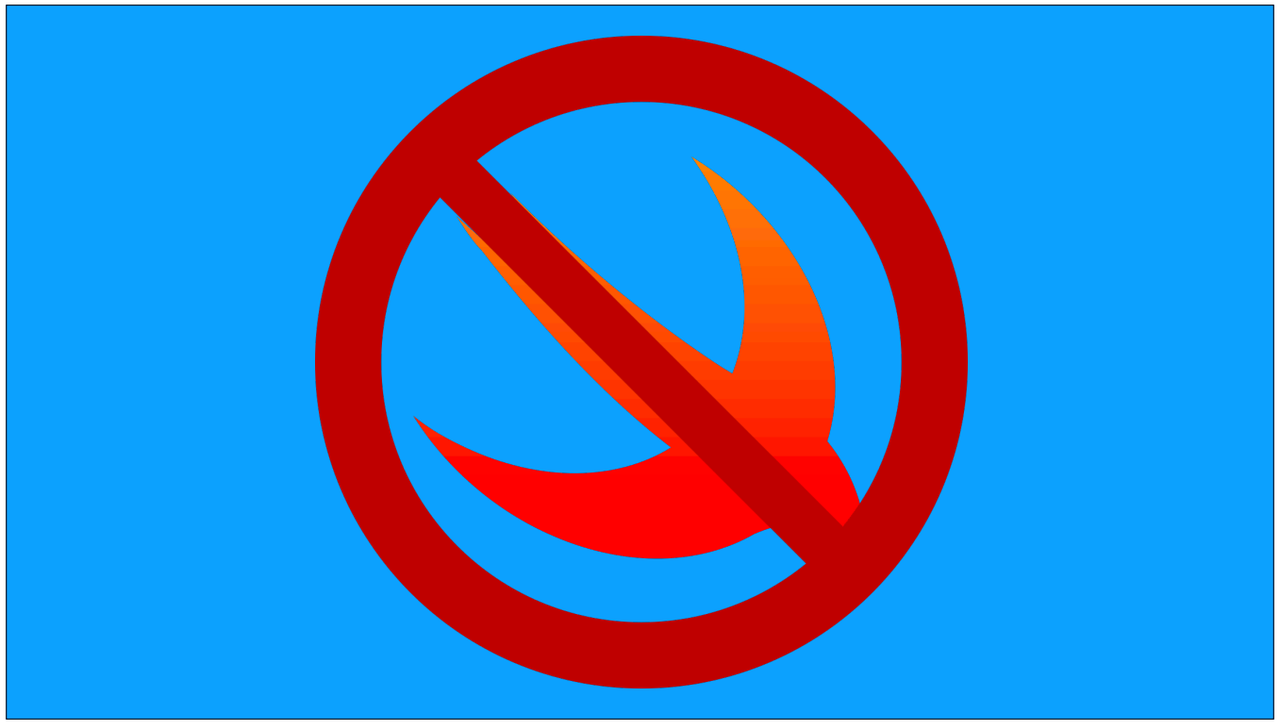
- Who’s made an Android app before?
- Who’s written Java before?

I’m Will. For those of you who don’t know me, some could describe me as a hackathon veteran, winning prizes at 6 different hackathons. On top of that, I’m one of the organisers for HackTheMidlands, as well as the founder for Hackathons for Schools Association, to help schools run their own hackathons. I also like to promote the hackathon wiki as much as possible….

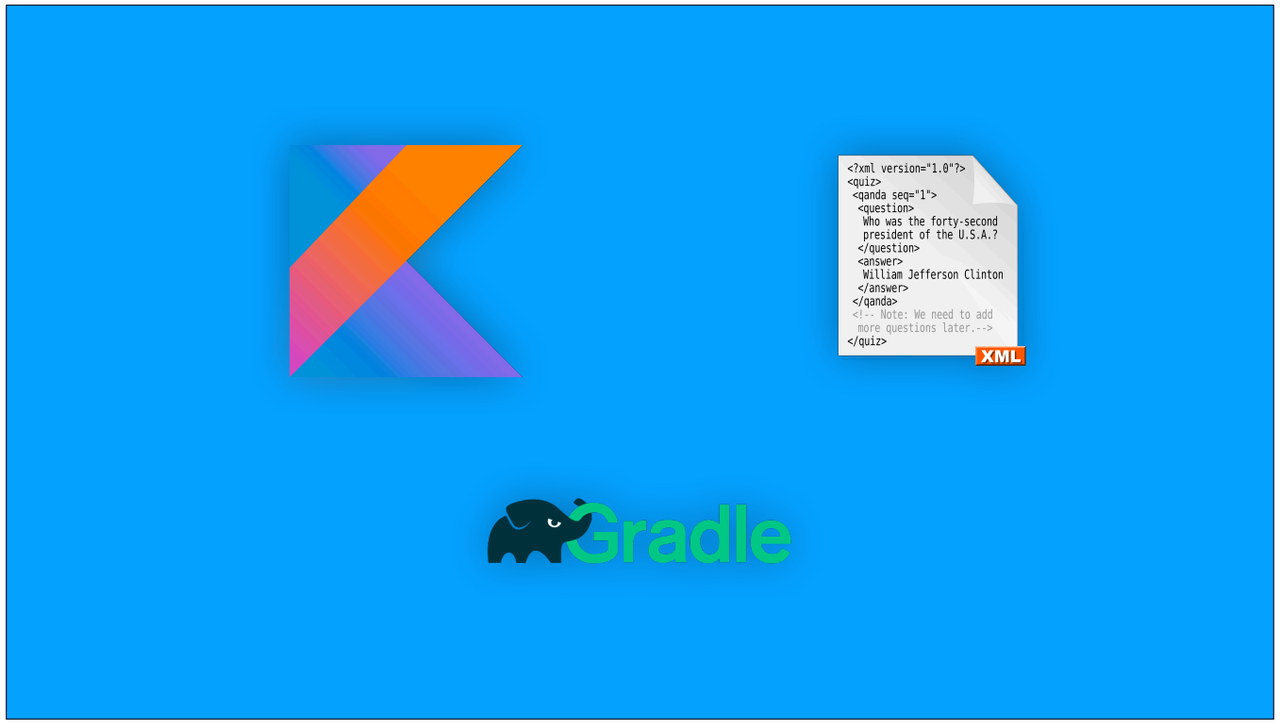
What makes up an Android app?

What are Android apps made of?

ASK AUDIENCE



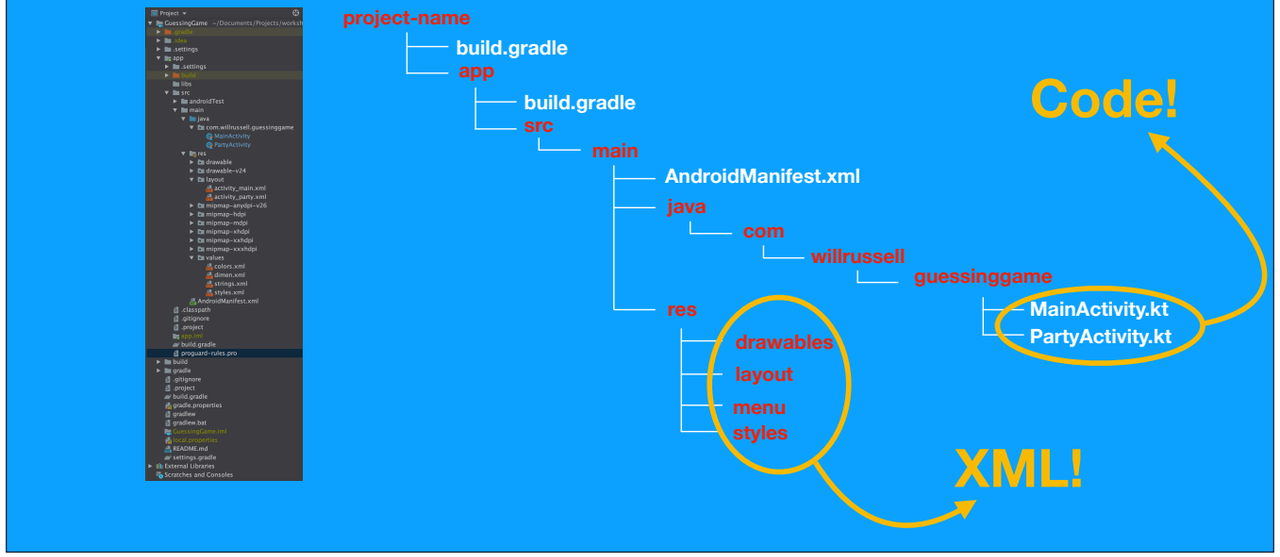
It’s definitely not Swift….



All the logic is written in Java
Your user interface is defined in XML normally. However, you can modify it with Java while the app is running.

Gradle helps builds the app, creating an APK (Android application package). The APK is the executable that Android runs, that has all the UI and logic components together. Think of it like an .exe on Windows, or a .dmg on macOS.

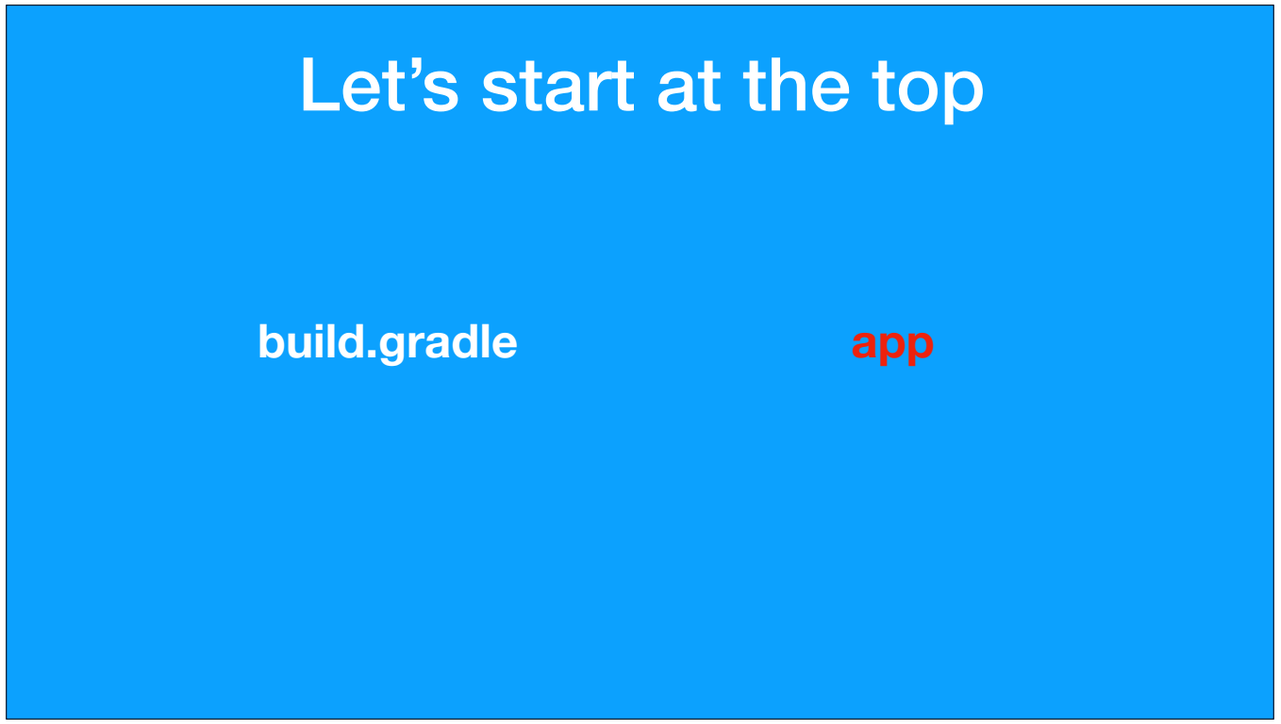
Kotlin has recently replaced Java as the default language for Android development.



Android apps have a lot going on.

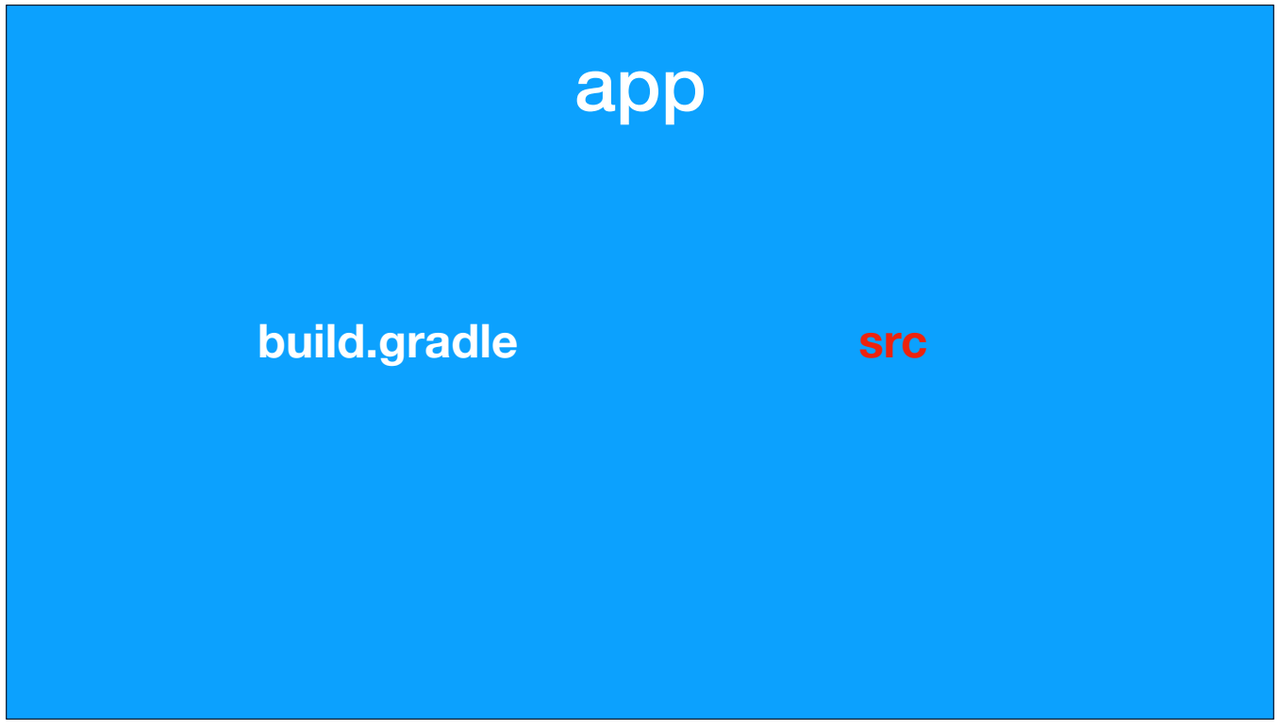
They can be quite intimidating, but don’t let this put you off.

We’re going to go through each layer of folders and break it down, starting at the top of the project!

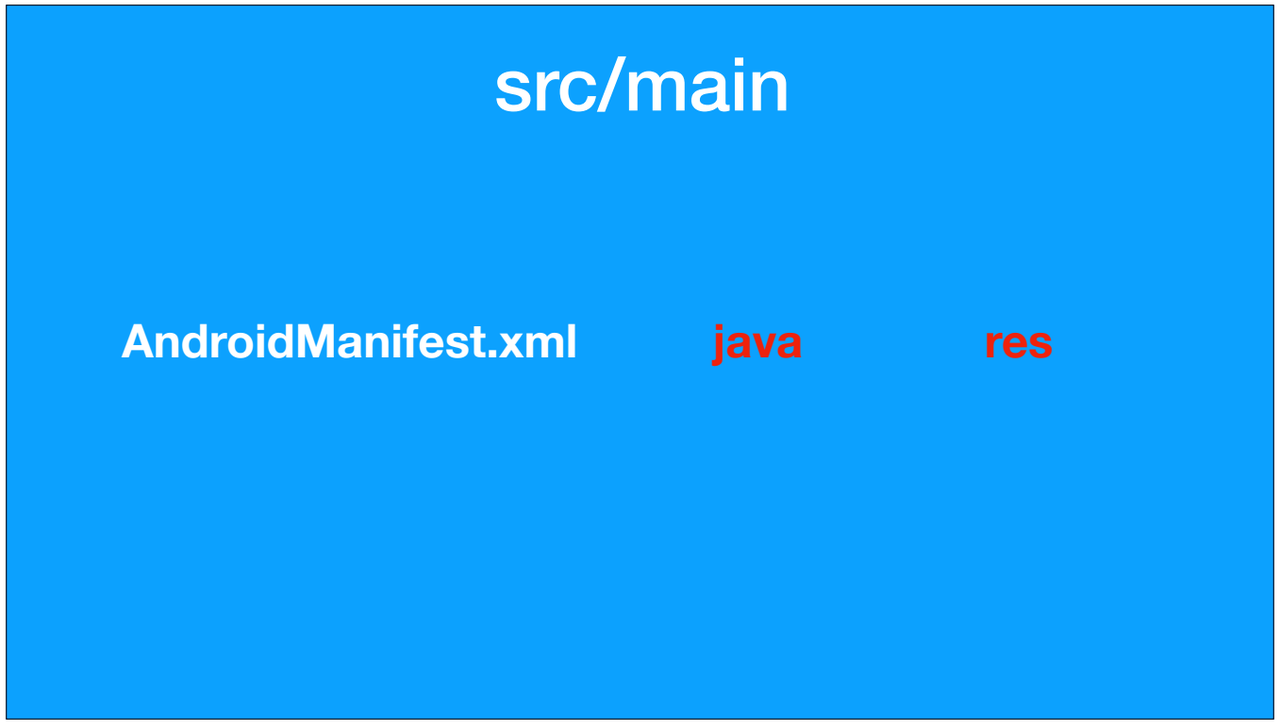


The 2 most notable things you’ll see are the build.gradle, and the app directory

Build.gradle is used to specify how Gradle is going to build your app. For the most part, you can leave this alone. The one that’s more interesting to us is inside the app directory.



Inside the app directory, we see another build.gradle file. Why 2 of these? Well for a basic app like this it’s not necessary, but this can come in handy when your app has a version for WearOS, Android TV, Android Tablets and Android phones. These might all need different requirements, so it’s useful being able to specify exactly what each module will use.



We’ll skip down 2 levels to main. In here, we see the AndroidManifest, and 2 directories which is where our code and UI sit.

AndroidManifest.xml

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

    <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=".PartyActivity"></activity>
        <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>
```

The AndroidManifest describes essential information about your app to the build tools, as well as the Android OS and Google Play.

The Manifest normally contains

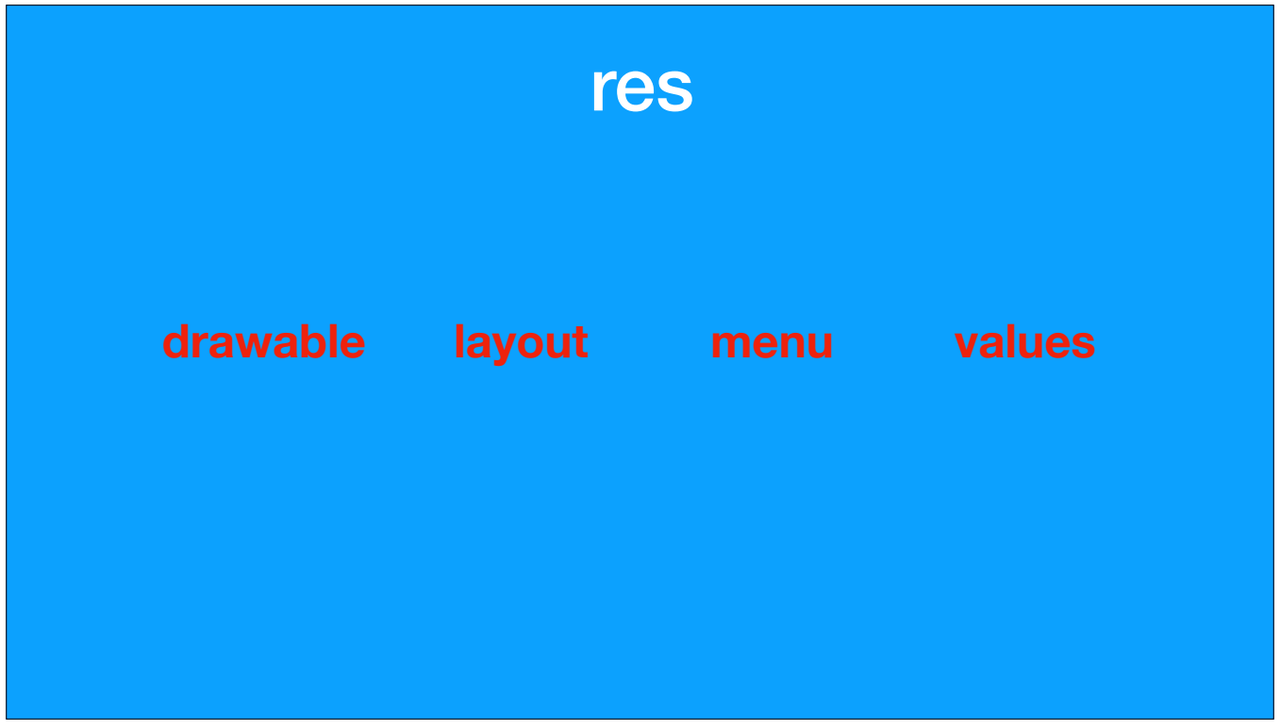
java

```kotlin
package com.willrussell.guessinggame

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class PartyActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_party)
    }

    fun tryAgain(v: View) {
        finish() // Finish the activity
    }
}
```

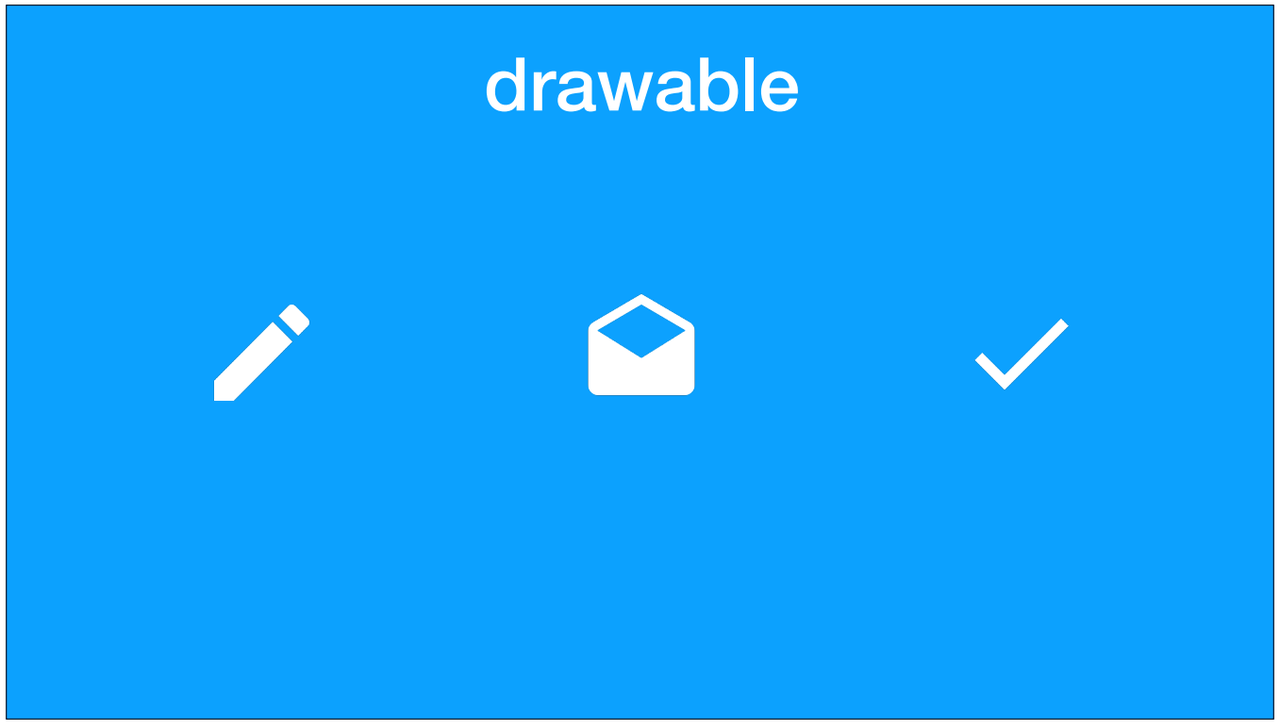
Inside the java folder, we find ourselves lots of code.

Here the java packages are defined, which is com.willrussell.guessinggame in this case.

The app we’re writing today will be in Kotlin so we’re going to have Kotlin here, but we could have Java if we wanted.



Res is short for resources. This is where we can define a lot of the app like icons, UI, menus, strings, colours, styles, and dimensions.

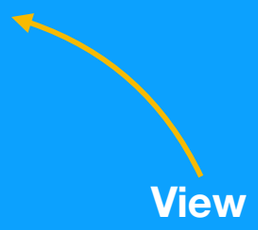


Icons, including the app icon, are stored inside of the drawable directory.

Android Studio lets you add Material Design icons into your apps quite easily. You can add your own SVGs or PNGs too!

layout

```
<TextView
    android:id="@+id/number_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/guess_label"
    android:textColor="@android:color/black"
    android:textSize="@dimen/title_size"
    app:layout_constraintBottom_toTopOf="@id/user_input"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



The layout directory contains all the code that describes our UI. This is written in XML, but we'll talk about this more later.

menu

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu 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"
    tools:context="com.will_russell.hackathon_demo_timer_android.MainActivity">

    <item
        android:id="@+id/action_countdown"
        android:orderInCategory="100"
        android:title="@string/countdown_mode"
        app:showAsAction="never" />
    <item
        android:id="@+id/action_time"
        android:orderInCategory="200"
        android:title="@string/time_mode"
        app:showAsAction="never" />
    <item
        android:id="@+id/action_about"
        android:orderInCategory="300"
        android:title="@string/title_activity_about_activity"
        app:showAsAction="never" />
</menu>
```

The menu directory, contains…. Well menus…..

This app doesn’t actually have any menus, but if it did, we’d put them here. These are also written in XML.

Here is a menu for another app I’ve been creating.

values

colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

dimen.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="input_margin">32dp</dimen>
    <dimen name="title_size">36dp</dimen>
</resources>
```

strings.xml

```xml
<resources>
    <string name="app_name">Guessing Game</string>
    <string name="guess_label">Guess a number</string>
    <string name="guess_correct_label">You guessed correctly!</string>
    <string name="guess_incorrect_label">You guessed incorrectly!</string>
    <string name="submit_button">Submit</string>
    <string name="try_again">Try again?</string>
</resources>
```

styles.xml

```xml
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Lastly, we have the values folder which contains all the other details of our app.

colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

We can define the colours used in our app here, in hex

dimen.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="input_margin">32dp</dimen>
    <dimen name="title_size">36dp</dimen>
</resources>
```

We can define the dimensions of our app here

strings.xml

```xml
<resources>
    <string name="app_name">Guessing Game</string>
    <string name="guess_label">Guess a number</string>
    <string name="guess_correct_label">You guessed correctly!</string>
    <string name="guess_incorrect_label">You guessed incorrectly!</string>
    <string name="submit_button">Submit</string>
    <string name="try_again">Try again?</string>
</resources>
```

We can put all the strings in here, so we can reference them easily, as well as update them from one place, than lots of other places.

styles.xml

```xml
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

We can change the theme of our app here, including things like Dark themes.

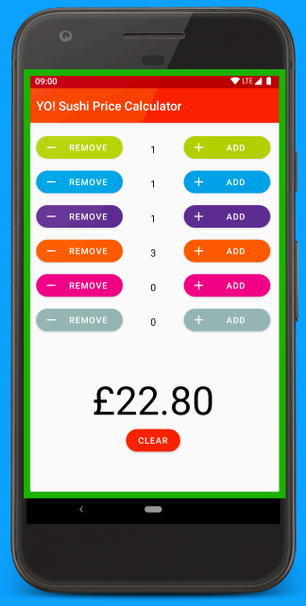
Components

Now we've talked about the project structure for an app, lets talk about what actually makes up an app!

Let's tart with components. There's 4 different types of components, which are essential building blocks for an app.

They are all entries into your app from both a user and system perspective.

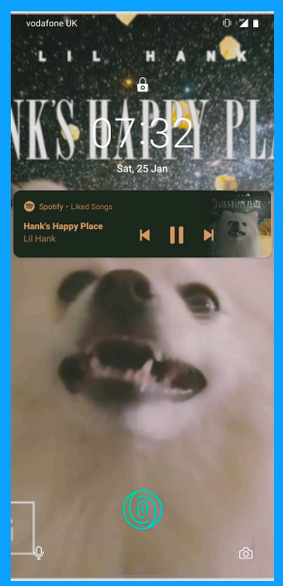
1. Activities



Entry point for interacting with the user

Single screen with UI (TRANSITION), such as a list of new emails, composing a new email, or reading an email.

2. Services



This is an entry point for keeping an app running in the background.

This could be music playback, or downloading a file for example. They don’t necessarily need to have a UI.

3. Broadcast Receivers 

Broadcast Receivers enable the system to deliver events.

These kind of events can be used for communication between the app and the operating system.

An example of this could be when an a phone goes into and out of airplane mode.

4. Content Providers

Content Providers handle the data and database management issues.

For example, this is how you would interact with an SQLite database on the web.

Using Components

Intents

So all these components sound great, but how do we use them?

3 of the 4 of these components use Intents, all but content providers.

An intent is an asynchronous message.

It binds individual components at runtime

Intents

```
val intent = Intent(this, PartyActivity::class.java)
```

They are created as objects in your code, which define the action to perform.

In this example, the intent object is going to start the Party Activity.

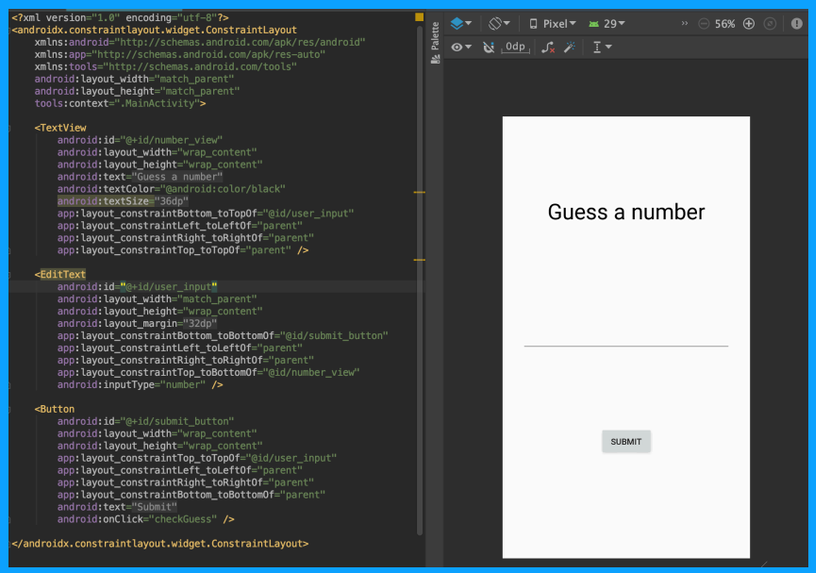
User Interface

Now we know how an app works under the hood, how can be display stuff to the user?

We need to write a user interface, but how does that work?

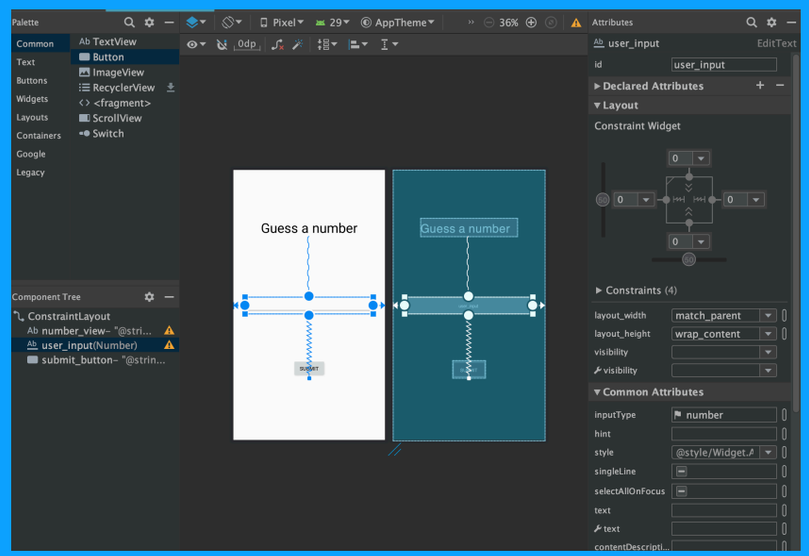
```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"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

    <TextView
        android:id="@+id/number_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/guess_label"
        android:textColor="@android:color/black"
        android:textSize="@dimen/title_size"
        app:layout_constraintBottom_toTopOf="@id/user_input"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/user_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/input_margin"
        app:layout_constraintBottom_toBottomOf="@id/submit_button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/number_view"
        android:inputType="number" />

    <Button
        android:id="@+id/submit_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="@id/user_input"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:text="@string/submit_button"
        android:onClick="checkGuess" />

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

The user interface is written in XML. This can look a bit daunting at first, but Android Studio lets you visualise what you’re looking at.



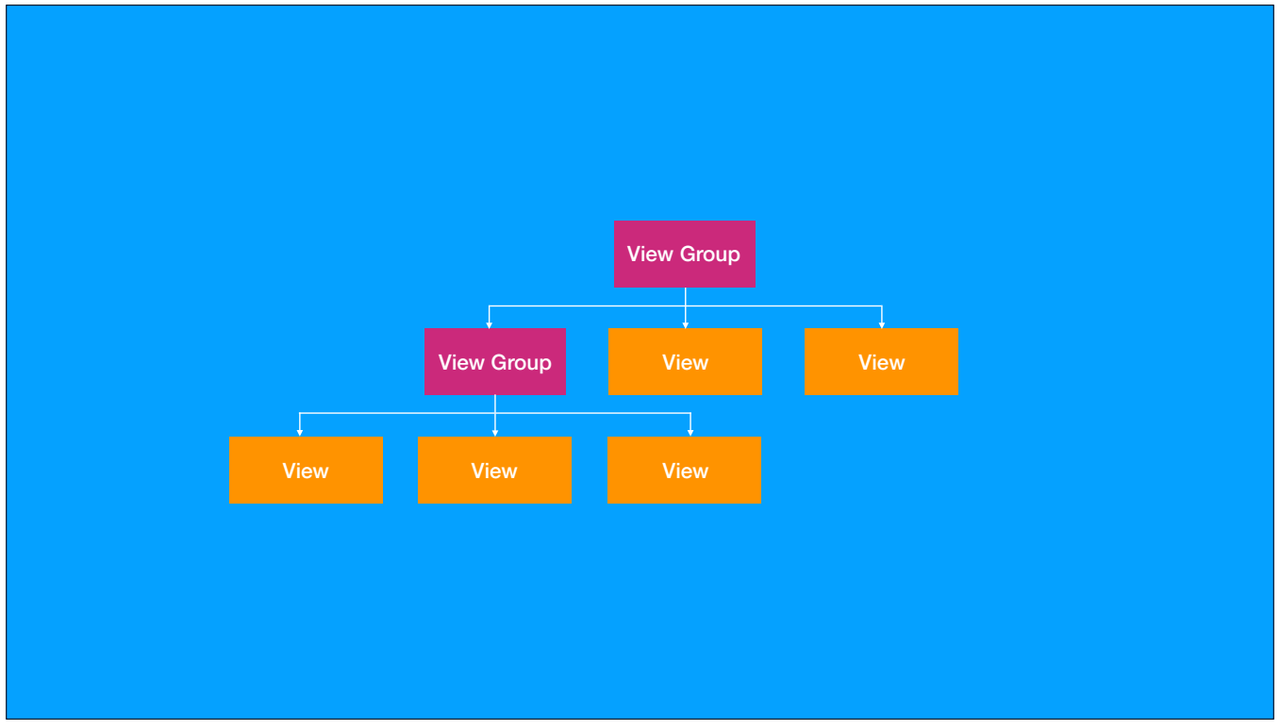
After a bit of practice, XML comes naturally as it’s quite human readable

If XML isn’t your thing, Android Studio gives you the ability to drag and drop views onto your screen



This drag and drop view definitely helps you get started, but can feel limiting.

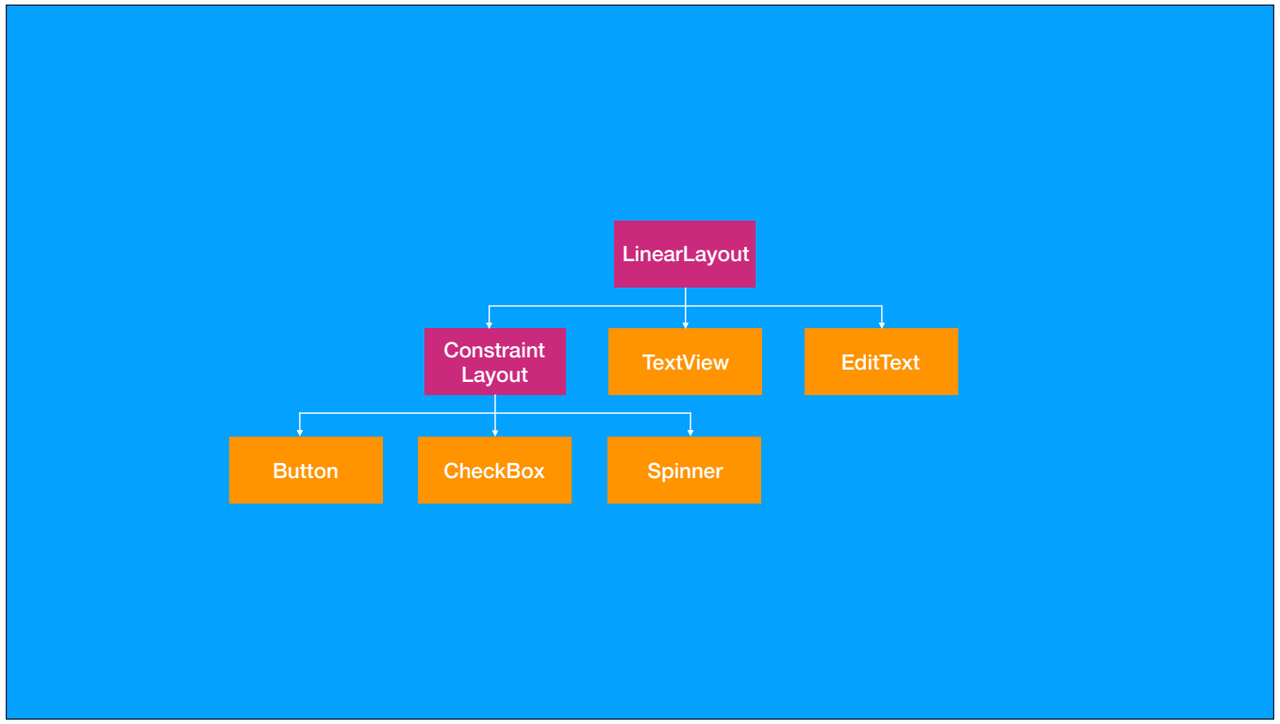
This is all well and good, but what actually makes up a UI?



The UI is made up of both Views and ViewGroups.

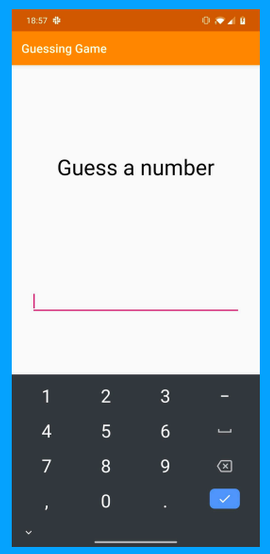
Views are single components on the UI. These can be text, labels, buttons, images, etc.

ViewGroups are also referred to as layouts which contain views, and tell them how they should display.



Here you can see an example of how Views might be grouped.

Time to make our own app!



Now it’s time for a live demo, where I’m going to show you how to create your own Guessing Game app from scratch.

- Don’t try and follow along
- All the code I am writing, as well as slides can be found online so don’t worry about trying to write down everything too.