Traffic Monitoring Service

Group No. 7

Report 2

Team Members

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

Instructor: Prof. Ivan Marsic

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

Revision History:

Version No.	**Date of Revision**
v.1	03/03/2013
v.2	10/03/2013

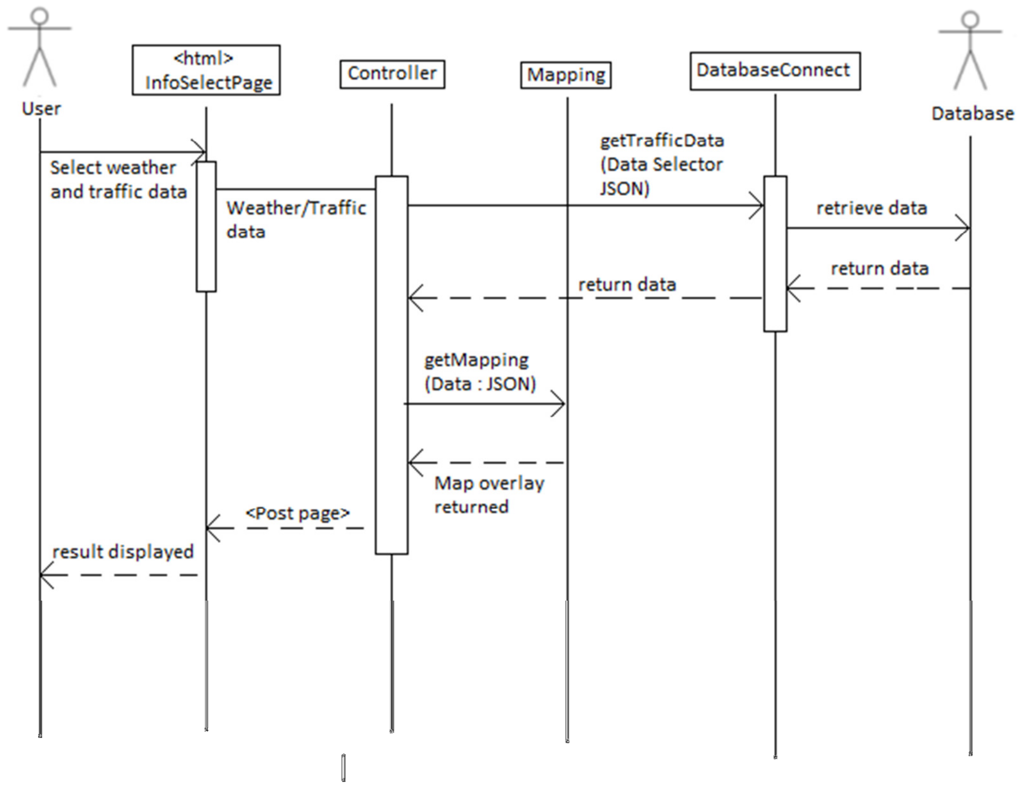
Breakdown of Contributions

All members contributed equally for this report.

Table of Contents

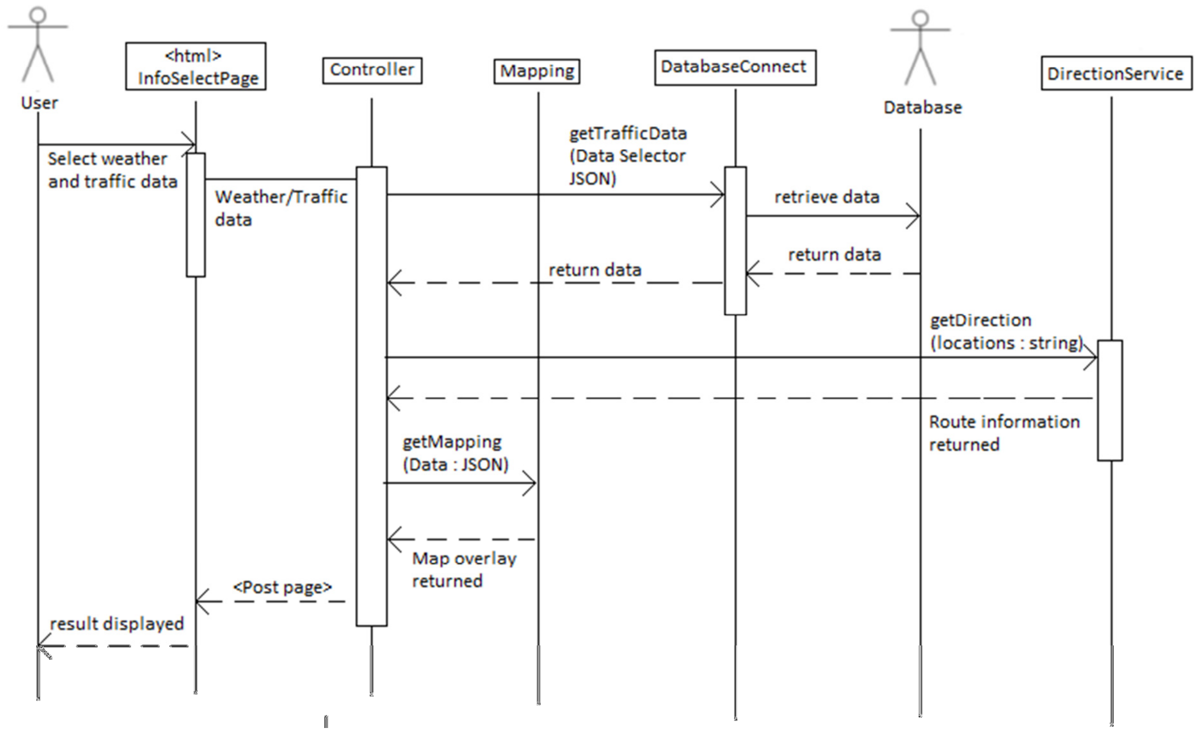
Interaction Diagrams

Use Case 1:



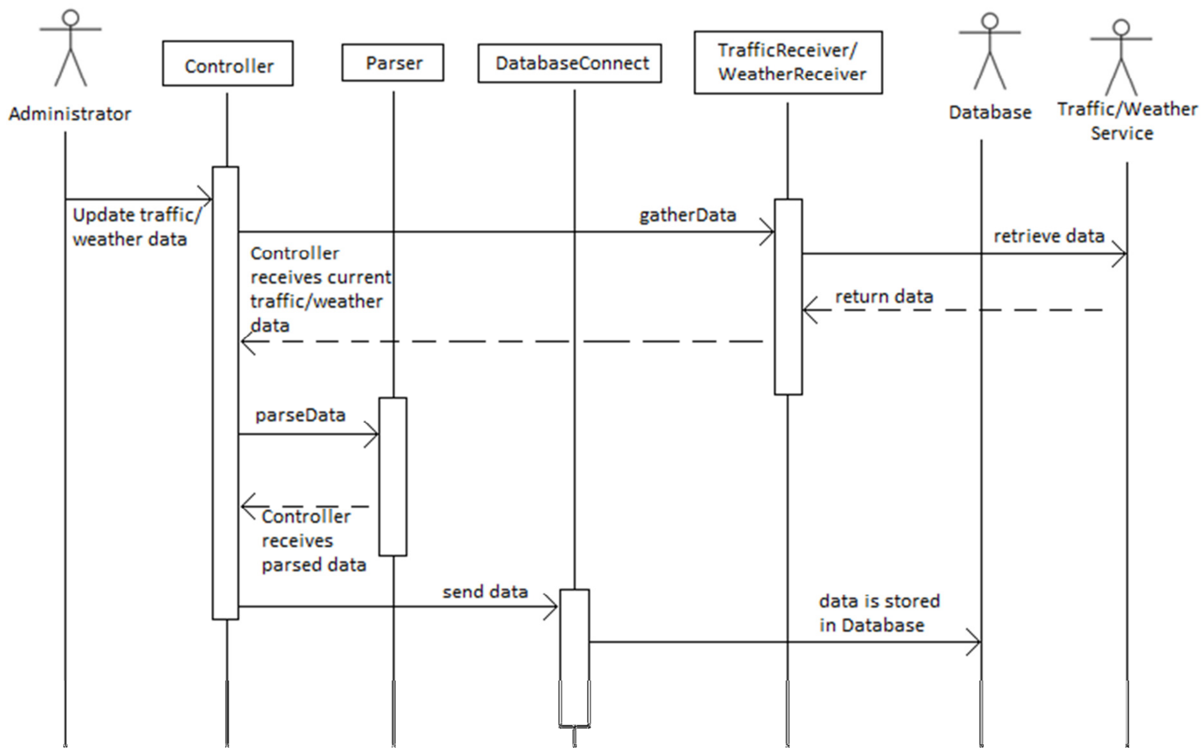
This use case has the responsibility of analyzing the inputs entered by the user to show the relevant traffic history for the area they desire. The web Application receives the data entered by the user, and using the Expert Doer principle, sends the data to the Controller. Using the High Cohesion principle, DatabaseConnect is used as an intermediate between the controller and the database. This ensures that the controller does not do too many computations when attempting to access data from the Database. The Database then returns the information through DatabaseConnect back to the Controller. The Controller sends the Database information to the Mapping service, and it returns a map overlay with the information posted. The Controller posts the map on the web Application, and the information is made available for the user.

Use Case 2:



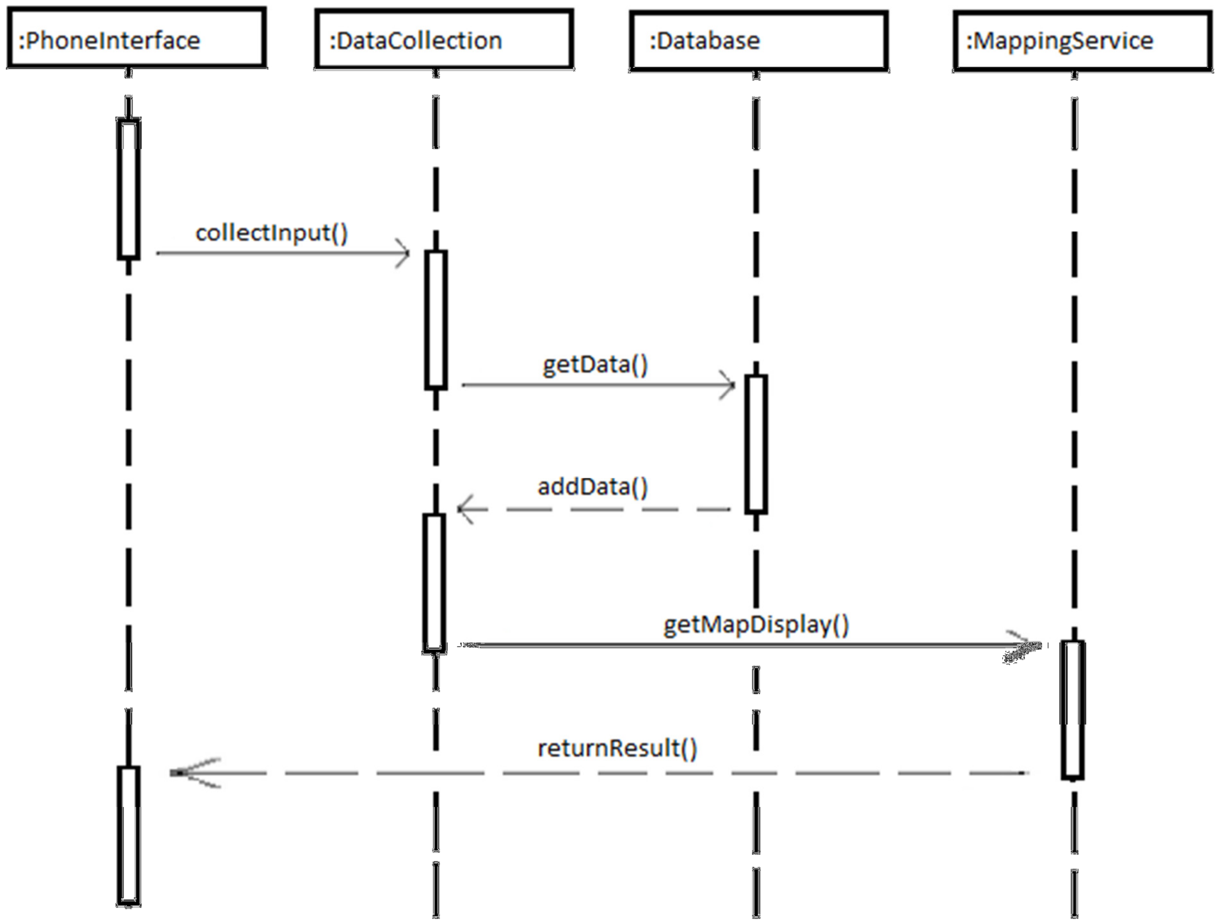
The methodology for the second use case is similar to that of Use Case 1. It employs the Expert Doer principle for the information sent, and it uses the High Cohesion principle in the form of DatabaseConnect. The difference from the first use case is the addition of a DirectionService. After the Database information is sent back to the Controller, the Controller then accesses the DirectionService to get the route information desired by the user. A list of directions is sent back to the Controller, and the Controller then adds the list of the directions to the object it sends to the Mapping service. The Mapping service returns the overlay of traffic history along with a route the user can follow to reach his or her destination. This information is sent to the web Application so that it can be viewed by the user.

Use Case 4 and 5:



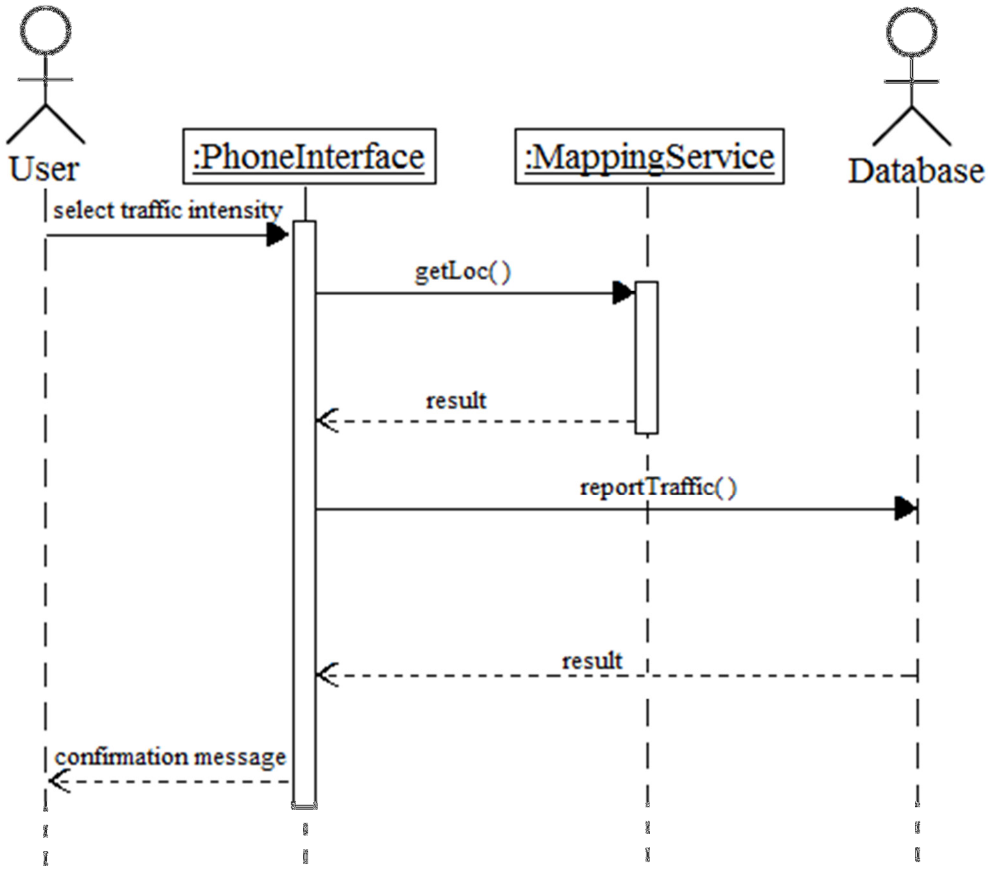
These use cases have the responsibility of accessing the Traffic and Weather Service websites to retrieve data at constant intervals. At every time interval, the Controller is told to update the traffic and weather data. Through the High Cohesion principle, the Controller sends for the TrafficReceiver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

Use Case 6:



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

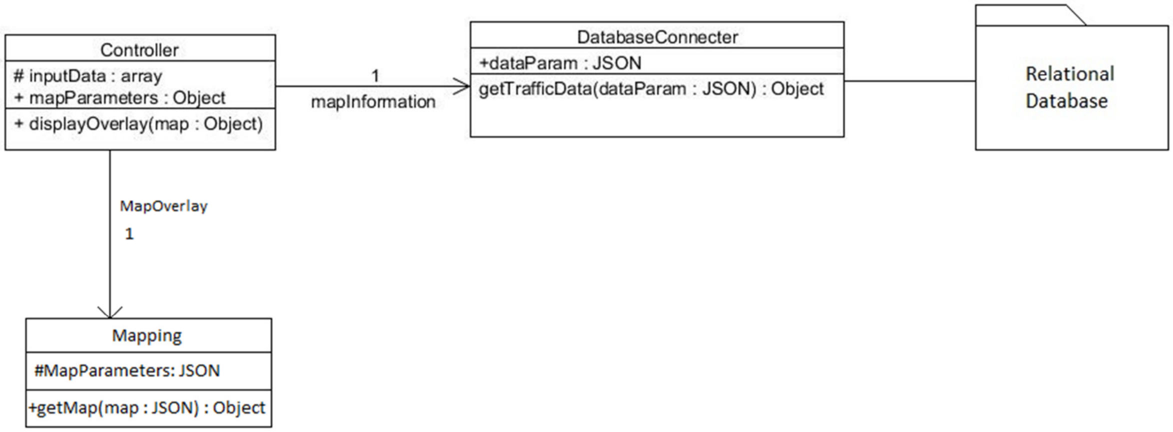
Use Case 7:



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

Class Diagram and Interface Specification

Use Case 1:

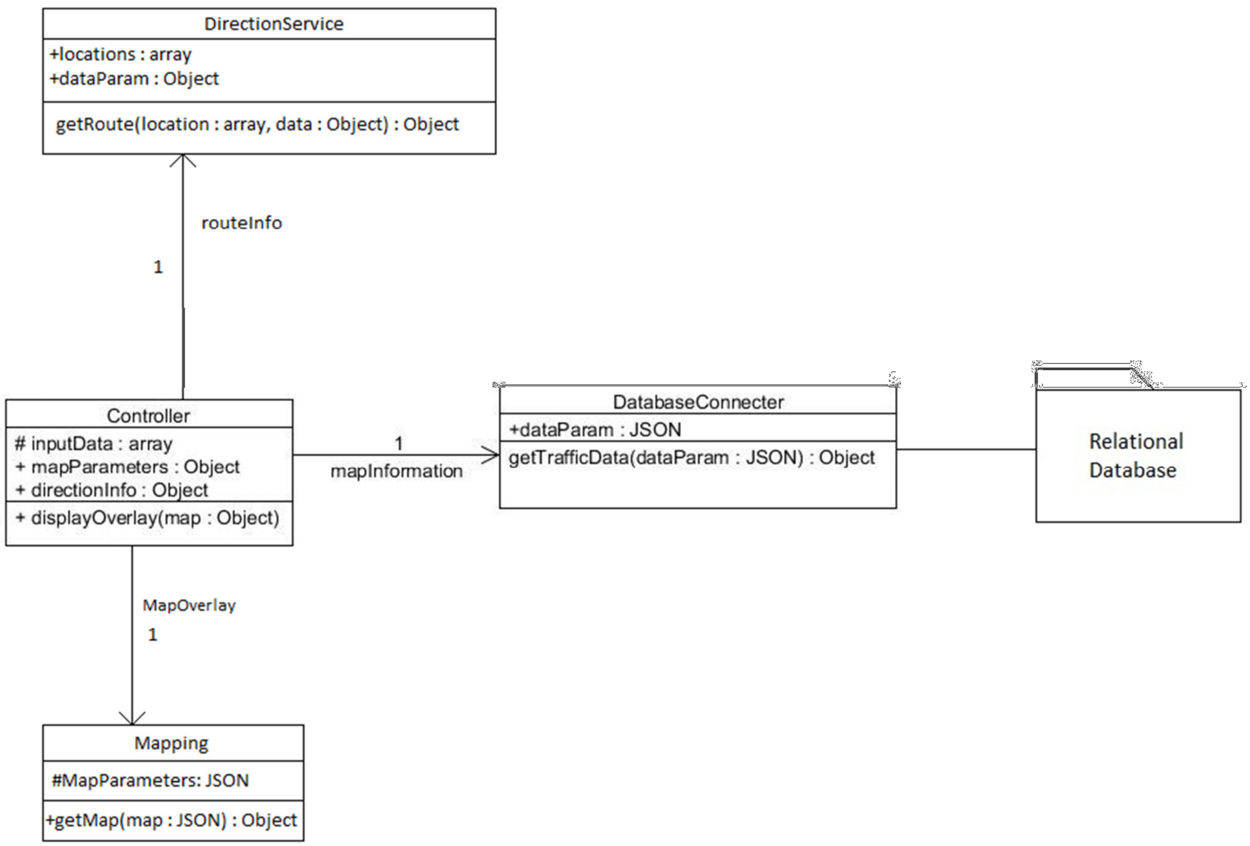


The controller class is the class that calls other classes to be executed and holds the data from the website. The controller will hold data sent from the website in the array of inputData which could become a hash if it is more convenient. The displayOverlay operation will use the object mapParameters and display the contained information to display an overlap.

The mapping class will get parameters to show the traffic information and return an object of that information. The method getMap will use mapParameters and get the mapping overlay based on the parameters.

The DatabaseConnector class will get the website input data and get the traffic information based on the data. The getTrafficData will get the traffic information and return an object of the data.

Use Case 2:



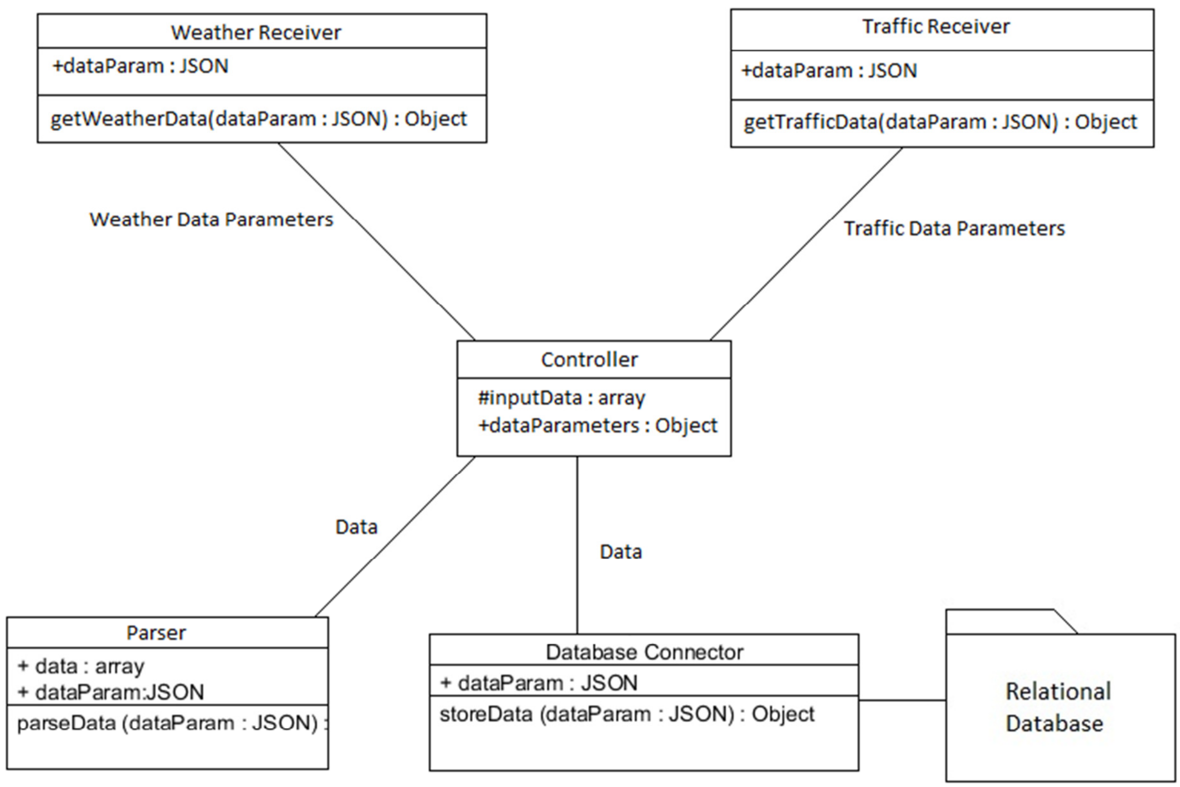
The controller class is the class that calls other classes to be executed and holds the data from the website. The controller will hold data sent from the website in the array of inputData which could become a hash if it is more convenient. It will have a directionInfo attribute to hold the route information. The displayOverlay operation will use the object mapParameters and directionInfo and display the contained information to display an overlap.

The mapping class will get parameters to show the traffic information and return an object of that information. The method getMap will use mapParameters and get the mapping overlay based on the parameters.

The DatabaseConnector class will get the website input data and get the traffic information based on the data. The getTrafficData will get the traffic information and return an object of the data.

The DirectionService class is to get the optimal route based on the start and end locations. The locations attribute will those values. The getRoute method will use an algorithm based on the traffic information to calculate a route.

Use Case 4,5:

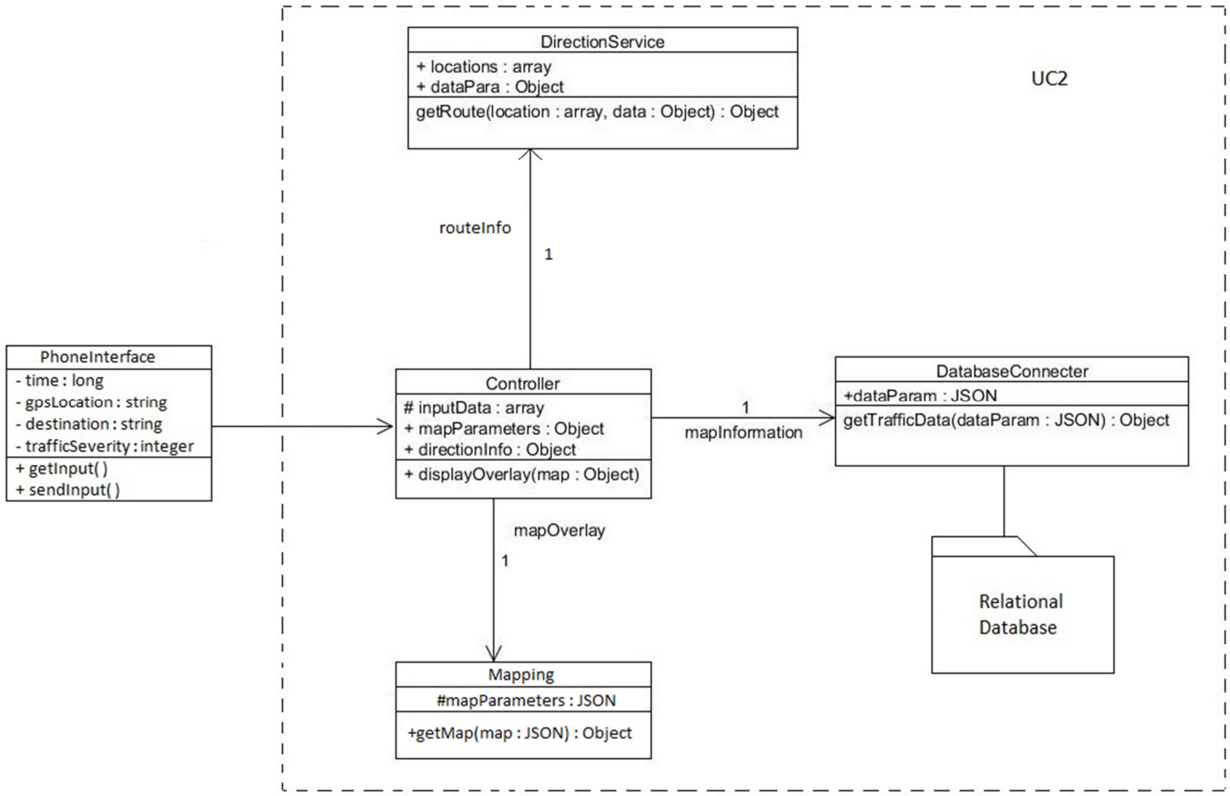


The controller class calls the other classes to execute their respective functions. Initially containing the desired data parameters as an object, the controller passes these parameters, along with an executable Javascript file, to either the traffic receiver or weather receiver. Once given the required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

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

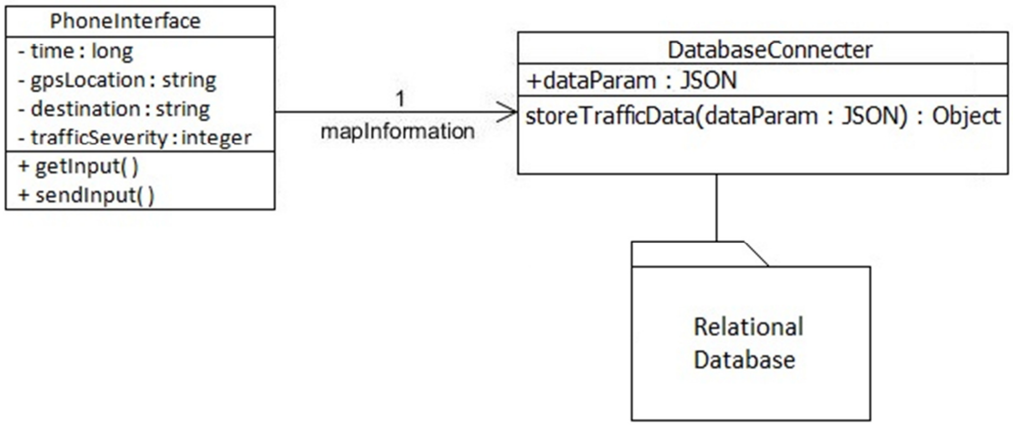
Afterwards, the controller gives the data to the parser, which extracts the necessary information. The controller then gives this information to the database connector, which have been correctly formatted by the parseData() function. The data is then stored in the database for future use, with storeData().

Use Case 6:



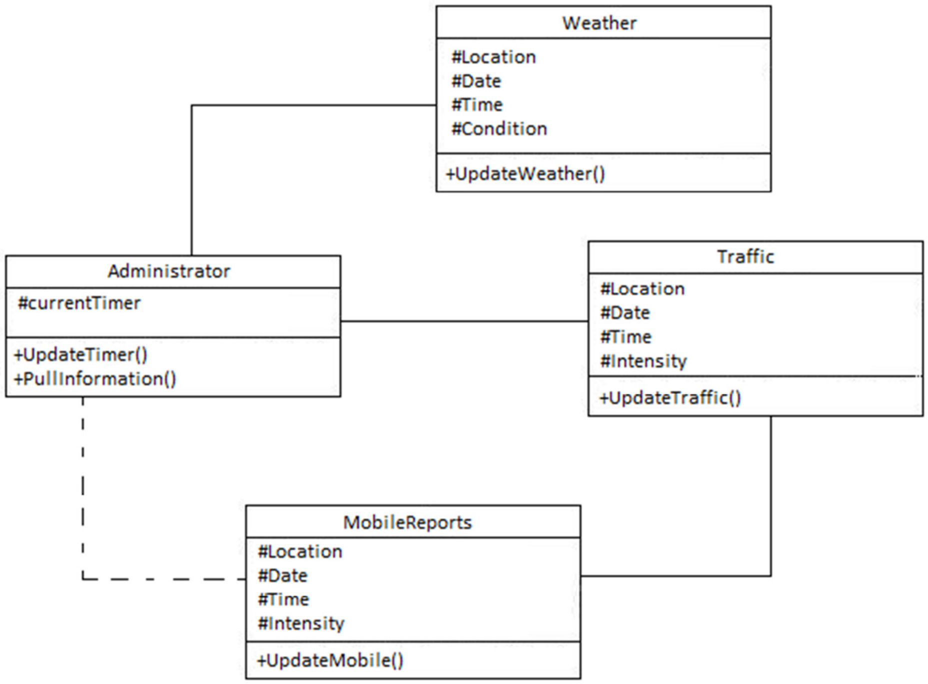
The PhoneInterface class is the mobile application that offers two options to the user. It holds data that is used both for obtaining traffic reports and updating the database. When getReport is invoked, the parameters for gpsLocation and time are collected along with the user input for destination, which are sent to the website to get directions. This leads into UC2.

Use Case 7:



The PhoneInterface class is the mobile application that offers two options to the user. It holds data that is used both for obtaining traffic reports and updating the database. When sendInput is invoked, the parameters for gpsLocation and time are collected along with the user input for trafficSeverity, which are sent to the database to be stored.

Relational Database:



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

Data Types and Operation Signatures

Controller:

- Attributes
  -String inputData[]
  -Object mapOverlay
- Operation
  +displayOverlay(Object map) (displays the overlay given to the controller by the mapping class)

Mapping:

- Attributes
  -JSON mapAttributes (contains the attributes given to the mapping class by the controller. These attributes help the mapping class decide which items to display on the overlay given to the controller)
- Operation
  +getMap(JSON mapAttributes) (Uses the Google Maps API to generate a map based on the requirements given in the parameters)

DatabaseConnector:

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

DirectionService:

- Attributes
  -String location[]
  -Object dataParam
- Operation
  getRoute(String location[], Object dataParam) (uses the locations given along with the dataParameters needed to generate a route.)

TrafficReceiver:

- Attributes
  -long

- Operation
  getTrafficData(JSON dataParam) (gets traffic data from web service)

WeatherReceiver:

- Attributes
  -JSON dataParam
- Operation
  getWeatherData(JSON dataParam) (gets weather data from web service)

Parser:

- Attributes
  -JSON dataParam
  -Object data[] (the final parsed version of the data collection)
- Operation
  parseData(JSON dataParam)

PhoneInterface:

- Attributes
  -long time
  -String gpsLocation
  -String destination
  -integer trafficSeverity
  -String inputData[]
- Operation
  getInput()
  sendInput(String inputData[])

Traceability Matrix

Because the class diagrams are based wholly on the use cases, the Traceability Matrix in Report 1 should provide sufficient information as to how the classes act. [In the full report, we can reference the page number it is on. For now, here is the diagram again.]

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

System Architecture and System Design

Architectural Styles

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

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

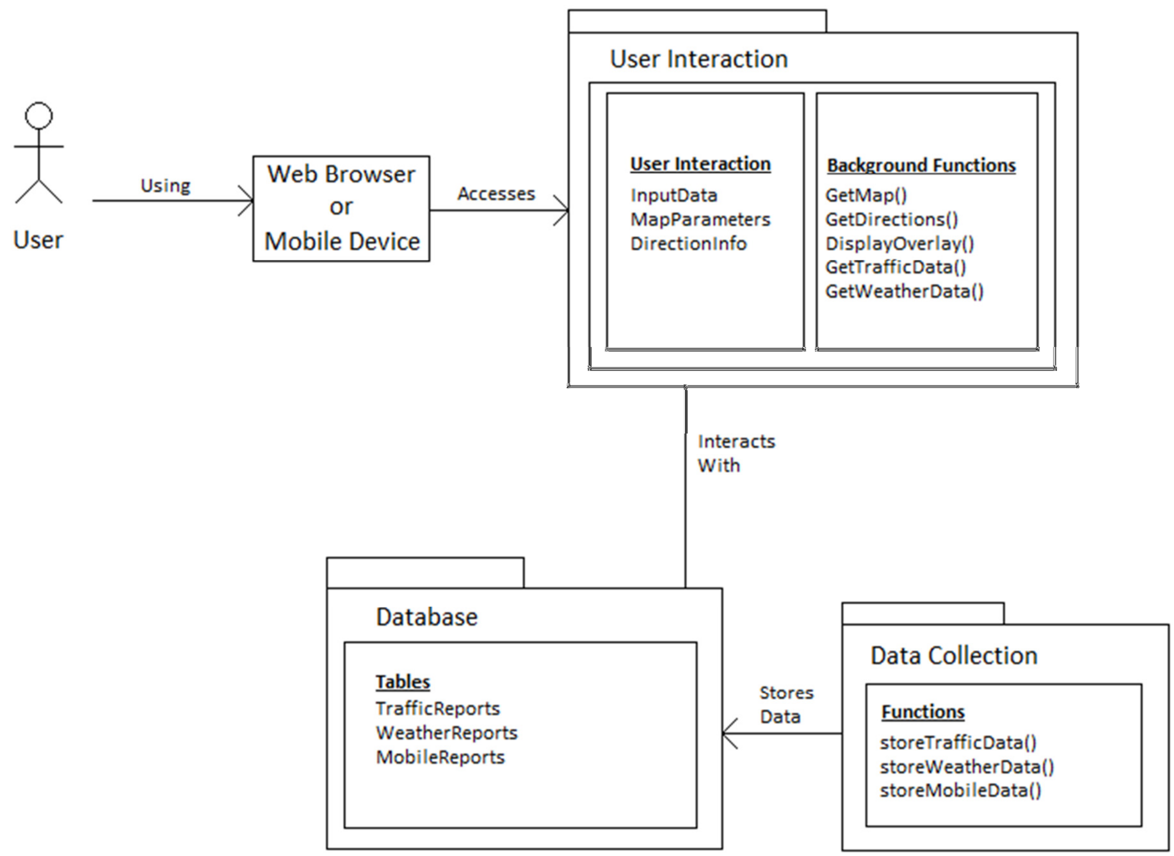
Identifying Subsystems

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

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

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

UML Package Diagram



Mapping Systems to Hardware

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

Persistent Data Storage

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

Database Tables			
Field	Type	NULL	Default
Traffic			
Time	Timestamp	YES	00-00-0000 (Date) 00:00:00 (Time)
Latitude	Decimal(0,0)	NO	0.00
Longitude	Decimal(0,0)	NO	0.00
Traffic Intensity	int	NO	0

Weather			
Time	Timestamp	YES	00-00-0000 (Date) 00:00:00 (Time)
City	varchar(255)	NO	----------
Condition	varchar(255)	NO	----------
Mobile Report			
Time	Timestamp	YES	00-00-0000 (Date) 00:00:00 (Time)
Latitude	Decimal(0,0)	NO	0.00
Longitude	Decimal(0,0)	NO	0.00
Traffic Intensity	int	NO	0

Global Control Flow

Execution Orderness

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

Time Dependency:

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

Concurrency:

No, our system does not use multiple threads.

Hardware Requirements

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

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

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

Project Management

Peter Lin and Matt Araneta are a team that will be working on web design and gathering data for traffic and weather. They will be in charge of use case 4 and 5. They will collaborate with Kevin and John on use cases 1, 2, and 3.

Geoff Oh and Mike Simio are a team that will be working on the mobile application of the project. They will be in charge of use case 6 and 7.

Kevin Hsieh and John Reed will be working on implementing an algorithm for finding the fastest route for directions, and managing the data gathered. They will work with Peter and Matt on use case 1, 2, and 3.

References

1. Marsic, Ivan. *Software Engineering*. 2012.
2. View Traffic Incidents. <511nj.org /IncidentList.aspx>.
3. View weather. <weather.com>.
4. Google Maps. <maps.google.com>.
5. Yahoo! Live Maps. <maps.yahoo.com>.
6. View Traffic Reports. <traffic.com>.