DEVELOPMENT OF INFOTAINMENT SYSTEM: TRAVEL COST CALCULATOR

By

HASEEFAH NAZAAHIAH BINTI AHMAD MOGHNI

FINAL PROJECT REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

DEVELOPMENT OF INFOTAINMENT SYSTEM: TRAVEL COST CALCULATOR

by

Haseefah Nazaahiah Binti Ahmad Moghni

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

pproved:

[illegible]

r. Noohul Basheer Bin Zain Ali
roject Supervisor

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

May 2011

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

Haseefah Nazaahiah Binti Ahmad Moghni

ABSTRACT

In-vehicle Infotainment (IVI) is a system that provides convenience for driver and passengers in a vehicle. The system has various applications that fulfill one need. In order to enrich the system, an application named Travel Cost Calculator (TCC) has been developed. TCC is an application that calculates the mileage between two locations, fuel cost estimation and toll fare that user has to pay. This application is built on Intel e-Menlow platform that is based on Intel Atom processor. TCC is implemented by using the Google Maps Application Programming Interface (API), Adobe Flash Builder, Hypertext Preprocessor (PHP) and MySQL database. In order to achieve the total up cost of a journey, the programming has to be done part by part. The driving direction approach is used to obtain the mileage between two locations. For the fuel cost estimation, the vehicle's mileage per gallon (MPG) needs to be known. It can be chosen from the car engine size (cc) options. Last but not least, the total toll fare could be achieved by using the automatic detection of toll station approach. Findings from these three parts will give the estimation of the total travel cost.

ACKNOWLEDGEMENTS

First of all, I would like to thank Allah the Almighty who has given me strength and blessings to complete my two semesters of Final Year Project. The utmost gratitude goes to my supervisor, Dr. Noohul Basheer Bin Zain Ali for his continuous guidance and support throughout my project. I would also like to give appreciation to Mr. Hau Tze-ming who has been a great teacher, and helping me getting through many challenges during this project. Besides that, I would like to thank Universiti Teknologi PETRONAS (UTP), especially the Electrical and Electronics Engineering Department whereby students are given the opportunity to experience doing research or design work. It is an opportunity for us, students to use the tools and techniques for problem-solving. Furthermore, the learning process is gained through hands-on experience. Last but not least, thank you to all my fellow friends and family for their love and support, I have successfully completed my Final Year Project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IVI	In-vehicle Infotainment
TCC	Travel Cost Calculator
GPS	Global Positioning System
API	Application Programming Interface
SDK	Software Development Kit
PHP	Hypertext Preprocessor
GUI	Graphical User Interface
CPU	Central Processing Unit
HDI	High Density Interconnect
TDP	Thermal Design Power
GMCH	Graphics Memory Controller Hub
POS	Point of Sale
LVDS	Low Voltage Differential Signaling
IDE	Integrated Development Environment
PHP	Hypertext Preprocessor

CHAPTER 1

INTRODUCTION

Chapter 1 gives the introduction of the Development of Infotainment System which will emphasize on Travel Cost Calculator (TCC). This chapter consists of the background study on IVI, the problem statement of IVI which includes problem identification and the significant of project, the objectives of the project, the scope of study, the relevancy of the project and the feasibility of the project within the scope and time frame.

1.1 Background Study

IVI is a rapidly growing and developing system that encompasses the use of smart phones, digital radio, Internet, DVD player, MP3-based systems, GPS navigation systems and roadside assistance services that provides not only the driver, but the individual needs of all passengers in their vehicles. "Infotainment" is generally referred to all digital applications including internal connectivity, navigation and location-based services, external communications, and radio that can be used by occupants of a vehicle. This technology acceleration, combined with the digitization of video, voice and data, and the evolution of the Internet, has brought IVI systems to a peak point. [1] – [3]

The "digital car" is becoming the natural and flawless extension of the digital home and office. The best feature that described IVI systems are as follow:

Navigation and location-based services:

The GPS that is based on 2D and 3D map not only display the route and directional information, but also points of interest and other location-based information and services.

Wireless connectivity:

There are several types of wireless embed on the system. Wireless personal area networks that is based on Bluetooth or Ultra-Wideband technology, wireless local area networks (WLAN) based on Wi-Fi (802.11) technology, and wide area networks (WANs) including WiMAX and 3G/4G cellular services.

Rear-seat entertainment:

The entertainment includes video-based games and on-demand video applications which are increasing the popularity of rear-seat multimedia entertainment systems, comprising systems that satisfy individual preferences of multiple users.

Seamless interoperability:

Interoperability is the ability of a system to grant services to and accept services from other systems. The services are then exchanged to enable them to work efficiently together. The interoperability in IVI includes video and audio content, MP3 audio and connections for cell phones and PDAs. [2]

1.2 Problem Statement

Inline with having the infotainment system in car, it would be great if the system have an application that is able to estimate the fuel consumption cost and toll cost such as Travel Cost Calculator (TCC). Instead of calculating the cost manually; the task can be done by using this application with just a click. TCC is an application that provides the mileage calculation between two locations, fuel consumption cost calculation and the toll cost determination. This will saved up a lot of time from doing the tedious calculations. Apart from that, the application is useful for users to plan their travel budget.

1.2.1 Problem Identification

Before any programming codes can be done, choosing the right hardware platform, operating system and software used is crucial. The need of knowing the hardware components is a must. The operating system used must also compatible with IVI system. Besides, learning the programming language such as Application Programming Interface (API) and Software Development Kit (SDK) are needed for the completion of the project. After completing those requirements, the process of programming the application can be done. The application is divided into three parts which are mileage calculation between two locations, fuel consumption cost calculation and toll cost determination. Therefore, a lot of research has to be done on the mileage calculation, fuel consumption cost calculation and also toll cost determination.

1.2.2 Significant of Project

The idea of the project is basically to build an application that is called TCC for IVI system. This application is built for user's convenience due to its fast response and flexibility. From this application, users can easily plan the travel budget before the journey. It is also designed for those who would like to claim the travel expenses if they have to go outstation within Malaysia.

For the mileage calculation between two locations, it is directly coded from Google Maps services where users get to know the estimated distance between two points that they inputted in the 'To' and 'From' boxes from the application. Some considerations are taken in order to perform the fuel consumption cost calculation such as vehicle's engine size (cc) so that the fuel economy can be calculated. The fuel economy is described using the car kilometers per liter (km/l). The fuel economy taken is the combination of the urban routes and extra urban routes. The average fuel economy is taken from several car manufacturers such as Perodua, Proton, Honda, Suzuki and Toyota. Apart from that, a database is built to store the toll fares from various toll entries to its exit. The summation of the fuel consumption cost and toll cost will result in the travel cost.

1.3 Objectives

The objective of this project is to build an application that provides benefits for users in terms of the mileage calculation between two locations, fuel consumption cost calculation and toll cost determination. This application is also concern for those who needs to claim their travel expenses and also for those who would like to plan their journey. Apart from that, the objective of the project is to program an automatic detection of the nearest toll station.

1.4 Scope of Study

The scope of study is divided into two sections which are hardware side and the software side. For hardware side, the IVI platform and the components reside on it must be recognized. The operating system used for this application is one of the Linux distributions. Therefore, studies on Linux shell languages must be mastered. The software involves around programming using Google Maps API, Adobe Flash Builder, Hypertext Preprocessor (PHP) and MySQL Database. Adobe Flash Builder is a development tool used to build this application.

Overall, the project scope can be divided into four stages which are platform assembly, requirement analysis and design, coding and testing. Platform assembly stage is where the hardware platform is setup together with the operating system. Next stage is the most important part where the project has to be planned and carried out systematically. Coding is where step-by-step procedures are made to program the TCC. Last but not least is the testing part. This is where the programming codes are tested and improvements are made on this stage. [4]

1.5 The Relevancy of Project

The project is relevant to the study of Structured Programming and Pervasive Systems as it focuses on programming the application using Adobe Flash Builder that has minimum basic of C language. The project is one of the recent technologies that are used in the digital car or namely IVI system. Even though the technology has already existed, but the project will prolong the ability of IVI system.

1.6 Feasibility of the Project within the Scope and Time Frame

The project is conducted starting with the collection of related materials based on internet resources, internet journals and technical papers specifically on IVI, Google Maps API, Adobe Flash Builder and TCC itself. Research has been done from time to time as to get a better understanding on the subject. The project is then focus on programming the application using Adobe Flash Builder software. Debugging is done in order to illustrate the result of the codes written. Based on the activities stated above, 14 weeks were given in the first semester for the researches and studies to be done as well as testing the source codes whereas another 14 weeks are set aside for the finalization of the design.

CHAPTER 2

LITERATURE REVIEW

This chapter contains the literature review that has been done for the research. It consists of hardware, operating system and software sections. The hardware section explains about Intel eMenlow platform. The operating system section explains briefly about Ubuntu operating system. The software section describes about the Google Maps API, Adobe Flash Builder, PHP and MySQL Database. Plus, there are also theories on Specific Data Model of Smart Fuel Consumption Cost Estimator and Nearest Point Calculation using Haversine Formula.

2.1 Hardware

There is a hardware tool used in this project which is a motherboard or platform called eMenlow. It is produced by Intel.

2.1.1 eMenlow Platform

Intel eMenlow platform is one of the hardware platforms generated by Intel itself for embedded products. Intel eMenlow platform is based on the Intel Menlow platform. The small letter 'e' from eMenlow stands for embedded. This platform is based on Intel Atom processor Z5xx series which operated at 1.6GHz speed. The Central Processing Unit (CPU) is based on Low Power Microarchitecture on 45nm technology. It is in an ultra small 13x14mm High Density Interconnect (HDI) package or 22x22mm. The performance and power optimization are for small form factor that is ultra low-power embedded platform that has Thermal Design Power (TDP) 2 or 2.2W. Other advanced feature for Intel Atom CPU is it has Hyper-threading technology. Hyper-threading technology allows user to run multiple demanding applications and one time and keep the system more protected, efficient and organisable. [4] – [9]

The system chipset is US15W System Controller Hub (SCH). It consists of integrated Graphics Memory Controller Hub (GMCH) and I/O Controller Hub (ICH) on a single chip. It also has an ultra-low power integrated graphics with 2D and 3D hardware accelerator. Integrated High Definition Video Codec is also part of US15W chipset. Last but not least is expansion capability with most widely adopted USB 2.0 and PCI-E standards. [9]

The compact platform provides benefits in low-power system and handheld for mobile devices that are used in various applications such as portable Point of Sale (POS), mobile kiosks, mobile gaming devices and digital signage. The low-power platform has a combined thermal design power fewer than 5 Watts. [5]

Intel eMenlow platform supported a single channel 24-bit Low Voltage Differential Signaling (LVDS) display. LVDS has a high speed and low power data transmission for display. Type II Compact Flash socket and mini SD card slot are also supported by this platform. Compact Flash is a mass storage device format used in handy electronic devices. Type II is 5mm thick and can draw up to 500mA. Apart from that, full hardware acceleration of H.264, MPEG2, VC1 and WMV9 is supported [6] - [7]. Other specifications will be described in the Table 1.

Table 1: Specification of Intel eMenlow Platform [7]

CPU	On board Intel® Atom (eMenlow) 1.6GHz or 1.1GHz with a 533/400MHz FSB
System Chipset	Intel® US15W
BIOS	AMI BIOS
System Memory	1 x 200-pin single channel DDR2 SO-DIMM 533MHz up to 1GB
Ethernet	10/100Mbps Realtek RTL8102E Ethernet Chipset (PCIe Interface)
I/O Interface	3 x RS-232 (by pin header) 1 x RS-232/422/485 (by pin header) 5 x USB 2.0 (4 Host USB + 1 Client USB ;by pin header) 1 x IDE (44-pin header) 1 x pin header for KB/MS
Expansion Digital I/O	8-bit digital I/O, 4-bit input/ 4-bit output
Super I/O	SMSC 3114NU
Display Interface	24-bit single channel LVDS VGA support by LVDS to VGA Module (LVDS-VGA-R10)
Audio	HD interface (Realtek ALC883) by Audio Kit
SSD	1x CF type I/II 1x micro SD
Watchdog Timer	Software programmable supports 1~255 sec. System reset
Power Supply	5V only, AT/ATX support
Temperature	Operating: 0°C ~ +60°C
Humidity	Operation: 5%~95% non condensing
Dimension	3.755"(95.89mm) x 3.550" (90.17mm)

2.2 Ubuntu Operating System

Ubuntu is a Debian-based Linux operating system. It is not only focuses on desktops and laptops users, but also mobile and embedded devices. The Ubuntu distribution for the embedded devices is called Ubuntu Mobile. Besides, it is a cross-platform operating system. Ubuntu can be installed on any hardware platform including Intel eMenlow platform.

It is an open source project. There are a lot of features that provide encouragement to use Ubuntu as the operating system for the project. Ubuntu is a user-friendly operating system besides it has various applications installed. Ubuntu is a quick response operating system. It has a fast boot up time. [10]

2.3 Software

There are a few softwares used in order to build this project. The softwares used are Google Maps API, Adobe Flash Builder, Hypertext Preprocessor (PHP) and MySQL Database.

2.3.1 Google Maps API

Google Maps has a wide range of API that let developer embed the robust functionality and usefulness of Google Maps into their own website and applications, and overlay their own data on top of the Google Maps API family. The family consists of maps JavaScript API, Maps API for Flash, Google Earth API, Web Services and Maps Data API. Among the families, JavaScript API and Maps API for Flash are very useful for this project. [11]

The Google Maps can be embedded on any web pages by using the Google Maps JavaScript API. Version 3 of this API is particularly design to be faster and more relevant to mobile devices such as Android-based devices and the iPhone. The API provides a number of utilities for manipulating maps and adding content to the map through a range of services. The JavaScript Maps API V3 provides a free web service. [12]

The Google Maps API for Flash lets Flex developers embed Google Maps in Flash applications. This API provides the functionality similar to the Google Maps JavaScript API [13]. The only difference is that developers have the option of choosing the interface of the application; either web based or Graphical User Interface (GUI) based.

In order to start using the Google Maps API, developer has to sign up for a Google Maps API key. Then only, the developer is able to use the APIs. Next the developer has to download the Google Maps API SDK, go through the developer's guide and consult the API reference.

2.3.2 Adobe Flash Builder

Adobe Flash Builder (formerly Adobe Flex Builder) software is designed to assist software developers develop cross-platform internet-based applications rapidly using the open source Flex framework. It takes in support for intelligent coding, debugging and visual design. It also has the feature of powerful testing tools that speed up development and lead to higher performing applications. [14]

Adobe Flash Builder has a few excellent features. It has powerful coding tools such as Eclipse based Integrated Development Environment (IDE) that includes editors for MXML, the ActionScript language, and CSS, as well as syntax colouring, statement completion, code-collapse, interactive step-through debugging, and automatic generation of common code [14]. The tools are very useful for building TCC.

Adobe Flash Builder gives support for Adobe AIR. Adobe AIR lets user quickly develop the application for GUI using the same skills and codebase used to build application for the web based. [14]

The codes in MXML is able to be restructured by renaming all references to a class, method or variable. This is due to the refactoring feature. ASDoc is used to display the comments in MXML and ActionScript editors. Adobe Flash Builder visually design and preview user interface layout, appearance, and behaviour using a rich library of built in components. [14]

It also has data-centric development. Introspect Java, PHP, Adobe ColdFusion, REST, and SOAP services are used to display methods and properties in the new Data/Service Explorer. This data-centric development is useful in building a database for toll cost which is part of the project. An easy drag-and-drop approach is used to bind methods to UI components. [14]

2.3.3 Hypertext Preprocessor

PHP is a broadly used for general purpose scripting language that is especially suited for Web development. PHP is a server-side language, cross-platform, HTML-embedded scripting language. PHP's syntax is mostly borrowed from C, Java and Perl with a few exclusive PHP-specific features thrown in. The language target is to let web developers write dynamically and generated pages quickly. The other benefit is that it makes a lot of easier to manage large web sites by placing all components of a web page in a single html file. [14] – [15]

As stated earlier, PHP is a cross-platform language meaning that the core PHP code can be extended to all of PHP's libraries and all code written in PHP. PHP includes built-in support for many databases (including Access, LDAP, Oracle, and MSSQL), networking support, zip archiving and an excellent set of built-in functions. PHP is easy to learn as the syntax structure borrows heavily from C. PHP is also the oldest HTML-embedded scripting language, giving it a head start off on all the others. [14] – [15]

In order to proceed with the project, some efforts were made by studying on the basic tutorials of PHP. Since most of the syntax and methods are borrowed from C language, the time taken to study on PHP is lesser. The studies include:

1. Introduction
2. Displaying Information and Variables
3. If, Else and Then Statements
4. Loops and Arrays
5. PHP with Forms [16]

2.3.4 MySQL Database

MySQL is the database chosen to store the data used in this project. There are quite a number of tables needed in the database such as to store the locations of the toll stations and to store the toll fares at each exit. MySQL are chosen as it is fast response, high reliability, east of use and cost savings. Besides, MySQL has been the most popular database used around the world [17]. The PHP codes are used to store the data in MySQL database. These two softwares have been a lot easier with the phpMyAdmin tool.

2.4 Specific Data Model of Smart Fuel Consumption Cost Estimator

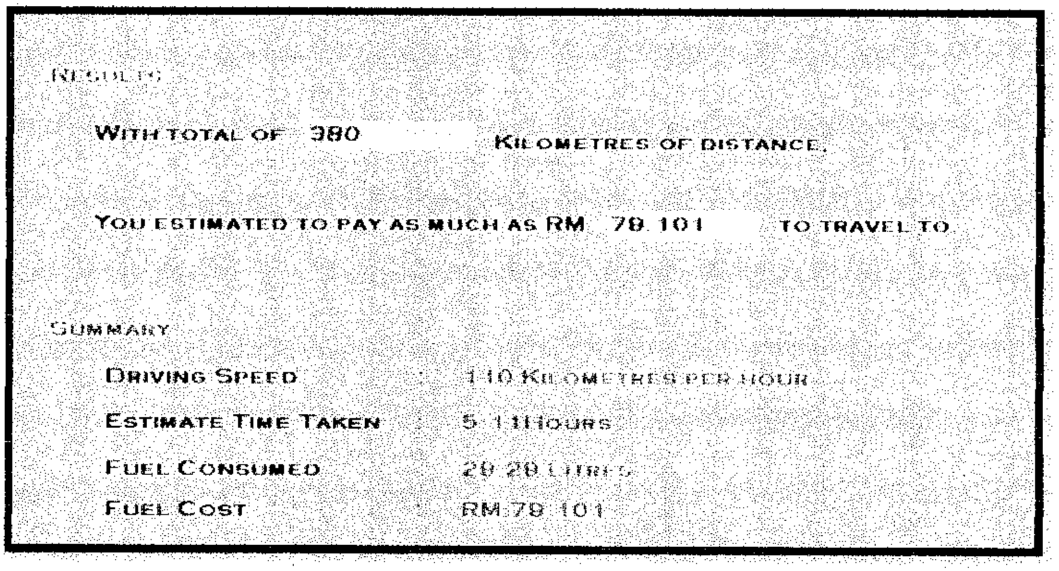
The Smart Fuel Consumption Cost Estimator (SFCCE) is automated software that calculates the total amount of fuel cost for a car to travel based on the distance given, driving speed, weight capacity and current market fuel price. Users will obtain the summary of the journey by using SFCCE and it is considered as the best solution to estimate the minimum cost towards the desired destination. [23]

In order to accomplish TCC, SFCCE is used as a reference to develop the fuel consumption cost calculation. SFCCE calculates based on the user input data and provide a result on fuel consumption to reach to the destination. In particular, estimating methods represent important processes for the fuel consumption cost calculation. SFCCE compares several results to produce the best result by using mathematical based estimating methods. [23]

SFCCE is based on several factors such as vehicle's engine (cc), transmission type, driving style, average speed on road, passenger and luggage weight capacity, air conditional status, vehicle service and distance covered. SFCCE calculates by converting each of the factors into a certain value to produce an estimation of fuel consumed. The amount of fuel consumed will be multiplied by the current fuel price in Malaysia. Thus, the estimated amount of cost can be displayed. [23]

SFCCE module is to create an engine to provide an estimation of fuel consumed. The methods used are one of the mathematical estimations that called Parametric Estimating methods. Statistical analysis is done on the data to find correlations between cost drivers and other system parameters. The analysis produces cost equations or cost estimating. [23]

Figure 1 shows the result of SFCCE where the application will summarize the result of fuel consumption cost and other related information.


RESULTS

WITH TOTAL OF 380 KILOMETRES OF DISTANCE.

YOU ESTIMATED TO PAY AS MUCH AS RM 79.101 TO TRAVEL TO.

SUMMARY

DRIVING SPEED	:	110 KILOMETRES PER HOUR
ESTIMATE TIME TAKEN	:	5.11HOURS
FUEL CONSUMED	:	29.29 LITRES
FUEL COST	:	RM 79.101

Figure 1: Result and Summary from SFCCE [23]

2.5 Nearest Point Calculation using Haversine Formula

Haversine formula is widely used in navigation. The great-circle distances between two points represented by the latitudes and longitudes can be calculated by using the Haversine formula, assuming that the earth can be represented by a perfect sphere [25]. By referring to Figure 2, let x refer to latitude and y refer to longitude. Therefore, the great circle distance, d between $P_1=(x_1, y_1)$ and $P_2=(x_2, y_2)$ is to be calculated [27]. The latitudes and longitudes are assumed to be in radians. Also, R indicates the Earth's radius which is 6371km or 3959 miles and a indicates the chord length between the two points.

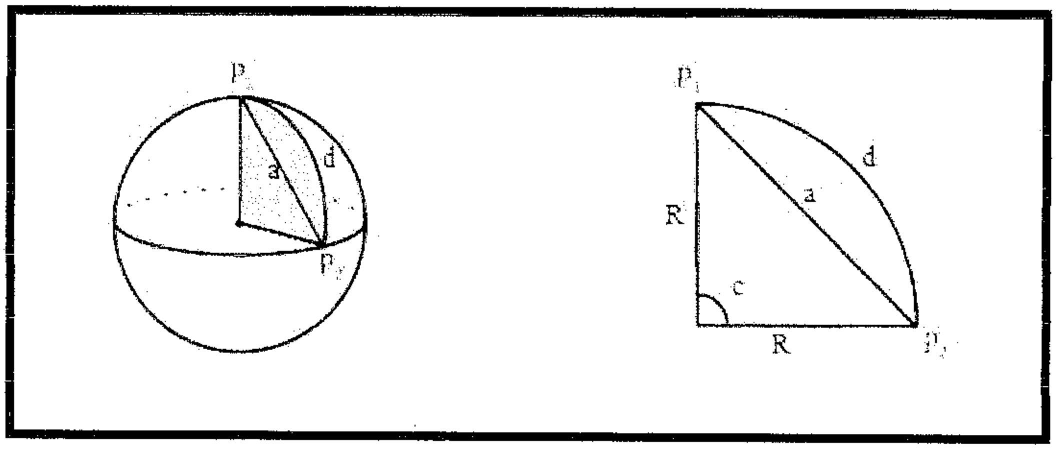


Figure 2: Great circle and tunnel through distances [27]

The general great circle distance, *d* or arc length formula is shown below:

$$d = R * c \qquad \text{Eq. (1)}$$

Whereby, *c* is the great circle distance in radians,

R =6371 km=3959 miles

Haversine formula states that:

$$a = \sin^2\left(\frac{\Delta x}{2}\right) + \cos x_1 * \cos x_2 * \sin^2\left(\frac{\Delta y}{2}\right) \qquad \text{Eq. (2)}$$

$$c = 2 * a\tan 2\left(\sqrt{a}, \sqrt{1-a}\right) \qquad \text{Eq. (3)}$$

Whereby, $\Delta x = x_2 - x_1$,

$\Delta y = y_2 - y_1$,

a is square of half the chord length,

$$a\tan 2\,(x, y) = \tan^{-1}\left(\frac{y}{x}\right) \qquad \text{Eq. (4)}$$

Therefore, the great circle distance can be calculated using the above information. The Haversine formula will more accurate for small distances compared to Cosine formula [25].

CHAPTER 3

METHODOLOGY

This chapter describes about the methodology used in the project. Procedure identification is done in order to achieve the aim of the project. A project work flow diagram is used to describe the project activities. The tools and equipments needed for the accomplishment of the project is then explained in this chapter.

3.1 Procedure Identification

In order to achieve the aim of the project, some research had been done by internet resources and technical papers. For the first phase, gathering information needs to be done on the hardware, software, operating system and TCC itself. Research on eMenlow platform is done for the hardware section. For software, Google Maps API, Adobe Flash Builder, PHP, MySQL are taken into consideration. The Ubuntu capabilities are studied in order to build the application on mobile-based operating system. For the TCC application, the mileage calculation between two locations, fuel consumption cost calculation and toll cost determination are reviewed and implemented in order to achieve the objectives.

After all the studies have been done, the next phase will be platform assembly whereby the hardware is setup together with the operating system. For this project, Ubuntu is installed on the eMenlow platform. After the installation completed, Adobe Flash Builder is installed on it as the development tool for building the application.

Then, next phase is the requirement analysis and design. The objective of this phase is to plan and carry out the project systematically. Firstly, the project is designed for mileage calculation between two locations. The result is then used in the second part which is fuel consumption cost calculation. Other information needed such as current market fuel price and fuel economy in order to come out with the calculation. Then, the toll cost can be determined by using the user select input approach. An automatic detection of the nearest toll station is then design to increase the benefits of TCC. After all of these results are obtained, the total up cost is calculated.

Next is the coding phase where step-by-step programming can be done after having some knowledge about the project. First step is programming the Driving Directions. Driving Directions will direct user to the destination using step-by-step method. From Driving Directions, user will get to know the total distance encountered and time taken to arrive to the destination. Then, the coding part is continued with the fuel consumption cost calculation. For this part, few factors have been taken into account to make the calculation accurate such as the vehicle's engine size and the average fuel economy. The vehicle's engine size is used to determine the fuel economy of the vehicle. The average fuel economy is calculated from several car manufacturers. The fuel economy is the combination of urban routes and extra urban routes. Next, a database consists of the toll entries, toll exits, coordinates and toll fares are needed in order to determine the toll cost. This database which is built using PHP and MySQL is then parsed to be in XML format. The user select input approach is implemented by using the knowledge of creating two related combo boxes [26]. The travel cost is calculated based on the values retrieved from previous parts.

Last but not least is the testing phase. In this phase, all coding that has been program earlier are tested. As to improve the project, an automatic detection of the nearest toll station is then programmed by using the advantage of the SQL *SELECT* statement. The automatic detection of the nearest toll station is then tested to meet the requirement of the designed application. A lot of trials and errors are done in order to come out with the best result.

3.2 Project Activities

Figure 3 describes the flow chart of the project and the implementation throughout the time.

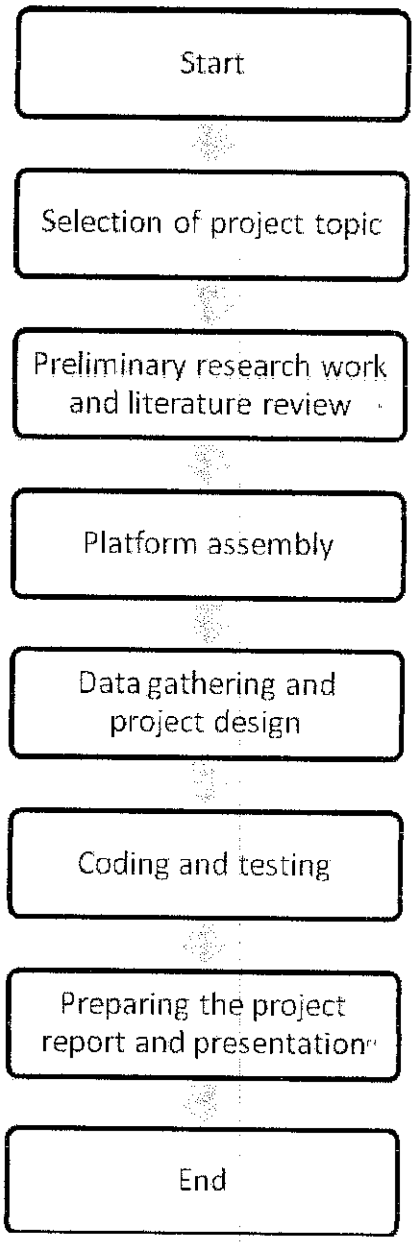

Start
Selection of project topic
Preliminary research work and literature review
Platform assembly
Data gathering and project design
Coding and testing
Preparing the project report and presentation
End

Figure 3: Project Work Flow

The first step is to select the project topic, conduct the preliminary researches and literature review. The literature review for mileage calculation between two points, fuel consumption cost calculation and toll cost determination is completed within this step. Along with studying on the literature review, the eMenlow platform is complete with the components and software needed.

The next step is data gathering and the project design. As mentioned earlier, the project is divided into three parts which are mileage calculation between two points, fuel consumption cost calculation and toll cost determination. Additional feature is then added to improve the application. The automatic detection of the nearest toll station is then designed.

The next step is the coding and testing. This will be described in the Results and Discussion chapter. The following step is preparing the report that needs to be submitted to the Final Year Project Committee. There is also an oral presentation which will take place towards the end of the semester.

3.3 Gantt Chart

The Gantt chart is provided together with the report in the Appendix A. The Gantt chart is a guideline for the project timeline. It can be changed from time to time depending on certain circumstances.

3.3.1 Project Milestone

As depicted in the Gantt chart itself, the first two weeks of the semester is the time to port the application on the platform. Week 3 and Week 4 is allocated for create and populate the toll cost database. After the database is ready, the results are parsed into XML format. The results are then combined and total up cost is implemented. In the mean time, three weeks are allocated for research and implemented the automatic detect toll station. Submission of Progress Report is done during Week 8. Any other improvements could be made in two weeks time before Pre-EDX. In Week 12 is the submission of Interim Draft Report while in Week 13 is the submission of Final Report and Technical Report. Last but not least is the oral presentation which will be held in Week 14.

3.4 Tools and Equipments Required

Hardware and software tools are needed for the accomplishment of the project. As for hardware tools, an eMenlow platform is used together with the PCI Express network card. This network card is needed for the internet connection as the application requires the connectivity to the internet.

There are needs for software such as Google Maps API and Adobe Flash Builder. These are the main software used to program the application. Ubuntu 10.10 is used as the operating system. Apart from that, phpMyAdmin tool is needed to build the database for toll cost determination. This tool consists of PHP and MySQL database.

CHAPTER 4

RESULTS AND DISCUSSION

This chapter consists of the results and the discussion of the project. There are two main sections which are Data Gathering and Analysis and also Modelling. The Data Gathering and Analysis consists of map display and driving directions, fuel consumption cost calculation, toll cost calculation, and automatic detection of nearest toll station. The Data Gathering and Analysis section explains how the data are retrieved in order to obtain the results. The Modelling section explains the design of each parts of the application.

4.1 Data Gathering and Analysis

This section explains how the data are retrieved in order to obtain the results. The data is needed in order to design the application.

4.1.1 Map Display and Driving Directions

In order to accomplish this project, the application is built using Google Maps API on Adobe Flash Builder as the interface. The knowledge of this two programming languages are needed in order to achieve the first objective of this project which is to calculate the mileage between two locations. The programming knowledge is used to display the map in the application. Then, the driving directions method is used in order to calculate the mileage between two locations.

4.1.2 Fuel Consumption Cost Calculation

For fuel consumption cost calculation, users have the options to get the fuel economy or the kilometers per liter (km/l) value by selecting the vehicle's engine size. Some of the vehicle manufacturers are taken into account such as Perodua, Proton, Kia, Honda, Suzuki and Toyota. From that, certain models from the vehicle manufacturers have been listed down and the fuel economy is categorized according to the vehicle's engine size. Then, the average of the fuel economy is taken as the km/l reading. The fuel economy and the vehicle's engine size are tabulated as below:

Table 2: Fuel Economy Based on Vehicle's Engine Size

Vehicle's Engine Size (cc)	Fuel Economy (km/l)
1000cc and less	22.53
1100cc – 1500cc	20.70
1600cc – 1900cc	18.20
2000cc and more	14.84

When the fuel economy is known, the cost per kilometer (cpkm) can be calculated using Equation 5:

$$cpkm\ (RM/km) = \frac{Fuel\ Price\ (RM)}{Fuel\ Economy\ (km/l)} \qquad \text{Eq. (5)}$$

When the cost per kilometer is known, the estimated fuel consumption can be calculated using Equation 6:

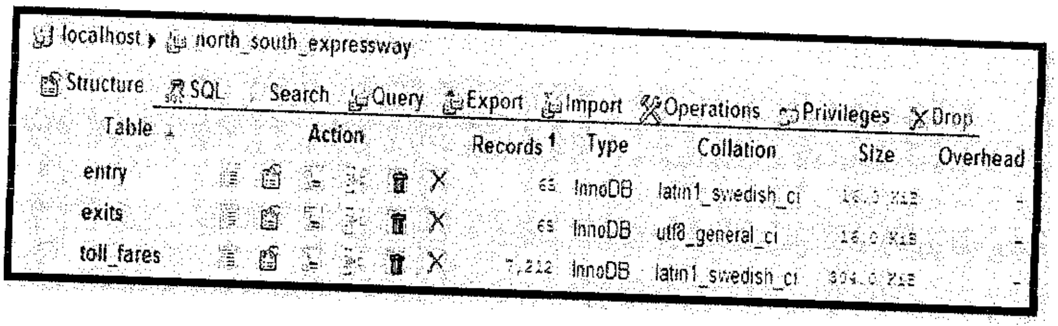
$$Fuel\ Cost\ (RM) = Total\ Distance(km) \times cpkm\ (RM/km) \qquad \text{Eq. (6)}$$

4.1.3 Toll Cost Calculation

In order to accomplish the Toll Cost Calculation, extra tools are needed. PhpMyAdmin and MySQL are needed to build the database for the toll stations. The scope for the toll stations has been narrowed down to North-South Expressway and the toll rate is also focused on Class 1 vehicle which is vehicles with two axles with three or four wheels excluding taxis [*20*].

4.1.3.1 Database Development

To start off, the coordinates for each toll station along the North-South Expressway are determined by using [21]. Next, using [22] and the toll station locations, the database is built up. The database name is given as North-South Expressway. Three tables are built separately. The first table consists of the toll entries with their latitudes and longitudes. The same goes for the second table which contains the information for the toll exits. Last but not least is the toll fares table whereby it is the combination of the first and second tables. The information contains are the toll entries, toll exits and the toll fares to each exits respectively. The structure of the database is shown in Figure 4.


localhost ▸ north_south_expressway

Structure SQL Search Query Export Import Operations Privileges Drop

Table	Action	Records	Type	Collation	Size	Overhead
entry		65	InnoDB	latin1_swedish_ci	16.0 KiB	-
exits		65	InnoDB	utf8_general_ci	16.0 KiB	-
toll_fares		7,222	InnoDB	latin1_swedish_ci	304.0 KiB	-

Figure 4: Structure of Database

The tables are then filled with data contains the information needed. The data that is taken from Plus Expressway Toll fare is first saved from excel spreadsheet into .csv (comma-separated values) format. This can be done using Microsoft Excel. The .csv file is then checked for validity through a text editor. The file is edited to make sure there is no whitespace left or extra commas. Then only the data can be imported. Figure 5 shows how the data is imported.

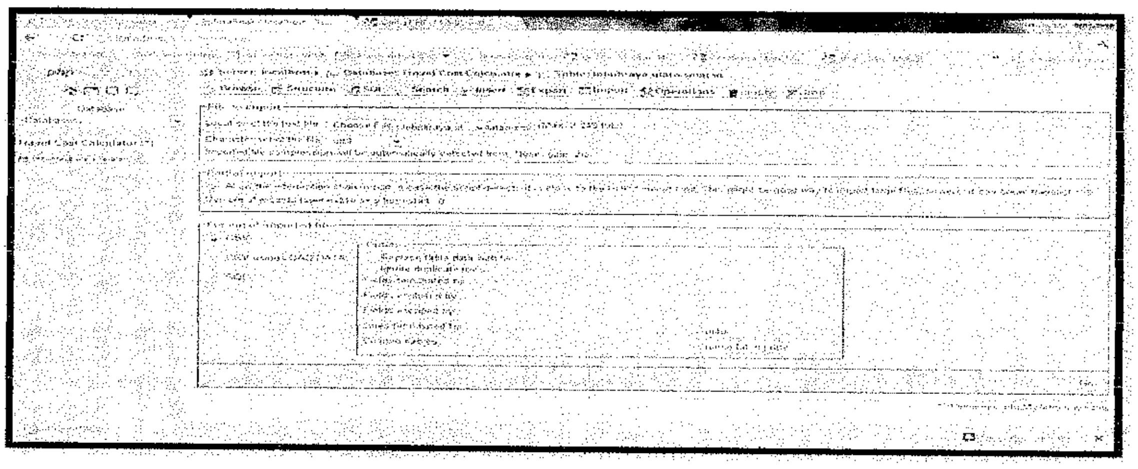
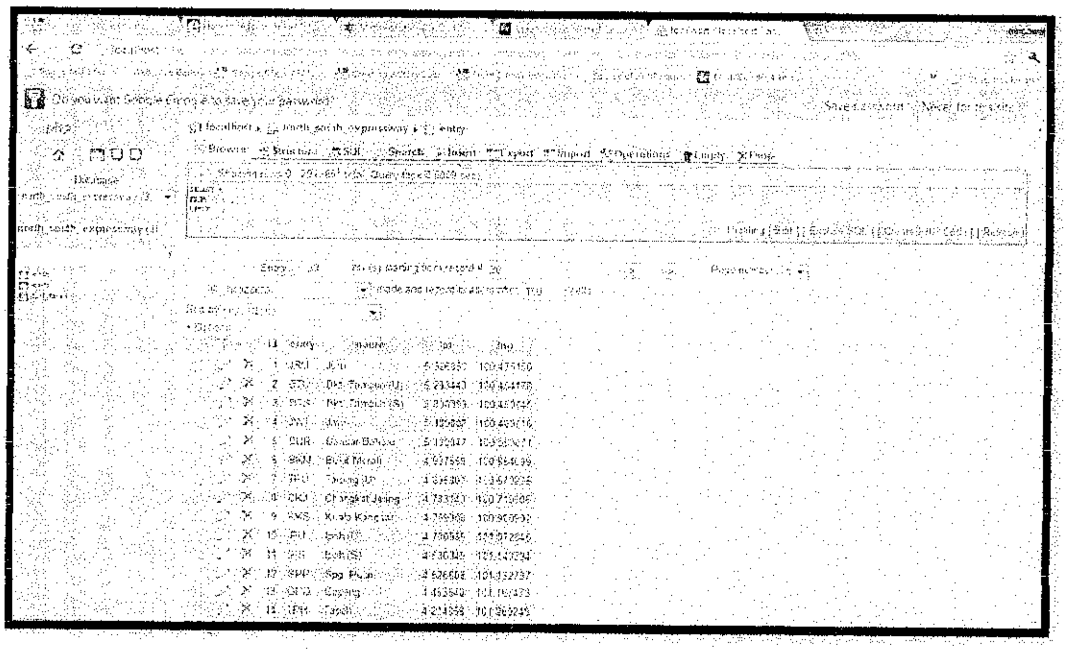


Figure 5: Import Data from .csv File

Then, if there is no error, the database can already be generated. The database is displayed in a webpage where the server is localhost. Figure 6 shows how the database looks like in a webpage.



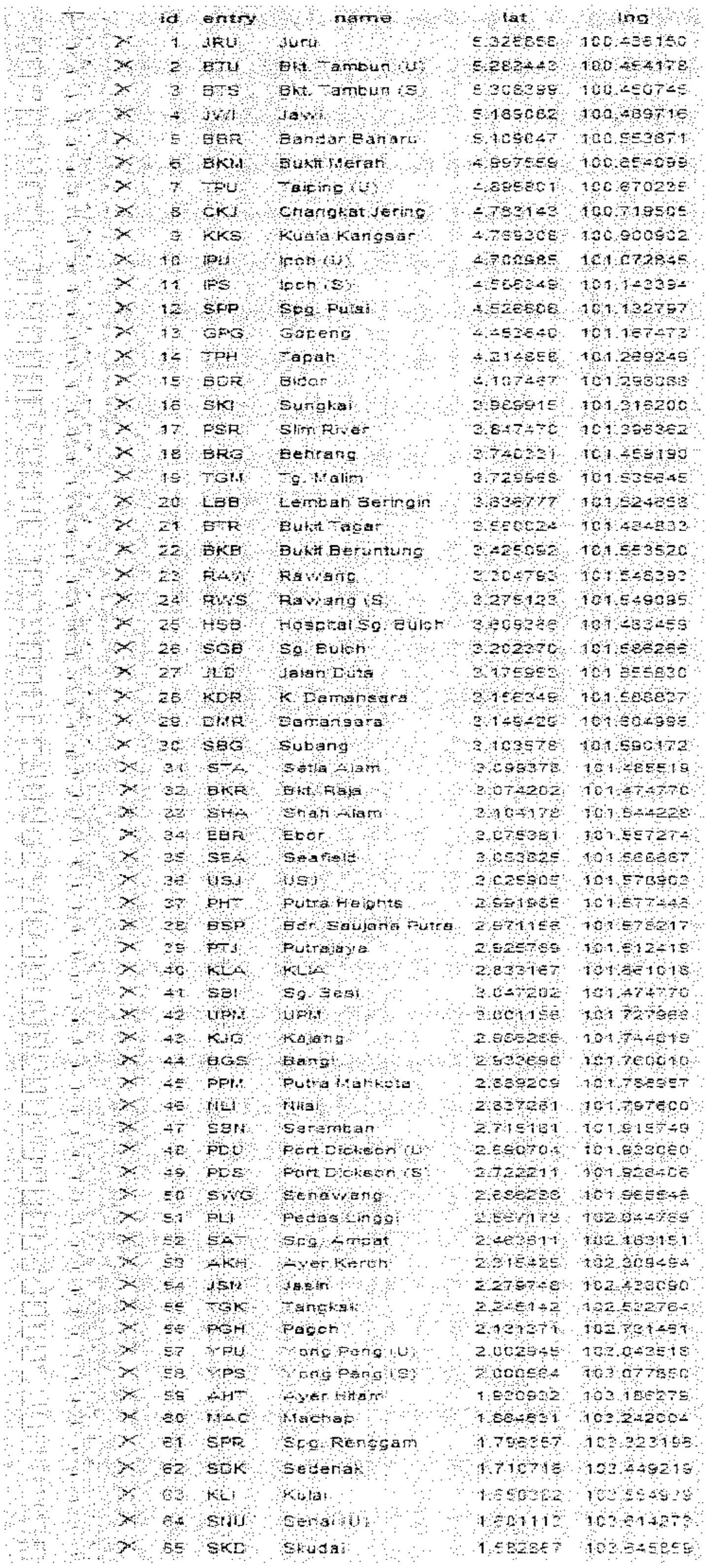
	id	entry	name	lat	lng
✕	1	JRU	Juru	5.326350	100.475150
✕	2	[illegible]	Bkt Tambun (U)	5.283440	100.464178
✕	3	[illegible]	Bkt Tambun (S)	[illegible]	[illegible]
✕	4	JWI	Jawi	[illegible]	[illegible]
✕	5	[illegible]	[illegible]	[illegible]	[illegible]
✕	6	[illegible]	Bukit Merah	4.937559	[illegible]
✕	7	[illegible]	Taiping (U)	[illegible]	[illegible]
✕	8	CKJ	Changkat Jering	4.733720	[illegible]
✕	9	KKS	Kuala Kangsar	[illegible]	[illegible]
✕	10	[illegible]	Ipoh (U)	[illegible]	[illegible]
✕	11	[illegible]	Ipoh (S)	[illegible]	[illegible]
✕	12	SPP	Spg Pulai	4.526606	[illegible]
✕	13	[illegible]	Gopeng	4.453540	[illegible]
✕	14	[illegible]	Tapah	[illegible]	[illegible]

Figure 6: Snapshot of North-South Expressway Database

4.1.3.2 Result of Database

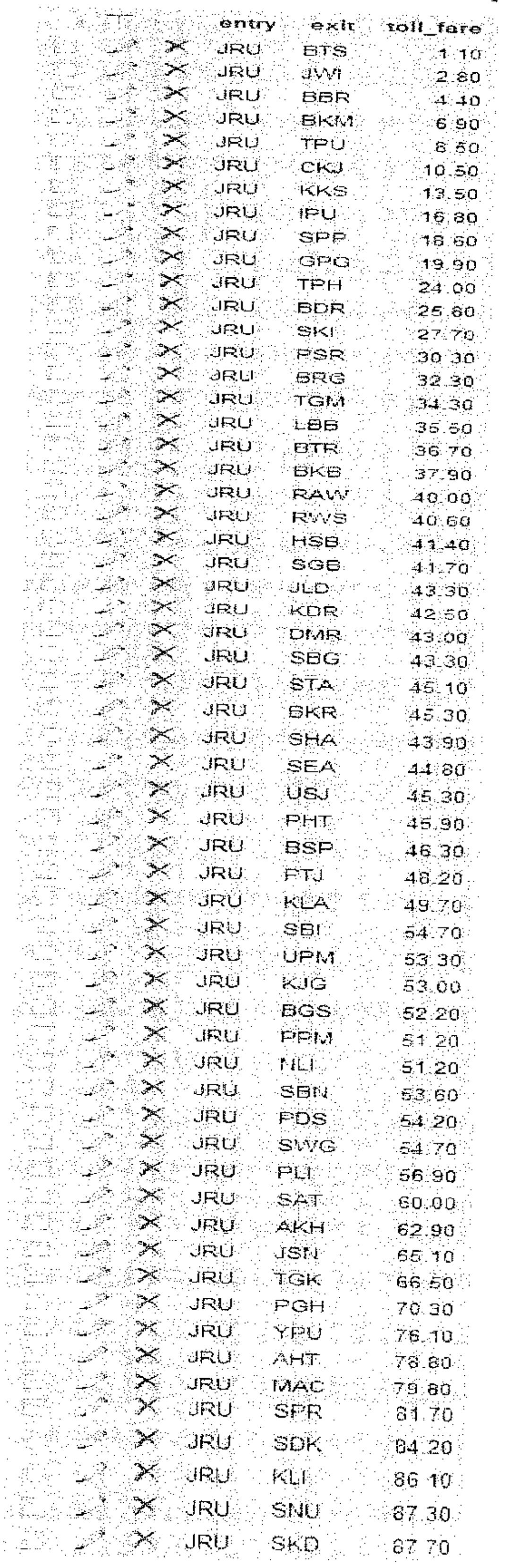
The full result consists of 65 toll stations along the expressway. Since the table for toll entries and toll exits are identical, Table 3 depicts the results of the table whereas Table 4 shows the partial results of toll fares, from Juru to all of its exits.

Table 3: Entry and Exit Data for North-South Expressway



id	entry	name	lat	lng
1	JRU	Juru	5.325858	100.436150
2	BTU	Bkt. Tambun (U)	5.282443	100.454178
3	BTS	Bkt. Tambun (S)	5.308399	100.450745
4	JWI	Jawi	5.169062	100.489716
5	BBR	Bandar Baharu	5.109047	100.553871
6	BKM	Bukit Merah	4.997559	100.654099
7	TPU	Taiping (U)	4.895801	100.670225
8	CKJ	Changkat Jering	4.783143	100.719505
9	KKS	Kuala Kangsar	4.799308	100.900902
10	IPU	Ipoh (U)	4.700985	101.072845
11	IPS	Ipoh (S)	4.566349	101.142394
12	SPP	Spg. Pulai	4.526808	101.132797
13	GPG	Gopeng	4.452640	101.167473
14	TPH	Tapah	4.214858	101.289249
15	BDR	Bidor	4.107487	101.298088
16	SKI	Sungkai	3.969915	101.316200
17	PSR	Slim River	3.847470	101.396362
18	BRG	Behrang	3.740221	101.459190
19	TGM	Tg. Malim	3.729568	101.535845
20	LBB	Lembah Beringin	3.636777	101.524858
21	BTR	Bukit Tagar	3.550024	101.484832
22	BKB	Bukit Beruntung	3.425092	101.553520
23	RAW	Rawang	3.304793	101.546393
24	RWS	Rawang (S)	3.275123	101.549095
25	HSB	Hospital Sg. Buloh	3.209249	101.483459
26	SGB	Sg. Buloh	3.202270	101.586288
27	JLD	Jalan Duta	3.175993	101.655830
28	KDR	K. Damansara	3.156349	101.588827
29	DMR	Damansara	3.149429	101.604995
30	SBG	Subang	3.103578	101.590172
31	STA	Setia Alam	3.095378	101.485519
32	BKR	Bkt. Raja	3.074202	101.474770
33	SHA	Shah Alam	3.104178	101.544228
34	EBR	Ebor	3.075381	101.557274
35	SEA	Seafield	3.053825	101.566667
36	USJ	USJ	3.025905	101.578902
37	PHT	Putra Heights	2.991985	101.577442
38	BSP	Bdr. Saujana Putra	2.971156	101.575217
39	PTJ	Putrajaya	2.925769	101.612419
40	KLA	KLIA	2.833167	101.661018
41	SBI	Sg. Besi	3.047202	101.474770
42	UPM	UPM	3.001156	101.727969
43	KJG	Kajang	2.965286	101.744819
44	BGS	Bangi	2.932698	101.760010
45	PPM	Putra Mahkota	2.889209	101.758957
46	NLI	Nilai	2.827261	101.797600
47	SBN	Seremban	2.715181	101.915749
48	PDU	Port Dickson (U)	2.590704	101.823080
49	PDS	Port Dickson (S)	2.722211	101.928406
50	SWG	Senawang	2.696236	101.965546
51	PLI	Pedas Linggi	2.557173	102.044769
52	SAT	Spg. Ampat	2.462511	102.163151
53	AKH	Ayer Keroh	2.315425	102.309494
54	JSN	Jasin	2.279748	102.423090
55	TGK	Tangkak	2.245142	102.522764
56	PGH	Pagoh	2.121271	102.721451
57	YPU	Yong Peng (U)	2.002945	102.042518
58	YPS	Yong Peng (S)	2.000564	103.077850
59	AHT	Ayer Hitam	1.920932	103.186275
60	MAC	Machap	1.884831	103.242004
61	SPR	Spg. Renggam	1.798357	103.223195
62	SDK	Sedenak	1.710718	103.449219
63	KLI	Kulai	1.650362	103.594929
64	SNU	Senai (U)	1.601112	103.614273
65	SKD	Skudai	1.582867	103.645859

Table 4: Toll Fares Data for North-South Expressway



entry	exit	toll_fare
JRU	BTS	1.10
JRU	JWI	2.80
JRU	BBR	4.40
JRU	BKM	6.90
JRU	TPU	8.50
JRU	CKJ	10.50
JRU	KKS	13.50
JRU	IPU	16.80
JRU	SPP	18.60
JRU	GPG	19.90
JRU	TPH	24.00
JRU	BDR	25.80
JRU	SKI	27.70
JRU	PSR	30.30
JRU	BRG	32.30
JRU	TGM	34.30
JRU	LBB	35.50
JRU	BTR	36.70
JRU	BKB	37.90
JRU	RAW	40.00
JRU	RWS	40.60
JRU	HSB	41.40
JRU	SGB	41.70
JRU	JLD	43.30
JRU	KDR	42.50
JRU	DMR	43.00
JRU	SBG	43.30
JRU	STA	45.10
JRU	BKR	45.30
JRU	SHA	43.90
JRU	SEA	44.80
JRU	USJ	45.30
JRU	PHT	45.90
JRU	BSP	46.30
JRU	PTJ	48.20
JRU	KLA	49.70
JRU	SBI	54.70
JRU	UPM	53.30
JRU	KJG	53.00
JRU	BGS	52.20
JRU	PPM	51.20
JRU	NLI	51.20
JRU	SBN	53.60
JRU	PDS	54.20
JRU	SWG	54.70
JRU	PLI	56.90
JRU	SAT	60.00
JRU	AKH	62.90
JRU	JSN	65.10
JRU	TGK	66.50
JRU	PGH	70.30
JRU	YPU	76.10
JRU	AHT	78.80
JRU	MAC	79.80
JRU	SPR	81.70
JRU	SDK	84.20
JRU	KLI	86.10
JRU	SNU	87.30
JRU	SKD	87.70

4.1.3.3 XML Parsing

The results from the toll fares table are then parsed to XML format. First, the parseToXML function is used to encode a few special entities to be XML friendly. For example, the character < is replaced by *&lt;* and the character > is replaced by *&gt;*. Other entities can be found in the Figure 7.

```
function parseToXML($htmlStr)

{
$xmlStr=str_replace('<','&lt;',$htmlStr);
$xmlStr=str_replace('>','&gt;',$xmlStr);
$xmlStr=str_replace('"','&quot;',$xmlStr);
$xmlStr=str_replace("'",'&#39;',$xmlStr);
$xmlStr=str_replace("&",'&amp;',$xmlStr);
return $xmlStr;
}
```

Figure 7: parseToXML Function

Next is to connect the database to MySQL server using *mysql_connect*. As depicted below, *localhost* is the name of the server, and then followed by the username and password for phpMyAdmin tools. The database that will be used is then chosen using *mysql_select_db* function. Execute all the rows in the *toll_fares* table using *SELECT ** function. The codes can be found in Figure 8.

```
// Opens a connection to a MySQL server
$connection=mysql_connect ('localhost', 'root', 'mysql');
if (!$connection) {
 die('Not connected : ' . mysql_error());
}

// Set the active MySQL database
$db_selected = mysql_select_db('north_south_expressway', $connection);
if (!$db_selected) {
 die ('Can\'t use db : ' . mysql_error());
}

//$result = mysql_query("set names 'utf8'");

// Select all the rows in the toll_fares table
$query = "SELECT * FROM toll_fares WHERE 1";
$result = mysql_query($query);
if (!$result) {
 die('Invalid query: ' . mysql_error());
}

header("Content-type: text/xml");
```

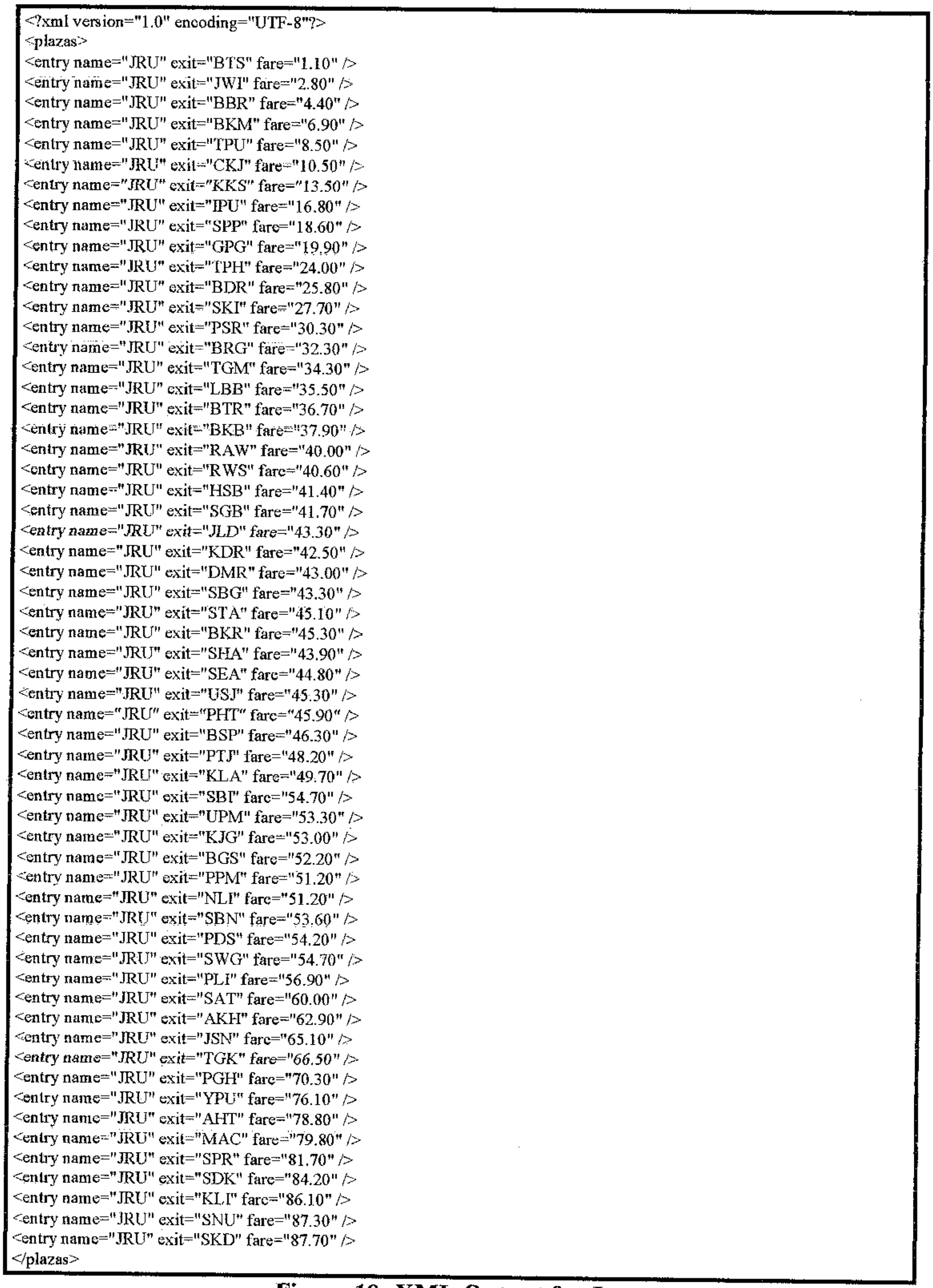
Figure 8: Connecting a Database

The *echo* function is used in order to parse the data to XML format. The parent *plazas* node is echoed out. The XML node for each row in the table needed to be echoed out by sending the entry, exit and fare through *parseToXML* function. The script is then finished by closing the *plazas* tag. Figure 9 shows the echo function.

```
// Start XML file, echo parent node
echo '<plazas>';

// Iterate through the rows, printing XML nodes for each
while ($row = @mysql_fetch_assoc($result)){
 // ADD TO XML DOCUMENT NODE
 echo '<entry ';
 echo 'name="' . parseToXML($row['entry']) . '" ';
 echo 'exit="' . parseToXML($row['exit']) . '" ';
 echo 'fare="' . parseToXML ($row['toll_fare']) . '" ';
  echo '/>';
}

// End XML file
echo '</plazas>';
```

Figure 9: Echo Function

The full code is attached in Appendix B. The resulting XML output for Juru toll entry is as Figure 10.



```
<?xml version="1.0" encoding="UTF-8"?>
<plazas>
<entry name="JRU" exit="BTS" fare="1.10" />
<entry name="JRU" exit="JWI" fare="2.80" />
<entry name="JRU" exit="BBR" fare="4.40" />
<entry name="JRU" exit="BKM" fare="6.90" />
<entry name="JRU" exit="TPU" fare="8.50" />
<entry name="JRU" exit="CKJ" fare="10.50" />
<entry name="JRU" exit="KKS" fare="13.50" />
<entry name="JRU" exit="IPU" fare="16.80" />
<entry name="JRU" exit="SPP" fare="18.60" />
<entry name="JRU" exit="GPG" fare="19.90" />
<entry name="JRU" exit="TPH" fare="24.00" />
<entry name="JRU" exit="BDR" fare="25.80" />
<entry name="JRU" exit="SKI" fare="27.70" />
<entry name="JRU" exit="PSR" fare="30.30" />
<entry name="JRU" exit="BRG" fare="32.30" />
<entry name="JRU" exit="TGM" fare="34.30" />
<entry name="JRU" exit="LBB" fare="35.50" />
<entry name="JRU" exit="BTR" fare="36.70" />
<entry name="JRU" exit="BKB" fare="37.90" />
<entry name="JRU" exit="RAW" fare="40.00" />
<entry name="JRU" exit="RWS" fare="40.60" />
<entry name="JRU" exit="HSB" fare="41.40" />
<entry name="JRU" exit="SGB" fare="41.70" />
<entry name="JRU" exit="JLD" fare="43.30" />
<entry name="JRU" exit="KDR" fare="42.50" />
<entry name="JRU" exit="DMR" fare="43.00" />
<entry name="JRU" exit="SBG" fare="43.30" />
<entry name="JRU" exit="STA" fare="45.10" />
<entry name="JRU" exit="BKR" fare="45.30" />
<entry name="JRU" exit="SHA" fare="43.90" />
<entry name="JRU" exit="SEA" fare="44.80" />
<entry name="JRU" exit="USJ" fare="45.30" />
<entry name="JRU" exit="PHT" fare="45.90" />
<entry name="JRU" exit="BSP" fare="46.30" />
<entry name="JRU" exit="PTJ" fare="48.20" />
<entry name="JRU" exit="KLA" fare="49.70" />
<entry name="JRU" exit="SBI" fare="54.70" />
<entry name="JRU" exit="UPM" fare="53.30" />
<entry name="JRU" exit="KJG" fare="53.00" />
<entry name="JRU" exit="BGS" fare="52.20" />
<entry name="JRU" exit="PPM" fare="51.20" />
<entry name="JRU" exit="NLI" fare="51.20" />
<entry name="JRU" exit="SBN" fare="53.60" />
<entry name="JRU" exit="PDS" fare="54.20" />
<entry name="JRU" exit="SWG" fare="54.70" />
<entry name="JRU" exit="PLI" fare="56.90" />
<entry name="JRU" exit="SAT" fare="60.00" />
<entry name="JRU" exit="AKH" fare="62.90" />
<entry name="JRU" exit="JSN" fare="65.10" />
<entry name="JRU" exit="TGK" fare="66.50" />
<entry name="JRU" exit="PGH" fare="70.30" />
<entry name="JRU" exit="YPU" fare="76.10" />
<entry name="JRU" exit="AHT" fare="78.80" />
<entry name="JRU" exit="MAC" fare="79.80" />
<entry name="JRU" exit="SPR" fare="81.70" />
<entry name="JRU" exit="SDK" fare="84.20" />
<entry name="JRU" exit="KLI" fare="86.10" />
<entry name="JRU" exit="SNU" fare="87.30" />
<entry name="JRU" exit="SKD" fare="87.70" />
</plazas>
```

Figure 10: XML Output for Juru

The code needs to be edited using the XML Editor to make it more meaningful. The result for Juru entry and all of its exits is displayed in Appendix C.

4.1.4 Automatic Detection of the Nearest Toll Station

The application would be better if it could detect the nearest toll entry automatically from the start point and nearest toll exit from the end point. This can be implemented by using SQL statement based on the Haversine formula that is shown in Equation (1) to (3). Since the entry and exits tables are identical, either one can be used to find the nearest toll station within a certain radius and distance restriction.

The SQL *SELECT* statement will find the closest toll station that is within the radius of 25 to the 5, 101 coordinate. This reference coordinate is based on trial and error method whereby it will provide the distance. The distance is calculated based on the latitude and longitude of the rows in the table and the targeted point. The distance value with less than 25 will be chosen and the toll station is limited to one as we were finding the nearest toll station. The SQL statement is shown in Figure 11[28].

```
SELECT id, ( 3959 * acos( cos( radians(5) ) * cos( radians( lat ) ) * cos( radians( lng ) - radians(101) ) + sin( radians(5) ) * sin(
radians( lat ) ) ) ) AS distance FROM entry HAVING distance < 25 ORDER BY distance LIMIT 0 , 1;
```

Figure 11: SQL Statement

Next is to generate the XML file by using the PHP DOM's function. It is basically the other method used to generate XML apart of *echo* function. The only difference is that the parameters latitude and longitude are needed from the URL. This can be shown as below:

```
// Get parameters from URL
$center_lat = $_GET["lat"];
$center_lng = $_GET["lng"];
$radius = $_GET["radius"];
```

Figure 12: PHP DOM's Function

The *SELECT* statement is executed by distance query. The *mysql_real_escape_string* is used to avoid SQL injection since the PHP script sent user input in SQL statement. The code that has been explained is shown in Figure 13.

```
// Search the rows in the markers table
$query = sprintf("SELECT  name, lat, lng, ( 3959* acos( cos( radians('%s') ) * cos( radians( lat ) ) * cos( radians( lng ) - radians('%s') ) + sin( radians('%s') ) * sin( radians( lat ) ) ) ) AS distance FROM entry HAVING distance < '%s' ORDER BY distance LIMIT 0 , 1",
 mysql_real_escape_string($center_lat),
 mysql_real_escape_string($center_lng),
 mysql_real_escape_string($center_lat),
 mysql_real_escape_string($radius));
$result = mysql_query($query);

if (!$result) {
 die("Invalid query: " . mysql_error());
}
```

Figure 13: Avoiding SQL injection

The full PHP script is attached in Appendix D. The resulting XML file can be checked using the server URL together with the parameter needed. It can be written as http://localhost/phpsqlsearch_genxml.php?lat=5&lng=101&radius=25. The XML output will give the nearest toll station to the point of 5,101 with radius of 25. Therefore, the XML output is displayed in Figure 14.

```
<?xml version="1.0"?>
<markers><marker name="Kuala Kangsar" lat="4.769308" lng="100.900902"
distance="17.33895024457618"/></markers>
```

Figure 14: XML Output

4.2 Modeling

This section explains how the application is design part by part by using the data from the previous section. This section consists of map display, driving directions, toll cost calculation, fuel consumption cost calculation, automatic detection of the nearest toll station and travel cost calculation.

4.2.1 Map Display

One of the purposes of this application is for user to estimate the mileage of the journey. A map is added for the ease of the user to look at the routes taken to reach destination. Besides, the map will help user to visualize the routes better.

A program is build to display the Malaysia map based on the programming knowledge retrieved after learning on Google Maps API and Adobe Flash Builder. The source codes written are attached in Appendix E. Figure 15 displays the result after debugging the codes.

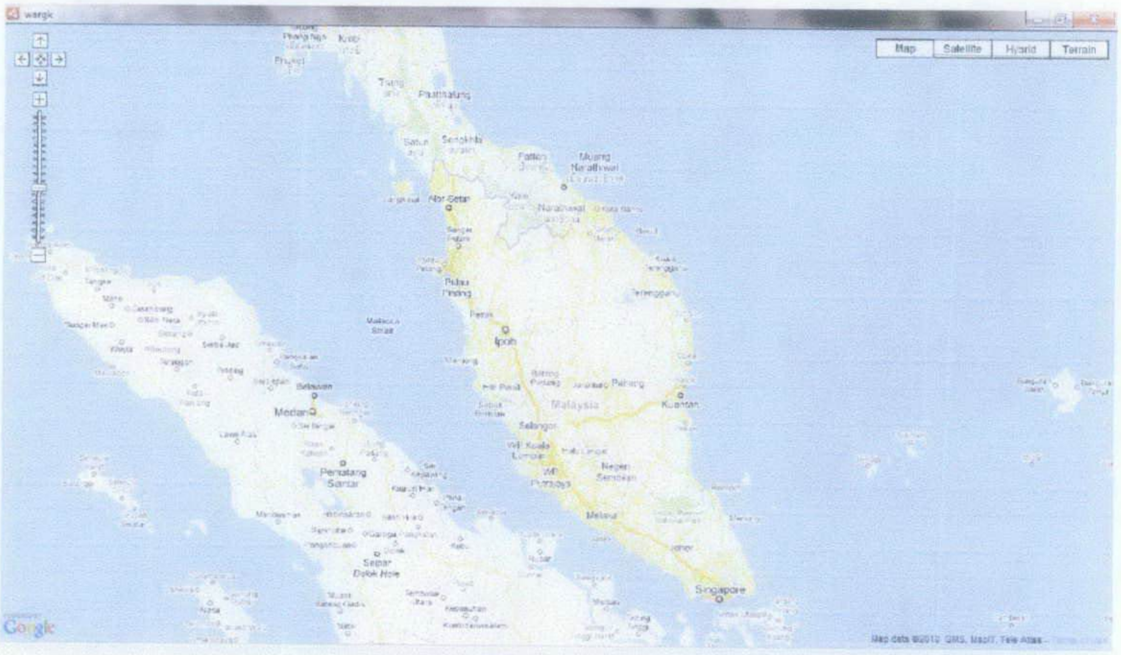


Figure 15: Map Display

4.2.1.1 Program Elaboration

There are two ways of displaying the map which are web application or desktop application. For this project, the map is displayed in a desktop application and it is declared as *<mx:WindowedApplication>*.

The map is initialized before going into next step of programming. The codes in Figure 16 show how the map is initialized.

```
<maps:Map xmlns:maps= "com.google.maps.*" id="map"
            mapevent_mapready="onMapReady(event)"
    width="100%" height="100%" url="http://localhost"
    sensor= "true"
key="ABQIAAAAckGGZPBUbdEHyikOnJMaWRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxTSKoJS776YBnwqN2UvA1m14-
r56g"/>
```

Figure 16: Map Initialization

A Map object is declared as a child of the *<mx:WindowedApplication>*. A maps namespace is defined to the reference code from *com.google.maps.**. Other parameters that must be defined are id, *mapevent_mapready* handler, an API key together with its URL and sensor. The *<maps:Map>* declaration also specifies width and height parameters to define how the Map will appear within the application. [29]

The Google Maps API for Flash consists of the Action Script codes. The *<mx:Script>* object holds the reference to Action Script code. The `<![CDATA[` and `]]>` delimiters are needed to inform the MXML parser to abandon the Action Script code because MXML is a type of XML. This is shown in the codes in Figure 17 [29].

```
<mx:Script>
  <![CDATA[

  // ActionScript Code

  ]]>
</mx:Script>
```

Figure 17: Action Script Codes

The Google Maps API libraries are imported with the import declarations. The libraries for the map, latitude and longitude, map event and map type are needed in order to build the basic map. The control function libraries such as zoom control, position control and map type control are added for user to control the map. The libraries used are Figure 18.

```
//basic map
import com.google.maps.LatLng;
import com.google.maps.Map;
import com.google.maps.MapEvent;
import com.google.maps.MapType;
//add control
import com.google.maps.controls.ZoomControl;
import com.google.maps.controls.MapTypeControl;
import com.google.maps.controls.PositionControl;
```

Figure 18: Import Libraries Declarations

Action Script is an event-driven programming language. In the declaration part, the Map declares an event listener to the Map object for the *MapEvent.MAP_READY* event through the use of the special parameter called *mapevent_mapready*. The event handler acts like a node for the initialization of the Google Maps API for Flash application. The event handler is defined as a private function. When the map receives the event, it will call the *onMapReady* function. [29]

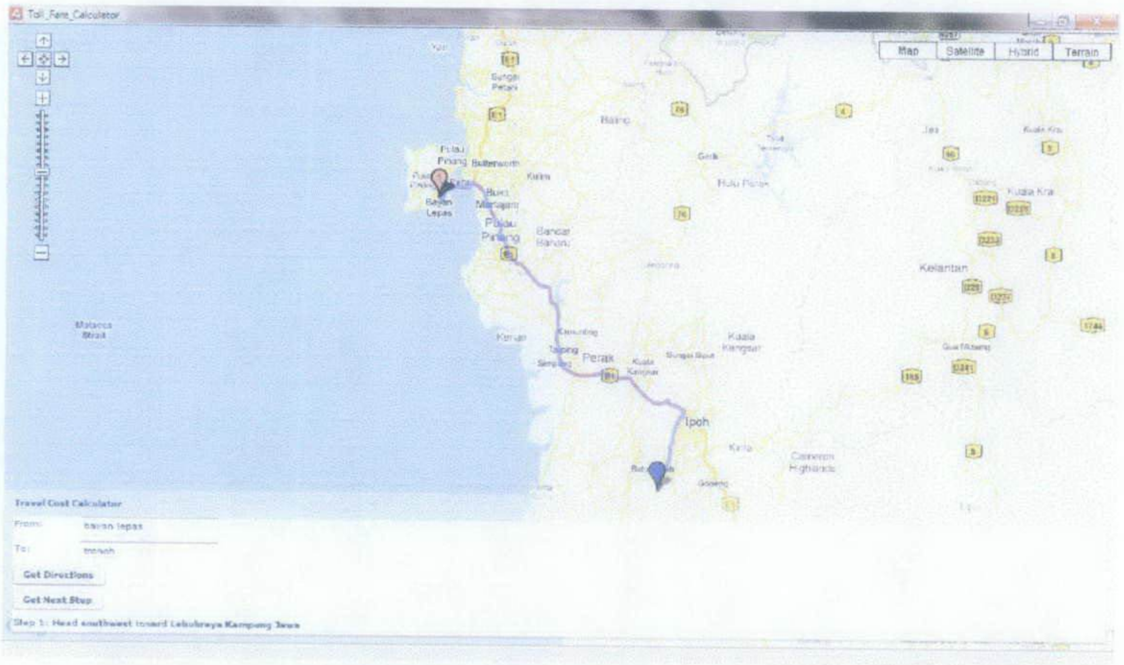
The *onMapReady()* function will pass the event parameter and call the *setCenter()* using the parameters that define the center of the map, the default zoom level and default map type. The *addControl()* are then called. The zoom control is added so that user can zoom in and out the map. Position control is added for the user to pan the map. User also can choose the map type that they prefer. The map types vary from the normal Map, Hybrid, Satellite and Terrain [4].The codes are shown in Figure 19.

```
private function onMapReady(event:Event):void {
this.map.setCenter(new LatLng(4.532618,101.777344), 7, MapType.NORMAL_MAP_TYPE);
//setting contol functions
this.map.addControl(new ZoomControl());
this.map.addControl(new PositionControl());
this.map.addControl(new MapTypeControl());
}
```

Figure 19: Main Function of Map Display

4.2.2 Driving Directions

In order for user to calculate the travel cost, the start location and the destination should be known. The Flex User Interface (UI) components used in programming the Driving Directions are *HBox*, *Label*, *TextInput* and *Button*. The horizontal boxes, *HBox* are added as a form for user to fill in the location and destination. Next will be two buttons for Get Directions and Get Next Step. The first button will show the directions to the destination. By clicking the Get Next Step button, user will get step-by-step directions each time the button is clicked. Apart from knowing that, there will be information about the time taken when user reach the destination. The service also provides the distance of the journey. The distance travelled is going to be used in calculating the fuel consumption cost of the journey. The source codes of the driving directions are attached in Appendix F. Figure 20 shows the result after debugging the codes.



Travel Cost Calculator
From: bayan lepas
To: tronoh
Step 1: Head southwest toward Lebuhraya Kampung Jawa

Figure 20: Driving Directions

4.2.2.1 Program Elaboration

A panel is added as a container for the Driving Directions information. In creating the UI, labels and text inputs for the users to enter *To* and *From* addresses are added. *HBox* components are added to make sure the labels and the text inputs are aligned. The id attributes are defined at the text inputs so that it can be referred systematically in the program codes. The codes in creating the UI are shown in Figure 21.

```
<mx:Panel title="Travel Cost Calculator" width="100%" height="170" bottom="0">
            <mx:HBox>
                        <mx:Label text="From: " width="70"/>
                        <mx:TextInput id="from" text=" " width="100%"/>
            </mx:HBox>
            <mx:HBox>
                        <mx:Label text="To: " width="70"/>
                        <mx:TextInput id="to" text=" " width="100%"/>
            </mx:HBox>
```

Figure 21: Creating UI

Button components named as Get Directions and Get Next Step are added. The Get Directions button's click event is connected to the *processDirections* function while the Get Next Step button's click event is sent to the *processTurnByTurn* function as shown in Figure 22.

```
                <mx:Button id="getDirections" label="Get Directions" click="processDirections(event);"/>
                <mx:Button id="getTurnByTurnDirections" label="Get Next Step" click="processTurnByTurn();"
enabled="false"/>
                <mx:Text id="step" htmlText="" width="100%"/>
        </mx:Panel>
```

Figure 22: Creating Buttons

Before the *processDirections()* function is called, the direction requests need to be initialized in *onMapReady()* function. Driving directions are requested using *Directions.load()* method. This method takes a query string or in other words the address. The *Directions* object needs to be requested from the Google's server. Therefore, the *Directions* object defines two *DirectionEvent* events which are *DIRECTION_SUCCESS* and *DIRECTION_FAILURE* as shown in Figure 23 [30].

```
dir = new Directions();
dir.addEventListener(DirectionsEvent.DIRECTIONS_SUCCESS, onDirLoad);
dir.addEventListener(DirectionsEvent.DIRECTIONS_FAILURE, onDirFail);
```

Figure 23: Direction Event

The *processDirections()* function is called when Get Directions button is clicked. The driving directions requested is perform using the *Directions.load()* method whereby the directions are based on the requested *To* and *From* inputs which is written as Figure 24.

```
dir.load("from: " + from.text + " to: " + to.text);
```

Figure 24: Load Directions Method

If the direction is successful, the *onDirLoad()* will be called. In this function, polylines and markers are to be created [30]. A polyline is added to the map to represent the entire directions. While there is start and end markers which are indicated in blue color. The *clearOverlays()* is called on the map to remove the previous query and *addOverlay(dir.createPolyline())* is called to create the polyline from the directions object. The map is then set to automatically zoom to fit the route created by the polyline. This can be accomplished by finding the bounds of the polyline, and then set the center of the map to the center of the polyline bounds. The *map.getBoundsZoomLevel* is used to find the optimal zoom level that fits the whole polyline. The explanations are depicted in the code in Figure 25.

```
var startMarker:Marker;
var endMarker:Marker;

map.clearOverlays();
map.addOverlay(dir.createPolyline());
map.setZoom(map.getBoundsZoomLevel(dir.bounds));
map.setCenter(dir.bounds.getCenter());

startMarker = new Marker(dir.getRoute(0).startGeocode.point, new MarkerOptions({fillStyle: {color:Color.BLUE}}));
endMarker = new Marker(dir.getRoute(0).endGeocode.point, new MarkerOptions({fillStyle: {color:Color.BLUE}}));
map.addOverlay(startMarker);
map.addOverlay(endMarker);
```

Figure 25: Successful Directions

Each route or direction from one place to another consists of one or more *Step* objects which contain the turn-by-turn directions. The routes can be achieved by *Directions.getRoute(i:Number)* method. Route object store the number of steps of that route, starting and ending geocode of that route, distance and time taken to arrive at the destination. Meanwhile, steps within a route can be retrieved using the *Route.getStep(i:Number)* method. Each *Step* object contains the description for the query and the distance computed. The codes are as in Figure 26 [30].

```
if (turnCounter <= dir.getRoute(0).numSteps) {
stepText = dir.getRoute(0).getStep(turnCounter-1).descriptionHtml;
stepMarker = new Marker(dir.getRoute(0).getStep(turnCounter-1).latLng, new MarkerOptions({label: turnCounter.toString()}));
map.addOverlay(stepMarker);
step.htmlText = "Step " + turnCounter + ": " + stepText;
                                        }
else {
        getTurnByTurnDirections.enabled = false;
        step.htmlText = "Arrive at " + to.text + " : " +dir.getRoute(0).summaryHtml;
                                        }
```

Figure 26: Step-by-step Directions

If Google server is not able to find the driving directions, the *onDirFail()* function is called. It will show the service status alert. Table 5 shows the service status codes after response.

Table 5: Service Status Codes of Google Maps [4]

Name	Value	Description
GEO_ABORTED_REQUEST	10101	Status code indicating that the request was aborted due to a newer request.
GEO_BAD_KEY	610	Status code indicating a missing or invalid API key.
GEO_BAD_REQUEST	400	Status code indicating that the request could not be processed properly by the servers.
GEO_BAD_STATUS_START	600	The lower numerical limit of bad status codes.
GEO_MISSING_ADDRESS	601	Status code indicating that the address parameter is missing.
GEO_MISSING_QUERY	601	Status code indicating that the query parameter is missing. This is a generalization of the GEO_MISSING_ADDRESS code introduced in the geocoder.
GEO_REQUEST_DENIED	650	Status code indicating that the request was denied.
GEO_SERVER_ERROR	500	Status code indicating problem with the geocode server. For example, this value may be returned if the geoserver is down.
GEO_SUCCESS	200	Status code indicating a successful geocoding query. .
GEO_TOO_MANY_QUERIES	620	Status indicating that we received too many requests in the 24h period.
GEO_UNAVAILABLE_ADDRESS	603	Status indicating that for legal or other reasons we may not return a reply to this particular query.
GEO_UNKNOWN_ADDRESS	602	Status code indicating an unknown or incorrect address.
GEO_UNKNOWN_DIRECTIONS	604	Status code indicating that no directions could be computed.
GEO_UNKNOWN_ERROR	651	Status code indicating an unknown error.

By using the driving directions method, the distance taken to reach the destination can be estimated together with the duration. This information is then used to calculate the fuel consumption cost.

4.2.3 Toll Cost Calculation

The custom toll cost calculator is designed using two related combo boxes which user will have to select the entry and exit. The related combo box is needed as in this case, the toll exit options are based on the selection of the toll entry. As a result, the toll cost of the journey will be displayed.

4.2.3.1 Program Elaboration

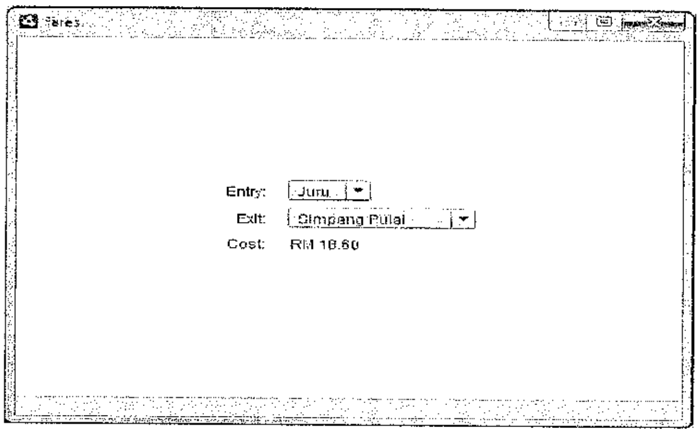
The resulting XML file for the toll cost is used as the data source namely *test.xml*. The *creationComplete* event is used to trigger a service call. In this chunk of code, the event called the rate which is the XML file. The XML contents need to be bind using the XMLList function. Then, for the toll entry combo box, the data is taken from the result of the bind XML file which is indicated by *plazaList*. The *labelField* is allocated in array method which is *@name*. The data for toll exits is taken from *entryCB.selectedItem.exit* where *entryCB* is the identification for the toll entry combo box. Consequently, the toll cost is displayed from this piece of code, *exitCB.selectedItem.fare*. The relationship between the toll entry, toll exit and the toll cost has been explained. The codes are shown in Figure 27.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication name="EstimateForm_test"
        xmlns:mx="http://www.adobe.com/2006/mxml"
                                        layout="vertical"
                                        creationComplete="rate.send()"
                                        verticalAlign="middle"
                                        backgroundColor="white">
    <mx:HTTPService id="rate" resultFormat="e4x" url="test.xml" result="handleResult(event)"/>
    <mx:Script>
            <![CDATA[
                    import mx.rpc.events.ResultEvent;

                    [Bindable]
                    private var plazaList:XMLList;

                    private function handleResult(event:ResultEvent):void{
                            plazaList = event.result.entry as XMLList
                    }
            ]]>
    </mx:Script>
    <mx:Form>
            <mx:FormItem label="Entry:">
                    <mx:ComboBox id="entryCB" dataProvider="{plazaList}" labelField="@name" />
            </mx:FormItem>
            <mx:FormItem label="Exit:">
```

```
                              <mx:ComboBox id="exitCB" dataProvider="{entryCB.selectedItem.exit}"
labelField="@name" />
                        </mx:FormItem>
                        <mx:FormItem label="Cost:">
                              <mx:Label text="RM {exitCB.selectedItem.fare}"/>
                        </mx:FormItem>
            </mx:Form>
</mx:WindowedApplication>
```

Figure 27: Related Combo Boxes

The resulting from the above program is shown in Figure 28.



Entry: Juru

Exit: Simpang Pulai

Cost: RM 18.60

Figure 28: Toll Cost Calculator

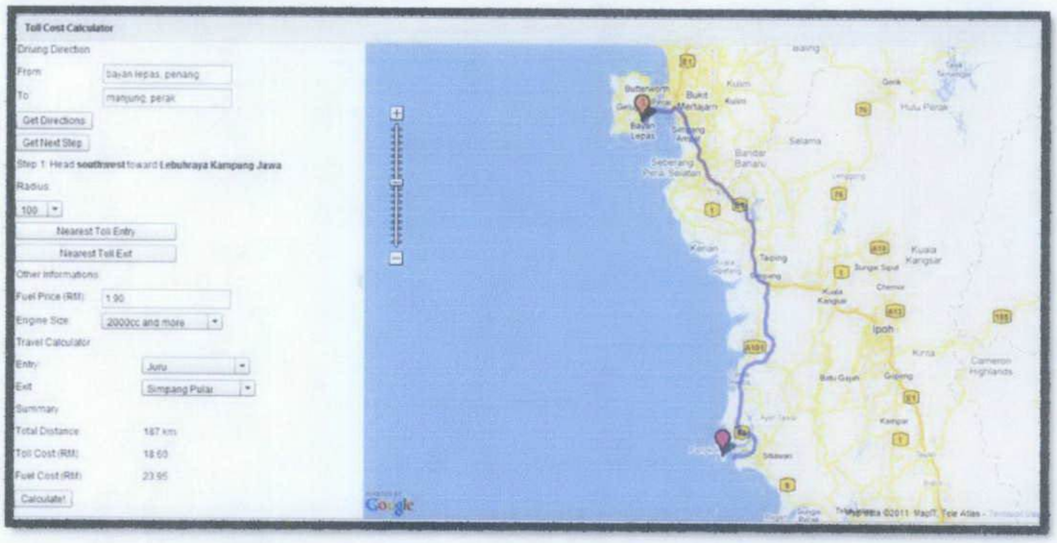
4.2.4 Fuel Consumption Cost Calculation

There are a few measurements need to be known in order to perform the *Fuel Consumption Cost Calculation* such as the current price of fuel, vehicle miles per gallon (MPG), user location and destination [24].

Therefore, the driving direction codes is modified by adding *HBox* components that could display the total distance travelled in kilometers and fuel price in Ringgit Malaysia (RM). The total distance will be calculated automatically after users hit the Get Directions button. For the fuel price, user has to input in the fuel price or else it will follow the current price which is RM1.90 by default.

For the vehicle's engine size, it has been narrowed down to four categories which are *1000cc and less*, *1100cc to 1500cc*, *1600cc to 1900cc* and *2000cc and more*. A combo box is added for user to select the vehicle's engine size. A button named *'Calculate!'* is added on the application. This button will perform the calculation and the estimated fuel consumption cost will be displayed in the Fuel Cost section.

Figure 29 and 30 illustrates the fuel consumption cost calculator and the source code is attached in Appendix G. The first red box indicates the information used to calculate the fuel consumption cost while the second red box is the result of the fuel consumption cost.



Toll Cost Calculator

Driving Direction

From: bayan lepas, penang

To: manjung, perak

Get Directions

Get Next Step

Step 1: Head **southwest** toward **Lebuhraya Kampung Jawa**

Radius: 100

Nearest Toll Entry

Nearest Toll Exit

Other Informations

Fuel Price (RM): 1.90

Engine Size: 2000cc and more

Travel Calculator

Entry: Juru

Exit: Simpang Pulai

Summary

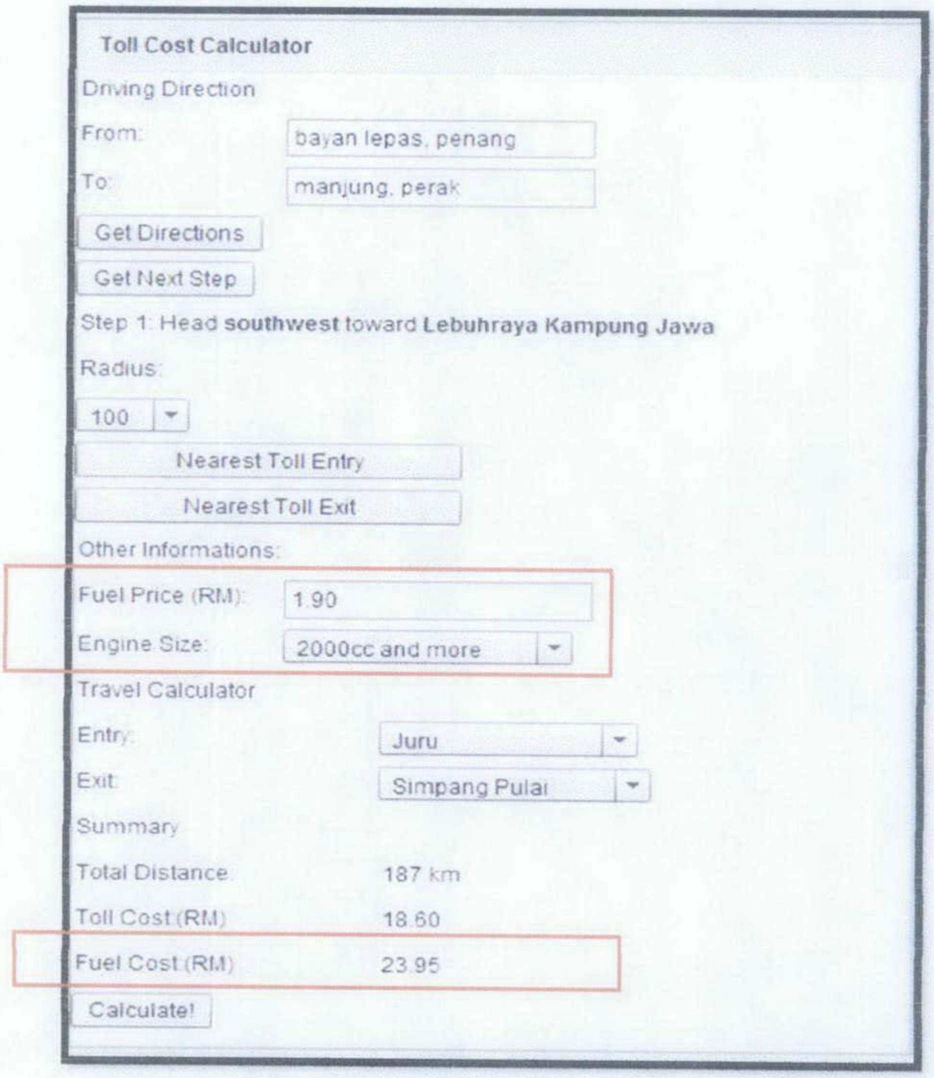
Total Distance: 187 km

Toll Cost (RM): 18.60

Fuel Cost (RM): 23.95

Calculate!

Figure 29: Fuel Consumption Cost Calculator with Map


Toll Cost Calculator

Driving Direction

From: bayan lepas, penang

To: manjung, perak

Step 1: Head **southwest** toward **Lebuhraya Kampung Jawa**

Radius: 100

Other Informations:

Fuel Price (RM): 1.90

Engine Size: 2000cc and more

Travel Calculator

Entry: Juru

Exit: Simpang Pulai

Summary

Total Distance: 187 km

Toll Cost (RM): 18.60

Fuel Cost (RM): 23.95

Figure 30: Fuel Consumption Cost Calculator

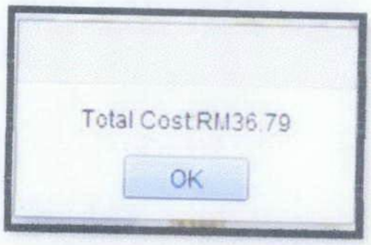
4.2.5 Travel Cost Calculation

The travel cost can be calculated by using the information gained from driving directions, fuel consumption cost and toll cost. The total travel cost is the sum of the fuel consumption cost and the toll cost of the journey. The travel cost is displayed using the alert function from the Adobe Flash Builder. It can be written as in Figure 31.

```
total=fuel_cost+parseFloat(exitCB.selectedItem.fare);
Alert.show("Total Cost:RM" + total.toFixed(2));
```

Figure 31: Travel Cost Calculation

When the *Calculate!* button is clicked, the calculation is performed and the result is displayed as Figure 32.



Total Cost RM36.79

Figure 7: Travel Cost

4.2.6 Automatic Detection of the Nearest Toll Station

In order to search for the nearest toll station, the driving directions part is extended. Since the PHP scripts took the latitude and longitude parameters in order to perform search, the *ClientGeocoder* function will pass the address from the *From* textbox to the *GeocodingEvent* function. Once the *GeocodingEvent* is successful, it will send the location to the *searchNearEntry* function as shown in Figure 33 [28].

```
private function searchEntry(event:Event):void {
                        var geocoder:ClientGeocoder = new ClientGeocoder();
                        geocoder.addEventListener(
                                GeocodingEvent.GEOCODING_SUCCESS,
                                function(event:GeocodingEvent):void {
                                        var placemarks:Array =
event.response.placemarks;
                                        if (placemarks.length > 0) {
                                                searchEntryNear(placemarks[0].point);
                                        }
                                });
                        geocoder.addEventListener(
                                GeocodingEvent.GEOCODING_FAILURE,
                                function(event:GeocodingEvent):void {
                                        Alert.show("Geocoding failed");
                                });
                        geocoder.geocode(from.text);
                }
```

Figure 33: SearchEntry Function

The *searchNearEntry* function will pass the latitude and longitude values to the PHP scripts. The *URLLoader* function is used to load the XML output. The code for *searchNearEntry* function is shown in Figure 34 [28]:

```
public function searchEntryNear(center:LatLng):void {
                        var searchUrl:String =
'http://localhost/phpsqlsearch_genxml.php?lat=' + center.lat() + '&lng=' + center.lng() + '&radius=' + radius.text;
                        var xmlString:URLRequest = new URLRequest(searchUrl);
                        var xmlLoader:URLLoader = new URLLoader(xmlString);
                        xmlLoader.addEventListener("complete", parseLocationsXml);
                }
```

Figure 34: SearchNearEntry Function

The same coding goes for the toll exit. The result for the automatic detection of the nearest toll station is shown in Figure 35, 36 and 37 whereas the full coding is displayed in Appendix H.

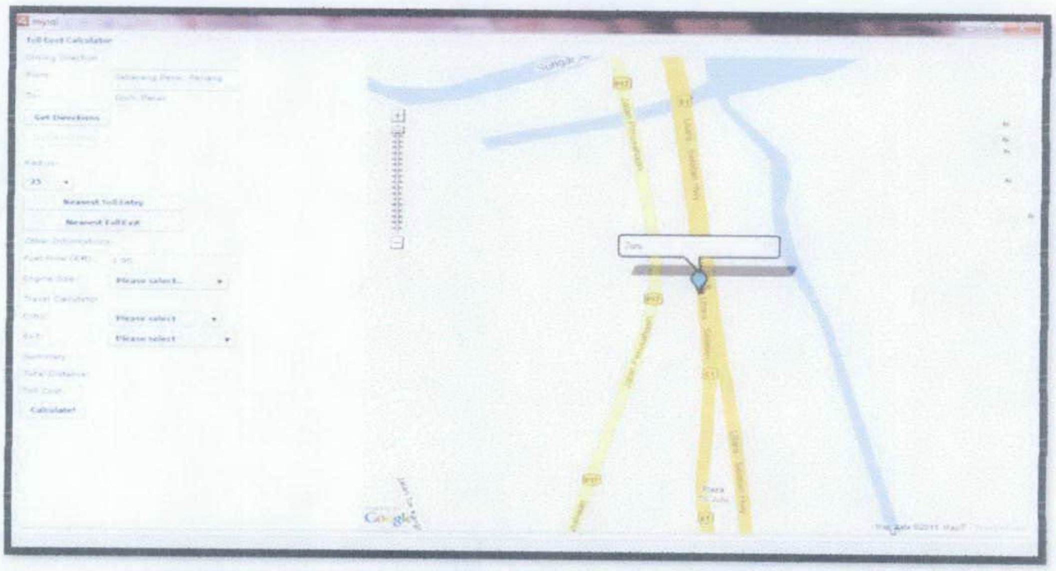
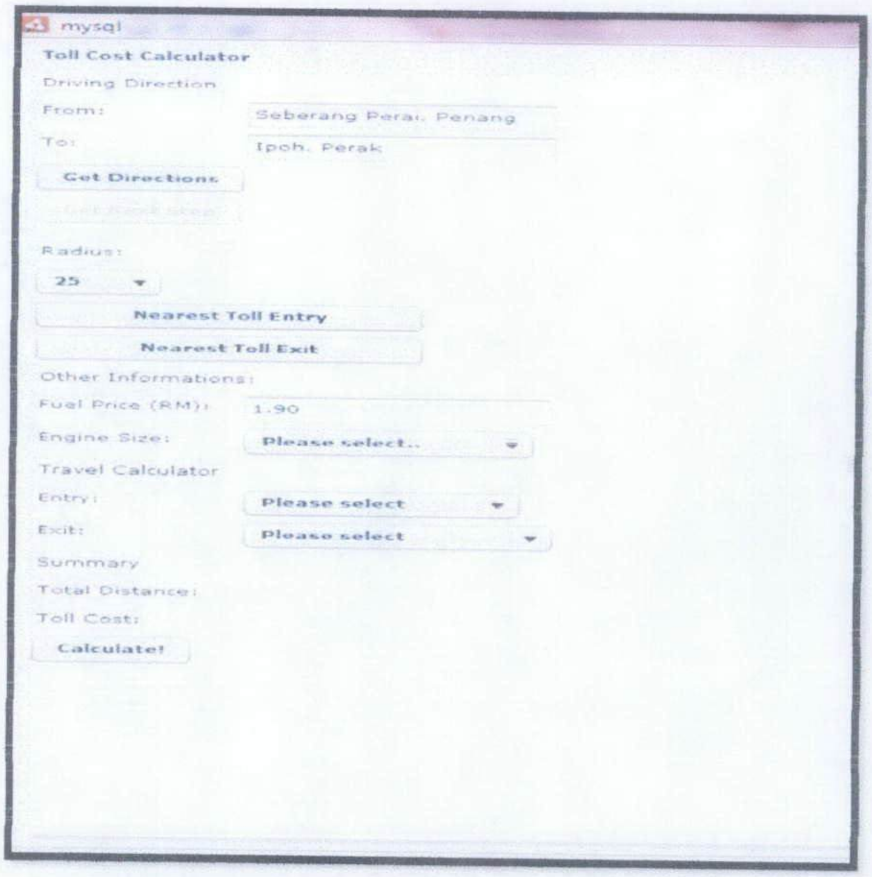

Toll Cost Calculator
Driving Direction
From: Seberang Perai, Penang
To: Gurh, Perak
Radius:
25
Other Informations:
Fuel Price (RM): 1.95
Engine Size: Please select...
Travel Calculator
Entry: Please select
Exit: Please select
Summary
Total Distance:
Toll Cost:

Figure 85: Travel Cost Calculator with Auto-Detect Function


mysql

Toll Cost Calculator

Driving Direction

From: Seberang Perai, Penang

To: Ipoh, Perak

Get Directions

[illegible]

Radius:

25

Nearest Toll Entry

Nearest Toll Exit

Other Informations:

Fuel Price (RM): 1.90

Engine Size: **Please select..**

Travel Calculator

Entry: **Please select**

Exit: **Please select**

Summary

Total Distance:

Toll Cost:

Calculate!

Figure 36: Zoom-in Travel Cost Calculator

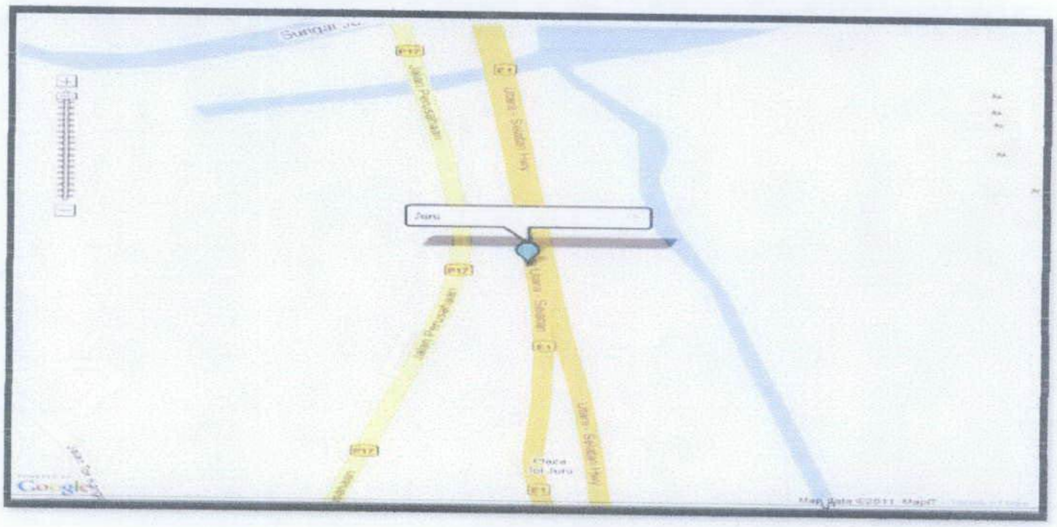


Figure 37: Automatic Detection of the Nearest Toll Station

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The requirement analysis and design has been developed to illustrate the TCC application. The project has been divided into three parts which are mileage calculation between two locations, fuel consumption cost calculation and toll cost determination. The project is completed with the use of software such as Google Maps API, Adobe Flash Builder, PHP and MySQL database. The application is also improved by adding automatic detection of the nearest toll station feature. TCC provides benefits for those who would like to plan their journey. Apart from that, the application is also benefits in terms of fast response and user friendly.

5.2 Recommendations

Some improvements are needed to be done in order to improve the accuracy for the fuel consumption cost calculation. Rather than using the combined fuel economy, the calculation will be more accurate if the fuel economy for urban routes and extra urban routes are taken into consideration. Besides, user average driving speed will also affect the fuel consumption cost. Thus, the driving speed should be considered. Therefore, these recommendations are hoped to be done in order for the calculation to be more accurate.

REFERENCES

[1] Genivi, "About GENIVI", Genivi. [Online]. Available: http://www.genivi.org. [Accessed: 14 August 2010]

[2] "Digital Lifestyle Takes to the Road", Whitepaper, [Online]. Available: http://download.intel.com/design/embedded/infotainment/docs/313714.pdf [Accessed: 14 August 2010]

[3] Palmer, S.; "The Value of an Open Infotainment Platform", Autotech Daily, September 10, 2008, [Online]. Available: http://download.intel.com/design/embedded/infotainment/docs/ Intel_Viewpoint-clean.pdf [Accessed: 14 August 2010]

[4] Ahmad Moghni,H.N; *Industrial Internship Final Report.* Perak,MY: Universiti Teknologi Petronas, 2010.

[5] Global Resources, "Industrial-level Motherboard uses Intel Atom eMenlow Platform", Global Sources, October 10, 2008. [Online]. Available: http://www.globalresources.com/gsol/I/Industrial-computer/a/9000000100993.htm. [Accessed: 14 August 2010]

[6] Applied Data Systems, "Catalyst Based on Intel eMenlow platform, the Catalyst module beefs up performance while maintaining low power", Applied Data Systems, [Online]. Available: http://www.applieddata.net/developers/SBC/Catalyst.asp. [Accessed: 14 August 2010]

[7] "Embedded Solutions with Intel Atom Processors", [Online]. Available: http://www.contradata.it/cataloghi/IEI_Atom_solutions.pdf. [Accessed: 14 August 2010]

[8] Intel, "Intel Hyper-Threading technology", Intel, [Online]. Available: http://www.intel.com/technology/platform-technology/hyper-threading/index.htm. [Accessed: 14 August 2010]

[9] "Advantech technology for Medical Devices", [Online]. Available: http://www.tecnoimprese.it/ [Accessed: 14 August 2010]

[10] "Why Use Ubuntu?", Ubuntu.com, [Online]. Available: http://www.ubuntu.com/ubuntu/why-use-ubuntu. [Accessed: 28 April 2010]

[11] Google Code, "Google Maps API Family", Google Code, [Online]. Available: http://code.google.com/apis/maps/. [Accessed: 16 August 2010]

[12] Google Code, "Google Maps Javascript API V3", Google Code, [Online]. Available: http://code.google.com/apis/maps/documentation/javascript/. [Accessed: 16 August 2010]

[13] Google Code, "Google Maps API for Flash", Google Code, [Online]. Available: http://code.google.com/apis/maps/documentation/flash. [Accessed: 16 August 2010]

[14] Adobe Flash Builder, "Develop Cross-platform Rich Internet Applications", Adobe, [Online]. Available: http://www.adobe.com/africa/products/flashbuilder/ [Accessed: 16 August 2010]

[15] PHP, "What is PHP?", PHP, [Online]. Available: http://www.php.net/ [Accessed: 21 September2010]

[16] Davis, P.; "A Brief Introduction to PHP: Hypertext Preprocessor", [Online]. Available: http://www.webguys.com/pdavis/Programs/What_Is_PHP [Accessed: 21 September 2010]

[17] Webmaster Help, “PHP Tutorials”, Freewebmasterhelp.com, [Online]. Available: http://www.freewebmasterhelp.com/tutorials/php. [Accessed: 19 September 2010]

[18] Oracle, “MySQL”, Oracle, [Online]. Available: http://www.sun.com/software/products/mysql/. [Accessed: 22 September 2010]

[19] Google Code, “Using PHP/MySQL with Google Maps”, Google Code, [Online]. Available: http://code.google.com/apis/maps/articles/phpsqlajax.html. [Accessed: 19 September 2010]

[20] PLUS, “Toll System”, Plus.com, [Online]. Available: http://www.plus.com.my/ [Accesed : 19 September 2010]

[21] Google Maps, “Google Maps Malaysia”, Google, [Online]. Available: http://maps.google.com.my/maps?hl=en&tab=wl [Accesed : 21 September 2010]

[22] Lembaga Lebuhraya Malaysia, “Closed-loop Toll Rate” llm.gov, [Online]. Available: http://www.llmnet.gov.my/serverpages/highway/highway_info.aspx. [Accesed : 21 September 2010]

[23] Ahmad, N.; “Specific Data Model of Smart Fuel Consumption Cost Estimator," *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, vol.1, no., pp.429-432, 13-15 Nov. 2009

[24] Jacobson, K.; “Fuel Cost and Driving Directions,” Blogspot.com, [Online]. Available: http://solidcoding.blogspot.com/2008/05/fuel-cost-and-trip-planner.html [Accessed: 12 October 2010]

[25] Sinnott, R. W.; "Virtues of the Haversine," in *Sky and Telescope*, vol. 68, no. 2, 1984, pp. 159

[26] Deehan, P.; "Creating two Related ComboBoxes" in Flex Examples, August 2007, [Online]. Available: http://blog.flexexamples.com/2007/08/04/creating-two-related-comboboxes/ [Accessed: 19 March 2011]

[27] Mackey, G. E.; "Efficient Nearest Neighbor Searches in N-ABLE^{TM}," Sandia National Lab., Albuquerque, New Mexico, Sandia Rep., July 2010.

[28] Fox, P.; "Creating a Store Locator with PHP, MySQL & Google Maps", in Google Code, January 2008 , [Online].
Available:
http://code.google.com/apis/maps/articles/phpsqlsearch.html#findnearsql
[Accessed: 19 February 2011]

[29] Google Code, "Google Maps API for Flash Basics", Google Code, [Online].
Available: http://code.google.com/apis/maps/documentation/flash/basics.html
[Accessed: 1 April 2011]

[30] Fox, P.; "Creating a Driving Directions Flex App Using the Google Maps API for Flash", Adobe.com, January 2009, [Online].
Available: http://www.adobe.com/devnet/flex/articles/googlemaps_api.html
[Accessed: 1 April 2011]

APPENDIX A

GANTT CHART FOR FYP II

No	Details/Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Port application on the machine	▨	▨												
2	Create and populate database			▨	▨										
3	XML parsing					▨	▨								
4	Port database together with the application						▨								
5	Combine completed stages and total up cost							▨							
6	Improvement (automated location)					▨	▨	▨							
7	Submission of Progress Report								○						
8	Other improvements/changes (integrate traffic infos)									▨	▨				
9	Pre-EDX											○			
10	Submission of Draft Report												○	○	
11	Submission of Final Report + Technical Report														
12	Oral Presentation														○

APPENDIX B

PHP SCRIPTS FOR XML PARSING

```
<?php

function parseToXML($htmlStr)
{
$xmlStr=str_replace('<','&lt;',$htmlStr);
$xmlStr=str_replace('>','&gt;',$xmlStr);
$xmlStr=str_replace('"','&quot;',$xmlStr);
$xmlStr=str_replace("'",'&#39;',$xmlStr);
$xmlStr=str_replace("&",'&amp;',$xmlStr);
return $xmlStr;
}

// Opens a connection to a MySQL server
$connection=mysql_connect ('localhost', 'root', 'mysql');
if (!$connection) {
  die('Not connected : ' . mysql_error());
}

// Set the active MySQL database
$db_selected = mysql_select_db('north_south_expressway', $connection);
if (!$db_selected) {
  die ('Can\'t use db : ' . mysql_error());
}

//$result = mysql_query("set names 'utf8'");

// Select all the rows in the toll_fares table
$query = "SELECT * FROM toll_fares WHERE 1";
$result = mysql_query($query);
if (!$result) {
  die('Invalid query: ' . mysql_error());
}

header("Content-type: text/xml");

// Start XML file, echo parent node
echo '<plazas>';

// Iterate through the rows, printing XML nodes for each
while ($row = @mysql_fetch_assoc($result)){
  // ADD TO XML DOCUMENT NODE
  echo '<entry ';
  echo 'name="' . parseToXML($row['entry']) . '" ';
  echo 'exit="' . parseToXML($row['exit']) . '" ';
  echo 'fare="' . parseToXML ($row['toll_fare']) . '" ';
   echo '/>';
}

// End XML file
echo '</plazas>';

?>
```

APPENDIX C
EDITTED XML STRUCTURE

```
<?xml version="1.0" encoding="UTF-8"?>
<plazas>
<entry name="Juru" >
<exit name="Bkt. Tambun (S)" >
<fare>1.10</fare>
</exit>
<exit name="Jawi">
<fare>2.80
</fare>
</exit>
<exit name="Bandar Baharu">
<fare>4.40
</fare>
</exit>
<exit name="Bkt. Merah">
<fare>6.90
</fare>
</exit>
<exit name="Taiping (U)">
<fare>8.50
</fare>
</exit>
<exit name="Changkat Jering">
<fare>10.50</fare>
</exit>
<exit name="Kuala Kangsar">
<fare>13.50</fare>
</exit>
<exit name="Ipoh (U)">
<fare>16.80</fare>
</exit>
<exit name="Simpang Pulai">
<fare>18.60</fare>
</exit>
<exit name="Gopeng">
<fare>19.90</fare>
</exit>
<exit name="Tapah">
<fare>24.00</fare>
</exit>
<exit name="Bidor">
<fare>25.80</fare>
</exit>
<exit name="Sungkai">
<fare>27.70</fare>
</exit>
<exit name="Slim River">
<fare>30.30</fare>
</exit>
<exit name="Behrang">
<fare>32.30</fare>
</exit>
<exit name="Tg. Malim">
<fare>34.30</fare>
</exit>
<exit name="Lembah Beringin">
<fare>35.50</fare>
</exit>
<exit name="Bkt. Tagar">
<fare>36.70</fare>
</exit>
<exit name="Bkt. Beruntung">
<fare>37.90</fare>
</exit>
<exit name="Rawang">
<fare>40.00</fare>
</exit>
```

```
<exit name="Rawang (S)">
<fare>40.60</fare>
</exit>
<exit name="Hospital Sg. Buloh">
<fare>41.40</fare>
</exit>
<exit name="Sg. Buloh">
<fare>41.70</fare>
</exit>
<exit name="Jalan Duta">
<fare>43.30</fare>
</exit>
<exit name="Kota Damansara">
<fare>42.50</fare>
</exit>
<exit name="Damansara">
<fare>43.00</fare>
</exit>
<exit name="Subang">
<fare>43.30</fare>
</exit>
<exit name="Setia Alam">
<fare>45.10</fare>
</exit>
<exit name="Bkt. Raja">
<fare>45.30</fare>
</exit>
<exit name="Shah Alam">
<fare>43.90</fare>
</exit>
<exit name="Seafield">
<fare>44.80</fare>
</exit>
<exit name="USJ">
<fare>45.30</fare>
</exit>
<exit name="Putra Heights">
<fare>45.90</fare>
</exit>
<exit name="Bdr. Saujana Putra">
<fare>46.30</fare>
</exit>
<exit name="Putrajaya">
<fare>48.20</fare>
</exit>
<exit name="KLIA">
<fare>49.70</fare>
</exit>
<exit name="Sg. Besi">
<fare>54.70</fare>
</exit>
<exit name="UPM">
<fare>53.30</fare>
</exit>
<exit name="Kajang">
<fare>53.00</fare>
</exit>
<exit name="Bangi">
<fare>52.20</fare>
</exit>
<exit name="Putra Mahkota">
<fare>51.20</fare>
</exit>
<exit name="Nilai">
<fare>51.20</fare>
</exit>
<exit name="Seremban">
<fare>53.60</fare>
</exit>
<exit name="Port Dickson (S)">
<fare>54.20</fare>
</exit>
<exit name="Senawang">
<fare>54.70</fare>
</exit>
<exit name="Pedas Linggi">
```

```
<fare>56.90</fare>
</exit>
<exit name="Spg. Ampat">
<fare>60.00</fare>
</exit>
<exit name="Ayer Keroh">
<fare>62.90</fare>
</exit>
<exit name="Jasin">
<fare>65.10</fare>
</exit>
<exit name="Tangkak">
<fare>66.50</fare>
</exit>
<exit name="Pagoh">
<fare>70.30</fare>
</exit>
<exit name="Yong Peng (U)">
<fare>76.10</fare>
</exit>
<exit name="Ayer Hitam">
<fare>78.80</fare>
</exit>
<exit name="Machap">
<fare>79.80</fare>
</exit>
<exit name="Spg. Renggam">
<fare>81.70</fare>
</exit>
<exit name="Sedenak">
<fare>84.20</fare>
</exit>
<exit name="Kulai">
<fare>86.10</fare>
</exit>
<exit name="Senai (U)">
<fare>87.30</fare>
</exit>
<exit name="Skudai">
<fare>87.70</fare>
</exit>
</entry>
<entry name="Bukit Tambun (U)">
<exit name="Juru">
<fare>1.10
</fare>
</exit>
</entry>
<entry name="Bukit Tambun (S)">
<exit name="Jawi">
<fare>1.90
</fare>
</exit>
<exit name="Bandar Baharu">
<fare>3.50
</fare>
</exit>
<exit name="Bkt. Merah">
<fare>6.10
</fare>
</exit>
<exit name="Taiping (U)">
<fare>7.60
</fare>
</exit>
<exit name="Changkat Jering">
<fare>9.60</fare>
</exit>
<exit name="Kuala Kangsar">
<fare>12.60</fare>
</exit>
<exit name="Ipoh (U)">
<fare>15.90</fare>
</exit>
<exit name="Simpang Pulai">
<fare>17.70</fare>
```

```
</exit>
<exit name="Gopeng">
<fare>19.00</fare>
</exit>
<exit name="Tapah">
<fare>23.10</fare>
</exit>
<exit name="Bidor">
<fare>24.90</fare>
</exit>
<exit name="Sungkai">
<fare>26.80</fare>
</exit>
<exit name="Slim River">
<fare>19.40</fare>
</exit>
<exit name="Behrang">
<fare>31.40</fare>
</exit>
<exit name="Tg. Malim">
<fare>33.40</fare>
</exit>
<exit name="Lembah Beringin">
<fare>34.60</fare>
</exit>
<exit name="Bkt. Tagar">
<fare>35.80</fare>
</exit>
<exit name="Bkt. Beruntung">
<fare>37.00</fare>
</exit>
<exit name="Rawang">
<fare>39.10</fare>
</exit>
<exit name="Rawang (S)">
<fare>39.70</fare>
</exit>
<exit name="Hospital Sg. Buloh">
<fare>40.50</fare>
</exit>
<exit name="Sg. Buloh">
<fare>40.80</fare>
</exit>
<exit name="Jalan Duta">
<fare>42.40</fare>
</exit>
<exit name="Kota Damansara">
<fare>41.60</fare>
</exit>
<exit name="Damansara">
<fare>42.10</fare>
</exit>
<exit name="Subang">
<fare>42.40</fare>
</exit>
<exit name="Setia Alam">
<fare>44.20</fare>
</exit>
<exit name="Bkt. Raja">
<fare>44.40</fare>
</exit>
<exit name="Shah Alam">
<fare>44.30</fare>
</exit>
<exit name="Seafield">
<fare>43.90</fare>
</exit>
<exit name="USJ">
<fare>44.40</fare>
</exit>
<exit name="Bdr. Saujana Putra">
<fare>45.40</fare>
</exit>
<exit name="Putrajaya">
<fare>47.30</fare>
</exit>
<exit name="KLIA">
```

```
<fare>48.80</fare>
</exit>
<exit name="Sg. Besi">
<fare>53.80</fare>
</exit>
<exit name="UPM">
<fare>52.40</fare>
</exit>
<exit name="Kajang">
<fare>52.10</fare>
</exit>
<exit name="Bangi">
<fare>51.30</fare>
</exit>
<exit name="Putra Mahkota">
<fare>50.30</fare>
</exit>
<exit name="Nilai">
<fare>50.30</fare>
</exit>
<exit name="Seremban">
<fare>52.70</fare>
</exit>
<exit name="Port Dickson (S)">
<fare>53.30</fare>
</exit>
<exit name="Senawang">
<fare>53.80</fare>
</exit>
<exit name="Pedas Linggi">
<fare>56.00</fare>
</exit>
<exit name="Spg. Ampat">
<fare>59.10</fare>
</exit>
<exit name="Ayer Keroh">
<fare>62.00</fare>
</exit>
<exit name="Jasin">
<fare>64.20</fare>
</exit>
<exit name="Tangkak">
<fare>65.60</fare>
</exit>
<exit name="Pagoh">
<fare>69.40</fare>
</exit>
<exit name="Yong Peng (U)">
<fare>75.20</fare>
</exit>
<exit name="Ayer Hitam">
<fare>77.90</fare>
</exit>
<exit name="Spg. Renggam">
<fare>80.80</fare>
</exit>
<exit name="Sedenak">
<fare>83.30</fare>
</exit>
<exit name="Kulai">
<fare>85.20</fare>
</exit>
<exit name="Senai (U)">
<fare>86.40</fare>
</exit>
<exit name="Skudai">
<fare>86.80</fare>
</exit>
</entry>
</plazas>
```

APPENDIX D

PHP SCRIPTS FOR SEARCH NEAREST TOLL STATION

```
<?php
//require("phpsqlsearch_dbinfo.php");

// Get parameters from URL
$center_lat = $_GET["lat"];
$center_lng = $_GET["lng"];
$radius = $_GET["radius"];

// Start XML file, create parent node
$dom = new DOMDocument("1.0");
$node = $dom->createElement("markers");
$parnode = $dom->appendChild($node);

// Opens a connection to a mySQL server
$connection=mysql_connect ('localhost', 'root', 'mysql');
if (!$connection) {
  die("Not connected : " . mysql_error());
}

// Set the active mySQL database
$db_selected = mysql_select_db('north_south_expressway', $connection);
if (!$db_selected) {
  die ("Can\'t use db : " . mysql_error());
}

// Search the rows in the markers table
$query = sprintf("SELECT  name, lat, lng, ( 3959* acos( cos( radians('%s') ) * cos( radians( lat ) ) * cos( radians( lng ) -
radians('%s') ) + sin( radians('%s') ) * sin( radians( lat ) ) ) ) AS distance FROM entry HAVING distance < '%s' ORDER BY
distance LIMIT 0 , 1",
  mysql_real_escape_string($center_lat),
  mysql_real_escape_string($center_lng),
  mysql_real_escape_string($center_lat),
  mysql_real_escape_string($radius));
$result = mysql_query($query);

if (!$result) {
  die("Invalid query: " . mysql_error());
}

header("Content-type: text/xml");

// Iterate through the rows, adding XML nodes for each
while ($row = @mysql_fetch_assoc($result)){
  $node = $dom->createElement("marker");
  $newnode = $parnode->appendChild($node);
  $newnode->setAttribute("name", $row['name']);
// $newnode->setAttribute("address", $row['address']);
  $newnode->setAttribute("lat", $row['lat']);
  $newnode->setAttribute("lng", $row['lng']);
  $newnode->setAttribute("distance", $row['distance']);
}

echo $dom->saveXML();
?>
```

APPENDIX E
SOURCE CODES FOR DISPLAY MAP

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
        <maps:Map     xmlns:maps="com.google.maps.*"     id="map"     mapevent_mapready="onMapReady(event)"
width="100%" height="100%"
                        url="http://localhost"                    sensor=                "true"
key="ABQIAAAAckGGZPBUbdEHyikOnJMaWRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxTSKoJS776YBnwqN2UvA1m14-
r56g"/>
        <mx:Script>
                <![CDATA[
                        //basic map
                        import com.google.maps.LatLng;
                        import com.google.maps.Map;
                        import com.google.maps.MapEvent;
                        import com.google.maps.MapType;
                        //add control
                        import com.google.maps.controls.ZoomControl;
                        import com.google.maps.controls.MapTypeControl;
                        import com.google.maps.controls.PositionControl;

                        private function onMapReady(event:Event):void {
                                this.map.setCenter(new          LatLng(4.532618,101.777344),          7,
MapType.NORMAL_MAP_TYPE);
                        //setting contol functions
                                this.map.addControl(new ZoomControl());
                                this.map.addControl(new PositionControl());
                                this.map.addControl(new MapTypeControl());
                        }
                ]]>
        </mx:Script>
</mx:WindowedApplication>
```

APPENDIX F
SOURCE CODES FOR DRIVING DIRECTIONS

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
<maps:Map xmlns:maps="com.google.maps.*" id="map" mapevent_mapready="onMapReady(event)" width="100%"
height="100%"
                              url="http://localhost" sensor= "true"
key="ABQIAAAAckGGZPBUbdEHyikOnJMaWRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxTSKoJS776YBnwqN2UvA1m14-
r56g"/>
        <mx:Panel title="Travel Cost Calculator" width="100%" height="170" bottom="0">
                <mx:HBox>
                        <mx:Label text="From: " width="70"/>
                        <mx:TextInput id="from" text=" " width="100%"/>
                </mx:HBox>
                <mx:HBox>
                        <mx:Label text="To: " width="70"/>
                        <mx:TextInput id="to" text=" " width="100%"/>
                </mx:HBox>

                <mx:Button id="getDirections" label="Get Directions" click="processDirections(event);"/>
                <mx:Button id="getTurnByTurnDirections" label="Get Next Step" click="processTurnByTurn();"
enabled="false"/>
                <mx:Text id="step" htmlText="" width="100%"/>
        </mx:Panel>
        <mx:Script>
                <![CDATA[
                        //basic map
                        import com.google.maps.*;
                        import com.google.maps.overlays.*;
                        import com.google.maps.services.*;
                        //add control
                        import com.google.maps.controls.ZoomControl;
                        import com.google.maps.controls.MapTypeControl;
                        import com.google.maps.controls.PositionControl;

                        import mx.controls.Alert;
                        import flash.events.Event;

                        private var dir:Directions;
                        private var turnCounter:uint = 0;

                        private function onMapReady(event:Event):void {
                                this.map.setCenter(new LatLng(4.532618,101.777344), 7,
MapType.NORMAL_MAP_TYPE);
                                //setting contol functions
                                this.map.addControl(new ZoomControl());
                                this.map.addControl(new PositionControl());
                                this.map.addControl(new MapTypeControl());
                                dir = new Directions();
                                dir.addEventListener(DirectionsEvent.DIRECTIONS_SUCCESS, onDirLoad);
                                dir.addEventListener(DirectionsEvent.DIRECTIONS_FAILURE, onDirFail);
                        }

                        private function processDirections(event:Event):void {
                                dir.load("from: " + from.text + " to: " + to.text);
                                getTurnByTurnDirections.enabled = true;

                                // Reset turnCounter to zero for new directions
                                turnCounter = 0;
                                step.htmlText = "Start at " + from.text;
                        }

                        private function onDirFail(event:DirectionsEvent):void {
                                Alert.show("Status: " + event.directions.status);
                                step.htmlText = "";
                        }

                        private function onDirLoad(event:DirectionsEvent):void {
                                //var dir:Directions = event.directions;
                                var startMarker:Marker;
```

```
                                        var endMarker:Marker;

                                        map.clearOverlays();
                                        map.addOverlay(dir.createPolyline());
                                        map.setZoom(map.getBoundsZoomLevel(dir.bounds));
                                        map.setCenter(dir.bounds.getCenter());

                                        startMarker = new Marker(dir.getRoute(0).startGeocode.point, new
MarkerOptions({fillStyle: {color:Color.BLUE}}));
                                        endMarker = new Marker(dir.getRoute(0).endGeocode.point, new
MarkerOptions({fillStyle: {color:Color.BLUE}}));
                                        map.addOverlay(startMarker);
                                        map.addOverlay(endMarker);
                              }

                              private function processTurnByTurn():void {

                                        var stepText:String;
                                        var stepMarker:Marker;
                                        turnCounter++;

                                        if (turnCounter <= dir.getRoute(0).numSteps) {
                                                  stepText = dir.getRoute(0).getStep(turnCounter-1).descriptionHtml;
                                                  stepMarker = new Marker(dir.getRoute(0).getStep(turnCounter-
1).latLng, new MarkerOptions({label: turnCounter.toString()}));
                                                  map.addOverlay(stepMarker);
                                                  step.htmlText = "Step " + turnCounter + ": " + stepText;
                                        } else {
                                                  getTurnByTurnDirections.enabled = false;
                                                  step.htmlText = "Arrive at " + to.text + " : " +
dir.getRoute(0).summaryHtml;
                                        }
                              }




                    ]]>
          </mx:Script>
</mx:WindowedApplication>
```

APPENDIX G

SOURCE CODES FOR FUEL CONSUMPTION COST CALCULATOR

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
        <maps:Map xmlns:maps="com.google.maps.*" id="map" mapevent_mapready="onMapReady(event)"
width="100%" height="100%"
                          url="http://localhost" sensor= "true"
key="ABQIAAAAckGGZPBUbdEHyikOnJMaWRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxTSKoJS776YBnwqN2UvA1m14-
r56g"/>
        <mx:Panel title="Travel Cost Calculator" width="30%" height="350" bottom="0" left="0" >
                <mx:HBox>
                        <mx:Label text="From: " width="100"/>
                        <mx:TextInput id="from" text=" " width="100%"/>
                </mx:HBox>
                <mx:HBox>
                        <mx:Label text="To: " width="100"/>
                        <mx:TextInput id="to" text=" " width="100%"/>
                </mx:HBox>
                <mx:HBox>
                        <mx:Label text="Total Distance:  " width="100"/>
                        <mx:TextInput id="distance" htmlText="0.000" width="100%"/>
                </mx:HBox>
                <mx:HBox>
                        <mx:Label text="Fuel Price (RM):              " width="100"/>
                        <mx:TextInput id="price" text="1.80" width="100%"/>
                </mx:HBox>
                <mx:HBox>
                        <mx:Label text="Known km/l :   " width="100"/>
                        <mx:TextInput id="kml" text="20" width="100%"/>
                </mx:HBox>
                <mx:HBox>
                        <mx:Label text="Engine Size:    " width="100"/>
                        <mx:ComboBox  id="cmbox" width="150" >
                                <mx:ArrayCollection>
                                        <mx:Object label="less than 1.0cc" data="22.53KM/L"/>
                                        <mx:Object label="1.0cc - 1.5cc" data="20.7KM/L"/>
                                        <mx:Object label="1.5cc - 2.0cc" data="18.2KM/L"/>
                                        <mx:Object label="more than 2.0cc" data="14.84KM/L"/>

                                </mx:ArrayCollection>
                        </mx:ComboBox>
                </mx:HBox>

                <mx:Button id="calculator" label="Calculate!" click="calculate();"/>
                <mx:Button id="getDirections" label="Get Directions" click="processDirections(event);"/>
                <mx:Button id="getTurnByTurnDirections" label="Get Next Step" click="processTurnByTurn();"
enabled="false"/>
                <mx:Text id="step" htmlText="" width="100%"/>
        </mx:Panel>
        <mx:Script>
                <![CDATA[
                        import com.google.maps.*;
                        import com.google.maps.controls.MapTypeControl;
                        import com.google.maps.controls.PositionControl;
                        import com.google.maps.controls.ZoomControl;
                        import com.google.maps.overlays.*;
                        import com.google.maps.services.*;

                        import flash.events.Event;

                        import flashx.textLayout.formats.Float;

                        import mx.collections.ArrayCollection;
                        import mx.controls.Alert;
                        import mx.events.DropdownEvent;

                        private var dir:Directions;
                        private var turnCounter:uint = 0;

                        private function onMapReady(event:Event):void {
```

```
                                    this.map.setCenter(new LatLng(4.532618,101.777344), 7,
MapType.NORMAL_MAP_TYPE);
                                    //setting contol functions
                                    this.map.addControl(new ZoomControl());
                                    this.map.addControl(new PositionControl());
                                    this.map.addControl(new MapTypeControl());
                                    dir = new Directions();
                                    dir.addEventListener(DirectionsEvent.DIRECTIONS_SUCCESS, onDirLoad);
                                    dir.addEventListener(DirectionsEvent.DIRECTIONS_FAILURE, onDirFail);
                        }

                        private function processDirections(event:Event):void {
                                    dir.load("from: " + from.text + " to: " + to.text);
                                    getTurnByTurnDirections.enabled = true;

                                    // Reset turnCounter to zero for new directions
                                    turnCounter = 0;
                                    step.htmlText = "Start at " + from.text;
                        }

                        private function onDirFail(event:DirectionsEvent):void {
                                    Alert.show("Status: " + event.directions.status);
                                    step.htmlText = "";
                        }

                        private function onDirLoad(event:DirectionsEvent):void {
                                    //var dir:Directions = event.directions;
                                    var startMarker:Marker;
                                    var endMarker:Marker;

                                    map.clearOverlays();
                                    map.addOverlay(dir.createPolyline());
                                    map.setZoom(map.getBoundsZoomLevel(dir.bounds));
                                    map.setCenter(dir.bounds.getCenter());

                                    startMarker = new Marker(dir.getRoute(0).startGeocode.point, new
MarkerOptions({fillStyle: {color:Color.BLUE}}));
                                    endMarker = new Marker(dir.getRoute(0).endGeocode.point, new
MarkerOptions({fillStyle: {color:Color.BLUE}}));
                                    map.addOverlay(startMarker);
                                    map.addOverlay(endMarker);

                                    //to display distance in km
                                    distance.htmlText=dir.getRoute(0).distanceHtml;
                        }


                        private function processTurnByTurn():void  {

                                    var stepText:String;
                                    var stepMarker:Marker;
                                    turnCounter++;

                                    if (turnCounter <= dir.getRoute(0).numSteps) {
                                                stepText = dir.getRoute(0).getStep(turnCounter-1).descriptionHtml;
                                                stepMarker = new Marker(dir.getRoute(0).getStep(turnCounter-
1).latLng, new MarkerOptions({label: turnCounter.toString()}));
                                                map.addOverlay(stepMarker);
                                                step.htmlText = "Step " + turnCounter + ": " + stepText;
                                    } else {
                                                getTurnByTurnDirections.enabled = false;
                                                step.htmlText = "Arrive at " + to.text + " : " +
dir.getRoute(0).durationHtml;

                                    }
                        }

                        private function calculate():void {
                                    var constant:Number=new Number (1000);
                                    //distance in meters
                                    var dstance:Number=dir.getRoute(0).distance;
                                    //to convert distance in meters into kilometers
                                    var dstance_km:Number =  dstance / constant;
                                    var kmperl:Number = Number (kml.text);
                                    //1 MPG = 0.43KM/L
                                    //mpg is actually km/liter in the case below (i just name it mpg)
```

```
                                        //choose the car cc
                                        if (kmperl == Number(km1.text)){
                                                var cpkm:Number= Number(price.text)/kmperl;
                                                var total_price:Number=dstance_km*cpkm;
                                                Alert.show("Total Cost:RM" + total_price.toFixed(2));
                                        }

                                        else if (cmbox.selectedLabel == "less than 1.0cc") {
                                                var mpg1:Number=new Number(22.53);
                                                //cost per km (cpkm)
                                                var cpkm1:Number= Number(price.text) / mpg1;
                                                //total cost
                                                var total_price1:Number= dstance_km*cpkm1;
                                                Alert.show("Total Cost:RM" + total_price1.toFixed(2));
                                        }

                                        else if (cmbox.selectedLabel == "1.0cc - 1.5cc") {
                                                var mpg2:Number=new Number(20.7);
                                                //cost per km (cpkm)
                                                var cpkm2:Number= Number(price.text) / mpg2;
                                                //total cost
                                                var total_price2:Number= dstance_km*cpkm2;
                                                Alert.show("Total Cost:RM" + total_price2.toFixed(2));
                                        }

                                        else if (cmbox.selectedLabel == "1.5cc - 2.0cc") {
                                                var mpg3:Number=new Number(18.2);
                                                //cost per km (cpkm)
                                                var cpkm3:Number= Number(price.text) / mpg3;
                                                //total cost
                                                var total_price3:Number= dstance_km*cpkm3;
                                                Alert.show("Total Cost:RM" + total_price3.toFixed(2));
                                        }

                                        else if (cmbox.selectedLabel == "more than 2.0cc") {
                                                var mpg4:Number=new Number(14.84);
                                                //cost per km (cpkm)
                                                var cpkm4:Number= Number(price.text) / mpg4;
                                                //total cost
                                                var total_price4:Number= dstance_km*cpkm4;
                                                Alert.show("Total Cost:RM" + total_price4.toFixed(2));
                                        }
                                }



                ]]>
        </mx:Script>
</mx:WindowedApplication>
```

APPENDIX H

SOURCE CODES FOR AUTO-DETECT TOLL STATION

```
<?xml version="1.0" encoding="utf-8"?>
        <mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:maps="com.google.maps.*"
layout="absolute" width="100%" height="100%" creationComplete="rate.send()">
                <mx:HTTPService id="rate" resultFormat="e4x" url="test.xml" result="handleResult(event)"/>
                <mx:Panel title="Toll Cost Calculator" width="100%" height="100%">
                        <mx:HDividedBox width="100%" height="100%">
                                <mx:VBox width="50%" height="100%">
                                        <mx:Label text= "Driving Direction" />
                                        <mx:HBox>
                                                <mx:Label
                                                        text="From: " width="100"/>
                                                <mx:TextInput
                                                        id="from" text="Seberang Perai, Penang"
width="100%" />
                                        </mx:HBox>
                                        <mx:HBox>
                                                <mx:Label text="To: " width="100"/>
                                                <mx:TextInput id="to" text="Ipoh, Perak" width="100%"/>
                                        </mx:HBox>
                                        <mx:Button id="getDirections" label="Get Directions"
click="processDirections(event);"/>
                                        <mx:Button id="getTurnByTurnDirections" label="Get Next Step"
click="processTurnByTurn();" enabled="false"/>
                                        <mx:Text id="step" htmlText="" width="100%"/>
                                        <mx:Label text="Radius: "       width="100"/>
                                        <mx:ComboBox id="radius">
                                                <mx:dataProvider>
                                                        <mx:Array>
                                                                <mx:String>25</mx:String>
                                                                <mx:String>100</mx:String>
                                                                <mx:String>200</mx:String>
                                                        </mx:Array>
                                                </mx:dataProvider>
                                        </mx:ComboBox>
                                        <mx:Button
                                                id="submitButton1" label="Nearest Toll Entry" width="200"
                                                click="searchEntry(event);"/>
                                                <mx:Button
                                                        id="submitButton2" label="Nearest Toll Exit"
width="200"
                                                        click="searchExit(event);"/>
                                                <mx:Label text="Other Informations: " width="150" />
                                                <mx:HBox>
                                                <mx:Label text="Fuel Price (RM): " width="100"/>
                                                <mx:TextInput id="price" text="1.90" width="100%"/>
                                        </mx:HBox>

                                        <mx:HBox>
                                                <mx:Label text="Engine Size: "  width="100"/>
                                                <mx:ComboBox  id="cmbox" prompt="Please select.."
width="150" >
                                                        <mx:dataProvider>
                                                                <mx:Object label="less than 1.0cc"
data="22.53"/>
                                                                <mx:Object label="1.0cc - 1.5cc"
data="20.7"/>
                                                                <mx:Object label="1.5cc - 2.0cc"
data="18.2"/>
                                                                <mx:Object label="more than 2.0cc"
data="14.84"/>
                                                                </mx:dataProvider>
                                                </mx:ComboBox>
                                        </mx:HBox>

                                        <mx:Label text="Travel Calculator" />
                                        <mx:HBox>
                                                <mx:Label text="Entry: " width="100"/>
                                                        <mx:ComboBox id="entryCB"
dataProvider="{plazaList}" prompt="Please select" labelField="@name" />
                                        </mx:HBox>
                                        <mx:HBox>
                                                <mx:Label text="Exit: " width="100"/>
                                                        <mx:ComboBox id="exitCB"
dataProvider="{entryCB.selectedItem.exit}" prompt="Please select" labelField="@name" />
                                        </mx:HBox>

                                        <mx:Label text= "Summary" />
                                        <mx:HBox>
                                                <mx:Label text="Total Distance: " width="100"/>
                                                <mx:Label id="distance" htmlText="" />
                                        </mx:HBox>
                                        <mx:HBox>
                                                <mx:Label text="Toll Cost: " width="100"/>
                                                <mx:Label text="RM {exitCB.selectedItem.fare}"/>
                                        </mx:HBox>
                                        <mx:Button id="calculator" label="Calculate!" click="calculate();"/>
                        </mx:VBox>
                                <maps:Map
                                        id="map"

        key="ABQIAAAAckGGZPBUbdEHyikOnJMaWRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxTSKoJS776YBnwqN2Uv
Alm14-r56g"
                                        url="http://localhost" sensor="true"
                                        mapevent_mapready="onMapReady(event)"
                                        width="100%" height="100%"/>
                        </mx:HDividedBox>
                </mx:Panel>
                <mx:Script>
                        <![CDATA[
                                import com.google.maps.InfoWindowOptions;
                                import com.google.maps.LatLng;
                                import com.google.maps.LatLngBounds;
                                import com.google.maps.Map;
                                import com.google.maps.MapEvent;
                                import com.google.maps.MapMouseEvent;
                                import com.google.maps.MapType;
                                import com.google.maps.controls.ZoomControl;
                                import com.google.maps.overlays.Marker;
                                import com.google.maps.overlays.MarkerOptions;
                                import com.google.maps.services.*;
                                import com.google.maps.services.ClientGeocoder;
                                import com.google.maps.services.GeocodingEvent;

                                import flash.events.Event;

                                import flashx.textLayout.formats.Float;

                                import mx.collections.ArrayCollection;
                                import mx.controls.Alert;
                                import mx.core.IUIComponent;
                                import mx.events.DropdownEvent;
                                import mx.events.ListEvent;
                                import mx.rpc.events.ResultEvent;

                                private var dir:Directions;
                                private var turnCounter:uint = 0;
                                //XML file for combo box purpose
                                [Bindable]
                                private var plazaList:XMLList;

                                private function handleResult(event:ResultEvent):void{

                                        plazaList = event.result.entry as XMLList;

                                }

                                private function onMapReady(event:Event):void {
                                        map.enableScrollWheelZoom();
                                        map.enableContinuousZoom();
                                        map.setCenter(new LatLng(4.532618,101.777344), 7);
                                        map.addControl(new ZoomControl());
                                        dir = new Directions();
                                        dir.addEventListener(DirectionsEvent.DIRECTIONS_SUCCESS,
onDirLoad);
                                        dir.addEventListener(DirectionsEvent.DIRECTIONS_FAILURE,
onDirFail);
```

```
					}

					private function searchEntry(event:Event):void {
						var geocoder:ClientGeocoder = new ClientGeocoder();
						geocoder.addEventListener(
							GeocodingEvent.GEOCODING_SUCCESS,
							function(event:GeocodingEvent):void {
								var placemarks:Array =
event.response.placemarks;

								if (placemarks.length > 0) {
									searchEntryNear(placemarks[0].point);
								}
							});
						geocoder.addEventListener(
							GeocodingEvent.GEOCODING_FAILURE,
							function(event:GeocodingEvent):void {
								Alert.show("Geocoding failed");
							});
						geocoder.geocode(from.text);
					}

					public function searchEntryNear(center:LatLng):void {
						var searchUrl:String =
'http://localhost/phpsqlsearch_genxml.php?lat=' + center.lat() + '&lng=' + center.lng() + '&radius=' + radius.text;
						var xmlString:URLRequest = new URLRequest(searchUrl);
						var xmlLoader:URLLoader = new URLLoader(xmlString);
						xmlLoader.addEventListener("complete", parseLocationsXml);
					}

					private function searchExit(event:Event):void {
						var geocoder:ClientGeocoder = new ClientGeocoder();
						geocoder.addEventListener(
							GeocodingEvent.GEOCODING_SUCCESS,
							function(event:GeocodingEvent):void {
								var placemarks:Array =
event.response.placemarks;

								if (placemarks.length > 0) {
									searchEntryNear(placemarks[0].point);
								}
							});
						geocoder.addEventListener(
							GeocodingEvent.GEOCODING_FAILURE,
							function(event:GeocodingEvent):void {
								Alert.show("Geocoding failed");
							});
						geocoder.geocode(to.text);
					}

					public function searchExitNear(center:LatLng):void {
						var searchUrl:String =
'http://localhost/phpsqlsearch_genxml.php?lat=' + center.lat() + '&lng=' + center.lng() + '&radius=' + radius.text;
						var xmlString:URLRequest = new URLRequest(searchUrl);
						var xmlLoader:URLLoader = new URLLoader(xmlString);
						xmlLoader.addEventListener("complete", parseLocationsXml);
					}

					public function parseLocationsXml(event:Event):void{
						map.clearOverlays();
						var entryXML:XML = new XML(event.target.data);
						var entry:XMLList = entryXML..marker;
						var entryCount:int = entry.length();
						if (entryCount == 0) {
							map.setCenter(new LatLng(4.532618,101.777344), 7);
							Alert.show("No nearest toll station found. Try a different
address");

							return;
						}
						var bounds:LatLngBounds = new LatLngBounds();
						for (var i:Number = 0; i < entryCount; i++) {
							var markerXml:XML = entry[i];
							var name:String = markerXml.@name;
							//var address:String = markerXml.@address;
							var distance:Number = new
Number(markerXml.@distance);

							var latlng:LatLng = new LatLng(markerXml.@lat,
markerXml.@lng);

							//var storeInfo:String = "<b>" + name + "</b>\n<br/>";
```

```
                                        var marker:Marker = createMarker(latlng, name);
                                        //storeLocations.addItem({ Distance: distance.toFixed(2),
Marker: marker});

                                        bounds.extend(latlng);
                                        map.addOverlay(marker);
                                }
                                //storeLocations.refresh();
                                map.setCenter(bounds.getCenter(),
map.getBoundsZoomLevel(bounds));
                        }

                        public function createMarker(latlng:LatLng, name:String):Marker {
                                var marker:Marker = new Marker(latlng, new MarkerOptions({fillStyle:
{color:0x00ffff}}));
                                marker.addEventListener(MapMouseEvent.CLICK,
function(e:MapMouseEvent):void {
                                        marker.openInfoWindow(new
InfoWindowOptions({contentHTML:name}));
                                });
                                return marker;
                        }

                        private function processDirections(event:Event):void {
                                dir.load("from: " + from.text + " to: " + to.text);
                                getTurnByTurnDirections.enabled = true;

                                // Reset turnCounter to zero for new directions
                                turnCounter = 0;
                                step.htmlText = "Start at " + from.text;
                        }

                        private function onDirFail(event:DirectionsEvent):void {
                                Alert.show("Status: " + event.directions.status);
                                step.htmlText = "";
                        }

                        private function onDirLoad(event:DirectionsEvent):void {
                                //var dir:Directions = event.directions;
                                var startMarker:Marker;
                                var endMarker:Marker;

                                map.clearOverlays();
                                map.addOverlay(dir.createPolyline());
                                map.setZoom(map.getBoundsZoomLevel(dir.bounds));
                                map.setCenter(dir.bounds.getCenter());

                                startMarker = new Marker(dir.getRoute(0).startGeocode.point, new
MarkerOptions({fillStyle: {color:0xff00ff}}));
                                endMarker = new Marker(dir.getRoute(0).endGeocode.point, new
MarkerOptions({fillStyle: {color:0xff00ff}}));
                                map.addOverlay(startMarker);
                                map.addOverlay(endMarker);

                                //to display distance in km
                                distance.htmlText=dir.getRoute(0).distanceHtml;
                        }


                        private function processTurnByTurn():void {

                                var stepText:String;
                                var stepMarker:Marker;
                                turnCounter++;

                                if (turnCounter <= dir.getRoute(0).numSteps) {
                                        stepText = dir.getRoute(0).getStep(turnCounter-
1).descriptionHtml;
                                        stepMarker = new
Marker(dir.getRoute(0).getStep(turnCounter-1).latLng, new MarkerOptions({label: turnCounter.toString()}));
                                        map.addOverlay(stepMarker);
                                        step.htmlText = "Step " + turnCounter + ": " + stepText;
                                } else {
                                        getTurnByTurnDirections.enabled = false;
                                        step.htmlText = "Arrive at " + to.text + " : " +
dir.getRoute(0).durationHtml;

                                }
```

```
                                        }

                                        private function calculate():void {
                                                var constant:Number=new Number (1000);
                                                //distance in meters
                                                var dstance:Number=dir.getRoute(0).distance;
                                                //to convert distance in meters into kilometers
                                                var dstance_km:Number =  dstance / constant;

                                                //1 MPG = 0.43KM/L
                                                //mpg is actually km/liter in the case below (i just name it mpg)

                                                //choose the car cc


                                                if (cmbox.selectedLabel == "less than 1.0cc") {
                                                        var mpg1:Number=new Number(22.53);
                                                        //cost per km (cpkm)
                                                        var cpkm1:Number= Number(price.text) / mpg1;
                                                        //total cost
                                                        var fuel_cost1:Number= dstance_km*cpkm1;
                                                        var
total1:Number=fuel_cost1+parseFloat(exitCB.selectedItem.fare);
                                                        Alert.show("Total Cost:RM" + total1.toFixed(2));
                                                }

                                                else if (cmbox.selectedLabel == "1.0cc - 1.5cc") {
                                                        var mpg2:Number=new Number(20.7);
                                                        //cost per km (cpkm)
                                                        var cpkm2:Number= Number(price.text) / mpg2;
                                                        //total cost
                                                        var fuel_cost2:Number= dstance_km*cpkm2;
                                                        var
total2:Number=fuel_cost2+parseFloat(exitCB.selectedItem.fare);
                                                        Alert.show("Total Cost:RM" + total2.toFixed(2));
                                                }

                                                else if (cmbox.selectedLabel == "1.5cc - 2.0cc") {
                                                        var mpg3:Number=new Number(18.2);
                                                        //cost per km (cpkm)
                                                        var cpkm3:Number= Number(price.text) / mpg3;
                                                        //total cost
                                                        var fuel_cost3:Number= dstance_km*cpkm3;
                                                        var
total3:Number=fuel_cost3+parseFloat(exitCB.selectedItem.fare);
                                                        Alert.show("Total Cost:RM" + total3.toFixed(2));
                                                }

                                                else if (cmbox.selectedLabel == "more than 2.0cc") {
                                                        var mpg4:Number=new Number(14.84);
                                                        //cost per km (cpkm)
                                                        var cpkm4:Number= Number(price.text) / mpg4;
                                                        //total cost
                                                        var fuel_cost4:Number= dstance_km*cpkm4;
                                                        var
total4:Number=fuel_cost4+parseFloat(exitCB.selectedItem.fare);
                                                        Alert.show("Total Cost:RM" + total4.toFixed(2));
                                                }
                                        }

                        ]]>
                </mx:Script>

</mx:WindowedApplication>
```