Software Engineering 332:452

Traffic Monitoring Project
Group 13 – March 15, 2015
Full Report #2

http://justincoding.com/traffics.html

Maxine Dienes

Charu Jain

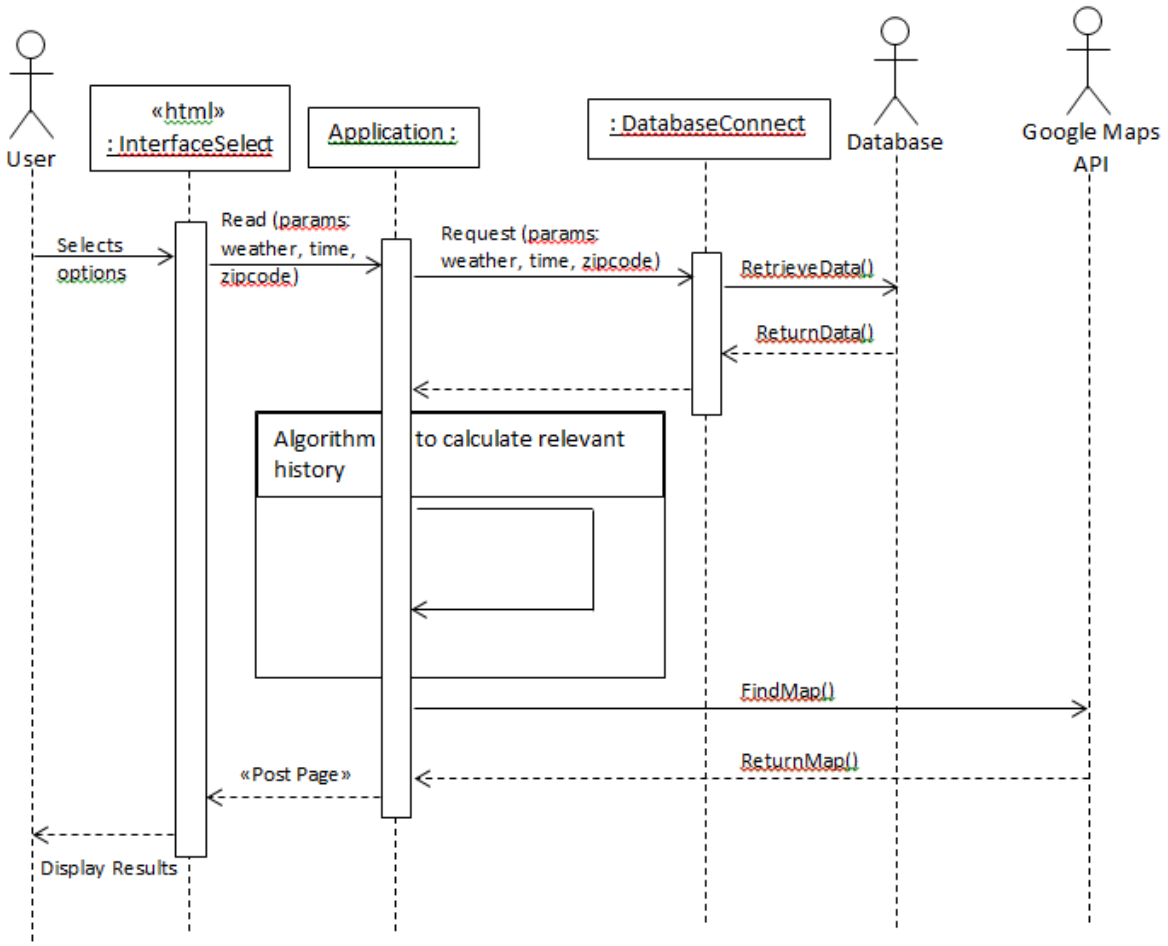
Mehul Salhotra

Akshay Sardana

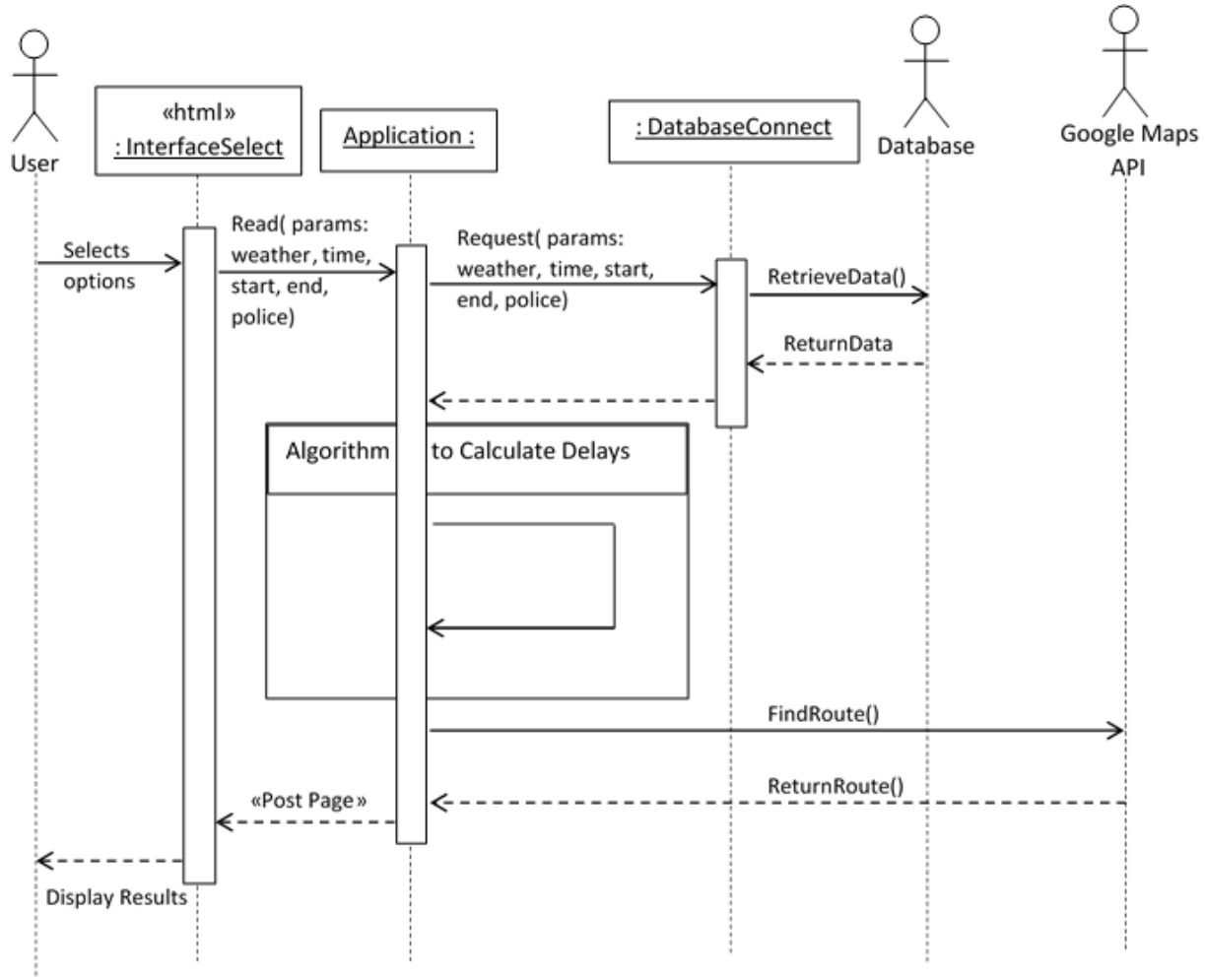
Justin Silang

Jason Yang

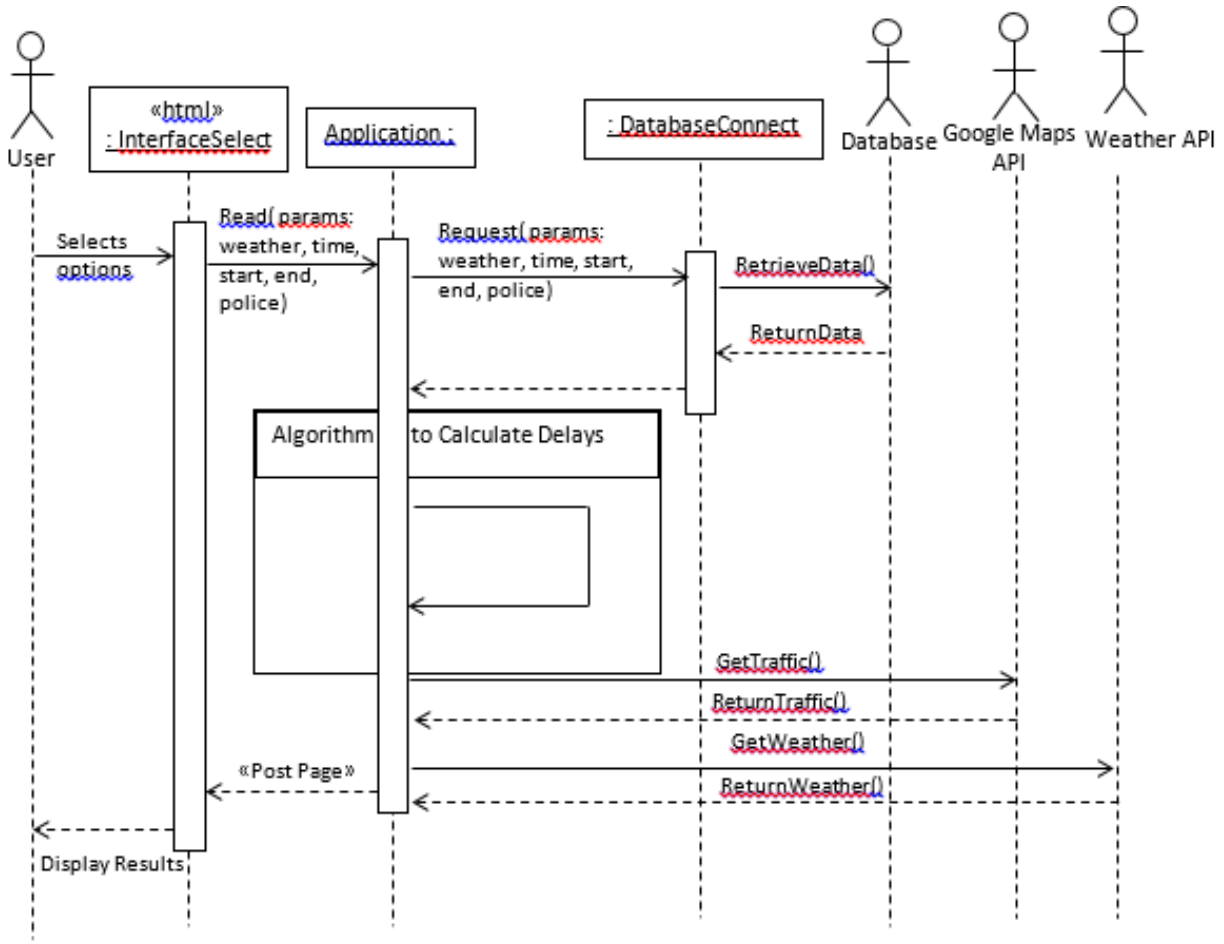
Use Case 1


User
«html»
: InterfaceSelect
Application :
: DatabaseConnect
Database
Google Maps
API
Selects
options
Read (params:
weather, time,
zipcode)
Request (params:
weather, time, zipcode)
RetrieveData()
ReturnData()
Algorithm
history
to calculate relevant
FindMap()
ReturnMap()
«Post Page»
Display Results

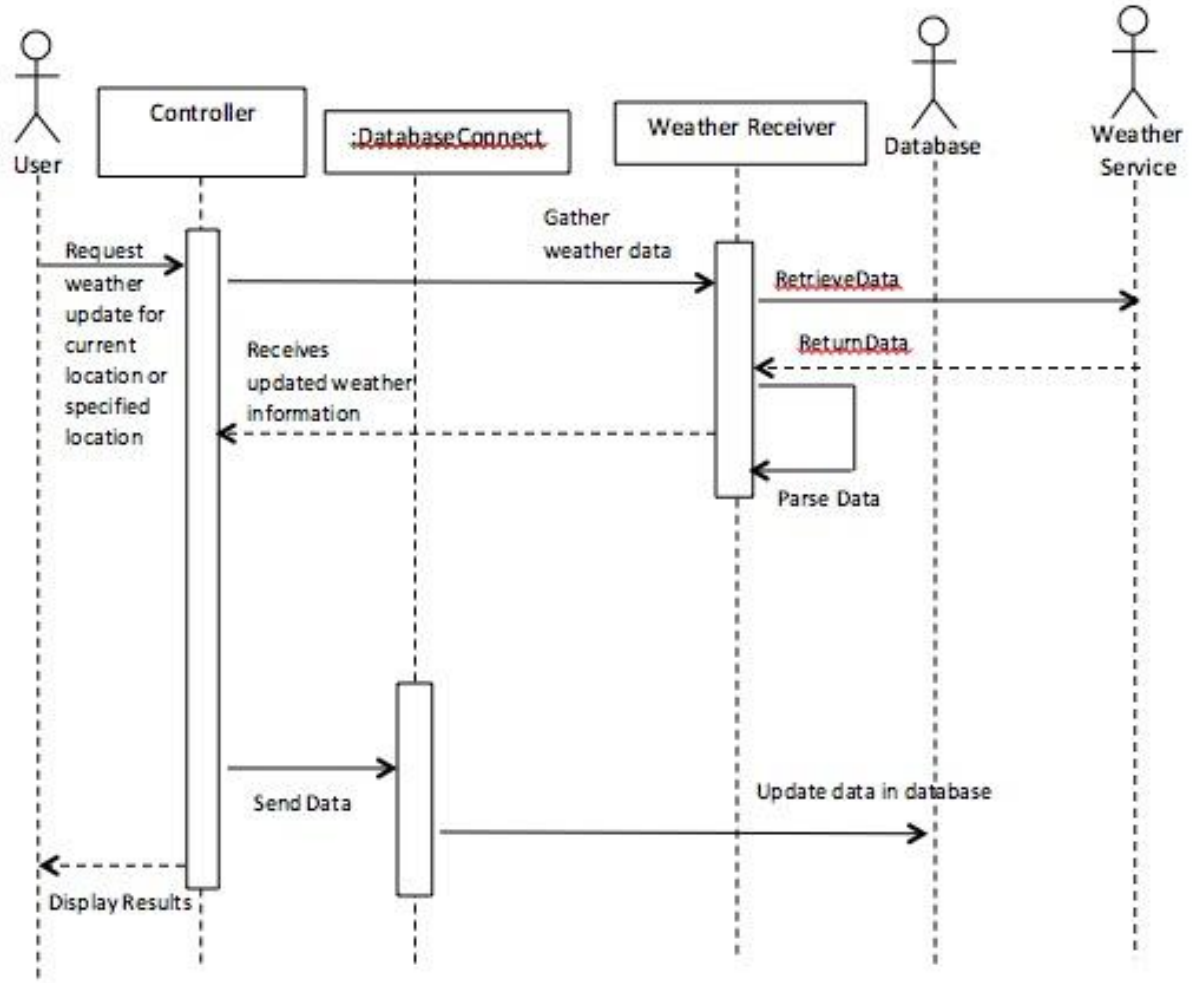
Use Case 2


User
«html»
: InterfaceSelect
Application :
: DatabaseConnect
Database
Google Maps
API
Selects
options
Read(params:
weather, time,
start, end,
police)
Request(params:
weather, time, start,
end, police)
RetrieveData()
ReturnData
Algorithm
to Calculate Delays
FindRoute()
ReturnRoute()
«Post Page»
Display Results

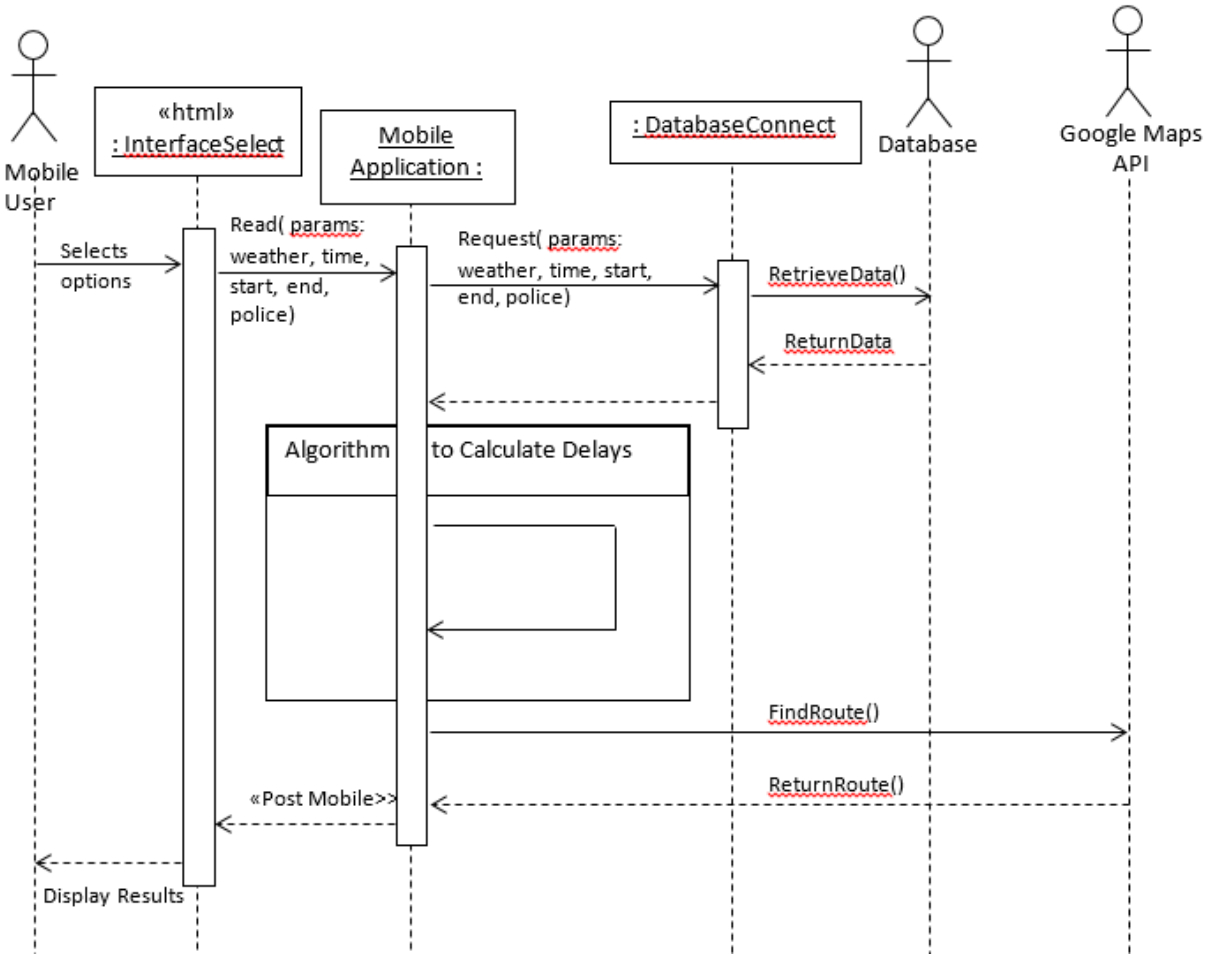
Use Case 4


User
«html»
: InterfaceSelect
Application :
: DatabaseConnect
Database
Google Maps API
Weather API
Selects options
Read(params: weather, time, start, end, police)
Request(params: weather, time, start, end, police)
RetrieveData()
ReturnData
Algorithm to Calculate Delays
GetTraffic()
ReturnTraffic()
GetWeather()
ReturnWeather()
«Post Page»
Display Results

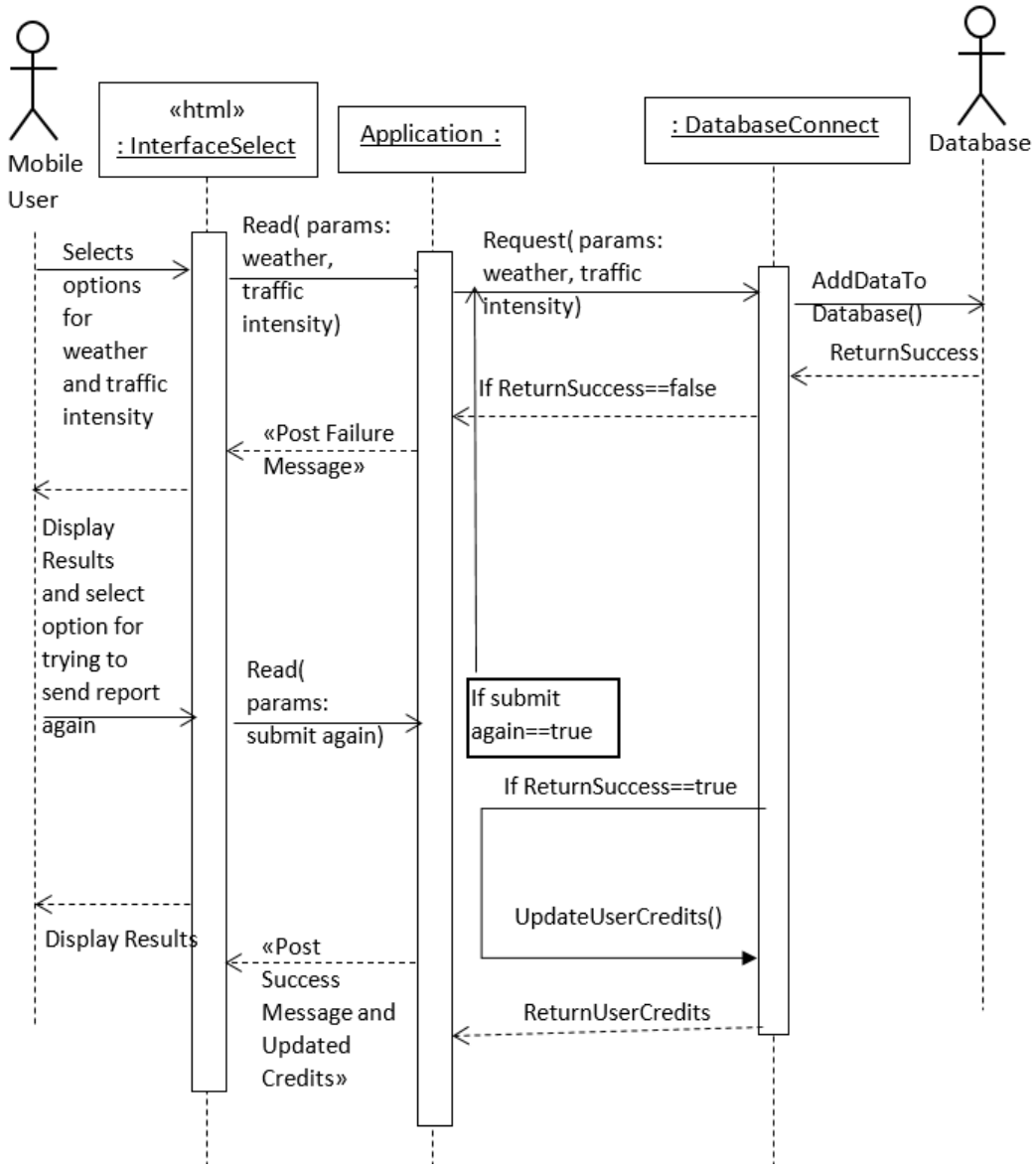
Use Case 5


User
Controller
:DatabaseConnect
Weather Receiver
Database
Weather Service
Request weather update for current location or specified location
Gather weather data
RetrieveData
ReturnData
Receives updated weather information
Parse Data
Send Data
Update data in database
Display Results

Use Case 6

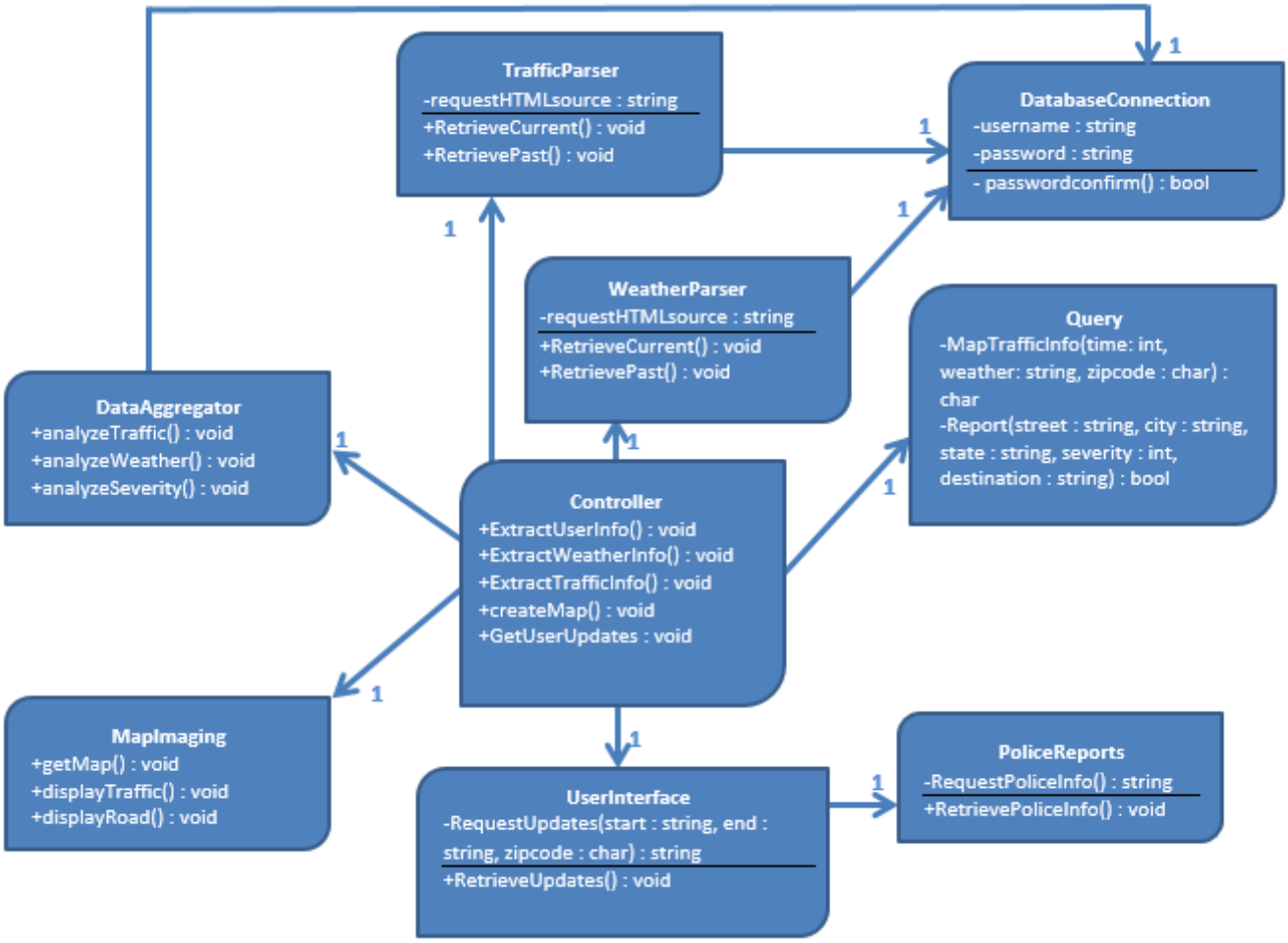

Mobile User
«html» : InterfaceSelect
Mobile Application :
: DatabaseConnect
Database
Google Maps API
Selects options
Read(params: weather, time, start, end, police)
Request(params: weather, time, start, end, police)
RetrieveData()
ReturnData
Algorithm to Calculate Delays
FindRoute()
ReturnRoute()
«Post Mobile»
Display Results

Use Case 9



Project Management: Each Group Member Submitted A Single Use Case

Class Diagram



Data Types and Operation Signatures

WeatherParser
-requestHTMLsource : string -htmltag: string -count: int -list: string -location: string -weather: string -precipitation: string -temperature: double -forecast: string -occurrencepossibility: int *(percentage chance it is going to be certain weather i.e. 10% chance precipitation)* -sockets: http socket
+RetrieveCurrent() : void +RetrievePast() : void

TrafficParser
-requesthtmlsource : string -htmltag: string -count: int -incidents: string *(amount of car accidents/incidents that happened)* -latitude: double -longitude: double -road: string -sockets: http socket
+retrieveCurrent() : void +retrievePast() : void

Both parsers retrieve and parse data, one in regards to traffic, and the other for the weather. There are several different attributes and operations, but they seem pretty self-explanatory. Yet, the ones that are more opaque are defined.

Query
-time: int -weather: string -zipcode: char -street: string -city: string -state: string -severity: int -destination: string
-mapTrafficInfo() : char -report() : bool

UserInterface
-start: string -end: string -zipcode: char
+retrieveUpdates() : void -requestUpdates() : string

DatabaseConnection
-username : string -password : string
-passwordconfirm() : bool

To access the database, the username and password much match.

MapImaging
-startlocation : string **-endlocation: string** **-routetype: string** ***(i.e. scenic route, fastest route, least use of highways)***
+getMap() : void **+displayTraffic() : void** **+displayRoad() : void**

MapImaging class is used for displaying the route. At this time, we are unsure about the various attributes, but know that some are more likely to be used than others. Therefore, we have included what we foresee to be necessary for this class.

Controller
To be determined when implementation begins
+ExtractUserInfo() : void **+ExtractWeatherInfo() : void** **+ExtractTrafficInfo() : void** **+createMap() : void** **+GetUserUpdates : void**

DataAggregator
-trafficintensity: float **-weather: string** **-time: int**
+analyzeTraffic() : void **+analyzeWeather() : void** **+analyzeSeverity() : void**

PoliceReports
To be determined when implementation begins
+retrievePoliceInfo() : void **-requestPoliceInfo(): string**

At this time, these are the attributes and operations we believe are going to be utilized for these various classes. If we feel any desire to improve the functionality, this portion will be updated in the future, along with any changes we made in the process. We have not begun implementation, thus there may be room for error. Yet, when these classes have been made, we will have the correct and updated class specifications available.

Traceability Matrix

	Class								
Domain Concepts	MapIm-aging	DataAggr-egator	UserInt-erface	Database Connecti-on	Weather Parser	Traffic Parser	Police Report	Query	Contro-ller
Application	X	X	X	X	X	X	X	X	X
Database		X		X	X	X	X	X	
Mapping Service	X	X						X	X
Directions Service		X				X		X	X
Police Service		X					X	X	X
Weather Service		X			X			X	X
Social Media/Music		X							
Mobile Application									

The classes for our system are listed across the top of the graph, while the domain concepts are listed along the left. From the domain concepts, we obtained the derived classes. First, the Mobile Application domain concept does not map out to any class, because its implementation is largely the same as the main application. Therefore, it is redundant to map the concept to other classes. Also, the Social Media/Music concept maps to very few classes, since it is an extra feature that will receive lower priority than the more critical components of the system. In contract, the main application maps to all classes since it is involved with each of them.

The DataAggregator class serves mainly to gather various types of data (such as weather, traffic and police data), and distribute it wherever it sees fit. This means that it will be involved with all of the main domain concepts, as it will usually act as an intermediate step between most processes. For each of the WeatherParser, TrafficParser, and PoliceReport classes, all three are derived from their respective services in the domain concepts. They also map to databases, because the above classes interact with the database by storing their respective data inside it.

Like the DataAggregator, the Controller and Query both are derived from most of the primary domain concepts. The query class will query the database for data regarding traffic, police data, and weather, so it will be derived from those classes. And the Controller acts as the main hub, delegating different tasks for each module, meaning that it is involved with many domain concepts. Finally, since the DatabaseConnection only acts as a channel to the database, it is soley derived from Database.

Architectural Styles

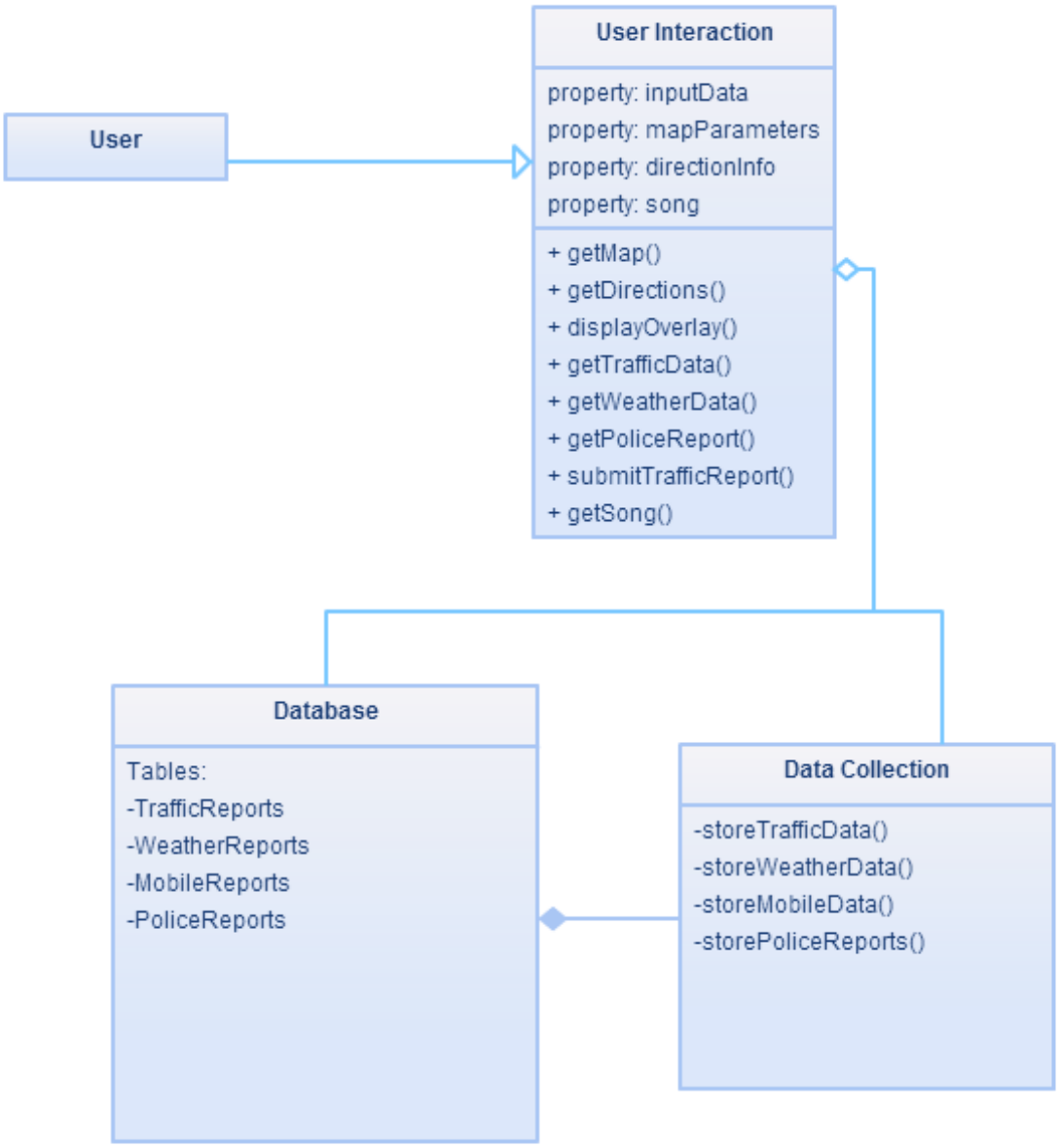
This system uses the Client/Server style as its main architectural style. The system is separated into two distinct areas. On the client side of the system, the user interacts with the interface to send the input information and view the output traffic information. On the server side of the system, the system performs functions on data based on input parameters and outputs data that can be used to for traffic information. The client side sends input requirements over to the server side, andrequests this output information from the server side, and this is the only capacity in which the client side and server side will interact. That way, the data that the client side receives will only be the data necessary for the user in that given instance, and the server side would not have to deal with changing inputs from the client side before they submit their request.

Identifying Subsystems

The first subsystem is the User Interaction subsystem. The user accesses the system through this subsystem on the website or the Android application, and the user will only interact with the system through this subsystem. It contains the display, the user's input data, the map output, the directions output to accompany the map output, the traffic report submission for users who want to submit their current traffic information, and the song output for those using the music functionality. This subsystem is an aggregation of the Database subsystem, as the database only works through the user interaction subsystem but the database also exists regardless of the current user interaction.

The second subsystem is the Database subsystem, which contains the traffic, weather, mobile, and police data gathered for the User Interaction subsystem to get and work with for desired user output.

The third subsystem is the Data collection subsystem, which contains the functions that store the data to the database from outside sources like the User Interaction system or relavant web services. It is a composition of the database because these functions only exist with the existance of the database.


User
User Interaction
property: inputData
property: mapParameters
property: directionInfo
property: song
+ getMap()
+ getDirections()
+ displayOverlay()
+ getTrafficData()
+ getWeatherData()
+ getPoliceReport()
+ submitTrafficReport()
+ getSong()
Database
Tables:
-TrafficReports
-WeatherReports
-MobileReports
-PoliceReports
Data Collection
-storeTrafficData()
-storeWeatherData()
-storeMobileData()
-storePoliceReports()

Mapping Subsystems to Hardware

Subsystems will be on the same computer.

Persistent Data Storage

MYSQL controls the database storage. There are four tables that the database stores: Traffic, Weather, MobileReports, and PoliceReports.

Database Tables

Field	*Type*	*NULL*	*Default*
Traffic Display			
Index	AUTO_INCREMENT	No	0
Date_Time	Timestamp	No	00-00-0000 (Date) 00:00:00 (Time)
Zipcode	Varchar(5)	No	N/A
Latitude	Decimal(0, 0)	No	0.00
Longitude	Decimal(0, 0)	No	0.00
Traffic Intensity	int	No	0
Address	Varchar(100)	Yes	N/A
County	Varchar(20)	No	N/A
Weather	Varchar(20)	Yes	N/A
Time_Value	Int	No	0
Weather			
Index	AUTO_INCREMENT	No	0
Timestamp	Timestamp	Yes	00-00-0000 (Date) 00:00:00 (Time)
Zipcode	Varchar(5)	No	N/A
Condition	Varchar(20)	No	N/A
Police Report			
Index	AUTO_INCREMENT	No	0
Date_Time	Timestamp	No	00-00-0000 (Date) 00:00:00 (Time)

Zipcode	Varchar(5)	No	N/A
Latitude	Decimal(0, 0)	No	0.00
Longitude	Decimal(0, 0)	No	0.00
Short_Descrip	Varchar(200)	No	N/A
Mobile Report			
Index	AUTO_INCREMENT	No	0
Date_Time	Timestamp	Yes	00-00-0000 (Date) 00:00:00 (Time)
Zipcode	Varchar(5)	No	N/A
Latitude	Decimal(0, 0)	No	0.00
Longitude	Decimal(0, 0)	No	0.00
Severity	int	No	0
Address	Varchar(100)	Yes	N/A
County	Varchar(20)	No	N/A
State	Varchar(20)	No	N/A
Short_Descrip	Varchar(200)	No	0

Network Protocol

The network protocol will be HTTP. The traffic system service will be accessed through web. Using HTTP, this will allow us to get a high amount of users.

Global Control Flow

Execution Orderness

As a navigation device our system will always have procedure and events driven attributes. The procedure-driven aspect of our system is that a user will always have the ability to go through the standard setup of manually entering a location and receiving traffic and mapping data. The system is events-driven because it automatically updates its mapping service based upon traffic and weather.

Time Dependency

There are no time dependencies on our system. Our navigation devices puts no limits on our user, they can enter new destination points at any time. The only timer is on our weather updating service, which only pulls new data every 30 minutes. This is because more often than not, weather services do not update their weather information as frequently; it is usually in intervals of an hour.

Concurrency

Our system does use multiple threads. One thread strictly deals with the interactions between the users and the application. Another thread handles the interaction between the application and web services, which includes weather and traffic information.

Hardware Requirements

The user must have Java and Flash installed to access the web application. Our application has been optimized for a 16:9 display ratio and can adjust to the popular screen resolutions.

The android application only requires 50 MB of space since the database stores all traffic and weather related information. The android phone must be able to access the Internet and preferably have a GPS chip.

Algorithms and Data Structures

The algorithm for traffic severity relies on Bing or Mapquest's severity value.

For a more clear and concise view of traffic history, we developed our own algorithm to clean up the traffic data. This allows for less clutter needed to be displayed on a map and can give the User easier map readability. Our database contains a weather table which contains the weather at certain times for all supported zipcodes. There second table is traffic table that contains all the traffic incidents with time and position of the incident. The third table is the traffic incident table that has the condensed traffic incidents and used to display. The algorithm takes all traffic incidents of the day and

constructs a hash of incidents with position as a key and the severity as the value. The hash is then used to get all incident points along a certain road at the same time with the same weather. The algorithm also creates the same hash from the condensed traffic incident table. The hashes are compared by street. Along that same street traffic incidents of the day are added to the closest point within .5 miles. If there are no points within .5 miles that incident location becomes a new location to be displayed in the condensed traffic incident table. The severity values are updated each time a traffic incident is added to a condensed traffic incident point.

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

The algorithm to determine the route for directions uses the condensed traffic points. We have the start and end points. Using the start and end points, a bounded box is formed. All points within that box that have high traffic intensities are then found in a list. Using the condensed traffic points we can construct a call to the Directions Service. In that call to Directions Service, we can indicate points between the start and end points that the route should avoid. We weight the points to be avoided based on the condensed traffic point's severity value. No complex data structures are used in this project. The majority of the complexity deals with manipulation of the data stored in the database.

User Interface Design and Implementation

Minimal changes have been made to modify the initial screen mock-ups, as these designs were minimalistic and straightforward in order to reduce user effort.

Using HTML, CSS, and Javascript for the webpage, and Android development tools for the phone application, we were able to produce the interfaces as described in the initial screen mock-ups for the system.

Design of Test Cases

Our test cases will test the key features of our navigation system. Multiple test cases will serve different purposes. One test case will require individual inputs of every property. The second test case will not require a single input, it will use location services to find the zipcode, use the computers clock to get the time, and automatically pull the current weather from a weather service.

Test Case 1:

Inputs:
Starting Address: 604 Bartholomew Road, Piscataway, NJ 08854
Destination Address: 43 College Ave, New Brunswick, NJ 08901
Time of Travel: 3:15 PM
Weather: Sunny
Ideal Output: Mapped directions with estimated arrival time of 6 minutes

Test Case 2:
Inputs:
Starting Address: taken using location services
Destination Address: can be anything
Time of Travel: current time taken from device
Weather: pulled from weather services from starting address
Ideal Output: Mapped directions with accurate arrival time

One of the key differences between the mobile application and the website is that there will be additional integration on the mobile application. Aside from that, both applications should be able to pass the same test cases.

In regards to integration testing, we will focus on going from the nitty gritty details to the overall output. For example, we will start by connecting the source code to the website and application we have developed. From there, we will test both applications to compare and contrast the expected output from the given output. In order to test whether the system can identify and handle incorrect values we will input incorrect initialization values and see how the system reacts.

Firstly, we will focus on integration testing between the application and the database. Our database should have a quantifiable amount of information on traffic data which the application will call on for various different reasons. Thus, we will have

to test to see whether the application is capable of determining the type of traffic data that the database contains. Also, we will check to see if the application will ignore non-existent or null values in the Database tables.

We will also make sure to check whether the requirements we determined previously will be properly and professionally implemented in the source code. This sort of testing is known as requirement testing and is a great preliminary method for us to use as a baseline check. For example, we will mainly look to compare the algorithm that determines the user's different routes in close detail. As this algorithm will have to take in various values from the Database and properly intake, process, and output to the user. Therefore, we will have to test to see whether these three attributes have been properly accessed. Also, we will confirm that the website we have developed has the proper functionality and image has the team has envisioned. Therefore we will do some GUI testing to see if the icons we have put on the website and different subpages are in the proper order.

Police Monitoring is also another important feature of our application. This can be tested by adding a police report via the mobile application and then verifying its existence in the universal database.

Project Management and Plan of Work

Merging the contributions from Individual Team Members

Everyone in this group has contributed equally; we ensure this to keep a positive environment and to keep the project moving forward. We all work towards the same format so that when the time comes to create a full report, formatting and consistency are not issues. We work towards the most modern look possible so that the reports do not seem dry and boring. We have successfully achieved a simple compilation of the final copy by heavy communication so that every member is aware of their task and how to implement it.

As with any project, issues do arise. One issue that we dealt with at first was the distribution of the workload, which we quickly fixed so that no one member was doing too much work – we strive for everyone to contribute equally. Another was the scheduling of group meetings to discuss programming implementations and project ideas. As engineers it is very hard to find time for everyone to be available, as everyone has extracurricular activities, jobs, and studying to do. We found that the best way for

everyone to interact was through Google Hangouts, Google Docs and GroupMe messaging. These methods of communication and report implementation allow for every member to commit a lot of time during the day, however at their own convenience. We were all able to discuss the report breakdowns, any questions at hand, and write the report whenever we had time during the day. This solution has led to a much better group dynamic and overall improved awareness of the project at hand.

Project Coordination and Progress Report

We are first working on implementing use case 1 (TrafficAccount) and use case 2 (GetDirections) as a whole as they are the basis of this project. We plan to have it completely implemented by the end of Spring Break, as we are working out some kinks with the user interface. We also plan to implement the MapService use case very soon so that we can extract the information from the map API's, something that affects a large portion of the project. Once we can get the User interface to work efficiently, we will include the ReportTraffic and ReportPoliceActivity use cases. In a more project management perspective, we are a little behind schedule, as the plan of work we had from the last report may have been much tighter than we realized. We are bearing in mind all new dates and deadlines that we have set and are working towards achieving all of them. Currently, we are using a GitHub account that has shared access for each member. We are using this to communally collaborate on code for the project.

Plan of Work

Breakdown of Responsibilities

As of right now, we have not broken down the responsibilities for developing, coding, and testing different modules and classes. As a group we are trying to move forward with setting up the correct environments to work with. We also plan on working on a lot of the different components of this project together, so we do not rely on separate pieces done by individual team members; we think this will help the execution of the project. Once we have broken down the team members that are *mostly* working on a single piece, we will have them perform the integration testing for that piece.

References

511 Traffic Incidents:
http://www.511nj.org/

Google Maps:
https://maps.google.com

“Software Engineering” by Ivan Marsic:
http://www.ece.rutgers.edu/~marsic/books/SE/

“The Costs of Highway Congestion” by David Morgan
http://abcnews.go.com/US/story?id=94064

Traffic.com:
https://www.here.com/traffic

Voice activation for android application:
http://developer.android.com/reference/android/speech/RecognizerIntent.html

Weather.com RSS Feed:
http://www.weather.com/

MapQuest API:
http://developer.mapquest.com/

Bing Traffic API:
https://msdn.microsoft.com/en-us/library/hh441725.aspx