<u>Summary of Twitter project – John B. Hinkel, III</u>

1. Overview:
    a. Professor Henry Kautz at the University of Rochester is doing a Big Data analytics project where he uses a data collection service (DataSift) to gather tweets from Twitter. He uses these tweets to predict health trends (the spread of flu, food poisoning, etc). His overall plan is to use these tweets in a system that would allow users to see where health issues lie in order to raise awareness when users go to a less-than-healthy area. More details of his research are in his New York Times article here: http://www.nytimes.com/2013/06/23/opinion/sunday/theres-a-fly-in-my-tweets.html?_r=0
    b. My role within this project is to develop an analytics tool that Professor Kautz's research group can use to analyze tweets. He specifically wanted the ability to
        i. Plot tweets as markers on a google-maps-style interface.
        ii. See satellite map views as well as street map views so he can see exactly where each tweet marker is.
        iii. Be able to click on each tweet marker and see the username that tweeted it and what they tweeted from that location
    c. Basically, this app would allow him to analyze heat-map-style distributions and see exactly where the particular health trend occurs at a street-level granularity.
2. Steps required to create the application
    a. Database creation
        i. There was no central location where Professor Kautz and his research group stored tweets, so I had to create a database that could be accessed from anywhere. Doing that involved creating a server from scratch to hold the database.
        ii. The server is hosted on Rackspace, a cloud hosting service. Basically, the server runs within a virtual Linux environment (specifically the Debian distribution of Linux). Rackspace gave me a blank install that had very little installed, so I needed to spend some time completely configuring the server. Configurations included
            1. Installing the ability to run java programs (the java and javac command utilities) and configuring the server to recognize the commands to run those utilities.
            2. Creating an administrator user on the system.
            3. Installing the database system (couchdb), and creating an HTTP server within this virtual environment so that any user can access the graphical database management utility that comes with the couchdb database system.
            4. Adding a line to the virtual environment's boot routines that automatically starts the database when the virtual environment is activated.
    b. Data processing to allow uploads to server
        i. There were two types of tweets: legacy tweets that were downloaded using a different API, and tweets from DataSift.
            1. Tweets that come from this DataSift service are in JSON format (a parseable delimited format). The legacy tweets were also in JSON, but the difference was the line spacing. Legacy tweets had one line of JSON for each tweet. However, DataSift had all tweets that were returned from a specific download in a nested array.
        ii. Professor Kautz wanted the ability to upload each tweet as an individual database entry, regardless of whether it came from legacy data or DataSift data.

iii. Therefore, I had to write two separate shell scripts (scripts that can be run from a terminal and process files in certain locations) to take a file, parse it into an uploadable format and upload it.
    1. The shell script for legacy data executed the following steps:
        a. Take the file and output it into the standard input stream.
        b. Read from the standard input stream, line by line.
        c. As each line is being read, upload it to the database.
    2. Since the DataSift data is only one line, the shell script for DataSift data
        a. Takes the file and outputs it into the standard input stream.
        b. Uploads the entire standard input stream at once to the database.
    3. For the datasift data, I then had to use the graphical database management utility for Couchdb to parse the data into separate tweets (the utility had the ability to use javascript to form “data views” with the data structured the exact way you wanted it).

c. Now that the server and the data upload scripts were all set, I had to focus on actually creating the application. The application had four different parts to it
    i. The map interface API
        1. The interface API came with a basic application that had the map screen and controls (zooming, layering of points, panning the map, etc) already set up.
        2. I use this application as a base for the program and expanded it by adding in the ability to plot tweets from the database.
        3. Additionally, Professor kautz wanted to plot tweets with a google-maps-style marker instead of the circle that came default with the software, so I wrote a class that extended the API to handle using images from a folder as markers. Each image marker holds a latitude, longitude, and ID of the tweet it corresponds to.
        4. Now, the API was completely ready to plot tweets.
    ii. The couchdb database query API
        1. Database queries into couchdb are run using the custom views I had created during the data processing steps described above.
        2. Within the code, I specified which view I wanted to return data from and how much data I wanted to return using a ViewQuery object (built into the API).
        3. Then, to query the database, you run a query with a method call and store the results of the query in a ViewResult object.
        4. This ViewResult object is then passed to a method that handles the parsing of the tweet metadata the view just returned.
    iii. Tweet Parsing
        1. Using regular expression matchers that are built into Java, I parsed the numbers that corresponded to the latitude and longitude into two separate arrays of numbers (one array for the latitudes and one for the longitudes).
        2. After forming these two arrays, I looped through each array and plotted a marker on the map with latitude and longitude that corresponds to each number in the two arrays. (i.e. the first index in the latitude array, coupled with the first index in the longitude array, constitutes the location information for a whole tweet marker).
    iv. How the username and tweet content are displayed to the user

1. All the markers generated in the Tweet Parsing step are stored in an array. Each marker in this array has latitude and longitude properties that I can access.
2. The Map interface API captures the longitude and the latitude of the mouse cursor when you click on a certain spot on the map and returns it to the user.
3. Using this feature, I get the latitude and longitude of the mouse cursor and compare it to the longitude and latitude of the tweet the user wants to click on. This comparison is what tells me the user has clicked on a map marker (tweet).
    a. However, only the tip of the map marker points to the true latitude and longitude of the tweet. To enable the user to click anywhere on the marker and still have the system interpret that correctly, I compare the latitude and longitude and tweet with a bounding box that surrounds the cursor.
    b. If the latitude and longitude of that tweet are anywhere within that bounding box, the system reacts as if you have clicked that marker.
4. After finding the marker that the tweet corresponds to, I query the database using the ID of that marker as a key. The database then returns the corresponding delimited tweet metadata.
5. After getting the metadata, I had to do some more string parsing with regular expressions in order to get the tweet content and the username. Once I got those out, I stored each piece of information to a variable.
6. I then showed the information to the user via a messagebox popup.
7. Everything in this subsection happens only when the mouse clicks on a tweet.

d. Future enhancements planned:
    i. Optimize the search process when the user clicks on a tweet. Right now the system takes around 3 seconds to return the username and tweet content in a message box, which isn't optimal.