Ottawa Traffic Application

COMP 4905 - Computer Science Honours Project

Final Report

Carleton University

Author: Fahad Tariq

Student ID: 100907078

Supervisor: Dr. Dwight Deugo

August 18th, 2015

Table of Contents

i. List of Figures

ii. List of Tables

iii. Abstract

To address various problems faced by commuters, I have used data provided by Ottawa Open Data to create an android application. I have focused on two such datasets pertaining to traffic and cycling. There are a wide range of useful data available for both motorists and cyclists such as red light traffic camera locations, traffic incidents (collisions) causing delays, road construction information, parking & carpool lots locations & capacities, bicycle ring & post locations, and many more. Up to the minutes screenshots from traffic cameras across Ottawa can be viewed at the touch of a button. All of this information is neatly presented in an easy to use android application. If used, such information can be of great assistance when commuting around the city or while visiting Ottawa as a tourist.

1. Introduction

1.1 Problem

Nowadays, smartphones are a common entity among drivers and with the Ottawa Open Data initiative, there is a potential to address a variety of problems faced by commuters in the city. If there is a traffic jam or road/lane closures due to construction, you rely on roadside digital displays and billboards to get updated information. If you live in the suburban areas of Ottawa, it would be beneficial to know exactly where park and ride (carpool) lots located. If you are visiting Ottawa as a tourist, you will need help in finding parking lots around the city. If you are a cyclist, then knowing which roads have bike lanes is advantageous and searching for rings/posts is also a problem. All of these problems can be mitigated by utilizing data provided by Ottawa Open Data.

The data addresses traffic related problems for everyday commuters by letting them know if there are any traffic jams on their commute due to traffic incidents and alerts. It also assists new travellers to the city by notifying them about construction events and parking lot locations around the city. There is also data to help cyclists find rings/posts when they are wandering around the city. It also solves another problem for cyclists by allowing them to see which roads have bike lanes, segregated bike lanes, paved shoulders. This will allow cyclists to plan their route optimally using this information in mind.

1.2 Motivation

The data provided by the City of Ottawa is aimed towards helping citizens. By creating a mobile application that pulls the updated data, it helps people to access it conveniently on the go when they need it. Traffic data can assist users to make wise decisions for travelling within the city based on delays caused by incidents or construction work. Cycling data would be ideal for people in the summers to find the closest cycle rings and posts and also see which path is suitable for travel based on bike lanes data.

1.3 Goals

My primary goal is to make efficient use of the provided data to assist people on the go.

- The mobile application should be able to display all traffic alerts (incidents, construction, special events). This information should be displayed on top of Google maps (with traffic data enabled) so that users can easily see if certain alerts are responsible for slow traffic.
- The application should provide traffic camera locations and associated images
- The application should also provide locations for red light camera locations, parking lots, part and rides, cycling ring and posts
- For cycling network, the application should provide the cycling paths in the city on the map as well as information for bike lanes, segregated bike lanes, paved shoulder, etc.
- If there is time, a special feature will be made for red light camera locations. If a user activates detection of red light camera location, the application will monitor the GPS location and play a sound when approaching a red light camera.

1.4 Objectives

- The mobile application will display traffic events on top of Google maps interface
- Traffic camera locations and associated images can be viewed through the application
- The application should also provide locations for red light camera locations, parking lots, park and rides, cycling ring and posts
- For cycling network, the application should provide the cycling paths in the city on the map as well as information for bike lanes, segregated bike lanes, paved shoulder, etc.

- Given enough time, red light camera detection will be implemented based on GPS location to alert user with a sound

2. Background

The main purpose of this project for me was to gain experience using JAVA and more specifically, android development. Coming from a background of web development, I wanted to see what Android had to offer and hence I chose a project involved with Google Maps (an integral part of android) and Ottawa Open Data APIs [1]. Open Data is a great initiative and the city of Ottawa has made available a lot of interesting datasets for developers to consume. After going through various datasets available through the Ottawa Open Data portal, what caught my eye was the amount of information available pertaining to commuters. Surely, a mobile application which gives you real-time information on traffic incidents, construction, parking lots, carpool lots, bicycle ring & posts locations would be beneficial to a good subset of Ottawa residents and visitors. For me personally, working with various different APIs would be a great experience to getting started with Android development.

After I had a rough idea of the datasets that I wanted to build the application on, I researched other related traffic applications which might be similar to mine. After searching for “traffic” on Google Play Store, I came across a couple of popular applications, one made for Utah and the other for Malaysia. UDOT Traffic [2] application was built by Utah Department of Transportation to provide commuters and travelers with information such as emergency alerts, road condition alerts, incidents, special event alerts, construction alerts as well as lane closures and weather information. After using the application, I got a few ideas on how to structure my proposed application. Firstly, by having alerts, incidents, construction separately, it seemed to be a hassle for the user to use them all. I decided to have my application be more user-friendly by displaying such related information together on the map and giving the user the ability to filter out each specific item if they chose to do so. This approach would result in lesser items in the main menu and hence less complex to navigate through. Also, having google maps be on traffic mode was very user friendly as it allowed me to easily see which incidents and construction was responsible for traffic jams. Traffic camera images were also incorporated in the application but required the user to navigate away from the map activity to view it. A better user experience would be to have the image be displayed as an alert or dialog on top of the map screen.

The other application I researched was MY-Traffic [3]. This application is labelled as a community traffic monitoring system for Malaysian residents. It shows traffic incidents, road statistics, traffic camera images, as well as a community-driven traffic incident system. This application had all traffic related markers on the map together with no way to filter out specific ones, such as camera images or incidents. Traffic camera images were included in the marker info window so the user didn't have to navigate away from the map at all. I wanted my application to be closer to MY-Traffic in the user interface area since it seemed to be a much better experience then UDOT.

In terms of bicycle related data, I found an application called Warsaw Bike Path [4], which shows the bike paths in the city, bike lanes, bike racks. It provided all this information using various different colours on the map to show the paths. With the cycling network resource provided by Ottawa Open Data, a similar type of information could be displayed for the city of Ottawa. For cyclists, this type of a resource is very helpful when planning out a safe riding route around the city. The locations of bike racks is also an invaluable resource as you don't have to keep riding around the area in search of places to stow your bike. The cycling network resources provided by Ottawa Open Data consisted of KMZ, SHP, CSV, dwg, XML, and GeoJSON. The overall bike paths were conveyed only by KMZ, SHP, dwg and GeoJSON as CSV only shows the type of paths and their lengths whereas the XML resource could not be loaded due to a server error. After researching on each of KMZ, SHP, dwg and GeoJSON, I decided to go with GeoJSON since it seemed to be the simplest option when it came to parsing. GeoJSON [5] is an open standard format used to encode geographical features using Javascript Object Notation (JSON). After going through the GeoJSON specification, it didn't seem very difficult to create a parser and hence I decided to parse it myself instead of using a library such as GeoTools [6].

Similarly, all other datasets that I would be using were in JSON format which can be handled using Java's JSON libraries. One of the datasets, related to bicycling, was in CSV format which is also simple to parse without using any external libraries. Due to my limited exposure to Android and Java programming, I wanted to rely less on external libraries and build stuff myself. I also had to research Android features, such as network connectivity and application perimissions, using the Android User Guides [7] to see how Android handled such stuff and the limitations imposed by it.

2.1 Project Scope

The main purpose of the proposed android application is to expose to the general audience the variety of datasets available from Ottawa Open Data. Traffic datasets would be presented with a simple interface on top of Google Maps so that users can easily explore the information. In addition to mapping the traffic camera list provided by one of the datasets, there is also the ability to view up-to the minute images from such cameras. Traffic delays caused all around Ottawa due to collisions, extreme weather, etc. are neatly reported on top of a Google Maps interface which shows traffic congestion on the roads. Hence, users can identify why certain traffic jams are caused by viewing the nearby traffic incidents, construction or special events markers.

The software is also favorable towards bicyclists as it shows locations of bicycle ring & posts around the city as well as specifying bike paths and bike lanes available. Parking lots as well as carpool parking lots around the city can be viewed easily through the application.

3. Approach

3.1 Overview

Since all datasets that I would be incorporating are aimed at commuters, it was a no brainer to develop a mobile application which is readily available on smartphones, for people on the go. One of the first things I considered was developing an intermediary web server between the mobile application and the Ottawa open data APIs. It would be up to the web server to handle all incoming requests as well as store the latest traffic camera images so that the application wouldn't be restricted to Traffic Ottawa's limitation of use of their APIs. After weighing the pros and cons of a web server, I ultimately decided against making one since it didn't seem like it would be too beneficial. There were also legal ramifications of storing traffic camera images on the web server without having explicit permission to do so.

For the mobile application, I wanted to keep the UI as simple as possible since this application is aimed at commuters who would be using it on the go and as such require, a simple UI which they can easily and quickly navigate. Also, all data had a location on the map so incorporating everything with Google Maps was essential. To maximize the usefulness of the various APIs, I decided to group together similar datasets and in the case of traffic related sets, Google Maps was set to show traffic mode to give users a visual depiction of traffic problems alongside the data items.

3.2 Architecture and System Design

3.2.1 Overview

This section provides the approach used to design the system and the overall architecture used for the application.

3.2.2 High-Level Subsystem Decomposition

The high-level subsystem decomposition shows the major components of the system: models, business logic, and activities.

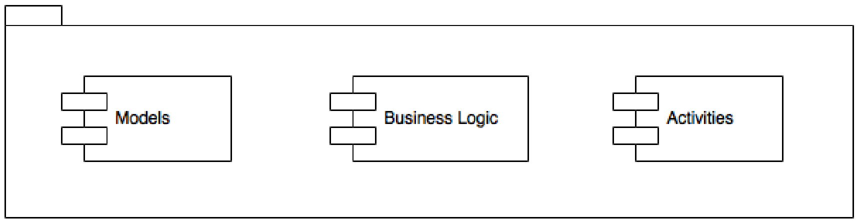


Figure 1 - High-level Subsystem Diagram

The above figure shows the high-level subsystems in the system. Since there is only one layer in the application, all of these subsystems exist on the same application layer.

3.2.3 Detailed Subsystem Decomposition

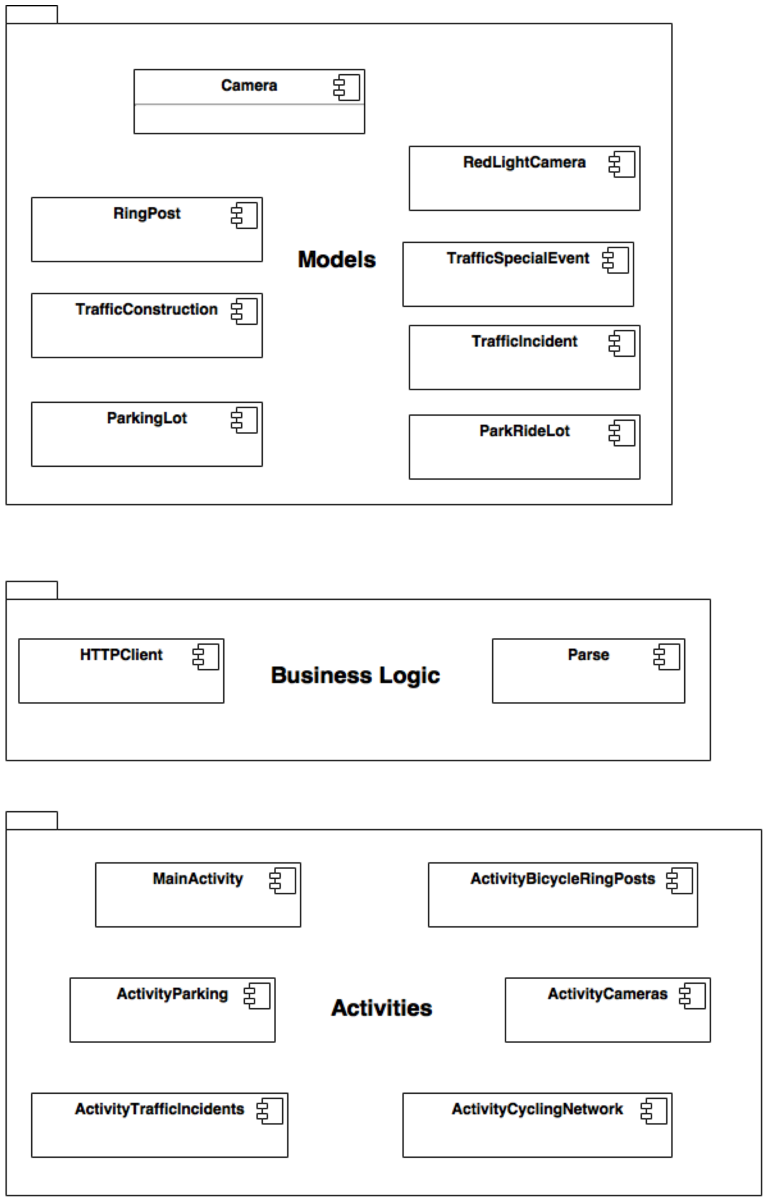


Figure 2- Detailed Subsystem Diagram

The above diagram shows the different subsystems in the application and the components within them. The models subsystem contains all the Java objects that are used throughout the application and they resemble very closely to their associated JSON representation as received from the open data APIs. The activities subsystem consists of the various activities in the application. Activities are a mix on view and controller functionalities from the MVC design architecture [8] and hence keeping separate views for each distinct group of dataset was necessary. To avoid code duplication, common code between different activities was modularized and put under the business logic subsystem. The business logic subsystem consists of the parsing and HTTP connectivity functionality which are utilized by all activities in the system. Both these inner classes conform to different design patterns which are discussed in the next section.

3.2.4 Application Architecture and Design decisions

Instead of forcing a new architecture, I chose to stick with the default Android architecture which loosely resembles a Model-View-Controller (MVC) approach. The benefits of MVC stem from the fact that loosely coupled relations between each component (models, views, and controllers) allows for a very modular approach of coding but Android does not incorporate a true MVC since Activities act both as controllers and views. This decreases the coupling between views and controllers and degrades the full benefits of MVC. Even though it may not be a true MVC architecture, it still results in neat organization of the code, both physically and logically in the application itself.

Also, I chose to separate some of the business logic away from controllers to decrease the coupling and make the system design more modular. Two major pieces of the system, the Parsing subsystem and the HTTP connection subsystem, are both loosely coupled with the activities which makes for more robust code. To implement the parsing subsystem, I used the Strategy design pattern [9]. Although only two strategies were needed for the complete system (CSV and JSON parsing), the strategy design pattern allows us to easily add more strategies if the need arises. It makes for very loosely coupled code and offers a great advantage for extensibility in the future versions of the application. Adopting this design pattern offers us the flexibility of incorporating any type of API into the system. Even if the API is a different format, other than CSV and JSON, it is easy to use the design pattern to add more strategies to cover the new format. This makes the application much easier to maintain in the future since any updates in the source API formats can be handled easily.

Whenever an android application is running, the foreground processing is done in the User Interface (UI) thread. Since this thread deals with all the UI related stuff as well as listeners, it can remain quite busy throughout the running time of the application. To make use of multiple threads, android provides an AsynchTask [10] class which gives you access to non-UI threads to complete processing intensive tasks or slow tasks such as network connections. The HTTPClient connection subsystem is constructed to maximize the usage of Android's built-in AsynchTask class. This means that all network operations are carried out on background threads instead of the UI thread which makes the application much more accessible and reliable for users. Since AsynchTask requires a context to an Activity, there is some coupling between the two subsystems. Also, by using a delegation style callback interface, our controller code remains in the activity subsystem, instead of the HTTPClient class, making the coupling between activities and HTTPClient as low as possible.

The models hold all the data that we parse and process and it is used only during run-time. Since this data needs to be kept updated, there is no persistent data management in the software design. I considered a database design to assist this users' needs but the processing overhead of synchronizing local and remote data from the APIs would have been considerable, given the size of some of the APIs that are used in the application. Data retrieved from the city of Ottawa servers is held temporarily using the object model during the runtime of the application. This has the added benefit of not consuming a lot of space on the user's mobile phone as well as not wasting processing time to store the data into a database or flat files. The drawback is that the application is not usable without an internet connection which hardly seems like a drawback at all since this type of information is only beneficial in real-time.

4. Results/Validation

4.1 Evaluation of Project Proposal Objectives

To compare the results of the system, I will be evaluating each of the objectives that were mentioned in the project proposal.

Objectives	Evaluation
1) The mobile application will display traffic events on top of Google maps interface	Successfully completed All traffic events provided by Ottawa Open Data were successfully integrated with the application.
2) Traffic camera locations and associated images can be viewed through the application	Successfully completed A successful implementation of camera locations and associated real-time images were incorporated into the application.
3) The application should also provide locations for red light camera locations, parking lots, park and rides, cycling ring and posts	Successfully completed All mentioned items in the objective have been successfully employed into the application.
4) For cycling network, the application should provide the cycling paths in the city	Partially completed/few bugs & glitches

on the map as well as information for bike lanes, segregated bike lanes, paved shoulder, etc.	The sheer size of the cycling network data proved to be bothersome and caused slowdowns in the application. Explained in more detail below.
5) Given enough time, red light camera detection will be implemented based on GPS location to alert user with a sound	Not attempted due to not enough time remaining This was an optional objective which would have been pursued, given enough time, but unfortunately there was no time remaining for it.

Table 1 - Evaluation of Objectives

4.1.1 Objective #1 Results/Evaluation

Objective: *The mobile application will display traffic events on top of Google maps interface*

This objective was set up so that a successful implementation of Google Maps along with the API data can be used to show to the user a visual representation of all traffic events relative to their position in the city. If somebody is residing in the east end of the city, then viewing traffic events for western Ottawa are of little importance, hence the detection of the user's current location allows to show relevant traffic events quickly and without the hassle of extra zooming around the city.

4.1.2 Objective #2 Results/Evaluation

Objective: *Traffic camera locations and associated images can be viewed through the application*

The dataset provided a list of traffic cameras around the city and Traffic Ottawa has an API which allows to see the real-time traffic camera image. Access to this resource intensive API requires signing up and declaring the intended usage of the API. The usage of the API has the ability to designate a user ID which meant that each different installation of this application required a distinct user id when requesting traffic images. After researching a way to get some sort of a unique phone ID, I didn't find an exact answer since the corresponding android function "Secure.ANDROID_ID" is sometimes not implemented properly by third-party smartphone manufacturers (such as Motorola, Samsung, etc) which meant the same android_id being returned for every phone. After researching on a workaround, I stumbled across an answer on stack overflow by Joe [11] which suggested the use of several different IDs, together with a hashing function. This approach seemed like it would work on almost all android phones hence I went along with it. The actual traffic image is shown briefly as a toast notification (built-in android temporary notification on screen) so that the user does not have to needlessly interact with it again to close the image and open another one.

4.1.3 Objective #3 Results/Evaluation

Objective: *The application should also provide locations for red light camera locations, parking lots, park and rides, cycling ring and posts*

All the different datasets mentioned in the objective were implemented in various activities throughout the application. I grouped similar datasets together hence red light camera locations are accessed alongside traffic camera information and so on for other datasets as well. Being able to identify your current location on the map and find the nearest parking lot or the nearest cycling post can be easily achieved hence this objective is completed successfully.

4.1.4 Objective #4 Results/Evaluation

Objective: *For cycling network, the application should provide the cycling paths in the city on the map as well as information for bike lanes, segregated bike lanes, paved shoulder, etc.*

The cycling network objective was meant to show the user all the different cycling paths in the city. There was also enough information present in the given API to distinguish between bike lanes, segregated bike lanes, paved shoulder. It would have been an invaluable tool for cyclists all over the city.

Despite the many different formats available to consume the data, the limitation of developing for the Android Google Maps API meant that only a few of the formats were feasible. It came down to KML and GeoJSON in the end and despite the Javascript Google Maps API having the ability to load KML files, there is no such equivalent feature for the Android Google Maps API. The choice left was to parse either the KML or the GeoJSON myself and I went with GeoJSON.

After completing the parsing of the GeoJSON file and drawing the route paths on the google maps using Polylines, it was very much apparent that speed was going to be an issue. Parsing the GeoJSON itself was fast but the polyline function of Google Maps to draw every path took its toll on processing speed as well as the RAM of the mobile phone (application started to hang if too many routes were drawn on the map). The paths itself were correctly drawn on the map but the slothfulness of the drawing paths and the sluggishness of the UI meant that the cycling network isn't really usable right now. It is not apparent if this sluggishness is due to limitations of the polyline function in Google Maps or due to the sheer number of the paths that need to be drawn on the map.

To improve this situation, reviewing the algorithm of drawing paths for possible improvements would be a start. Another possible direction to explore would be to give KML a try to see if it might be parsed faster with Google maps. Due to lack of time available to spend on the project, I couldn't explore these options.

4.1.5 Objective #5 Results/Evaluation

Objective: *Given enough time, red light camera detection will be implemented based on GPS location to alert user with a sound*

This optional objective wasn’t reached due to the difficulty encountered with the bicycle cycling network. Had the cycling network functionality not met with numerous glitches, there might have been time to tackle this objective to implement a red light camera detection based on the users location and checking if they are approaching a red light camera. It would have been a fun exercise but unfortunately there wasn’t time to work on it.

4.2 Results/Self-Evaluation of Application

This project culminated with the creation of an android application. The application can be installed on most smartphones running the android operating system and since there are no services nor a local database, it doesn’t bog your phone down at all. Meeting most of the objectives means the overall project can be considered successful.

Since I had no background in android development, I had to learn everything through the android docs and by experimenting myself. My background in using PHP frameworks tailored towards MVC patterns such as CodeIgniter [12], definitely helped to get a grasp on how things are done in android. The various Java libraries means that you don’t have to reinvent the wheel yourself every time. The usage of an AMD processor computer also stifled my progress somewhat since only Intel CPUs support Intel Hardware Accelerated Execution Manager (HAXM) [13] which meant that I couldn’t use the built-in emulator of Android Studio. Instead, I had to rely on an external emulator such as Genymotion [14] as well as using USB debugging with my actual smartphone.

4.3 Validation & Application Screens

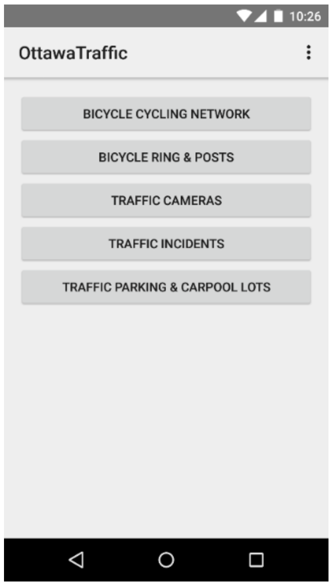


Figure 3 - Main Menu

Once the OttawaTraffic application is copied over to the phone, you can install it after accepting the permissions it requires. It requires the following permissions:

- Internet
- Access Network State
- Read Phone State
- Read GServices (use of Google Maps requires google play services to be enabled on the phone)
- Access Coarse Location
- Access Fine Location

The OttawaTraffic application has no user registration nor any synching mechanism with an intermediary web server hence when you open the application, you are taken straight to the main menu. I grouped together similar datasets so that the menu isn't cramped with a lot of different items. Similar items such as traffic construction, traffic incidents and special events are grouped under the same heading of traffic incidents. Similarly parking lots and park and ride (carpool) lots are under the same heading. This makes it so that the user isn't overwhelmed by all the different options present in the application. Making the application more professional looking or stylish wasn't a priority but if it were then a loading splash screen as well as stylized buttons would be the first improvements.

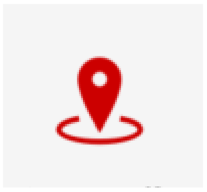
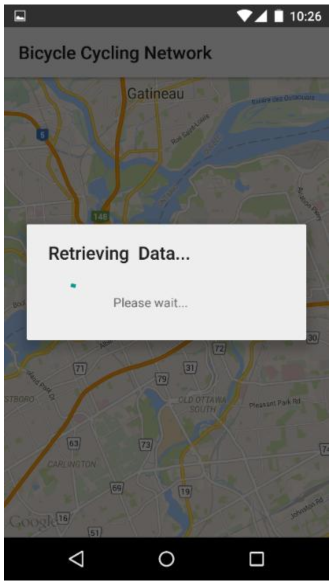


Figure 4 - App Logo



Retrieving Data...

Please wait...

Figure 5 – Downloading GeoJSON and drawing Paths on the map

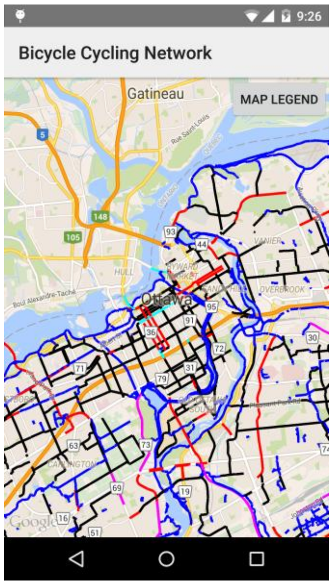


Figure 6 – The Cycling Network with all paths drawn on the map

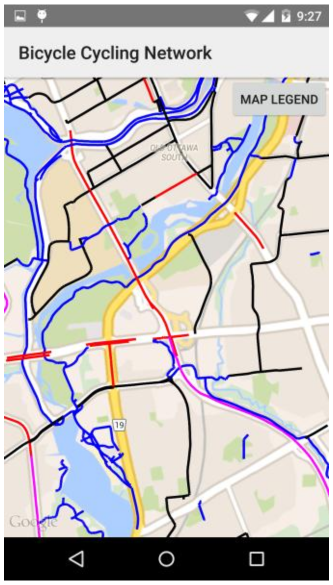
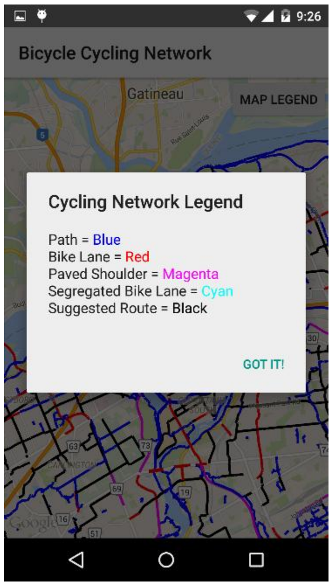


Figure 7 – Zoomed in display of some paths on the map



Cycling Network Legend

Path = Blue
Bike Lane = Red
Paved Shoulder = Magenta
Segregated Bike Lane = Cyan
Suggested Route = Black

Figure 8 - Map Legend

The bicycle network downloads approximately a 4.5MB GeoJSON file when it is opened. After it is completely downloaded, the parsing begins and polylines are drawn on the map. Depending on the type of path being drawn, the polyline color is changed. A map legend button in the corner shows the user an alert dialog so they can see exactly which colors correspond to which features on the map. Zooming in close on the map allows you to see exactly where the routes go to. As discussed in section 4.1.4, there is a sluggishness when using polylines in Google Maps. I am not sure if it is due to the sheer amount of polylines drawn on the map or problems with my algorithm but the cycling network is not fully functional at the moment. It takes more than an hour to draw all 6920 features on the map and the Google Maps itself becomes very unresponsive and slow as a result. I have the code restricted to show only the first 200 features as of now so that it can be viewed and tested. Improvements to the code base or the use of an alternative algorithm to draw the paths on the map might result in better speeds.

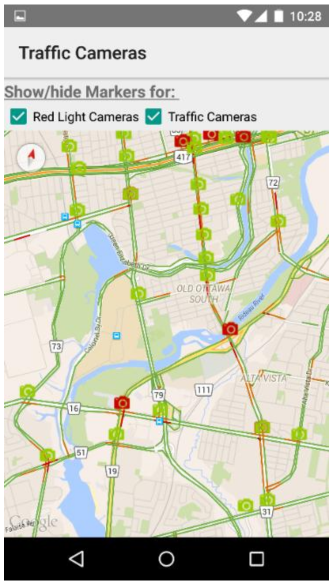


Figure 9 - Traffic Cameras

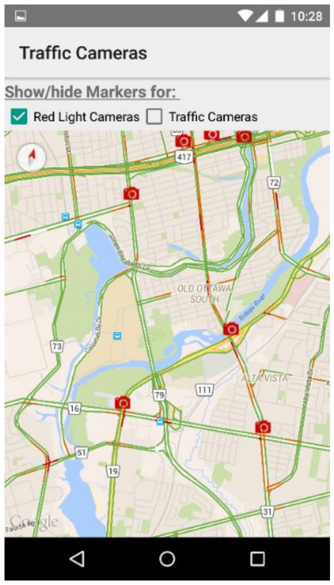


Figure 10 - Only showing Red Light Cameras

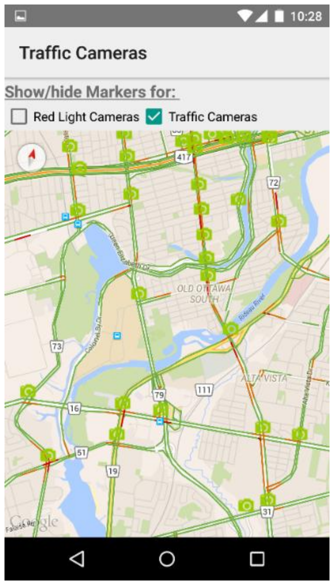


Figure 11 - Only showing Traffic Cameras now

The Traffic Cameras activity handles the display of all CCTV traffic cameras and red light camera locations. The checkboxes allow the user to filter out which items are showed on the map. By default, when this view is opened, both checkboxes are checked so that all items are shown on the map. The custom marker images on the map make it easy to differentiate between the normal traffic cameras and red light cameras.

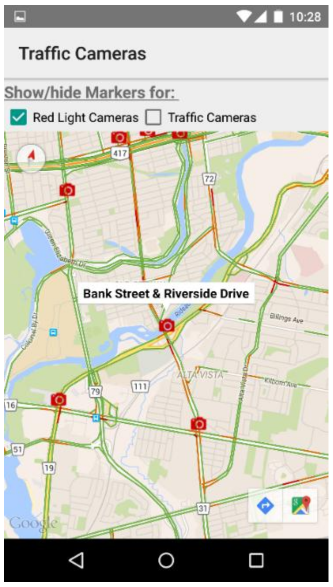


Figure 12 - Upon click of Red Light Camera

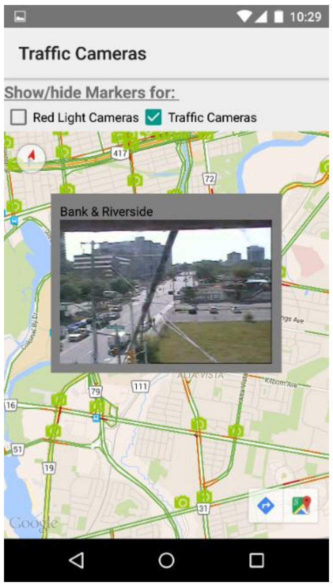


Figure 13 - Upon click on a Traffic Camera

If you click a red light camera marker, you will see the name of the intersection that it is located at. This data is provided by the API and not calculated manually. On the other hand, a click on a normal traffic camera displays a toast notification with a custom view. The custom view uses an ImageView element to show the traffic camera image on the screen. The location of the intersection is displayed above the camera image so that the user knows which intersection is being displayed. As is the nature of toast notifications, after a few seconds the notification automatically disappears without user input. This makes it easy for the users to view multiple camera images in quick succession. Some cameras are operated by the Ministry of Transportation (MTO) and their title indicates this. The majority of cameras are city cameras though hence there is no indication of a city camera within the title. Based on the restriction imposed by Ottawa Traffic, the images can only be accessed every 60 seconds.

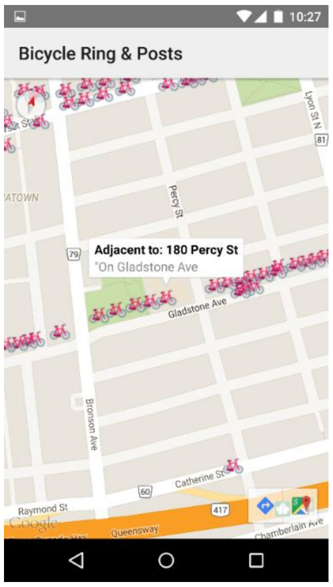


Figure 14 - Rings & Posts

The bicycle ring and posts section shows all ring and posts locations around the city. Although the majority of the locations are based in and around the downtown area. A custom marker image of a bicycle is used to go along with the kind of data that is shown on the map. Upon clicking a bicycle, the user is shown the address that the ring/post is adjacent to in the title of the popup dialog. Although not present for every ring/post, some also have a description which tells you exactly which street the ring/post is located on. I could definitely see this information being valuable to a number of cyclists in the summer months so that they can easily identify where ring/posts are located. The two bottom right icons, automatically created by Google Maps when a marker is clicked, allow the user to either open the specific marker location in a stand-alone Google Maps application or to get directions to the location.

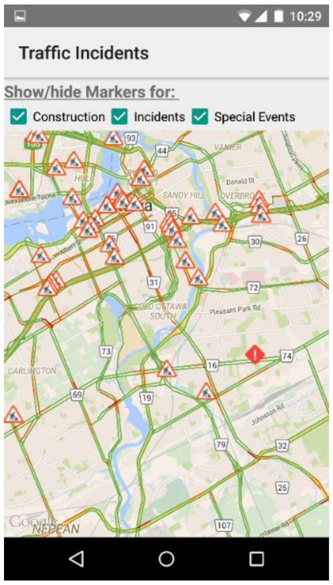
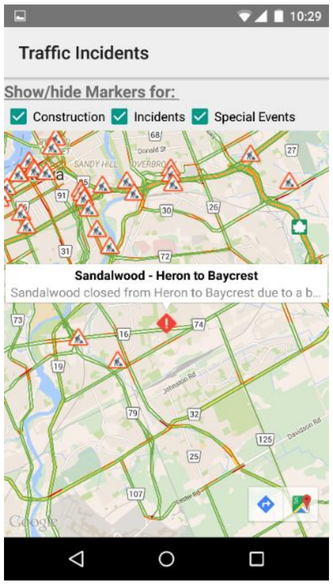
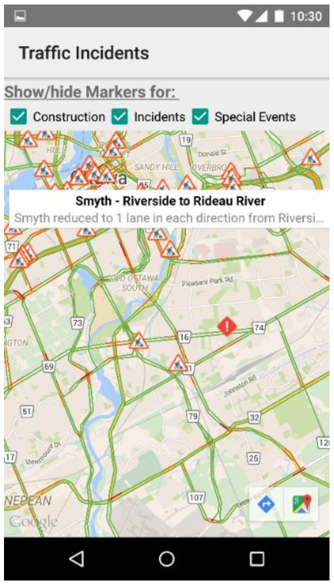


Figure 15 - All Traffic Incidents showing



Sandalwood - Heron to Baycrest
Sandalwood closed from Heron to Baycrest due to a b...

Figure 16 - Clicking an incident marker



Smyth - Riverside to Rideau River
Smyth reduced to 1 lane in each direction from Riversi...

Figure 17 - Clicking a construction marker

The above three screenshots of the application show the gist of the traffic incidents page. The traffic incidents view allows the user to view 3 key traffic related information: traffic incidents, construction, and special events. The triangle marker identifies a construction element whereas an exclamation marker is for traffic incidents and lastly, a blue star shows special events. Due to it being summer, the map is dominated by construction markers all around and clicking one of them will show you the details as well as any lane closure information associated with the work. Similarly, clicking a traffic incident shows you the details in a popup window on the map. It has to be noted that the popup window uses a custom info window contents so that the message is not stripped off at the end. Since construction messages tend to be longer, it was necessary to implement this so that the user is able to read the whole message quickly. During the course of the project, I wasn't able to see any special events being popped up but the specs of the API on the Ottawa Traffic website made it possible to implement it.

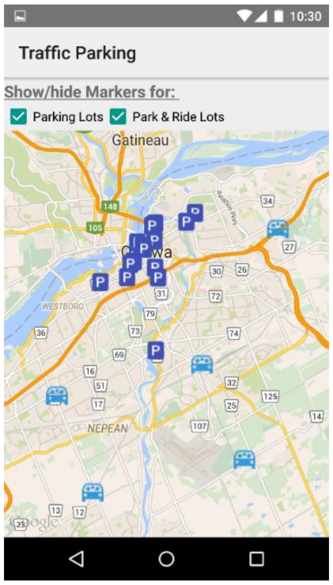


Figure 18 - Showing all the parking and carpool lots

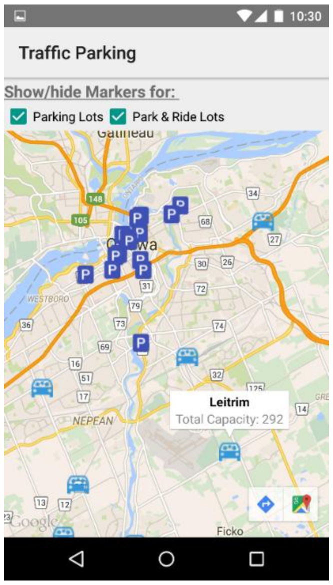


Figure 19 - Upon clicking a carpool lot

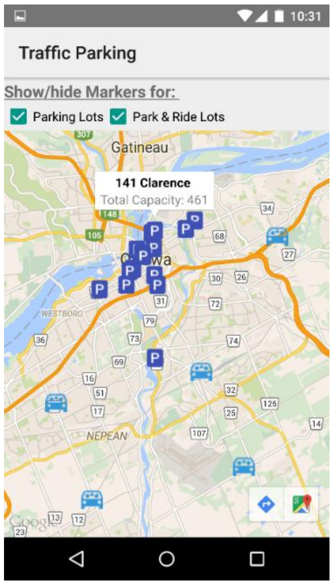


Figure 20 - Upon clicking a parking lot marker

The above images show the working of the parking section of the application. It shows the custom marker locations for both parking lots and park and ride lots. Clicking on either marker shows you the exact address of the parking lot as well as the capacity information in the message. Nothing complex here but if the API in the future is able to update the status of the parking lots (percentage filled) then it could be very useful in the busy summer months. Also, this data can benefit both Ottawa residents and travelers as any venture into the downtown area results in the search of parking lot locations nearby. Having this application handy can save a few minutes of browsing around as well as money saved in gas bill.

5. Conclusion

To evaluate myself based on my work on this project, I think I did an okay job. Having never worked with Android before and having used Java only in a few university courses, I think I coped well with the added workload. Completing almost all the proposal objectives successfully supports this claim. This is not to say that I don't regret certain parts of the project. Firstly, I should have adhered much more strictly to the deadlines and reported to the supervisor on time. Also, had I been able to dedicate more time on the project and estimate the workload correctly, I would have been able to meet the timelines and possibly worked on the optional objective I set out for myself.

Regarding the application itself, I am pleased with the final outcome as it sets out nicely and achieves what I intended it to. To be more content with my work, I would have liked to have the cycling network work flawlessly as well, but I didn't anticipate the slowdowns that I experienced in Google Maps. The optional objective that I set for myself was also interesting and I regret not having enough time to work on it. Lastly, the user interface could have been constructed more nicely to make the application more user-friendly and pleasing for the eyes.

5.1 Future Work

There are a number of ways to improve upon this application. Some of the most obvious ones are:

1. Improve the Bicycle Cycling Network in terms of both speed and usability. The initial download time cannot be improved but the actual drawing of the paths as well as the overall performance can be greatly improved.

2. Make the application bilingual (since the source APIs make this possible) and have a default language selectable by users. Since a lot of French speaking people from the neighboring Quebec province visit Ottawa, having bilingual feature could be put to good use.
3. Red Light camera detection to alert user when approaching a red light camera.
4. Improve User Interface to make it a more stylish and sleek design. The visual representation of an application has a great influence on how the application is perceived by users. So a nice design and modern graphics can be used to improve the look of the application.
5. Implement a client-side database to store the model data so that users can access data when offline (would be useful for some datasets such as parking lots, cycling network, etc.)

6. Software & Resources Utilized

Over the course of the project, there are many resources that I utilized and relied on. Some resources below, like Stack Overflow and Android Developer Guide, were accessed many times throughout the project with multiple pages accessed in total.

1. Android Developer Guides - https://developer.android.com/guide/index.html
2. Google Maps Android API - https://developers.google.com/maps/documentation/android/intro
3. Open Data Ottawa - http://data.ottawa.ca/
4. Ottawa Traffic Open Data API - http://traffic.ottawa.ca/map/opendata_info
5. Wikipedia (for Design Patterns among several other topics) - https://en.wikipedia.org/wiki/Design_Patterns
6. Stack Overflow for various Android/JAVA coding questions/reviews - http://stackoverflow.com/
7. Genymotion – https://www.genymotion.com/#!/
8. Draw.io – To make subsystem diagrams - https://www.draw.io/
9. Microsoft Word – To make project report
10. Android Studio – To make Android project

7. References

[1] OpenData Ottawa. (2015). Retrieved from OpenData Ottawa: http://data.ottawa.ca/

[2] UDOT Traffic - Android Apps on Google Play. (2015). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.transcore.android.commuterLink&hl=en

[3] MY-Traffic - Android Apps on Google Play. (2015). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.fazlimnor.mytraffic&hl=en

[4] Warsaw Bike Path - Android Apps on Google Play. (2015). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=pl.wtopolski.android.warsawbikepath&hl=en

[5] GeoJSON. (2015). Retrieved from GeoJSON: http://geojson.org/

[6] GeoTools The Open Source Java GIS Toolkit. (2015). Retrieved from GeoTools: http://geotools.org/

[7] Introduction to Android. (2015). Retrieved from Android Developers: http://developer.android.com/guide/index.html

[8] Model–view–controller. (2015). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Model-view-controller

[9] Strategy pattern. (2015). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Strategy_pattern

[10] AsynchTask. (2015). Retrieved from Android Developers: http://developer.android.com/reference/android/os/AsyncTask.html

[11] java - Is there a unique Android device ID?. (2014). Retrieved from Stack Overflow: http://stackoverflow.com/questions/2785485/is-there-a-unique-android-device-id

[12] CodeIgniter Web Framework. (2015). Retrieved from CodeIgniter: http://www.codeigniter.com/

[13] Android* - Intel® Hardware Accelerated Execution Manager. (2015). Retrieved from Intel Developer Zone: https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager

[14] Genymotion. (2015). Retrieved from Genymotion: https://www.genymotion.com/#!/