Traffic Monitoring Service

Group No. 7

Final Report

Team Members

Name	**Email**
Kevin Hsieh	**hsieh63@eden.rutgers.edu**
John Reed	**johnreed@eden.rutgers.edu**
Geoff Oh	**geoffrey.oh@rutgers.edu**
Mike Simio	**michaelsimio@gmail.com**
Peter Lin	**Peterlin741@gmail.com**
Matt Araneta	**maraneta@eden.rutgers.edu**

Instructor: Prof. Ivan Marsic

Project URL: http://traffichistory.co.nf/

Revision History:

Version No.	**Date of Revision**
Part 1	05/03/2013
Part 2	05/12/2013

Breakdown of Contributions

All members contributed equally for this report.

Table of Contents

Customer Statement of Requirements

Many Services that give information about traffic only account for current traffic incidents. The majority of these services collects and releases data at certain intervals and then releases their views of the current traffic concentrations. These services, such as those given by "Traffic.com" and "Yahoo! Maps Live Traffic" use this method to monitor traffic. These types of services are the most widely accepted in terms of traffic aggregation and information.

This project attempts to move away from this method of only using current traffic data to give information about traffic concentrations. If traffic data can be continually taken for long periods of time, the traffic trends can be shown for particular routes. This can be taken further as to collect the weather along the route and the time of day for each piece of data. This new method of traffic monitoring is important to understanding the trends that occur in traffic. Current traffic monitoring services only release incidents and congestions as they occur, which are not as useful to someone looking at the current traffic before leaving for their destination. With this service, the user would be able to see the projected traffic for the roads they intend to travel, and they can find the likelihood of traffic congestions in those areas. To supplement this traffic monitoring system, live traffic updates can be included as well. This can help to account for outliers in the traffic probabilities that are created. One of the main focuses of the project will be giving the user enough information to make the correct route choices for their situation.

While previous iterations of this project have only included highways in New Jersey, we intend to extend the scope of the service to the entire Tri-State area. This area includes New Jersey, New York, and Pennsylvania. This change was made to cater to the expected users of this service. Many people who live in the Tri-State area have to travel to other areas or states for their jobs or schools. Therefore the primary users of this service are expected to be commuters. Temporary visitors, such as tourists would not necessarily find meaning in historical traffic data. However, commuters travel the same route many days a week, and they benefit the most from possible route improvement. The time spent using this service will easily be offset by the amount of time the commuter will end up saving with an improved route.

There are several major scenarios that we consider this traffic monitoring system to be useful. The most important scenario is for the user to be able to monitor traffic on major highways during rush hour. By allowing the user to view previous traffic data, the user can make plans for an alternate route, or to find a time where traffic is at a minimum. Another useful scenario is allowing the user to decide which highway to use under certain weather conditions, time of day, and day of the week for road trips or other planned drives. The user can plan ahead for longer drives and achieve minimal traffic through the use of observing traffic history. A final and more interesting use would be to use the traffic history data to observe if road infrastructure can be improved. Through the collection of traffic incidents, the number of damaged highways could be recorded. Similarly, if a certain area sees constant congestions, a recommendation could be made to either widen the highway or create an alternate route to move drivers away from the congested areas. The traffic monitoring service has many possible applications.

The desired traffic monitoring service will have multiple ways of viewing the traffic predictions. One of the methods should be to view the traffic projections by state. This should allow the user to view the traffic concentrations in their state, so that the user can avoid the roads that typically have traffic in their local areas. This allows the user to see in a broader scale the traffic concentrations that typically occur in the areas they regularly travel. This process is outlined below:

(1) "Traffic reports within a Zip Code" – the user is given the following choices:
- (A) "Select target Zip Code"
  - a. The User enters a Zip Code of his or her choice
- (B) "Time"
  - a. The user can choose from intervals of one hour
- (C) "Weather"
  - a. User selects which weather was occurring during the traffic they wish to see

Another method of viewing traffic reports will involve showing the traffic incidents along a route of the user's choice. The user will be able to enter a starting point and a destination, and the service will consult a directions service, such as Google Maps or Mapquest in order to find the fastest route. This route will not consider the traffic projections initially, and it will show the user the expected traffic along the given route. The service will ideally be able to suggest routes that avoid areas with a very high projected concentration of traffic. This will allow the user to find an ideal route with minimal exertion. This process is outlined below:

(2) "Traffic reports along a route" – The User will be displayed a low-traffic route
- (A) "Enter Route" – The user will have two text boxes in which to enter their starting location and destination. The service will display the path of the route on the GUI
- (B) "Time"
  - a. The user can choose from intervals of one hour

In this method, the Application will find a route with a low amount of traffic for the User. Commuters generally have to travel along several highways to get to their final destination, and this method will show the route they should take to avoid areas of high traffic. This can be enlightening information to the user, who might find out that they have been taking a highway that has a substitute with a much lower traffic concentration.

The above descriptions describe the user accessible front-end of the traffic monitoring system. The back-end of the system is comprised of the "Weather collection" and "Traffic collection" services. These services will collect data on weather and traffic conditions, will parse the information into a database, and will perform analyses that can be used to show the traffic projections on the front-end. These back-end services are only accessible by the administrator. The users will not have access to the collection service databases.

The "Weather Collection" provides the following function: it provides the weather along a certain highway, and it gives the time that the weather occurred. This service will update along a time interval of the administrator's choosing. The weather data will be retrieved through a weather forecasting service such as "Wunderground.com." This collection service will be able to retrieve data only specified highways and areas, and the data will be able to be used in conjunction with the "Traffic collection" service to allow the user to observe traffic patterns in different types of weather.

The "Traffic collection" service provides similar functions to the "Weather collection" service. It retrieves data from traffic incidents or congestions and records it in a database. This service will also update at a specified time interval of the administrator's choosing. This data can be taken from live traffic monitoring services, such as "511nj.org" or "Traffic.com." The service should update every time interval, making sure that it only records new traffic incidents, as multiple instances of traffic incidents would cause false positives to affect the traffic projection algorithms. The traffic and weather collection services can run in parallel in order to achieve a more accurate prediction of future traffic.

In addition to the version of the traffic monitoring service that the users can access on their computers, there should also be a mobile component to the service. The mobile component of the traffic report system should allow users to use the system on the go. The mobile application needs to offer the user the ability to get traffic reports on the go. This will expand the portability of the system and allow the user to get traffic updates no matter where they are. The app should also allow users to share their current traffic status. This will increase the accuracy of traffic predictions and allow future users to get a more accurate reading of their current traffic situation. The application should focus on simplicity and ease of use as the user will most likely already be travelling. The phone should automatically collect any the data the user does not need to explicitly provide.

The primary function of the mobile application should allow the user to get a traffic report on the go. The user should just need to enter their destination and allow the application to automatically collect the rest of the required information. This data should be sent to the web system and be processed as if a user of the web service had just entered those parameters. The report should then be returned to the user's smartphone and be presented in a clear manner.

Next, the user should be able to send their current traffic status to the database at any time. The accuracy of the database is limited to information pulled from online sources. The addition of real time updates would increase the accuracy of all future traffic conditions. If the user has found a road with high traffic intensity they should be able to warn future drivers. A simple radio button selection should allow the user to select their traffic intensity and have it be sent to the database with one button. Limiting the traffic intensity to a few selections will aid in quantifying the data for analysis in the database.

The mobile component of the system will expand the functionality of the entire system. The system will now be available to users on the go from their smartphones. Now if a user does not have access to a computer or has suddenly changed their route they will be able to access the system remotely. Furthermore, the addition of real time traffic updates will greatly increase the accuracy of the system. If a user has encountered heavy traffic, they can send a report to instantly update the database. Future users of the system, using either the mobile app or the website will have a more accurate traffic report.

Glossary of Terms

Administrator - Someone who oversees the website and is responsible for overseeing the data intake and manipulation

Application – The main program that the user will be interacting with. This program is made to be user friendly and is located on the website.

Current Traffic – Traffic that is currently stored on the *Traffic Service* websites at the time the user is utilizing the traffic monitoring service. This data is concrete, and describes the known conditions of traffic at the moment

Database - server or entity that will contain user data, traffic information, and weather information

Developer - Someone who is involved with creating the website's front-end and its back-end

Directions Web Service – An API that will be used to help calculate the route for the Directions portion of the Application. In this case, Mapquest API is used.

Dropdown box - Box with dropdown options that cannot be changed, only selected

Geocode – Receiving inputs of areas in the form of text and transforming it into Latitude and Longitude. Reverse Geocoding is the opposite.

Graphical User Interface (GUI) – A type of interface that allows the user to interact with the graphical components, including buttons, dropdown menus, and any maps that appear

Mobile application - Software for an Android smartphone that is available to all users of the traffic system. Mimics the basic functions of the web Application.

Mobile application user - Someone who will use the mobile app

Mobile device - A device that is meant primarily for mobile use, including tablets and smartphones

Mobile friendly site - site which is easier to navigate on a mobile device

Radio Button - A family of buttons from which the user can select their traffic intensity.

Traffic Incident Point – A point with a Latitude and Longitude that has a recorded traffic intensity associated with it.

Traffic Web Service – A website or web service that contains traffic data that can be collected, parsed, and stored into a database

Weather Web Service – A website or web service that contains weather data that can be collected, parsed, and stored into a database

User – A person who intends to use the traffic monitoring system to access historical traffic data

Summary of Changes

Project Objectives

The Application no longer has the option of showing live traffic updates. This was done to ease working with the Database. With live traffic updates, the time of each submission to the Database must be parsed and checked very often, a function which was not reasonable to implement at this time.

The Mobile Application cannot display use the directions function of the Application as of yet. This function is included in Future Work.

Use Case Descriptions

Instead of asking the User to input an Area to display the Traffic History, the Application now asks them for a Zip Code. This was done because of constraints on number of points for using the Google Maps API

The Application no longer asks for the day of the week as an input.

The Directions Use Case (UC-2) Uses the Google Maps API. Mapquest API was found to have a better API for the use of directions and avoiding points with high traffic intensities.

Use Case 2 now displays a route that avoids areas of high traffic intensity. There is only map as an output, and the list of directions for that route is also provided.

The Mobile Application initially planned to give the traffic report for a given route. The system was ultimately designed to give the traffic report for a given location.

System Design

The main Application no longer calls the Database in the first two Use Cases. The two classes it calls: MapService and DirectionsService now call the Database. This was done to reduce coupling between the Application and those classes.

The DirectionsService class now forms a route in the following way: It receives the inputs given by the user, Geocodes them, and finds a box bounded by those two points. The Mapquest API is then told to find the fastest route between those two points, while avoiding all points with a high traffic intensity within the bounded box. The Route is then found and displayed to the user along with a list of the directions.

We did not realize that location services must be explicitly available on the smartphone to collect the location. A network connection is not sufficient for the preconditions.

Functional Requirements Specification

Stakeholders

- User
- Mobile User
- Administrator

Actors and Goals

- User
    - Initiating Actor
    - The user has the ability to access the main Application to receive the traffic history reports and directions they desire
- Administrator
    - Initiating Actor
    - The administrator has the ability to run and provide upkeep for the traffic and weather collection services.
- Application
    - Participating Actor
    - The goal of the Application to provide the user with a clear and concise result of their traffic and directions queries in the form of text or images. The Application is run on a website and can be run on a variety of web browsers.
- Database
    - Participating Actor
    - The goal of the Database is to serve as a depository for all our data to be used. It will store the weather and traffic data sent by the Weather and Traffic Services. It will also store the parsed traffic data that is used to display the traffic history. It can be expanded to hold much more data.
- Mapping Service
    - Participating Actor
    - The goal of the Mapping Service is to plot traffic history data on a map that will be appealing and easy for the user to understand.
- Directions Service
    - Participating Actor
    - The goal of the Directions Service is to get a route from starting to end location. It will also have the goal to influence the route based on traffic history .
- Geocoding Service
    - Participating Actor
    - The goal of the Geocoding Service is to get the latitude and longitude of an address.

- Weather Service
  - Participating Actor
  - The goal of the Weather Service is to provide weather data concerning the roads used in the Application. This data will be stored in the Database.
- Traffic Service
  - Participating Actor
  - The goal of the Traffic Service is to provide traffic data concerning the roads used in the Application. This data will be stored in the Database.

Casual Description of Use Cases

- UC1: ViewTrafficHistory
  - The web interface will ask the user for the time of day, zipcode, and weather. After the user inputs the data, it will display in that zipcode which roads have a history of traffic and how severe it is.
- UC2: GetDirections
  - The web interface will ask the user for the time of day, area, and weather. The User will then input their starting location and destination. The interface will display on the map which route our systems suggests based on routing information for Mapquest and traffic history in the area of the route.
- UC3: MapInterface
  - Use the Mapping Service to display a clear result of the User's query. The Map Interface will be using the Google Maps API to provide much of its function. The map provided will ideally be a non-static map that the Users will be able to transform to more fit what they desire to see.
- UC4: GetTrafficData
  - Provide traffic data requested by the Traffic Service. The Traffic Service and its specifications are controlled by the Administrator.
- UC5: GetWeatherData
  - Provide weather data requested by the Weather Service. The Weather Service and its specifications are controlled by the Administrator.
- UC6: GetMobileReport
  - Provides the user with a traffic report in a given location via the mobile application on an Android smartphone.
- UC7: ReportTrafficCondition
  - Allows the user to report the traffic in their area via the mobile application on an Android smartphone. The mobile reports will be added to the database.

System Requirements

Functional Requirements:

Identifier	PW	Requirement
REQ1	5	The system's interface shall allow the user to select pre-populated options for traffic history based on time, area, and weather.
REQ2	4	The system's interface shall include text boxes in which the user can input their starting location and destination for directions.
REQ3	3	The system shall allow the user to view any part of the map and zoom in and out of different areas.
REQ4	5	The system shall pull data from various traffic sites (511.nj.org, google.maps.com, mapquest.com, and others) to be stored in the database.
REQ5	5	The system shall pull data from weather sites every 1-6 hours.
REQ6	5	The system shall have a mobile app with capabilities similar to that of the website version of the traffic monitoring service
REQ7	5	The system shall collect user data from the mobile app

Non-Functional Requirements:

Identifier	PW	Requirement
REQ8	2	The system shall provide a non-cluttered, user friendly, easy to understand web page.
REQ9	4	The system shall display Mapquest’s suggested route as well as the suggested route to take based on traffic history.
REQ10	2	The system shall provide a mobile friendly website which can be viewed on a mobile device.
REQ11	5	The system shall use an algorithm which determines the intensity of traffic.
REQ12	2	The system shall have preferences for route such as toll roads.
REQ13	4	The system shall allow the administrator to control the frequency and time of data gathering scripts.
REQ14	3	The mobile application shall provide a traffic history report and any current traffic in the user’s current area.

On-Screen Appearance Requirements:

Identifier	PW	Requirement
REQ15	5	The system shall use a display containing the Google Maps or Mapquest interface.
REQ16	4	The system shall have a concise and simple mobile app GUI similar to the website version of the traffic monitoring system

FURPS Table:

Functionality	•Features Web and Mobile based Applications for more ease of use for Users •Features inputs such as Zip Code, Weather, and Time of Day in order for the User to get a more accurate output of the traffic history they wish to see.
Usability	•There is a main page on the website Application that has a simple menu that links to all of the other pages on the site. •All of the pages on the site are created with a similar interfact, as not to confuse the User. •Input boxes are clearly labeled
Reliability	•Checks are done to see if Users entered valid data on the website Application as well as the Mobile Application. •Version Control of the Applications is done extensively to ensure that no progress is lost as a result of an accident.
Performance	•Multiple Users are able to use the either Application at the same time. •Efficiency checks are made when creating the maps in to prevent processing too much unimportant information
Supportability	•Website Application runs on many supported browsers •Mobile Application runs on all updated Android devices •Classes are separated clearly in order to increase ease of expanding the Application

Illustration of REQ15

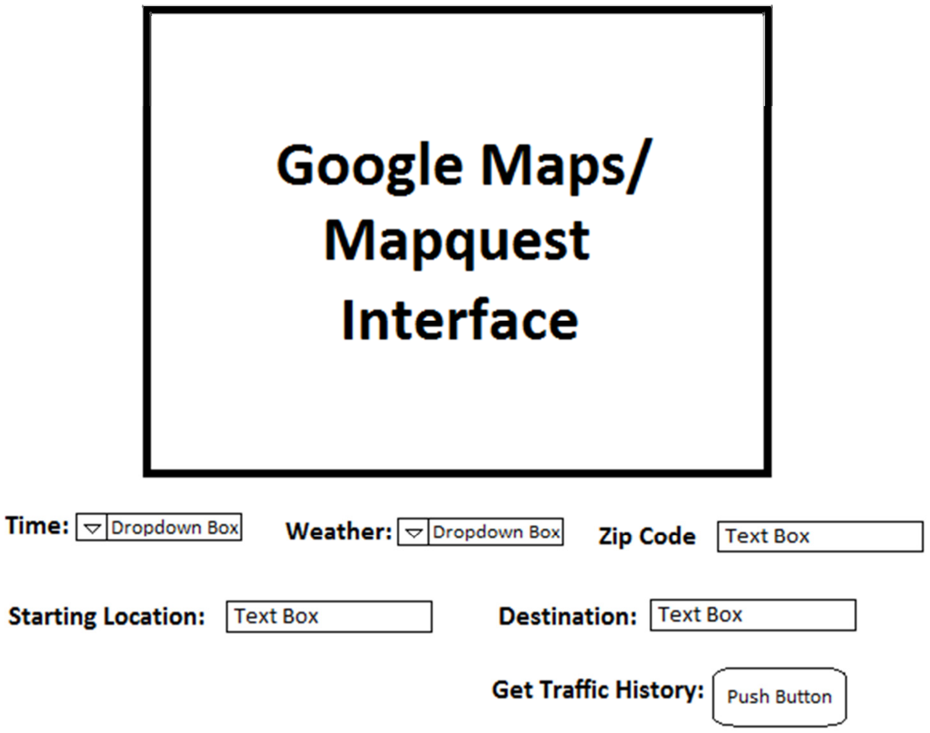
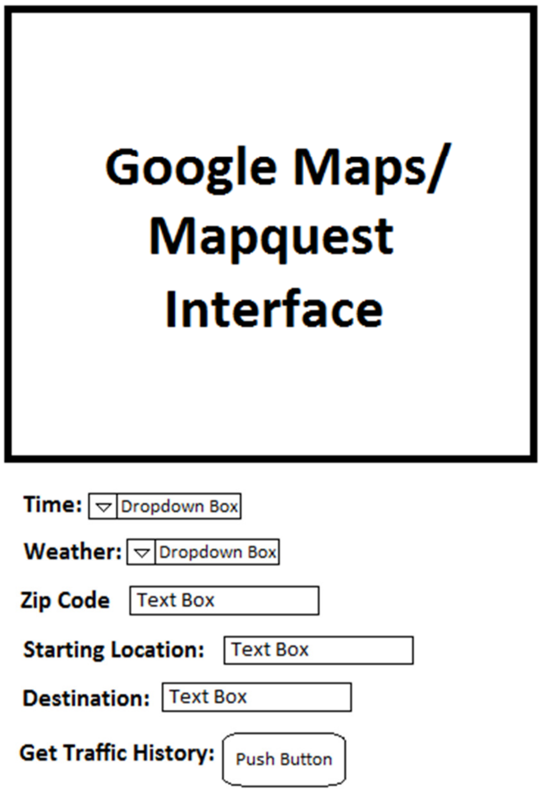


Illustration of REQ16



Traceability Table:

	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7
REQ 1	X						
REQ 2		X					
REQ 3	X	X	X				
REQ 4				X			
REQ 5					X		
REQ 6						X	X
REQ 7							X
REQ 8	X	X	X				
REQ 9		X				X	
REQ 10						X	
REQ 11				X			
REQ 12		X				X	
REQ 13				X	X		
REQ 14						X	
REQ 15	X	X	X				
REQ 16						X	X
Total PW:	15	20	10	14	9	21	14

Fully-dressed Descriptions of Use Cases

- Use Case 1: View Traffic History

Initiating Actor:	User
Goal:	To view traffic history of the desired Zip Code, weather conditions, and time.
Participating Actors:	Application, Database, MapService, Mapping Web Service
Preconditions:	Application is available Database is not empty All Services are available
Postconditions:	Traffic history is displayed for the user on the Application on the map.

Main Success Scenario:

1) The User uses drop-down boxes to selects the time of day and weather conditions. The User will input a valid Zip Code into the text box.
2) The User clicks the “Submit” button on the Application.
3) The Application will send query to the Database based on Zip Code, weather, and time of day.
4) The Database will return the information in an array.
5) The Application will parse the information and place them into a new array based on their severity. The Application will construct how the Mapping Service is called.
6) The Mapping Service is called and it returns the map image to be displayed.
7) The Application displays the map image for the user to view.

Extensions:

2. User enters invalid zipcode
   2.1. Application will detect error on submition
   2.2. Application stops submition and requests User to reenter a correct key
5. Application gets an empty array for reasons either no traffic history in the area, outside bounds of New Jersey, Pennsyvlnia, New York
   5.1. Application will handle the empty area by constructing call to Mapping Service to only show the zipcode
   5.2. Mapping Service is called and the map image of the zipcode is returned
   5.3. Application displays the map image and noting that the area has no traffic history.

Summary of Changes: The original options that had been planned to be given to the user proved unfeasible to implement. The APIs that are being used cannot show general areas using little processing power. This was changed to give the User the option to enter a Zip Code to show. The day of the week was also omitted in the final product. These options can be further looked into in future implementations.

- Use Case 2: Get Directions

Initiating Actor:	User
Goal:	To obtain driving directions based on traffic conditions along with an image of the route on a map.
Participating Actors:	Application, Database, Mapping Service, Geocoding Service, DirectionsService.
Preconditions:	Application is available Database is not empty All Services is available
Postconditions:	Directions and route image are displayed for the user on the Application.

Main Success Scenario:

1) The User uses drop-down box to select the time of day. The User uses text boxes on the website to input their starting location and their destination.
2) The User clicks the "Submit" button on the Application.
3) The Application takes the inputted addresses and calls the Geocoding Service.
4) Geocoding Service returns the longitude and latitude of the addresses
5) The Application uses the longitude and latitude to create a call to the Database to find all the traffic points between the starting and end location.
6) The Database returns all the traffic points and the severity.
7) The Application takes the traffic points and creates the call to the Directions Service
8) The Directions Service returns a JSON of the route which has been optimized with the traffic severity points.
9) The Application calls the Mapping Service with the route information
10) The Mapping Service returns an image of the map
11) The Applications displays the map of the route for the User along with the directions.

Extensions:

4. The User input a invalid or not found address.
    4.1. The Application will stop processing and display on the page the address was invalid or not found.
7. The Application received no traffic data from the database
    7.1. The Application continues processing. The route will not be influenced by traffic data and will be a simple quickest route.

Summary of Changes: When this Use Case was first created, there was no plan for implementation, therefore the Use Case was vague in its description. The final product ended up with the following plan: Geocode the two entered areas given by the user. Find the latitutde/longitude box bounded by the two given points. Enter into the Mapquest API to find the fastest route while avoiding all of the traffic points found in the bounded box found earlier. Output the route on a static map from the Mapquest API, and show the list of directions for the User.

- Use Case 4: Get Traffic Data

Initiating Actor:	Administrator
Goal:	To obtain traffic data
Participating Actors:	Database, Traffic Service
Preconditions:	Database is available Traffic site is available 1 hour has passed since last call.
Postcondtitions:	Traffic data stored into database Begin countdown to next call
Main Success Scenario:	1) Administrator executes the script that obtains the traffic data. 2) The script queries the traffic site and obtains data 3) The script parses the data to be used in the database.
Extensions:	Nothing to Mention.

- Use Case 5: Get Weather Data

Initiating Actor:	Administrator
Goal:	To obtain weather data
Participating Actors:	Database, Weather Service
Preconditions:	Database is available Weather site is available 1 hour has passed since last call.
Postcondtitions:	Weather data stored into database Begin countdown to next call
Main Success Scenario:	1) Administrator executes the script that obtains the traffic data. 2) The script queries the weather site and obtains data. 3) The script parses the data to be used in the database.
Extensions:	Nothing to mention.

- Use Case 6: Get a Traffic Report on a Mobile Device

Initiating Actor:	Mobile User
Goal:	To obtain a traffic report on a mobile device
Participating Actors:	Mobile Application, Database, Mapping Service
Preconditions:	Network communication available
Postcondtitions:	Nothing important to mention

Main Success Scenario:

1) User enters destination.
2) Application collects location and time from phone.
3) Application sends data to web based system.
4) Web based system returns report to application.
5) Application displays the results.

Extensions: The user enters an invalid destination. The user is notified and asked to try again.

- Use Case 7: Report a Traffic Condition from a Mobile Device

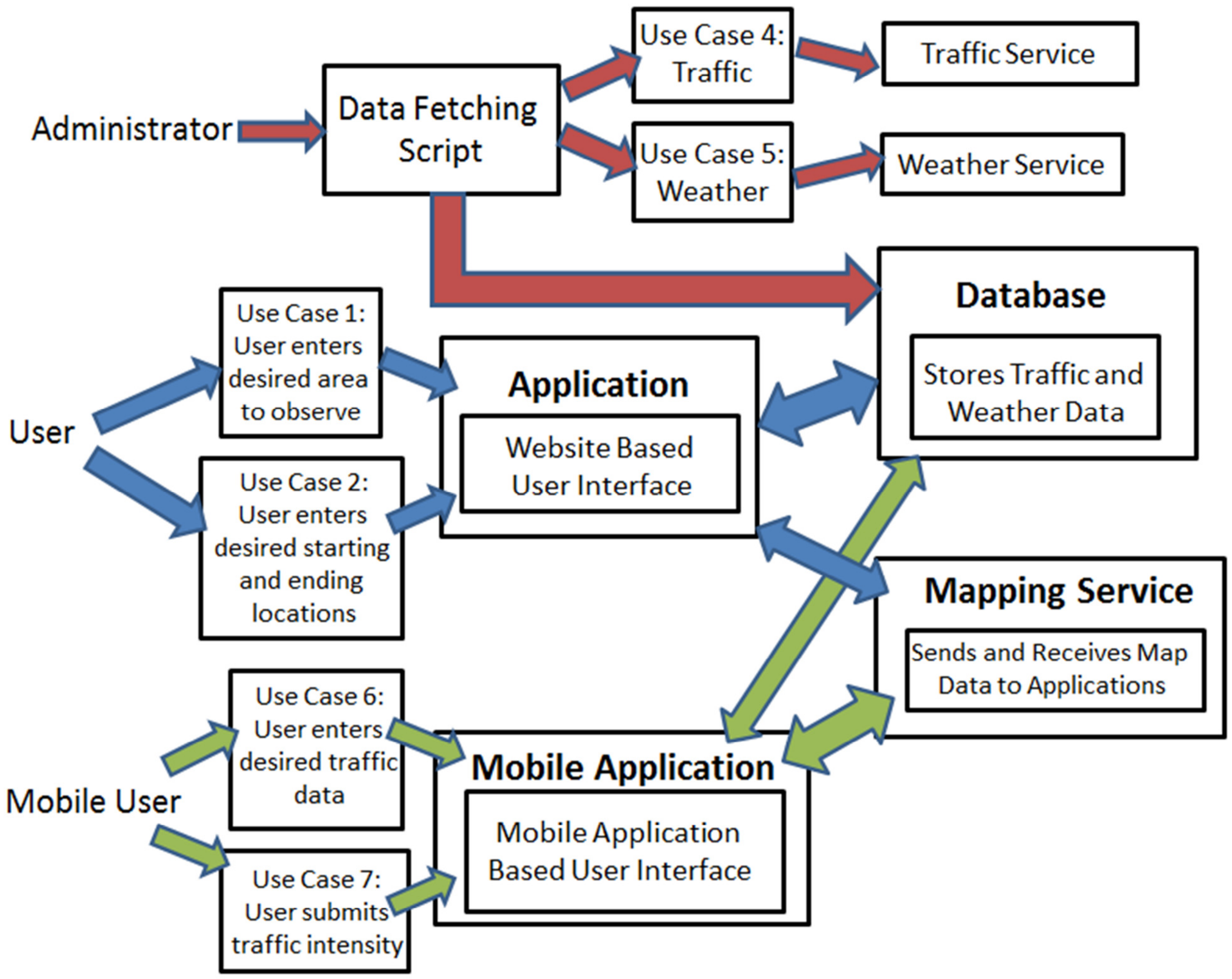
Initiating Actor:	Mobile User
Goal:	Allow User to share traffic condition
Participating Actors:	Mobile Application, Database, Mapping Service
Preconditions:	Network communication and GPS Location available
Postcondtitions:	Nothing important to mention

Main Success Scenario:

1) User selects traffic intensity from a variety of choices.
2) Report is sent to the database

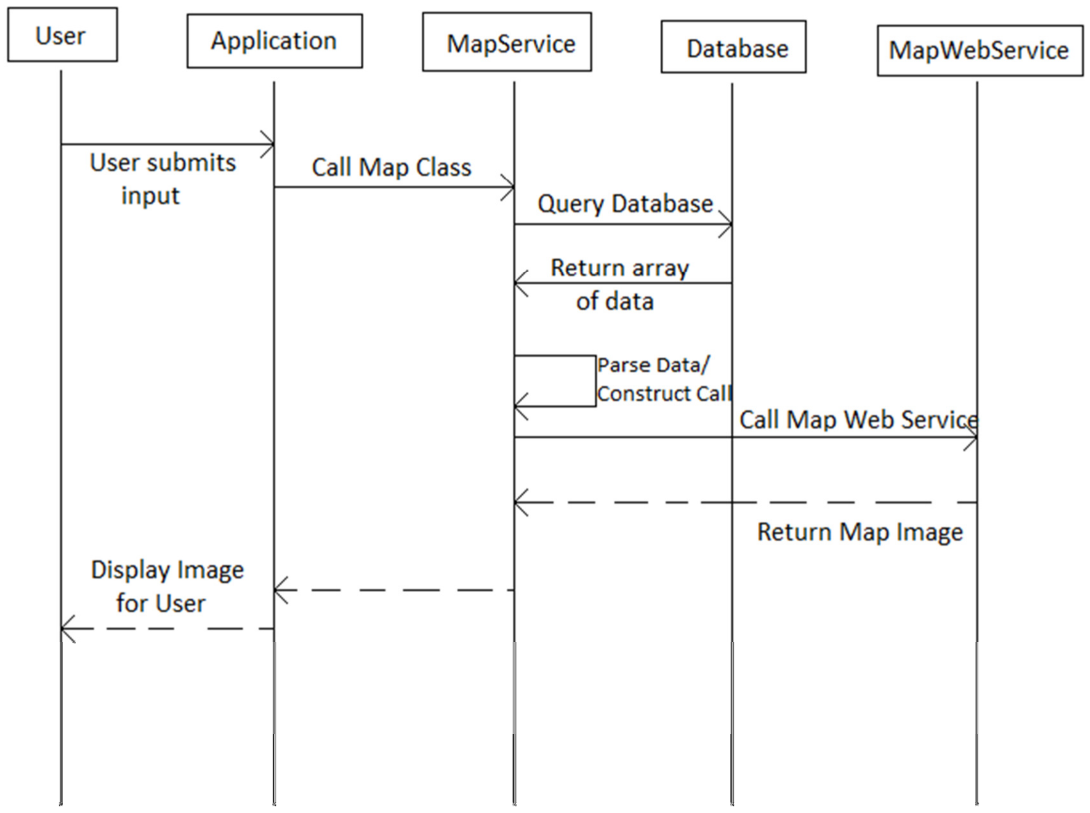
Extensions: The database fails to collect the data. The user is informed and instructed to try again.

Use Case Diagram


Administrator
Data Fetching Script
Use Case 4: Traffic
Traffic Service
Use Case 5: Weather
Weather Service
Database
Stores Traffic and Weather Data
User
Use Case 1: User enters desired area to observe
Use Case 2: User enters desired starting and ending locations
Application
Website Based User Interface
Mapping Service
Sends and Receives Map Data to Applications
Mobile User
Use Case 6: User enters desired traffic data
Use Case 7: User submits traffic intensity
Mobile Application
Mobile Application Based User Interface

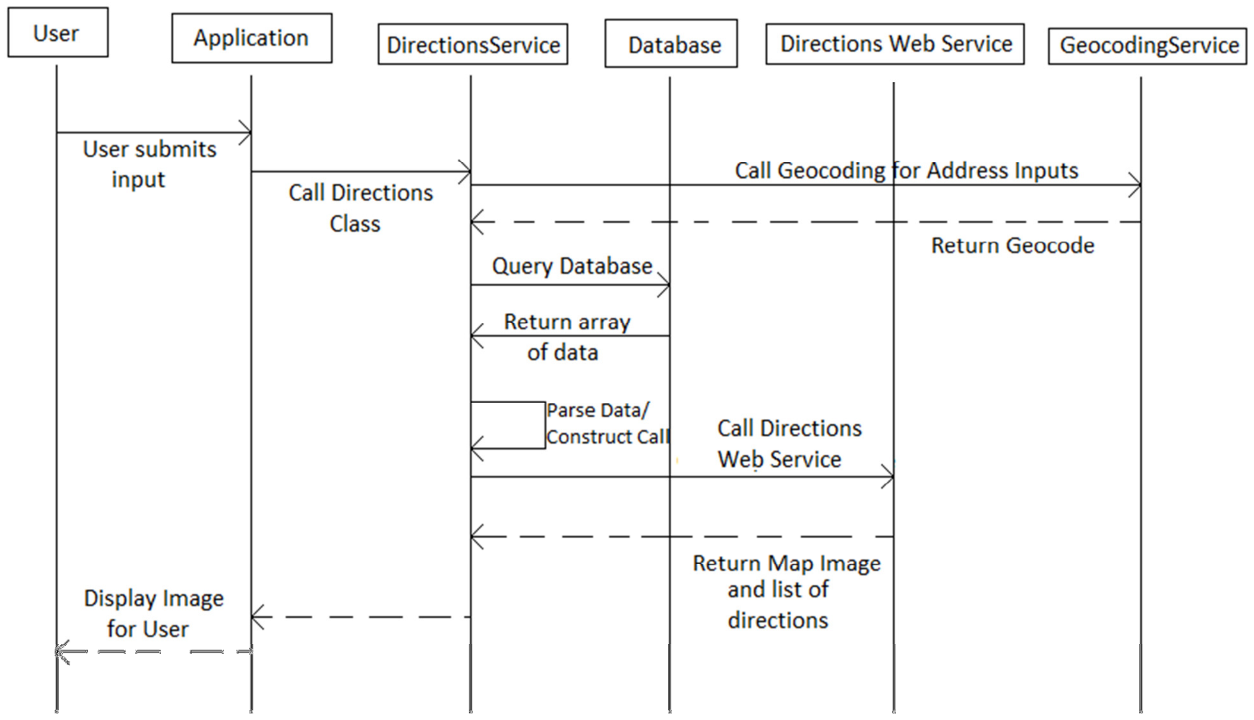
System Sequence Diagrams

Use Case 1:



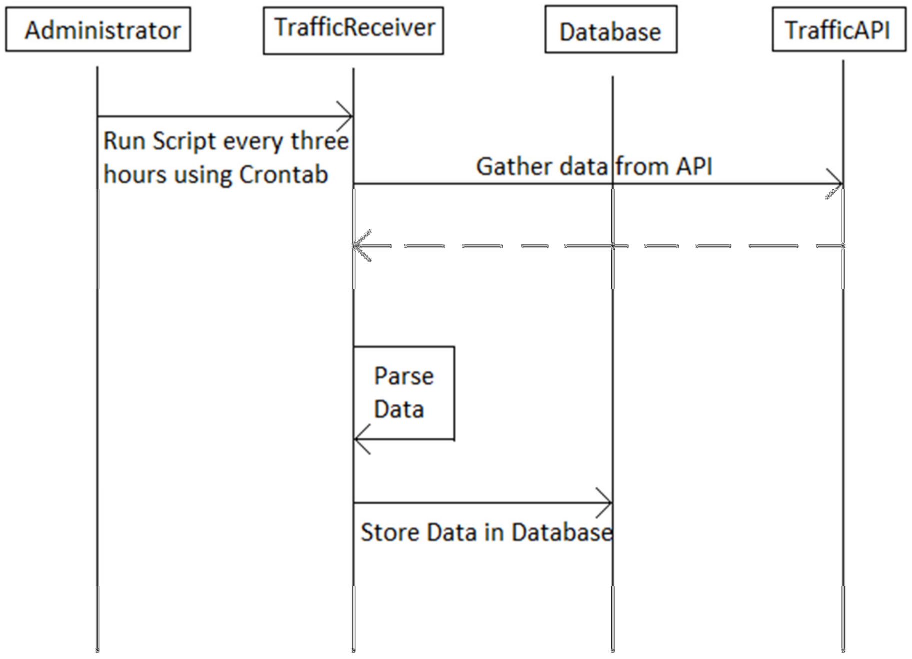
This system diagram shows the normal flow of use case 1 and the alternate flow for if the array is empty. The User will enter the data and submit it on the Application. The Application queries the database for traffic within the inputted zip code. The Application then calls the MapService class to process these inputs, and calls the Database requesting the traffic points in the area desired. The Database returns an array of data. The MapService class parses the information and constructs the link to the Mapping Web Service. The Mapping Web Service is called and returns a map image to the Application. The Application will display the image for the user. For the alternate flow if the array is empty, the map will show the zip code still, but a text box will appear at top the show that the zip code has no traffic history.

Use Case 2:



This system diagram shows the normal flow of use case 2 and the alternate flow for if the array is empty. The User will enter the data and submit it on the Application. The Application calls the DirectionsService class with the given inputs. The DirectionsService class then calls the Geocoding Service and validates the returned geocode. The DirectionsService uses the geocode to query the database for traffic history between the geocoded points. The Database returns an array of results. The DirectionsService uses the result to construct a call to Directions Web Service. The Directions Web Service returns JSON data of the route to take. The DirectionsService class parses the JSON data and constructs an image URL to display. The Application displays the image of the route along with the directions.

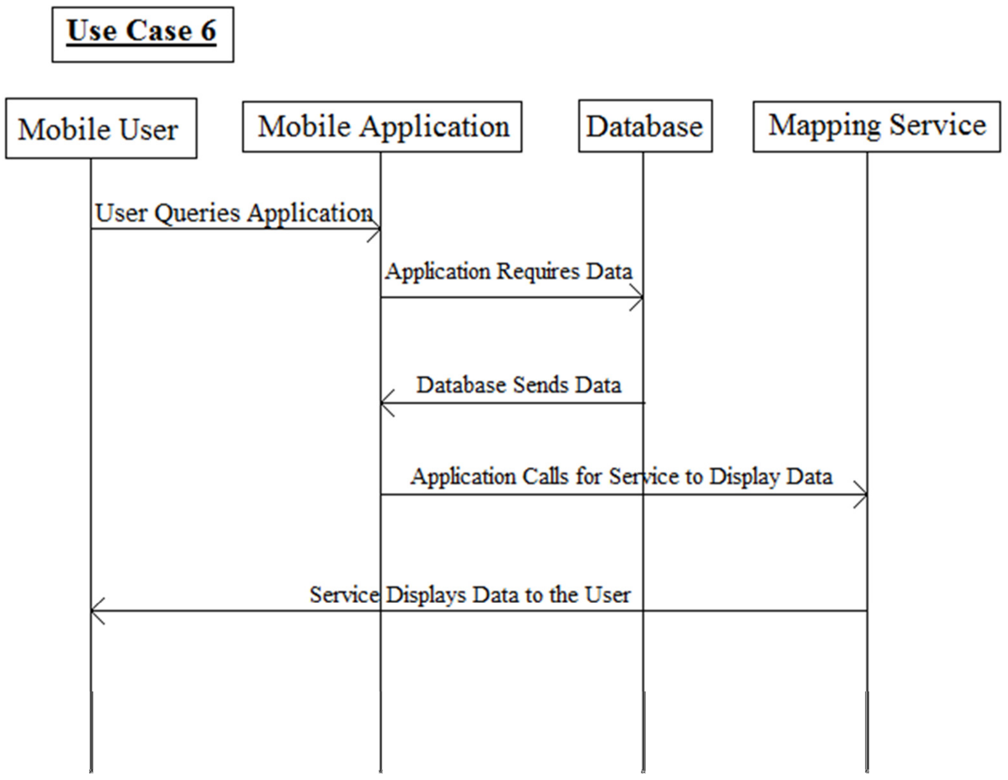
Use Case 4, 5 – System Sequence Diagram:



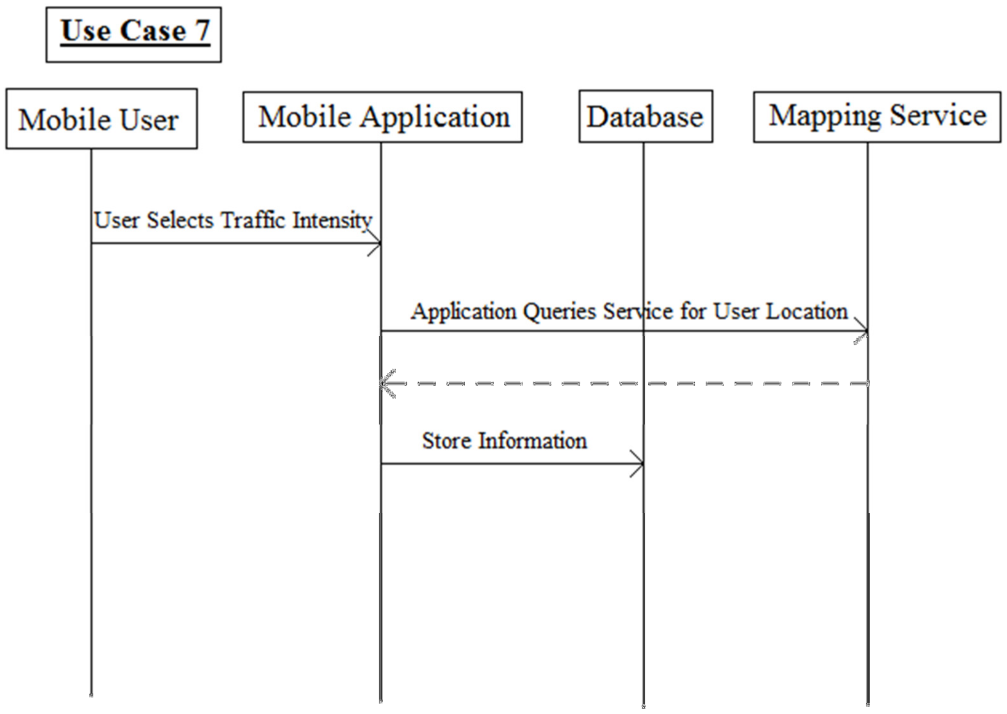
Use cases four and five are very similar and both have the same system sequence diagram. Above is the diagram for use case 4; simply replace TrafficReceiver and Traffic API with WeatherReceiver and Weather API to obtain the diagram for use case 5.

The ‘TrafficReceiver’ comes in the form of a Perl script which collects data. The Administrator runs this script every three hours using the Crontab capabilities of our website (Crontab can set a command or script to run at specific times every day, week, etc.). When the script is run, it gathers information from a traffic/weather API which we have access to. This information may come in the form of JavaScript Objects (JSON) or XML files. The script parses through this data and obtains the data we want for our database. For Use Case 4 (traffic), this may include zip code, time, latitude, longitude, severity, description, address, county, and state. For Use Case 5 (weather), this data includes time, weather, and zipcode. Finally, this data is sent to and stored in the MySQL database on our website server. This is done by the Perl script (the ‘TrafficReceiver’) using a database connection and SQL commands.

Previously, we had the ‘Administrator’ query the traffic/weather service, parse the data, and send the information to the database. This is inaccurate; the administrator is not actually doing these things – the Perl scripts are. Therefore, it was best to include TrafficReceiver and WeatherReceiver, which compose of the Perl scripts which accomplish these tasks. The Administrator is in charge of running these data collection scripts every few hours.



In Use Case 6, the User enters information to the Mobile Application to receive data on the traffic and weather conditions for the roads they desire. The Mobile Application sends which data is required to the Database, and the Database returns the needed data. The Mobile Application calls for the Mapping Service to display the necessary data to the User, and a map containing the desired information is then shown to the User.



In Use Case 7, The User desires to enter the traffic intensity of the area they are currently in. The User chooses an option for traffic intensity and sends it to the Mobile Application. The Mobile Application receives the data, and queries the Mapping Service to find where the User is located. This data is sent back to the Mobile Application, and the Mobile Application stores the User's location and entered traffic intensity into the database.

Effort Estimations

Actor type	Descript of how to recognize actor type	Weight
Simple	The actor is a system our system interacts with	1
Average	The actor is a person interacting with the system through text-based interface	2
Complex	The actor is a person interacting with our system through a graphical user interface	3

Actor name	Description of characteristics	Complexity	Weight
User	User interacts through a graphical user interface	Complex	3
Mobile User	Mobile user interacts through a graphical user interface	Complex	3
Administrator	Administrator interacts with the system through command line or text based interface	Average	2
Application	The Application is a system interacting with our system.	Simple	1
Database	The Database is a system interacting with our system.	Simple	1
Mapping Service	The Mapping Service is a system that our system interacts with.	Simple	1
Weather Service	The Weather Service is a system that our system interacts with..	Simple	1
Traffic Service	The Traffic Service is a system that our system interacts with.	Simple	1

UAW = 2 * Complex + 1 * Average + 5 * Simple = 13

Use case category	Descript of how to recognize actor type	Weight
Simple	Simple interface design. Up to one participating actor. Number of steps for success scenario: <4	5
Average	Moderate interface design. Up to two participating actor. Number of steps for success scenario: <6	10
Complex	Complex interface design. Up to four participating actor. Number of steps for success scenario: <12	15

Use case	Description of characteristics	Category	Weight
View traffic history (UC-1)	Moderate interface design. Three particapating actors. 7 steps for main success scenario.	Complex	15
Get directions (UC-2)	Moderate interface design. Four particapating actors. 11 steps for main success scenario.	Complex	15
Get traffic data (UC-4)	Simple interface design. Two participating actors. 3 steps for main success scenario.	Simple	5
Get weather data (UC-5)	Simple interface design. Two participating actors. 3 steps for main success scenario.	Simple	5

Get traffic report on mobile device (UC-6)	Moderate interface design. Three particapating actors. 5 steps for main success scenario.	Average	10
Report traffic condition from mobile device (UC-7)	Moderate interface design. Three particapating actors. 5 steps for main success scenario.	Average	10

UUCW = 2 * Complex + 2 * Average + 2 * Simple = 60

UUCP = UAW + UUCW = 73

TCF Parameters

Technical factor	Description	Weight
T1	Distributed system	2
T2	Performance objectives	1
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install	.5
T7	Easy to use	.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent use	1
T11	Special security features	1
T12	Provides direct access for third parties	1
T13	Special user training facilities are required	1

Technical factor	Description	Weight	Perceived complexity	Calculated factor
T1	Distributed Web based system	2	2	4
T2	User expect fast loading web pages. Scripts should be run with good performance	1	4	4
T3	End-user efficiency has no exceptional demands	1	3	3
T4	Internal processing is complex and deals with large data	1	4	4
T5	Reusability of web pages and database tables	1	3	3
T6	Install is not required	.5	1	.5
T7	Use is very simple	.5	4	2
T8	Portable, but not a concern	2	2	4
T9	Changes can be made easily by developers1	1	3	3
T10	Concurrent use is available.	1	2	2
T11	Security concern is only with user passwords and sql injections	1	3	3

T12	No direct access for third parties	1	0	0
T13	No unique training needs	1	0	0

Technical Factor Total : 32.5

TCF = C1 + C2 * TFT = .6 + (.01 * 32.5) = .925

ECF Parameters

Environmental factor	Description	Weight
E1	Familiar with the development process	1.5
E2	Application problem experience	.5
E3	Paradigm experience	1
E4	Lead analyst capability	.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time staff	-1
E8	Difficult programming language	-1

Environmental factor	Description	Weight	Perceived complexity	Calculated factor
E1	Beginner familiarity with UML-based development	1.5	2	3
E2	Some application problem experience	.5	3	1.5
E3	Knowledge object oriented approach	1	4	4
E4	Moderate lead analyst	.5	3	1.5
E5	Motivated, but team members slacking	1	3	3
E6	Stable requirements expected	2	2	4
E7	No part-time staff	-1	4	-4
E8	Programming language is of average difficulty	-1	3	-3

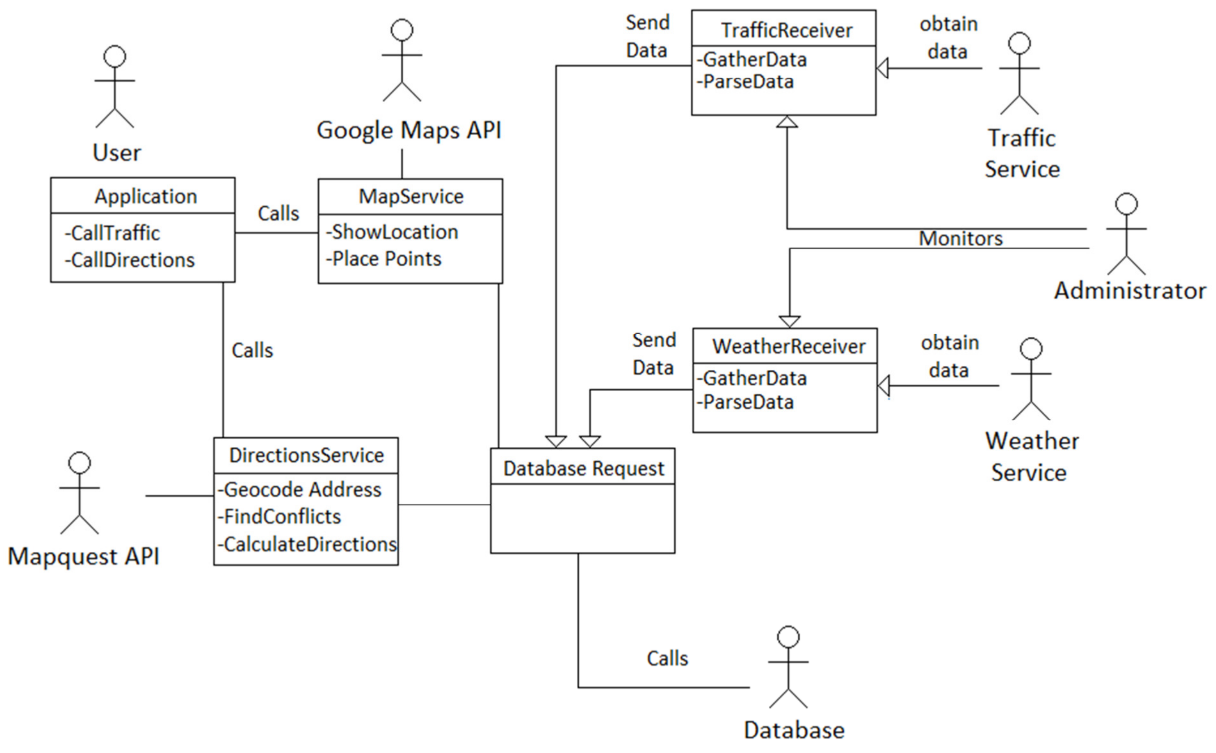
Environmental Factor Total : 10

ECF = C1 + C2 * EFT = 1.4 + (-.03 * 10) = 1.1

UCP = UUCP × TCF × ECF
UCP = 73 * .925 * 1.1 = 74.2775

Domain Analysis

Domain Analysis for Website Users

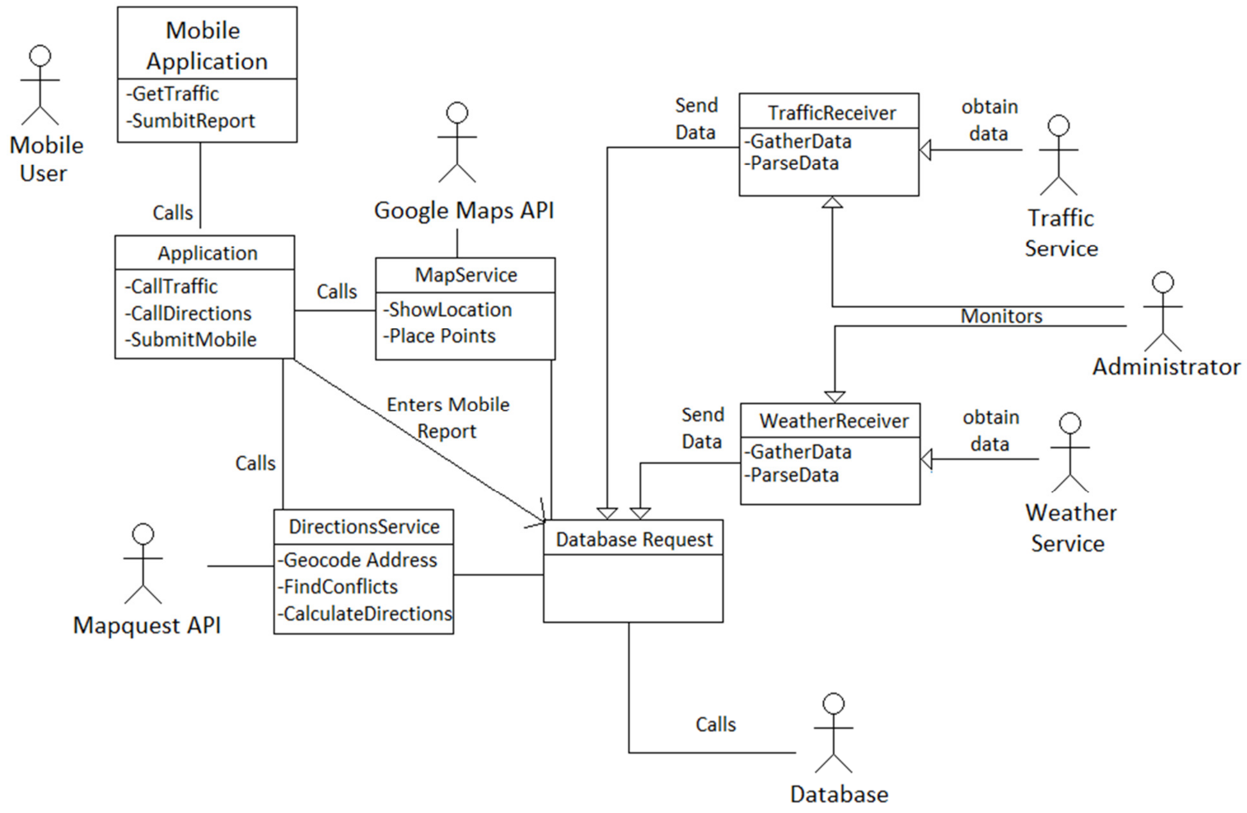


The Domain Analysis shows that the User only has access to the Application. This boundary is made simple to create the easiest user experience. The Application can call either the Map or Directions Service. Although they seem similar in nature, the relevant processes needed to complete each action require different APIs to be used for each function. The Google Maps API is much better at placing unique points onto a map, while the Mapquest API is better at finding routes while avoiding locations that have high traffic severities. Both of these functions call the Database to receive the relevant information about the traffic points they wish to analyze or display.

The Administrator has control of the two scripts that can receive data from Web Services. The traffic and weather scripts gather and parse the data received from the Web Services, and they store that data in the database to be used by the main Application.

Changes: In the original Domain Analysis diagram, the application called the MapService and the DirectionService along with making the Database Request. This functionality was changed, for if the Application were to make the database calls, it would require much more data being passed between the classes. This changed reduced the coupling between the Application and its two functions by a significant amount. Reducing the number of functions in Application also made it a simpler boundary for the User.

Domain Analysis for Mobile Users



The Mobile Application is made to mimic the functionalities of the website Application. Because of this, the best way to replicate the functions of the website would be to use the code of the website itself. For Use Case 6, The Mobile Application gets a series of inputs from the Mobile User, and the Mobile Application sends that data to the main Application. The Application then parses the data from the Mobile Application. At this point, the Application behaves in the exact same way as the previous Domain Diagram.

For Use Case 7, The Mobile Application receives a series of inputs for a mobile traffic report. These inputs are again sent to the main Application. This time, the Application parses the data received, and it enters it straight into the database, bypassing the MapService and DirectionsService.

Changes to this Domain include: the changes included in the previous diagram concerning the main Application, and the fact that the Mobile Application works entirely through the main Application to receive its functionality.

Use Case 1: View Traffic History

Concept Definitions

Responsibility Description	Type	Concept Name
Takes in user input to find which areas the user is requesting traffic history. Outputs traffic information once the data is processed	D	Application
Uses the Google Maps API to display the areas the user is requesting.	D	MapService
Contains the traffic and weather data relevant to the user. This data can be accessed by the application	K	Database

Association Definitions

Concept Pair Name	Associated Definition	Association Name
Application ↔ Map Service	Application sends user locations to the MapService. The MapService returns the map of the area to the user.	Provides Data
MapService ↔ Database	The MapService requests Database information to be used in its functions in order to display the correct information on the map	Provides Data

Attribute Definitions

Concept	Attributes	Attribute Description
Application	Weather	Weather of the area the user requires
	Time	Time of day user requires
	Day of Week	Day of the week user requires
	Area	Location user desires
Database	Traffic Data	Traffic intensities associated with times and locations
	Weather Data	Weather history linked to stored locations

Use Case 2: Get Directions

Concept Definitions

Responsibility Description	Type	Concept Name
Takes in user input to find which areas the user is requesting traffic history. Outputs direction information once the data is processed	D	Application
Contains the traffic and weather data relevant to the user. This data can be accessed by the application	K	Database
Receive starting location and destination from the user. Avoid points of high traffic intensity. Find directions and display map showing route.	D	DirectionsService

Association Definitions

Concept Pair Name	Associated Definition	Association Name
Application ↔ DirectionsService	Application sends user starting location and destination. DirectionsService returns list of directions and map showing the route.	Provides Data
DirectionsService ↔ Database	The DirectionsService accesses the data in the Database to accurately form the route, and to avoid locations of high severity.	Provides Data

Attribute Definitions

Concept	Attributes	Attribute Description
Application	Weather	Weather of the area the user requires
	Time	Time of day user requires
	ZipCode	Location user desires
	StartingLocation	Starting location of the user
	Destination	Requested Destination of the user
Database	TrafficData	Table of traffic intensities associated with times and locations
	WeatherData	Table of weather history linked to stored locations
DirectionsService	ListOfDirections	List of directions user can take to get to their destination
	AvoidedPoints	List of points to avoid when creating the route

Use Case 4: Get Traffic Data

Concept Definitions

Responsibility Description	Type	Concept Name
Retrieves traffic history from traffic service websites.	D	TrafficReceiver
Contains traffic history previously sent in by TrafficReceiver	K	Database

Association Definitions

Concept Pair Name	Associated Definition	Association Name
TrafficReceiver → Database	TrafficReceiver obtains information from traffic service websites and stores data into the database.	Provides Data

Attribute Definitions

Concept	Attributes	Attribute Description
TrafficReceiver	Location	Location of traffic that occurred
	Time	Time of traffic that occurred
	TrafficType	Type of traffic that occurred
Database	TrafficData	Table of traffic intensities associated with times and locations

Use Case 5: Get Weather Data

Concept Definitions

Responsibility Description	Type	Concept Name
Retrieves weather history from weather service websites.	D	WeatherReceiver
Contains weather history previously sent in by WeatherReceiver	K	Database

Association Definitions

Concept Pair Name	Associated Definition	Association Name
WeatherReceiver → Database	WeatherReceiver stores information of weather associated with stored locations into database.	Provides Data

Attribute Definitions

Concept	Attributes	Attribute Description
WeatherReceiver	Location	Location of weather that occurred
	Time	Time of weather that occurred
	WeatherType	Type of weather that occurred
Database	TrafficData	Table of weather history linked to stored locations

Use Case 6: Get Mobile Traffic Data

Concept Definitions

Responsibility Description	Type	Concept Name
Takes in user input to find which areas the user is requesting traffic history. Outputs traffic information once the data is processed.	D	MobileApplication
Receives the inputs from Mobile Application and sends them to the necessary locations to be processed.	D	Application
Uses the Google Maps API to display the areas the user is requesting.	D	MapService
Contains the traffic and weather data relevant to the user. This data can be accessed by the application	K	Database

Association Definitions

Concept Pair Name	Associated Definition	Association Name
MobileApplication ↔ Application	Mobile Application sends its inputs to the main Application to receive its functionality	
Application ↔ MapService	Application sends user locations to the MapService. The MapService returns the map of the area to the user.	Provides Data
MapService ↔ Database	The MapService reads the weather and traffic data relevant to the user.	Provides Data

Attribute Definitions

Concept	Attributes	Attribute Description
MobileApplication	Weather	Weather of the area the user requires
	Time	Time of day user requires
	Zip Code	Zip Code User desires
Database	TrafficData	Traffic intensities associated with times and locations
	WeatherData	Weather history linked to stored locations

Changes: The DirectionsService was removed from this Domain Analysis because of time constraints. The Mobile Application sends inputs through the main Application to get the traffic history of the area the User wishes to see.

Use Case 7: Submit Mobile Traffic Report

Concept Definitions

Responsibility Description	Type	Concept Name
Allows user to submit the traffic intensity of the area they are currently in.	D	MobileApplication
Receives the inputs from the Mobile Application and sends those to the Database.	D	Application
Stores the location and traffic intensity submitted by the user.	K	Database

Association Definitions

Concept Pair Name	Associated Definition	Association Name
MobileApplication ↔ Application	MobileApplication receives input from the User and sends that data to the Application	Provides Data
Applicaiton ↔ Database	The Application stores the received data into the database.	Provides Data

Attribute Definitions

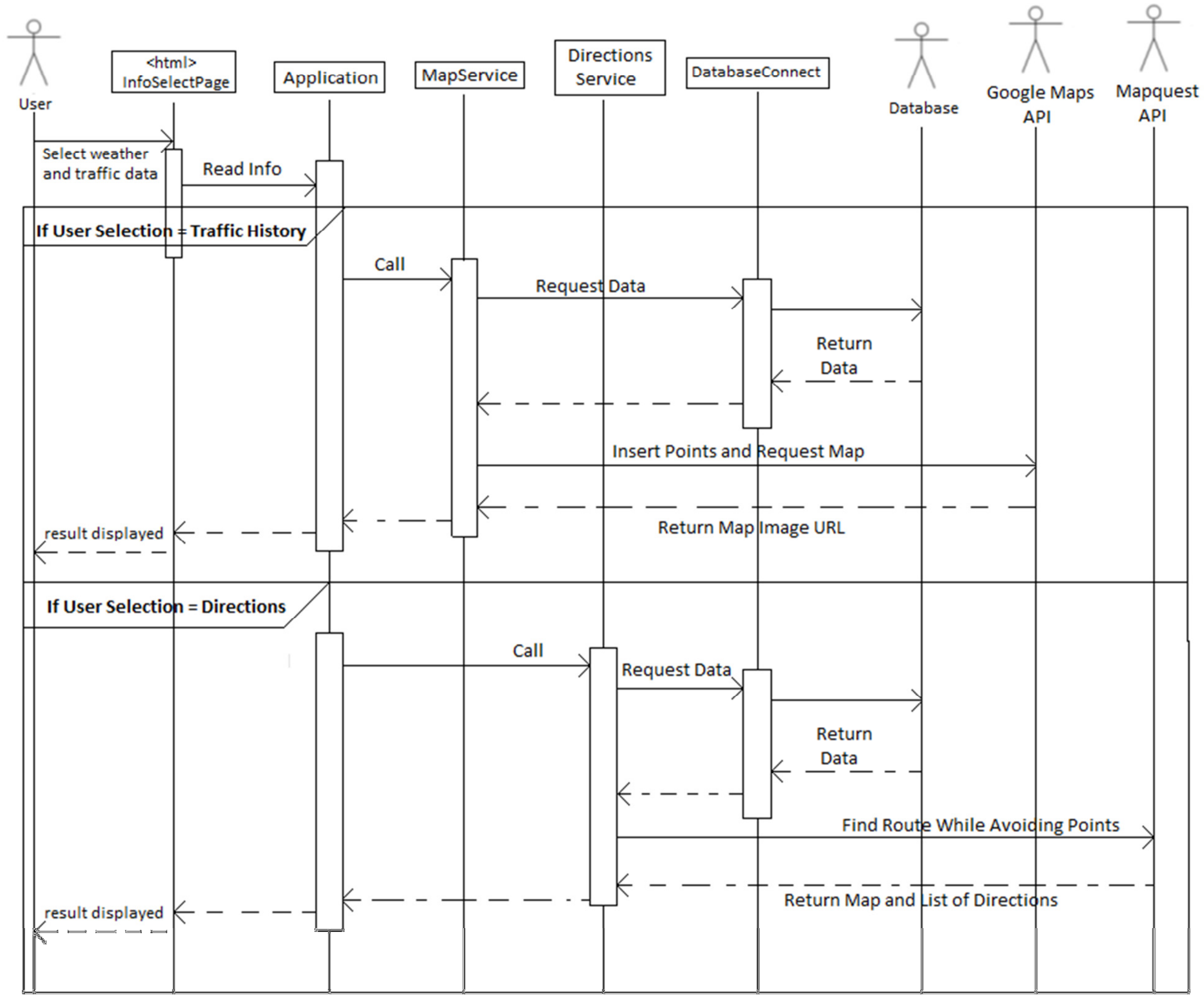
Concept	Attributes	Attribute Description
MobileApplication	Time	Time of day user submits
	TrafficIntensity	Level of traffic user observes
	Location	Location of the user
Database	TrafficData	Traffic intensities associated with times and locations

Domain Traceability Matrix

	Application	MapService	DirectionsService	Database Request	WeatherService	TrafficService	Mobile Application
UC-1	Receives inputs from User	Uses inputs to process Map Data		Accessed by MapService			
UC-2	Receives inputs from User		Uses inputs to process Directions	Accessed by Directions Service			
UC-3	Map is shown on Application						
UC-4				Stores Traffic Data		Processes information from Traffic Web Service	
UC-5				Stores Weather Data	Processes information from Weather Web Service		
UC-6	Receives data from Mobile App, sends data to MapService	Uses inputs to process Map Data		Accessed by MapService			Receives inputs from Mobile User
UC-7	Receives data from Mobile App, sends data to Database			Stores Mobile Traffic Report			Receives inputs from Mobile User

Interaction Diagrams

Use Cases 1 and 2:



These Use Cases show the options the User has when accessing the Application. The User is prompted to enter information regarding the traffic history or directions they require. Based on these inputs, the Application will move to one of two states: Traffic History or Directions.

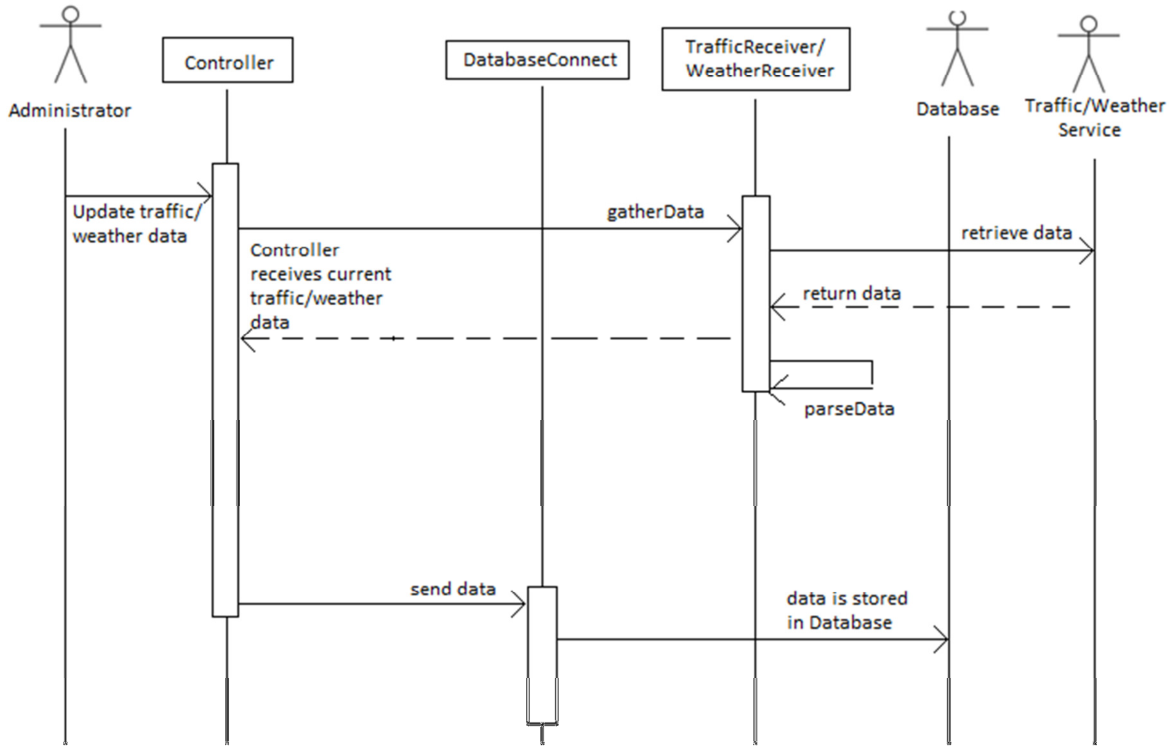
In the Traffic History State, the MapService calls the Database using DatabaseConnect, an example of the High Cohesion Principle. DatabaseConnect is used as an intermediate between the controller and the database. This ensures that the controller does not do too many computations when attempting to access data from the Database. The Database then returns the information through DatabaseConnect back to the MapService, where the MapService will use that data to call the Google Maps API and place points on a map that is to be returned to the User.

The Directions State is very similar to the Traffic History State and employs many of the same methodologies. It calls and receives data from the Database, however the DirectionsService calls Mapquest API. DirectionsService uses Mapquest API to find a route while avoiding all locations that have a high traffic severity along that route. The Mapquest API returns both a map and list of directions that DirectionsService can pass back to the User.

Changes: that Occurred in these Use Cases are as follows: The main Application no longer calls the Database. The Database is called by the main classes of the two states: MapService and DirectionsService. This was done to decrease coupling and increase processing time, further explained in the Domain Analysis. Another change is that DirectionsService uses Mapquest API rather than the Google Maps API. This was necessary because Google Maps does not contain the features that would allow to avoid certain points with high traffic severities.

Finally, The system was changed to behave using the State Design Pattern. The Application begins in its base state, waiting for input. When it Receives input, it checks if the User desires Traffic Information and Directions, and it moves to the appropriate state. After the State has completed its processes, it moves back into the base state of Applications, waiting for data yet again.

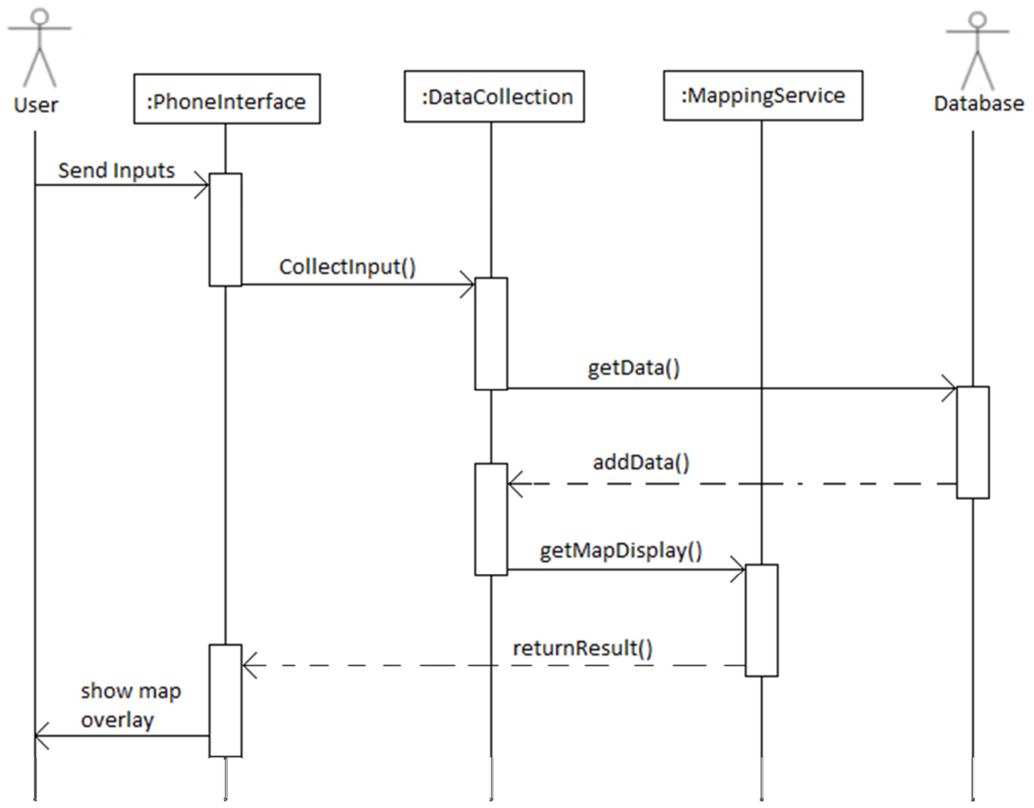
Use Case 4 and 5:



These use cases have the responsibility of accessing the Traffic and Weather Service websites to retrieve data at constant intervals. At every time interval, the Controller is told to update the traffic and weather data. Through the High Cohesion principle, the Controller sends for the TrafficReceiver or WeatherReceiver to access the data from the web services. This data is then sent back to the Controller via the TrafficReceiver and WeatherReceiver as JSON objects. The TrafficReceiver and WeatherReceiver then parses the data into a format that is easier to use. The Controller then uses this parsed data and sends it to the DatabaseConnect. The Low Coupling principle is used, to ensure that the Database has the least number of connections possible. The only pieces of the system that should interact with the Database should be those specifically made to do so. With the database storage, this use case is complete.

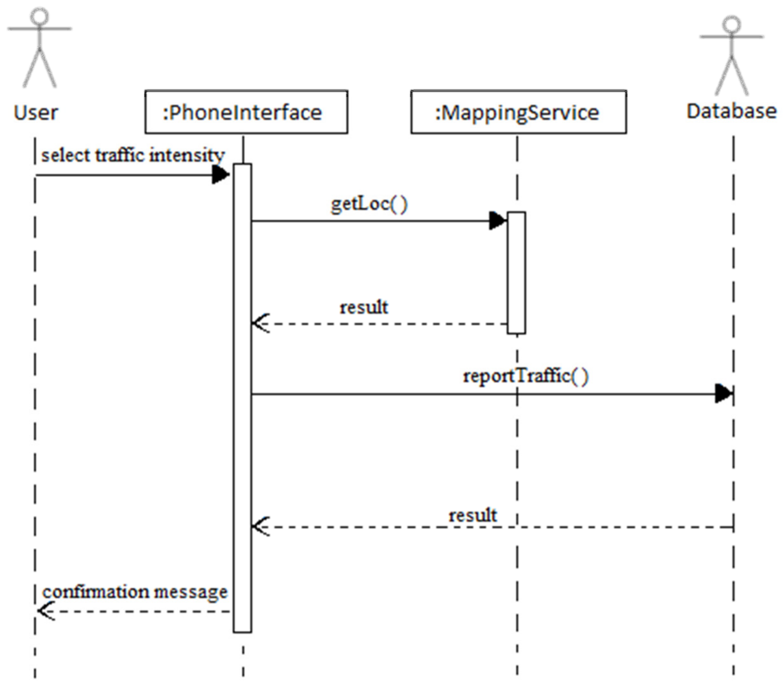
Changes: The Parser was removed because that functionality is already included in the scripts for the traffic receiver and weather receivers.

<u>Use Case 6:</u>



This use case has the responsibility of getting traffic history data shown to a mobile user. Similar to Use Case 1, it employs the Expert Doer principle. The user enters his or her data to the Mobile Application and sends it to the controller of this system, DataCollection. The DataCollection uses the data sent by the user to retrieve the Database information relevant to his or her request. The data is sent back to DataCollection, and this data is then sent to the MappingService. The MappingService uses the data sent by DataCollection to show a map overlay that shows the relevant traffic history requested by the user. This overlay is sent to the Mobile Application to be seen by the user.

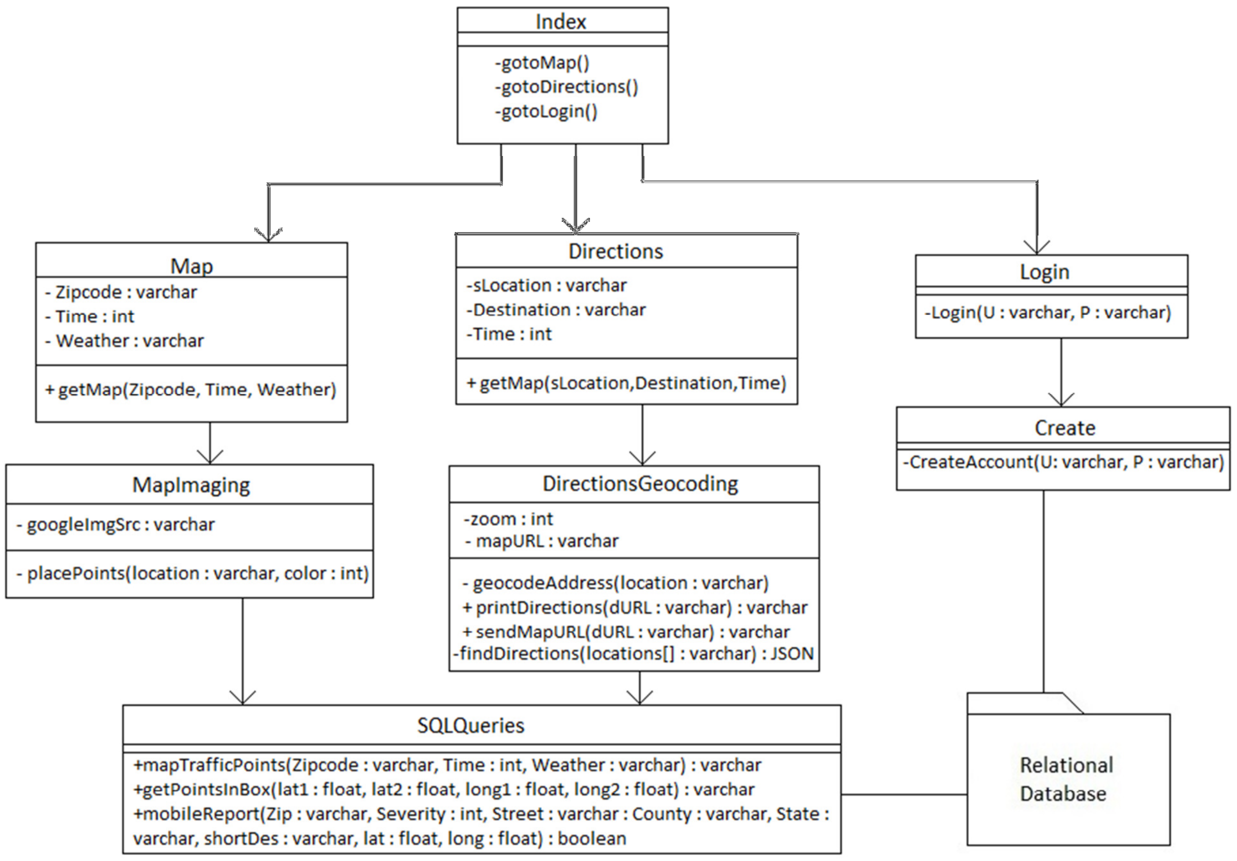
Use Case 7:



This use case handles the responsibility of receiving the traffic intensity data entered by the user. This use case employs the Expert Doer principle, as the user inputs the traffic intensity for his or her location, and sends it to the controller of this system, the PhoneInterface. The PhoneInterface must then retrieve the user's location, so it accesses the MappingService to do so. After the user's location is returned, the PhoneInterface can send the traffic intensity and user's location to the Database to be stored for future use. The user is sent a confirmation message to show that his or her data has been received.

Class Diagrams

Class Diagram for Website:


Index
-gotoMap()
-gotoDirections()
-gotoLogin()
Map
- Zipcode : varchar
- Time : int
- Weather : varchar
+ getMap(Zipcode, Time, Weather)
Directions
-sLocation : varchar
-Destination : varchar
-Time : int
+ getMap(sLocation,Destination,Time)
Login
-Login(U : varchar, P : varchar)
Create
-CreateAccount(U: varchar, P : varchar)
MapImaging
- googleImgSrc : varchar
- placePoints(location : varchar, color : int)
DirectionsGeocoding
-zoom : int
- mapURL : varchar
- geocodeAddress(location : varchar)
+ printDirections(dURL : varchar) : varchar
+ sendMapURL(dURL : varchar) : varchar
-findDirections(locations[] : varchar) : JSON
SQLQueries
+mapTrafficPoints(Zipcode : varchar, Time : int, Weather : varchar) : varchar
+getPointsInBox(lat1 : float, lat2 : float, long1 : float, long2 : float) : varchar
+mobileReport(Zip : varchar, Severity : int, Street : varchar : County : varchar, State : varchar, shortDes : varchar, lat : float, long : float) : boolean
Relational Database

Use Case 1:

The index class is the class that calls other classes to be executed and holds the data from the website. One of the destinations of the index page is the Map page. The Map class reads the input sent by the User and sends it to the MapImaging class. The MapImaging class finds all traffic points in that area by using an SQL query to the database.

Once the array of locations is returned, the MapImaging class will enter the points according to severity into the URL it plans to return. Once all points have been processed, the MapImaging class will return the URL of the image to be displayed to the Map class.

Use Case 2:

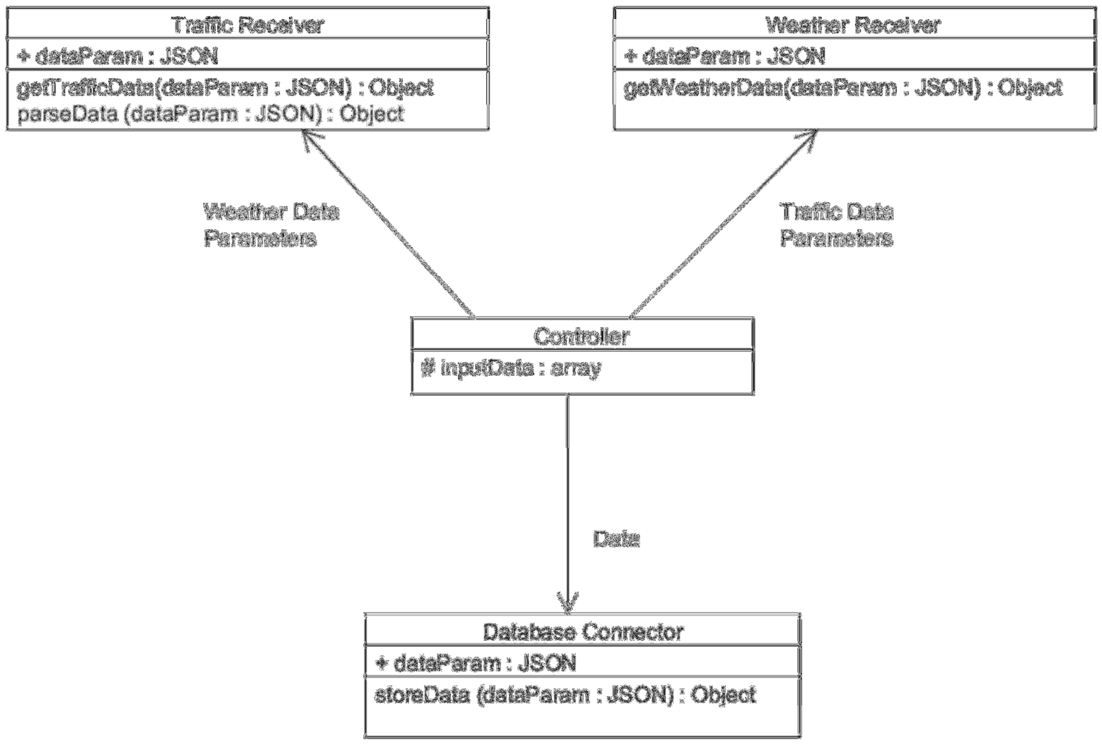
From the index class, the Directions class can be reached. On the directions page, the User must enter their starting location, destination, and time that they plan to leave. These items are sent to the DirectionsGeocoding class.

First, the addresses the User entered must be geocoded to find their latitudes and longitudes. Once these have been found, the DirectionsGeocoding class queries the database to find all locations within a box bounded by the starting location and destination.

Once all points have been returned, DirectionsGeocoding then finds the route that avoids all points of high severity in the form of a JSON object. From that JSON object, the list of maneuvers that need to be followed can be returned. The mapURL can also be completed by finding the necessary zoom for the map, as well as the route needed. Both the list of directions and the map are returned to the User.

The website Class Diagram somewhat follows the Design Pattern for state. Once the User reaches the main index page, the User is allowed the option of going to either page, the Map or Directions. Once the User has reached one of those pages, that state's functions will complete, and the User will have the option of repeating his or her actions with different inputs, or to go back to the main page.

Class Diagram for Scripts:



Use Cases 4 and 5:

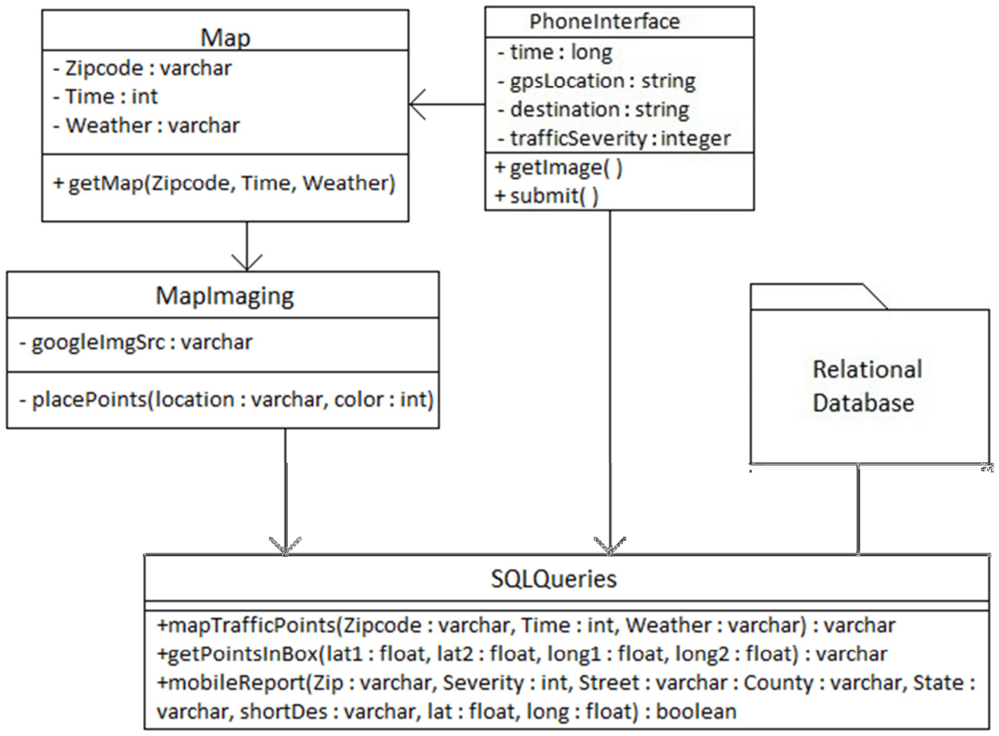
The controller class calls the other classes to execute their respective functions. Initially containing the desired data parameters as an object, the controller passes these parameters, along with an executable Javascript file, to either the traffic receiver or weather receiver. Once given the required data parameters, the receivers will run the scripts.

The traffic receiver will take data from the traffic sites, using the getTrafficData() function, and the weather receiver will take data from weather sides, with the getWeatherData() function. These functions will save the data in a text file.

Afterwards, the controller gives the data to the parser, which extracts the necessary information. The controller then gives this information to the database connector, which have been correctly formatted by the parseData() function. The data is then stored in the database for future use, with storeData().

These classes were designed to follow the Publisher/Subscriber Design Pattern. When the receivers post that they have received an item, the controller is let known and info is stored into the database

Class Diagram for Mobile Application:


Map
- Zipcode : varchar
- Time : int
- Weather : varchar
+ getMap(Zipcode, Time, Weather)
PhoneInterface
- time : long
- gpsLocation : string
- destination : string
- trafficSeverity : integer
+ getImage( )
+ submit( )
MapImaging
- googleImgSrc : varchar
- placePoints(location : varchar, color : int)
Relational Database
SQLQueries
+mapTrafficPoints(Zipcode : varchar, Time : int, Weather : varchar) : varchar
+getPointsInBox(lat1 : float, lat2 : float, long1 : float, long2 : float) : varchar
+mobileReport(Zip : varchar, Severity : int, Street : varchar : County : varchar, State : varchar, shortDes : varchar, lat : float, long : float) : boolean

Use Cases 6 and 7:

The PhoneInterface class is the mobile application that offers two options to the user. It holds data that is used both for obtaining traffic reports and updating the database.

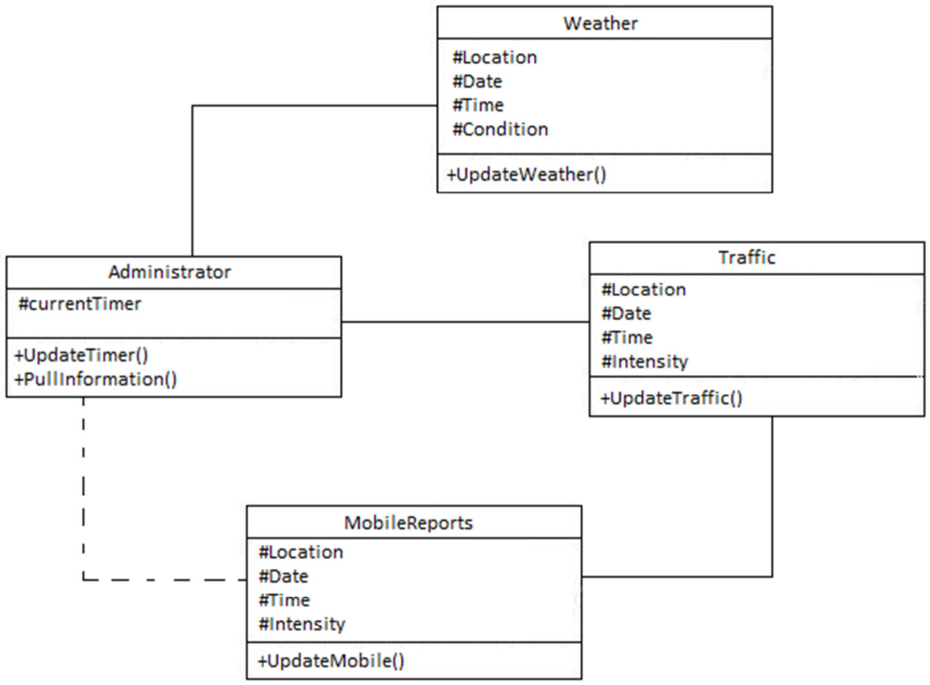
When getImage is invoked, it asks the User for Zipcode, Time, and Weather, similar to Use Case 1. The functions of Use Case 1 are then completed by the main application, and the resultant image is sent back to the PhoneInterface.

When submit is invoked, the PhoneInterface asks the user to submit the traffic severity of the location they are currently in. The GPS location of the Mobile User is found, and the severity along with the User's location are stored into the database through the SQLQueries class.

Design Patterns:

For obtaining a traffic report and submitting a traffic update, the HttpPost used by the mobile application is like a remote proxy in that the required information is bundled into the HttpPost object and sent to the server to be handled. The webservice receives the input and either correctly retrieves the mapInformation for use case 6 or correctly handles inputting an entry into the database for use case 7.

Relational Database:



The information that is stored in the database can be viewed in the diagram above. The Administrator is in charge of controlling when the weather and traffic measurements will be taking place (See Use Case 4 and 5). For weather, the database holds the location, date, time, and location of the particular occurrence. For traffic, the database stores the location, date, time, and traffic intensity for each traffic congestion that occurs. The database also stores the traffic reports given by mobile users. The Administrator can access these reports, but controlling them is not the Administrator's primary concern. The mobile reports connect to the traffic section of the database to combine web service data with the user data given.

Data Types and Operation Signatures

Index:

- Operation
  -gotoMap – brings the User to the Map interface
  -gotoDirections – brings the User to the Directions interface
  -gotoLogin – brings the User to the Login screen

Map:

- Attributes
  -Zipcode – User input of the Zipcode
  -Time – User input of time
  -Weather – User input of weather
- Operation
  +getMap(Zipcode, Time, Weather) – invokes the MapImaging class to place an image showing the traffic intensity points for the inputs received

MapImaging:

- Attributes
  -googleImgSrc – the image URL of the final image found by placing points on the static map.
- Operation
  +placePoints(location, color) – appends the googleImgURL with text that displays a latitude/longitude combination with a color to show its traffic intensity

Directions:

- Attributes
  -sLocation – User input of starting location
  -Destination – User input of final destination
  -Time – User input of time planned to leave
- Operation
  getMap(sLocation,Destination,Time) – returns the list of directions along with a map showing the route generated by the DirectionsGeocoding class.

DirectionsGeocoding:

- Attributes
  -zoom – the calculated zoom value based on the length of the route found
- Operation

geocodeAddress(address) – finds the latitude and longitude of the address input by the User.
findDirections(locations[]) – finds a route from the User's starting location to destination while avoiding the points with high traffic severities.
printDirections(dURL) – uses the directions URL found, and finds the list of maneuvers inside. These inputs are then sent to be displayed on the Directions class.
sendMapURL(dURL) – sends the mapURL to the directions class to display the image of the route.

SQLQueries:

- Operation
  mapTrafficPoints(Zipcode, Time, Weather) – get the list of traffic points found in the database given the User inputs for Use Case 1
  getPointsInBox(lat1, lat2, long1, long2) – find all points within bounded box set by the User's starting location and destination
  mobileReport(Zipcode, Severity, Street, County, State, shortDes, Lat, Long) – inputs the report given by the Mobile Application into the database.

Login:

- Operation
  Login(Username, Password) – checks if a valid user, then logs the user in.

Create:

- Operation
  createAccount(Username, Password) – checks if account already exists, and creates the new account.

DatabaseConnector:

- Attributes
  -JSON dataParam (contains the parameters requested by the controller)
- Operation
  getTrafficData(JSON dataParam) - get
  storeData(JSON dataParam)

TrafficReceiver:

- Attributes
  - JSON dataParam – list of parameters received by web service
- Operation
  getTrafficData(JSON dataParam) - gets traffic data from web service

WeatherReceiver:

- Attributes
  -JSON dataParam - list of parameters received by web service
- Operation
  getWeatherData(JSON dataParam) - gets weather data from web service
- Operation
  parseData(JSON dataParam)

PhoneInterface:

- Attributes
  -long time
  -String gpsLocation
  -String destination
  -integer trafficSeverity
  -String inputData[] – combines list of attributes into an array of items to be submitted.
- Operation
  getImage() – gets the image of the map desired by the Mobile User
  submit(String inputData[]) – submits traffic report

	UC 1	UC 2	UC 4	UC 5	UC 6	UC 7
Index	Allows User to access Map page	Allows User to access directions page				
Map	Receives User Inputs and displays final map				Receives User Inputs and displays final map	
MapImaging	Creates image of points on map				Creates image of points on map	
Directions		Receives User input and displays list of directions and image				
Directions Geocoding		Geocodes User inputs and finds necessary route to be displayed				
SQLQueries	Finds points within User input parameters	Finds list of points within User parameters				Inputs mobile user report into database
Login						
Create						
Database Connector			Submits traffic report into database	Submits weather report into database		
Traffic Receiver			Finds traffic information from webservice			
Weather Receiver				Finds weather information from webservice		
Phone Interface					Receives User inputs and displays map image	Allows user to submit mobile report

Object Constraint Language (OCL) Contracts

Map
Invariants: Registered Users and non-registered Users can access this page.
Precondition: User has filled out all inputs. User has input a zipcode within the Tri-state area.
Postcondition: Map is displayed for the User. If User is logged in, this search is saved on recent searches.

Directions
Invariants: Registered Users and non-registered Users can access this page.
Precondition: User has filled out all inputs. User has input locations within the Tri-state area.
Postcondition: Map is displayed for the User. If User is logged in, this search is saved on recent searches.

CreateAccount
Invariants: Page is only available to Users not logged in
Precondition: Username must not already be taken
Postcondition: New account is created. Brings User back to home page.

Login
Invariants: Page is only available to Users not logged in
Precondition: Username and Password combination must match that of one in the database
Postcondition: User is logged in, and brought back to main page.

SubmitMobileReport
Invariants: Page is available to all Mobile Users
Precondition: all inputs must be filled out. User must be located within the Tri-state area.
Postcondition: New entry is placed in database. Mobile User is notified of result.

System Architecture and System Design

Architectural Styles

The most important architectural style in this project is the Client/Server style. The user is only meant to interact with the basic user interfaces of the Web and Mobile Applications. After the user inputs their relevant information, it is the server's job to process all of the information given in order to display what the user needs. The server sends requests to server-side classes which the user will never interact with. This type of architectural style is ideal in this situation, for it makes it the easiest for the user. All the user will control are a series of inputs (text boxes, dropdown menus, radio buttons, etc.) which are all very simple to interact with. Through using these simple inputs, a complex output is displayed for the user.

The Client/Server architectural style is most reliable when the classes execute in a similar way each time. If there were many possibilities for function use when the inputs are given, another architectural style should be used. However, the same functions and classes are used every time, so the Client/Server style is a good fit.

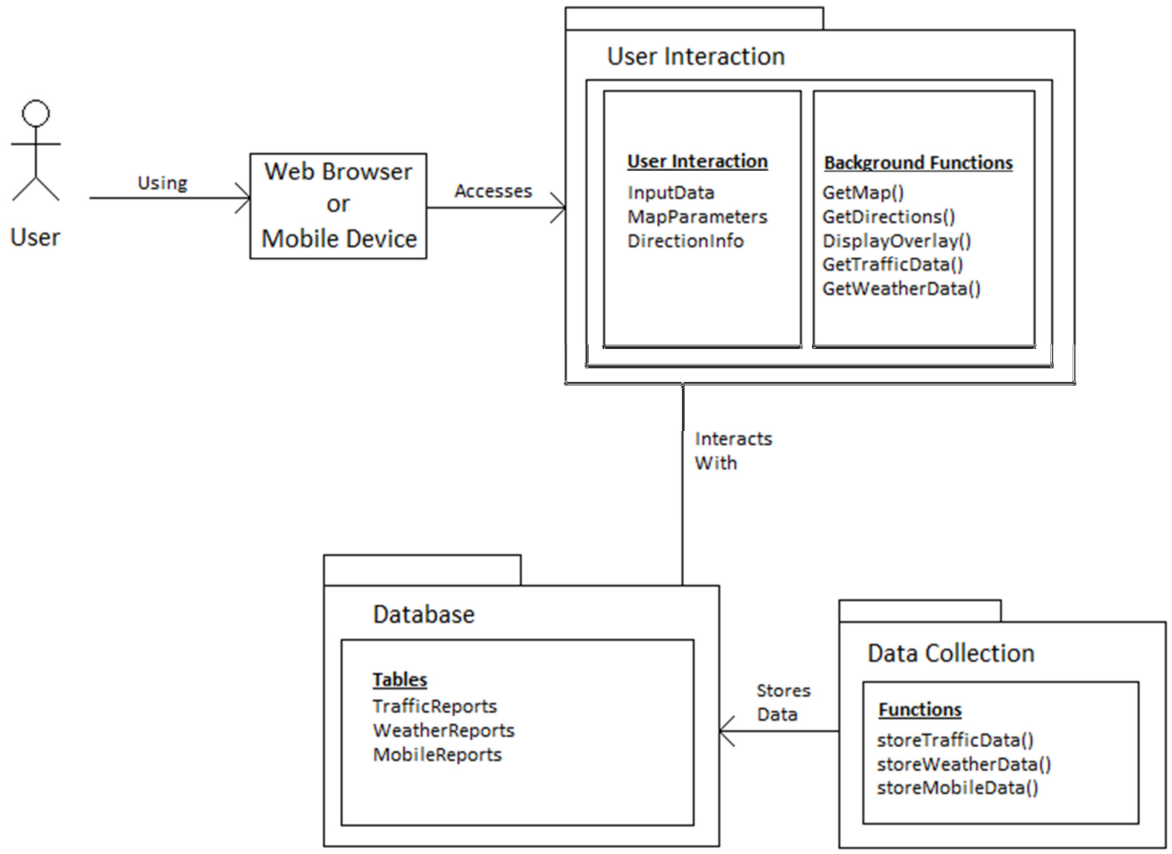
Identifying Subsystems

The traffic monitoring service will be a website that the clients will interact with. These users will access the web Application through the browser of their choice. The network protocol in this case is HTTP. This allows us to reach the highest number of users through a web Application. All user interaction with the system takes place in the User subsystem. The user subsystem contains all of the user inputs, the map output, and the directions output given by the system. These are all of the objects the user will interact with during their use of the Application. This subsystem connects directly to the database of the system. The database holds all of the information necessary to achieve the output the user desires.

The second subsystem is the Data Collection subsystem. This system involves the traffic and weather gathering services. The traffic and weather receivers access the relevant web services in order to find more data to store into the database. Because it stores data into the database, this subsystem clearly also connects to the database.

The final subsystem is the database. Its functions are shown through its interactions with the previous two subsystems.

UML Package Diagram



Mapping Systems to Hardware

The data stored in the traffic monitoring system is stored in a database. Preferably we will have a server to deploy the website and another server or multiple servers to store all of the data in the database. Currently we are relying on outside web hosting that may host deployment and data storage on the same server.

Persistent Data Storage

MYSQL is the chosen language to control database storage. There are three tables that the database stores: Traffic, Weather, and MobileReports.

Database Tables			
Field	Type	NULL	Default
Traffic_Display			
Index	AUTO_INCREMENT	NO	0
Date_Time	Timestamp	NO	00-00-0000 (Date) 00:00:00 (Time)
Zipcode	Varchar(5)	NO	-----------
Latitude	Decimal(0,0)	NO	0.00
Longitude	Decimal(0,0)	NO	0.00
Traffic Intensity	int	NO	0
Address	Varchar(100)	YES	-----------
County	Varchar(20)	NO	-----------
Weather	Varchar(20)	YES	-----------
Time_Value	Int	NO	0
Weather			
Index	AUTO_INCREMENT	NO	0
Timestamp	Timestamp	YES	00-00-0000 (Date) 00:00:00 (Time)
Zipcode	varchar(5)	NO	----------
Condition	varchar(20)	NO	----------
Mobile Report			
Index	AUTO_INCREMENT	NO	0
Zipcode	Varchar(5)	NO	-----------
Date_Time	Timestamp	YES	00-00-0000 (Date) 00:00:00 (Time)
Latitude	Decimal(0,0)	NO	0.00
Longitude	Decimal(0,0)	NO	0.00
Severity	int	NO	0
Address	Varchar(100)	YES	-----------
County	Varchar(20)	NO	-----------
State	Varchar(20)	NO	---------
Short_Descrip	Varchar(200)	NO	---------

Global Control Flow

Execution Orderness

Our system is two- fold, both procedure- driven and event- driven. The system is procedure-driven in that the user can request to use the traffic data any time. All information from the time of request as well as previously stored information is immediately retrieved. It is also event-driven because the weather and traffic receivers take available data from websites and store them, and must wait one hour to repeat their functions.

Time Dependency:

Our system is an event- response type, with no concern for real time. The user can request to use traffic data at any point. All information from the time of request as well as previously stored information is immediately retrieved. The only timers in the system are for the weather and traffic receivers, which take available data from websites and store it into a database, every hour. However, this is not a constraint on the system.

Concurrency:

The mobile application makes use of two threads: a main thread which deals with the UI and user input, and a network thread that handles the connection from the app to the web service. The network thread sets up the HttpPost with the correct post keys and interacts with the web script. The returned image is displayed by the main UI thread.

Hardware Requirements

User is required to have a functional computer, with a screen resolution of at least 800 x 600. The user must be able to use a browser compatible with the web Application.

Alternatively, in order to use the mobile application, the user must have an Android mobile device, capable of accessing the internet.

The database requires 2 GB in order to store traffic and weather data gathered from web services.

Algorithms and Data Structures

The algorithm to calculate traffic severity has been changed to rely on Mapquest's severity value.

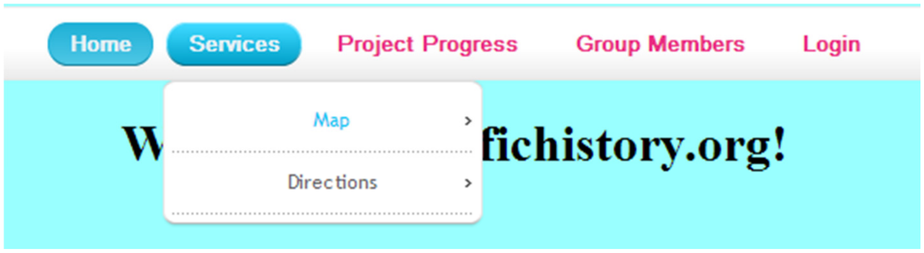
For a more clear and concise view of traffic history, we developed our own algorithm to clear up the traffic data. This allows for less points needed to be displayed on a map and can give the User an easy time reading a map. Our database contains a weather table which contains the weather at certain times for all zipcodes we support. There is another table that contains all the traffic incidents with time and position of the incident. The third table is the traffic incident table that has the condensed traffic incidents and used to display. The algorithm takes all traffic incidents of the day and constructs a hash of incidents with position as a key and the severity as the value. The hash is then used to get all incident points along a certain road at the same time with the same weather. The algorithm also creates the same hash from the condensed traffic incident table. The hashes are compared by street. Along that same street traffic incidents of the day are added to the closest point within .5 miles. If there are no points within .5 miles that incident location becomes a new location to be displayed in the condensed traffic incident table. The severity values are updated each time a traffic incident is added to a condensed traffic incident point.

The algorithm to decide how severe a condensed traffic incident point is using the severity value and dividing it by the number of traffic calls the data collection script makes.

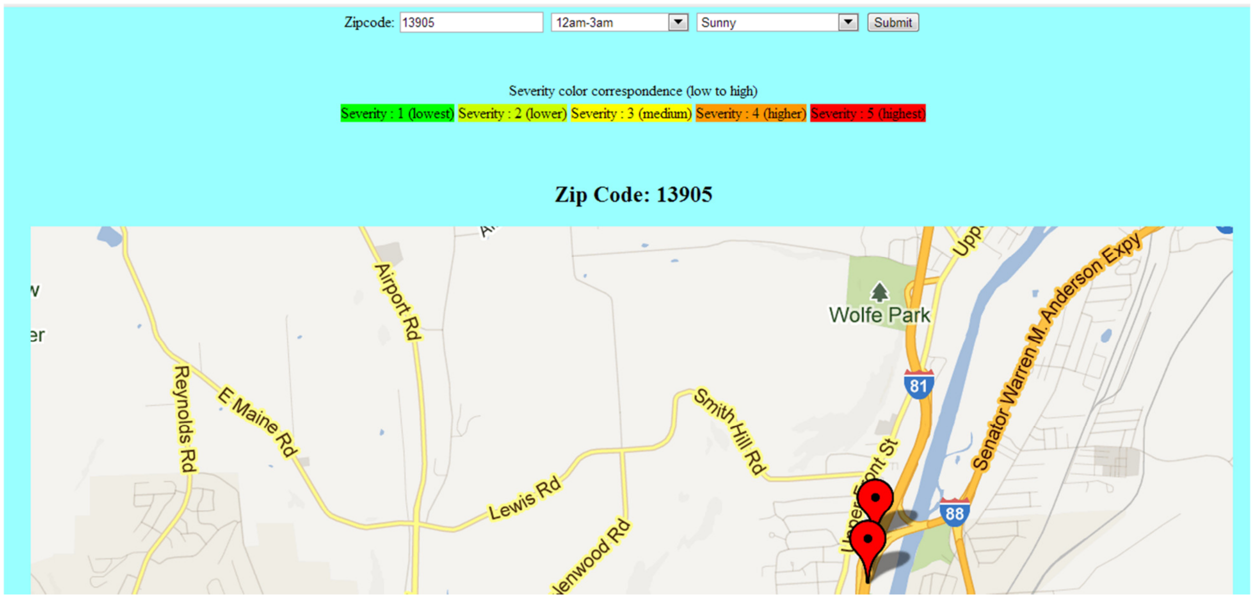
The algorithm to determine the route for directions uses the condensed traffic points. We have the start and end points. Using the start and end points, a bounded box is formed. All points within that box that have high traffic intensities are then found in a list. Using the condensed traffic points we can construct a call to the Directions Service. In that call to Directions Service, we can indicate points between the start and end points that the route should avoid. We weight the points to be avoided based on the condensed traffic point's severity value.

No complex data structures are used in this project. The majority of the complexity deals with manipulation of the data stored in the database.

User Interface Specifications



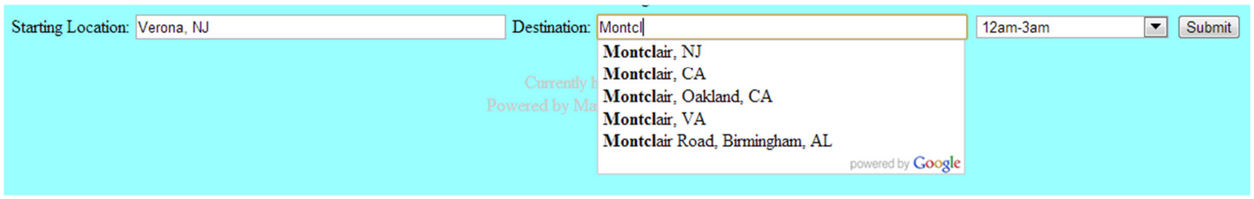
On the starting index page, the User is allowed to choose whether to go to the Map or Directions pages.


Zipcode: 13905 12am-3am Sunny Submit

Severity color correspondence (low to high)
Severity : 1 (lowest) Severity : 2 (lower) Severity : 3 (medium) Severity : 4 (higher) Severity : 5 (highest)

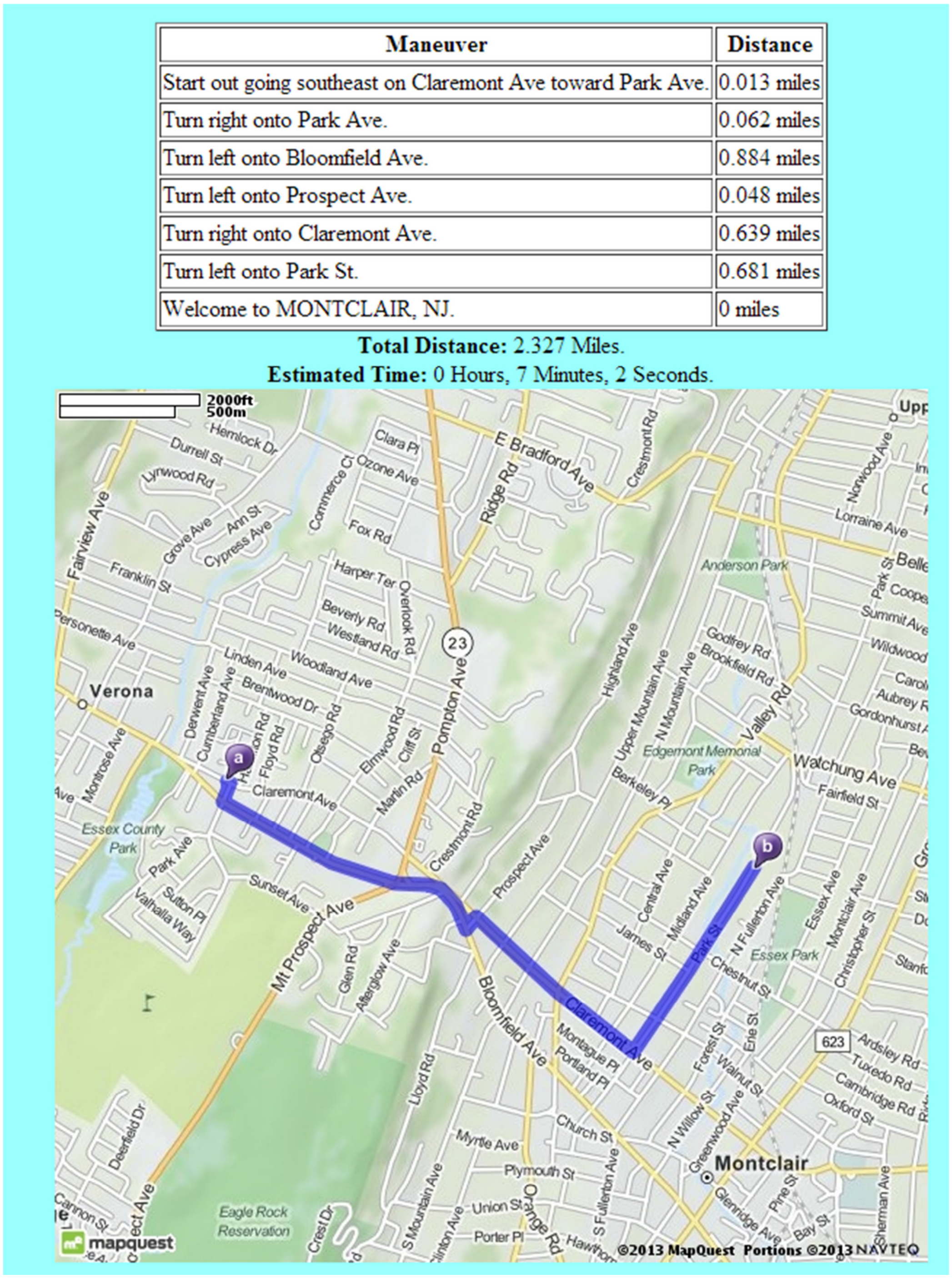
Zip Code: 13905

The Map page gives the User a textbox to enter the zip code, and dropdown menus for time and weather. After submission, a map is displayed showing the traffic intensity points.


Starting Location: Verona, NJ Destination: Montcl 12am-3am Submit

Montclair, NJ
Montclair, CA
Montclair, Oakland, CA
Montclair, VA
Montclair Road, Birmingham, AL

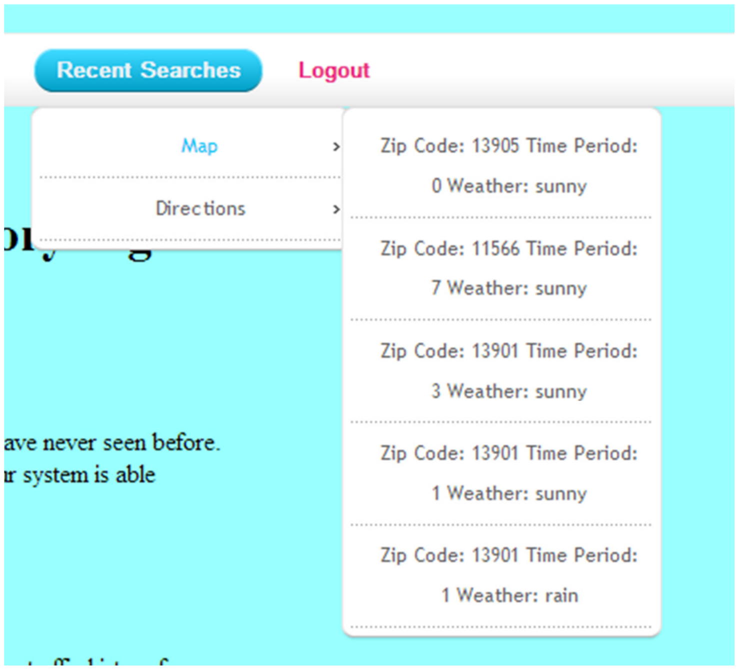
On the directions page, the user has inputs of textboxes to show the starting location and destination. An autocomplete implementation by Google is used to allow the User to more easily choose the location of their choice.



Maneuver	Distance
Start out going southeast on Claremont Ave toward Park Ave.	0.013 miles
Turn right onto Park Ave.	0.062 miles
Turn left onto Bloomfield Ave.	0.884 miles
Turn left onto Prospect Ave.	0.048 miles
Turn right onto Claremont Ave.	0.639 miles
Turn left onto Park St.	0.681 miles
Welcome to MONTCLAIR, NJ.	0 miles

Total Distance: 2.327 Miles.
Estimated Time: 0 Hours, 7 Minutes, 2 Seconds.

Example of the result for the directions interface. Shows list of directions along with the chosen route.



Zip Code: 13905 Time Period: 0 Weather: sunny

Zip Code: 11566 Time Period: 7 Weather: sunny

Zip Code: 13901 Time Period: 3 Weather: sunny

Zip Code: 13901 Time Period: 1 Weather: sunny

Zip Code: 13901 Time Period: 1 Weather: rain

Once the User has logged in, a recent searches function is available to them. This function displays the last 5 searches done on the Map and Directions Interfaces.

Design of Tests

The user will input the area of traffic history they desire, the weather, the time of day, and day of the week. The output will construct a map with the traffic history overlay points on the map. 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 (Map): User will input:
Zip Code: 19305
Weather: Sunny
Time: 12am-3am

Test Case 2 (Map): User will input:
Area: 11566
Weather: Sunny
Time: 9pm-12am

Test Case 3 (Directions): User will input:
Starting Location: Verona, NJ
Destination: Montclair, NJ
Time: 9am-12pm

Test Case 4 (Mobile): User will input:
Zip Code: 19305
Weather: Sunny
Time: 12am-3am

Additional Testing was done between the website Application and the Mobile Application. At first, we were unsure as to how to allow the Mobile Application to access the Database tables. We concluded that it makes the most sense to work through the website Application to access all of the information. To accomplish this, data must be sent from the Mobile Application to the website Application. We decided to make the Mobile Application send an object containing all of the user inputs, and to create a method in the website Application that can parse the object that is being sent. The website Application can then function as normal to get the necessary data, and it can then send the map overlay to the Mobile Application.

History of Work, Current Status, and Future Work

Our previous plan of work had a linear progression to it. We expected each step to be done sequentially, but had given each of subgroup the flexibility to work past due dates. We had also split the team into groups for data collection/database management, traffic and direction design/implementation, and mobile application development. The data collection/database management group consisted of Peter and Matt. The traffic and direction design/implementation group consisted of Kevin and John. The mobile application development group consisted of Geoff and Mike. We had no central time keeper that may have been useful to keep track of how long each sub group worked on the section of the code. Our initial use cases and design were sufficient for the project.

Initial Plan of work

Milestone	Original Deadline	Date completed	Initial Group	Final Group
Script collection	2/28	Tentative 3/23 (basics done)	Matt and Peter	Kevin, John, Matt, and Peter
Mobile Application Page	3/7	Tentative 3/30 (basics done)	Mike and Geoff	Mike and Geoff
Traffic Algorithm	3/14	Tentative 3/23 (basics done)	Kevin and John	Kevin and John
Traffic map display	3/21	Tentative 3/27 (basics done)	Kevin and John	Kevin and John
Mobile traffic map	3/24	Tentative 3/27(basics done)	Mike and Geoff	Mike and Geoff
Integration of mobile	3/17	Tentative 3/30 (basics done)	Mike, Geoff, Kevin, John	Mike, Geoff, Kevin, John

The plan of work changed considerably from what we had initially assigned. Some milestones could not be met in a week due to other class constraints or obligations. In the end Mike and Geoff managed to complete most of the mobile-side application with integration testing with Kevin and John. Stated before is the flexibility of working past due dates, which were pushed to the limits by the scripting team which lead to an increased work load on other teams. Kevin and John eventually took over the data collection scripts and database management since it was vital to the traffic and direction algorithm. This lead to most of the work done between demo 1 and demo 2.

Final history of work:

Milestone	Date completed	Group
Initial Database Scripts	3/10	Peter and Matt
Traffic data collection	3/12	Kevin
Weather data collection	3/15	John and Peter
Traffic algorithm code	3/19	Kevin and John
Traffic map display	3/21	Kevin and John
Direction algorithm code	3/26	Kevin and John
Traffic map display	3/27	Kevin and John
Mobile report traffic	3/19	Mike and Geoff
Mobile traffic map	3/5	Mike and Geoff
Mobile integration testing	3/21	Mike, Geoff, Kevin, John
Website ease of use	4/30	Peter and Matt

We had some miscommunication and delays, but we managed to complete most of our core objectives.

Current accomplishments

- Traffic and Weather data collection
- Zipcode table
- Traffic algorithm
- Traffic map display
- Directions Algorithm
- Directions map display
- Mobile traffic map
- Mobile traffic report

Our current product is a website that the user can access. On the website there are different pages for directions and traffic history viewing. These pages have inputs for the user and validation checks. The pages will submit the inputs and return the map image or desired output. There is also a severity chart for the user to see how severe a point is on the map. Our data collections scripts have been running without problem. The mobile application is usable and can report traffic and view traffic history of a zip code.

Future work:

There is plenty of development into more mobile features, however we were limited by our own processing and storage limits. We wanted to store information of the user while the app was on to record speed of roads, time, weather, and location of user. Eventually we can build up a database of information of each road's average speed and incidents. Using the average speed of roads and the relevant data, we can process the data and determine which roads have slow speeds at which times. This would allow us to predict in more detail where traffic prone areas are instead of relying on third party information about traffic incidents.
For the website and data collection aspect, we had scripts to trawl the net for more traffic reports. This would allow for more accurate reporting. The missing key is a standard traffic severity algorithm. Using an array of key words seem to be too unreliable, but given no other data there is not much else to use. There is also possibility of expansion from the tri-state to the country to the world. This would rely on much more storage or perhaps daily caching of traffic data. Some data collection improvement would be change it to be event driven, so the database is updated whenever a traffic incident occurs rather than to check every 3 hours.

Future implementations would be to integrate all data of an area into the page. This would allow the user to not only see the traffic history, but the current traffic, weather reports, public transportation, or additional data of the area. The area could be expanded to see which places are frequented by users. Also there could be work to be done to create our own interactive map and move away from outside mapping services. The mapping website could eventually be expanded to rival the bigger map services, but that would require a full-time staff to develop and the funds necessary to accomplish that.

References

1. Marsic, Ivan. *Software Engineering*. 2012.
2. 511 NJ - *View Traffic Incidents*. 24 February 2013. <511nj.org /IncidentList.aspx>.
3. 511 PA – *511 PA. Travel Info to Go*. 24 February 2013. <511pa.org>.
4. 511 NY – *511 NY. Get Connected to Go* 24 February 2013. <511ny.org>.
5. Wunderground – *Weather Forecast and Report*. 3 March 2013. <http://www.wunderground.com/>
6. Google Maps. 15 February 2013. <maps.google.com>.
7. Mapquest API – *Mapquest Platform Web Service*. 4 April 2013 <mapquestapi.com>.
8. Learning Perl – *The Perl Programming Language*. < http://www.perl.org/>.
9. W3Schools – *W3Schools Online Web Tutorials*. 7 March 2013. < http://www.w3schools.com/>
10. Learning JQuery – *JQuery*. 15 March 2013. < http://jquery.com/>
11. Google Maps for Developers – *Google Maps API.* 24 February 2013. < https://developers.google.com/maps/>
12. Learning PHP – *PHP – Hypertext Preprocessor.* 1 March 2013. < http://us.php.net/>
13. Android API – *Package Index – Android Developers*. 27 February 2013 < developer.android.com/reference>