

TRAFFIC MONITORING SYSTEM

Trick-The-Traffic

GROUP NUMBER: 7



GROUP MEMBERS:

Kush Patel
Tejas Ravi
Jeffrey Gillen
Shreyas Bhandare
Kisholoy Vinayak Ghosh
Aaditya Shukla

kush.patel@rutgers.edu
tr332@scarletmail.rutgers.edu
jeffrey.gillen@rutgers.edu
shreyas.bhandare@rutgers.edu
kisholoy.v.ghosh@rutgers.edu
aaditya.shukla@rutgers.edu

Breakdown of Individual Contribution

Task	Kush Patel	Jeffrey Gillen	K.Vinayak Ghosh	Tejas Ravi	Aaditya Shukla	Shreyas Bhandare
Summary of changes	equally	equally	equally	equally	equally	equally
Customer Statement of Requirements	equally	equally	equally	equally	equally	equally
Glossary of Terms	equally	equally	equally	equally	equally	equally
System Requirements	25%	0%	25%	0%	25%	25%
Functional Requirement Specifications	20%	20%	20%	0%	20%	20%
Effort Estimation	equally	equally	equally	equally	equally	equally
Domain Analysis	23%	10%	23%	0%	23%	23%
Interaction Diagrams	25%	0%	25%	0%	25%	25%
Design Patterns	equally	equally	equally	equally	equally	equally
Class Diagrams and Interface Specification	25%	0%	25%	0%	25%	25%
OCL contracts specification	25%	0%	25%	0%	25%	25%
System Architecture	25%	0%	25%	0%	25%	25%
Algorithms and Data Structure	0%	0%	0%	100%	0%	0%
User Interface design and implementation	25%	0%	25%	0%	25%	25%
Design of Tests	equally	equally	equally	equally	equally	equally
History of Work and Future Work	equally	equally	equally	equally	equally	equally
References	equally	equally	equally	equally	equally	equally

Table of Contents

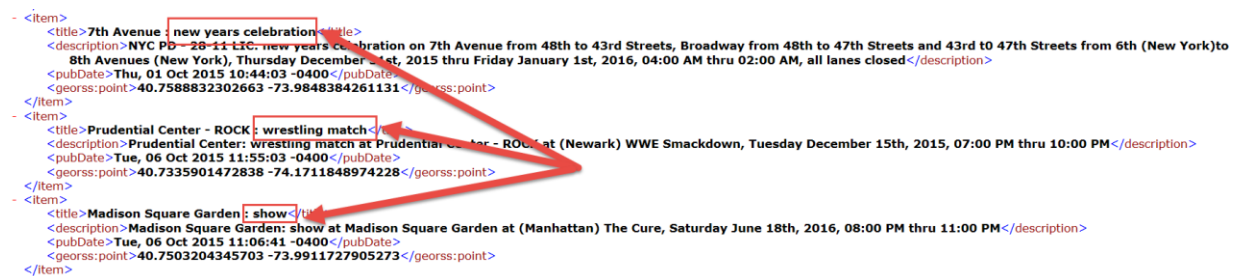
I.	Breakdown of Contributions.....	2
II.	Table of Contents.....	3
III.	Summary of Changes.....	4
IV.	Customer Statement of Requirements.....	5
V.	Glossary of Terms.....	8
VI.	System Requirements.....	9
VII.	Functional Requirements Specifications.....	13
VIII.	System Sequence Diagrams.....	27
IX.	User Effort Estimation.....	31
X.	Domain Analysis.....	35
XI.	Interaction Diagrams.....	43
XII.	Class Diagrams.....	52
XIII.	System Architecture and System Design.....	59
XIV.	Algorithms and Data Structures.....	65
XV.	User Interface Design and Implementation.....	68
XVI.	Design of Tests.....	76
XVII.	History, Current and Future Work.....	77
XVIII.	References.....	80

Summary of Changes

Traffic Data Source:

Initially we had selected 511NJ.org as a data source for accident and traffic data. However after collecting some data, we noticed that the data feed is not updated as frequently. Since the 511NJ.org is a local state webpage for displaying data, we know the original data feed is from users calling in and reporting issues. The input seemed to be manually entered. An example is shown in the image below.

The data source has the “Type of Incident” in the “Title” tag of the XML file. For every new entry, it is hard to predict that the correct text will be used to represent that entry. For example, for category type “Events”, the text needs to be consistent. Below XML has different strings for same “Event” type. So if anyone is parsing through and searching for specific type of incidents, it can be easily missed out due to this reason.



```
- <item>
  <title>7th Avenue : new years celebration</title>
  <description>NYC PD - 28-11-11C: new years celebration on 7th Avenue from 48th to 43rd Streets, Broadway from 48th to 47th Streets and 43rd to 47th Streets from 6th (New York) to 8th Avenues (New York), Thursday December 31st, 2015 thru Friday January 1st, 2016, 04:00 AM thru 02:00 AM, all lanes closed</description>
  <pubDate>Thu, 01 Oct 2015 10:44:03 -0400</pubDate>
  <georss:point>40.7588832302663 -73.9848384261131</georss:point>
</item>
- <item>
  <title>Prudential Center - ROCK : wrestling match</title>
  <description>Prudential Center: wrestling match at Prudential Center - ROCK at (Newark) WWE Smackdown, Tuesday December 15th, 2015, 07:00 PM thru 10:00 PM</description>
  <pubDate>Tue, 06 Oct 2015 11:55:03 -0400</pubDate>
  <georss:point>40.7335901472838 -74.1711848974228</georss:point>
</item>
- <item>
  <title>Madison Square Garden : show</title>
  <description>Madison Square Garden: show at Madison Square Garden at (Manhattan) The Cure, Saturday June 18th, 2016, 08:00 PM thru 11:00 PM</description>
  <pubDate>Tue, 06 Oct 2015 11:06:41 -0400</pubDate>
  <georss:point>40.7503204345703 -73.9911727905273</georss:point>
</item>
```

Additionally, the traffic data did not have any congestion severity parameters. Although it could be easy to map this out if sufficient information is provided. The data was lacking such mapping pointers.

The data source was changed to WazeAPI for Traffic, YelpAPI for restaurants and ParkWiz API for parking lots. These have the necessary information need to accomplish goals.

Requirements/Use Cases:

After some feedback from TA, we had to modify REQ since they were too general and did not describe the project use cases appropriately. Those were changed when new Traffic API was selected after the first demo. The use cases were broken down as well because multiple use cases were combined into one as part of our first report.

Android Development:

In the later phase of our project, we decided to build an android application that incorporated some of the features of the web application. This added an additional actor however the participating actors remained the same because the data feed should be same as the web application.

Customer Statements of Requirements

Problem Statement

The internet has advanced tremendously over the past decade. In today's day and age, our lives have become dependent on using the internet. From checking email to deciding what food to eat, every activity is governed by various websites. Websites also offer map applications that offer services such as navigation and traffic updates. Navigation is very essential if someone wants to go to an unknown location. Live time traffic is also very essential to determine congestion on a particular route and plan routes accordingly.

There are several services that offer this feature; for example, "Traffic.com" and "Yahoo! Maps Live Traffic" etc. All these services only account for current traffic conditions in a given geographical area. Even though current traffic models give a good idea of the congestion on a particular highway, it still is not completely accurate because the prediction can be based off of very few traffic reports. If there are no traffic reports, one cannot know what the traffic levels look like at a specific time of the day.

With the help of current traffic intimation models, a user will get to know about current traffic updates dynamically as he travels along the route. However, it would be more meaningful if the user knows what to expect in terms of traffic even before starting from his starting location. This project aims to take traffic prediction to another level. Instead of just using current traffic patterns, we will use historic traffic data to give the user a better estimate of traffic patterns. Below we have proposed a solution to this problem. Several questions one would ask are What? How? Why? Who? Etc. We will explain each aspect below.

Additionally, there are researchers and government agencies who plan our road developments such as adding lanes to highways, building bridges and minimizing the scheduling for constructions on roads. Such agencies can benefit from applications that will give them severity information for specific ZipCode. If a specific area has high congestion, due to volume of cars (not because of accidents) the government will need to think about adding an extra lane on the highway to reduce traffic.

Another problem that users encounter is not knowing where police traps are. Users pay awful amount of money in fines due to speeding tickets. If they can know beforehand where police traps are, they can slow down and avoid the ticket.

Solution Statement

This project will create a One Stop Site for a user on a busy weekday or a relaxed weekend. It will give the user the ability to query for traffic information using a zip code, day of the week and time of the day. So when a user does perform a query, the traffic prediction would be based on the current predictions as well as the historic traffic data that the route experienced over the past few months. We will be using a prediction algorithm that will predict approximately how long it will take (in seconds) in traffic along a road. In this way, the user will be able to see what route is experiencing congestion and hereby decide whether to take another route. Additionally, traffic congestion is of different kinds namely low, medium low, medium, medium high and high. The service will ideally be able to suggest routes that avoid areas with a very high projected concentration of traffic. User will be able to add the severity layer on top of his navigation route and hereby differentiate congestion via severity level. This will allow the user to find an ideal route with minimal exertion. The assimilated data and traffic information could be used in the future to help in improving road infrastructure or traffic diversion based on traffic patterns.

Normal traffic can be predicted. However, there are random events that may occur along a road that cannot be predicted. These random events can be Accidents, Hazards, and Construction etc. Just knowing where traffic may help, however knowledge of random events can be very helpful. If an accident occurs today, the prediction model might not be able to get the most accurate traffic delay (time) because of few reasons. There might not be enough traffic reports or the historical traffic data may not reflect an accident for a given location which means there may not be congestion. Hence these incidents can identify possible major delays and user can plan accordingly. As an add on to live alerts, we will also be having Police Trap alerts that tells the user whether a cop has a speed trap at a given intersection. To verify accuracy of these traps, the map GUI will display the “number of likes” the given trap has gotten from users in the past hour. The user can therefore use caution while driving and possibly avoid a speeding fine. This application helps in saving time, and now money too!

We have already emphasized how much important time is. With this application, we will be saving the user more time by providing them information such as parking lots and restaurants. In this way, user would not need to switch to another application to obtain external information. This data is obtained from reliable sources such as Yelp and ParkWiz for restaurants and parking lots respectively.

The front end of the software is divided up into two separate parts. They are the following:

1. Website Application: User can register home zip code and have map give traffic prediction for home zip code upon login. They can enter custom zip code anytime. All the above features will be available on the web application.
2. Android Application: User can obtain live alerts and incidents on their portable smartphone. They can also see the parking lots and restaurants.

Lastly, the web application will also allow the user to run custom reports by zipcode. Through this report, they will be able to trend average traffic severity per hour and compare those with each weekday. Additionally, they will also have ability to run chart that shows total number of accident counts as well as police reports. In this way, the user can know which cities are more prone to accidents etc.

Glossary of Terms

1. User: A person who wishes to use the traffic monitoring application to view traffic congestion in a user-entered zip code and be able to choose a faster route to destination
2. ZipCode: A 5 digit number that defines a city or town in the United States. This will be an entry field for the user
3. MySQL Database: A relation database management system that will be used to store traffic incident data as well as weather information
4. Waze API: An application programming interface (API) that our system will use to access traffic incident data provided by third party application that stores Waze Data. This data will be stored into the MySQL Database
5. Yelp API: An application programming interface (API) that our system will use to obtain Restaurant Data to be used with the Google Map
6. ParkWiz API: An application programming interface (API) that our system will use to obtain Parking Lot Data to be used with the Google Map
7. Google Maps JavaScript API: An application programming interface (API) provided by Google that allows our team to integrate congestion, traffic alerts and other data points onto a map.
8. Desktop Application: A web application that a user will be able to access on their web browser of a laptop/desktop computer
9. Mobile Android Application: An android application that a user can access on their android smartphone. The application will provide all the features a desktop application provides

System Requirements

Functional Requirements

Identifier	PW	Requirement
REQ 1	3	The website should allow a user to create an account with their home/preferred Zipcode.
REQ 2	3	The website should allow users to login using their correct account credentials (i.e. username and password)
REQ 3	4	The website home page should load traffic severity information for user's home/preferred zip code that was registered as per REQ1
REQ 4	5	The web application should allow the user to enter any zip code (NJ/NY) and plot congestion data points for that zip code with different severity levels.
REQ 5	4	The website should allow users to access most up-to-date traffic accident alerts
REQ 6	4	The website should allow users to access most up-to-date traffic road construction alerts
REQ 7	4	The website should allow users to access most up-to-date traffic police trap alerts
REQ 8	5	The website should allow user to obtain traffic information for any particular day for a specific hour. User Input would be Day of the Week and Hour. Severity Information for all zip codes would be represented on the map.
REQ 9	4	The website should allow the user to have a navigation route overlaid on top of the severity info obtained in REQ 6. User input would be Start and End Destination Address.
REQ 10	3	The web application shall give the user to plot the restaurants on the map and also display their full name.
REQ 11	3	The web application shall give the user to plot the parking lots on the map and also display the parking lot name.
REQ 12	2	The website application shall allow users to trend total number of accidents for any Zipcode (NY/NJ). Trend would be an hourly curve for the past week. User Input would be Zipcode.
REQ 13	2	The website application shall allow users to trend total number of police reports for any Zipcode (NY/NJ). Trend would be an hourly curve for the past week. User Input would be Zipcode.

REQ 14	2	The website application shall allow users to trend average severity for any Zipcode (NY/NJ). Trend would be an hourly curve for the past week. User Input would be Zipcode.
REQ 15	3	The android application shall give the user to plot the parking lots on the map and also display the parking lot name.
REQ 16	3	The android application shall give the user to plot the restaurants on the map and also display the restaurant name
REQ 17	4	The android application shall give the user to plot the police alerts on the map and also display the number of thumbs up
REQ 18	4	The android application shall give the user to plot the accident alerts and construction alerts on the map
REQ 19	4	The android application shall give the user option to do navigation from a starting point to an ending point on the map
REQ 20	5	The system shall query traffic data from WazeAPI once every hour
REQ 21	5	The system shall query restaurant data from YelpAPI upon admin's zip code query
REQ 22	5	The system shall query parking lot data from ParkWiz API upon admin's zip code query
REQ 23	5	The system shall query Geonames API to map a traffic data point to a Zipcode
REQ 24	5	The system shall store data to MySQL database for REQ20, REQ 21, REQ 22.
REQ 25	5	The system shall use prediction algorithm on database data and write prediction to database column "predicted"
REQ 26	5	The android application shall give the user option to do navigation from a starting point to an ending point

Non-Functional Requirements

Identifier	PW	Requirement
REQ 27	3	The website shall provide the user with a neat and easy to use Interface for the user
REQ 28	2	The system shall allow the administrator to control the frequency of reports for data collection
REQ 29	2	The systems shall allow the administrator to define Bounding Box Co-

		Ordinates for an area to which collect the data for.
REQ 30	3	The application shall provide traffic incident graphing and charting abilities on the webpage (trend over last week) for informational purposes
REQ 31	5	The application shall use a local algorithm (Bayesian) to predict time in traffic observed (In seconds).
REQ 32	1	The application database shall have enough memory to store traffic incident data for over 2 months for different zip codes
REQ 33	1	The application shall have an authentication system before the user can proceed with the services
REQ 34	1	The website application shall be compatible with any browser (Internet explorer, Google Chrome etc)
REQ 35	3	Multiple users should be able to access the website without experiencing slow response time.
REQ 36	4	The data collection script should be independent of the website in the sense that if the data collection script fails due to any reason, the website shall still load and display the latest information from the database. The website should still load and display the stored data.
REQ 37	2	The android application will be compatible with any manufacture device that has Android OS on it
REQ 38	3	The android application shall provide user with neat and easy to use interface

On-Screen Appearance Requirements

Identifier	PW	Requirement
REQ 39	3	Based on the prediction made by the application using the user's input, the application shall display the relevant information in an ergonomic way.
REQ 40	3	The application website shall provide a Google Maps interface which will display a large size map with pan and zoom functionality.
REQ 41	1	The application website shall provide a key along the side of the map to understand the color code seen on the roads in the map.
REQ 42	2	The application website shall user different color markers for different

		congestion levels (0, 1, 2, 3).
REQ 43	1	The application shall provide a drop down menu for all the different features that it provides and for fields that requires the user to make a choice (Ex: time, day of week etc.)
REQ 44	1	The application shall use appropriate icons to represent traffic alerts. For example: Cop Badge for Speed Traps, Car Accident for Accidents, Restaurants icon for Restaurants.
REQ 45	3	The system shall have a concise and simple mobile app GUI similar to the website version of the traffic monitoring system
REQ 46	1	The website application and the Mobile application should have buttons that allow users to toggle the data points on the map layout

Functional Requirement Specification

Stakeholders

- Users
- Administrators

Actors and Goals

- User:
 - Initiating Actor
 - The user will be accessing the application on their web client to obtain traffic predictions and also navigation between a start and end point. The application will also be providing other features that will go hand in hand with the traffic predictions
 - The user will also be using the android application to use some of the features we have presented in the android application.
- Administrator:
 - Initiating Actor
 - The administrator's primary role is to ensure the traffic data, restaurant data and parking lot data collection modules are working smoothly. The administrator must also ensure the webserver is in sync with the database (display the most up-to date traffic information).
- Database:
 - Participating Actor
 - The database will serve as a central repository for all the traffic data, parking lot data, Parking Lot data written by the backend script. It will also store the user account information with their password and username for authentication.
- Mapping Service: Google Maps
 - Participating Actor
 - The goal of the mapping service is to plot the output of the traffic prediction algorithm on the map for the user to display. It will have different colors to depict different congestion levels.
- Traffic Service: Waze
 - Participating Actor
 - The goal of the traffic service is to provide data to our application to store into the database. The traffic service will have the data available and the backend script will be grabbing the data from the API.
- Restaurant Service: Yelp
 - Participating Actor
 - The goal of the restaurant service is to provide data to our application to store into the

database. The restaurant service will have the data available and the backend script will be grabbing the data from the API.

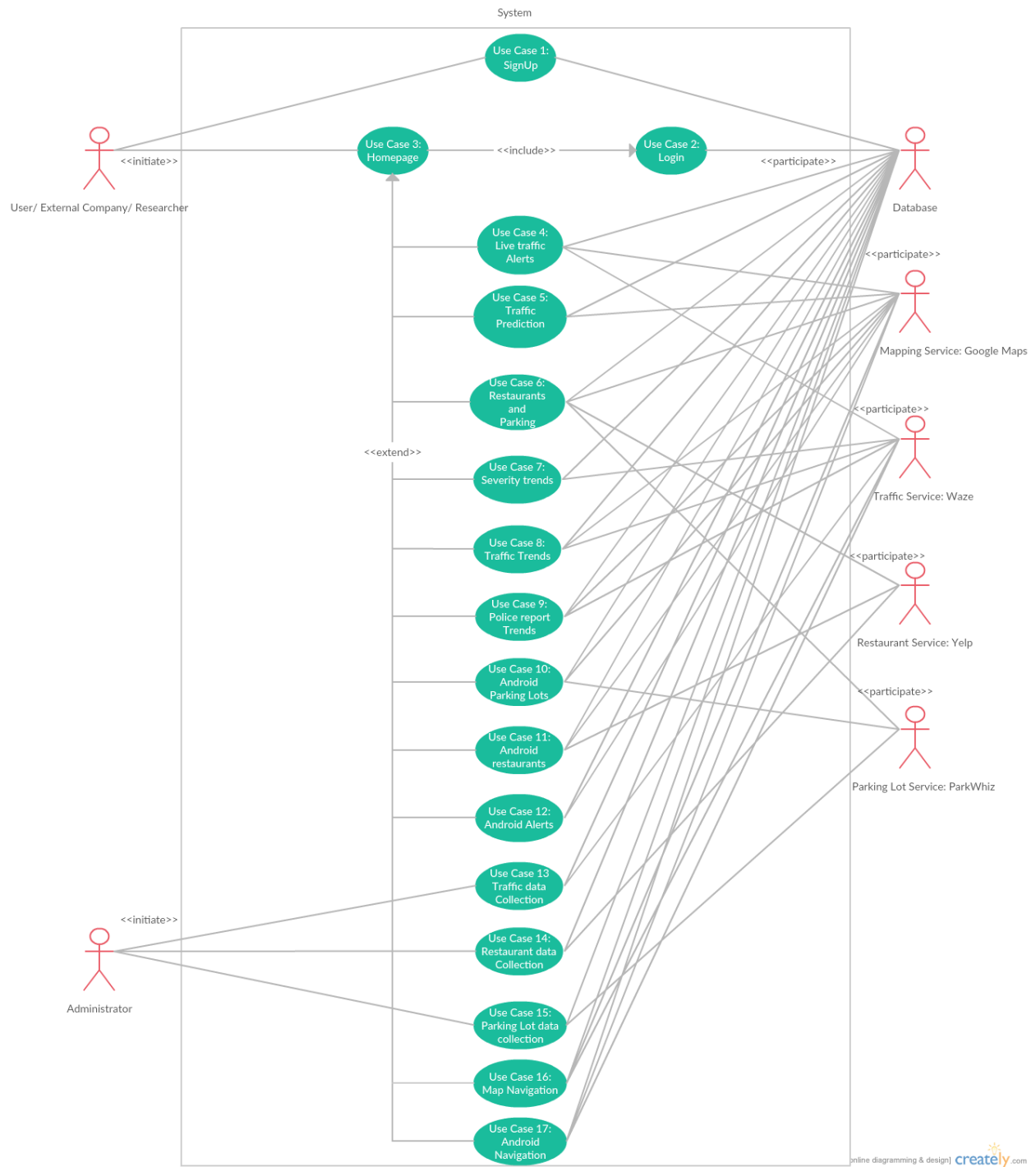
- ParkingLot Service: ParkWiz
 - Participating Actor
 - The goal of the parking lot service is to provide data to our application to store into the database. The parking lot service will have the data available and the backend script will be grabbing the data from the API.
- External Companies and researchers
 - Initiating Actor
 - A researchers goal would be similar to a user's goal which is to view trends in different Zipcodes for severity trends. They can also view total number of accident reports as well as Police reports reported for that zip code.

Use Case Casual Description

Use Case #	Name	Description	Requirements
UC1	SignUp	User should be able to signup/register to use the web application	REQ1, REQ27
UC2	Login	User should be able to securely log into the system	REQ2, REQ27, REQ33, REQ34
UC3	HomePage	Query for Traffic Congestion for any NY/NJ ZipCode and obtain different severity level markers	REQ3, REQ4, REQ35, REQ40
UC4	LiveTraffic Alerts	User should be able to view last updated traffic accidents, construction and police traps	REQ5, REQ6, REQ7, REQ40, REQ41, REQ33, REQ36
UC5	TrafficPrediction	User should be able to query for traffic congestion for a particular day and hour. Output will present congestion with predicted delay in Sec	REQ8, REQ31, REQ33, REQ39, REQ40, REQ41, REQ42, REQ43

UC6	Restaurants and Parking	Users should be able to plot restaurants and parking lots on the map centered on a zip code (NY/NJ)	REQ10, REQ11, REQ33, REQ36, REQ40, REQ44, REQ46
UC7	Severity Trends	User should be able to trend average severity for a specific Zipcode. Output is hour trend for the past week.	REQ12, REQ13, REQ14, REQ30, REQ36, REQ41, REQ42
UC8	Accidents Trends	User should be able to trend total accident counts for a specific Zipcode. Output is hourly trend for the past week.	REQ12, REQ30, REQ36, REQ41, REQ42
UC9	Police Report Trends	User should be able to trend police report for a specific Zipcode. Output is hour trend for the past week.	REQ4, REQ13, REQ30, REQ36, REQ41,
UC10	Android: Parking Lots	User should be able to plot parking lots on android app	REQ15, REQ37, REQ38, REQ45, REQ46
UC11	Android: Restaurants	User should be able to plot restaurant on android app	REQ16, REQ37, REQ38, REQ45, REQ46
UC12	Android: Alerts	User should be able to plot traffic alerts on android app	REQ17, REQ18, REQ37, REQ38. REQ45, REQ46
UC13	Traffic DataCollection	Collect traffic data from Waze API and store to database	REQ20, REQ23, REQ24, REQ29
UC 14	Restaurant Data Collection	Collect YelpAPI data for a zip code and store it to database	REQ21, REQ24, REQ28, REQ36
UC 15	Parking Lot Data Collection	Collect ParkWiz API data for a zip code and store it to database	REQ22, REQ24, REQ28, REQ36
UC 16	Map Navigation	Obtain a navigation route for a starting point to an ending point on the Web Application	REQ9, REQ40, REQ23
UC 17	Android Navigation	Obtain a navigation route for a starting point to an ending point on the Android Application	REQ19, REQ37, REQ38, REQ46

Use Cases Diagram



Traceability Matrix

Requirement	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
REQ-1	3	X								
REQ-2	3		X							
REQ-3	4			X						
REQ-4	5			X						X
REQ-5	4				X					
REQ-6	4				X					
REQ-7	4				X					
REQ-8	5					X				
REQ-9	4									
REQ-10	3						X			
REQ-11	3						X			
REQ-12	2							X	X	
REQ-13	2							X		X
REQ-14	2							X		
REQ-15	3									
REQ-16	3									
REQ-17	4									
REQ-18	4									
REQ-19	4									

REQ-20	5									
REQ-21	5									
REQ-22	5									
REQ-23	5									
REQ-24	5									
REQ-25	5									
REQ-26	5	X	X							
REQ-27	3									
REQ-28	2									
REQ-29	2									
REQ-30	3							X	X	X
REQ-31	5					X				
REQ-32	1									
REQ-33	1		X		X	X	X			
REQ-34	1		X							
REQ-35	3			X						
REQ-36	4						X	X	X	X
REQ-37	2									
REQ-38	3									
REQ-39	3					X				
REQ-40	3			X	X	X	X			
REQ-41	1					X		X	X	X
REQ-42	2					X		X	X	
REQ-43	1					X				

REQ-44	1				X		X			
REQ-45	3									
REQ-46	1						X			
Weight		8	10	15	17	21	16	16	12	15

Traceability Matrix Continued...

Requirement	PW	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17
REQ-1	3								
REQ-2	3								
REQ-3	4								
REQ-4	5								
REQ-5	4								
REQ-6	4								
REQ-7	4								
REQ-8	5								
REQ-9	4							X	
REQ-10	3								
REQ-11	3								
REQ-12	2								
REQ-13	2								
REQ-14	2								
REQ-15	3	X							
REQ-16	3		X						

REQ-17	4			X					
REQ-18	4			X					
REQ-19	4								X
REQ-20	5				X				
REQ-21	5					X			
REQ-22	5						X		
REQ-23	5				X			X	
REQ-24	5				X	X	X		
REQ-25	5								
REQ-26	5								
REQ-27	3								
REQ-28	2					X	X		
REQ-29	2				X				
REQ-30	3								
REQ-31	5								
REQ-32	1								
REQ-33	1								
REQ-34	1								
REQ-35	3								
REQ-36	4					X	X		
REQ-37	2	X	X	X					X
REQ-38	3	X	X	X					X
REQ-39	3								
REQ-40	3							X	

REQ-41	1								
REQ-42	2								
REQ-43	1								
REQ-44	1								
REQ-45	3	X	X	X					
REQ-46	1	X	X	X					X
Weight		12	12	17	17	16	16	12	10

Fully Dressed Use Cases

Use Case UC-3: Get Traffic Congestion by Zip Code

Related Requirement: REQ 3,4,35,40

Initiating Actor: User/ Researcher

Actor's Goal: Get Severity info for a specific Zip Code

Participating Actors: Database, Mapping Service, Traffic Service

Preconditions: User has logged into the application and is a valid user

Postconditions: Application has displayed traffic predictions

Success End Condition: If Zip code is part of NY/NJ.

Flow of Events for Main Success Scenario:

1. User launches web browser and uses website address
2. System will display log in page
3. User will enter username and password
4. System will validate user against database stored accounts
5. Home Page will have text box to input Zipcode and drop down box to select day of the week. User would enter zip code and hit "Run Query"
6. Webpage will reload and user needs to select "Severity 0", "Severity 1", "Severity 2" and "Severity 3"
7. The corresponding markers for each severity level will be displayed on the map.

Flow of Events for Alternate Scenarios

1. User enters invalid information (ZipCode is not NY/NJ)
2. Requested data is unavailable
 - a. System prompts user to re-enter zip code
 - a. System will display that there is no severity information reported for a specific Zipcode

Use Case UC-4: Live Traffic Alerts

Related Requirements: REQ5, REQ6, REQ7, REQ40, REQ41, REQ33, REQ36

Initiating Actor: User

Actor's Goal: To display latest traffic alerts such as Accidents/Construction/Police alerts

Participating Actors: Database, Mapping Service, Traffic Service

Preconditions: User had logged into the application

Postconditions: Application has displayed all the traffic alerts on the webpage

Success End: Zipcode is valid area for NY/NJ. Accidents and other incidents have been reported for a zipcode

Failed End: Accidents and Police incidents are not reported for a zipcode and has 0 returned marker points. Also if ZipCode is invalid.

Flow of Events for Main Success Scenario:

1. User launches the "Traffic Alerts" webpage
2. System loads page with latest traffic alerts and presents buttons "Accidents", "Construction", "Police Alerts".
3. User clicks on any one or all of these buttons
4. Mapping service will display the appropriate markers on the map

Flow of Events for Alternate Scenarios:

- 1) User enters invalid information (ZipCode is not NY/NJ)
 - a. System prompts user to re-enter zip code
- 2) Requested data is unavailable
 - a. System will display that there is no severity information reported for a specific Zipcode

Use Case UC-5: Traffic Prediction

Related Requirements: REQ46REQ8, REQ31, REQ33, REQ39, REQ40, REQ41, REQ42, REQ43

Initiating Actor: User

Actor's Goal: To obtain traffic prediction for a specific day and hour

Participating Actors: Database, Mapping Service, Traffic Service

Preconditions: User has logged into the application with valid credentials

Postconditions: Application has displayed all the traffic alerts on the webpage

Success End: Zipcode is valid area for NY/NJ. Accidents and other incidents have been reported for a zipcode.

Failed End: Accidents and Police incidents are not reported for a zipcode and has 0 returned marker points. Additionally, use case can fail if zip code is invalid.

Flow of Events for Main Success Scenario:

1. User launches the "NNJ Severity" webpage
2. System loads page and give user text box to enter "Hour" and "Day of the week"
3. User clicks on "Run Query" button
4. System loads page with severity information and has buttons ready with "Severity 0", "Severity 1", "Severity 2" and "Severity 3"
5. User will click button and enters starting location and destination into the text fields
6. Mapping Service will load the navigation route on the map with the appropriate markers for severity
7. User will click on the markers and they will display the predicted traffic "Congestion" delay in seconds.

Flow of Events for Alternate Scenarios:

1. User enters invalid information (ZipCode is not NY/NJ)
 - a. System prompts user to re-enter zip code
2. Requested data is unavailable
 - a. System will display that there is no severity information reported for a specific
3. Entered Starting and Ending Destinations are not valid
 - a. System will display that there are no returned routes Zipcode

Use Case UC-6: Restaurants and Parking

Related Requirements: REQ10, REQ11, REQ33, REQ36, REQ40, REQ44,

Initiating Actor: User

Actor's Goal: To obtain restaurants and parking for a zip code

Participating Actors: Database, Mapping Service, Restaurant Service and Parking Service

Preconditions: User has logged into the application with valid credentials

Postconditions: Application has displayed all the traffic alerts on the webpage

Success End: Zip code is valid area for NY/NJ. Accidents and other incidents have been reported for a zipcode

Failed End: Accidents and Police incidents are not reported for a zip code and has 0 returned marker points. Additionally, use case can fail if zip code is invalid.

Flow of Events for Main Success Scenario:

1. User launches the "Restaurant and Parking" webpage
2. System loads page with zip code information
3. User will enter zipcode and click on submit button
4. Restaurant Service and Parking Service will load the data onto the map.

Flow of Events for Alternate Scenarios:

1. User enters invalid information (ZipCode is not NY/NJ)
 - a. System prompts user to re-enter zip code
2. Requested data is unavailable

Use Case UC-7: Severity Trending

Related Requirements: REQ12, REQ13, REQ14, REQ30, REQ36, REQ41, REQ42

Initiating Actor: User

Actor's Goal: To obtain severity trends for a specific zipcode

Participating Actors: Database, Traffic Service

Preconditions: User has logged into the application with valid credentials

Postconditions: Application has displayed the trends for the entered Zipcode

Success End: Average traffic severity is displayed

Failed End: Zipcode entered is invalid. Chart will show zero for all values

Flow of Events for Main Success Scenario:

1. User launches “Severity Trends” webpage
2. System loads page and give user text box to enter “ZipCode”
3. User enters ZipCode and clicks on “Run Query” button
4. System loads page with information with hourly chart trend lines for each weekday
5. User can hover over the trend lines to see the value of “Average Severity” observed by that Zipcode for any hour.

Flow of Events for Alternate Scenarios:

- 1) User enters invalid information (ZipCode is not NY/NJ)
 - a. System prompts user to re-enter zip code

Use Case UC-12: Android alerts

Related Requirements:

Initiating Actor: User

Actor’s Goal: To get the latest traffic alerts such as Accidents/Construction/Police alerts on android app

Participating Actors: Database, Mapping Service, Traffic Service

Preconditions: Database is updated

Postconditions: Android App has displayed all the traffic alerts

Success End: Zipcode is valid area for NY/NJ. Accidents and other incidents have been reported for a zip code

Failed End: Accidents and Police incidents are not reported for a zipcode and has 0 returned marker points. Also if Zip Code is invalid.

Flow of Events for Main Success Scenario:

1. User launches the android app
2. System loads page with latest traffic alerts and presents buttons “Accidents”, “Construction”, “Police Alerts”
3. User clicks on any one or all of these buttons

4. Mapping service will display the appropriate markers on the map

Flow of Events for Alternate Scenarios:

- 1) User enters invalid information (Zip code is not NY/NJ)
 - a. System prompts user to re-enter zip code
- 2) Requested data is unavailable
 - a. System will display that there is no severity information reported for a specific Zip code

Use Case UC-13,14,15: Data Collection

Related Requirements: REQ20, REQ23, REQ24, REQ29, REQ28, REQ21, REQ22

Initiating Actor: Admin

Actor's Goal: To update the database with latest traffic alerts, congestion, parking lot and restaurant data.

Participating Actors: Database, Traffic Service, Restaurant Service and Parking Lot Service.

Preconditions: Admin is logged in.

Postconditions: Database tables are populated with updated traffic data, restaurant data and parking lot data.

Success End: Data collection script completed successfully.

Failed End: There is no updated data returned from Traffic Service, Restaurant Service and Parking Lot Service.

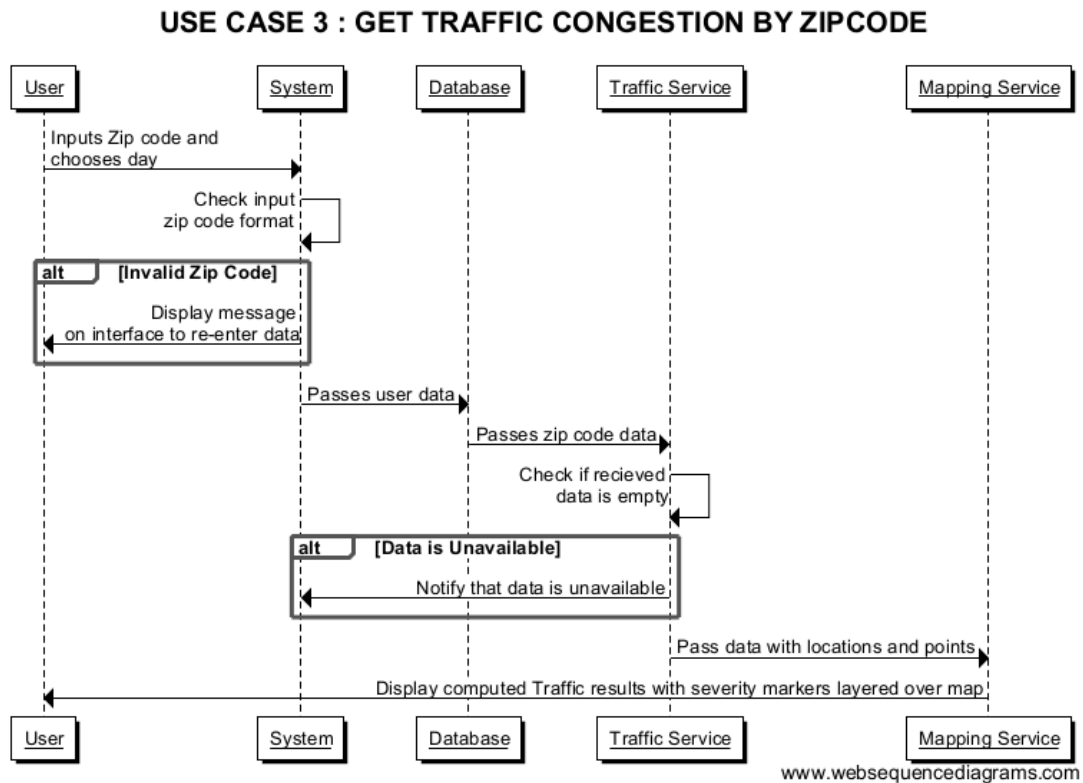
Flow of Events for Main Success Scenario:

1. Admin logs in.
2. He runs the data collection script.
3. Traffic Service, Restaurant Service and Parking Lot Service sends the request to get updated data.
4. Returned JSON data is parsed and stored to database.

Flow of Events for Alternate Scenarios:

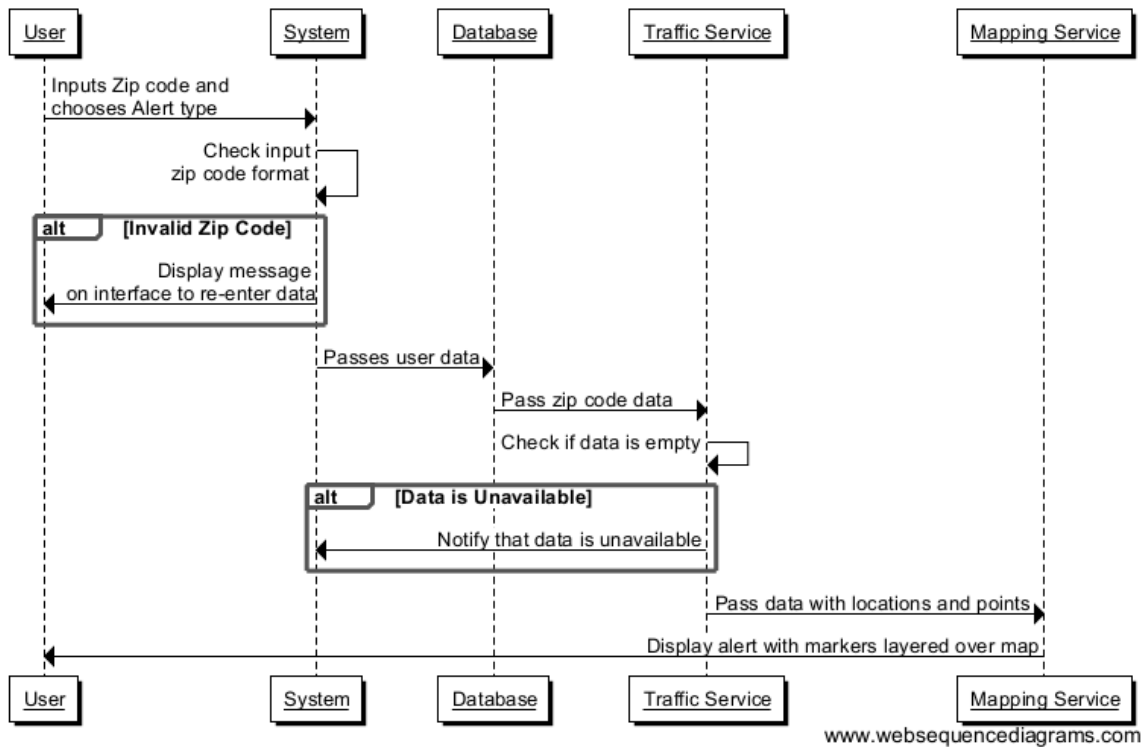
- 1) There is no updated data for entered zip code.
- 2) Requested data is unavailable by third party servers (Parkwiz API, Yelp API and Waze API)

System Sequence Diagrams



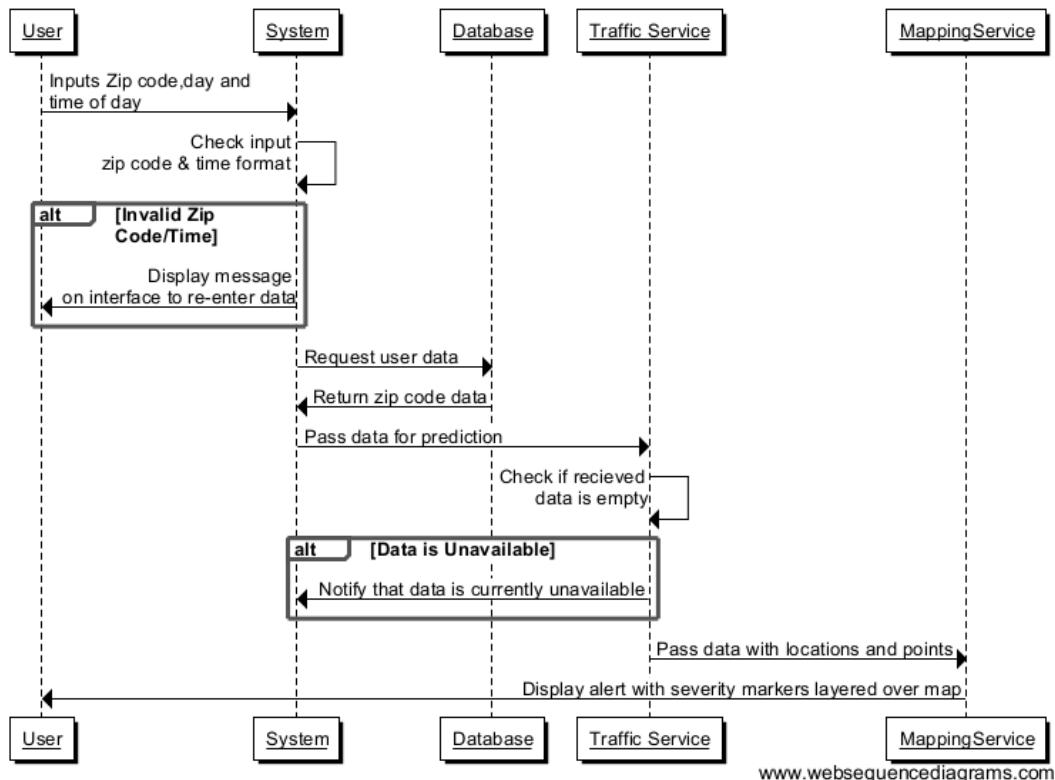
The user will enter the zip code they want to see congestion data for. The database will send the relevant zip code data to the Traffic Service, which will send the congestion information to the Mapping Service. The Mapping Service then displays the information.

USE CASE 4 : LIVE TRAFFIC ALERTS



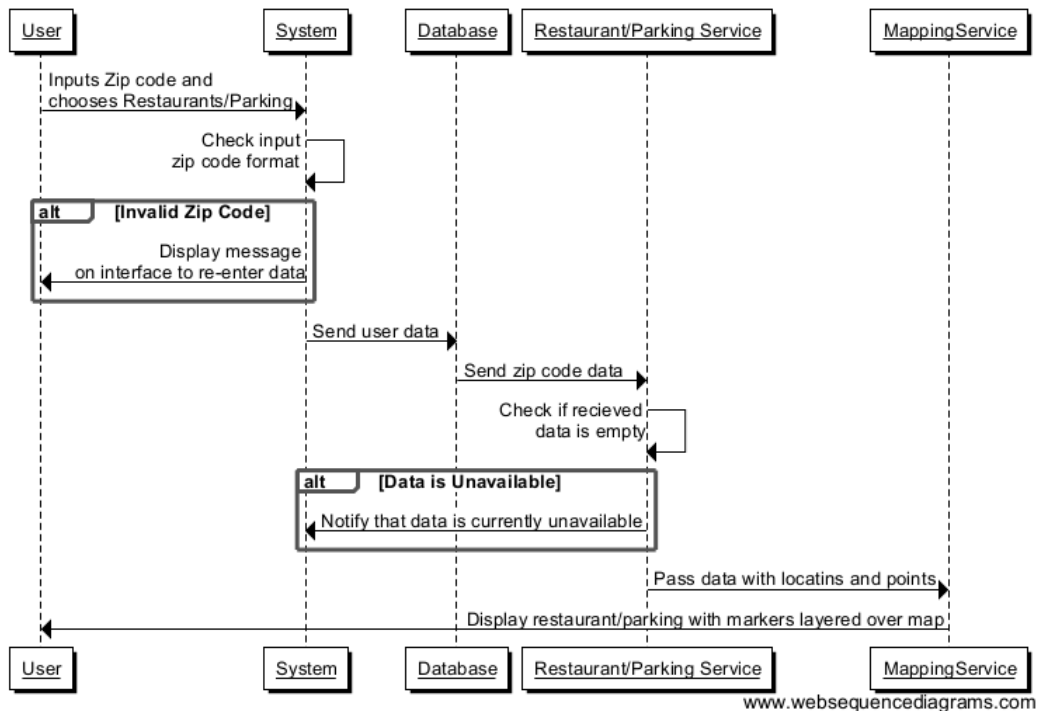
The user will enter the zip code they want to receive traffic alerts for. The database will send the relevant zip code data to the Traffic Service, which will send the traffic alert information to the Mapping Service. The Mapping Service then displays the information.

USE CASE 5: TRAFFIC PREDICTION



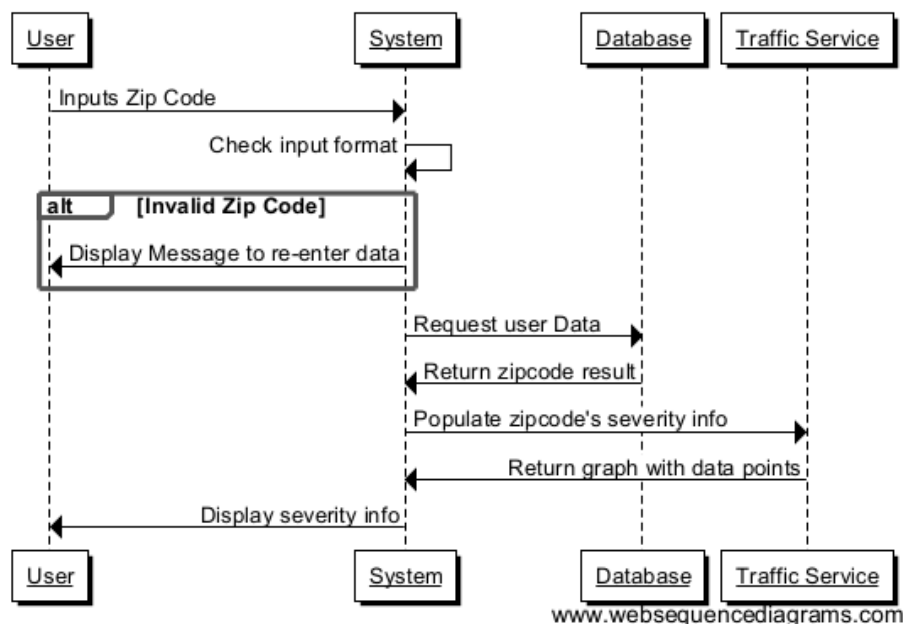
The user will enter the zip code they want to see traffic prediction for. The database will send the relevant zip code data to the Traffic Service, which will send the traffic prediction information to the Mapping Service. The Mapping Service then displays the information.

USE CASE 6: RESTAURANT AND PARKING



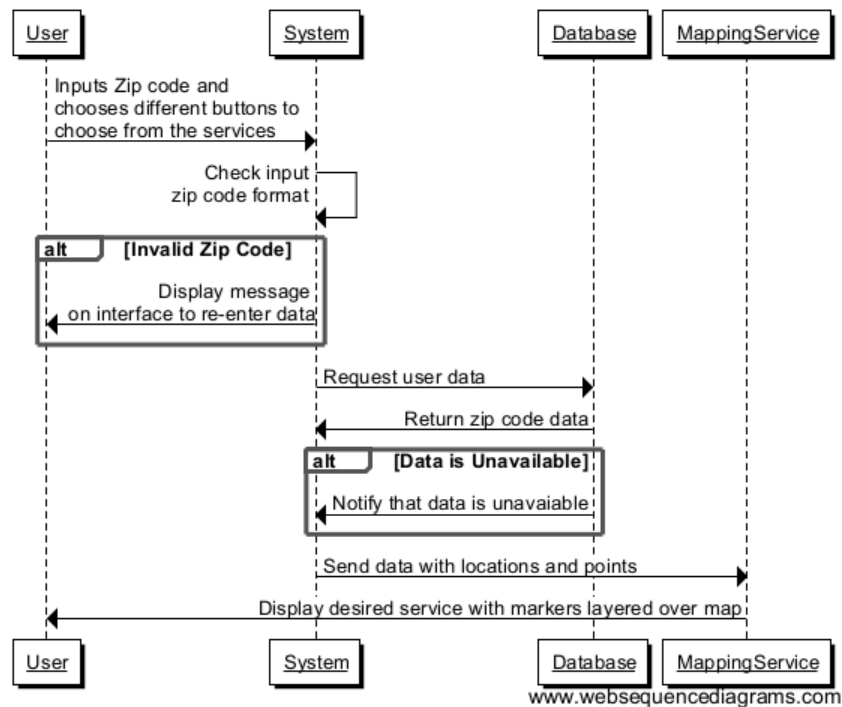
The user will enter the zip code they want to see restaurant and parking information for. The database will send the relevant zip code data to the Traffic Service, which will send the restaurant and parking information to the Mapping Service. The Mapping Service then displays the information.

USE CASE 7: SEVERITY TRENDING



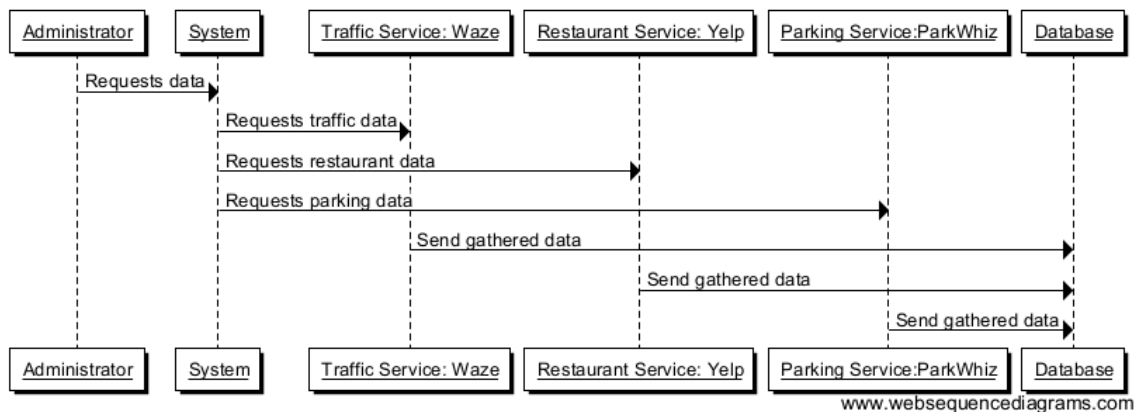
The user will enter the zip code they want to see severity trends for. The database will send the relevant zip code data to the Traffic Service, which will send the severity trends to the Mapping Service. The Mapping Service then displays the information.

USE CASE 12: ANDROID ALERTS



The user, on their android device, will enter the zip code they want to receive traffic alerts for. The database will send the relevant zip code data to the Traffic Service, which will send the traffic alert information to the Mapping Service. The Mapping Service then displays the information to the android device.

USE CASE 13,14,15 : DATA COLLECTION



The administrator will request that new traffic, restaurant and parking data be gathered. The traffic service, restaurant service, and parking service will all gather the most recent data and then store it in the database.

User Effort Estimation

Actor Type	Description of 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 the system through a graphical user interface	3

Actor Name	Description of characteristics	Complexity	Weight
User	The actor is a system our system interacts with	Complex	3
Mobile User	The actor is a person interacting with the system through text based interface	Complex	3
Researcher	The actor is a person interacting with the system through a graphical user interface	Complex	3
Administrator	Administrator interacts with the system through command line or text based interface	Average	2
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
Restaurant Service	The Restaurant 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
Parking Lot Service	The Parking Lot Service is a system that our system interacts with.	Simple	1
Application	The Application is a system interacting with our system.	Simple	1

$$UAW = [3 * \text{Complex}] + [1 * \text{Average}] + [6 * \text{Simple}] = [9] + [2] + [6] = 17$$

Use Case Category	How to recognize	Weight
-------------------	------------------	--------

Simple	Number of steps for success ≤ 3 . One participating actor	5
Average	Number of steps for success < 7 . Two participating actors	10
Complex	Number of steps for success ≥ 7 . Three or more participating actors	15

Use Case #	Name	Number of steps and Actors	Category	Weight
UC1	SignUp	Steps: 5, Actor: 1	Average	10
UC2	Login	Steps: 3, Actor: 1	Simple	5
UC3	HomePage	Steps: 7, Actor: 3	Complex	15
UC4	LiveTraffic Alerts	Steps: 3, Actor: 3	Average	10
UC5	TrafficPrediction	Steps: 9, Actor: 3	Complex	15
UC6	Restaurants and Parking	Steps: 4, Actor: 2	Average	10
UC7	Severity Trends	Steps: 2, Actor: 2	Simple	5
UC8	Accidents Trends	Steps: 2, Actor: 2	Simple	5
UC9	Police Report Trends	Steps: 2, Actor: 2	Simple	5
UC10	Android: Parking Lots	Steps: 2, Actor: 2	Simple	5
UC11	Android: Restaurants	Steps: 2, Actor: 2	Simple	5
UC12	Android: Alerts	Steps: 2, Actor: 2	Simple	5
UC13	Traffic Data Collection	Steps: 3, Actor: 2	Average	10
UC 14	Restaurant Data Collection	Steps: 2, Actor: 2	Simple	5
UC 15	Parking Lot Data Collection	Steps: 2, Actor: 2	Simple	5
UC 16	Map Navigation	Steps: 3, Actor: 1	Simple	5
UC 17	Android Navigation	Steps: 3, Actor: 1	Simple	5

$$UUCW = [2 * \text{Complex}] + [4 * \text{Average}] + [11 * \text{Simple}] = [30] + [40] + [55] = 125$$

$$UUCP = UAW + UUCW = 125 + 17 = 142$$

Technical Factor	Description	Weight
T1	Distributed system (running on multiple machines)	2
T2	Performance objectives (are response time and throughput performance critical?)	1
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install (are automated conversion and installation included in the syststem)	0.5
T7	Easy to use (including operations such as backup, startup, and recovery)	0.5
T8	Portable	2
T9	Easy to change (to add new features or modify existing ones)	1
T1	0 Concurrent use (by multiple users)	1
T1	1 Special security features	1
T1	2 Provides direct access for third parties (the system will be used from multiple sites in different organizations)	1
T1	3 Special user training facilities are required	1

Technical Factor	Technical Description	Weight	Perceived Complexity	Calculated Factor
T1	Distributed, Web-based system	2	3	6
T2	Users expect good performance but nothing exceptional	1	4	4
T3	End-user expects efficiency but there are no exceptional demands	1	4	4
T4	Internal processing is relatively simple	1	3	3
T5	No requirement for reusability	1	2	2
T6	Ease of install is moderately important (will probably be installed by technician)	0.5	1	0.5
T7	Ease of use is very important	0,5	4	2
T8	No portability concerns beyond a desire to keep database vendor options open	2	1	2
T9	Easy to change minimally required	1	2	2
T10	Concurrent use is required	1	3	3
T11	Security is a significant concer	1	2	2
T12	No direct access for third parties	1	0	0
T13	No unique training needs	1	0	0

Total TCF = 30.5. TCF = 0.6 + (0.01 *30.5) = 0.905

Environmental Factor	Description	Weight
E1	Familiar with the development process	1.5
E2	Application problem experience	0.5
E3	Paradigm experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time staff	-1
E8	Difficult programming language	-1

Environmental Factor	Description	Weight	Complexity	Calculated Factor
E1	Beginner familiarity with UML-based development	1.5	3	4.5
E2	Some application problem experience	0.5	2	1
E3	Knowledge object oriented approach	1	4	4
E4	Moderate lead analyst	0.5	4	2
E5	Motivated, but team members slacking	1	2	2
E6	Stable requirements expected	2	3	6
E7	No part-time staff	-1	3	-3
E8	Programming language is of average difficulty	-1	3	-3

Environmental Factor = 13.5

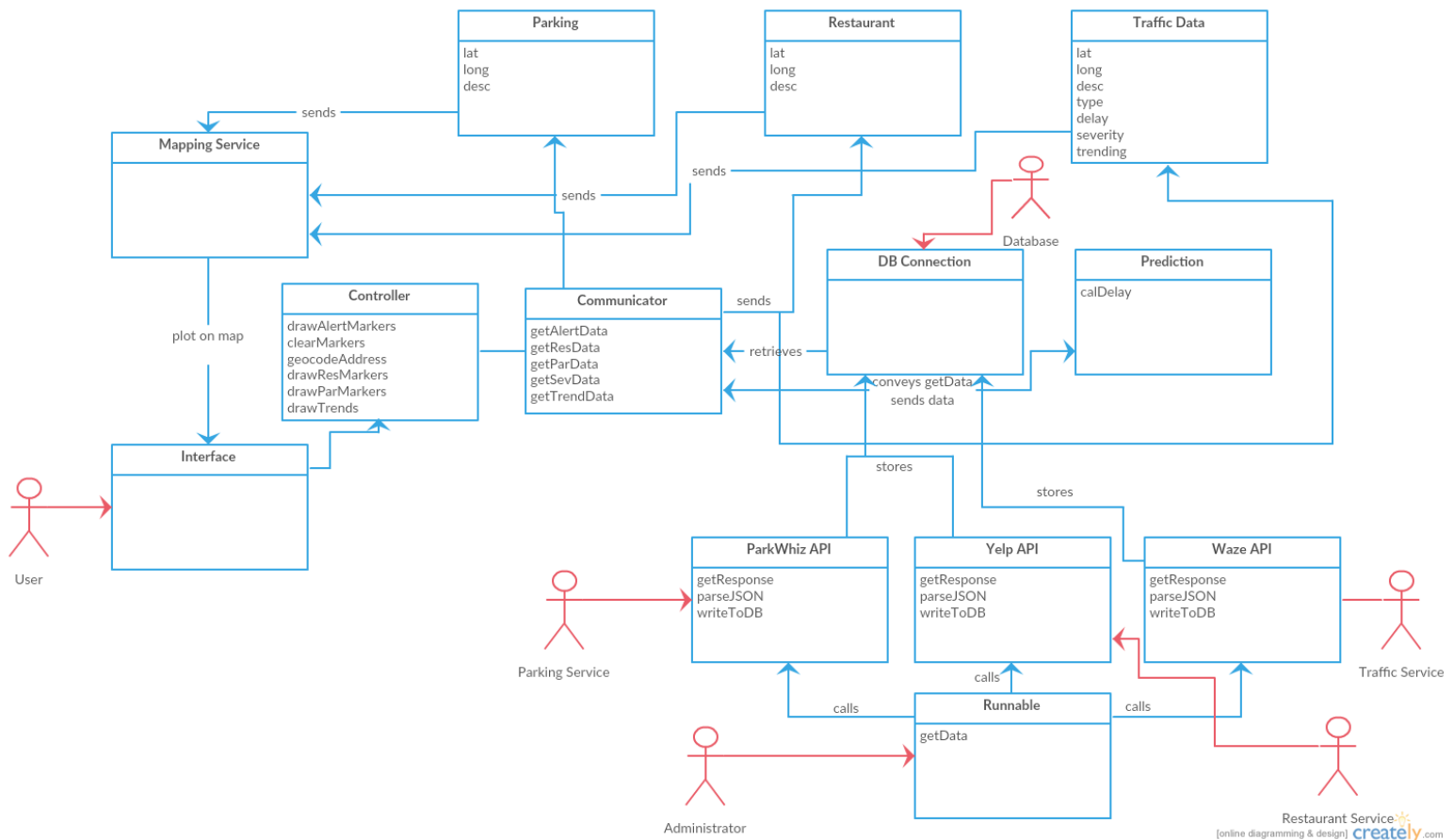
$ECF = [C1] + [C2 * 13.5] = 1.4 + [-0.03 * 13.5] = 0.995$

$UCP = UUCP * TCF * ECF$

$UCP = 142 * 0.905 * 0.995 = 127$

The project has 127 Use Case Points

Domain Analysis



The Domain Analysis figure above shows the analysis of related software systems in a domain to find their common and variable parts. From the above diagram, it is clear that the User has access only to the Interface. To keep the User experience easy and personalized, the User interface is made basic and has a login to prompt. The login details of the user are verified from the database and his home zip code is displayed in the homepage. A google map instance is also created at this point which is zoomed in at the user's home zip code area. Based on the input from the User, the Interface calls Controller. All of the functionality of the application happens through this specified Controller Class. There are action methods in controller class which calls the appropriate methods from Communicator class. In our project Communicator class is talking to database and getting the required data. All the code for Communicator class is in PHP. Once the communicator gets the required data from database it then passes it to respective helper class. Construction, Restaurant, Parking, Event, etc., All these helper class have an array of latitude, longitude and description arrays. Now we are passing these data to our Mapping service which is GOOGLE API and then the data is plotted on the user interface of map instance. If the user is selecting the Navigation interface then he will enter start location and end location. Our controller class will trigger the Mapping service API and will be able to see navigation on google map instance. If the user selects to query the severity data, they need to enter the day and time of day. Controller class will call the prediction class and it will return the result to user interface.

The Administrator has control of the other features that can receive data from Web Services. The administrator will monitor the data that comes from these Services. These scripts gather and parse the data received from the Web Services. The Administrator can use these scripts as per the requirement of the application and then pass required information to be stored in the database which in turn is used by the main web application. To provide a personalized touch to the application, a login feature has been incorporated. This functionality allows the application to access the database at the time of login and prompt or predict the user's general choice location and time to display information based on the users data stored in the database. The application sends and receives data from the database where the User details are stored. Also, a few new features have been added keeping in mind a user who is travelling to an unknown location. The user will be able to find parking lots which are accessible to the general public based on the data that we receive from the Web service which is parsed through the ParkingWhizAPI Class and stored in our Database. The application will also provide restaurant locations at the destination location. The data of the restaurants which are queried from the Web Services will also be parsed through the YELPAPI Class and stored in the database. The data for the parking lots and the restaurants will be fed into the database fewer times than the traffic data because it is less prone to frequent changes.

TrafficModule class (WAZE API) will get the alerts and congestion data. Alerts data will consist of construction, events, road jams, police data,etc.,

Concept Definition

Responsibility Description	Type	Concept Name
Display the alerts, severity trends, accident charts, parking lot and restaurant data.		Interface
Coordinate activity and delegate work originated from the user query.	D	Controller
Calls the MappingService: Google API which will display the navigation on the google map instance of the user interface.	D	Controller
Calls the Prediction class to fetch the severity data based on user queried day and time	D	Controller
plot the alert markers on the map based on user selected	D	Controller

alert type		
Gets the data for user queried alert type and pass it to respective helper entity	D	Communicator
Gets the severity trend data from database and pass it to the prediction	D	Communicator
Gets the accident and police data reports and pass it to Trending charts	D	Communicator
Manage interactions with the database		DBConnection
Stores the marker data and pass it to the MappingService to plot the markers on map.	D	Restaurant
Stores the marker data and pass it to the MappingService to plot the markers on map.	D	Parking
Stores the severity data for past 5 days and	K	TrafficData
Stores the accident and police data for past 5 days	K	TrafficData
Stores type of alert and respective latitude and longitude values and pass it to the MappingService to plot the markers on map.	D	TrafficData
Stores the trend data for past 5 days and pass it to the MappingService to plot the markers on map.	D	TrafficData
calculate the delay in seconds and pass it to traffic data.	D	Prediction
Plots the data on user interface	D	MappingService
gets the http response, parse the JSON data and writes to DB	D	ParkWhizAPI
gets the http response, parse the JSON data and writes to DB	D	YelpAPI

gets the http response, parse the JSON data and writes to DB	D	WazeAPI
calls the respective API to get the data	D	Runnable

Association definitions

Concept Pair	Association description	Association name
Communicator <-> DBConnection	Communicator retrieves the data from DBConnection based on user query.	Retrieves
Communicator <-> Prediction	Prediction conveys the data which it requires for predicting the delay to communicator class, the required data is then sent back to prediction from communicator.	conveysGetData, sendsData
Communicator <-> TrafficData	Communicator sends the latitude, longitude, description, type, delay, severity, trending data to traffic data entity.	sends
Communicator <-> Restaurant	Communicator sends the latitude, longitude and description data to Restaurant entity.	sends
Communicator <-> Parking	Communicator sends the latitude, longitude and description to Parking entity.	sends
Parking<-> MappingService	Parking sends the geocode object to MappingService	sends
Restaurant<-> MappingService	Restaurant sends the geocode object to MappingService	sends
TrafficData<-> MappingService	TrafficData sends the geocode object to MappingService	sends
Runnable <-> ParkWhizAPI	calls the ParkWhizAPI to get the response and parse the JSON data	calls
Runnable <-> YelpAPI	calls the YelpAPI to get the response and parse the JSON	calls

	data	
Runnable <-> WazeAPI	calls the WazeAPI to get the response and parse the JSON data	calls
WazeAPI<-> DBConnection	writes the parsed JSON response to db	stores
YelpAPI<-> DBConnection	writes the parsed JSON response to db	stores
ParkWhizAPI<-> DBConnection	writes the parsed JSON response to db	stores

Attribute Definitions

Concept	Attributes	Attribute Description
Controller	drawAlertMarker	pass the query parameter to communicator
Controller	clearMarker	pass the query parameter to communicator
Controller	geocodeAddress	pass the query parameter to communicator
Controller	drawResMarker	pass the query parameter to communicator
Controller	drawParMarker	pass the query parameter to communicator
Controller	drawTrends	pass the query parameter to communicator
Communicator	getAlertData	queries the data from database for selected parameter
Communicator	getResData	queries the data from database for selected parameter
Communicator	getParData	queries the data from database for selected parameter
Communicator	getSevData	queries the data from database for selected parameter
Communicator	getTrendData	queries the data from database for selected parameter

Parking	lat	latitude value of parking lot
Parking	lng	longitude value of parking lot
Parking	desc	name of parking lot
Restaurant	lat	latitude value of restaurant
Restaurant	lng	longitude value of restaurant
Restaurant	desc	name of restaurant
TrafficData	lat	latitude value of traffic data
TrafficData	lng	longitude value of traffic data
TrafficData	desc	description of traffic data
TrafficData	type	type of traffic data
TrafficData	delay	delay value of congestion
TrafficData	severity	severity of congestion
TrafficData	trending	trend data
Prediction	calDelay	calculates the delay
ParkWhizAPI	getResponse	make request to get the data from api
ParkWhizAPI	parseJSON	parses the JSON response
ParkWhizAPI	writeToDb	writes the parsed data to database
YelpAPI	getResponse	make request to get the data from api
YelpAPI	parseJSON	parses the JSON response
YelpAPI	writeToDb	
WazeAPI	getResponse	make request to get the data from api
WazeAPI	parseJSON	parses the JSON response
WazeAPI	writeToDb	
Runnable	getData	

Domain Traceability

<u>Use Case</u>	<u>Weight</u>	<u>Interface</u>	<u>Controller</u>	<u>Communicator</u>	<u>DBConnection</u>	<u>Prediction</u>	<u>MappingService</u>	<u>TrafficData</u>	<u>Restaurant</u>	<u>Parking</u>	<u>Runnable</u>	<u>WazeAPI</u>	<u>YelpAPI</u>	<u>ParkWhizAPI</u>
UC-3	15	x	x	x	x		x	x						
UC-4	17	x	x	x	x		x	x						
UC-5	21	x	x	x	x	x	x	x						
UC-6	16	x	x	x	x		x		x	x				
UC-7	16	x	x	x	x		x	x						
UC-12	17	x	x	x	x		x	x						
UC-13,14,15	17				x						x	x	x	x

System Operation Contracts

1) Operation: Writing into database

Precondition: Memory is allocated for different type of data in database. Runner class calls the respective API class which will request for the JSON data from Server

Post condition: Response data is parsed and the result is stored into the database.

2) Operation: Displaying alert, parking, restaurant data on Web Page

Pre-condition: Database it updated with live alerts, Parking and Restaurant Data.

Post-condition: Data is displayed on the web page once the user clicks on a particular event.

3) Operation: Displaying alert, parking and restaurant data on Android App

Pre-condition: Database it updated with live alerts, Parking and Restaurant Data.

Post-condition: Data is displayed on the android app once the user clicks on a particular event.

3) Operation: Service request from database

Precondition: There is enough data in the database provided by Traffic Service, Restaurant Service and Parking Service.

Post condition: This operation returns user with markers overlaying on the map. If there is no data in database, it returns nothing to the mapping devices showing only the blank map. That notifies user that there is no data available in the database for particular user request criteria.

4) Operation: Calculate results

Precondition: Based on the Criteria provided by the user, selected data is sent to prediction class.

Post condition: According to Bayesian prediction, calculations for traffic predictions are performed and returned to the application.

5) Operation: Get Severity data

Precondition: Based on the user search criteria get the severity data from database.

Post condition: This operation returns the severity marker and user can plot them on the graph.

Description of these markers will have severity value.

6) Operation: Navigation on Android App

Precondition: User has selected start and end location.

Post condition: Application was successfully able to provide navigation from start location to end location.

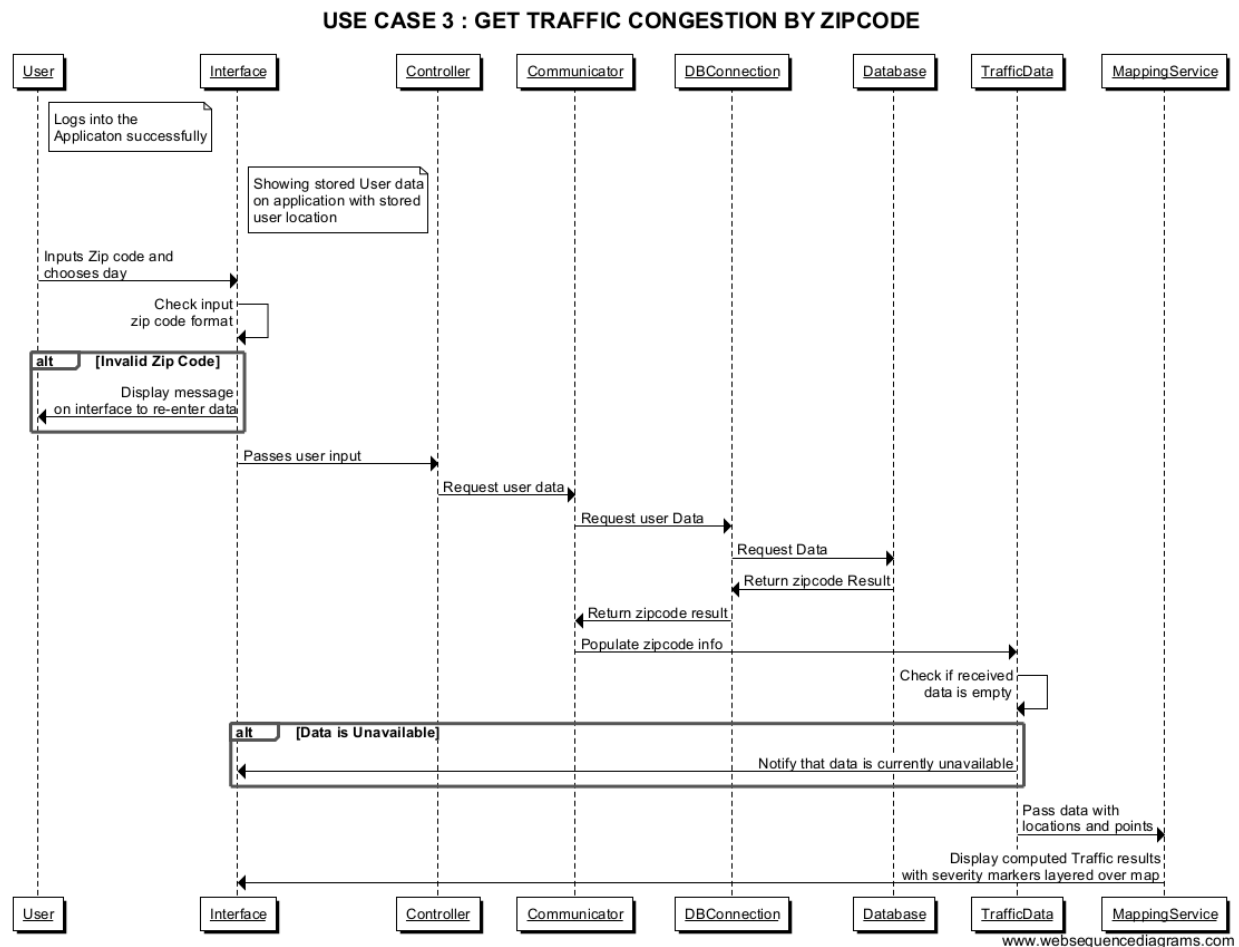
7) Operation: Navigation on Web page

Precondition: User has selected start and end location.

Post condition: Application was successfully able to provide navigation from start location to end location.

Interaction Diagrams

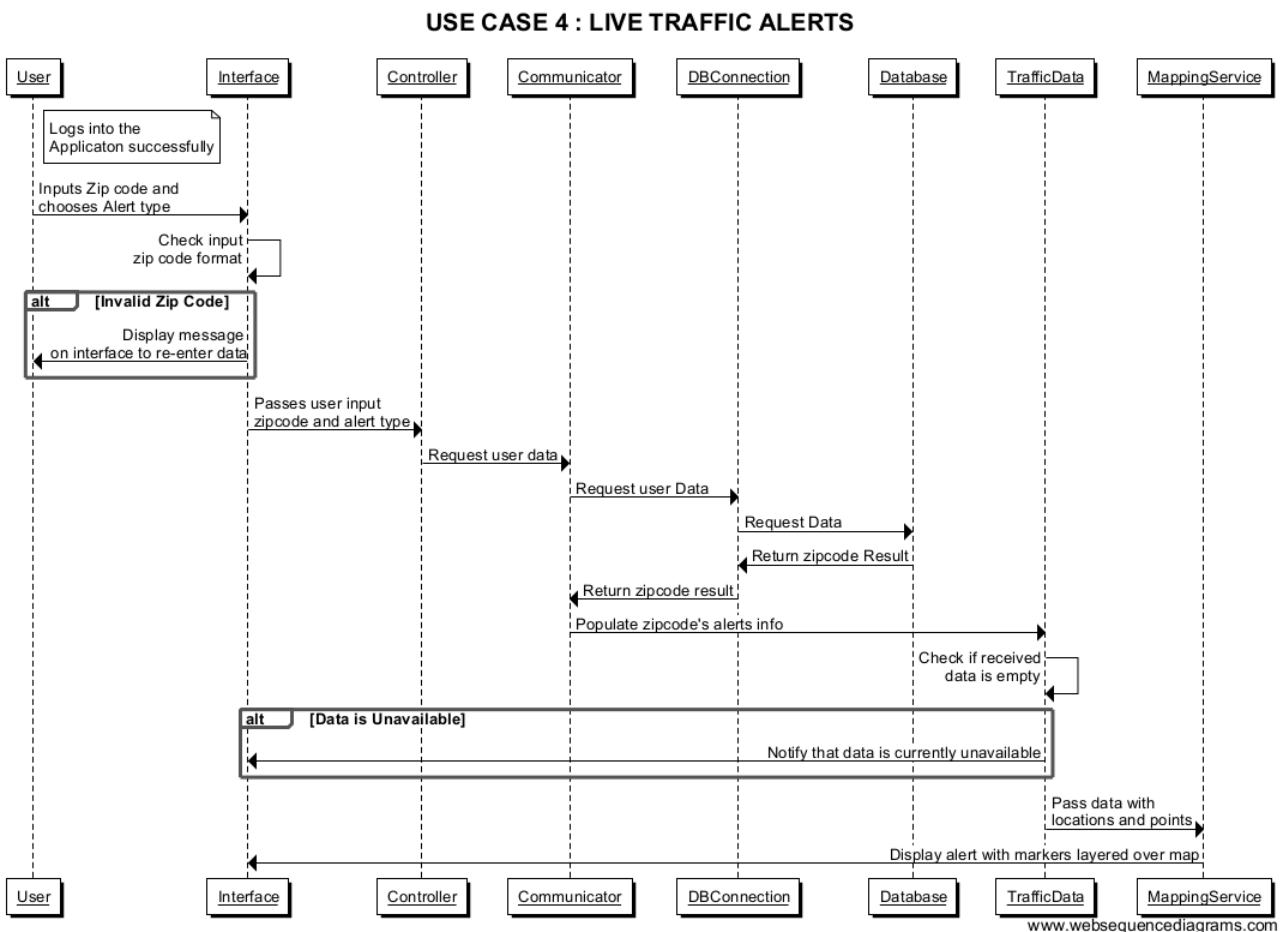
Sequence diagrams which show the interaction between the various entities included in the domain concepts help to understand how the domains go about achieving the results for the Use Cases. The interaction between the user and the various system components are shown below:



The above interaction diagram shows the interaction between the system components and the sequence of events when the user chooses to view traffic congestion in a desired zip code (**Use Case 3**). As per our implementation, once the user logs into the application, the homepage for the user is displayed and the screen shows the map of the user's stored home zip code location. The user can now input a zip code into the text field and also choose a day from the drop down menu to complete the input for this use case. Once the user submits the input, the interface calls the Controller entity. The Controller entity acts as a mediator between the server side script and the webpage. It will link the user events (button clicks) and convey the type of response needed to the Communicator. The Communicator then uses this input to connect to the Database through the DBConnection. The Communicator queries the Database to get the desired data for the specified zip code. This resultant data is passed back to the Communicator through DBConnection. The Communicator now uses the type of the user input to choose where this data has to be sent. The data in this Use Case is Traffic data and hence, the data is

passed to TrafficData. The TrafficData will hold all the relevant information that could be used to plot the User's desired options to complete this Use Case. The TrafficData will then pass this data to the MappingService(Google Maps API) which is responsible for plotting this data at the defined locations on the Map. The MappingService will drop markers on the map to indicate the Traffic Congestion present in that zipcode.

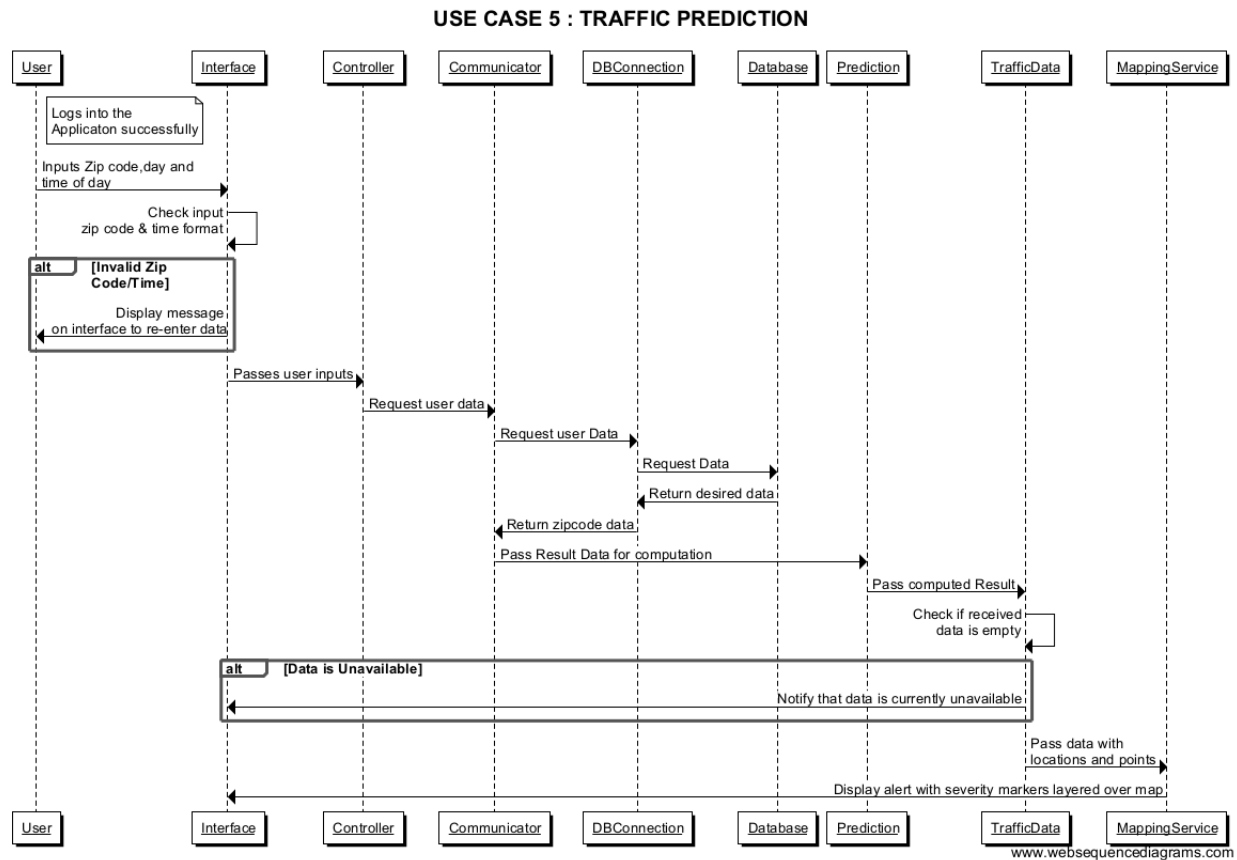
In the Alternate scenario for this Use Case, we have considered two scenarios i.e when the user enters invalid zip codes or the data for the specified zip code is not available. In the case that the entered zip code is erroneous the interface will check if the entered zip code is in the right format and if not, it will display an error to the user asking the user to re-enter the data correctly. In the case where the data for a particular zip code is not present, once the user enters the input and submits it, the interface passes this data to the Controller which acts as the mediator and passes the information to the Communicator. The Communicator queries the Database through the DBConnection for the desired data. The database will return an empty data set which will be passed to the TrafficData entity through the Communicator. On receiving the empty data, the TrafficData entity will send out a message to the user notifying the user that the severity data of that specified area is not available currently.



The above interaction diagram shows the interaction between the system components and the sequence of events when the user chooses to view live alerts in a desired zip code (**Use Case 4**). As per our implementation, the user can access this feature from the scrollbar on the left side of the page. The user can now input a zip code into the text field and also choose from the

various alerts (construction, accidents or police alerts) to complete the input for this use case. Once the user submits the input, the interface calls the Controller entity. Just as the previous use case, here the Controller entity acts as a mediator between the server side script and the webpage. It will link the user events (button clicks) and convey the type of response needed to the Communicator. The Communicator then connects to the Database through the DBConnection with this user input. The Communicator queries the Database to get the desired data for the specified zip code. This resultant data is passed back to the Communicator through DBConnection. The Communicator now uses the type of the user input to choose where this data has to be sent. The data in this Use Case is traffic related alert data and hence, the data is passed to TrafficData. The TrafficData holds data related to traffic and alerts and the data from the Communicator will hold all the relevant information that could be used to plot the User's desired options to complete this Use Case. The TrafficData will then pass this data to the MappingService (Google Maps API) which is responsible for plotting this data at the defined locations on the Map. The MappingService will drop markers on the map to indicate the Traffic Congestion present in that zipcode.

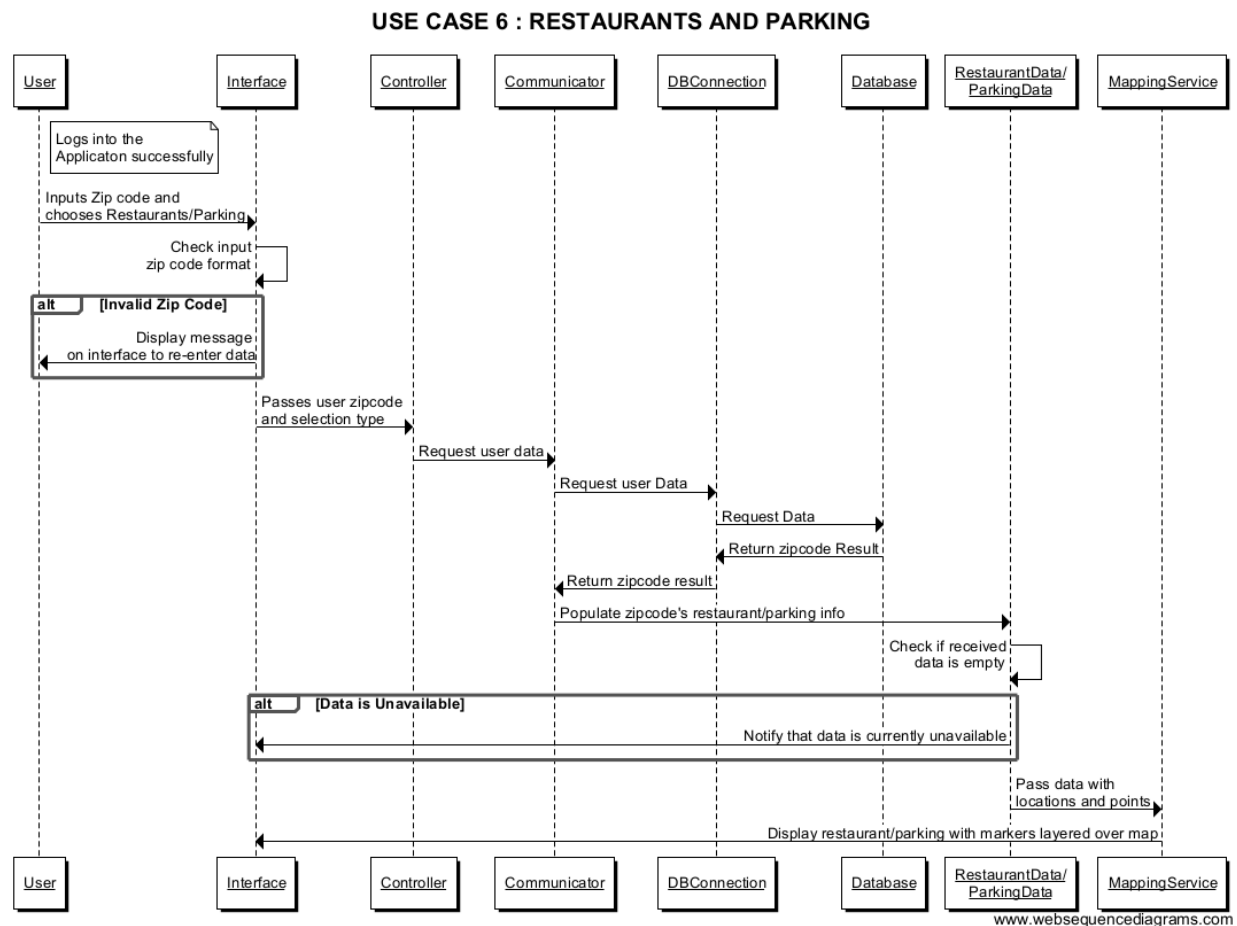
In the Alternate scenario for this Use Case, we have considered two scenarios i.e when the user enters invalid zip codes or the data for the specified zip code is not available. In the case that the entered zip code is erroneous the interface will check if the entered zip code is in the right format and if not, it will display an error to the user asking the user to re-enter the data correctly. In the case where the data for a particular zip code is not present, once the user enters the input and submits it, the interface passes this data to the Controller which acts as the mediator and passes the information to the Communicator. The Communicator queries the Database through the DBConnection for the desired data. The database will return an empty data set which will be passed to the TrafficData entity through the Communicator. On receiving the empty data, the TrafficData entity will send out a message to the user notifying the user that the alerts data of that specified area is not available currently.



The above interaction diagram shows the interaction between the system components and the sequence of events when the user chooses to view Predicted Traffic for locations in a desired zip code (**Use Case 6**). As per our implementation, the user can access this feature from the scrollbar on the left side of the page. The user can now input a zip code into the text field, choose the day for the desired traffic intensity and also input a time of the day to complete the input for this use case. Once the user submits the input, the interface calls the Controller entity. The Controller entity conveys the type of response needed to the Communicator. The Communicator then connects to the Database through the DBConnection with this user input. The Communicator queries the Database to get the desired data for the specified zip code. This resultant data is passed back to the Communicator through DBConnection. The Communicator now uses the type of the user input to choose where this data has to be sent. The data in this Use Case is traffic related alert data and hence, the data is passed to TrafficData. The TrafficData holds data related to traffic severity and this Use Case has a result in the form of delays that can be seen on the severity markers on the map. The TrafficData will then pass this data to the MappingService (Google Maps API) which is responsible for plotting this data at the defined locations on the Map. The MappingService will drop markers on the map to indicate the Traffic Congestion present in that zipcode.

In the Alternate scenario for this Use Case, we have considered two scenarios i.e when the user enters invalid zip codes or the data for the specified zip code is not available. In the case that the entered zip code or the time format is erroneous the interface will check if the entered zip code and time is in the right format and if not, it will display an error to the user asking the user to re-enter the data correctly. In the case where the data for a particular zip code is not

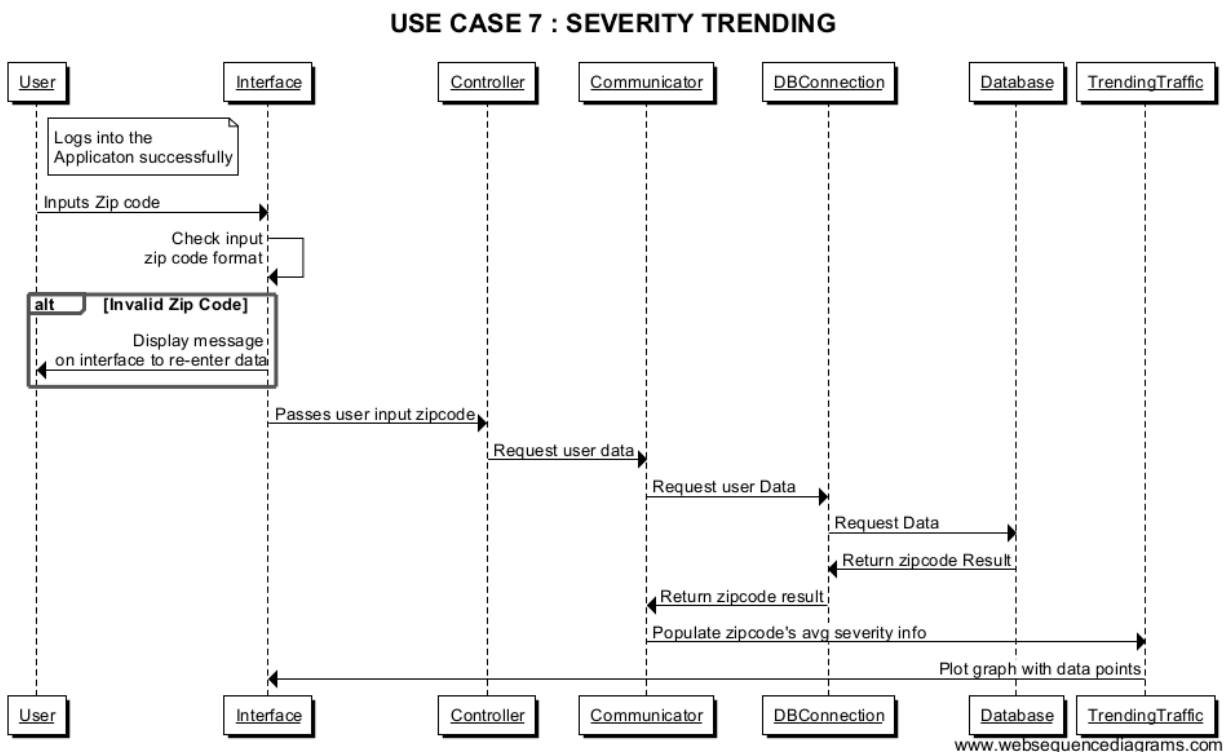
present, once the user enters the input and submits it, the interface passes this data to the Controller which acts as the mediator and passes the information to the Communicator. The Communicator queries the Database through the DBConnection for the desired data. The database will return an empty data set which will be passed to the TrafficData entity through the Communicator. On receiving the empty data, the TrafficData entity will send out a message to the user notifying the user that the alerts data of that specified area is not available currently.



The above interaction diagram shows the interaction between the system components and the sequence of events when the user chooses to view Restaurant or Parking Lot locations in a desired zip code (**Use Case 6**). As per our implementation, the user can access this feature from the scrollbar on the left side of the page. The user can now input a zip code into the text field and also choose from the various alerts (construction, accidents or police alerts) to complete the input for this use case. Once the user submits the input, the interface calls the Controller entity. Just as the previous use case, here the Controller entity acts as a mediator between the server side script and the webpage. It will link the user events (button clicks) and convey the type of response needed to the Communicator. The Communicator then connects to the

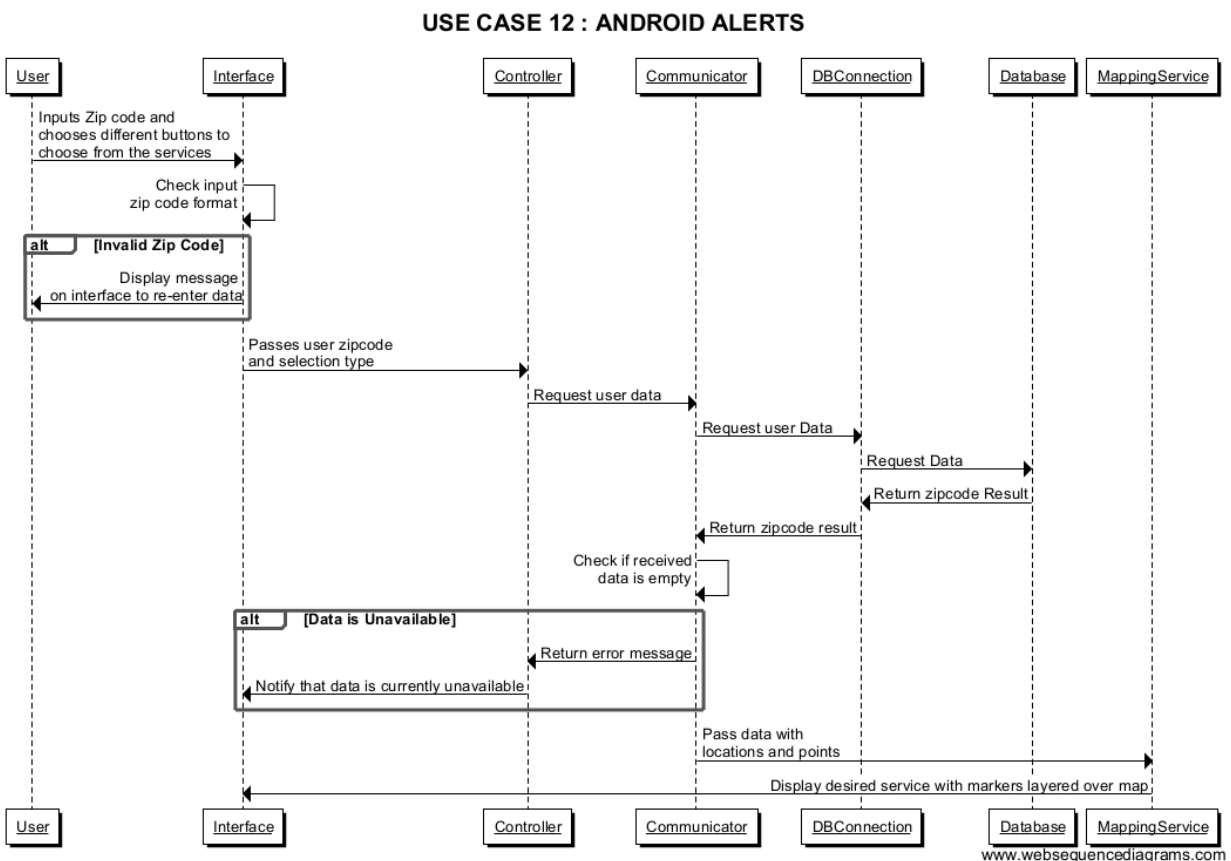
Database through the DBConnection with this user input. The Communicator queries the Database to get the desired data for the specified zip code. This resultant data is passed back to the Communicator through DBConnection. The Communicator now uses the type of the user input to choose where this data has to be sent. The data in this Use Case is traffic related alert data and hence, the data is passed to TrafficData. The TrafficData holds data related to traffic and alerts and the data from the Communicator will hold all the relevant information that could be used to plot the User's desired options to complete this Use Case. The TrafficData will then pass this data to the MappingService (Google Maps API) which is responsible for plotting this data at the defined locations on the Map. The MappingService will drop markers on the map to indicate the Traffic Congestion present in that zipcode.

In the Alternate scenario for this Use Case, we have considered two scenarios i.e when the user enters invalid zip codes or the data for the specified zip code is not available. In the case that the entered zip code is erroneous the interface will check if the entered zip code is in the right format and if not, it will display an error to the user asking the user to re-enter the data correctly. In the case where the data for a particular zip code is not present, once the user enters the input and submits it, the interface passes this data to the Controller which acts as the mediator and passes the information to the Communicator. The Communicator queries the Database through the DBConnection for the desired data. The database will return an empty data set which will be passed to the TrafficData entity through the Communicator. On receiving the empty data, the TrafficData entity will send out a message to the user notifying the user that the alerts data of that specified area is not available currently.



The above interaction diagram shows the interaction between system components when the user accesses the Severity trends feature in the application (**Use Case 7**). In this use case, the user can view the traffic trends from the different zip codes by inputting the zip code of the desired area in a text box and hitting the submit button. Once the user hit the submit button, if the zip code is valid, the interface passes the user input to the Controller. The Controller then connects to the Communicator and passes the user information to it. The Communicator will then query the Database using the DBConnection and retrieve the relevant information for the desired zip code. After that, the data is passed to the TrendingTraffic entity in the form of arrays of the average severity data and the TrendingTraffic entity then plots them on the interface in the form of charts for the different days in the respective zip code.

The alternate scenario considered for this Use case is where the user enters an invalid zip code in the Text box. In this scenario, the interface will check the format of the entered zip code and if the format is wrong, an error message will appear asking the user to re-enter the data.

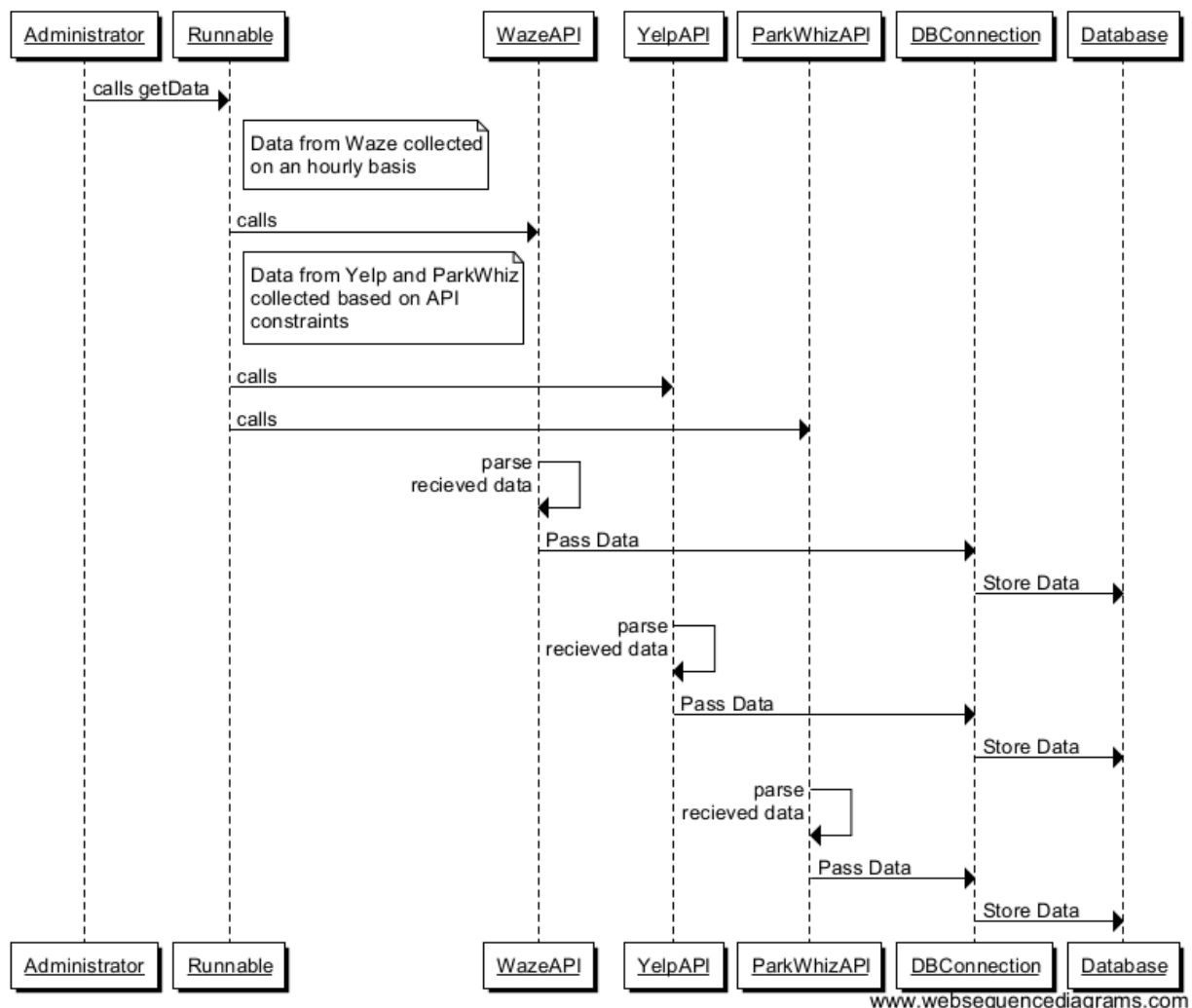


The above interaction diagram shows the interaction between the various system components that are needed when the user accesses the mobile Android application (**Use Case 12**). As we can see from the interaction diagram, the basic components are similar to the web application. This is kept similar so that the android application has a similar feel as the web application. The interaction starts with the home page of the user device. In this the user will enter a zip code of the desired location he wants to access the data for. If the zip code is in the right order, the map is zoomed to the area around where the zip code is located. Now the user has buttons

located on the screen which trigger the various features of the app. When the user enters the zip code and selects a feature, the interface passes this user information to the Controller. The Controller forwards this information to the Communicator with the type of information that needs to be given back to the user. The Communicator then connects to the Database using the DBConnector and queries for the data of that area. Once the data for that area is got, the Communicator calls the MappingService which is the Google Maps API and the information about the latitude, longitude and description of the alert are passed to it. The MappingService will then plot the points with the designated markers at the desired locations.

In the alternate scenario for this Use Case, we consider 2 cases: where the inputted zip code is wrong and where the data for the destination is not present. In the 1st case we can see that when the user inputs a wrong zip code, the interface will check for the format and send an error message asking the user to re-enter the data if it is wrong. In the other scenario, where the data is not present, once the Communicator received the information from the database, it will check if the data is empty. If the data is empty, it prompts the Controller to inform the user that the data for the service is currently unavailable.

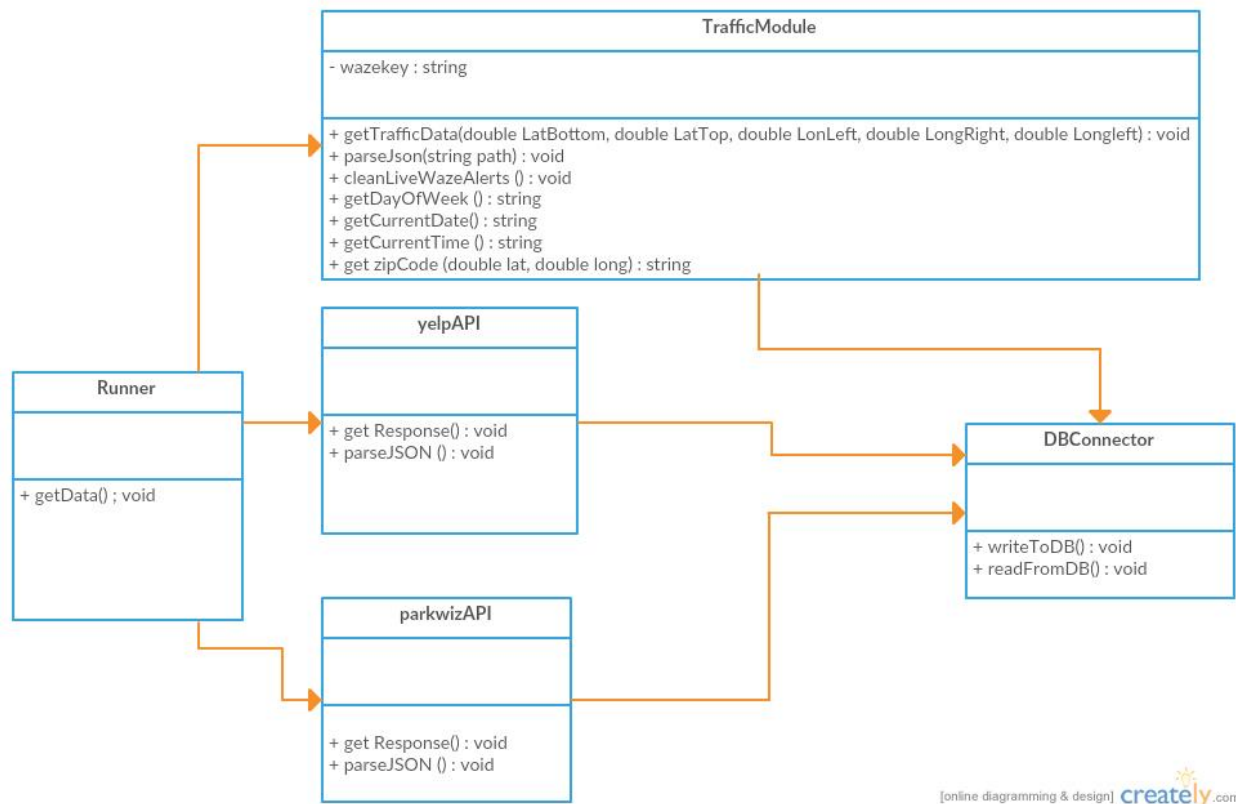
USE CASE 13,14,15 : DATA COLLECTION



The above interaction diagram shows the system interaction for Data Collection (**Use Case 13,14 & 15**). The 3 use cases have been clubbed because the interaction for the various entities is similar in all the use cases. Here, we can see that the Administrator is the Initiating Actor. The administrator will invoke the backend script which will call the entity called Runnable. This entity is responsible for running the code to collecting the information from the different API's to store in the Database. The Runnable entity will call the different entites viz. WazeAPI, YelpAPI and ParkWhizAPI. These entities perform a similar function and that is to get response from the respective API's and parse the information. The WazeAPI information runs every hour so that the latest alert and uniform traffic information is stored in the Database. The Yelp and ParkWhiz API collection are limited by the number of calls that can be done to the API and so are run at a lesser frequency. The WazeAPI, ParkWhizAPI & YelpAPI get the JSON responses from the API's, parse the information and then store it into the Database by connecting to the DBConnector. The respective data is stored in the various tables and updated periodically.

Class Diagrams

Class Diagram for data collection:



Use Cases 13,14,15 (Use cases for data collection):

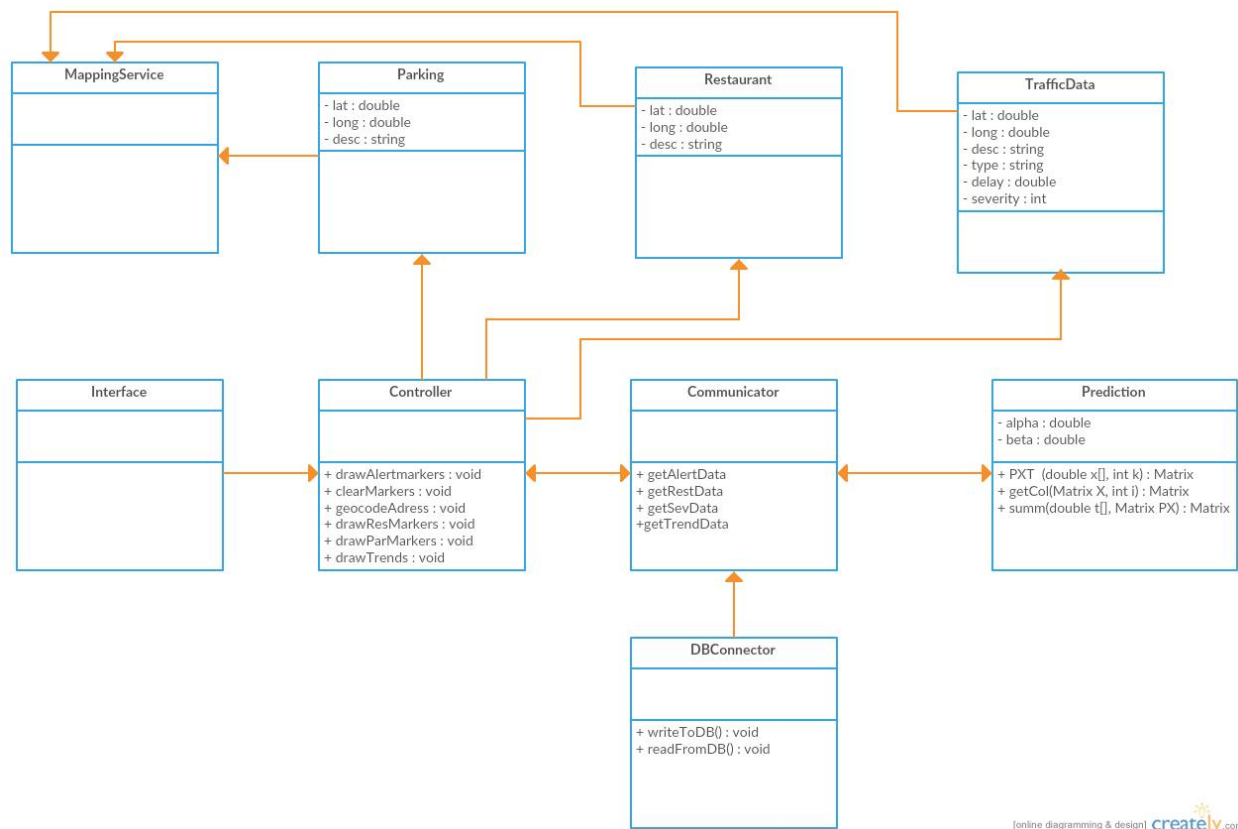
Class **Runner** will have a function called **getData ()** which will call class **TrafficModule**, class **yelpAPI** and class **parkwizAPI** on timely basis to run PHP script inside them for data collection.

Class **TrafficModule** will query for traffic data from waze API. Waze will return JSON response to it. This response will be break down and only selected parameters such as latitude, longitude, day, date, zipcode and time will be separated out from it and this data is stored to database using class **DBConnector**.

Similarly, class **yelpAPI** and **parkwizAPI** will get a response from **yelp.com** and **parkiz.com** in JSON format and they will parse this data using **parseJSON()** and store it into the database using class **DBConnector**.

Class Diagram for Mapping and Prediction

Algorithm:



Use Case 3:

This use case deals with mapping live congestion according to the severity level for given zipcode. From the interface, controller will call drawAlertMarkers() which will communicate with Database Connector using Communicator. DBConnector will read the data from database and pass it to getAlertData. Now for marking this data, Controller will call class TrafficData which has certain parameters for markers such as latitude, longitude etc. Finally, class MappingService will plot this data on google maps.

Use case 4:

This use case deals with mapping construction, police, accidents for given zipcode. Again the same procedure will be followed. From the interface, controller will call drawAlertMarkers() which will communicate with Database Connector using Communicator. DBConnector will read the data from database and pass it to getAlertData. Now for marking this data, Controller will call class TrafficData which has certain parameters for markers such as latitude, longitude etc. Finally, class MappingService will plot this data on google maps.

Use case 5:

This use case deals with predicting the traffic. User should be able to query for traffic congestion for a particular day and hour. Output will present congestion with predicted delay in Sec. For this from the interface, controller will call drawAlertMarkers() which will communicate with Database Connector using

Communicator. DBConnector will read the data from database and pass it to class Prediction. Class Prediction will run the Bayesian prediction algorithm and return the data to the class Communicator. Now for marking this data on map, Controller will call class TrafficData which has certain parameters for markers such as latitude, longitude etc. Finally, class MappingService will plot this data on google maps.

Use case 6:

This use case deals with mapping restaurants and parking slots for given zipcode. From the interface, controller will call drawResMarkers() and drawParMarkers() which will communicate with Database Connector using Communicator. DBConnector will read the data from database and pass it to getResData() and getParkData(). Now for marking this data, Controller will call class Parking and class Restaurant which has certain parameters for markers such as latitude, longitude, description etc. Finally, class MappingService will plot this data on google maps.

Use Case 7:

User should be able to trend average severity for a specific Zipcode. Output is hour trend for the past week. From the interface, controller will call drawTrends() which will communicate with Database Connector using Communicator. DBConnector will read the data from database and pass it to getTrendData. Finally, controller will connect to class MappingService to plot this data on Webpage. (using JavaScript).

Use case 8,9:

These use cases are similar to use case 7, only they are specific for police report data and accidents data respectively.

Design Pattern:

The above class diagram follows MVC design pattern in which A controller can send commands to the model to update the model's state. It can also send commands to its associated view to change the view's presentation of the model A model stores data that is retrieved according to commands from the controller and displayed in the view. A view generates an output presentation to the user based on changes in the model. A view controller generates an output view and an embedded controller.

Data Types and Operation Signatures

1) TrafficModule ->

Attributes:

- wazekey : string - waze API key variable

Operations:

+ getTrafficData(double LatBottom, double LatTop, double LonLeft, double LongRight, double Longleft) : void – API requesting function

+ parseJson(string path) : void – parsing JSON data.

+ cleanLiveWazeAlerts () : void – clear live waze alerts.

- + getDayOfWeek () : string – getting day of the week from waze response.
- + getCurrentDate() : string - getting current date.
- + getCurrentTime () : string – getting current time
- + get zipCode (double lat, double long) : string – getzipcode from waze response.

2) yelpAPI ->

Operations:

- + getResponse() : void – getting response from yelp API in JSON format.
- + parseJSON () : void – parsing JSON response and extracting only specific data out of it.

3) parkwizAPI ->

Operations:

- + getResponse() : void – get response from parkwiz API
- + parseJSON () : void - parsing JSON response and extracting only specific data out of it.

4) DBConnector ->

Operations:

- + writeToDB() : void – writing data to database
- + readFromDB() : void – reading the data from database.

5) Runner ->

Operations:

- + getData() : void – call parkwizAPI, yelpAPI and TrafficModule classes and run script time to time to get the data.

6) Prediction ->

Attributes:

- alpha : double - Bayesian constant
- beta : double - Bayesian constant

Operations:

- + summ(double t[], Matrix PX) : Matrix – returns predicted severity matrix for given zipcode.

7) Restaurant ->

Attributes:

- lat : double – latitudes of the restaurants to be plotted.
- long : double - latitudes of the restaurants to be plotted.
- desc : string - description of the restaurant slots

8) Parking ->

Attributes:

- lat : double – latitudes of the parking slots to be plotted.
- long : double - latitudes of the parking slots to be plotted.
- desc : string – description of the parking slots

9) TrafficData ->

Attributes:

- lat : double - latitudes of the traffic alert/severity to be plotted.
- long : double - latitudes of the traffic alert/severity to be plotted.
- desc : string - description of the traffic alert/severity
- type : string – type of traffic alert
- delay : double – delay in seconds according to severity
- severity : int – a number range from 0-4

10) Controller ->

Operations:

- + drawAlertmarkers : void – calls Mapping service class to plot alert markers
- + clearMarkers : void - calls Mapping service class to clear all markers
- + geocodeAdress : void - calls Mapping service class to mark geocode.
- + drawResMarkers : void - calls Mapping service class to plot restaurant markers
- + drawParMarkers : void - calls Mapping service class to plot parking markers
- + drawTrends : void – display weekly historical traffic trends on webpage.

11) Communicator ->

Operations:

- + getAlertData : void – get alert data from database
- + getRestData : void - get restaurant and parking data from database
- + getSevData : void - get severity data from database
- + getTrendData : void - get trend data points from database

Traceability Matrix:

	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-13	UC-14	UC-15
Runner								Initiates PHP script run for traffic data collection	Initiates PHP script run for restaurant data collection	Initiates PHP script run for parking data collection
TrafficModule								Runs the PHP script, get the data from waze API and parses it		
yelpAPI									Runs the PHP script, get the data from yelp API and parses it	
parkwizAPI										Runs the PHP script, get the data from parkwiz API and parses it
DBConnector	retrieves the data when Controller asks for it	retrieves data when Controller asks for it	Reads the data from database	retrieves data when Controller asks for it	retrieves data when Controller asks for it	retrieves data when Controller asks for it	retrieves data when Controller asks for it	Connects to DB and writes the data into it	Connects to DB and writes the data into it	Connects to DB and writes the data into it
Prediction			Runs the Bayesian prediction algorithm							
MappingService	Plots the live congestion data on google map	Plots the accident s , police alerts constrciton data on google map	Plot the the predicted data on google map	Plots the restaurant and parking data on google map	Plots the historical trend on webpage	Plots the historical accidents trend on webpage	Plots the historical police data trends on webpage			
Parking				Helper class for Mapping service to plot parking data						
TrafficData	Helper class for Mapping service to plot traffic data	Helper class for Mapping service to plot traffic data	helper class for plotting the data		Helper class to plot traffic trends	Helper class to plot accident trends	Helper class to plot police data trends			
Restaurant				Helper class for Mapping service to plot restaurant data						
Communicator	Communicates with DB connector to retrieve the data	Communicates with DB connector to retrieve the data	Communicates with DB connector to retrieve the data	Communicates with DB connector to retrieve the data	Communicates with DB connector to retrieve the data	Communicates with DB connector to retrieve the data	Communicates with DB connector to retrieve the data			
Controller	Initiates the plotting action by requesting the data from database	Initiates the plotting action by requesting the data from database	Initiates the plotting action by requesting the data from database	Initiates the plotting action by requesting the data from database	Initiates the plotting action by requesting the data from database	Initiates the plotting action by requesting the data from database	Initiates the plotting action by requesting the data from database			

Object Constraint Language (OCL) Contracts:

MappingService

Invariants: Registered Users and non-registered Users can access this page.

Pre-conditions: User has filled out all inputs.

Post-conditions: Markers are plotted on map/trends are displayed on webpage.

Runner:

Invariants: Administrator has enabled the data collection scripts.

Pre-conditions: Ready to run the scripts for receiving API response

Post-conditions: Received the API response from corresponding classes.

DBConnector:

Invariants: Database will respond to the queries

Pre-conditions: Data is ready to be written/to be read

Post-conditions: Data is retrieved from databases/stored into the database.

System Architecture & System Design

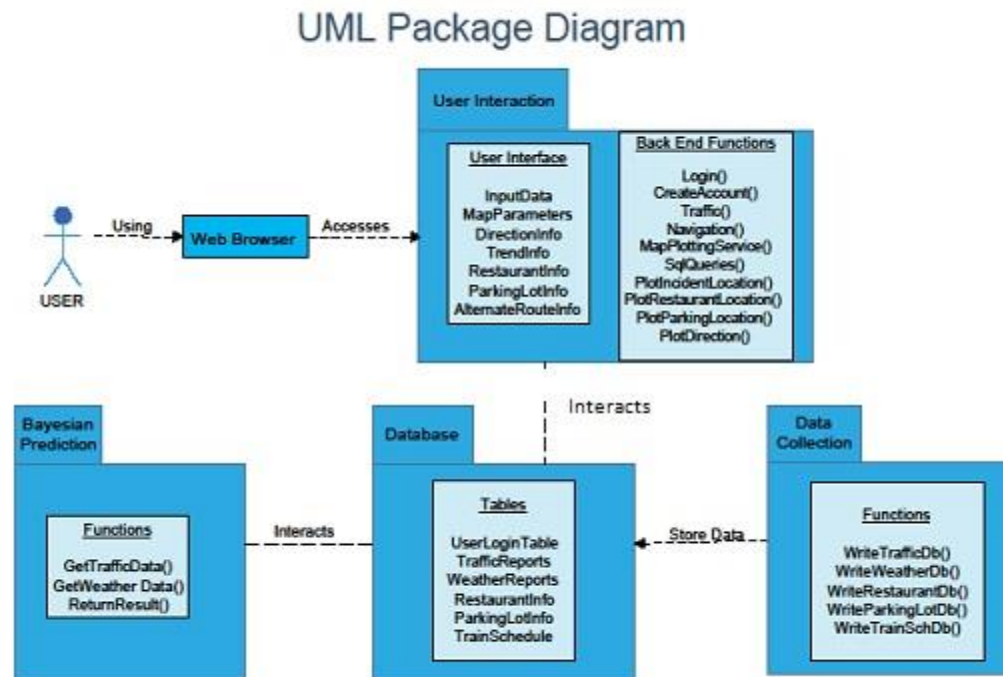
Architectural Types:

The most important actors of the traffic monitoring system are the users who will use the application and the system (which includes the application, database etc.) that'll provide the information after processing the data. Hence, the primary architectural style used in this project is the Client- Server style. The Client–Server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. In our case, the client is going to be the user and the server will be the system.

The clients and servers will communicate over a computer network on separate hardware. Clients and servers exchange messages in a request–response messaging pattern: The client sends a request, and the server returns a response. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. The user of the application is only meant to interact with the basic user interfaces of the Web Application. 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.

For an easy user experience, it is important to reduce the number of interactions of the user with the many processes or classes that will be present in the application to process the user information. In this project the user will need to control only a series of inputs (text boxes, dropdown menus, radio buttons, etc.) and has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service. These aspects of the project are all very simple to interact with and designed to have minimum user effort. The server sends requests to server side classes based on the input from the user and hence, the user will never interact with these classes. Using these simple inputs from the user, a complex output is displayed for the user, by the server. The classes execute in a similar way each time. 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 with POST request method. 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 Interaction subsystem. The user interaction subsystem contains all of the user inputs, and the services that have to be displayed by the system. These services include the map output, the directions output, the add-ons like restaurant locations, parking lot locations and train schedules. The user will interact with only the interface and provide his inputs. The other part of the subsystem includes the backend functions that will run 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. These functions are not accessible by the user.

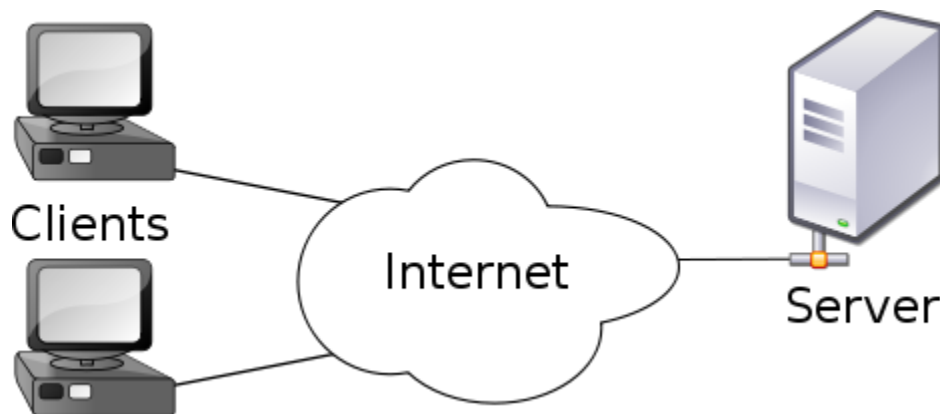
The second subsystem is the Data Collection subsystem. This system involves the functions to get and write the information got from the web services to the Sql Database. The various service receivers access the relevant web services in order to find more data to store into the database. Because it stores data into the database, this subsystem clearly also connects to the database.

The next subsystem is the Prediction subsystem. This system is responsible for generating results based on the input from the user and the data stored in the database. This system uses

the data from the user and the database data to predict the traffic that could be expected in any given area. This subsystem returns the result in the form of the predicted values for the traffic. Hence, this subsystem is connected to the Database and interacts with the Database.

The final subsystem is the database. The database is responsible for storing the data got from all the other subsystems. This database will make tables which will hold information like the user login details, the traffic data, weather data, the restaurant location data, the parking lot location data and the train schedules. The other subsystems will access and interact with this subsystem whenever it requires to access historical data.

Mapping Subsystem to hardware

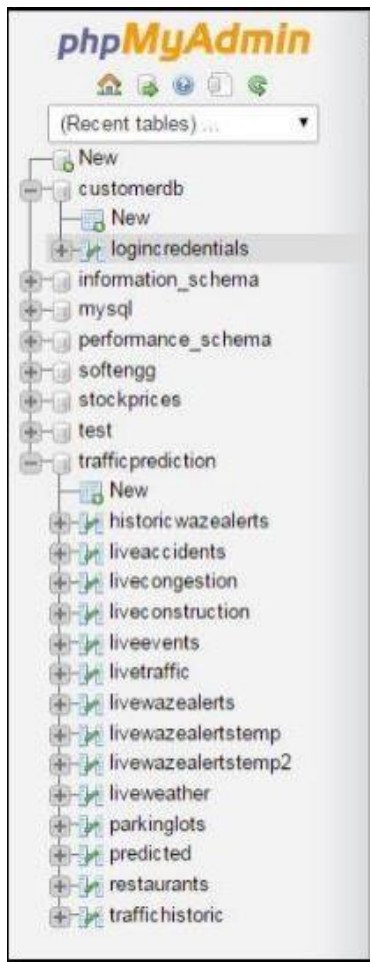


We are using the client server subsystem. We are hosting our database and php file with the help of webserver called wamp. Wamp as a server component provides a function or service to one or many clients, which initiate requests for such services.

WampServer refers to a software stack for the Microsoft Windows operating system consisting of the Apache web server, OpenSSL for SSL support, MySQL database and PHP programming language.

Persistent Data Storage

We are using MySQL to store our data. We have two databases named "customerdb" and "trafficprediction". We have a data collection script which is getting the JSON response through http get method and after parsing the JSON response we are storing the data into the Database. We have few extra tables in our database that is because we changed our source for data collection from 511nj to waze. The Database schema is attached below:



1) We have one name table named logincredentials which is storing user data in it. It has the field as mentioned below.

#	Name	Type	Collation	Attributes
<input type="checkbox"/> 1	<u>username</u>	varchar(15)	latin1_swedish_ci	
<input type="checkbox"/> 2	password	varchar(15)	latin1_swedish_ci	
<input type="checkbox"/> 3	FirstName	varchar(20)	latin1_swedish_ci	
<input type="checkbox"/> 4	Email	varchar(50)	latin1_swedish_ci	
<input type="checkbox"/> 5	ZipCode	char(5)	latin1_swedish_ci	

2) trafficprediction - This database has several tables in it. livewazealerts and historicwazealerts have same structure and they are storing the data which we are collecting from waze API.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	ID	int(11)			No	None	AUTO_INCREMENT
2	ZipCode	char(6)	latin1_swedish_ci		No	None	
3	numofThumbs	int(11)			No	None	
4	incidenttype	varchar(40)	latin1_swedish_ci		No	None	
5	incidentsubtype	varchar(100)	latin1_swedish_ci		No	None	
6	placenearby	varchar(100)	latin1_swedish_ci		No	None	
7	latitude	double			No	None	
8	longitude	double			No	None	
9	Day	text	latin1_swedish_ci		No	None	
10	Date	date			No	None	
11	Time	time			No	None	

3) Table livecongestion is storing the congestion data from waze API. Table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	Id	int(11)			No	None	AUTO_INCREMENT
2	ZipCode	char(6)	latin1_swedish_ci		No	None	
3	Date	date			No	None	
4	Time	time			No	None	
5	Day	varchar(10)	latin1_swedish_ci		No	None	
6	Severity	int(11)			No	None	
7	CongestionType	varchar(200)	latin1_swedish_ci		No	None	
8	EndStreet	varchar(200)	latin1_swedish_ci		No	None	
9	StartStreet	varchar(200)	latin1_swedish_ci		No	None	
10	Street	varchar(200)	latin1_swedish_ci		No	None	
11	StartLatitude	double			No	None	
12	StartLongitude	double			No	None	
13	EndLatitude	double			No	None	
14	EndLongitude	double			No	None	
15	Delay	double			No	None	

4) Table parkinglots is storing the data from parkwhiz API. Table structure is as follows:

#	Name	Type	Collation	Attributes	Null
1	Latitude	double			No
2	Longitude	double			No
3	Name	varchar(29)	latin1_swedish_ci		No

5) Table restaurant is storing the data from YELPAPI. Table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Ext
<input type="checkbox"/> 1	Latitude	double			No	None	
<input type="checkbox"/> 2	Longitude	double			No	None	
<input type="checkbox"/> 3	Name	varchar(29)	latin1_swedish_ci		No	None	

Global Control Flow:

1) Execution Orderness-

The system is event driven as users can generate actions in different orders each time they use the system. For example, a user could use request traffic and weather predictions, or they could request navigation. While navigating they could select a restaurant, or a parking lot, or neither.

2) Time Dependency-

The rate that traffic and weather data is updated is on a timer. The rate that the data is updated can be adjusted by an administrator. There are no timers that are related to what a user would be using the system for. Users are able to perform actions at any time.

Hardware Requirements:

The website will be hosted on a remote server. And the data collected is stored in a database, on multiple servers, that is the host server deploying the website and the project members. The database being used is MySQL and the server being used presently is Apache TOMCAT 8.0. We are currently working on making sure the Database containing the traffic data and the website are on the same host.

Algorithms and Data Structures

How does our algorithm predict traffic congestion? A lot of the traffic websites like Yahoo, Bing, and MapQuest use only the current traffic incident reports to predict traffic. A typical use case would be a user logging into Yahoo at 7am to check traffic conditions for a particular highway. Due to it being 7am, Yahoo will show no congestions because normally the congestion starts around 8am. If the user has a long commute, they would get stuck in traffic because by 8AM, the traffic has increased tremendously and it's too late for him to take a detour. Hence to handle that use case, our algorithm would also take into account previous day's data. So when a user does login to our website, they will be given an option to select a time and if the time is past the current time (future), the algorithm will query the database for previous historic data for the same time and make a decision accordingly.

Another aspect of traffic prediction is accident conditions. We all know accidents do not occur every day on the same road; if an accident caused traffic congestion today, it would not cause traffic congestion tomorrow. In order to handle this use case, we will be using a weighted algorithm prediction hereby analyzing current traffic data as well as historic traffic data.

Let's assume the user wants to predict traffic for time 9am and the query was executed at 8am. This means that our current database has traffic incident data up to 8am. Our weighted algorithm will analyze the current traffic data and check whether there has been any change in traffic conditions in the past hour (between 7AM-8AM). If there have been any changes, we know traffic congestion is starting to increase. In this case, our traffic predictions will put more weight on the current traffic and less weight on yesterday's traffic. The reason we chose to have more weight on today's traffic is because if there was an accident in the morning, we simply do not want to predict light traffic based on yesterday's traffic. Additionally, we are also using the weather to help with predictions. If the weather is raining or snowing, it will perform a query on the database for snow/rain days and use the traffic incident data for those days. The weather data will have a weight factor associated as well, if and only if the weather was bad, meaning it rained/snowed

Prediction Algorithm:

These are machine-learning algorithms we use to predict the future value, when a set of past events is given.

Curve fitting is the basis of prediction. It is a process of drawing a curve that best fits with a given data set. Given a particular set of points, we draw a graph that passes through all the points or closest to all the points and passing through most of them. Thus making it possible for predicting further values. In this process it is important to carefully select the order, there is a chance that the curve may pass too far from the points if the order is too low and if the order is too high there is a chance that there are unnecessary oscillations and the prediction may not be right.

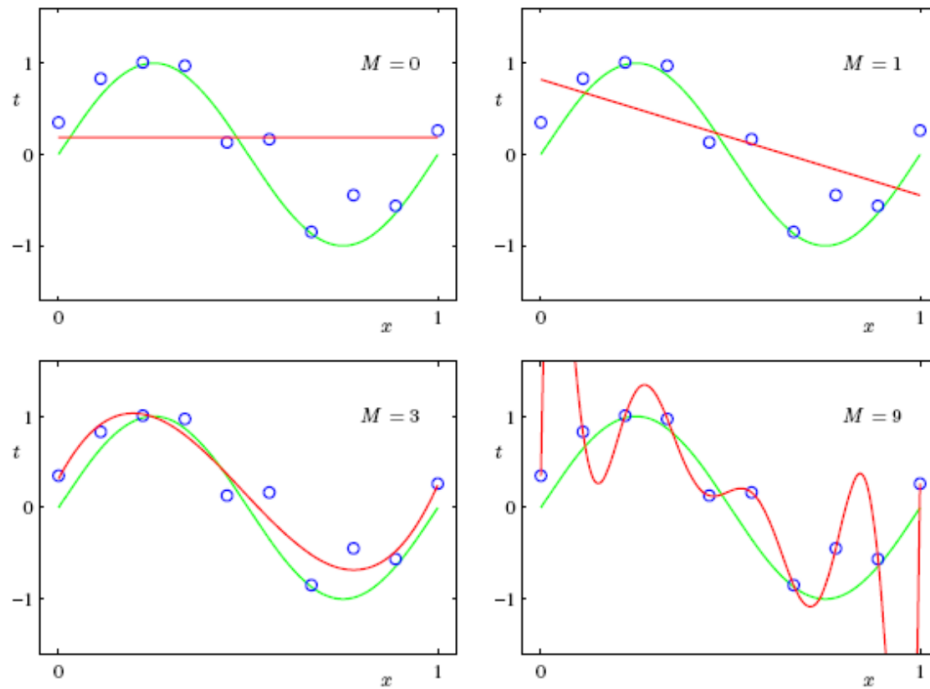


Fig: It shown the curve for different values of order M .

There are many types and strategies used to implement curve fitting. We will be using a couple of them in the project to achieve our results with the most probability of it being right. Some of the strategies used have been described below.

Bayesian Curve Fitting:

The Bayesian Curve fitting uses Bayesian probability, that is we try to predict the distribution of data. We consider the previous data as training data X and t and another test point x to predict another t . So the distribution we get is $p(t|x,X,t)$.

$$p(t|x, x, t) = p(t|x,w) * p(w|x, t) dw$$

Use of weighted average method in the prediction strategy, giving more weight to inputs that are more relevant to output than the ones less relevant. In this case when you take the case of a particular time, for example, 5P.M. is the time when people start coming home and traffic suddenly increases. If we use the live data to see the traffic congestion, it is hard for the user to estimate how much time it is going to take to reach the destination. If we use the prediction then, we can get a precise idea of the delays on the road at the preferred time, the user wants to leave.

Traffic congestion	Weight
High	3
Moderate	2
Low	1
No traffic	0

So in the equation:

$$p(t|x, x, t) = p(t|x, w) * p(w|x, t) dw$$

We take t value, as the Delay value obtained from the congestion table and x is date/time so we predict the value of t depending on the test set, which is the future date/time. So that the value we obtain is the congestion Delay at the particular time. We consider all the Delays from the same day and time intervals for example, Thursday, 1P.M-2P.M. to estimate the traffic delay on the next Thursday, 1P.M-2P.M. it might be in an hour or on other day.

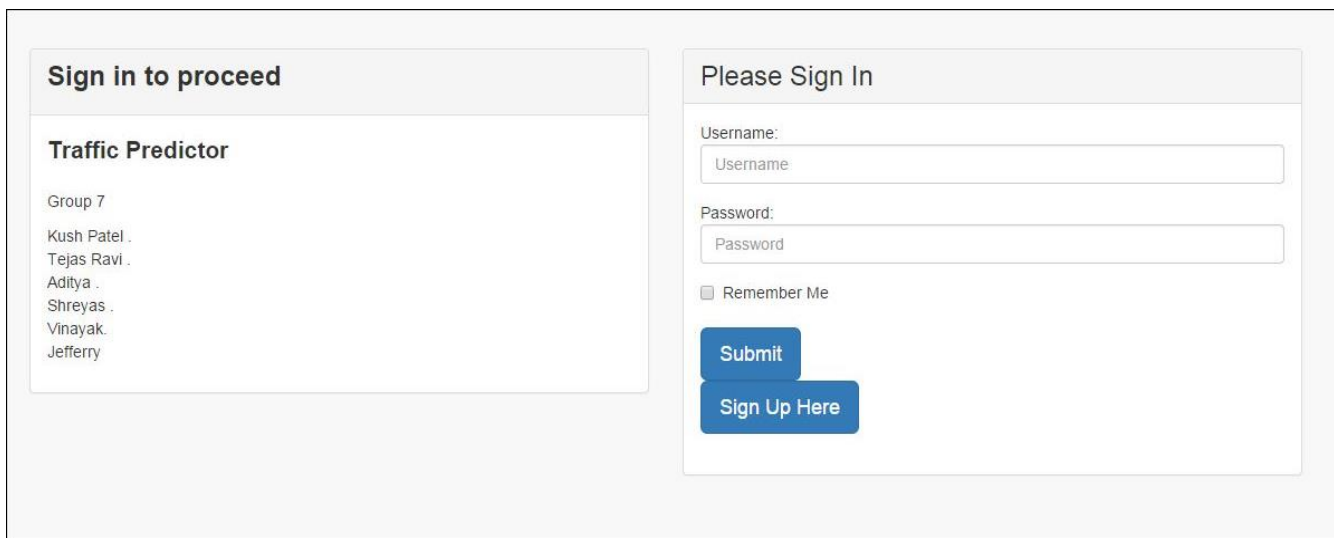
The delays are predicted for every street available in the Database table and stored into another table named predicted along with the Latitude, Longitude and the delay, which is used to plot it on the map in the website under Severity.

No complex data structures are used in this project.

User Interface Design and Implementation

Since the main objective of the project is to display results on a web page which can be easily understood by the user, the interface implementation plays a key role in achieving this goal. We have tried to create an interface which is easy to use and requires the user to follow a similar procedure to access the various features of the application.

In the preliminary design of the User interface, the approach adopted by us was to provide drop down menus to select from among the various services that could be accessed by the user. However, while implementing the Interface, we realized that we could make a template for the user interface which will be identical for the various pages, to a great extent. Hence, we decided to create a sidebar which would be present on the different pages of the interface to toggle between the various services of the application. This reduces the user effort by at least 1 mouse click each time he/she wants to switch between services. Also, all the data is plotted on the interface with interactive markers and graphs which make the interface understandable and easy to use. The markers used are general (day to day) images which are self-descriptive and easy for a user to identify. With the current implementation of our interface, the various pages can be seen below:



The screenshot displays a web interface with a sidebar on the left and a main content area on the right. The sidebar, titled 'Sign in to proceed', contains a section 'Traffic Predictor' listing 'Group 7' members: Kush Patel, Tejas Ravi, Aditya, Shreyas, Vinayak, and Jeffery. The main content area, titled 'Please Sign In', features a login form with fields for 'Username' and 'Password', a 'Remember Me' checkbox, and two buttons: 'Submit' and 'Sign Up Here'.

Fig: Sign In/Sign Up Page.

The first page that a user comes across on loading the website is the user login page. The implementation has a 'Remember Me' button which can be used to save the user credentials which will save the user credentials and will reduce the effort of the user the next time he/she loads the website. The user login is important because our application is trying to personalize the user experience by storing the user information and providing certain features based on this data.

Once the user logs into the website using the user credentials, the following page appears which could be called as the User's Home page. This home page is where the user can select

among the various features that the application is built for. Each of these features, once clicked will link to a different page and the user can go back to any page by using the scroll bar on the page.

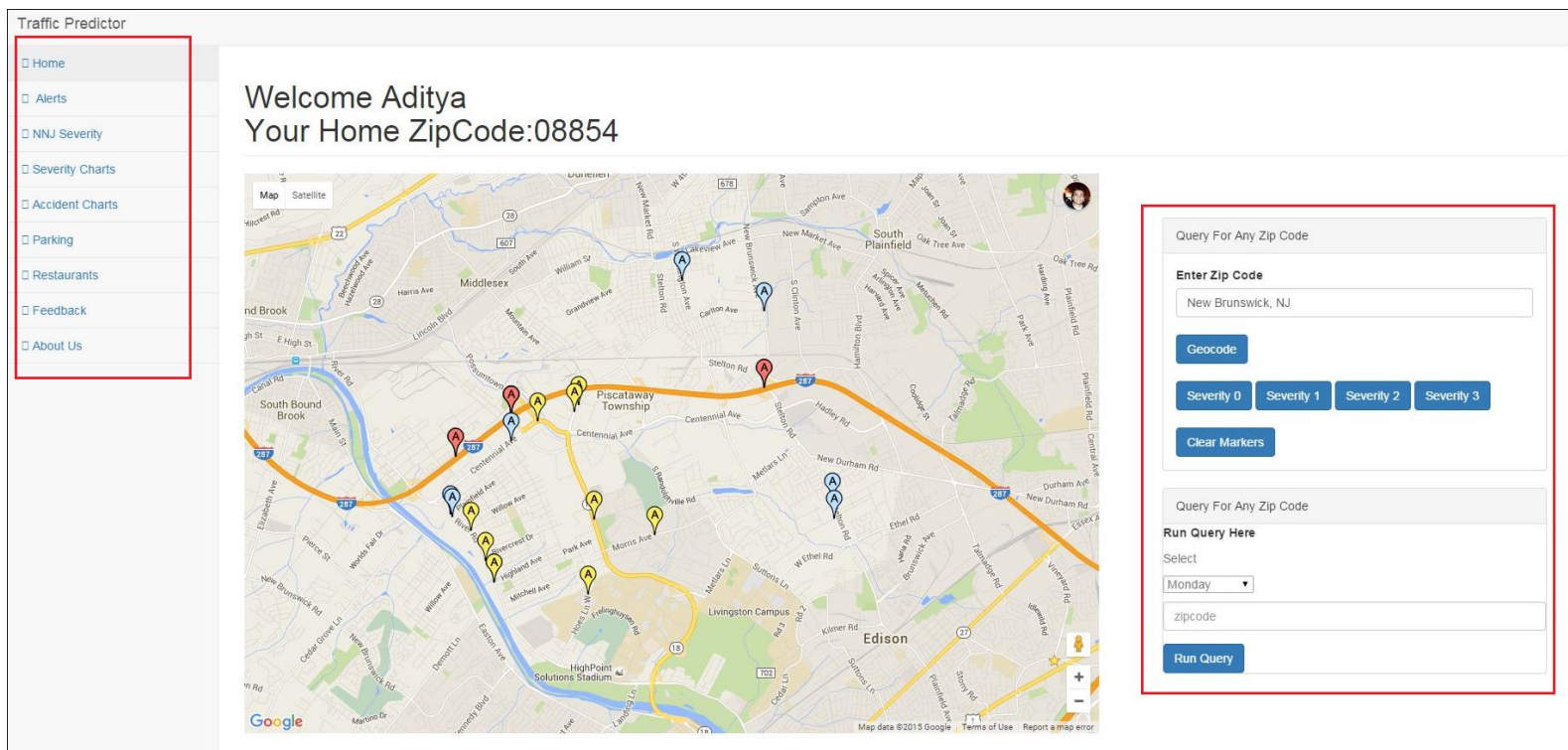


Fig: Home page

The home page screenshot above has 2 highlighted areas (marked by the red box). As seen from the screenshot, the Home page loads the area of the user's stored Home zip code. The main intention of the application is to plot the traffic intensities around the area the user chooses. By default, the area that is shown on the page is the Home Zip code and the user can choose a day (drop down at the bottom right of the page) and then plot the various severities that are seen from the data in that area.

The markers for the various severities are marked as colored markers with Severity markers varying from "Light blue" to "Red" to signify increasing traffic intensity. The future implementation of this project could try and plot the intensities in the form of lines on the roads that the severities are seen.

In the previous user interface implementation design our approach was that the user would have to input zip code of the desired destination. While implementing the code we figured out that we can geocode areas by taking the input in the form of addresses. So, in the current implementation, the user can plot and clear the severities in different areas by just entering addresses which makes the user experience better as it reduces the effort of the user to remember the zip codes of various areas. The minimum user effort considering zip codes would include 5 keystrokes followed by 3 mouse clicks to choose a day and submit and 1 mouse click to plot for each severity information.

The red block on the left top corner of the page shows the sidebar which holds the various features that a user can access by one mouse click.

The next feature that the user can access would be the Alerts page. When we were collecting data to store the congestion history, we realized that the data had data about the alerts in the areas which would include accidents, closed roads and police locations. The screenshot of the page is below:

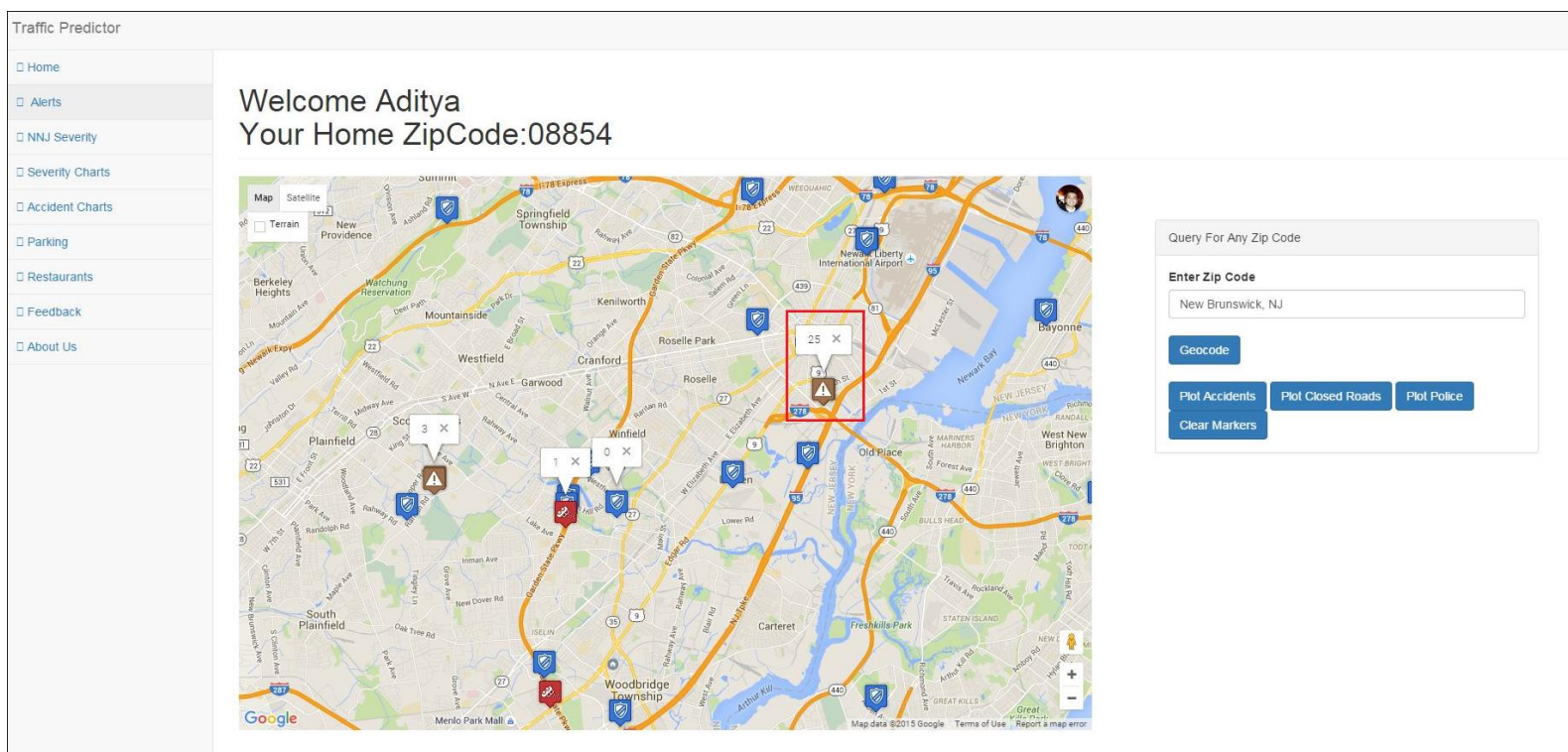


Fig: Alerts Page

As we can see, the interface layout is similar to the home page and requires the user to enter data in a similar manner as the home page. This makes the user experience and effort better. The user will enter the address for the area in which the alerts need to be plotted. The user can use one mouse click to plot the various markers on the map. These markers are informative and with a single click of a mouse the user can get information about the delay that will be caused due to these incidents. The data seen is in the order of seconds. As an added feature in the Police markers, the user can see the number of likes that actual users of the Waze application have entered, which authenticates the actual presence of Police in that location. The minimum user effort estimated for this page would include 5 key strokes for the zip code, followed by 1 mouse click for submitting it and then 1 mouse click to plot each incident marker.

The next feature that the user can view is the NNJ severity page. The user can navigate from a start location to a destination location. The navigated path will be highlighted in blue along the route. The user also can plot the severities of all the traffic that is present around the New York / New Jersey area and make an informed decision about which path he/she wants to take while travelling. The information on the markers are the delay (in seconds) that can be expected at those spots. The minimum user effort estimated to navigate through this page 18 key strokes for entering Start location, Destination location and time. It would be followed by 1 mouse click

to navigate and 1 mouse click each to plot the markers for the various severities. A screenshot of the page is shown below:

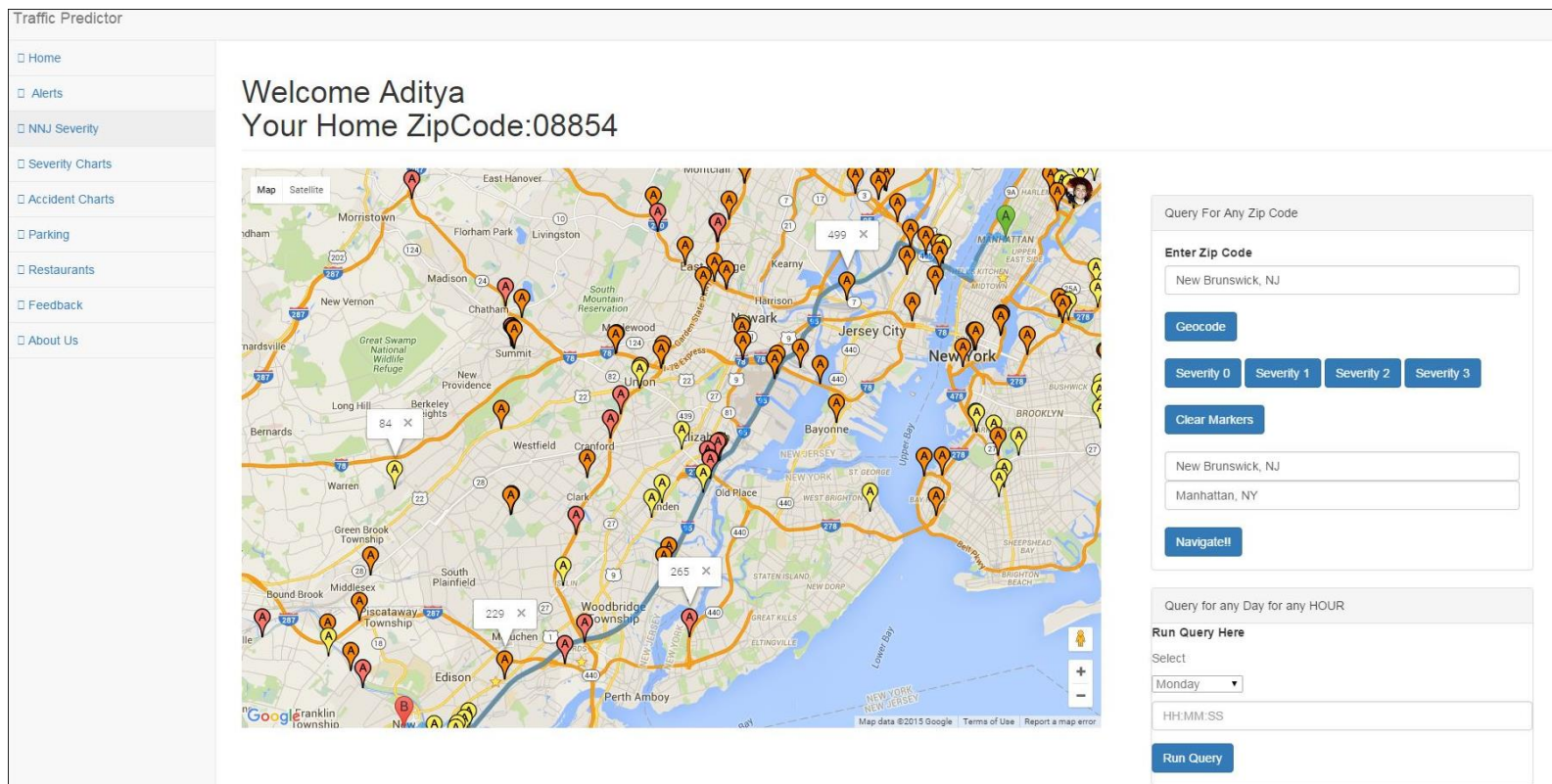


Fig: NNJ Severity & Navigation Page

The next 2 pages are dedicated to showcasing the data that has been accumulated over the days and the traffic and alert trends that have been noticed in given areas for each day. These pages display the data that is calculated across the desired zip code and show the aggregate of the information gathered from that area. The screenshots for these can be seen below:

[Home](#)

08854



[Home](#)

08876



Page | 72

The last 2 features of the applications have similar functionality and user effort required to navigate through these pages are the same. These services include the Restaurant and Parking information. In these services, the user can find restaurants and plot parking lots in and around the area that the user defines. The markers will pop a message to identify the name of the parking lot or restaurant name. The user effort estimated to navigate through this page would include a minimum of 5 keystrokes to enter the zip code of the desired location followed by 1 mouse click to submit the information and a click each to plot the restaurants or the parking spots. The screenshots of these pages can be seen below:

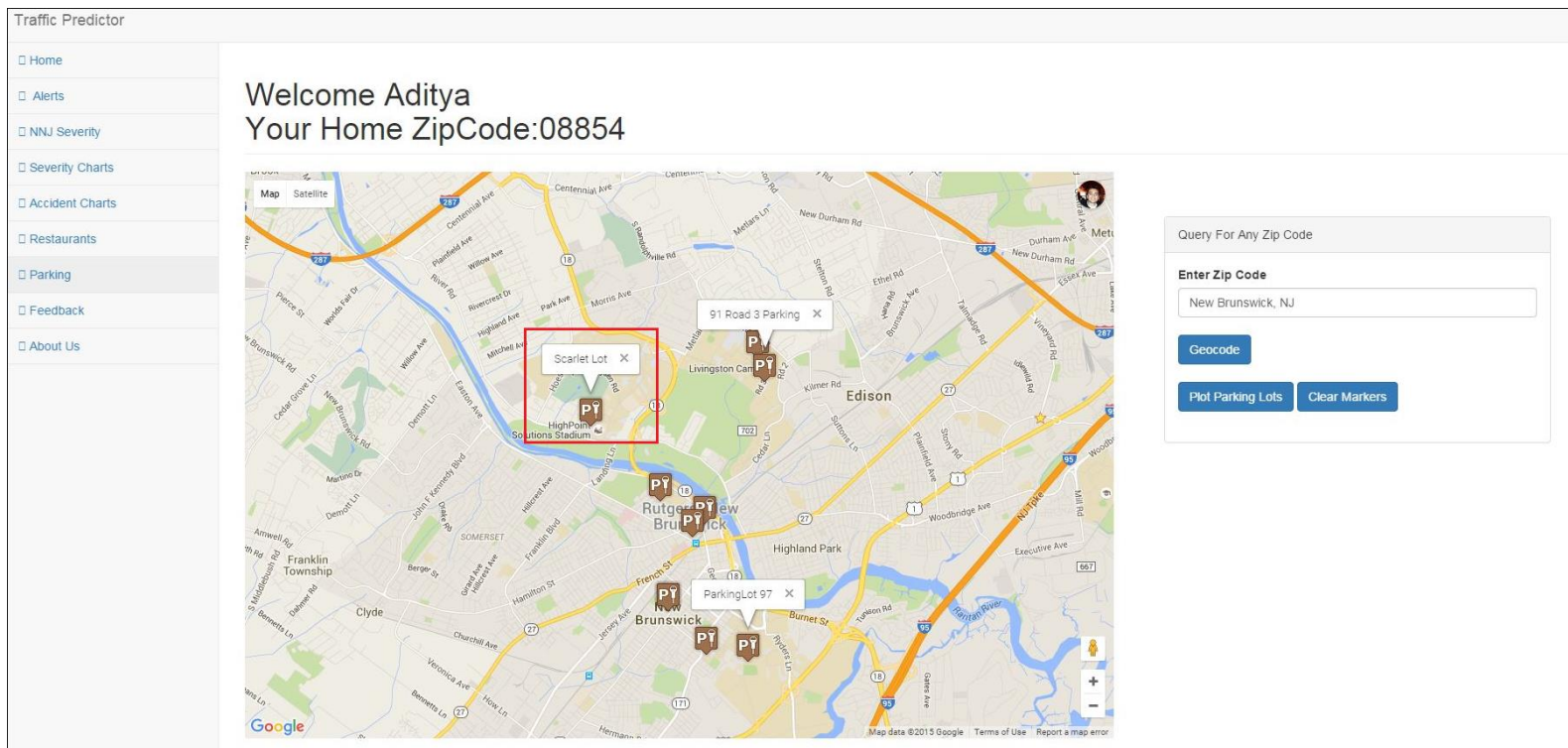


Fig: Parking Page

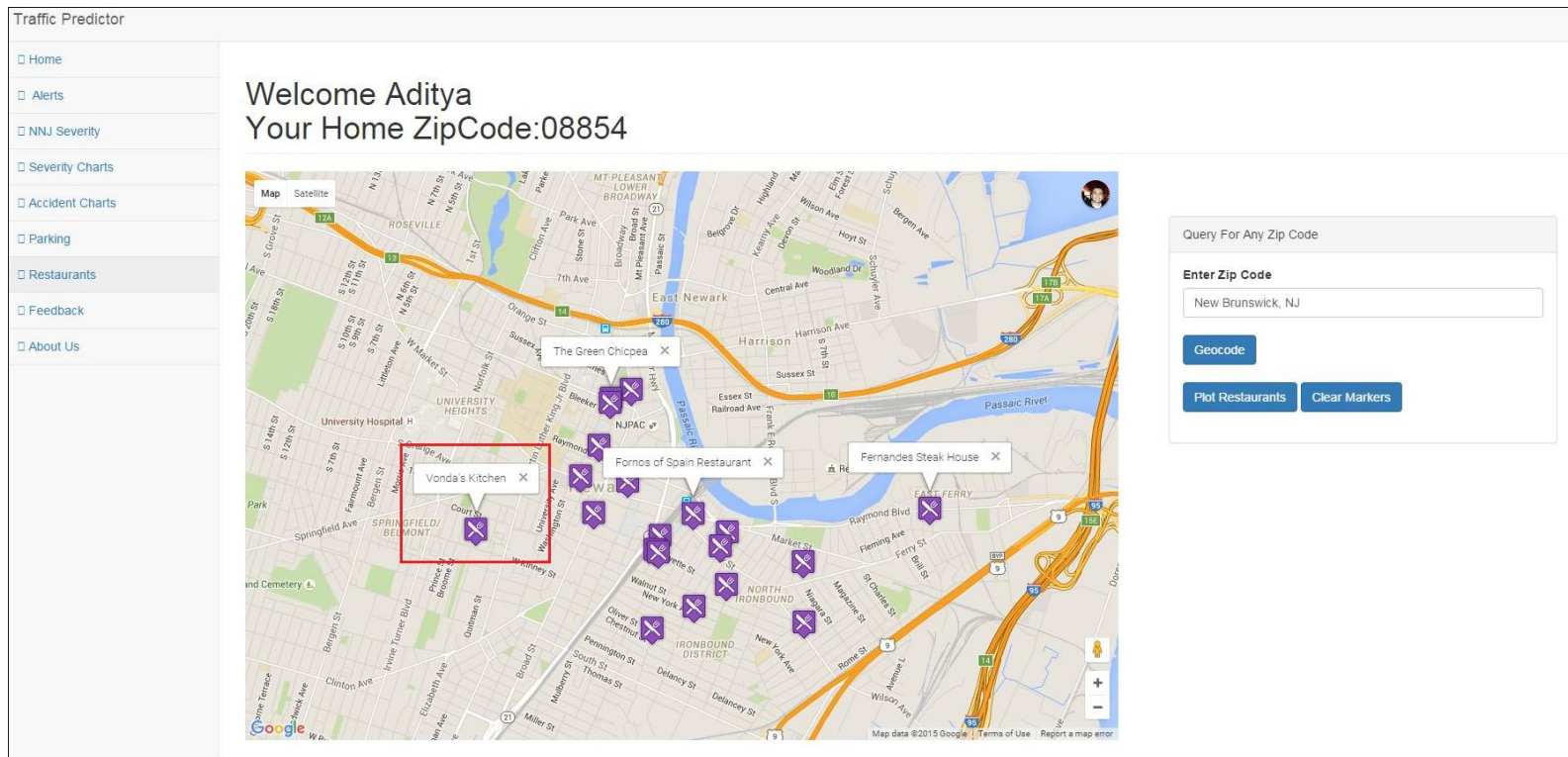


Fig: Restaurants Page

As an add-on, we created an android application which would perform the same functions as our web application. The feel of the application is similar to the website and any user that has used the website will be able to navigate through the application with ease. The screen of the mobile application is shown below:

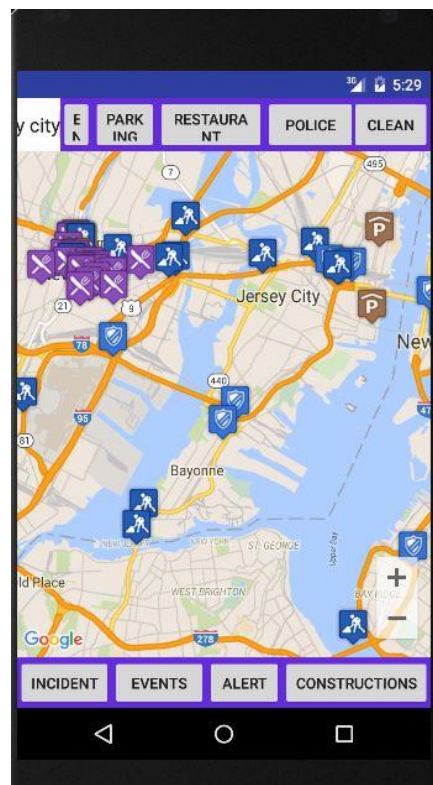


Fig: Android Application screen

The screen above includes buttons to recreate the same activities as the web application. There is a text box which needs the user to fill with an address of the destination. The android application reduces user effort by using an auto complete feature which prompts the user with possible data that needs to be fed in that field. Once the user enters the information, the map recalibrates and zooms into the desired location. Now the user has an option to plot any of the information from the buttons that are present on the screen. Having all the services present on the same screen reduces user effort of toggling between pages to view information about the same area. The user can clear each information whenever he/she wants. Each marker on the map is interactive just like the web application and will pop a message about that type of marker when clicked by the user. The effort estimated for the user would include 5 key strokes to input the zip code followed by one click to enter the information. Beyond this point the user will need 1 click to plot each marker and 1 click to obtain information about it.

Design of Tests

In this phase we have designed our website to open a particular location when loaded, that is New Brunswick. Since it is the place where the server is located.

The server contains of a Database with multiple number of tables. All the information stored in the database is obtained in the form of JSON files the JSON files are then parsed and stored in the database. For example Congestion data is obtained from Waze.com, restaurant data is obtained from Google Places and the parking lot data is obtained from parkwhiz API. Each table has its designated purpose. That is they are required to satisfy the following test cases.

Test case 1, Severity: The page shows a map and different severity level buttons and a blank space to enter a desired zip code. After the zip code is entered the maps is centered to its latitude and longitude. The severity level buttons can be used to mark the streets which come under a particular severity level. There are 4 levels of severity available given by 0- No Traffic, 1- Low, 2-Moderate, 3-High.

Once the user clicks on a particular “Severity Level” markers indicating the particular severity appear on the streets present on the map being displayed and once the marker is clicked, it displays the predicted delay. This data is fetched from a Predictions table which contains the Street name, Lat& Long, day, time, Predicted Delay, severity on the street. Once a particular button is pressed the table is parsed and relevant data is displayed on the screen.

Test case 2, Navigation: In the navigation.html user can enter a desired destination address and the source or starting point. Then the page gives directions to the particular location. Then the User can use the section below the map which shows "Accidents", "Construction" and, "Special events". To display where there is congestion on road. This is quite useful when the user wants to go out and check if there is congestion on any route to avoid it and take an alternate route.

The navigation is provided using a “GoogleMaps API” and the markers for the events, accidents, and congestion are obtained by using the latitude and longitude of an incident for example accident or police which is available in the Incidents Table and parsing through it. The table contains the kind or incident, Latitude and Longitude, description, and zip code.

Test case 3, Current traffic: User enters a particular location. It may be their home or any other location, then the map is centered to the entered location. The below the map section many options can be seen- Events, Construction, Accidents. The user clicks on the desired section which they want to see, either one or all of them, which can be cleared at any time.

Test case 4, Restaurants and Parking lots: The DataBase contains tables containing the information of restaurants and parking lots where we get the restaurant data from “Google Places API” and the parking lot data from parkwhiz along with the latitude, longitude and the zip code. Once given and input of zip code the map is centered on the particular area and upon clicking on a button we can see the restaurant/ parking lots with respective markers.

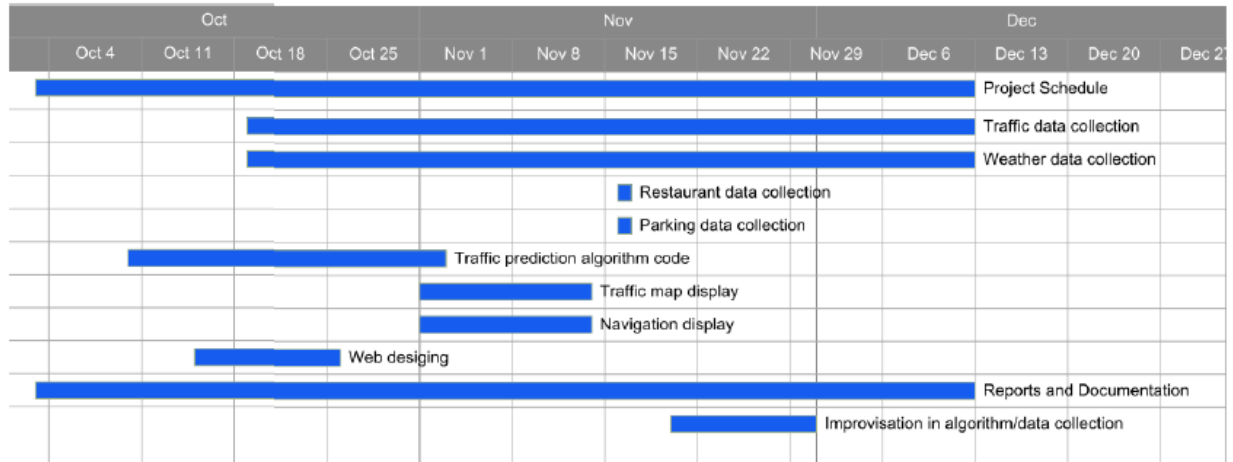
Integration Test: An android application has been integrated to the website. A user in possession of an android device can dump the .apk on the android device and run it to perform the same functions which the web application does. Mobile app is being made, since it is convenient to carry a mobile device. And can be used while traveling. The mobile application can also be tested on eclipse, with an android sdk or with Android studio and an emulator running an android 5.0.1 or higher versions of android.

History of Work & Future Work

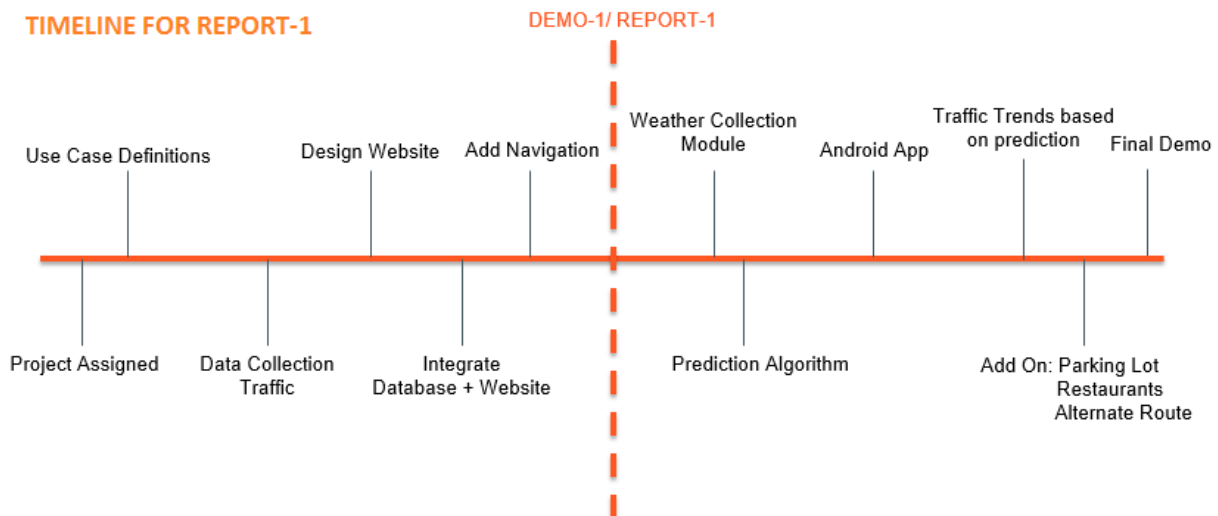
Clearly, the project progress can be divided into two sub-parts – from start to demo-1 and from demo-1 to demo-2

1) Milestones (report-1/demo-1)

What we planned:



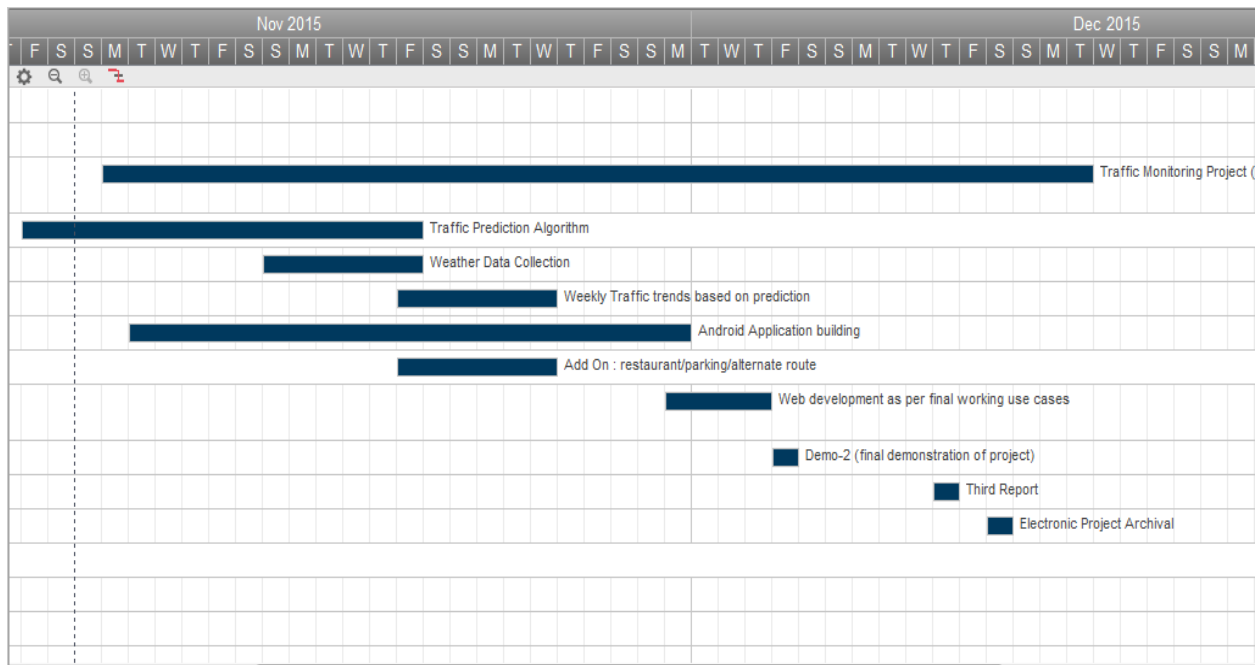
What we Achieved:



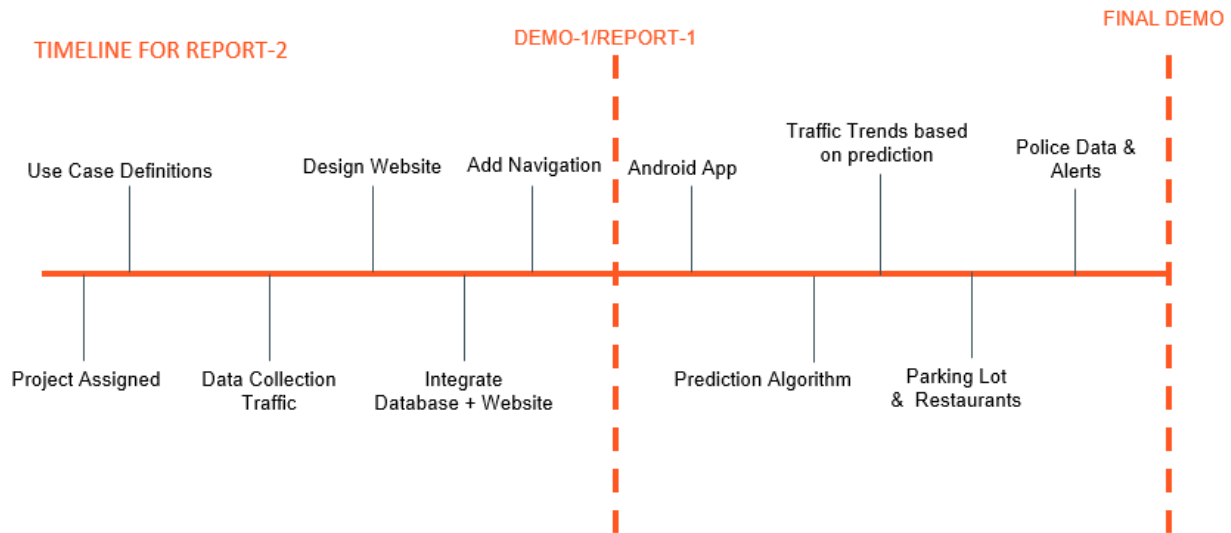
This timeline is taken from our demo presentation and If you compare this with the Gantt chart that we submitted in report-1, (given below), we were able to achieve traffic data collection, website design, navigation and integrate database with website before presenting demo-1. Only thing that we couldn't achieve in this time was developing traffic prediction algorithm.

2) Milestones (report-2/demo-2)

What we planned:



What we Achieved:



From demo-1 to demo-2 period, we changed our data source from 511nj.com to waze.com. We discarded weather information from our scope because it was unnecessary add-on. However, we were able to build algorithm for traffic prediction, android application building, traffic trends based on prediction and several add-ons such as restaurants and parking lot display on map.

Major Accomplishments:

- Collect the data from waze.com, 511nj.com, parkwiz.com, yelp.com
- Design a suitable website using html, CSS and JavaScript.
- Display live traffic incidents – congestion, construction, special events for given zipcode, time and day (511nj.com)

- Display live traffic incidents based on severity level. (waze.com)
- Build a Bayesian prediction algorithm for traffic prediction using historical data for given zipcode, time and day.
- Display weekly traffic trends for user entered zipcode based on prediction model
- Navigate from source to destination and show traffic across route
- Display parking, restaurant and police data from user entered zipcode (parkwiz.com and yelp.com)
- Build Android phone application for displaying live traffic incidents, navigation and police alerts.
- Documentation of report-1, report-2 and report-3 according to Prof. Marsic's requirements.

Future Work:

- Find a reliable and consistent data source which gives free, unlimited traffic data.
- A better prediction algorithm for predicting traffic according to the severity.
- Android Application providing complete features as website.
- Following up software engineering design basics from Prof. Marsic's book.
- Extend the scope of the project from NJ, NY to all over the united states.

References

- [1] Chen, H., Roseman, R. and Rule, A. (2015). *Curve Fitting & Multisensory Integration*.
[online] <http://www.cogsci.ucsd.edu/>. Available at:
http://www.cogsci.ucsd.edu/~ajyu/Teaching/Cogs202_sp14/Slides/lect3.pdf [Accessed 15 Oct. 2015].
- [2] Creately.com, (2015). *Online Diagram Software to draw Flowcharts, UML & more / Creately*. [online] Available at: <http://creately.com> [Accessed 15 Oct. 2015].
- [3] Dev.virtualearth.net, (n.d.). [online] Available at:
http://dev.virtualearth.net/REST/v1/Locations?postalCode=07657&key=AvITzUaX5nqGg0JKU43QSP30BB_ovX37MIND9WahtRbBQpeV4ugoB3HQ60LRxQL8&o=xml
[Accessed Oct. 2015].
- [4] GenMyModel, (2015). *Software Modeling in the Cloud*. [online] Available at:
<http://genmymodel.com> [Accessed 16 Oct. 2015].
- [5] <https://developers.google.com/maps/?hl=en>, (n.d.). *Google Maps API*. [online] Available at:
<http://google.com> [Accessed Oct. 2015].
- [6] Marsic, I. (2012). *Software Engineering*.
- [7] Msdn.microsoft.com, (n.d.). *Find a Location by Address*. [online] Available at:
<https://msdn.microsoft.com/en-us/library/ff701714.aspx> [Accessed Oct. 2015].
- [8] Msdn.microsoft.com, (n.d.). *Get Traffic Incidents*. [online] Available at:
<https://msdn.microsoft.com/en-us/library/hh441726.aspx> [Accessed Oct. 2015].
- [9] Msdn.microsoft.com, (n.d.). *Traffic Incident Data*. [online] Available at:
<https://msdn.microsoft.com/en-us/library/hh441730.aspx> [Accessed Oct. 2015].
- [10] Openweathermap.org, (n.d.). *OpenWeatherMap current weather and forecast*. [online] Available at: <http://openweathermap.org> [Accessed 4 Oct. 2015].
- [11] Project Report Spring 2011, (2011). *Project: Traffic Monitoring*. [online] Available at:
<http://www.ece.rutgers.edu/~marsic/books/SE/projects/Traffic/2011-g7-report3.pdf>
[Accessed 2015].

- [12] Project Report Spring 2013, (2013). *Traffic Monitoring Service*. [online] Available at: <http://www.ece.rutgers.edu/~marsic/books/SE/projects/Traffic/2013-g7-report3.pdf> [Accessed 2015].
- [13] Smartsheet, (n.d.). *Online Project Management Software / Smartsheet*. [online] Available at: <http://smartsheet.com> [Accessed 15 Oct. 2015].
- [14] Stackoverflow.com, (n.d.). *Stack Overflow*. [online] Available at: <http://stackoverflow.com/questions/33069298/bing-traffic-api-not-returning-any-data-for-accidents-delays> [Accessed 2 Oct. 2015].
- [15] W3schools.com, (2015). *W3Schools Online Web Tutorials*. [online] Available at: <http://w3schools.com> [Accessed 10 Oct. 2015].
- [16] Websequencediagrams.com, (n.d.). *WebSequenceDiagrams.com - Draw and Edit Sequence Diagrams in seconds*. [online] Available at: <http://websequencediagrams.com> [Accessed 8 Oct. 2015].
- [17] Wikipedia, (2015). *Curve fitting*. [online] Available at: https://en.wikipedia.org/wiki/Curve_fitting [Accessed 15 Oct. 2015].
- [18] Youtube.com, (n.d.). *YouTube*. [online] Available at: <http://youtube.com> [Accessed 7 Oct. 2015].