SEM2220 Assignment Three
Report

Assignment Three - An Android Homepage Widget

Submitted in partial fulfillment of
the requirements for the award of the degree of

MEng
in
Software Engineering

Submitted by

crh13 Craig Heptinstall

Department of Computer Science
Aberystwyth University

13th May 2016

Chapter 1

1.1 Introduction

This paper looks at the solution provided for an assignment requiring the implementation of a homepage widget to be written for the android operating system. This app required the user to be able to see data about current temperature for multiple sensors in a house. The data to be shown on the app was hosted on two web addresses (one for each sensor), and either should be selectable for showing the data on the widget.

Upon reading the initial requirements, the main features/ tasks that had to be completed to satisfy the brief and add potential flair was interpreted as follows:

- Create a widget that is an appropriate size for a range of screen sizes and for the amount of data it should be showing.
- Display the data in a clean and compliant way following the Android design model.
- Pull data from APIs, and parse this data in a way it can be shown effectively.
- Provide the user with options applicable to the data and how it is displayed (e.g. Sensor selection).
- Provide means of refreshing the data on the widget and also a way to access the settings from the widget(Should be clickable).

The requirements above were designed to be abstract, so that under the implementation stage, if any issues arose, the widget design could be changed.

1.2 Design aspects considered

Because of the large amount of screen sizes available today, and the vast range of device pixel densities, designing a widget meant taking into consideration that different users may see the widget differently to others. Usefully, the android developer site outlines some UI guidelines [1] for widgets.

Because widgets are always placed on a cell-based page, it is important to find out how much space is available for the size of widget. The guidelines provided by Google give a simple formula (Available size(dp- dots per inch) = 70 x Number of cells). By using this, I was able to find out how much space I had available to use, and how many cells would be required for my widget.

From looking at how other apps, particularly weather apps used space in widget form, it was decided to keep the design rather minimalistic, therefore to keep the widget easy to read and use. Because of only the few details required in the brief to be displayed on the app, it made deciding a size in cells easier. A single row was decided, to keep the space free on a users' screen, and to not show too much blank space on the widget itself. Because this was a widget that notifies the user about data they may feel important, the option for cell width would be left to them (though due to the amount of text possible 4 cells minimum was decided.) By giving them the option to increase the width of the widget, they would be easily to stretch it across the entire width of the screen, to make it more aesthetically pleasing.

As for the colours of the app, to keep in trend with the general theme of the android operating system apps, simple pastel colours where chosen. By having a contrast of the same colour divided into sections, this meant different aspects of the data would be clearly defined. The

same simplicity reached the text type and colour, which again followed the design principles of standard android apps.

As for actual content on the widget, this was kept to what was required on the brief, with two additional options: refreshing and configuration buttons. Both of which used the android icon library [2], in order to continue following general design principles. The configuration window used generic control elements and simple text to give the user a clear and well defined way of using it.

1.3 Creating the application

Android studio made it very easy to create a widget application with the project wizard. This meant that the widget size could be defined (4 x 1 in this case) and the size of the widget was already defined in the project manifest file. The only change required here was changing the manifest file to allow the user to only change the size of the app horizontally and not vertically.

Once the skeleton was created, various tutorials on app layouts helped decide which layout style would fit best for what layout was required. In this case, it was decided that the use of a simple linear layout would be used, to give more flexibility in placement of content. Alongside this, using the 'layout weight' attribute, each element was easily placed in a grid like fashion, allowing them to expand naturally if the user made the widget wider. A tutorial on this attribute [3] helped explain how to use it to allocate portions of the widget space to elements based on their weights. Android principles were followed when it came to adding the icons for refreshing and configuring, and the widget preview image, by using different density images in appropriate folders so that higher density pixel phones would get better results in terms of quality.

Once the basic layout of the app was complete, populating it with actual data was then attempted. Being a basic XML server, and with the Java language backend of android apps, this meant many libraries and different means of getting and using the data was quite straightforward. In this case, the Dom builder library, as explained in another online tutorial [4] helped gather and get the various data attributes from the servers provided in the brief.

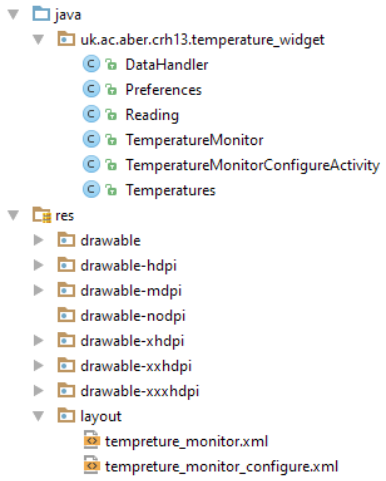

Figure 1.1: The codebase for the widget, showing object orientated classes, and different folders for the range of pixel densities.

With the data gathered by servers, in order to keep the code as maintainable as possible, this was done in an object-orientated means (see figure 1.1) as much as possible. Things such as temperature readings and user configurations were all stored as separate objects to make them easy to access and manipulate. By organising the data in the app, populating text views on the widget was done easily.

The next requirement saw the addition of a configuration window, and refreshing the data on the widget. The first of these meant in addition to the widget activity, another activity was required in order to allow the user to open and view a separate window when configuring options. Using Intents, this made navigating between the different activities, and passing data between them easy to implement. To grasp how to use Intents in the correct way, and make sure Android guidelines were being followed, useful tutorials [5] from the web were consulted

throughout.

Having created a basic configuration window with two options: sensor source and unit of temperature measurement, it was decided that rather than passing these directly over the Intent to the widget, they would be saved more consistently.

After looking over a few different means of saving data on an Android device, the SharedPreferences interface was decided upon due to the small amount of data being saved for this widget. Because there would only be two values to save (actual temperature data would simply be refreshed and pulled again when the user started their phone/ app up again): units and sensor choice, a database would be over-complicated and unnecessary. The SharedPreferences interface allowed the app to save the user-choices, in the config window, and then bring them back into the widget activity when refreshing and showing the data. Following the API on the Android site [6], this was quite straightforward, though it meant making sure the widget ID would used to store the values, so that other apps could not access data.

1.3.1 Issues encountered and solutions

With any project, there were issues that appeared that had to be solved throughout the project. Looking back at the start of the implementation, and without previous knowledge of layout properties such as weights, it meant the widget layout took slightly longer than expected. Having read various sources mentioned earlier, and by thoroughly testing the layout on different screen widths and cell sizes, the most appropriate and simple solution was found.

There was an initial problem pulling data from the server, due to both permissions in the android manifest, and that the main UI thread that apps run on do not allow network activity. Having realised this, the solution decided was to use the AsyncTask class [7], which acted like a helper class for managing different threads when trying to perform both UI management and other tasks at the same time. By utilising this class, the refresh on data could be called by the refresh button, and the user could continue to use the app.

The final two issues en-counted, were to do with refreshing the data from the widget. The first issue was having two buttons (one for opening the config window, and one for refreshing data.), and that when trying to perform an action on click, it was unknown which action was being called. By reading the developer documentation again, it was discovered that Intents could be assigned 'action' values which let them be assigned names. This also fixed the issue that on resize of the widget, the data refreshed, by knowing the 'onRecieve' method was not triggered by an Intent, the call could be ignored.

1.4 Testing and running the app

1.4.1 Emulating the application

As the device was developed on a AMD based system, the Intel emulator was not available. Due to this, using a real Android device was sufficient for testing, and allowed live debugging still in Android studio. The images following highlight features and work flow of the widget.

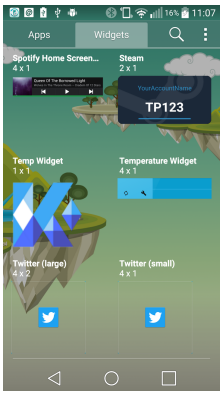

Figure 1.2: The 'Temperature Widget' shown in the widget list.

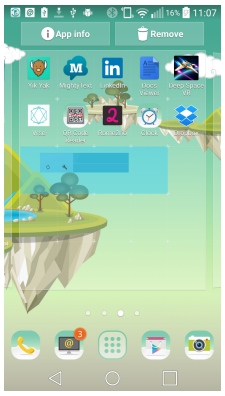

Figure 1.3: Dragging the widget to a homescreen. 4 cells are shown selected initially

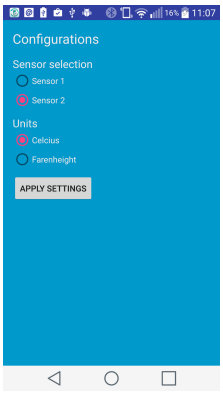

Figure 1.4: The config window opened when adding the widget for the first time.

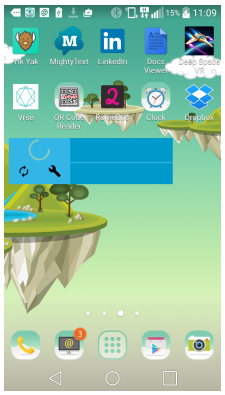

Figure 1.5: After accepting the config options, the widegt begins an initial refresh, showing the refresh spinner.

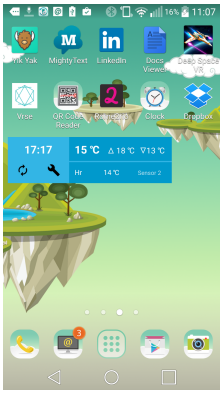

Figure 1.6: The content now loaded. Shown in initial state, Celsius and sensor 2.

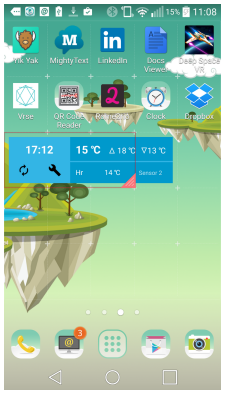

Figure 1.7: Attempted resizing the width smaller than allowed. Red shows it is not allowed.

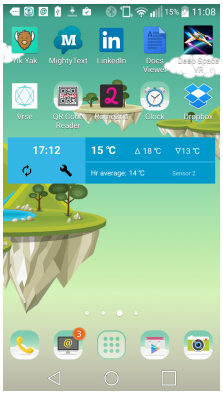

Figure 1.8: The widget after being stretched wider than 4 cells. The content is still readable and looks professional.

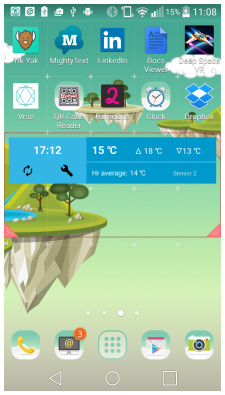

Figure 1.9: Attempted height resize. Not allowed becuase of the specified rule in the app manifest.

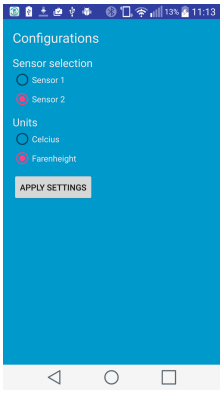

Figure 1.10: Changing the configuration to show the temperatures in Fahrenheit.

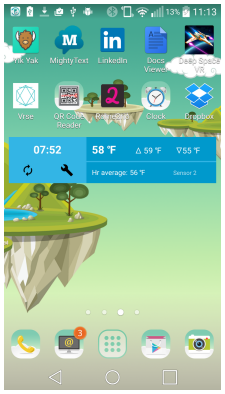

Figure 1.11: The widget showing temperatures in Fahrenheit after the update. Content has also been refreshed.

1.4.2 Testing

The following test table 1.1 explains how the widget was tested, in all usability and data aspects.

Table 1.1: My caption

Test #	Test	Expected Result	Actual	Pass?
1	Drag widget to cells	Config window opens	As expected	Pass
2	Close initial config window after dragging	Widget does not show on device	As expected	Pass
3	Confirm default values after dragging	Widget shows size 4 X 1, with content after loading	As expected	Pass
4	Stretch widget wider than 4 cells wide	Widget stretches to any width and shows all content the same	As expected	Pass
5	Make widget smaller than 4 cells wide	Widget does not allow smaller than 4 cells wide	As expected	Pass
6	Make widget larger than 1 cell high	Widget does not allow higher than 1 cell	As expected	Pass
7	Click config open icon	Config window should open	As expected	Pass
8	Make changes but click back	Widget should remain the same	As expected	Pass
9	Change unit to Fahrenheit	Temperatures should update and be in Fahrenheit	As expected	Pass
10	Change sensor to sensor 2	Temperatures should change to data from sensor 2	As expected	Pass
11	Open config from widget	Radio buttons should still be selected appropriately	As expected	Pass
12	Press refresh button	Content on widget should refresh, pulling new data	As expected	Pass
13	Press refresh button with no data connection	'No Data!' should show on widget	As expected	Pass
14	Restart phone	Widget should show refresh spinner, then show content	As expected	Pass

1.5 Conclusion

As can be seen in table 1.1, all usability tests passed, and due to making sure errors were caught and dealt with, the widget runs smoothly with no unattractive traits. During this project, many skills were learned and built upon, from creating a basic widget layout, to allowing communication of data flow between activities and external sources. The SharedPreferences interface is also a new skill that would prove very usefully in future projects. The main lesson to pull from tis project is that by maintaining use of Android guidelines provided directly by Google, it is quite easy to create clean looking apps and widgets that conform to the the day-to-day apps that can be found on any Android smartphone.

Following the work performed, I can now provide personal scores and reasons for the amounts:

Topic	Score	Reason
Documentation	24/30	Good description of what features were implemented, problems addressed, and how the widget was created. Clear test table, and display of the widget in use. Design choices were justified accordingly. Page limit exceded only because of widget screenshots.
Implementation	39/50	Well commented code, coded following android guidelines. Debugged and tested inside Android studio, using Android 5.0 and above features as required. OO style of code. Good use of SharedPreferences to store user preferences.
Flair	5/10	Added option to change the unit of temperature, and refreshing spinner animation appears when refreshing data. Widget almost looks like part of the standard Android UI.
Testing	8/10	Use of Android studio debugging, with the use of a real Android device to test. Clear test table and screenshots of using the widget.

Overall mark: 76%

References

[1] Google, *App Widget Design Guidelines*, `http://developer.android.com/guide/practices/ui_guidelines/widget_design.html`, 2016.

[2] Google, *Material icons- Google design*, `https://design.google.com/icons/index.html`, 2016.

[3] C. Andrews, *Using Android's layout weight attribute*, `http://www.101apps.co.za/index.php/articles/using-android-s-layout-weight-attribute.html`, 2014.

[4] TheoryApp, *Parse XML using DOM in Java*, `http://theoryapp.com/parse-xml-using-dom-in-java/`, 2012.

[5] C. Andrews, *Creating app widgets*, `http://www.101apps.co.za/index.php/articles/app-widgets-tutorial-part-2.html`, 2014s.

[6] Google, *SharedPreferences documentation*, `http://developer.android.com/reference/android/content/SharedPreferences.html`, 2016.

[7] Google, *Async Class documentation*, `http://developer.android.com/reference/android/os/AsyncTask.html`, 2016.