## CET3013N: - Building an Android UI

**Week 2 Exercises: Using Android Studio to build a simple application**

###### Objectives – What this lesson is trying to achieve.

You are trying to learn how to:

1. Use Android Studio to build Graphic Layouts in both Landscape and Portrait mode
2. Connect Android views with java code using **findViewById**
3. Use the Android Emulator, and monitor applications using **Log.i**
4. **implement** an **onClick** callbackmethod for the button.
5. Implement the view binding in the project.

### Start the Emulator

An emulator is a significant application, so you should just start one per lab session and **leave it running**. See Week 1 for details

### Define a new Project

Android build most of the files we need. It is simplest to choose an empty activity. If you try something else just ignore the additional generated files.

1. Start a new Empty View Activity Project
2. Application Name: Week 2 CET3013  
   Domain: default
3. Use API 28 (Pie)
4. Continue Finish until your project files appear in Studio

### Start an Emulator and run the App

Android Studio always creates a ‘Hello World’ in your new project. It is often a good idea to run this to check all the SDK components are installed.

1. Select 'Run App' from the Android Studio Menu (allow it to install instant run should it request)
2. When the app starts (it will just show 'Hello World') use the back arrow on the phone to go to the home screen
3. Find your app name and start it to restart the app
4. DO NOT STOP the emulator, but swap back to studio

### Build a working one button App

In this task, we will add some code to a simple single button application. The application is designed in the interface designer, whilst the code is edited by editing the java file. Java files are stored in the src folder tree.

1. In Studio, double click ‘activity\_main.xml’ (or 'content\_layout.xml' in some other startup activity types). You will find under 'res/layout' . This will switch to the interface designer we looked at in week 1.
2. Add a single button to the layout
3. Click the button and open its properties. (These may be in a collapsed tab to the right)
4. Change the button name to 'Accept',
5. Change the button's ID to **accept\_button**
6. Finally, find the **onClick** property, and type **accept\_Clicked**
7. Find the java class **MainActivity** (in the java tree)
8. **Inside** the class, we are just going to use a System Log Call based on Log class. As this is for info, we use Log.i.  
     
   Add this method this to the mainActivity class under the **onCreate** method

**func accept\_Clicked(view:View){**

**Log.i("CET3013Test1", "Button Clicked")**

**}**

1. Run the App!

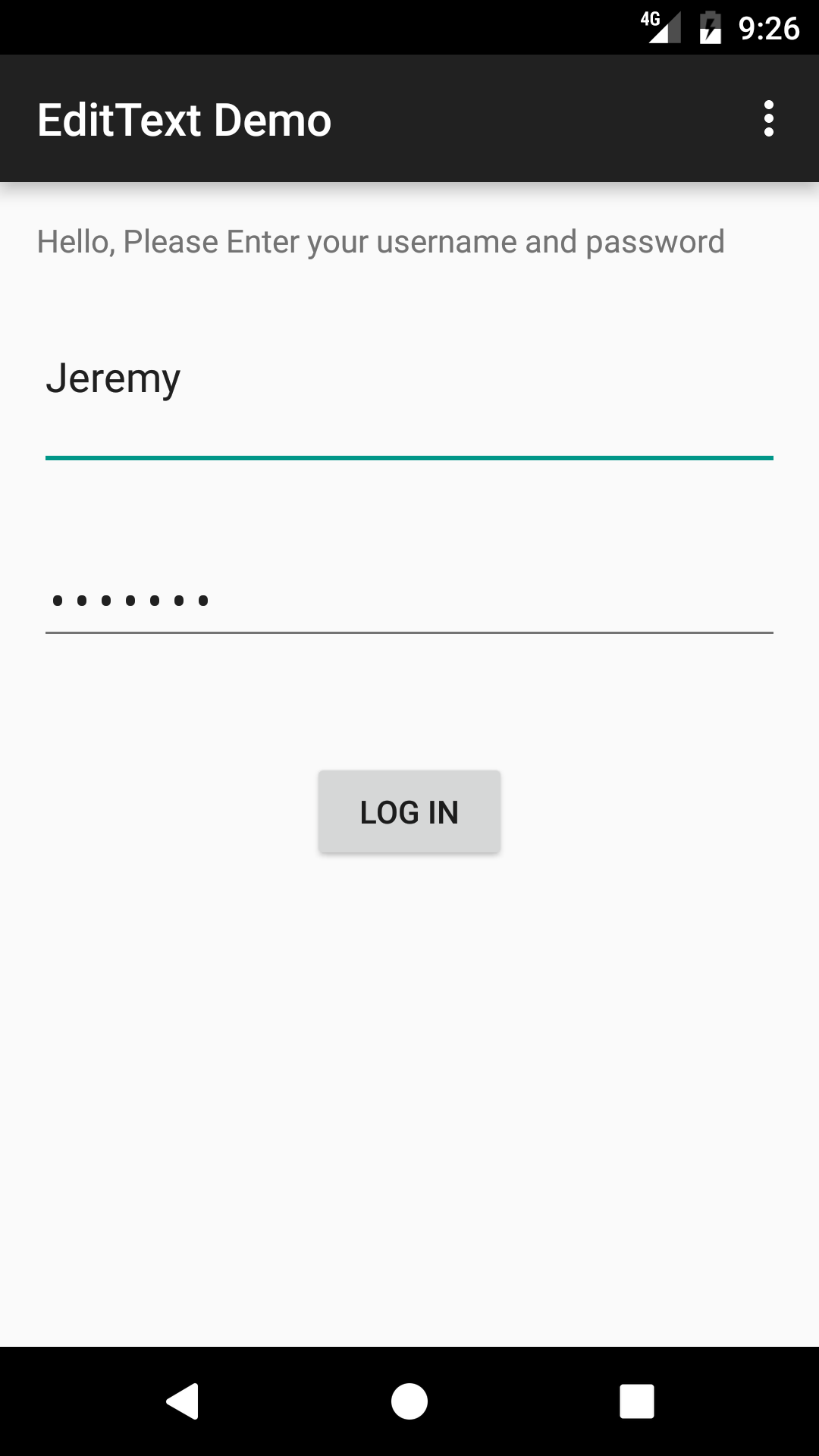
### Checking logcat

The Android Studio monitor links to applications running in Android (Virtual) Devices. This is very important for debugging.

1. In Studio, bring up the Android Monitor, which is usually, at the bottom of the screen as the pane labelled **6: Android Monitor**
2. Show the logcat pane
3. In your emulator, you should see text from your application saying 'Button Clicked' when you click the button. The message is being generated by the **Log.i** method call from the previous task. You will see many other messages being generated by other applications on the phone/emulator. These can usually be ignored

### A Login Screen

In this task, we are going to create a simple application in which view elements will be linked to java code. The application will look like this.

****

1. Using the Interface Designer Create a Layout such as that shown above. This is a RelativeLayout, which includes a **TextView,** two **EditTexts**, for a username, and the other for passwords. Underneath there is a **Button**, labelled 'Accept'.
2. Check the Layout shows well in the emulator. Try typing in both text boxes. You will need to tab between them
3. In Studio, bring the **MainActivity** into the java editor pane
4. Add private instance variables in the **MainActivity** class that correspond to views in the layout. Their types may be found in the layout XML (text) file. Call these **textview**, **username**, **password**, and **accept**
5. Assign these variables to their objects at the end of the OnCreate method using **findViewById**. Remember to cast the result to the correct type like this:  
    **username = findViewById<EditText>(R.id.username)**
6. Write a 'LogIn' method as in the earlier task, and attach this to the **accept** button.
7. This button should take the username and password text and display them using **Log.i** You can get the text from the relevant views by typing a '.' next to the variable and the appropriate method should pop up out of the menu.
8. Test the application in the emulator. Your objective is to type text into the username and password boxes, click accept and get log messages to show in the Android Monitor that report the text that you typed into the Emulator.

### Extend the Login Screen

Whilst it can be useful attaching code to a working view which we did with **Accept\_Clicked** above, a generic solution is preferable. Here we extend the base class to implement a listener.

1. Override the system default **onClick** method for the login button.

button\_accept.setOnClickListener {}

1. Record that the button was clicked (or not) using **Log.i** as previously
2. Test the application in the emulator. You should be able see the same messages as previously.

### Some programming logic

You should now extend the call back for the **accept** button as suggested below. Android Studio will suggest get and set text methods for each of the views when a ‘.’ Is placed next to the object name. The other operations are basic java.

1. When accept is clicked, If the username, and password contain at least four letters each, set the welcome message to be “Welcome, username”, where username is what you typed.
2. If the username has four or more letters, but the previous condition fails, change the welcome message to say “Weak Password”
3. If the username has fewer than four characters change the welcome message to say “Poor Username”

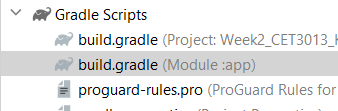
### Landscape

Android phones will automatically rotate an application when a phone is rotated. However, a design built for portrait might not look good in landscape mode. Consequently, it is useful to prepare a separate landscape layout. This will be automatically inflated whenever the screen is rotated to a landscape orientation. This is done as follows:

1. Open the activity\_main.xml file in Design view.
2. Click on the icon in the top-left and select Create Landscape Variation.
3. Drag and rearrange the onscreen widgets to form a more pleasing use of space.
4. Now create a res/layout-land folder in addition to the layout folder (Studio will hide this, and show the landscape layout labelled (land)
5. To include a layout file for landscape orientation you should place a file with the same name and widget IDs in the res/layout-land folder in addition to the portrait version in the res/layout folder. This file can then be edited in any way we like.

### View Binding

1. Open and configure the build.gradle module file.



1. Add in the viewBinding attribute inside the android {} block.

buildFeatures **{** viewBinding true  
**}**

1. Create the View Binding class in the MainActivity.

var binding: ActivityMainBinding? = null

1. Add in the following to configure the view binding structure:

binding = ActivityMainBinding.inflate(*layoutInflater*)  
val view: View = binding!!.*root*

1. Comment out the existing setContentLayout and replace with the following code:

setContentView(view)

1. Comment out all the findViewById statements and you can access the view directly using the binding object. For example:

binding.buttonLogin.setOnClickListener(v -> **{**

1. Convert all the existing with the viewBinding implementation.