Design and Implementation of Real Time Bi-directional Traffic Management Support System with GPS and WebSocket

A project submitted

by

MD. Rahatur Rahman (ID: 2008-2-96-005)

Under the supervision of

Dr. Md. Shamim Akhter

Department of Computer Science & Engineering
Faculty of Science & Engineering
East West University

May 2015

Design and Implementation of Real Time Bi-directional Traffic Management Support System with GPS and WebSocket

A project report submitted to the Computer Science & Engineering department of the science faculty, East West University in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering.

MD. Rahatur Rahman (ID: 2008-2-96-005)

Department of Computer Science & Engineering
Faculty of Science & Engineering
East West University - Bangladesh

May 2015

Declaration

This is to certify that this project is my original work. No part of this work has been submitted elsewhere partially or fully for the ward of any other degree or diploma program. Any material reproduce in this project has been properly acknowledged.

Students' names & signatures:

1. MD. Rahatur Rahman (ID: 2008-2-96-005) __________________

Approval

The project report titled “Design and implementation of Real Time Bi-directional Traffic Management Support System with GPS and WebSocket” has been submitted to the following respected members of the Board of Examiners of the faculty of Science in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering on 10th May 2015 by the following student and has been accepted as satisfactory.

1. MD. Rahatur Rahman (2008-2-96-005)

Dr. Md. Shamim Akhter
Supervisor & Assistant Professor
Department of Computer Science &
Engineering, EWU

Dr. Shamim Hasnat Ripon
Chairperson & Associate Professor
Department of Computer Science &
Engineering, EWU

Acknowledgements

First of all I express our gratefulness to the Almighty, without His divine blessing it would not be possible for me to complete this project successfully. I sincerely like to thank my honorable project supervisor Dr. Md. Shamim Akhter sir, Assistant Professor, Department of Computer Science & Engineering, East West University (EWU) for his great inspiration and proper guidance throughout the whole project work. I am also grateful to all of my respected teachers without whom I would never reach at this educational stage. I am thankful for their time and ideas and appreciate their availability whenever I turned to them.

Contents

Index of Figures and Tables

Abstract

In this paper, I present the principles of a low operational-cost but flexible Internet-based traffic management support system. The main core of the system is an extensible web server created by Microsoft called Internet Information Services (IIS, formerly Internet Information Server). The server application runes on IIS.

The Traffic Management Support System is a smart system to assist the personnel involved in the management of the traffic system. It provides vital information to the management to make better decisions. As the traffics are growing rapidly (day by day) but the road systems/resources are not increasing in the same way to meet necessary demands, so it is important to manage the traffic system properly, with better utilization of the existing roads and resources.

In the past various systems were proposed and implemented to overcome the problems stated above. Most of the systems were implemented with machine vision (image processing), RFID gateways, areal surveillance, and/or remote sensors technologies. But the major drawback of the above mentioned systems are their installation difficulties, maintenance over the time, error handling, coverage and cost effectiveness. Nevertheless, they consist uni-cast/ one way communication (client to server only).

The demand for a feasible solution to the above mentioned problems is very high for a metro city- like Dhaka. Losing valuable time (approx 200%) being stuck in traffic jams and increases CO2 emissions (approx 300%) due to the present traffic situation. As a result, an up-to-date technological based traffic management support system/tool has become a need for such metro cities.

Thus, our proposed system will bring the real time bidirectional communication between the clients and the servers. Accurate and real-time traffic situation can be plotted at any given time. In addition, it would make better utilization of the existing technologies to achieve these goals. In addition, users will be able to get help/advice from the system, find nearby places like hospitals, police stations, vehicle service stations. Users can also find the optimized and cost effective way to reach from source to destination or vice versa.

Chapter 1 - Introduction

1.1 Introduction

In this work, I propose to use the WebSocket over http technology with real time bi-directional communication to solve the problems discussed above. The main reason behind the selection of the technology is low cost implementation, flexibility, maintainability and infrastructure security.

In this section I present a blend of all above features and technologies to introduce a generic architecture for the proposed real time system.

The vehicles referred as clients has access to GPS which allows them to collect the present location in terms of Latitude and Longitude.

The clients are also equipped with any device capable to handle the web request/response over the Http. The device should also have support for the IETF as RFC 6455 standards. The architecture comprises a WebSockets server that listens for incoming connections and the client browsers that should be compatible with HTML5 and WebSockets.

The clients will be able to get the services from the server(s). Also they will have to relay their present location through a WebSocket server.

At first an Html5 client requests a web-page from a web server that includes the required javascript for establishment of the WebSocket connections (or any fallback connection if required). Then, the client connects to the WebSockets server using a WebSocket connection and starts the process of sending the location updates in predefined intervals.

The communication server(s) will be responsible to handle the incoming client connections and the database server manages the client data as well as the data of the traffic system.

The routes are calculated in the database server in predefined intervals and are served to any requesting client.

The HTTP signaling between clients and web servers assumes access to the appropriate web page by each client.

Figure bellow illustrates the components of our proposed architecture along with the involved signaling.

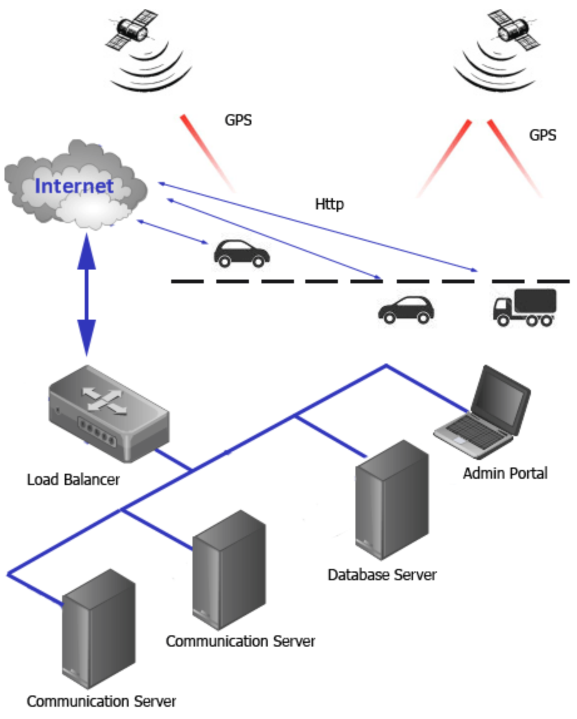


Figure 1: System Architecture

Apart from the location updates the clients can also request for exposed services like traffic route suggestions, find nearby places, view traffic alerts related to the road and the transportation industry etc. by the servers.

The server application will generate most relevant data based on the facts available at time of the request and make decisions for the client. The server will relay the decision/suggestion to the clients whenever the data is processed and ready.

1.2 Background of the project

Over the time different solutions have been proposed to solve the traffic jam problem. The solutions vary in their core technology; as some of them use Infrared sensors, CCTV cameras and image processing, GSM and cellular towers, RFID gates, Sound sensors etc.

However, many of those solutions are either very expensive or difficult to install and maintain over the time.

As a result I tried to develop a system which will allow us to achieve the same goals without suffering from the shortcomings mention above.

1.3 Objective

Specific objectives of this project includes:

- Study and analysis of the existing traffic monitoring systems and their shortcomings.
- Design and implementation of client applications which will allow users to consume services.
- Design and Implementation of an administrative dashboard for real-time road traffics monitoring.

The benefits of the proposed system is flexibility, cost effective and low infrastructure maintenance and installation costs. From this perspective, the primary benefit is the ability to use the same mobile phone that is used to make phone calls, can be used as a mobile tool to consume different road traffic related services. The system will also allow users to get direction to a destination, find nearby places, view and send traffic alerts. The system will be low on costs. So in quest for developing such an application it has been decided to design and implement a WebSocket based real time communication system.

1.4 Organization of the project

As I try to develop a WebSocket based real time communication system, first of all I will do some cost-effective comparison with the existing parallel technologies which can deliver the same type of outputs. Then I will try to explain the WebSocket technology and how it works. I shall describe the communication technology used to monitor real time traffics. Eventually protocols and the interfacing will be discussed too. Next comes the security discussion where the security of the system will be discussed following by source code listing.

- <u>Chapter 2 – Analysis</u>

  This chapter will provide an overview of the WebSocket technology and its various technical aspects such as handshaking and bandwidth consumptions and also an overview of the application designs.
- <u>Chapter 3 – Implementation</u>

  The third chapter will provide an eagle's eye view of the entire system using UML diagrams. I have provided separate class diagrams and Use case diagrams of the application.
- <u>Chapter 4 – Test Input Output</u>

  This chapter consists of the results of the tests I performed on our system in various angles and I have demonstrated that our system can handle (gracefully) all sorts of inputs.
- <u>Chapter 5 – Conclusion</u>

  I conclude my project report by acknowledging the things I learned while developing the system and the future improvements that can be made to the system.

Chapter 2 - Analysis

2.1 WebSocket and the http protocol

So far, real time media communication between various client devices, either one way (server to client streaming) or two (conference), was, more or less, a static and monolithic operation served by customized platform-specific solutions.

Current attempts to provide real-time web applications largely revolve around polling and other server-side push technologies, the most notable of which is Comet, which delays the completion of an HTTP response to deliver messages to the client. Comet-based push is generally implemented in JavaScript and uses connection strategies such as long-polling. [5]

2.1.1 Polling

With polling, the browser sends HTTP requests at regular intervals and immediately receives a response. This technique was the first attempt for the browser to deliver real-time information.

Obviously, this is a good solution if the exact interval of message delivery is known, because you can synchronize the client request to occur only when information is available on the server. However, real-time data is often not that predictable, making unnecessary requests inevitable and as a result, many connections are opened and closed needlessly in low-message-rate situations. [5].

Figure bellow demonstrates a polling scenario between a client and a server.

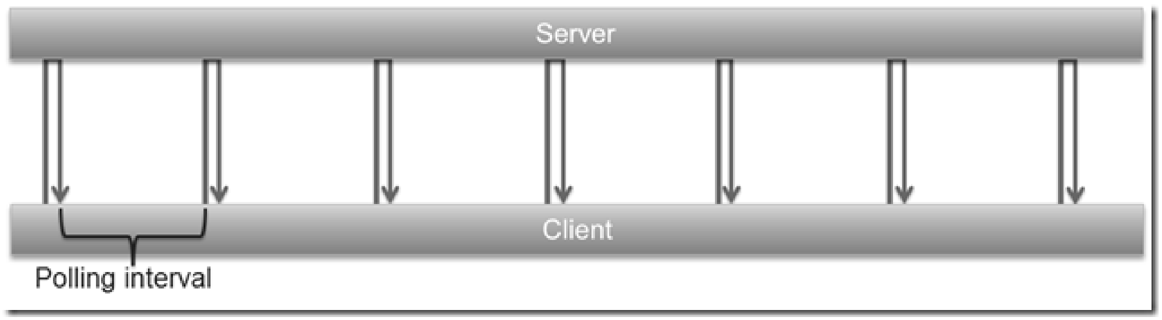


Figure 2: Polling [10]

2.1.2 Long Polling

With long-polling, the browser sends a request to the server and the server keeps the request open for a set period. If a notification is received within that period, a response containing the message is sent to the client. If a notification is not received within the set time period, the server sends a response to terminate the open request. It is important to understand, however, that when you have a high message volume, long-polling does not provide any substantial performance improvements over traditional polling. In fact, it could be worse, because the long-polling might spin out of control into an un-throttled, continuous loop of immediate polls. [5]

Figure bellow demonstrates a long polling scenario between a client and a server.

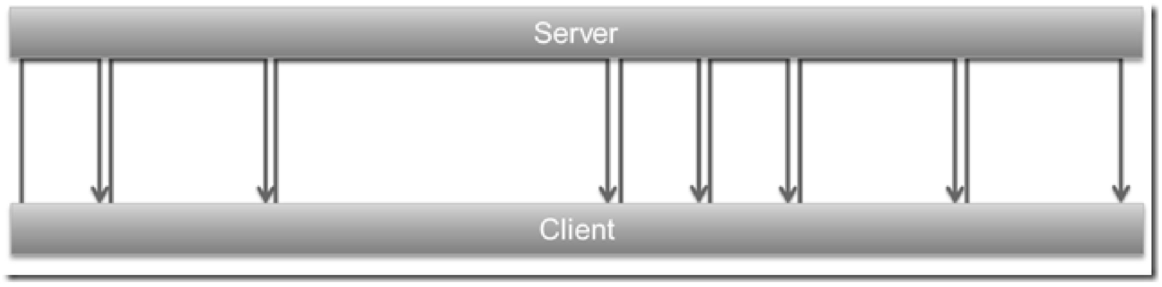


Figure 3: Long-polling [10]

2.1.3 Server-Sent Events

SSEs are sent over traditional HTTP. That means they do not require a special protocol or server implementation to get working. SSE connections can only push data to the browser. Which is why this technology is not suitable for my application.

Figure bellow demonstrates a Server-Sent Event scenario between a client and a server.

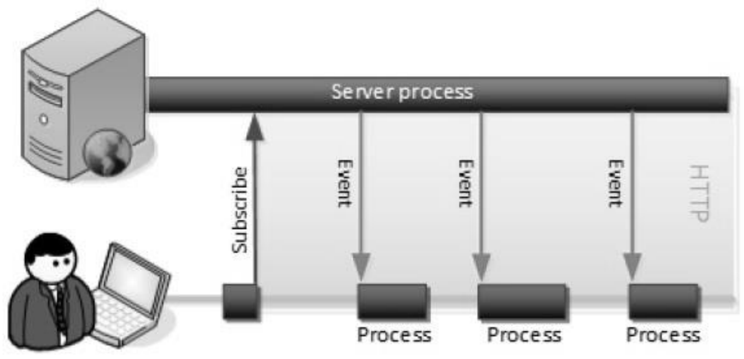


Figure 4: Server-Sent Event

2.1.4 WebSocket

Ultimately, all of these methods for providing real-time data involve HTTP request and response headers, which contain lots of additional, unnecessary header data and introduce latency. On top of that, full-duplex connectivity requires more than just the downstream connection from server to client. In an effort to simulate full-duplex communication over half-duplex HTTP, many of today's solutions use two connections: one for the downstream and one for the upstream. The maintenance and coordination of these two connections introduces significant overhead in terms of resource consumption and adds lots of complexity. [5]

The WebSocket specification - developed as part of the HTML5 initiative - introduced the WebSocket JavaScript interface, which defines a full-duplex single socket connection over which messages can be sent between client and server. The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management.
WebSocket represents the next evolutionary step in web communication compared to Comet and Ajax. [4]

WebSocket defines a full-duplex communication channel that operates through a single socket over the Web. HTML5 Web Sockets is not just another incremental enhancement to conventional HTTP communications; it represents a colossal advance, especially for real-time, event-driven web applications. [4]

It provides dynamic improvements over conventional full-duplex connection over http.

The main difference in WebSockets – compared to the usual network traffic over HTTP – is that the WebSocket protocol does not follow the traditional request-response convention. Once a client and a server have opened a WebSocket connection, both endpoints may asynchronously send data to each other. The connection remains open and active as long as either the client or the server closes the connection. [7]

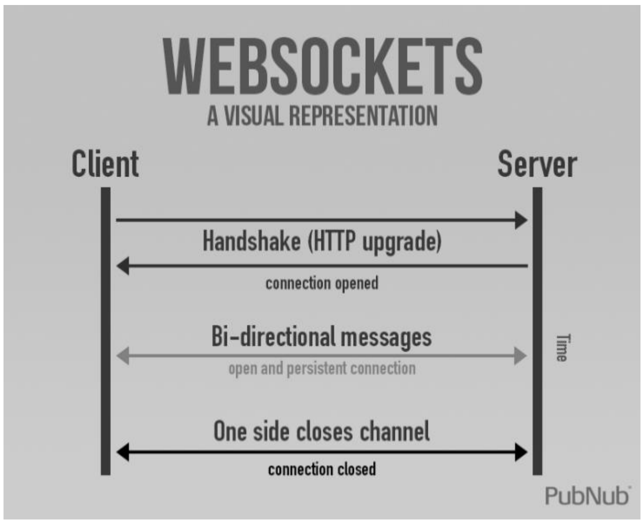


Figure 5: WebSocket connection [8]

2.1.5 Browser supports for WebSocket

Table 1: Browser Support for WebSocket

IE	Firefox	Chrome	Safari	Opera	iOS Safari*	Opera Mini*	Android Browser*	Chrome for Android
11	37	42	8	27	8.3	8	40	42

*The Safari browser for iOS is tied to the operating system, so the numbers used are based on the OS version.
*In most cases Opera Mini processing is done via Opera servers, which often prevents JS from working correctly.
As of version 8, the default mode of Opera Mini on iOS uses the iOS Safari engine, though Mini mode can be enabled.
*Android browser/WebView version numbers through 4.4 refer to the version of Android OS. Support listed is for the Android core; it should be noted that many hardware vendors (Samsung, HTC, etc.) use altered version of their default browser which may include more/less/buggy support.
Starting in Android 5, the web engine can be updated separately, so the latest Chromium version number is used instead.[17]

2.2 Performance

2.2.1 Payload

If overhead of network is considered the total HTTP request and response header information calls for overhead. The size of header may vary with respect to application here for example let us consider that header contains 871 bytes and that does not even include any data. The size of header may increase up to 2000bytes in some cases. For analysis let us consider a polling application is deployed for large number of users. Then network throughput for just HTTP request and response header data for two different set of users becomes as follows:

- Use case A: 1,000 clients polling every second: Network throughput is (871 x 1,000) = 871,000 bytes = 6,968,000 bits per second (6.6 Mbps)
- Use case B: 10,000 clients polling every second: Network throughput is (871 x 10,000) = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)
- Use case C: 100,000 clients polling every 1 second: Network throughput is (871 x 100,000) = 87,100,000 bytes = 696,800,000 bits per second (665 Mbps)

That's an enormous amount of unnecessary network throughput! If only we could just get the essential data over the wire. [5]

Now if this application is rebuild with HTML5 WebSocket there is tremendous amount of reduction in the network throughput due to unnecessary data. Each of these messages is a WebSocket frame that has just two bytes of overhead (instead of 871).
Now let us consider above example with the WebSocket based application in this case the size of frame header is only 2 bytes. Let us analyze the effect:

- Use case A: 1,000 clients receive 1 message per second: Network throughput is (2 x 1,000) = 2,000 bytes = 16,000 bits per second (0.015 Mbps)
- Use case B: 10,000 clients receive 1 message per second: Network throughput is (2 x 10,000) = 20,000 bytes = 160,000 bits per second (0.153 Mbps)
- Use case C: 100,000 clients receive 1 message per second: Network throughput is (2 x 100,000) = 200,000 bytes = 1,600,000 bits per second (1.526 Mbps)

As you can see in the following figure, HTML5 Web Sockets provide a dramatic reduction of unnecessary network traffic compared to the polling solution. [5]

Following figure shows the comparison of the unnecessary network throughput overhead between the polling and the WebSocket applications:

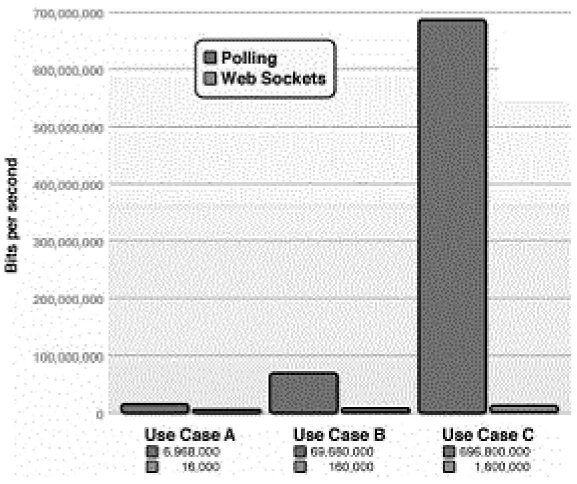


Figure 6: Polling vs WebSocket overhead [5]

2.2.2 Latency

What about the reduction in latency? Take a look at the following figure. In the top half, you can see the latency of the half-duplex polling solution. If we assume, for this example, that it takes 50 milliseconds for a message to travel from the server to the browser, then the polling application introduces a lot of extra latency, because a new request has to be sent to the server when the response is complete. This new request takes another 50ms and during this time the server cannot send any messages to the browser, resulting in additional server memory consumption. [5]

In the bottom half of the figure, you see the reduction in latency provided by the WebSocket solution. Once the connection is upgraded to WebSocket, messages can flow from the server to the browser the moment they arrive. It still takes 50 ms for messages to travel from the server to the browser, but the WebSocket connection remains open so there is no need to send another request to the server. [5]

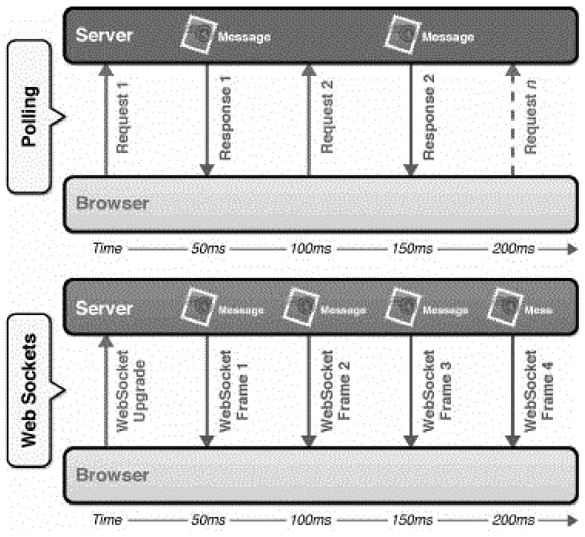


Figure 7: Polling vs Websocket latency [5]

2.3 WebSocket based System Design Overview

The WebSockets technology provides a bidirectional communication channel using a single TCP connection.

It is designed to be implemented in applications such as web browsers. Its API is being standardized by the W3C. The connections are established over the regular TCP port 80, which ensures that the system can run behind firewalls. The life-cycle of a WebSocket session is depicted in Figure 3.

At first the client that supports the WebSockets protocol, requests to establish a WebSocket connection. The positive response from the server denotes the start of such a WebSocket connection. The connection remains open for the whole session, until any endpoint requests its release with the specified procedure. As a WebSocketremains active; WebSocket frames can be transferred from server to client and vice versa with no preceding request. In our implementation the WebSockets server also hosts the service logic part of our web-applications, which is responsible for maintaining a listing of the client peers with active WebSockets and session management.

The server application has real time data to plot the situation of a region based on the data available. The server application is also responsible for data mining and business logic execution based on the data collected from various clients.

2.3.1 Hardware Requirements

- The server used for this implementation is an x86 based web server running Windows machine running IIS 8.0 or above with a minimum of 2GB memory.
- For client application, any mobile unit supporting the Html5 specification and GPS capability.

2.3.2 Software Requirements

We can use MySQL 5 or MS SQL Server 2012 Express or above for storing the client data.
The server application will be a web application built on top of open source Microsoft Asp.net MVC frameworks.

For the WebSocket implementation we can reuse the open source implementation of SignalR for Asp.net MVC framework.[5]

2.3.3 Description of the system components

2.3.3.1 Google Map

Google Maps is a desktop and mobile web mapping service application and technology provided by Google, offering satellite imagery, street maps, and Street View perspectives.[13]

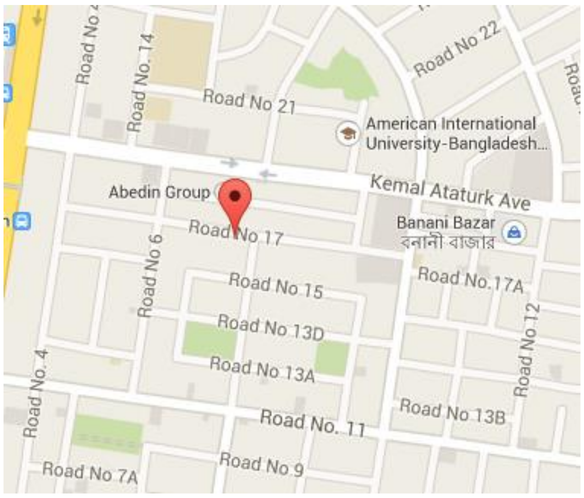


Figure 8: Google map

2.3.3.2 API

After the success of reverse-engineered mashups such as chicagocrime.org and housingmaps.com, Google launched the Google Maps API in June 2005 to allow developers to integrate Google Maps into their websites. It is a free service, and currently does not contain ads, but Google states in their terms of use that they reserve the right to display ads in the future. [13]

2.3.3.3 Licensing

Most websites and applications may use the Google Maps API free of charge. However, if you consistently generate a high amount of traffic, usage limits apply and you will need to pay for extra usage. If your site or application generates 25 000 map loads or more each day, for more than 90 consecutive days, we'll get in touch with you to talk about payment. Don't worry, if you go over the limits, we won't immediately shut off your API access or display error messages on your site. [14] [15]

2.3.3.4 GPS

The Global Positioning System (GPS) is a space-based satellite navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites. [11]

2.3.3.5 How GPS Works

GPS satellites circle the earth twice a day in a very precise orbit and transmit signal information to earth. GPS receivers take this information and use triangulation to calculate the user's exact location. Essentially, the GPS receiver compares the time a signal was transmitted by a satellite with the time it was received. The time difference tells the GPS receiver how far away the satellite is. Now, with distance measurements from a few more satellites, the receiver can determine the user's position. [12]
Figure bellow shows the satellites in orbits.

Figure 9: Satellites in orbits [12]

2.3.3.6 GPS Accuracy

Today's GPS receivers are extremely accurate, thanks to their parallel multi-channel design. [12]
GPS time is theoretically accurate to about 14 nanoseconds. However, most receivers lose accuracy in the interpretation of the signals and are only accurate to 100 nanoseconds. [11]

2.3.4 Fallback Mechanism

We know that in a real world scenario there will always be some system which will lack the latest technology. i.e. in this case our focused WebSocket may not be supported by a group of devices. As a result we always have to have some fallback mechanism so that our services are not interrupted.

Figure bellow will give an idea of the stack of the most preferred to the list preferred mechanisms.

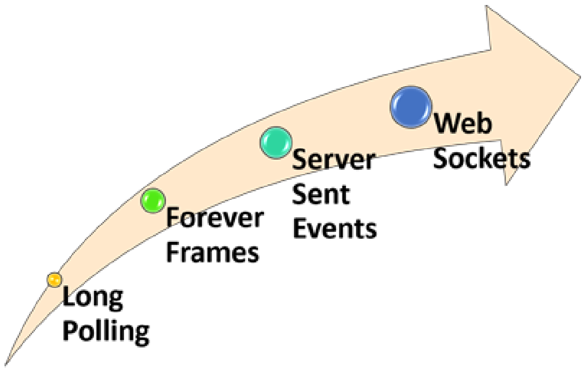


Figure 10: Preferred fallback stack [16]

2.3.5 User Authentication

To identify a client the server will maintain a challenge request/response which each of the clients will have to comply to consume the services. The server will maintain the sessions for each of the clients.

Chapter 3 - Implementation

3.1 Class Diagrams

Classes are typically modeled as rectangles with three sections: the top section for the name of the class, the middle section for the attributes of the class, and the bottom section for the methods of the class.

The class diagrams were generated using Visual Studio Ultimate 2013.

3.1.1 Data Models

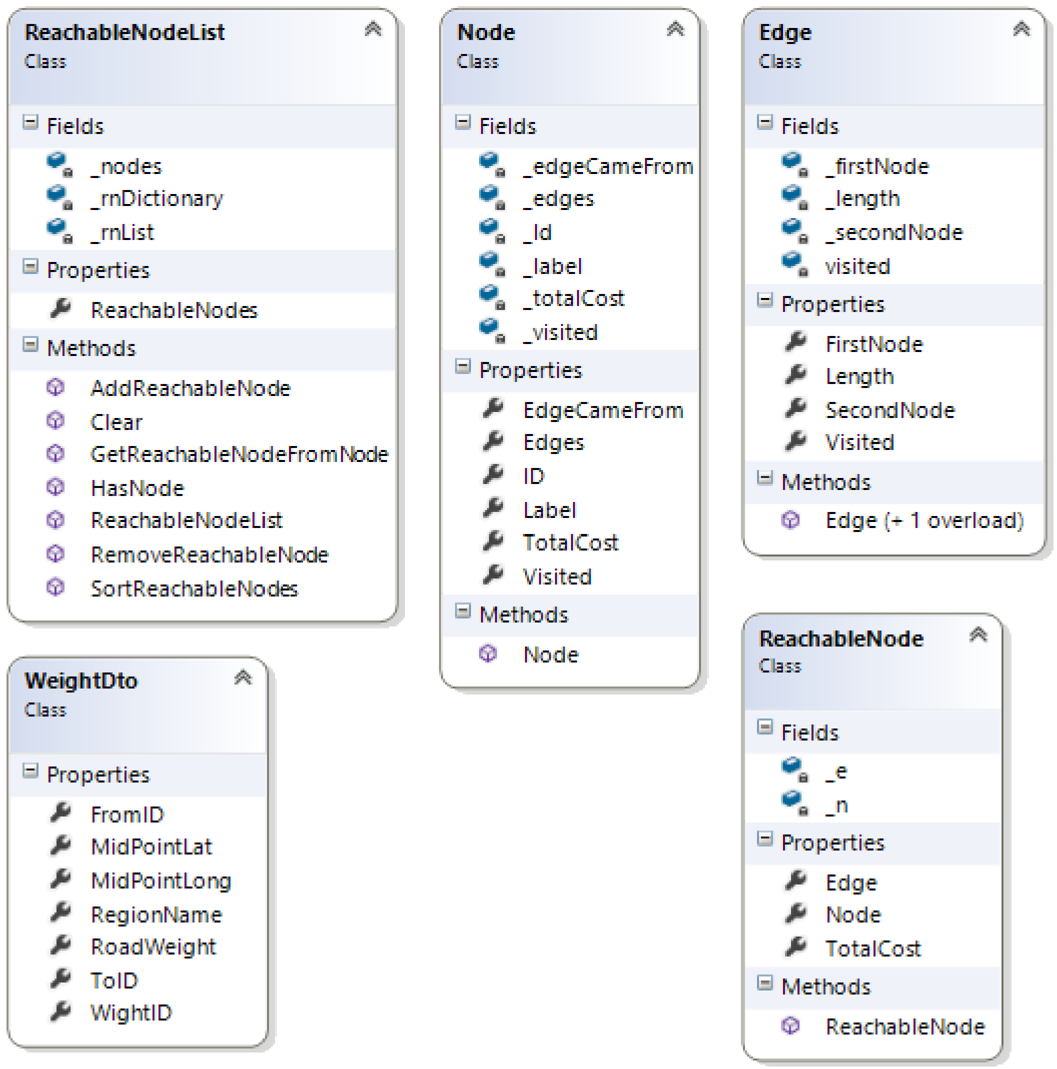


Figure 11: Graph Model

The diagram above shows the fields, properties and methods for the models used to represent the nodes of the graph and the weight table data.

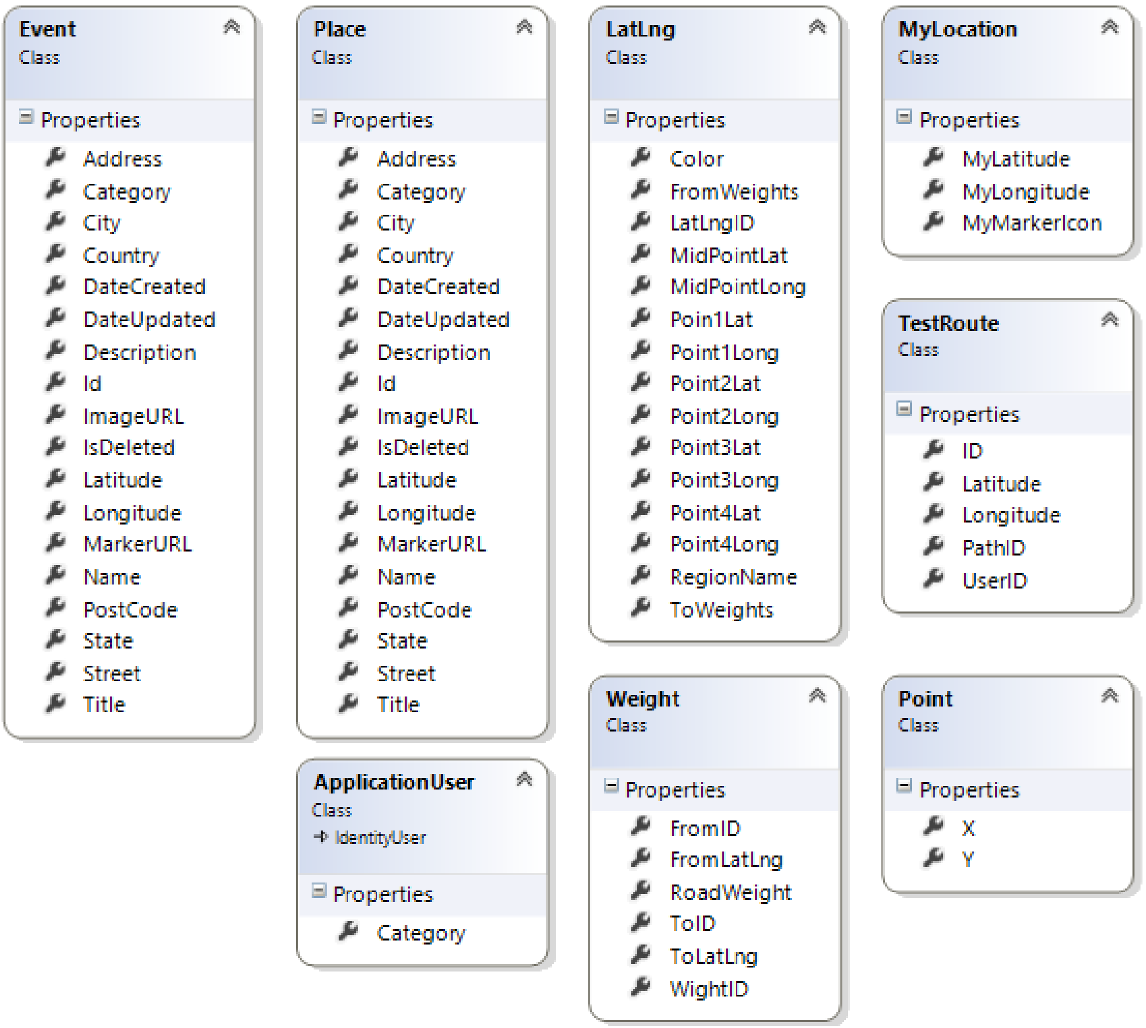


Figure 12: Data Models

The diagram above shows the fields, properties and methods for the models used to represent the weight table, Alert, Place, a Marker on the map, a Test route data.

3.1.2 Application Services

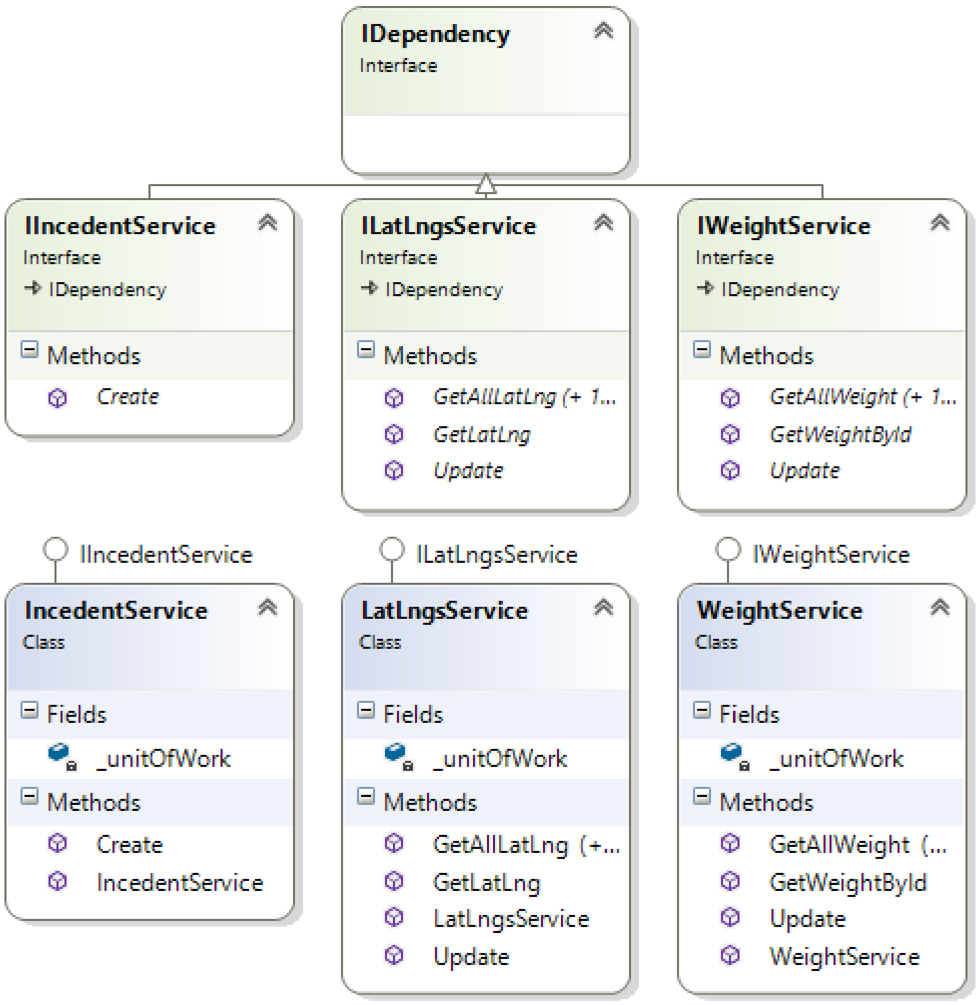


Figure 13: Services

The diagram above shows the fields, properties and methods for the services of Weight, Location and Alert. It also shows the interfaces of each of the classes.

3.1.3 Application Data Access

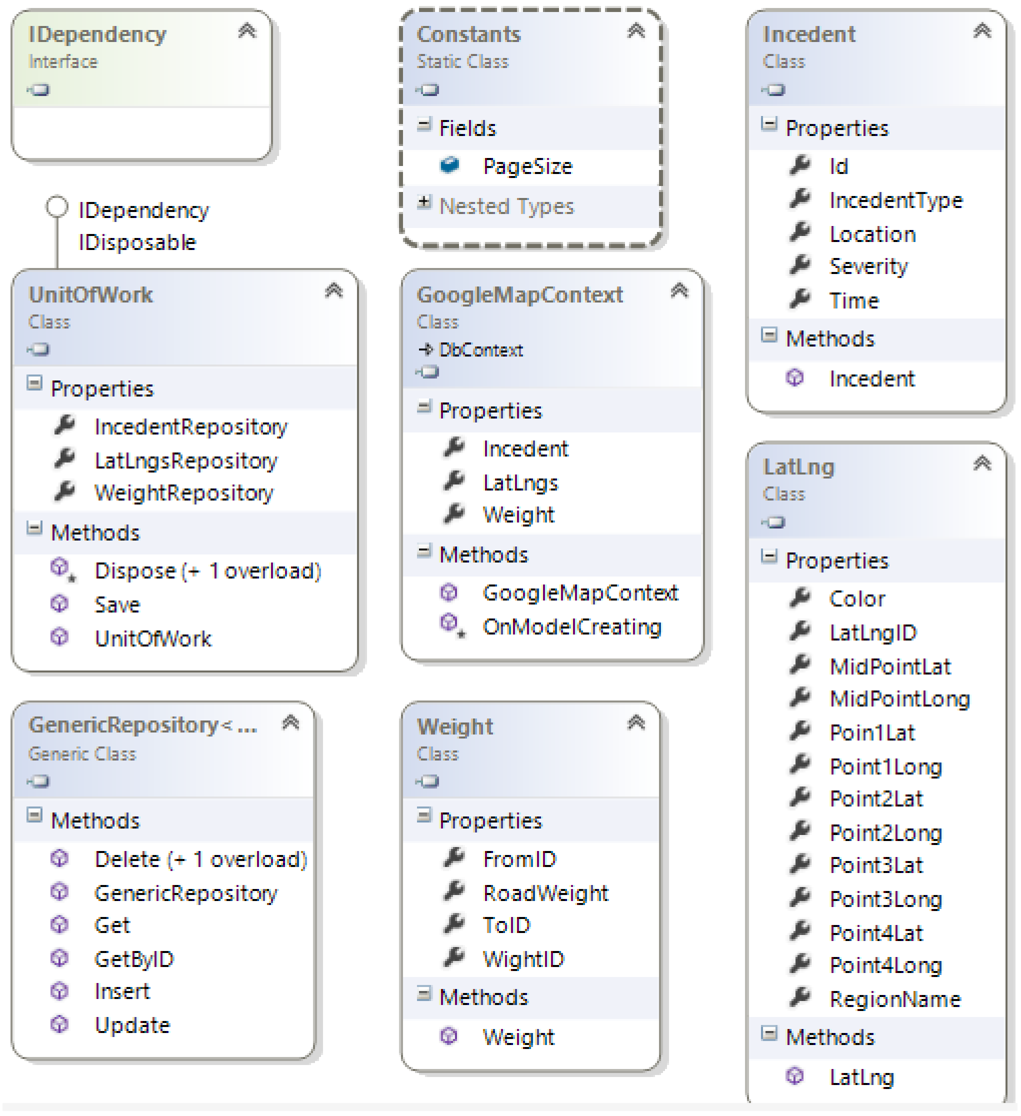


Figure 14: Data Access & Core

The diagram above shows the fields, properties and methods for the data models to store and retrieve the data from and to the database. The models shown above describes individual entities in the database.

3.1.4 Application Controllers

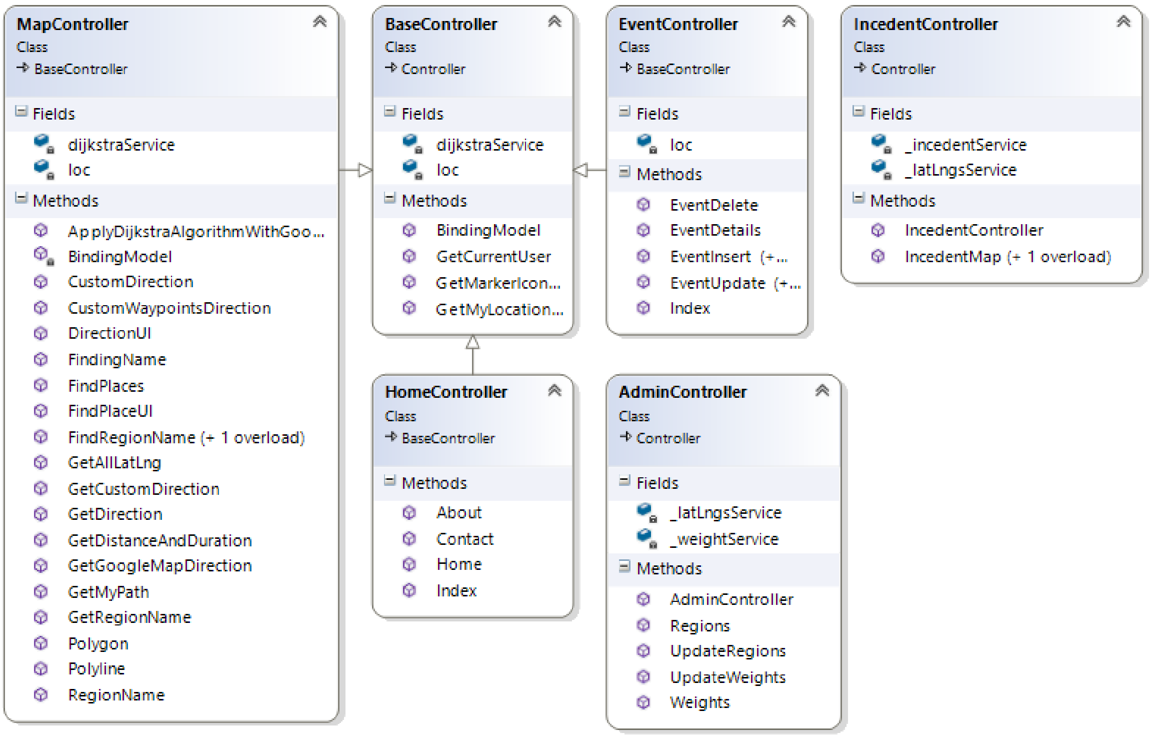


Figure 15: Application Controller

The diagram above shows the fields, properties and methods for the classes that control the display of the pages in the application.

3.2 Use Case Diagrams

Use case diagrams in the following sections describe the sequence of actions that provide something of measurable value to actors. The General options for both the server and the client applications are shown here.

3.2.2 Application Services

The following UMS 2.5 Use case diagram presents the services exposed by the application.

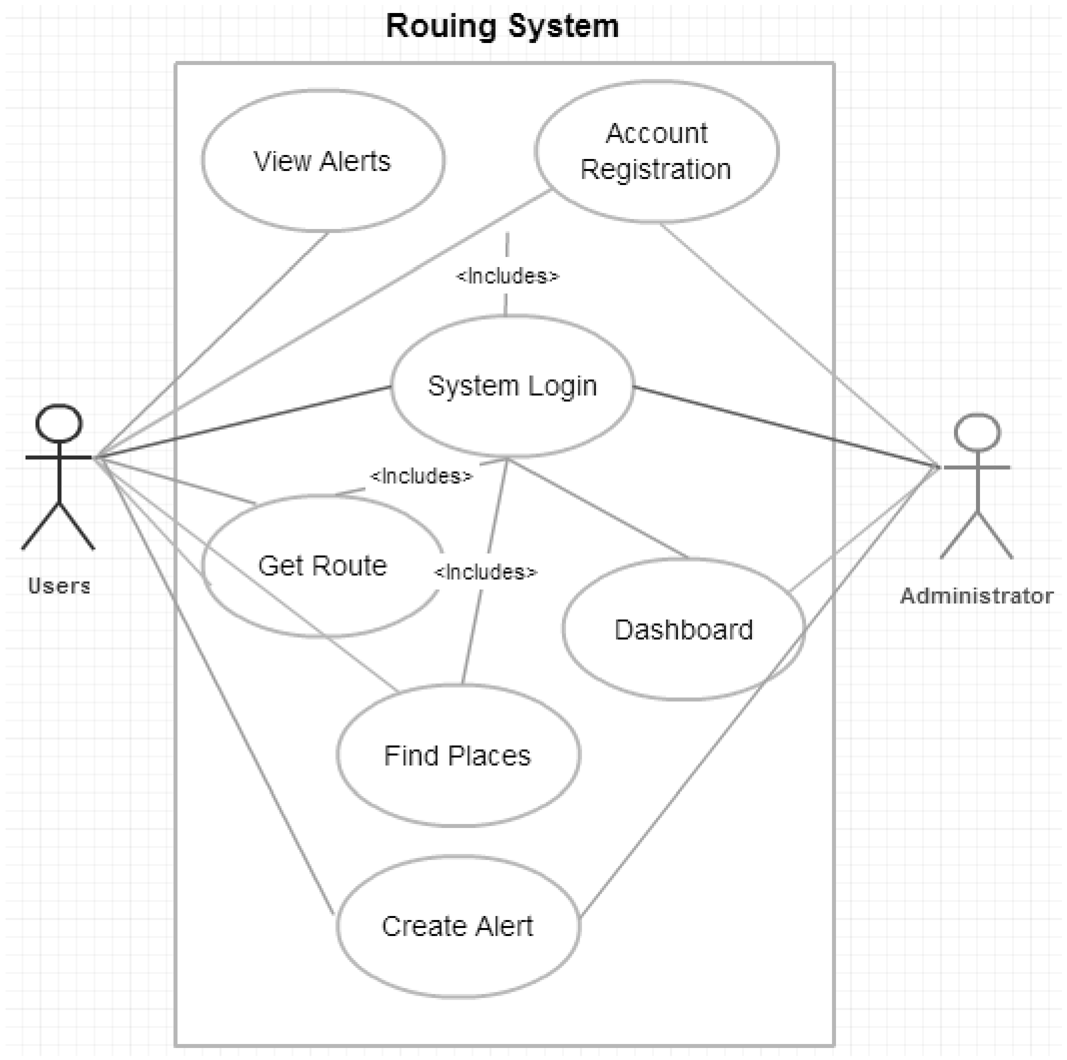


Figure 16: Application Use case diagram

3.3 Screen Shots

The screen shots of the application with detailed description of the features will help in understanding how the application functions.

3.3.1 Home Screen

Figure below is the home page for the users

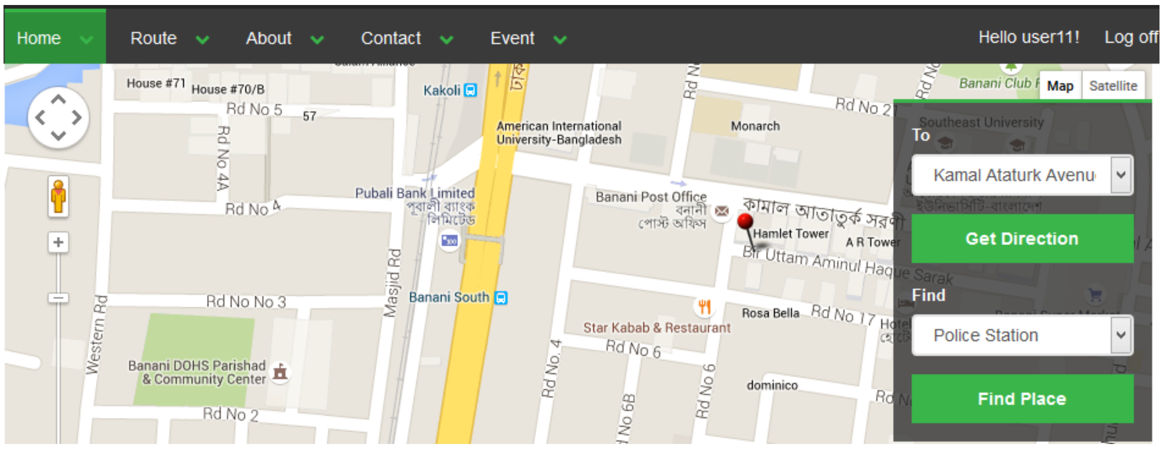


Figure 17: Home Page

On the home page users can see their current location on the map. A marker icon will display the location. The marker icon changes as the users move. A user can only see hei or her location.

In addition, from the home page uses can get direction to a place. And also find nearby places.

3.3.2 Get Route Suggestions

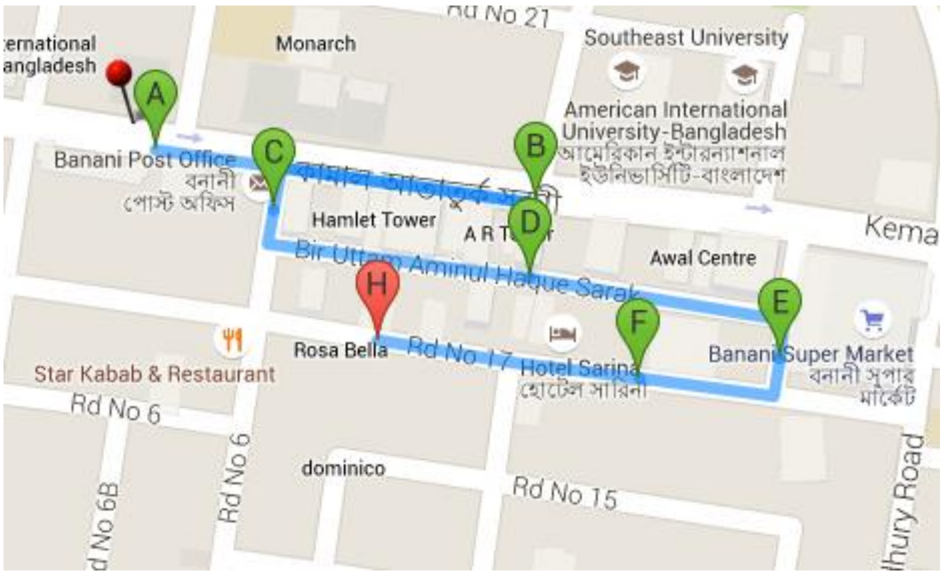


Figure 18: Route Suggestions

When a user requests a route suggestion the system does the followings:

- Consider the current location of the user as the 'Source'
- The selected "To Place" is considered as the destination
- Dijkstra's algorithm is used to find a better routing
- The weight matrix is applied to calculate an optimized route from source to destination

Based on the above criteria the suggested route is displayed on the screen. The direction is followed alphabetically.

3.3.3 Find Places

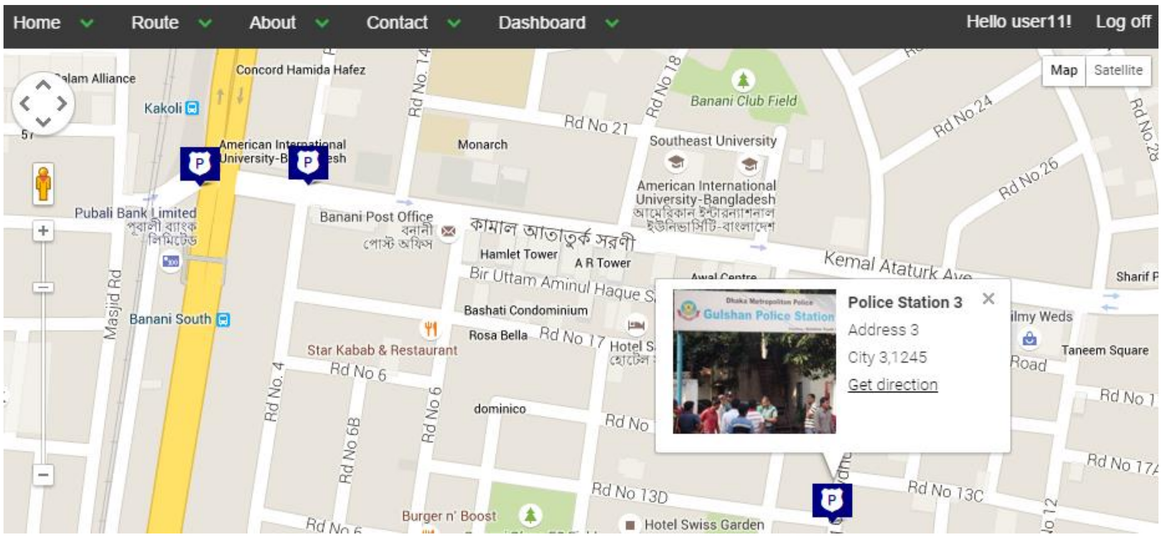


Figure 19: Find Places

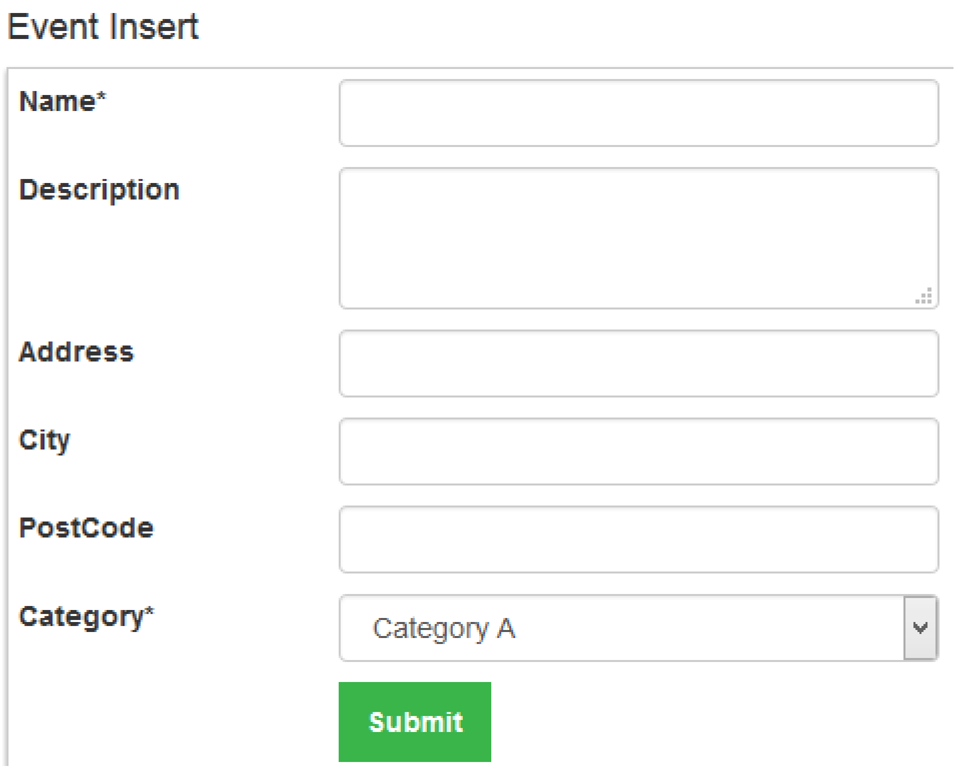
When a user requests to view the places the system does the followings:

- Gets the “category” of the places user is looking for
- Performs a search in the database for the listed places
- Displays the resulted places on the screen – on a map

Based on the above criteria the places are displayed on the screen. If the user clicks the marker icon a basic information of the place would be displayed.

Users will have a “Get Direction” link to find a better route to the selected place.

3.3.4 Create an Alert/Event

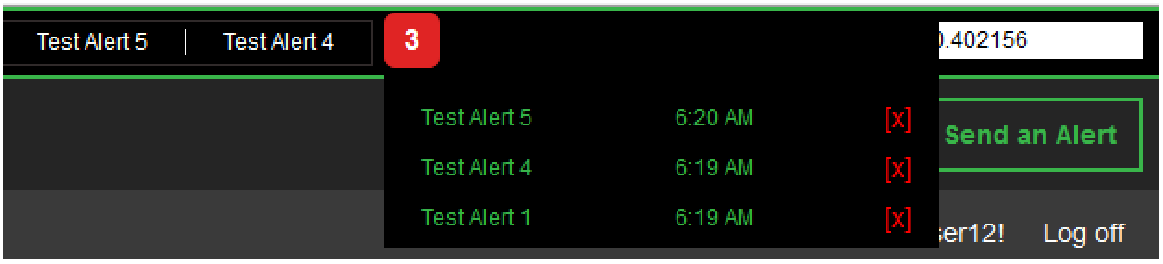

Event Insert

Name*	
Description	
Address	
City	
PostCode	
Category*	Category A

Figure 20: Add Alert/Event

This page allows the user to report and incident/event. The name of the event/Alert is mandatory. When a user submits an alert it gets broadcasted to all connected users.

3.3.5 Alert Notification



Test Alert 5	6:20 AM	[x]
Test Alert 4	6:19 AM	[x]
Test Alert 1	6:19 AM	[x]

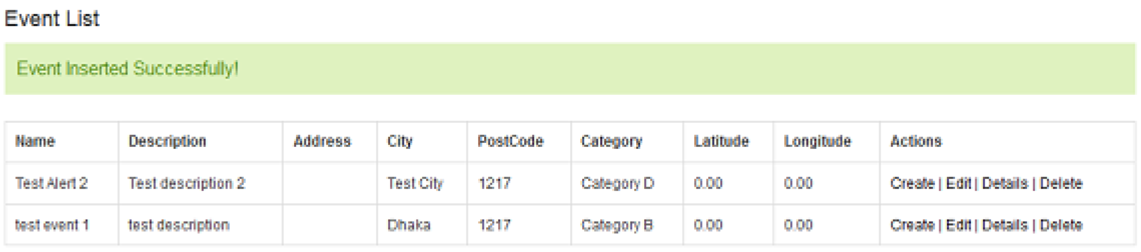
Figure 21: Alert/Event Notification

On the top right corner of the screen there are the alert icon. If a message is received by the system it displays the count of the alerts in this section.

If the user clicks the alert icon the list of alerts is displayed.

From the list the user can click an Alert and view the details of the alert.

3.3.6 Alert/Event List


Event List

Event Inserted Successfully!

Name	Description	Address	City	PostCode	Category	Latitude	Longitude	Actions
Test Alert 2	Test description 2		Test City	1217	Category D	0.00	0.00	Create | Edit | Details | Delete
test event 1	test description		Dhaka	1217	Category B	0.00	0.00	Create | Edit | Details | Delete

Figure 22: Alert/Event List

On the Alert/Event list page users can view all the alerts. So if any user misses an alert he/she can find the alerts on this page. The Alerts are sorted date descending. Which means the latest one is usually on top.

3.3.7 Dashboard

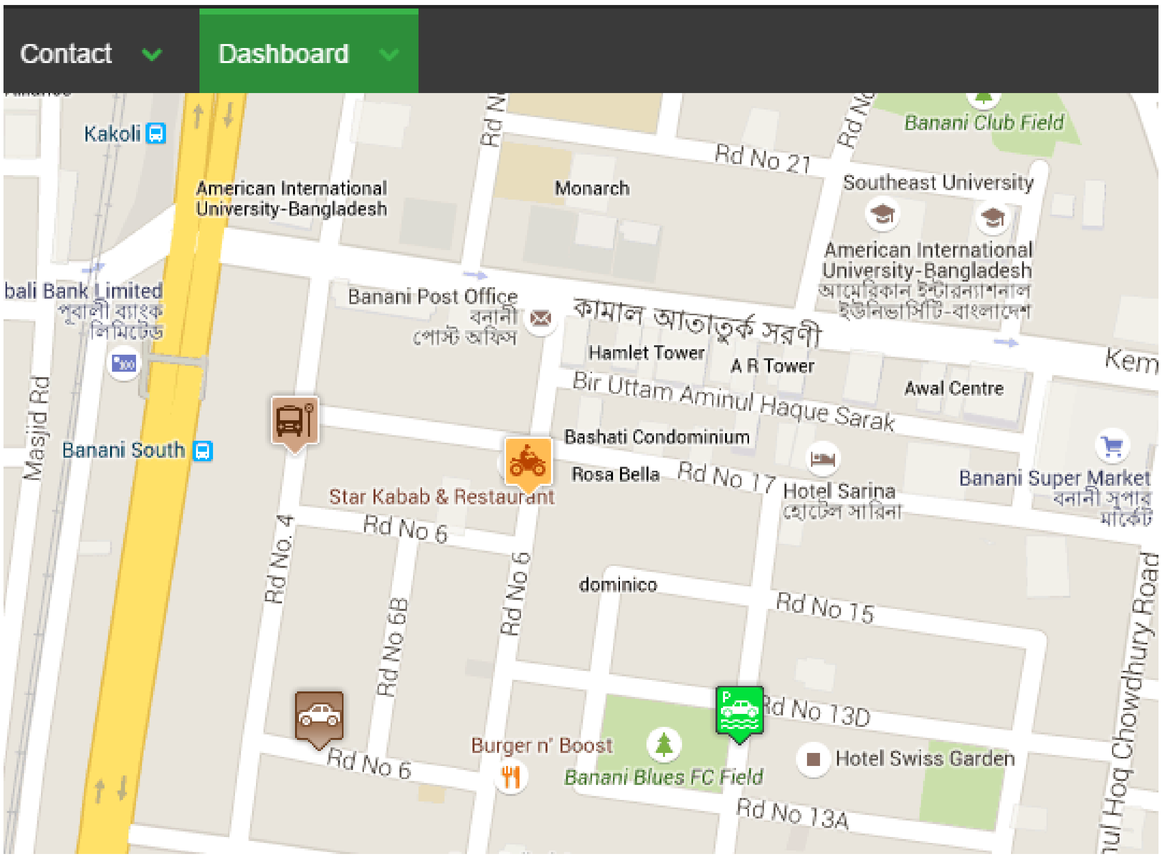


Figure 23: Dashboard

The Dashboard is the page where the admin can monitor all the traffic and their locations. The live marker for each user gets displayed here. The markers move as the user moves. From the dashboard it is possible to view the information of a particular user. It is also possible to send an alert to that particular user.

Chapter 4 - Test Input-Output

This chapter layouts the test IO tables according to the available options in the application. Each table has four columns. First one is for the description of the option for which we are going to list the input-output and the last one (‘Remarks’) is used to describe if the output is the direct result to server input or reformatted. The ‘Remarks’ sections identifies any output that may vary depending on the logic of the application.

Operation: Account Registration

Table 2: Test inputs for account registration

Description	Input	Output	Remarks
Successful creation of an account	User ID, Password	Auto login and redirect to home page	Dynamic Output
If account already exists for a given ID	User ID, Password	This account already exists	
If the given password id not well formatted	User ID, Password	Password must be at least 6 characters long	

The table above describes the probable account registration scenarios with the combination of inputs and their expected outputs. This is the IO table for the registration page in the application.

Operation: Login

Table 3: Test inputs for Login operation

Description	Input	Output	Remarks
Correct User Name wrong Password	User Name, Password	Invalid username or password.	
Wrong user name correct password	User Name, Password	Invalid username or password.	

Correct user name and correct password	User Name, Password	Auto login and redirect to home page	Dynamic Output
Wrong user name and wrong password	User Name, Password	Invalid username or password.	
Successful login	User Name, Password	Auto login and redirect to home page	Dynamic Output
Already logged in	User Name, Password	Auto login and redirect to home page	Discards previous session

The table above describes the probable login scenarios with the combination of inputs and their expected outputs. This is the IO table for the login page in the application.

Operation: Get Direction

Table 4: Test inputs for Route operation

Description	Input	Output	Remarks
If not logged in	To Location	Empty	Dynamic Output
If logged in	To Location	Display Route from source to destination	Dynamic Output
If the ‘To’ destination do not match any in the database	To Location	Empty	Dynamic Output

The table above describes the probable direction scenarios with the combination of inputs and their expected outputs. This is the IO table for the “Get Direction” functionality in the application.

Operation: Find Place

Table 5: Test inputs for Find Place operation

Description	Input	Output	Remarks
Place exists	Category of Place	Display place	Dynamic Output
Place do not exists	Category of Place	No Place is displayed	
Place no in the database	Category of Place	No place is displayed	

The table above describes the probable place finding scenarios with the combination of inputs and their expected outputs. This is the IO table for the "Find Place" functionality in the application.

Operation: View Alerts/Events

Table 6: Test inputs for Alerts/Events

Description	Input	Output	Remarks
Alert for all users		Display in notification section	
Alert for current user		Display in notification section	

The table above describes the probable alert scenarios with the combination of inputs and their expected outputs. This is the IO table for the "Alert" functionality in the application.

Operation: Create an Alert/Event

Table 7: Test inputs for Alert/Event Creation

Description	Input	Output	Remarks
All necessary fields empty	Name, Category	The Name* field is required.	
All necessary fields available	Name, Category	Event Inserted Successfully!	

The table above describes the probable alert creation scenarios with the combination of inputs and their expected outputs. This is the table for the "Add/ Edit Alert" functionality in the application.

Operation: Alert Notification

Table 8: Test inputs for Alert Notification

Description	Input	Output	Remarks
Alert notification		Number of Alerts	Dynamic Output
Alert List		List of Alerts	Dynamic Output
View an alert	Alert ID	Alert Detail	Dynamic Output

The table above describes the probable alert notification scenarios with the combination of inputs and their expected outputs. This is the IO table for the "View Alert" functionality in the application.

Operation: Logout

Table 9: Test inputs for logout operation

Description	Input	Output	Remarks
When user logs out		Redirect to home page	User session gets cleared

The table above describes the probable logout scenario with the combination of inputs and their expected outputs. This is the IO table for the "Log out" functionality in the application.

Input Range: Input Ranges are fixed in the graphical user interface so that users cannot put extreme inputs. Null or blank inputs are also checked before requesting for any service.

Output Format: Sometime unformatted outputs (e.g. Empty) are generated on the server side and they are represented for the client. This is done to increase the readability of the application.

Input-Output Format: To make the inputs understandable by the application I have formatted user input and transformed the server's output in a human understandable form. On the server side the user input is parsed and mapped to the necessary data.

Chapter 5: Conclusion

5.1 Conclusion

The “Real Time Bi-directional Traffic Management Support System” system that has been successfully implemented with GPS and WebSocketby proved the adage of “mobility & flexibility” for traffic management. As a traffic support system the application provides opportunities for tactical and targeted communication.
The following point explains the accomplishment of the given objectives (that were set forth for the project):

- Studied and analyzed the WebSocket technology and its various technical aspects.
- Designed and implemented WebSocket based server that is capable of processing the client requests and acts accordingly.
- The server application was developed not only for monitoring traffic but also to provide services to the end users.

This WebSocket based system can be used in any geographical location anywhere in the world including: Forests, Mountains, Deserts, north or south poles where installation and maintenance of any ground based monitoring system is difficult.

Places where road conditions changes very frequently like tornedo zones, hazardous zones, congested traffic zones, this application may be very helpful for routing traffics in an emergency situation.

5.2 Future Work

The traffic management support system can be further enhanced by embedding the software into mobile devices. In addition, microcontroller based implementation with GPS and GSM/EDGE unit support will bring down the costs significantly. Future work may include improving application security so that an intruder will not compromise the network. Also develop a system so that every vehicle will contain the tee above device. This can be done with the vehicle registration authority to connect the vehicle registration system with the application. Another challenge is to deal with dynamic environment enhancing the routing and extending the coverage of the application.

Appendices

Appendix A: Acronyms

API - Application Program Interface

CCTV - Closed-circuit Television

GPS - Global Positioning System

HTTP - Hypertext Transfer Protocol

HTML - Hyper Text Markup Language

IETF - Internet Engineering Task Force

IP - Internet Protocol

IIS - Internet Information Services

JS - JavaScript

MS - Millisecond

RFC - Request for Comments

RFID - Radio-frequency identification

SSE - Server-Sent Events

Appendix B: Source Code

BaseController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MVC5.Web.ViewModels;
using MVC5.Web.Services;
using Dijkstra;
using MVC5.Web.Models;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity;

namespace MVC5.Web.Controllers
{
    public class BaseController : Controller
    {
        LocationService loc = new LocationService();
        DijkstraService dijkstraService = new DijkstraService();

        public void BindingModel(CustomWaypointsViewModel model)
        {
            var allCoordinates = loc.GetAll();

            if (model.FromID == 0 && model.ToID == 0)
            {
                model.FromID = 2;
                model.ToID = 2;

                var s = (from ad in allCoordinates
                         where ad.LatLngID == 2
                         select ad).FirstOrDefault();

                model.StartingMidPointLat = s.MidPointLat;
                model.StartingMidPointLong = s.MidPointLong;

                model.EndingMidPointLat = s.MidPointLat;
                model.EndingMidPointLong = s.MidPointLong;

            }
            else
            {
                var s = (from ad in allCoordinates
                         where ad.LatLngID == model.FromID
                         select ad).FirstOrDefault();

                var e = (from ad in allCoordinates
                         where ad.LatLngID == model.ToID
                         select ad).FirstOrDefault();

                model.StartingMidPointLat = s.MidPointLat;
                model.StartingMidPointLong = s.MidPointLong;

                model.EndingMidPointLat = e.MidPointLat;
                model.EndingMidPointLong = e.MidPointLong;
            }

            var shortestPathNodeList =
dijkstraService.GetMinDistancePathsLatlng(model.FromID,
model.ToID).Take(8).ToList();
            model.ShortestPathCoordinates = shortestPathNodeList;
            model.Coordinates = allCoordinates;
        }

        public MyLocation GetMyLocationByUserId(string userId)
        {
            MyLocation myLoc = null;
            var testRoute =
loc.GetTestRouteByUserId(userId).FirstOrDefault();
            if (testRoute != null)
            {
                myLoc = new MyLocation()
                {
                    MyLatitude = testRoute.Latitude,
                    MyLongitude = testRoute.Longitude,
                    MyMarkerIcon = GetMarkerIconByCategory()
                };
            }
            return myLoc;
        }

        public ApplicationUser GetCurrentUser()
        {
            ApplicationUser user = null;
            if
(ControllerContext.HttpContext.User.Identity.IsAuthenticated)
            {
                var store = new UserStore<ApplicationUser>(new
ApplicationDbContext());
                var userManager = new
UserManager<ApplicationUser>(store);
                user =
userManager.FindByNameAsync(User.Identity.Name).Result;
            }
            return user;
        }

        public string GetMarkerIconByCategory()
        {
            string markerIcon = string.Empty;
            var user = GetCurrentUser();
            if (user != null)
            {
                switch (user.Category)
                {
                    case "Category A":
                        markerIcon =
"/content/images/markers/Category_A.png";
                        break;
                    case "Category B":
                        markerIcon =
"/content/images/markers/Category_B.png";
                        break;
                    case "Category C":
                        markerIcon =
"/content/images/markers/Category_C.png";
                        break;
                    case "Category D":
                        markerIcon =
"/content/images/markers/Category_D.png";
                        break;
                    default:
                        markerIcon =
"/content/images/markers/Category_A.png";
                        break;
                }
            }

            return markerIcon;
        }
    }
}
```

AccountController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin.Security;
using MVC5.Web.Models;
using System.Web.Security;
using MVC5.Web.Services;

namespace MVC5.Web.Controllers
{
    [Authorize]
    public class AccountController : Controller
    {
        LocationService loc = new LocationService();
        public AccountController()
            : this(new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(new ApplicationDbContext())))
        {
        }

        public AccountController(UserManager<ApplicationUser>
userManager)
        {
            UserManager = userManager;
        }

        public UserManager<ApplicationUser> UserManager { get; private
set; }

        //
        // GET: /Account/Login
        [AllowAnonymous]
        public ActionResult Login(string returnUrl)
        {
            ViewBag.ReturnUrl = returnUrl;
            return View();
        }

        //
        // POST: /Account/Login
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model,
string returnUrl)
        {
            if (ModelState.IsValid)
            {
                var user = await UserManager.FindAsync(model.UserName,
model.Password);
                if (user != null)
                {
                    await SignInAsync(user, model.RememberMe);
                    return RedirectToLocal(returnUrl);
                }
                else
                {
                    ModelState.AddModelError("", "Invalid username or
password.");
                }
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        //
        // GET: /Account/Register
        [AllowAnonymous]
        public ActionResult Register()
        {
            RegisterViewModel model = new RegisterViewModel();

            model.CategoryList = loc.GetCategoryList();
            return View(model);
        }

        //
        // POST: /Account/Register
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterViewModel
model)
        {
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser() { UserName =
model.UserName, Category = model.Category };
                var result = await UserManager.CreateAsync(user,
model.Password);
                if (result.Succeeded)
                {
                    await SignInAsync(user, isPersistent: false);
                    return RedirectToAction("Home", "Home");
                }
                else
                {
                    AddErrors(result);
                }
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        //
        // POST: /Account/Disassociate
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Disassociate(string
loginProvider, string providerKey)
        {
            ManageMessageId? message = null;
            IdentityResult result = await
UserManager.RemoveLoginAsync(User.Identity.GetUserId(), new
UserLoginInfo(loginProvider, providerKey));
            if (result.Succeeded)
            {
                message = ManageMessageId.RemoveLoginSuccess;
            }
            else
            {
                message = ManageMessageId.Error;
            }
            return RedirectToAction("Manage", new { Message = message
});
        }

        //
        // GET: /Account/Manage
        public ActionResult Manage(ManageMessageId? message)
        {
            ViewBag.StatusMessage =
                message == ManageMessageId.ChangePasswordSuccess ? "Your
password has been changed."
                : message == ManageMessageId.SetPasswordSuccess ? "Your
password has been set."
                : message == ManageMessageId.RemoveLoginSuccess ? "The
external login was removed."
                : message == ManageMessageId.Error ? "An error has
occurred."
                : "";
            ViewBag.HasLocalPassword = HasPassword();
            ViewBag.ReturnUrl = Url.Action("Manage");
            return View();
        }

        //
        // POST: /Account/Manage
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Manage(ManageUserViewModel
model)
```

```
        {
            bool hasPassword = HasPassword();
            ViewBag.HasLocalPassword = hasPassword;
            ViewBag.ReturnUrl = Url.Action("Manage");
            if (hasPassword)
            {
                if (ModelState.IsValid)
                {
                    IdentityResult result = await
UserManager.ChangePasswordAsync(User.Identity.GetUserId(),
model.OldPassword, model.NewPassword);
                    if (result.Succeeded)
                    {
                        return RedirectToAction("Manage", new { Message
= ManageMessageId.ChangePasswordSuccess });
                    }
                    else
                    {
                        AddErrors(result);
                    }
                }
            }
            else
            {
                // User does not have a password so remove any
validation errors caused by a missing OldPassword field
                ModelState state = ModelState["OldPassword"];
                if (state != null)
                {
                    state.Errors.Clear();
                }

                if (ModelState.IsValid)
                {
                    IdentityResult result = await
UserManager.AddPasswordAsync(User.Identity.GetUserId(),
model.NewPassword);
                    if (result.Succeeded)
                    {
                        return RedirectToAction("Manage", new { Message
= ManageMessageId.SetPasswordSuccess });
                    }
                    else
                    {
                        AddErrors(result);
                    }
                }
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        //
        // POST: /Account/ExternalLogin
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public ActionResult ExternalLogin(string provider, string
returnUrl)
        {
            // Request a redirect to the external login provider
            return new ChallengeResult(provider,
Url.Action("ExternalLoginCallback", "Account", new { ReturnUrl =
returnUrl }));
        }

        //
        // GET: /Account/ExternalLoginCallback
        [AllowAnonymous]
        public async Task<ActionResult> ExternalLoginCallback(string
returnUrl)
        {
            var loginInfo = await
AuthenticationManager.GetExternalLoginInfoAsync();
            if (loginInfo == null)
            {
                return RedirectToAction("Login");
            }

            // Sign in the user with this external login provider if the
user already has a login
            var user = await UserManager.FindAsync(loginInfo.Login);
            if (user != null)
            {
                await SignInAsync(user, isPersistent: false);
                return RedirectToLocal(returnUrl);
            }
            else
            {
                // If the user does not have an account, then prompt the
user to create an account
                ViewBag.ReturnUrl = returnUrl;
                ViewBag.LoginProvider = loginInfo.Login.LoginProvider;
                return View("ExternalLoginConfirmation", new
ExternalLoginConfirmationViewModel { UserName =
loginInfo.DefaultUserName });
            }
        }

        //
        // POST: /Account/LinkLogin
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult LinkLogin(string provider)
        {
            // Request a redirect to the external login provider to link
a login for the current user
            return new ChallengeResult(provider,
Url.Action("LinkLoginCallback", "Account"), User.Identity.GetUserId());
        }

        //
        // GET: /Account/LinkLoginCallback
        public async Task<ActionResult> LinkLoginCallback()
        {
            var loginInfo = await
AuthenticationManager.GetExternalLoginInfoAsync(XsrfKey,
User.Identity.GetUserId());
            if (loginInfo == null)
            {
                return RedirectToAction("Manage", new { Message =
ManageMessageId.Error });
            }
            var result = await
UserManager.AddLoginAsync(User.Identity.GetUserId(), loginInfo.Login);
            if (result.Succeeded)
            {
                return RedirectToAction("Manage");
            }
            return RedirectToAction("Manage", new { Message =
ManageMessageId.Error });
        }

        //
        // POST: /Account/ExternalLoginConfirmation
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model,
string returnUrl)
        {
            if (User.Identity.IsAuthenticated)
            {
                return RedirectToAction("Manage");
            }

            if (ModelState.IsValid)
            {
                // Get the information about the user from the external
login provider
                var info = await
AuthenticationManager.GetExternalLoginInfoAsync();
                if (info == null)
                {
                    return View("ExternalLoginFailure");
                }
                var user = new ApplicationUser() { UserName =
model.UserName };
                var result = await UserManager.CreateAsync(user);
                if (result.Succeeded)
                {
                    result = await UserManager.AddLoginAsync(user.Id,
info.Login);
                    if (result.Succeeded)
                    {
                        await SignInAsync(user, isPersistent: false);
                        return RedirectToLocal(returnUrl);
                    }
                }
                AddErrors(result);
            }

            ViewBag.ReturnUrl = returnUrl;
            return View(model);
        }

        //
        // POST: /Account/LogOff
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult LogOff()
        {
            AuthenticationManager.SignOut();
            return RedirectToAction("Home", "Home");
        }

        //
        // GET: /Account/ExternalLoginFailure
        [AllowAnonymous]
        public ActionResult ExternalLoginFailure()
        {
            return View();
        }

        [ChildActionOnly]
        public ActionResult RemoveAccountList()
        {
            var linkedAccounts = 
UserManager.GetLogins(User.Identity.GetUserId());
            ViewBag.ShowRemoveButton = HasPassword() || 
linkedAccounts.Count > 1;
            return (ActionResult)PartialView("_RemoveAccountPartial", 
linkedAccounts);
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing && UserManager != null)
            {
                UserManager.Dispose();
                UserManager = null;
            }
            base.Dispose(disposing);
        }

        #region Helpers
        // Used for XSRF protection when adding external logins
        private const string XsrfKey = "XsrfId";

        private IAuthenticationManager AuthenticationManager
        {
            get
            {
                return HttpContext.GetOwinContext().Authentication;
            }
        }

        private async Task SignInAsync(ApplicationUser user, bool 
isPersistent)
        {

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ExternalCookie)
;
            var identity = await UserManager.CreateIdentityAsync(user, 
DefaultAuthenticationTypes.ApplicationCookie);
            AuthenticationManager.SignIn(new AuthenticationProperties() 
{ IsPersistent = isPersistent }, identity);
        }

        private void AddErrors(IdentityResult result)
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error);
            }
        }

        private bool HasPassword()
        {
            var user = UserManager.FindById(User.Identity.GetUserId());
            if (user != null)
            {
                return user.PasswordHash != null;
            }
            return false;
        }

        public enum ManageMessageId
        {
            ChangePasswordSuccess,
            SetPasswordSuccess,
            RemoveLoginSuccess,
            Error
        }

        private ActionResult RedirectToLocal(string returnUrl)
        {
            if (Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            else
            {
                return RedirectToAction("Home", "Home");
            }
        }

        private class ChallengeResult : HttpUnauthorizedResult
        {
            public ChallengeResult(string provider, string redirectUri)
: this(provider, redirectUri, null)
            {
            }

            public ChallengeResult(string provider, string redirectUri,
string userId)
            {
                LoginProvider = provider;
                RedirectUri = redirectUri;
                UserId = userId;
            }

            public string LoginProvider { get; set; }
            public string RedirectUri { get; set; }
            public string UserId { get; set; }

            public override void ExecuteResult(ControllerContext
context)
            {
                var properties = new AuthenticationProperties() {
RedirectUri = RedirectUri };
                if (UserId != null)
                {
                    properties.Dictionary[XsrfKey] = UserId;
                }

context.HttpContext.GetOwinContext().Authentication.Challenge(properties
, LoginProvider);
            }
        }
        #endregion
    }
}
```

DashboardController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVC5.Web.Controllers
{
    public class DashboardController : Controller
    {
        public ActionResult Dashboard()
        {
            return View();
        }
      }
}
```

EventController.cs

```
using Microsoft.AspNet.SignalR;
using MVC5.Web.Models;
using MVC5.Web.ServerHub;
using MVC5.Web.Services;
using MVC5.Web.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVC5.Web.Controllers
{
    public class EventController : BaseController
    {
        LocationService loc = new LocationService();

        public ActionResult Index()
        {
            var allEvents =
loc.GetAllEvents().OrderByDescending(x=>x.DateUpdated);
            return View(allEvents);
        }

        public ViewResult EventDetails(int id)
        {
            var ev = loc.GetEventById(id);

            return View(ev);
        }

        public ActionResult EventInsert()
        {
            var model = new EventViewModel();
            model.CategoryList = loc.GetCategoryList();
            return View(model);
        }

        [HttpPost]
        public ActionResult EventInsert(EventViewModel model)
        {
            try
            {
                if (ModelState.IsValid)
                {
                    var e = new Event()
                    {
                        Name = model.Name,
                        Description = model.Description,
                        City = model.City,
                        PostCode = model.PostCode,
                        Category = model.Category,
                        Latitude = model.Latitude,
                        Longitude =model.Longitude,
                        DateCreated = DateTime.Now,
                        DateUpdated = DateTime.Now
                    };
                    loc.InsertEvent(e);
                    TempData["Success"] = "Event Inserted
Successfully!";

                                EventHub.BroadcastFromServer(e);

                    return RedirectToAction("Index");
                }
                model.CategoryList = loc.GetCategoryList();
                return View(model);
            }
            catch
            {
                model.CategoryList = loc.GetCategoryList();
                return View(model);
            }
        }

        public ActionResult EventUpdate(int id)
        {
            var ev = loc.GetEventById(id);
            EventViewModel model = null;
            if (ev != null)
            {
                model = new EventViewModel()
                {
                    Id = ev.Id,
                    Name = ev.Name,
                    Description = ev.Description,
                    City = ev.City,
                    Address = ev.Address,
                    Category = ev.Category,
                    PostCode = ev.PostCode,
                    CategoryList = loc.GetCategoryList()
                };
            }

            return View(model);
        }

        [HttpPost]
        public ActionResult EventUpdate(int id, EventViewModel model)
        {
            try
            {
                if (ModelState.IsValid)
                {
                    var ev = loc.GetEventById(id);
                    if (ev != null)
                    {
                        ev.Name = model.Name;
                        ev.Address = model.Address;
                        ev.Category = model.Category;
                        ev.Description = model.Description;
                        ev.City = model.City;
                        ev.PostCode = model.PostCode;
                        ev.DateUpdated = DateTime.Now;
                        loc.UpdateEvent(ev);
                    }
                    TempData["Success"] = "Event Updated Successfully!";
                    return RedirectToAction("Index");
                }
                model.CategoryList = loc.GetCategoryList();
                return View(model);
            }
            catch
            {
                model.CategoryList = loc.GetCategoryList();
                return View(model);
            }
        }

        public ActionResult EventDelete(int id)
        {
            var ev = loc.GetEventById(id);
            loc.DeleteEvent(ev);
            TempData["Success"] = "Event Deleted Successfully!";
            return RedirectToAction("Index");
        }

    }
}
```

MapController.cs

```
using Dijkstra;
using Map.Contex;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using MVC5.Web.Controllers;
using MVC5.Web.Models;
using MVC5.Web.Services;
using MVC5.Web.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Xml.Linq;

namespace Map.Controllers
{
    public class MapController : BaseController
    {
        LocationService loc = new LocationService();
        DijkstraService dijkstraService = new DijkstraService();

        public ActionResult Polygon()
        {
            return View();
        }

        public ActionResult Polyline()
        {
            return View();
        }

        public ActionResult GetAllLatLng()
        {
            var model = loc.GetAll();
            return Json(model);
        }

        public ActionResult GetDistanceAndDuration()
        {
            return View();
        }

        public ActionResult GetRegionName()
        {
            return View();
        }

        public ActionResult FindingName()
        {
            return View();
        }

        public ActionResult RegionName(double latitude, double
longitude)
        {
            var regionName = loc.GetRegionFromLatLong(latitude,
longitude);
            return Json(regionName);
        }

        [HttpGet]
        public ActionResult FindRegionName()
        {
            return View();
        }
        [HttpPost]
        public ActionResult FindRegionName(string latlng)
        {
            return Json(loc.GetRegionFromServerSide(latlng));
        }

        public ActionResult GetGoogleMapDirection()
        {
            return View();
        }

        public ActionResult CustomDirection(GooglemapViewModel model)
        {
            model.AllRegionName = loc.GetAll();
            return View(model);
        }

        public ActionResult CustomWaypointsDirection(GooglemapViewModel
model)
        {
            model.AllRegionName = loc.GetAll();

            return View(model);
        }

        public ActionResult GetDirection(CustomWaypointsViewModel model)
        {
            BindingModel(model);
            return View(model);
        }

        public ActionResult
ApplyDijkstraAlgorithmWithGooglemap(CustomWaypointsViewModel model)
        {
            BindingModel(model);
            return View(model);
        }

        //[HttpGet]
        //public ActionResult GetShortestPathCoordinateList(int fromID,
int toID)
        //{
        //    var shortestPathNodeList =
dijkstraService.GetMinDistancePathsLatlng(fromID, toID).Take(8);

        //    return Json(shortestPathNodeList,
JsonRequestBehavior.AllowGet);
        //}

        #region Partial Actions

        [OutputCache(Duration = 3600)]
        public PartialViewResult DirectionUI()
        {
            CustomWaypointsViewModel model = new
CustomWaypointsViewModel();
            BindingModel(model);
            return PartialView("_GetDirectionUI", model);
        }
        [OutputCache(Duration = 3600)]
        public PartialViewResult FindPlaceUI()
        {
            PlaceViewModel model = new PlaceViewModel();
            List<SelectListItem> list = new List<SelectListItem>();
            list.Add(new SelectListItem() { Text = "Police Station",
Value = "Police Station", Selected = true });
            list.Add(new SelectListItem() { Text = "Fire Service", Value
= "Fire Service" });
            list.Add(new SelectListItem() { Text = "Hospital", Value =
"Hospital" });
            list.Add(new SelectListItem() { Text = "DMP Booth", Value =
"DMP Booth" });
            list.Add(new SelectListItem() { Text = "Service Center",
Value = "Service Center" });

            model.CategoryList = list;

            //return PartialView("_FindPlaceUI", new SelectList(list,
"Text", "Value"));

            return PartialView("_FindPlaceUI", model);
        }

        public ActionResult GetMyPath(string userId)
        {
            var testRoutes = loc.GetTestRouteByUserId(userId);
            return Json(testRoutes, JsonRequestBehavior.AllowGet);
        }


        public ActionResult FindPlaces(PlaceViewModel model)
        {
            model.Places = loc.GetPlacesByCategory(model.PlaceCategory);
            model.MyLocation =
GetMyLocationByUserId(GetCurrentUser()!=null ? GetCurrentUser().Id :
null);
            return View(model);
        }

        public ActionResult GetCustomDirection(CustomWaypointsViewModel
model, string FLat, string FLan, string TLat, string TLan)
        {
            LatLng startingLocation = null;
            LatLng endingLocation = null;

            if (model.ToID > 1)
            {
                startingLocation =
loc.FindNearestLocation(model.StartingMidPointLat,
model.StartingMidPointLong);
                endingLocation = loc.GetLatLngById(model.ToID);
            }
            else
            {
                startingLocation =
loc.FindNearestLocation(decimal.Parse(FLat), decimal.Parse(FLan));
                endingLocation = loc.GetLatLng(decimal.Parse(TLat),
decimal.Parse(TLan));
            }
            if (startingLocation != null && endingLocation != null)
            {
                model.EndingMidPointLat = endingLocation.MidPointLat;
                model.EndingMidPointLong = endingLocation.MidPointLong;
                model.ToID = endingLocation.LatLngID;

                model.StartingMidPointLat =
startingLocation.MidPointLat;
                model.StartingMidPointLong =
startingLocation.MidPointLong;
                model.FromID = startingLocation.LatLngID;

                var totalShortestPathNodeList =
dijkstraService.GetMinDistancePathsLatlng(model.FromID,
model.ToID).ToList();
                model.ShortestPathCoordinates =
totalShortestPathNodeList.Take(8).ToList();
                model.TotalShortestPathCoordinates =
totalShortestPathNodeList;
            }

            model.MyLocation = GetMyLocationByUserId(GetCurrentUser() !=
null ? GetCurrentUser().Id : null);

            return View(model);
        }

        #endregion

        private void BindingModel(CustomWaypointsViewModel model)
        {
            var allCoordinates = loc.GetAll();

            if (model.FromID == 0 && model.ToID == 0)
            {
                model.FromID = 2;
                model.ToID = 2;

                var s = (from ad in allCoordinates
                         where ad.LatLngID == 2
                         select ad).FirstOrDefault();

                model.StartingMidPointLat = s.MidPointLat;
                model.StartingMidPointLong = s.MidPointLong;

                model.EndingMidPointLat = s.MidPointLat;
                model.EndingMidPointLong = s.MidPointLong;

            }
            else
            {
                var s = (from ad in allCoordinates
                         where ad.LatLngID == model.FromID
                         select ad).FirstOrDefault();

                var e = (from ad in allCoordinates
                         where ad.LatLngID == model.ToID
                         select ad).FirstOrDefault();

                model.StartingMidPointLat = s.MidPointLat;
                model.StartingMidPointLong = s.MidPointLong;
                model.EndingMidPointLat = e.MidPointLat;
                model.EndingMidPointLong = e.MidPointLong;
            }

            var shortestPathNodeList = 
dijkstraService.GetMinDistancePathsLatlng(model.FromID, 
model.ToID).Take(8).ToList();
            model.ShortestPathCoordinates = shortestPathNodeList;
            model.Coordinates = allCoordinates;
        }
    }
}
```

// Code for Routing //

Database Context Object:

```
public class GoogleMapContext : DbContext
    {
        public GoogleMapContext()
            : base("GoogleMapDbContex")
        {
            Database.SetInitializer<GoogleMapContext>(new 
CreateDatabaseIfNotExists<GoogleMapContext>());
        }

        public DbSet<LatLng> LatLngs { get; set; }
        public DbSet<Incedent> Incedent { get; set; }
        public DbSet<Weight> Weight { get; set; }

        protected override void OnModelCreating(DbModelBuilder 
modelBuilder)
        {

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

            modelBuilder.Entity<LatLng>().ToTable("LatLngs");
            modelBuilder.Entity<Incedent>().ToTable("Incedent");
            modelBuilder.Entity<Weight>().ToTable("Weights");

            Database.SetInitializer<DbContext>(null);
        }
    }
```

Get Shortest Path:

```
var totalShortestPathNodeList = 
dijkstraService.GetMinDistancePathsLatlng(model.FromID, 
model.ToID).ToList();
```

Repository Methods – Insert/Update/Delete/ Get:

```
    public class GenericRepository<TEntity> where TEntity : class
    {
        internal GoogleMapContext context;
        internal DbSet<TEntity> dbSet;

        public GenericRepository(GoogleMapContext context)
        {
            this.context = context;
            this.dbSet = context.Set<TEntity>();
        }

        public virtual IEnumerable<TEntity> Get(
            Expression<Func<TEntity, bool>> filter = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>>
orderBy = null,
            string includeProperties = "")
        {
            IQueryable<TEntity> query = dbSet;

            if (filter != null)
            {
                query = query.Where(filter);
            }

            foreach (var includeProperty in includeProperties.Split
                (new char[] { ',' },
StringSplitOptions.RemoveEmptyEntries))
            {
                query = query.Include(includeProperty);
            }

            if (orderBy != null)
            {
                return orderBy(query).ToList();
            }
            else
            {
                return query.ToList();
            }
        }

        public virtual TEntity GetByID(object id)
        {
            return dbSet.Find(id);
        }

        public virtual void Insert(TEntity entity)
        {
            dbSet.Add(entity);
        }

        public virtual void Delete(object id)
        {
            TEntity entityToDelete = dbSet.Find(id);
            Delete(entityToDelete);
        }

        public virtual void Delete(TEntity entityToDelete)
        {
            if (context.Entry(entityToDelete).State ==
EntityState.Detached)
            {
                dbSet.Attach(entityToDelete);
            }
            dbSet.Remove(entityToDelete);
        }
```

```
    public virtual void Update(TEntity entityToUpdate)
    {
        dbSet.Attach(entityToUpdate);
        context.Entry(entityToUpdate).State = EntityState.Modified;
    }
}
```

References

[1] Traffic Pollution Kills Thousands," BBC News, 2000. http://news.bbc.co.uk/2/hi/health/905016.stm (Accessed on 10th May, 2015)

[2] "Cutting Greenhouse Gas Emissions has Major Direct Health Benefits," The Lancet Medical Journal, December 2009. http://climateprogress.org/2009/12/01/the-lancetmedical-journal-cutting-greenhouse-gas-emissions-hasmajor-direct-health-benefits (Accessed on 10th May, 2015)

[3] "Road Pollution Can Damage Kid's Lungs, Hearts," MSNBC Health, 2007. http://www.msnbc.msn.com/id/16831975/ (Accessed on 10th May, 2015)

[4] https://www.websocket.org

[5] HTML5 Web Sockets: A Quantum Leap in Scalability for the Web. http://www.websocket.org/quantum.html (Accessed on 10th May, 2015)

[6] http://www.sapub.org/global/showpaperpdf.aspx?doi=10.5923/j.web.20130201.01

[7] http://juerkkil.iki.fi/files/writings/websocket2012.pdf (Accessed on 10th May, 2015)

[8] http://www.pubnub.com/websockets/ (Accessed on 10th May, 2015)

[9] http://article.sapub.org/10.5923.j.web.20130201.01.html (Accessed on 10th May, 2015)

[10] http://css.dzone.com/articles/techniques-real-time-client (Accessed on 10th May, 2015)

[11] http://en.wikipedia.org/wiki/Global_Positioning_System (Accessed on 10th May, 2015)

[12] http://www8.garmin.com/aboutGPS/ (Accessed on 10th May, 2015)

[13] http://en.wikipedia.org/wiki/Google_Maps (Accessed on 10th May, 2015)

[14] https://developers.google.com/maps/documentation/javascript/usage (Accessed on 10th May, 2015)

[15] https://developers.google.com/maps/licensing (Accessed on 10th May, 2015)

[16] http://signalr.blogspot.com/ (Accessed on 10th May, 2015)

[17] Browser versions support for WebSocket.
http://caniuse.com/#feat=websockets (Accessed on 10th May, 2015)

[18] Unified Modeling Language
http://en.wikipedia.org/wiki/Unified_Modeling_Language (Accessed on 10th May, 2015)

[19] Asp.net user authentication
http://www.asp.net/web-forms/overview/older-versions-security/introduction/forms-authentication-configuration-and-advanced-topics-cs (Accessed on 10th May, 2015)