Health Monitor Analytics

GROUP 1

Report 3

D. Takacs

G. Bati

J. Abdulbaqi

I. Paraskevakos

K. Dong

Individual Contributions Breakdown

"All team members contributed equally"

Table of Contents

Summary of Changes

Game System

Deletion of REQ – 10 of Report 1: This requirement is removed, because there was no time to analyze it and implement it.

Deletion of REQ – 11 of Report 1: The reason is that it is mainly the same with REQ – 12 of Report 1.

Change in REQ-12 of Report 1:: This is the most important change in the most important change in the requirements of the Game Feature. The requirement at the beginning was to create leaderboards for the Counties that are in a State. The first question, that had to be answered, is how are we going to map a tweet from the "City, State" or its geo coordinates? A quick answer is to get the coordinates of the geographical center (geocenter) of each County in each State, draw a circle with radius the biggest distance from the geocenter of the county until its borders and everything inside that circle belongs to a specific County. If the area a County covers was close to a circle, the possible overlaps would not have been a major problem and would not change the results significantly. The geographical shape of Counties is irregular and the overlaps could affect the results significantly. Also decisions should have been made in which County and eventually State those points should been allocated. That arises the question how to properly allocate the points to a tweet that belongs to another State than the one that the County that takes those points is in. Since there was not sufficient time to do all the proper geographical analysis or find an API that can easily and without major restrictions to give the necessary information, the decision was made to change the requirement from Counties to Cities. The question that needs an answer here is how the Game distinguishes one City from another with the same name in the same State? Actually, it does not and the reason is that the way this system allocates coordinates to tweets does not take into consideration different cities with the same name in a State.

Deletion of REQ – 13 of Report 1: Since we had decided between the submission of Report 1 and the 1st demo that we will not have registered users in our system. This decision results to removing Requirement 13.

Recommendation System

Delete the requirements of 19, 20, 23, 24 in report 1. The problem is that we cannot get enough information from the interactive map system so far. It is pointless to recommended different types of sports based on the location as we have not enough data.

Change UC-10 of Report 1 into a getting weather requirements. The recommendation system would let the user to type a certain location and get the weather

Change UC-11 of Report 1 into getting a data from the interactive map.

These two changes of the UC are baed on the change of the Requirements.

1.Customer Statement of Requirements

We will produce a superior product via enhancements to the previous semester's Health Activities Monitoring and Analysis projects. We will divide the customer statement of requirements into 5 main parts, one part for each of the 5 main enhancements. The detail of the problem, the motivation and the vision will be in the following sub-sections.

1.1 Problem Statement

1.1.1 Problem:

The whole project is an extension of three previous project about health activities monitoring and analyzing. The former groups were able to successfully make a website which includes an interactive map. The map is a heat map that shows the number of tweets that people are posting based on geographical location. We want to improve the project from 5 different perspectives.

Firstly, we want to improve the previous heat map. The previous projects used less than optimal methods to collect accurate tweet data. Their heat maps suffered from a lack of data, because they could only use the 1% of all tweets which contained geological coordinates. Also, the previous projects used less than optimal keyword phrases, which resulted in catching a large percentage of unrelated tweets. And they lacked any analysis further than displaying count data of tweets containing keyword phrases by area.

Secondly, the previous team includes a ranking of areas from the number of tweets and use it as an inspiration for people to exercise. We want to make a system that can increase people's motivation, much more than the original method of offering simply a ranking display. To do this we will create a game based on exercising, to increase people's motivation for physical activity.

Thirdly, the original interactive map that shows the health activities includes a broad overview of these activities, which is great, but what if the users were interested in more detailed information, like the specific type of physical activities/sports? The previous project team did not design any kind of specific actions, they simply show the users the overall twitter hash-tags of the same field.

Fourthly, while having health monitoring software which would make suggestions to the

users, like what sport/physical activities are popular or which gym is nearby, is very useful, there is some drawback. The issue that arises is that recommending different exercise activities regardless of weather, could cause users not to follow recommendations. For example, recommending outdoor running during snowfall, or indoor activities during beautiful weather. We plan to give a full recommendation which includes weather forecast, geographical information, and twitter information.

Finally, we noticed that many people around the world wish to lose weight in the beginning of every year. Weight loss is one of the most popular goals. Unfortunately, a good number of these people get discouraged before reaching their goal. Previous versions showed little tangibles about the actual effectiveness of the different activities. For example, the duration or calories burned. Using tweets based on calories burned could provide useful visualizations for extra motivation.

1.1.2 Motivation:

We noticed several problems that we found from the previous projects, that we want to make improvements upon. Below, we will review our motivation about those problems.

For the improvement of the heat map, we think that by improving upon our data collection and analysis methods, we can gather more robust data. We can then draw more accurate logical conclusions regarding exercise activity by area. This accuracy and our ability to draw more analytic insights will make our product more appealing to users.

For the improvement of the old ranking board to our new gaming system. We want to find a way to motivate our users to exercise more and to actually persuade others to start exercising. We also want them to start tweeting about their exercise. What was realized from the previous projects was that area ranking was only done from the absolute number of tweets. We think that this is inefficient since more tweets from an area do not mean that this area actually exercises more than another area with fewer tweets.

Take for example New York City and New Brunswick. There may be several thousands of tweets from the first and just a couple of hundred from the second. If those numbers are compared to the actual population of the area, it can be erroneously concluded that New Brunswick’s residents exercise way less than New York's. But this is only due to New York having nearly 100 times the population size of New Brunswick.

To improve the old heat-map into a map that can show the per-capita distribution of tweet data. We want to develop an interactive electronic map which will show the users not just the count of the tweets for certain health activities, but it will show them normalized to their population size.

To improve the recommendation implementation, we will make a function of the system that can recommend physical activities for users. The function would let the user make make two binary choices. First, from “Low-Intensity” or “High-Intensity”, and then from “Individual” or “team”. The system would consider weather, geologic information and the most mentioned sports on twitter, to make a recommendation of several activities. The form of the recommendation would be “word cloud”

Finally, for inspecting people’s calorie burned. We would like to design a Calorie-Meter that tells user how many people around the area have already burned how many calories. This would be an inspiration for the user who wants to lose weight.

1.1.3 Vision:

For the improvement upon the heat map, we will use per capita population data combined with twitter data to create a map based data visualization that displays per capita data. This will give us more insight into regional differences among populations of the US than raw count data would.

Then for the gaming system, we will create a competitive game that will assign points and track scores by areas, such as states and counties. The point assignment will be based only on the tweets from each area. Briefly, based on the number of tweets or the relative number of tweets per area (e.i. the ratio of relevant tweets and the population), a scheme for point assignment will be devised. Also in order to keep the game competitive, we will also subtract points from areas where the residents are exercising less than they had been during previous periods. We mainly want to use the ratios because we think that it is more fair to reward communities in which their residents exercise more as a whole, and not be biased towards areas with larger populations.

Thirdly, for the interactive map, we can distribute display information. The map’s options will include many types of health and sports activities like types of sports (football, soccer … etc.) or fitness activities like working out at the gym. Briefly, these maps will show

you the collected data from twitter hash-tags for individual activities. The user will be able to choose the preferred activity from a list, then a map will show how this activity is distributed in the area that the user chooses. The list, which will be close to the map, will be updated whenever the user changes the area in the map, because the list will include the top number of the activities in the specific area only.

Then the recommendation part would have a form with two questions, after the user chooses the answer, the questions would disappear and would display a word cloud. Upon clicking each word in the word cloud, the cloud would display detailed information under the word cloud. The information would include the activity location, popularity, and so on. There would also be a “choose again” button to let the user change his choice.

Finally for the calorie meters, we will track automatic tweets that are generated by workout monitoring devices and mobile applications that mention the workout and the amount of burnt calories, we will then use the calorie gathered information to perform more tangible analyses. We will then keep track of calorie data, and display the results with a meter presented in the main page of our website to make sure that whenever a user visits the site he/she will be motivated to join the many people around the world who are burning calories and exercising. As the time of the semester permits, we can add many more features to this meter. For instance, classifying the top workouts and what percentage they contribute in terms of burning calories.

2 Glossary of Terms

Twitter: Short messages on micro blogging and social networking service.

Tweet: A message sent using Twitter (http://en.wikipedia.org/wiki/Tweet).

Hash-tag: Hash-tag is a word or a phrase prefixed by the symbol # in many social networking websites.

Google Maps: Google Maps is a web mapping service application and technology provided by Google.

OpenWeatherMap: OpenWeatherMap is an online service that provides free API to weather data including current weather data, forecasts and history data to the developers of web-services and mobile applications.

MongoDB: MongoDB (created by "humongous") is a cross-platform document-oriented database system. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schema (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open source software.

MongoDB Collection: A grouping of MongoDB documents. A collection is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection have a similar or related purpose.

JSON: JSON or JavaScript Object Notation, is an open standard format that use human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-

independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages. JSON's basic types are: Number, String, Boolean, Array, Object and null.

Heat map: A heat map is a graphical representation of data where the individual values contained in a matrix are represented as colors. Fractal maps and tree maps both often use a similar system of color-coding to represent the values taken by a variable in a hierarchy.

Map: The Interactive electronic map used to build this service like Google maps or any other web maps.

Guests: The person who uses the features of the application service.

Users: The subset of guests that register with the Website and could use the functions that are specific for registered users only.

List: Drop-down list that will be able to slide in a place close to the map.

Marker: A shape like an icon or any suitable shape will be used to illustrate the distribution of the health activities on the map.

Word Cloud: The function that displays a word cloud of recommended physical activities. The most recommended would have the largest font.

Information List: A list where one can click each word in the word cloud, that would cause the system to show detailed information about the activities in a list underneath the word cloud.

3.System Requirements

3.1 Enumerated Functional Requirements

Table 3.1 Functional Requirements of Health Activities Monitoring and Analysis System

Identifier	Description	PW[1]
REQ – 1 (D)	The system shall be able to display a heat map showing per capita data. (Normalized by population number)	5
REQ – 2 (D)	The system shall be able to display different views of data by choosing from an available selection offered	3
REQ – 3 (D)	The system shall provide a collection of different sets of data for the user to choose from.	3
REQ – 4 (D)	The system shall at a minimum, allow heat map to be displayed for major metropolitan cities	4
REQ – 5 (D)	The system should allow heat map to be displayed for less populated regions as well. (Less Discretized)	2
REQ – 6 (D)	The system should show estimated actual count data to be viewed based on our discussion in 3.c) above	1
REQ – 7 (I)	The System shall assign points to areas based upon their tweets	5
REQ – 8 (I)	The System should track the history of the tweet numbers over a period of time to assign extra points	2
REQ – 9 (I)	The system shall assign points to areas according to the number of relevant tweets	5
REQ – 10 (I)	The system shall show a leader board for States and Cities in a selected State	4
REQ – 11 (J)	The system shall allow the user to choose a specific activity from a list.	5
REQ – 12 (J)	The system should update the activity list as soon as the map area changed.	3
REQ – 13 (J)	The system shall display location markers on the map, for any activity chosen from the list.	4
REQ – 14 (J)	The system should display markers from different activities with different colors.	2

[1] Priority Weight

REQ – 15 (K)	The system shall recommend activity based on weather and geology information.	5
REQ – 16 (K)	The system shall show user a chart to display the eight most recommendation activities based on the interactive map system.	5
REQ – 17 (G)	The system shall use a list of hash-tags and automatic tweets to search for number of burnt calories and the related workouts.	5
REQ – 18 (G)	The system shall download all relevant tweets and then store them to a local database for analysis purposes.	5
REQ – 19 (G)	The system shall extract tweets from local database after the necessary calculations.	5
REQ – 20 (G)	The system shall allow users to see the counter in the main page of the website that are updated daily at 12:00am.	5
REQ – 21 (G)	The system shall allow users to sign up.	1
REQ – 22 (G)	The system shall allow users to log in with correct login ID and Password.	1
REQ – 23 (G)	The system should track the history of the tweets for a period of time to generate extra analysis.	3
REQ – 24 (G)	The current status of the counter should be shared through social media websites.	4

For the requirement that relates to data analysis and per capita heat map. This system aims to break away from the “Bombardment” of tweets from the most populated areas, to try and show how the tweet data is being communicated relative to the size of the population. REQ – 1 is the most important, followed by REQ – 4. Notice that ideally, we could produce a heat map on a very granulated detail.(REQ – 5) This might be too hard to implement in the given time frame, so I put this as a “should”. REQ – 4 would allow us to discretize the large cities. If we tried to use all locations (REQ – 5), then to combat the problems of statistical noise from smaller populations, we would have to devise a complicated method of weighted averaging nearby populations and tweets. But if we stick to just REQ – 4, the large numbers of tweets coming from the large populations will leave us less susceptible to statistical deviations.

REQ2 and REQ3 are related and will allow the user to view maps based on different data, instead of only one default mapping, which we deem to be necessary. REQ6 may not be feasible due to statistical noise and whether or not we have highly reliable aggregate US

data for the data source that we are modeling in the heat map. Because REQ6 is heavily dependent on factors of our system to be which will be unknown until completion, it is the least important and we will consider this if we have time to implement this.

For Requirements that related to the gaming experience system. REQ – 7 and REQ – 9 have the highest priority weight. Since each area, i.e. City, is also considered a player, it is obvious that point assignment is one of the key objectives. We have assumed that our users will continue to use Twitter as the main method of reporting their exercise activities. So we use their tweets to assign points to areas. This way, the user can continue playing and continuously see the community's results. REQ – 9 must also have the biggest priority because the system must also rank areas based primarily on the relevant density of tweet from each area. Since this is a game, there must be a way to rank areas and to know about it. The best way is by the points that they have. REQ – 10 is the one for this purpose. It has smaller priority that REQ – 7 and REQ – 9 because in order to have a game and create the leader board, you must first have point assignment. Also, the leader board must be able to show some level of detail. For that reason there will be a state leader board and a city leader board per state.

People generally lack the motivation to exercise, but it is possible that our community game will give them enough reason to start. So to some extent, our point assignment methodology should reward positive change. On the other hand, it should also punish for negative change. This means that for communities in which the number of people who exercise has increased, they should get additional points, and for communities in which the number of people who exercise has decreased, they should lose points (REQ – 8). Also, when people exercise, they probably are not doing just one specific type of exercise only.

For the Interactive map parts of the system. It is aimed to show the user a map with distributed markers according to the selected field from a list. REQ – 11 is about the main feature that this service owns, which is the ability to choose a specific activity to show its distribution on the map. Therefore, it should have a high priority. REQ – 12, this feature will make this service more dynamic, but it needs more work to be able to do this feature and it will not stop the entire service if it is not done, so its priority is 3. REQ – 13 is an essential part of the service to complete the main feature therefore its weight is 4. REQ – 14 is very nice to make this service fabulous, but again this may be needed extra work and time and if not complete they will not affect the whole service. Therefore, their priority is low and they depend on the time and available information to do it. A required policy should be clear for all the work needed for this service: IAM-WP1[2]: At any time, the maps

[2] IAM-WP Interactive Activities Map Work Policy

show only one type of activity from the list. This means that we have to keep the map related with only one activity. So each time the map moves to another selection, it should view one activity only.

This system is aimed to show a word cloud including the recommendation of physical activities. REQ – 15 and REQ – 16 are of the most important because they are the core of the recommendation system. REQ – 16 is that the system should make a suggestion after knowing that the user's choice of exercise category, the weather forecast, the nearby equipment (gym...), and the popular sports/physical activities nearby (twitter). The REQ – 16is that the system should make a display based on what it has done in REQ – 15.

One of the most important objectives of this system is to motivate people to exercise through gathering automatic tweets generated by common devices and mobile applications that show the type of workouts and burned calories associated with them, thus REQ – 17 and REQ – 20 carry the highest priority weight.

It is recommended that users should be able to see more analysis related to the tweets, as explained in REQ – 23, but this is not a top priority due to the fact that the semester is very short. Thus, REQ –23 has a lower priority than the previous mentioned ones. REQ – 21 and REQ – 22 have a lower priority since they are only desirable because many users like to be anonymous, as well as to avoid any security breaches or users' sensitive data loss. REQ – 24 is highly recommended to publicize our work and to encourage more people to check out our website daily, as they may check the weather and temperature. The system, however, does not specify which social networks it can support. We may introduce an option to allow the user to suggest some websites.

3.2 Non – Functional Requirements

Table 3.2 Enumerated Non - Functional Requirements

Identifier	Description	PW
REQ – 25 (J)	The system shall show a map of the initial area.	5
REQ – 26 (I)	The system shall update area points in fixed time intervals	5
REQ – 27 (I)	The system shall update the leader board in fixed time interval	5
REQ – 28	The system shall be accessed through a website.	5

(G)		
REQ – 29 (G)	The system shall update the counter depending on Twitter API's policy of frequency of gathering data.	5
REQ – 30 (G)	The system should be accessed through mobile applications.	1

REQ – 25 essential to perform all the functions of the map. That is why it has the highest priority.

In order for our game to be, let us say, live or real time, the points and the leader boards need to be updated several times a day. There are two main reasons. The first is that when a game is competitive the users want the accumulated points to be assigned to their community as soon as possible, almost immediately. So, it is important to update the points and the leader board many times. The second reason is that Twitter does not allow a system to ask for new data all the time, but obliges fifteen minutes intervals between data requests. So we cannot update in a true real time fashion the points and the leader boards. As a consequence, there is a trade-off in that a time interval between updates is required to be both as quick as possible, and also long enough that it does not violating Twitter's Policy. The above shows the reasons for REQ – 26 and REQ –27.

In order for the Calorie counter to be accurate, it has to gather data as soon as they are tweeted, as allowed by Twitter API (REQ – 29). Since many people have access to the internet, having this counter present in a website makes a lot of sense (REQ – 28). Also, many people use their cell phones to browse the internet, so developing an application for Calorie-Meter is desirable, but not a must (REQ – 30).

3.3. On-Screen Appearance Requirements

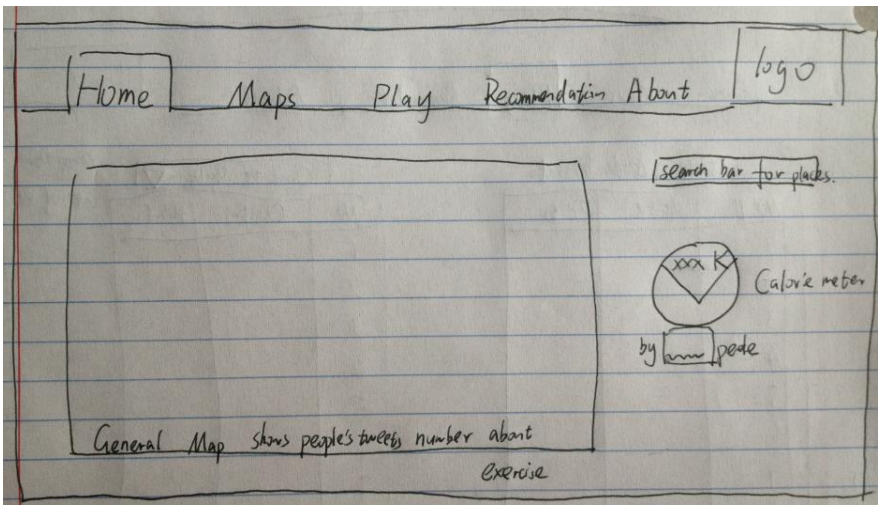

Figure 1 Home Page of Our Website

In the home page of the website, you can see the logo at the right up region of the page. There would be a heat map which shows the original tweet numbers of the USA about physical activities. The map can be zoomed in or out by using the mouse or the search bar. The Calorie meter would show how many people together burned how many calories today.

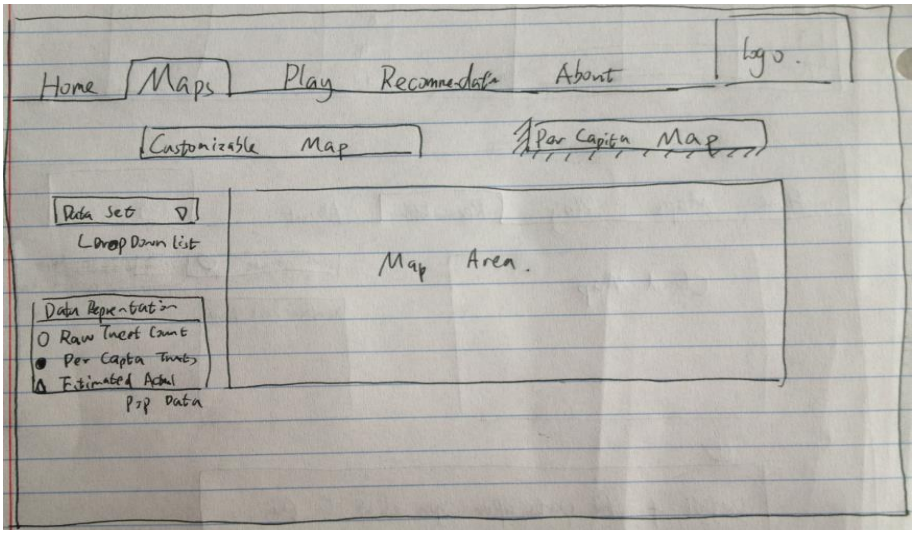

Figure 2 Per Capita Map of the Map Page

The main part of the page would be a per capita map, and on the left of the map would be some choices and a function list. The Customize Map (which would be the interactive map system mentioned in the problem statement) would be similar, just change the choice

to be different types of physical activities.

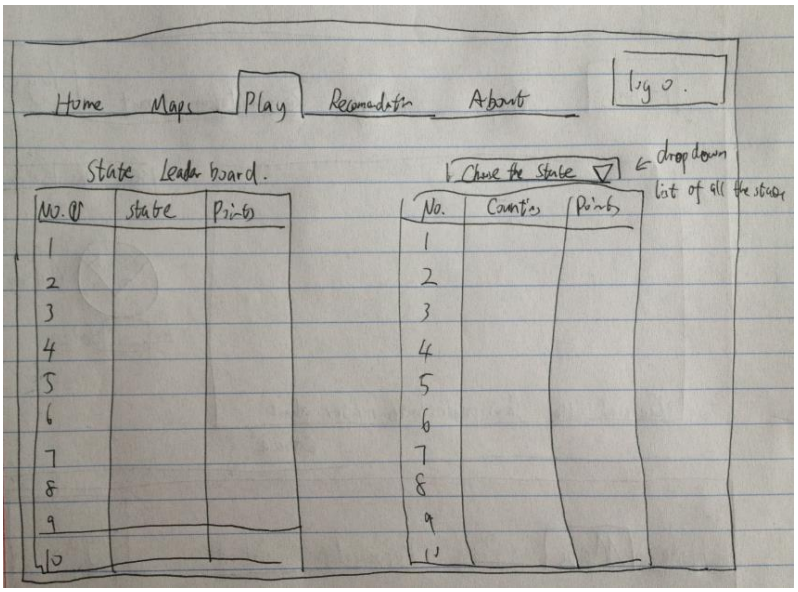

Figure 3 Play Page of the Website

The play page includes the gaming system mentioned before. The left hand side of the State Leader board will always be displayed there. The leader board will be a list from 1 to 10, showing the 10 states with the most points, along with their points also. On the right hand side there will be a drop-down list where all states will be presented. When the user selects a state, the state's points will be displayed and a list of the top 10 counties of that state similar to the above table.

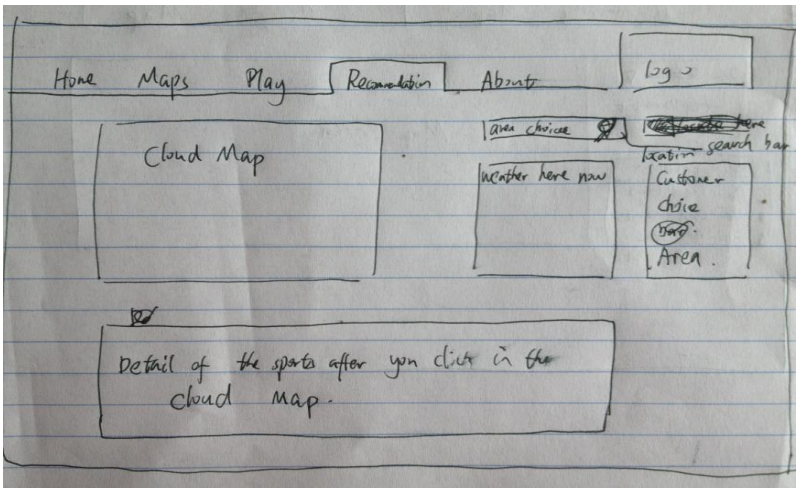

Figure 4 Recommendation Page of the Website

The left part of the page would be a cloud map area, while on the right hand, there would be a place for the user/guest to choose their location and category of the sports. The

bottom would be detailed information of one of the selected physical activities.

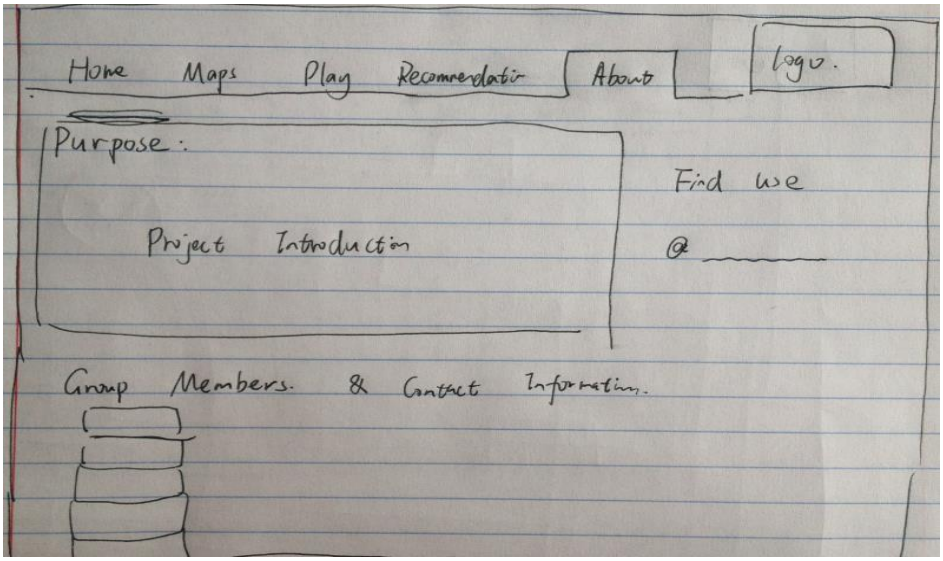

Figure 5 About Page of the Website

The About Page would include an introduction of the project. The name of each member of the team and the contact information.

4.Functional Requirements Specification

4.1 Stakeholders

Our system is mainly focusing on Health Activities Monitoring and Analysis. The groups of people who may be interested would include:

Exercise enthusiasts:

These people are the main customers of our project. They love to exercise and hope to get more information about it. They could use our website to easily check out what are the popular nearby physical activities, and what is the recommended activity is right now. There people are very likely to register on the website and take part in the exercise gaming we have.

Healthy lifestyle enthusiasts

These people may be more interested in overall health, such as nutrition, stress reduction, etc, but not as into actual exercise as the exercise enthusiast is, but they still may want to use our site to help with their healthy lifestyle. They can get information on our website like the recommendation page if and when they decide to do some exercises.

Government / Research Institutions

Government or Research Institutions may be interested about the most popular physical activities by different areas, or people's participation rates in different kinds of activities. Both of which can be displayed on our website.

Sports Company

For Sports Companies, they could use our website to know the popular sports nearby and in this way, adjust their commercial or selling strategy to make the company more successful.

Gym

For gym staffs, or similar exercising center staffs, they could use our website to know the popular sports nearby, and the basic ratio of different kinds of sports. And the information

would be helpful in situations such as modifying their gym for different kinds of exercise activities/sports.

4.2 Actors and Goals

User: (Initializing type)

Interacts with the system, gets the information they need, takes part in the game experience. A registered user can have access to all the information and services of the website.

Guest: (Initializing type)

Gets the information they need, but cannot take part in the gaming part of the system. Can sign up to be a user, in which case they will become a User, as above.

Administrator: (Initializing type)

The administrator is a special user and has top priority to access and change our database and all other user information.

Database: (Participating type)

Records all tweets about exercises, with location, person, and time of the exercise and username, password and profile of users.

Google database: (Participating type)

We get geology information from Google database into our database.

Twitter database: (Participating type)

We get twitter information from Twitter database into our database.

Open weather map database: (Participating type)

We get weather information from Open weather map database into our database.

4.3 Use Cases

4.3.1Casual Description

Table 4.1 Use Cases

UC-1 **Log in:** Allow the user to access the system Derived from REQ22
UC-2 **Sign up:** Allow the guest to create a new profile Derived from REQ21
UC-3. **Display map visualization:** Allow the user to view a map showing per capita data. Derived from REQ1, REQ4, and REQ5.
UC-4. **Display different data views:** Allow the user to choose between different sets of data that will subsequently be displayed on the map. Derived from REQ2 and REQ3.
UC-5. **Switch map to actual counts:** For certain data sets, allow the user to choose to switch from a per capita map to either a map that shows raw tweet counts, or to a map that estimates the actual counts of the US population as a whole. Derived from REQ3.
UC-6: **View Leaderboard**: Allow a Guest to see the State leaderboard of the States without being logged in to the system. Users will always be able to run this use case. UC – 1 will explicitly run to avoid screen clutter. Derived from REQ8.
UC-7 **View City Leaderboard:** Allow a Guest to select a State from a drop-down list and see the City Leaderboard for that State. Derived from REQ10

UC-8 **Change the activity:** Allow the guest and the user to select and change the activity from a list, then the markers on the map will be updated according to the new activity selected. Derived from REQ11, REQ13.
UC-9 **Change the place:** Allow the guest and the user to change the place to show another place on the map with the same activity selected. Derived from REQ12, REQ13.
UC-10 **Choose the location:** Allow the user to type the location to see the current weather Derived from REQ15.
UC-11 **Show Recommendation chart:** Allow the user to see a chart include the world's most popular activities. The sports are based on the data of the interactive map Derived from REQ16.
UC-12 **Show Calorie Meter:** Allow users to see the total number of burned calories done by people from a variety of exercises, from the historical data taking from tweets. Derived from REQ17, REQ18, REQ19, and REQ20.

4.3.2 Use Case Diagram

The whole system would have five subsystem, includes: Per Capita Map Visualization Subsystem, Gaming Subsystem, Interactive Activity Map Subsystem, Recommendation Subsystem, and Calorie Meter Subsystem. The use case diagram of the system would be displayed based on the boundary of each subsystem.

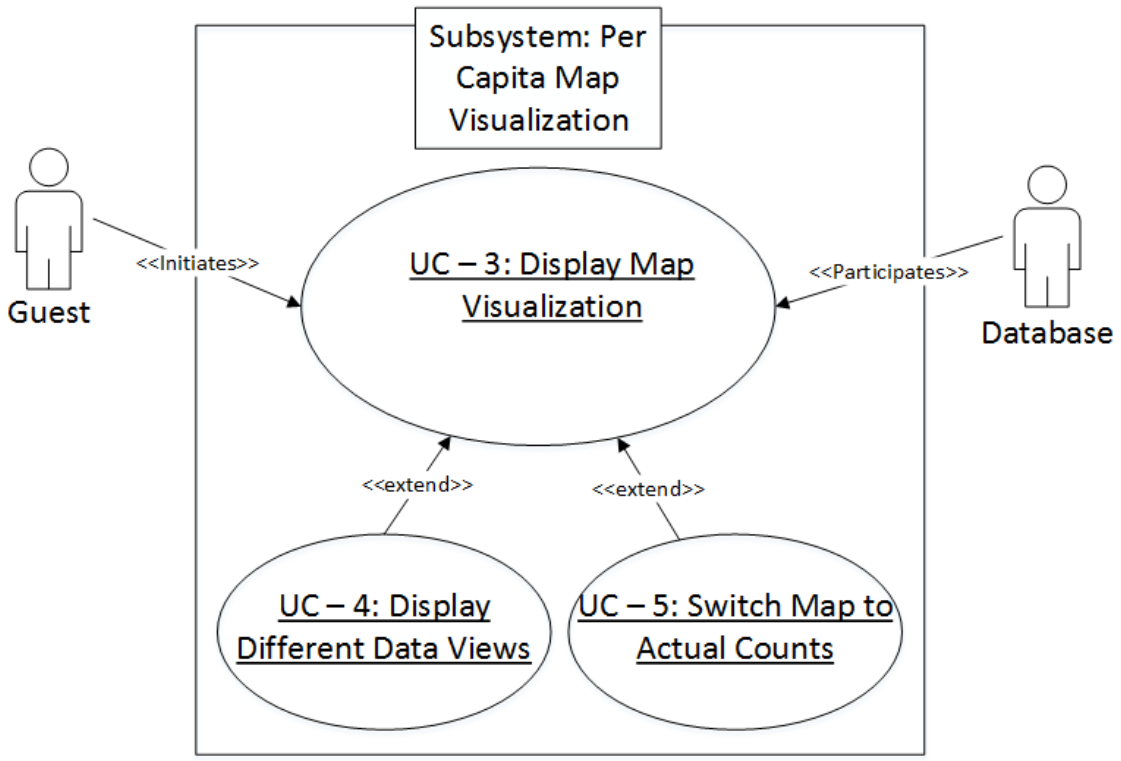

Figure 6 Use Case Diagram of Per Capita Map Visualization Subsystem

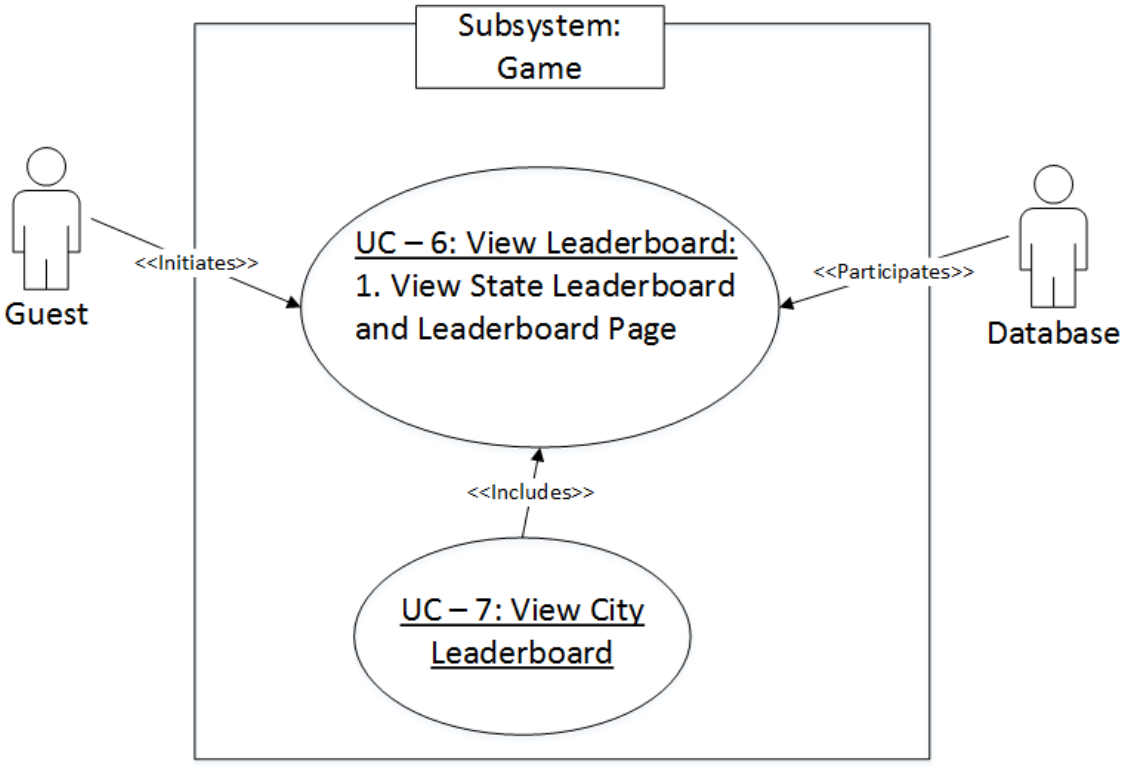

Figure 7 Use Case Diagram of Gaming Subsystem

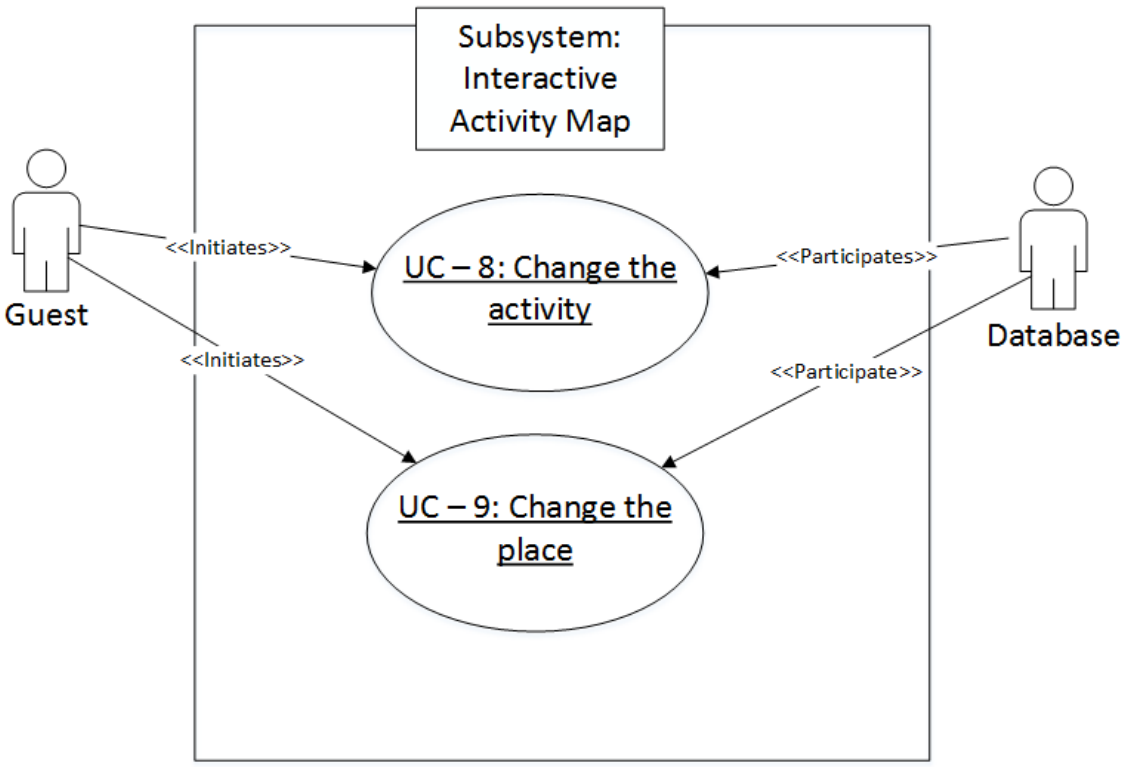

Figure 8 Use Case Diagram of Interactive Activity Map Subsystem

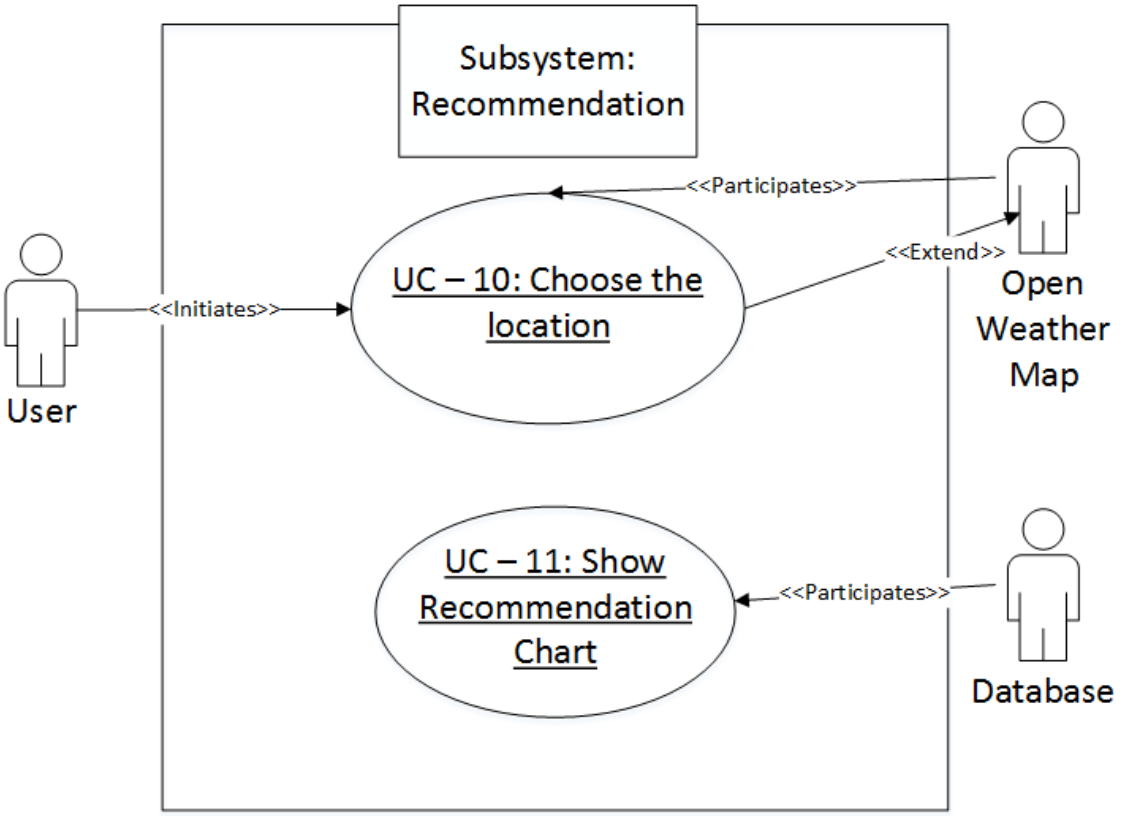

Figure 9 Use Case Diagram of Recommendation Subsystem

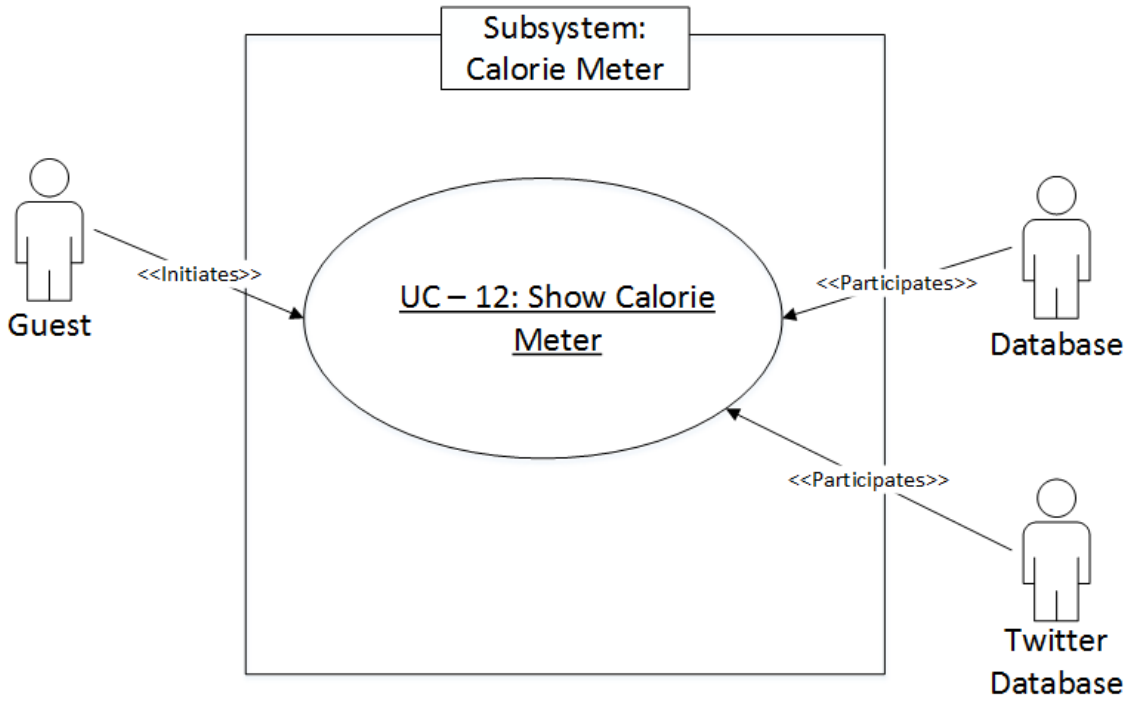

Figure 10 Use Case Diagram of Calorie Meter Subsystem

4.3.3 Traceability Matrix

REQ	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12
1	5			X									
2	3				X								
3	3				X	X							
4	4			X									
5	2			X									
6	1						X						
7	5												
8	2												
9	5												
12	4							X					
14	5								X				

15	3									X			
16	4								X	X			
17	2												
21	5										X		
22	5											X	
25	5												X
26	5												X
27	5												X
28	5												X
29	1		X										
30	1	X											
31	3												
32	4												
Max	PW	1	1	5	3	3	2	4	5	4	5	5	5
Total	PW	1	1	11	6	3	3	4	9	7	5	5	20

Table 4.2 Traceability Matrix

4.3.4 Fully-Dressed Description

Use Case UC – 3:	**Display Map Visualization**
Related Requirements:	REQ1, REQ4, REQ5
Initiating Actor:	User
Actor's Goal:	To visualize per capita data on a map
Participating Actors:	Database
Pre-conditions:	None
Success End Condition:	Map visualization is displayed
Failure End Condition:	Map visualization is not displayed
Flow of Events for Main Success Scenario: ← 1. System home screen displays map icon as one of the possible selections → 2. User/guest selects map icon. ← 3. System displays list of different data sets → 4. User/guest selects a particular data set. ← 5. System displays appropriate map visualization.	
Flow of Events for Extensions (Alternate Scenario): 3a. System displays an empty list 1. System notifies user that no data is contained in database and goes back to home screen in step 1.	

Use Case UC – 6:	**View Leaderboard**
Related Requirements:	REQ8
Initiating Actor:	Guest/User
Actor's Goal:	To see Leaderboard
Participating Actors:	Database
Pre-conditions:	None
Success End Condition:	View Game Web Page

Failure End Condition:	Game Web Page does not load. Either leaderboard does not load

Flow of Events for Main Success Scenario:

Include:: Use Case – 7

→ 1. User/Guest selects Game Web Page

← 2. System queries Database for the ten States with the most points

← 3. System loads the page with the State Leaderboard

← 4. System loads the City Leaderboard for New Jersey

→ 5. User/Guest selects a state from a drop-down list

← 6. System updates City Leaderboard with for the selected state

Flow of Events for Extensions (Alternate Scenario):

3a. System does not show State Leaderboard

← 1. System loads the page notifying the visitor that there are too few tweets to create a leaderboard.

Use Case UC – 8:	**Change the Activity**
Related Requirements:	REQ11 and REQ13.
Initiating Actor:	Guest/User
Actor's Goal:	To change the activity shown as markers on the map
Participating Actors:	Database, Google database
Pre-conditions:	The initial activity shown on the map will be the top activity in the initial location

Flow of Events for Main Success Scenario:

← 1. System shows map with an initial place with markers on this map according to the top activity in this location.

→ 2. User/Guest will click on the list to show the most top activities in this location and choose one from this list.

← 3. System will update the map with a new markers representing the new

activity chosen by the User/Guest.

Use Case UC – 9:	**Change the Place**
Related Requirements:	REQ12 and REQ13.
Initiating Actor:	Guest/User
Actor's Goal:	To change the place shown as markers on the map
Participating Actors:	Database, Google database
Pre-conditions:	The initial activity shown on the map will be the top activity in the initial location
Flow of Events for Main Success Scenario: ← 1. System shows map with an initial place with markers on this map according to the top activity in this location. → 2. User/Guest will drag the map to change the place in the map. ← 3. System will update the map with markers represent the same activity chosen by the User/Guest or initially chosen by the system (the top activity) for the new place chosen. ← 4. System will update the list according to the new place, but it shall show the same activity chosen before, after the User/Guest changes the place.	

Use Case UC – 11:	**Choose the location**
Related Requirements:	REQ15
Initiating Actor:	User
Actor's Goal:	To see the weather of one certain location
Participating Actors:	open weather map database, system, webpage
Pre-conditions:	None

Success End Condition:	Display the weather of that location
Failure End Condition:	Weather does not show correctly
Flow of Events for Main Success Scenario: → 1. User selects Recommendation Web Page → 2. User types certain location ← 3. System use the location to get data from Openweathermap API ← 4. System loads the page with the weather data	
Flow of Events for Extensions (Alternate Scenario): 3a. System does not show weather ← 1. System notify the user that the input of location may be wrong	

Use Case UC – 12:	**Show Calorie Meter**
Related Requirements:	REQ17, REQ18, REQ19, REQ20
Initiating Actor:	User
Actor's Goal:	Find number of burnt calories taken from people who posted related tweets.
Participating Actors:	Database, Twitter Database
Pre-conditions:	The server already searched related tweets from twitter online and stored them in our database.
Success End Condition:	Calorie Meter is displayed.
Failure End Condition:	Calorie Meter is not displayed.
Flow of Events for Main Success Scenario: ← 1. User visits the main page of our website. → 2. Server begins to get tweets from twitter database and transfer these related tweets to our database. System extracts and analyzes data and information in system database, then displays the total of burnt calories in a form of a counter.	

4.4 System Sequence Diagrams

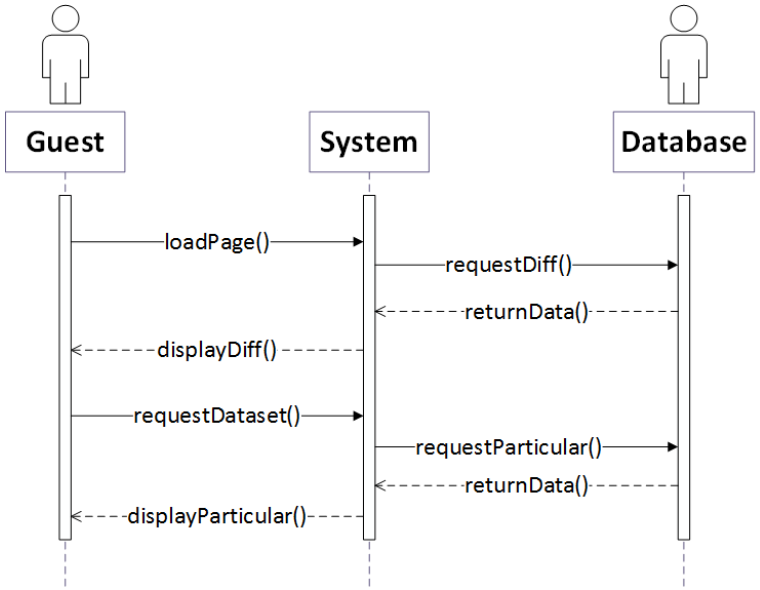

Figure 11 System Sequence Diagrams of UC-3

Figure 11 shows the System Sequence Diagrams of UC-3 which is the per capita map. When guest open the per capita map page, there would be an initial map displayed on the webpage. This initial map would include different kinds of data sets. Then when the guest choose a special data set, the system would request a particular dataset from the database, and then based on the return data to display a particular data set.

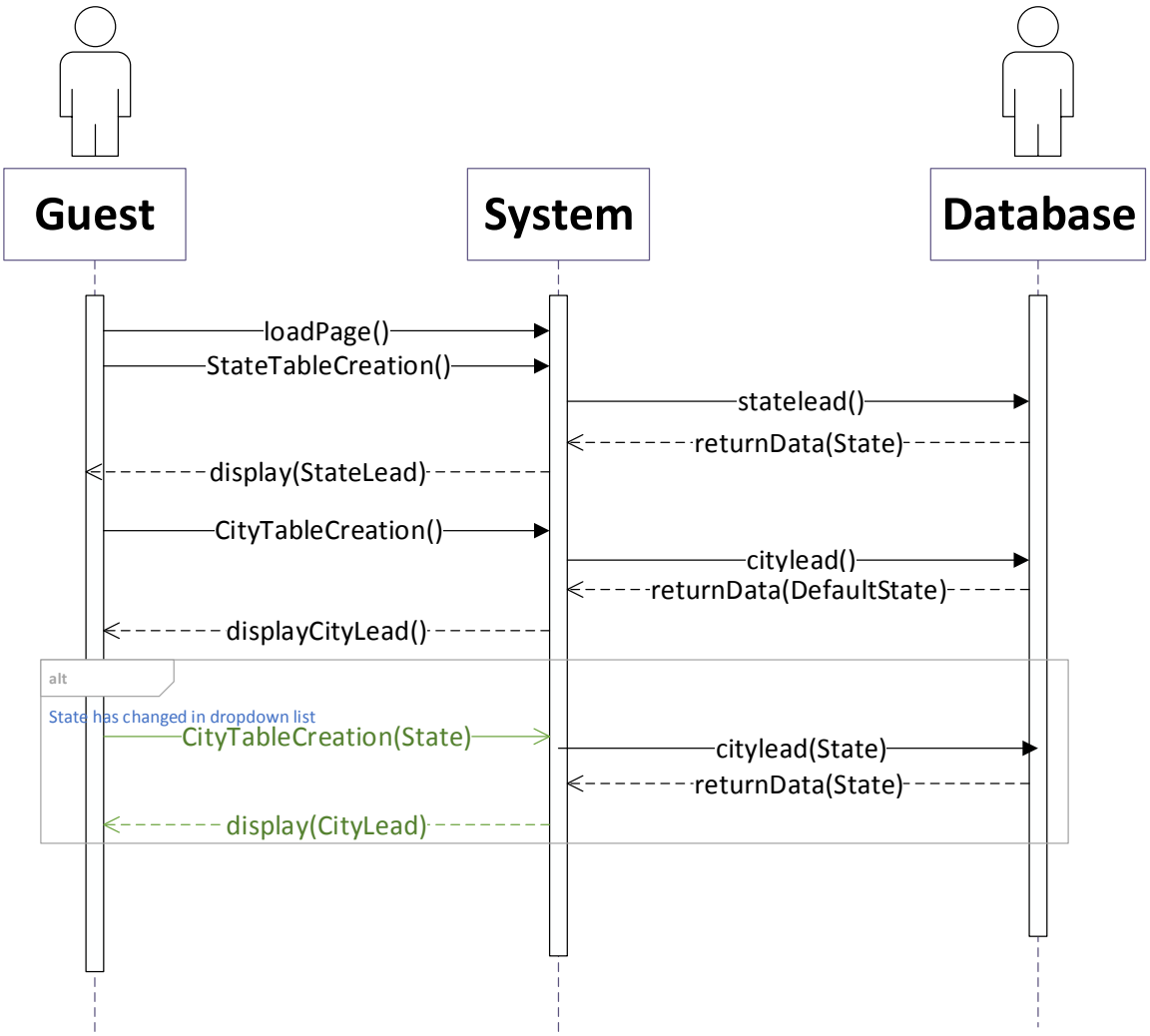

Figure 12 System Sequence Diagrams of UC-6

Figure 12 shows the System Sequence Diagrams of UC-6 which is the game. When a guest opens the play page, there would be an initial leaderboard displayed on the webpage. The system would request data from the database and display it to the user.

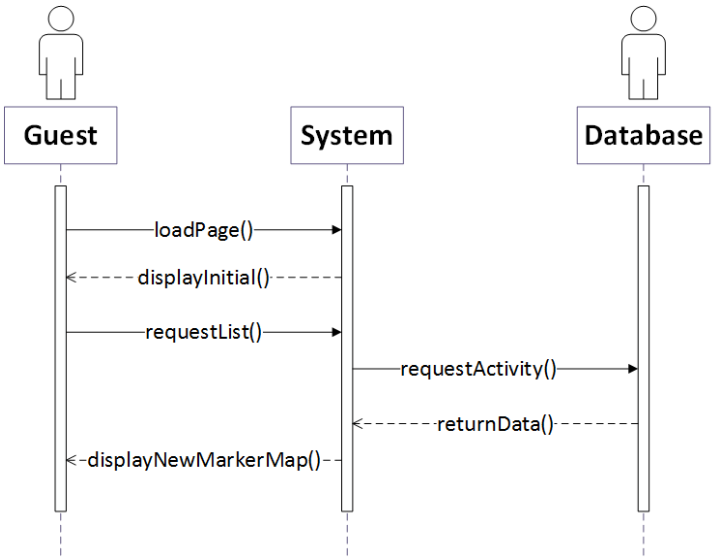

Figure 13 System Sequence Diagrams of UC-8

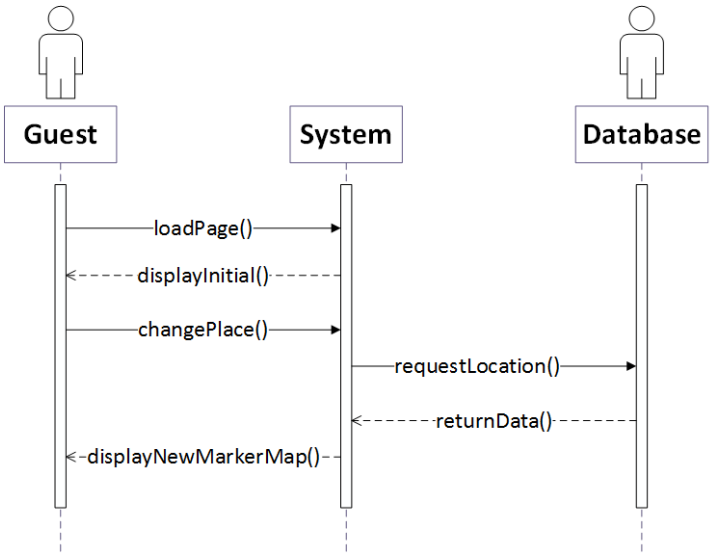

Figure 14 System Sequence Diagrams of UC-9

Figure 13 is the System Sequence Diagrams of UC-8, Figure 14 is the System Sequence Diagrams of UC-9. Both sequence diagrams belong to the interactive active map subsystem. The system would use the local database to collect data. Then based on the user's choice of sports, or upon moving the map location, the system would request different data from the database and then display them to the user.

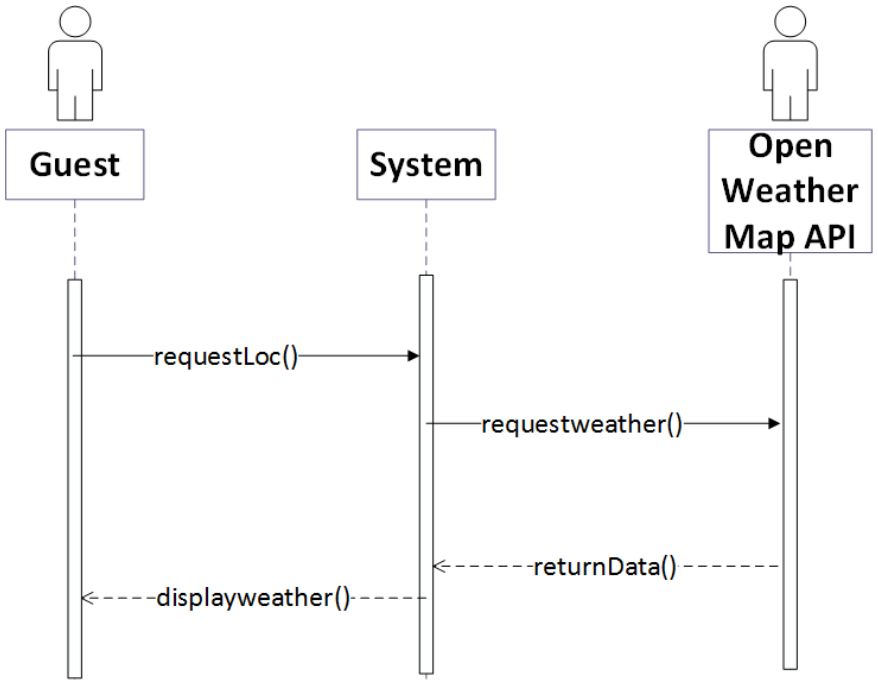

Figure 15 System Sequence Diagrams of UC-10

Figure 15 is the System Sequence Diagrams of UC-10. It is about the recommendation subsystem. The system based on a user's request, get information from Open Weather Map API. The API return the data to the system. The system would then use the data and display weather situation to the user.

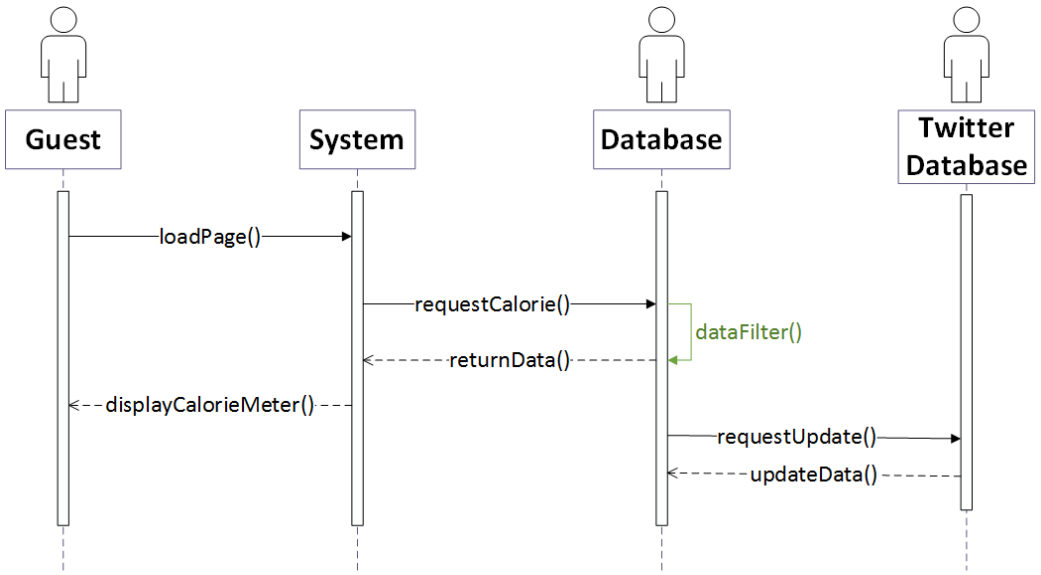

Figure 16 System Sequence Diagrams of UC-12

Figure 16 is the System Sequence Diagrams of UC-12. The system would show the user a

calorie meter when the user loads in the home page. The data would return from the local database, and the local database would update from the twitter database every 24 hours.

5.User Interface Specification

5.1 Preliminary Design

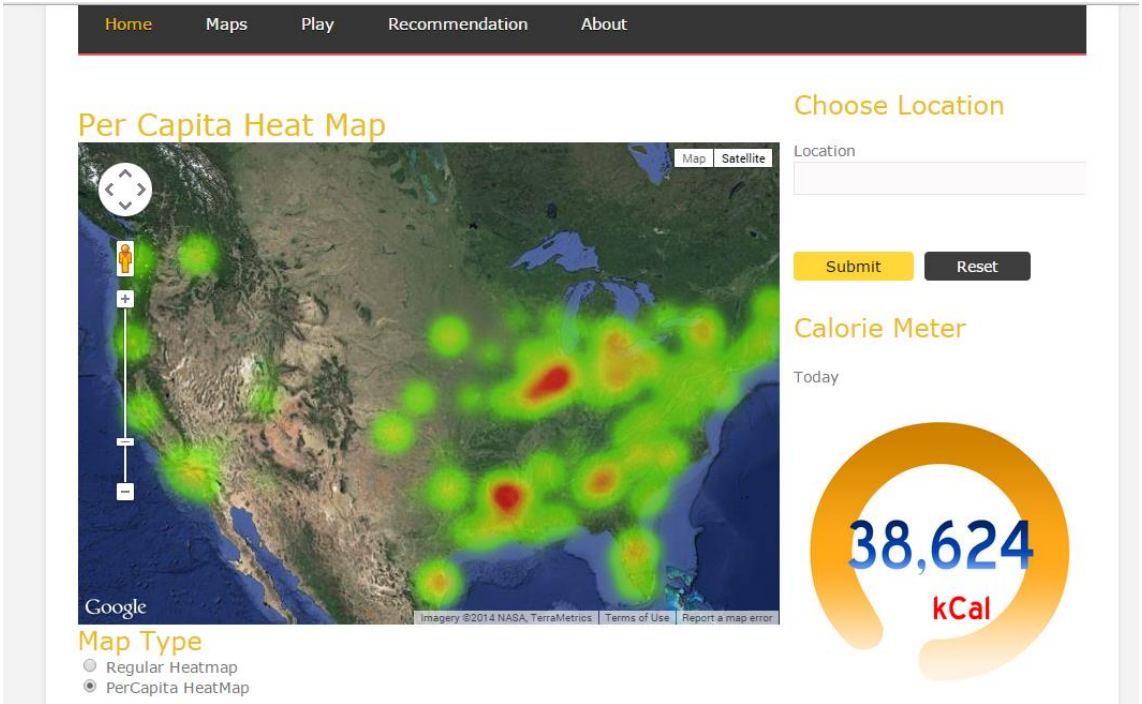

Figure 17 Home Page of the Website

Figure 17 is the home page of the website. It include the Per Capita Map. In the Per Capita Map a user will be able to select the data set that he/she would like to view, as well as the representation type of the data on the map. The Data Set will be a drop down list with only one possible selection. The Data Representation will be a list of radio buttons. The map will be fixed to the US. Upon any change in either list by the user, the map will refresh itself appropriately. Because the Estimated Actual Pop Data feature will only be available for some data sets, if it is selected when it cannot be used, an alert message will appear to prompt the user to select another data set or change the data representation type.

Figure 18 is the Map page. The Interactive Activity Map page would be very similar to the Per Capita Map page, The map would automatically display the 200 relative tweets around the center of the map.

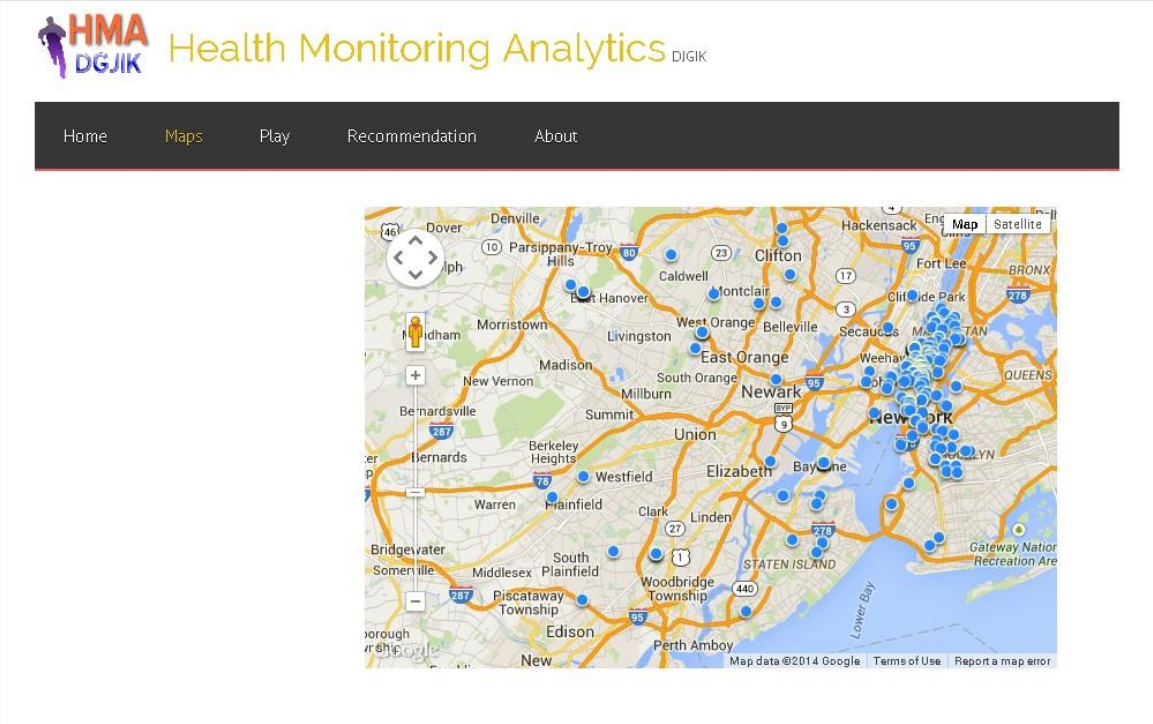

Figure 18 Map Page of the Website

Figure 19 shows the Play page. In this Gaming page on the left hand side shows the State Leaderboard which will always be displayed there. The leaderboard will be a list from 1 to 10, showing the 10 states with the most points with their points also. On the right hand side there will be a drop down list where all states will be presented. When the user selects a state, the state's points will be displayed along with a list of the top 10 cities of that state similar to the State Leaderboard.

In the Recommendation Page, shown by Figure 20, the user can type the city or place he or she lives. The right chart would automatically display the weather situation of the place. The left chart would display the world's most popular activities based on the data of interactive map. The user can compare the popular activities and the weather to make a sport decision.

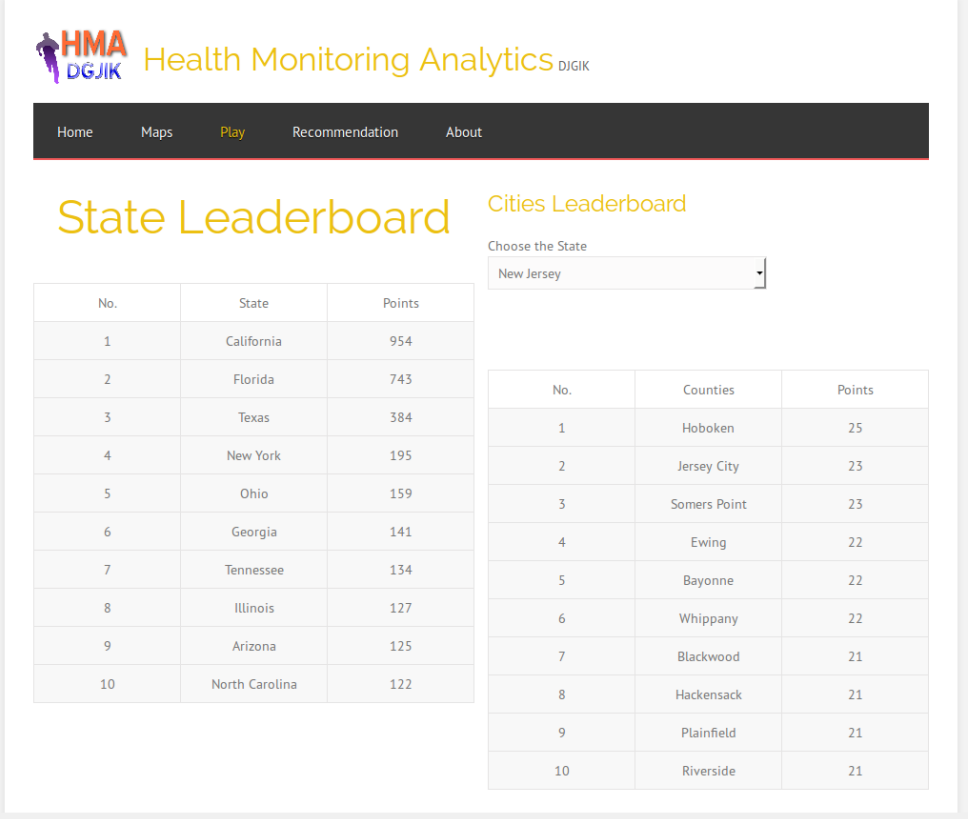

Figure 19 Play Page of the Website

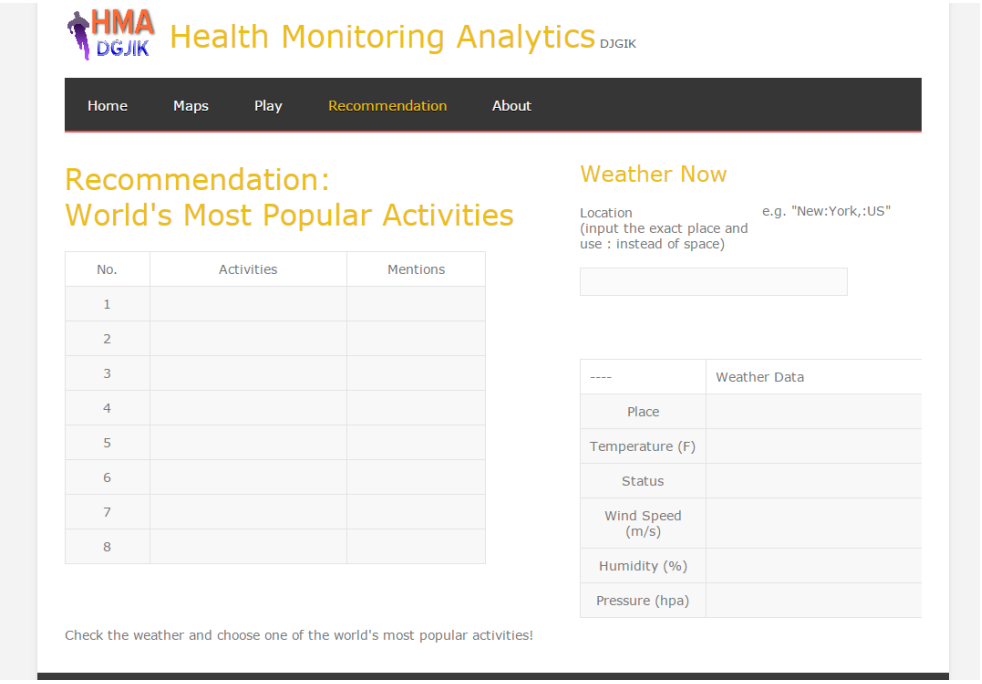

Figure 20 Recommendation Page of the Website

5.2 User Effort Estimation

Typical Usage Scenario 1: Guest wants to see a US heat map of per capita running activity.

(Total of 5 clicks)

A. From any page guest clicks “Map” on upper bar.

B. Guest clicks on “Per Capita Map” in upper left of screen.

C. Guest selects “running” data set from drop-down list.

D. Guest clicks radio button corresponding to heat map visualization to change map visualization type, and heat map of running activity is displayed.

Typical Usage Scenario 2: Guest wants to see the State Leaderboard and the City Leaderboard of the State he is living in

(Total of 3 clicks)

A. From any page, Guest clicks “Play” on the menu bar of the site

B. Guest selects the State he wants from a drop-down list on the right part of the page

Typical Usage Scenario 3: The Guest wants to see the Interactive Active Map for some kind of physical activity.

(Total of 4 clicks)

A. From any page, Guest clicks “Map” on the menu bar of the site

B. Guest clicks on “Customer Map” in upper left of screen.

C. Guest selects “running” from drop-down list.

Typical Usage Scenario 4: The User wants to use the Recommendation page to see the current weather of his location

(Total of 2 clicks and several keystrokes)

A. From any page, User clicks “Recommendation” on the menu bar of the site

B. Click the text bar and press several keys to type the “location”

Typical Usage Scenario 5: The guest wants to see the Calorie Meter:

(Total of 2 mouse clicks and several keystrokes)

A. Guest double clicks the cursor to "any Web Browser”

B. Guest presses several keys to type the “name of our website”

C. Guest presses the key “Enter” to navigate the Calorie Meter, which is presented in the Home Page of our website.

6.Domain Analysis

6.1 Domain Model

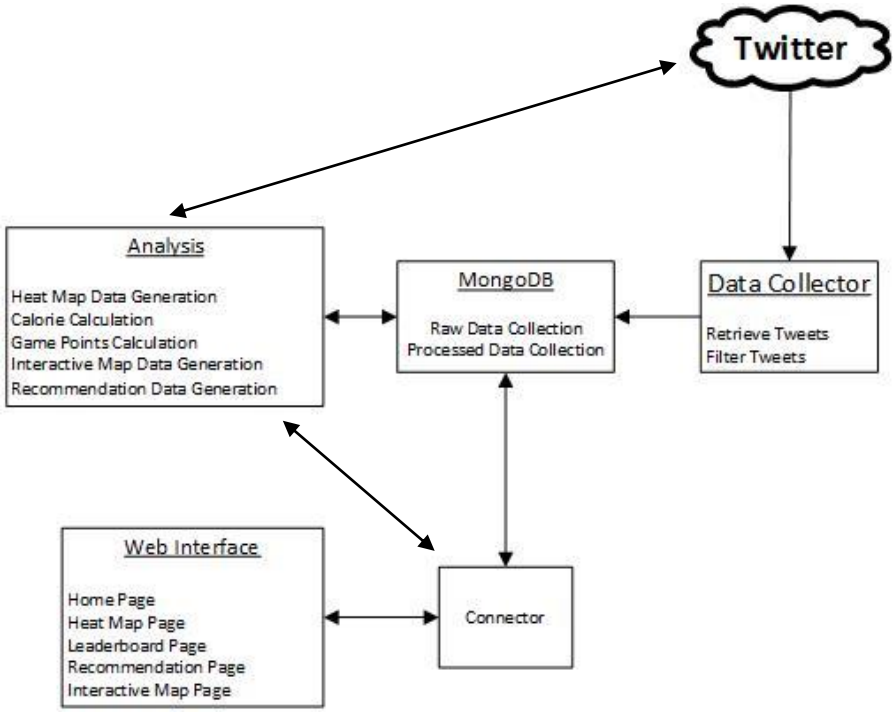

Figure 21 Domain Model

The Figure 21 shows the way we visualize our system as a whole. At the beginning, we did not believe that there was a need for the data collector or the connector, but if those two parts were missing the database would get data exactly as they are given from Twitter. The responsibility of separating the useful data would fall to the Analysis part and it would mean extra complexity for that part. Also, the Web Interface would have to be more complex since it would have to query the database for the necessary data and format them to the proper format for showing them to the Guest. Finally, in order to prevent extreme cases where multiple different access attempts are tried to our database, we divided them into two different databases based on the type of processing the data has gone through.

Our system can be divided into five distinct parts. Those are the Data Collector, the MongoDB, the Analysis, the Connector and the Web Interface. The Data Collector is the part that implements Twitter API to collect data according to the hash-tags, filter those tweets and finally store them in the Raw Data MongoDB Collection. There is a high probability that when very popular terms are used to retrieve data, many tweets will be irrelevant. Take for example the term “football”, it can be used in a tweet to show that

someone played football – relevant – but also can be used in a tweet that generally refers to football – irrelevant. Through the filtering process irrelevant tweets will be captured and discarded. After the filtering the rest of the data will be saved in the Raw Data MongoDB Collection. The MongoDB is the database of the system. It will two basic databases. The first database is the Raw Data which will contain the data that were just retrieved from Twitter and gone through the Data Collector and the second, the Processed Data, is the one that will contain the data created from the system's Analysis part. The Analysis is the part of the system that implements the system's features. It reads from the Raw Data database and writes data to the Processed Data database. The Connector is the part that reads data from the Processed Data database and gives them in the proper format to the Web Interface. The Web Interface is the only part of the system that is visible to the Guest and includes all the pages defined in the User Interface Requirements section. We reached this model by trying to make a logical division of the several parts of the system.

For the Caloriemeter and Recommendation part. The Analysis would connect the twitter API or weather API and the connector. They would not connect with the MongoDB database.

6.1.1 Concept Definitions

Table 6.1 Concept Definitions

Responsibility	Type	Concept	Use Case
R1: Store the tweets of different data sets, as well as the population data.	K	Database	UC-3 Display Map Visualization UC-8 Change the activity UC-9 Change the place
R2: Calculates Raw tweet data into efficient tables for quick Web Interface use.	D/K	Analysis	
R3: Changes web page via user clicks to navigate from home screen to interactive maps page.	D	Web Interface	
R4: Collect the user input for the display options.	D	Web Interface	
R5: Display the correct map of the specified data type.	D	Web Interface	
R6: Display Leaderboard Information	D	Interface	UC-6 View

R7: Knowledge of the points assigned to each City and State	K	Database	Leaderboard
R8: Check if points need updating and update according to the amount of new tweets per State for the States and per City for the Cities	D	Analysis	
R9: Store the new scores to the Database	D	Analysis	
R10: Existence of new tweets	K	Database	
R11: Web interface sends request to Database to get data.	D	Web interface	UC-11 Change Location
R12: Analyzing received data.	K	Database	UC-12 Show Calorie Meter
R13: Present data through web interface.	D	Web interface	

The Database is the part of the system that actually knows all the results from the Analysis subsystem. Responsibility R6 is performed by the Interface. When a Guest clicks the Leaderboard page, the interface will query the Database for the ten States and ten Cities from the selected State with the most points and display that information in a fashionable manner to the Guest. For this reason this Responsibility is a “Doing”, since it requires action from the interface.

By Analysis, we consider any type of processing to the raw data that our system receives from Twitter. The question here is how assigning points is part of the Analysis? Since the input of the system are raw data from Twitter some kind of analysis will take place to bring the data to the proper format. Also, since the Game in his system is to show which areas are those that their residents exercise the most, the results can also be considered as part of the Analysis. That is the reason that responsibilities R8 and R9 are assigned to the concept Analysis.

6.1.2 Association Definitions

Table 6.2 Association Definitions

Concept Pair	Association Description	Association Name
Data Collector ↔	Data Collector gets data from outside	Stores data

Mongo DB	database including Twitter, Google and Open Weather Map, and store the useful in the Mongo DB.	
Mongo DB ↔ Analysis	Analysis would do some calculation about the data in Mongo DB, and then store some useful information back into Mongo DB	Provides/ stores data
Mongo DB ↔ Connector	The Connector passes through some data from the web interface to the mongo DB. The Connector also passes through some data from the MongoDB to the web interface to do the display job.	Provides data
Web Interface ↔ Connector	Same as the last one.	Provides data

6.1.3 Attribute Definitions

Table 6.3 Attribute Definitions

Concept	Attributes	Attribute Description
Mongo DB (Database)	Send data request	Send data request to other database
	Crawl data	Crawl data from Twitter
	Return data	Return data to other database or web interface
Web Interface	Analyze request	Analyze request from Guest
	Heat Map Page	Display the heat map data on a map
	Leaderboard Page	Display the results from the Game
	Calorie Calculator	Display the calculated Calorie data
	Interactive Map Page	Display the Interactive Map data on a map
	Recommendation Page	Display the Word Cloud and the recommendation data
Data Collector	Tweet retrieving	Retrieve tweets from Twitter
	Filtering tweets	Filter the incoming tweets to discard noise

	Send tweets	Send the filtered tweets to the Raw Data Database
Connector	Retrieve Data	Retrieve data from the Processed Data Collection
	Receive Requests	Receive and satisfy Web Intaface's requests
Analysis	Heat Map	Calculated the heat map data
	Calorie	Calculate the burnt calories
	Game	Calculate the game points
	Interactive Map	Calculate the data for the Interactive Maps
	Recommendation	Calculate the data for the Recommendations

6.1.4 Traceability matrix

Table 6.4 Traceability matrix

Use Case	PW	Web Interface	Connector	Analysis	MongoDB	Data Collector
UC-1	1	X	X		X	
UC-5	1	X	X		X	
UC-3	11	X	X	X	X	X
UC-4	6	X	X	X	X	X
UC-5	3		X		X	X
UC-6	3	X	X	X	X	
UC-7	4	X	X			X
UC-8	9	X	X			X
UC-9	7	X	X			X
UC-10	5	X	X			X
UC-11	5	X	X	X	X	X
UC-12	20	X	X	X	X	

Max PW	20	20	20	20	11
Total PW	72	75	45	50	50

6.2 System Operation Contracts

ChangeMapType:
Preconditions: - Current page displayed is the map page.
Post-conditions: - Map type is supported by current data set- new map type is displayed with pre-selected data. - Map type is not supported by current data set- display alert

SelectInteractiveMaps:
Preconditions: - User is currently on any page of the website:
Post-conditions: - Interactive Maps is the current displayed page.

ChangeMapData:
Preconditions: - Current page displayed is the map page.
Post-conditions: - Map is updated displaying visualization of newly chosen data.

ChangeCountyLeaderboard
Preconditions: Current page displayed is the Leaderboard page. State Leaderboard is displayed and

County Leaderboard for default State is displayed.
Post-conditions: County Leaderboard of different State is displayed. There is not yet any data for this State – display alert.

ShowRecommendation:
Pre-condition: The server already searched related tweets from twitter online and stored them in our database.
Post-conditions: User asks for a recommendation on the Recommendation Page. Word cloud would not change until next request for recommendation

Show CalorieMeter:
Pre-condition: The server already searched related tweets from twitter online and stored them in our database.
Post-conditions: 1. User visits the main page of our website. 2. CalorieMeter displays the total of burnt calories in a form of a counter.

ChangeActivityType:
Preconditions: - Current page displayed is the map page.
Post-conditions: - The map will be updated with the new data from the database according to the new activity selected

ChangeMapPlace:
Preconditions: - Current page displayed is the maps present place.
Post-conditions: - The map will show us the new place with markers on it, according to the activity that is already selected.

6.3 Mathematical Model

6.3.1 Algorithm to Get Geo-coordinates for Location Names

Why It May Be Needed:

1. We need Geo Coordinates to create map data.

2. Only a small portion (roughly 1%) of tweets actually contain geo tags.

3. For many keyword phrases, we may have too little data if we are restricted to using only geo-tagged tweets.

If we could accurately map geo coordinates for location names (text strings), we could increase the amount of usable data for map based visualizations.

How it works:

Take the subset of all our collected tweets that contain geo tagged latitude and longitude.

Calculate the count of each unique location text string, such as “Windy City”, or “The Big Easy”.

For each unique location with a count greater than some small minimum number (say 8), we do the following.

For both the latitude and longitude, we trim off the bottom X% as well as the top X% of values. The amount trimmed off X% should be large enough to get rid of the outliers, which would be tweeters listing their location as somewhere else than where they are actually at. But, it should be big enough that in the event that there are 2 significantly sized locations with the same name, some of the data from the smaller like named city would still remain. (X could probably be reasonably set at 3%)

Then we find the median (could use average if median is too computationally expensive,

probably not though) and standard deviation of both the latitude and the longitude of each unique location string. Then if the standard deviation of the trimmed data, for both the latitude and the longitude is less than y miles (for some chosen y, potentially between5-10), we conclude that the data is in fact clustered around a specific geographic area. We would store the median latitude and median longitude as the geographic coordinates for that particular location name, in a separate look up table.

Then whenever we pull tweets in the future, for any tweets that are not geo-tagged, we can look up the coordinates for that location string and we will have geo location info for that tweet.

When a particular location name would fail our test:

If people listed their location as something other than a specific location, such as “Earth”, “North America”, “USA”, “The South”, then we would see the geo locations of the data spread out all over the map, and the standard deviation of the data latitude and longitude would be much higher than our cutoff value of y miles. Therefore, for these tweets, we would not be able to infer accurately the geo coordinates.

Also, if there were 2 significantly sized distinct locations, with the same name, then even after the x% trimmed, the remaining data from both locations would make the standard deviation too large to pass our cutoff test.

When This Algorithm May Potentially Not be required:

If Google API could both easily recognize slang names for cities, as well as allow us a very high number of look up attempts, then there would be no need for this algorithm. Also, if we are unable to get a large enough count of tweets that contained geo coordinates, then this algorithm would not be very effective.

6.3.2 Slecting Keywords/phrases

1. Finding balance between “high success” and high number of tweets

There is going to be a tradeoff between finding “high success” search phrases/keywords, and finding phrases/keywords that show up in enough tweets to give us enough data for accurate analysis.

For instance, compare “I played basketball” with “basketball”. We will probably get 1,000 times more tweet data containing just “basketball”, but the majority of tweets containing “basketball” will have nothing to do with that person actually exercising by playing the game. Conversely, the phrase “I played basketball” will have relatively few tweets, but it will be very “successful” in indicating that that particular person actually exercised by playing basketball.

The problem when using keywords that have low “success”, is that other factors not related to exercise will influence the regional frequencies of tweets containing that word. For instance, if the New York Knicks are in the National Basketball Championships, then New York City will have an unusually high rate of tweets containing “basketball”. We might incorrectly conclude that New York residents are exercising more by playing more basketball. On the other hand, the phrase “I played basketball” should not increase in frequency even if the local professional basketball team is in the spotlight.

This should highlight the problem associated with previous projects, in that they used many low success keywords that were subject to fluctuations from extraneous factors that have nothing to do with exercising.

2. Using “milestone” keywords

Certain words can be used to ascertain whether or not a person has met or exceeded a certain threshold. For instance, people tweeting the phrase, “did pullups”, if the phrase is “successful” at determining that that particular person performed pullups, shows that whoever tweeted this is at least strong enough to perform one pull up. Therefore, we can look at the data of people who tweet this to get an idea of the proportion of people in an area that have reached or exceeded this level of strength to bodyweight. This, in turn, could be used to compare strength differences between locations.

We could also use many other “milestone” phrases such as:

“I ran”+”mile”, “I did”+”pushups”, “I ran”+”marathon”…etc.

Notice that pullups require much more strength than pushups, so someone doing pullups would imply that they are also strong enough to do pushups, but the converse is not necessarily true. Also, the 2 phrases for running both would be a measure of cardiovascular health. But notice that running a marathon would imply that the person could run a mile, but running a mile does not necessarily imply that the person can run a marathon.

So while pushup, and marathon phrases would show a much higher level of conditioning, we would most surely get less people that could perform those and therefore have less data from which to draw conclusions.

On an even lower scale, we could choose “situp”, instead of “pushup” or “pullup”, and we could choose “I walked”+”mile” instead of “I ran”+”mile”. But notice that probably 95% of the population could do a situp or walk a mile, so if we use these to try and measure overall strength or cardiovascular fitness levels (in this milestone case we are focused on fitness levels, not workout frequency! People could walk often but still not have a high level of fitness) we would not be able to precisely discern whether one group is more fit than another.

Ideally, we would want to find a measure that, on average, is achievable by 50% of the population. Why 50%? Well we can view differences in the percentage of the population that can perform an certain activity, as shifts in entire distribution or "bell curve" of ability (in this case strength). Note that all we can tell is a binary "can do" or "can't do". So if we pick an activity that on average is done in the central point of the data, at the 50% mark, then if one location is has a standard deviation of plus one over the average that can perform that activity (assuming normality), than we should see a difference of about 15-20% between that location and the average location.

But imagine if we instead of choosing an activity that on average 50% of the total population can do, we choose an activity that 98% of the population can do. Well, then if the same area as before is stronger than the overall population by one standard deviation, then we will see that stronger location should show about a 99.5% "can do" rate. But because the differences between these two percentages are so low, we cannot accurately measure them without a very large set of data.

So in conclusion, we cannot choose a binary "can do", "can't do" variable that is either doable by the majority of the population, or conversely, not doable by the majority of the population.

6.3.3 Use per capita data to analyze exercise activity:

1.Using Ratios as Opposed to Count Data for Analysis

This will be the inspiration for one of our project features below.

a. Trying to compare data from a Small Town to data from a Big City

The previous groups limited themselves to using count data from tweets. Meaning that they first picked keywords/phrases, and then simply counted up the number of tweets that contained these phrases, and tried mapping them to a geographic location.

What is the problem here? Well, imagine that for one year's data, we counted up the total number of tweets in New York City that contained "I shoveled snow". Then we compared this to the number of tweets from say, Fargo, North Dakota, with the same phrase. We might see that 200,000 tweets with that phrase came from New York City, and 100,000 came from Fargo, ND.

What could we conclude? Well, absent all other information we might incorrectly conclude that it snows more often in NYC. Or that people are more willing to shovel their snow when it snows in NYC. But if we take a look at the total populations, we get a different story. The population of Fargo is 113,000 while the population of NYC is 8.3 million. Notice that if Fargo had as many occupants as NYC we could expect

100,000*80=8 million tweets containing those words!

Therefore, we can conclude that there are some factors, (MOST LIKELY COLD WEATHER AND SNOW!) that are causing the amount of tweets to the population level to be so high. Notice that if Fargo had 5,000 tweets with the phrase "I went jogging" and NYC had 50,000 during the same time frame, then the "tweet-population-ratio" is 5,000/113,000=0.044 for Fargo and 50,000/8.3m=0.0006 for NYC. This would be a huge indicator that people in Fargo go jogging much more than people in NYC on a per-capita basis.

b. How to Analyze Data on a Per-Capita Basis

Assumptions:

1) The amount of tweets from a location are linearly related to the population.

For the sake of simplicity we can assume that location has a limited impact on both the percentage of the population that uses twitter, and the average rate of tweets per twitter user.

Ideally, we could get the count of all tweets by each zip code, and calculate our ratio from that. We would probably need at least several tens of millions of tweets in order to get an estimation of the total twitter output by area. However, we have found US population data by location, which we can use as a proxy to "relative total" twitter output by location, along with our assumption above.

2) We can find keywords/phrases that both have a high success rating, as well as being present in enough tweets that we can have large enough data to draw conclusions.

3) Twitter users are equally likely to tweet about their health and fitness regardless of their geographic location.

For instance, if certain cities had a culture of tweeting all their activities, while others had a culture of not mentioning their exercise, then the data would be less reliable. But it seems that it is a very bold claim that (AT LEAST WITHIN THE USA) different areas would be more prone to tweet about their fitness and health activities than other locations. So we can assume there is no difference.

Implementation:

When we are collecting tweet data, for each keyword/phrases, and for a fixed time interval, we collect the count of the number of tweets from a city and divide it by that city's population. This will give us one ratio for city x, and another ratio for city y. Notice that the ratio itself is meaningless. For example, if 7 tweets per 1,000 people contain "I went jogging", this by no means implies that 0.7% of people jog. However, if in city x every 20 tweets per 1,000 people contains "I went jogging" and in city y every 45 tweets per 1,000 people contains "I went jogging", then we can make a very strong case

that not only in city y do people jog more, but using our assumptions above, we can estimate that the population goes jogging about 2 times as much in city y than x.

c. Getting Accurate Count Data from Per-Capita Ratio Differences combined with aggregate official US government data.

There are many available sources provide aggregate nationwide data for health and fitness. For instance, we can probably get accurate figures for the percentage of Americans that bike regularly. Let's say according to the government figures, (A reliable source) 5% of Americans bike regularly. Now let's go ahead and search the entire country for tweets from one a week that contain "I rode my bike". Let's say that we collect 500,000 matching tweets (this is just a fictional example and could be completely different from reality) throughout the country.

Now set the average ratio of tweets to population at 500,000/300mi= 5/3,000 with all of our earlier assumptions we can now say that 5/3,000 is the national corresponding ratio to 5% of Americans bike regularly. Imagine if in Tulsa, Oklahoma, we see that for their population, every week, the phrase "I rode my bike" shows up about 3 times for every 3,000 people. Then we could make a strong case that (3/3,000)/ (5/3,000)*5% = (3/5)*5%=3% of Tulsans bike regularly. Also, if NYC average 7 tweets with that phrase per every 3,000 people, then we can make a strong case that (7/3,000)/ (5/3,000)*5%=7% of New Yorkers bike regularly. Then we can translate this to actual figures by multiplying 7%*8mil=560,000 New Yorkers bike regularly.

What we have done is take aggregate data from a trusted source, take per capita tweet frequencies of arbitrary phrases, and use regional differences of the tweet frequencies per capita to find regional, actual data.

Drawbacks: It may be very difficult to gather enough data for many locations to be even reasonably, statistically significant. We very well may only be able to implement this using the larger cities.

7. Interaction Diagrams

7.1 Sequence Diagrams

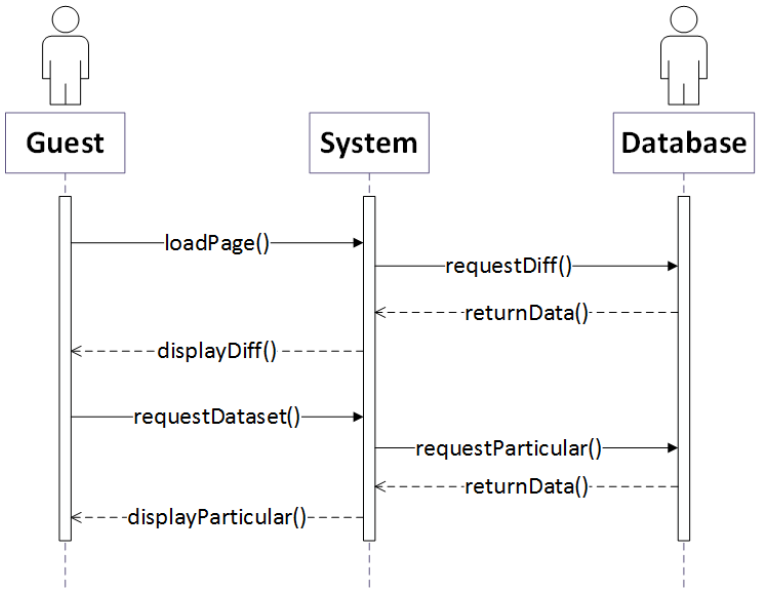

Figure 22 System Sequence Diagrams of UC-3

Figure 22 shows the System Sequence Diagrams of UC-3 which for the per capita map. When a guest opens the per capita map page, there would be an initial map displayed on the webpage. This initial map would include an option for different kinds of data sets. Then when the guest chooses a particular data set, the system would request that particular dataset from the database, and then use the returned data to display in the map.

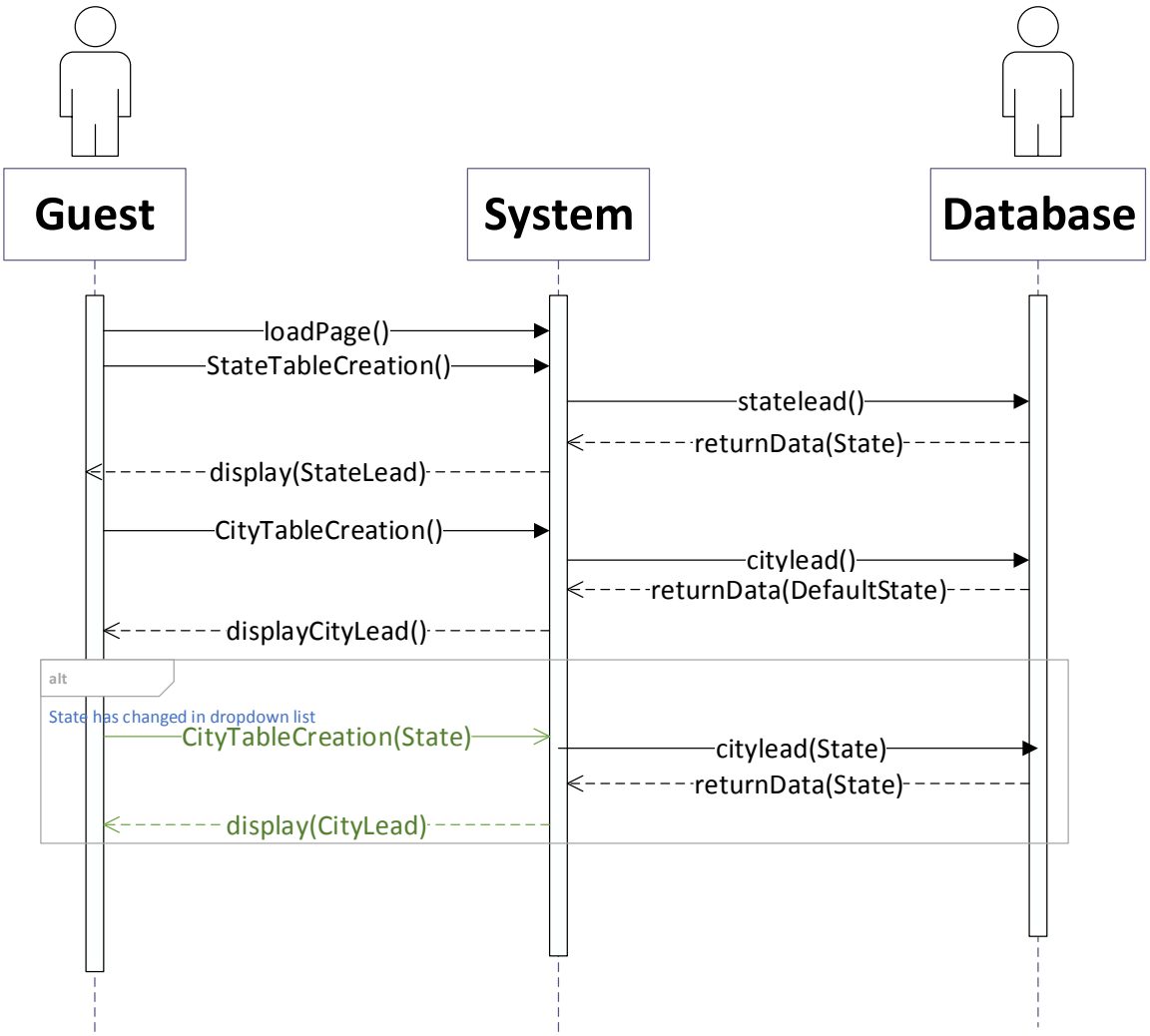

Figure 23 System Sequence Diagrams of UC-6

Figure 23 shows the System Sequence Diagrams of UC-6 for the game. When a guest opens the play page, there would be an initial leaderboard displayed on the webpage. The system would request data from the database and display it to the user. Also, an asynchronous call happens if the user has selected a different State from the default, which is New Jersey, for the City display. If the selected state is different, the page sends a request to the system with the new State. The system then queries the database for the City Leaderboard of the selected State. On the page load, the system automatically queries the database for the New Jersey City Leaderboard. In both alternatives, after the sequence of requests and queries, the results are displayed on the page.

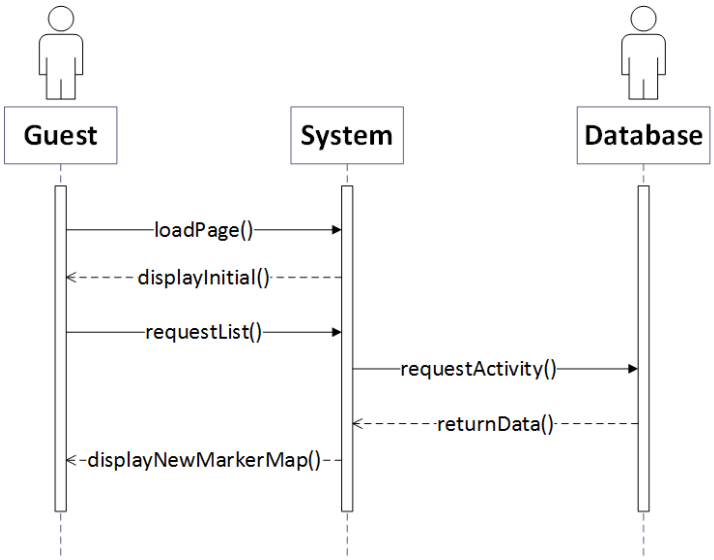

Figure 24 System Sequence Diagrams of UC-8

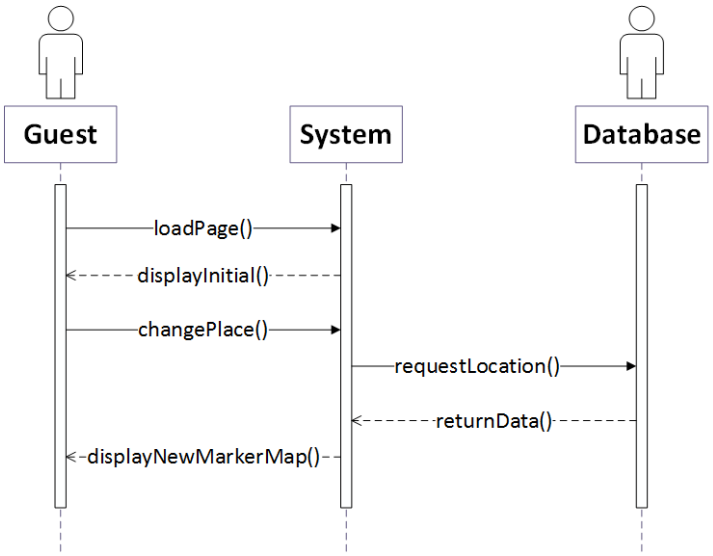

Figure 25 System Sequence Diagrams of UC-9

Figure 24 is the System Sequence Diagrams of UC-8, Figure 25 is the System Sequence Diagrams of UC-9. Both sequence diagrams belong to the interactive map subsystem. The system would use the local database to collect data. Then based on the user's choice of sports, or upon moving the map location, the system would request different data from the database and then display that particular data to the user.

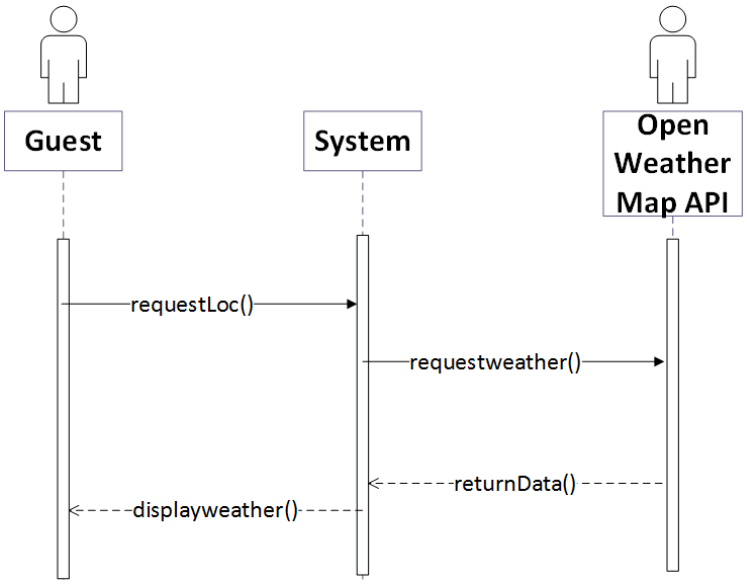

Figure 26 System Sequence Diagrams of UC-10

Figure 26 is the System Sequence Diagrams of UC-10. It is about the recommendation subsystem. The system based on a user's request, get information from Open Weather Map API. The API return the data to the system. The system would then use the data and display weather situation to the user.

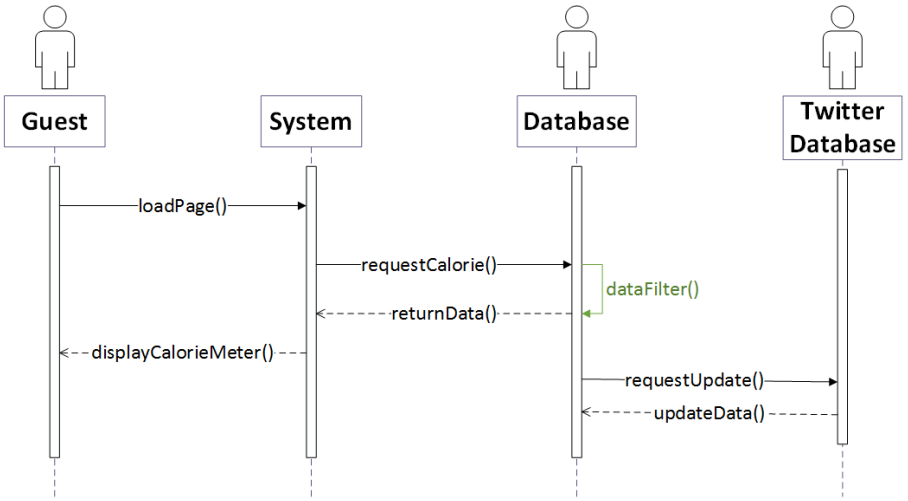

Figure 27 System Sequence Diagrams of UC-13

Figure 27 is the System Sequence Diagrams of UC-13. It is for the Calorie Meter subsystem. This subsystem would show the user a calorie meter when the user loads in the home page. The data would return from the local database, and the local database would update from the twitter database every 24 hours.

8. Class Diagram and Interface Specification

8.1 Class Diagram

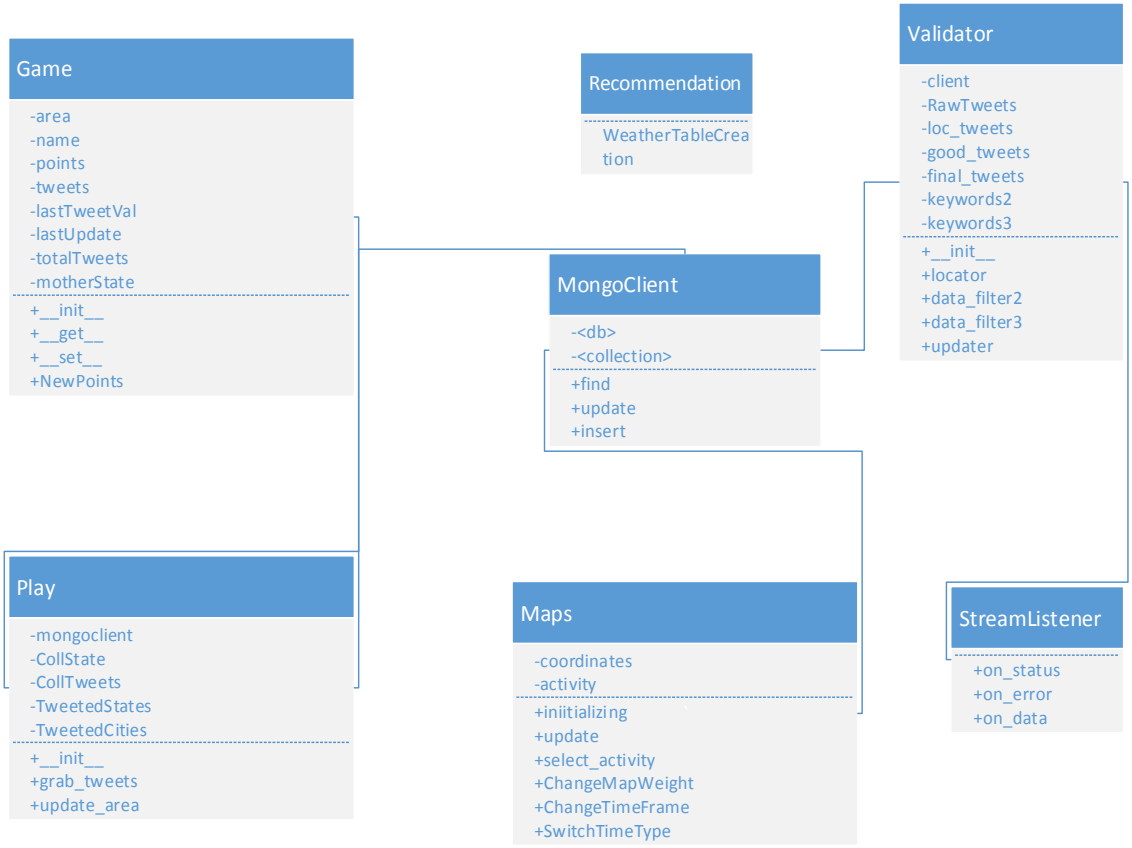

Figure 22 Class Diagram

8.2 Data Types and Operation Signatures

StreamListener:

This is the class that creates the link between the system to be and Twiiter Stream API. In this part of the document only the methods that are used will be explained, because we have not used any attribute from this class. For more information the reader is referenced to [reference to tweepy]

- on_status(self, status) : Called when a new status arrives
- on_data(self,raw_data): Called when raw data is received from connection.
- on_error(self, status_code) : Called when a non-200 status code is returned

Validator:

This class is responsible to receive the raw data, find which tweets are actually valid or

trusted information based on the criteria the system has and then save them inside the MongoDB. The attributes of the class are:

client : This is the actual connection with the MongoDB

RawTweets : This is a list that has all the tweets that were grabbed more resently from Twitter

loc_tweets : A list of the Geo Located tweets after the locator method is executed

good_tweets : A list of the tweets that the first filter let to pass

final_tweets : A list of the tweets that the filter filter let to pass

keywords2 : A list with the words used as the filtering terms for the first filter. It is read from keywords2.txt file

keywords3 : A list with the words used as the filtering terms for the first filter. It is read from keywords3.txt file.

The methods of this class are necessary to utilize the reception of new tweets and preprocess them in order to remove any "noise" that exists. "Noise" is any tweet that, although having the search terms in it, is not relevant to actual exercise or its location is not recognized.

__init__ : The constructor of the class must be present for python classes.

locator : This method checks all tweets and returns only those that are geotagged or uses the location of the user to decide the coordinates of a tweet.

data_filter2 : This method searches through all the Tweets for each word in the keyword list. If a keyword exists in the tweet's text and the tweet has not been added to the final list by a previous keyword success it is added to the tweets that will be returned.

data_filter3 : This function searches through all the Tweets for each word in the keyword list. If a keyword exists in the tweet's text and the tweet has not been discarded from the final list by a previous keyword success it is discarded from the tweets that will be returned.

updater : This method just updates the database with the new filtered and located data.

We have to note that this class was implemented after the second demo. During the demo we used data that were derived by using the procedural functions from the data_filtering file. We created the functions first because we needed to see whether an implementation of these functions can provide us with the additional information or not.

MongoClient:

This class represents the actual connection between the MongoDB and the rest of the system. The attributes of this class are the actual contents of the database. For example, if a database named raw_data exists in MongoDB there will be an attribute with that name. For that reason the attributes listed below are generic but give the basic idea of each attribute. For more information about MongoClient class the reader is referenced to [reference to pymongo]

- <db> : These attributes are the several databases that exist in the MongoDB server that is installed. It is an object of the pymongo Database collection.
- <collection>. These attributes are the collections that exist in each <db> in the MongoDB server that is installed. It is an object of the pymongo Collection class

The methods listed are the ones that the system uses to grab documents from the database, update them and insert new.

- find(self, *args, **kwargs) : This method is used to query the database with a specific type of query as it is defined by MongoDB. It returns a python list of python dictionaries.
- update(self,spec,document,upsert) : This method is used to update a specific document in the selected collection in the database. The parameters are

  -- spec : a dictionary specifying elements which must be present for a document to be updated

  -- document : a dictionary specifying the document to be used for the update.

  -- upsert : Boolean type (optional). When True, insert if object doesn't exist.

- insert(self, doc_or_docs) : a list of documents to be added in the database

Game:

This class is the one that actually creates the game feature of the system. An object of this class is constructed every time the points of an area must be updated. The attributes of this class are:

- area : The type of the area. It can be either 'State' or 'County'. This is a string
- name : The name of the Area. This is a string.
- points : The Number of points that the area has. An integer number.
- tweets : The number of unused tweets for this specific area from the last update. An integer number
- lastTweetVal : The number of Tweets that were used for the last update. An

integer number
- lastUpdate : This is the date of the last update with tweets. A list of integers that represent the following [yyyy,mm,dd]
- totalTweets : The total number of tweets. An integer number.
- motherState : In case of a County in which State it belongs. This attribute is a string with the name of the State that the County is in.

The methods of the class are:

- __init__(self,area,name,points,tweets,lastTweetVal,lastUpdate,totalTweets,motherState) : Class Constructor. The constructor initializes the class' attributes. The parameters are directly connected to the attributes with the same name.
- __get__(self,attr) : returns the value of an attribute. Parameter attr is a string and should agree with the attributes name. If the attribute does not exist the method returns 0.
- __set__(self,attr,val) : Sets the attribute with name attr with the value of val. attr is a string and val has the same data type of the specific attribute.
- NewPoints(self) : The basic point update method. It updates the points of an area based on how often it is updated, when tweets from that area are retrieved, what impact those new tweets have in the total number of tweets from that area and what is the difference from the last update.

Play:

This Class is the Class that connects MongoDB with the Game class. Contains all the necessary methods to enable the game. The reason that a different class exists to utilize the connection of the Game class and the database is to make the Game as independent as possible with the used database. The attributes of this class are:

- mongoclient : This is the actual connection with the MongoDB. This is a MongoClient object.
- CollState : This is the connection with the Collection that holds the data for the Game over the States. This is a pymongo Collection object
- CollTweets : This is the connection with the Collection that holds the data from the tweets pulled from Twitter. This is a pymongo Collection object.
- TweetedStates : This attribute has a list of the states mentioned in the Tweets. It a list of strings.
- TweetedCities : This attribute has a list of the states mentioned in the Tweets

The methods of this class are:

- __init__ (self): The constructor of the class must be present for python classes. Does not need any parameters.

- grab_tweets(self) : This method is to utilize the existing connection with the database and get the needed data. Does not need any parameters.
- update_area(self) : This method is used to call the game class and update the database entry for that specific area.

Connector

The role of this class is to get the data from the mongoDB and push it to the web interface through the web server as a HTTP messages.

Variables of this class:

-db : object repesenting the database name in the mongodb.

-coll: object representing the collection name in the mongodb.

-cursor: string representing the query data.

-doc :string collecting all the cursor data.

-ci: integer for iteration loop

-red: JSON format to push it

-i: integer for iteration loop

-File name: string representing the javascript and html file names.

-pubsub: JSON format to be pushed to the map

-st: string to get to run the mongodb function.

Functions of this class:

-Mongodb: used to get the queried data from mogodb.

-stream: used to push the data to the webpage map.

-tweets: creates the http links for files.

-runThread: excutes the Mongodb function.

Maps

The roles of this class is to get the data from mogodb and then send it to the maps (per capita and Interactive).

Variables of this class:

-Coordinates: float represents the coordinates of the tweets.

-Activity: string represents the top activity type.

Functions of this class:

- Initializing: Initializes the google map configurations.

- Update: updates the data for the maps

- select_activity: takes the new activity selected and requests an update according to the new one

- ChangeMapWeight: redraws data on the map using the new weighted data method.

- ChangeTimeFrame: changes the time frame used for the new algorithm.

- SwitchTimeType: switch between the time frames used above.

Recommendation

Recommendation is a separate part of the main system, it use a python file to get the weather information from the Openweathermap and one php file as a connector to put the information on the webpage

-WeatherTableCreation: it calls the php file to get the data from the openweathermap API when the user type anything in the text bar.

8.3 Traceability Matrix

Table 8.1 The Traceability Matrix of class and domain concepts

Domain Concepts	StreamListener	Validator	MongoClient	Game	Play	Recommendation	Maps
R1	X	X	X				
R2			X				
R3				X	X	X	X
R4					X	X	X
R5			X				X
R6				X			
R7			X	X		X	
R8				X		X	
R9			X				
R10	X	X	X				
R11			X				
R12				X		X	

To understand this traceability matrix, here is the description of each R1-R13:

R1: Store the tweets of different data sets, as well as the population data.

R2: Calculate the raw tweet data into efficient tables for quick Web Interface use.

R3: Change the web page via user clicks to navigate from home screen to the interactive maps page.

R4: Collect the user input for the display options.

R5: Display the correct map of the specified data type.

R6: Display Leaderboard Information

R7: Store knowledge of the points assigned to each County and State

R8: Check if points need updating and update according to the amount of new tweets per State for the States and per County for Counties

R9: Store the new scores to the Database

R10: Existence of new tweets

R11: Web interface sends request to Database to get data.

R12: Analyze the received data.

R13: Present data through the web interface.

9. System Architecture and System Design

9.1 Architectural Styles

Our system has a server client architectural model. The reason for this, is that our system is based on a website that is connected to a database. Therefor, we need the server and the client to be connected on the internet. The server would include the data storage and data calculations. There would be a database as well. For us it is the MongoDB, to store all the information and do several calculations based on the database. The client would communicate with the server effortlessly and display the server's data in a certain way. The communication would be done by JSON. This server client architecture can help us to achieve our project goal which is a website that can display calculated data from twitter.

9.2 Identifying Subsystems

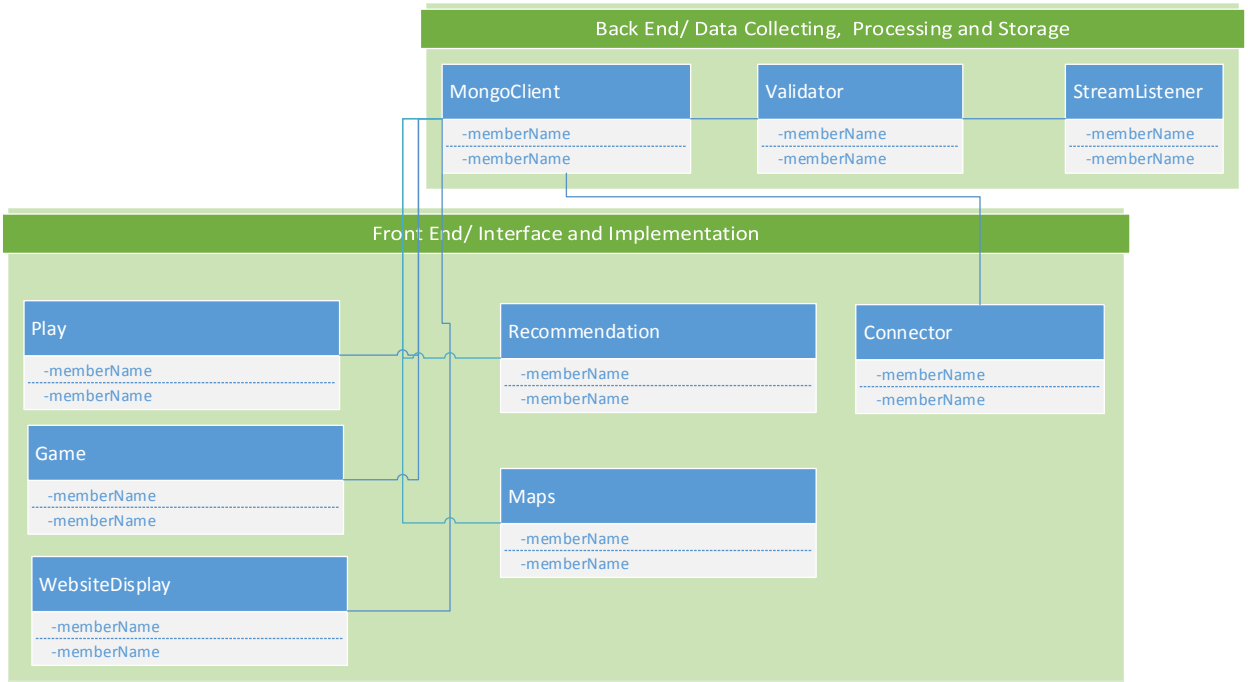

Figure 28 Subsystem

We can logically divide our entire system into 2 subsystems. The first subsystem, the back end, deals entirely with fetching twitter data, filtering/processing it, and then storing it. The second subsystem, the front end, then retrieves data that is output from the

database, which it then uses as input for all of its features that it implements and presents to the user through the interface.

Let us try to understand why we can, in fact split our system into these 2 subsystems. Well, each subsystem does a particular function (albeit very complex). This means that we could in theory replace a subsystem with another different “black box” that accepts the same input and provides the same output, and this would not affect the other half of our system. Let us imagine that in the near future, twitter upgrades its website to include a fully functional, on demand database, and tweet data filterer. We can get all the data we want, tailor fitted, directly from twitter. This feature makes our back end subsystem obsolete. Because the new twitter database feature can provide us with the same output as our obsolete subsystem, we can use its output as the input for our front end subsystem, completely get rid of our back end subsystem, and our system will still remain fully functional.

Conversely, imagine that some data analytics fitness company creates some “super front-end software” that can do everything that our front end subsystem could do and more. It makes our front end subsystem obsolete. The “super front-end software”, however, requires processed and filtered twitter data. We could keep our back end subsystem, delete the front end subsystem, and connect the “super front-end software” to the data output from our back end subsystem, and our system will still remain fully functional.

9.3 Mapping Subsystems to Hardware

For the back end (data fetching, processing, and storing). The database is MongoDB and is located on the same computer that is also running the server. The instructions for the front end subsystem are stored on the server computer as well, and these instructions, as well as the output from the back end subsystem (the processed, filtered, and stored data) are transferred to the client’s computer via HTTP over an internet connection. The actual implementation of the front end subsystem occurs on the client’s computer when the front end instructions and requested subset of data are processed.

9.4 Persistent Data Storage

Our system uses MongoDb to store the data coming from Twitter API. This data comes in JSON document format, which is highly compatible with MongoDB file format BSON (BSON is a Binary JSON). The data is stored in the same format, and includes all the fields that are offered by the Twitter API. These fields are specified by Twitter developers in https://dev.twitter.com/overview/api

The most important fields to us are:

Coordinates, entities, text, retweeted.

Coordinates: Nullable. Represents the geographic location of this Tweet as reported by the user or client application. The inner coordinates array is formatted as geoJSON (longitude first, then latitude).

Example:

```
"coordinates":
{
    "coordinates":
    [
        -75.14310264,
        40.05701649
    ],
    "type":"Point"
}
```

Entities: which have been parsed out of the text of the Tweet.
Example:

```
"entities":
{
    "hashtags":[],
    "urls":[],
    "user_mentions":[]
}
```

Text: The actual UTF-8 text of the status update. See twitter-text for details on what is currently considered valid characters.
Example:

```
"text":"Tweet Button, Follow Button, and Web Intents javascript now
support SSL http:\/\/t.co\/9fbA0oYy ^TS"
```

Retweeted :Perspectival. Indicates whether this Tweet has been retweeted by the authenticating user.
Example:

```
"retweeted":false
```

9.5 Network Protocol

Our software is a client-Server application. Therefore, the software parts use the internet communication network protocols TCP/IP and HTTP between webserver services (Apache with MongoDB) and the client (Browser).

9.6Global Control Flow

The system is event driven and time dependent at the same time. Part of the system, specifically the User Interface and the Connector, wait for a possible user to either load a new page of the system or make a request like, for example display a visualization map. This is a specific event that may happen at some point in time and the system must be triggered and respond to the user requests. Also the system is time dependent because it queries for new tweets and processes them in specific time intervals that is greater than the time required by the critical path of this system. The critical path is the path in the system that is defined as the point of getting the tweets until the moment that the slowest feature has finished processing and has stored the results in the database.

9.7 Hardware Requirements

In order for our system to function properly on the server side, the following hardware is needed. A server is needed to host the website, as well as to allow us to store tweets. MongoDB, Python, and Java IDE must be supported by the server. Also, it should have enough disk space to store data and to help in the future growth of the system. In addition, a sizeable amount of memory (RAM) is required to facilitate data processing. Since it is a server, it is obvious that it is capable of performing network communication.

When it comes to the client, he/she needs a computer that can browse websites and have a high-speed Internet connection.

10. Algorithms and Data Structures

10.1 Algorithms

10.1.1 Data Analysis Algorithms

Our Data analysis is basically a two phase process. Phase one isgetting the tweets data from the Twitter API by using twitter search with a combination of specific hashtags and terms related to the heath activities, as well as the hashtags belonging to wearable sports devices.

The second phase (under development) is analysing the data from the first phase to see how every hashtag (or term) or a group of hashtags (or terms) can be optimized via noise filtering to get a final product of, usable, trustable data. This phase can be broken to two distinct steps. The first part is called data_filter2 and the second data_filter3. The algorithms are:

data_filter2:

1. Take all tweets and the keywords
2. For each tweet and for each keyword
3. If the keyword is in the tweet keep it else discard it
4. Continue by going to 2 until all tweets have been checked

data_filter3:

1. Take all tweets and the keywords
2. For each tweet and for each keyword
3. If the keyword is in the tweet discard it else keep it
4. Continue by going to 2 until all tweets have been checked

10.1.2 Per Capita Map

When we are collecting tweet data, for each keyword/phrases, and for a fixed time interval, we collect the count of the number of tweets from a city and divide it by that city's population. This will give us one ratio for city x, and another ratio for city y. Notice that the ratio itself is meaningless. For example, if 7 tweets per 1,000 people contain "I went jogging", this by no means implies that 0.7% of people jog. However, if in city x every 20 tweets per 1,000 people contains "I went jogging" and in city y every 45 tweets per 1,000 people contains "I went jogging", then we can make a very strong case that not only in city y do people jog more, but using our assumptions above, we can estimate that the population goes jogging about 2 times as much in city y than x.

There are many available sources provide aggregate nationwide data for health and fitness. For instance, we can probably get accurate figures for the percentage of Americans that bike regularly. Let's say according to the government figures, (A reliable source) 5% of Americans bike regularly. Now let's go ahead and search the entire country for tweets from one a week that contain "I rode my bike". Let's say that we collect 500,000 matching tweets (this is just a fictional example and is much higher and completely different from reality) throughout the country.

Now set the average ratio of tweets to population at 500,000/300mi= 5/3,000 with all of our earlier assumptions we can now say that 5/3,000 is the national corresponding ratio to 5% of Americans bike regularly. Imagine if in Tulsa, Oklahoma, we see that for their population, every week, the phrase "I rode my bike" shows up about 3 times for every 3,000 people. Then we could make a strong case that (3/3,000)/ (5/3,000)*5% = (3/5)*5%=3% of Tulsans bike regularly. Also, if NYC average 7 tweets with that phrase per every 3,000 people, then we can make a strong case that (7/3,000)/ (5/3,000)*5%=7% of New Yorkers bike regularly. Then we can translate this to actual figures by multiplying 7%*8mil=560,000 New Yorkers bike regularly.

What we have done is take aggregate data from a trusted source, take per capita tweet frequencies of arbitrary phrases, and use regional differences of the tweet frequencies per capita to find regional, actual data.

10.1.3 Interactive Map

The algorithm used for the Interactive Map is based on analysing the tweets and finding which activity has the largest frequency (with respect to the coordinates).

So after finding the highest frequency activity, there will be a search for the next highest frequency activity up to 5 activities (just suggested number has no specific meaning).Highest frequency in our case mean that this hashtag/keyword has been repeated in tweets more than others for a specific area.

10.1.4 Recommendation

The Recommendation would based on the Interactive map, showing the highest frequency activities and shows it to the customer based on location and involving the weather API.

10.1.5 Game System

The Game feature of this system, tries to assign points to areas in as just a manner as

possible. The feature is more interested, in general, in the impact the number of tweets in each update of the game has to the whole number of tweets from that area, rather than simply adding numbers and sorting areas by sheer amount of tweets. Also, it is considered important to penalize areas when there is no tweets for a significant amount of time. In general, the pointing assignment system depends on the number of the new tweets compared with the last update, the time since the last update, and the impact of this update to the whole number of tweets from that area.

The feature can be broken down into two independent algorithms. The first algorithm, Game class, is the one used to assign points to a specific area based on specific inputs, as will be described. The second is the communication algorithm that realizes the connection between the pointing system and the data storage of the system, realized by the Play class. The data storage here is the MongoDB database. The reason for this separation is to make the pointing system as independent as possible from the type of the data storage and vice versa, meaning the data storage as independent of the pointing system.

First the Communication Algorithm is presented , because it helps show the data movement between the database and the pointing system, and the other way around.

Communication Algorithm:

1. Fetch the new tweets from the database
2. Count the number of tweets for each Area that appears in this set
3. Select an Area and get its statistics and points from the database
4. Update the Points of this Area by a pointing system
5. Update the Area's statistics and points in the database
6. Repeat steps 3 to 5 for the next Area

As it can be seen from the Communication Algorithm, during step 3 the actual update happens and the communication is independent from it. The next algorithm is the Pointing System.

Pointing Algorithm:

1. Find how much time has passed from the last update.
2. If more than a week has passed, and no new data has come from that area, and the area had previously been at zero points ***-10 new points and go to 12***
3. Calculate the difference between the number of tweets for this assignment and the number used in the previous update. This is the initial number of new points

4. If this is the first time an area get points, ***add 20 more points. Go to 12***
5. If the difference is positive, there is at least a 50% increase, and the new number of tweets is more than 10% of the total, ***add 10 more points. Go to 12***
6. If the difference is positive, there is at least 50% increase, and the new number of tweets is less or equal than 10% of the total, ***add 5 more points. Go to 12***
7. If the difference is positive, there is less than 50% increase, and the new number of tweets is more than 10% of the total, ***add 5 more points. Go to 12***
8. If the difference is positive, there is less than 50% increase, and the new number of tweets is less or equal to 10% of the total, ***add 0 more points. Go to 12***
9. If the difference is negative, the number of new tweets is at least 50% of the previous, and the new number of tweets is more than 10% of the total, ***increase the points to half. Go to 12***
10. If the difference is negative, the number of new tweets is less than 50% of the previous, and the new number of tweets is more than 10% of the total, ***increase the points to 75% of the initial. Go to 12***
11. For any other case ***do not change the number of points***.
12. If there is an update due to the time passed, ***update the points and go to 13***
13. Update the points, consider the new number of tweets as the one used for the last update, update the total number of tweets from the number when the last update happened.
14. Return the statistics and the points for the Area

10.1.6 CalorieMeter

CalorieMeter crawls the augmented amount of burned calories that are generated automatically by mobile apps and wearable devices. This algorithm allows us to ensure the accuracy of the collected tweets as well as decrease the noise in these data. We use carefully selected hashtags and keywords that guarantee this accurate collection of tweets. Also, the system will show top workouts done by people to burn these calories along with the most recent five tweets.

10.1.7 Additional Algorithm discussion

The main purpose of our software, is to show location based user activity. With that being said very few tweets are actually geotagged (about 1%). For almost every feature we have, we needed geo coordinates associated with a tweet, to allow us to even use it. Therefore, in order to increase the amount of usable data we had, we attempted to find ways to associate geo coordinates to tweets that were not already geo-tagged.

Original Geo-Mapping User Location String Algorithm

We were not able to successfully implement our initial location finding algorithm. As you recall, this algorithm consisted of collecting lists of all geo tagged coordinates for each unique user location string, and using a trimmed mean, and setting a cutoff standard deviation, to determine if the geo tags did in fact cluster around a specific location.

We have collected over 500,000 tweets. Of these 500,000 tweets, about 5,500 were geo-tagged. (A good rule of thumb is that about only 1% of tweets are actually geo – tagged for location. Assuming a random distribution of these tweets, 5,000 seemed like a very small, but potentially enough to determine geo coordinates for maybe 100 or so locations. However, what we found was the popular "80-20 rule" was also prevalent in the tweeting process. For example, of those 5,500 tweets that were geo tagged, we found that over 15 had a user location listed as "At my friend Bryan's House", many had "I live on the corner of cherry street". It seemed very unlikely that these locations were listed by different users, and, in fact, these very unique names were all from tweets that were tweeted by the same person respectively.

We then realized that any coordinate data that we got from one user, could only be considered one time, for statistical purposes. Otherwise, we could have someone tweeting 40 times and having their user location listed as "Hollywood, CA", but they could actually be tweeting in Argentina. We may have 15 other geo-tagged tweets, each by a different person, with their location also listed as "Hollywood, CA", who actually do live in Hollywood. If we incorrectly tried to use all 40 of the geo-tagged tweets from the Argentinian user to determine the actual geo coordinates associated with "Hollywood, CA", we would incorrectly associate a South American Latitude and Longitude for the user location of "Hollywood, CA". However, if we correctly only allowed one tweet per user to be factored into this, then our algorithm would correctly assign coordinates within Hollywood, CA, to the user location "Hollywood, CA".

But this requirement for uniqueness brought the amount of usable geo-tagged tweets down to under 1,000. Which is less than 0.2% of all tweets. Using less than 1,000 tweets was not worth implementing our algorithm, because, we found that only a few of the biggest cities, allowed the algorithm to be successful.

In the future, I would try to implement this algorithm with at least 100,000 geo-tagged tweets. This would, on average, require collecting (1/0.002)*100,000 = 50 million tweets.

Standard US Address User Location String Geo-Mapping Algorithm

Because of the failure to retrieve enough data for our original algorithm above, we then came up with another idea to try and increase the number of geo-tagged tweets we could

use. What we did was take a list of the top 3,700 U.S. cities, along with their geo-coordinates (latitude and longitude). We then took the 2 acceptable formats for proper location listing, the full name of the city followed immediately by a comma and a space and followed immediately by either the full state name or the 2 character state abbreviation. The 2 acceptable formats are “City, State” and “City, ST”. For example, the following are acceptable: “Newark, New Jersey” , “Philadelphia, PA” , “Newark, NJ” , “Minneapolis, Minnesota”

Our algorithm works as follows:

Every time a tweet is received from twitter, if it is already geo-tagged, we save those geo tags to a 2 new variables assigned to that tweet, finalGeoLat and finalGeoLog, respectively. If the tweet is not already geo-tagged, we then check the user location string again the list of 3,700*2 = 7,400 acceptable location names that we mentioned above. If the user location matches any of these, we then use the geo coordinates associated with that matched location, and assign it to 2 new variables, finalGeoLat and finalGeoLog, that are assigned to that particular tweet. Finally, if the tweet does not have geo tags, and does not match any location, we discard it. Interestingly enough. This simple algorithm worked very well, and allowed our usable number of tweets to rise almost 10 times the amount we had before applying the algorithm. In more detail the algorithm is a follows:

1. For all the tweets check if the tweet is geotagged if yes append it to the located tweets, else go to 2
2. For all the acceptable locations check if the users location match, repeat.
3. If the location does not use the abbreviation for tha state change it.
4. Append the tweets to the located tweets and return to 2.

Potential Insights into Using this Method

The first thought regarding this accuracy of using the person’s city and state to conclude that they are in fact from that location, involves the personality of that user. There are many user’s that list obviously fictional and/or nonsensical locations in their profile. We hypothesize that these user’s are also more likely to tweet out fictional, humorous, and sarcastic tweets. Conversely, I propose that a user who follows the standard address format of “City, State”, is less likely to tweet out fictional, humorous, or sarcastic tweets.

What does this mean? Collecting tweets that are not factual is bad, and serves to create noise and introduce less accuracy to our analysis. Therefore, if we can maximize the amount of factual tweets, we can get less noisy data. If, in fact, users whose location is in the standard form of “City, State” do tweet more factually, then by using this algorithm, we have in fact also increased the accuracy of our tweets.

Another issue at hand is whether the person tweeting is actually in the city that is listed as their location. We can be sure of their location if the location is geo-tagged using GPS, but we cannot be sure of their exact location when we use our "City, State" mapping algorithm. To address this, imagine that we take a city that is a major destination area, such as Orlando, FL. Let's imagine that someone from Chicago is in Orlando and decides to workout. If they happen to tweet about the exercise and to be geotagged by GPS, we will allocate their workout to Orlando. But is this the right allocation? I would make the case that when trying to see what cities workout the most, we should be more interested on only the citizens that actually live in that town. If many people from out of town work out in Orlando, should Orlando be rewarded, or should the towns where those people are from be rewarded? I believe that it makes more sense to actually allocate the home town of the people, and not the actual location in which the exercise took place. Therefore, it may actually be even BETTER, to use the user's listed location as opposed to the geo-tagged location from GPS, when presenting the exercise data.

Per Capita Heat Map Weighting Algorithm

The weighting algorithm is applied after all data collection algorithms are complete.

For the weighting algorithm, we want to estimate the population density for any given point of latitude and longitude in the country. We then want to use the inverse of the estimated population density as the weight assigned to that tweet. We first need to create a continuous population function for any given geo coordinates. We take our table with the top 3,700 hundred U.S. cities and their populations and geo coordinates. We also need the haversine formula function, for finding the distance in miles between 2 sets of geo coordinates.

The weighting function works as follows:

For each tweet, we get the finalGeoLat, and finalGeoLog, which are the geocoordinates. We then first check these coordinates against our list of 3,700 US cities. If the coordinates match one of the cities on the list, we simply find the population of that city, and set the weight of that tweet to be the reciprocal of that city's population.

If the geo coordinates do NOT match a city on the list, then we get a list of all the cities that are within the radius of influence, which we set to be 120 miles. We use the haversine function to calculate the distance of all cities, and any that are within 120 miles, we add to a temporary list.

Note that we optimize the algorithm a bit, by eliminating cities that are outside of the 120 mile range by virtue of their latitude or longitude alone, thus saving computation time of having to compute the haversine function 3,700 times. We only have to compute the haversine function for cities falling within the possible lat,long rectangle that is plus or minus 120 miles north, south, east, or west.

Then, we do the following:

If there are no cities in the list, meaning that the coordinates are not within 120 miles of any of the 3,700 US cities, we set the weighting to zero. This is to ensure that tweets outside of the country don't show up on the heatmap.

If there are cities in the list we do the following:

We find the city in the list that is the closest to the geo coordinates we have from the tweet. We save this distance as currentMin.

Then we do the following:

We loop through every city in the list. We set the highest weight possible (1) to the closest city to our geo tags. Every other city is given a weight that is the selected as the distance of the closest city divided by the distance of city x from our geoTags. This is then raised to a power of 2.4, in order to exaggerate the impact of the closes cities. Also, to prevent areas nearby large cities from having densely populated outskirts that are not properly accounted for, we introduce the bigCityMultiplier, which increases the weight based on the size of the city, if the city is over 300,000 people. This causes tweets near big cities to have their population lean more towards that of the big city. We slightly dampen this effect by raising the bigCitMultiplier ratio to the power of 0.8. The 2.4,0.8, and 300000 are 3 variables that should be calibrated in the future. Each city in the list is then assigned a weight and we take the weighted average of all the city's respective populations, to get our estimate of the population density of our tweet's geo coordinates. Then we assign to the tweet a weight that is the reciprocal of this estimated population density.

In the future, we could change up this algorithm to calculate in 2 dimensions. Right now, it is only calculating weights based off of one dimension, geographically speaking. For instance if a very small city is 20 miles due East of the geo tags, and a very big city is 80 miles due East in the same direction. A very good argument could be made against even considering that larger city in the population estimation, because its effect should radiate outward in a straight line, and should be stronger the closer the city is to it.

10.2 Data Structure

The tweets comes from Twitter REST APIs in JavaScript Object Notation (JSON) documents format. JSON is an open, human and machine-readable standard that facilitates data interchange, and along with XML is the main format for data interchange used on the modern web.JSON is a text format that is completely language independent but uses

conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. JSON supports all the basic data types you'd expect: numbers, strings, and boolean values, as well as arrays and hashes. Document databases such as MongoDB use JSON documents in order to store records, just as tables and rows store records in a relational database. A JSON database returns query results that can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages – reducing the amount of logic you need to build into your application layer.

MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes. BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages.

The MongoDB BSON implementation is lightweight, fast and highly traversable. Like JSON, MongoDB's BSON implementation supports embedding objects and arrays within other objects and arrays – MongoDB can even "reach inside" BSON objects to build indexes and match objects against query expressions on both top-level and nested BSON keys. This means that MongoDB gives users the ease of use and flexibility of JSON documents together with the speed and richness of a lightweight binary format.

The Game feature uses Python arrays and dictionaries as its data structures. Arrays are used in order to create a signal iterating structure to store all the areas that need to be updated each time. It helps to have the ability to reference each area to be updated, in the same manner, and not have to create a separate type for each one of them. Dictionaries, in Python, can be thought as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary).

11. User Interface Design and Implementation

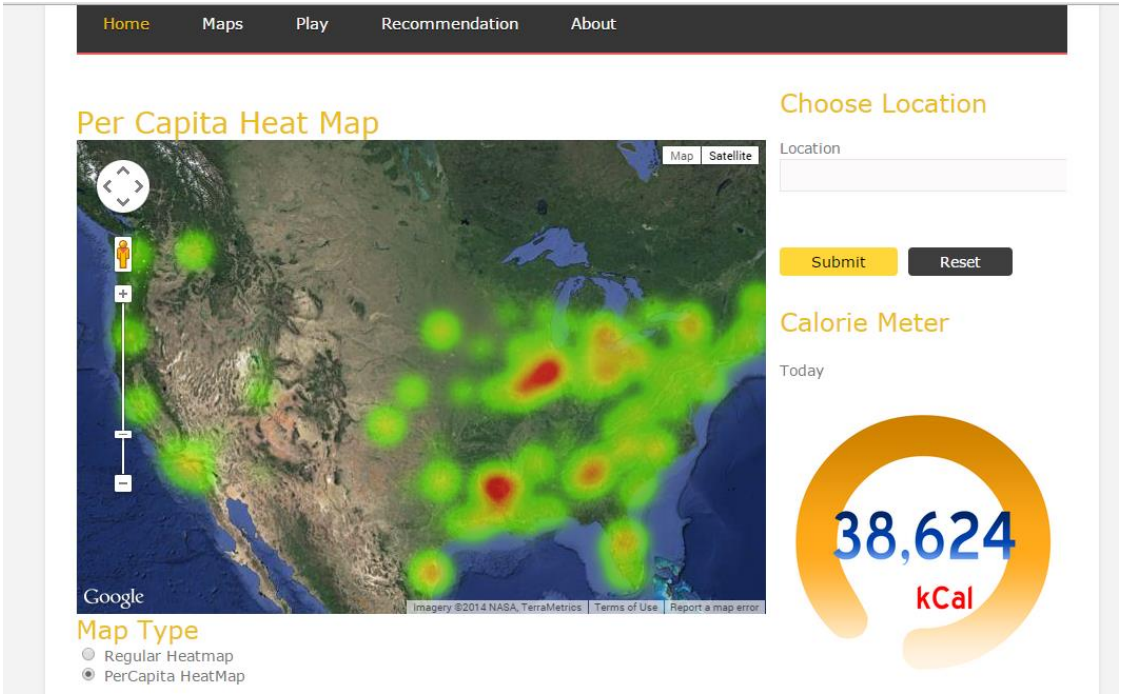

Figure 223 Home Page of the Website

Figure 29 is the home page of the website. It include the Per Capita Map. In the Per Capita Map a user will be able to select the data set that he/she would like to view, as well as the representation type of the data on the map. The Data Set will be a drop down list with only one possible selection. The Data Representation will be a list of radio buttons. The map will be fixed to the US. Upon any change in either list by the user, the map will refresh itself appropriately. Because the Estimated Actual Pop Data feature will only be available for some data sets, if it is selected when it cannot be used, an alert message will appear to prompt the user to select another data set or change the data representation type.

Figure 30 is the Map page. The Interactive Activity Map page would be very similar to the Per Capita Map page, The map would automatically display the 200 relative tweets around the center of the map.

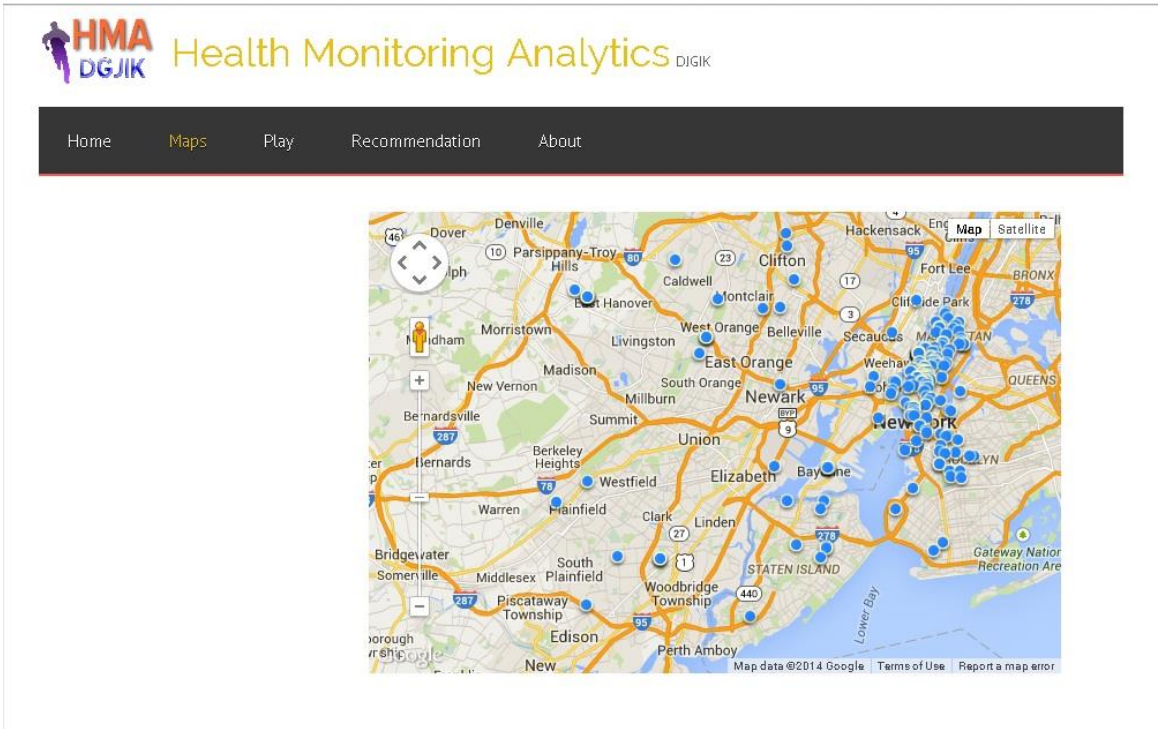

Figure 30 Map Page of the Website

Figure 31 shows the Play page. In this Gaming page on the left hand side shows the State Leaderboard which will always be displayed there. The leaderboard will be a list from 1 to 10, showing the 10 states with the most points, along with their points. On the right hand side there will be a drop down list where all states will be presented. When the user selects a state, the state's points will be displayed along with a list of the top 10 cities of that state similar to the State Leaderboard.

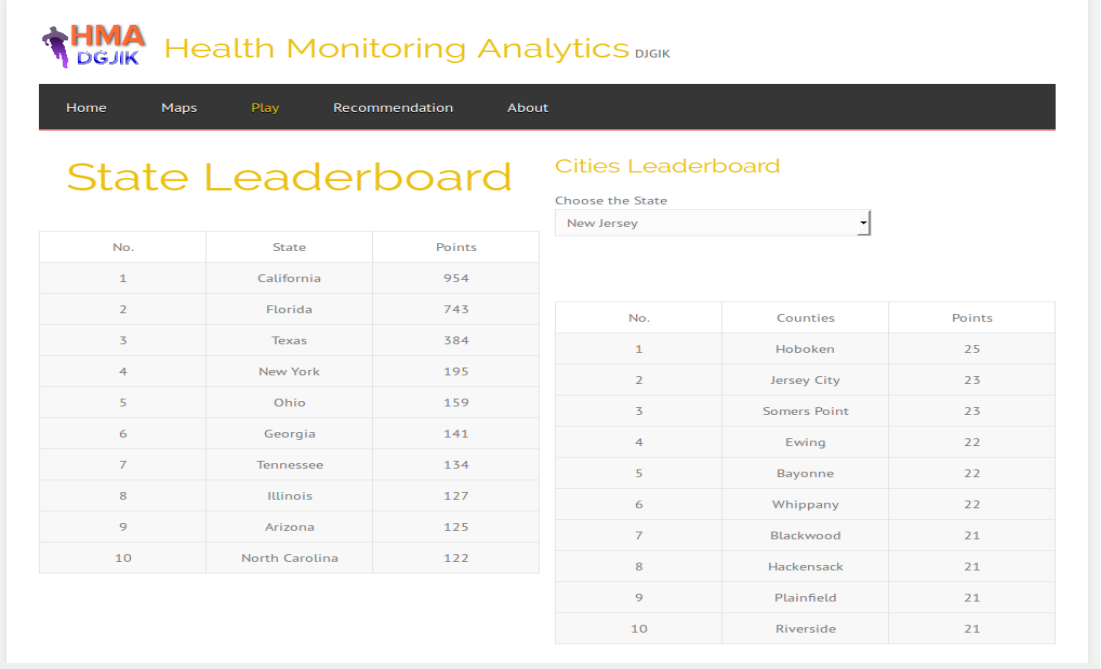

Figure 24 Play Page of the Website

In the Recommendation Page, shown by Figure 32, the user can type the city or place he or she lives. The right chart would automatically display the weather situation of the place. The left chart would display the world's most popular activities based on the data of interactive map. The user can compare the popular activities and the weather to make a sport decision.

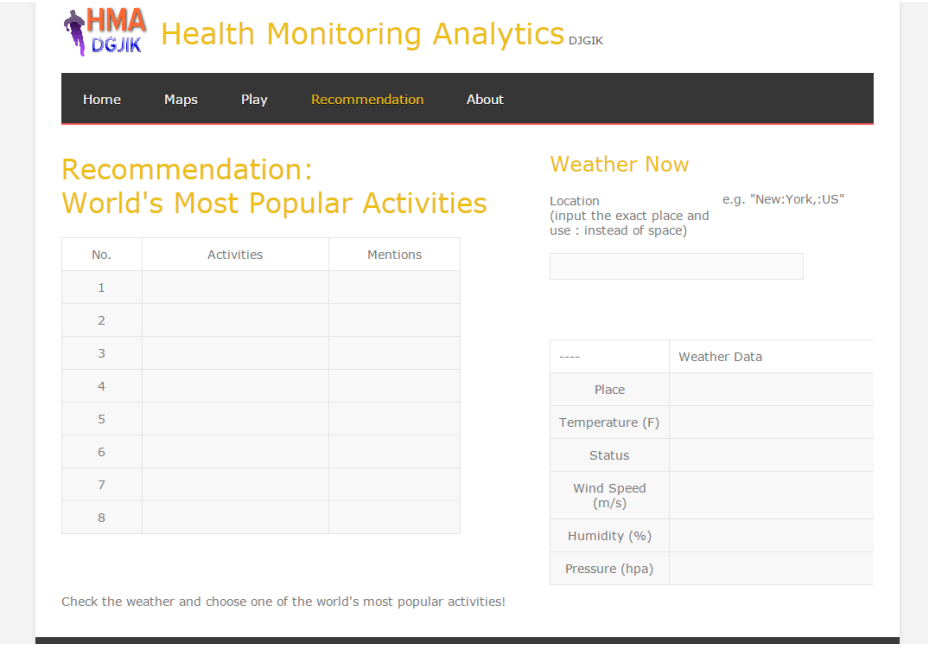

Figure 25 Recommendation Page of the Website

12. Design of Tests

12.1 The Unit Test for the PerCapita HeatMap

For the PerCapita Heat Map, we are testing the concept of using population based weighting for data that is generated by the local population (in this case tweets). In particular, we are testing whether using this type of weighting will produce the expected results for different random sets of data.

For this test we need only 2 files, both in the same folder:

HMTL file : AttemptforGoogleMapHTMLONLYwithGENERATE.html

Javascript file : PerCapitaJavaScriptCombinedWithGenerating.js

For this unit test we take 7 different US cities with different populations:

1. NYC
2. Los Angeles
3. Chicago
4. Houston
5. Jacksonville
6. Seattle
7. Denver

We then generate 400 random points of location data, which can fall in one of the 7 cities. For the probability set from which these random points are drawn, we use 3 different sets

1. Equal Probability, (1/7) for each city, regardless of population
2. Population-Based Probability, every person out of the total population of all 7 cities has an equal chance of generating the data (in our project's case generating the tweets).
3. A mix of both, where for each city the probability is the Equal Probability (1/7) plus the population-based probability, all divided by 2.

Then we allow 2 different views of this randomly generated data, one is a heatmap that equally weights all the points on the map. The other is a population weighted heat map, which essentially weights each tweet by dividing it by the corresponding population from the city from which the tweet originated.

When we run the html file, a map is generated with 6 different radio buttons. Each of these buttons correspond to all 6 possibilities of the 3 different probabilities for

generating data sets of 400 points and the 2 different map types, weighted vs regular, for displaying the results. Which are as follows:

1. Regular Heat Map with Data generated with equal probability (1/7) for each city regardless of the population. If this visualization works, then we should expect the map density to be equally spread out around the map. We can click the first button several times to verify that, in fact, this is the result that we expected.

2. Regular Heat Map with Data generated with population based probability. If this visualization works, then we should expect the map density to be more concentrated in cities with the highest populations. We can click the second button several times to verify that, in fact, this is the result that we expected.

3. Regular Heat Map with Data generated with a mix of both equally weighted and population weighted probability. If this visualization works, then we should expect the map density to still be more concentrated in cities with the higher populations, but not as much so as in number 2 above. We can click the third button several times to verify that, in fact, this is the result that we expected.

4. PerCapita (population weighted) Heat Map with Data generated with equal probability (1/7) for each city, regardless of the population. If this visualization works, then, we should expect the map density to be more concentrated in cities with the lowest populations. We can click the fourth button several times to verify that, in fact, this is the result that we expected.

5. PerCapita (population weighted) Heat Map with data generated with population based probability. If this visualization works, then we should expect the map density to be more concentrated in cities with the highest populations. We can click the fifth button several times to verify that, in fact, this is the result that we expected.

6. PerCapita (population weighted) Heat Map with data generated with a mix of both equally weighted and population-based probability. If this visualization works, then we should expect the map density to still be more concentrated in cities with the lower populations, but not as much so as in number 5 above. We can click the sixth button several times to verify that, in fact, this is the result that we expected. In conclusion, we can conclude the per capita weighted data to allow us to compare characteristics of the populations of different areas, regardless of the size differences between them.

12.2 The Unit Test for the Game System

12.2.1 Game class Unit Test

This unit test creates a mock area with all the necessary attributes set to specific values

and tests how the points of an area are updated. The cases it covers are the increase or decrease according to different number of new tweets for the update, the decrease of points due to no new updates for at least a week, and the get method of the class. The expected results are pre-calculated and the answer of each method is compared to that. In a case of failure the class should be changed in that part in order to implement correctly the case under test. One case that actually failed at the first run of the unit test was the decrease of points after a long time. The system initially changed the date that the area was updated to the date of the decrease. This issue was fixed so that the date was not changed. It is crucial to keep the actual date of an update due to new tweets and not because of any change of points, because the main point is to encourage people to exercise and tweet about it.

12.2.2 Play class Unit Test

This test needs to use a stub for the Game class and a mockup query from the database. It checks the correct connection between those two parts. In a case of a failure the interface, between either the Game class or the database, should be checked if it is according to the specification and changed accordingly.

12.2.3 UC – 6 Test

This is actually the integration test of the feature. Mock data will be used for the Leaderboard of the states and New Jersey, since that is the default state for the County Leaderboard. Also the database will be fed with a specific number of tweets from different areas that will cause well known results as the output. A success of this test will be considered to be such: that the pre-decided Leaderboards are displayed correctly in the Play page. A failure means such: that wrong data are displayed and the possible causes are that the page does not correctly communicate with the database. Also in the next refresh of the page, the new Leaderboard will be shown and the results are compared to the expected ones. If this part fails, the unit tests of the Play and Game classes and the integration test of the processing part of the system should be revised, since their success assures the success of the second part of the UC - 6 test. The detail of UC—6 is in the Appendix

12.3 The Unit Test for the Recommendation

The Unit Test for the Recommendation would be a python program that trying to get information from the Openweathermap API. The test would only use one certain location “Piscataway” and trying to use the openweathermap to get all the weather information we need, including temperature, status, wind, humidity and pressure.

12.4 The Unit Test for the CalorieMeter

Beside the Python code we created to test CalorieMeter as presented in Demo1, Tweetchup twitter analysis tool (3rd party tool) was used to test the accuracy of the

selected hashtag and keywords. The results were identical to our code and motivate us to continue in this direction.

13. History of Work, Current Status and Future Work

Per Capita Heatmap:

In regards to the future of the Per Capita heatmap system, much discussion of this is currently in the algorithms section above. In addition to improving the population density function/algorithm, I would like to improve upon the aesthetics of the map itself. Most importantly, I would have liked the map to be asynchronously updated. Also, I believe that the ability to observe the difference between different time periods would be very nice. Perhaps as a default, we could show a time lapse of changes that occurred over say, the last week.

Also, we could have deduced the caloric value of different activities based on the average of the scores of each activity from the calorie meter. We then could have also used the average calories of the particular exercise that the tweet mentions, and actually multiplied that number by the weight that we have already obtained from our original population weighting algorithm, to get “PerCapita Calories Burned” Map. This would be a great measure that we could introduce in the future.

Game System:

Regarding the Game feature we were able to create a fully functional first version. The algorithm created for assigning points is at an early stage. It can easily be optimized to become fairer and eventually use the population of each area as an extra parameter to assigning points. All the parts of the feature have been debugged so that everything works as it should. More specifically, until the first demo bits and pieces of the feature existed. The processing elements existed and work in a degree. Also the game page of the site was working with just mock up data to show mainly how it was visualized. Until the second demo the feature was fully integrated. The page was querying the database correctly with AJAX calls. Some bugs from the Game and Play classes were corrected mainly because real data were used and some corner cases, such as division with zero when an area, State or City, was pointed for the first time, become visible. During the second demo, a bug at how the database updated the entries for the cities existed, resulting in giving points only to one city per State. This bug was corrected after the second demo presentation and now the game feature is fully functional.

The missing, uncoded, parts of this feature are its tests, mainly the integration test for the feature and that should be the future work. These tests can show bugs that may not be corrected yet, because the system has not run for a significant amount of time. Also the tests may show how the pointing system behaves after multiple updates with different

amount of tweets per area.

Recommendation

The recommendation part is based on the interactive map. The sad thing is, as we cannot find a lot of data that can fulfilling the need of interactive map. The recommendation is pointless. For example, if you only find 200 tweets in a town and the most of them including “running” “run” and “bike”, it is impossible for you to recommend the most famous 8 sports here. So far, we use the data all over the world and that can get us no more than 10 sports. If in the future we can have large enough data, the recommendation part may be more pratical. We may can recommend multiple sports based on the location and the weather.

CalorieMeter

Regarding CalorieMeter, we were able to accomplish design a system that calculates the augmented number of burnt calories taken generated by mobile apps and wearable devices for the most recent 200 tweets as restricted by Twitter API. We would like to see many improvements and code factoring to this innovative idea and we suggest the following:

1- Listing in order the workouts that contributes the most to burning calories.
2- Giving the augmented amount of burnt calories for cities, counties, and states rather than for thw whole world.
3- We are waiting to the smart students to come to add many more innovative ideas.

Another good direction we suggest instead of using Twitter is to use HealthGraph (Runkeeper API). Here are some useful links related to it:

1- The official site: http://developer.runkeeper.com/healthgraph
2- Python Client Library for Health Graph API http://aouyar.github.io/healthgraph-api/
3- Objective C wrapper class for accessing the RunKeeper Health Graph API from iOS 4.0 or newer. https://github.com/brierwood/RunKeeper-iOS

Product Ownership

Ownership	D. Takacs	G. Bati	J. Abdulbaqi	I. Paraskevakos	K. Dong
Per Capita Map	√				
Game Experience				√	
Interactive Active Map			√		
Recommendation					√
Calorie Meter		√			

Report Distribution

Report	D. Takacs	G. Bati	J. Abdulbaqi	I. Paraskevakos	K. Dong
Problem Statement	√	√	√	√	√
Glossary of Terms		√			
Functional Requirements	√	√	√	√	√
Non-Functional Requirements		√	√	√	
On-Screen Appearance Requirements	√	√	√	√	√
Stakeholders					√
Actors and Goals					√
Casual Description	√	√	√	√	√
Use Case Diagram	√	√	√	√	√
Traceability Matrix					√
Fully-Dressed	√	√	√	√	√

Description					
System Sequence Diagrams			√	√	√
User Interface Specification	√	√	√	√	√
Domain Model	√	√	√	√	√
System Operation Contracts	√	√	√	√	√
Mathematical Model	√				
Interaction Diagrams	√	√	√	√	√
Class Diagram	√	√	√	√	√
Data Types and Operation Signatures				√	
Traceability Matrix					√
Architectural Styles					
Identifying Subsystems	√				
Mapping Subsystems to Hardware	√				
Persistent Data Storage		√			
Network Protocol			√		
Global Control Flow		√			
Hardware Requirements		√			
Algorithms	√	√	√	√	
Data Structures			√	√	
User Interface Design and Implementation					√
Design of Tests	√	√		√	

Project Management and Plan of Work				√	√
Reference		√			

Tasks and Gantt Charts

Tasks	**Starting Date**	**Duration**	**Ending Date**
Data Mining	9-Sep	46	25-Oct
Web Interface Design Phase 1	25-Oct	5	30-Oct
Customer Statement of Requirements	1-Oct	4	5-Oct
Glossary of Terms	15-Sep	5	20-Sep
References	14-Oct	1	15-Oct
System Specification	17-Oct	8	25-Oct
Domain Analysis	10-Oct	5	15-Oct
Web Interface Design Phase 2	30-Oct	31	30-Nov
Online Archiving	9-Sep	94	12-Dec
Twitter Query Optimization/ Data Analysis	20-Oct	31	20-Nov
Map Interface	15-Oct	14	29-Oct
First Gaming System	14-Oct	15	29-Oct
Interaction Diagrams	15-Oct	7	22-Oct
Class Diagrams and System Architecture	15-Oct	15	30-Oct
Report #1	20-Sep	25	15-Oct
Report #2	15-Oct	22	6-Nov
Visualizing the tweets on the website	11-Nov	20	1-Dec
Integrating CalorieMeter	11-Nov	23	4-Dec

Future enhancements for CalorieMeter	11-Nov	31	12-Dec
Using the weather API	11-Nov	20	1-Dec
Combine Recommendation with the Interactive maps	11-Nov	31	12-Dec
Future enhancements for Recommendation	11-Nov	31	12-Dec
Finalizing our keyword/phrases algorithms	12-Nov	14	26-Nov
Finding optimal query time rotation between keywords/phrases for visualizing different data for all systems	26-Nov	7	3-Dec
Finalizing population function for heatmap weighting and integrating it into the maps	26-Nov	5	1-Dec
Finding optimal phrase clusters for physical attributes	19-Nov	12	1-Dec
Finalizing visualizing the data using Google map with markups	12-Nov	19	1-Dec
Finalizing the implementation of querying the top activities from our database	12-Nov	19	1-Dec
Finalizing the new map with new selection control.	12-Nov	19	1-Dec
Integrate and Test Game	7-Nov	24	1-Dec
System Integration and Testing	14-Nov	28	12-Dec
Report #3	18-Nov	22	10-Dec
Demo 2	7-Nov	28	5-Dec

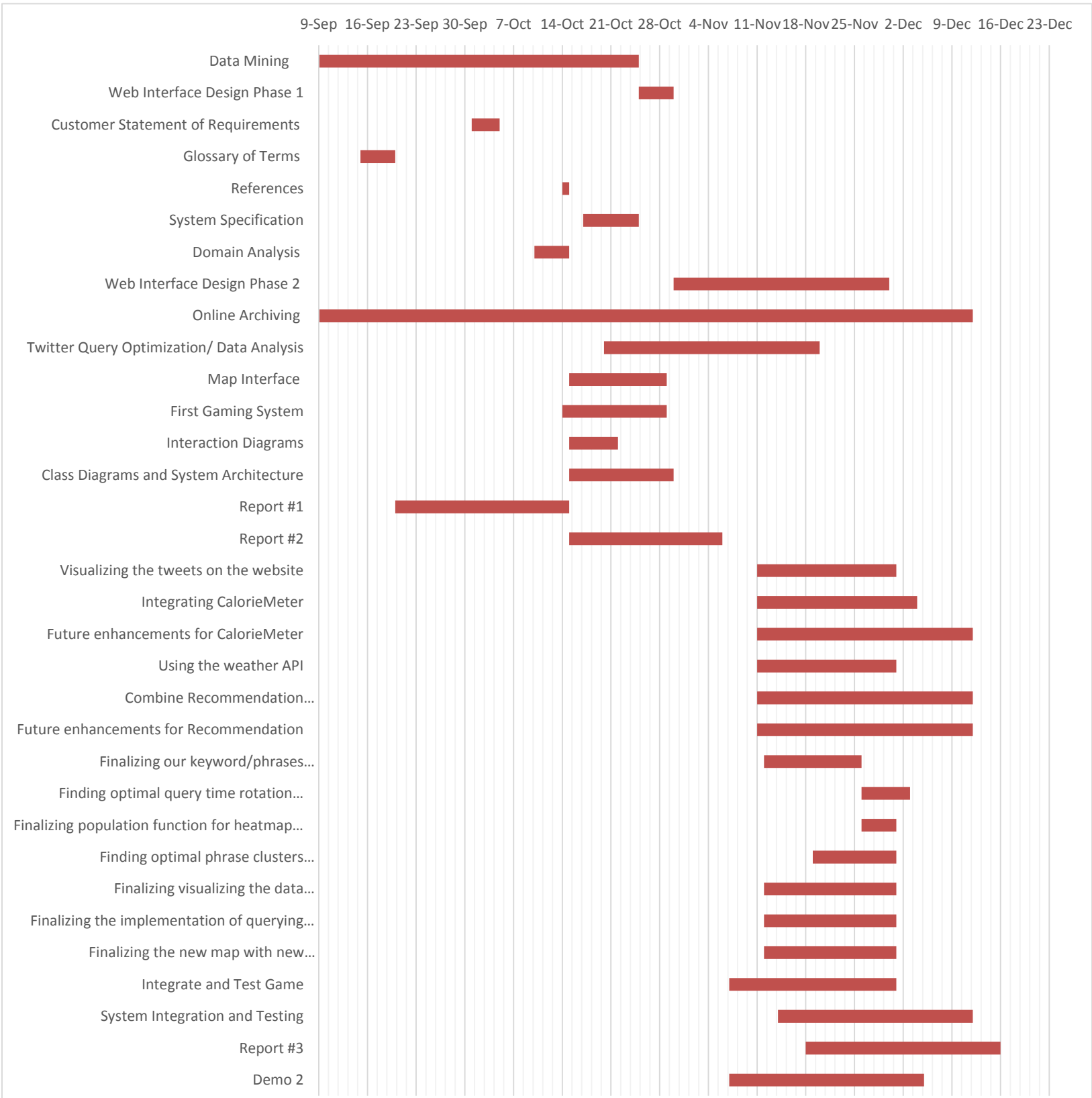

Figure 26 Ghant Chart of the Tasks over the whole project

Reference

[1] S. Kumar, F. Morstatter and H. Liu, "Twitter Data Analytics," Springer, 19 08 2013. [Online]. Available: http://tweettracker.fulton.asu.edu/tda/. [Accessed 15 10 2014].

[2] M. A. Russell, Mining the Social Web, Sebastopol, CA: O'Reilly Media, Inc., 2014.

[3] Twitter, "Twitter Developers," Twitter, [Online]. Available: https://dev.twitter.com/. [Accessed 09 09 2014].

[4] G. D. Clark, X. Gao, R. Xu, L. Xu, Y. Qian and a. X. Yu, "Group #1—Health Monitoring Analytics," 12 2013. [Online]. Available: http://www.tru-it.rutgers.edu/takmac/student/2013-g1-ProjectFiles.zip. [Accessed 10 10 2014].

[5] G. Yang, Y. Ji, S. He, H. Yu, C. Liang and a. X. Liao, "Group #2—Cities Activity Monitoring Analytics," 12 2013. [Online]. Available: http://www.tru-it.rutgers.edu/takmac/student/2013-g2-ProjectFiles.zip. [Accessed 10 10 2014].

[6] S. Yang, L. Liu, X. Chi, Y. Dong and a. Q. Shen, "Group #3—Health Monitoring Analytics," 12 2013. [Online]. Available: http://www.tru-it.rutgers.edu/takmac/student/2013-g3-ProjectFiles.zip. [Accessed 10 10 2014].

[7] I. Marsic, "Software Engineering book," 10 09 2012. [Online]. Available: http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf. [Accessed 10 10 2014].

[8] R. Kumar, "How To Setup MongoDB, PHP5 & Apache2 on Ubuntu," tecadmin, 20 05 2014. [Online]. Available: http://tecadmin.net/setup-mongodb-php5-apache2-ubuntu/. [Accessed 15 10 2014].

[9] J. Roesslein, "Tweepy Documentation," 26 04 2014. [Online]. Available: http://tweepy.readthedocs.org/en/v2.3.0/. [Accessed 07 11 2014].

[10] M. Dirolf and B. Hackett, "PyMongo 2.7.2 Documentation," 03 04 2014. [Online]. Available: http://api.mongodb.org/python/current/. [Accessed 07 11 2014].

[11] Python Software Foundation, "5.5 Data Structures Python 2.7.8 Documentation," Python Software Foundation, 28 10 2014. [Online]. Available: https://docs.python.org/2/tutorial/datastructures.html#dictionaries. [Accessed 12 11 2014].

[12] MongoDB, Inc., "MongoDB," MongoDB, Inc., 2013. [Online]. Available: http://www.mongodb.org. [Accessed 12 11 2014].

[13] BSON, "BSON - Binary JSON," BSON, [Online]. Available: http://bsonspec.org/. [Accessed 12 11 2014].

[14] JSON, "Introducing JSON," JSON, [Online]. Available: http://json.org/. [Accessed 12 11 2014].

Appendix

A.1 Group Discussion Records

A.1.1 Group Discussion Record 1

Time: 2014-09-16 4p.m.-7p.m.

I. Previous Project

Very similar, uses “combination words” to gather information, draw a heat map

II. New points for our project

a. Show ratio of favourite sports (distribution by states), draw a heat map with percentage of popularity of each sports. For example, if a state has low population and when in a traditional heat map it may not be seen on the map. However, if people there are more interest about one kind of physical activity, we can use this kind of heat map to see. (*Dean*)

b. “Game”. Including a game system in our application, people who exercise or invite more people to exercise can earn points. On the other hand, people who do not exercise will lose points. The game would encourage people to do exercise and be healthy. The detail of the game would be talked about later. (*Ioannis*)

c. Combine weather forecast in the application to help users plan their physical activities. For example, recommendation the nearest gym when in a rainy day. Or recommend of special sport events. (*Dean*)

d. Recommend the group activities from the Internet (like Yelp), for example, show group activities within 5km, or within 10km. (*Ghassan*)

e. Recommend to users a nutrition plan or sports activity based on individually customized inputs into the application including height, weight, or other inputs, in order to help people achieve their optimal bodyweight.

III. Other points that are mentioned

a. Use another map instead of google map. (*Ghassan and Jalal*)

b. Find several special hashtags, which are automatically created by other health applications, like mapmyrun, to see if we can use this to give us precise data of physical exercise. (*Ghassan*)

c. Dean has an idea about how we use the twitter to draw a precisely heat map, as only a small part of twitter has location information. But he may need more time to think and describe his idea clearly to us.

d. Information security system, for example, fingerprint confirmation. (*Ghassan*)

IV. Discussion about II & III

a. For III.a, it is a problem to display, and potentially it is not important to just use another map to show our output.

b. For III.b, it is very likely that we could not use the information from other applications. But at least we can analyze those tweets that have those specific hashtags.

c. For II.c and II.d, we originally just wanted a recommendation of indoor/outdoor activity considering the weather report. In addition, now we think we can increase the range of recommendations, including special events or group activities.

d. For II.e, we have some different ideas about the recommendation. At the beginning, *Ioannis and Ke* wanted it to encourage people to have better shape, like helping people lose weight, by recommend physical activities and meal plan. However, *Dean* points it out that for some people, they may just want to be strong, then using only one ideal weight standard may not be enough. People wanting to get stronger would want a heavier optimal bodyweight, than those wanting only to be healthier. Also, if people are already in a perfect shape, the application may not have any advice for them. So we may need some functions that allow the users to set up specific goals and then the application would give advice according those goals.

e. *Jalal* gives us a very important point. That we need to use the twitter database as a basic tool to accomplish our ideas. We need to use data as a reference in our II.c,d,e.

f. *Ke* thinks that for III.d, the security system may seem “away” from the project, finally we agree to make it an optional goal. Ghassan thinks it would not be too hard to achieve based on his experiences.

V. Other points

a. Everybody should read about the book about Twitter database, it will help us to discuss about what we can do or what we cannot do in the following discussion

b. *Ke* would set up a google drive and invite everybody else.

c. The regular meeting of the team would be Tuesday 4p.m.-7p.m.

d. *Jalal* would take responsible to set up a google website.

e. Try things the previous group do.

f. *Jalal* would summarize the proposal and after everyone satisfied. It will be sent to the Professor.

VI. Relevant Discussion

Ioannis tell us some basic ideas about UML.

A.1.2 Group Discussion Record 2

Time: 2014-9-23 4p.m.-7p.m.

I. Data Mining following Previous Work

a. The Twitter book has two parts, one is getting data, one is analyzing data. The whole book uses Java, and does not use MySQL. Instead, it uses MongoDB. So to make it easy to get all the previous work and sample code from the book, we decided to use ***Java*** and ***MongoDB*** for our project.

b. *Jalal* recommend a book called *"Data Mining from the Society Network"*, it mentions that we may have the ability to make something that also uses **Facebook** or other social medium website.

c. Ioanis says he would contact facebook, and then we could at least find out the way to get data from Facebook, even if we cannot do that actually.

II. Proposal Discussion

a. We first discuss about the old four "New Points" in the proposal, which are these:

1. Data analysis improvement
2. Relative popularity for different sports
3. Indoor/outdoor recommendation considering weather forecast
4. "Game"

We think that 2 and 4 are perfect for what we are going to do, while i and iii pose several problems.

After last week's lecture, Professor mentioned a lot of how to distribute the work. And it would be hard for the whole group if one person doing something like Data Mining. Also, We want to have five "New Points" to let each member of the team *own* a point.

In this way, we would not let data analysis to be one "new point", but a job needs to be done by all of the team members.

So the Structure of the whole project would not be compartmentalized.

Instead, everyone should know something about data mining and filter to make the project "survive" even if others drop the course. After that, each one would do something

about the data analysis and visualization about the function of his ownership.

In this way, we need two more "new points".

Jalal mentions that he wants to do something about the map, to let it show more information than just an original heat map. Then *Dean* gets confused about how it would be different from the original point 2(II.a.ii). We then agree that the map coming from II.a.ii or other information would mainly focus on relative popularity, and Jalal would do some discovery about other information which we may discuss in the following GD.

Ghassan has an idea of "Goal achievement", to gather information about people setting goals about sports or losing weight, then tracking the same person if he or she finishes his or her goals. It would be amazing if we knew how people around us finish their exercise goal or not. But unfortunately, we conclude that this is a "Mission Impossible". It is hard to precisely define when a twitter user sets a goal, among other similar tweets, and it would be very hard to know if he or she succeeds or not. Too many unpredictabilities would make the idea impossible to accomplish. Then *Ghassan* comes up with another idea called "Calorie Meter", it would tell people how many people around him or her burned how many calories today, this week or etc. We think it would be great to have this calorie meter to encourage people to exercise.

Thus finally, we have our updated "New Points" and also their ownership:

Chart / Histograms Output Implementation	Dean
Recommendation Implementation	Ke
Game	Ioannis
Visualizing Extra Information on Maps	Jalal
Calorie Meters	Ghassan

III. Other Discussion

a. About next meeting

Everyone would make a document like *Ioanis* already did. The document includes the requirements of his new points. We would discuss everyone's document next time and then discuss other requirements outside the five points

b. About Data Mining and Filter

Jalal, *Ghassan*, and *Ioanis* start to get to know how to accomplish the data mining. Meanwhile *Ke* and *Dean* would look at the filter part, try to accomplish what the previous group did. This mission starts right now.

A.1.3 Group Discussion Record 3

1) Ghassan quickly summarizes last lectures basic points. He says that professor Marsic considers Requirements Description very important. That way, he can understand the logic behind a requirement, and whether he agrees or not with the requirement. The discussion continues on how the requirements should be written.

2) Ghassan points out that some requirements may depend on other features. Ioannis gives as an example of one of his requirements that may depend on the feature that Ghassan is responsible for. Also, he suggests that requirements that are dependent to features of another group member can be optional. That way, they can be implemented if all goes well.

3) Continuing the discussion from the previous meeting about data mining from Twitter, Jalal says that there are some specific steps that need to be taken before someone is able to get data from Twitter. Those are:

a. An existing twitter account.

b. Sign in to Twitter Dev.

c. Give the necessary information that Twitter asks, such as application name, application code development site, etc. It is important to read the devel policy before accepting.

4) Jalal then shows the way that the data is taken from Twitter by presenting that the above procedure was for Twitter to provide the necessary authentication keys. Also, points out that the book “Mining the Social Web” uses python to extract data. Ioannis says that python can be an easy way to extract data quickly because python runs in user space and does not require installation of any specific IDE tools. Ioannis has to provide Dean with the python program that will help him draw Tweet data his geo-tag algorithm before the actual implementation in Java.

5) Ghassan shows us tweetchup.com. A Twitter analytics page that can be helpful and provide ideas for our project.

6) Continuing the discussion about data mining, Jalal explains the use of the authentication codes and how to run authentication for the REST and Streaming API. He also says that we should use the Streaming API, because it is more real time than the REST.

7) We should use MongoDB, because its logic is closer to how Twitter delivers data. It was noted that only Group 1 from previous year used PHP and MySQL.

8) Ioannis was concerned how to connect the MongoDB database with the Apache server and Ghassan said that there is Apache Camel, a MongoDB implementation for Apache.

9) The conversation continues with the requirements explanation. Jalal was the first and he showed a reference to his posted document for details. He also said that in the beginning he will use static activities but he would like to have dynamic.

10) Dean explains his requirement document. Ioannis proposes that we could use Google Maps API to get the coordinates, but Ghassan said that this way we might reach easily Google’s quota.

11) Towards the end Jalal pointed the importance to reach Thursday’s deadline and we started looking at what the first part of the first report should have.

12) The conversation continued about how the User Interface should look like. And we made a rough sketch. Look at the picture at the end.

13) Finally, we decided that each one of us has to finish what is needed for the first

part of the 1s report and update the proposal document. Also, we are leaving out the Plan of work and we will decode about it on the next meeting. Ghassan will send the first draft of the glossary of terms and the rest of us will add anything that might be needed.