**Android Development using Kotlin**

**An android project consists**

- Kotlin/Activity files for the core logic of the app(It is used for functionalities or backend purpose)

- A resource folder for static contents such as images , strings , colors and layouts(XML files for frontend )

- An android **Manifest file** that defines essential app details so the OS can launch your app . It also contains the permission statements for your app such as internet access , access to camera , contacts etc .

- **Gradle** is the **android build tool** which grab all the kotlin , layouts and external libraries bundle up them and compile them to generate an installable APK(Android Application Package) file . Or , It is used for building and running the application

**Activity , Layouts , Views and View-Groups**

**activity** - It is a kotlin/java class . It is responsible for drawing an app user-interface and for receiving input events. Activities have a associative layout files .

**layout -** Layouts are the XML files that express what the app actually look likes . They define **views** i.e texts , images , buttons etc and define their location on the screen .

- To connect a layout to an activity ,

***setContentView(R.layout.activity\_main)***

***-* R** is a class containing the definitions for all resources of a particular application package

**views -** On the screen all the individual elements are called views and These views are the children of specific **Viewgroups(Ex:- LinearLayout , ConstraintLayout , RelativeLayout , ScrollView) .** ex : textViews , image views , edit texts , buttons , checkboxes , sliders , menus etc .

- All the views are the children of View class .

- **dp(density independent pixel)** is the unit for expressing location and dimensions of views

On 160 dpi screen , 1dp = 1pixel

On 480 dpi screen , 1dp = 3pixels

- The android devices will automatically handle the conversion of dp to pixels .

**Padding and Margin -** Padding provides space between border and content of an element . Margin provides space between the border and outer elements .

- We can extract and collect all the formatting of a view(like , fontsize , padding , margin , textfont) in a **style** , and by setting this style to any view we could achieve the same formatting for that view too without repetition .

**android:contentDescription="@string/yellow\_star" -** contentDescription is used by screenreaders to describe images .

- These activities and layouts are connected by a process known as **Layout Inflation .**

**Layout Inflation -** This process is triggered when the activity starts . During Layout Inflation , the views defined in the XML layout files are turned/inflated into Kotlin view-objects. Once this happened ,the activity can draw these objects on the screen and also can dynamically modify them .

**View Groups -** View groups are responsible for holding multiple views on screen and helping to specify their positions . Ex:- Linear , Relative , Constraints , Scroll View etc .

Linear Layout - It is a viewgroup , where we can arrange views horizontally or vertically .

Scrollview - it is used for scrollable content

Constraint Layout - Any design can be made using it .

**Interaction of XML views to the Kotlin file to make them functional/modifiable**

- While defining the views like <TextView> ,<ImgaeView> ,<Button> ,we give them specific ids , and through these ids we grab the views in kotlin files .

**<android:id=”@+id/view\_id”>**

- To grab the XML views into Kotlin , we first declare a variable of the corresponding view datatype like , Button , TextView etc and then store the views into them by using **findViewById(R.id.view\_id)** method

**lateinit var txtCount : TextView 🡨 we use lateinit keyword while not initializing the variable**

**lateinit var btnCount : Button**

**txtCount = findViewById(R.id.txtCount)**

**btnCount = findViewById(R.id.btnCount)**

- Now after grabbing the views into the kotlin file , we can do some functional and modificational stuffs with the views .

- In order to get your button clicked and perform some task/output , we can set an **setOnClickListener{}** method to it .

- **setOnclickListener{}** takes a **lambda function** as an argument which gets triggered every time the button is clicked . Like **Toast**

**btnCount.setOnClickListener{function}**

**-** We can pass a **Toast** to the setOnClicklistener() as Toast is message which pop-ups on the screen for short period of time .

- By , passing Toast we can able to pop-up a message every time when our button is clicked and we can achieve this as follows ,

**btnCount.setOnClickListener{Toast.makeText(**context**:this ,** text**:”popup\_message ”, Toast.LENGTH\_SHORT/LONG”).show()}**

- The first parameter of Toast.makeText() is **context ,** which is an object/instance of an abstract class **Context** and this object is obtained by calling **getApplicationContext()**

- In simple words **context ,** is an sbstract class acts as the interface to global information about an application environment or it is the way to get the info of your app environment .

**-** in XML **android:** , **tools:** , are different namespaces , which holds different attributes like android:**src ,** android:**layout\_gravity** etc

- **tools** namespace holds variety of attributes which enable design/preview/compile-time features as its attributes get removed on run-time and your app remains unaffected after compiling .

**Vector Drawables and Androidx Compatibility/Support Library for Lower API levels**

**Vector Drawables** - These are the vector graphics(images) defined in an XML file . These vector drawables are image scalable . Or , It can be scaled(resized) without loosing quality .

- Vector drawables are only applicable for API level 21 and onwards . Hence , to make it applicable for lower API levels we need to do some arrangements .

- Since Vector drawables don’t work below 21 API , gradle converts these vecor drawables into PNGs and make them compatible for lower API levels . But , in this way the size of the APK unnecessarily increases .

- To get rid of this extra-size issue in lower API levels , we use **Androidx compatibility library** which makes the vector drawables to support upto API 7 .

Following are the steps to use this library:-

1 . Add androidx vector support to the build.gradle

**vectorDrawables.useSupportLibrary = true**

2 . Add new namespace to the layout

**xmlns:app=”http://schemas.android.com/apk/res-auto”**

3 . Change the **android:src** to **app:srcCompat** in the layout file