GRT TXT

GEOG 483: GeoWeb Project

Kwok, Calvin Mistry, Jaydeep Ong, Mark Tasneem, Zarrin

Prepared for: Dr. Johnson

GEOG 483: GeoWeb and Location Based Services

April 4, 2017

Table of Contents

1.0 Introduction

With the task of creating an open-ended technical development project, a native android application called GRT TXT was developed. It is a map utility app to see all Grand River Transit (GRT) bus stops and quickly send SMS messages for bus times. It packaged GRT bus stop open data from the City of Waterloo into a simple map interface so that the user can immediately SMS the GRT bus stop code to the EasyGo Next Bus Call & Text Messaging service (#57555). The app is a one stop solution to the need of sending SMS for GRT bus times because it displays the GRT bus stops on a map, and also sends the SMS from within the app.

1.1 Target Market

The target market for this app are people who are unaware of GRT bus stop numbers and their locations. There are over 2,600 unique GRT bus stops in Kitchener-Waterloo-Cambridge (KWC) with unique bus stop ID numbers, and remembering the most common ones used by a GRT customer can become an issue. Since the app can display all bus stops on the map, and show their bus stop names and ID numbers when tapped on, it can really improve the usability of the EasyGo Next Bus Texting service. Another key group of users are people who do not have access to mobile internet. Popular maps and navigation apps like Google Maps or GRT's EasyGo Mobile app are great for getting bus directions or real time feeds for bus locations, however, these apps are dependent on a steady internet connection for information. GRT TXT on the other hand can function without an internet connection because the full dataset of all GRT bus stops is saved locally on the app itself. Lastly, the interface for GRT TXT was designed to be

as simple as possible with no gimmicky features, all it does is display the GRT bus stops, and allows the user to immediately send SMS from within the app.

1.2 Data Source

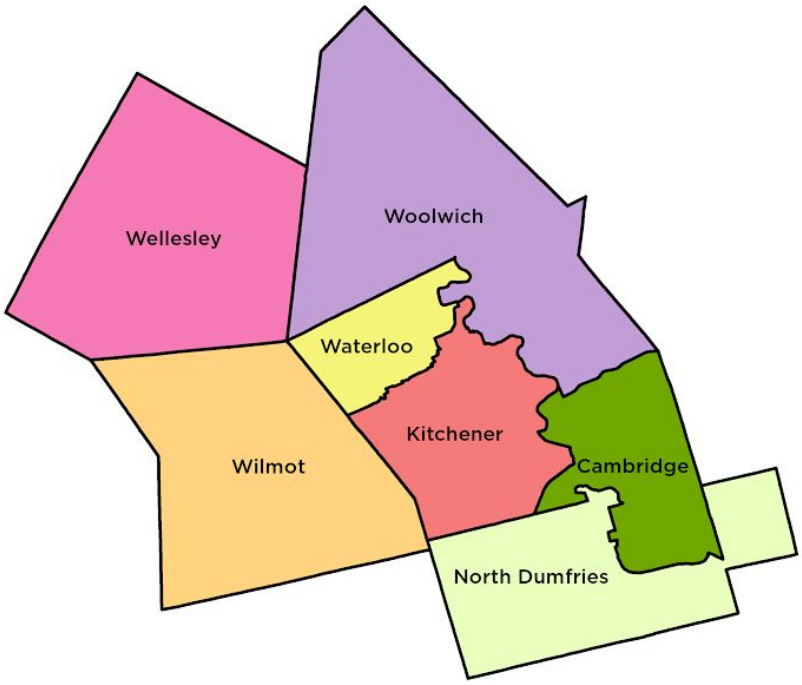


Figure 1: Map of Census Districts in KWC metropolitan area

For the scope of the project, only the GRT bus stops were considered to be included in the app because they are the ones that supported the service to SMS the bus stop code for bus times. More specifically the GRT bus stops from the City of Waterloo's open data portal were considered because they were last updated on October 3, 2016, and seemed to be most complete dataset. Another data source for the GRT bus stops was from the Region of Waterloo's Data Catalogue, but this dataset seemed unreliable and was only used to a small degree for quality control and assurance (QC/A). When observing all the bus stops for both the datasets combined,

Figure 1 shows the complete coverage of all the regions in the KWC metropolitan area, however, there were some bus stops that were listed in the dataset but not active in real life. Thus, indicating that the datasets needed further QC/A before including them in the app.

1.3 Data Licensing

For the both the datasets, GRT states that users of their open data are allowed to 'Copy, Publish, Distribute, and Transmit Information, meaning that we are allowed to do anything we want with the data and repurpose it for another use of needed. They also allow is to 'Use Information Commercially', meaning that we are allowed to use their open data in a commercial application if we choose to do so.

The app itself was made using Google Map's Application Programming Interface (API) and used the default Google Maps basemap for the basemap of the app interface. When using the Google Maps API however, we must sell or distribute the mobile application for free if we use their map because we do not own the basemap, or the functions available with the API. We are also required to avoid using private information about individuals because Google does not want malicious use with the location data of users.

As for the terms and conditions of Android Studio, the Android Development Package used to make the app, it can be used to solely develop applications for Android compatible implementations. We, as a third party developer, own the rights to our intellectual property. We can reproduce and distribute the app as an open source software license.

2.0 Methods

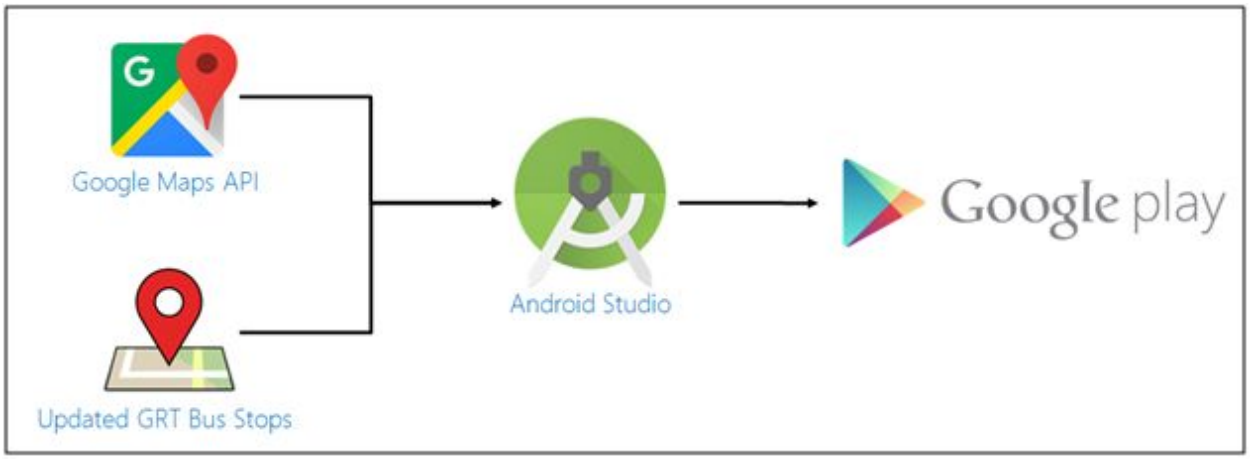


Figure 2: Workflow for app development and distribution

To develop GRT TXT, Android Studio was used as the development platform to create the application, as illustrated in Figure 2. Google Maps' API and the City of Waterloo's updated GRT Bus stop dataset were used as data components within Android Studio. Google Maps' API provides permission to access the Google basemap as well as the libraries for the API which was used to render GRT TXT. After building the application in Android Studio and testing it on our own phones, it was packaged into an Android Package (APK) which is an installer file to install the app onto any Android phone. Since the application is Android based, it made sense to choose Google Play as the distributing platform to share the application to the public instead of sharing the APK itself to others.

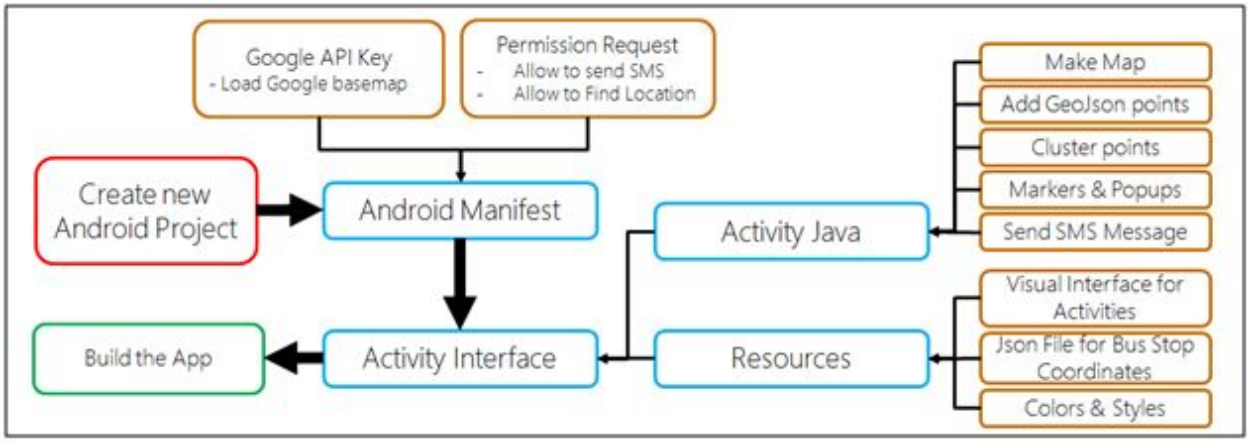


Figure 3: Main components of GRT TXT android studio application

Shown in Figure 3 are the main components of GRT TXT Android Studio package. After creating a new Android Project in Android Studio, all of the base files for the app are automatically created. We had to import the Google Maps Android Utility Library into the app's file system to allow the app to make maps, display points, and do much more with their API.

An XML file called Android Manifest holds the most important components of the app such as permissions, Google API key, and app parameters such as app name and version. The Google API key is required to load the Google Maps basemap, as well as use any of the functions available in the Google Maps Android Utility Library. Permissions are required to access the location of the user, as well as ask the user to allow the app to send SMS messages.

The Activity Interface is a XML file which represents what the user can do in the app. GRT TXT only has one activity, which is to display the map. The activity's Java file controls everything that goes on behind the scenes in the app from the moment the app loads. It controls tasks such as loading the map, reading the bus stops Json file, adding the bus stops as markers, binding popups to each marker, sending SMS messages, and clustering the markers. The activity

can call upon various resource files that render the visual interface, the raw files for the bus stop Json, as well as any colors and styles for the app. Appendix A to D will go into a deeper dive on the creation of the app, specifically into the code base of the Activity Interface's Java file. The entire code base is also available on Github https://github.com/jrmistry/GRText.

2.1 Data Quality Control and Quality Assurance

Perhaps one of the most important aspects of application development that utilizes large datasets would be the idea of data accuracy and consistency. Therefore, in order to produce accurate results, a significant amount of quality control and quality assurance procedures were required in order to reduce the amount of errors/inaccurate locations of bus stops when the points are added to the map. Figure 4 shows the general processing steps used to improve the accuracy and spatial integrity of the dataset.

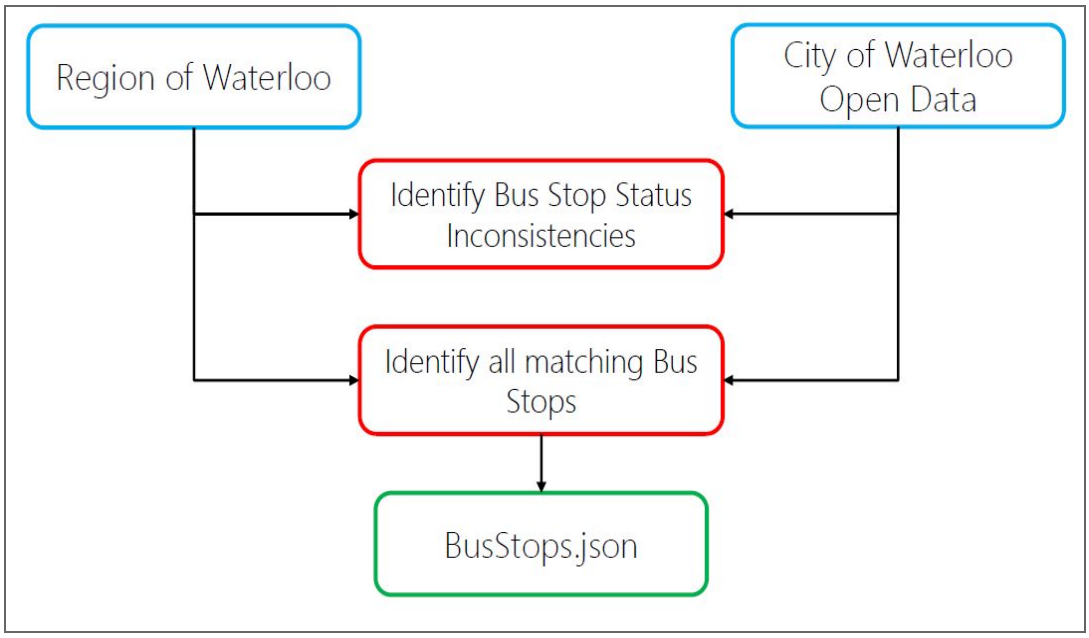


Figure 4: General QC and QA processing methodology

In essence, two GRT bus stops taken from the Region of Waterloo and City of Waterloo Open Data Portal were used in the quality control/assurance process. The reason for including the two datasets containing the same information was due to the large degree of inconsistency with regard to the geographical location of bus stops, duplication of bus stops, and active vs inactive stops. In order to remediate this issue, both datasets were imported into an Access database for comparison and processing using various SQL queries. First of all, it was important to identify all stops that were inactive (specified in the attribute table) in order to prevent data duplication between new and inactive stops with the same bus stop id. Once all of the inactive stops were removed, the next steps involved performing three database joins (Inner, Left, and Right) to identify all matching bus stops from both datasets based on a unique id field (Inner join), include all bus stops from the Region of Waterloo and only bus stops that match in the City of Waterloo Open Data Portal (Left join), and include all bus stops in the City of Waterloo Open Data Portal and only bus stops that match in the Region of Waterloo (Right join). By performing the three different joins it was discovered that 2339 bus stops had matching id's from both datasets. However, a significant portion of bus stops did not have matching id's from the Region of Waterloo (~ 500). In addition, while performing ground truthing of bus stops in google maps, it was observed that not only did the Region of Waterloo have reduced accuracy with regard to the individual bus stops, but also the geographical location of the stops when compared to the City of Waterloo data. For instance, it was observed that there were a cluster of 8 bus stops near the Davis Centre at the University of Waterloo (claimed to be active) but is not present on google maps as seen in Figure 5.Therefore, it was decided that the entirety of the City of Waterloo dataset was used instead of the Region of Waterloo to maintain data consistency.

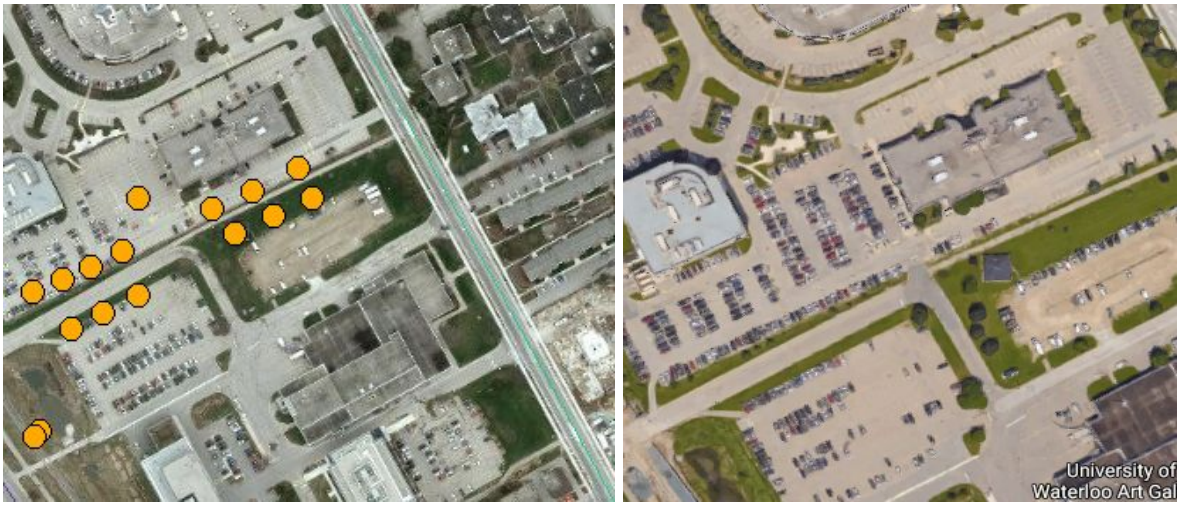


Figure 5: Cluster of bus stops (non existent) No bus stops present in google maps

Furthermore, the results of the different queries indicate that the City of Waterloo data was more accurate as 85 % of stops from the Region of Waterloo had matching id's. The other 15% (non-matching cases) included "problematic" bus stops from the Region of Waterloo including the cluster of bus stops from Figure 5.

Once the decision was made to use the City of Waterloo data, additional processing was done to remove duplicated bus stops from the data set. It was noticed that the duplicated stops were the result of the replacement of an old bus stop with a new one but keeping the bus stop id. However, this information was not updated in the data spreadsheet resulting in duplicated bus stop id's. Nevertheless, further investigation revealed that the newer bus stops had a larger FID field than the old bus stops. Therefore, a query was made to remove all duplicated bus stops while preserving the ones with a larger FID value. Once all of the duplicated bus stops were removed, the resulting database containing the updated GRT bus stop information was converted into a CSV file in which the Latitude and Longitude columns were formatted into a numerical

data type instead of text in order to be able to display the points in ArcGIS. Finally, the CSV was converted into a .JSON file using geojson.io which was placed in the GRT TXT app itself.

3.0 Results

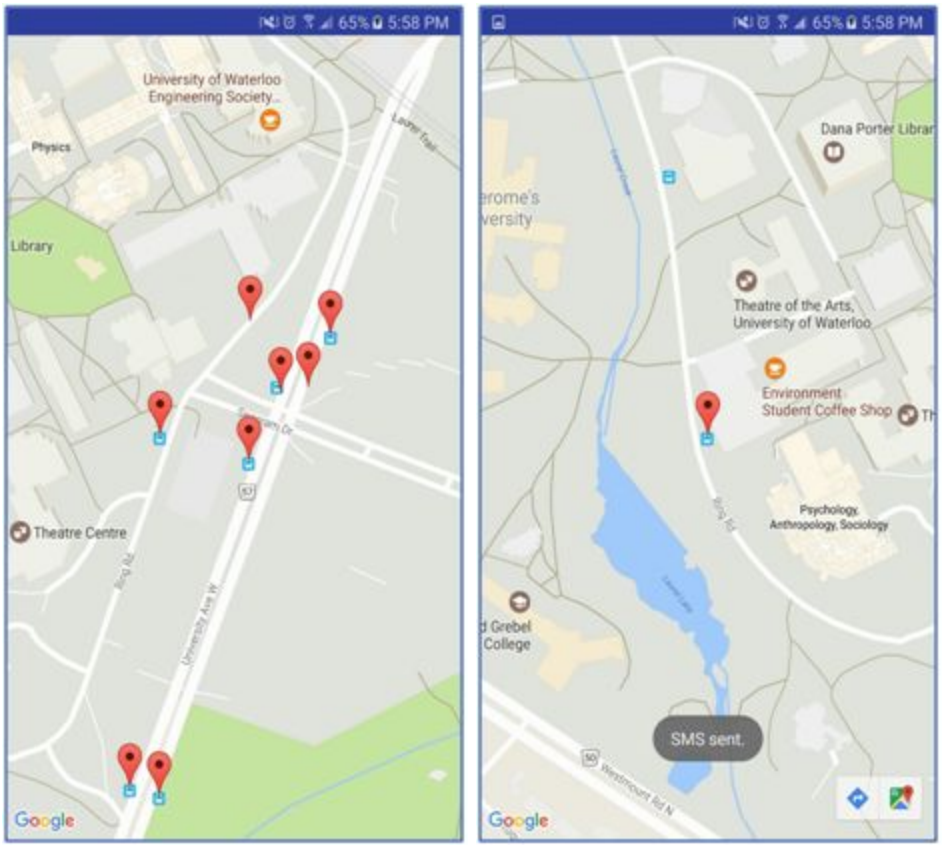


Figure 6: Map of GRT TXT and the sent SMS

The end product is a map utility application that enables users to inquire the arrival of buses to particular GRT bus stops. A map is immediately shown once the user opens GRT TXT, as seen in Figure 6. The application displays all available bus stops within the Region of Waterloo, and the user can zoom and pan throughout the Region to find active GRT bus stops. When the user zooms too far out of the map, the bus stops will get clustered to prevent overlapping of markers and allow improve readability.

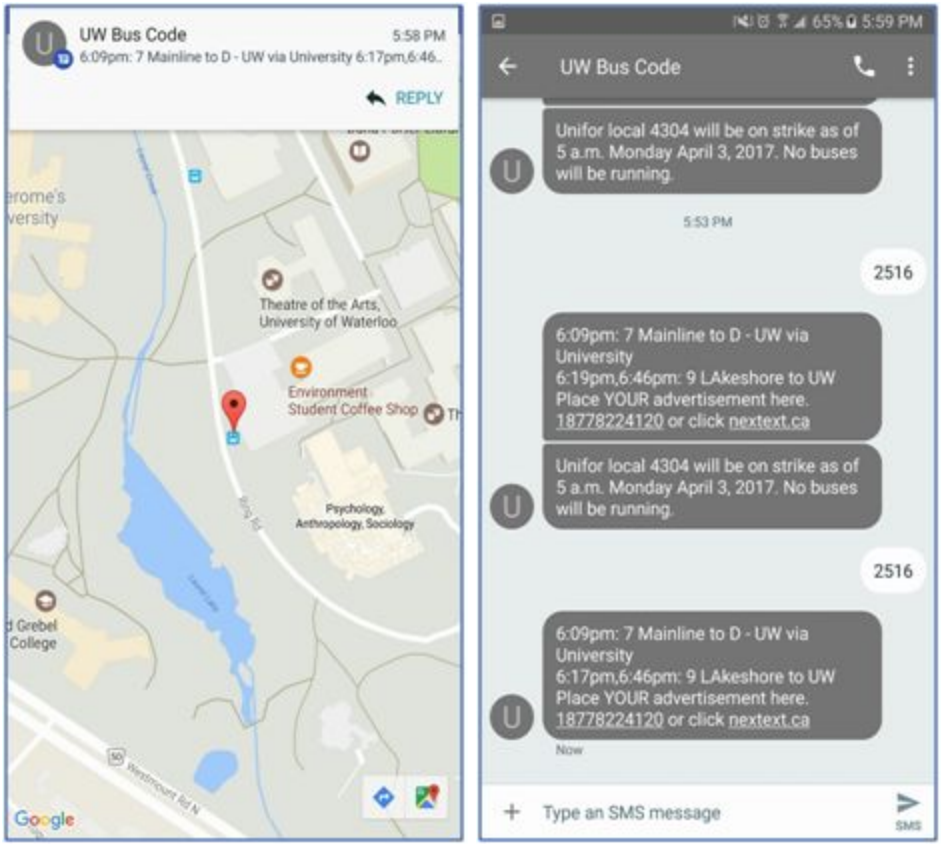


Figure 7: Notification of the bus time text and the text message of the bus timings

Clicking on a bus stop of interest will activate a popup message for the bus stop showing its bus stop code and name. Clicking the bus stop again will allow the user to send an SMS message to EasyGo to retrieve bus time information. The transaction is confirmed when GRT TXT tells the user that the SMS has been sent as seen in Figure 7. Once the transaction has been implemented, the user should expect an SMS reply message from EasyGo listing the next three buses scheduled to arrive in the bus stop selected.

3.2 Google Play

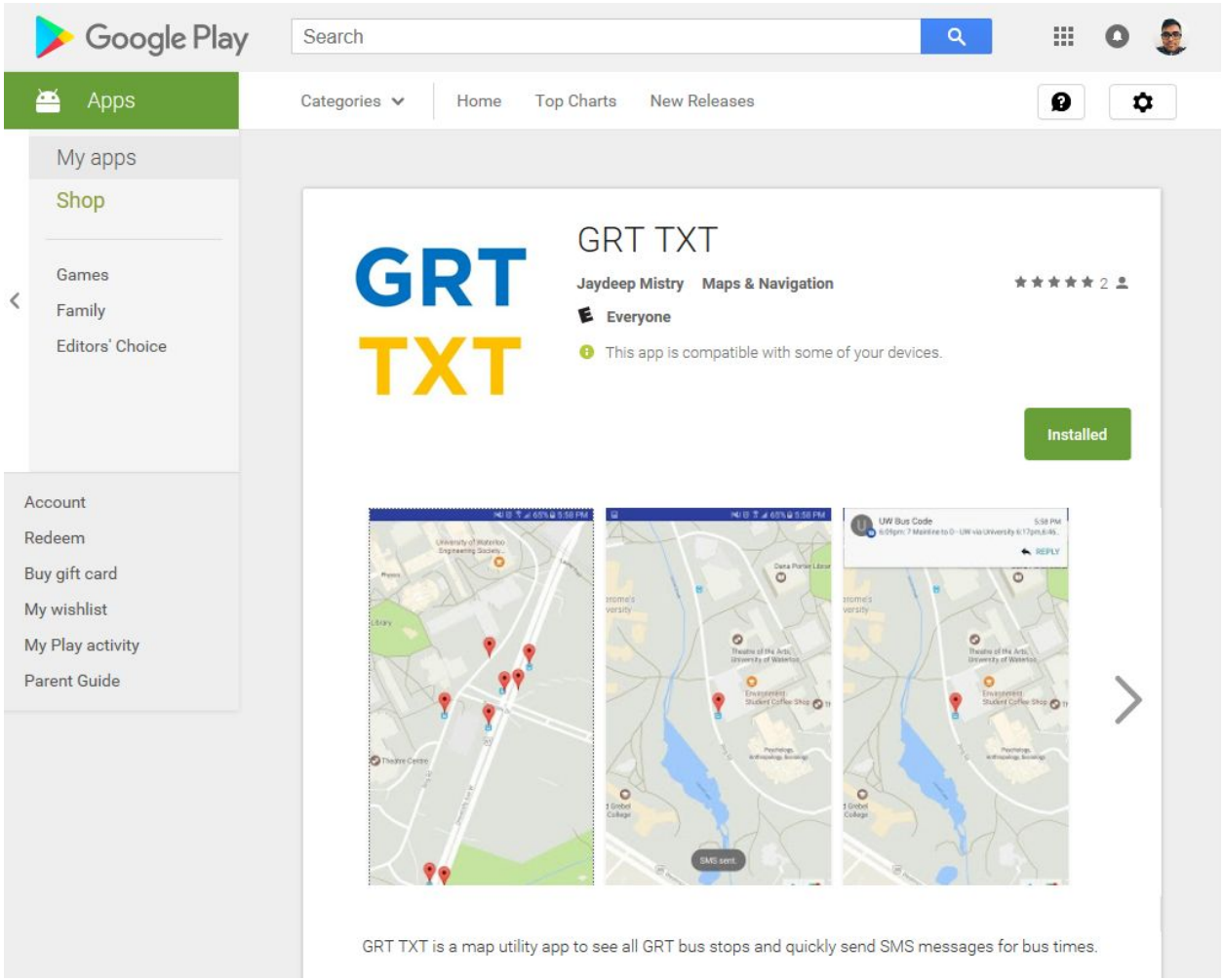


Figure 8: GRT TXT app available on Google Play

After developing the app, it was put up on the Google Play store for anyone to install and use. Making the app available on Google Play seemed to be the best decision for simple and easy distribution and sharing of the app. Since the release of the app on Saturday April 1st, 2017, there have been 12 install of the app and a whopping 5 Star average review of the app.

4.0 Discussions

4.1 Development Platform

The Android Studio was used because it allowed the ability to use the Google Maps library interface effortlessly, enabling the implementation of the code smoothly. With the Google Maps API library interface and the code, we can efficiently build the app on our phones. Google Play was used to easily distribute the application instead of sharing the APK installer of the application to each individual user.

4.2 Limitations

Throughout this development project, various limitations were encountered. Most notably however, would be the large degree of inaccuracies associated with the GRT bus stop datasets. For example, it became evident that there were duplicate bus stops for many locations as described earlier. Another prominent issue was that there were inaccuracies in the latitude and longitude coordinates of many bus stops, which ultimately reduced the accuracy of its geographical location. Despite efforts to eliminate all inactive bus stops, there were also some bus stops which were accurate, but would not respond to an SMS. Therefore, it is inevitable that users will receive error notifications claiming that a bus stop does not exist due to the degree of inconsistency present within the data.

Another limitation with the app correlates to the idea of interoperability in that it is only compatible with android devices and not Apple products that use the IOS operating system due to time constraints when building the app. Given more time, it would be possible to not only add more features to the app, but also develop for more platforms.

4.3 Cost

In regards to the total costs associated with building the app, 25 US dollars was required to create a Google Play developer account. However, the app itself is free for end users to install from the Google Play store. As for costs to develop the app itself, it took about 30+ hours of development time. For each line of code that was written, there was over 30 lines of documentation that was read to understand the functionality of the API and its properties.

4.4 Future Goals

Some future goals to consider when attempting to improve the application would be to improve the user interface through the integration of common functionalities such as the ability to geolocate the app user such that users would be able to identify all bus stops surrounding them as opposed to having a default map extent when the app is launched. Furthermore, since the bus stops layer is plagued with inconsistencies as mentioned, the ability to flag missing or incorrect bus stops would be beneficial. In addition, similar to bookmarking a favourite website, a functionality should be added to allow users to bookmark frequently used bus stops to augment the efficiency of obtaining bus arrival times. Finally, developing for apple's IOS will further

improve the interoperability of the app thus encompassing a greater range of telecommunication users.

5.0 Conclusion

Overall, it is highly recommended that Google Maps Utility API for android be used for development as it can not only leads to creative application solutions such as the our app, but also make the corresponding workflow seamless. In regards to data quality and assurance of open datasets, GRT needs to perform quality checks more often to ensure their data is accurate and consistent for end users so that issues associated with data quality don't arise when other developers try to create an app using data from the GRT website. Overall, it was possible to transform open data into a useful service by packaging it into an application with a user friendly interface. This entire application would not have been feasible to make if the GRT bus stops data was not freely available and we had to digitize the bus stops IDs ourselves. With future improvements to GRT's bus stop datasets, our app is also expected to improve in the overall accuracy and usefulness.

References

Android Studio (2016). *Terms and Conditions* Retrieved March 22, 2017, from https://developer.android.com/studio/terms.html

Android Studio (2017). *Documentation* Retrieved March 22, 2017, from https://developer.android.com/reference/org/w3c/dom/Document.html

City of Waterloo (2016). *Open Data Catalogue* Retrieved March 20, 2017, from http://opendata-city-of-waterloo.opendata.arcgis.com/datasets/e01be588229b4695b628f9801edf08e9_3?geometry=-81.891%2C43.307%2C-79.277%2C43.655

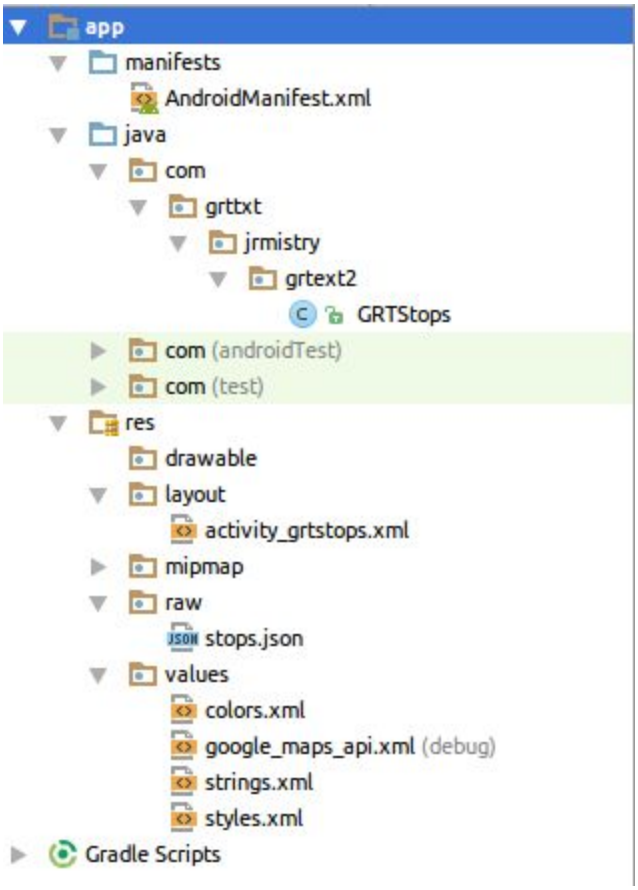
Google Maps APIs | Google Developers (2017). *Terms of use* Retrieved March 22, 2017, from https://developers.google.com/maps/terms

Google Play (2017). *Terms of Service* Retrieved March 22, 2017, from https://play.google.com/intl/en-us_us/about/play-terms.html

Region of Waterloo. (2010). *Data Catalogue* Retrieved March 20, 2017, from http://www.regionofwaterloo.ca/en/regionalGovernment/GRT_Stops.asp

Appendices

Appendix A: Android Studio



This represents the file system of the application. The major folders/components are the manifest, the java files, and res (short for resources). The manifest controls the major components of the app. The res holds all visual design elements and raw data. The java folder holds all java files that control the activities of the app.

Appendix B: Android Manifest

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.grttxt.jrmistry.grtext2">
4
5      <!--
6           The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
7           Google Maps Android API v2, but you must specify either coarse or fine
8           location permissions for the 'MyLocation' functionality.
9      -->
10     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
11     <uses-permission android:name="android.permission.SEND_SMS" />
12
13     <application
14         android:allowBackup="true"
15         android:icon="@mipmap/ic_launcher"
16         android:label="GRT TXT"
17         android:roundIcon="@mipmap/ic_launcher_round"
18         android:supportsRtl="true"
19         android:theme="@style/AppTheme">
20
21         <meta-data
22             android:name="com.google.android.gms.version"
23             android:value="9877000" />
24
```

Above is a section of the Android Manifest which defines the app permission required (lines 10,11). Lines 14 - 19 hold the name of the application and other features.

```
33         <meta-data
34             android:name="com.google.android.geo.API_KEY"
35             android:value="AIzaSyD9fj0ErpekAeFinguuLXQEU4Eiea5_VHo" />
36
37         <activity
38             android:name="com.grttxt.jrmistry.grtext2.GRTStops"
39             android:label="@string/title_activity_grtstops">
40             <intent-filter>
41                 <action android:name="android.intent.action.MAIN" />
42
43                 <category android:name="android.intent.category.LAUNCHER" />
44             </intent-filter>
45         </activity>
46     </application>
47
```

Above is the lower section of the manifest, most important thing to note is the Android Google Maps API key listed in line 35.

Appendix C: Activity Interface XML

```
1  <fragment xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:map="http://schemas.android.com/apk/res-auto"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:id="@+id/map"
5      android:name="com.google.android.gms.maps.SupportMapFragment"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context="com.grttxt.jrmistry.grtext2.GRTStops" />
9
```

The activity interface's visual component is written in XML. For our app, the file is very small because there isn't much to render. There is only one box, representing the map will be rendered and it will take up 100% of the space of the allowable screen.

Appendix D: Activity Java Script

```
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;

import com.google.android.gms.maps.model.Marker;
import com.google.maps.android.clustering.ClusterItem;
import com.google.maps.android.data.Feature;
import com.google.maps.android.data.geojson.GeoJsonFeature;
import com.google.maps.android.data.geojson.GeoJsonLayer;

import com.google.maps.android.clustering.ClusterManager;

import org.json.JSONException;

import android.support.v4.content.ContextCompat;
import android.util.Log;
import android.widget.Toast;

import java.io.IOException;
//import java.util.jar.Manifest;

import android.Manifest;
import android.telephony.SmsManager;
```

This specifies all the imports of the app. Imports being the external libraries required for the app to function and be used in the Java file.

```
35  public class GRTStops extends FragmentActivity implements OnMapReadyCallback, GoogleMap.OnInfoWindowClickListener {
36
37      //private static final String KEY_CAMERA_POSITION = "camera_position";
38      //private static final String KEY_LOCATION = "location";
39
40      private static final int MY_PERMISSION_REQUEST_SEND_SMS = 0;
41      String phoneNo = "57555";
42      String message = "";
43
44      private final static String mLogTag = "GeoJsonDemo";
45      private GoogleMap mMap;
46
47      private ClusterManager<MyItem> mClusterManager;
```

Above is the initial variables to be declared. This can include the permission to send an SMS, the phone number of the GRT service, the message, as well as the Google Maps layer for the GeoJson, as well as the cluster manager. The map object on line 45 has to be globally declared for it to be accessible by other components of the app. The Cluster Manager is needed to cluster the points when they get cluttered.

```
53      @Override
54      protected void onCreate(Bundle savedInstanceState) {
55          super.onCreate(savedInstanceState);
56          setContentView(R.layout.activity_grtstops);
57          // Obtain the SupportMapFragment and get notified when the map is ready to be used.
58          SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
59                  .findFragmentById(R.id.map);
60          mapFragment.getMapAsync(this);
61      }
```

Above is the OnCreate function that is called when the activity loads. The @Override statement on line 53 means that we are going to change the default implementation of the onCreate function to our own version and what we want it to do.

```
73      @Override
74      public void onMapReady(GoogleMap googleMap) {
75          mMap = googleMap;
76
77          // Add a marker in Sydney and move the camera
78          LatLng waterloo = new LatLng(43.468712, -80.539103);
79          //mMap.addMarker(new MarkerOptions().position(waterloo).title("Marker in Waterloo"));
80          mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(waterloo, 17));
81
82          mClusterManager = new ClusterManager(this, mMap);
83
84          mMap.setOnCameraIdleListener(mClusterManager);
85          mMap.setOnMarkerClickListener(mClusterManager);
86
87          mMap.setOnInfoWindowClickListener(this);
88
89          retrieveFileFromResource();
90      }
```

Above we will override the onMapReady function with our own implementation. The onMapReady function is called whenever the map is loaded, and passes a GoogleMap object type that represents the map in the activity. We are associating the GoogleMap object to our variable mMap so we can use it later on, and also asynchronously by multiple functions. Since

the app does not ask for the user's location, the default location to open it is set on line 80, where we move the camera to the coordinates set by the variable called Waterloo on line 78. We associate the Cluster Manager variable to cluster points for the map in the object mMap (line 82). We also have to allow the Cluster Manager to listen in on any changes to the view of the map or the markers on line 84 and 85. We also called setOnInfoWindowClickListener for the map object to this, meaning that whenever the map object is clicked, the function will pass this, being the map itself, as a parameter to itself when running the code. It seems a bit confusing but it is needed for the pop-up windows to function. Lastly, we call the function retrieveFileFromResource on line 89 to load the bus stops from the JSON file.

```
92      private void retrieveFileFromResource() {
93          try {
94              GeoJsonLayer layer = new GeoJsonLayer(mMap, R.raw.stops, this);
95
96              int x = 0;
97              for (GeoJsonFeature swag  : layer.getFeatures()) {
98                  if (x >= 0) {
99                      String str[] = (swag.getGeometry().getGeometryObject().toString()).split("(,)|(\\()|(\\))");
100                     //passToast(str[1] + " " + str[2]);
101                     //passToast(swag.getGeometry().getGeometryObject().toString());
102                     //passToast(swag.getProperty("stop_name") + " " + swag.getProperty("stop_id"));
103                     MyItem newCI = new MyItem(Double.parseDouble(str[1]), Double.parseDouble(str[2]), swag.getProperty("
104                     mClusterManager.addItem(newCI);
105                 }
106                 x = x + 1;
107             }
108
109             //addGeoJsonLayerToMap(layer);
110         } catch (IOException e) {
111             Log.e(mLogTag, "GeoJSON file could not be read");
112         } catch (JSONException e) {
113             Log.e(mLogTag, "GeoJSON file could not be converted to a JSONObject");
114         }
115     }
```

Above is the implementation for retrieveFileFromResource function. It has a try and catch statement, meaning that it will try to load the bus stop points from the json file, and if it fails, it will catch the error and not crash the app. Inside the try, we make a GeoJsonLayer and instantly load all the points using one command on line 94. Once the points are loaded, we go through a for loop, essentially looping through all bus stop points in the GeoJson and creating a cluster item for each point, and then adding it to the Cluster Manager. Since the Cluster Manager was

associated to the map object earlier, adding a point to the Cluster Manager will also add it to the map automatically.

```
118     public class MyItem implements ClusterItem {
119         private final LatLng mPosition;
120         private String mTitle = "";
121         private String mSnippet = "";
122
123         public MyItem(double lat, double lng) { mPosition = new LatLng(lat, lng ); }
126
127         public MyItem(double lat, double lng, String title, String snippet) {
128             mPosition = new LatLng(lat, lng );
129             mTitle = title;
130             mSnippet = snippet;
131         }
132
133         @Override
134         public LatLng getPosition() { return mPosition; }
137
138         @Override
139         public String getTitle() { return mTitle; }
142
143         @Override
144         public  String getSnippet() { return mSnippet; }
147     }
```

Above is the implementation of each cluster item. All the cluster item holds is the Latitude and Longitude coordinates of the point, the title and the snippet (body) of the content that we will display in the popup. The cluster item is a pre-existing class, however, MyItem is our own implementation of it because we want to allow added functionality to it.

```
149     @Override
150     public void onInfoWindowClick(Marker mark) {
151         //passToast("Info window clicked");
152         //passToast(mark.getTitle());
153         sendSMSMessage(mark.getTitle());
154     }
155
```

Above is the onInfoWindowClick function that we will override with our own implementation. Earlier we had a setOnInfoWindowClickListener function on the map which will pass this when called, so onInfoWindowClick will receive *this*, which is the marker that was tapped. Here we

will send an SMS whenever the user will tap the pop-up of the marker, and pass the title of the marker, which was the bus stop code.

```
168     protected void sendSMSMessage(String stop) {
169         message = stop;
170
171         if (ContextCompat.checkSelfPermission(this,
172                 Manifest.permission.SEND_SMS)
173                 != PackageManager.PERMISSION_GRANTED) {
174
175             if (ActivityCompat.shouldShowRequestPermissionRationale(this,
176                     Manifest.permission.SEND_SMS)) {
177
178             } else {
179                 ActivityCompat.requestPermissions(this,
180                         new String[] {Manifest.permission.SEND_SMS},
181                         MY_PERMISSION_REQUEST_SEND_SMS);
182             }
183         } else {
184             SmsManager smsManager = SmsManager.getDefault();
185             smsManager.sendTextMessage(phoneNo, null,message,null,null);
186             passToast("SMS sent.");
187         }
188     }
```

Above is the code for sending the sms message. The string stop will be the bus stop code we saw earlier. Here we will check first to see if the app does NOT have permission to send the SMS. Whenever it is the first time of using the app after installing it, this permission will be asked for. If permissions were not granted to send SMS, then it will request for permissions, and if they were granted, then it will send the SMS immediately.

```
190     @Override
191     public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
192         switch (requestCode) {
193             case MY_PERMISSION_REQUEST_SEND_SMS: {
194                 if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
195                     SmsManager smsManager = SmsManager.getDefault();
196                     smsManager.sendTextMessage(phoneNo, null,message,null,null);
197                     passToast("SMS sent.");
198                 } else {
199                     passToast("SMS failed, please try again.");
200                 }
201             }
202         }
203     }
204 }
```

Above is the code the request for permissions and store them once accepted.