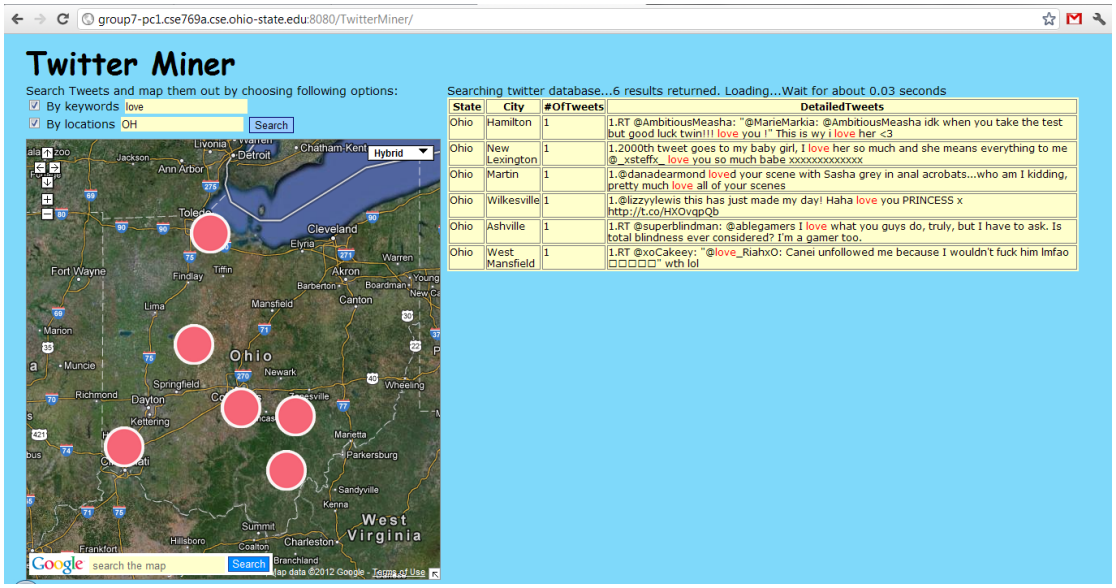
Group 7 TwitterMiner Project
User and Developer Guide

CSE 769 SP2012

Instructor: Dr. Rajiv Ramnath

Team members:

Wei Chen, Yu Qiao, Igor Tolkachev



Acknowledgement

This research project would not have been possible without the support of many people. The authors wish to express their deepest gratitude to course instructor, Dr. Rajiv Ramnath for his invaluable mentoring, guidance and help over our course of finishing the project. Thanks also go to our T.A. Andrew Yates for his constructive feedbacks and suggestions at every step of our work and Shaun Rowland for preparing a server for hosting and testing our project.

CONTENT

Introduction

TwitterMiner is a baby version JavaEE-based Web application that is developed to enhance the Twitter search capabilities provided by existing Twitter Search API. This application is developed partly to fulfill the course requirement of CSE 769 (Applied Enterprise Distributed Computing for Engineers and Scientists).

In general, there are two ways to search Tweets through the Twitter interface provided by Twitter.com. The first method is to use the search toolbar on Twitter's homepage, which is similar to the one provided by a search engine, to search Tweets by either hashtag or keywords. The second method is to use the newly released #Discover tool to find certain Twitter topics. Both search utilities are limited in the sense that they only provide interfaces to search by keyword, rather than other constrains such as location and time. We believe an option that allows location-based search of Tweets is important to enhance the overall user experience of interacting with Twitter search utility.

In our TwitterMiner application, we developed a simple enterprise Java Web application to allow the exploration of tweets on both topical and spatial basis. By interacting with a simple TwitterMiner interface, one could retrieved a list of most recent Tweets by specifying either keywords or locations, or both and then visualize the retrieved Tweets on Google Maps. Twitter data one can search are automatically pulled from Twitter.com through an ad hoc Java program we developed based on Twitter Search REST API. Tweets are retrieved according to predefined spatial sampling locations across the entire states and are stored in H2 database. Only the latest 10,000 tweets are saved in the database and our Java application repeatedly fetch a new set of Tweets about every 20 minutes.

The front page is a screen shot of the main interface of our application. This reports is comprised of two sections: a user guide and a developer guide. Code is attached as Appendix I. Below is a list of the sites that are related to this project:
TwitterMiner project URL:
http://group7-pc1.cse769a.cse.ohio-state.edu:8080/TwitterMiner/
Project site: https://sites.google.com/site/sp12cse769
Google code site: https://code.google.com/p/twitter-java-cse769sp-project/
Google group site: http://groups.google.com/group/cse-769-project
CSE769SP12 class site: https://sites.google.com/site/cse76920122spring

User Guide

System components

We propose a framework including three major system components: Java Web application component, Twitter database component and Google Maps. These three components are proposed based on our knowledge of the way distributed data is shared by different parts of the enterprise system.

Java Web application component

Java Web application component provides a Web interface for end users to issue a search to Twitter database by either keywords or location or both.

Twitter database component

Twitter database component of our system provides the interface for both retrieving data from Twitter via Twitter search API and providing search return to the user. The ad hoc Twitter database is different from the one provided by Twitter in the sense that the one we create is a small, but randomly sampled, subset of the entire gigantic Twitter database provided by Twitter.com. We distill location information from the Tweets by intentionally searching Tweets based on a (lat,lon,radius) search constrain defined by Twitter Search API.

Google Maps component

A mapping component is another essential component of our application. We here choose Google Maps to provide mapping functions mainly because it is the most popular online mapping system out there today and most online users are familiar with it. Also, the Google Maps Javascript API provides some essential and easy-to-code functions to mashup user data with Google Maps.

Use cases

Our system includes two major use cases: tweet search and database update respectively.

ID:	UC-1
Title:	tweet search
Description:	The tweet search use case allows a user to specify a search constrain through a Web interface and retrieve Tweets that satisfy keywords and/or location constrains. Extracted Tweets will be used to populate a table and also mapped out on Google Maps.
Primary Actor:	online user
Preconditions:	User's browser supports JavaScript
Postconditions:	Result table and a map is represented to the user.

Main Success Scenario:	1. Result table is populated with Tweets. 2. Map is generated with graduated symbols on it representing the number of Tweets.
Extensions:	The input area provides example search constrains for both keyword and location. The system will show an error message if the input area is empty when the user press the search button.
Frequency of Use:	Always used by users of the sytem
Status:	Pending Review
Owner:	Wei, Yu, Igor
Priority:	P1 - High
ID:	UC-2
Title:	database update
Description:	An administrator is in control of the database update process. This is done by running a standalone Java application which automatically retrieves Tweets from Twitter.com via Twitter Search API. An administrator has the privilege to run or terminate this program whenever necessary. By default, this program is always running.
Primary Actor:	administrator
Preconditions:	Connection to Twitter.com is active.
Postconditions:	New tweets are populated into our H2 database.
Main Success Scenario:	More recent tweets are added to the database.
Extensions:	Locations with no tweets retrieved from will be specially handled.
Frequency of Use:	Mostly run automatically with minimum administration.
Status:	Pending Review
Owner:	Wei, Yu, Igor
Priority:	P2 - Medium

Screen flow

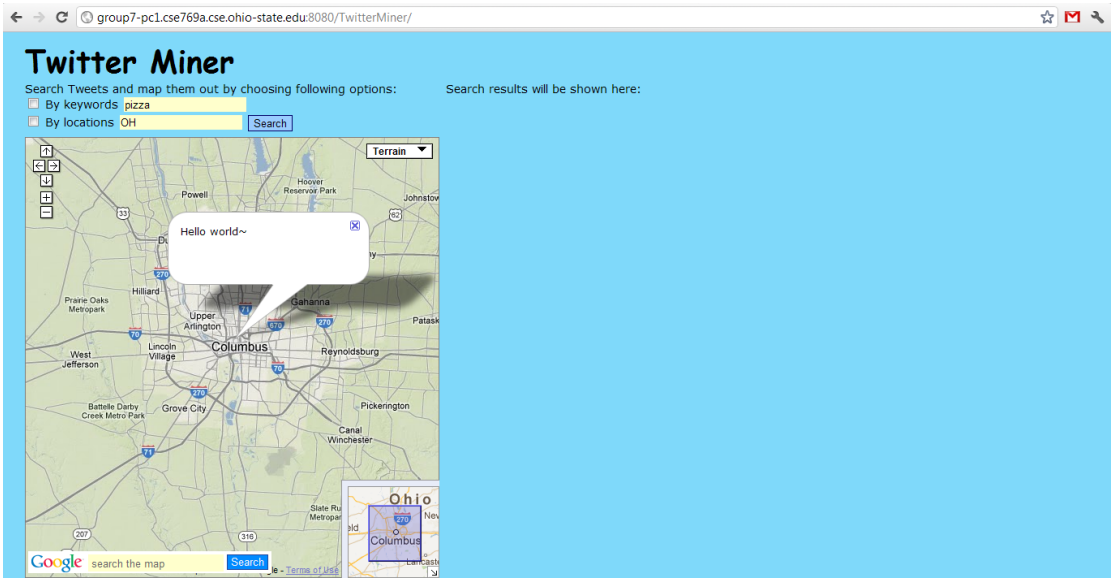


Figure 1 Initial screen

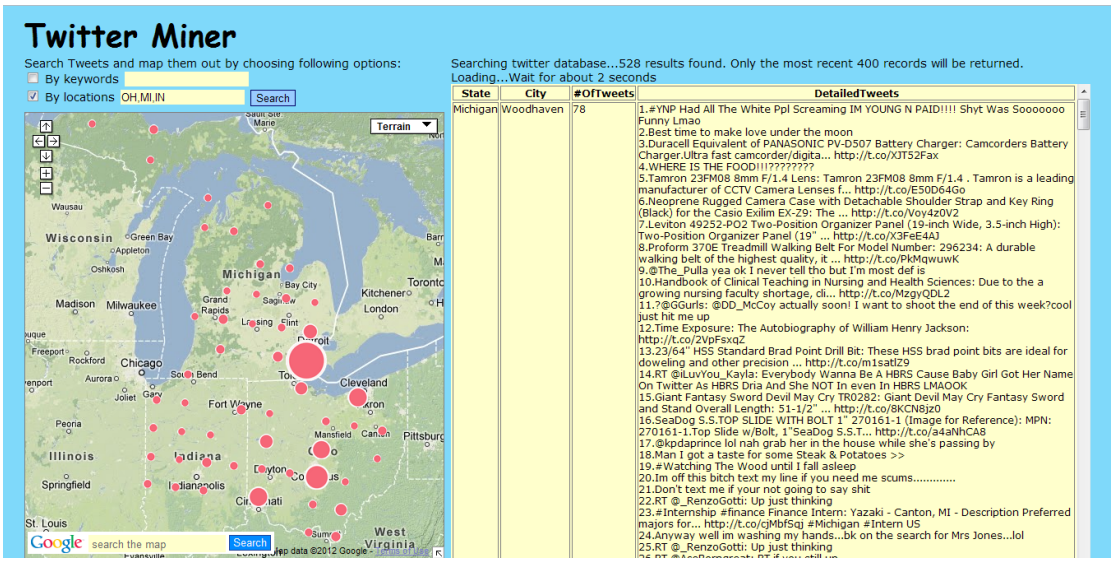


Figure 2 Search tweets by locations

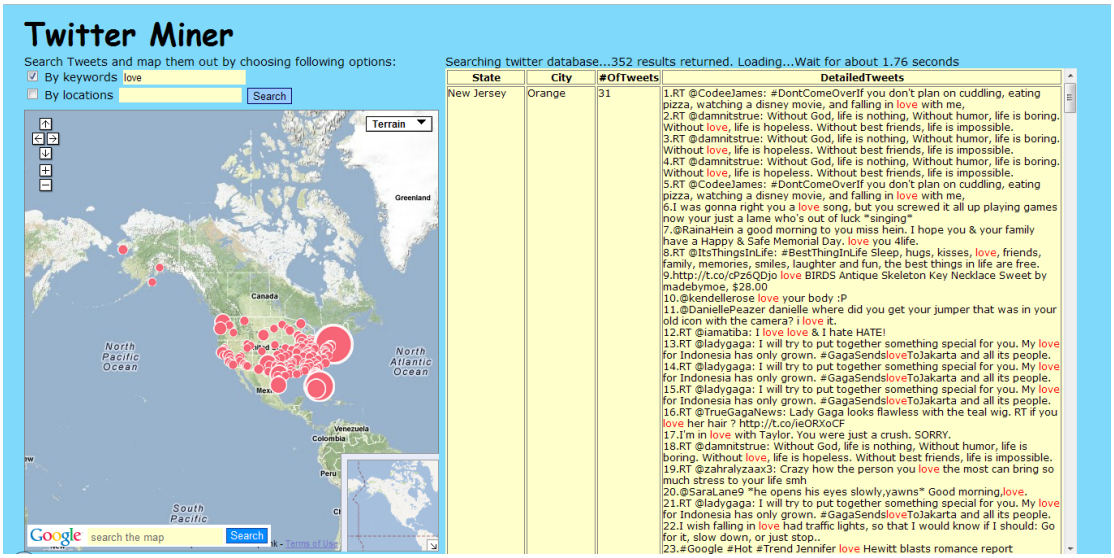


Figure 3 Search tweets by keywords

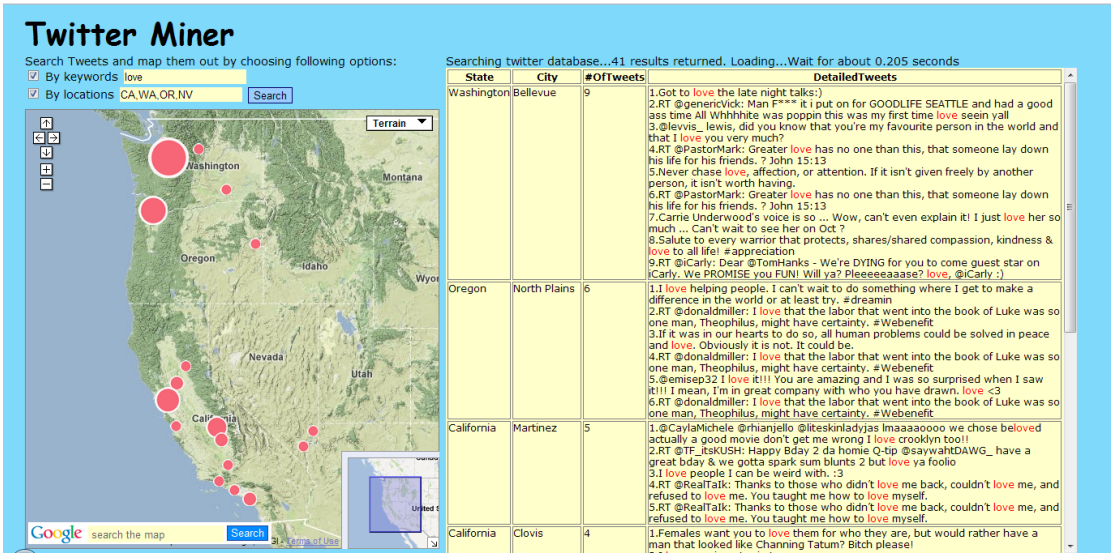


Figure 4 Search tweets by both keywords and locations

Developer Guide

System Design

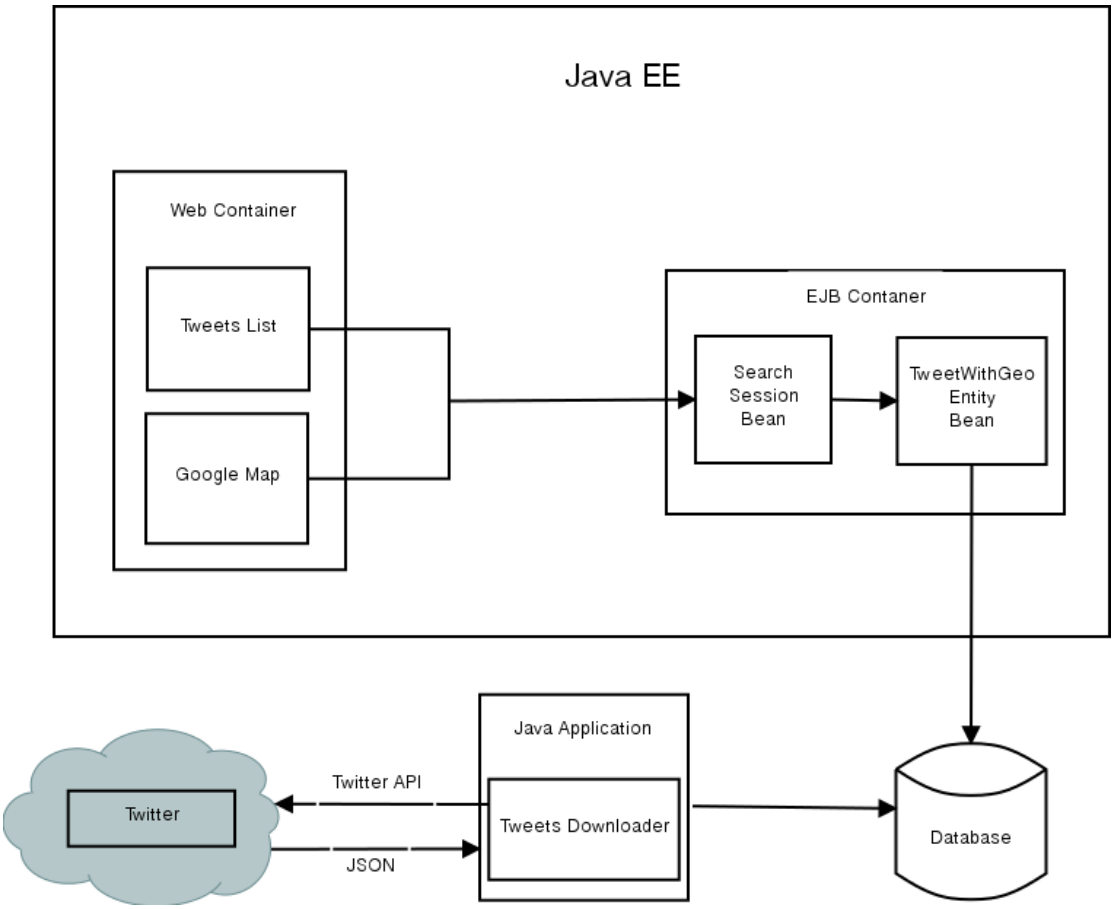


Figure 5 Overview of the system

As discussed previously, our application includes two main components: Java EE component that allows one to retrieve tweets from the H2 database by keywords or locations and Java application component used for downloading data from Twitter and populating the H2 database. These two components are loosely dependent.

Further, the Java EE component consists of two primary containers: web container and Enterprise Java Beans (or EJB) container. The web container is built upon HTML, Java Servlet and AJAX. The EBJ container includes entity beans and session beans to proviede functions for the frontend to interact with the database.

Domain classes

Below is a class diagram of all Java classes created for our application. The diagram is created using ObjectAid UML Explorer, a UML modeling plugin in Eclipse. The class diagrams of the ObjectAid UML Explorer are based on the OMG's UML 2.0

specification (see http://www.omg.org/uml/). They can contain existing Java classes, interfaces, enumerations, annotations (collectively called classifiers henceforth in accordance with UML 2.0) as well as packages and package roots (i.e. JARs and source folders).

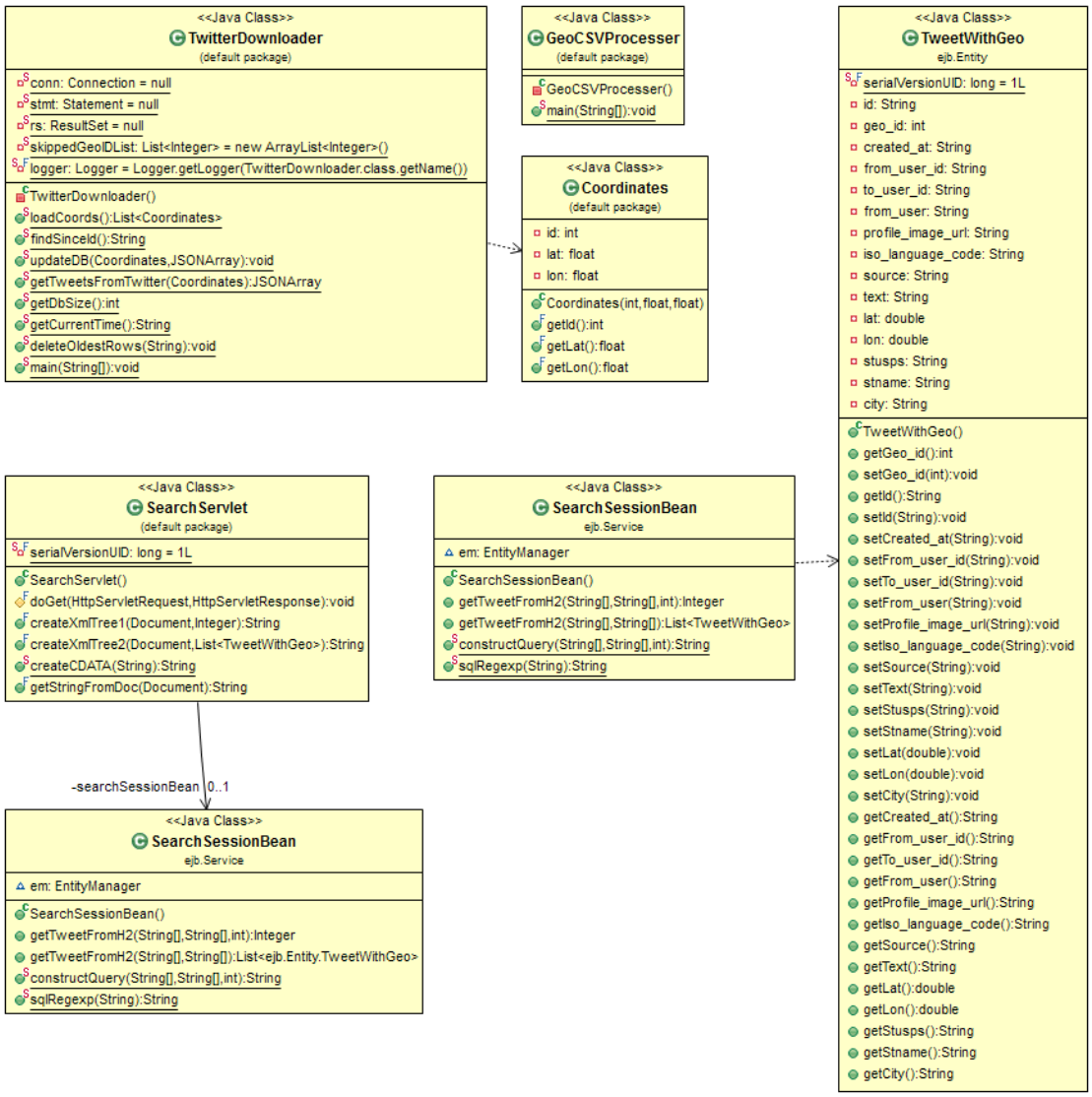


Figure 6 Domain classes diagram

Database schema

Before we discuss the two component detailed, it necessary to introduce the database schema of our application. There are two tables and one view in the database: “tweet”, “geo” and “tweet_geo_view”. “tweet” is populated by “Tweets downloader” component which will be discussed later. “geo” is a fixed table. It is populated by importing a cvs file.

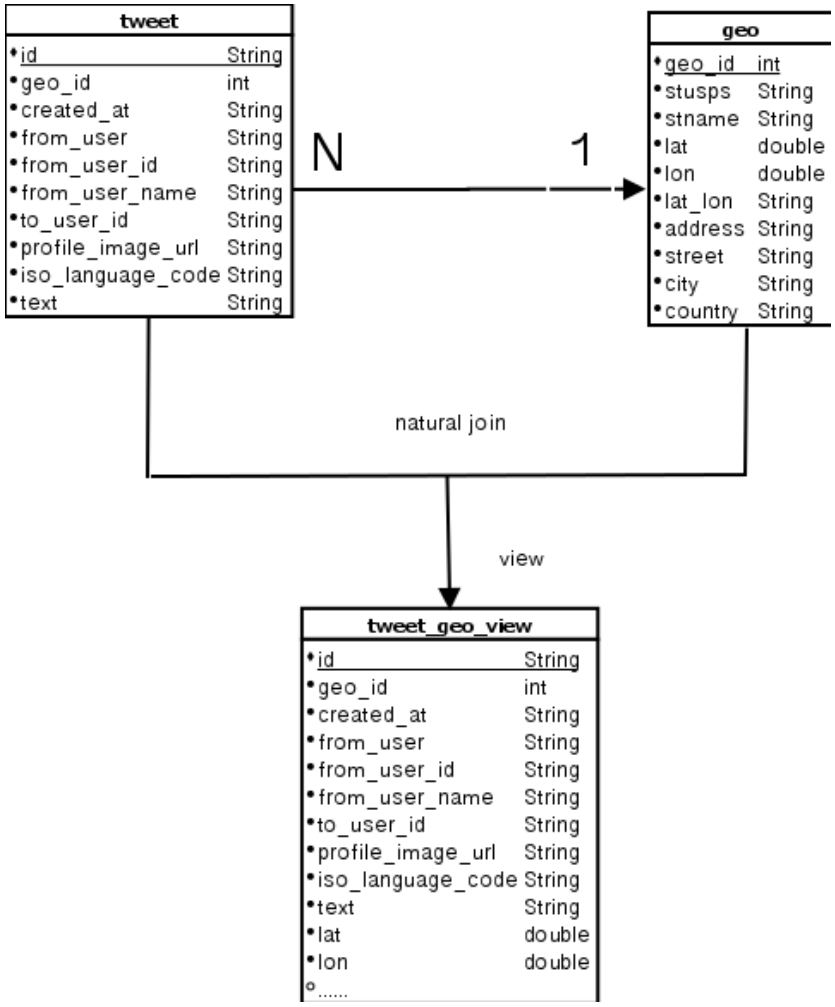


Figure 7 Database schema

Every row in "tweet" represents a tweet got from twitter. It has ten attributes fully describing a tweet:

- id – The unique id of the tweet. It is generated by twitter.
- geo_id – Foreign key point to the primary key of a row in "geo".
- created_at – The time when the tweet is created.
- from_user – The username who created the tweet.
- from_user_id – The user id of the user who created the tweet.
- from_user_name – The name of the user who created the tweet.
- to_user_id – The id of the user whom the tweet is sent to.
- profile_image_url – The URL of the profile image.
- iso_language_code – ISO language code of the tweet.
- text – The content of the tweet.

There are 2503 rows in "geo" and every row in "geo" represents an area in USA. The 2503 areas cover most of the area of USA. The table has fourteen columns to describe an area:

- geo_id – The unique id of the area.

- stusps – The abbreviation of the state which the area belongs to.
- stname – The name of the state which the area belongs to.
- lat – The latitude value of the area's center.
- lon – The longitude value of the area's center.
- lat_lon – The coordinate value composited by lat and lon.
- address – The address of the area's center.
- street – The street of the area's center.
- city – The city name of the area's center.
- country – The country which the area belongs to. In this database, it is always USA.

"tweet" has a "many-to-one" relationship with "geo". In the other word, several tweets could come from the same location. The two tables are associated with "geo_id".

"tweet_geo_view" is a view. It is the result of using natural join to "tweet" and "geo". It is created for retrieving data. Because this view contains all the information, we only need one entity bean mapping it.

Enterprise Java Beans

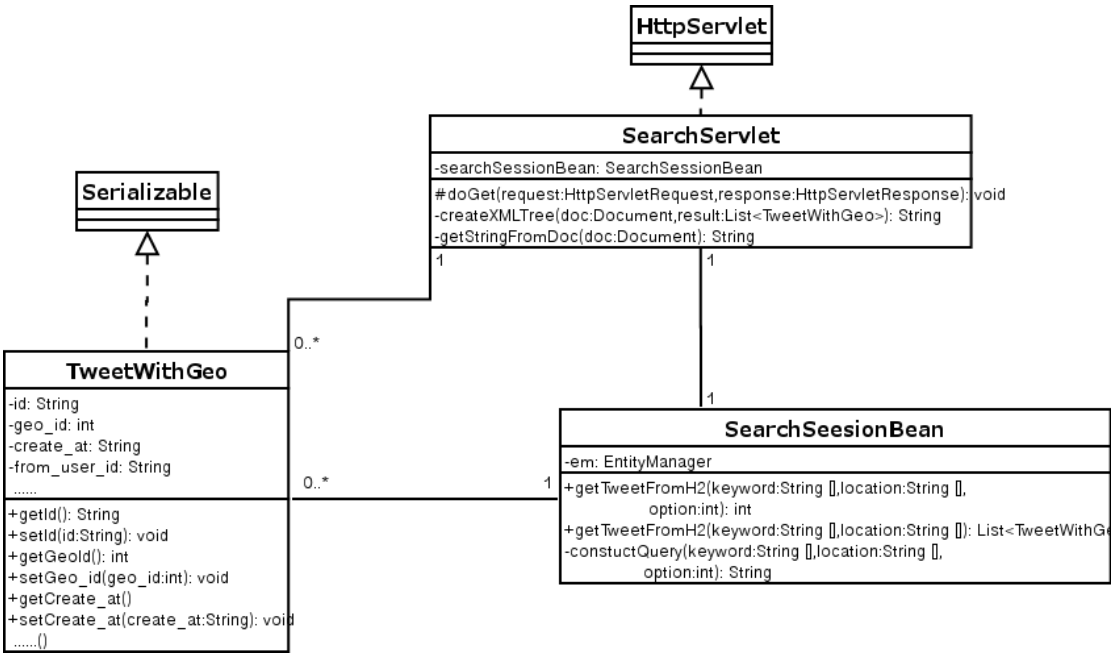


Figure 8 EJB classes diagram

The class diagram of EJB is shown in the figure above. There are one entity bean and one session bean. These beans are called in servlet.

Entity bean

"TweetWithGeo" is an entity bean mapping to the view "tweet_geo_view" in the

database. It represents a tweet with its geo information.

Session bean

"SearchSessionBean" is a session bean used for searching tweets in the database. It is a stateless bean because it doesn't have to track the state of the client. This bean has two overloaded public methods: getTweetFromH2(). Both the two methods use keyword or location or both to search records in "tweet_geo_view" in the database by using query. This first one with one more argument returns the number of records. The other method returns a list of "TweetWithGeo" objects.

Sequence diagram

The following sequence diagram shows the interaction between servlet and EBJ components.

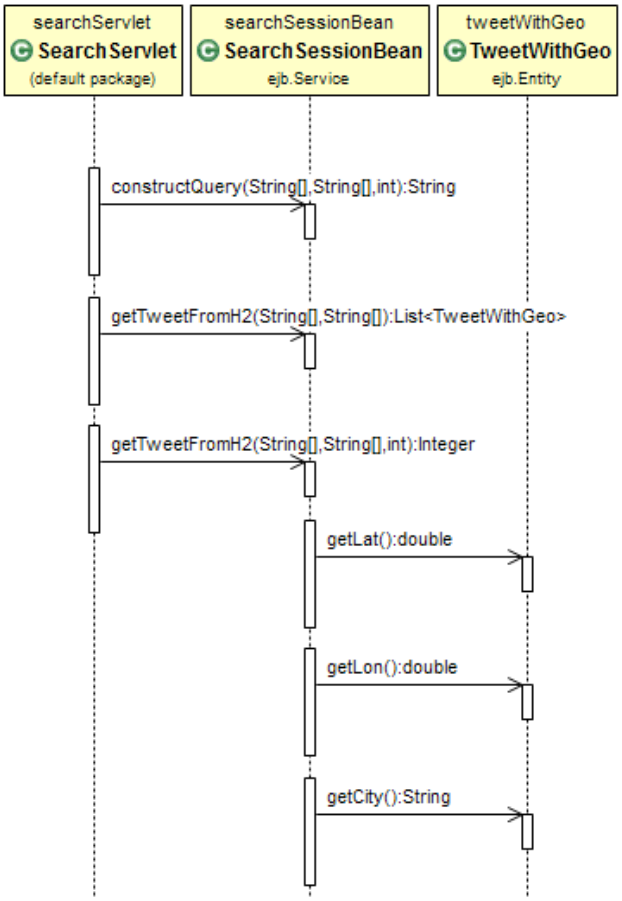


Figure 9 The sequence diagram of the servlet and EJB components.

Web Client

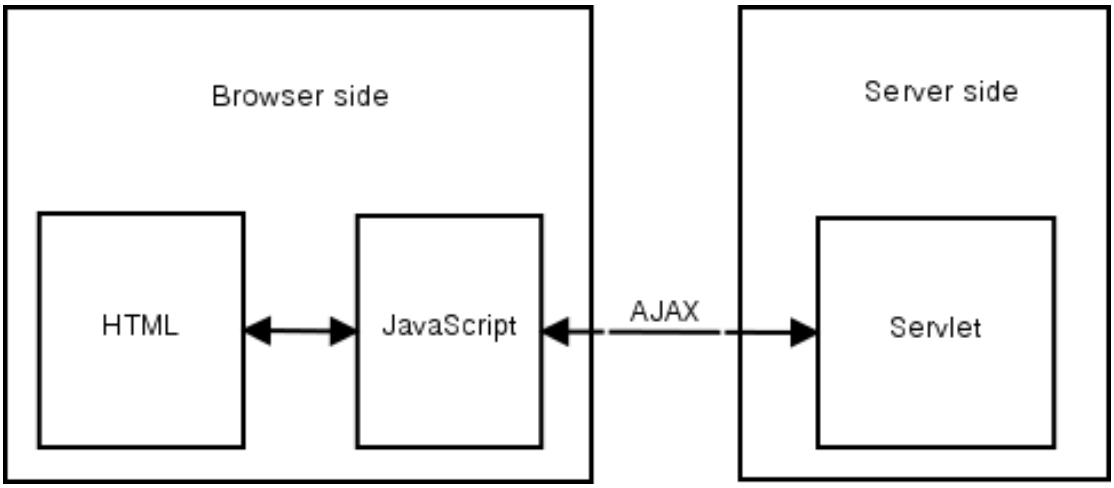


Figure 10 Web client architecture

Figure 9 shows the architecture of web client. At the browser side, we use html and JavaScript to implement the user interface. Our application only has one web page. On the web page, there are three areas: input area, results table area and map area, as shown in figure 10.

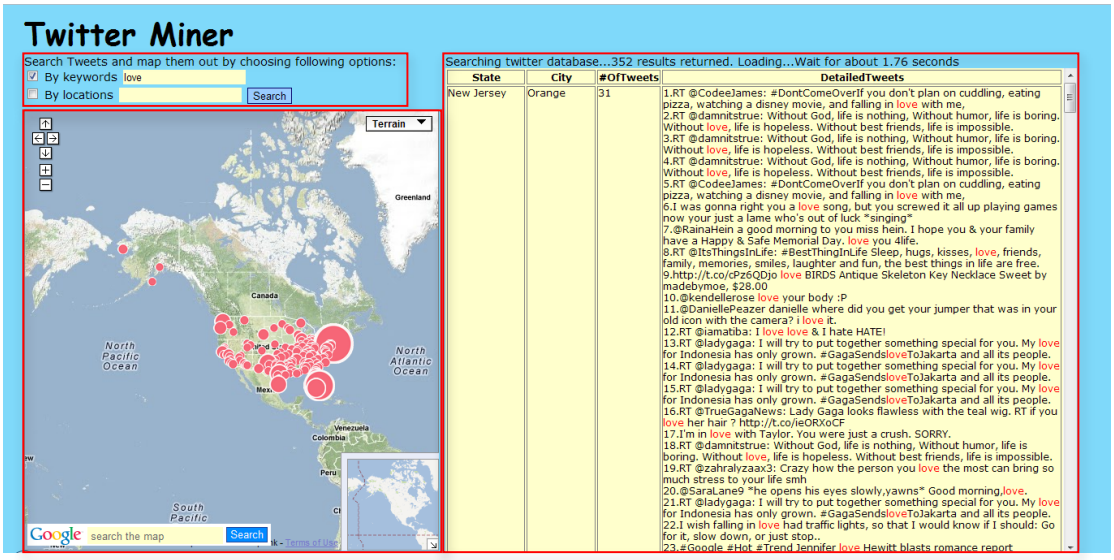


Figure 11 Three areas of the web page

After user inputs keywords or locations or both and check the checkboxes, user clicks search button. This event will call JavaScript function. Then this function will call servlet as asynchronous way. The servlet will call methods of session bean and get the number of the tweets and a list of tweets. The servlet then constructs the number and the tweets into two XML and pass them to JavaScript. Because getting the list of tweets and constructing may take long time, the servlet pass the XML containing the number of tweets first.

After the servlet returns XML, JavaScript will call a callback function. This function will parse the XML. If the XML contains the number of tweets, it will show it on the web page and calculate the approximate time it will cost to get the table of result. If the XML contains the tweets with their geo information, the function will put the tweets into table and place the geo points on the map.

We use JavaScript API provided by Google to implement the map. To show the locations where the tweets come from, we place red circles on the locations. The radium of the circle indicates the number of tweets come from this location. The location that most tweets come from has a fixed size of circle placed on it. Other circles are scaled based on the ratio of number of tweet against the max number.
The results of tweets are shown in a table. The table has four columns: state, city, number of tweets and detail of tweets. When user clicks any row in the table, the circle at the location that the tweets in the row come from will be focused. Also, if user uses keywords to search tweets, the keywords in the result table will be shown in red color.

XML schemas and/or DTDs

In our application, XML is used to exchange data between servlet and JavaScript. XML could be validated by schema or DTD. See Appendix to check the code.

Non-functional requirements

Availability

As mentioned before, our application has a separate component that is used for populating the database. This component is running 24*7 hours to update the tweets pulling from Twitter. Theoretically, our application works 24*7 hours. And the tweets in the database are the most recent data.

Portability

Our application is based on Java whose greatest advantage is the good portability. Our application could work on any platform supporting Java Virtual Machine.

Performance

The performance of our application mainly depends on the performance of the database query and parsing the XML file returned by servlet. The more tweets are retrieved, the more time it will take. To make sure it won't take much time, we only show the first 400 tweets retrieved. The detail about the performance analysis is discussed in section "Performance Analysis".

Usability

Our application is very easy to use. There is one area for user to input keyword and specify and option (whether search by keyword or location or both). The result is shown in a table and the locations are shown in a map. The visualization of results is designed for the usability.

Reusability

All components of our application have certain reusability. For example, the component for populating database could be used for other application that also needs use tweets in the database.

Constrains and limits

Our application implements several use cases and meets several non-functional requirements. Though, it is a baby version and has some constrains and limits.

- The number of results can not be over 1000. This is because of the limits of Dom String. If the number of results is over 1000 and we do not limit it, when we construct it into XML, there will be an exception: "too long to fit into Dom String".
- If user inputs multiple keywords, we only use "OR" logic to make the query. This means if user uses "love, happy" to search tweets, the result will be tweets containing either "love" or "happy". Not all the users want this kind of result. Some may want get the tweets containing both "love" and "happy".

How to build and run the software

Install and configure Java and Eclipse

- Get the latest version JavaSE (currently Java Platform (JDK) 7u4) from http://www.oracle.com/technetwork/java/javase/downloads/index.html
- DO NOT choose the version with Java EE (such as JDK 7u3 + Java EE) because we will install JBoss later separately to provide JavaEE facility. The default Glassfish server bundled with JDK 7u3 + Java EE install may potentially conflict with JBoss server and cause configuration problems.
- For windows server, set environmental variables JAVA_HOME and classpath for JDK install.

Install and configure JBoss Java enterprise server

(shamelessly borrowed from Rajiv's original instruction)

- Install JBOSS as follows:

    - Download the latest Final release of JBoss Application Server (currently 7.1.1) from: http://www.jboss.org/jbossas/downloads
    - Unzip JBoss to the location where you would like it to be installed. No further installation is required.
        - Hint: Best to pick a directory path name with no spaces
    - Locate the deployments directory within the JBOSS installation (hint: it is a sub-directory under standalone). That is where your code will end up being deployed.
    - Locate the server configuration file standalone.xml.
    - Look inside this file to find the datasource of the embedded H2 database (jdbc:h2:mem:test). Bring up the H2 console (that you installed in Part 3) and inspect this data source.
- Install JBOSS Application Server (AS) 7 components into Eclipse as follows.
    - Start Eclipse, then Help > Install New Software... >
    - Paste this URL: http://download.jboss.org/jbosstools/updates/development/indigo/ Hit Enter.
    - When the site loads, click Select All.
    - To properly resolve all dependencies, check [x] Contact all update sites during install to find required software
    - Click Next, agree to the license terms, and install.
    - To test the installation, do the following:
    - In Eclipse, navigate to the Servers tab. If the Servers tab does not show, select it from Windows->Show View.
    - Create a new AS 7 server by right-clicking on the Servers area, and selecting New->Server
    - Start the server by right-clicking on the server and selecting Start
    - After the server starts, go to http://localhost:8080/ in your browser and see if it shows the Welcome page of the server.

Import, build, deploy and run the application

- To get a copy of our project source code and import them into Eclipse, please use SVN Eclipse plugin.

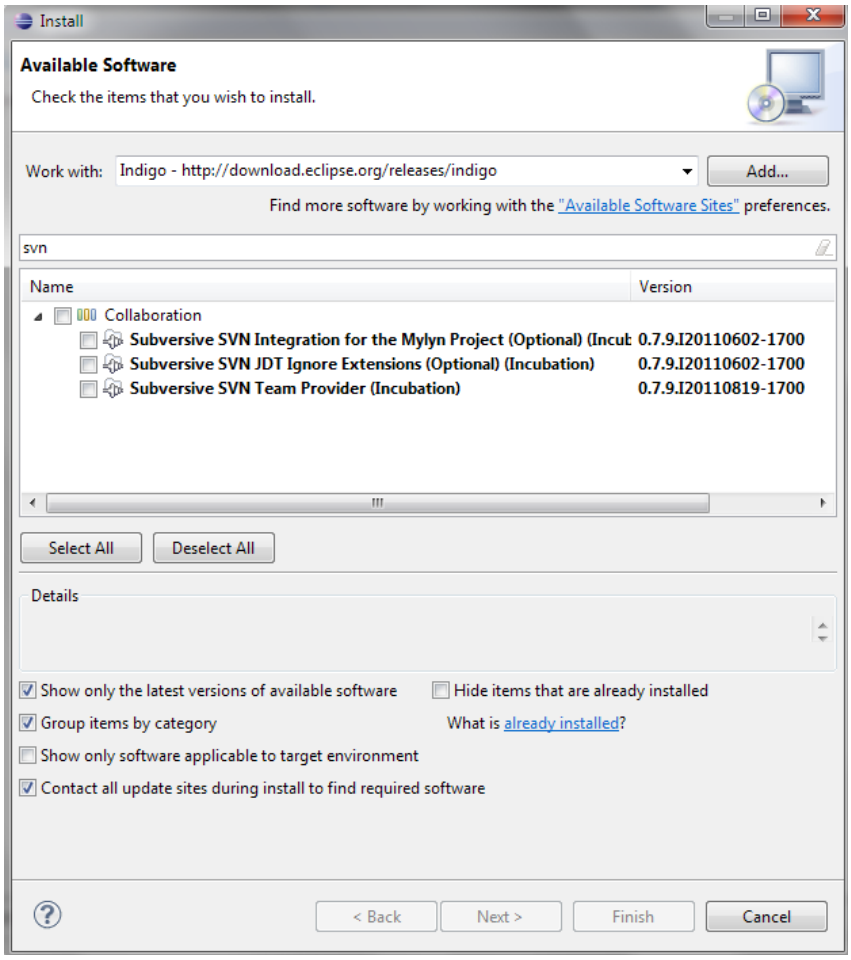


Figure 12 SVN install

- We host our project on code.google.com and the checkout page of our source code is https://code.google.com/p/twitter-java-cse769sp-project/source/checkout.
- To check out the source code from our project's SVN repository, in Eclipse go to->view->SVN repository. Right click->New->Repository location and enter the following information.

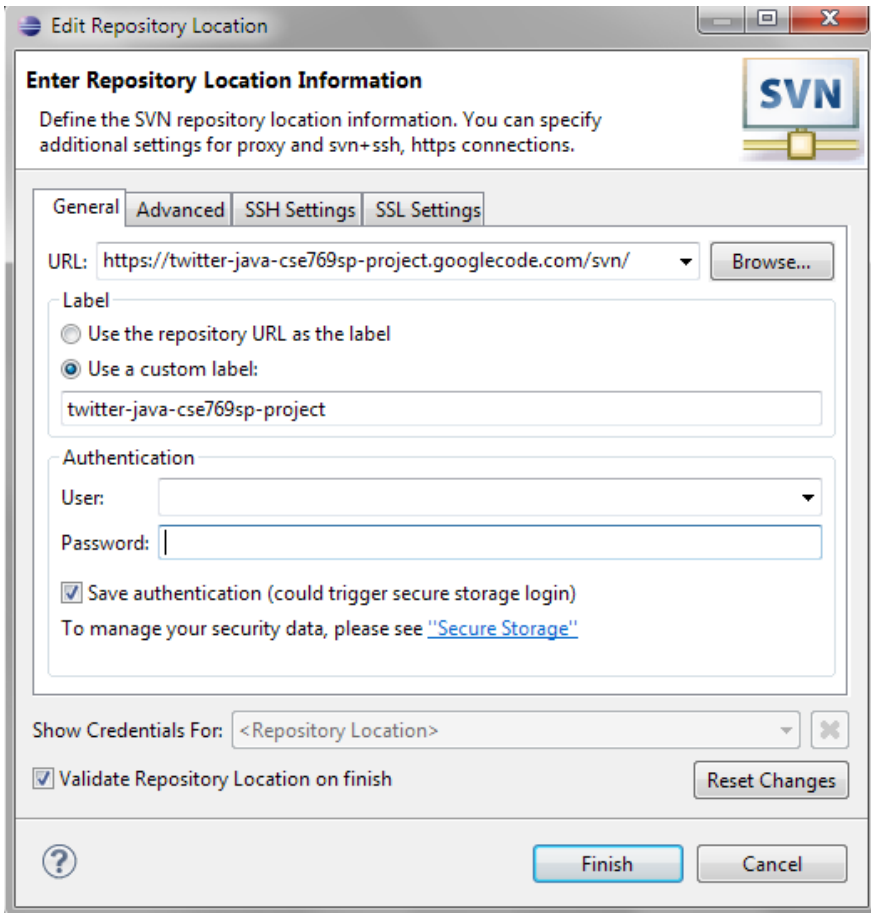


Figure 13 Connect to Google Code repository

Test and performance

The following performance analysis studies the variation in xml retrieval performance by different number of tweets counted and retrieved. As we can see, in Figure 14 and 15 the performance of xml creation and return doesn't scale with the increase of the number of tweets counted or returned. This is because we set up a maximum number of 400 as the cap if there are more than 400 tweets found. Also, the way we use DOM based xml generation rather than simple printing may be accountable for this performance improvement.

Table 1 Performance analysis results

ID	# of tweets counted	xml retrieval time	# of tweets retrieved	detailed xml retrieval time (count>400:400,count)
1	258	0.2494	258	0.9048
2	1901	0.2851	400	1.2428
3	2065	0.1643	400	1.0235
4	302	0.1665	302	0.9056
5	136	0.2912	136	1.8379
6	2177	0.2695	400	0.9990
7	3345	0.1879	400	0.9714
8	1080	0.2673	329	1.0738
9	1983	0.2247	400	1.1332
10	1184	0.1654	351	0.9646
11	219	0.2288	219	1.3718
12	1157	0.2804	268	1.4184
13	2761	0.2287	400	0.9852
14	2212	0.2276	365	1.0226
15	1531	0.2460	365	1.1035
16	1583	0.1950	376	1.0489
17	701	0.1971	285	1.1682
18	688	0.2546	244	1.3951
19	1959	0.2545	334	1.2018
20	2487	0.2282	382	1.0039
21	1872	0.2368	365	1.0631
22	1557	0.2205	370	1.0762
23	1142	0.1961	330	1.1085
24	695	0.2259	264	1.2816
25	1323	0.2546	289	1.2985
26	2223	0.2413	358	1.1029
27	2179	0.2325	373	1.0335
28	1715	0.2286	367	1.0696
29	1594	0.2126	334	1.0860
30	1280	0.2457	304	1.2291

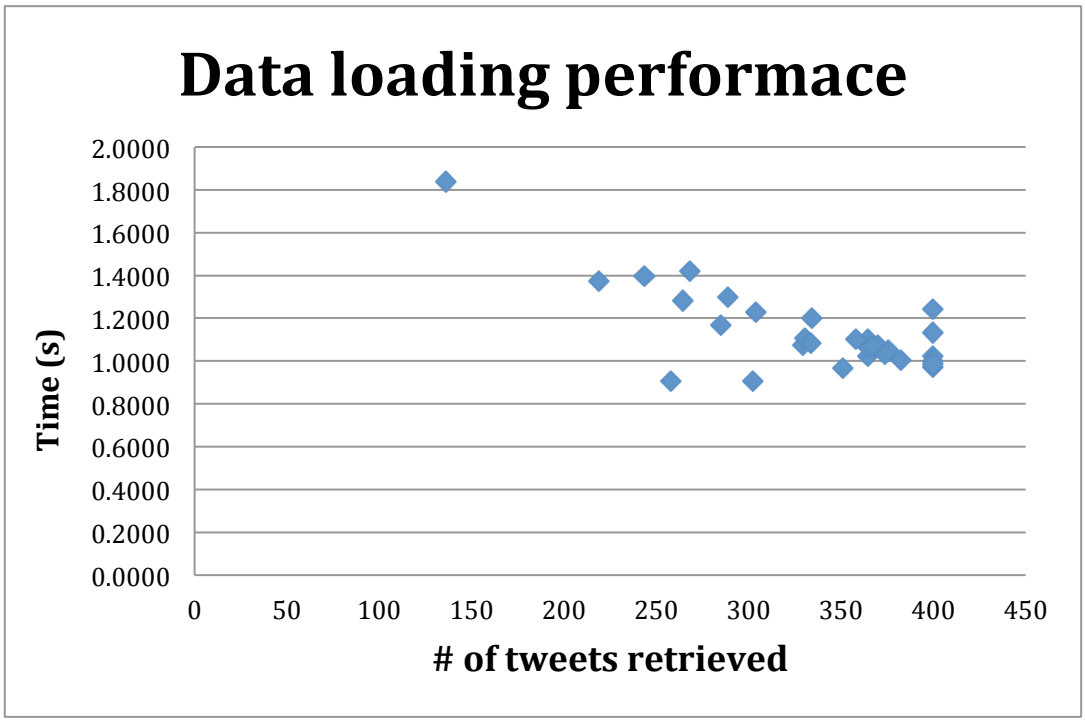


Figure 14 XML performace by the number of tweets retrieved.

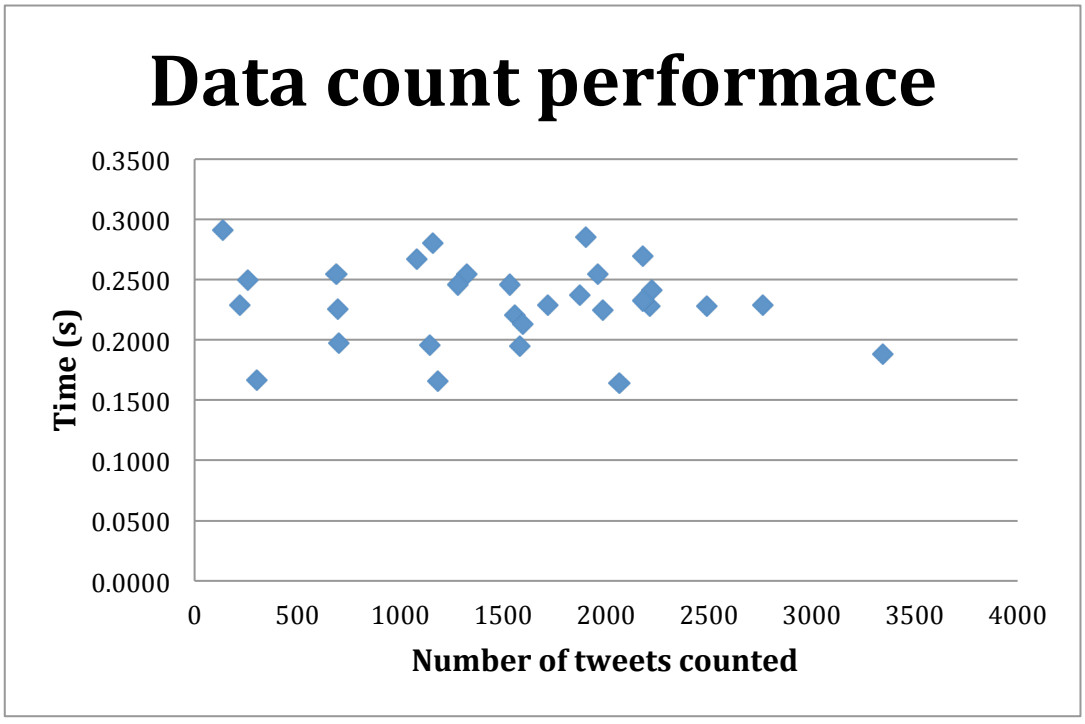


Figure 15 XML performance by the number of tweets counted.

Summary

The TwitterMiner we develop is a baby version enterprise Java Web application to allow users to explore tweets on a keyword and locational basis. By interacting with a Web interface, one could retrieve a list of Tweets that conform to the keywords and locations constrains. Also, a map is provided along with a result table to allow visualizing the Tweets on Google Maps.

Twitter data are automatically pulled from Twitter.com through an ad hoc Java program we developed based on Twitter Search REST API. Tweets are retrieved according to predefined sampling locations across the entire US and are stored in H2 database. Only the most recent 10,000 tweets are persisted in the H2 database and our Java application repeatedly fetch a new set of Tweets about every 20 minutes.

An initial evaluation of the application shows that our implementation permits efficient retrieval of query results based on Ajax calls. The average response time of interaction is less than one second.

Appendix I Source Code

DTD and Schema

XML DTD

```
<!ELEMENT root (state*)>
<!ELEMENT state (locaiton+)>
<!ATTLIST state
    name CDATA #REQUIRED>
<!ELEMENT location (tweet+)>
<!ATTLIST location
    city CDATA #REQUIRED
    geoId CDATA #REQUIRED
    lat CDATA #REQUIRED
    lon CDATA #REQUIRED>
<!ELEMENT tweet (#CDATA)>
```

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">



    <xs:element name="root">
        <xs:complexType>
            <xs:sequence>
                <xs:element      name="state"      maxOccurs="unbounded"
minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element                    name="location"
maxOccurs="unbounded" minOccurs="1">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element        name="tweet"
type="xs:string" maxOccurs="unbounded" minOccurs="1" />
                                        </xs:sequence>
                                        <xs:attribute          name="city"
type="xs:String" use="required" />
                                        <xs:attribute          name="geoId"
type="xs:string" use="required" />
                                        <xs:attribute            name="lat"
type="xs:string" use="required" />
                                        <xs:attribute            name="lon"
type="xs:string" use="required" />
                                    </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute     name="id"     type="xs:string"
use="required" />
                        <xs:attribute name="created_at" type="xs:string"
use="required" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required" />
        </xs:complexType>
    </xs:element>

</xs:schema>
```

EJB

Entity bean

```
package ejb.Entity;

import java.io.Serializable;
import javax.persistence.*;

/**
 * TweetWithGeo EJB entity class.
 * @author Yu Qiao, Wei Chen, Igor.
 *
 */
@Entity
@Table(name="TWEET_GEO_VIEW")
public class TweetWithGeo implements Serializable{
    private static final long serialVersionUID = 1L;
    private String id;
    private int geo_id;
    private String created_at;
    private String from_user_id;
    private String to_user_id;
    private String from_user;
    private String profile_image_url;
    private String iso_language_code;
    private String source;
    private String text;

    private double lat;
    private double lon;
    private String stusps ;
    private String stname;
    private String city;

    /**
     * Get ID, the primary key.
     * @return id.
     */
    @Id
    @Column(name="id")
    public String getId(){
        return this.id;
```

```
    }
    /**
     * Get geo_id.
     * @return geo_id.
     */
    @Column(name="geo_id")
    public int getGeo_id(){
        return this.geo_id;
    }
    /**
     * Get created_at.
     * @return created_at.
     */
    @Column(name="created_at")
    public String getCreated_at(){
        return this.created_at;
    }
    /**
     * Get from_user_id.
     * @return from_user_id.
     */
    @Column(name="from_user_id")
    public String getFrom_user_id(){
        return this.from_user_id;
    }
    /**
     * Get to_user_id.
     * @return to_user_id.
     */
    @Column(name="to_user_id")
    public String getTo_user_id(){
        return this.to_user_id;
    }
    /**
     * Get from_user.
     * @return from_user.
     */
    @Column(name="from_user")
    public String getFrom_user(){
        return this.from_user;
    }
    /**
     * Get profile_image_url.
     * @return profile_image_url.
```

```
     */
    @Column(name="profile_image_url")
    public String getProfile_image_url(){
        return this.profile_image_url;
    }
    /**
     * Get iso_language_code.
     * @return iso_language_code.
     */
    @Column(name="iso_language_code")
    public String getIso_language_code(){
        return this.iso_language_code;
    }
    /**
     * Get source.
     * @return source.
     */
    @Column(name="source")
    public String getSource(){
        return this.source;
    }
    /**
     * Get text.
     * @return text.
     */
    @Column(name="text")
    public String getText(){
        return this.text;
    }
    /**
     * Get lat.
     * @return lat.
     */
    @Column(name="lat")
    public double getLat(){
        return this.lat;
    }
    /**
     * Get lon.
     * @return lon.
     */
    @Column(name="lon")
    public double getLon(){
        return this.lon;
```

```
    }
    /**
     * Get stusps.
     * @return stusps.
     */
    @Column(name="stusps", columnDefinition = "char(2)")
    public String getStusps(){
        return this.stusps;
    }
    /**
     * Get stname.
     * @return stname.
     */
    @Column(name="stname")
    public String getStname(){
        return this.stname;
    }
    /**
     * Get city.
     * @return city.
     */
    @Column(name="city")
    public String getCity(){
        return this.city;
    }
    public void setGeo_id(int geo){
        this.geo_id = geo;
    }
    public void setId(String id){
        this.id=id;
    }
    public void setCreated_at(String created_at){
        this.created_at=created_at;
    }
    public void setFrom_user_id(String from_user_id){
        this.from_user_id=from_user_id;
    }
    public void setTo_user_id(String to_user_id){
        this.to_user_id=to_user_id;
    }
    public void setFrom_user(String from_user){
        this.from_user=from_user;
    }
    public void setProfile_image_url(String profile_image_url){
```

```
        this.profile_image_url=profile_image_url;
    }
    public void setIso_language_code(String iso_language_code){
        this.iso_language_code=iso_language_code;
    }
    public void setSource(String source){
        this.source=source;
    }
    public void setText(String text){
        this.text=text;
    }
    public void setStusps(String stusps){
        this.stusps = stusps;
    }
    public void setStname(String stname){
        this.stname=stname;
    }
    public void setLat(double lat){
        this.lat=lat;
    }
    public void setLon(double lon){
        this.lon=lon;
    }
    public void setCity(String city){
        this.city=city;
    }
}
```

Session bean

```
package ejb.Service;
import java.util.List;
import java.math.BigInteger;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import ejb.Entity.TweetWithGeo;

import static java.lang.System.out;
/**
 * This EJB session bean has two methods to implement searching functions
 * @author Yu Qiao, Wei Chen and Igor
 *
 */

@Stateless
public class SearchSessionBean {
    /**
     * The name of the persistence unit as defined in the persistence.xml
file.
     */
    @PersistenceContext(unitName="examples-769-EJB")
    /**
     * Entity manager.
     */
    EntityManager em;
    /**
     * get tweets from H2 database through the entity manager in JPA.
     * @param keyword a list of keywords.
     * @param location a list of locations.
     * @param option an indicator whether the query is to return count
or actual tweets.
     * @return
     */
    public Integer getTweetFromH2(String [] keyword, String [] location,
int option){
        String query=constructQuery(keyword,location,1);
        return
((BigInteger)em.createNativeQuery(query).getSingleResult()).intValue(
);
    }
```

```
    /**
     * @param keyword a list of keywords.
     * @param location a list of locations.
     * @return a list of Tweets.
     */
    public List<TweetWithGeo> getTweetFromH2(String [] keyword, String
[] location){
        String query=constructQuery(keyword,location,2);
        query+=" limit 400";
        return
em.createNativeQuery(query,ejb.Entity.TweetWithGeo.class).getResultLi
st();
    }
    /**
     * Construct a native sql query based on user input parameters and
option.
     * @param keyword a list of keywords.
     * @param location a list of locations.
     * @param option whether return count or tweets, 1 for count, 2 for
actual tweets.
     * @return
     */
    public static String constructQuery(String [] keyword, String []
location, int option){
        String query=null;
        String inClauseLocation = null;
        String regExpClause = null;
        if(option==1){
            query="SELECT count(*) FROM TWEET_GEO_VIEW ";
        }else if(option==2){
            query="SELECT * FROM TWEET_GEO_VIEW ";
        }
        if (location != null){
            inClauseLocation = " ('"+location[0]+"'";
            if(location.length>0){
                int i;
                for(i=1;i<location.length;i++){
                    inClauseLocation += "," + "'" + location[i] + "'";
                }
            }
            inClauseLocation += ") ";


        }
```

```
        if(keyword != null)
        {
            regExpClause = "UPPER(text) "+sqlRegexp(keyword[0]);
            if(keyword.length>0){
                int i;
                for(i=1;i<keyword.length;i++){
                    regExpClause    +=    "    OR    UPPER(text)    "    +
sqlRegexp(keyword[i]);
                }
            }
        }
        //only use location to search
        if(regExpClause == null){
            query += "where UPPER(stusps) in" + inClauseLocation + "or
UPPER(stname) in" + inClauseLocation;
        }
        //only use keyword to search
        else if (inClauseLocation == null){
            query += "where " + regExpClause;
        }
        //use both keyword and location to search
        else{
            query += "where (UPPER(stusps) in" + inClauseLocation + "or
UPPER(stname) in" + inClauseLocation
                    + ") and (" + regExpClause + ")";
        }
        return query;
    }
    /**
     * A utility function to construct a regular expression of where
constrain.
     * @param s a string to be converted to regular expression.
     * @return a regexp string.
     */
    public static String sqlRegexp(String s){
        return "REGEXP '[^A-Z]"+s+"[^A-Z]'";
    }

}
```

Servlet

```
package ejb.Servlet;
import static java.lang.System.out;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSSerializer;

import ejb.Entity.TweetWithGeo;
import ejb.Service.SearchSessionBean;

/**
 * @author Wei Chen, Yu Qiao, Igor. Class SearchServlet.
 */
@WebServlet("/SearchServlet")
public class SearchServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * Field searchSessionBean is an enterprise java session bean object.
     *
     */
    @EJB
    private SearchSessionBean searchSessionBean;
```

```
    /**
     * @see HttpServlet#HttpServlet()
     */
    public SearchServlet() {
        super();
    }
    /**
     * The doGet method for handling an HTTP get request.
     */
    protected final void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
IOException {
        final long startTime = System.nanoTime();
        // code

        List<TweetWithGeo> result = null;
        Integer count = null;
        String[] locationArray = null;
        String[] keywordArray = null;
        final int option = 0;
        if (request.getParameter("locations") != null) {
            locationArray                                                =
(request.getParameter("locations")).toUpperCase()
                    .split(",");
        }
        if (request.getParameter("keywords") != null) {
            keywordArray                                                 =
(request.getParameter("keywords")).toUpperCase()
                    .split(",");
        }

        final PrintWriter writer = response.getWriter();
        response.setCharacterEncoding("UTF-8");
        response.setContentType("text/xml");

        String xmlString = "";
        try {
            final      DocumentBuilderFactory      builderFactory      =
DocumentBuilderFactory
                    .newInstance();
            final DocumentBuilder docBuilder = builderFactory
                    .newDocumentBuilder();
            final Document doc = docBuilder.newDocument();
```

```
            if (request.getParameter("option") != null) {
                count                                                    =
this.searchSessionBean.getTweetFromH2(keywordArray,
                        locationArray, option);
                xmlString   =   new   SearchServlet().createXmlTree1(doc,
count);
                out.print(xmlString);
            } else {
                result                                                   =
this.searchSessionBean.getTweetFromH2(keywordArray,
                        locationArray);
                xmlString   =   new   SearchServlet().createXmlTree2(doc,
result);
            }
        } catch (Exception e) {
            System.out.println(e);
        }
        writer.write(xmlString);
        writer.close();
        final long endTime = System.nanoTime();
        final float scale = 1000000000F;
        out.println("Took " + (endTime - startTime) / scale + " s");
    }

    /**
     * Create a string representation of the XML with the number of returned
tweets.
     * @param doc the XML document object to be appended.
     * @param count the number of tweets retrieved.
     * @return a string representation of the doc.
     */
    public final String createXmlTree1(Document doc, Integer count)
            throws Exception {
        // This method creates an element node; Element implements Node
        final Element root = doc.createElement("root");
        // adding a node after the last child node of the specified node.
        // out.println("1");
        doc.appendChild(root);
        final Element newNode = doc.createElement("count");

    newNode.appendChild(doc.createTextNode(Integer.toString(count)));
        root.appendChild(newNode);
        return getStringFromDoc(doc);
    }
```

```
    /**
     * Create a string representation of the XML with the number of returned
tweets.
     * @param doc the XML document object to be appended.
     * @param result a list of TWeetWithGeo objects.
     * @return a string representation of the doc.
     */
    public final String createXmlTree2(Document doc, List<TweetWithGeo>
result)
            throws Exception {
        // This method creates an element node; Element implements Node
        // out.println("the size is "+result.size());
        final Element root = doc.createElement("root");
        // adding a node after the last child node of the specified node.
        // out.println("1");
        doc.appendChild(root);
        if (result == null) {
            return getStringFromDoc(doc);
        }
        // out.println("2");
        final XPath xpath = XPathFactory.newInstance().newXPath();

        XPathExpression exp;
        Object exprResult;
        NodeList nodeList;

        for (int i = 0; i < result.size(); i++) {
            final TweetWithGeo a = result.get(i);
            // out.println(a.getStname());
            exp = xpath.compile("//state[@name=\"" + a.getStname() +
"\"]");
            exprResult = exp.evaluate(doc, XPathConstants.NODESET);
            nodeList = (NodeList) exprResult;
            if (nodeList.getLength() > 0) { // if state exists
                final Node curState = nodeList.item(0);
                exp     =     xpath.compile("//location[@geoId=\""     +
a.getGeo_id()
                        + "\"]");
                exprResult           =           exp.evaluate(curState,
XPathConstants.NODESET);
                nodeList = (NodeList) exprResult;
                if (nodeList.getLength() > 0) { // if location exists
                    final Node curLocation = nodeList.item(0);
```

```
                final Element newTweet = doc.createElement("tweet");
                // 1

    newTweet.appendChild(doc.createTextNode(createCDATA(a
                        .getText())));
                curLocation.appendChild(newTweet);
            } else {
                final Element newTweet = doc.createElement("tweet");
                // 2

    newTweet.appendChild(doc.createTextNode(createCDATA(a
                        .getText())));

                final         Element         newLocation         =
doc.createElement("location");
                newLocation.setAttribute("geoId",
                        Integer.toString(a.getGeo_id()));
                newLocation.setAttribute("city", a.getCity());
                newLocation
                        .setAttribute("lat",
Double.toString(a.getLat()));
                newLocation
                        .setAttribute("lon",
Double.toString(a.getLon()));

                newLocation.appendChild(newTweet);
                curState.appendChild(newLocation);
            }
          } else {
            final Element newState = doc.createElement("state");
            newState.setAttribute("name", a.getStname());

            final Element newTweet = doc.createElement("tweet");
            // 3

    newTweet.appendChild(doc.createTextNode(createCDATA(a.getText()))
);

            final          Element          newLocation          =
doc.createElement("location");
            newLocation.setAttribute("geoId",
                    Integer.toString(a.getGeo_id()));
            newLocation.setAttribute("city", a.getCity());
            newLocation.setAttribute("lat",
```

```
Double.toString(a.getLat()));
              newLocation.setAttribute("lon",
Double.toString(a.getLon()));

              newLocation.appendChild(newTweet);
              newState.appendChild(newLocation);
              root.appendChild(newState);
           }
       }
       return getStringFromDoc(doc);
   }

   public static String createCDATA(String s) {
       // return "<![CDATA["+s+"]]>";
       return s;
   }

   /**
    * Write a Doc object into a string.
    * @return a string representation of the doc object.
    */
   public final String getStringFromDoc(Document doc) {
       final       DOMImplementationLS       domImplementation       =
(DOMImplementationLS) doc
              .getImplementation();
       final           LSSerializer           lsSerializer           =
domImplementation.createLSSerializer();
       return lsSerializer.writeToString(doc);
   }

}
```

Web page

```
<!-- Plain HTML page that kicks us into the app -->

<html>
<head>
    <!--For iPhone-->
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no"
/>
    <link href="css/style.css" rel="stylesheet" type="text/css" />
    <script
src="http://maps.google.com/maps?file=api&v=2&key=AIzaSyBCUJrJ8VqAyah
onwNz0VmXONc0Tdo-8e8"
        type="text/javascript"></script>

    <script src="js/main.js" type="text/javascript"></script>

    <title>Twitter Enterprise Java Application--CSE 769--Wei
Chen(David), Yu Qiao, Igor Tolkachev</title>

</head>
<body onload="init()" onresize="adjustWindow()">
    <h1 >Twitter Miner</h1>
    <div id="leftPanel" style="float: left;" >
        <div id="inputContainer" style="float: left; margin-bottom:
5px;">
            <form action="SearchServlet" method="get" >
                <div>Search Tweets and map them out by choosing following
                    options:</div>
                <input type="checkbox" id="checkKeyword"/> By keywords
                <input type="text" id="keywordList" title="comma
seperated keyword." value="pizza"/><br />
                <div style="float: left">
                <input type="checkbox" id="checkLocation"/> By locations
                <input type="text" id="locationList" title="comma
seperated state name or state abbr." value="OH"/>
                </div>
                <input type="button" value="Search" style="float:left;
margin-left:5px;" onclick="search()"></input>
            </form>
        </div>
        <div id="map" title="Map" style="float: left; border: thin grey
solid"></div>
    </div>
```

```
    <div id="resultContainer" style="float:left;" class="result">
        Search results will be shown here:
    </div>
</body>
</html>
```

Javascript

```
// authors: Wei Chen, Yu Qiao, Igor
function Point(state,city,lat,lon,numOfTweets,tweetText){
    this.state=state;
    this.city=city;
    this.lat=lat;
    this.lon=lon;
    this.numOfTweets=numOfTweets;
    this.tweetText=tweetText;
}

var markers;
var bounds =null;
var keywordLst=null;

//construct query string from the user inputs in the form.
function getQueryString(){
    var para="";
    keywordLst=null;
    if(document.getElementById("checkKeyword").checked==false
        && document.getElementById("checkLocation").checked==false){
        alert("Please choose at least one of the following search
options.");
        return;
    }else{
        var value=null;
        if(document.getElementById("checkKeyword").checked==true){

    if((value=document.getElementById("keywordList").value)!=""){
                para+="keywords="+value;
                keywordLst=value.split(",");
            }
        }
        if(document.getElementById("checkLocation").checked==true){


    if((value=document.getElementById("locationList").value)!=""){
                //alert(value);
                if(para!=""){
                    para+="&locations="+value;
                }else{
                    para="locations="+value;
                }
```

```
            }
        }
    }
    if(para==""){
        alert("Please fill in at least one of the following search text
area.");
    }
    return para;
}

//Ajax search function. Get the number of tweets may be retrieved.
function search(){
    document.getElementById("resultContainer").innerHTML="Searching
twitter database...";
    map.clearOverlays();
    markers=[];
    var para=getQueryString();
    GDownloadUrl("SearchServlet?"+para+"&option",       function(data,
responseCode) {
         if(responseCode == 200) {
            var xml = GXml.parse(data);
            var                          count                          =
xml.documentElement.getElementsByTagName("count")[0].textContent;
            if(count>400){

document.getElementById("resultContainer").innerHTML+=count+" results
found. " +
                  "Only the most recent 400 records will be returned. "
+
                  "Loading...Wait for about "+400/200+" seconds";
            }else{

document.getElementById("resultContainer").innerHTML+=count+" results
returned. " +
                  "Loading...Wait for about "+count/200+" seconds";
            }
            loadData(para);
         } else if(responseCode == -1) {
           alert("Data request timed out. Please try later.");
         } else {
           alert("No results returned.");
           document.getElementById("resultContainer").innerHTML="No
results yet...";
         }
```

```
    });
}

//Ajax load data function. Get the actual number of tweets.
function loadData(para) {
    GDownloadUrl("SearchServlet?"+para, function(data, responseCode) {
          // To ensure against HTTP errors that result in null or bad data,
          // always check status code is equal to 200 before processing
the
            // data
        /*state        city    # of tweets    first 5 tweets*/
          if(responseCode == 200) {
            var xml = GXml.parse(data);

            var                              states                              =
xml.documentElement.getElementsByTagName("state");
            //alert(states.length);
            if(states==null){
            alert("No tweets retrieved!");
            return;
            }
            var points=[];
            for (var i = 0; i < states.length; i++) {
            var locations=states[i].getElementsByTagName("location");
            for(var j=0;j<locations.length;j++){
                var tweets=locations[j].getElementsByTagName("tweet");
                var tweetTexts=[];
                for(var k=0;k<tweets.length;k++){

    tweetTexts.push((k+1).toString()+"."+tweets[k].textContent);
                }
                point=new Point(states[i].getAttribute("name"),
                        locations[j].getAttribute("city"),
                        locations[j].getAttribute("lat"),
                        locations[j].getAttribute("lon"),
                        tweets.length);
                var text;
                if(tweetTexts.length>100){
                    text=tweetTexts.slice(0,100).join("<br/>");
                }else{
                    text=tweetTexts.join("<br/>");
                }
                //alert(keywordLst.length);
                //var o="ohio";
```

```
                if(keywordLst!=null){
                    for(var k=0;k<keywordLst.length;k++){
                        //var pattern="[^a-z]("+keywordLst[k]+")[^a-z]";
                        //alert(pattern);
                        var re = new RegExp(keywordLst[k],"gi");
                        text=text.replace(re,
                                "<span
style='color:red'>"+keywordLst[k]+"</span>");
                        //alert(text);
                    }
                }
                point.tweetText=text;
                points.push(point);
            }
            }
            points.sort(compare);
            createTable(points);
            drawMarkers(points);
          } else if(responseCode == -1) {
            alert("Data request timed out. Please try later.");
          } else {
            alert("No results returned.");
            document.getElementById("resultContainer").innerHTML="No
results returned.";
          }
        });
}

//Draw markers on Google Maps.
function drawMarkers(points){
    var maxNumOfTweets=0,sumNumOfTweets=0;
    for(var i=0;i<points.length;i++){
    //alert(points[i].city);
    sumNumOfTweets+=points[i].numOfTweets;
    if(points[i].numOfTweets>maxNumOfTweets){
        maxNumOfTweets=points[i].numOfTweets;
    }
    }
    //alert(sumNumOfTweets);
    var tweetPercentages=[];
    for(var i=0;i<points.length;i++){
    //alert(points[i].city);
    tweetPercentages.push(points[i].numOfTweets*1.0/sumNumOfTweets);
    }
```

```
    //alert(tweetPercentages.join(","));
    bounds= new GLatLngBounds();
    for(var i=0;i<points.length;i++){
    var point=points[i];
    var                  latlon                  =                  new
GLatLng(parseFloat(point.lat),parseFloat(point.lon));
        bounds.extend(latlon);
        var                 html                 =                 "<div
class='info'>"+point.city+","+point.state+"<br/>"
            +point.lat+ "," +point.lat
            + "<br/># of tweets:"+ point.numOfTweets+"</div>";
        var title=point.city;
        var ratio=point.numOfTweets*1.0/maxNumOfTweets;
        //alert(latlon.lat()+" "+latlon.lng());
        //exit(0);
        var    marker    =    createMarker(latlon,    html,    title,
createIcon(ratio));
        map.addOverlay(marker);
        markers.push(marker);
    }
    map.setCenter(bounds.getCenter(),map.getBoundsZoomLevel(bounds));
    //alert(map.getBoundsZoomLevel(bounds));
    //adjustTable(document.getElementById("theOnlyTable"),tweetPercen
tages);
}

//Create the table of tweets.
function createTable(points){
    var div=document.createElement("div");
    var resultTable=document.createElement("table");
    resultTable.id="theOnlyTable";
    div.appendChild(resultTable);

div.style.height=document.getElementById("resultContainer").clientHei
ght-10;
    div.style.overflow="auto";
    //alert(div.style.height+div.style.overflow);
    document.getElementById("resultContainer").appendChild(div);
    var tbody=document.createElement("tbody");

//resultTable.style.height=document.getElementById("resultContainer")
.clientHeight;
    resultTable.appendChild(tbody);
    var tr=document.createElement("tr");
```

```
    tbody.appendChild(tr);
    var th=document.createElement("th");
    th.style.width="40px";
    th.innerHTML="State";
    tr.appendChild(th);

    th=document.createElement("th");
    th.style.width="40px";
    th.innerHTML="City";
    tr.appendChild(th);

    th=document.createElement("th");
    th.style.width="50px";
    th.innerHTML="#OfTweets";
    tr.appendChild(th);

    th=document.createElement("th");

th.style.width=document.getElementById("resultContainer").width-130;
    th.innerHTML="DetailedTweets";
    tr.appendChild(th);
    for(var i=0;i<points.length;i++){
        tr=document.createElement("tr");
        tr.setAttribute("valign","top");
        tr.setAttribute("index",i);
        td=document.createElement("td");
        td.innerHTML=points[i].state;
        tr.appendChild(td);

        td=document.createElement("td");
        td.innerHTML=points[i].city;
        tr.appendChild(td);

        td=document.createElement("td");
        td.innerHTML=points[i].numOfTweets;
        tr.appendChild(td);

        td=document.createElement("td");
        var div=document.createElement("div");
        div.style.overflow="auto";
        div.margin="0px";
        div.innerHTML=points[i].tweetText;
        td.appendChild(div);
        tr.appendChild(td);
```

```
        GEvent.addDomListener(tr, 'click',
                function() {
                GEvent.trigger(markers[this.getAttribute("index")],
'click');
        });
        GEvent.addDomListener(tr, 'mouseover', function(){
            this.style.backgroundColor="#ff6";
        });
        GEvent.addDomListener(tr, 'mouseout', function(){
            this.style.backgroundColor="#ffc";
        });
        tbody.appendChild(tr);
    };
}

//Adjust the look of the result table.
function adjustTable(thisTable,tweetPercentages){
    //alert(tweetPercentages.length);
    var resultContainer=document.getElementById("resultContainer");
    var minHeight=1000;
    var
trs=thisTable.getElementsByTagName("tbody")[0].getElementsByTagName("
tr");
    for(var i=1;i<trs.length;i++){
        if(trs[i].offsetHeight<minHeight){
            minHeight=trs[i].offsetHeight;
        }
    }

    //alert(thisTable.offsetHeight+","+trs.length);
    //trs=trs.splice(1,trs.length);
    //alert(minHeight);
    for(var i=1;i<trs.length;i++){
        //alert(trs[i].style.height);
        var
div=trs[i].getElementsByTagName("td")[3].getElementsByTagName("div")[
0];

    div.style.height=(resultContainer.clientHeight-trs[0].offsetHeigh
t-minHeight-80)*tweetPercentages[i-1]+minHeight;
        //alert(tweetPercentages[i-1]);

    //alert(trs[i].style.height+","+trs[i].offsetHeight+","+trs[i].cl
ientHeight);
```

```
    }
}

//crate marker.
function createMarker(latlon, html, title, icon)
{
    var marker = new GMarker(latlon, {
        title: title,
        icon: icon
    });
    GEvent.addListener(marker, 'click', function ()
    {
        this.openInfoWindowHtml(html);
    });
    return marker;
}

//create icon.
function createIcon(ratio)
{
    //alert("ratio is "+ratio);
    var maxRadius=50;
    var minRadius=10;
    var icon = new GIcon();
    icon.image = "img/circle.png";
    //default size 20x34
    var radius=(maxRadius-minRadius)*ratio+minRadius;
    icon.iconSize = new GSize(radius, radius);
    icon.iconAnchor = new GPoint(radius/2, radius/2);
    icon.infoWindowAnchor = new GPoint(radius/2, radius/2);
    icon.infoShadowAnchor = new GPoint(radius/2, radius/2);
    return icon;
}

//sort tweets by count.
function compare(a,b) {
    if (a.numOfTweets > b.numOfTweets)
        return -1;
    if (a.numOfTweets < b.numOfTweets)
        return 1;
    return 0;
}
var map;
```

```
function init(){
    adjustWindow();
    map = new GMap2(document.getElementById("map"));
    var columbus=new GLatLng(39.966596, -83.009377);
    map.setCenter(columbus,10);
    map.setMapType(G_PHYSICAL_MAP);
    map.addControl(new GMenuMapTypeControl());
    map.addControl(new GScaleControl());
    map.addControl(new GOverviewMapControl());
    map.enableScrollWheelZoom();
    map.enableGoogleBar();
    map.openInfoWindowHtml(columbus,      "<div      class='info'>Hello
world~</div>");
    map.addControl(new GSmallMapControl());
}

//adjust the look of the DOM container.
function adjustWindow(){
    var heighCompressRatio=0.1;
    var widthCompressRatio=0.05;
    var maxHeight = document.body.clientHeight*(1-heighCompressRatio);
    var maxWidth = document.body.clientWidth*(1-widthCompressRatio);
    var dividentRatio=0.4;
    var divLeftPanel = document.getElementById('leftPanel');
    var divMap = document.getElementById('map');
//  /var marginLeft=10;
    divLeftPanel.style.width = maxWidth * dividentRatio;
    divLeftPanel.style.height=maxHeight;
    //divRightPanel.style.marginLeft=maxWidth                          *
dividentRatio+marginLeft;
    divMap.style.height                                                =
maxHeight-document.getElementById("inputContainer").clientHeight-5;
    divMap.style.width                                                 =
document.getElementById("leftPanel").offsetWidth-10;
/*  if(navigator.userAgent.search('Firefox')!=-1){

    }else        if(navigator.userAgent.search('Safari')!=-1        ||
navigator.userAgent.search('Chrome')!=-1){

    }else if(navigator.userAgent.search('MSIE')!=-1){

    }   */
    var resultContainer=document.getElementById('resultContainer');
    //alert(maxWidth+","+divLeftPanel.style.width);
```

```
    resultContainer.style.width=maxWidth-parseInt(divLeftPanel.style.
width);
    resultContainer.style.height=maxHeight;
    //alert(resultContainer.style.width+","+resultContainer.style.hei
ght);
}
```