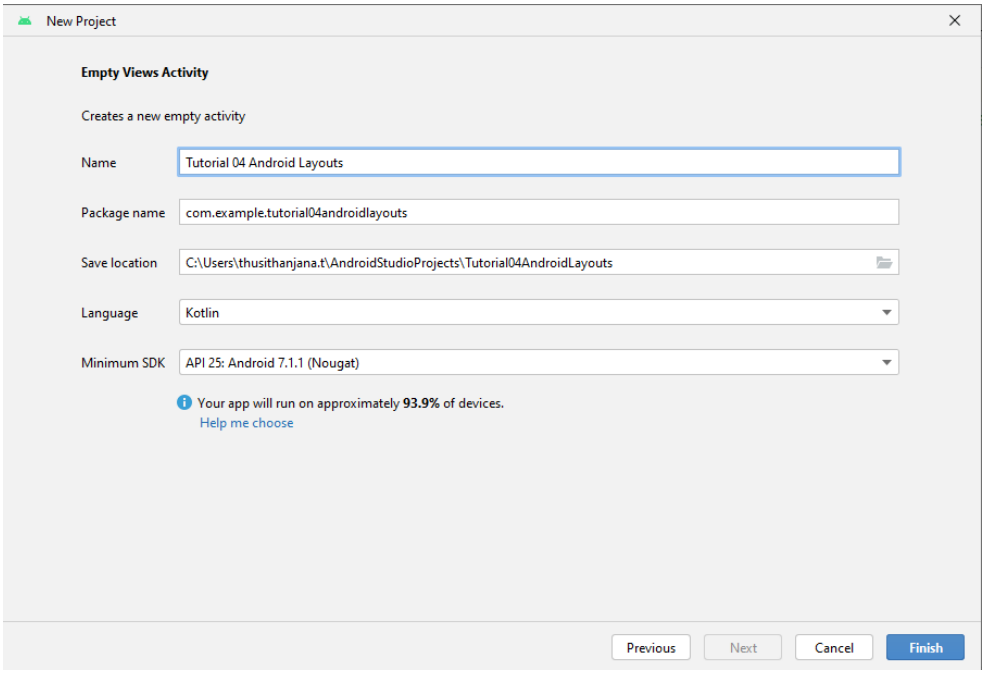
FACULTY OF COMPUTING

IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT
2024 - Tutorial

Android XML Layouts

In this tutorial, you'll learn the fundamentals of creating user interfaces using XML layouts in Android app development. XML layouts are essential for defining the structure and appearance of your app's user interface.

1. Start a new Android project named “Tutorial 04 Android Layouts”.



2. Remove the existing textview from the activity_main.xml.
3. Add the following Views to the screen and observe the UI.
    a. TextView
    b. EditText
    c. Button
    d. ImageView

4. Observe the attributes of each view.
    a. id
    b. text
    c. layout_width, layout_height
    d. textColor
    e. background
    f. hint

Android units and usages

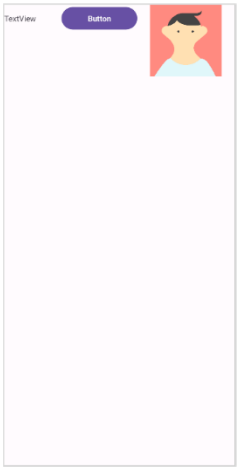
1. **dp (density-independent pixels)**:
    - Commonly used for defining sizes of UI elements.
    - Ensures consistent sizes across different screen densities.
    - Example: **android:layout_width="20dp"**
2. **sp (scaled pixels)**:
    - Used for specifying text sizes.
    - Adjusts the text size based on the user's font size preference.
    - Example: **android:textSize="14sp"**
3. **px (pixels)**:
    - Represents actual pixels on the screen.
    - Use sparingly as it can lead to inconsistent sizes on different screen densities.
    - Example: **android:padding="8px"**
4. **pt (points)**:
    - Often used for text sizes, similar to sp.
    - 1 pt = 1/72 of an inch.
    - Example: **android:textSize="12pt"**
5. **in (inches)**:
    - Used less frequently for specifying sizes in inches.
    - Example: **android:layout_width="2in"**
6. **mm (millimeters)**:
    - Used less frequently for specifying sizes in millimeters.

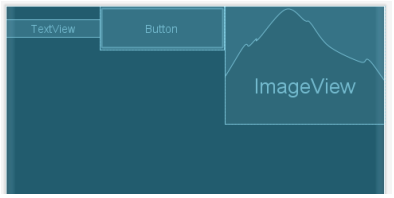
- Example: **android:layout_height="50mm"**

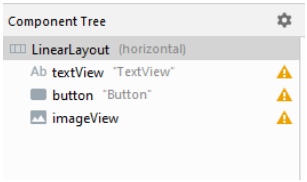
7. **% (percentage)**:
    - Represents a percentage of the parent's size.
    - Used to create responsive layouts that adapt to different screen sizes.
    - Example: **android:layout_weight="0.3"**
8. **wrap_content**:
    - Dynamic sizing based on the content's dimensions.
    - Used to make a view's size wrap around its content.
    - Example: **android:layout_width="wrap_content"**
9. **match_parent (fill_parent)**:
    - Fills the available space in the parent view.
    - Used to make a view take up all available space.
    - Example: **android:layout_height="match_parent"**

Layouts

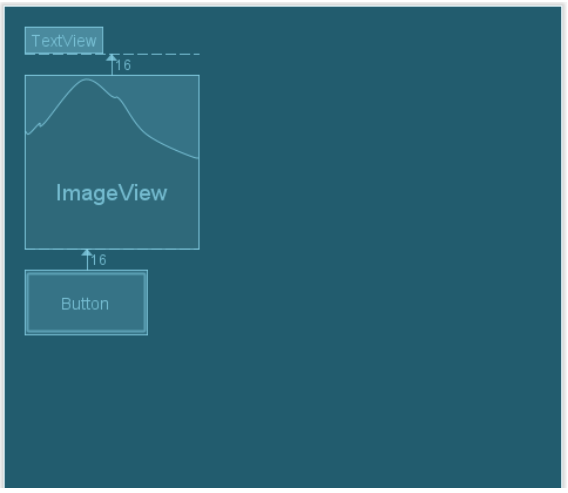
1. **Linear Layout**: A Linear Layout is a type of layout in Android that arranges its child views in a linear, either horizontal or vertical, fashion. Child views are placed one after the other, and you can control their arrangement using attributes like weight, gravity, and margins. It's simple and easy to use for basic UI structures.







2. **Relative Layout**: A Relative Layout is a type of layout that arranges its child's views relative to each other or the parent layout. Views are positioned based on their relationships to other views or the parent, allowing for more complex and flexible UI designs compared to Linear Layouts.



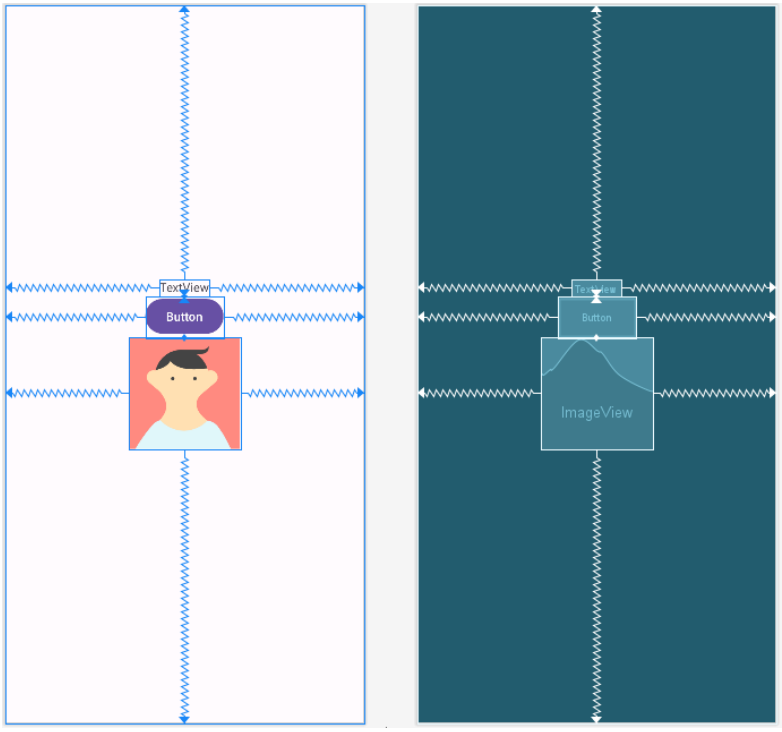

```
<TextView
   android:id="@+id/textView2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="TextView"
   android:layout_marginStart="16dp"
   android:layout_marginTop="16dp"
   />

<ImageView
   android:id="@+id/imageView2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   tools:srcCompat="@tools:sample/avatars"
   android:layout_below="@id/textView2"
   android:layout_marginStart="16dp"
   android:layout_marginTop="16dp"
   />

<Button
   android:id="@+id/button2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Button"
   android:layout_below="@id/imageView2"
   android:layout_marginStart="16dp"
   android:layout_marginTop="16dp"
   />
```

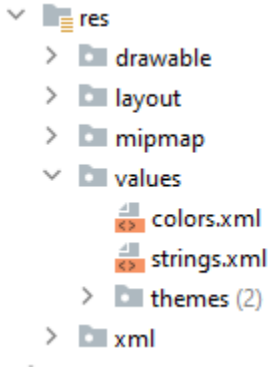
3. **Frame Layout**: A Frame Layout is a simple layout that allows you to place a single child view within it. It's often used for single-purpose or layered UI components. Views placed in a Frame Layout are stacked on top of each other, so you can control their visibility and positioning in a straightforward manner. We will be using Frame Layouts in Fragments

4. **Constraint Layout**: A Constraint Layout is a powerful and flexible layout in Android that allows you to create complex UIs with a flat view hierarchy. It relies on constraints to define the position and size of child views relative to each other and the parent layout. Constraint Layout is great for designing responsive and adaptive UIs that work well across different screen sizes and orientations.



Using Resources

String Resources

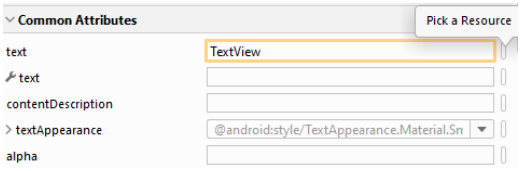


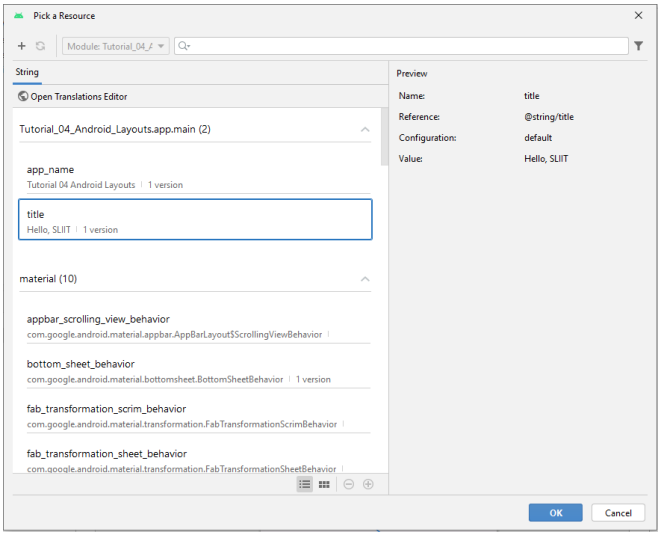
Will be using string resources to keep all the readable text in one location. Hardcoding text in xml and Kotlin files will make it harder to maintain the projects. You may need to change multiple files for one change of a text element. Separating text from the UI will allow you to introduce multi-language support to the application as well.

Add the following to the strings.xml file.

```
<string name="title">Hello, SLIIT</string>
```

Use it on the text field as follows.



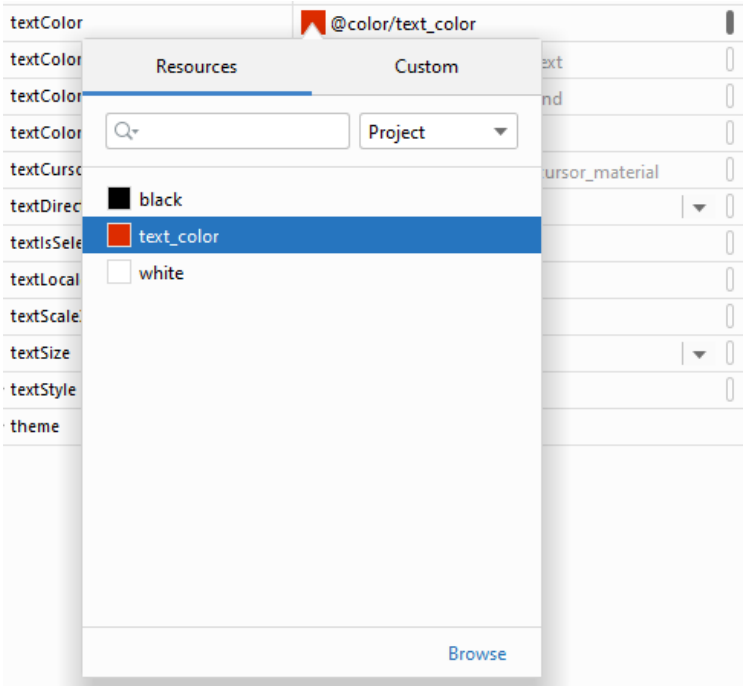


Color Resources

Add the following to the colors.xml file.

```
<color name="text_color">#DD2C00</color>
```

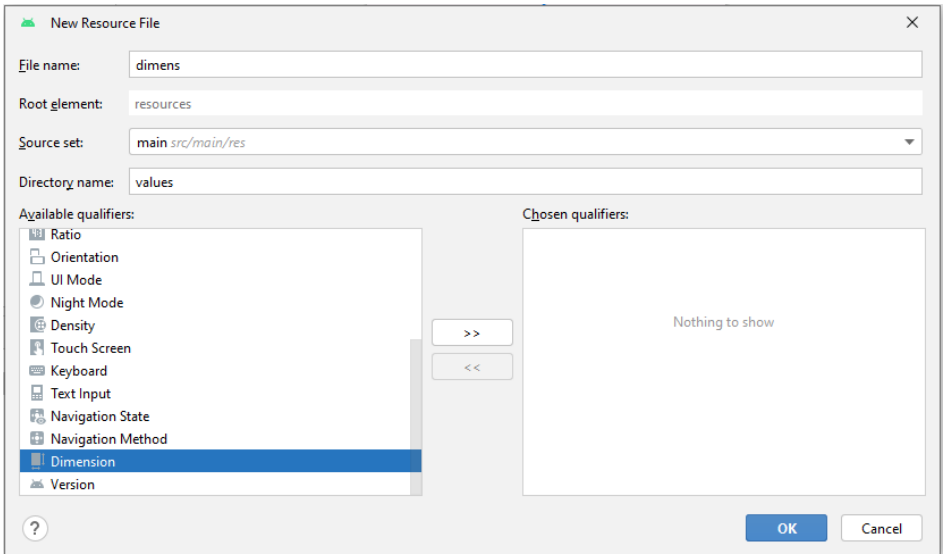
Use it on the text field.



Dimension Resources

This file is not available by default. You must manually create it. You can maintain the dimensions in this file, and it makes the designing process easy. No need to add them manually to each element.

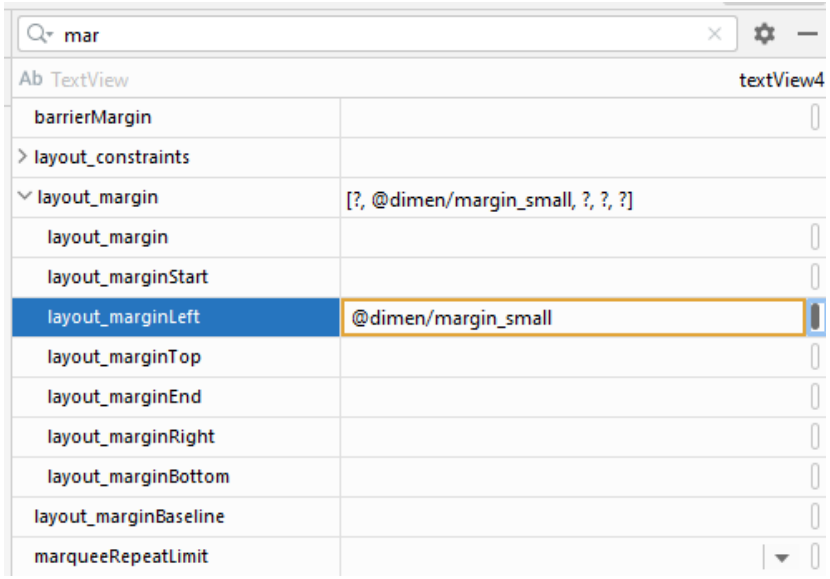
Create the dimens.xml file.



Add the following to the file.

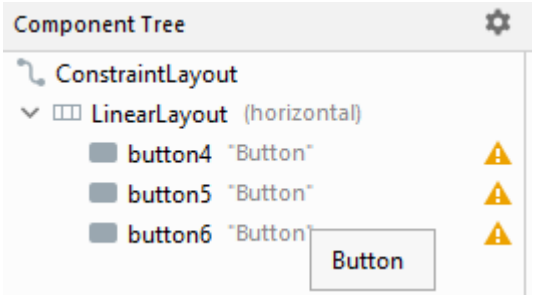
```
<dimen name="margin_small">16dp</dimen>
```

Use it on the text field.



Creating Complex Layouts

You can use nested layouts to create complex layouts. One example would be using linear layouts to add a group of buttons in constraint layouts.

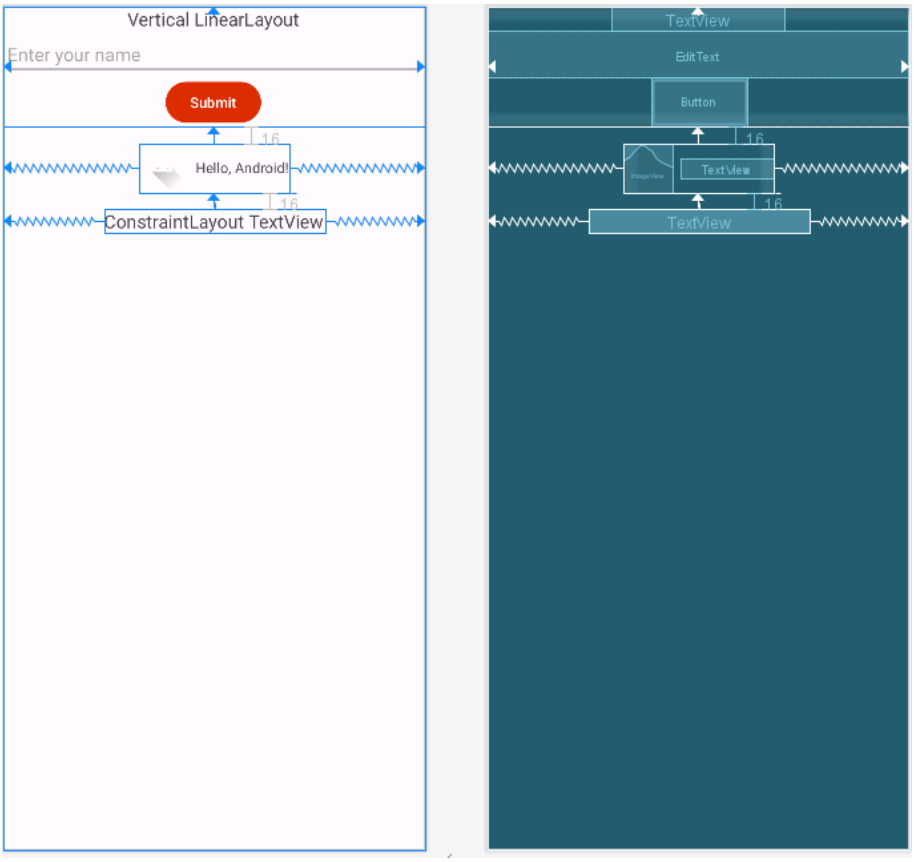


Best Practices for XML Layouts

- Using Descriptive IDs
  - txtTitle
  - btnOk
  - edtUserName
- Keeping the Hierarchy Simple
- Using Styles and Themes

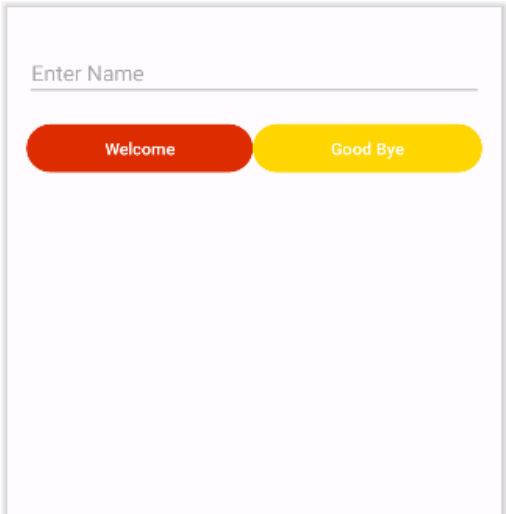
Exercise

Design the following UI.

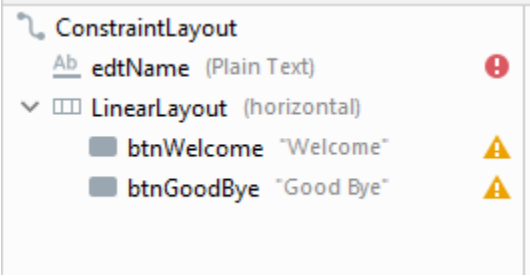


Handling User Interactions

Design the following UI.



Use descriptive ids.



Create objects in the Kotlin file for each view.

```
lateinit var edtName:EditText
lateinit var btnWelcome:Button
lateinit var btnGoodBye:Button
```

initialize them in the onCreate method.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    edtName = findViewById(R.id.edtName)
    btnWelcome = findViewById(R.id.btnWelcome)
    btnGoodBye = findViewById(R.id.btnGoodBye)
}
```

Set on click listeners to all buttons.

```
btnWelcome.setOnClickListener {

}

btnGoodBye.setOnClickListener {

}
```

create two functions as follows. And call them in the OnClickListener methods.

```
fun sayHello(){

}

fun sayGoodBye(){

}
```

Now we can use toast messages to display the messages to the user.

```
class MainActivity : AppCompatActivity() {
    lateinit var edtName:EditText
    lateinit var btnWelcome:Button
    lateinit var btnGoodBye:Button
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        edtName = findViewById(R.id.edtName)
        btnWelcome = findViewById(R.id.btnWelcome)
        btnGoodBye = findViewById(R.id.btnGoodBye)

        btnWelcome.setOnClickListener {
            sayHello()
        }

        btnGoodBye.setOnClickListener {
            sayGoodBye()
        }
    }

    fun sayHello(){
        Toast.makeText(this, "Hello, ${edtName.text.toString()}",Toast.LENGTH_LONG).show()
    }

    fun sayGoodBye(){
        Toast.makeText(this, "Hello, ${edtName.text.toString()}",Toast.LENGTH_LONG).show()
    }
}
```