Software Requirements Specification

GeoSkyNet

Version 1.0

Prepared By Team BRX

Aaron Dodson (gonfunko@gmail.com)
Anthony Coccia (anthony.coccia@gmail.com)
Kenny Bier (kenbier@gmail.com)
Matthew Greenfield (mattgreenfield1@gmail.com)
Peter Huang (peterhuang913@gmail.com)

With Mentors, Aerospace

Nehal Desai (nehalmeister@gmail.com)
Setso Metodi (t.metodi@gmail.com)

Instructor

Chandra Krintz

T.A.

Stratos Dimopoulos

Course

CMPSC 189A

Date

February 1st, 2013
Revision 1: March 4th, 2013

Table of Contents

1. Introduction

1.1 Purpose

This specification is built to ensure that all parties understand the requirements in the development of GeoSkyNet, a software system intended to extract location information from social networking messages. The specification is intended for the development team, the sponsors at the Aerospace Corporation, and the UCSB CAPSTONE faculty staff.

1.2 Scope

GeoSkyNet is a software system intended to extract location information from natural language social media messages and convert it into geospatial coordinates. Initially, the focus will be on processing messages from Twitter, with the goal of being able to identify and display the locations of selected food trucks within the United States. The system will consist of two primary parts: a backend responsible for data collection, processing, analysis and storage, and a user-facing frontend that offers a basic map view of food truck locations based on the data generated by the backend.

Initially, a list of food truck Twitter accounts will be monitored by processes designated to grab new tweets and store them in the database. When unprocessed tweets exist in the database, the message broker system will queue messages to multiple sensor processors that will attempt to identify location information. The results of each sensor processor is then appended onto the document stored in the database. If the confidence in the location information for a tweet is high enough, it can be queried for by the frontend.

1.3 Definitions, Acronyms and Abbreviations

- **Middleware** - a mediator between the database and the user interface
- **Natural Language** - the language in which humans speak: complex, ambiguous, and littered with idioms. Unlike mathematics, natural language is not exact and can be difficult for a computer to interpret.
- **NLTK** - Stands for “Natural Language Toolkit”; A library for Python that parses and stores natural language into data structures

1.4 References

NTLK – http://nltk.org
MongoDB – http://www.mongodb.org
Twitter – http://www.twitter.com
RabbitMQ – http://www.rabbitmq.com

1.5 Revisions

- **3/4/2013:** Added suggestion feedback from Chandra

2. Overall description

2.1 Product Perspective

GeoSkyNet will be an application that displays on a map the locations of food trucks. The goal is to automate the process of extracting accurate and specific geocoordinate data contained in an unstructured social media communication format. The food trucks data is an example data set that the final system can be deployed with.

While research, development, and tools exist in the problem domain, the GeoSkyNet software will be a new and standalone product. The software will use resources and algorithms that already exist combined with new solutions specific to our problem set.

2.2 Functional Overview

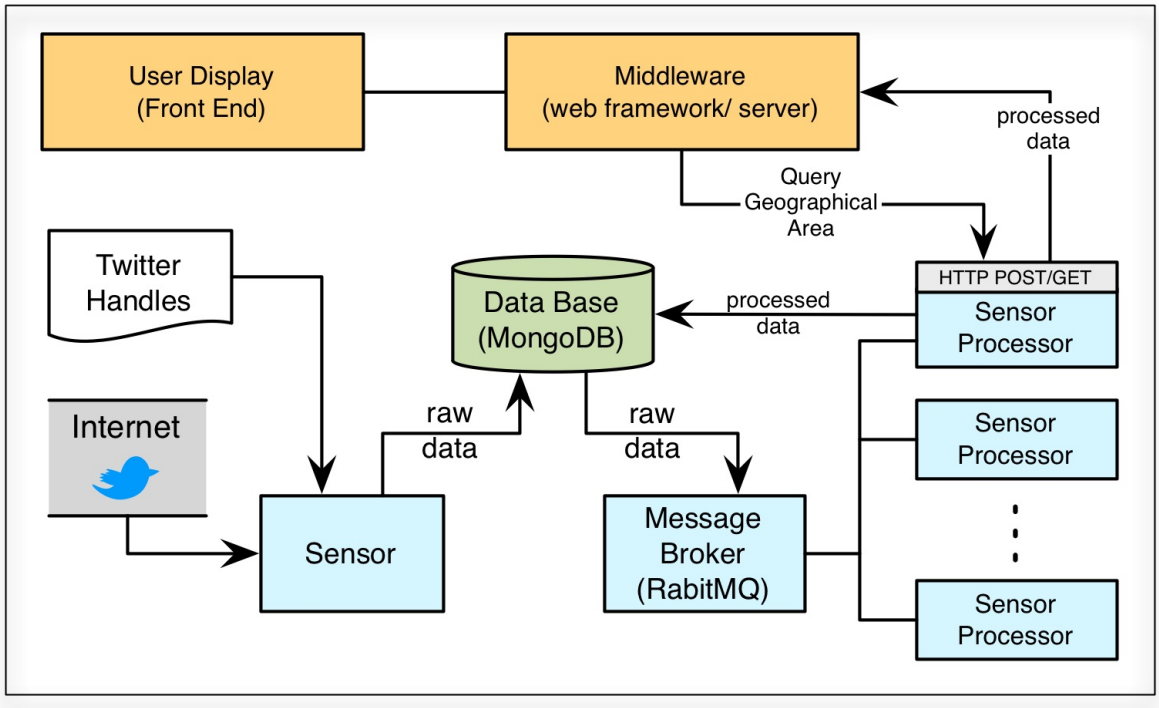

Figure 1: A functional decomposition of the Application

As shown in Figure 1, the application is composed of the following components:

- **Sensor(s):** The Sensors monitor a specific set of twitter handles in real time and pass the Tweets to the Message Broker as they arrive. Since we are focusing on food trucks, there could be a single sensor that handles all Twitter food truck traffic or multiple sensors that collect data for specific geographical regions, or even a sensor for each Twitter handle. The idea of sensors is that they are independent lightweight Python functions with the task of collecting very specific data.

- **Message Broker:** The Message Broker routes incoming sensor data to the appropriate sensor processor and is also in charge of arbitrating data exchange between the sensor processors. It is implemented using the Python-based RabbitMQ framework.

- **Sensor Processor(s):** The Sensor Processors transform the raw sensor data to its final form(s) and pass the data through the Message Broker to other sensor processors. One sensor processor is designed to interface with the web framework through a generic HTTP Post/Get interface in order to deliver the data to the front end for display to the user. Together, the sensor processors comprise a set of methods, written in Python, that Implement the data processing algorithms for extracting the geo-locations contained in each tweet. The sensor processors also update the data with routing logic that is used by the broker to determine which sensor processor to route the data to next.

- **Middleware and Front End:** The middleware implements a web framework that serves the data to the user through the front end. It interfaces through HTTP (GET/POST) with one of the sensor processors. The front end displays the data as geotags on a map. The information contained in each geotag consists of the food truck's geo coordinates (for a specific time period), associated address, and the original raw Tweet or combination of Tweets used to extract the geo coordinates from.

- **Database**: The database is central repository for storing all data for offline and generally long-term analysis.

2.3 User Characteristics

The primary users of the system are individuals interested in identifying the location of food trucks in the United States. They are not expected to be technically proficient, so the end-user facing part of the system will be designed to allow a novice user to find food trucks simply by visiting the website. The user will be able to query food truck locations based on

1. Current location
2. A specified location
3. A specific food truck
4. Food truck type

2.4 Operating Environment

At a minimum, the backend will require two Amazon EC2 instances running Ubuntu 12.04 with Python, MongoDB, RabbitMQ, and a web server installed. The user-facing website must be compatible with the latest versions of Safari, Chrome, Firefox, Internet Explorer and Opera running on Windows, Linux and Mac systems (where supported by the respective browsers). Additionally, the website should be functional on the latest versions of the built-in mobile browsers for Android and iOS.

2.5 Design and Implementation Constraints

All sensors, sensor processors, and message broker must be implemented using the Python programming language. Requests to Twitter will be subject to its API rate limits (https://dev.twitter.com/docs/rate-limiting/1.1). Additionally, queries to Google's Geocoding API (https://developers.google.com/maps/documentation/geocoding/) are limited to 2,500 requests per day.

2.6 Assumptions and Dependencies

The algorithms will rely on Twitter's feeds and APIs as well as Google's Geocoding API. As long as the available APIs remain functional and Twitter does not shut down, the algorithms should work. Additionally, availability of the system is also dependent on the status of Amazon EC2 on which it will be operating.

3. Specific requirements

3.1 External Interface Requirements

3.1.1 User Interface

- The primary feature of the user interface will be a large zoomable map
- "Pins" will indicate the current location of food trucks, and clicking/tapping them will provide more information, including the name of the food truck and the tweet(s) from which its location was inferred, as well as the time frame it will be there (if available).
- The user's current location will be indicated by a pin or dot of different color or identifying text
- A search field will allow the user to search for a particular food truck or a location.

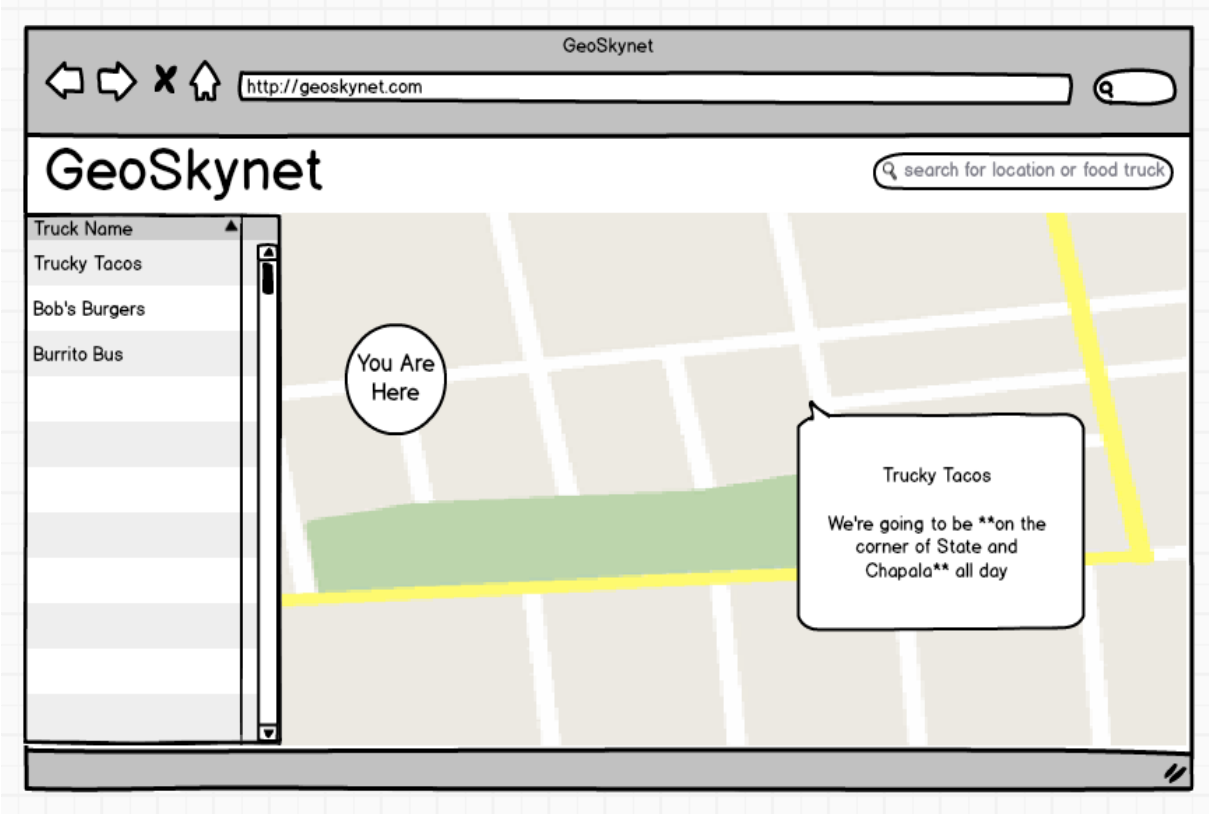

Figure 2: Frontend user interface mockup

3.1.2 Hardware Interface

- The backend will be agnostic to the exact nature of the hardware it is running on, so long as the hardware provides at least 8 GB of RAM, 500 GB of disk space and a multi-core 2.0 GHz or faster x86-64 processor.

3.1.3 Software Interface

- The backend will store the text of processed tweets, the range of the tweet believed to identify a location, a measure of the confidence of the location's accuracy, and the actual location in latitude and longitude coordinates in a MongoDB (version 2.2) database. The frontend will read this data when displaying the map to the user.
- One or more sensors will maintain a continuous HTTP request to Twitter's Streaming API endpoint and collect data as it arrives. It will tag this data and dispatch it to a RabbitMQ (version 3.0) queue that distributes it to various sensor processors depending on the tagged metadata.
- A series of sensor processors will receive data from a RabbitMQ queue, process it, potentially with the Natural Language Toolkit framework (to identify locations, degrees of confidence, map textual locations onto coordinates, etc) and optionally place their processed output back into the queue with additional or different metadata to cause it to be routed to another sensor processor for further processing.
- An Apache (version 2.4) web server will respond to requests from users' web browsers via HTTP GET and POST and pass requests to one or more sensor processors

responsible for looking up requested information in the database, formatting it and returning a response.
- The frontend will consist of Javascript running in the user's browser, which will issue requests to the Apache web server for information and interface with the Google Maps and Geocoding APIs to identify and plot the extracted locations on a map, as well as responding to user input such as search queries.

3.1.4 Communications Interface

- Requests to third party APIs will be made using HTTP GET and POST requests
- Communication within the backend will be handled by RabbitMQ routing and work queues.
- Requests to the backend from the frontend will be made via HTTP GET and POST requests.

3.2 Functional Requirements

3.2.1 Load Tweets

Precondition: Valid Twitter API credentials exist and the system is logged in.

1. The system will send an HTTP GET request to Twitter's streaming API
2. The system will continuously and indefinitely listen for responses
3. When a response is received, the tweets it contains will be passed to the RabbitMQ dispatch queue

Exceptional conditions: In the event of a lost connection, the system will attempt to reconnect. In the event of an API status error or repeated connection failures, the system will back off exponentially and notify an administrator by email.

3.2.2 Extract Locations

Precondition: An unprocessed tweet is available

1. The system will check for the presence of geolocation metadata associated with the tweet, and use it if found
2. If geolocation metadata is not found, the system will analyze the tweet using the Natural Language Toolkit and run a series of heuristics on the result to attempt to identify location strings
3. If a location is found, it is stored with the tweet in the database and the tweet is sent to another sensor processor to convert the extracted location string to coordinates

Exceptional conditions: if no location is found, the tweet is discarded

3.2.3 Convert Location String to Coordinates

Precondition: A location string has been identified, API credentials for Google Geocoding API exist

1. The system will send an HTTP request to the Google Geocoding API with the location string and API credentials
2. The system will wait to receive a response

3. Assuming the response indicates success, the coordinates will be associated with the original tweet in the database

Exceptional conditions: If the request fails due to rate limiting, the system will notify an administrator via email and not issue further requests for 24 hours. If the response indicates that coordinates could not be established, the system will log the failure and carry on.

3.2.4 Find Food Trucks Near Location

Preconditions: A location is specified in latitude and longitude coordinates, and a radius of interest is specified

1. The database will be queried for tweets whose latitude and longitude each lie within the interest radius of the specified location
2. The distance of each returned result will be calculated to ensure that both the latitude and longitude are not close to the maximum possible values, and the results will be filtered accordingly (eg only those locations within a circle with the specified radius of the point of interest, as opposed to within a square with dimensions equal to the diameter)
3. The exact coordinates, original tweets, and Twitter user ID of the author will be returned in JSON format.

Exceptional conditions: If no matching tweets are found or the input parameters are invalid, an empty response will be returned

3.3 Performance requirements

The system should be capable of actively monitoring and analyzing the tweets of at least 1,000 Twitter users for location data. Once operational, the system should maintain 99% uptime, except in cases of Twitter or Google API outages or other circumstances beyond its direct control. The user-facing website should be able to handle at least 1,000 concurrent users.

3.4 Safety and Security Requirements

The data stored is available to anyone that has access to the internet through the front-end interface. However, the database and other components must be protected from malicious attacks. Firewalls and code that handle user requests should prevent attempts to shutdown the system or fill the database with inaccurate content.

Appendix A: Group Log

January 18th, 2013: Aerospace folks created document skeleton
January 30th, 2013: BRX added additional sections from Capstone template
January 30th, 2013: BRX met with Aerospace to come to agreement on individual sections
February 1st, 2013: BRX made additional modifications as per meeting with Aerospace
February 8th, 2013: Got the sign-off from Aerospace for SRS completion