WEATHER MOBILE APPLICATION

CSCI 5408 – Mobile Computing

Submitted by:

Harsh Pamnani

B00802614

Submitted on March 22, 2019

TABLE OF CONTENTS

LIST OF TABLE AND FIGURES

INTRODUCTION

Overview

The main task for this assignment is to develop a weather application, which fetches the weather details from OpenWeatherMap API. Edit text input for the user to enter city name, temperature, clouds, humidity, minimum temperature, maximum temperature, weather, and other details mentioned in the assignment specification document are the must have for this application. The default city for this application is set to Halifax. Hence, when the application launches, it will display Halifax's weather. Figure-1 shows Halifax weather on application launch.

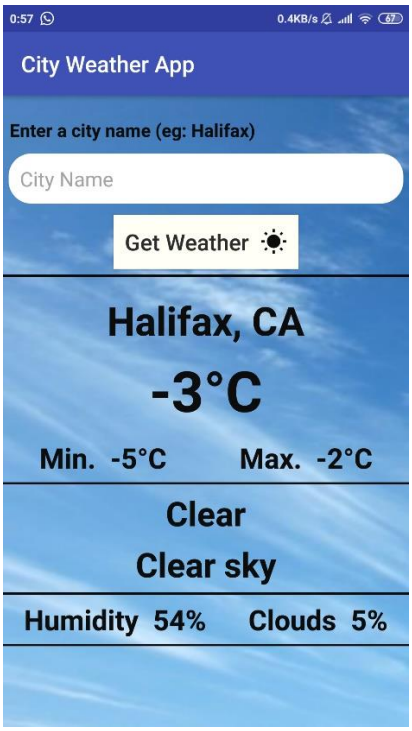

Figure 1 Default city Halifax's weather

What is unique?

There are two main unique features of my application:

1. Autocomplete text view for city name suggestions
2. Dynamic background images

The **autocomplete text view will show the city name suggestion when the user starts typing** for the city name. In this way, it will help the user to accelerate the task they are doing. All the frequent cities are taken into consideration. Also, I have not hard-coded these values. The values are fetched dynamically from String array at run time. So, if we want to add a new city in the drop-down in future, we can easily add it by adding a new item in the string array. Figure-2 shows autocomplete when the user enters character "t" in the edit text.

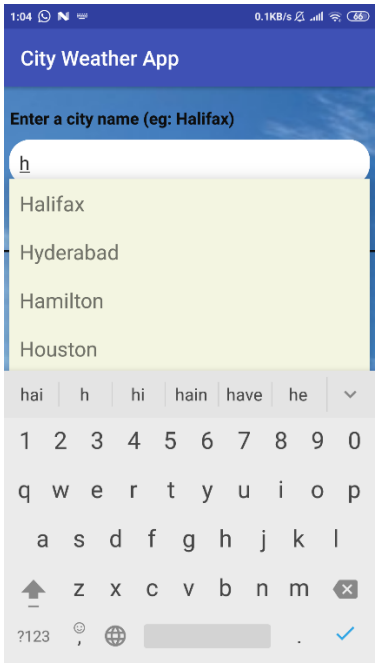

Figure 2 Autocomplete text view for city name suggestion

Another unique feature of my application is **dynamic background images according to the weather.** When the data is received from the weather API, I will display all the necessary details on the screen and the main layout's background image will change dynamically based on the weather description received in the response from API. For example, if the city has clouds, cloud's background image will be displayed. Similarly, for rain, Rain background image will be displayed. Figure-3 shows numerous kinds of dynamic backgrounds like Clouds, Mist, Rain, etc.

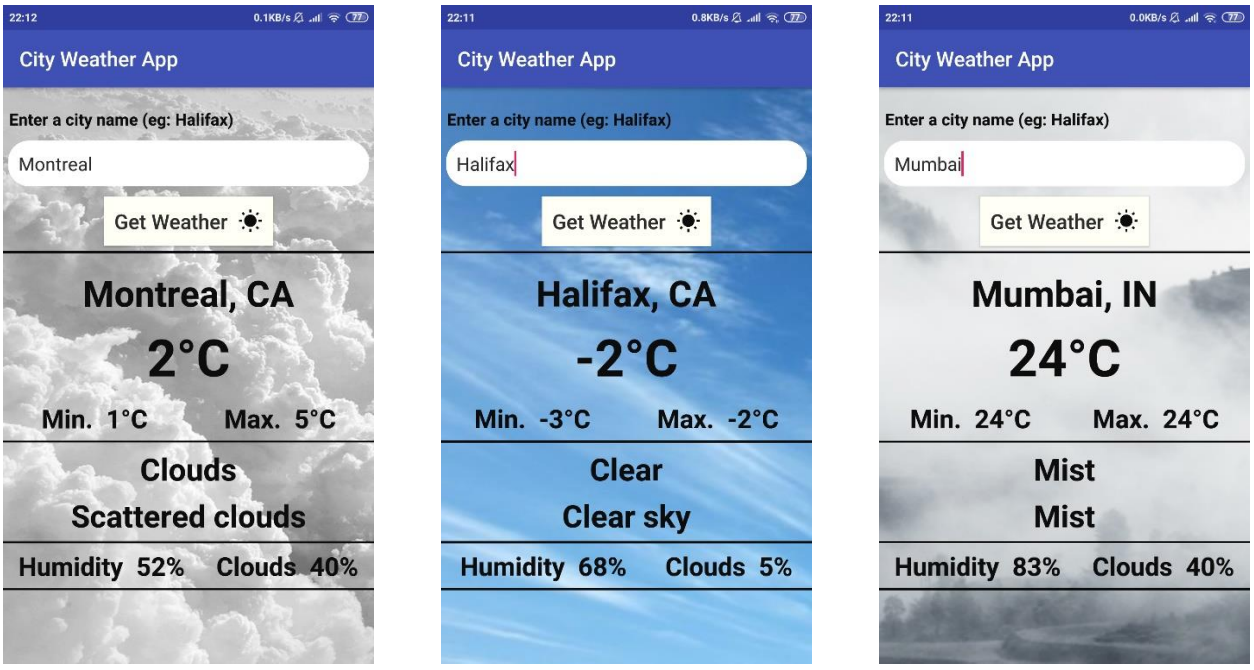

Figure 3 Dynamic background images

SOFTWARE DESIGN AND ARCHITECTURE

Software used and dependencies

I have used Android Studio 3.3 for building this application. Following are the dependencies used in my build.gradle file.

```
implementation 'com.android.support:appcompat-v7:28.0.0'
implementation 'com.android.support.constraint:constraint-layout:1.1.3'
testImplementation 'junit:junit:4.12'
androidTestImplementation 'com.android.support.test:runner:1.0.2'
androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
```

Figure 4 Dependencies

Apart from this, I am using OpenWeatherMap API to get the weather information for user-entered city. HTTP GET method is used to request information from the API. Following is screenshot from application code for the URL. As shown in figure-5, I have used static final variables for declaring these constants. Hence, if OpenWeatherMap changes the URL in future, there will not any changes in my code, we just have to change the values of these constants.

```
private static final String URL_REQ_START = "http://api.openweathermap.org/data/2.5/weather?q=";
private static final String URL_REQ_END = "&APPID=";
```

Figure 5 OpenWeatherMap API request URL

Wireframe for UI design

Before starting to create the application, I have built a wireframe for UI design, which gives the schematic representation of what the actual application will look like. Figure-6 below shows the wireframe.

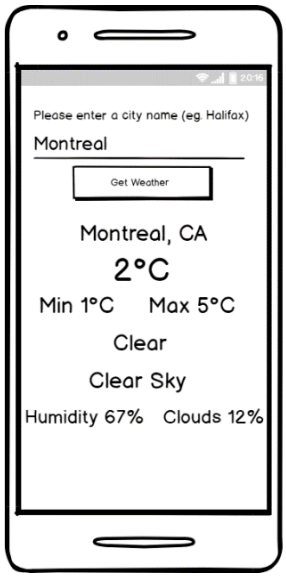

Figure 6 Wireframe

IMPLEMENTATION STEPS

UI design

After completing the wireframe, the first step of making any android application is to make user-friendly UI design. In UI, there is a hint displayed to the user on the top “Please enter a city name (e.g. Halifax)”. After this, there is an edit text for the user to enter the city name. I have implemented Auto Complete text view, so that it helps the user for completing the city names. Below that, submit button “Get Weather” is kept, using which the user can get the weather details for the entered city name.

The main layout is LinearLayout. Inside that, text views are placed for displaying the city name, current temperature, minimum temperature, maximum temperature, weather description, humidity and clouds percentages. I have also kept 4 dividers on the UI, to easily identify the various information.

Application Logic

Once the UI is completed, the second step is to build a logic behind the application. For auto-complete city names, I have created an array of city names in the strings.xml resource. Most of the popular cities are considered while building this array. Once the user enters the city name, it is validated that it doesn’t contain any characters other than alphabets.

If city name contains numbers or any other special characters, an error message will be displayed to the user. If the city name contains only alphabets, the “Get Weather” button will make a request to OpenWeatherMap API to get the weather details for that city. AsyncTask helper class is used, so that the request to API is made in background and not on the main UI thread.

ANALYSIS OF NIELSEN'S USABILITY HEURISTICS

Error Prevention

I am showing city name suggestions in drop-down. In this way, as soon as the user starts typing, the city names will be displayed in the dropdown. Hence, the user will not enter the wrong spelling for city names. Although the drop-down list is static for now, it can be easily extended in future because I am using string array for storing city names. So, new cities can be added in the string array.

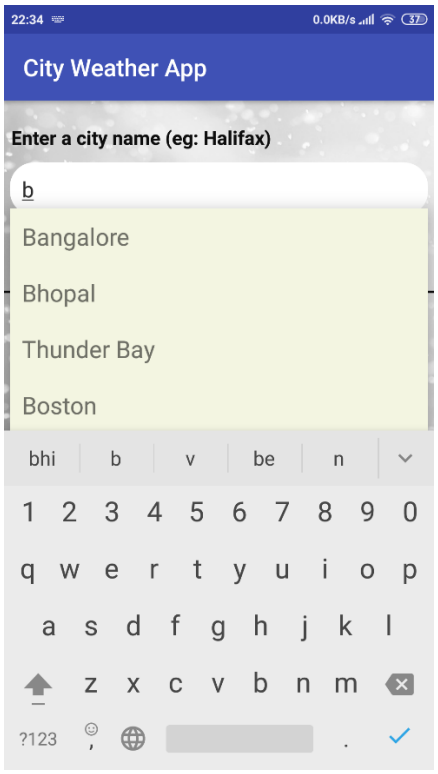

Figure 7 AutoComplete for city names' suggestion

Help users recognize, diagnose, and recover from errors

If the user any makes any error, toast messages are displayed to help the user to recognize the error and suggest a solution in a constructive way.

1. If user doesn't enter any city name any directly press "Get Weather" button, then a toast message saying "Please enter city name" will be displayed. Figure-8a shows the toast for this error message.
2. If user enters wrong city name, then a toast message saying "Please enter valid city name" will be displayed. Figure-8b shows the toast for this message.
3. If the internet connection is not present and if the user tries to get the weather, then a toast message saying "Please check your internet connection and try again" will be displayed. Figure-8c shows the toast for this error message.

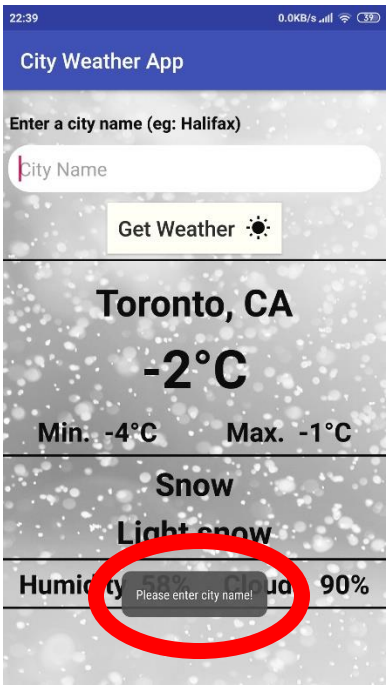

Figure-8a

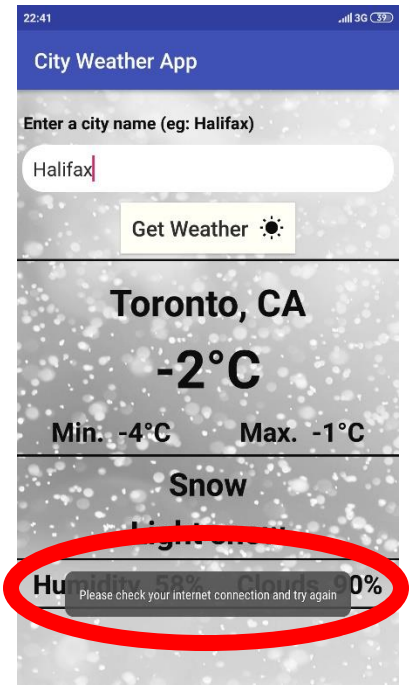

Figure-8b

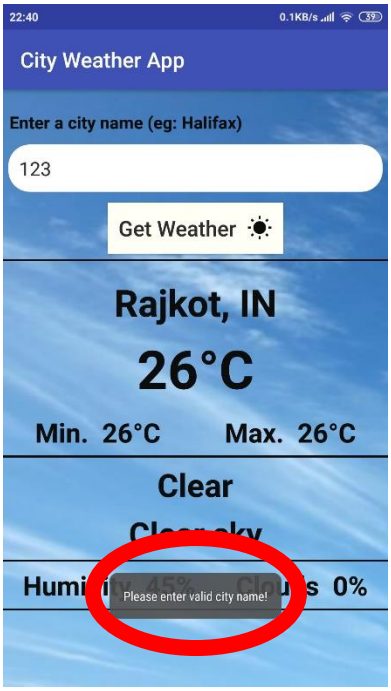

Figure-8c

Figure 8 Showing user-friendly error Toast messages

Aesthetic and minimalist design

The weather application contains only relevant information. There is no irrelevant information shows on the UI.

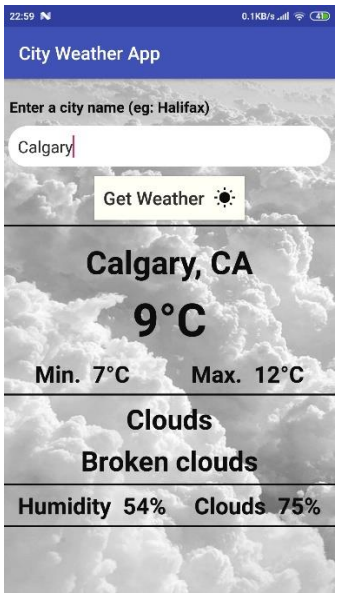

Figure 9 Aesthetic and minimalist design

IMPLEMENTATION PROBLEMS AND SOLUTIONS

AsyncTask problem

When I first started building the application, I build the logic of HTTP GET request without using AsyncTask class. The application was able to fetch data, but as the thread has already passed from there, it was not setting the UI element. So, when I press the button second time, it fetches the new details from API, but displays the old value from previous run.

As a solution to this problem, I found that we must use AsyncTask to enable proper and easy use of UI thread. The AsyncTask has two override methods **doInBackground** and **onPostExecute**. The **doInBackground** method will fetch the weather data from API in the background process. Once the data is retrieved from the API, we can access the result in **onPostExecute** method and set the UI elements according to our requirements.

AutoComplete drop-down problem

When I first started the application, I did not put any auto-complete drop-down suggestion for city names. Once the application was built, I thought it would be a good idea to put auto-complete for city names. However, I did not know how to set adapter for auto-complete. Hence, for the syntactical part of it, I was facing problems.

I resolved this problem by looking for the syntax of setting the adapter on it. And then, I created a separate array in strings.xml for city names. The list is kept small as of now, but if we want to insert more cities, then we can easily add new city by entering new item in the string array.

Internet connectivity problem

The weather application was crashing when “Get Weather” button is pressed, if the internet connectivity is not present. So, for this problem, I wanted to check whether the device has connectivity or not. If the device has connectivity, then only proceed further, otherwise show the toast message “Please check your internet connection and try again” is displayed to the user. For this purpose, I have added “**ACCESS_NETWORK_STATE**” permission in the **Android_Manifest.xml** file.

TESTING METHODOLOGY AND TEST PLANS

Test Cases List

I have prepared following list of test cases for my application. These test cases will be tested in **Unit Testing and Alpha Testing.**

1. User enters correct city name and gets the weather information.
2. User enters city name which contains digits in it.
3. User doesn't enter city name and directly presses "Get Weather" button.
4. Internet connectivity is not available in the device, and user tries to get the weather information.
5. Temperature is > 0 °C
6. Temperature is = 0 °C
7. Temperature is < 0 °C
8. Same conditions as 5, 6, and 7 for Minimum and Maximum temperature.
9. Humidity is 0%.
10. Humidity is between 0% and 100%.
11. Humidity is 100%.
12. Same conditions as 9, 10, and 11 for Clouds.
13. Background image changes to **Rain** background, when weather description contains "**Rain**" or "**Drizzle**" in it.
14. Background image changes to **Clouds** background, when weather description contains "**Clouds**" or "**Cloudy**" in it.
15. Background image changes to **Smoke** background, when weather description contains "**Smoke**" in it.
16. Background image changes to **Clear Sky** background, when weather description contains "**Clear**" in it.
17. Background image changes to **Sunny** background, when weather description contains "**Sunny**" in it.
18. Background image changes to **Storm** background, when weather description contains "**Storm**" or "**Thunderstrom**" in it.
19. Background image changes to **Haze** background, when weather description contains "**Mist**" or "**Haze**" in it.
20. Background image changes to **Snow** background, when weather description contains "**Snow**" in it.
21. Background image changes to **Default** background, when weather description does not contain any of the above terms I it in it.

Unit Testing

Unit testing is the testing performed for individual components of the application and check all the components are working as expected. If there are multiple screens in any android application, then all the screens are tested individually. Once all the screens are tested completely, integration of all the screens is done.

In the weather application, there is only one screen where the user can enter the city name and get the weather information. So, for Unit Testing only one screen is tested. For the unit testing I have performed following tasks:

- Check whether “Please enter a city name (e.g. Halifax)” is displayed at the top or not.
- Check edit text is displayed to the user for entering the city name.
- Check “Get Weather” button is displayed below edit text.
- Check “Get Weather” button has “Sun” drawable icon right side to it.
- Check what happens when user enter incorrect city name.
- Check what happens when user enters special characters in city name.
- Check what happens if no internet connectivity is there and user tries to get the weather information.
- Check that the background images changes for all the weather conditions mentioned in Unit Testing.

Alpha Testing

Alpha Testing is the first phase of user acceptance testing, which is performed to diagnose all the possible issues and bugs for any product before releasing it to the market. Main focus of this type of testing is to replicate how real users might use this application. For alpha testing, the user are internal users. So, in my case I have tested my application and asked two of my friends to test the application. During alpha testing, I found two of the important bugs of my application.

- **Application crashes when there is no internet connection**

  The application was crashing if there is not internet connection and user tries to get the weather details from the API. So, to solve this crash, I am first checking whether internet connection is available or not. If it is not available, then I ma displaying a toast “Please check your internet connection and try again.” to the user.

- **Application displayed “Please enter valid city name” even if user does not enter any city name**

  When the user directly pressed the “Get Weather” button, without entering any city name, then also it displayed “Please enter valid city name”. So now I am checking when the user presses the button, the edit text is not empty. If the edit text is empty, then I am displaying the toast “Please enter a city name” to the user. So, now it displays the correct message.

- **Application displayed weather information for city name like “123”**

  The application was displaying weather information for city names like “123” and “Hami,,,,”. This problem occurs because the OpenWeatherMap API has some strange cities like this. To resolve this problem, I am checking that the city name should contain only alphabets.

REFERENCES

[1] H. Corners, "How to create EditText with rounded corners?", *Stack Overflow*, 2019. [Online]. Available: https://stackoverflow.com/questions/3646415/how-to-create-edittext-with-rounded-corners. [Accessed: 23- Mar- 2019].

[2] "AsyncTask | Android Developers", *Android Developers*, 2019. [Online]. Available: https://developer.android.com/reference/android/os/AsyncTask. [Accessed: 23- Mar- 2019].

[3] A. Jasmin, "Detect whether there is an Internet connection available on Android", *Stack Overflow*, 2019. [Online]. Available: https://stackoverflow.com/questions/4238921/detect-whether-there-is-an-internet-connection-available-on-android. [Accessed: 23- Mar- 2019].

[4] "Check if a string contains only alphabets in Java using Regex - GeeksforGeeks", *GeeksforGeeks*, 2019. [Online]. Available: https://www.geeksforgeeks.org/check-if-a-string-contains-only-alphabets-in-java-using-regex/. [Accessed: 23- Mar- 2019].

[5] H. Mansouri, "How to set the size of an AutoCompleteTextView Result?", *Stack Overflow*, 2019. [Online]. Available: https://stackoverflow.com/questions/5539469/how-to-set-the-size-of-an-autocompletetextview-result. [Accessed: 23- Mar- 2019].

[6] "Alpha Testing" *Guru99.com*, 2019. [Online]. Available: https://www.guru99.com/alpha-beta-testing-demystified.html. [Accessed: 23- Mar- 2019].

[7] "Unit Testing", *Software Testing Fundamentals*, 2019. [Online]. Available: http://softwaretestingfundamentals.com/unit-testing/. [Accessed: 23- Mar- 2019].

[8] "Home Screen Background Image", *Shutterstock*, 2019. [Online]. Available: https://www.shutterstock.com/video/clip-6796867-white-clouds-disappear-hot-sun-on-blue [Accessed: 25- Mar- 2019].

[9] "Thick Clouds Background Image", *Pinterest*, 2019. [Online]. Available: https://www.pinterest.ca/pin/323062973239792308/. [Accessed: 25- Mar- 2019].

[10] "Sunny Background Image", *Sketchappsources.com*, 2019. [Online]. Available: https://www.sketchappsources.com/free-source/2501-iphone-app-background-sketch-freebie-resource.html. [Accessed: 25- Mar- 2019].

[11] "Black Raindrops Background Image", *Pinterest*, 2019. [Online]. Available: https://www.pinterest.ca/pin/477522366732474301/?lp=true. [Accessed: 25- Mar- 2019].

[12] G. Productions, "Weather Application Icon", *Iconfinder*, 2019. [Online]. Available: https://www.iconfinder.com/icons/2990925/mixed_rain_sun_weather_icon. [Accessed: 25- Mar- 2019].