

Traffic Monitoring Project

Group #13

Xuan Li

Chih-Ting Cho

Ting-Chieh Huang

Jonathan Hong

Jan Racoma

Kevin Cundey

Software Engineering 14:332:452

Second Report

<https://sites.google.com/site/452trafficmonitoring/home>

Individual Contributions Breakdown

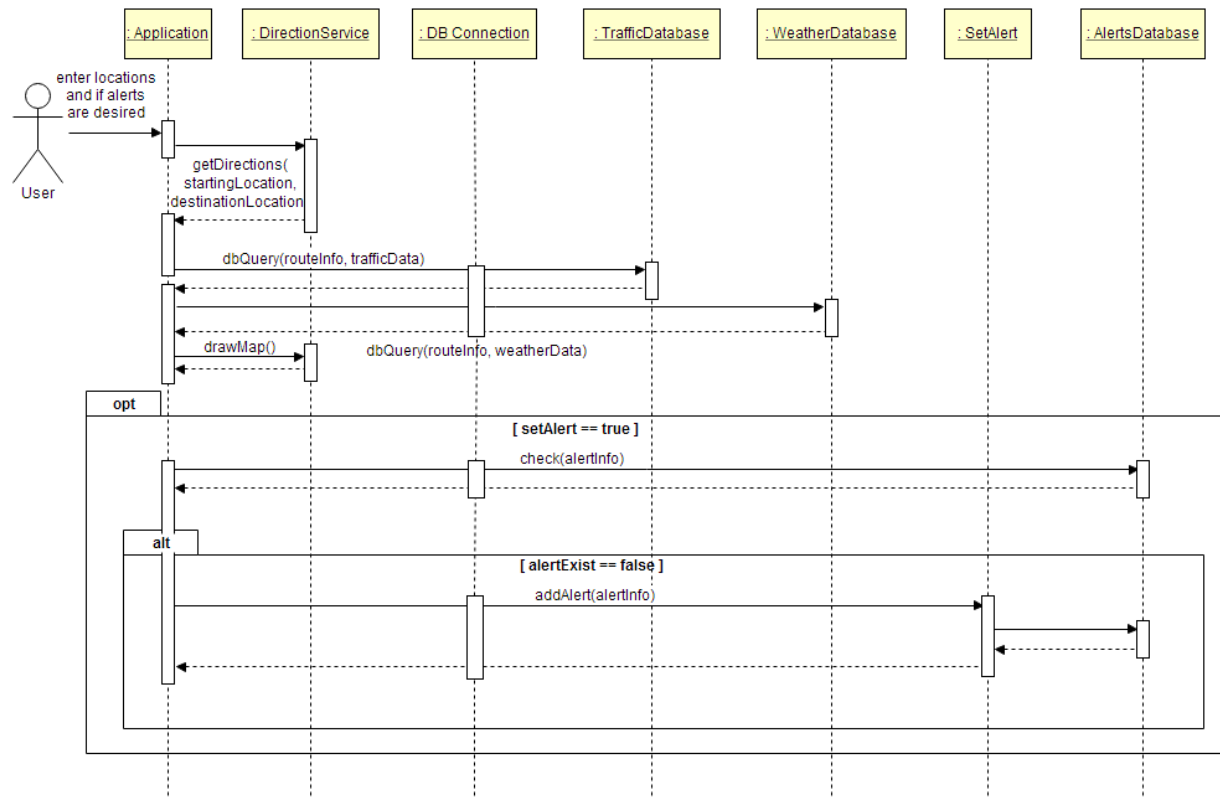
All team members contributed equally.

Table of Contents

<u>Page Title</u>	<u>Page No.</u>
<u>Section 1: Interaction Diagrams</u>	4
1.1 Interaction Diagrams	4
<u>Section 2: Class Diagram and Interface Specifications</u>	7
2.1 Class Diagram	7
2.2 Data Types and Operation Signatures	8
2.3 Traceability Matrix	11
<u>Section 3: System Architecture and System Design</u>	13
<u>Section 4: Algorithms and Data Structures</u>	17
<u>Section 5: User Interface Design and Implementation</u>	18
<u>Section 6: Design of Tests</u>	19
<u>Section 7: Project Management and Plan of Work</u>	21
<u>Section 8: References</u>	24

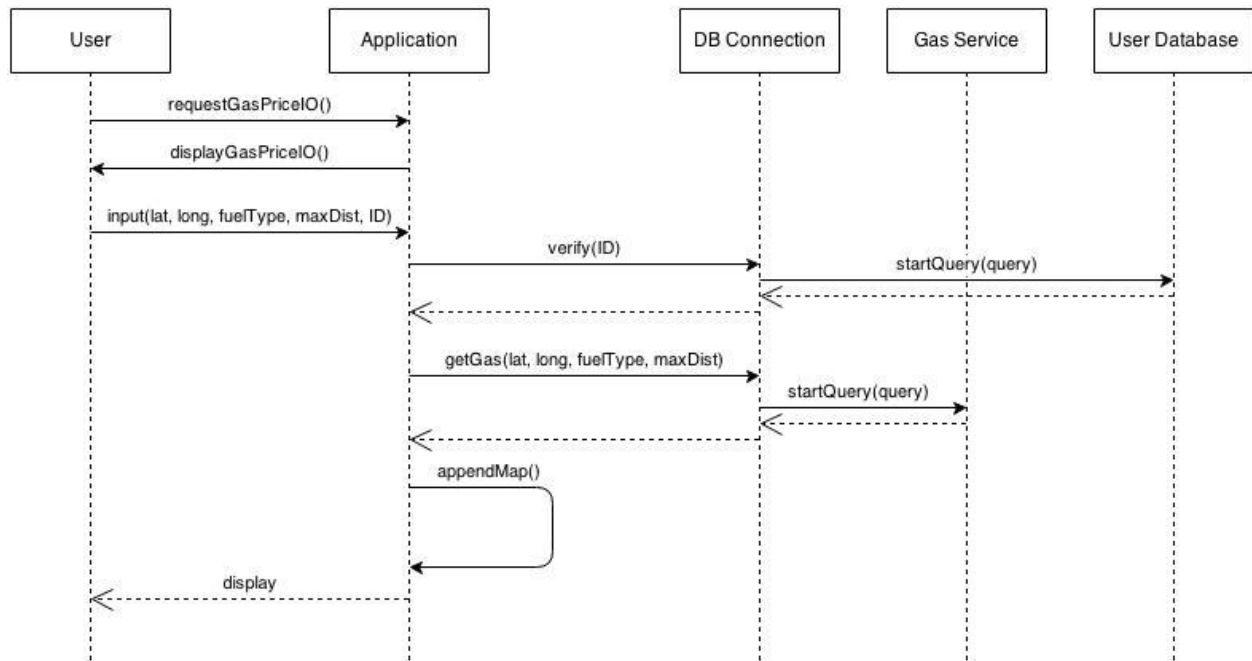
Interaction Diagrams

Use Case 1:



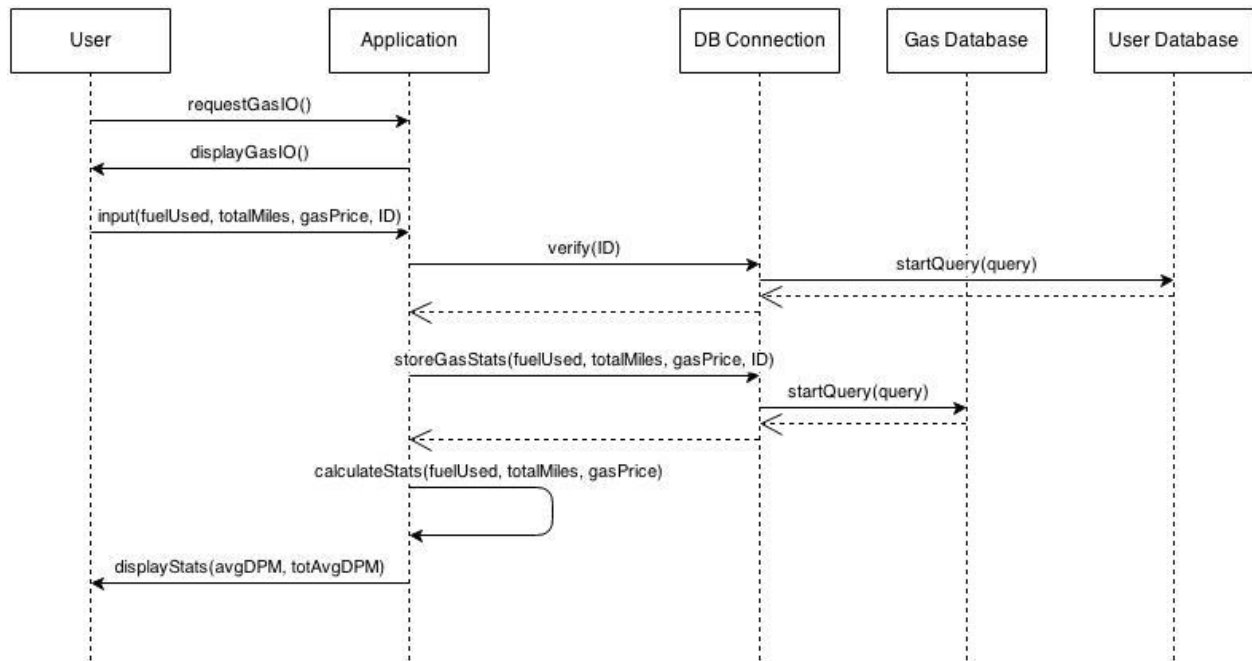
The system will request directions from DirectionService. It will then take the route information and retrieve traffic data from the traffic and weather database along the intended route. After all information is retrieved from the databases, the application will draw the map from the direction service. If the user selected to set a new alert, the application will query the AlertsDatabase if the alert exists. If it does not exist, the application will pass the alert information to be stored into the database. The “High Cohesion Principle” is followed as there are no computations in this particular use case. Since the system heavily relies on communication between the DB Connection and the databases, the “Low Coupling Cohesion Principle” is difficult to employ.

Use Case 2:



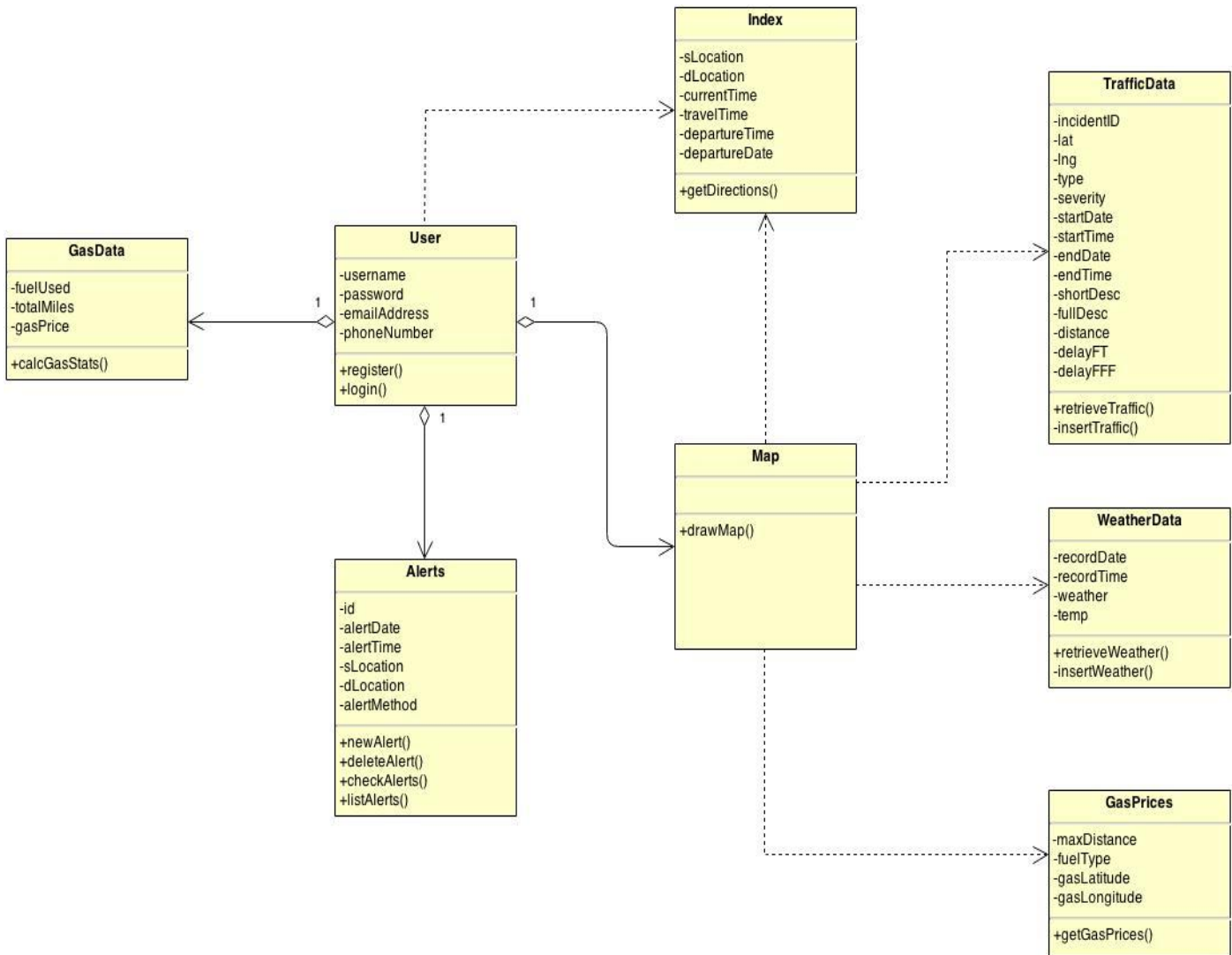
The user will request to view gas prices in the nearby area. The application will then display the gas price interface. After the user inputs the data, the application will communicate to the database connection to verify that the user is valid. After receiving the response, the application request for gas price data. The database connection will contact the gas service and return the response. After this is accomplished the application will then append the locations on to the interactive map, then display this to the user. The “Low Coupling Principle” is not feasible as the program heavily relies on communication between the application and various databases and services. The “High Cohesion Principle” is followed as there are not many computation responsibilities required in this use case. The “Expert Doer Principle” is assigned to the application object as it “knows” the most and uses this knowledge to perform tasks within itself.

Use Case 8:



The application was assigned all three types of responsibilities (knowing, doing, and communicating.) This is because this is the hub for the user's interaction. Although the application follows has all types of responsibilities, it follows two important design principles for this use-case. The application only does one computation, following the "High Cohesion Principle," and it follows the "Expert Doer Principle" as it does not off-load many tasks. Unfortunately, it is not feasible to design this object with the "Low Coupling Principle" as it is the main hub for user interaction.

Class Diagram



Data Types and Operation Signatures

Index:

Attributes

- sLocation:varchar -- user input starting point.
- dLocation:varchar -- user input destination.
- currentTime:time -- user input current time.
- travelTime:time -- the travel time from starting point to the user input destination.
- departureTime:time -- user input departure time.
- departureDate:date -- user input departure date.

Operations

- +getDirections(in sLocation:varchar, in dLocation:varchar) -- get the direction and show the route from starting point to the user input destination.

Map:

Operations

- +drawMap() -- invokes index class, show the direction from starting point to the user input destination on the map.

TrafficData:

Attributes

- incidentID:int -- identification of each incident that system detects
- lat:decimal -- latitude value
- lng:decimal -- longitude value
- type:int -- type of the incident
- severity:int -- severity of the incident (scale 1- 4, 4 is the most severe)
- startDate:date -- incident starting date
- startTime:time -- start time of the incident
- endDate:date -- end time of the incident
- endTime:date -- end date of the incident
- shortDesc:varchar -- short description of the incident
- fullDesc:varchar -- full description of the incident
- distance:decimal -- distance from the incidents that affected
- delayFT:decimal -- delay from typical
- delayFFF:decimal -- delay from freeflow

Operations

- +retrieveTraffic(in lat:decimal, in lng:decimal) -- retrieve the incident's latitude and longitude
- insertTraffic() -- insert the retrieved traffic into user database.

WeatherData:

Attributes

- recordDate:date -- date of weather in record
- recordTime:time -- time of weather in record
- weather:varchar -- type of weather
- temp:decimal -- temperature of weather

Operations

- +retrieveWeather(in recordDate:time, in recordTime:time) -- retrieve the weather information from weather websites based on the recorded date and time.
- insertWeather() -- insert the weather information that is retrieved from weather websites to the user database.

GasPrices:

Attributes

- maxDistance:int -- user input the maximum distance from their location
- fuelType:varchar -- the type of fuel
- gasLatitude:decimal -- gas station latitude
- gasLongitude:decimal -- gas station longitude

Operations

- +getGasPrices(in fuelType:varchar, in maxDistance:int, in gasLatitude:decimal, in gasLongitude:decimal) -- according to the input maximum distance, retrieve the nearby gas stations positions and display their prices respectively.

User:

Attributes

- username:varchar -- user input of user's name
- password:varchar -- user input of user's password
- emailAddress:varchar -- user input of user's email address
- phoneNumber:varchar -- user input of user's phone number

Operations

- +register(in username:varchar, in password:varchar, in emailAddress:varchar, in phoneNumber:varchar) -- register user's name, password, email address and phone number into system
- +login(in username:varchar, in password:varchar) -- access user's information and history by input user's name and password

GasData:

Attributes

-fuelUsed:decimal -- user input of the amount of the fuel that has been used.
-totalMiles:decimal -- user input of total miles that he/she has run with the previous input amount of fuel
-gasPrice:decimal -- invoke GasePrices class, and display the gas prices.

Operations

+calcGasStats(in fuelUsed:decimal, in totalMiles:decimal, in gasPrice:decimal) -- calculate how much user spends on gas for each mile.

Alerts:

Attributes

-id:int -- identification of alert
-alertDate:date -- user input alert date
-alertTime:time -- user input alert time
-sLocation:vchar -- user input starting location
-dLocation:vchar -- user input the destination
-alertMethod:vchar -- user input alert methods (email/text message)

Operations

+newAlert() -- allow user to create new alert
+deleteAlert() -- allow user to delete the old/existing alert
+checkAlerts() -- retrieve the existing alerts
+listAlerts() -- display the existing alerts

Traceability Matrix

Domain Concepts

Classes	D B C o n n e c t i o n	A p p l i c a t i o n	I n t e r f a c e	D i r e c t i o n S e r v i c e	G a s S e r v i c e	P a g e M a k e r	U s e r D a t a b a s e	T r a f f i c D a t a b a s e	W e a t h e r D a t a b a s e	A l e r t s D a t a b a s e	G a s D a t a b a s e
Index	X	X	X	X			X	X	X	X	
GasPrices	X		X		X		X				
UserAlert	X	X	X			X	X			X	
User	X	X	X				X				
TrafficData	X	X		X			X	X			
Map	X		X	X							
WeatherData	X	X							X		
Gas Data	X				X		X				X

There are 11 domain concepts. DB connection's responsibility is to retrieve, display, and send data to other concepts. We need it for all of our classes. In our domain Application, we have all the information about date, time, destination, starting position and user information. While we are deriving our classes, we need separate classes for weather, traffic, gas, user, alert, basic interface information, and map. Then, we have Classes Index, GasPrices, UserAlert, User, TrafficData, Map, WeatherData and GasData. Thus, for domain Gas Service, we need two

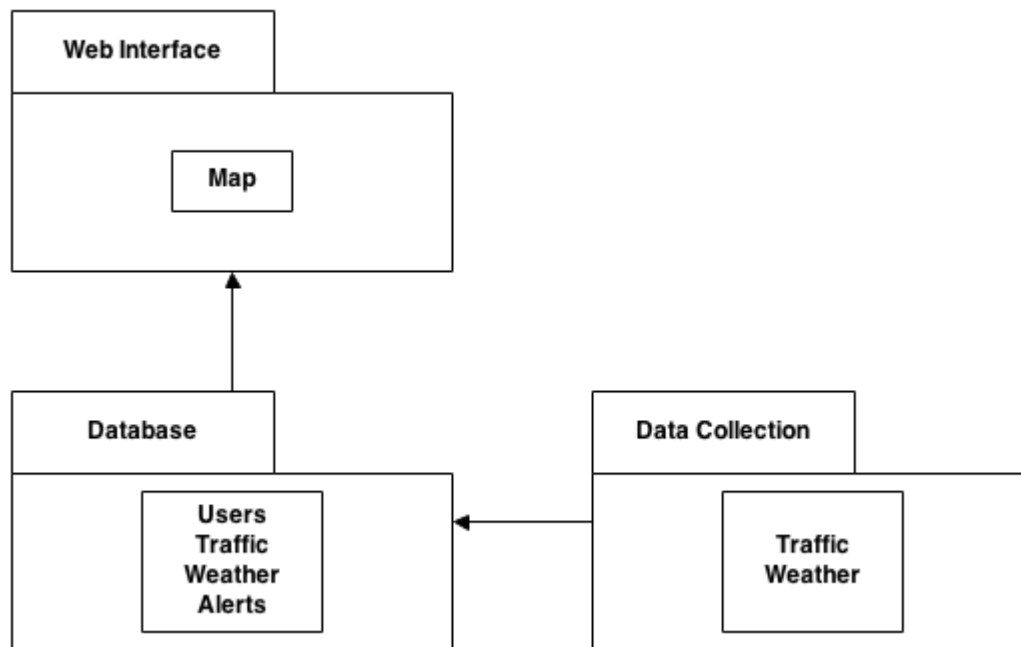
classes, GasPrices and Gas data. For domain Traffic Database, we need TrafficData class to store the traffic information. For Alert Database, we need UserAlert to allow user to create, delete and check alerts. The DirectionService needs three classes: index, trafficData and Map.

System Architecture and System Design

Architectural Styles

At the core, Traffic Monitoring follows the Client/Server model. The server component provides the core functions to the many user clients. The main server plays the roles of a web server and a database server. The web server runs and waits for incoming requests from clients. Once the initial connection has been established, the client and server begin exchanging messages using a request-response message pattern. The client controls its inputs through the main web interface and the server takes these inputs, processes, and returns response. Since the server response is similar, the Client/Server model is the best fit.

Identifying Subsystems



Mapping Subsystems to Hardware

The core server will serve both as a web server and a database server. The web server will allow access through a web browser and generate pages using PHP which will pull stored data from the MySQL database.

Persistent Data Storage

MySQL will be used to store the collected data.

trafficData Database Schema:

Field Name	Type	NULL	Default
id	int(10)	NO	0
lat	decimal(10,6)	YES	NULL
lng	decimal(10,6)	YES	NULL
type	int(1)	YES	NULL
severity	int(1)	YES	NULL
startDate	date	YES	NULL
day	varchar(4)	YES	NULL
startTime	time	YES	NULL
endDate	date	YES	NULL
endTime	time	YES	NULL
shortDesc	varchar(255)	YES	NULL
fullDesc	varchar(255)	YES	NULL
distance	decimal(4,4)	YES	NULL
delayFromTypical	decimal(4,4)	YES	NULL
delayFromFreeFlow	decimal(4,4)	YES	NULL

weatherData Database Schema:

Field Name	Type	NULL	Default
recordDate	date	NO	0000-00-00
recordTime	time	NO	00:00:00
weather	varchar(255)	YES	NULL
temp	decimal(3,1)	YES	NULL

userDatabase Database Schema:

Field Name	Type	NULL	Default
username	varchar(255)	NO	None
password	varchar(255)	NO	None
emailAddress	varchar(255)	NO	None
phoneNumber	varchar(255)	YES	NULL

alertsDatabase Database Schema:

Field Name	Type	NULL	Default
id	int(11)	NO	None
username	varchar(255)	NO	None
alertDate	date	NO	None
alertTime	time	NO	None
sLocation	varchar(255)	NO	None
dLocation	varchar(255)	NO	None
alertMethod	varchar(255)	NO	None

Network Protocol

Applications on the world web web primarily use the HyperText Transfer Protocol (HTTP) and so will this application. There is no need to re-invent the wheel of a standard.

Global Flow Control

- Execution Orderness:

The system is both *procedure-driven* and *event-driven*. The system is procedure-driven in that the user can request to use the system anytime and every user goes through the same steps. The system is event-driven because the traffic and weather databases collect data from websites at periodic intervals.

- Time Dependency:

The collection subsystem is dependent on the system time. A traffic and weather collection script runs once an hour and records data into the MySQL Database. Also,

SendAlert will also run once an hour to send out the alerts to users.

Hardware Requirements

The user is required to have functional computer with a minimum screen resolution of 800x600. The user must use a browser that is compatible with the web application. This also applies to users viewing the web application on a mobile device. The database requires 2 GB in order to store traffic and weather data and gas prices gathered from web services.

Algorithms and Data Structures

Algorithms

The main algorithm used is the calculation of average traffic intensity. The algorithm was kept simple yet inclusive of key factors. It was kept simple so that it would not greatly burden the ultimate goal of the project but needed to be extensive enough to encompass all traffic factors, such as severity which was provided by the traffic website. Weather was also gathered but simplified in that only one zip code was used for collection. It's assumed to be sufficient since the scope radius was not too large.

$$\frac{\text{Number of Incidents In Sector}}{\text{Total Incidents on Highway} * \text{Total Days of Collection}}$$

Data Structures

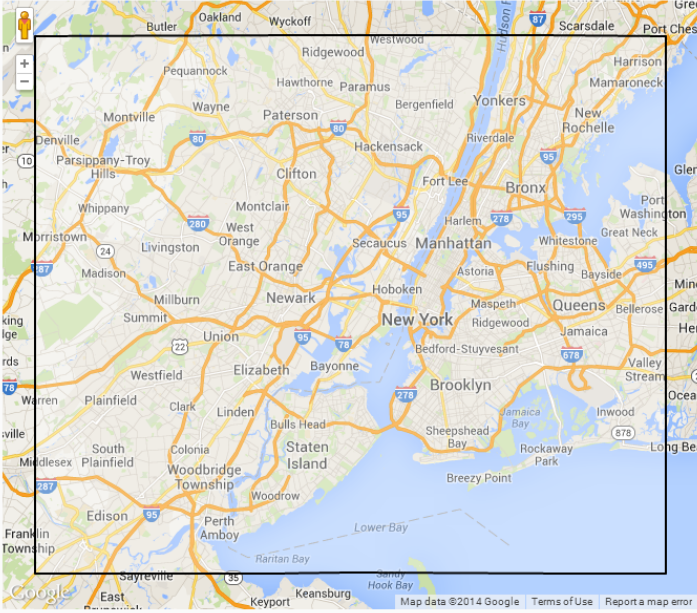
The main data structure is the MySQL Database. The complexity is mainly due to manipulation of the data in order to result in the correct query.

User Interface Design and Implementation

The user interface was not changed significantly from the initial design. Only minor tweaks occurred.

[Traffic Monitoring](#) [Alerts](#) [More Stuff](#) [Signup](#) [Login](#)

Get Directions



A Google Map of the New York City metropolitan area. A black line indicates a route starting from Butler, NJ, heading south through Paterson, NJ, then east through Manhattan, NY, and ending in Port Chester, NY. The map shows major highways like I-80, I-95, and I-287. Various city names and landmarks are labeled.

Starting Location:

Destination Location:

Departure Time:

Departure Date: +10 Days Max

Set Alert? ☐ Alert Method:

Design of Tests

The user will input the starting location, destination location, time of departure, switch alert on/off. If on, choose alert method, and type in either phone number/email address. The output will construct a map with the traffic history shown on the map. The alerts, if any, will reach the user on expected time and date. The test cases will test the code's output with an expected result, and check to see if they match. The test cases should also be aware of empty or null data entries.

Test Case 1 (Directions): User will input:

Starting Location: 08854

Destination Location: 11351

Time of Departure: Depart By

Date: Today

Time: 05:00

Set Alert: On

Alert Method: test@testing.com

Test Case 2 (Directions): User will input:

Starting Location: 08901

Destination Location: 19019

Time of Departure: Leave Now

Set Alert: Off

Test Case 3 (Directions): User will input:

Starting Location: Verona, NJ

Destination: Montclair, NJ

Time of Departure: Depart By

Date: Tomorrow

Time: 10:00

Set Alert: On

Alert Method: 848-565-5938

Test Case 4 (Mobile): User will input:

Starting Location: Piscataway, NJ

Destination: NYC, NY

Time of Departure: Depart By

Date: 03/28/2014

Time: 13:00

Set Alert: On

Alert Method: 205-203-2323

Test Case 5 (Gas Prices): User will input:

Nearby Gas Prices: On

Fuel Type: Regular

Maximum distance from user's location: <5mi

The expected result of these tests would be that the direction is correctly shown on the screen. The alerts reach the email/phone number that the user typed in. Also, it should allow users to manage their existing alerts to delete the alerts that they do not want. In the case that the user enables the nearby gas prices, the nearby gas stations with their prices will be shown on the map and overlaid with user's route.

For the test coverage, first our test inputs are totally random, second, we are going to use boundary testing to test the boundary inputs. For example, the alert date should be within 10 days.

Our plan for integration testing is Bottom-Up Integration.

Project Management and Plan of Work

- ☐ Condense traffic data to major highways and routes
- ☐ Divide map into sectors: grid
- ☐ Condense weather to five main categories: clear, cloudy, rain, heavy rain, snow
- ☐ Get Register/Login working
- ☐ GetDirections() working
- ☐ Add traffic info
- ☐ Add weather info
- ☐ Get alert sending working
- ☐ Add alerts to database
- ☐ Take scheduled departure and factor in traffic for alert time
- ☐ List alerts
- ☐ Delete alerts
- ☐ Add gas usage information to database
- ☐ Calculate gas usage
- ☐ Get gas price retrieval working

Plan of Work

Original Plan of Work:

There are six people in our group and due to the limited time constraints and the demanding task of data collection and analysis, any fewer than the allotted six members would prove difficult. Even though the project workload will be divided evenly among the six members, specific groups of two will focus on certain aspects due to strengths and abilities as follows:

Jan & Jonathan: Departure Alerts & Public Transportation

Chih-Ting & Ting-Chieh: Probable Delays & Alternate Routes

Kevin & Xuan: Fuel Prices & Consumption

As of February 23, 2014:

In an attempt to make this project as agile as possible, we will maintain the set of groups working on their separate features. Each minigroup has been working on their separate use cases, test cases, and design. As of now, the system has been collecting data since the beginning of February.

The plan is to develop the website and maintain a weekly goal. The goal is to have a fully functional website by the March 23, the week prior to the first demo. This last week should be spent fine tuning small details and ironing out small kinks. This plan of action will most likely be altered as time goes and individuals might shift around or take extra work as needed.

As of March 19, 2014:

The traffic data stored in the MySQL database has been condensed to major roadways and routes. The coverage area of our program has been set and divided into 100 zones (10x10 grid). The graphical user interface has been implemented. Currently, the website displays a map (Google Maps API) and the input fields. Using the Google Directions API, we are able to display directions on the screen given starting and destination locations. The next step is to do error checking in order to ensure that the route fits within the program's coverage area. In addition, we will need to integrate the traffic data into `GetDirections()` and offer alternative routes when needed.

Traffic Monitoring	Start Date	End Date	Time	Team	2/24/2014 - 3/2/2014	3/3/2014 - 3/9/2014	3/10/2014 - 3/16/2014	3/17/2014 - 3/23/2014	3/24/2014 - 3/28/2014
					Implement..				
Main Scenario	24-Feb	7-May	72	Jan/Jon	Statistics on Data	Map Overlays / Alternate Routes	Total System Integration	Testing/Last Minute Changes...	First Demo
GasUsage	24-Feb	7-May	72	Kevin/Chih-Ting	Interface for Data Input	Usage Calculations & Statistics	Total System Integration	Testing/Last Minute Changes...	First Demo
Alerts	24-Feb	7-May	72	Ting-Chieh/Xuan	Interface for ManageAlerts	Send out Alerts	Total System Integration	Testing/Last Minute Changes...	First Demo

References

1. Marsic, I. 2012. *Software Engineering*.
2. *MapQuest Platform Web Services : Traffic (v2)*. [Online]. Available: <http://www.mapquestapi.com/traffic/>. [1 February 2014].
3. *API | Weather Underground*. [Online]. Available: <http://www.wunderground.com/weather/api/d/docs?d=index>. [1 February 2014].
4. *myGasFeed.com - Api requests for developers*. [Online]. Available: <http://www.mygasfeed.com/keys/api>. [1 February 2014].
5. *W3Schools Online Web Tutorials*. [Online]. Available: <http://www.w3schools.com/>. [1 February 2014].