MODUL 1
Introduction to Android Development

THEME DESCRIPTION

An introduction to Android, where you will set up your environment and focus on the fundamentals of Android development.

WEEKLY LEARNING OUTCOME (SUB-LEARNING OUTCOME)

Students will have gained the knowledge required to create a basic Android app from scratch and install it on a virtual or physical Android device.

TOOLS/SOFTWARE USED

- Android Studio

CONCEPTS

Android Studio Installation and Setup

Before you start this module, you will need to install Android Studio **Electric Eel (or higher)**, which is the software you will be using throughout the lab modules. You can download Android Studio from https://developer.android.com/studio.

On **macOS**, launch the **DMG** file and drag and drop Android Studio into the Applications folder. Once this is done, open Android Studio. On **Windows**, launch the **EXE** file. If you're using **Linux**, unpack the **ZIP** file into your preferred location. Open your Terminal and navigate to the androidstudio/bin/ directory and execute studio.sh.

Kotlin

Kotlin is the main coding language that you will be using throughout the lab modules. It is the base of what you will be creating using the Android Studio IDE that you've just installed. It is superior to Java in terms of verbosity, handling null types, and adding more functional programming techniques.

PRACTICAL STEPS

Part 1 - Creating a new project (Hello World)

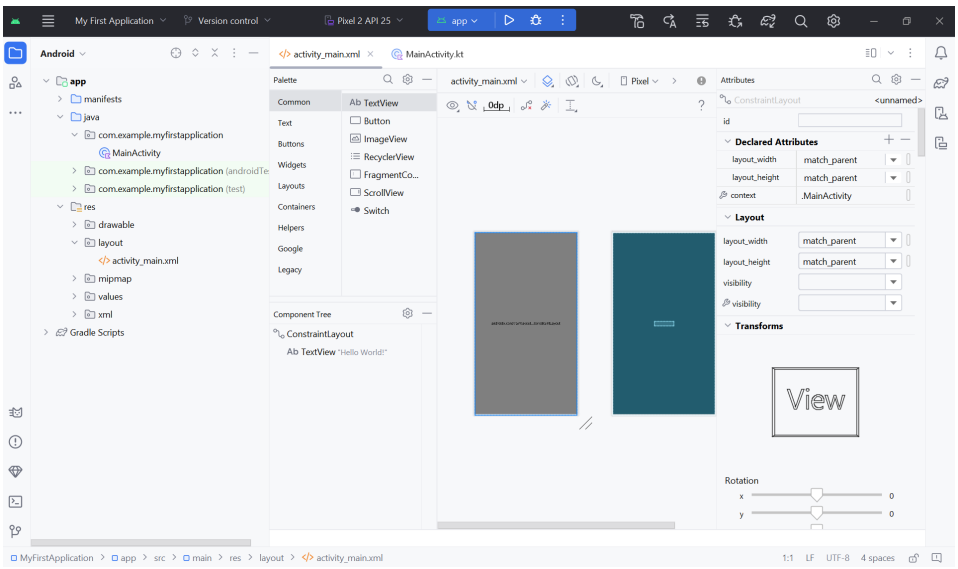
1. Open Android Studio and click **New Project.**
2. You will see a bunch of options for activity templates. You can call an activity a page for your application. Choose the **Empty Activity** to start with.
3. Name your application **"LAB_WEEK_01"**. The other fields will be automatically updated based on your application name.

4. Set the minimum SDK to **“API 24: Android 7.0 (Nougat)”** as it runs on most android devices. SDK stands for software development kit which determines what version of android your application will be based on.

ANDROID PLATFORM VERSION		API LEVEL	CUMULATIVE DISTRIBUTION
4.4	KitKat	19	
5.0	Lollipop	21	99.5%
5.1	Lollipop	22	99.2%
6.0	Marshmallow	23	97.7%
7.0	Nougat	24	95.4%
7.1	Nougat	25	93.9%
8.0	Oreo	26	92.4%
8.1	Oreo	27	90.2%
9.0	Pie	28	84.1%
10.	Q	29	72.2%
11.	R	30	54.4%
12.	S	31	31.3%
13.	T	33	15.0%

Last updated: May 30, 2023

5. Click **Finish**, and let your android application build itself. Depending on your system, it may take quite some time because **Gradle** build is a heavy process. Gradle is a build automation tool known for its flexibility to build software.
6. Just like other IDEs, you can see the file structures on the left side and the main editor on the right side.



7. There are 7 important folders present in the file structure:
    - **manifests** - Stores important descriptions about your application for runtime.
    - **java** - The brain of your application.
    - **res/layout** - Stores all your activity layouts.
    - **res/drawable** - Stores all your image assets and such.
    - **res/mipmap** - Stores your application icon.
    - **res/values** - Stores all your resource values (texts, colors, etc.).
    - **Gradle Script** - Stores build configuration (app version, dependencies, etc.).

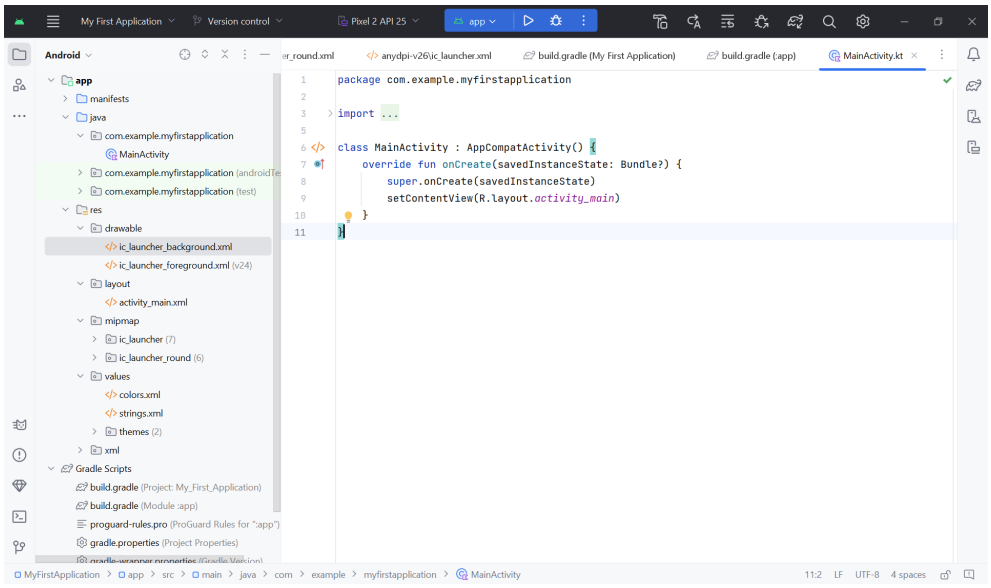
8. First, let's look at the **manifests** file. Click your **AndroidManifest.xml**, and it should look like this.

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

   <application
       android:allowBackup="true"
       android:dataExtractionRules="@xml/data_extraction_rules"
       android:fullBackupContent="@xml/backup_rules"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/Theme.MyFirstApplication"
       tools:targetApi="31">
       <activity
           android:name=".MainActivity"
           android:exported="true">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

9. The **manifests** file describes essential information about your app to the Android build tools, the Android operating system, and Google Play. It is also written as an **XML** file.
10. For now, you can ignore all the attributes in the **application** element, instead you can focus more on the **activity** element. The **activity** element defines an activity to be recognized by the system app. For Instance if you have 3 activities, then you need to

define 3 separate activity elements. The starting point activity is marked with the **“android.intent.action.MAIN” action** element.

11. **Manifests** file is also used to define **permissions** which your app is allowed to use (camera, internet, etc.).
12. Next, click on **MainActivity.kt** in the **java folder**, This file controls all of the logic for your main activity. It’ll look like this in the editor.

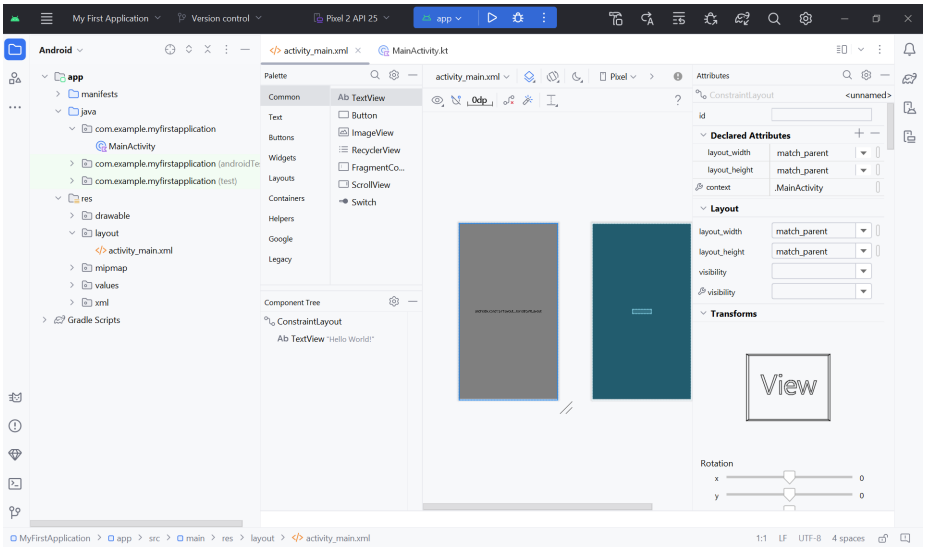


13. After experiencing **Java** in **Object Oriented Programming**, **Kotlin** shouldn’t feel that much different, just way shorter and more efficient.

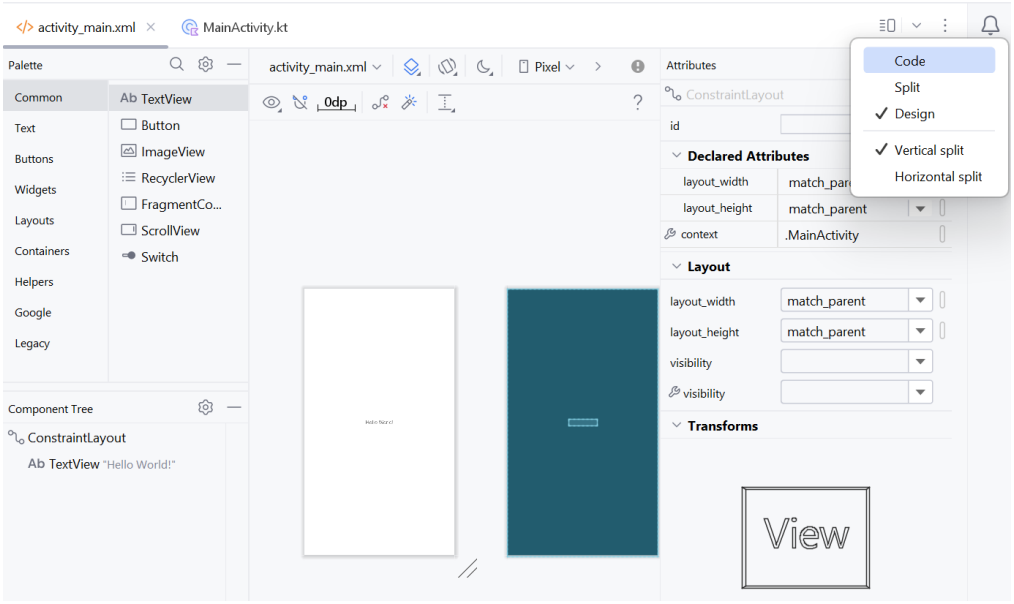
```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
   }
}
```

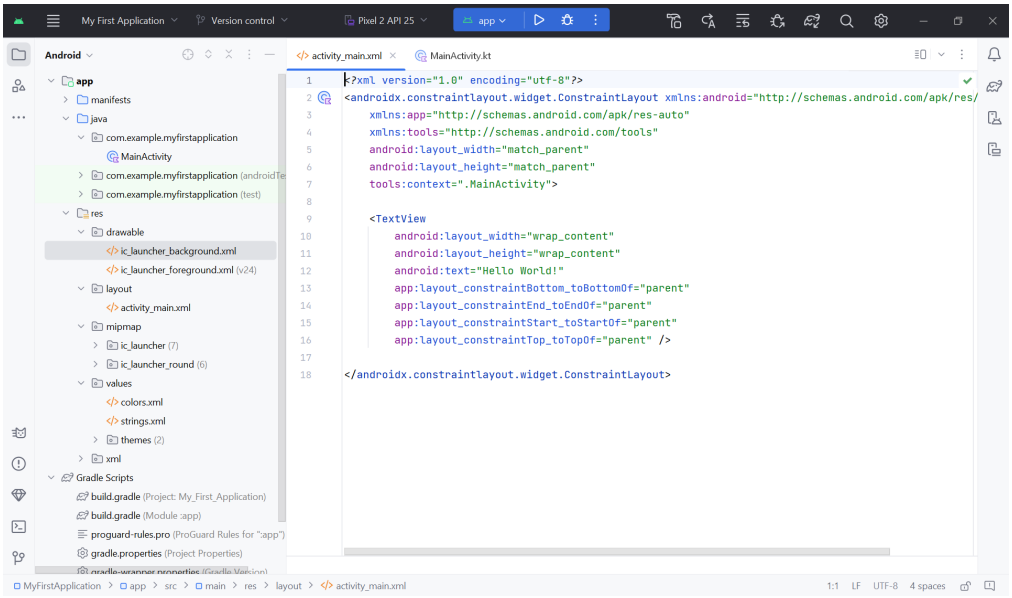
14. Here’s the explanation for the code above:
    - **MainActivity** - The name of your activity.
    - **AppCompatActivity** - Important library for your activity.
    - **onCreate** - Triggers when your activity starts at runtime.
    - **savedInstanceState** - Stores previous state data.
    - **setContentView(R.layout.activity_main)** - Shows your started activity onto the user screen (in this case, it references **activity_main.xml** from the **res/layout** folder).

15. Now click on **activity_main.xml** in the **res folder**, this file controls the layout of the main activity. It’ll look like this in the editor.



16. In android studio, **xml files** can be edited in 2 different ways; **code** or **design**. You can try enabling the other option from the menu at the top right of your editor (or you can try split mode).



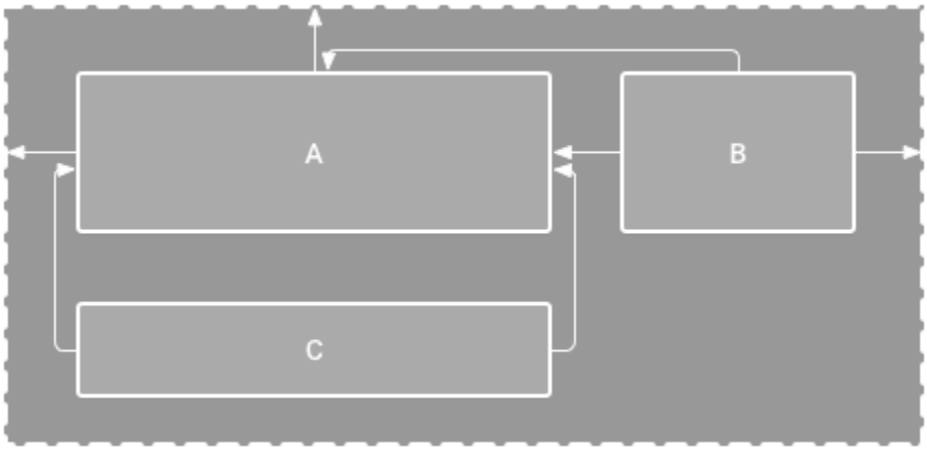


17. **XML** is also not that much different from **HTML**. They are both **tag**-based.

```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>
```

18. Here's the explanation for the code above:
    - **ConstraintLayout** - One of the **layouts (view groups)** that you can use for your app. As its name says, it is constraint-based.
    - **xmlns:android, xmlns:app, xmlns:tools** - Important namespaces.
    - **tools:context** - Defines the associated activity for the layout (in this case MainActivity.kt from the java folder).
    - **TextView** - One of the **widgets (views)** that you can use for your layout. This one lets you show a text onto your screen.
    - **android:layout_width** - Defines the width of the element.
    - **android:layout_height** - Defines the height of the element.
    - **android:text** - Defines the shown text for the element.
    - **app:layout_constraint** - Defines the constraints of which the view is bound to.

19. Here are some basic view groups and views that you might want to know before building your application.
    View Groups:
    - ConstraintLayout
    - LinearLayout
    - RelativeLayout
    - GridLayout
    - TableLayout



ConstraintLayout

Views:

- **TextView** - Defines a text
- **Button** - Defines a clickable button
- **ImageView** - Defines an image
- **EditText** - Defines an input field

20. Next, let's check the **res/values** folder. Inside the folder, there are 3 XML files:
    - **Colors.xml** - Stores all color resources

```
<?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>
</resources>
```

    - **Strings.xml** - Stores all string resources

```
<resources>
   <string name="app_name">My First Application</string>
   <string name="error_message">Error!</string>
</resources>
```

    - **Themes.xml** - Stores all style resources

```
<resources xmlns:tools="http://schemas.android.com/tools">
   <!-- Base application theme. -->
```

```
    <style name="Theme.MyFirstApplication"
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>

    <!-- Your custom styles -->
    <style name="text_input"
parent="Widget.MaterialComponents.TextInputLayout.OutlinedBox">
        <item name="android:layout_margin">8dp</item>
    </style>
</resources>
```

21. Resources are values stored as **variables** that you can use throughout your app. With resources, you can avoid using any hard coded values in your application. You will see the implementation in the next tutorial.

22. Lastly, let's look at the **Gradle** file. Click on the **build.gradle (Module :app)** file. It should look similar to this.

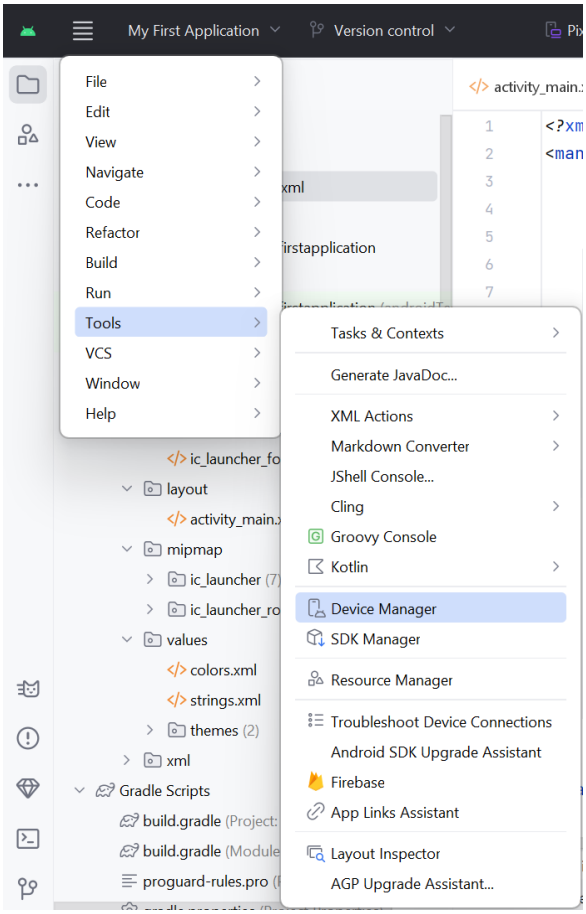
```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'com.example.myfirstapplication'
    compileSdk 33

    defaultConfig {
        applicationId "com.example.myfirstapplication"
        minSdk 24
        targetSdk 33
        versionCode 1
        versionName "1.0"
```

```
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.8.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
}
```

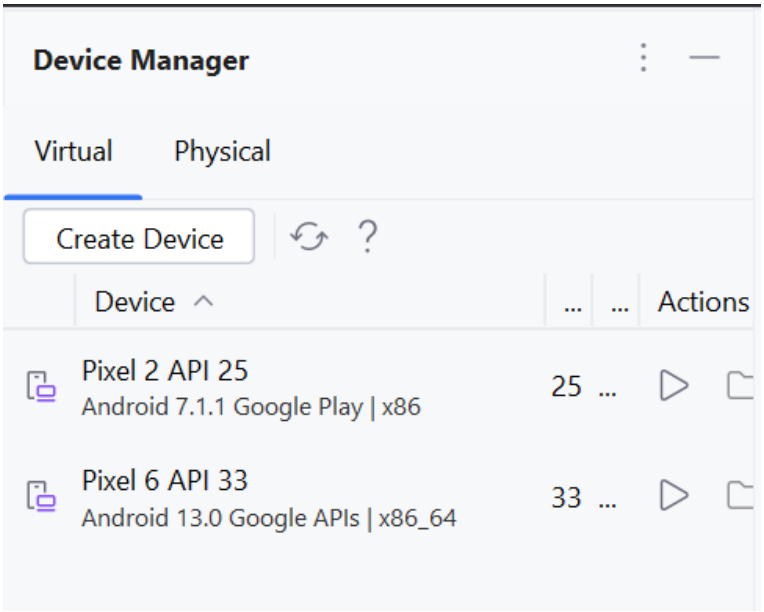
23. For now, you can ignore everything and focus on the **dependencies**. This section defines all your libraries used for your application. If you add a new library to the java folders, you don't need to write it manually in the **Gradle** file again as it does it automatically for you. Most of the time, you rarely need to touch this file.

Part 3 - Running your application

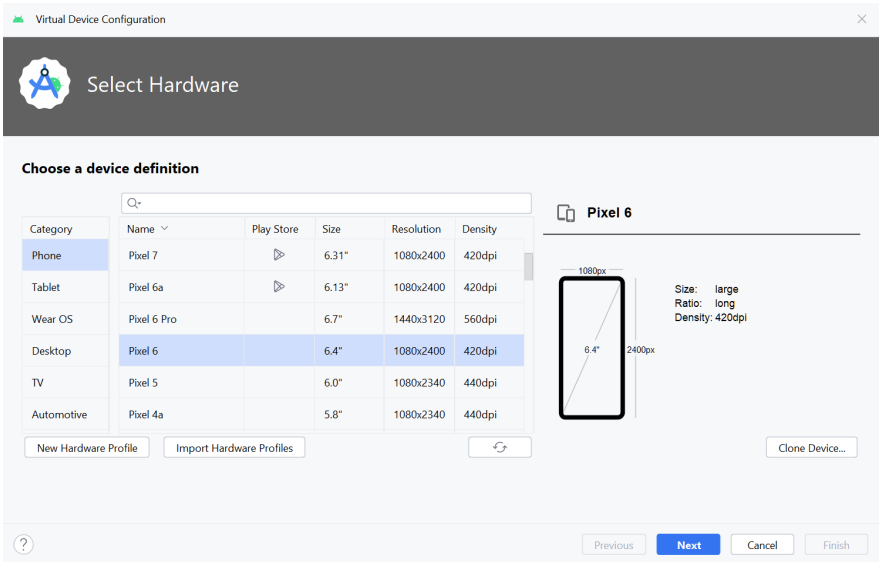
1. Go to **Tools** > **Device Manager**.



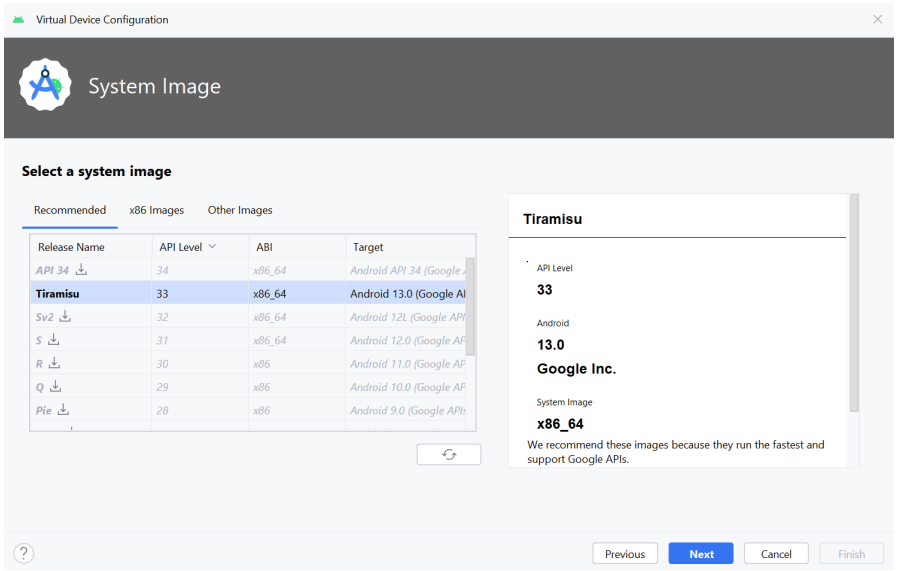
2. There are 2 ways of running your application: **virtually (Emulator)** or **physically (Physical device)**.



3. First let's try the virtual way. Choose the **Virtual Tab** and click **Create Device**.
4. Pick the **Pixel 6** phone and click **Next**.

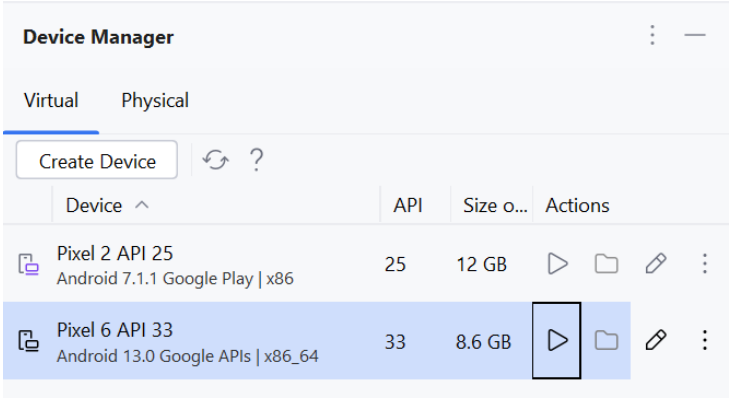


5. Download and pick the **Tiramisu or API 34** system image and click **Next**.

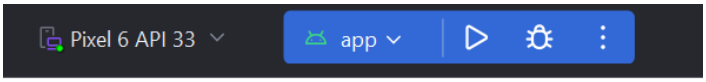


6. Click **Finish**.

7. Now start your emulator by clicking the **Play** button for the created device.

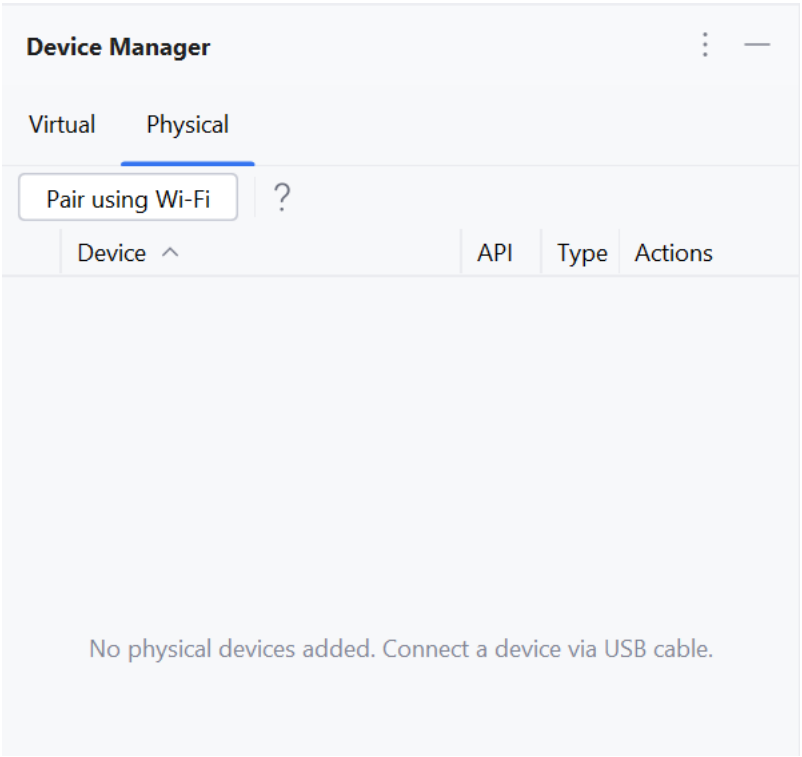


8. After the emulator has finished loading, click the **Play** button to start your application at the top of your editor.



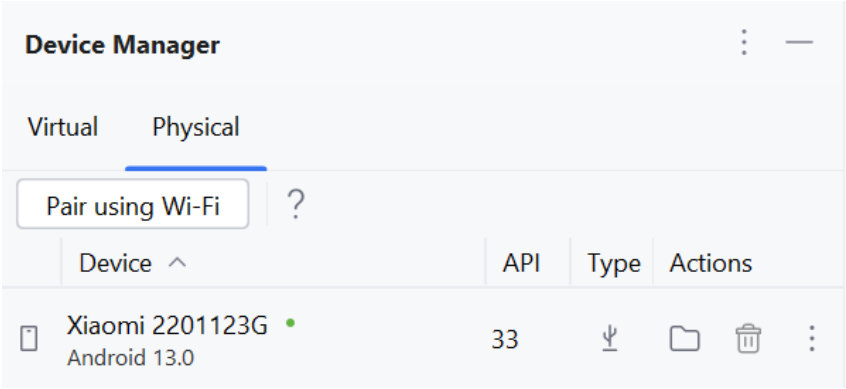
9. Congratulations, you have successfully run your app virtually.

10. Now let's try running it physically. From the device manager, choose the **physical tab**.

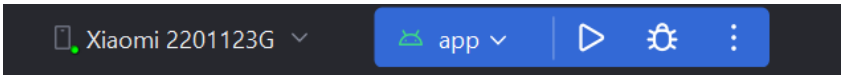


11. For faster pairing and debugging, **USB cable** is recommended.
12. Now try connecting your PC/Laptop with your Android phone through a **USB cable**.
13. After it's connected, go to your phone and enable the **developer mode**.

14. Depending on what Android phone you're using, the steps may vary, so you might want to do some research.
15. After **developer mode** is activated, now go to **developer settings > USB debugging** and set it to **True**.
16. Still in the **developer settings > Install via USB** and set it also to **True**.
17. Now your device should appear in your device manager.



18. Simply run your application by clicking the **Play** button at the top of your editor.



19. Congratulations, you have successfully run your app on your own physical device.

Part 4 - findViewById and click events

1. After you managed to create your very own "Hello world" app, let's create something more advanced. Start off by making the application front-end first.
2. Go to **activity_main.xml**. Modify your **TextView** into the code below.

```
<TextView
    android:id="@+id/name_display"
    style="@style/text_display"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/name_value"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

3. Below the **TextView**, add an input field using the **TextInputLayout** viewgroup. Generally, you can use **EditText** instead, but **TextInputLayout** provides the input field with an additional floating label.

```
<TextView ... />
<com.google.android.material.textfield.TextInputLayout
   android:id="@+id/name_wrapper"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   style="@style/text_input"
   android:hint="@string/name_hint"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toBottomOf="@id/name_display">
   <com.google.android.material.textfield.TextInputEditText
       android:id="@+id/name_input"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:inputType="text"/>
</com.google.android.material.textfield.TextInputLayout>
```

4. Below the **TextInputField**, add a button using the **Button** view.

```
<com.google.android.material.textfield.TextInputLayout ... />
<Button
   android:id="@+id/name_submit"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   style="@style/button"
   android:text="@string/button_name_value"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toBottomOf="@id/name_wrapper" />
```

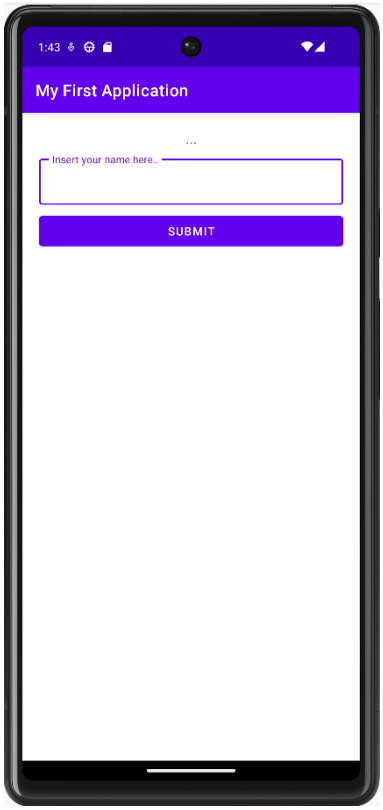
5. You may notice that all the styles and strings are highlighted in red. That's because we haven't defined them yet. Go to **strings.xml**, and add the necessary string resources.

```
<resources>
   <string name="app_name">My First Application</string>
   <string name="name_hint">Insert your name</string>
   <string name="name_value">...</string>
   <string name="button_name_value">Submit</string>
   <string name="name_greet">My name is: </string>
   <string name="name_empty">Field is empty!</string>
</resources>
```

6. Now go to **themes.xml**, and add the necessary styles.

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.MyFirstApplication"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        ...
    </style>

    <!-- Your custom styles -->
    <style name="text_input"
parent="Widget.MaterialComponents.TextInputLayout.OutlinedBox">
        <item name="android:layout_margin">8dp</item>
    </style>
    <style name="button">
        <item name="android:layout_margin">8dp</item>
        <item name="android:gravity">center</item>
    </style>
    <style name="text_display"
parent="@style/TextAppearance.MaterialComponents.Body1">
        <item name="android:layout_margin">8dp</item>
        <item name="android:gravity">center</item>
        <item name="android:layout_height">40dp</item>
    </style>
    <style name="screen_margin">
        <item name="android:layout_margin">12dp</item>
    </style>
</resources>
```

7. Now run your app, and it should look like this.



8. The front-end itself is done, now let's move on to the front-end logic. Go to **MainActivity.kt**.
9. First, you need to bind the UI elements from the front-end with the logic in your kotlin file. You can do that by using the **findViewById** command which gets the reference of any UI elements based on their **id**. Add the code below inside the **onCreate** function.

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

    val nameDisplay = findViewById<TextView>(R.id.name_display)
    val nameSubmit = findViewById<Button>(R.id.name_submit)
}
```

10. Next, we need to set a **Click Listener** for the button that we've just created. You can do that by using the **.setOnClickListener** command. Add the code below continuing the previous code.

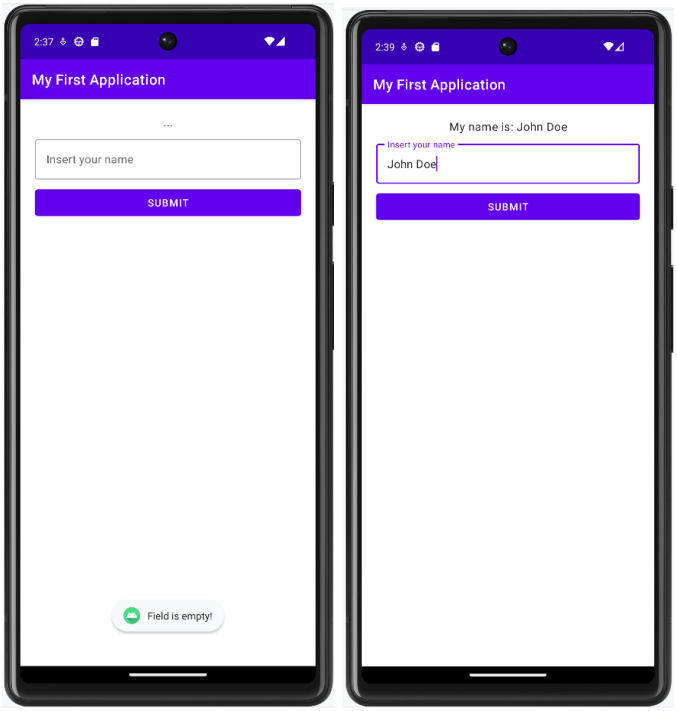
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
```

```
    val nameDisplay = findViewById<TextView>(R.id.name_display)
    val nameSubmit = findViewById<Button>(R.id.name_submit)

    nameSubmit.setOnClickListener {
        val nameInput = findViewById<TextInputEditText>(R.id.name_input)
        ?.text.toString().trim()
    }
}
```

11. Now, the input field needs to be checked if it's empty or not. If it is, then send an error message using the **Toast** library, but if it's not, then update the **TextView** with the new value. Add the code below continuing the previous code.

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

    val nameDisplay = findViewById<TextView>(R.id.name_display)
    val nameSubmit = findViewById<Button>(R.id.name_submit)

    nameSubmit.setOnClickListener {
        val nameInput = findViewById<TextInputEditText>(R.id.name_input)
        ?.text.toString().trim()
        if(nameInput.isNotEmpty()){
            nameDisplay?.text = getString(R.string.name_greet).plus("
").plus(nameInput)
            // or you can use
            // nameDisplay?.text = "${getString(R.string.name_greet)} ${nameInput}"
        }
        else {
            Toast.makeText(this, getString(R.string.name_empty), Toast.LENGTH_LONG)
                .apply {
                    setGravity(Gravity.CENTER, 0, 0)
                    show()
                }
        }
    }
}
```

12. **Run** your app, and try submitting the name with an empty field. An error message should pop up at the bottom. Now try submitting it after entering your name into the field. The display name at the top should update.



13. Congratulations, you've made your first interactable app.

ASSIGNMENT

Continue the tutorial (**"LAB_WEEK_01"**) and add a **student number** as a second input field! The input has to be **11 digits** long. If it has more or less, then put out the error message **"Student number has to be 11 digits long"**.