Health Monitor Analytics

GROUP 1

Report 2

D. Takacs

G. Bati

I. Paraskevakos

J. Abdulbaqi

K. Dong

Project Website: https://github.com/mohjaba/Health-Activity-Monitoring

Individual Contributions Breakdown:

"All team members contributed equally."

Table of Contents

1. Interaction Diagrams

1.a Sequence Diagrams

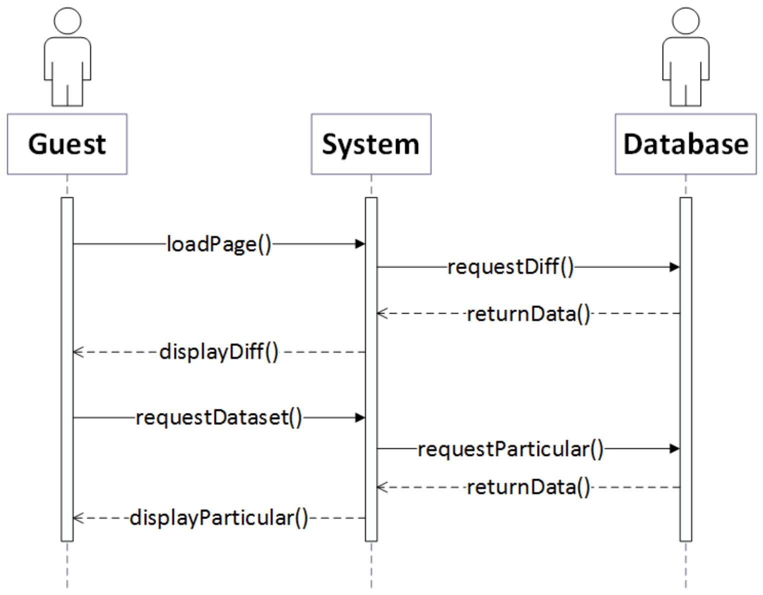


Figure 1 System Sequence Diagrams of UC-3

Figure 1 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.

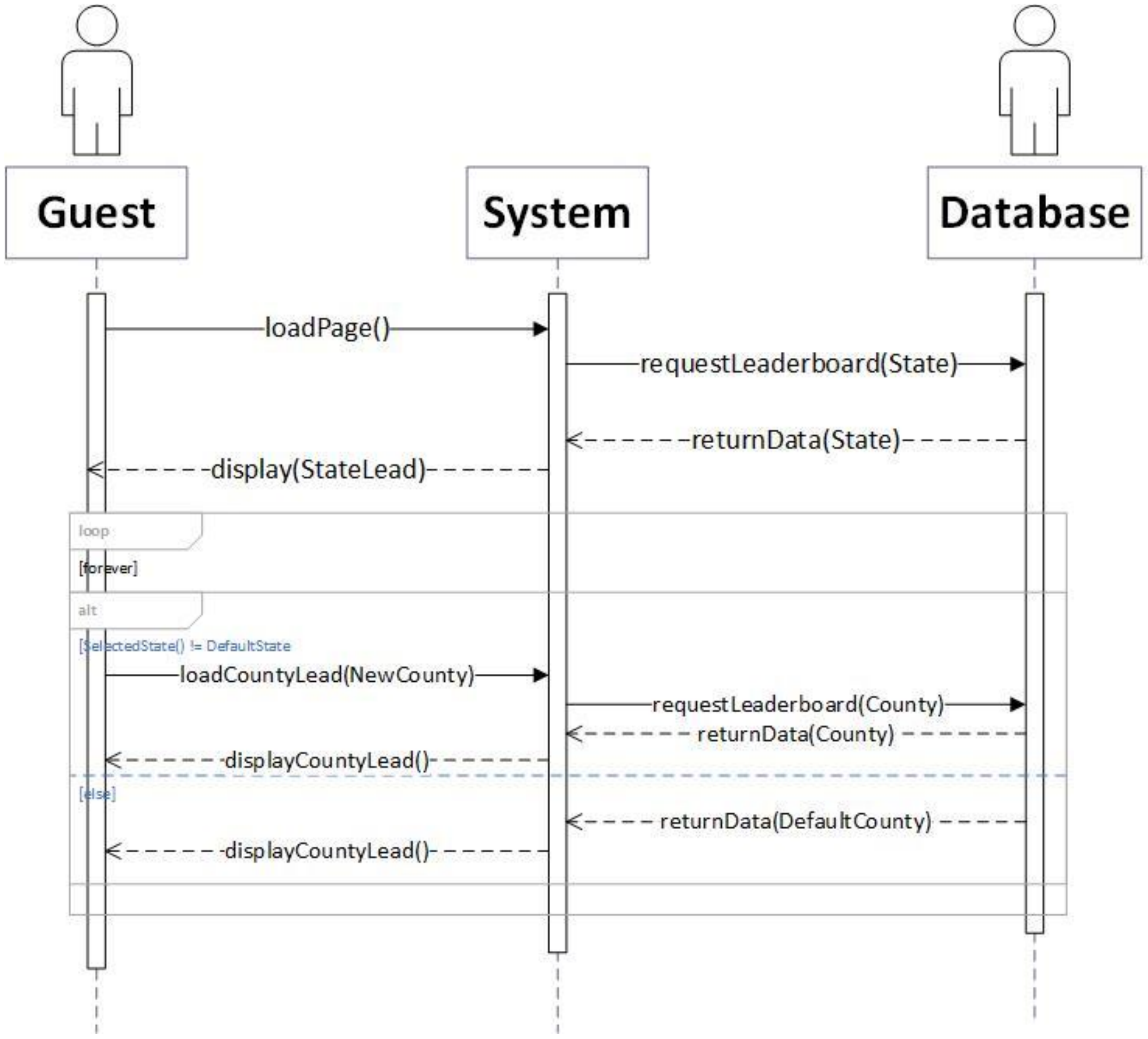


Figure 2 System Sequence Diagrams of UC-6

Figure 2 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, a forever loop checks if the user has selected a different State from the default, which is New Jersey, for the County 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 County Leaderboard of the selected State, otherwise the system automatically queries the database for the New Jersey County Leaderboard. In both alternatives, after the sequence of requests and queries, the results are displayed on the page.

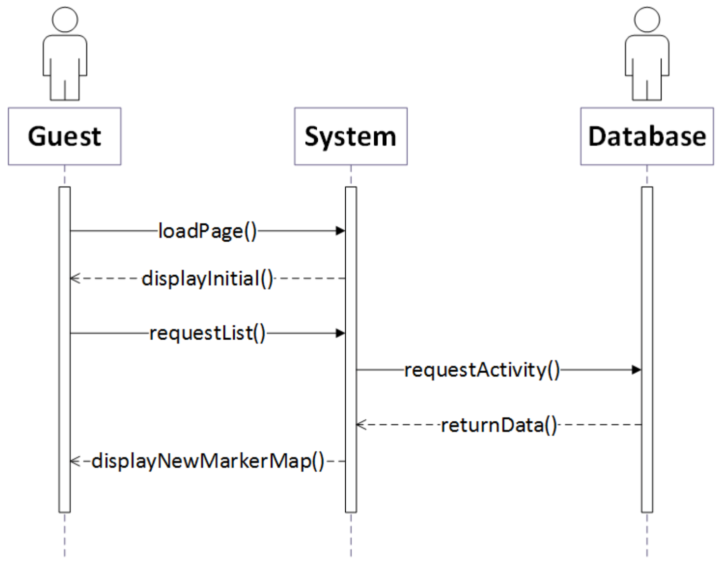


Figure 3 System Sequence Diagrams of UC-8

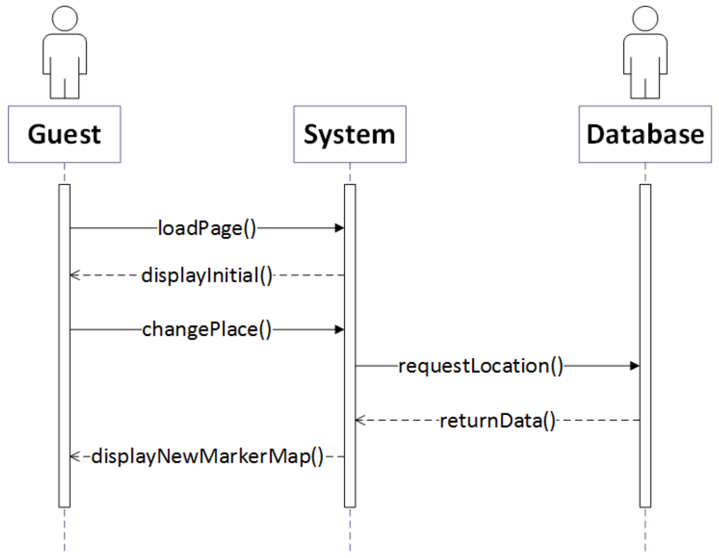


Figure 4 System Sequence Diagrams of UC-9

Figure 3 is the System Sequence Diagrams of UC-8, Figure 4 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.

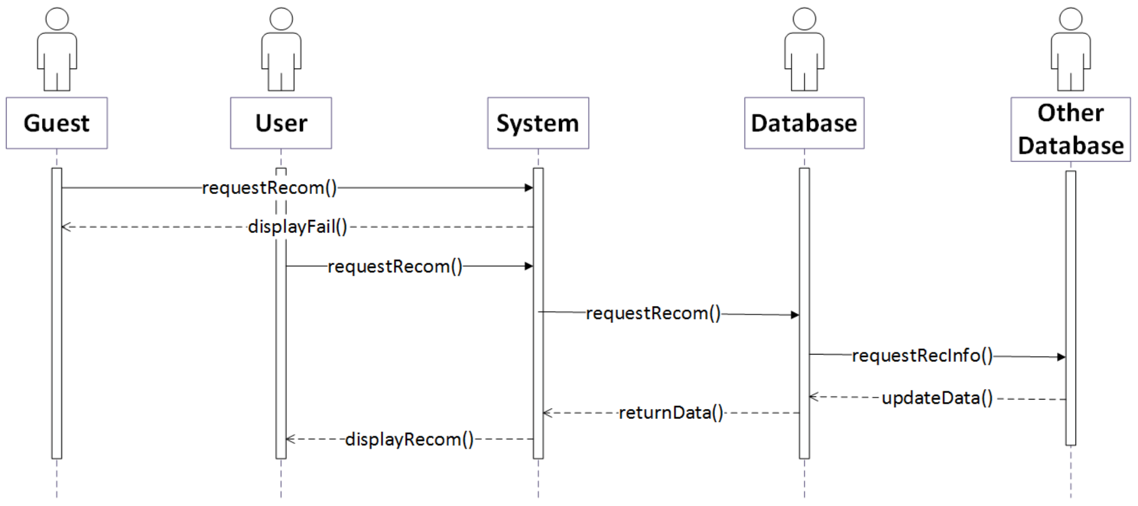


Figure 5 System Sequence Diagrams of UC-11

Figure 5 is the System Sequence Diagrams of UC-11. It is for the recommendation subsystem. The recommendation subsystem would deny any request from an unregistered guest (because a login is necessary for this subsystem), then based on a user's request, get information from the local database. The local database would then request the additional required data from the other database (such as weather data) and return the data to the local database. The system would then use the data and display the recommendation to the user.

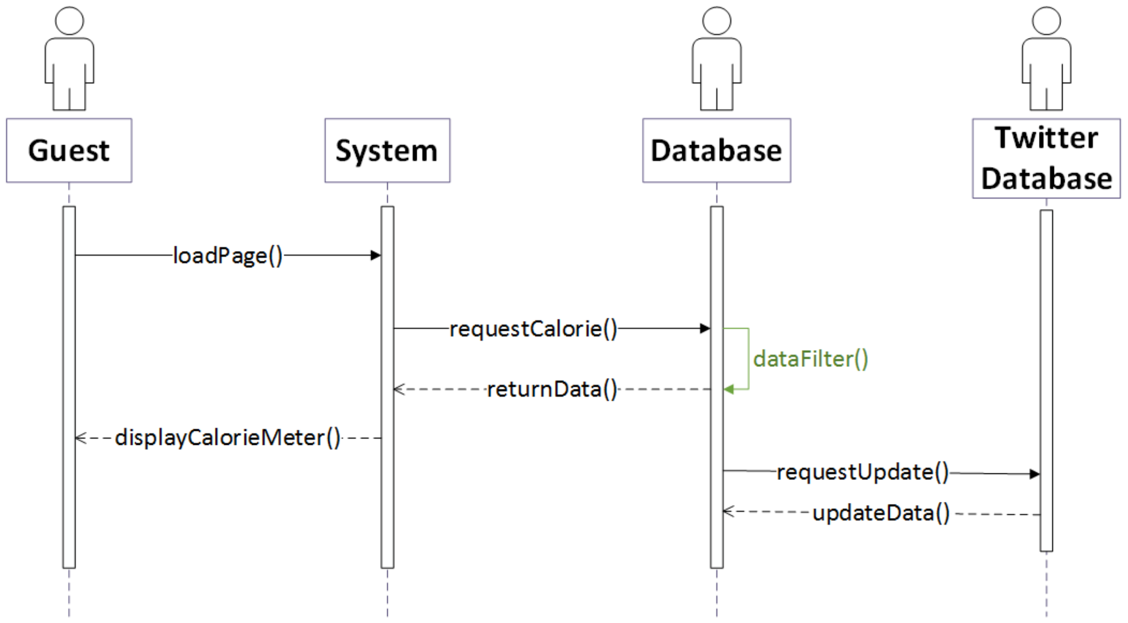


Figure 6 System Sequence Diagrams of UC-13

Figure 6 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.

2. Class Diagram and Interface Specification

2.a Class Diagram

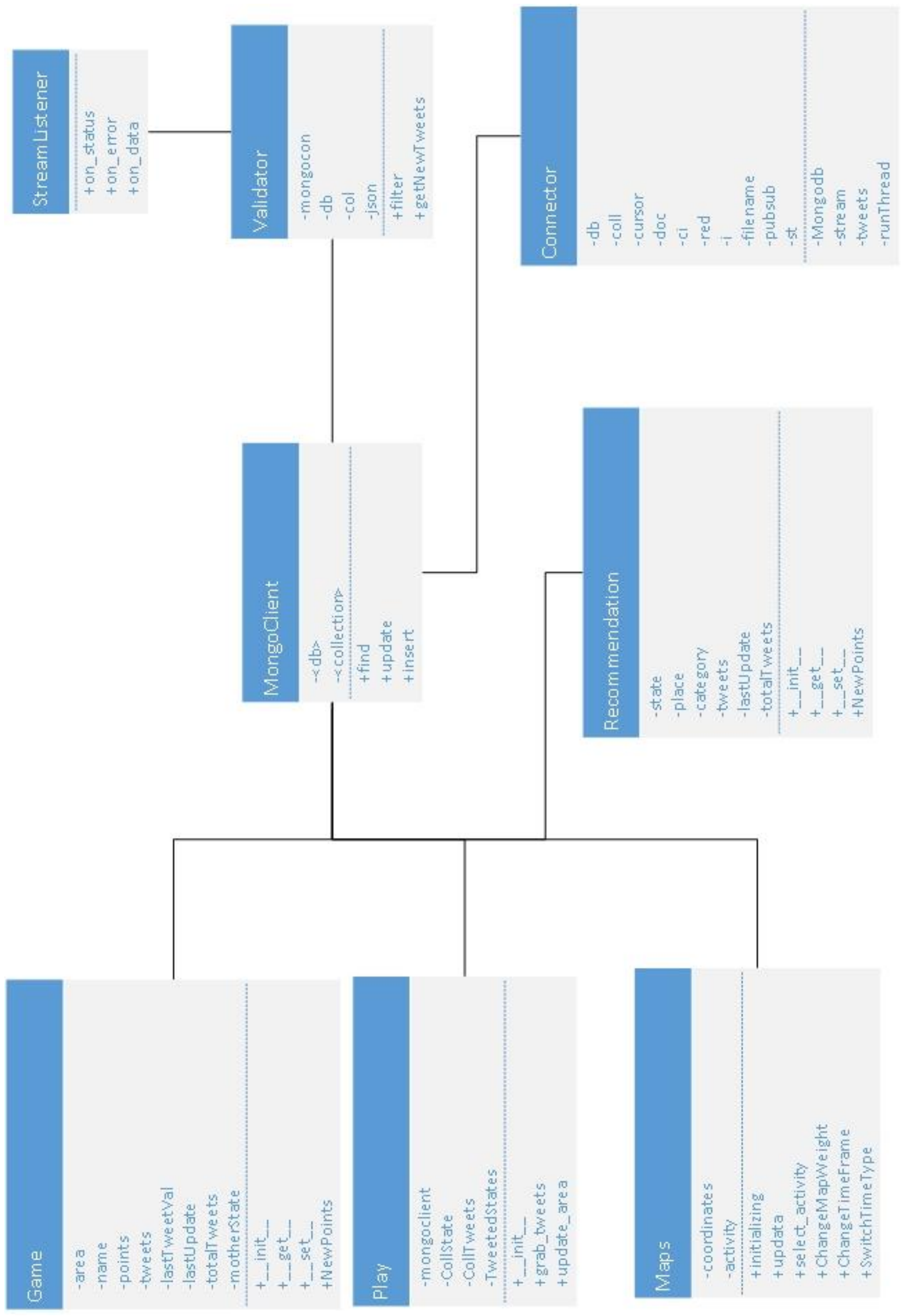


Figure 7 Class Diagram

2.b 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 this class are:

- mongocon : Object of the MongoClient class. This is the actual connection between an object of the Validator class and MongoDB
- db : The database that the data will be stored in. This is an object of pymongo Database class [reference to pymongo]
- col: MongoDB database collection. This is an object of the pymongo Collection class [reference to pymongo]
- json : This is the json file descriptor that the data will be written in. This is a python module.

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.

- Filter(self,tweets) : This method gets as input the raw tweets that have been received by Twitter’s Stream API. It analyzes the tweets, discards all those tweets that are not relevant to exercise, and stores them inside the collection of MongoDB.
- getNewTweets(self, StreamListener) : This method gets as input an object of the StreamListener class and utilizes the method that Twitter API has available to get data from Twitter

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.

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,tweets,area,name,motherstate) : This method is used to call the game class and update the database entry for that specific area. The parameters are:

    - tweets: The number of new tweets that will be used to update a specific area.

    - area: This parameter shows the type of the area. It is a string that can take the values 'state' or 'county'

- name: A string with the name of the area. It is a string.

- motherstate: This parameter has the acronym of the State that the County in is. It is a string.

Recommendation:

This class is the one that accomplish the recommendation and connect to the MongoDB. The attributes of this class are:

- state : This is a string of which state it is
- place : The name of the place should be smaller than the state. This is a string.
- category : The category of sports. This is a string
- tweets : The number of unused tweets for this state of this specific category from 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 of the state. An integer number.

The methods of the class are:

- __init__(self, state, place, category, tweets, lastUpdate ,totalTweets) : Class Constructor. The constructor initializes the class' attributes. The parameters are directed 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 state 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 the difference was from the last update.

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.

2.c Traceability Matrix

Table 1 The Traceability Matrix of class and domain concepts

Domain Concepts	StreamListener	Validator	MongoClient	Game	Play	Recommendation	Maps	Connector
R1	X	X	X					
R2			X					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					X
R12			X	X		X		
R13				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.

3. System Architecture and System Design

3.a 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.

3.b Identifying Subsystems

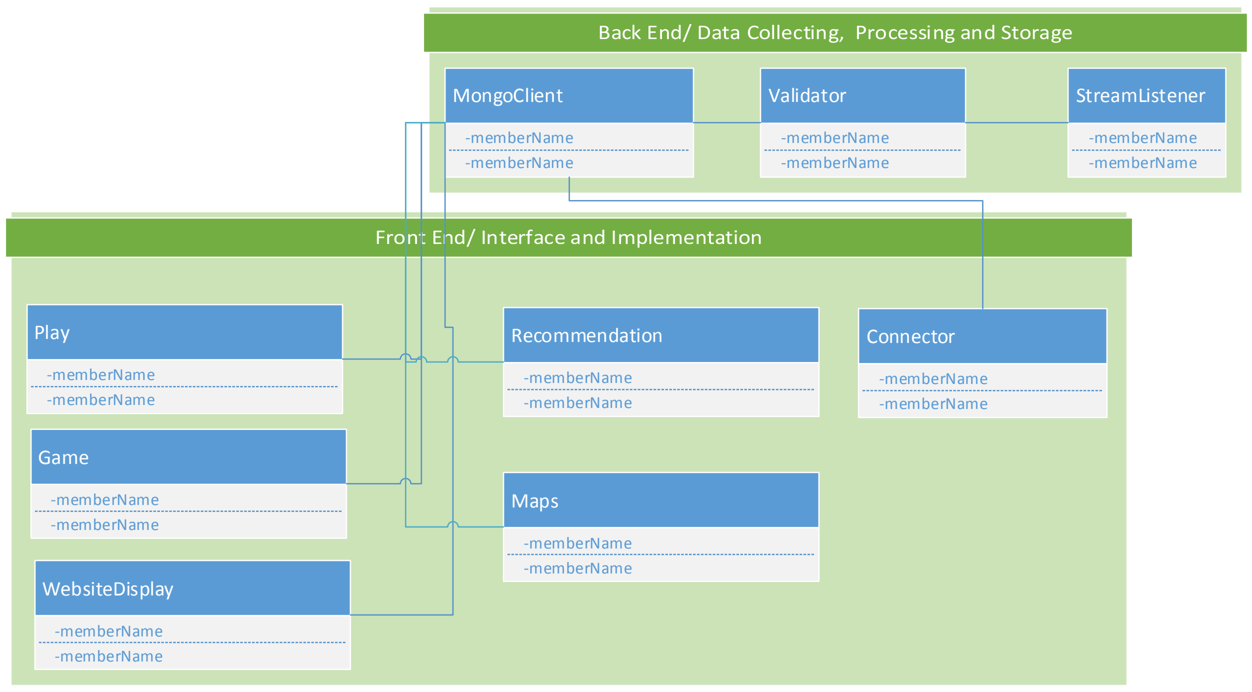


Figure 8 Subsystem

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.

3.c 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.

3.d 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
```

3.e 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).

3.f Global 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.

3.g 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.

4. Algorithms and Data Structures

4.a Algorithms

4.a.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.

4.a.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.

4.a.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.

4.a.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.

4.a.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 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. 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 11***

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 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 11***

5. 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 11***

6. 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 11***

7. 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 11***

8. 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 11***

9. 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 11***

10. For any other case ***do not change the number of points***.

11. If there is an update due to the time passed, ***update the points and go to 13***

12. 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.

13. Return the statistics and the points for the Area

4.a.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.

4.b 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).

5. User Interface Design and Implementation

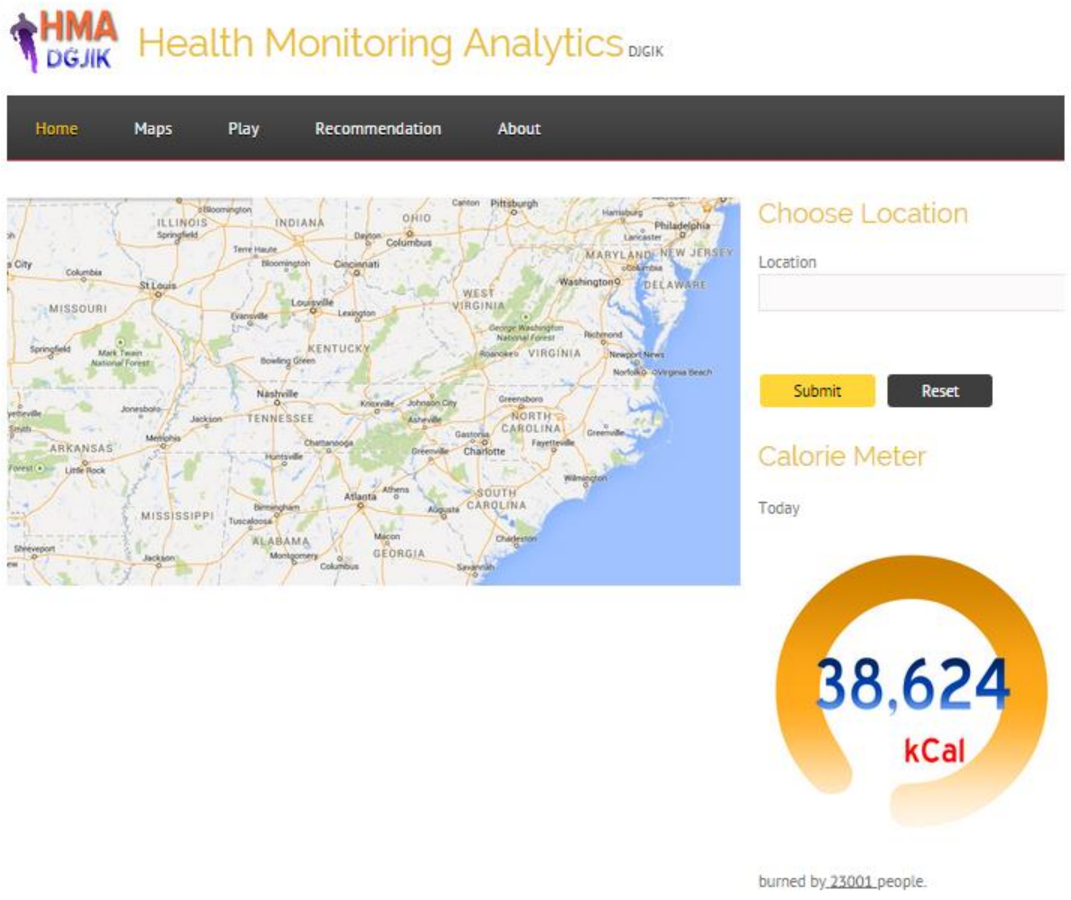


Figure 9 Home Page of the Website

Figure 9 is the home page of the website. In the home page, there will be an old-type heat map with a calorie meter. A user will be able to see the total of the burned calories taken from tweets generated by certain devices and mobile applications, in form of a visitors' counter. This counter alike shall be presented in the main page of the website. The Calorie Meter shall be updated and zeroed daily.

Figure 10 is the Map page. 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. The Interactive Activity Map page would be very similar to the Per Capita Map page, only the user would click a drop down list to choose between the different kinds of sports activities.

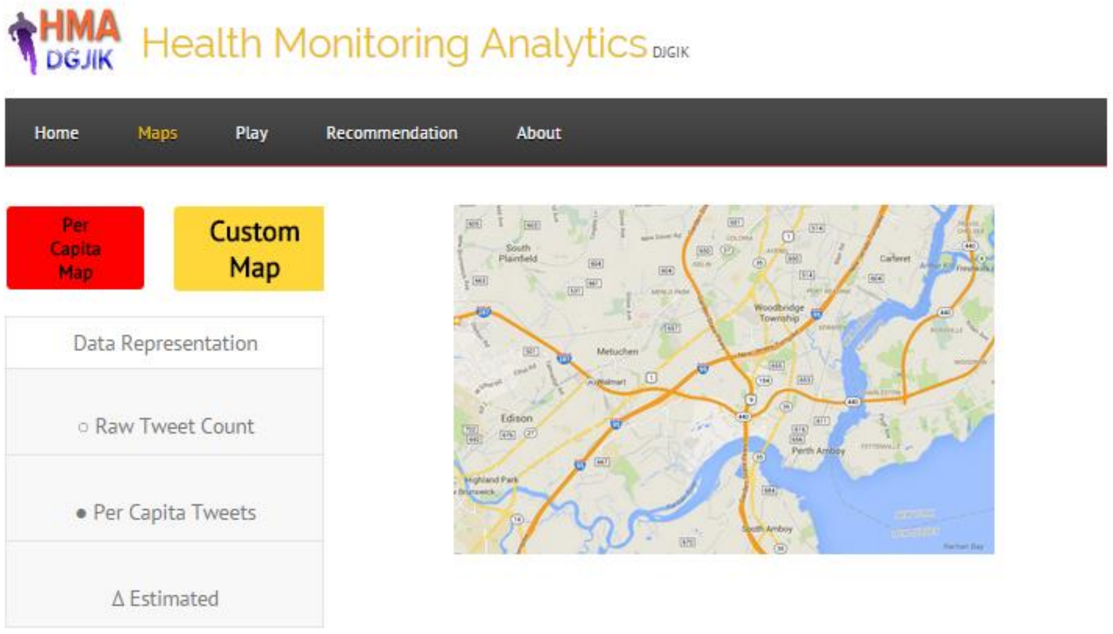
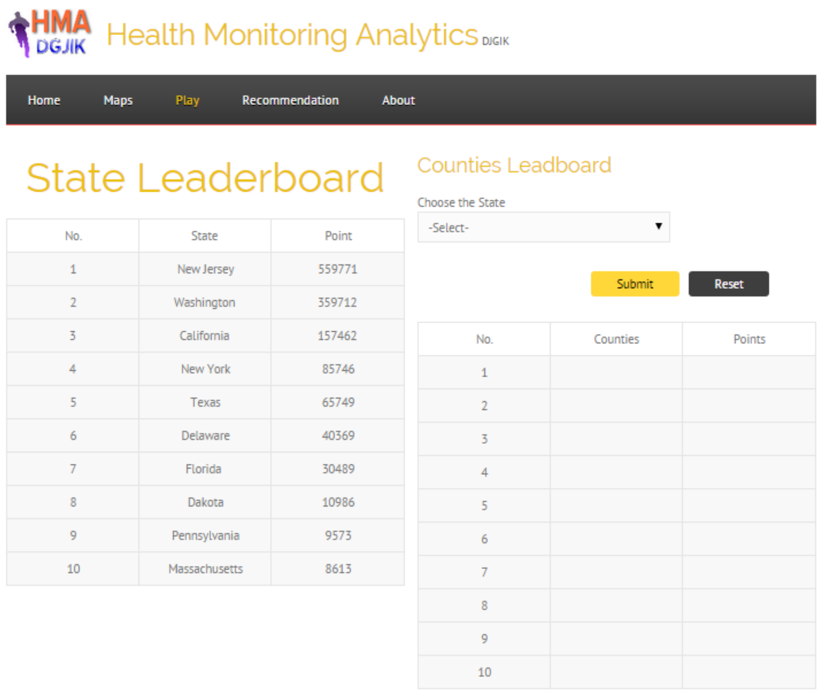


Figure 10 Map Page of the Website

Figure 11 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 counties of that state similar to the State Leaderboard.



State Leaderboard

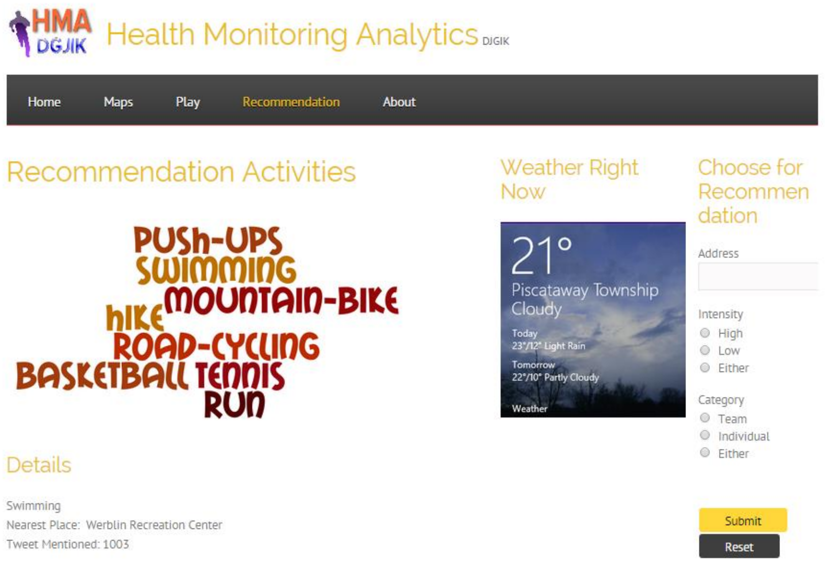
No.	State	Point
1	New Jersey	559771
2	Washington	359712
3	California	157462
4	New York	85746
5	Texas	65749
6	Delaware	40369
7	Florida	30489
8	Dakota	10986
9	Pennsylvania	9573
10	Massachusetts	8613

Counties Leadboard

No.	Counties	Points
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Figure 11 Play Page of the Website

In the Recommendation Page, shown by Figure 12, the user can click to choose the sports category, then click submit. The word cloud would be shown on the left part of the page. Then the user can click each word of the word cloud, and after that, the detail of that sport would be shown on the window below the word cloud.



Swimming
Nearest Place: Werblin Recreation Center
Tweet Mentioned: 1003

Figure 12 Recommendation Page of the Website

6. Design of Tests

6.a 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.

6.b The Unit Test for the Game System

6.b.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.

6.b.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.

6.b.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

6.c 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.

7. Project Management and Plan of Work

7.a Product Ownership

Table 7-1 Product Ownership

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

7.b Breakdown of Responsibilities (Report 2)

Report 2	D. Takacs	G. Bati	J. Abdulbaqi	I. Paraskevakos	K. Dong
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		√			

7.c 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

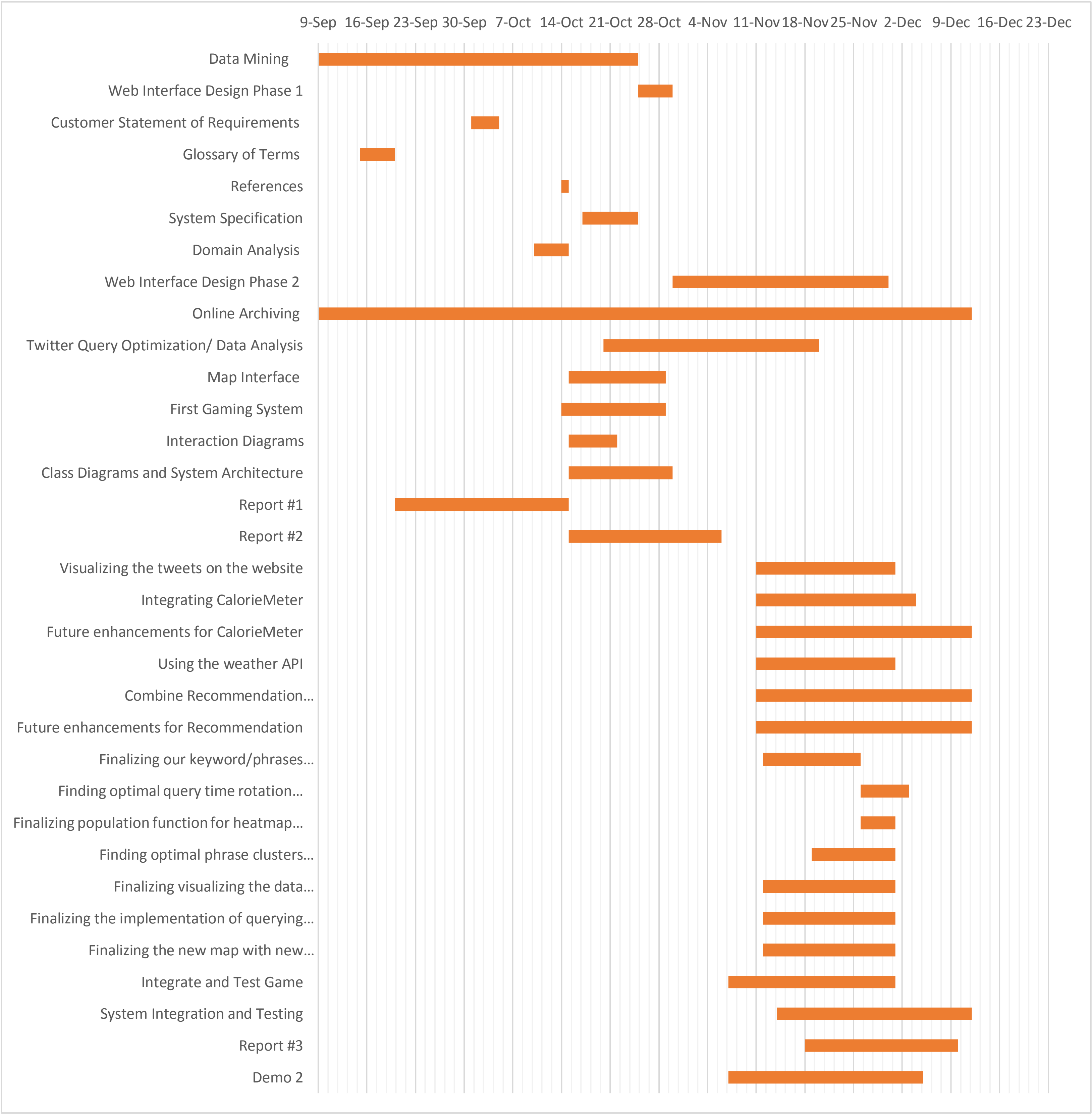


Figure 13 Ghant Chart of Tasks

Bibliography

[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

Appendix.A Use Cases Causual Description

UC-1 **Log in:** Allow the user to access the system Derived from REQ30
UC-2 **Sign up:** Allow the guest to create a new profile Derived from REQ29
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, REQ11.
UC-7 **Change County:** Allow a Guest to select a State from a drop-down list and see the County Leaderboard for that State Derived from REQ12
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 REQ14, REQ16.

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 REQ15, REQ16.
UC-10 Choose the Recommendation Category: Allow the user to choose recommendation sports in categories. The user shall choose one from “Low-intensity” or “High-intensity” and then choose one from “Individual” or “Team”. Derived from REQ19 and REQ20.
UC-11 Show Recommendation Word Cloud: Allow the user to see a word cloud after choosing the sports category, the word cloud would include 8 sports/activities that are recommended. The more recommended activity will have a bigger font size. Derived from REQ21 and REQ22.
UC-12 Show Recommendation Details: Allow the user to click on the word cloud and get detailed information about this sport. Derived from REQ23 and REQ24.
UC-13 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 REQ25, REQ26, REQ27, and REQ28.

Appendix.B 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	The system shall at a minimum, allow heat map to be displayed for major	4

[1] Priority Weight

(D)	metropolitan cities	
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 should assign points for different types of exercise.	1
REQ – 11 (I)	The system should show a leader board per area	2
REQ – 12 (I)	The system shall show a leader board for States and County in a selected State	4
REQ – 13 (I)	The system should allow users to set community goals and assign points based on their completion	2
REQ – 14 (J)	The system shall allow the user to choose a specific activity from a list.	5
REQ – 15 (J)	The system should update the activity list as soon as the map area changed.	3
REQ – 16 (J)	The system shall display location markers on the map, for any activity chosen from the list.	4
REQ – 17 (J)	The system should display markers from different activities with different colors.	2
REQ – 19 (K)	The system should allow the user to choose activity type between “High-intensity” or “Low-intensity”	2
REQ – 20 (K)	The system should allow the user to choose activity type between “Individual” or “Team”	2
REQ – 21 (K)	The system shall recommend activity based on weather, geology information and activity population nearby.	5
REQ – 22 (K)	The system shall show user a word map to display the eight most recommendation activities.	5

REQ – 23 (K)	The system should allow user to click every word in the word cloud, and show detail information in the information list.	3
REQ – 24 (K)	The system should show change the information list when user click another word in the word cloud.	3
REQ – 25 (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 – 26 (G)	The system shall download all relevant tweets and then store them to a local database for analysis purposes.	5
REQ – 27 (G)	The system shall extract tweets from the local database after the necessary calculations.	5
REQ – 28 (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 – 29 (G)	The system shall allow users to sign up.	1
REQ – 30 (G)	The system shall allow users to log in with correct login ID and Password.	1
REQ – 31 (G)	The system should track the history of the tweets for a period of time to generate extra analysis.	3
REQ – 32 (G)	The current status of the counter should be shared through social media websites.	4