Traffic Monitoring System

Technical Documentation

Group #7

Matt Araneta, Kevin Hsieh, Peter Lin, Geoff Oh, John Reed, Michael Simio

May 12, 2013

Website-based Application Descriptions

List of files and their basic descriptions:

File name	Short description
index.php	Home page, short introduction
header.php	Constructs head of web page
footer.php	Constructs end of web page
menu.php	Constructs menu bar
progress.php	Displays table of progress
repxxx.php	Displays corresponding report pdf
groupMember.php	Displays group member
map.php	Displays input to construct map
mapImaging.php	Server-side code of map.php constructs map
sqlQueries.php	Functions to connect to database
polyEncode.php	Not complete
css/main.css	CSS document for all webpages
css/menu.css	CSS document to style menu bar
Images folder	Contain images for menu bar
Pdf folder	Contains all report pdfs
scripts/jquery-1.9.0.min.js	Jquery source code
scripts/menu.js	Menu source code
scripts/pdfobject.js	PDF viewer source code
create.php	Creates user account
directions.php	Webpage for directions
directionsGeocoding.php	Backend for directions.php
login.php	Login page
mobileReport.php	Used for posting data from the molile application

Full Descriptions

index.php

This is the homepage. This contains a short intro of what the website is about. This page also checks whether or not $_SESSION['loggedin'] == TRUE. If it is, the page display a messaging saying 'Welcome back, <username>'.

header.php and *footer.php*

These files construct the beginning and end of each web page, to assure consistency between pages.

menu.php

This file contains the menu's html seen on the website. which has different tabs a user can select to bring them to different pages of the website.

I added several tabs to the menu, and conditional loops which determine when tabs will appear. If the user is not logged in, the 'Login' tab will appear. Otherwise, the 'Logout' tab appears. Since the login page I created will display a logout button if the user is already logged in, both of these tabs will link to the login.php page.

If the user is logged in, I use the SQL query: "SELECT recentMap, recentDirections FROM Login WHERE Username = '$username'". I determine whether there is a string in either of these columns; if they are NULL, there are no recent searches. Otherwise, I add another tab to the menu called Recent Searches, which display any recent searches the user has.

To do this, I obtain the string from the SQL query, use explode() to turn it into an array, and display any searches in that array. Using conditional statements, the search will only appear in the menu if it exists. For example: if(isset($recentMap[4])). This checks whether the second search exists in the array. If it does, I add HTML code which adds another search to select in the menu. I link every one of these searches to either the map.php or directions.php page, making sure to include the corresponding search data in the link. When the user reaches either of these pages after clicking on one of their recent searches, the page retrieves this data and automatically use it to create a map (with mapImaging.php or directionsGeocoding.php) containing the user's desired information. Kevin contributed in implementing this part.

progress.php

This file constructs a table of our current progress and archives of our reports in a table format.

repxxx.php

These files build the page to display our reports in a PDF viewer.

groupMember.php

This file displays the group members.

map.php

This is the page that displays the input for the user. The inputs are within a form, and on submit of the form it uses Jquery AJAX to perform a post back with the values entered by user. The inputs are a text box limited to numbers and two dropdown select menus for weather and time. There is also JavaScript validation using JQuery. It will require that the zip code entered is 5 characters long and is all digits and there must be a selected value for weather and time. If none of the requirements are met , the page will not submit the data. If the inputs are correct, the form is submitted and the response of the post back is shown in the div formResult and shows the intensity color chart. Within the div formResult, an image is displayed which is the map of the zip code the user put in.

mapImaging.php

This is the server-side file that processes the post back from map.php. It will call a function defined in sqlQueries.php with the post back values. From the function, it will fill an array that contains the longitude and latitude of each point to construct and the point's severity. This file adds each point and color severity to the url to Google maps. With the url complete the file builds a html img tag with the src set to the url and returns the tag to map.php. It also returns a label for the not yet implemented zoom feature. The page also checks if the user is logged in. If they are not, mapImaging skips all the login user code and proceeds to display the map. If the user is logged in, the information they typed into the input fields is retrieved, manipulated, and the recentMap column in our database is altered accordingly. The data is stored using one really long string which stores the user's 5 most recent searches. Since there are three variables; zip code, time, and weather, the code would store up to 15 different strings in the database, separated by columns. Every three strings represent one of the user's recent searches. The first three represent the most recent search, the second three represent the second most recent search, and so on until the 5th most recent search. These strings are updated with the new values. The string is ordered newest to oldest.

sqlQueries.php

This file will be for all functions that require a database connection. The function in the file called mapTrafficPoints. The function takes a weather, time slot, and zip code. It opens a connection to the database and performs a select query based on these conditions. It will access the Traffic_Incident table for the longitude, latitude, and traffic severity. It takes the results and parses it so that it is an array of arrays of longitude latitudes and severity color. It then closes the database connection and returns the result array. The other function defined is the getPointsInBox which takes a top left latitude, bottom right latitude, top left longitude, and bottom right longitude. It then performs a select statement in the database and returns an array of all the points within the boundaries. It fills the array and returns that array. Another function defined is mobileReport which takes zip code, severity, street, county, state, short description, latitude, and longitude. It then inserts the data into the database table Traffic_Mobile and returns value based on whether or not the insert is successful.

directions.php

This page is the webpage displayed to the user for directions input. There is two input boxes and one dropdown menu. The input boxes are linked to google's auto complete address which provides possible addresses based on the characters you type in. There is also JQuery validation to ensure the boxes are filled and there is a selected dropdown value. On click of submit, if the requirements are not met then there is no postback. If the requirements are met, the data is posted back and the return is an image of the entire route and a table of direcitons turn by turn.

directionsGeocoding.php

This page is the server-side file that processes the information from directions.php. It first geocodes the input addresses to obtain a latitude and longitude point. It also checks that it is a valid address. If it is not the page returns that the address is not found. If the address is found it then calls the function getPointsInBox from sqlQueries.php. The return is used to optimize the route by adding avoidance weight to each point so the route avoids heavy incident locations. It will then use the data to construct a url to MapQuest. The return is a JSON object. The file parses the object to obtain the image link and the each leg of the route. It will return the table of direcitons for each leg and the route image. The page also checks if the user is logged in. If they are not, direcitonsGeocoding.php skips all the login user code and proceeds to display the map. If the user is logged in, the information they typed into the input fields is retrieved, manipulated, and the recentMap column in our database is altered accordingly. The data is stored using one really long string which stores the user's 5 most recent searches. Since there are three variables; zip code, time, and weather, the code would store up to 15 different strings in the database, separated by columns. Every three strings represent one of the user's recent searches. The first three represent the most recent search, the second three represent the second most recent search, and so on until the 5th most recent search. These strings are updated with the new values. The string is ordered newest to oldest.

login.php

This is the login page of our website. In order to keep a user logged in, I made use of PHP sessions, which allow information storage on the web server. Basically, a sessions create a unique id for each visitor and store variables based on each id. This makes it very easy and efficient to maintain information about a user. The login page I created consists of numerous conditional loops. I use a session variable called $_SESSION['loggedin'] to keep track of whether a user is logged in or not. If it's set to TRUE, the user is logged in, and the logout button is displayed on the page. Otherwise, I determine whether or not they have just attempted to log in with the submit button. If they just attempted to log in, I connect to the database using mysqli_connect, determine whether or not their username and password are valid (using an SQL query), and log them in if they are a valid user, setting $_SESSION['loggedin'] to TRUE. Once this occurs, I direct the user to the home page of the website, index.php. If they enter invalid information, I display an 'Invalid login.' prompt on the screen and a 'Try again' button, which refreshes the login page so the user can again enter their username and password.

create.php

If the user clicks the 'Create Account' HTML form, they will be directed to this page. If the user has not attempted to create an account yet, an HTML form is displayed. The user has to enter a unique username, their desired password, and retype their password to confirm it. Once they press 'Submit', the page refreshes and I set a variable called 'create' equal to 'yes'. When create == 'yes', I first determine whether the username is between 4 and 20 characters and whether the password is between 3 and 20 characters. I also check whether the two passwords match. If any of these conditions are not met, they are displayed on the screen. If they are all met, I create a database connection and use an SQL query (SELECT COUNT(*) FROM Login WHERE Username = '$username') to determine whether that username already exists in the table Login. If it does (if count == 1), then a message is displayed saying that username cannot be used. If the username is unique, then they have successfully created an account, and I use an SQL insert to add this new username and password to the Login table.

Mobile-based Application Description

Basic Description

postData()	Communicate with web service
MainActivity.java	Display main page for Application
Report.java	Displays input to construct Map
Submit.java	Submits mobile traffic report

Full Descriptions

postData()

This function is called when the user presses the “Get Report” button on the get traffic report screen. It obtains a map image with traffic intensity overlay from the webservice and returns it on an Android ImageView on the lower half of the screen. The function first obtains the data from the fields entered by the user (zip code, time, and weather), and creates an Android Http Client to interact with the mapImaging.php part of the web service. The response is an HTML tag. The postData function gets the message of the body and converts it to an InputStream, which is a stream of bytes. This stream of bytes is read into a StringBuffer and converted into a String data type. At this point, the content of the String is the text that makes up the HTML tag. A URL pointing to the map image with overlay is parsed from the String and the image is returned to the user.

MainActivity.java

This is the home page for the mobile application. From this page, the Mobile User has the option of going to the Map page or the page to submit a mobile report.

Report.java

This page allows the User to input zip code, time, and weather. This data is sent to the main application to be processed. Once the main application has finished placing the necessary points on the map, an image URL is returned. The image is then shown to the Mobile User to observe traffic history.

Sumbit.java

This page allows the Mobile User to submit a traffic report into the database. The User is asked to enter the current traffic intensity of the area they are in. Once the Mobile User presses “Submit,” the GPS location of the Mobile User is found through using a geocoder of the User’s current address. This data is then sent to the main application, where it constructs an SQL query to input the data into the database.

Data Collection Documentation

List of files and their basic descriptions:

File Name	Short Description
googleReverseGeo.pl	Reverse geocoding script calling google api
mapquestTrafficApi.pl	Traffic incident script calling mapquest api
trafficAlgorithm.pl	Algorithm script to apply our algorithm to collected traffic reports
Weather.pl	Weather script to collect weather information
Weather2.pl	Weather script to collect weather information
Weather3.pl	Weather script to collect weather information

Full descriptions:

googleReverseGeo.pl

This script is run after the mapquestTrafficApi.pl script is run. We first use MapQuest's reverse geocoding to obtain addresses for longitude and latitude. MapQuest does not always obtain an address for every longitude and latitude, so Google's reverse geocoding API is called to fill in the empty addresses.

mapquestTrafficApi.pl

This script is run every 3 hours starting from 12am. This script will get all traffic incidents within a bounding box that we define. We have hard coded bounding boxes to cover all of New Jersey, Pennsylvania, and New York. This script gets the longitude and latitude of the traffic incident and then calls MapQuest's reverse geocoding API to obtain the street address, county, city, and zip code. There are checks to make sure that each incident is in the tri-state area. Then the script will insert the data into the database. The data inserted is zip code, latitude, longitude, severity, a short description, address, county, and state. It will also update the traffic call counter which keeps track of how many times traffic reports are inserted into the database.

trafficAlgorithm.pl

This script is run once a day at 11:30 pm. The script will collect all of today's traffic incidents from the Traffic_Mapquest table. It then puts the data into a multiple level hash map. It also calls the another table Traffic_Display which contains the points we display on our map. It puts that data into a similar multiple level hash map. The hash map is defined as weather->time->street->longitude->latitude = array of severity number and index. Then in loop it compares all points from Traffic_Mapquest against Traffic_Display with the same weather, time, and street. The points that are within .5 miles which is calculated using the longitude and latitude have their severity values added to that traffic point from Traffic_Display. If there are no points within .5

miles, that point is added to the Traffic_Display hash map. At the end the script inserts/updates the Traffic_Display table in the database. The script checks whether to insert or update if there is a defined index. The data inserted is time, address, county, weather, severity, longitude, latitude, and zip code.

Weather.pl

This script is run every 3 hours starting from 12 am. It first gets a zip code per county. The weather for one zip code in a county will apply to all zip codes in that county. This script will cal l then call the weather.com API to get the weather for each zip code. The weather is gotten and is further simplified into weather conditions that we defined. The possible weather conditions we get back were too many, so certain weather conditions were recorded as others. The script then inserts into the database the weather condition, zip code, and timestamp.

Weather2.pl

This script was created to API call limitations.

Weather3.pl

This script was created to API call limitations.

Database Implementation

List of tables and their basic descriptions:

Table Name	Short description
Traffic_Counter	Counter of traffic report calls
Traffic_Display	Contains relevant data to display on map
Traffic_Mapquest	Contains all traffic incidents from MapQuest
Traffic_Mobile	Contains all traffic reports from the mobile app
Weather	Contains all weather conditions
Zip_Codes	Contains all zip codes that we support
Login	Contains user accounts

The database is MSSQL and the only insertion and updates are from the perl scripts. The table whose data is used for display on the website should only be the Traffic_Display table.

Full descriptions:

Traffic_Counter

This table has the columns Index(int) and Counter(int). This table was created specifically to contain only the number of times that traffic incidents are retrieved. This number is necessary for our traffic algorithm.

Traffic_Display

This table has the columns Index(int), Date_Time(timestampe), Zipcode(varchar), Latitude(decimal), Longitude(decimal), Address(varchar), County(varchar), Severity(int), Weather(varchar), Time_Value(int). This table is created to hold traffic incident points to be displayed on the map. The Date_Time is just used to observe the last time a traffic incident point's severity value was updated or inserted.

Traffic_Mapquest

This table has the columns Index(int), Date_Time(timestampe), Zipcode(varchar), Latitude(decimal), Longitude(decimal), Address(varchar), County(varchar), Severity(int), Short_Descrip(varchar), and State(varchar). This table is used to hold all the traffic incidents pulled from Mapquest.

Traffic_Mobile

This table has the columns Index(int), Date_Time(timestampe), Zipcode(varchar), Latitude(decimal), Longitude(decimal), Address(varchar), County(varchar), Severity(int), Short_Descrip(varchar), and State(varchar). This table is used to hold the traffic reports from the mobile application.

Weather

This table has the columns Index(int), Timestamp(timestampe), Weather(Varchar), and Zipcode(varchar). This table is used to hold the weather conditions for one zip code in each county.

Zip_Codes

This table has the columns Index(int), County(varchar), and Zipcode(varchar). This table should not be changed and contains all the zip codes in the tri-state area and the corresponding county

Login

This table has the columns Index(int), Username(varchar), Password(varchar), recentMap(varchar), and recentDirections(varchar). This table holds all user accounts and the most 5 recent searches the user has made.