Framework Design for Map-Based Navigation in Google Android Platform

Aryo Pinandito[1,3], Agi Putra Kharisma[2,3] and Rizal Setya Perdana[1]
[1]*Information System Department, Computer Science Faculty, Universitas Brawijaya.*
[2]*Information Technology, Computer Science Faculty, Universitas Brawijaya.*
[3]*Mobile, Game, and Media (MGM) Research Group, Computer Science Faculty, Universitas Brawijaya.*
aryo@ub.ac.id

***Abstract*—This research proposes a design of mobile application framework that allow mobile application developers to develop their own outdoor mobile navigation application with the possibilities of utilizing multiple different pathfinding methods by using Abstract Factory design pattern for an optimized mobile navigation application in Google Android platform. The proposed framework has main functionality of providing navigation path from source or user location to a particular or user-specified location and then represents it visually on a digital map. Dijkstra and A-Star algorithms are implemented to show the effectiveness of the proposed mobile application framework design. An Android application prototype is constructed using the application framework and it has been successfully developed and satisfies the specified requirements. Mobile application framework design, performance comparison of pathfinding methods implementation, and a recommendation in specifying pathfinding method to use during application runtime are also provided.**

***Index Terms*—Star; Android; Dijkstra; Map; Navigation.**

I. INTRODUCTION

The abundant use of mobile devices is affected by the growing popularity of services that is offered to mobile users such as location-based mobile games, Global Positioning System (GPS)-based navigation services, transportation tracking [1], mobile travelling, and many others. Statistics shows that mobile travelling applications have significant growth between 2011 and 2015, while 85% of international travelers use mobile device while they were travelling, and they have been proven to increase the revenue of many travel industries [2], thus invites mobile application developers to build mobile applications that are intended for use on mobile devices while maintaining high level of mobile application usability.

There are many mobile applications that provide map-based public directions and navigation in the Android platform such as (Google) Maps, Polaris, Scout, Sygic, and Waze. Navigation services is often realized by utilizing the GPS sensor that is embedded in a mobile device. Google as the developer of Android platform provides their Location and Sensors Application Programming Interface (API), maps and direction service as part of Google Play Services library. The library is included in the Android Standard Development Kit (Android SDK). Several transportations and traveling applications provide maps and direction functionalities by utilizing Google Maps API and Google Maps Directions API. Hence, the development of maps-based Google Android and Location-Based Services (LBS) applications in Android platform become faster and easier.

Several research and implementation related with travelling and navigation system that is utilizing electronic or digital maps have been conducted. Tourist guide mobile application, which is built under Java 2 Mobile Edition (J2ME) has been designed and developed to support multiple mobile platforms [3]. It utilizes GPS to obtain user location information and performs client-server RESTful communication using XML data format. The application also utilizes Google Maps API to display electronic maps and application geographical data. However, mobile applications, which are developed using J2ME, were still restricted to use mobile resources and Internet connection, hence they will suffer from scalability and reliability [3].

Another similar research and application development about campus navigation system on Google Android platform has been conducted to help academicians to find locations along with the travelling route to the intended location [4][5]. The developed application allows users to find information related with events that is currently happening in the area, thus allowing users to obtain updated information that visualized on their map. However, they still have several problems on how to find an alternative route to a particular destination and how to show several places of another interesting information along the route. The geographical information, which represents a route from an origin to a destination, is obtained from external systems or API, i.e., Google Maps API and Google Geocoder API. Therefore, most Android developers, which uses Google APIs in their applications, cannot change, adjust, or modify the APIs by default.

Google also have their own native Google Maps application for the Android platform. The application allows users to find alternative route from origin to destination. However, the application has several limitations that also faced in research by [5]. The problem faced is how can people find a traveling route in a private area such as in university campus or tourist resort area. Finding the shortest path from origin to destination in these private areas cannot be performed by only relying to services provided by Google Maps application. Moreover, problems related to how people able to perform indoor navigation by using their mobile device without using GPS sensor reading also become an interesting issue. Therefore, researches related to outdoor and indoor navigation system become issues that need to be resolved.

This research tries to resolve several navigation problems in an Android based mobile device by proposing a design of Android application framework. The proposed application

framework design should allow mobile application developers to develop their own mobile outdoor navigation application with the possibilities of utilizing multiple different path finding methods and improving the computation of their methods for an optimized mobile navigation application in Android platform. This research also tries to implement two path finding methods, i.e. Dijkstra and A* algorithm, that are wrapped into navigation library component used in the proposed application framework. Several performance measurements and analysis of the implementation were performed in this research to obtain the suitability characteristics of the proposed application framework and navigation library for use in an Android mobile application.

II. Literature Review

A. Dijkstra Algorithm

Dijkstra Algorithm is one of widely used routing technique to solve single-sourced the shortest path problem between two locations, nodes, or vertices in such a road network or a mesh that represented by graph. The algorithm traces the network from one source node or vertex and spans all nodes or vertices that are reachable from source. For the algorithm to be applicable, the cost or weight for an edge between two nodes or vertices must be non-negative [6][7].

If given a graph (G) with edges (E) and vertices (V), and a specified source vertex (S), then Dijkstra algorithm could be described with the following pseudocode [8]:

```
Initialize all vertices (V) distance (d) in graph (G) to infinity
Set distance of S to 0
Create set of resolved vertices (P)
Create set of unresolved vertices (U)
Add S to U

While (U is not empty): Then
    Pick one vertex V from U with the lowest distance from S
    Remove V from U
    Get adjacent vertices (W) from vertex V
    For each adjacent vertex (v) in W: Do
      Obtain weight (w) from V to v defined by E
      If (d of v) > (d of V) + w: Then
        Set (d of v) with (d of V) + w
        Set path to v from S
      : end If
      Add v to U
    Add V to P
    : end For
: end While
```

B. A* Algorithm

A* algorithm is probably the most popular pathfinding algorithm, especially in computer games programming [9]. A* algorithm is an improvement of Dijkstra algorithm that uses heuristic technique in determining estimated distance between source and destination nodes or vertices. However, the heuristic function has to be admissible which means that the heuristic value returned is never overestimates the actual distance between source and destination nodes.

Dijkstra algorithm selects an adjacent node based on the lowest cost or distance to the adjacent nodes, A* algorithm selects an adjacent node by minimizing the total cost estimation of a path ($f(n)$) by summing the cost of source node to a node n ($g(n)$) and its heuristic value to node n that estimates the cost from node n to a destination node ($h(n)$) as in Equation (1).

$$f(n) = g(n) + h(n) \quad (1)$$

The A* algorithm stops when destination node has been reached and the computed $f(n)$ value of destination node is smaller than any other unexplored nodes in queue or until there are no unexplored nodes in queue and destination node has not been reached. This behavior become the main difference between A* and Dijkstra algorithm because Dijkstra algorithm will always continue the search until all possible nodes has been explored thus making A* algorithm better.

C. Abstract Factory Creational Pattern

Application frameworks are becoming increasingly common and important, as they make up a reusable design for a specific software purpose. The design is usually implements a particular object-oriented design pattern. There are more than 20 design patterns that categorized into Structural, Creational, and Behavioral Pattern [10] and some of these patterns apply to Android application. Creational patterns are abstracting the instantiation of processes or objects. The pattern helps making systems independent from its internal creation, composition, and representation while providing the same functionalities.

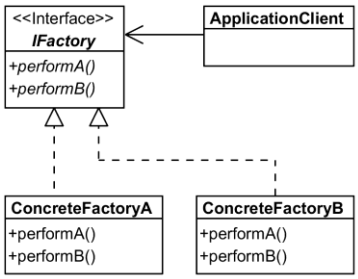

Figure 1: Basic abstract factory design pattern

One of popular Creational design pattern used to separate application with is Abstract Factory. Abstract Factory design pattern provides an interface to create groups of related or dependent objects without specifying their concrete classes [10]. Applications are allowed to specify concrete classes that provides similar functionality to perform the required functionalities during runtime. Therefore, Abstract Factory design pattern is the recommended design pattern when application developers want to decouple the main application code from the creation of instances of class and its working configuration with multiple families of classes [11]. The basic class diagram of Abstract Factory design pattern is shown in Figure 1.

III. Methodology

A. System Requirement Analysis

Android application framework, which allows developers to develop their own optimized shortest path algorithm, was developed in this research. The requirements of the

proposed framework designed in this research were composed based on several features. The main functionality that the framework should possess is, it should be able to provide navigation path from source or user location to a particular or user-specified location in such a way that the navigation path and related information should be able to be presented visually on user's mobile device digital map.

The system was analyzed by the following scenario: by using a mobile device application, a user is expected to know the shortest path to follow when they want to go to a particular location from their origin. The path to the destination will be displayed on a map on their mobile device screen. Therefore, the proposed system framework design should be able to point the nearest known location from user current location, locate the nearest known location to user-specified destination location, provide the shortest path to the specified location, and display the path on a map.

B. Framework Design

In providing the navigation path, the designed framework should provide freedom and flexibility to application developer in developing and optimizing their shortest path or navigation algorithm. Therefore, the designed framework provides a Navigation Library component that should meet the requirements. The proposed framework architectural design is depicted in Figure 2.

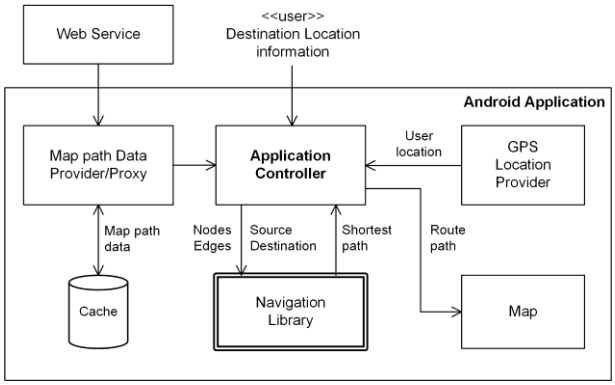

Figure 2: Proposed system architectural framework

Most of the proposed application framework's components functionalities were performed inside the Android application. The core flow of the application is controlled by an Application Controller which could be implemented in Android as an Activity. The map path information is obtained from a Map Path Data Provider component. The map path data contains geographical information related to paths and crossroads on the map. This information could be obtained from a web service, from a local cache, or both. However, the primary data source used in this research is provided by a web service using HTTP RESTful protocol in JavaScript Object Notation (JSON) format. JSON format is used in this research as it is known superior compared to XML format in terms of its processing speed, CPU utilization, and memory usage for a common usage scenario [12] and more suitable as a data-loading tool in asynchronous web-based application [13] which is considered important when it comes into mobile device computing. The Data Provider component is responsible to convert it into Nodes and Edges and store its information into cache for future use.

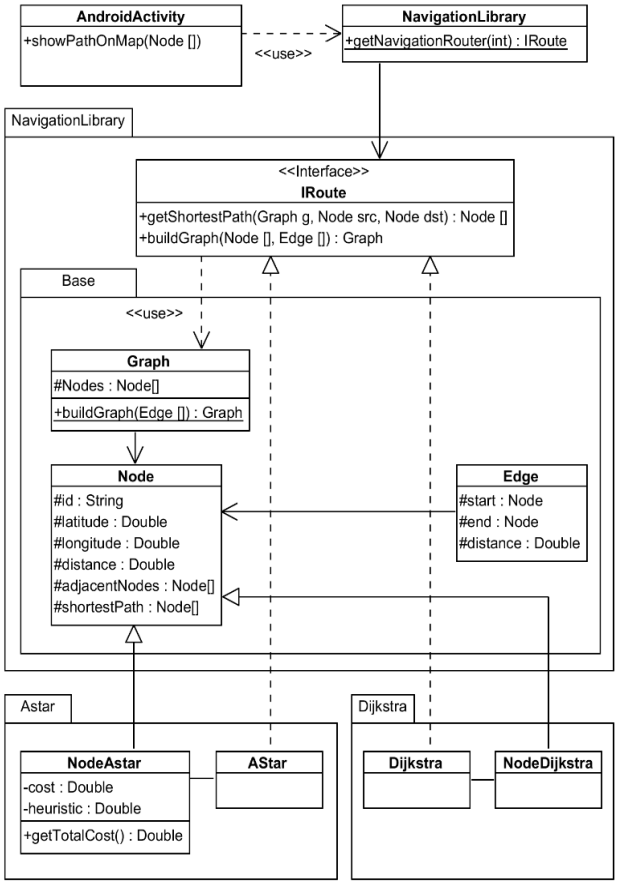

Figure 3: Navigation library class diagram

The application requires two location information as input in order to provide the route between them. The source location is predetermined by user's current geolocation where such information can be obtained from GPS sensor reading in user's mobile devices. The information is provided by application components namely GPS Location Provider. An Application Controller has responsibility to collect and control data that application user, Map Path Data Provider, and GPS Location Provider provide to perform the intended application functionalities. Thus, Application Controller communicates with Navigation Library to obtain possible shortest path to be displayed on a map.

Apart from providing navigation functionality, the proposed framework should also provide base classes that represents the problem domain for navigation algorithm implementation reuse. These base classes should represent basic abstractions of road intersection or turn and the path between two nodes. One Node class will represent each road intersection or turn and one Edge will represent the connection or path between two Nodes. It is possible that one Node may be connected to more than one other Node by several Edges to represents a crossroad or road intersection. The information contained in an Edge were used to determine the adjacent Nodes of a Node. Lastly, a set of Nodes and their respective adjacent Nodes information were wrapped as a Graph. Node, Edge, and Graph class relationship is shown in Figure 3 and for the sake of simplicity, getter and setter methods have been omitted from the diagram.

The internal design of navigation library was designed by implementing the concept of Abstract Factory design pattern as shown in Figure 3. The Abstract Factory pattern is used when a system should be configured with one of multiple

groups of objects and only reveals the interface rather than the concrete implementations [10]. In this research, the product is the implementation of shortest path algorithm inside the navigation library. Therefore, an Android application, which requires the shortest path finding functionality, does not need to know the concrete algorithm implementation of the navigation library, thus making it possible for the development, improvement, and optimization of navigation library in the future.

C. Implementation and Evaluation

One prototype of Android application that implements the designed application framework is developed. The framework and its navigation library component were implemented using Java programming language under Google Android application framework using Google Android SDK. The Android application and its navigation library were implemented in linked Android application and library projects respectively. The compiled mobile application prototype was deployed on a Google Nexus 6 Android device that runs an Android Operating System (OS) with API level 24 or Android version 7.0 Nougat. However, the prototype was developed using AppCompat Support Library from Google Android SDK. The minimum required level for the application prototype to run was set to API level 15 or Android version 4.0.3 Ice Cream Sandwich. Therefore, it should be able to run correctly on devices with an Android 4.0.3 Ice Cream Sandwich onwards.

A web service has been developed to provide and manage geolocation Nodes and Edge information from a private DBMS that store the information. The web service was built under Linux, Apache, MySQL, and PHP (LAMP) stack so that the developed mobile application could act as RESTful client.

In order to evaluate the effectiveness of the designed framework, this research implements two shortest path algorithms, i.e., Dijkstra and A* as such that one of these algorithms can be used to solve the navigation route discovery problem between two specified locations. Both algorithms were evaluated using a same data set from the same data source. The implementation of A* algorithm used in this research modifies the implementation of Dijkstra algorithm in terms of distance calculation of a node and in determining node to expand based on the heuristic value of a node.

In determining heuristic value of distance between two nodes in the implementation of A* algorithm, a Euclidean distance formula is used. However, a better accuracy in distance calculation between two geolocations over the Earth surface could be obtained using Haversine formula [14].

Geolocation data of street intersections, turns, and corners were obtained by performing data collection of the road network inside the main campus area of Universitas Brawijaya. The data then be verified by rendering them to a digital map such as Google Maps or Mapbox Map and stored in a MySQL DBMS afterwards.

In order to visualize the intended output, the Android application prototype employ Mapbox Mobile SDK for Google Android platform instead of Google Maps API. Mapbox Mobile SDK has ambient caching feature and allows application to pre-fetch the maps from Mapbox Server in advance, hence opening the possibility to render maps and its associated resources when the device lacks of network connectivity [15]. The SDK is used to render digital maps, draw point of interest markers overlay, and, lastly visualize the returned path from application Navigation Library on mobile device's screen.

Several performance measurement tests and fitness analysis were performed to the developed mobile application prototype and its Navigation Library implementation to obtain empirical results. The test was performed on a road network that consists of 134 nodes and 154 edges.

IV. Results and Discussion

Based on the functionality tests, the proposed framework satisfies the requirements. The developed Android application prototype able to obtain road interconnection information in JSON format from a web service using RESTful protocol and cache it to the application local storage. As shown in Figure 4, the application able to obtain user geolocation and display it as a marker on a map. It also shows several points of interest nearby to be selected as destination. Once a destination marker has been selected, the shortest path could be obtained from the implementation of navigation library and display it on the map.

Based on the performance test performed to the Android application prototype, it is known that the average time required for the Dijkstra algorithm to complete is 1.22 milliseconds while in most scenarios, A* algorithm perform significantly better. A* algorithm requires 0.1 milliseconds in average to find the shortest path for a nearby node (C to NC), 0.21 milliseconds in average for a medium range node in the center area from the side of the area (C to Ce), and 1.16 milliseconds in average when the source and destination nodes both are in the opposite sides (C to FC).

As shown in Figure 5, the algorithm execution time of A* algorithm perform better than Dijkstra algorithm in finding the shortest path. In average, A* algorithm perform more than 60% faster than Dijkstra algorithm in all scenarios. However, A* algorithm has an overhead processing time when it calculates the heuristic values of all nodes to a particular destination node. The average processing time of heuristic estimation process for 134 nodes in this research test scenarios is 0.26 milliseconds. This processing time should be added to the total processing time of A* algorithm.

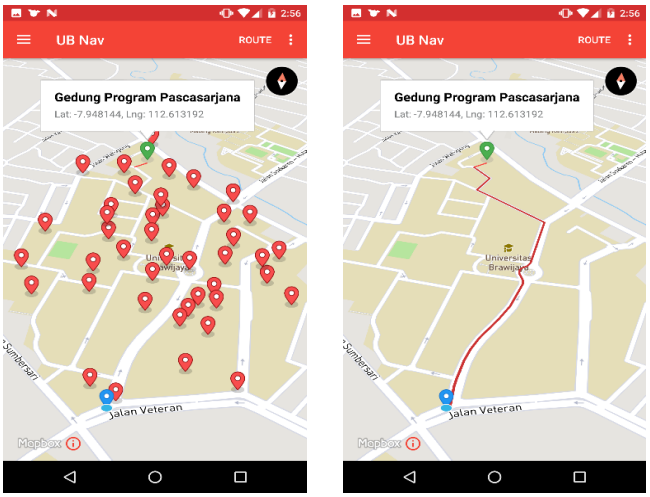

Figure 4: Android application prototype showing several point of interests (left) and showing the route to the user-specified location (right)

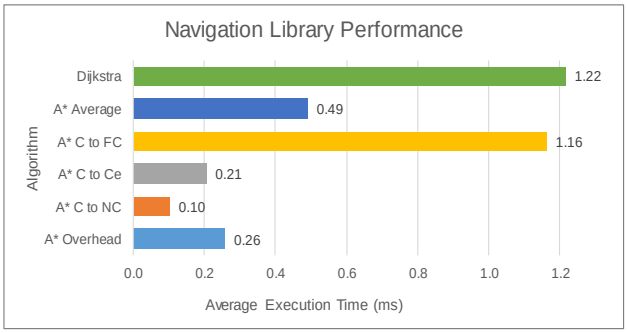

Figure 5: Shortest path algorithm average execution time (lower is better)

The worst scenario case for A* algorithm is, when it has to traverse to almost all possible nodes as much as Dijkstra algorithm does. This is most likely happened when the source node is on the opposite side of the destination node. Even though in all test scenarios the A* algorithm perform better, the Dijkstra algorithm would perform better than A* algorithm in a scenario that the application needs to find several shortest paths from a single source node. This is due to the nature of Dijkstra algorithm that caches all the shortest path to every node which has been explored.

Based on this research test case scenarios, both Dijkstra and A* algorithm perform the routing calculation in milliseconds. Therefore, both algorithm was applicable in mobile devices and would not tend to be sluggish. However, if the desired destination of user could be predefined or predicted, the application could perform a pre-caching calculation using Dijkstra algorithm for a better and faster user experience.

As future works, this research area could be extended to the following topics:

a. Implementing another shortest path or navigation method while keep optimizing the implementation of application navigation library for mobile devices;
b. Conducting usability analysis to improve application User Interface or User Experience (UI/UX) usage scenarios. The analysis and refinement could be performed by several usability improvement methods as the development of mobile application UI/UX with limited user-interaction access become important issues;
c. Integrating the application framework and the implemented navigation library in other Android or other mobile application platforms that requires similar functionalities or features;
d. Extending the application prototype to support additional functionalities to provide better usability;
e. Extending the concepts introduced in this research, which are related with the application framework, data structure, and application design patterns, to help mobile application users perform turn-by-turn indoor navigation.

V. Conclusion

In this research, the Abstract Factory design pattern allow the design and implementation of navigation application to utilize different type of pathfinding algorithms, i.e., Dijkstra algorithm and A* algorithm. Both Dijkstra and A* algorithms are able to be implemented natively in Android platform.

An Android application prototype, which implements the Abstract Factory design pattern as designed in the proposed framework, has been successfully developed and meets the specified requirements. The developed application prototype is able to obtain road interconnection geographical information from a RESTful web service using JSON data format and locally caches the information. User's geographical location information could be obtained from user's device GPS sensor to automatically determine the nearest node as starting point of the navigation path. Lastly, the Android application prototype that employ Mapbox Android SDK is able to visually display maps, point of interest markers, and display computed navigation path from and to a particular location.

This research also shows that A* algorithm perform significantly better than Dijkstra algorithm in most pathfinding scenarios for up to 71% faster and 60% in average. However, when it comes to a scenario when several shortest paths to several particular nodes needs to be found from a single source node at once, Dijkstra algorithm would probably perform better than A* algorithm as it caches the discovered shortest path information of every node as it travels.

By conducting the implementation and test of the proposed framework to the Android application prototype, this also proves that the proposed application framework design allows an Android application to dynamically specify and use an implementation of shortest path algorithm during runtime.

Acknowledgment

This research is partially funded by DIPA of Computer Science Faculty, Universitas Brawijaya. Part of this research is supported by Mobile, Game, and Media (MGM) Research Group and Mobile Application Development Laboratory. We are thankful to all of our friends and colleagues at Unit TIK Universitas Brawijaya, Information System, Computer Science, and Information Technology Department of Computer Science Faculty, Universitas Brawijaya who contribute their expertise in assisting this research.

References

[1] Khalid, S. K. A., Salleh, N. S. M., & Samsudin, N. A. (2016). "A Bus Tracking Information System using Consumer Grade GPS: A Case Study." Journal of Telecommunication, Electronic and Computer Engineering, 8 (4), 47–51.
[2] Frederic Gonzalo. April 12, 2016. "16 Stats About Mobile Travel in 2016", in Frederic Gonzalo [Online]. http://fredericgonzalo.com/en/2016/04/12/16-stats-about-mobile-travel-in-2016/
[3] Alshattnawi. Sawsan, (2013) "Building Mobile Tourist Guide Applications using Different Development Mobile Platforms," in International Journal of Advanced Science and Technology vol. 54, pp. 13–22
[4] Anpat, V., "Campus Navigation on Android Platform," Int. J. Sci. Technol. Eng., vol. 2, no. 10, pp. 452–458, 2016.
[5] Jana, S. and Chattopadhyay, M., "An event-driven university campus navigation system on android platform," Proc. - Int. Conf. 2015 Appl. Innov. Mob. Comput. AIMoC 2015, pp. 182–187, 2015.
[6] X. Zhang, Y. Zhang, Y. Hu, Y. Deng, and S. Mahadevan, "An adaptive amoeba algorithm for constrained shortest paths," Expert Syst. Appl., vol. 40, no. 18, pp. 7607–7616, 2013.
[7] K. Rohila, P. Gouthami, and P. M, "Dijkstra's Shortest Path Algorithm," Int. J. Innov. Res. Comput. Commun. Eng., vol. 2, no. 10, pp. 6139–6144, 2014.
[8] Baeldung. "Dijkstra Algorithm in Java, " in Baeldung, 2017, [Online]: http://www.baeldung.com/java-dijkstra

[9] Cui, X., & Shi, H. (2011). "A*-based Pathfinding in Modern Computer Games." IJCSNS International Journal of Computer Science and Network Security, 11(1), pp. 125–130.
[10] Ruchi Mittal, Ipsita Bhattacharya, M.P.S. Bhatia. "Innovative Framework for Data Structure Using Design Pattern," in Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011), Volume 2, pp. 197-204, 2012
[11] Joydip Kanjilal, "The factory method and abstract factory design patterns: managing object creation efficiently" in InfoWorld, Jun 5, 2015 [Online] http://www.infoworld.com/article/2931441/c-sharp/the-factory-method-and-abstract-factory-design-patterns-managing-object-creation-efficiently.html
[12] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, Clemente Izurieta. "Comparison of JSON and XML Data Interchange Formats: A Case Study," in ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE (2009), pp. 157-162
[13] Lin, B., Chen, Y., Chen, X., & Yu, Y. (2012). "Comparison between JSON and XML in Applications Based on AJAX," in 2012 International Conference on Computer Science and Service System, CSSS 2012, pp. 1174–1177.
[14] Chris Veness. "Calculate distance, bearing and more between Latitude/Longitude points, " in Movable Type Scripts, 2017 [Online]: http://www.movable-type.co.uk/scripts/latlong.html
[15] Mapbox. " Offline maps with Mapbox Mobile, " in Mapbox Help, 2017 [Online] https://www.mapbox.com/help/mobile-offline/