**Intelligent Web Assignment**

team name: unkonwn

member: Pengyuan Zhao, Likang Cao, Shangshu Lu

# 1 Introduction

The majority requirement of this assignment have been achieved. The project can respond queries of users correctly, show result on front page in particular format, and retrieval information in database. However, there are some flaws in data query, some responses are delayed.

# 2 For each task in section 1

## 2.1 Querying the social Web

### 2.1.1

**Issues:**

The task of this part is getting relevant messages through Twitter according to queries that users input. Based on the requirement, it can limit the query to a specific geographic region(latitude/longitude). Moreover, a demonstration of people who retweeted each message must be achieved. And the most difficult thing is to select tweets that are really poplar retweeted by other users and contains the keyword and combine these query requirements with location limitation.

**Design and its Motivations:**

First part we use Tweeter ‘Search/Tweets’ api to get useful information according to user’s input. According to the requirements, we should use keywords and geographic region(like latitude and longitude) to get discussions. Firstly, we have to get users’ key word query and optional region data. This program use html5 get geolocation function which can get user’s location when it is allowed by user. As for keywords, the form can handle texts which contains useless words such as ‘a’, ‘the’ and others. This maintains the keywords are really popular and meaningful. Then twitter ’Search/Tweets’ apiprovides three useful parameters which are query, coordinates and result\_type which specify the requirements and return correct information. When parsing the return Json format data of the request, there are users, texts and relevant information can be stored in an array and shown on the web interface.

Second part is providing a hyperlink which helps users to check who have retweeted the selected popular tweet. We select ‘Retweet/:id’ API function which use tweet’s unique id to query who retweeted the information and relevant information. The tweet’s id was stored in the first part querying topics and the response information also provide retweets’s author profile url and author name. All these information helps make the result and web interface more beautiful.

Finally, the advantages of this solution works quite efficiently with tweeter API and comply with the requirements well. But the disadvantages of that are program can not get the latest tweets which were created no more than a hour or less and didn’t handle the results with hash tag and detect the popularity of the tweets.

**Requirements:**

We can fully satisfy the requirements, which are showing the relevant information and displaying the users who retweeted the messages. And below is the requirements form:

|  |  |
| --- | --- |
| Requirements | Complete degree |
| Input keywords and handle input texts | Fully complete |
| Limit query to specific geographic region | complete |
| Output message | complete and in right format |
| Display who retweeted message | Fully complete with user profile |

**Limitations:**

According to the resolution, the program can’t get latest popular topics which may resulted that if theres an emergency like earthquake or fire, the program may not get results immediately.

### 2.1.2a

**Issues:**

The objective of this part is extracting tweets from specific users and analyze the high frequent keywords existed in the last few days. The keywords will be sorted depending on the total amount. The most frequent keywords Y used in the last days Z must be displayed. Moreover, the keywords must common to all users.

The first challenge of this part is that the tweets from users will contain a lot of punctuations and blank. This part must be considered, otherwise there would be a plenty of illegal keywords come out. Secondly, the high frequency keywords may not exist in a specific user's tweets, which should be deleted from the whole wordlist to satisfy the requirement of "common to all users".

**Design and its Motivations:**

First of all, we need extract tweets from a specific users. The days and users' screen\_name can be controlled by the search/tweets API. Following by token the tweets and eliminate the punctuations and blanks, the data be stored into arrays. Meanwhile, a stoplist is set to get rid of irrelevant words, for instance, preposition and article.

Secondly, the frequency of each words can be calculated and sort them in the order. Moreover, the zero frequency of keywords must be deleted to ensure that are common to all users. At last, according to the keywords number required, the sorted array can be spliced to satisfy the requirement.

**Requirements:**

The parameters of names, keywords and days will be transmitted from page. The implement function can handle these parameters and return the table about names and high frequent keywords to users.

**Limitations:**

The process of getting and handling data is quite complex. All parts of implements are designed in the same callback function, there is lacking in several interface and encapsulation to control these functions. Which will increase the difficulties to debug implement function.

### 2.1.2b

**Issues:**

As I mentioned, when people click ‘Query’ button to search visited locations, the result can not be shown immediately, user need to click the refresh to view results.

**User operation:**

For using the search function of part 2(b), the user should click the NAVIGATOR button and select Question2B, then the user need to input the user name (i.e. screen name of twitter) and days as he/she wish. After clicking the “Query” button, the user also has to click “Refresh to see venue list” to view the result in the table, clicking the external button to get search result is also a serious weakness of this system. After that, two buttons and a list of names, address, categories, descriptions, urls of visited venues will be shown to the user. When clicking the “POI FROM DBPEDIA” button, the DBPedia page of venue’s city will be displayed to the user, when clicking the “SHOW POINTS OF INTERST” button 10 points of interests near the venue will be marked on the Google map in a new page, and the marker on the map was clickable, it will link to the related Foursquare page.

**Design choices:**

For solving the problem of finding out user’s check-in information, the twitter search API was used, then set “swarmapp/com/c/” as query, set user’s name (i.e. screen name of twitter) as parameter “from ”, and I also used setDate() to get the days we need. Then, the user’s check-in id was used to find visited venue id through the check-in API “<https://api.foursquare.com/v2/checkins/resolve>” of foursquare. Finally I extracted needed information by venue’s id through the foursquare API “<https://api.foursquare.com/v2/venues/VENUE_ID>” such as venue’s name, venue’s address and venue’s description.

**Requirement**

|  |  |
| --- | --- |
| Requirement | Achievement |
| Input user’s name and number of days as query parameter | The user’s name and number of days can be transmitted from browser to the server. |
| Call twitter search API | Used to extract check-in id. |
| Call foursquare check in API | Used to extract visited venues’ id |
| Call foursquare venue search API | Get require information of each venues such as photo, category name, address, URL and description. It also returns 10 points of interest near the visited location. |
| Points of interest were marked on Google map | It was achieved by call Google map API. |
| Marker on Google map were clickable | The markers of PoS were clickable to link venue’s home page in foursquare. |
| Link to DBPedia page of individual venue | For achieving goal by click the “POI FROM DBPEDIA” button. |
| Producing data for Web | RDFa was produced and write to rdfa.html in the root directory of folder. |

### 2.1.2c

**Issues:**

This part is to get a list of user by querying a venue’s name and check who have visited this venue in a limit days. If user input last X days text is empty, the program should using twitter streaming api to capture information when it is published. There are two main challenges of this section. First one is to select between twitter ‘Rest api’ and twitter ‘Streaming api’ according to users’ input. Second challenge is to query and store venue data and push checked in users’ in the venue array.

**Design and its Motivations:**

The design of this part use two APIs to get the correct output, first is using foursquare venue search API to get the venue’s coordinate and then use tweeter search API to get the users by querying their check in information. In order to get the correct result - a user list that appeared in a venue in the past few days, the first part is to get the correct location coordinates using Foursquare search api combined with location keywords parameter. The parameter is what user input in the form and handled by script functions that ensure user’s input is in right format. After getting a list of venues which contains venue keywords or related words, the server can compare the venue name with user’s input and return a real venue location information. Then the server use tweeter api to search the user list. In this part, if users’ input days are empty, the server could use tweeter Streaming API to search users and if days are integer, the server will use tweeter REST api. All returned data will be stored in a venueAndUser array and can be shown on the interface after well styled both on the map and a user list. In this part, user can index a selected checked in user’s past 100 tweets using Tweeter Timeline:id api.

The advantage of this design choice is that server can get correct venue information and checked in user list based on that information. The usage of ‘Streamging api’ and ‘Rest api’ make the user list keeps updating in real time. The disadvantage of this design is the venue might not be quite accurately shown on the map due to some unfixed bugs.

**Requirements:**

This solution partly fulfilled the requirement. All complete degree is shown below:

|  |  |
| --- | --- |
| Requirements | Complete degree |
| Input venues and days and handle input texts | Fully complete |
| Use Twitter streaming api and rest api | Partly Complete with few bugs using streaming api |
| Display user name, id, etc. | Complete |
| Display user’s 100 tweets | Complete |
| Information about users current whereabout | Partly complete with showing user’s location |
| Venues information List and show maps | Complete |

**Limitations:**

When users’ input keywords(such as Shop, Library, Museum etc.) are too normal to search in Foursquare, the results shown on the map are not so accurately. And users list will be too large because the server will add all users who have checked in all venues.

## 2.2 Querying the Web of Data

**Issues:**

The aim of this section is to find information about the surroundings of a venues visited by a person. To achieve function button “show point of interest”, we have to use DBPedia and Foursquare to query the information. The challenge of this part is to get correct information using SPARQL searching language.

**Design and its Motivations:**

The design of this part are searching recommended data from foursquare and DBPedia website. The difference between these two part are querying methods. When searching point of interest venues using Foursquare api, venue/search api with intent and radius parameters will be used. As for DBPedia, we use SPARQL searching language to search recommended venues’ latitude and longitude and then show these locations on the map. All venue locations will be shown on the map and popping up markers on the map. When clicking the markers the page will jump to foursquare related venue page. When press DBPedia point of interest button, it will jump to DBPedia recommended venue page.

As for the disadvantage of this design, if server cannot get a url link from DBPedia querying result, the page will not jump.

**Requirements:**

All requirements complete degree are shown below:

|  |  |
| --- | --- |
| Requirements | Complete Degree |
| Retrieve data from DBPedia and Foursquare | Complete |
| Output can be plotted on a map and clickable | Complete |
| Show information by opening its related webpage | Complete (Some venue information may can not get from [DBPedia.org](http://DBPedia.org) because of venue name not in wiki database) |

**Limitations:**

All query request and returned data is not in real time.

## 2.3 Producing Data for the Web

**Issues:**

The produced data should be persistent presented in the HTML documents, however the produced data in rdfa.html was not persistent

**Design and its Motivations:**

After producing the data of visited venues, the data will store into the rfda.html document, it includes the names, photos, addresses, categories, URL and description of venues.

**Requirements:**

|  |  |
| --- | --- |
| **Requirement** | **achievement** |
| **Getting venue information** | **It was produced through foursquare API** |
| **Store RDFa in a html page** | **The produced RDFa write to rfda.html** |

**Limitations:**

The data was not persistent presented in rdfa.html documents.

## 2.4 Storing information

**Issues:**

The objective of this part is that storing the information have been tracked, for instance, the keywords they have used, the people they have contacted and the venues they have visited, into the database of DCS. The biggest issues we faced is that to make all tables data relational. Different kind of data are gathered by individual function or API, must be stored in separated tables. The connection and relation between tables need to be handled to make the data updating and searching more efficiently and properly.

**Design and its Motivations:**

Three tables are constructed to store the information we get from the web, USERSINFO, KEYWORD, VENUEINFO. The userid can be recognized as an unique identification for one particular user account. So we can make this column as key to connect all these tables.

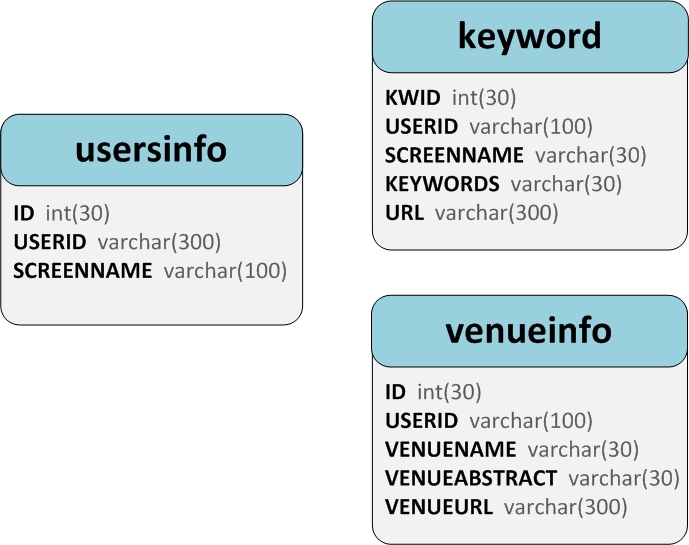


Figure 1 Design of tables in database

**Requirements:**

Users information include users' id and screen name. Screen name can be repetition but USERID is the unique identification for each twitter account, so can be primary key to build the connect between separated tables. The keyword information be stored into KEYWORD table, which are many-to-many relationship to userid. Moreover, venue information which include venue name, venue abstract and venue photo, are stored in VENUEINFO table.

Users can query keywords data and venue data from searching history page.

**Limitations:**

Because the relationship between keywords and users is many-to-many, which means one word can be used by many people and individual can send many words, meanwhile, the algorithm of handling keyword is complicated, the progress of redundant processing become tedious. It will decrease the efficiency of data retrieval.

## 2.5 Web Interface

**Issues:**

For this part, we need to design an webpage for users to send their requests to the server. Because the requirement of this program is divided into three individual part for each teammate, the biggest problem we faced is that associating different parts to the final webpage we designed.

**Design and its Motivations:**

The web interface directly influent user’s experience. To ensure all the web requests shown on the page and transfer user’s input correctly, the web interface should maintain its webpage style beautiful and stable. Multiple CSS styles are used in this project and all webpages are created by using .ejs file, which is a page format combines data and a template to produce HTML.

**Requirements:**

|  |  |
| --- | --- |
| Requirements | Complete degree |
| System implemented as HTML/Javascript sets | Fully complete |
| System controlled through browser-based interface | Fully complete |
| Interface served by node.js server and all interaction with social web done through node.js server | Complete but server name is app.js |

**Limitations:**

For the purpose of satisfying some limitation of inputs, several script code are embed into pages. It would affect the response speed of WebPages.

# 3 Division

|  |  |  |
| --- | --- | --- |
| Members | Task | Division |
| Pengyuan Zhao | **Design:** Database, 1.1.1a. 1.1.2b, 1.2 Query the web data, 1.3 Producing web data  **Implement:** 1.1.2b query, 1.2 the web data query, 1.3 producing data of web,  **Documentation:** 2.1.2b,2.3 | 33.3% |
| Likang Cao | **Design:** Web Interface, Database, 1.2a query.  **Implement:** Database, Keyword query, Web interface, Data query.  **Documentation:** 2.1.2a, 2.4, 2.5 | 33.3% |
| Shangshu Lu | **Design:** Web Interface, Database, 1.1 and 1.2c query, Web data query.  **Implement:** Web Interface, Database, 1.1 and 1.2c query, Web data query.  **Documentation:** 2.1.1, 2.1.2c, 2.2, 2.5 | 33.3% |

**Appendix: Diagram of the ontology:**

