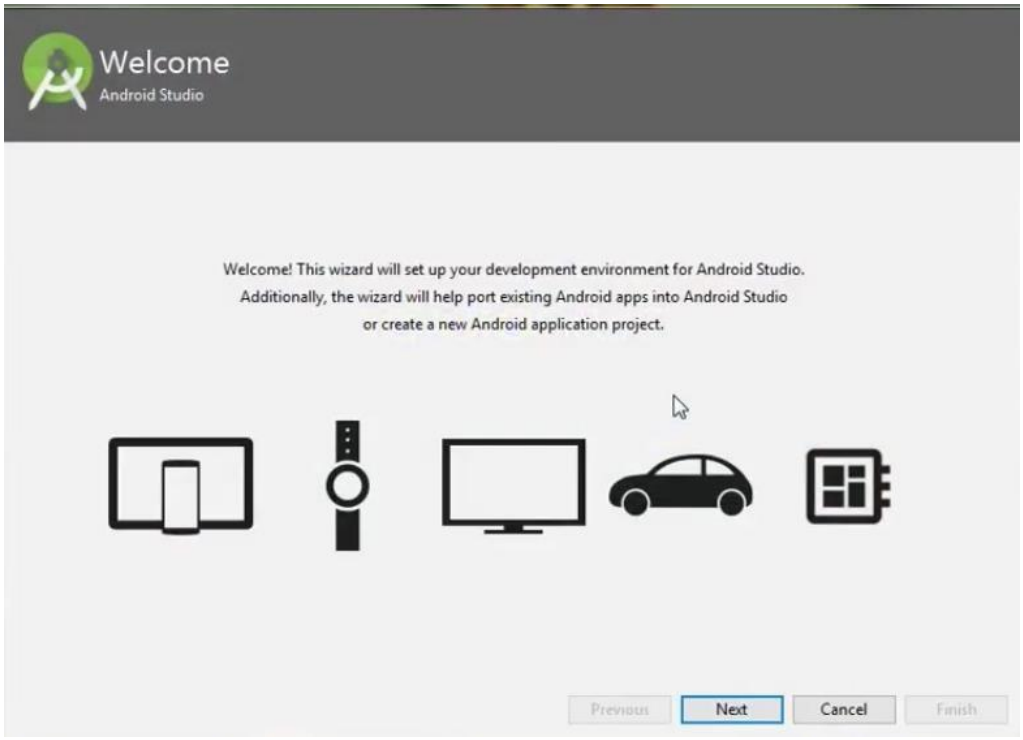
Click the OK button.

Step 6: This will start the Android Studio.

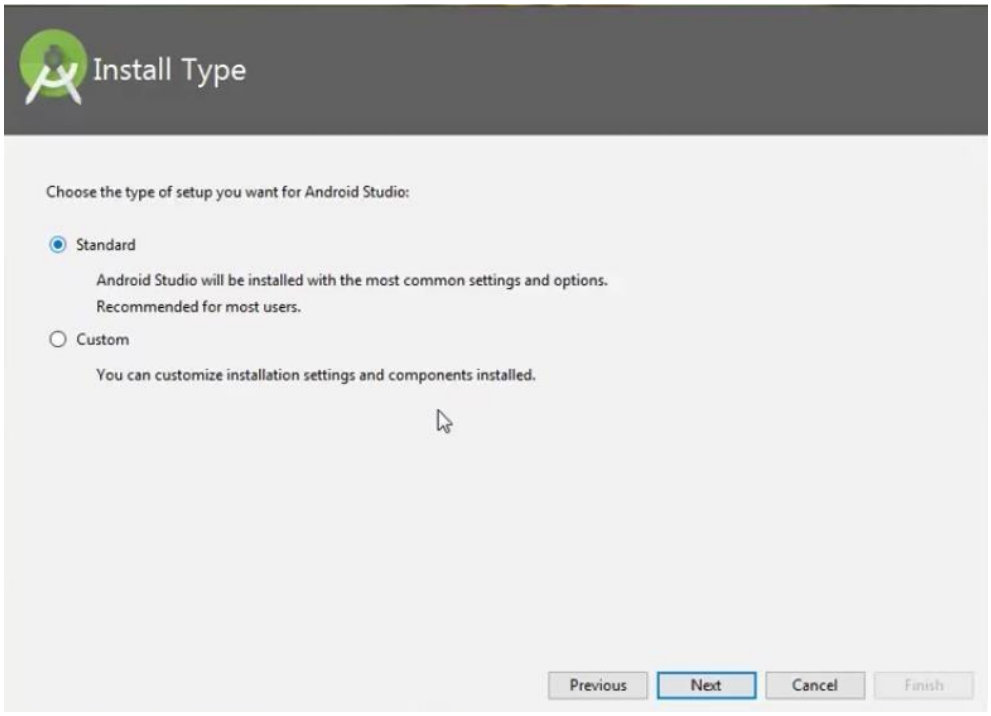
Meanwhile, it will be finding the available SDK components.



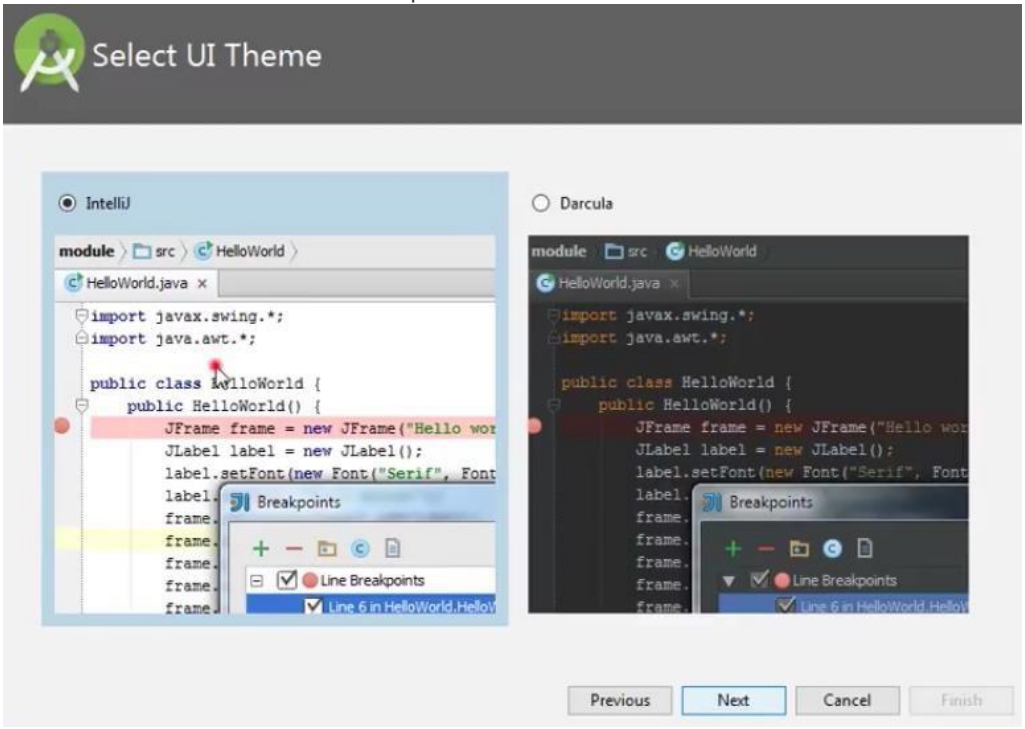
Step 7: After it has found the SDK components, it will redirect to the Welcome dialog box.



Click on Next .

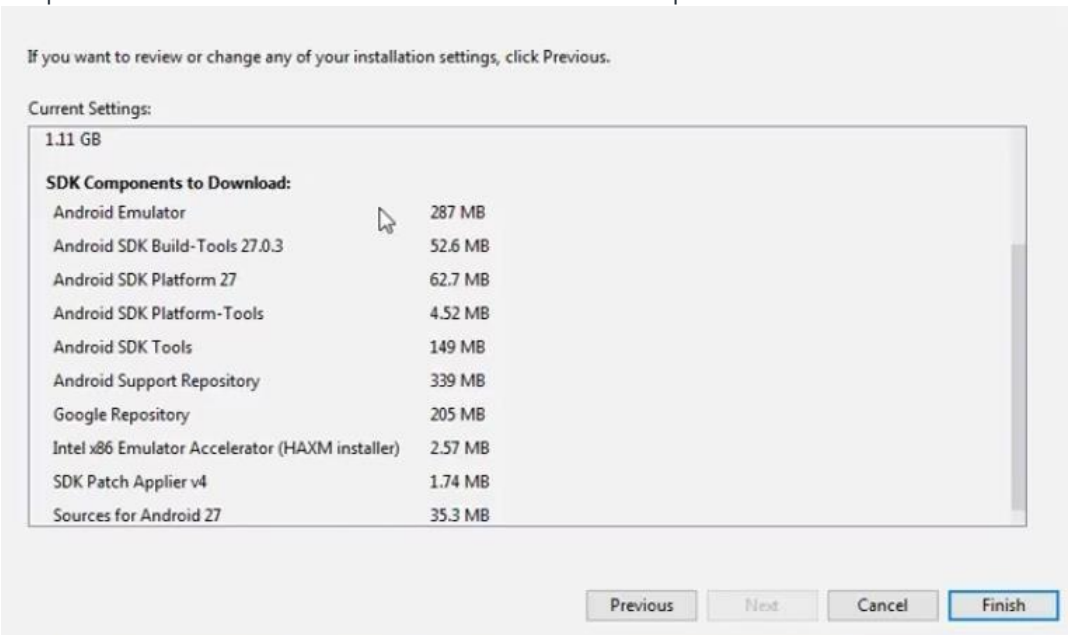


Choose Standard and click on Next. Now choose the theme, whether the Light theme or the Dark one. The light one is called the IntelliJ theme whereas the dark theme is called Dracula . Choose as required.

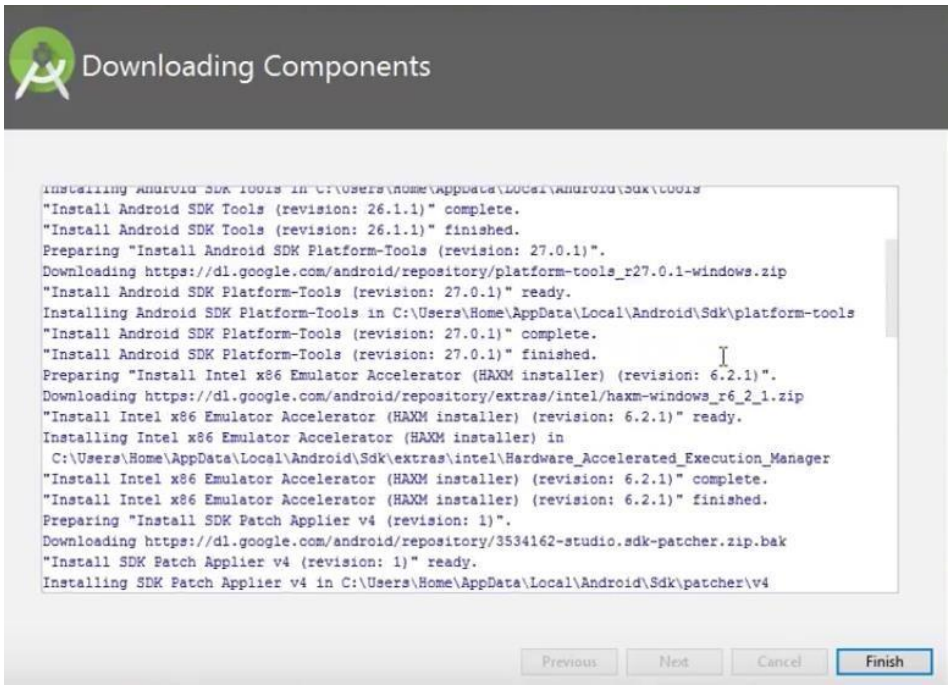


Click on the Next button.

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

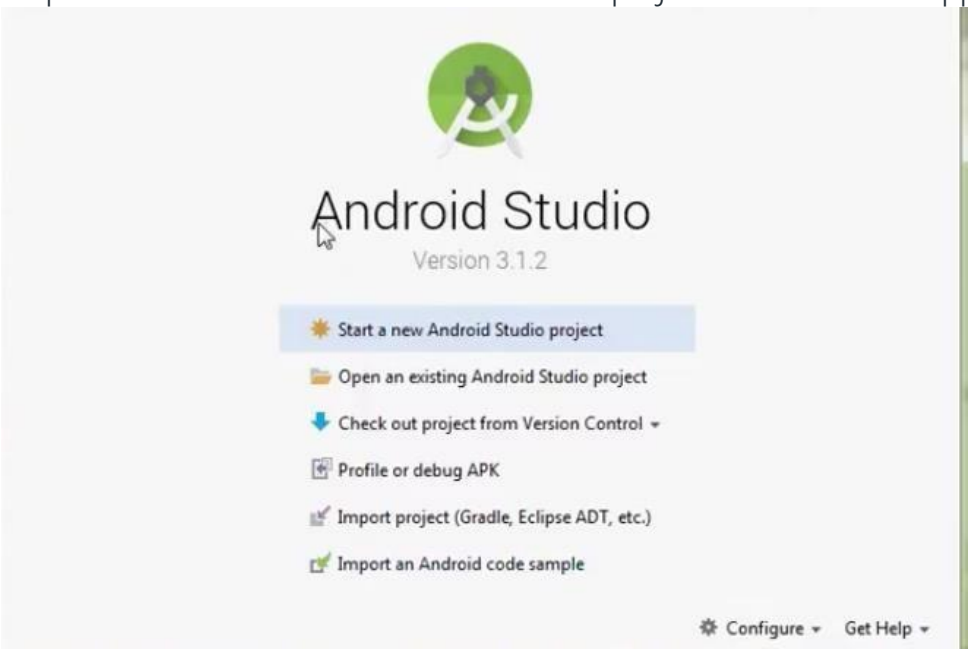


Click on Finish. Components begin to download let it complete.



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.



To run your first android app in Android Studio you may refer to [Running your first Android app.]

❖ First Android Application

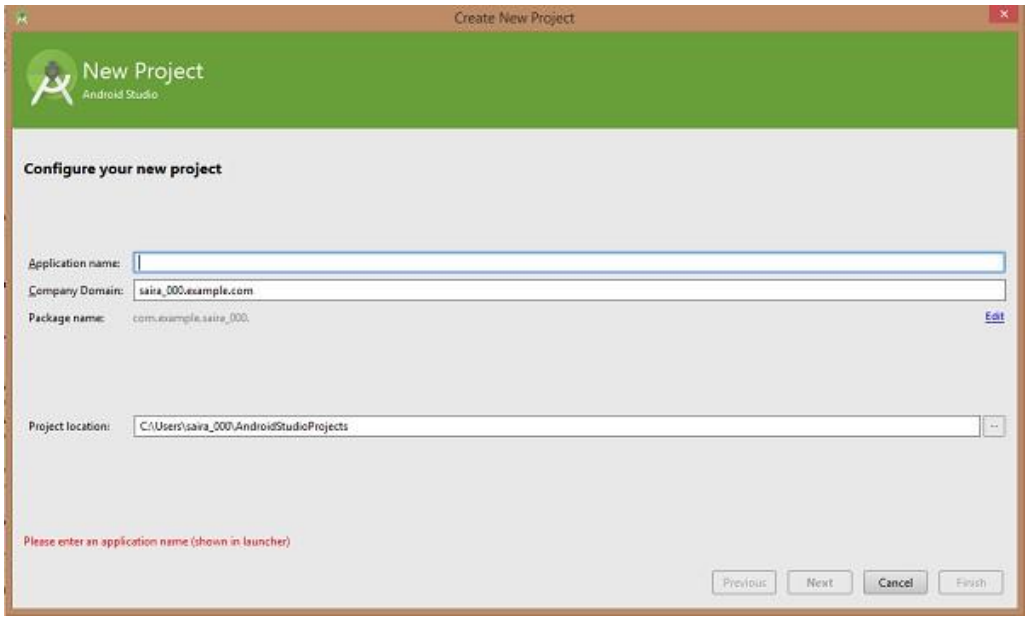
Let us start actual programming with Android Framework. Before you start writing your first example using Android SDK, you have to make sure that you have set-up your Android development environment properly as explained in Android - Environment Set-up tutorial. I also assume that you have a little bit working knowledge with Android studio.

So let us proceed to write a simple Android Application which will print "Hello World!".

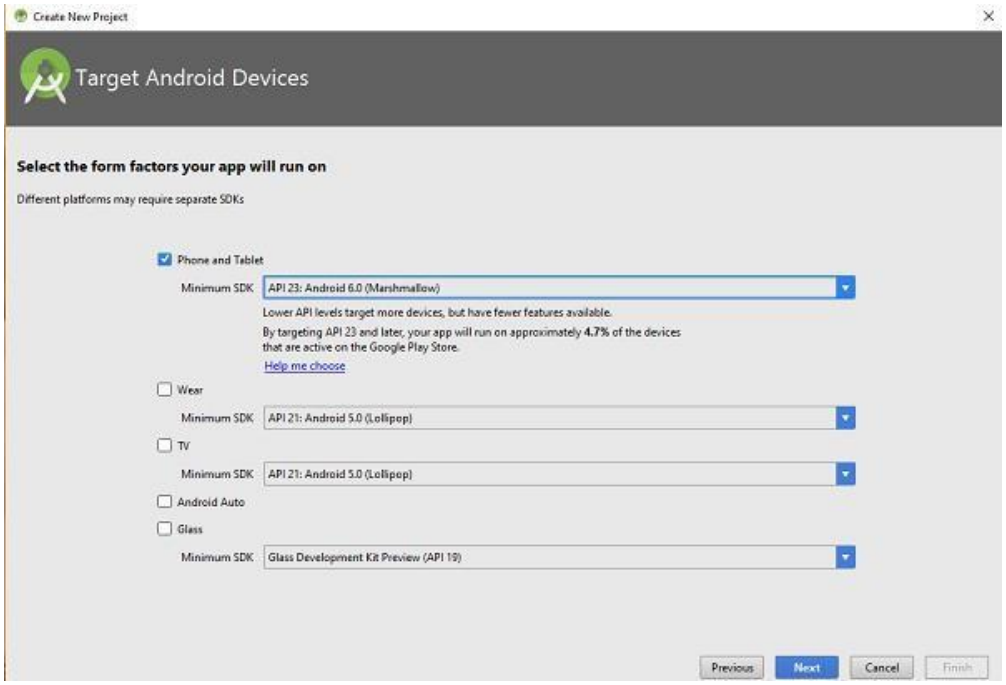
Create Android Application

The first step is to create a simple Android Application using Android studio. When you click on Android studio icon, it will show screen as shown below

You can start your application development by calling start a new android studio project. in a new installation frame should ask Application name, package information and location of the project.–

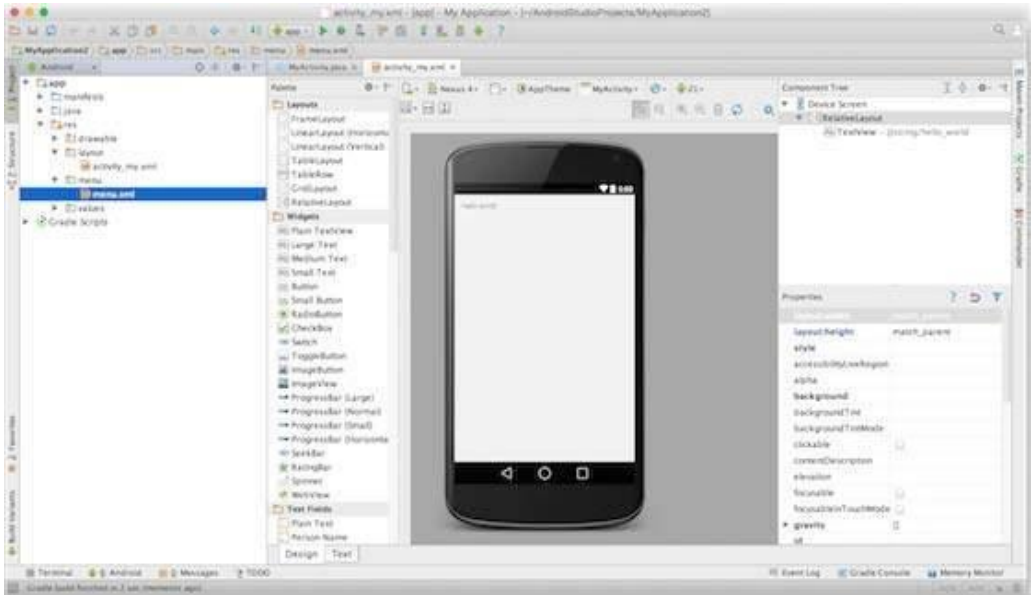


After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Mashmallow) –



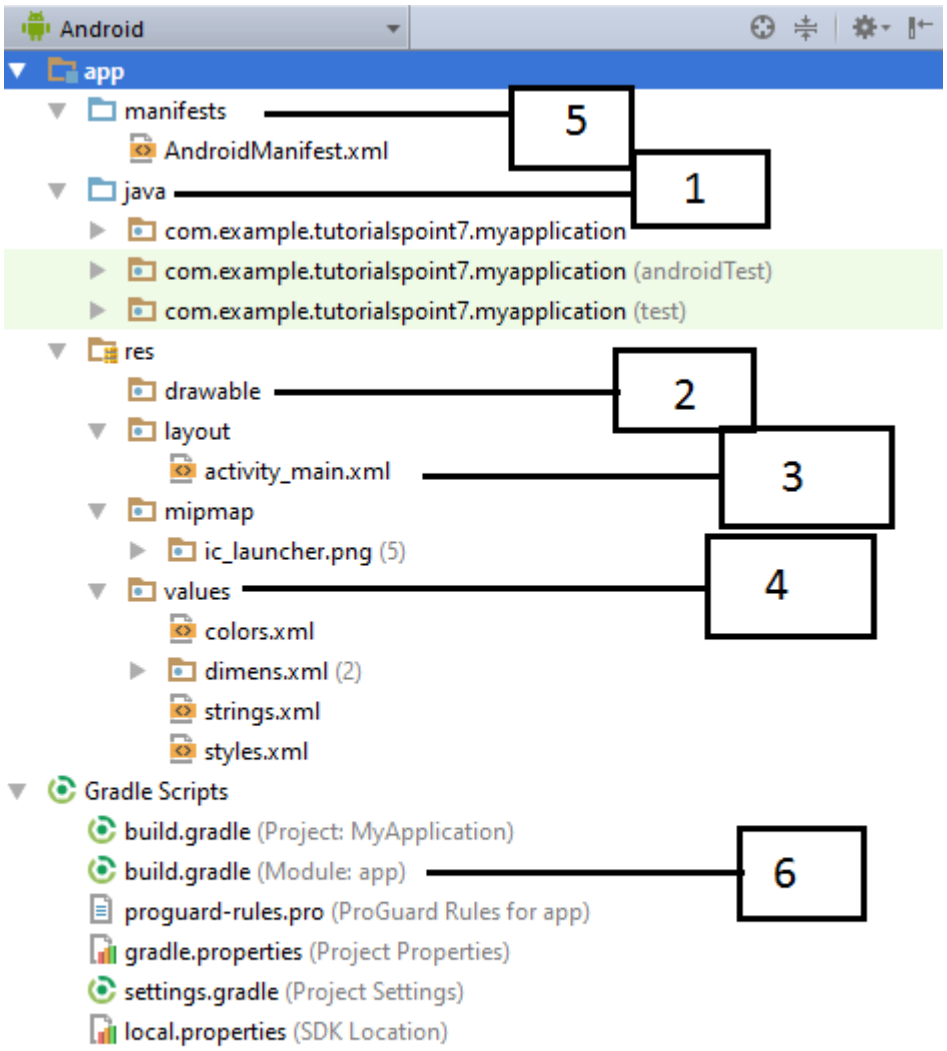
The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications.

At the final stage it going to be open development tool to write the application code.



Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project –



Sr.No.	Folder, File & Description
1	Java This contains the .java source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
2	res/drawable-hdpi This is a directory for drawable objects that are designed for high-density screens.
3	res/layout This is a directory for files that define your app's user interface.
4	res/values This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
5	AndroidManifest.xml

	This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
6	Build.gradle This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

Following section will give a brief overview of the important application files.

The Main Activity File

The main activity code is a Java file MainActivity.java. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application –

```
package com.example.helloworld;

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);
   }
}
```

Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are figured when an activity is loaded.

The Manifest File

Whatever component you develop as a part of your application, you must declare all its components in a *manifest.xml* which resides at the root of the application project directory. This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file –

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tutorialspoint7.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      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>
```

Here <application>...</application> tags enclosed the components related to the application. Attribute *android:icon* will point to the application icon available under *res/drawable-hdpi*. The application uses the image named ic_launcher.png located in the drawable folders

The <activity> tag is used to specify an activity and *android:name* attribute specifies the fully qualified class name of the *Activity* subclass and the *android:label* attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

The action for the intent filter is named *android.intent.action.MAIN* to indicate that this activity serves as the entry point for the application. The category for the intent-filter is named *android.intent.category.LAUNCHER* to indicate that the application can be launched from the device's launcher icon.

The *@string* refers to the *strings.xml* file explained below.
Hence, *@string/app_name* refers to the *app_name* string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application.

Following is the list of tags which you will use in your manifest file to specify different Android application components –

- <activity>elements for activities
- <service> elements for services

- <receiver> elements for broadcast receivers
- <provider> elements for content providers

The Strings File

The strings.xml file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as following file –

```
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="hello_world">Hello world!</string>
   <string name="menu_settings">Settings</string>
   <string name="title_activity_main">MainActivity</string>
</resources>
```

The Layout File

The activity_main.xml is a layout file available in *res/layout* directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout –

```
<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" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:padding="@dimen/padding_medium"
      android:text="@string/hello_world"
      tools:context=".MainActivity" />

</RelativeLayout>
```

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The *TextView* is an Android control used to build the GUI and it have various attributes like *android:layout_width*, *android:layout_height* etc which are being used to set its width and height etc.. The *@string* refers to the strings.xml file located in the res/values

folder. Hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

Running the Application

Let's try to run our Hello World! application we just created. I assume you had created your AVD while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window –

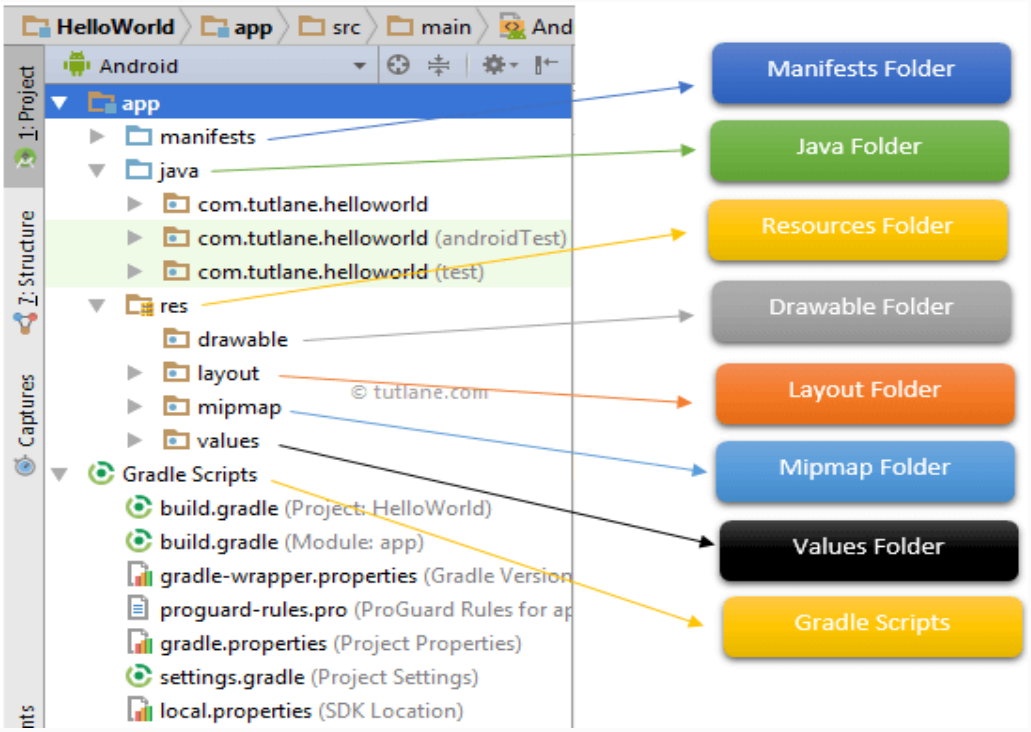
Congratulations!!! you have developed your first Android Application.

UNIT-II

❖ Anatomy of an Android

To implement android apps, Android Studio is the official IDE (Integrated Development Environment) which is freely provided by Google for android app development.

Once we setup android development environment using android studio and if we create a sample application using android studio, our project folder structure will be like as shown below. In case if you are not aware of creating an application using an android studio please check this Android Hello World App.



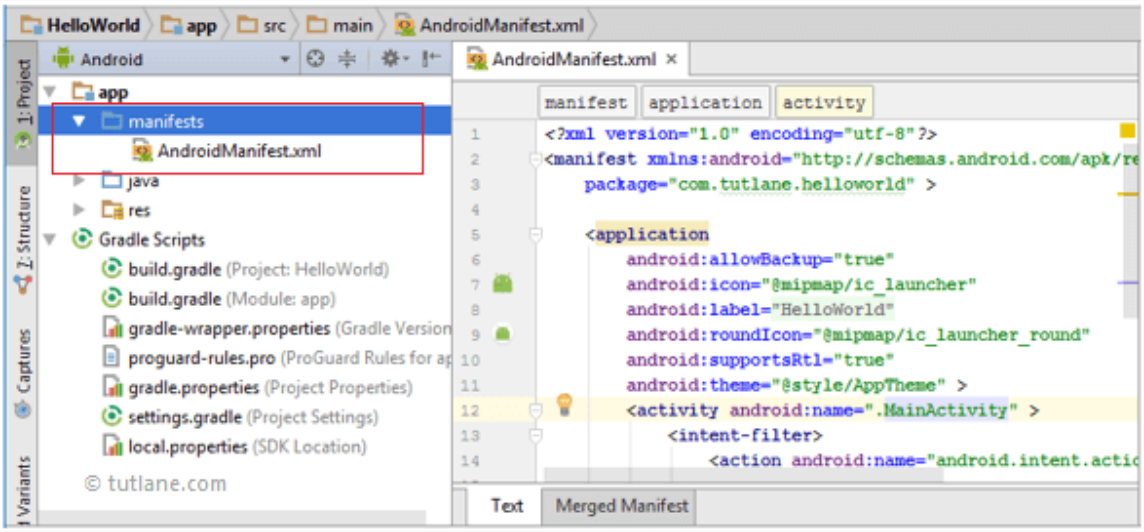
The Android project structure on the disk might be differs from the above representation. To see the actual file structure of the project, select Project from the Project dropdown instead of Android.

The android app project will contain different types of app modules, source code files, and resource files. We will explore all the folders and files in the android

App.

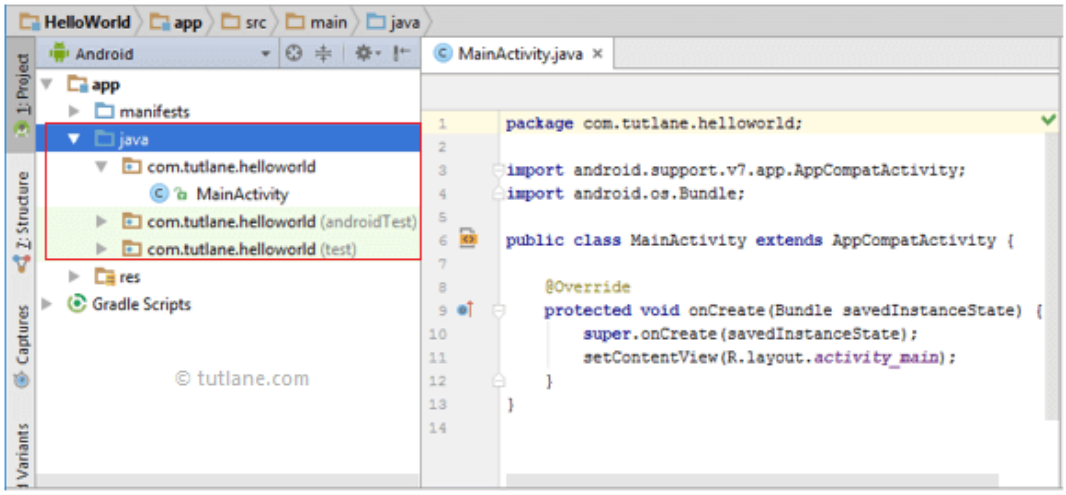
1. Manifests Folder

This folder will contain a manifest file (AndroidManifest.xml) for our android application. This manifest file will contain information about our application such as android version, access permissions, metadata, etc. of our application and its components. The manifest file will act as an intermediate between android OS and our application.Following is the structure of the manifests folder in the android application.



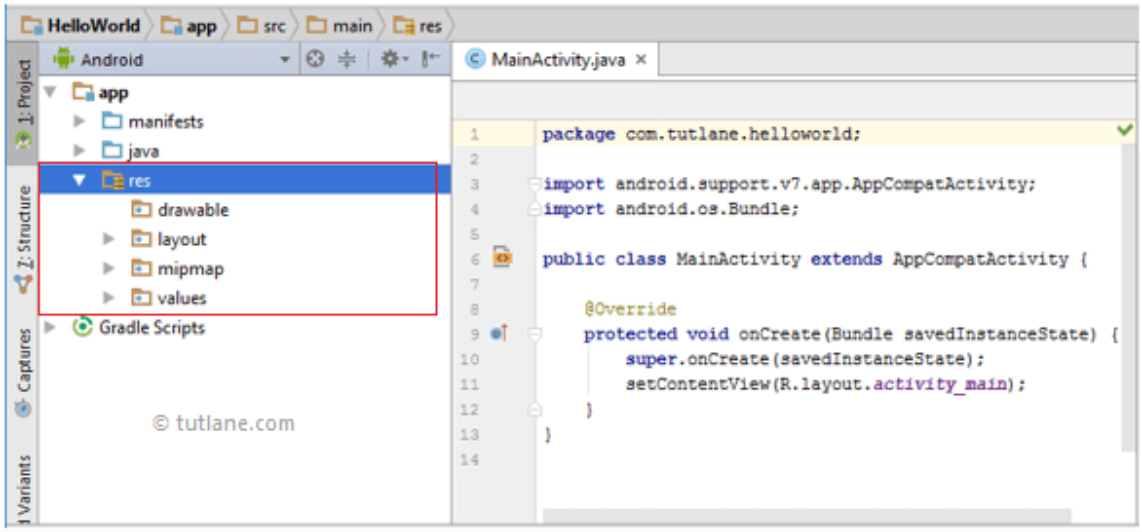
2. Java Folder

This folder will contain all the java source code (.java) files which we'll create during the application development, including JUnit test code. Whenever we create any new project/application, by default the class file MainActivity.java will create automatically under the package name "com.tutlane.helloworld" like as shown below.



3. res (Resources) Folder

It's an important folder that will contain all non-code resources, such as bitmap images, UI strings, XML layouts like as shown below.



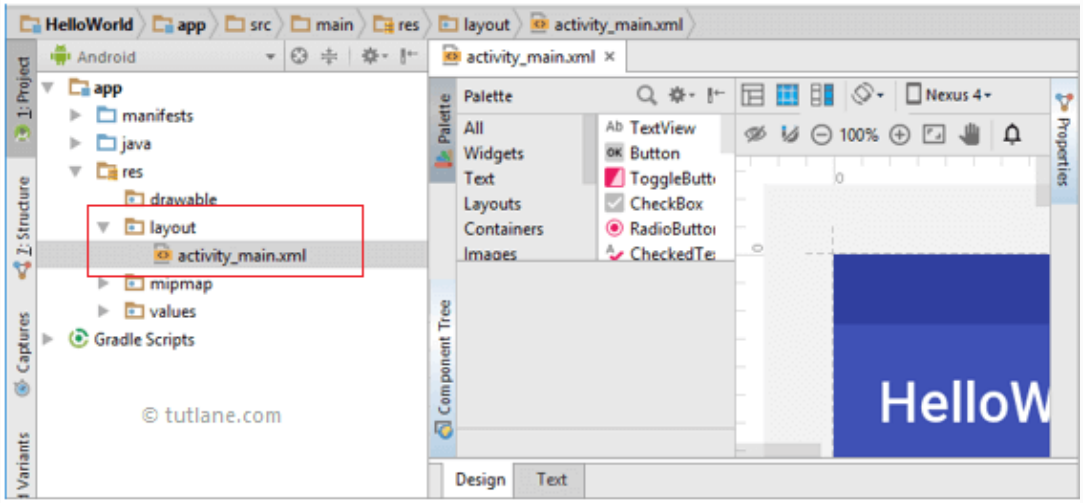
The res (Resources) will contain a different type of folders that are

3.1. Drawable Folder (res/drawable)

It will contain the different types of images as per the requirement of application. It's a best practice to add all the images in a drawable folder other than app/launcher icons for the application development.

3.2. Layout Folder (res/layout)

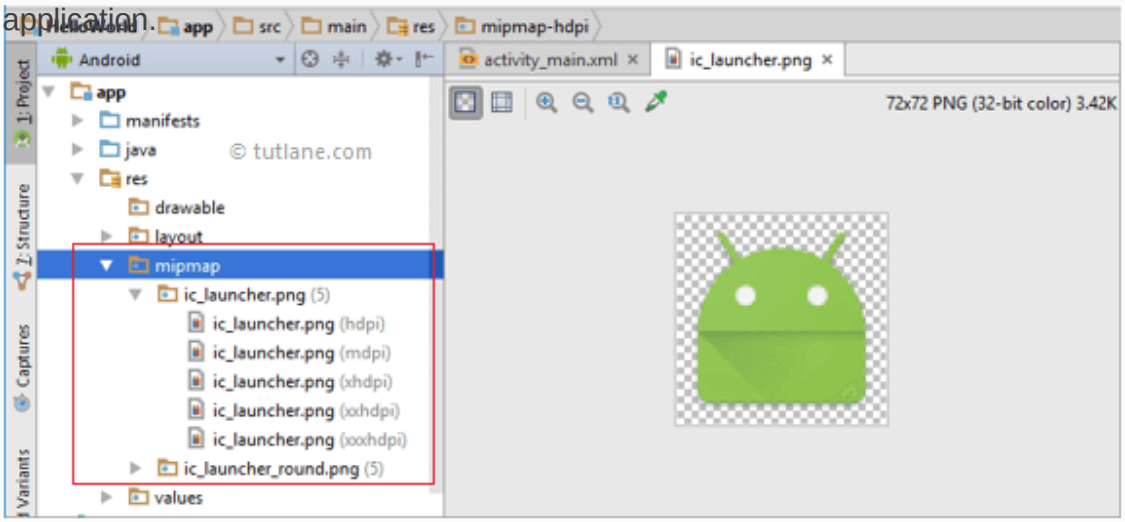
This folder will contain all XML layout files which we used to define the user interface of our application. Following is the structure of the layout folder in the android application.



3.3. Mipmap Folder (res/mipmap)

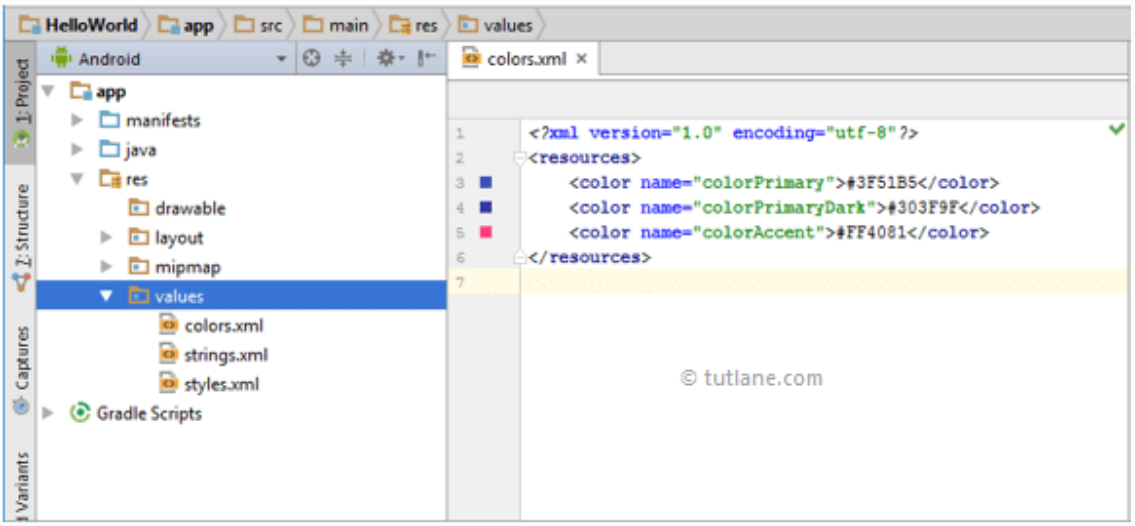
This folder will contain app / launcher icons that are used to show on the home screen. It will contain different density type of icons such as hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi, to use different icons based on the size of the device.

Following is the structure of the mipmap folder in the android application.



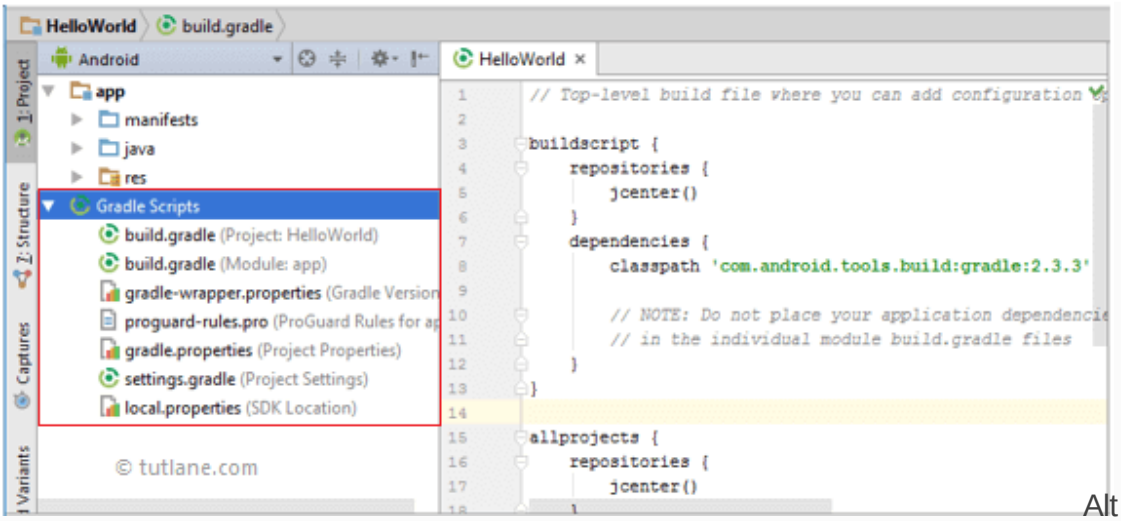
3.4. Values Folder (res/values)

This folder will contain various XML files, such as strings, colors, style definitions and a static array of strings or integers. Following is the structure of the values folder in android application.



4. Gradle Scripts

In android, Gradle means automated build system and by using this we can define a build configuration that applies to all modules in our application. In Gradle build.gradle (Project), and build.gradle (Module) files are useful to build configurations that apply to all our app modules or specific to one app module.Following is the structure of Gradle Scripts in the android application.

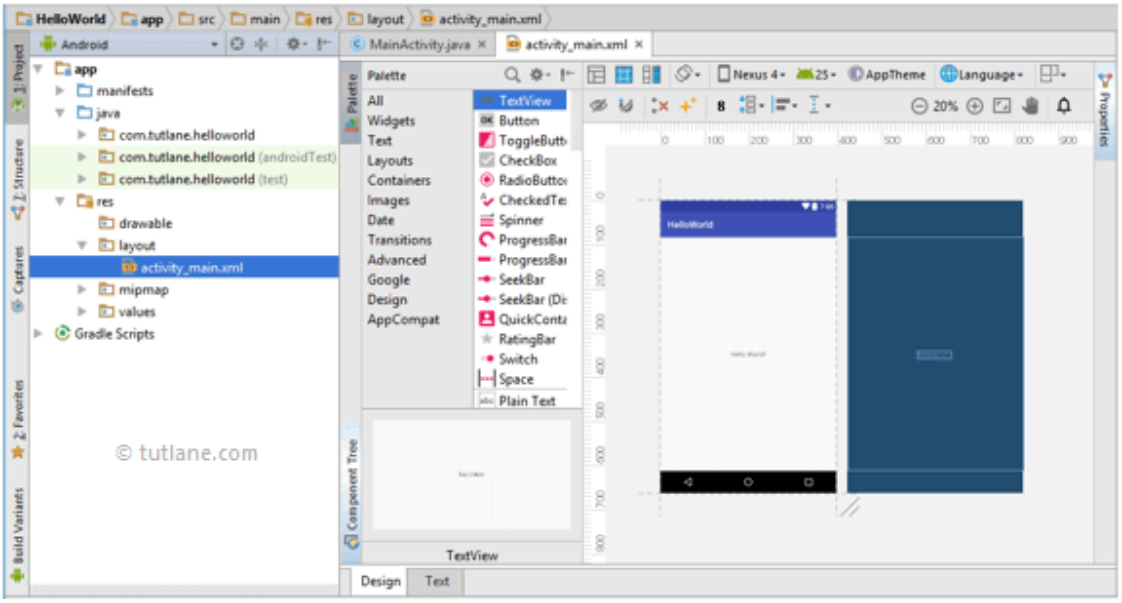


Alt Text

Following are the important files which we need to implement an app in android studio.

5. Android Layout File (activity_main.xml)

The UI of our application will be designed in this file and it will contain Design and Text modes. It will exists in the layouts folder and the structure of activity_main.xml file in Design mode like as shown below.



We can make required design modifications in activity_main.xml file either using Design or Text modes. If we switch to Text mode activity_main.xml file will contain a code like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/andro
   id" 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="com.tutlane.helloworld.MainActivity">
  <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Hello World!"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />
    </android.support.constraint.ConstraintLayout>
```

6. Android Main Activity File (MainActivity.java)

The main activity file in the android application is MainActivity.java and it will exist in the java folder. The MainActivity.java file will contain the java code to handle all the activities related to our app.

Following is the default code of MainActivity.java file which is generated by our HelloWorld application.

```
package com.tutlane.helloworld;
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);
    }
}
```

7. Android Manifest File (AndroidManifest.xml)

Generally, our application will contain multiple activities and we need to define all those activities in the AndroidManifest.xml file. In our manifest file, we need to mention the main activity for our app using the MAIN action and LAUNCHER category attributes in
intent filters (). In case if we didn't mention MAIN action or LAUNCHER category for the main activity, our app icon will not appear in the home screen's list of apps.

Following is the default code of AndroidManifest.xml file which is generated by our HelloWorld application.

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

    <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>
```

These are the main folders and files required to implement an application in android studio. If you want to see the actual file structure of the project, select Project from the Project dropdown instead of Android.

❖ Terminologies Correlated to Android

XML file

The preeminent file is used for the structure of an android project. It has complete information about all the components and packages. It initializes the API that is further used by an application.

View

It is the component of the User Interface that occupies the rectangular area on the screen.

Layout

It properly aligned the views on the screen.

Activity

Activity is a User interface screen through which the user interacts. Users have a right to place the UI elements in any way according to the Users choice.

Emulator

The emulator is the virtual device smartphone provided with an android studio. You can run your created application on the emulator and test its UI and function according to the needs.

Intent

It acts as a communicating object. You can establish a communication between two or more than two components as services, broadcast receivers. It is used to start and end the activity and services components.

Services

It is used to run the process even in the background. There is no defined UI for service. Any component can start the service and end the services. You can easily switch between the applications even if the services are running the background.

Content Provider

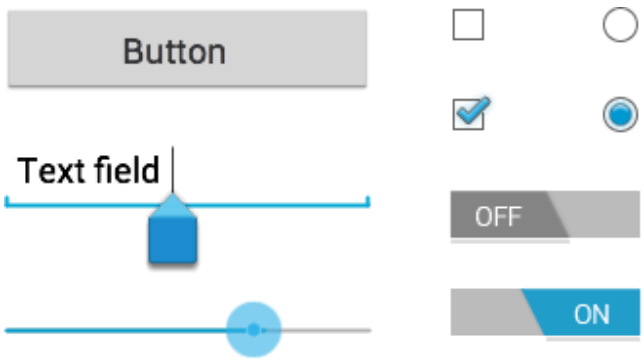
It implemented in two ways:

You can use implement the existing content provider in your application. However, you can also create a new content provider that will provide or share the data with other applications.

❖ *Android UI Controls (Textview, EditText, Radio Button, Checkbox)*

In android **UI** or **input** controls are the interactive or View components that are used to design the user interface of an application. In android we have a wide variety of UI or input controls available, those are TextView, EditText, Buttons, Checkbox, Progressbar, Spinners, etc.

Following is the pictorial representation of user interface (UI) or input controls in android application.



Generally, in android the user interface of an app is made with a collection of **View** and **ViewGroup** objects.

The **View** is a base class for all UI components in android and it is used to create interactive UI components such as TextView, EditText, Checkbox, Radio Button, etc. and it is responsible for event handling and drawing.

The **ViewGroup** is a subclass of **View** and it will act as a base class for layouts and layout parameters. The ViewGroup will provide invisible containers to hold other Views or ViewGroups and to define the layout properties.

To know more about View and ViewGroup in android applications, check this Android View and ViewGroup.

In android, we can define a UI or input controls in two ways, those are

- Declare UI elements in XML
- Create UI elements at runtime

The android framework will allow us to use either or both of these methods to define our application's UI.

Declare UI Elements in XML

In android, we can create layouts same as web pages in HTML by using default **Views** and **ViewGroups** in the XML file. The layout file must contain only one root element, which must be a **View** or **ViewGroup** object. Once we define the root element, then we can add additional layout objects or widgets as a child elements to build View hierarchy that defines our layout.

Following is the example of defining UI controls (TextView, EditText, Button) in the XML file (**activity_main.xml**) using LinearLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/fstTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter Name" />
    <EditText
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"/>
    <Button
        android:id="@+id/getName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get Name" />
</LinearLayout>
```

In android, each input control is having a specific set of events and these events will be raised when the user performs particular action like, entering the text or touches the button.

Load XML Layout File from an Activity

Once we are done with the creation of layout with UI controls, we need to load the XML layout resource from our activity **onCreate()** callback method like as shown below.

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

If you observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout_file_name**. Here our xml file name is **activity_main.xml** so we used file name **activity_main**.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

Create UI Element at Runtime

If we want to create UI elements at runtime, we need to create our own custom **View** and **ViewGroup** objects programmatically with required layouts.

Following is the example of creating UI elements (TextView, EditText, Button) in LinearLayout using custom **View** and **ViewGroup** objects in an activity programmatically.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView textView1 = new TextView(this);
        textView1.setText("Name:");
        EditText editText1 = new EditText(this);
        editText1.setText("Enter Name");
        Button button1 = new Button(this);
        button1.setText("Add Name");
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.addView(textView1);
        linearLayout.addView(editText1);
        linearLayout.addView(button1);
        setContentView(linearLayout);
    }
}
```

By creating a custom **View** and **ViewGroups** programmatically, we can define UI controls in layouts based on our requirements in android applications.

Width and Height

When we define a UI controls in a layout using an XML file, we need to set width and height for every **View** and **ViewGroup** elements using **layout_width** and **layout_height** attributes.

Following is the example of setting width and height for **View** and **ViewGroup** elements in the XML layout file.

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

```
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/fstTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter Name" />
</LinearLayout>
```

If you observe above example, we used different values to set layout_width and layout_height, those are

- match_parent
- wrap_content

If we set value **match_parent**, then the **View** or **ViewGroup** will try to match with parent width or height.

If we set value **wrap_content**, then the **View** or **ViewGroup** will try to adjust its width or height based on the content.

Android Different Types of UI Controls

We have a different type of UI controls available in android to implement the user interface for our android applications.
Following are the commonly used UI or input controls in android applications.

- TextView
- EditText
- AutoCompleteTextView
- Button
- ImageButton
- ToggleButton
- CheckBox
- RadioButton

Android TextView

In android, **TextView** is a user interface control that is used to display the text to the user.

Android EditText

In android, **EditText** is a user interface control which is used to allow the user to enter or modify the text.

Android AutoCompleteTextView

In android, **AutoCompleteTextView** is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.

Android Button

In android, **Button** is a user interface control that is used to perform an action when the user clicks or tap on it.

Android Image Button

In android, **Image Button** is a user interface control that is used to display a button with an image to perform an action when the user clicks or tap on it.
Generally, the Image button in android looks similar as regular Button and perform the actions same as regular button but only difference is for image button we will add an image instead of text.

Android Toggle Button

In android, **Toggle Button** is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.

Android CheckBox

In android, **Checkbox** is a two-states button that can be either checked or unchecked.

Android Radio Button

In android, **Radio Button** is a two-states button that can be either checked or unchecked and it cannot be unchecked once it is checked.

❖ Application Context in Android

The Application Context is a context tied to the lifecycle of the application and is used across the app's components to access application-level resources and services. It remains available throughout the app's lifetime and is not tied to the lifecycle of a specific activity or fragment.

I) Key Features of Application Context

1. Global Scope
   The Application Context is not linked to a specific activity or view; it persists as long as the application is running.
2. Resource Access
   It provides access to resources like:
   - Shared Preferences: Storing small key-value pairs persistently.
   - File System: Accessing app-specific files in internal or external storage.
   - Databases: Interacting with SQLite or Room databases.
3. System Services
   Application Context enables access to Android system-level services, such as:
   - ConnectivityManager: Checking or managing network connectivity.
   - LocationManager: Accessing location-related functionality.
   - AlarmManager: Setting alarms.
   - NotificationManager: Managing notifications.

II) When to Use Application Context

1. Long-lived Objects:
   Use Application Context when the context needs to outlive the activity or fragment, such as:
   - Background tasks.
   - Application-wide singleton instances.

2. Avoiding Memory Leaks:
   It helps prevent memory leaks since the Application Context is tied to the application's lifecycle rather than the activity's.
3. Accessing Application Resources:
   When working with components that need resources independent of a specific UI component, such as:
   - Custom adapters.
   - Database helpers.

III) Example:

Accessing Shared Preferences

Here's how you can use SharedPreferences with the Application Context:

```
SharedPreferences sharedPreferences =
    getApplicationContext().getSharedPreferences("MyPrefs", Context.MODE_PRIVATE);


SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("username", "Ram");
editor.putInt("age", 30);
editor.apply(); // or editor.commit();
```

- Mode:
  - Context.MODE_PRIVATE: Only the app can access the file (default).
  - Other modes like MODE_WORLD_READABLE and MODE_WORLD_WRITEABLE are deprecated.
- apply() vs. commit():
  - apply(): Asynchronous, does not return a result, faster for UI thread.
  - commit(): Synchronous, returns a boolean indicating success or failure.

2. Retrieving Data

```
SharedPreferences sharedPreferences =

    getApplicationContext().getSharedPreferences("MyPrefs", Context.MODE_PRIVATE);


String username = sharedPreferences.getString("username", "DefaultName"); int

age = sharedPreferences.getInt("age", 0);


System.out.println("Username: " + username);

System.out.println("Age: " + age);
```

- The second parameter in getString() or getInt() is the default value returned if the key is not found.

❖ *Activities in Android*

Activities are part of the basic component of the Android application. Users cannot interact with an application on their mobile device without an activity. Activities form the bedrock upon which our mobile application is built. It consists of the layout, text fields, images, and different UI elements which the user can interact with.

Every mobile application you open, the first screen you see is an activity. In Android Studio activities are created using the Java or Kotlin programming language.

Activities are linked through intents, that are used for communication and navigation between different activities. Intents can be used to start a new activity, pass data between activities, or even trigger actions in other components of the app.

This article will provide a detailed explanation of activities, their importance, and activity life cycle along with its methods.

An activity

Activity is more like a container for one or more multiple screens in your app. It's not just seen as a screen but also as a unit that users interact with in your application.

Activities contain information about whether a user is currently interacting with the screen or if the activity is in the background.

It also serves as an entry point for your app. We can have multiple screens in an application that are bundled together into an activity.

For example, A user profile page activity can contain other screens like information details of the user.

However, Jetpack Compose has made everything easier. Using the jetpack compose UI we can have only one main activity with multiple screens while staying in the single main activity.

The main activity in Jetpack Compose serves as the entry point of our application. The main characteristic of activities in Android is the `LIFECYCLE` which means at some point our activity is created, destroyed, paused, etc.

Basic components of an activity

1. `Layout XML:` the layout XML file represents the user interface of an activity. These files contain view-groups (linear layout, constraint layout) and views such as buttons, images, app bars, etc.

   The XML file defines the structure and positioning of these components which the user will interact with like clicking a button to open a new activity.

   However, in Jetpack Compose you can define and structure the features of your app inside the main.activity.kt file.
2. `Activity class:` An activity class comprising a single screen with a user interface is represented by an instance(object) of the Activity class in Android.

   To create a custom activity you must create a subclass of the Activity class and override its lifecycle methods and other crucial methods to the application's needs.
3. `Life cycle methods:` these methods perform various behaviors in activity and can be overridden in different stages. The methods are:`onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`, etc.

- ***Services***

A Service is an application component that can perform long-running operations in the background. It does not provide a user interface. Once started, a service might continue running for some time, even after the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Types of Services

These are the three different types of services:

Foreground

A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services must display a Notification. Foreground services continue running even when the user isn't interacting with the app.

When you use a foreground service, you must display a notification so that users are actively aware that the service is running. This notification cannot be dismissed unless the service is either stopped or removed from the foreground.

Background

A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

Bound

A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

❖ *Intents in Android*

Intents are messaging objects used to communicate between app components, such as Activities, Services, or Broadcast Receivers. They enable actions like launching activities, starting services, or delivering broadcasts.

Types of Intents

1. Explicit Intent

- Definition: Specifies the target component (e.g., an Activity or Service) by name.
- Use Cases:
  - Navigating between activities within the same app.
  - Starting a service.
- Key Features:
  - Target is explicitly defined using the component name or class.
  - Ideal for intra-app communication.
- Example: Navigating to another activity.

```
Intent intent = new Intent(this, SecondActivity.class); startActivity(intent);
```

- Example: Starting a service.

```
Intent intent = new Intent(this, MyService.class);
startService(intent);
```

2. Implicit Intent

- Definition: Describes a general action to be performed, allowing the Android system to determine which component (app or service) can handle it.
- Use Cases:
  - Sharing data.
  - Opening a web page or sending an email.
  - Choosing apps to handle user actions (e.g., opening a file).
- Key Features:
  - Does not explicitly name the target component.
  - Relies on intent filters in the manifest to match actions and categories.
- Example: Opening a web page.

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.example.com"));
startActivity(intent);
```

- Example: Sharing data.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain"); intent.putExtra(Intent.EXTRA_TEXT,
"Hello, world!"); startActivity(Intent.createChooser(intent,
"Share via"));
```

Components of an Intent

1. Action: The operation to be performed (e.g., ACTION_VIEW, ACTION_SEND).
2. Data: The data to be acted upon, often specified as a URI.
3. Category: Provides additional information about the action (e.g., CATEGORY_DEFAULT).
4. Extras: Key-value pairs for passing additional data.
5. Flags: Instructions on how to launch the activity (e.g., FLAG_ACTIVITY_NEW_TASK).

Passing Data with Intents

You can use the putExtra() method to pass data and retrieve it in the target component using getExtras().

- Sending Data:

```
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra("KEY_NAME", "Ram");
intent.putExtra("KEY_AGE", 25);
startActivity(intent);
```

- Receiving Data:

```
Intent intent = getIntent();
```

```
String name = intent.getStringExtra("KEY_NAME"); int
age = intent.getIntExtra("KEY_AGE", 0);
```

Intent Filters

For an Implicit Intent to be handled, the target component must define an intent filter in its manifest file.

- Example:

```
<activity android:name=".SecondActivity">
 <intent-filter>
 <action android:name="android.intent.action.VIEW"/>
 <category android:name="android.intent.category.DEFAULT"/>
 <data android:scheme="https" android:host="www.example.com"/>
</intent-filter>
 </activity>
```

❖ Passing Data Between Activities using Extras

We can pass data between different activities when their started and when they end. We start activities by creating `intents`. We can "put extra" information in an intent that starts a new activity.
Gmail has at least two Activities. Let's imagine there is a `ViewInbox` activity and a `ReadEmail` activity. When someone clicks on an email in the inbox the `ViewInbox` activity, the activity needs to create a new intent to start a `ReadEmail` activity. Simply starting the `ReadEmail` activity isn't enough.
The `ReadEmail` activity needs to know what email it's supposed to display.
The `ViewInbox` activity puts extra information in the intent that specifies which email should be displayed.
Here's some simple examples of how to put extra information in an intent, and how to get it out.

When creating new intents, you can also give it *extra* data. Here's an example:

```
Intent intent = new Intent(EmailListActivity.this, ReadEmailActivity.class);
intent.putExtra("ID", 123);
intent.putExtra("SENDER",  "RAM");
```

The Intent class has a handful of helper methods you can call to get and store extra data. The main one is `putExtra()`, which takes two parameters: a String that gives the data a name, and the data itself.
With `Intent.putExtra()`, you can put data inside the intent (including Strings, numbers, booleans, certain objects).
Once you start a new activity, you can retrieve the Intent and get the sent data, as follows:

```
// get the intent that started this activity
Intent intent = getIntent();

// get the data from the intent
int id = intent.getIntExtra("ID", 0);
String sender = intent.getStringExtra("SENDER");
```

Again, the Intent class has a handful of getters for extra data, usually formatted like get_Extra. Examples, `getIntExtra()`, `getStringExtra()`, `getBooleanExtra()`, etc.

❖ Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents –

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

Creating the Broadcast Receiver

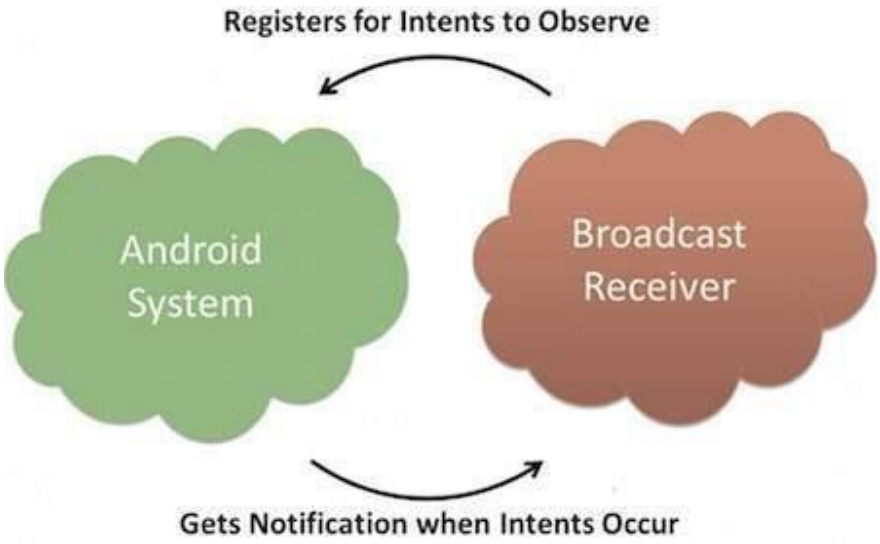
A broadcast receiver is implemented as a subclass of BroadcastReceiver class and overriding the onReceive() method where each message is received as a Intent object parameter.

```
public class MyReceiver extends BroadcastReceiver { @Override
```

```
  public void onReceive(Context context, Intent intent) { Toast.makeText(context,
    "Intent Detected.", Toast.LENGTH_LONG).show();
  }
}
```

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in AndroidManifest.xml file. Consider we are going to register MyReceiver for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.



Broadcast-Receiver

```
<application android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <receiver android:name="MyReceiver">

    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED">
      </action>
    </intent-filter>

  </receiver>
```

```
</application>
```

Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver MyReceiver and implemented logic inside onReceive() will be executed.

There are several system generated events defined as final static fields in the Intent class. The following table lists a few important system events.

Sr.No	Event Constant & Description
1	android.intent.action.BATTERY_CHANGED Sticky broadcast containing the charging state, level, and other information about the battery.
2	android.intent.action.BATTERY_LOW Indicates low battery condition on the device.
3	android.intent.action.BATTERY_OKAY Indicates the battery is now okay after being low.
4	android.intent.action.BOOT_COMPLETED This is broadcast once, after the system has finished booting.
5	android.intent.action.BUG_REPORT Show activity for reporting a bug.
6	android.intent.action.CALL Perform a call to someone specified by the data.
7	android.intent.action.CALL_BUTTON The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8	android.intent.action.DATE_CHANGED The date has changed.
9	android.intent.action.REBOOT Have the device reboot.

❖ *The Android Manifest File*

The Android Manifest File (AndroidManifest.xml) is an essential configuration file in every Android application. It provides information about the app's structure and requirements

to the Android operating system. Here's an overview of its purpose and common settings:

Purpose of the Android Manifest File

1. Declare Application Components:
   - Registers app components such as activities, services, broadcast receivers, and content providers.
   - Ensures the Android system is aware of the components and their configurations.
2. Request Permissions:
   - Specifies permissions the app needs to access system features (e.g., camera, location, internet).
3. Define Hardware and Software Features:
   - Declares the hardware or software features required for the app (e.g., camera, GPS).
4. Set Application Metadata:
   - Provides metadata like app name, version, icons, and themes.
5. Filter Device Compatibility:
   - Restricts app installation on incompatible devices based on hardware, features, or API levels.
6. Control App Behavior:
   - Specifies application-level configurations such as launch mode, process behavior, and backup options.

Example AndroidManifest.xml

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

   <!-- Versioning Information -->
   <uses-sdk android:minSdkVersion="21"
      android:targetSdkVersion="33" />

   <!-- Permissions -->
   <uses-permission  android:name="android.permission.INTERNET"  />
```

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

<!-- Hardware Features -->
<uses-feature android:name="android.hardware.camera" android:required="true" />

<!-- Application Block -->
<application
   android:allowBackup="true"
   android:icon="@mipmap/ic_launch
   er"
   android:label="@string/app_name"
   android:theme="@style/AppTheme
   ">

   <!-- Main Activity -->
   <activity
      android:name=".MainActivity"
      android:label="@string/app_na
      me" android:exported="true">
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>

   <!-- Secondary Activity -->
   <activity
      android:name=".SecondActivity"
      android:label="@string/second_activity"
      android:exported="true">
      <intent-filter>
         <action android:name="com.example.myapp.ACTION_VIEW" />
         <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
   </activity>

   <!-- Service -->
   <service
      android:name=".MyBackgroundService"
```

```
        android:enabled="true"
        android:exported="false" />

    <!-- Broadcast Receiver -->
    <receiver
        android:name=".MyBroadcastReceiver"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>

    <!-- Content Provider -->
    <provider android:name=".MyContentProvider"
        android:authorities="com.example.myapp.provider"
        android:exported="false" />

    <!-- Metadata -->
    <meta-data android:name="com.google.android.maps.v2.API_KEY"
        android:value="your_api_key" />

  </application>
</manifest>
```

Explanation of the Program

1. <manifest>:
   - The root tag includes the app's package name and XML namespace declaration.
2. Versioning:
   - <uses-sdk> defines the minimum and target SDK levels.
3. Permissions:
   - <uses-permission> declares permissions like internet access, location, and camera.
4. Hardware Features:
   - <uses-feature> ensures that the app requires a camera and will not run on devices without one.

5. Application Settings:
   - <application> defines global app settings, including backup, theme, and the app's icon.
6. Activity:
   - <activity> registers MainActivity as the launcher activity with an intent filter for MAIN and LAUNCHER.
   - SecondActivity demonstrates handling a custom action.
7. Service:
   - <service> registers a background service (MyBackgroundService).
8. Broadcast Receiver:
   - <receiver> registers MyBroadcastReceiver to handle system events like BOOT_COMPLETED.
9. Content Provider:
   - <provider> declares a custom content provider (MyContentProvider) with a unique authority.
10. Metadata:
    - <meta-data> is used to store custom data like API keys for Google Maps.

❖ *Android Emulator*

The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device. The emulator offers these advantages:

- Flexibility: In addition to being able to simulate a variety of devices and Android API levels, the emulator comes with predefined configurations for various Android phone, tablet, Wear OS, Android Automotive OS, and Android TV devices.
- High fidelity: The emulator provides almost all the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.
- Speed: Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.

In most cases, the emulator is the best option for your testing needs. This page covers the core emulator functionalities and how to get started with it.

Alternatively, you can deploy your app to a physical device. For more information, see Run apps on a hardware device.

Get started with the emulator

The Android Emulator lets you test your app on many different devices virtually. The emulator comes with Android Studio, so you don't need to install it separately. To use the emulator, follow these basic steps, which are described in more detail in the sections that follow:

1. Verify that you have the system requirements.
2. Create an Android Virtual Device (AVD).
3. Run your app on the emulator.
4. Navigate the emulator.

This page covers the steps to set up and explore your virtual testing environment in more detail. If you already have your app running on the emulator and are ready to use more advanced features, see Advanced emulator usage.

If you're experiencing issues with the emulator, see Troubleshoot known issues with Android Emulator. Depending on your needs and resources, it might be worth delving into system requirements and technical configurations, or it might be better to use a physical device.

Emulator system requirements

For the best experience, you should use the emulator in Android Studio on a computer with at least the following specs:

- 16 GB RAM
- 64-bit Windows 10 or higher, MacOS 12 or higher, Linux, or ChromeOS operating system
- 16 GB disk space

❖ *Create an Android Virtual Device*

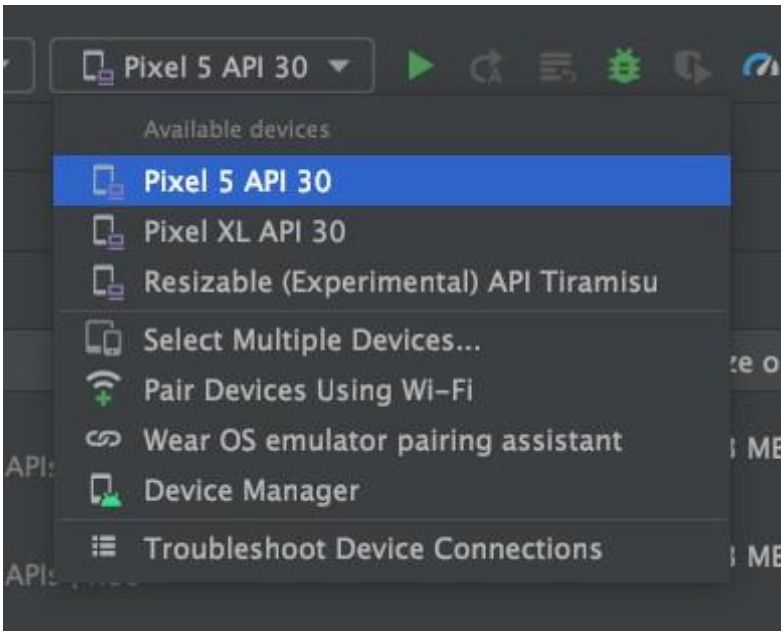
Each instance of the Android Emulator uses an Android virtual device (AVD) to specify the Android version and hardware characteristics of the simulated device. To create an AVD, see Create and manage virtual devices.

Each AVD functions as an independent device with its own private storage for user data, SD card, and so on. By default, the emulator stores the user data, SD card data, and cache in a directory specific to that AVD. When you launch the emulator, it loads the user data and SD card data from the AVD directory.

Run your app on the emulator

After you have created an AVD, you can start the Android Emulator and run an app in your project:

1. In the toolbar, select the AVD that you want to run your app on from the target device menu.



The target device menu.

2. Click Run. The emulator might take a minute or so to launch for the first time, but subsequent launches use a snapshot and should launch faster. If you experience issues, see the troubleshooting guide.

Once your app is installed on your AVD, you can run it from the device as you would run any app on a device. Any time you want to deploy new changes, you need to click Run or Apply Changes again.

Wear OS pairing assistant

If you want to test your app with Wear OS devices, the Wear OS pairing assistant guides you step-by-step through pairing Wear OS emulators with physical or virtual phones directly in Android Studio. To learn more, see Use the Wear OS emulator pairing assistant.

Navigate the emulator

While the emulator is running, you can use your computer mouse pointer to mimic your finger on the touch screen and use the emulator panel to perform common actions.

Navigate the emulator screen

Use your computer mouse pointer to mimic your finger on the touchscreen, select menu items and input fields, and click buttons and controls. Use your computer keyboard to type characters and enter emulator shortcuts.

UNIT-III

❖ User Interface (UI) Screen Elements

In Android development, the User Interface (UI) Screen Elements are the building blocks that help design interactive layouts for apps. These elements are primarily composed of View and ViewGroup components. Below are the key UI elements:

I. Basic UI Elements

- ✓ TextView → Displays text to the user.
- ✓ EditText → Input field for user text input (e.g., username, password).
- ✓ Button → Clickable element that performs actions.
- ✓ ImageView → Displays images.
- ✓ ProgressBar → Shows loading progress.
- ✓ CheckBox → Allows selection of multiple options.
- ✓ RadioButton & RadioGroup → Allows selection of a single option from a group.
- ✓ Switch → A toggle button for ON/OFF states.

II. Layouts (ViewGroups)

These define how UI elements are arranged:

- ✓ LinearLayout → Arranges elements in a single row or column.
- ✓ RelativeLayout → Positions elements relative to each other.
- ✓ ConstraintLayout → A flexible layout that uses constraints for positioning.
- ✓ FrameLayout → Holds a single child view, useful for overlays.
- ✓ TableLayout → Organizes elements in rows and columns.
- ✓ ScrollView → Enables vertical scrolling.

III. UI Containers

- ✓ Dialog → Shows pop-up messages or input prompts.
- ✓ PopupMenu → Displays a small menu when a button is clicked.
- ✓ DrawerLayout (Navigation Drawer) → A side menu for navigation.

- ✓ BottomSheet → A modal or persistent UI component that slides up from the bottom.

Basic UI Elements

View Components (Basic UI Elements) in Android

View components are the essential building blocks of an Android app's UI. They are responsible for displaying content and handling user interactions. Below are some of the most commonly used View components in Android development:

1. TextView

- Used to display static text to the user.
- Cannot be edited by the user (unlike EditText).
- Supports text formatting, styling, and

linking. Example:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello, World!"
   android:textSize="18sp"
   android:textColor="@android:color/black"/>
```

2. EditText ✍

- Allows the user to enter text (input field).
- Can be used for passwords, numbers, emails,

etc. Example:

```
<EditText
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Enter your name"
android:inputType="text"/>
```

3. Button

- Used to perform an action when clicked.
- Supports different styles (e.g., default, elevated,

outlined). Example:

```
<Button android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Click Me"
android:onClick="handleClick"/>
```

(In the Java/Kotlin file, define handleClick(View view) to handle button clicks.)

4. ImageView

- Displays images or icons.
- Can load images from resources or

```
<ImageView android:layout_width="100dp"
android:layout_height="100dp"
```

URLs. Example:

```
    android:src="@drawable/sample_image"/>
```

5. CheckBox

- Allows users to select multiple options

independently. Example:

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I agree to the terms"/>
```

6. RadioButton & RadioGroup

- RadioButton allows users to choose only one option from a group.
- RadioGroup ensures that only one option is selected at a

time. Example:

```
<RadioGroup android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Option 1"/>

    <RadioButton
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 2"/>
</RadioGroup>
```

7. ProgressBar

- Used to indicate loading or processing

progress. Example (Indeterminate Circular

Progress):

```
<ProgressBar
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
```

Example (Determinate Horizontal Progress):

```
<ProgressBar
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  style="@android:style/Widget.ProgressBar.Horizontal"
  android:progress="50"
  android:max="100"/>
```

8. SeekBar

- Allows users to select a value by

sliding. Example:

```
<SeekBar
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"/>
```

Layouts (ViewGroups)

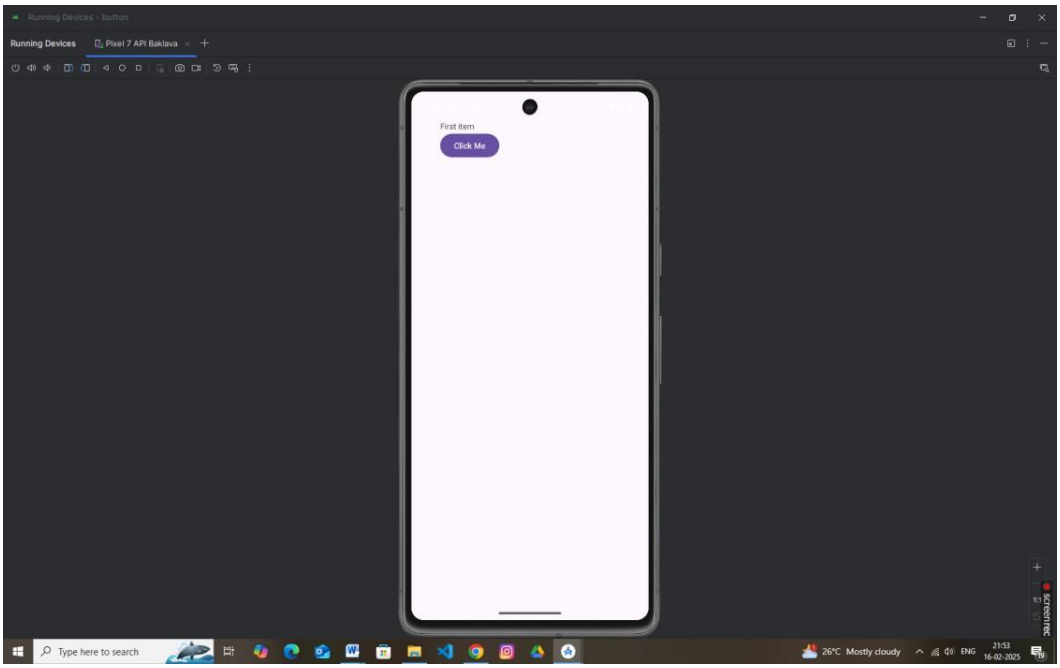
These layout containers help you structure the user interface of your Android app. Here's an overview of how each one works and how to use them in Java:

1. LinearLayout

LinearLayout arranges child elements either in a row (horizontal) or in a column (vertical).

- XML (LinearLayout Example):

```
<LinearLayout android:id="@+id/linearLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="First item" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />
</LinearLayout>
```



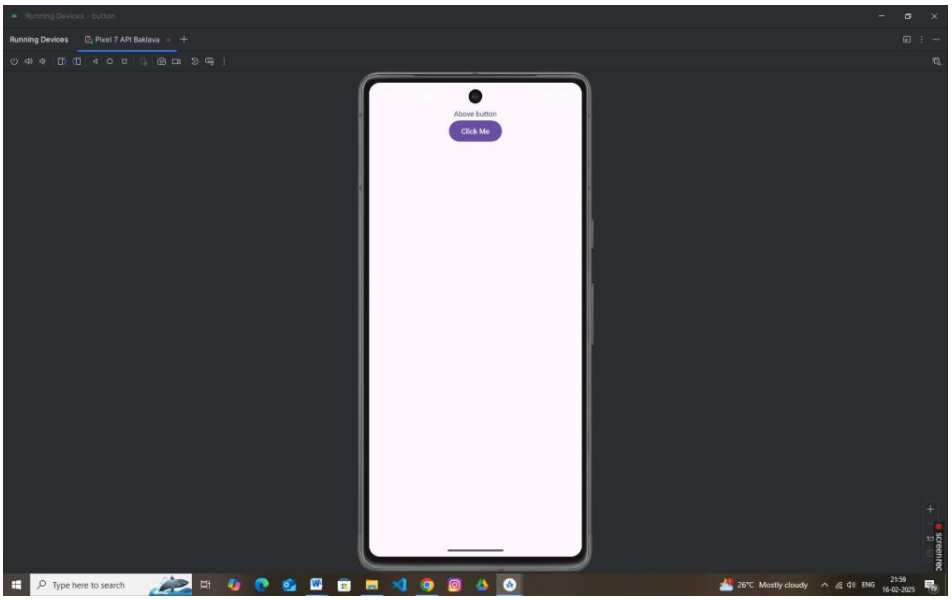
2. RelativeLayout

RelativeLayout positions elements relative to each other. You can position elements by specifying attributes like android:layout_toEndOf, android:layout_below, etc.

- XML (RelativeLayout Example):

```
<RelativeLayout android:id="@+id/relativeLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Above button"
        android:layout_centerHorizontal="true" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_below="@id/textView"
```

```
        android:layout_centerHorizontal="true"  />
</RelativeLayout>
```



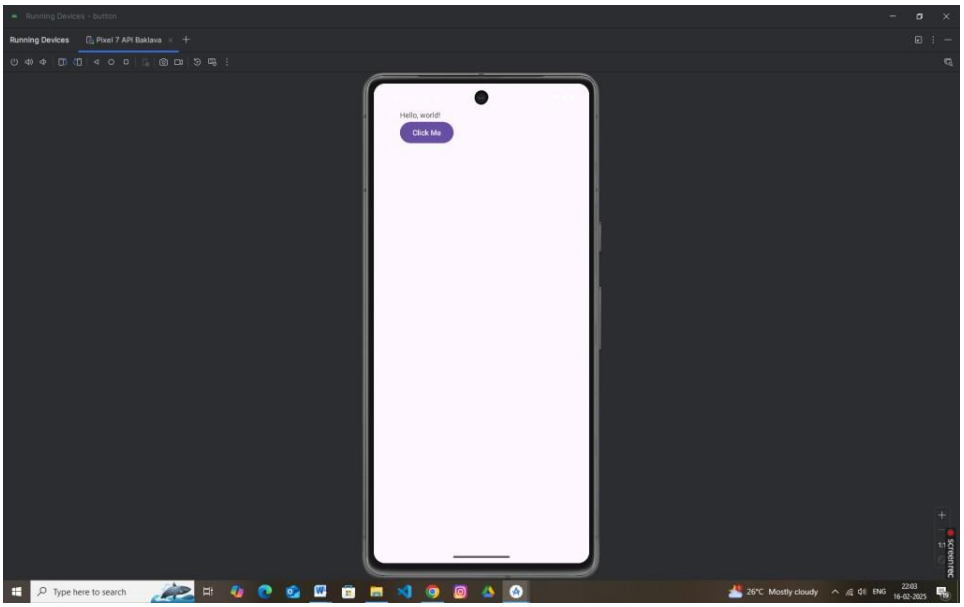
3. ConstraintLayout

ConstraintLayout is the most flexible layout container, allowing you to define complex layouts with flexible positioning and relationships between elements. It's recommended for modern UI design.

- XML (ConstraintLayout Example):

```
<androidx.constraintlayout.widget.ConstraintLayout
   android:id="@+id/constraintLayout"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <TextView android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, world!"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintStart_toStartOf="parent" />

   <Button
```

```
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Click
        Me"
        app:layout_constraintTop_toBottomOf="@id/textView"
        app:layout_constraintStart_toStartOf="parent" />

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



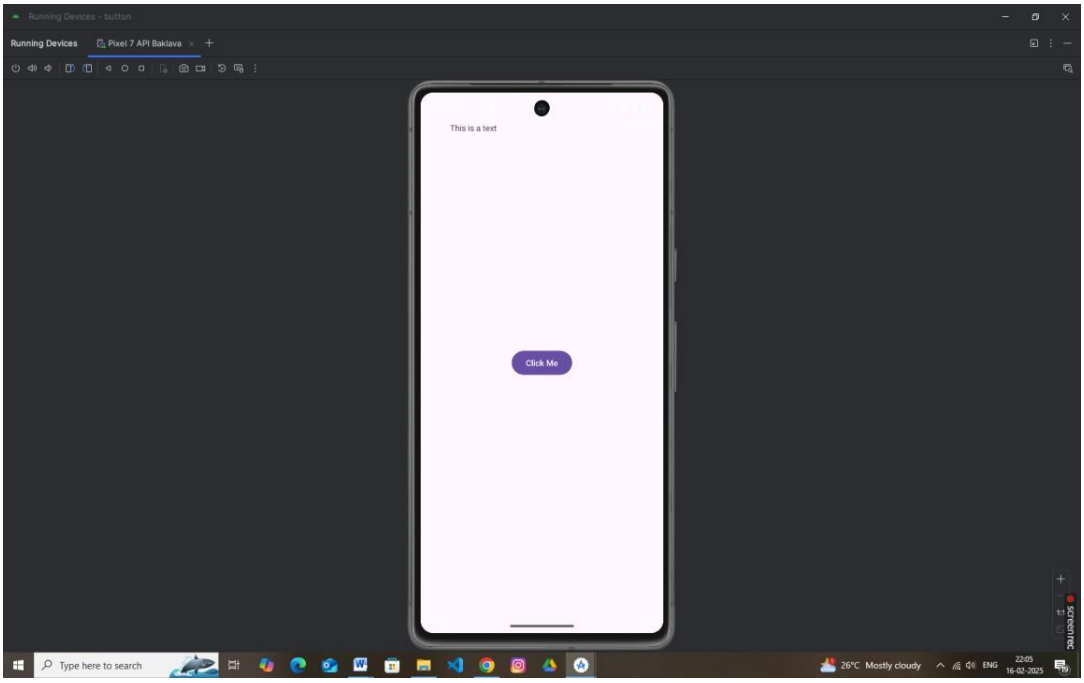
4. FrameLayout

FrameLayout is used to stack child elements on top of each other. This layout is commonly used for displaying one element at a time (like showing a fragment or image).

- XML (FrameLayout Example):

```
<FrameLayout android:id="@+id/frameLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

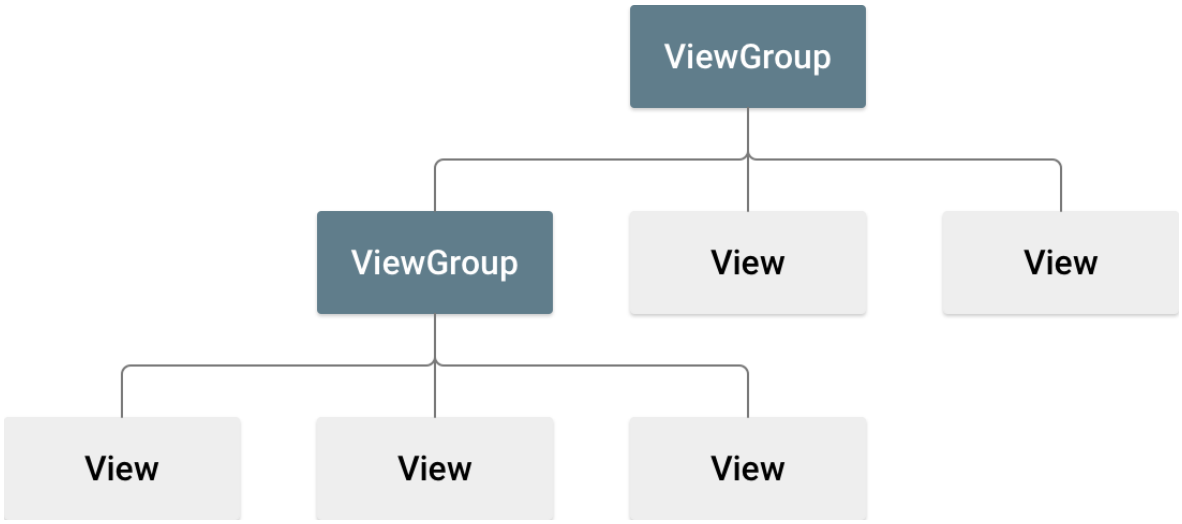
```
        android:text="This is a text" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_gravity="center" />
</FrameLayout>
```



❖ Designing User Interfaces with Layouts

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects.

A View usually draws something the user can see and interact with. A ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects



Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View ListView is a view group that displays a list of scrollable items.

7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the view.
2	android:layout_width This is the width of the layout.
3	android:layout_height This is the height of the layout
4	android:layout_marginTop This is the extra space on the top side of the layout.
5	android:layout_marginBottom This is the extra space on the bottom side of the layout.
6	android:layout_marginLeft This is the extra space on the left side of the layout.
7	android:layout_marginRight This is the extra space on the right side of the layout.
8	android:layout_gravity This specifies how child Views are positioned.
9	android:layout_weight This specifies how much of the extra space in the layout should be allocated to the View.
10	android:layout_x This specifies the x-coordinate of the layout.
11	android:layout_y This specifies the y-coordinate of the layout.
12	android:layout_width This is the width of the layout.
13	android:paddingLeft This is the left padding filled for the layout.

14	android:paddingRight This is the right padding filled for the layout.
15	android:paddingTop This is the top padding filled for the layout.
16	android:paddingBottom This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height –

- android:layout_width=wrap_content tells your view to size itself to the dimensions required by its content.
- android:layout_width=fill_parent tells your view to become as big as its parent view.

View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

android:id="@+id/my_button"

Following is a brief description of @ and + signs –

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources.

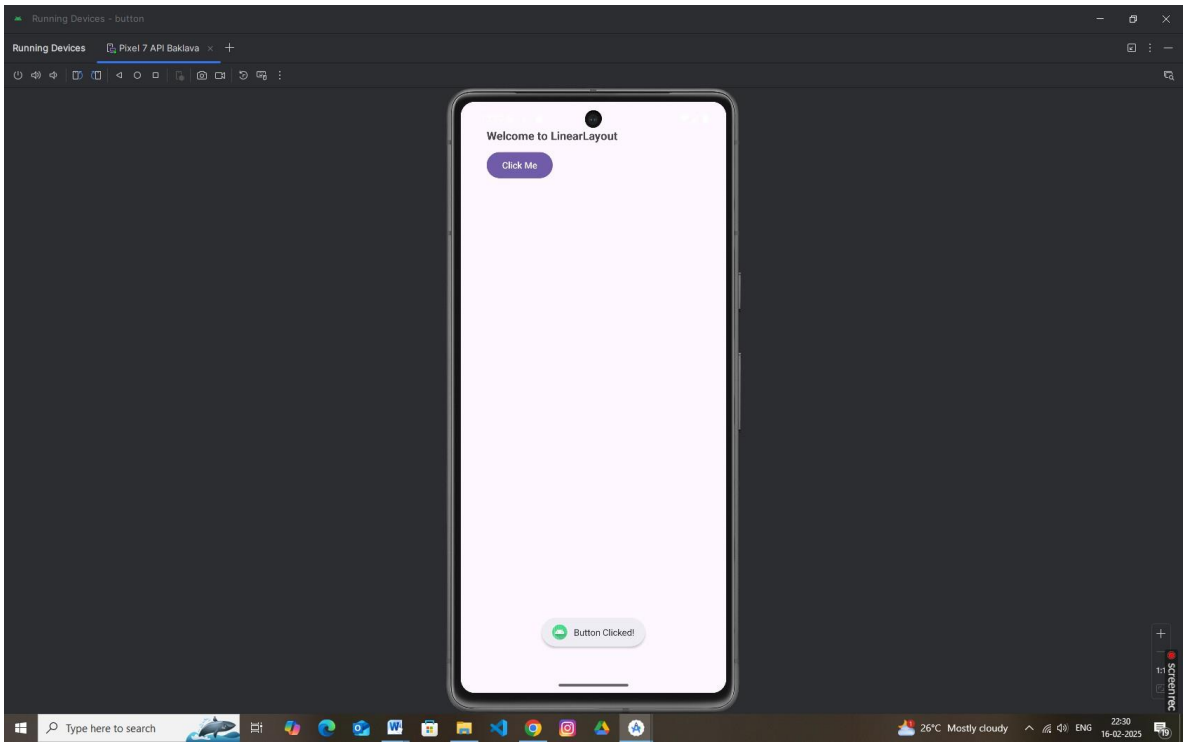
MainActivity.java

```
package com.example.button;

import android.os.Bundle;
```

```
import android.widget.Button;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button = findViewById(R.id.button);
        button.setOnClickListener(v ->
                Toast.makeText(this, "Button Clicked!",
Toast.LENGTH_SHORT).show()
        );
    }
}
```

activity_main.xml

```
<?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"
    android:padding="40dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to LinearLayout"
        android:textSize="18sp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_marginTop="10dp"/>

</LinearLayout>
```



❖ Animation in Android with Example

Animation is the process of adding a motion effect to any view, image, or text. With the help of an animation, you can add motion or can change the shape of a specific view. Animation in Android is generally used to give your UI a rich look and feel.

Here's example—this time using a **slide-in animation** that makes a view slide in from the left.

Step 1: Create a Slide-In Animation Resource

Create a new XML file called slide_in_left.xml in your project's res/anim folder:

```
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="-100%"
    android:toXDelta="0%"
    android:duration="1000" />
```

Explanation:

- fromXDelta="-100%": The view starts off-screen to the left.
- toXDelta="0%": The view ends at its natural position.
- duration="100": The animation lasts for 500 milliseconds.

Step 2: Create a Layout with a View

For this example, let's create a layout with a Button that will slide in from the left.

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16sp"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/slideButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Slide In" />

</LinearLayout>
```

Step 3: Load and Start the Animation in Your Activity

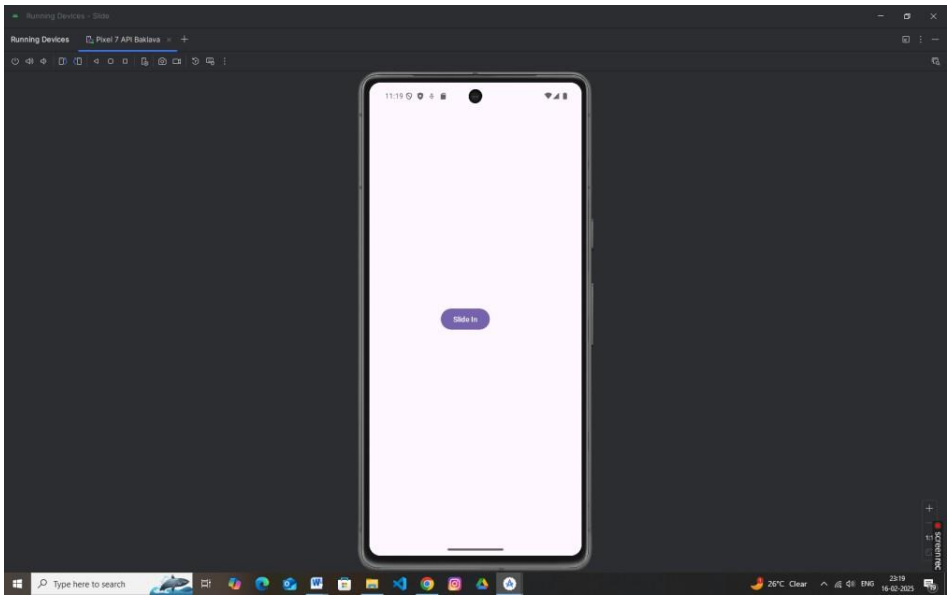
In your MainActivity.java, load the slide-in animation using AnimationUtils and apply it to the button.

MainActivity.java

```
package com.example.slide;

import android.os.Bundle;
import android.view.animation.Animation;
import
android.view.animation.AnimationUtils;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity extends AppCompatActivity {

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

        // Find the button by its ID
        Button slideButton = findViewById(R.id.slideButton);

        // Load the slide-in animation from the resource
        Animation slideAnimation =
AnimationUtils.loadAnimation(this,R.anim.slide_in_left);

        // Start the animation when the activity starts
        slideButton.startAnimation(slideAnimation);
    }
}
```

Explanation:

- AnimationUtils.loadAnimation(this, R.anim.slide_in_left): Loads the slide-in animation from the XML resource.
- slideButton.startAnimation(slideInAnimation): Applies the animation to the Button.



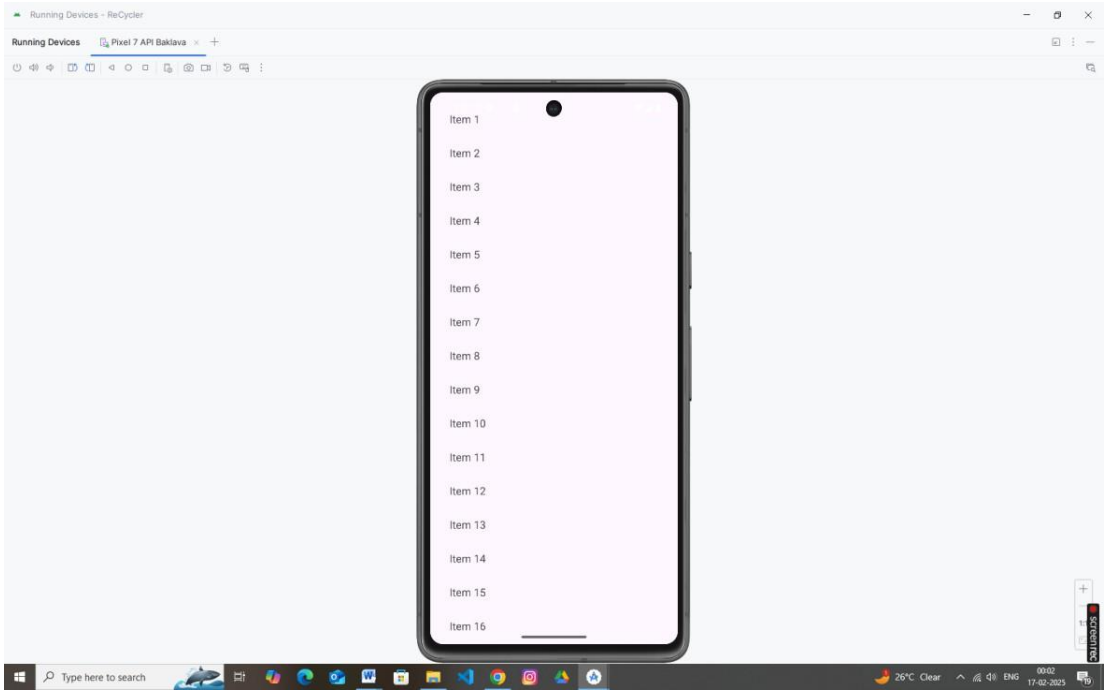
❖ RecyclerView

RecyclerView makes it easy to efficiently display large sets of data. You supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed.

As the name implies, RecyclerView recycles those individual elements. When an item scrolls off the screen, RecyclerView doesn't destroy its view. Instead, RecyclerView reuses the view for new items that have scrolled onscreen. RecyclerView improves performance and your app's responsiveness, and it reduces power consumption.

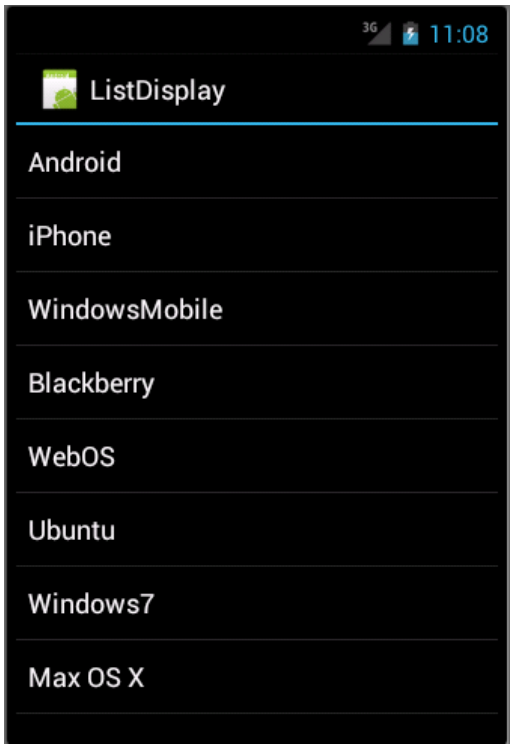
- RecyclerView is the ViewGroup that contains the views corresponding to your data. It's a view itself, so you add RecyclerView to your layout the way you would add any other UI element.
- Each individual element in the list is defined by a view holder object. When the view holder is created, it doesn't have any data associated with it. After the view holder is created, the RecyclerView binds it to its data. You define the view holder by extending RecyclerView.ViewHolder.
- The RecyclerView requests views, and binds the views to their data, by calling methods in the adapter. You define the adapter by extending RecyclerView.Adapter.

- The layout manager arranges the individual elements in your list. You can use one of the layout managers provided by the RecyclerView library, or you can define your own. Layout managers are all based on the library's LayoutManager abstract class.



❖ Android ListView

Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.



List View

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are ArrayAdapter,Base
Adapter, CursorAdapter, SimpleCursorAdapter,SpinnerAdapter and WrapperListAdapter. We will see separate examples for both the adapters.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView. Consider you have an array of strings you want to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array −

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

Here are arguments for this constructor −

- First argument this is the application context. Most of the case, keep it this.
- Second argument will be layout defined in XML file and having TextView for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

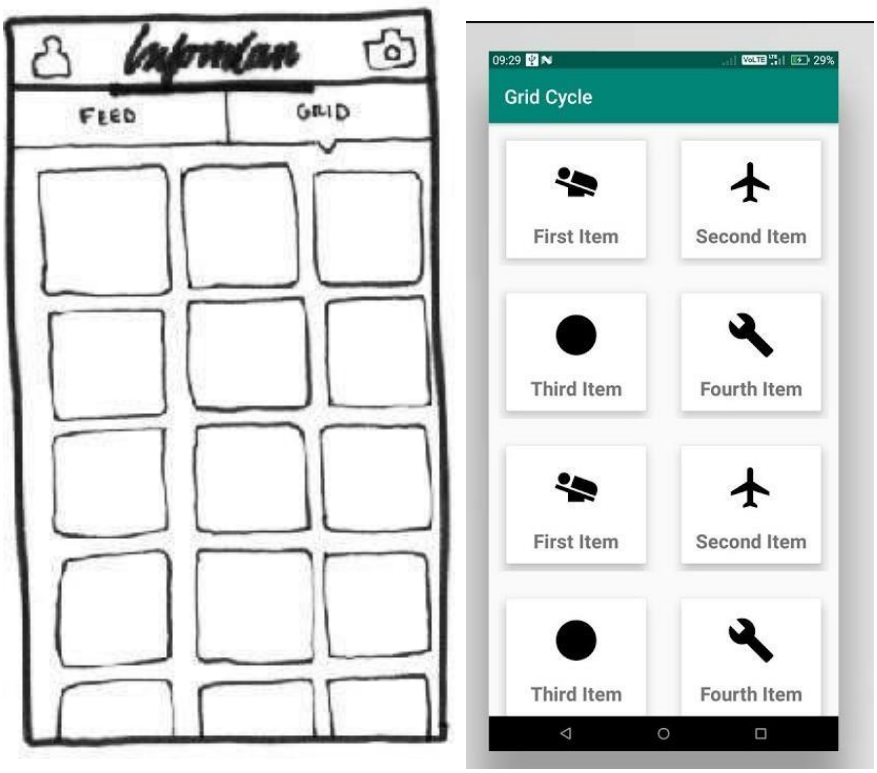
Once you have array adapter created, then simply call setAdapter() on your ListView object as follows −

```
ListView listView = (ListView) findViewById(R.id.listview); listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

❖ Android GridView

Android GridView shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a ListAdapter



An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

GridView Attributes

Following are the important attributes specific to GridView –

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:columnWidth This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
3	android:gravity Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	android:horizontalSpacing Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
5	android:numColumns Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.

❖ Input Controls

Below is a comprehensive guide to various input controls and UI components in Android. Each section includes an explanation and a simple code example. You can mix and match these concepts to build robust and interactive user interfaces.

1. Buttons

In Android, Button represents a push button. A Push buttons can be clicked, or pressed by the user to perform an action. There are different types of buttons used in android such as CompoundButton, ToggleButton, RadioButton.

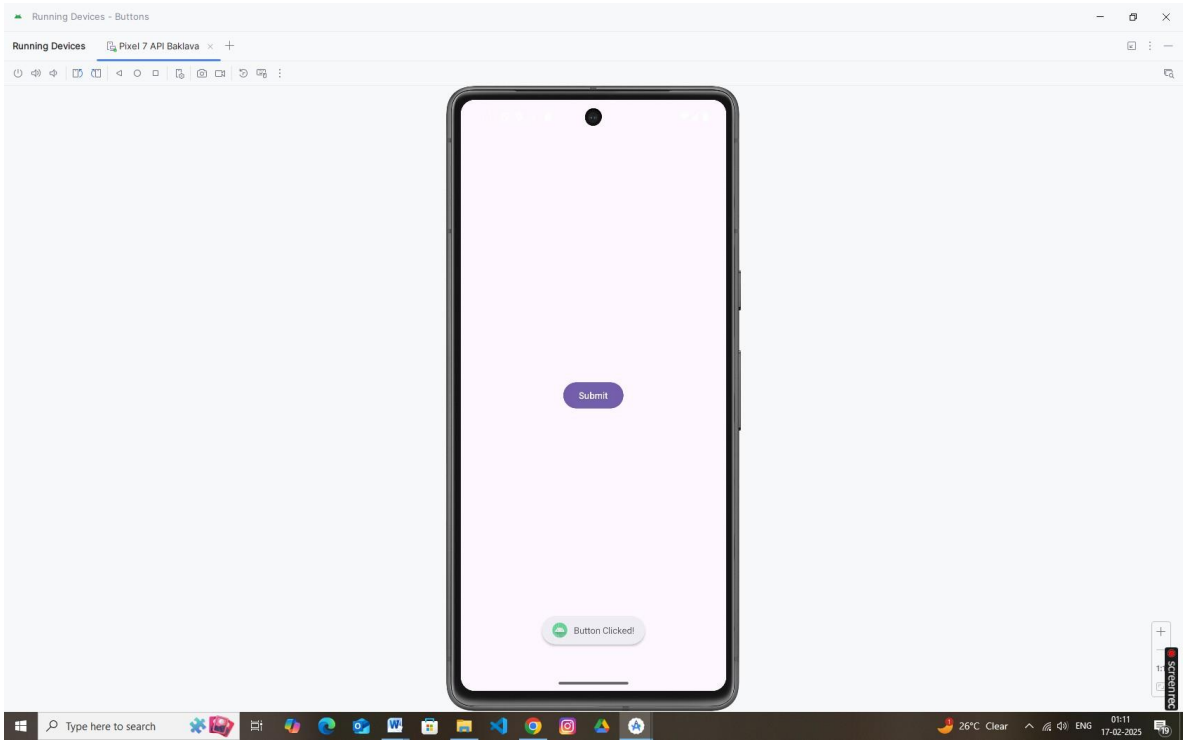
Button is a subclass of TextView class and compound button is the subclass of Button class. On a button we can perform different actions or events like click event, pressed event, touch event etc.

XML Example:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit" />


</LinearLayout>
```

Java Example:

```
package com.example.buttons;

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

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnSubmit = findViewById(R.id.btnSubmit);
        btnSubmit.setOnClickListener(v ->
                Toast.makeText(this, "Button Clicked!",
Toast.LENGTH_SHORT).show()
        );

    }
}
```



2. Checkboxes

In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBox class.

In android there is a lot of usage of check box. For example, to take survey in Android app we can list few options and allow user to choose using CheckBox. The user will simply checked these checkboxes rather than type their own option in EditText. Another very common use of CheckBox is as remember me option in Login form.

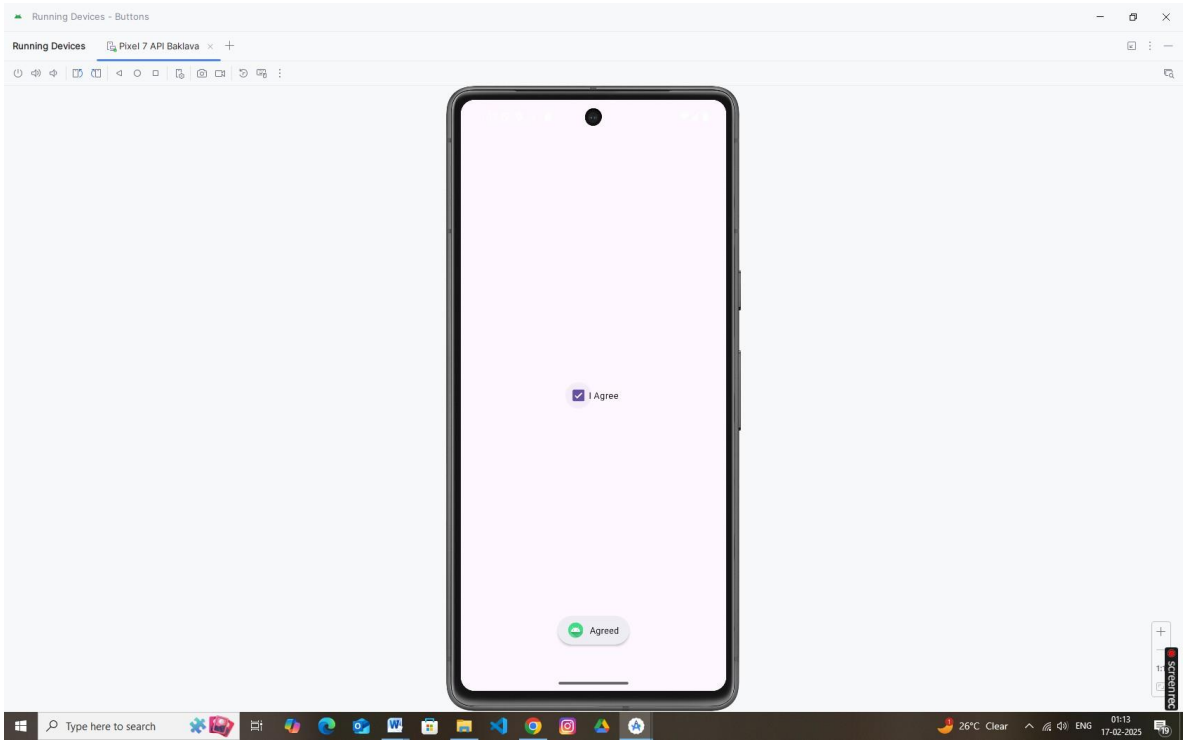
XML Example:

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

```
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <CheckBox
        android:id="@+id/checkboxAgree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I Agree" />


</LinearLayout>
```

Java Example:

```
package com.example.buttons;

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

public class MainActivity extends AppCompatActivity {

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

        CheckBox checkboxAgree = findViewById(R.id.checkboxAgree);
        checkboxAgree.setOnCheckedChangeListener((buttonView, isChecked) -> {
            String msg = isChecked ? "Agreed" : "Not Agreed";
            Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
        });


    }
}
```



3. Radio Buttons

In Android, RadioButton are mainly used together in a RadioGroup.
In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group.
The most common use of radio button is in Quiz Android App code.

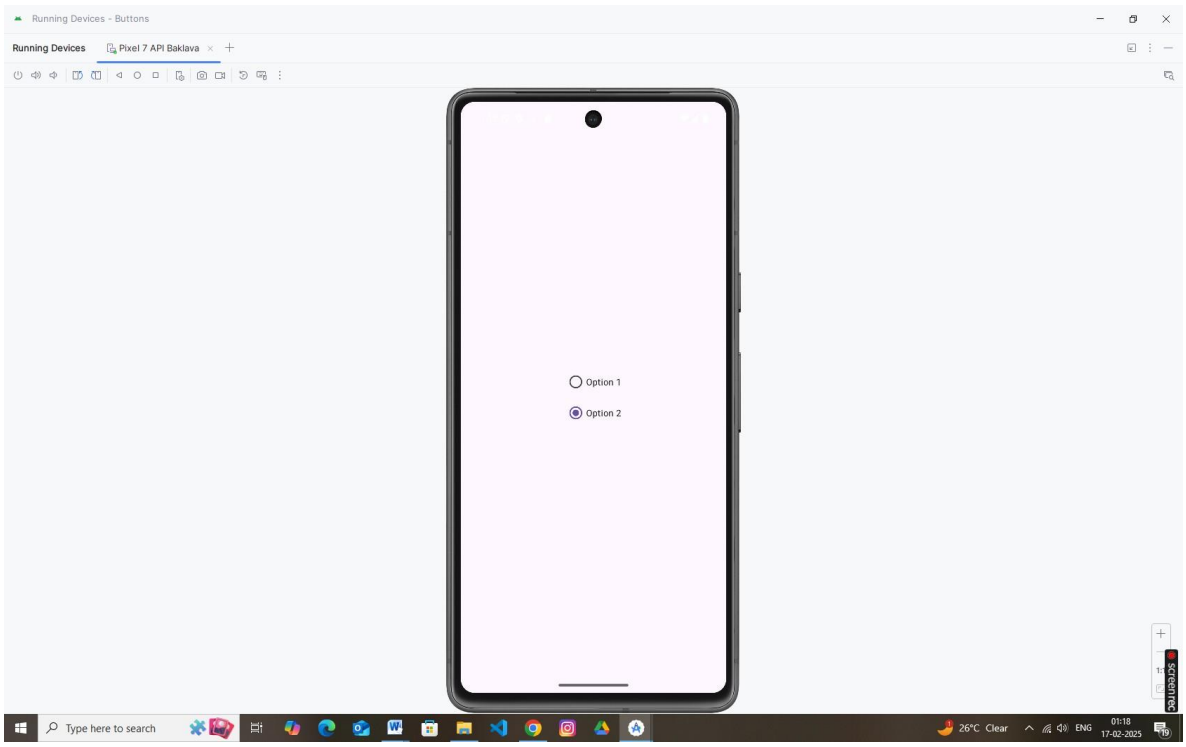
XML Example (with RadioGroup):

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
```

```
        <RadioButton
            android:id="@+id/radioOption1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Option 1" />

        <RadioButton
            android:id="@+id/radioOption2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Option 2" />
    </RadioGroup>



</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        RadioGroup radioGroup = findViewById(R.id.radioGroup);
        radioGroup.setOnCheckedChangeListener((group, checkedId) -> {
            RadioButton radioButton = findViewById(checkedId);
            Toast.makeText(this, "Selected: " + radioButton.getText(),
Toast.LENGTH_SHORT).show();
        });


    }
}
```



4. Toggle Buttons

In Android, ToggleButton is used to display checked and unchecked state of a button. ToggleButton basically an off/on button with a light indicator which indicate the current state of toggle button. The most simple example of ToggleButton is doing on/off in sound, Bluetooth, wifi, hotspot etc. It is a subclass of compoundButton.

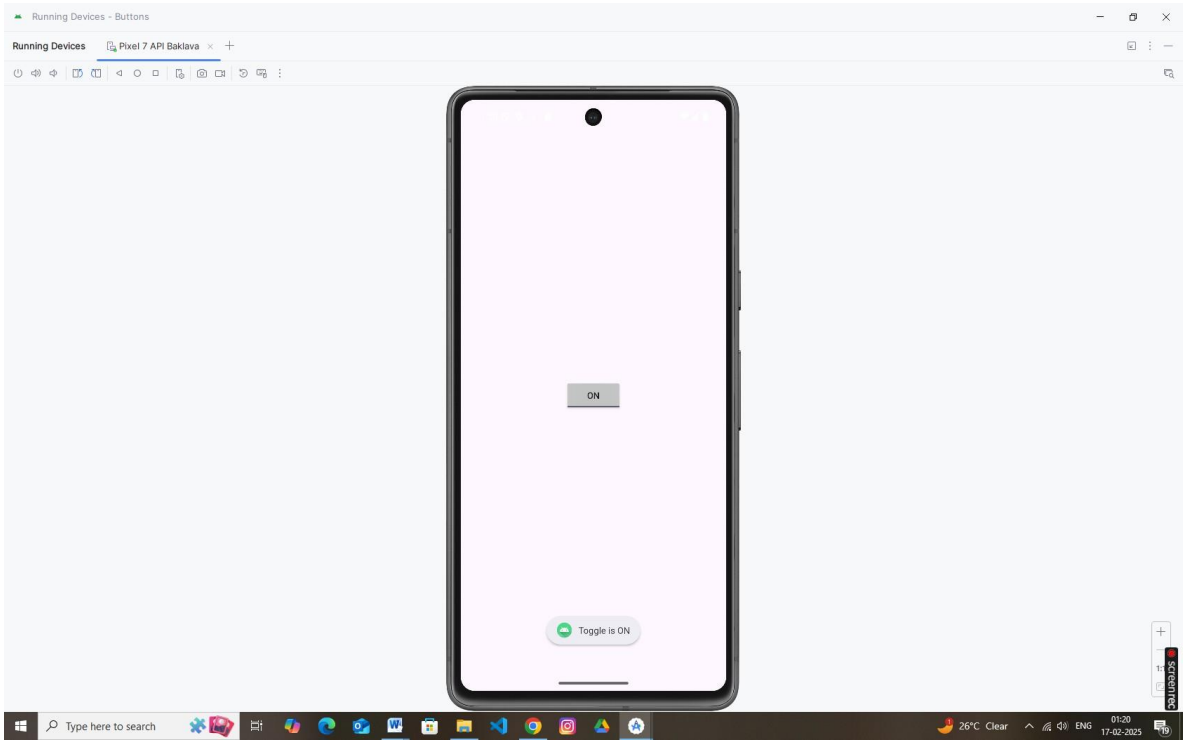
XML Example:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="ON"
        android:textOff="OFF" />
```

```
</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
import android.widget.ToggleButton;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        ToggleButton toggleButton = findViewById(R.id.toggleButton);
        toggleButton.setOnCheckedChangeListener((buttonView, isChecked) -> {
            String status = isChecked ? "Toggle is ON" : "Toggle is OFF";
            Toast.makeText(this, status, Toast.LENGTH_SHORT).show();
        });


    }
}
```



5. Spinners

In Android, Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list seen in other programming languages. In a default state, a spinner shows its currently selected value. It provides a easy way to select a value from a list of values.

In Simple Words we can say that a spinner is like a combo box of AWT or swing where we can select a particular item from a list of items. Spinner is a sub class of AsbSpinner class.

Important Note: Spinner is associated with Adapter view so to fill the data in spinner we need to use one of the Adapter class.

XML Example:

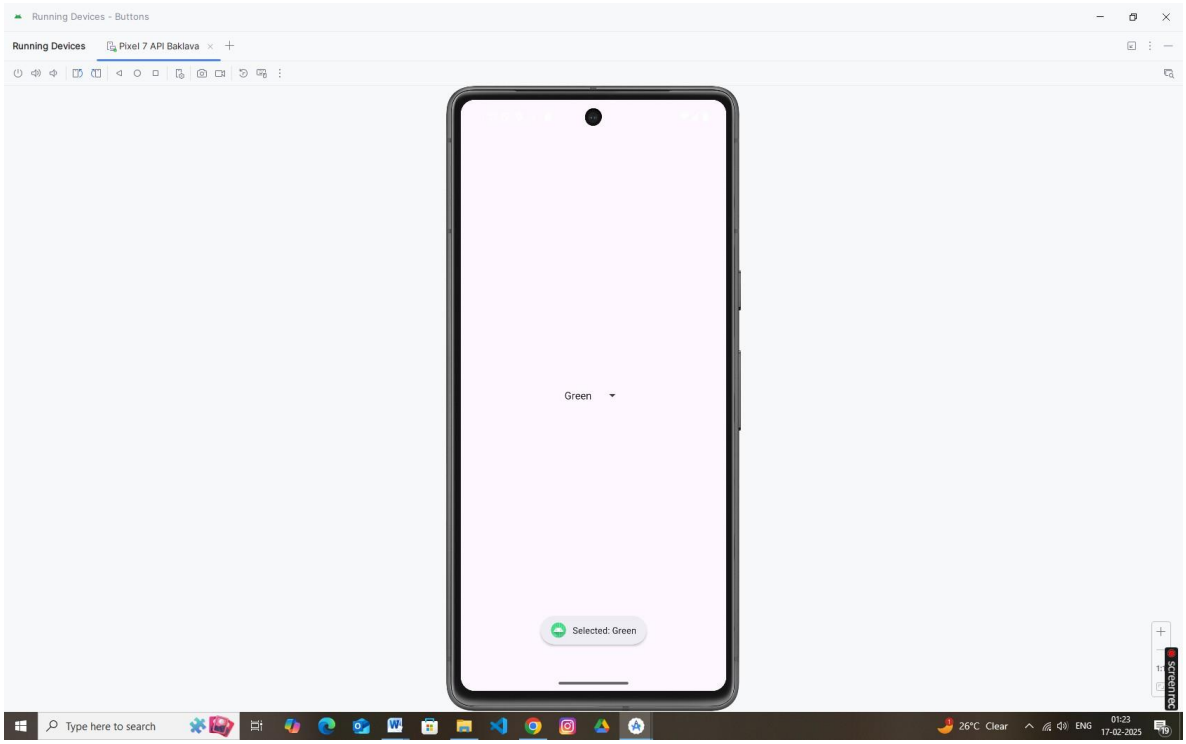
```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <Spinner
        android:id="@+id/spinnerOptions"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />


</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.view.View;
import
android.widget.AdapterView;
import
android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        Spinner spinner = findViewById(R.id.spinnerOptions);
        String[] options = {"Red", "Green", "Blue"};
        ArrayAdapter<String> adapter = new
        ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, options);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
        spinner.setAdapter(adapter);

        spinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
int position, long id) {
                Toast.makeText(getApplicationContext(), "Selected: " +
```

```
options[position], Toast.LENGTH_SHORT).show();
                }
                @Override
                public void onNothingSelected(AdapterView<?> parent) {}
            });


    }
}
```



6. Input Events

In android, Input Events are used to capture the events, such as button clicks, edittext touch, etc. from the View objects that defined in a user interface of our application, when the user interacts with it.

To handle input events in android, the views must have in place an event listener. The View class, from which all UI components are derived contains a wide range event listener interfaces and each listener interface contains an abstract declaration for a

callback method. To respond to an event of a particular type, the view must register an appropriate event listener and implement the corresponding callback method.

For example, if a button is to respond to a click event it must register to View.onClickListener event listener and implement the corresponding onClick() callback method. In application when a button click event is detected, the Android framework will call the onClick() method of that particular view.

Generally, to handle input events we use Event Listeners and Event Handling in android applications to listen for user interactions and to extend a View class, in order to build a custom component.

Android Event Listeners

In android, Event Listener is an interface in the View class that contains a single call-back method. These methods will be called by the Android framework when the View which is registered with the listener is triggered by user interaction with the item in UI.

The following are the call-back methods included in the event listener interface.

Method	Description
onClick()	This method is called when the user touches or focuses on the item using navigation-keys or trackball or presses on the "enter" key or presses down on the trackball.
onLongClick()	This method is called when the user touches and holds the item or focuses on the item using navigation-keys or trackball and presses and holds "enter" key or presses and holds down on the trackball (for one second).
onFocusChange()	This method is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

Method	Description
onKey()	This method is called when the user is focused on the item and presses or releases a hardware key on the device.
onTouch()	This method is called when the user performs a touch event, including a press, a release, or any movement gesture on the screen.
onCreateContextMenu()	This method is called when a Context Menu is being built.

Example (Button Click):

```
Button btnAction = findViewById(R.id.btnSubmit);
btnAction.setOnClickListener(v -> {
    // Handle click event here
    Toast.makeText(this, "Button was clicked!", Toast.LENGTH_SHORT).show();
});
```

You can also override other event methods (like onTouchEvent or onKeyDown) in your activity for more granular control.

7. Menus

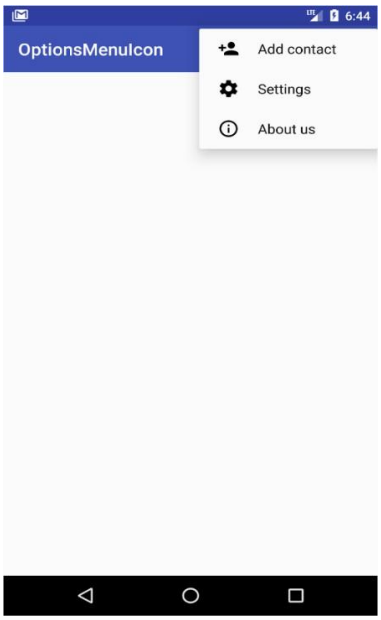
The menu is a part of the User Interface (UI) component, used to handle some common functionality around the app. To utilize the menu, you should define it in a separate XML file and use that file in your app based on your requirements. You can also use menu APIs to represent user actions and other options in your app activities.

Different types of menus

Android provides three types of menus. They are as follows:

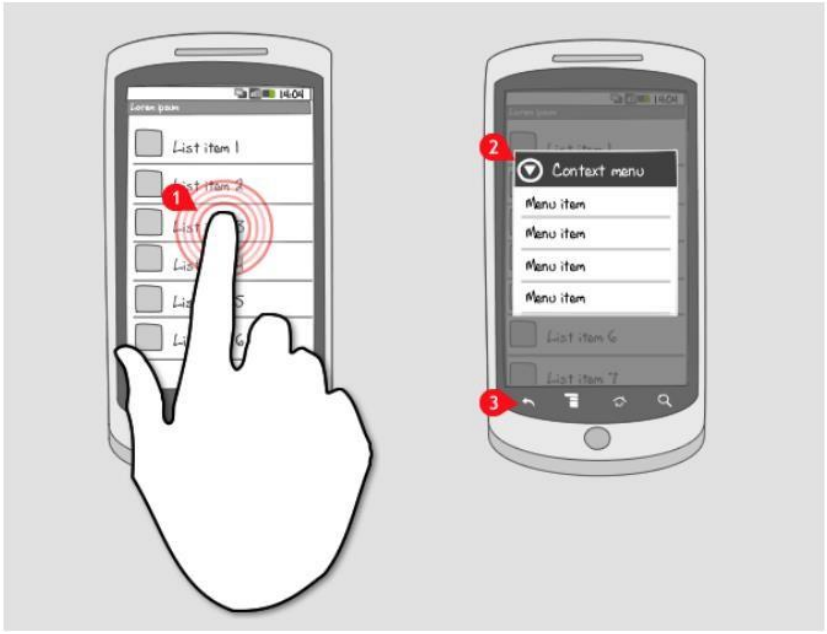
i. Option Menu

This type of menu is a primary collection of menu items in an app and is useful for actions that have a global impact on the searching app. The Option Menu can be used for settings, searching, deleting items, sharing, etc.



ii. Context Menu

This type of menu is a floating menu that only appears when a user presses for a long time on an element and is useful for elements that affect the selected content or context frame.



iii. Popup Menu

Using Popup Menu we can display a list of items in a vertical list which presents the view that invokes the menu. Popup Menu is useful since it can provide an overflow of actions which are related to any specific content.

To define a menu in the XML file
In this step, we will write the menu's code in an XML format to define the type of menu and its items. First, you should create a new menu folder inside of your project resource folder (res/menu) to define the menu. Add a new XML file (res/menu/file.xml) to build your menu. This XML (res/menu/file.xml) file can be given any name that you provide.
There are the following important elements of a menu:

1. <menu>: A <menu> element defines a menu, which is a container for menu items that holds one or more elements. It must be the root of a file.
2. <items>: This element is used to create items in the menu. An <items> element can contain a nested <menu> element to create a submenu.
3. <group>: This element is an optional, invisible container for <item> elements. <group> allows categorizing menu items so they share properties such as active state and visibility.

8. Toast

In android, Toast is a small popup notification that is used to display an information about the operation which we performed in our app. The Toast will show the message for a small period of time and it will disappear automatically after a timeout.

Generally, the size of Toast will be adjusted based on the space required for the message and it will be displayed on the top of the main content of activity for a short period of time.

For example, some of the apps will show a message like "Press again to exit" in toast, when we pressed a back button on the home page or showing a message like "saved successfully" toast when we click on the button to save the details.

If you observe above syntax, we defined a Toast notification using makeText() method with three parameters, those are

Parameter	Description
context	It's our application context.
message	It's our custom message which we want to show in Toast notification.
duration	It is used to define the duration for notification to display on the screen.

We have two ways to define the Toast duration, either
in LENGTH_SHORT or LENGTH_LONG to display the toast notification for a short or longer period of time.

```
Toast.makeText(getApplicationContext(), "This is a Toast message",
Toast.LENGTH_SHORT).show();
```

9. Dialogs

Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

Android AlertDialog is composed of three regions: title, content area and action

buttons. Android AlertDialog is the subclass of Dialog class.

Example: AlertDialog

```
new AlertDialog.Builder(this)
        .setTitle("Exit")
        .setMessage("Are you sure you want to exit?")
        .setPositiveButton("Yes", (dialog, which) -> finish())
        .setNegativeButton("No", null)
        .show();
```

10. Styles and Themes

Styles and themes on Android let you separate the details of your app design from the UI structure and behavior, similar to stylesheets in web design.

A style is a collection of attributes that specifies the appearance for a single View. A style can specify attributes such as font color, font size, background color, and much more.

A theme is a collection of attributes that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply a theme, every view in the app or activity applies each of the theme's attributes that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Define a Style (res/values/styles.xml):

```
<resources>
    <style name="CustomButtonStyle">
        <item name="android:background">#FF5722</item>
        <item name="android:textColor">#FFFFFF</item>
        <item name="android:padding">10dp</item>
    </style>
</resources>
```

Apply Style in XML:

```
<Button
    style="@style/CustomButtonStyle"
    android:id="@+id/styledButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Styled Button" />
```

Themes:
Themes are applied at the app or activity level (see the AndroidManifest.xml or your styles.xml for your app theme).

11. Creating Lists (Using ListView)

Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

XML Layout (res/layout/activity_list.xml):

```
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Java Example:

```
ListView listView = findViewById(R.id.listView);
String[] items = {"Apple", "Banana", "Cherry", "Date", "Elderberry"};
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
        android.R.layout.simple_list_item_1, items);
listView.setAdapter(adapter);
```

12. Custom Lists (Using Custom Adapter for ListView)

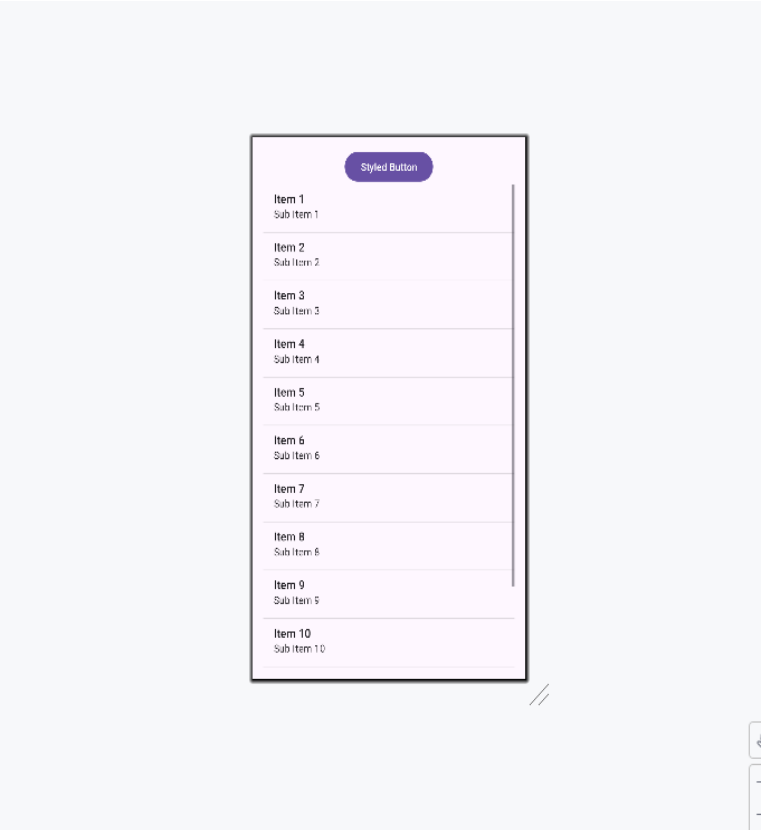
In this tutorial we'll use a CustomAdapter that populates the custom rows of the Android ListView with an ArrayList. Also to enhance the user experience, we'll animate the ListView while scrolling.

Android ListView Custom Adapter Overview

The simplest Adapter to populate a view from an ArrayList is the ArrayAdapter. That's what we'll implement in this tutorial. There are other adapters as well, such as the CursorAdapter which binds directly to a result set from a Local SQLite Database and it uses a Cursor as it's data source.

Custom Row Layout (res/layout/custom_row.xml):

```
<Button
    style="@style/CustomButtonStyle"
    android:id="@+id/styledButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Styled Button" />


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


<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">

    <ImageView
        android:id="@+id/imgIcon"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:src="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="10dp"
        android:textSize="18sp" />
</LinearLayout>
```

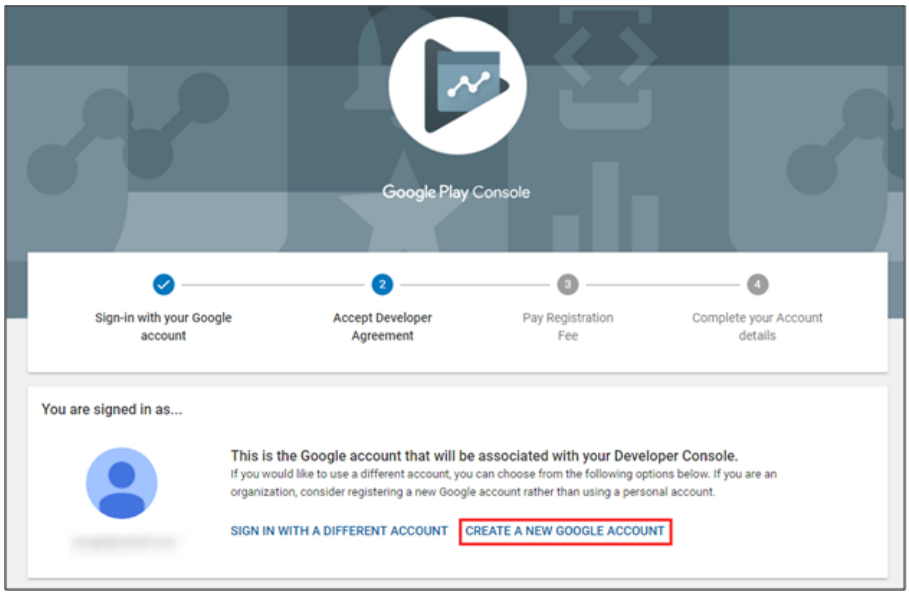

Styled Button
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Item 8
Sub Item 8
Item 9
Sub Item 9
Item 10
Sub Item 10

UNIT -4

❖ Publish Android App on the Play Store

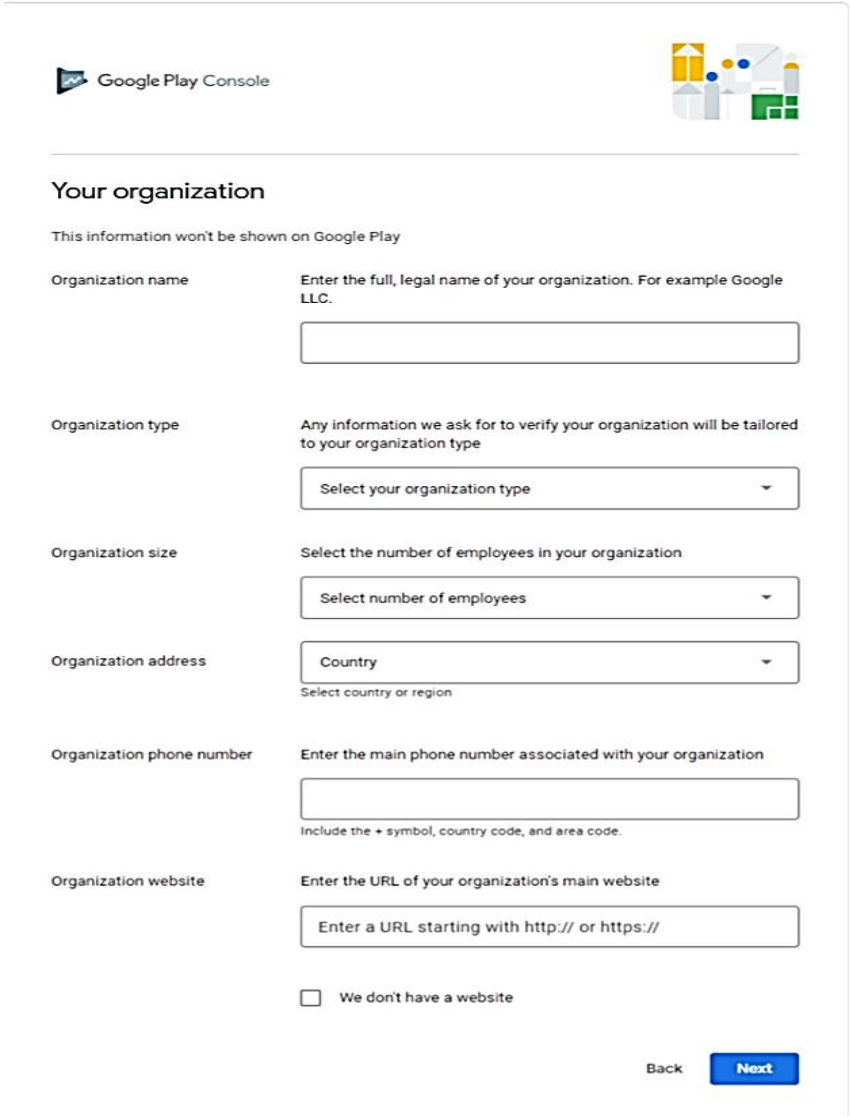
Step 1 – Google Play Console Developer Account.

One of the primary steps of publishing an Android application on the Play Store is creating a Google developer account on the Google Play Console. Google Play Console can be understood as a backend operating system for all the apps published on the Play Store. Developers who intend to publish an app on the Play Store must create a developer account on the console and pay a one-time fee of $25, payable using a credit card or through online banking methods. Also, remember that post submission, the developer account can take up to 48-hours to activate.



.

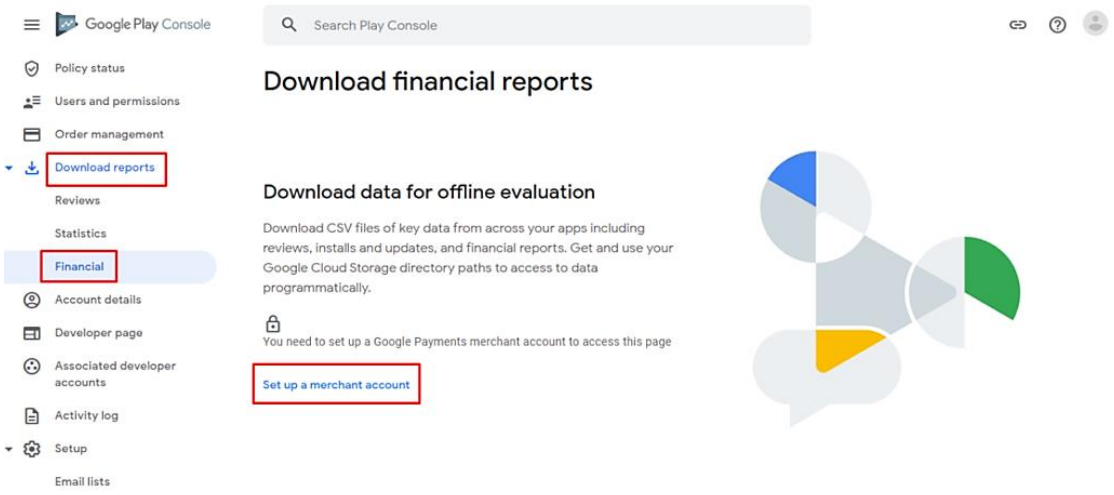
Finally, ensure you fill out all the credentials asked while creating the account.



Step 2 – Set-up up a Google Merchant Account.

If your app involves in-app purchases, the next essential step is to link your developer account with your merchant account. If you already have an existing merchant account, you can navigate to Download reports and select Financial. However, to access this page, you must create a Google Merchant Account.

And to create one, click on setup up a merchant account.

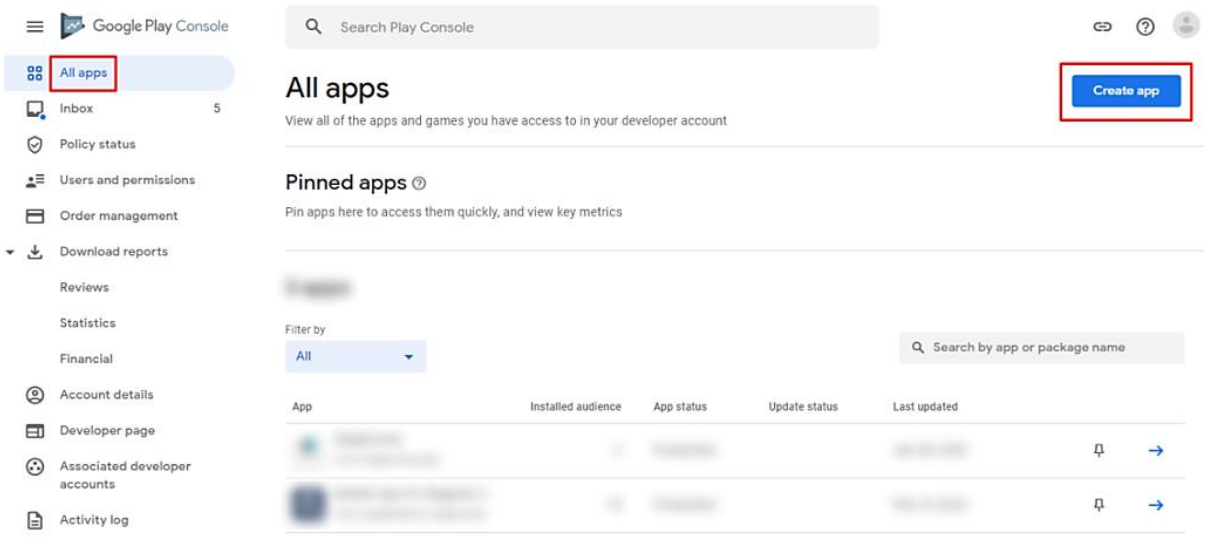


Once you create your new merchant account, it will automatically get linked to your Google Play Console account and enable you to monitor, manage, and analyze the app sales and generate reports.

Step 3 – Create Application

Once the merchant account is linked to your Google Play Console, the next step is to create an application. And for creating an application, there are a few essential steps that you need to follow:

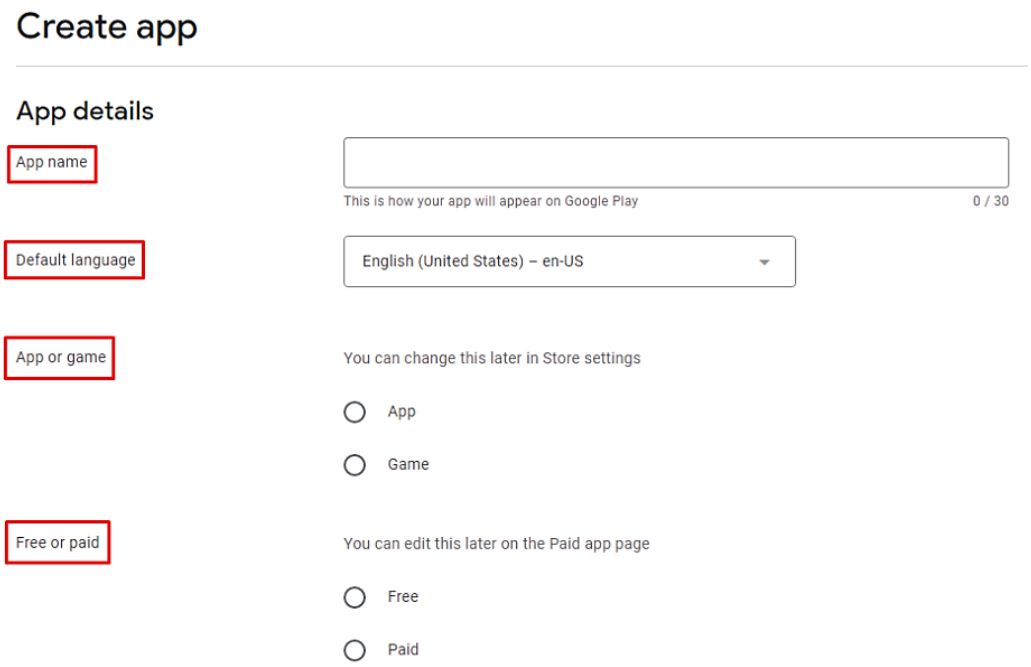
- Click on – Menu > All applications
- Select the 'Create Application' option.



Next, the play console will ask you to enter some basic app details. For instance,

- App Name – You must enter a 30-character long name in this field which will be displayed on the Google Play Console. However, this app name can be changed afterward.

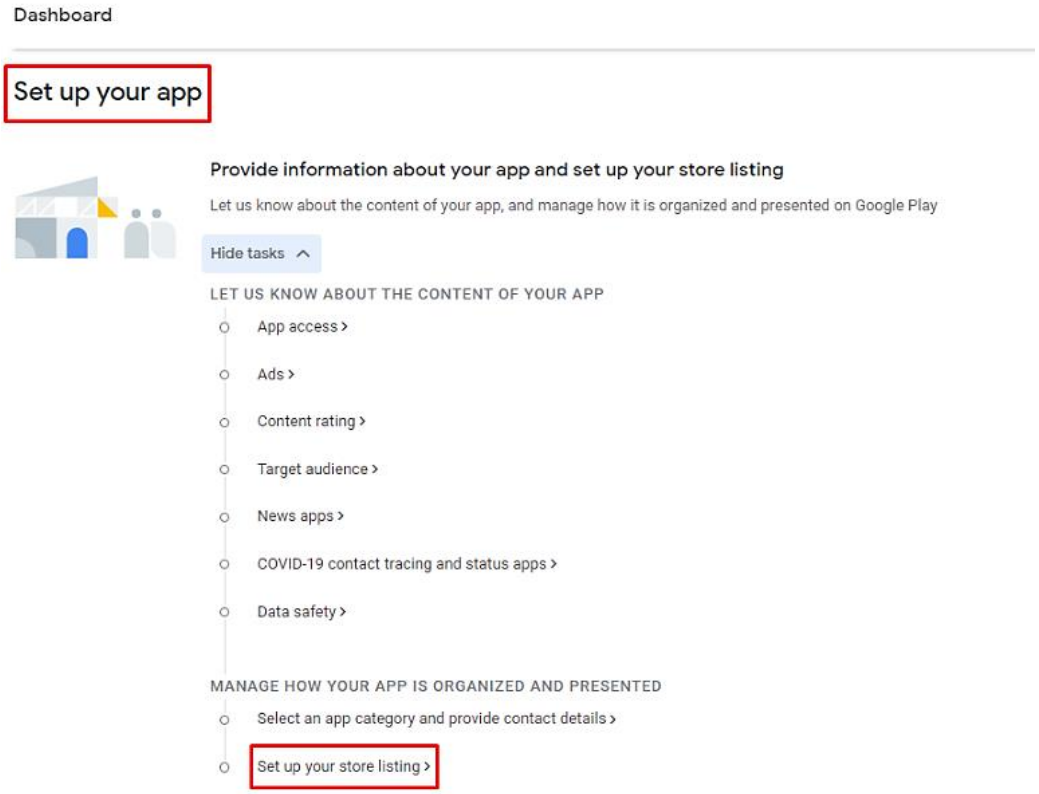
- Default language – Another essential field is the setup of the app language. You can navigate to the drop-down menu and set a default language for your app.
- App or game – The next step is to define whether you upload an app or a game, but this can again be revised afterward from the store settings.
- Free or paid – Define whether your app will be available free of cost or will require the user to pay for it. The free or paid section can be updated from the Paid app page later, but only until you publish your app. Once the app is live, you cannot transform your app from free to paid.



Once all of the above information is filled and verified, the Google Play Console will enquire for affirmations from you. Ensure that your app matches the Google policies of the Developer Program and Accepts US export laws. As soon as you agree to the terms and conditions, click **Create App**.

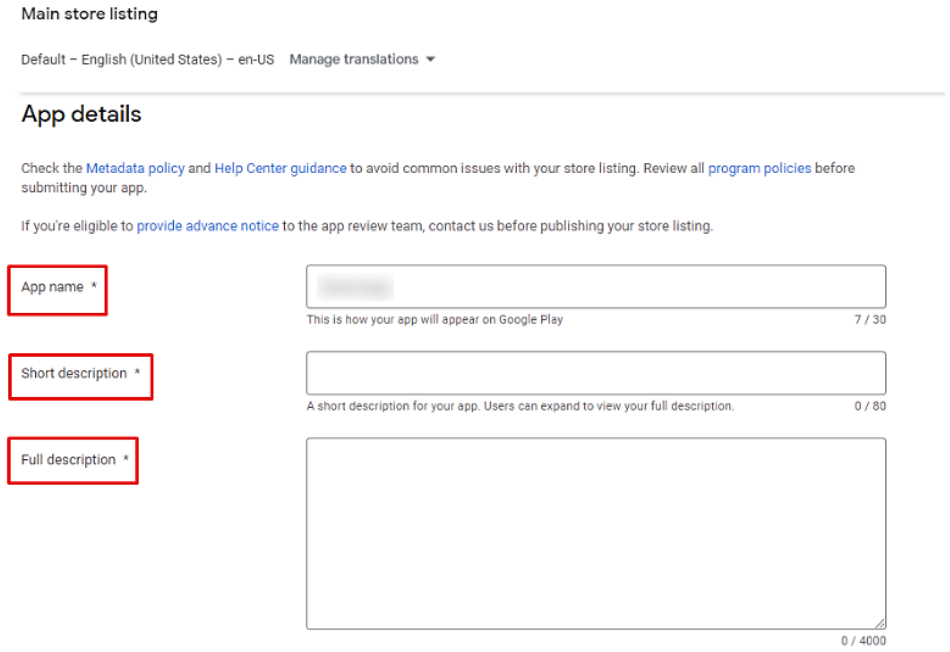
Step 4. App Store Listing Details

The next step to uploading the app on the Google Play console is filling in the essential information regarding the application listing. Once you click on the 'Create App,' the Play Console will automatically take you to a consolidated dashboard wherein you'll have to enter the necessary details to set up your app.



The Google Play Console will enquire about the following:

- App name – As you have already entered the app name in the previous step, you need not enter the same name again; however, if you wish to revise the title, this is where you can change the name.
- Short description – This field requires you to enter an 80-character-long description that best describes your app.
- Full description – The following field to the short description enables you to explain your app in detail. You can expand the word limit to 4000 characters and leverage your targeted keywords to lead Google to share your app with the relevant audience.



Once all this information is added to the Google console, the next step is adding the app graphics, category of the app, and the privacy policy. Remember, we asked you to keep high-quality images ready before beginning the app publishing process; this is precisely where all those images will be leveraged.

Further, here are the details you would require:

Particulars	Details
Screenshots	-2 to 8 in number, JPG, or PNG. The ratio shouldn't exceed 2:1
Icon	– 512 X 512– PNG– Maximum file size: 1024KB
Localization	-If your app comes in several languages, you need to mention them and add additional translations of your app's information to appeal to the users coming and checking out your app.
Application type and categorization	– Navigate to the Drop-down menu and select application type – game or app.- Pick a category suitable for your app- Rate your content after uploading your APK.
Contact details	-Provide the necessary contact forms so users can contact you.
Privacy Policy	-To escape the breach of app privacy, Google mandates adding a privacy policy while publishing your app. -If you need a break, click Save Draft and complete it later.

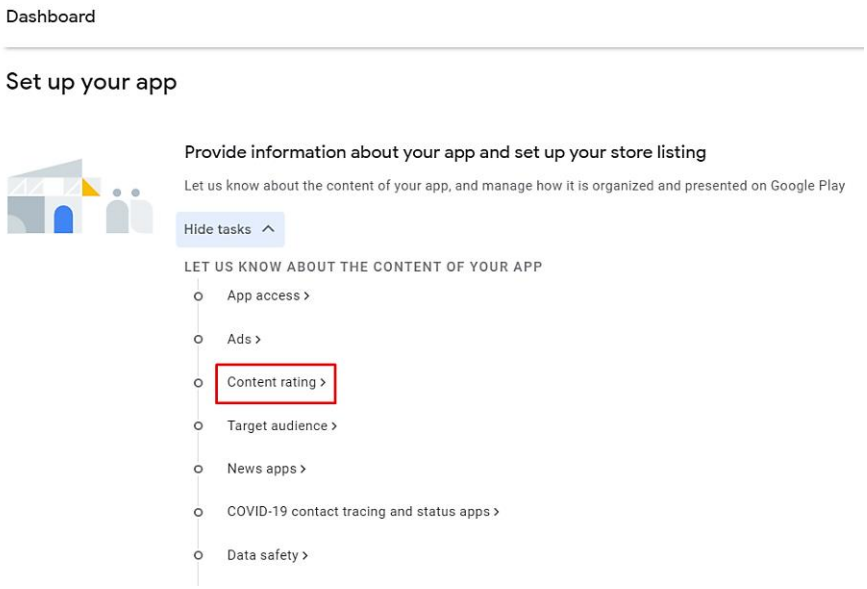
Once you are done uploading details, Hit the Save button.

Also, Read- Mobile App Security: A Comprehensive Guide to Secure Your Apps

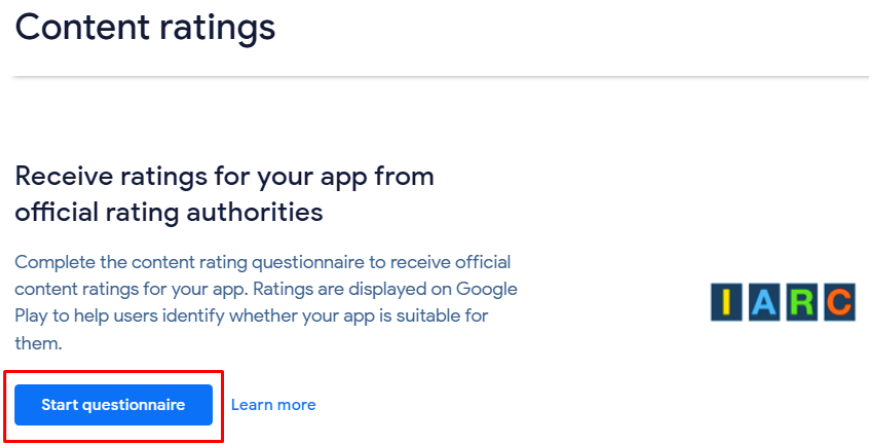
Step 5 – Content Rating

The next most crucial step is the content rating questionnaire. Without rating your app's content, Google will consider it an Underrated app and will certainly remove it from Google Play Store. And since you might not want it, let's learn the steps of adding a content rating.

To add the content rating, you'll have to navigate to the main dashboard, set up your app, and select the Content rating option.

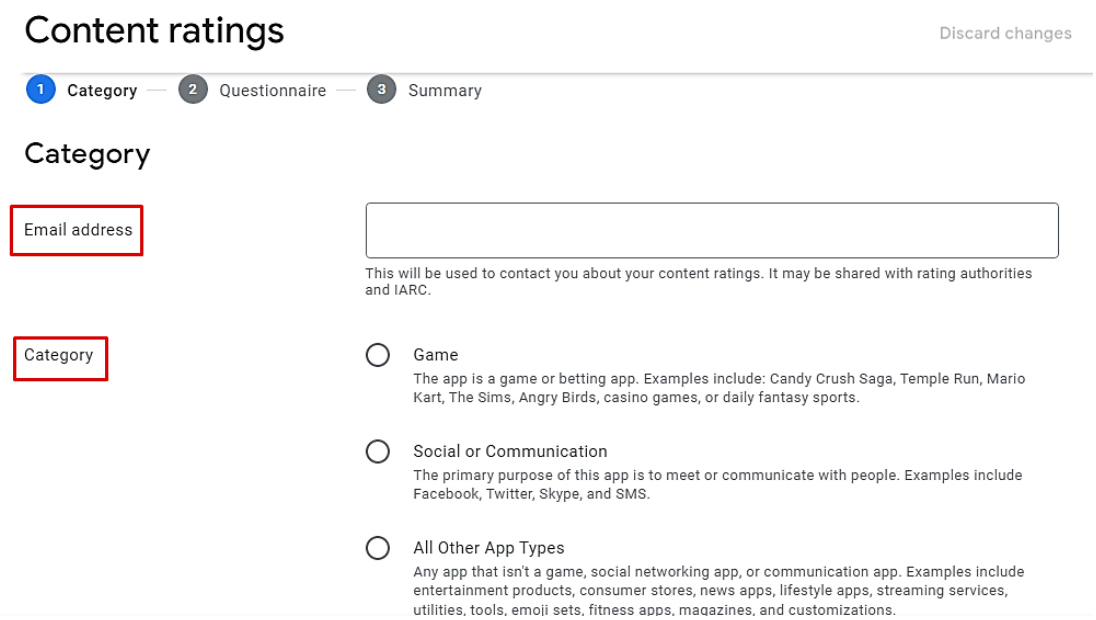


The Next dashboard will pop up, and you'll be able to navigate the "Start Questionnaire" button; you have to click the tab and get started.

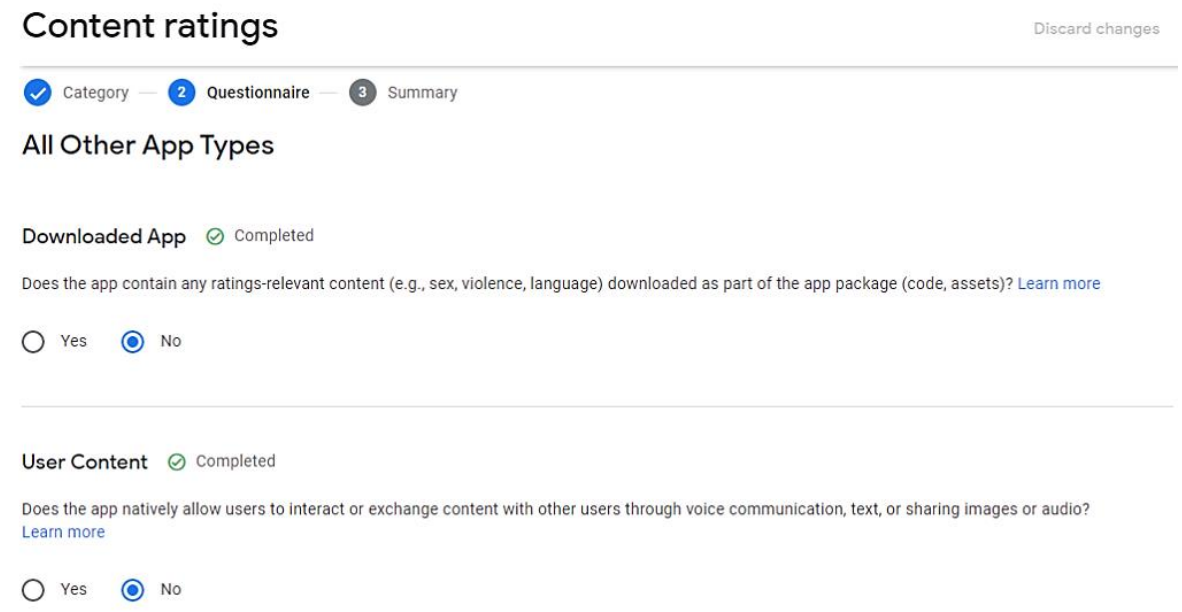


You'll have to enter basic information about your app in the content rating section. This section is divided into three sub-sections – Categories, Questionnaire, and Summary.

In the Category section, you have to enter the email address that the users can leverage to contact you and the category of the app you are publishing on the Play Store.



Once you are done filing the above fields, click the Next button, and you'll be redirected to the questionnaire section. The questionnaire section lets Google explore more about your app to understand your target audience better.



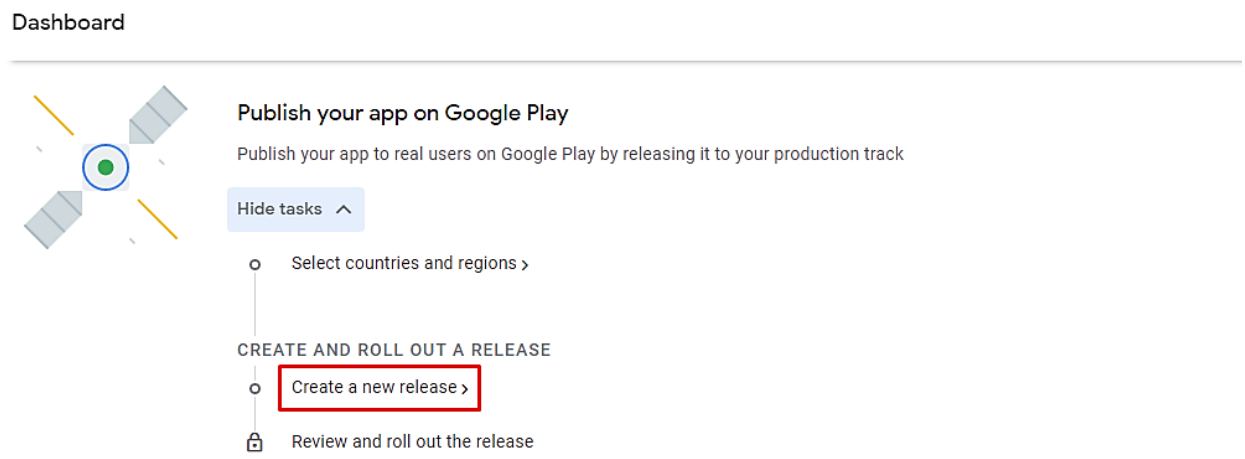
Once all the details are filled in, you can look at the content rating summary and hit 'Submit' to apply the changes.

Step 6 – Create & Upload Android App to Google Play

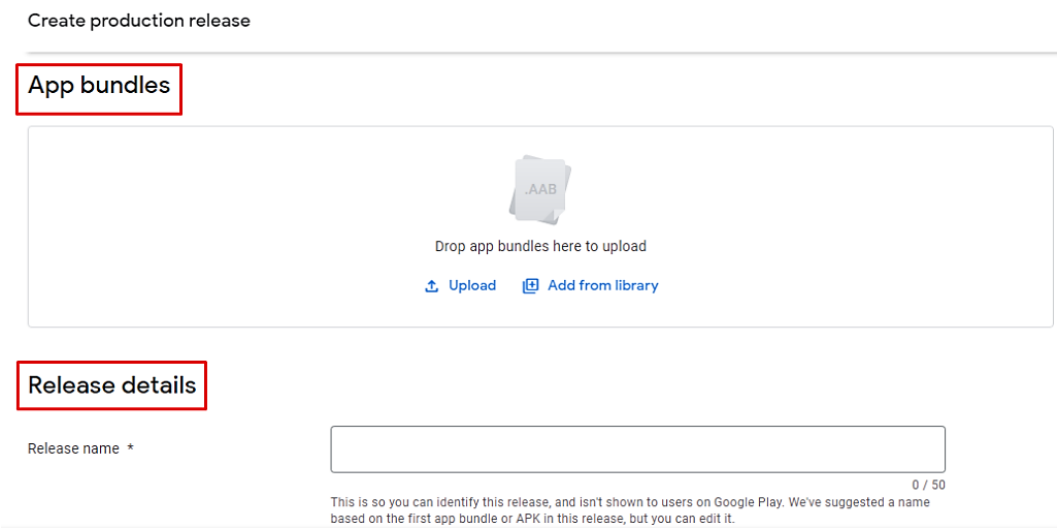
Uploading the APK to the Google Play Console is the foremost step of the app publishing process, where the app is finally uploaded and submitted for Google to review and go

live.

Once you have decided on the testing, you can go to the dashboard and select "**Create a new release.**"



Post selecting "**Create a new release**," you'll be redirected to a dashboard, wherein you will upload the app bundles and the release details.



Once you enter all the details, confirm that everything is correct, and take the last step of this guide on Google Play app upload and add the application to the platform. Then, navigate to the 'App Releases' tab and select 'Manage Production' followed by 'Edit Release'; click on 'Review' and then the 'Start rollout to production option.

Select the 'Confirm' option, and that's it!

You are done with the successful upload of your app to the Google Play Store account for free.

Once the Google Play app upload is done, you must patiently wait for your application to get reviewed and approved by Google. The Google app approval process can take a few hours or extend up to 7days, so ensure that you are well prepared for both successful publishing of the app or for doing any revisions if required.

❖ Android Shared Preferences

Shared Preferences allows activities and applications to keep preferences, in the form of key-value pairs similar to a Map that will persist even when the user closes the application. Android stores Shared Preferences settings as XML file in **shared_prefs** folder under DATA/data/{application package} directory. The DATA folder can be obtained by calling Environment.getDataDirectory(). **SharedPreferences** is application specific, i.e. the data is lost on performing one of the following options:

- on uninstalling the application
- on clearing the application data (through Settings)

As the name suggests, the primary purpose is to store user-specified configuration details, such as user specific settings, keeping the user logged into the application. To get access to the preferences, we have three APIs to choose from:

- **getPreferences()** : used from within your Activity, to access activity-specific preferences
- **getSharedPreferences()** : used from within your Activity (or other application Context), to access application-level preferences
- **getDefaultSharedPreferences()** : used on the PreferenceManager, to get the shared preferences that work in concert with Android's overall preference framework

In this topic we'll go with getSharedPreferences(). The method is defined as follows: getSharedPreferences (String PREFS_NAME, int mode) **PREFS_NAME** is the name of the file. **mode** is the operating mode. Following are the operating modes applicable:

- **MODE_PRIVATE**: the default mode, where the created file can only be accessed by the calling application
- **MODE_WORLD_READABLE**: Creating world-readable files is very dangerous, and likely to cause security holes in applications
- **MODE_WORLD_WRITEABLE**: Creating world-writable files is very dangerous, and likely to cause security holes in applications
- **MODE_MULTI_PROCESS**: This method will check for modification of preferences even if the Shared Preference instance has already been loaded
- **MODE_APPEND**: This will append the new preferences with the already existing preferences
- **MODE_ENABLE_WRITE_AHEAD_LOGGING**: Database open flag. When it is set, it would enable write ahead logging by default

The activity_main.xml layout consists of two EditText views which store and display name and email. The three buttons implement their respective onClicks in the MainActivity.

```
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" >

    <Button
        android:id="@+id/btnSave"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:onClick="Save"
        android:text="Save" />

    <Button
        android:id="@+id/btnRetr"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:onClick="Get"
        android:text="Retrieve" />

    <Button
        android:id="@+id/btnClear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/etEmail"
        android:layout_centerVertical="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:onClick="clear"
        android:text="Clear" />

    <EditText
        android:id="@+id/etEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Email"
        android:inputType="textEmailAddress"
        android:layout_below="@+id/etName"
        android:layout_marginTop="20dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <EditText
        android:id="@+id/etName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Name"
        android:inputType="text"
        android:layout_alignParentTop="true"
        android:layout_alignLeft="@+id/etEmail"
        android:layout_alignStart="@+id/etEmail" />
```

```
</RelativeLayout>
```

```
package com.example.sharedpreferences;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {
    SharedPreferences sharedpreferences;
    TextView name;
    TextView email;
    public static final String mypreference = "mypref";
    public static final String Name = "nameKey";
    public static final String Email = "emailKey";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = (TextView) findViewById(R.id.etName);
        email = (TextView) findViewById(R.id.etEmail);
        sharedpreferences = getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);
        if (sharedpreferences.contains(Name)) {
            name.setText(sharedpreferences.getString(Name, ""));
        }
        if (sharedpreferences.contains(Email)) {
            email.setText(sharedpreferences.getString(Email, ""));

        }

    }

    public void Save(View view) {
        String n = name.getText().toString();
        String e = email.getText().toString();
        SharedPreferences.Editor editor = sharedpreferences.edit();
        editor.putString(Name, n);
        editor.putString(Email, e);
        editor.commit();
    }

    public void clear(View view) {
        name = (TextView) findViewById(R.id.etName);
        email = (TextView) findViewById(R.id.etEmail);
        name.setText("");
        email.setText("");

    }

    public void Get(View view) {
        name = (TextView) findViewById(R.id.etName);
        email = (TextView) findViewById(R.id.etEmail);
        sharedpreferences = getSharedPreferences(mypreference,
                Context.MODE_PRIVATE);

        if (sharedpreferences.contains(Name)) {
            name.setText(sharedpreferences.getString(Name, ""));
```

```
        }
        if (sharedpreferences.contains(Email)) {
            email.setText(sharedpreferences.getString(Email, ""));

        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

}
```

❖ Managing application resources in a hierarchy

Managing application resources in a hierarchy means organizing your app's assets like images, layouts, strings, and other data in a structured tree-like structure, where each resource is categorized and nested within parent categories, allowing for easier access, organization, and management, often with the ability to inherit properties from higher levels in the hierarchy; essentially mirroring a file system structure within your project, with different folders representing different resource types and their variations based on device configurations.

Key points about managing application resources hierarchically:

- **Centralized location:**

  All app resources are typically stored within a dedicated directory within your project, like the "res" folder in Android development.

- **Resource types:**

  Different types of resources are separated into distinct subfolders within the main resource directory, such as "drawable" for images, "layout" for screen layouts, "values" for strings and dimensions.

- **Qualifiers:**

  You can further categorize resources by adding qualifiers to the filenames, allowing you to provide different versions of a resource based on device factors like screen density, orientation, or language.

- **Inheritance and overriding:**

  In some systems, settings or access controls defined at a higher level in the hierarchy can be inherited by child resources, allowing for consistent application behavior across different parts of your app while still enabling customization at lower levels.

  Example of a resource hierarchy in Android development:

- "res" directory:
  - "drawable": Contains image files for different screen densities (e.g., "logo.png", "logo_hdpi.png")
  - "layout": Holds layout files for different screen sizes (e.g., "activity_main.xml", "activity_main_tablet.xml")
  - "values": Contains default values for strings, dimensions, and styles (e.g., "strings.xml", "dimens.xml")
  - "mipmap": Used for app icons that are optimized for different launcher densities

Benefits of hierarchical resource management:

- Organization:

  Keeps your app resources well-structured and easy to find
- Maintainability:

  Simplifies updating resources across different parts of your app by making changes at higher levels in the hierarchy
- Device compatibility:

  Allows you to provide tailored resources for different device configurations using qualifiers
- Access control:

  In some systems, you can set permissions at different levels in the hierarchy to manage who can access specific resources

❖ Working with Different Types of Resources in Android

In Android, resources are external elements such as strings, images, colors, and layouts that help manage UI components efficiently. They allow for better maintainability, localization, and adaptability across different screen sizes and configurations.

Types of Resources in Android

1.Drawable Resources

- Used for images, vector graphics, and XML-based shapes.
- Stored in res/drawable/.

- **Examples:** PNG, JPG, SVG, and XML-based drawables.

🔹 **Example of a Vector Drawable (res/drawable/ic_launcher.xml)**

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
   android:width="24dp"
   android:height="24dp"
   android:viewportWidth="24"
   android:viewportHeight="24">
   <path
      android:fillColor="#FF0000"
      android:pathData="M12,2L15,8H9L12,2Z" />
</vector>
```

2 Layout Resources

- Define the structure of the user interface.
- Stored in res/layout/.

🔹 **Example (res/layout/activity_main.xml)**

```
<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/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"/>
</LinearLayout>
```

3 String Resources

- Used for defining text values (useful for localization).
- Stored in res/values/strings.xml.

🔹 **Example (res/values/strings.xml)**

```
<resources>
   <string name="app_name">MyApplication</string>
   <string name="hello_world">Hello, World!</string>
</resources>
```

🔹 **Accessing String Resources**

- In XML: android:text="@string/hello_world"

- In Java/Kotlin:

4 Color Resources

- Used to define color values.
- Stored in res/values/colors.xml.

🔹 Example (res/values/colors.xml)

```
<resources>
   <color name="primaryColor">#6200EE</color>
   <color name="accentColor">#03DAC5</color>
</resources>
```

🔹 Using Color in XML

```
<TextView
   android:textColor="@color/primaryColor"/>
```

5 Dimension Resources

- Used for defining sizes, margins, and paddings.
- Stored in res/values/dimens.xml.

🔹 Example (res/values/dimens.xml)

```
<resources>
   <dimen name="text_size">16sp</dimen>
   <dimen name="margin">8dp</dimen>
</resources>
```

🔹 Using in XML

```
<TextView
   android:textSize="@dimen/text_size"
   android:layout_margin="@dimen/margin"/>
```

6 Style and Theme Resources

- Define UI consistency across the app.
- Stored in res/values/styles.xml and res/values/themes.xml.

🔹 Example (res/values/styles.xml)

```
<style name="CustomButton" parent="Widget.MaterialComponents.Button">
   <item name="android:background">@color/primaryColor</item>
   <item name="android:textColor">@color/white</item>
```

```
</style>
```

Applying Style to a Button in XML

```
<Button
   style="@style/CustomButton"
   android:text="Click Me"/>
```

7 Menu Resources

- Defines app menus (e.g., options menu, context menu).
- Stored in res/menu/.

Example (res/menu/main_menu.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/action_settings"
      android:title="@string/settings"
      android:icon="@drawable/ic_settings"
      android:showAsAction="always"/>
</menu>
```

8 Animation Resources

- Define animations using XML.
- Stored in res/anim/.

Example (res/anim/fade_in.xml)

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
   android:duration="1000"
   android:fromAlpha="0.0"
   android:toAlpha="1.0"/>
```

9 Font Resources

- Used to define custom fonts.
- Stored in res/font/.

Example (res/font/roboto_medium.xml)

```
<font-family xmlns:android="http://schemas.android.com/apk/res/android">
   <font android:font="@font/roboto_medium" android:weight="500"/>
</font>
```

Using in XML

```
<TextView
   android:text="Hello, World!"
   android:fontFamily="@font/roboto_medium"/>
```

UNIT – 5

❖ API in Android App development

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

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

Here are some common Android APIs categorized by their functionalities:

1. Data Storage & Management APIs

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

2. Networking & Web APIs

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

3. Telephony & Messaging APIs

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

4. Location & Mapping APIs

- **Google Maps API** – Embed maps and enable location-based services.
- **GPS & Location API** – Retrieve the user's location using GPS, Wi-Fi, or mobile networks.
- **Geofencing API** – Trigger actions when a device enters or leaves a specific geographical area.
- **Fused Location Provider API** – Efficiently retrieve the user's location with minimal battery drain.

5. Sensors & Connectivity APIs

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

6. Multimedia & Camera APIs

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

7. UI & Animation APIs

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

8. Security & Permissions APIs

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

9. Background Processing APIs

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

❖ Internal storage

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

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

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

Writing file

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

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

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

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

Reading file

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

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

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

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

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

Apart from the the methods of write and close, there are other methods provided by

the **FileOutputStream** class for better writing files. These methods are listed below –

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

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

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

❖ Android external storage

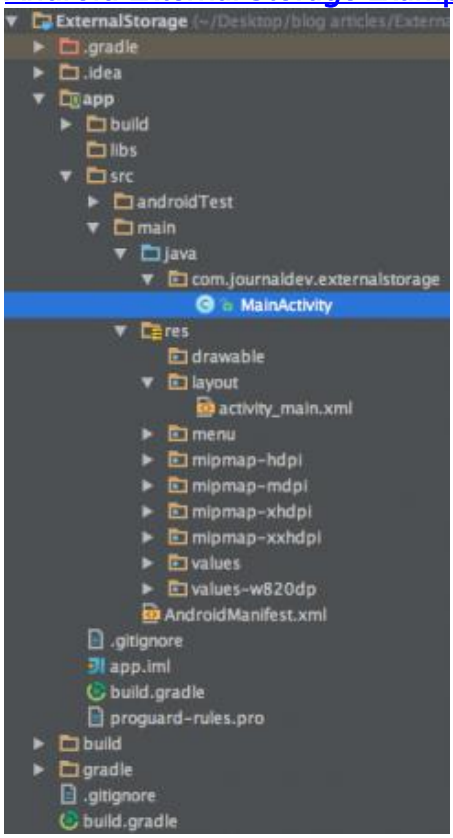
Android external storage can be used to write and save data, read configuration files etc.

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

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

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

Android External Storage Example Project Structure



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

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

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

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

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

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

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

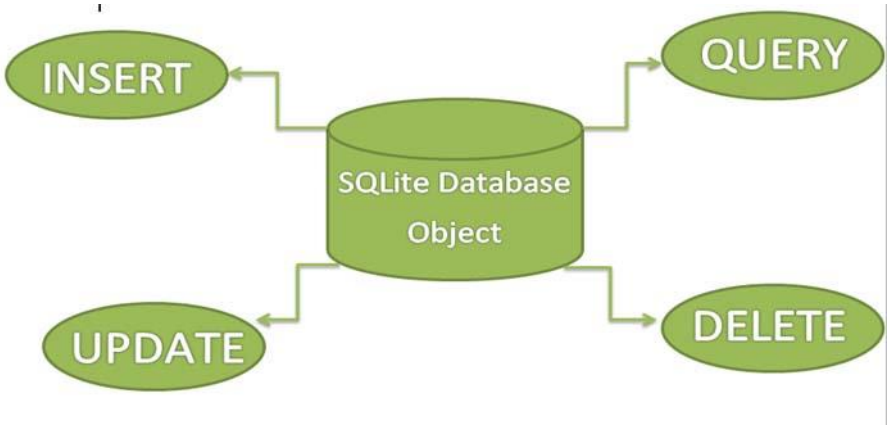
getExternalStorageState() is a static method of Environment to determine if external storage is presently available or not. As you can see if the condition is false we've disabled the save button.

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

❖ SQLite Database

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

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



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

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

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

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

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

❖ Managing data using Sqlite

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

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

Database - Package

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

Database - Creation

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

Its syntax is given below

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

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

Database - Insertion

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

Its syntax is given below

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

Database - Fetching

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

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

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

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.

2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

Database - Helper class

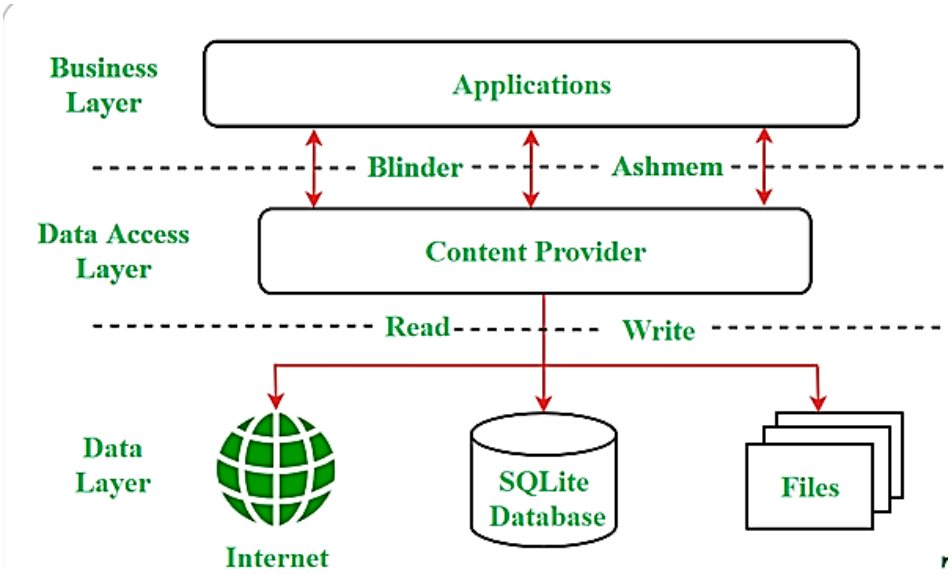
For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

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

❖ Sharing data between Applications with Content Providers

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

to other applications to interfere with the data.



Content URI

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

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

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

Details of different parts of Content URI:

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

Operations in Content Provider

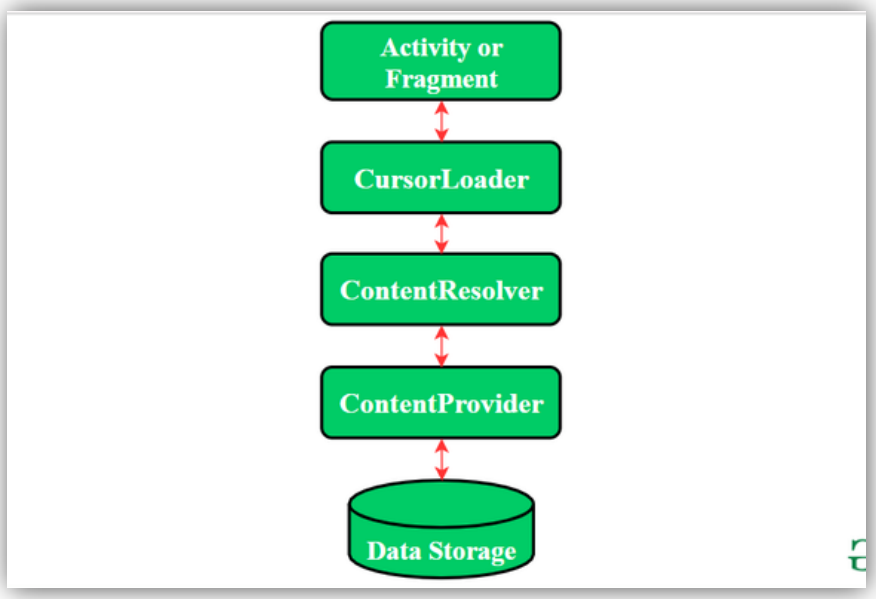
Four fundamental operations are possible in Content Provider namely **Create** , **Read** , **Update** , and **Delete** . These operations are often termed

as **CRUD operations** .

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

Working of the Content Provider

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



Creating a Content Provider

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

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

Following are the six abstract methods and their description which are essential to

override as the part of ContenProvider class:

Abstract Method	Description
query()	A method that accepts arguments and fetches the data from the desired table. Data is retired as a cursor object.
insert()	To insert a new row in the database of the content provider. It returns the content URI of the inserted row.
update()	This method is used to update the fields of an existing row. It returns the number of rows updated.
delete()	This method is used to delete the existing rows. It returns the number of rows deleted.
getType()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
onCreate()	As the content provider is created, the android system calls this method immediately to initialise the provider.

❖ Using Android Networking APIs

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

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

Checking Network Connection

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

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

Once you instantiate the object of ConnectivityManager class, you can

use **getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

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

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

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

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

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

Performing Network Operations

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

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

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

After that you need to call **openConnection** method of url class and receive it in a

HttpURLConnection object. After that you need to call the **connect** method of HttpURLConnection class.

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

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

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

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

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

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

❖ web content within your Android app

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

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

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

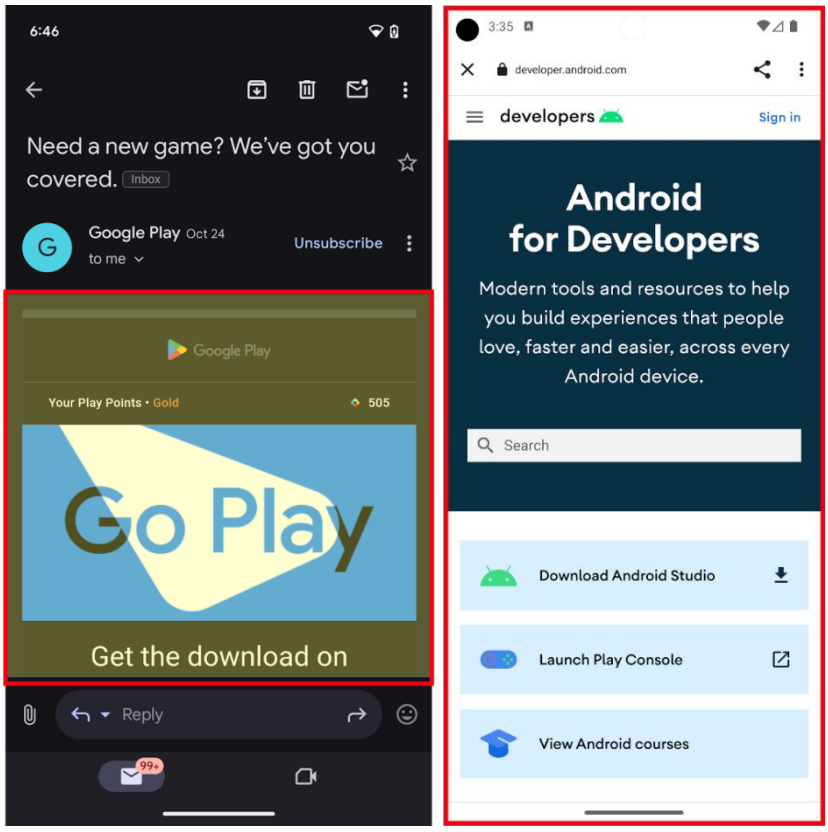


Figure 1. WebView (left) and Custom Tab (right) outlined in red.

Embed web content

- Efficiency: Reuse existing code from your website. Build on existing web technologies and content.
- Integration: Leverage external content from 3P providers, such as Media, Ads, within your app.

- Flexibility: Update content dynamically without being constrained to predefined UIs, or without releasing app updates.

use web content

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

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

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

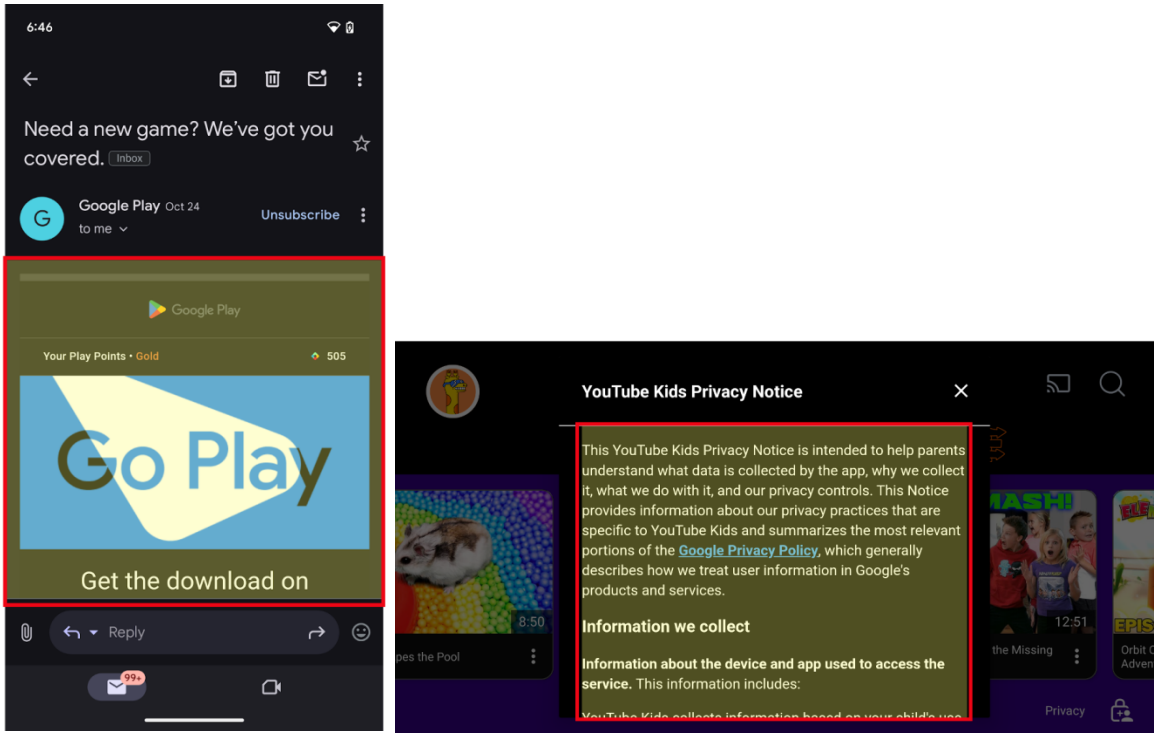


Figure 2. Web content embedded within the app with WebViews as primary (left) and supporting content (right).

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

- Have a full in-app browsing experience for when users click a link and you want to keep them in the app, instead of leaving to an external browser.
- Note: For large screen devices such as tablets and foldables, there are additional options to help apps take advantage of additional space:
- Apps can open weblinks in split screen using launch an adjacent multi-window experience. This enables users to multitask between your app and a browser at the same time. OR

- Custom Tabs have a side panel option that can open in the same task, but next to your existing app content.
- The Custom Tab is powered by the user's default browser, for browsers which support Custom Tabs.
- While it's possible to use a WebView and provide a highly customizable in-app browsing experience, we recommend Custom tabs for an out-of-the-box browser experience and seamless transition for when a user wants to open a web link in the browser.

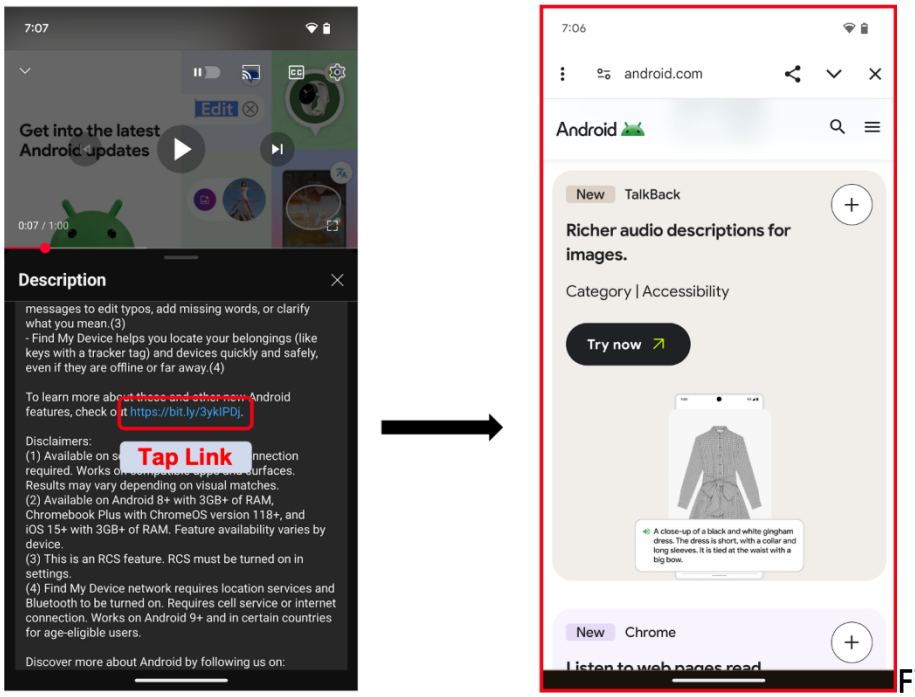


Figure 3. Clicking on an in-app link (left) and opening an in-app browser (right).

3. Login or Authentication flows within your app

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

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

❖ JSON Parsing

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

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

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

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

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

❖ Using the Telephony API

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

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

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

TelephonyDemo.java

```
package com.telephone;

import android.app.Activity;
```

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

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

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

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

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

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

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

main.xml

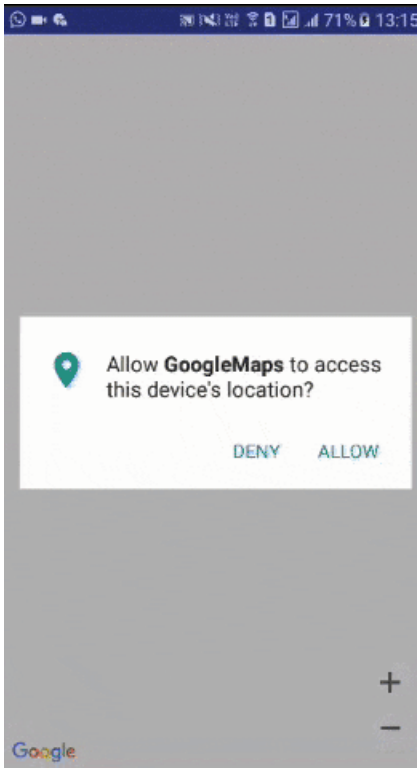
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
```

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

Output

- **Google Maps in Android**

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

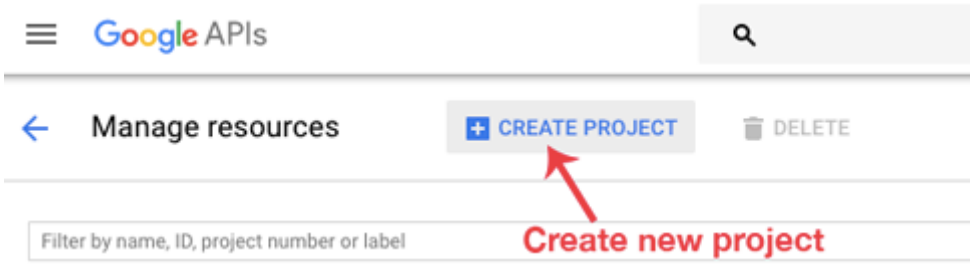


Steps For Getting The Google Maps Api Key:

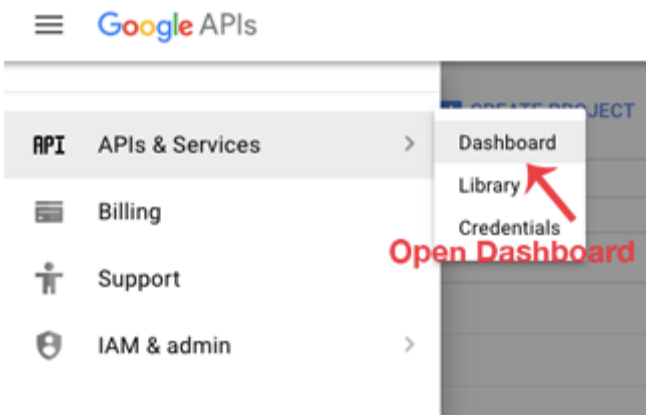
An API key is needed to access the Google Maps servers. This key is free and you can use it with any of your applications. If you haven't created project, you can follow the below steps to get started:

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

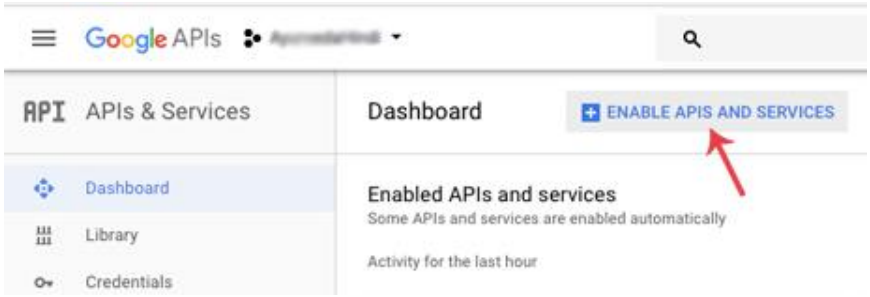
Step 2: Now create new project. You can create new project by clicking on the **Create Project** button and give name to your project.



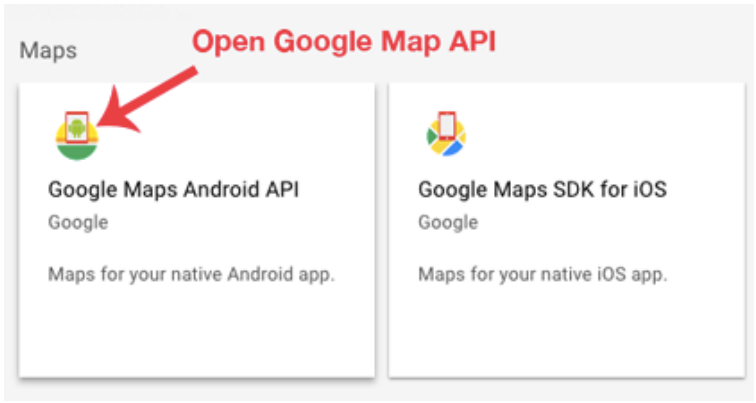
Step 3: Now click on APIs & Services and open **Dashboard** from it.



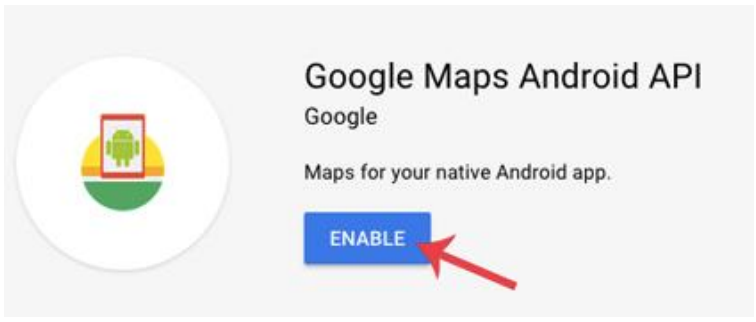
Step 4: In this open **Enable APIS AND SERICES**.



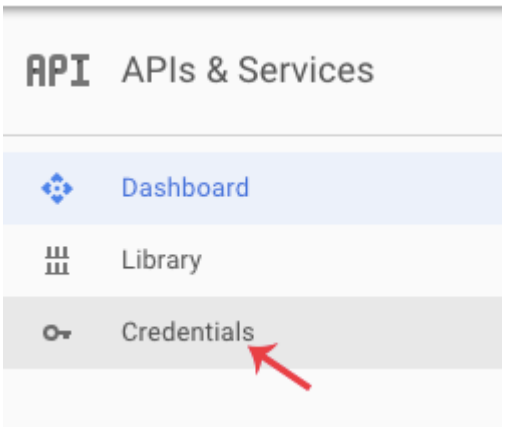
Step 5: Now open Google Map Android API.



Step 6: Now enable the Google Maps Android API.



Step 6: Now go to **Credentials**



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

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

Google Maps Example

Step 1: Add Google Maps SDK Dependency

gradle

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

Step 2: Get a Google Maps API Key

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

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

Step 3: Add a Map Fragment to Your Activity

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

```
    android:layout_height="match_parent"/>
```

Step 4: Load the Map in Java

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

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

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

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