IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT

2023 – Lab Sheet

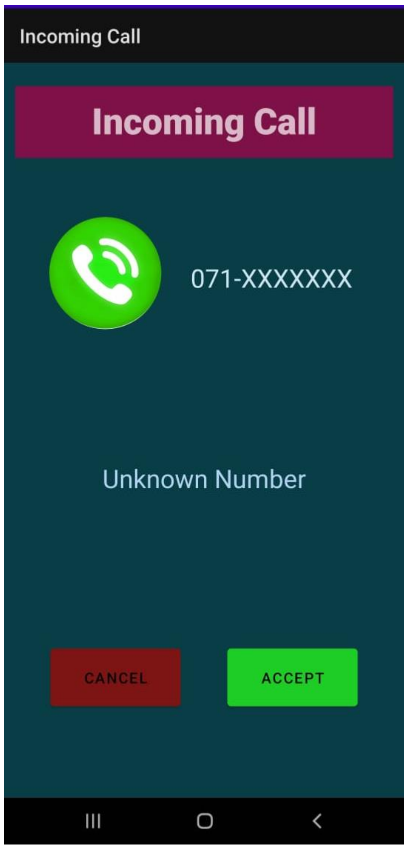
1. Getting Started

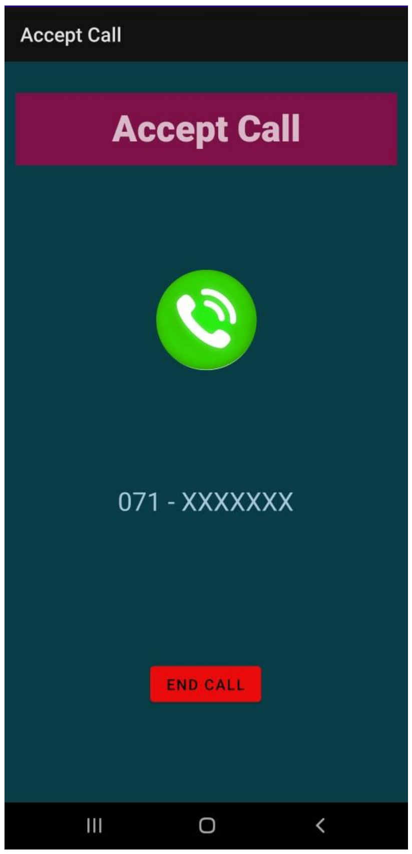
XML stands for ***eXtensible Markup Language***, which is a way of describing data using a text-based document. Because XML is extensible and very flexible, it's used for many different things, including defining the UI layout of Android apps. XML is case-sensitive, it requires each tag is closed properly, and preserves whitespace. XML contains only tags.

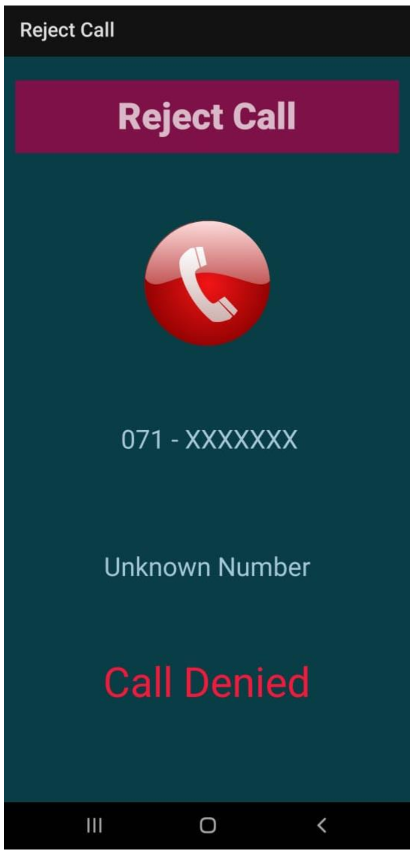
The simple syntax of XML :

<tag_name> Hello World! </tag_name>

Here's what the finished app will look like: UI Design



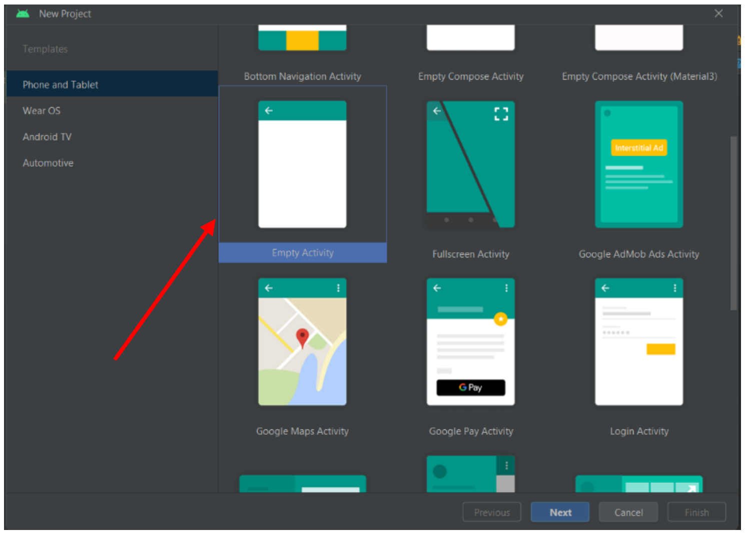




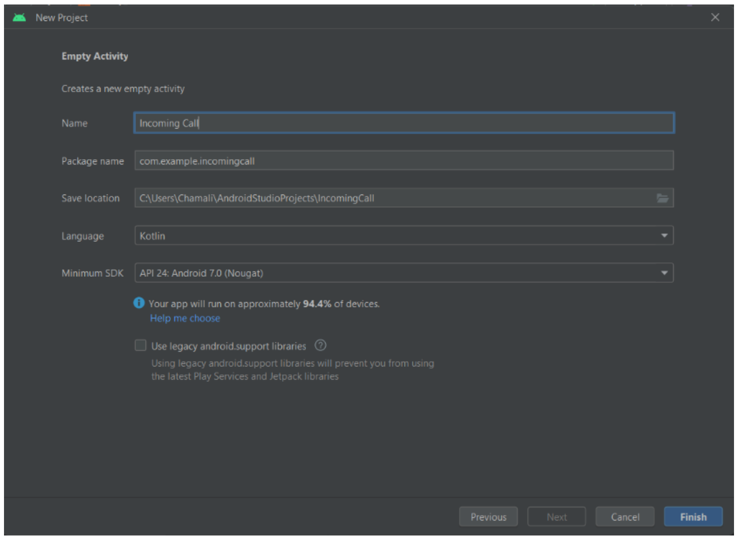
2. Start the Project

2.1 Create a new Project in Android Studio

01) You must open android studio application then you will appear a new window, then select **Phone and Tablet** in the left side pane, after that select the **empty activity** from the templates and click **Next** button.



02) Give your application name. such as My_First_App. In here we give **Incoming Call** as an application name. you can give save location as well. Make sure that language must be set to **kotlin**. In here SDK set to **API 24: Android 7.0 (Nougat).** Leave the defaults for the other fields.



Empty Activity

Creates a new empty activity

Name	Incoming Call
Package name	com.example.incomingcall
Save location	C:\Users\Chamali\AndroidStudioProjects\IncomingCall
Language	Kotlin
Minimum SDK	API 24: Android 7.0 (Nougat)

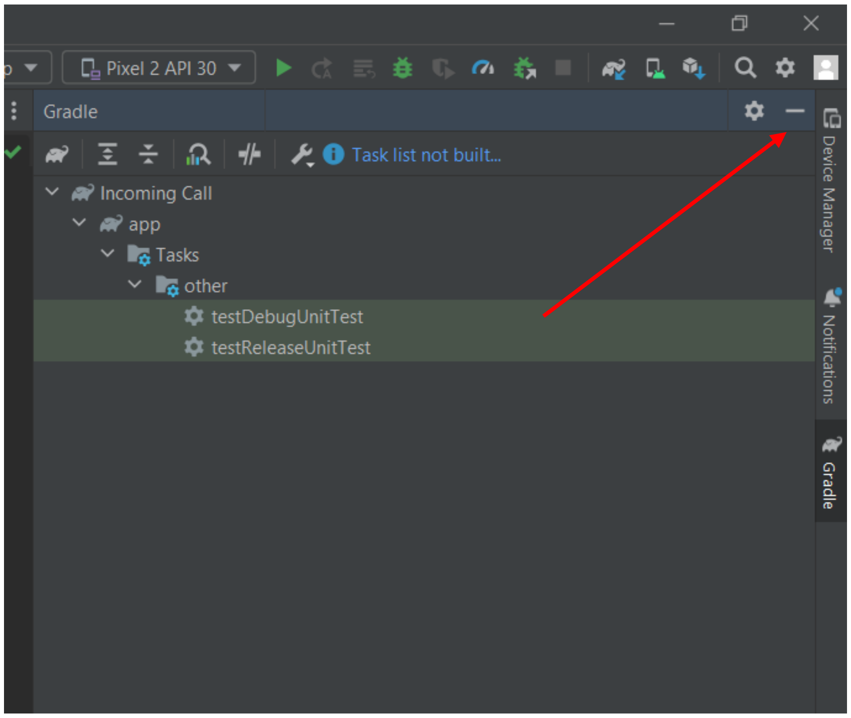
Your app will run on approximately **94.4%** of devices.
Help me choose

☐ Use legacy android.support libraries
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

2.2 Get Your Screen Setup.

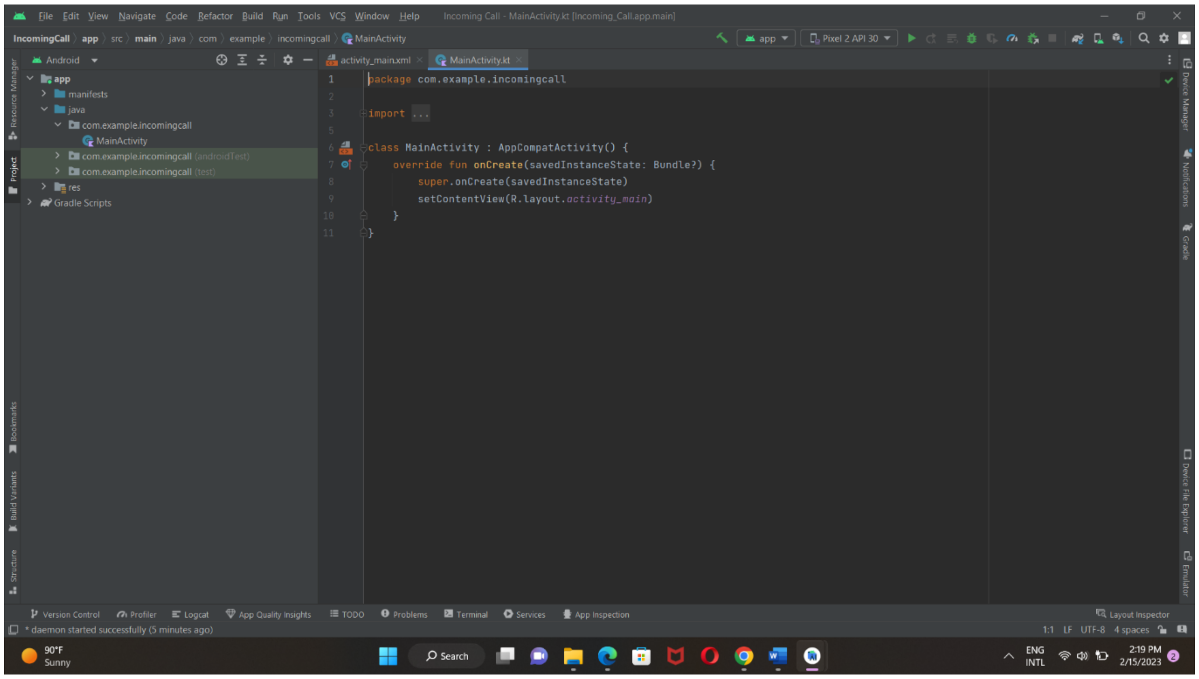
When your project first opens in Android Studio, there may be a lot of windows and panes open. To make it easier to get to know Android Studio, here are some suggestions on how to customize the layout.

If there's a **Gradle** window open on the right side, click on the minimize button **(**—) in the upper right corner to hide.



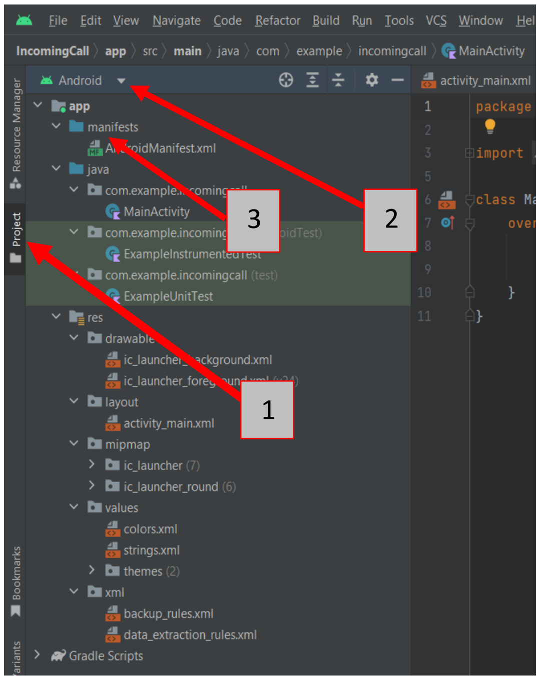
Depending on the size of your screen, consider resizing the pane on the left showing the project folders to take up less space.

At this point your window should be similar to the screenshot shown below.



```
package com.example.incomingcall

import ...

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

2.3 Explore the project structure and layout.



The upper left of the Android Studio window should look similar to the following structure:

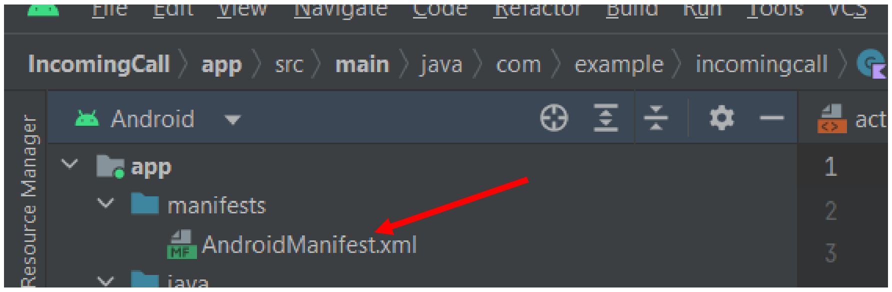
According to the Empty Activity template for your project, Android Studio has setup several files for you. You can look at the hierarchy of the files for your application.

- Double-click the **app (3)** folder to expand the hierarchy of app files.
- If you click **Project (1)**, you can hide or show the **Project** view.
- The current **Project** view selection is **Project > Android(2)**.

1. **Project View**: It shows your files and folders structured in a way that is convenient for working with an Android Studio project.

2. **Android:** In Android view you can see top level folders.
3. **App**: below your app folder, like manifests, java, res, Gradle scripts.

➢ Expand the **manifests** folder.


File Edit View Navigate Code Refactor Build Run Tools VCS
IncomingCall 〉 app 〉 src 〉 main 〉 java 〉 com 〉 example 〉 incomingcall
Android
app
manifests
AndroidManifest.xml
java
Resource Manager

This folder contains **AndroidManifest.xml.** This file describes all the components of your Android app and is read by the Android runtime system when your app is executed.

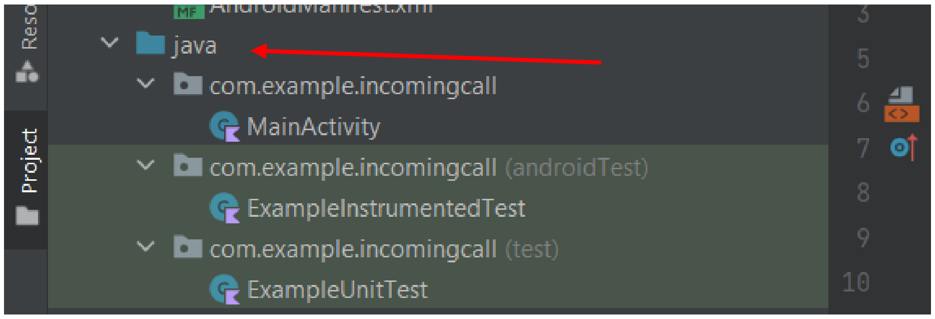
➢ Expand the **java** folder.

All your Kotlin language files are organized here; Android projects keep all Kotlin language files in this folder, together with any Java sources. The **java** folder contains three subfolders:

com.example.incoming call (or the domain name you have specified)**:** This folder contains the Kotlin source code files for your app.

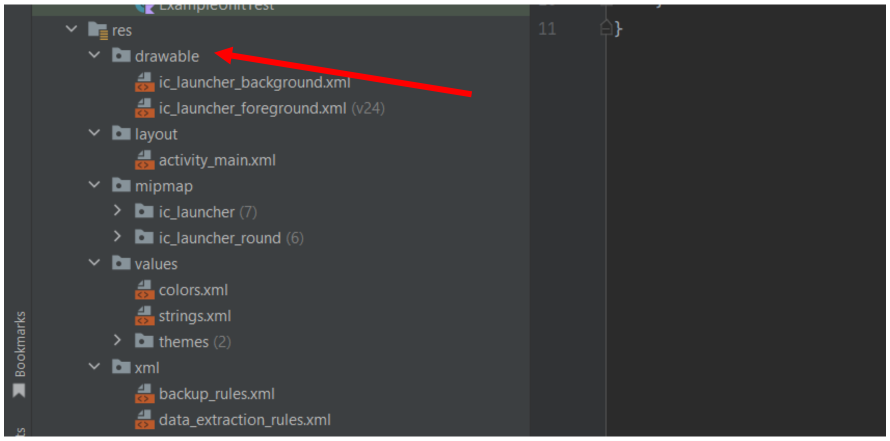
com.example.incoming call (androidTest): This folder is where you would put your instrumented tests, which are tests that run on an Android device.

com.example.incoming call (test): This folder is where you would put your unit tests. Unit tests don't need an Android device to run.



- Expand the **res** folder.

This folder contains all the resources for your app, including images, layout files, strings, icons, and styling. It includes these subfolders:



drawable: All your app's images will be stored in this folder.

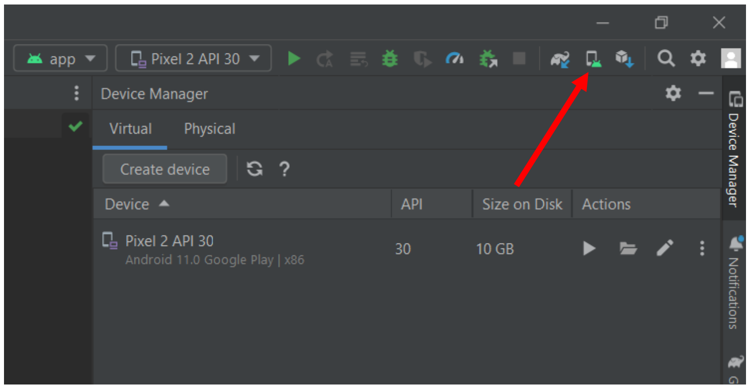
mipmap: This folder contains the launcher icons for your app.

values: Contains resources, such as strings and colors, used in your app.

2. Create Virtual Device (Emulator)

In this task you will use the Android Virtual Device (AVD) manager to create a virtual device (or emulator) that simulates the configuration for a particular type of Android device.

- In Android Studio Tools > AVD Manager or click the AVD Manager icon in the toolbar.

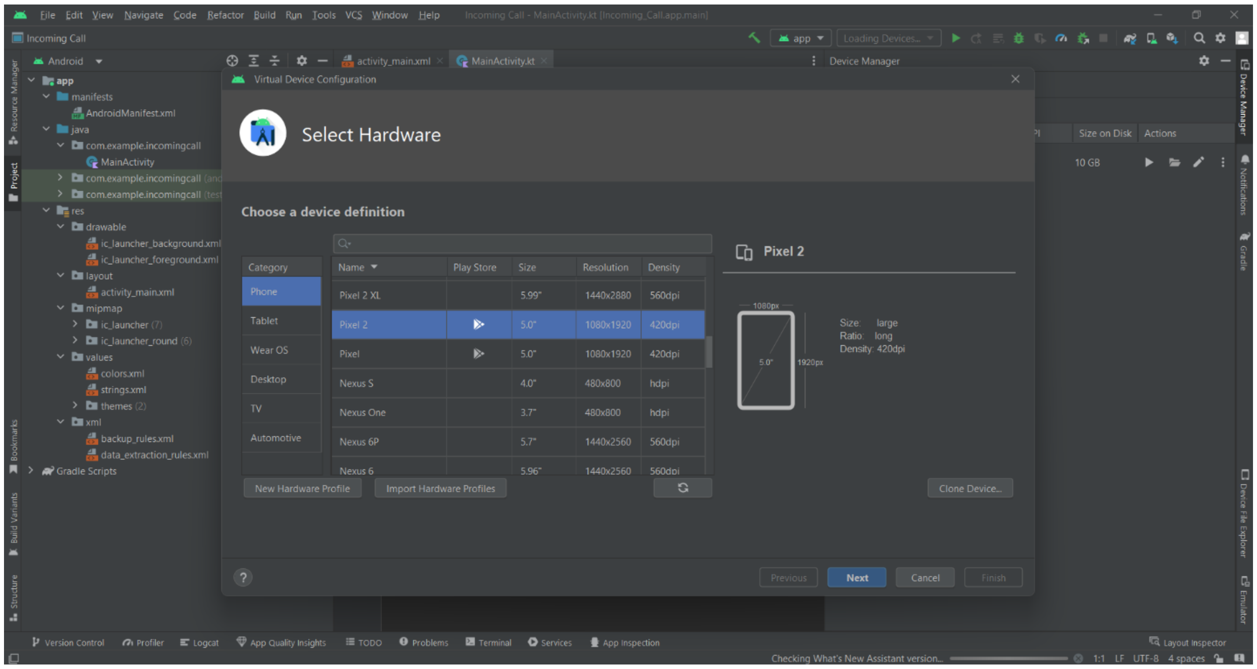


Device	API	Size on Disk	Actions
Pixel 2 API 30 Android 11.0 Google Play | x86	30	10 GB	

- Click **+Create Virtual Device**. (If you have created a virtual device before, the window shows all of your existing devices and the **+Create Virtual Device** button is at the bottom.)

- The **Select Hardware** window shows a list of pre-configured hardware device definitions.

- Choose a device definition such as **pixel 2** and then click **Next**.

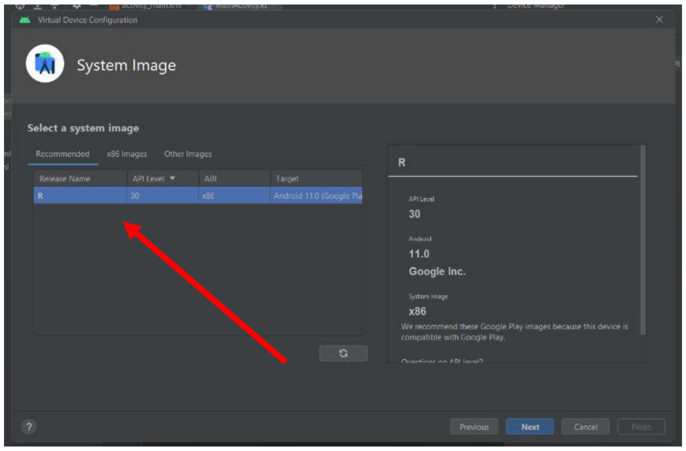


Category	Name	Play Store	Size	Resolution	Density
Phone	Pixel 2 XL		5.99"	1440x2880	560dpi
Tablet	Pixel 2	▶	5.0"	1080x1920	420dpi
Wear OS	Pixel	▶	5.0"	1080x1920	420dpi
Desktop	Nexus S		4.0"	480x800	hdpi
TV	Nexus One		3.7"	480x800	hdpi
Automotive	Nexus 6P		5.7"	1440x2560	560dpi
	Nexus 6		5.96"	1440x2560	560dpi

Pixel 2
Size: large
Ratio: long
Density: 420dpi

- In the **System Image** dialog, from the **Recommended** tab, choose the latest release. In here we choose the **R**.

- If a **Download** link is visible next to a latest release, it is not installed yet, and you need to download it first. If necessary, click the link to start the download, and click **Next** when it's done. This may take a while depending on your connection speed.

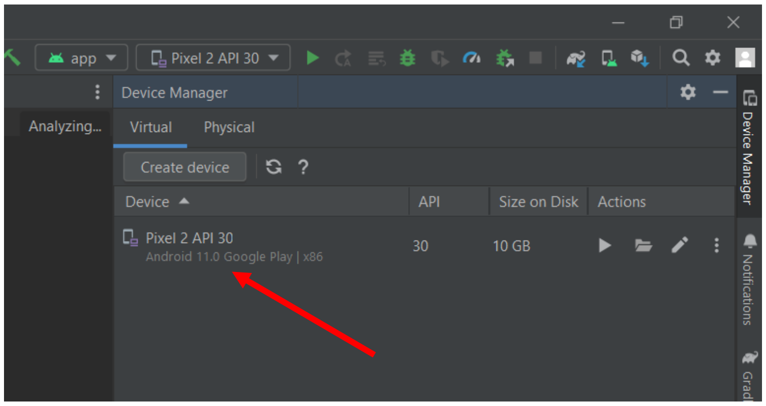


Release Name	API Level	ABI	Target
R	30	x86	Android 11.0 (Google Pla

R
API Level
30
Android
11.0
Google Inc.
System Image
x86
We recommend these Google Play images because this device is compatible with Google Play.

- In the next dialog box, accept the defaults, and click **Finish**.

- The AVD Manager now shows the virtual device you added.

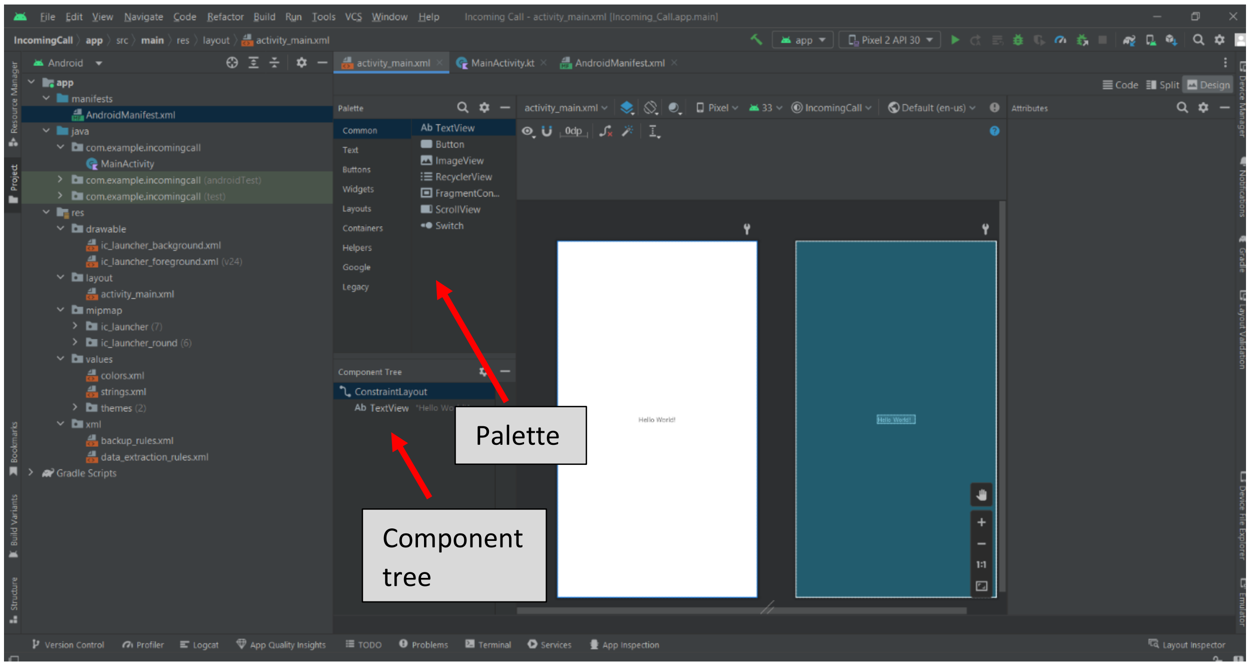


Device	API	Size on Disk	Actions
Pixel 2 API 30 Android 11.0 Google Play | x86	30	10 GB	

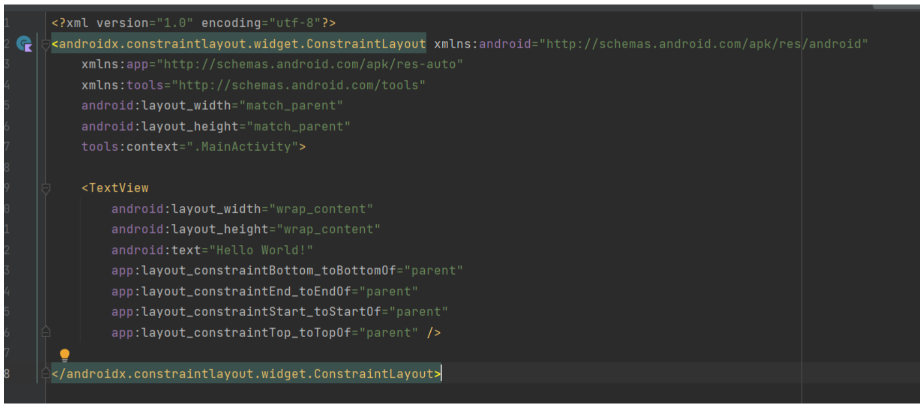
3. Create UI design using XML

3.1 Identify elements in activity_main.xml

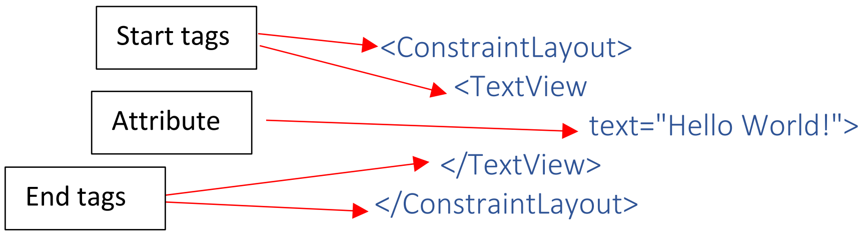
- Open **activity_main.xml** (res>layout>activity_main.xml)
- You can see the design view with “ hello world”.
- Then you can see **icon palette** and **component tree**.
- In here component tree describes that hierarchy of UI elements. You can see **constraint layout** and **child** views under the component tree, For a example , a constraint layout(the parent) can contain **TextViews**, **ImageViews**, **Buttons** or other views(child views)



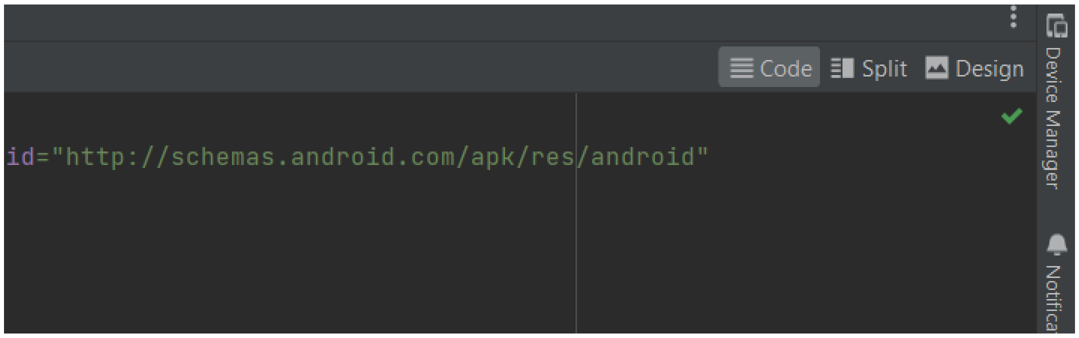
- Each UI element is represented by an XML element in the XML file. Each element starts and ends with a tag, and each tag starts with a < and ends with a >
- You can set attributes in UI elements using the **Layout Editor (design),** the XML elements can have attributes too. In the XML file UI elements can be like this:



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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

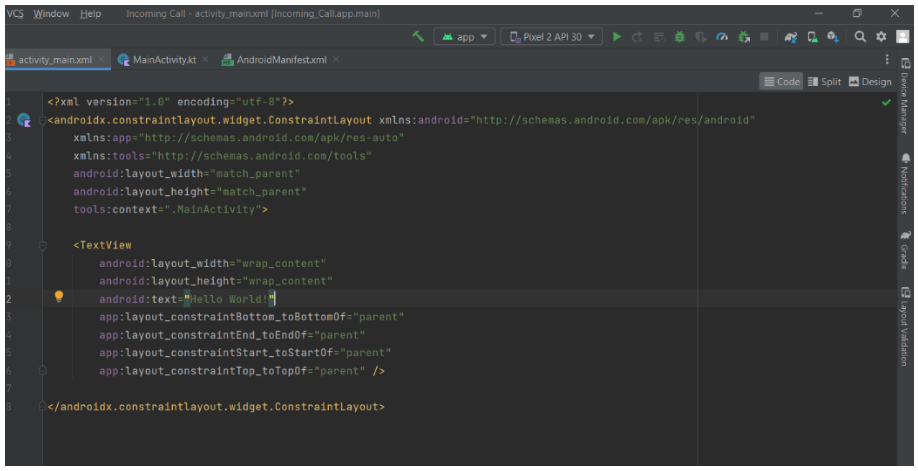


- Find the options for **Code**, **Split**, and **Design** views in the upper right of the Layout Editor.



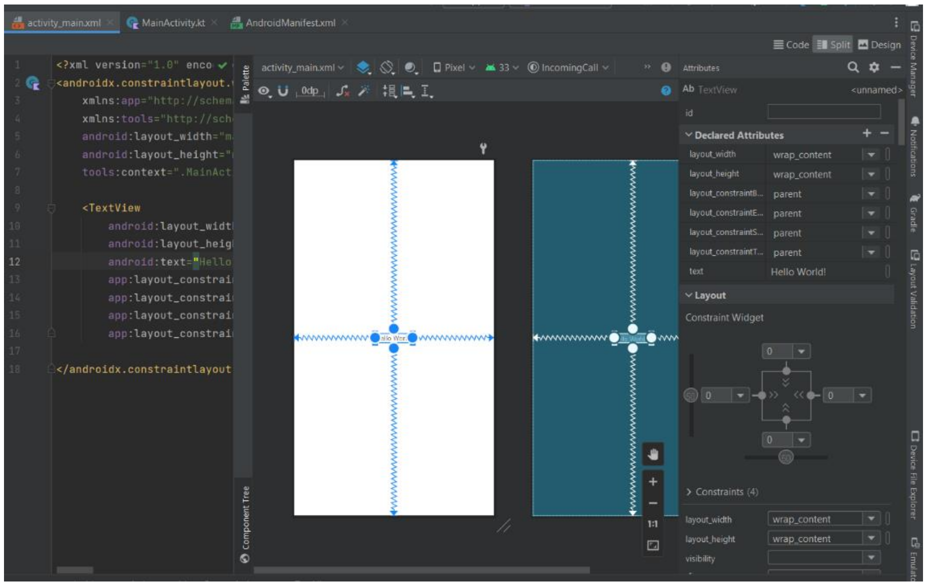
```
id="http://schemas.android.com/apk/res/android"
```

- If you click **Code** view then you can see XML code in **activity_main.xml**.



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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

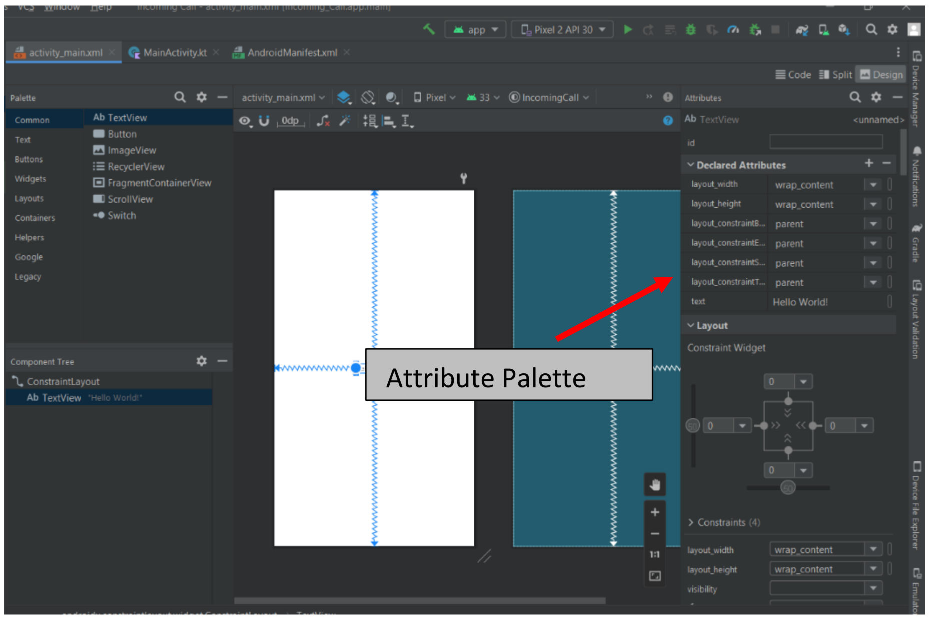
- In **Split View** you can see both **code** view and **Design** View.



```xml
<?xml version="1.0" enco
<androidx.constraintlayout.
    xmlns:app="http://schem
    xmlns:tools="http://sch
    android:layout_width="m
    android:layout_height="
    tools:context=".MainAct

    <TextView
        android:layout_widt
        android:layout_heig
        android:text="Hello
        app:layout_constrai
        app:layout_constrai
        app:layout_constrai
        app:layout_constrai

</androidx.constraintlayout
```

```
id
layout_width            wrap_content
layout_height           wrap_content
layout_constraintB...   parent
layout_constraintE...   parent
layout_constraintS...   parent
layout_constraintT...   parent
text                    Hello World!
layout_width            wrap_content
layout_height           wrap_content
visibility
```

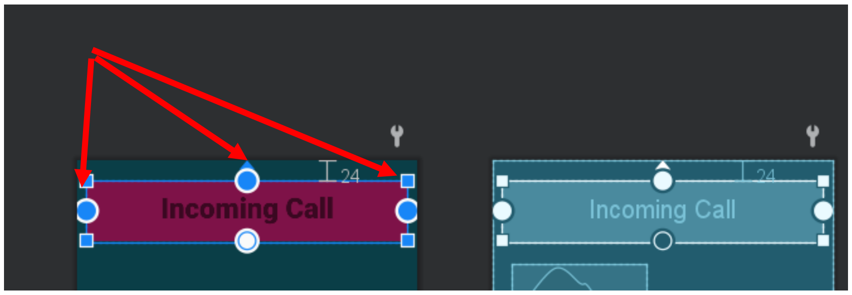
- In **Design** View you can see Design of the project.



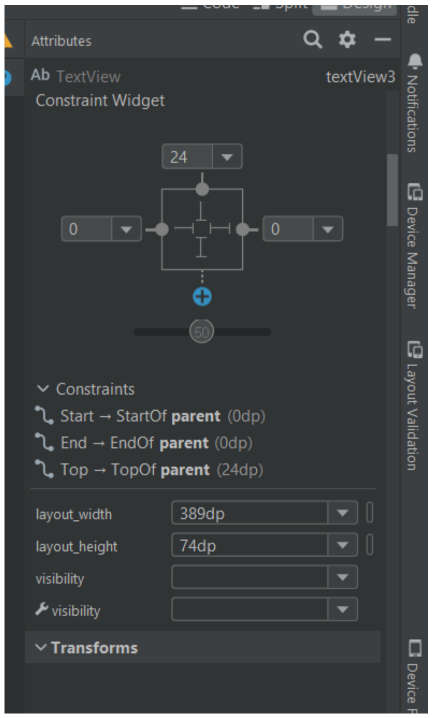
- Identify Attribute palette of the **Design** View. It shows attributes of the elements. Such as TextView, Buttons etc.

3.2 Let's Start create the UI design

- Step 01: First we drag and drop a **TextView** from the palette to the Design.
- Then give **id** for TextView such as textview1 using the **Attribute** palette and give Text as Incoming Call.
- Step 02: Set a **Constraints** to the TextView. A constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline. If you do not set the constraints it may be shows the error.



It will be appear in Constraint layout in the Attribute palette.



Attributes

Ab TextView textView3

Constraint Widget

Constraints

Start → StartOf **parent** (0dp)

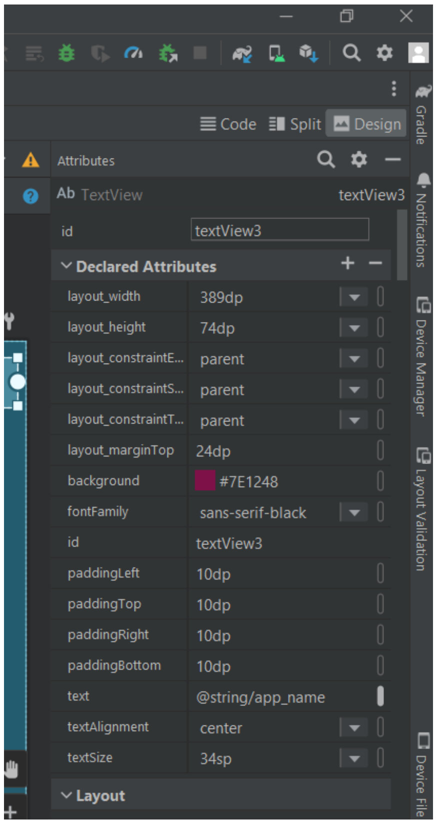
End → EndOf **parent** (0dp)

Top → TopOf **parent** (24dp)

layout_width	389dp
layout_height	74dp
visibility	
visibility	

Transforms

Step 03: Then you can do more formatting using the **Attribute** palette. Such as Text editing, change background color, padding etc.



Attributes

Ab TextView textView3

id textView3

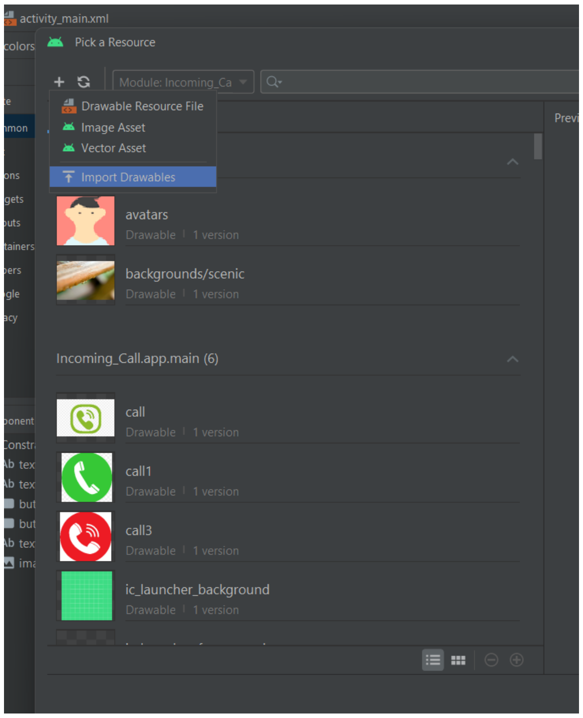
Declared Attributes

layout_width	389dp
layout_height	74dp
layout_constraintE...	parent
layout_constraintS...	parent
layout_constraintT...	parent
layout_marginTop	24dp
background	🟥 #7E1248
fontFamily	sans-serif-black
id	textView3
paddingLeft	10dp
paddingTop	10dp
paddingRight	10dp
paddingBottom	10dp
text	@string/app_name
textAlignment	center
textSize	34sp

Layout

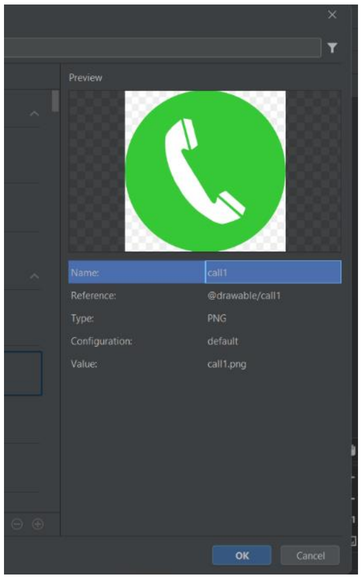
Step 04: Adding Images : First, you should drag and drop a **ImageView** from the palette. Then you will appear new window like this:

Then click **+** mark in the upper left corner and click **import drawables**.



Then you can import any saved images from your computer and give a name for it.

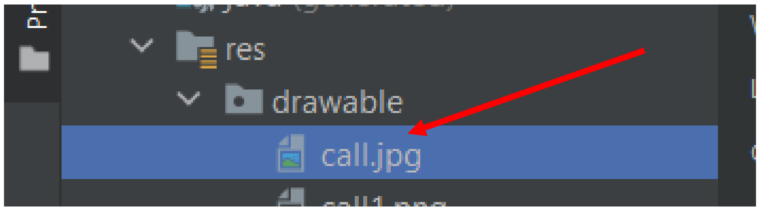
We downloaded it from the website.
Link is available below.



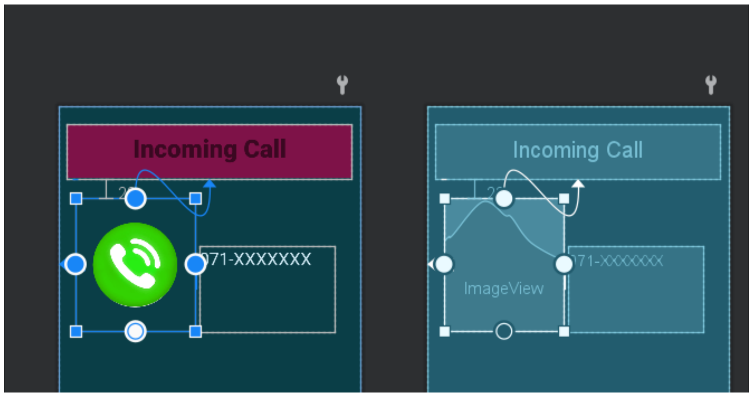
Name:	call1
Reference:	@drawable/call1
Type:	PNG
Configuration:	default
Value:	call1.png

https://www.google.com/search?q=incoming+call+button+picture+png&sxsrf=AJOqlzUcCVELXTL0cxs53RIxJbkJ6hBSUQ:1676537872451&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiCutui1pn9AhUxT2wGHUEtAnAQ_AUoAXoECAEQAw&biw=1536&bih=688&dpr=1.25#imgrc=zeGTp-G7cMTOOM

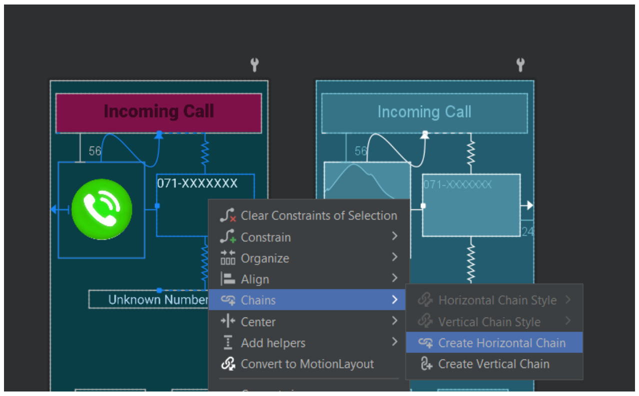
Then click **OK.** After that you can see **ImageView** which you add earlier, it is add to the **drawable** folder under the resources.



Step 05: Then set constraints for ImageView. In here we give constraints like this:

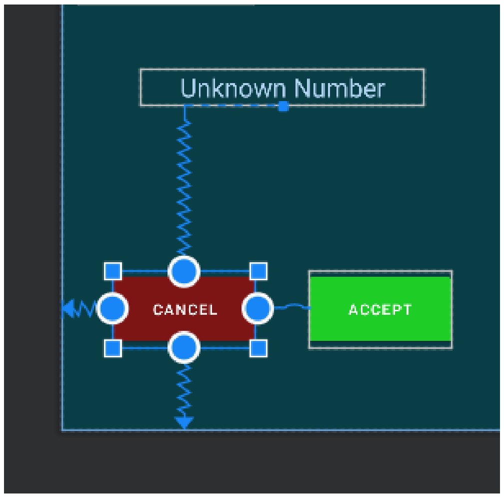


Step 06: After adding another TextView (071-XXXXXXX) then you have to select the both ImageView and TextView then right click after that go to the **Chains** and select the **create the Horizontal Chain**.



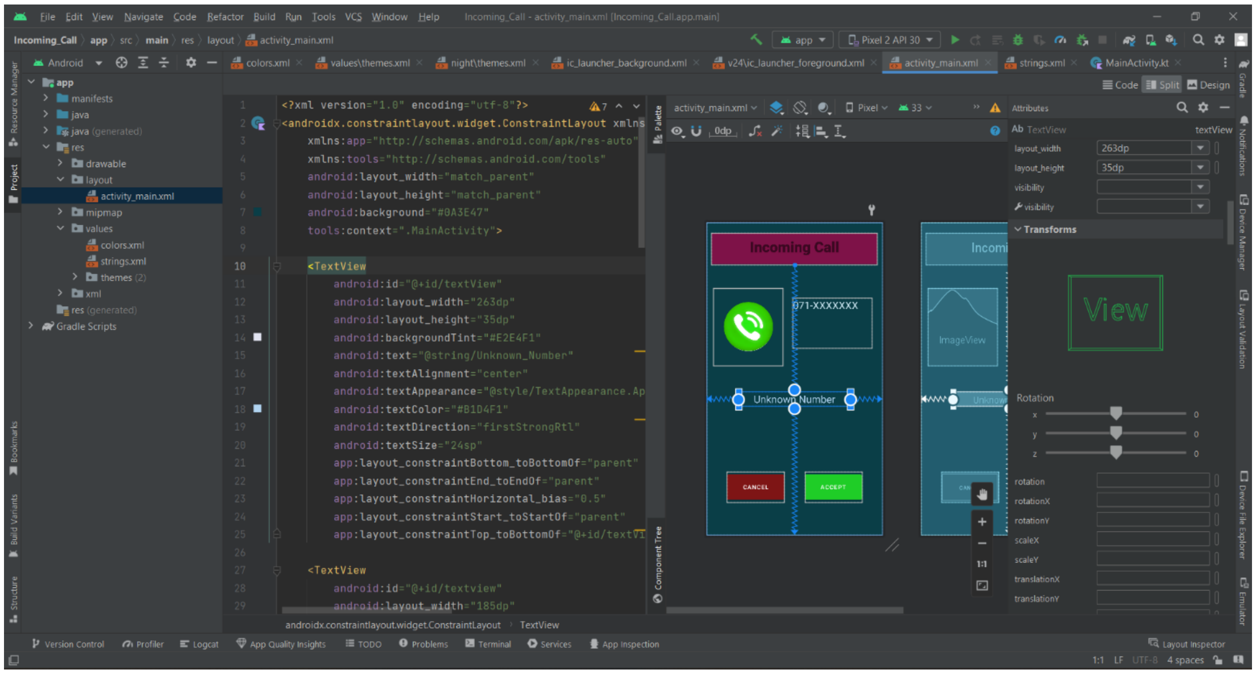
Step 07: **Adding Buttons** : you can drag and drop a **Button** from the palette and give id and Name. then set constraints to it like this:

After adding Accept Button you can select both Cancel and Accept buttons and right click go to the **chains** and select **Create Horizontal Chain**.



If you switch to the Split screen view to see the XML beside the Design Editor. The Design Editor allows you to preview your UI layout.

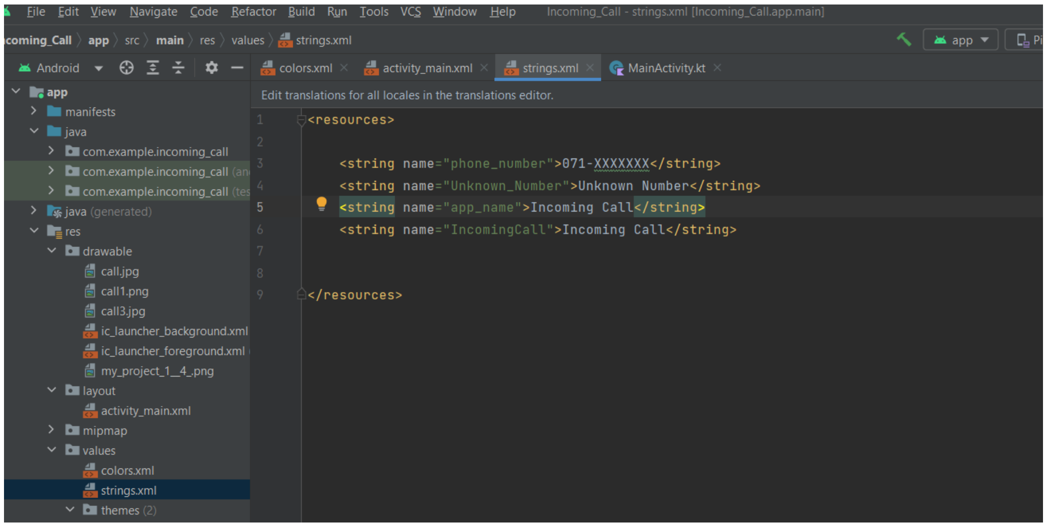
Then try clicking the different lines, one below the ConstraintLayout, and then one below the TextView, and notice that the corresponding view is selected in the **Design Editor**. The reverse works, too—for example, if you click on the TextView in the **Design Editor**, the corresponding XML is highlighted.



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#0A3E47"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="263dp"
        android:layout_height="35dp"
        android:backgroundTint="#E2E4F1"
        android:text="@string/Unknown_Number"
        android:textAlignment="center"
        android:textAppearance="@style/TextAppearance.Ap
        android:textColor="#B1D4F1"
        android:textDirection="firstStrongRtl"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textVi

    <TextView
        android:id="@+id/textview"
        android:layout_width="185dp"
```

Step 08: String.xml file: This file contains texts for all the TextViews widgets. Location of the strings.xml file:

App>res>values



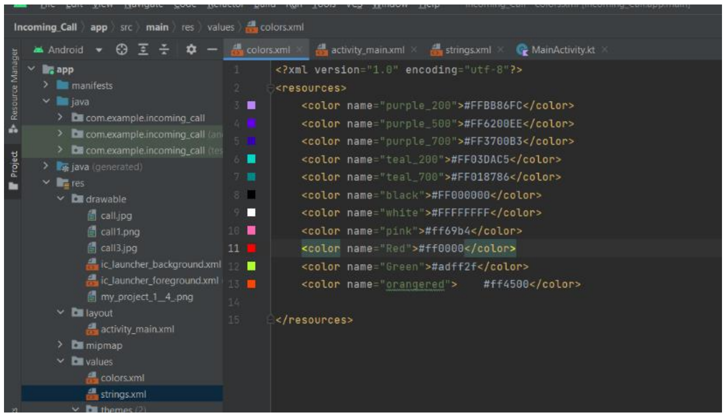
Edit translations for all locales in the translations editor.

```
<resources>

    <string name="phone_number">071-XXXXXXX</string>
    <string name="Unknown_Number">Unknown Number</string>
    <string name="app_name">Incoming Call</string>
    <string name="IncomingCall">Incoming Call</string>


</resources>
```

Step 09: colors.xml file: it is responsible to hold all the types of colors required for the application. The colors need to be defined in the **hex code format**.

Location of colors.xml file: ***app>res>values***

It should be like this:

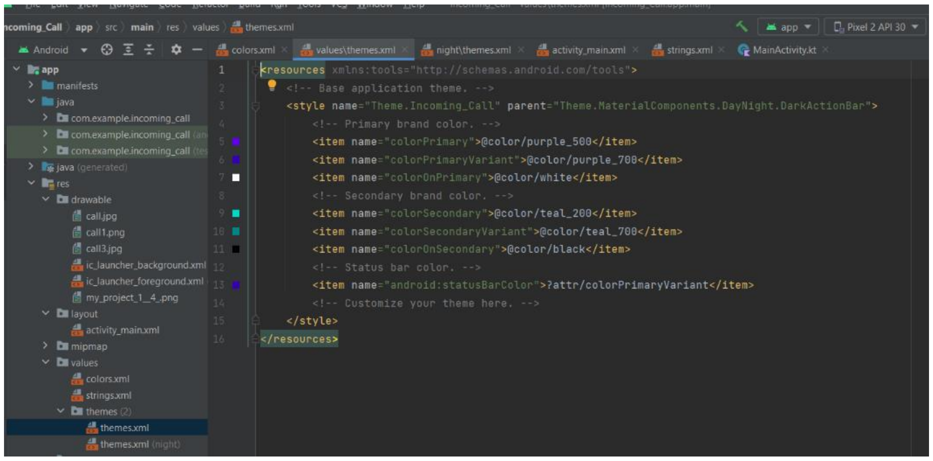


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="pink">#ff69b4</color>
    <color name="Red">#ff0000</color>
    <color name="Green">#adff2f</color>
    <color name="orangered">    #ff4500</color>

</resources>
```

Step 10: themes.xml: this file defines the base theme and customized themes of the application. It also used to define styles and looks for the UI of the application. By defining styles we can customize how the views or widgets look on the UI. Location of themes.xml file:

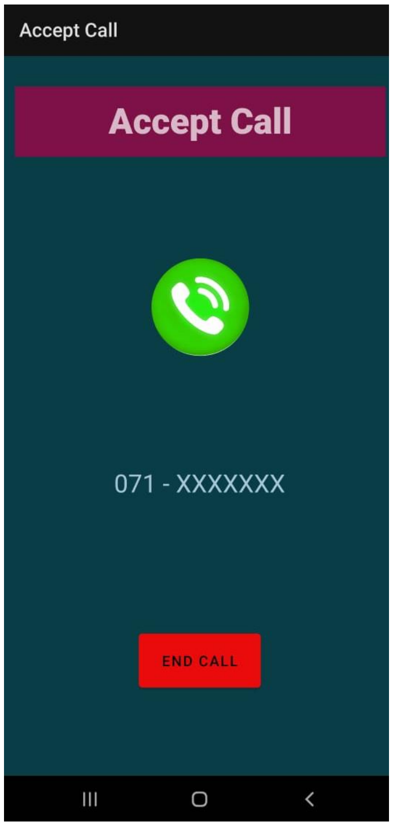
app>res>values

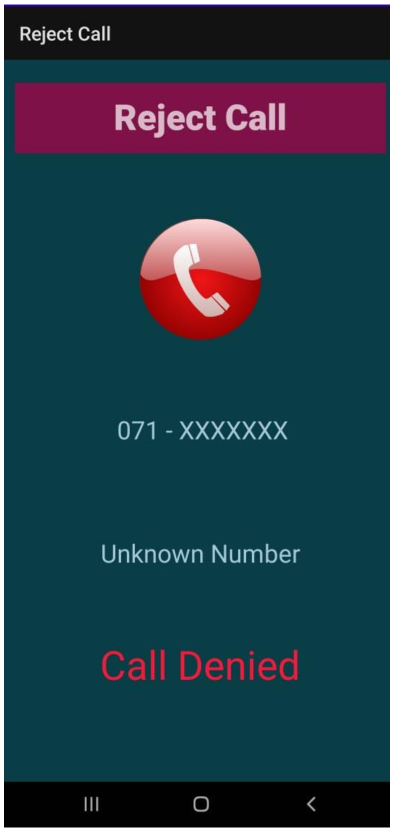
it will look like this:



```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Incoming_Call" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

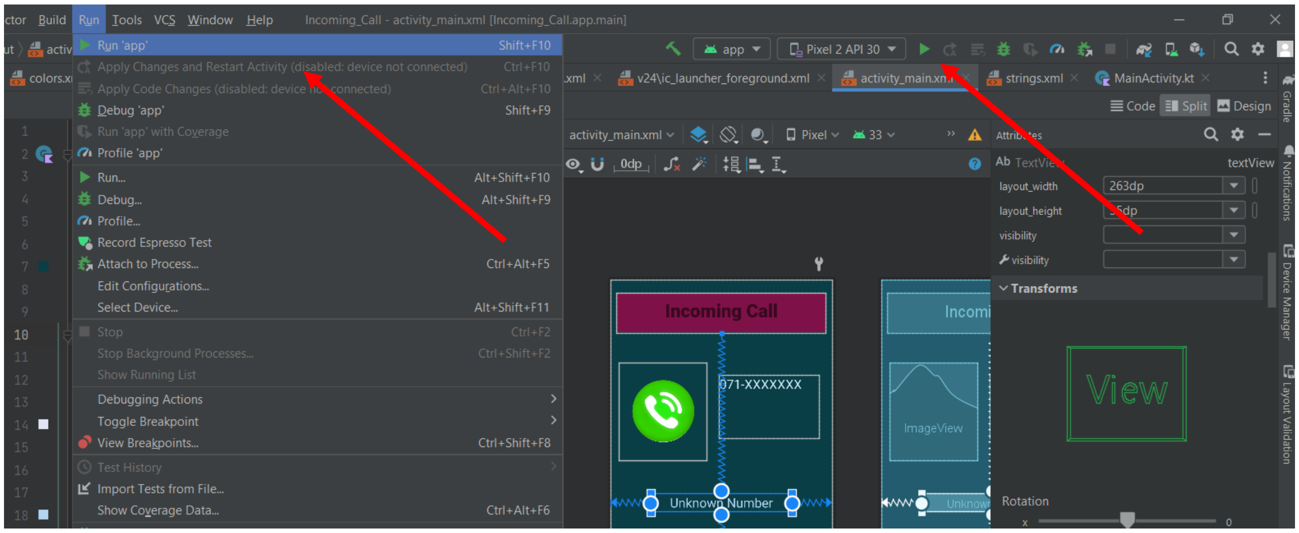
Step 11: In here we create additional two user interfaces for Accept Call and Decline Call. They will look like this:





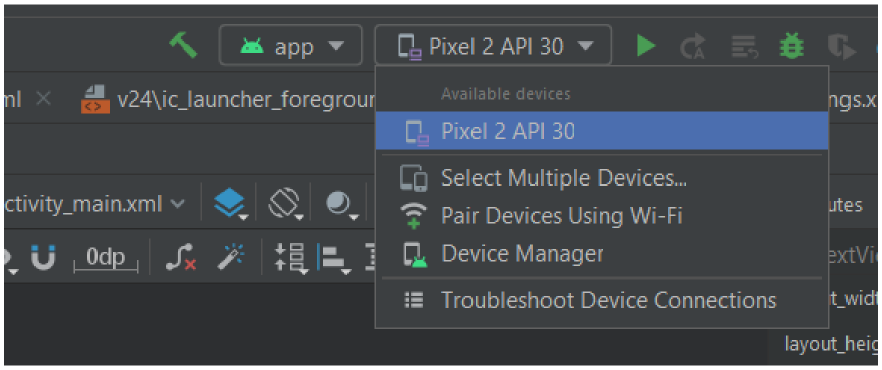
4 Run your User Interface on Emulator.

Step 01: In Android Studio , select Run > Run 'app', or click the Run icon in the toolbar. Simply you can give it **Shift +F10** .

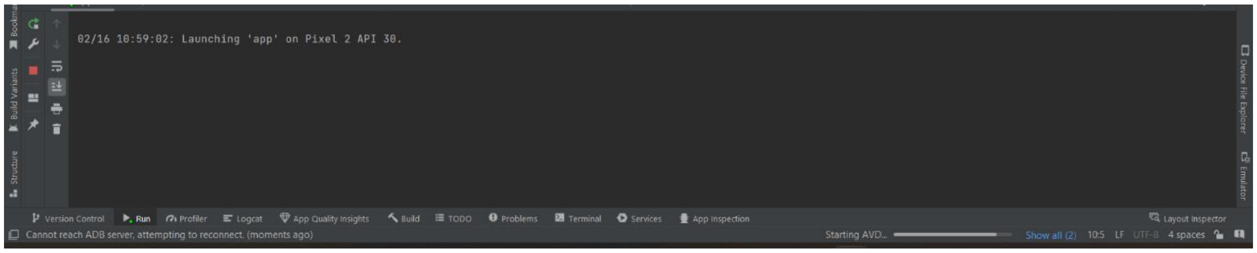


Then you can see the icon changes once your app is running.

Step 02: In Run > select Device, under the Available Devices, select the virtual device that you installed earlier. A dropdown menu also appears in the toolbar.



The emulator starts and boots in your computer. Depending on the speed of your computer, this may take a while. You can look in the small horizontal status bar at the very bottom of Android Studio for messages to see the progress.



```
02/16 10:59:02: Launching 'app' on Pixel 2 API 30.
```

```
Cannot reach ADB server, attempting to reconnect. (moments ago)        Starting AVD...
```

Step 03: Run your app on a device.

What you need:

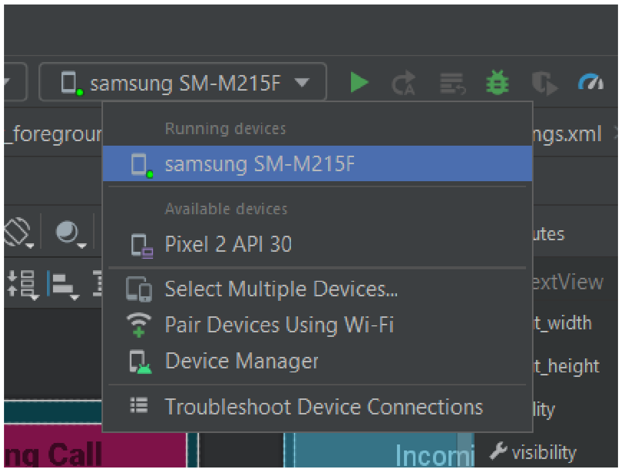
- An Android device such as phone or tablet.
- A data cable to connect your Android phone to your computer via the USB port.
- In your Android Phone you must turn on the USB Debugging .

Steps to enable USB Debugging in your Android Phone:

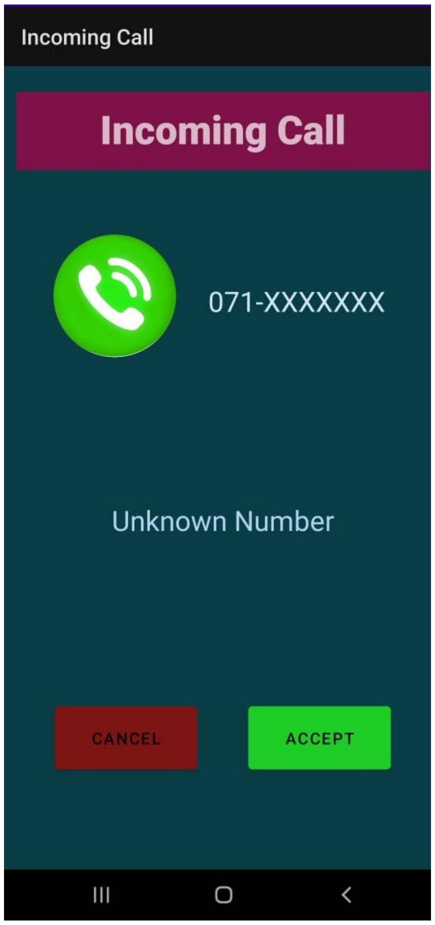
Open Settings>Developer Options and double tap on it.

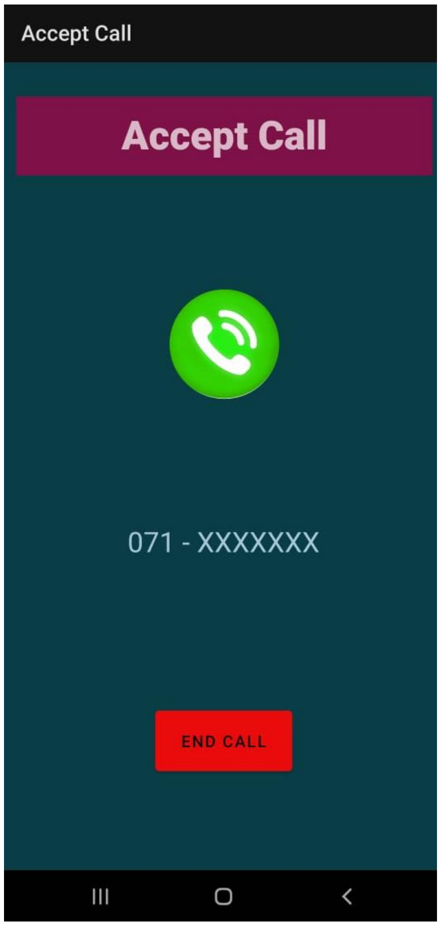
Then enable USB Debugging. You might need to allow USB from your mobile device. After that in Android Studio select your device in the dropdown menu which appears in the tool bar.

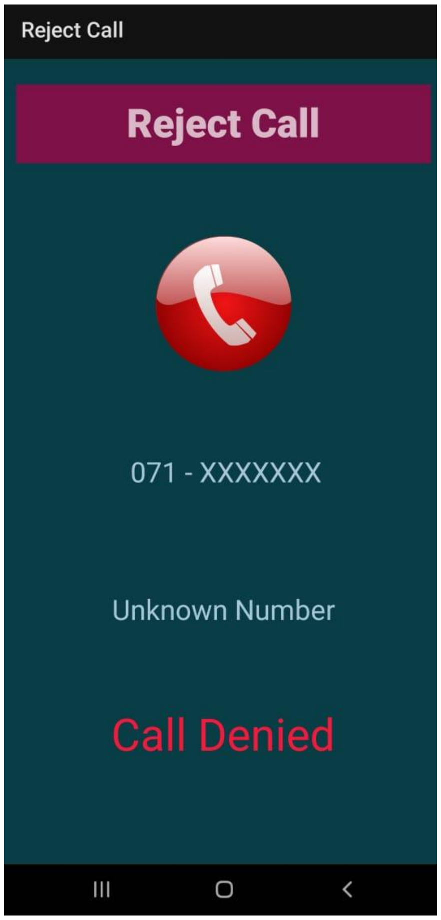
Android Studio installs the app on your mobile device and runs it.



Finally your UI will be look like this.


Incoming Call
Incoming Call
071-XXXXXXX
Unknown Number
CANCEL
ACCEPT


Accept Call
Accept Call
071 - XXXXXXX
END CALL


Reject Call
Reject Call
071 - XXXXXXX
Unknown Number
Call Denied