University at Buffalo

Submitted By:
Santosh Dubey

Information Retrieval

Table of Contents

Chapter 1
Abstract
&
Introduction

Abstract

With the growing age of technology and quick results which are required and can be very rigorously needed by the today's users. With the demand of time the technology improvement in the current scope is needed. There are many research which are going on in the fields like Search, Voice over IP, quick result, and response on time etc.

As we are well aware of that every users nowadays who uses Internet wants to search for anything and everything like educational colleges, about Information Technology, books, news etc. using Search Engines. This has become the need for everyone.

This project is focusing on the same concept as in this mainly 3 search engines(Wikipedia, Yahoo, Bing) have been used which are exactly used to get the result from the passed query , the comparison of the result and their performance have also been tried in order to get faster query result.

In order to develop this project several algorithms have been used and query processing which can be in result to give faster response.

The main aim of this project is not only showing the response of passed query but it also dynamically get the contents from the Internet through HTTP request/response which is quite challenging.

There are many search engine and they are fully distributed among several places, and they uses very high machine learning , query processing technique, algorithms etc henceforth to handle these pre-requisite process and to handle millions of request/response they uses very high performance computing technology and servers.

This could be the main hurdle in this project as this project is working on the local machine. So it could be obviously not possible to make it very high performance computer to handle millions of request/response henceforth on small scale this project had given very tremendous processing results.

The details of each terms and approaches will be discussed in the later part.

Introduction

Today approximately 55% people uses search engine for their daily routine query like news, place, about something or someone etc. but 25% of people uses search engine for the business purposes and rest are used as research purposes. As the query length increases it become bottleneck to the company to extract the query and get the response.
According to the survey from google in 2010 today 40% people insert 3-4 lines of query in google to get the response, so as the size of query increases the processing time and query handling time also increases which is quite challenging for all search engines. They uses many heuristic-based algorithm and they also proposed the technique for the computation of the resulting relation size after a projection of the result/query.

In this project the same logic and concepts have been tried to make one individual search engine which can be even used as distributed search engine. So the aim of this project is to present results from multiple sources for a given query. In this project the database are mounted on other side of the system in separate directory which can be then taken into consideration for the further processing.

Henceforth for this purpose, the below three sources have been taken to make the static database of the project.

- Web : This contains all general information of the fired query.
- Travel : This have been taken to give the information about the places. This data source have details of all places in it.

Similarly there are other 3 databases (Bing, Yahoo, and Wikipedia) which contains dynamic data sets into it.
The above data sources have been selected to show the results based on deep research and survey results. In this project each source is maintained in a separate index and results are combined from multiple sources for a given query when being displayed to the user. A webpage is created for a given query, showcasing different results from different sources.
Since the survey result shows different graph pitch so the above mentioned sources have been bifurcated into separate databases, every time user fires query the result will be taken from all those data sources. The query response in this sense will depend on the results and the query size but many times the database tool (Solr) gives very slow response due to database sizes.

This is the survey result which shows how much query have been fired from particular fields over a decade.

- As we can see the survey result shows 92% of the search was related to travel or place.
- 91% for "General/Web" search.

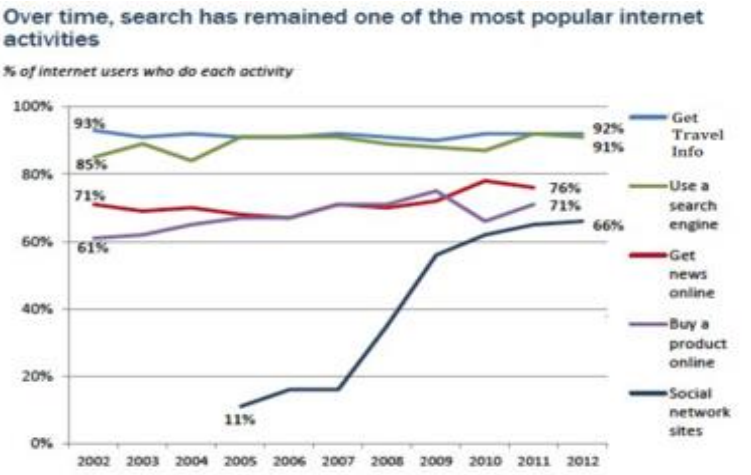


fig: 1 (ref: American Life Project tracking surveys, 2002-2012)

It is very vital step to take some survey before proceeding to make the database of any search engine. So once the results have been found then the data-sources have been selected.
This was the main reason behind selecting only these 2- datasrouces.
But now next most difficult task was like how to select data for all these survey results in order to make database. Then after deep research and thinking the final conclusion was to keep some limitation in the data the main reason behind this was that the process is going to run on local machine hence the data should be limited as there are limitation of the hardware and software, otherwise system can be crashed.
So to ensure different results to be related, 2 distinct data sources have been selected after keeping **Location as a theme** for search criteria. As there are the data sources available at developer sources location on Wikipedia which contains all live data in uncompressed XML format which contains huge amount of the data which are not exactly required in this project in order to get own data and make data sets many contents have been taken from the different sources from the web, after taking these contents and to make the databases it have been converted into XML format with their specific fields of the data.

So in order to get only data these data have been taken into this project and mounted on the database. Pages from Web, Travel data are parsed and only those pages are indexed whose title pertains to countries and major cities in the world. There have been many process applied once the data was ready to parse which can be discussed later in the part. To have the data dynamically the crawler have been implemented which will generate the XML format of the data once it finishes retrieving, it will get title, id and link of the pages from the search engine like Yahoo, Bing, Wikipedia. These search have been selected based on their popularity.

However the database related to these search engines contain dynamic data so in that sense data in the solr are dynamics and each of them contains distinct data values into it.

However the ranking of this data which are in these cores are not possible since they may contains different titles with different links, as the of the project is mainly based on location so there can be possibilities that each title of the documents may contains long string or phrase or sentences in which it is not possible to boost their score and ranking.

So solr renders all results for any specific query so to search those query response user can go onto the next page UI and get the result as required.

Why particular topic chosen?

- In today's world everyone needs information about everything instantly.
- Which becomes bottleneck to the company fulfill this requirements.
- As the number of query increases the, the number of response also. Which becomes very challenging for the company. There are many research going on in this fields as this fields have been evolved tremendously.

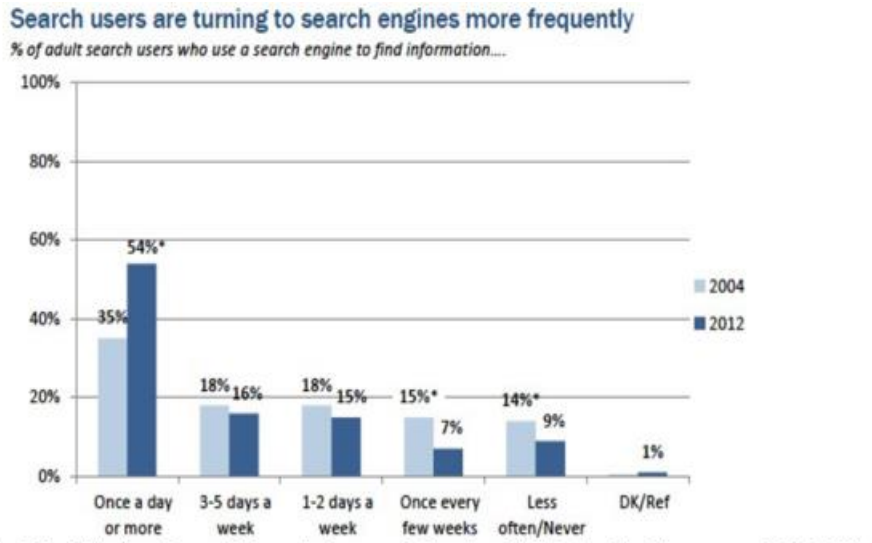


fig: 2 (ref: The Pew Research Center's Internet & American Life Project Tracking survey 2002-2012)

- So taking those into consideration this project was the best topic which can be selected and to perform experiment at research level.
- The person uses the search engine to find out their search result. In order to gather information the user has to go through several links to find out the best suited result. The aim of this project is to provide users a single link to find out the only best suited link to their search result.
- The search is mainly based on the location however the dynamics search allow user to get the result as required by the user; the user can retrieve their search result through three sources i.e. from Web and Travel.
- The project has been chosen based on user comforts. Whenever we go online we are provided with several links. So we have to go through all the links and this takes more time to read all the data to find out the best suited results.
- So keeping this situation in mind this topic have been selected to provide users with a single link the most suitable link that can be selected to find out the result in less time instead of checking out all the links related to that search topic. It is mainly used to provide less time and more accurate to find out the result compared to what the entire search engine provides.

Problem Definition

Defining the problem is one of the most important activities of the project. The objective is to define precisely the problem to be solved and thereby determine the scope of the new proposed system. In the today's existing search system users access Internet and getting the information from different Web sites. The approach to find information from web-sites is a difficult task. To ease for searching of various information' over the web-sites by giving search keywords requires software. To search any information in this system is long process and takes long time to search.

The new proposed search engine system ensures the end user to get the information by firing the query into specified in the database. Search engines are the most widely used means of finding information on the Internet. The proposed search engine provides the collective wisdom of all the top search result with the updated information.

The project phase consists of two main tasks:-

- The first task within the activity is to review the needs that originally initiated the project.
- The second task is to identify, at an abstract or general level, the expected capabilities of the proposed system.

A clear understanding of the problem will help us to build a better system and reduce the risk of project failure. It also specifies the resources that are to be made available to the project

The problem that is to be solved by the new system includes the following:-

- Unfortunately, those who can't handle mouse or type on keyboard or those who are lazy in typing the text can simply say the name of the location and that will directly redirected to search result.
- Users don't have to go through all the links provided to them based on their search.
- Users don't have to check all the links to find out the appropriate result based on their search.
- Users are provided with a single link to their searched queries. To allow the user to see their searched result by clicking a single link

- To allow the user to see three different searched results simultaneously from different sources.
- Save the user from having to use multiple search engines separately.
- Simplify information search and cut the time by pointing to the most relevant source documents
- Ad hoc access to multiple heterogeneous sources from a single point of access.
- Provide a dynamic, scalable access infrastructure that can be integrated with other features.

Chapter 2
Research Review

Literature Background of the project

i) Working of Search Engine :

This project is aiming on both static data as well as on dynamic data. To make static data two data-sources have been selected and the data have been selected from different web sources. Wikipedia provides live latest data in compressed XML format. These compressed data are very large in size approximate 100 to 200GB in size. So to make individual data sets in order to make full functional database some of data have been selected from the Wikipedia.

So as based on the survey report (mentioned above) two data-sources :

1) Web
2) Travel (Wikivoyage)

Role of Super Computer

Once these compressed data have been found now time has come to extract only the required data for the project since all the data could not be used in the process due to local system limitation. So for the same reason the "Python" code have been written in order to extract the data from the compressed data format. Since the size of the data is very large so this could not run on local system else there would have been the chances that system could have been crashed. So this process had been performed on "Super Computer" which had taken nearly 1 hours to parse whole documents.

To overcome this problem the data selection have been given location boundary. Before implementing the parsing "Python" code one manual text file have been created which contains important city and country name based on this text file the python code works. Since compressed data which have been taken from Wikipedia contains many fields but all fields are not required else that can cause more complication in extraction. The python code retrieved the data from the compressed data file based on the id, title, url, iLink, eLink, text rest fields will be ignored. After getting this fields values the python code will generate new XML data file which will only contains above mentioned fields.

Role of Eclipse Parsing

Since the data are in XMl format and it is not in user readable format so it should be parse.

So in eclipse the parsing code have written which parse all these huge data files files. The code have been written in "REGEX" and "JAVA". While parsing this data it removes all wiki-tags and XML-tags in order to get in desired text from the file. Once these data have been parsed it will be mounted on solr. Each databases have its own parsing code with its corresponding "Runner" this runner is responsible for activating all of its related code file.

But the challenge had been faced when it comes to parse the huge file in eclipse since eclipse JVM only allows 256MB which can be extended but it could not be more than 1GB. So to overcome this problem the java data structure have been used in the code which keeps on parsing the file also the external libraries of the solr have been used in order to mount the data simultaneously on the solr with the parsing process. So with this the eclipse never faced any problem to parse whole data file.

Role of User Interface

All the HTML codes have been deployed inside the tomcat. Since this project is running on the local machine so in order to process request/response we need local server. So the tomcat has been configured in such a way that whatever the query have been passed and it will understand that the result has to be fetched from Solr. Once this data is mounted on Solr, browser starts checking whether the results can be retrieved or not.

ii) **Role of Crawler**

The crawler in search engines plays very vital role to fetch the dynamic data across the internet. So the crawler is working very differently in the project, since till now the data was kind of static and if the user fire some query outside of the scope then it could be bit disappointing to get the results. So the crawler is working dynamically. Since the results can be different if we jump on different search engine with the same query. So in this project the main reason behind using crawler was to compare the results from different search engines and

show them to the users. Then accordingly the user can selected best suited results or links.

So in order to do that possible assumptions in practical crawler have been implemented, This crawler will crawl the page from giant search engines like :

a) Yahoo!
b) Bing
c) Wikipedia

The query can be passed and based on the query the crawler will crawler these 3 search engines and get the result on the screen. Now in order to give the users more flexibility in crawling pages the level of the search/crawl facility have been provided. In crawler searching according to the level plays important role which can superimposed to get good results.

a) **Level 1 :** This is light crawling, in this no redirect links can be crawl and this is not based on the "Recursive" based algorithms.
b) **Level 2 :** At this level all the links from the page as well as re-directs links of that page can crawled. This is based on the "Recursive" based algorithms.

Once user fires query from crawler the crawler starts crawling the pages based on the level of the crawling given. Firstly the crawler will only crawl links and it will keep on assigning unique ID to all the links. There can be possibility that some of the links can be repeated because of the re-directs pages and arrival of the links in later stage so in crawler user have given an options to remove the duplicate links according to its unique ID. Once the duplicate links have been removed the user can now get the TITLE from the retrieved links with according to its corresponding unique ID after deleting duplicate links. Crawling of the title takes couple of minutes this depends on the level of search or crawl. Because the crawler takes each links which have been crawled and based on that it recursively fetch the Title of that link. So if the user defined and started crawling at level 2 then he might get approximately 2000-100000 of the links in crawler. So to get the Title of this much is very time consuming also there can be other chances that crawler will start crawling homepage of the search engines which will cause processing delay in the crawling so to avoid this another filter have been used which will eliminate homepage links.

The reason behind not getting title and links simultaneously is that each links do not have TITLE on it so that might cause problem to the crawler to crawl the pages. So the best way was to first get the links and based on the links after applying different algorithms like BSF, RECURSIVE the result can be taken. User have given an option in which they can crawl three search engines parallel after which they can compare the results of each search engines.

Once the result have been retrieved then the crawler will generate XML format file of the result with its documents page. After getting this generated XML file the parsing code from eclipse will parse this XML and mount it on solr and from solr the query processing will take place and then later can be displayed on the screen.

iii) Role of MAP Search

As the search is mainly based on the "LOCATION" so user should get an added advantage to search through MAP. This the main feature of the search. MAP has been plotted form google inside the UI but this was the plain MAP. Now in order to show the city and country the same name as MAP coordinates should be printed or shown on the MAP. In order to do that "place.csv" have been manually created which contains major cities and countries name (Approximately 468) with its latitude and longitude values. The database in MYSQL has been created which in which all this cities and countries have been stored.

Once all the names with its corresponding longitude & latitude are stored then the PHP code file (which are stored in Webapp folder) which will then pick up all these values from that database through LAMP Server (Apache2) , phpmyadmin on the MAP bifurcating city into red marker and country into blue marker.

The latitude and longitude have been taken based on the median of the centroids geographical coordination values of each country and city manually. On each country and city the marker contains one small form contains the name of the city as a link. When the user clicks on this it will redirect the user on the result page and find out the values from the solr and display on the result page.

When user will clicks on the city it will be in the form of the link on the MAP but it will go as a query inside the solr. The process between sending query and

receiving response from solr and displaying results on UI JavaScripts will be called which will handle all this process.

iv) Role of Voice to Text Search

Since this facility supports only chrome so the JavaScript's which have been written will take the database from chrome to recognize the voice. JavaScript have been used for voice to text. The user needs MIC enabled systems from which user can give command as in query format once the voice has been recognized it will then covert it into text and then redirects it on the result page. This process will again hit Solr.

Solr Implementation Theoretical Background

There are certain thing which should be understood about implementation/Configuration of the Solr :

- **Synonym Filter :**
  Two filters have been created containing some common spelling mistakes and handling some countries which have short forms of their names. The mapping of the country names to their languages so if the user types Japanese it will fetch the documents with Japan.

- **Zone Scoring :**
  Here **zone weights** have been added while mounting them to SOLR. We set the zone weight for a match on the title field higher than the zone weight for a match on the text field. Zone weight is applied to give title the maximum weight.

- **Spell Check :**
  The spell check feature is file based. In this a new request handler have been created which is called spell. If we query against this handler it returns the results as well as spell suggestions.

- **Porter Stemmer Filter :**
  This filter applies the Porter Stemming Algorithm for English.

- **Shingle Filter Factory :**
  This filter constructs shingles, which are token n-grams, from the token stream. It combines runs of tokens into a single token. The number of tokens per single is 2 by default and we retain this default. Shingling is necessary to store indexes as Bi-grams which can later help in auto suggest feature.

Languages Used

REGEX :

This language have been used in order to remove the wiki-tags and XML-tags from the data when it is going for the parsing. Since links has very fix pattern so in order to match those patter REGEX is highly recommended apart from this to remove unnecessary tags and special characters from the text it will be very incumbersome if we implement code in pure java. So to avoid hassles and make things more simple REGEX have been used which actually match the pattern which have been written in the code and based on that it parse whole documents.

JAVA:

Java is very useful language in order to work with huge documents it makes work easier. Apart from this the datastructures of the java is very easy to implement in the code which is very tough to implement in other languages.

Also there are many advantages of the java, It will be important to mention while implementing code it is very easy to debug the code errors and the problem.

JSON :

Since all the parsing and other functionality of the code is in java so it will be better if we choose the other database language in java based platform only. So after mounting all the data in the Solr it gives option to convert the languages n many form so It is based on the user which languages they want to choose like " JSON, XML, PYTHON, RUBY, PHP and CSV" . So in this project JSON have been selected as it stored all the data into array format which makes things easier while fetching the data from the solr.

Python :

This language is very easy and compatible to implement. This has been implemented to count average documents once that documents have been parsed and converted into XML. This language have been used in order to get only location based documents rest had been ignored.

JavaScritps & JSP :

Uses of the javascripts as in front-end is highly recommended as it gives flexibility to the user to implement the code very easily also it can be integrated into UI. This languages had been used to fire the query and get the response from the Solr. As there are other alternatives like this process could have been done using Servlets with JSP but implementing servlets for this procedure could have made task more complicated. Because in this we could have made container to store the results and then accordingly convert it into HTML format after applying JSP it could have been displayed on the screen, there were many more complication in uses of the servlets, So the uses of javascritps with JSP was more flexible way.

In this project JavaScritps/JSP is working :

1. Firing query into Solr in JSON format.
2. Getting response into JSON format from Solr and converting into HTML format after applying JavaScript's/JSP and displaying it on the UI result page.

Since the query response is in JSON which are in java based platform so there must be some platform in the UI which can handle response of this type so the result page have been implemented in JSP. Which display result after getting it from the Javascripts DOM.

HTML & CSS :

This is used to design UI. This web development languages have been used in order to give good look feel of the UI.

AJAX :

This is for the autosuggestion purpose. Since in autosuggestion it is required that the process should be very quick enough to give the response. So this language had been used. When we insert the query in the search box that time for each query o character it hit Solr to get proper response of that query. So every time it hit solr. So this process should be fast enough to handle the query. In autosuggestion the process is basically based on the **"TITLE" using One-GRAM, BI-GRAM, TRI-GRAM.** The reason behind giving autosuggestion only

on Title using above mentioned features was it is not very easy task to implement the autosuggestion without using any algorithms. There are many algorithms in order to implement the full-fledged autosuggestion. At this moment the autosuggestion have been given on TITLE. But in future prospective the “ELASTIC” autosuggestion can be implemented.

Algorithms used in the Project

This is one of the most important part of the project since this give strong point to decide how the query will passed and how the query processing will be carried out. As there are many information retrieval based model like Boolean and probabilistic model but it depends on the approach which we take in order to accomplish to this project.

Algorithms used in the Query Processing :

In order to complete the project and get correct query output, based on the condition of the output result in this project BM25 algorithm is most suitable. In information retrieval, Okapi BM25 is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. It is based on the probabilistic retrieval framework. The BM25 is basically depends on the calculation of the TF-IDF in the documents. As in this project the term frequency (tf) and documents frequency maintained in order to rank the pages. BM25 is ranking algorithm of probabilistic models. In the solr when documents are been mounted they are stored in an array in JSON. But when the user request the query to the solr that time based on the query it calculate the terms appears in that particular documents. The more appearance of the terms in the documents the highest the score will be of that particular query.

For example: if the user typed "Mumbai" in the query and when this query will hit solr that time the query processing will check appearance or occurrences of the terms in the particular documents. Now if suppose there are two documents named as "Navi Mumbai" and "Mumbai" now let's take the document "Mumbai" have more occurrences of "Mumbai" terms compare to "Navi Mumbai" document. Then in this case the ranking of the documents will take place so here in this assumption "Mumbai" document will be placed on the rank one of the total documents.

Hence based on the score of the terms and their weight the scoring of the terms will take place.

Let's take another assumption :

The query "New York" which contains two terms will be passed. No relevance information (r and R are zero)
N = 500,000 documents
"New" occurs in 40,000 documents (n1= 40,000)
"York" occurs in 300 documents (n2= 300)
"New" occurs 15 times in doc (f1= 15)
"York" occurs 25 times (f2= 25)

Document length is 90% of the average length (dl/avdl= .9)
Typical TREC value for *k1* is 1.2, *k2* varies from 0 to 1000, b = 0.75
k1= 1.2, b = 0.75, and k2= 100
K = 1.2 !(0.25 + 0.75 !0.9) = 1.11

Formula:

$$\sum_{i \in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i}$$

Solution :

$$\begin{aligned} BM25(Q,D) \quad = \quad & \log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\ & \times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\ & + \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\ & \times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1} \end{aligned}$$

$$\begin{aligned} = \quad & \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\ & + \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\ = \quad & 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\ = \quad & 5.00 + 15.66 = 20.66 \end{aligned}$$

So based on the calculation procedure of the BM25 this model which made vital to make use of this model in to this project.

Algorithm used in Crawler :

These other kind of algorithms are used in focused crawlers to determine an optimal order in which the URLs are visited. Even though many different search algorithms have been tested in focused crawling, the two most popular ones are: Breadth-first Search and Best-first Search.

In this project **Best-first search** algorithm have been used since it is currently the most popular search algorithm used in focused crawlers. In best-first search, URLs are not simply visited in the order they are discovered; instead, some heuristics (usually results from Web analysis algorithms) are used to rank the URLs in the crawling queue and those that are considered more promising to point to relevant pages are visited first recursively. Non-promising URLs are put to the back of the queue where they rarely get a chance to be visited. Clearly, best-first search has advantages over breadth-first search because it probes only in directions where relevant pages locate and avoids visiting irrelevant pages.

The **Implementation Pseudo code** behind "BestFirst Search"

1. Define a list, OPEN, consisting solely of a single node, the start node, *s*.
2. IF the list is empty, return failure.
3. Remove from the list the node *n* with the best score (the node where *f* is the minimum), and move it to a list, CLOSED.
4. Expand node *n*.
5. IF any successor to *n* is the goal node, return success and the solution (by tracing the path from the goal node to *s*).
6. FOR each successor node:
    a) Apply the evaluation function, *f*, to the node.
    b) IF the node has not been in either list, add it to OPEN.
7. Looping structure by sending the algorithm back to the second step.

The crawler's algorithm works based on this algorithm recursively to crawl the title and URL of that title or vice-versa.

Objectives, Purpose and Scope of Proposed System

Objectives:

- The main objective behind this project is to provide results from multiple sources using static and dynamic data.
- It is mainly useful for the people who are mostly interested to find out information about the location.
- To provide flexibility to the user to retrieve the results from multiple sources as well as from crawler.
- To provide user with a best searched single link based on the highest score.
- Simplify information search and cut the time by pointing to the most relevant source documents.
- Lazy users or motor impaired can use voice input to search for data.

Purposes/Scope of this project are as follows:

- To retrieve the results from multiple sources of any given query.
- To integrate data from multiple internal and external sources.
- Parse fairly involved Wikipedia markup.
- Index a descent sized subset of the Wikipedia corpus.
- Create multiple indexes on the page data as well as metadata.
- Provide an index introspection mechanism that can be later built on support queries.
- To crawl the data (URL, TITLE) from 3 giant search engines i.e. Yahoo, Bing, Wikipedia and parsing those data through parser and mount those data into solr. In this process the crawler can get any N number of documents. After getting this documents the number of results are stored and maintained into XML format which contains these data into documents pages.
- Getting familiar with world widely used strongest open source database system i.e. Solr.
- Experimenting with own search engine at distributed scale level.
- Familiarizing with query processing and implementing new system based on the predefined algorithms.

Chapter 3
System Analysis

Requirement Gathering

As being the sole responsible developer of this project the brainstorming session have been conducted with various search engines and experimented with their performance query processing, algorithms applied on query, their response time etc. After this session we had come up with different needs of the users.

Like :

- There should be the best results of the query which should be given to the users.
- Secondly user should get the flexibility to compare all results from all different sources or search engines.
- Still the companies are required to change query processing style like they can give the suggestion or rating of the retrieved results.
- They should give suggestion that why user should select only those URL or topics.
- Companies are still researching on this particular topics.

Keeping those views in minds, our platform offers following features:

- Easy to use interface
- Less complexity of tabs
- Easy maintenance of pages and results.
- Comparing the results from all other search engine and showing them to the users.

These are the features that will help users take less time understanding the features. Henceforth for the same several session had been kept to analyze requirements of the project as well as the processing of the project. In order to do that it is very important to analyze the Hardware as well as software system

requirements also the data sizes, query processing and algorithms which will be used in the project.

So it is very important ti list down the Software & Hardware requirements.

1. Software Requirement Specification:

- **Eclipse :** It is very powerful developer tools , in this project this has been used to implement the parsing code and mounting code which will mount the parsed data on the solr.
- **Geany :** this is used to develop HTML and CSS code in UBUNTU.
- **MySQL :** This is the database which is used to store all the values of the cities and countries which will later be used to show the markers on the MAP.
- **Chrome :** This is the browser which will be used for Voice to Text conversion since the implemented code will access the database of the chrome to recognize the English language also this browser will be used to access the Solr open data source database on localhost:8080.
- **Python :** This is the developer tool of the python. This tool have been used to implement the python code.
- **Tomcat Server :** This is the main server through which the user fires the query through UI to solr. Since Solr and UI have been deployed inside the tomcat.
- **Ubuntu :** This is the operating system which is required as this will make works more easy in order to configure this whole project.
- **Solr :** The solr should be deployed inside the tomcat in order to work on the server else the user cannot fire any query because user will fire query on port 8080 because the UI is deployed inside tomcat and solr will run on different port which is it's default port number 8983.

2. Hardware Requirement Specification :

- At least 6GB of storage memory
- Core 2 Processor
- RAM 2GB
- Internet Access
- MIC Enabled Laptop

Feasibility Study

The feasibility study proposes one or more conceptual solutions to the problem set for the project. The objective in assessing feasibility is to determine whether a development project has a reasonable chance of success. The following are the criteria that are considered to confirm project feasibility:

- **Technical Feasibility:**

At first it is necessary to check whether the proposed system is technically feasible or not and to determine the technologies and skills necessary to carry out the implementation of the system. If they are not available, then find the solution to obtain them.

- **Operational Feasibility:**

The system has been specifically designed to be user friendly so that the system could be easy to use also the system has been designed to mirror the real world scenario so that the process flow can be better understood by the other end users. They have in depth knowledge of the entire process in the real world. Hence migrating to the system does not pose much hindrance.

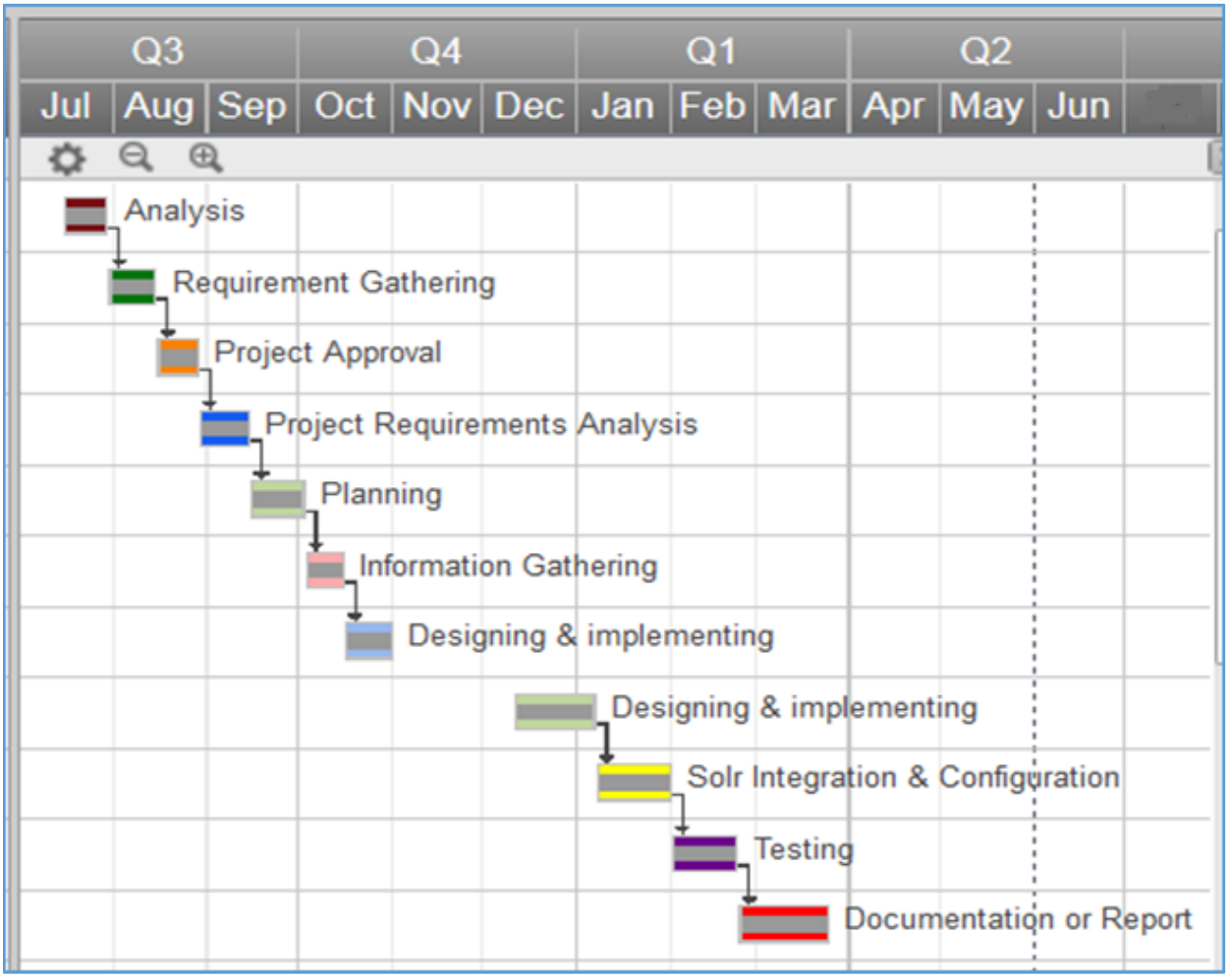
- **Economic Feasibility:**

The system is implemented based on the low cost so the user should not be worry about the cost of the system. Apart from that it is very handy to use.

Chapter 4
Project Planning
&
Description

Gantt Chart

	Task Name	Duration	Start	Finish	Predecessors	Assigned To	% Complete	Status
1	Analysis	15	07/15/13	07/29/13		Analysis	100%	Complete
2	Requirement Gathering	16	07/30/13	08/14/13	1	Requirement Gathering	100%	Complete
3	Project Approval	14	08/15/13	08/28/13	2	Project Approval	100%	Complete
4	Project Requirements Analysis	17	08/29/13	09/14/13	3	Project Requirements Analysis	100%	Complete
5	Planning	18	09/15/13	10/02/13	4	Planning	100%	Complete
6	Information Gathering	13	10/03/13	10/15/13	5	Information Gathering	100%	Complete
7	Designing & implementing	16	10/16/13	10/31/13	6	Designing & implementing	100%	Complete
8	Designing & implementing	27	12/11/13	01/06/14		Designing & implementing	100%	Complete
9	Solr Integration & Configuration	25	01/07/14	01/31/14	8	Solr Integration & Configuration	100%	Complete
10	Testing	22	02/01/14	02/22/14	9	Testing	100%	Complete
11	Documentation or Report	30	02/23/14	03/24/14	10	Documentation or Report	100%	Complete



Risk Analysis

In risk analysis it should be very important to mention the risk scenario which had been aroused while implementing this project.

- **Size of Documents :**

As the size of the static document was very large so it was very important and complicated task to break that into smaller chunks. In order to do that the evaluation and deep study was required to of each documents. Since this was the main step towards the implementation else the parsing of the documents could have been taken into picture. In order to do that it was crucial task to first calculate the number of the documents presents into the compressed data. Since the compressed data could not be open into the system due to its overwhelming size so the python code implemented which actually counts all the present data and take its average count into consideration.

After getting this data then the factor arises when it was the time to break that huge data into smaller data file. In order to do that another python code have been written which takes this huge data file and based on the fields declared (TITLE, ID, iLink, eLink, TEXT) it parsed whole files and taken only these data based on the location defined into the txt file. This code matches the location which are defined into the txt file with the data file. Matched data based on the above fields will be sorted out. After it finishes the parsing it generates new XML data file which only contains extracted data. So it reduces the size of the whole data file also most of the data have been taken and parsed from several web sources.

- **Processing & Parsing :**

This is another risk analysis related to the data and processing the data into in order to get the result in proper format. After getting the final data file (into XML format) the eclipse parsing code will parse these data and mount those data on the solr but while it is parsing the developer should have made sure that the process should not stop in between else the data can be corrupted or it will not be mounted on the solr. User should make sure that they are mounting genuine data file to parse else the process could not be completed.

- **Validation and Verification :**

This is the very important step towards the output of the results, the main problem can be arises when solr is giving response. So for the sake of simplicity the solr configuration have been taken into consideration. Before mounting the data on the solr the several schema and schema configuration have been written so that solr can response in the way user is expecting. There could have been many scenario when the output could be have been given the solr in not expected way for instance if user typed “Mumbai” then he could have got the results as “Navi Mumbai” or “Maharashta” so this is the problem with the synonyms filter of the solr in order to make correction into this the synonyms of the several places have been written which can handled the case mentioned above as well as case-sensitiveness of the query.

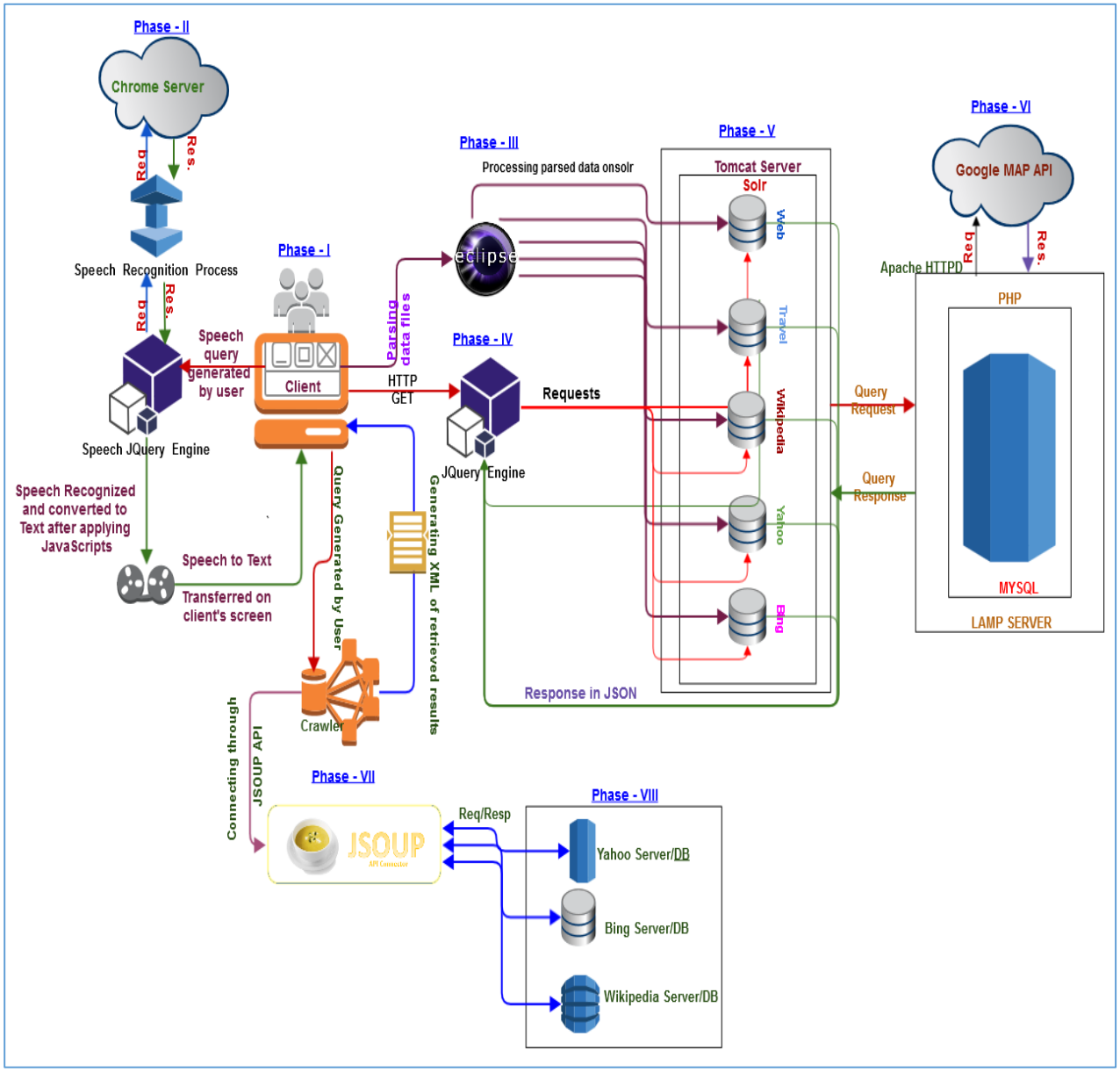
There had been many other validation given like making URL from the TITLE and getting that implemented URL into the Web. So in order to make this URL it was required that it should in the URL format so for the same URL matching code have been implemented.

- **Query Processing Speed :**

In any search engine this step is very important that the query request and response should be as minimum as it can be else user will not feel to get into it. In order to handle this process there are two possibilities which can be taken into consideration :

- Implementing query processing handler into the solr.
- Removing the bugs from the implemented code.
- Reducing the query request time interval duration.
- Getting the response from the Solr as soon as the request hit the Solr.

Architecture of the Project



- Solr will be deployed in a Tomcat server.
- There are separate cores for Web and Travel.
- User will be presented with a web based interface.
- UI will be hosted by Tomcat as well.
- UI is built on JSP architecture and I have client side JavaScript.
- User will type a free text query and submit the form.
- JQuery will convert the query to Solr query syntax and wait for the HTTP response.
- Solr will reply in JSON format.

- JQuery script will parse the JSON and make object from the result response.
- JavaScript will render the results page in HTML format.
- The MAP UI which will show countries and major cities around the world.
- User will start search by simply clicking the location on the Map.
- Google maps API has been used to show the location of countries and major cities.
- JavaScripts have been applied on the received/recognized voice and based on that it will convert it into text. In this project it will make use of the chrome database to recognize the English language.
- The Crawler will be stand-alone application will run separately in user machine and it is responsible for crawling the data from the Yahoo, Wikipedia, Bing search engine based on the query given on mentioned level.
- In order to connect with the multiple servers JSOUP API have been used which parse the query and crawl the pages accordingly.
- All the data requests/response will pass through API only.
- After getting this data into the crawler the next step is converting those data into XML format with corresponding data (Id, title, url) and send it for parsing purposes.
- After converting this data into XML format it will go for parsing and then mounted on the solr. From solr again same process will followed up and it will be given on the UI.
- At a time based on the level of the search the crawler can crawl "N" number of links and title.
- Apart from the above it also maintain history of entered text which can be used by the ADMIN in order to see what query have been passed and to check whether that data is available or not in the databases. Based on this ADMIN crawls those data and mount it into the solr.
- Crawler have 2 levels to search which defines at what level user want to search the query.
- Many time it will happen that crawler will take bit long time to crawl pages if the approximate or average crawled links are above 1000, the reason behind this is that crawler maps all the links into its queue and take each links one by one to crawl web pages to get the title of that links.
- May times there could be a possibility that links are redirected from many pages so in that case crawler will simple discard this link to get the title.
- Also in order to make an exceptions for not to crawl home pages of the servers the multiple filters have been used.

Pre-Parsing

As the theme of the multi-core search is **location based**. Python scripts have been implemented to extract the information about the places. These places are stored in places.txt file. If the document was location related it was extracted and made into and XML document.

Places.txt file has been manually created by fetching the data from the cities and countries list in from the Geonames website. **http://www.geonames.org/**

Another problem was that the data files were too large to extract. So the Python scripts have been implemented to parse these data files. This python scripts decompress 10 MB of binary data at time in memory from these data file using bz2 decompression and python libraries. It also checks whether the documents mention the name of the place is there or not. If the document mentions the name of the place in the title or the first paragraph then it extract the data into proper XML format. Using this technique the size reduction have been taken place.

a) Web

i) Original Size: 120 GB +
ii) Reduce Size: 800 MB

b) Travel

i) Original Size: 943 MB
ii) Reduced Size: 53 MB

This was a **crucial step** in the extraction of the documents related to the theme of the project and based on the **places.txt** file to plot the location in the MAP UI and the run spell-checks against the file. MAP UI is functioning fully different with regards to the other function of the UI.

Solr Schema

The first step was to choose an appropriate schema for SOLR. Following fields for SOLR schema have been chosen :

- **Title:** This is displayed to the user when a query returns the document
- **URL of the page:** This is displayed to the user when a query returns a document. URL's are valid Wikipedia URL's derived from the title.
- **Text:** This field contains the entire Wikipedia Text.
- **All internal links in a wiki page:** These are Wikipedia links that are referenced within a Wikipedia document.
- **All external links in a wiki page:** These are non-Wikipedia links that direct us to external sources to get related information.
- **Document ID:** is a unique field for each document which is auto incrementing integer.
- **Timestamp :** is the unique date and time of the documents, this time is depends on the upgradation of the data or the academic evaluation of the data.

The next step is to identify what fields to index and needs to be just stored. So in this project just store the URL, Internal Links and External Links have been stored and the index of the Title and text have been taken into consideration.

Standard Tokenizer Factory: This tokenizer splits the text field into tokens, treating whitespace and punctuation as delimiters. Delimiter characters are discarded, with the following exceptions:

- Periods (dots) that are not followed by whitespace are kept as part of the token, including Internet domain names.
- Words are split at hyphens, unless there is a number in the word, in which case the token is not split and the numbers and hyphen(s) are preserved.
- The "@" character is among the set of token-splitting punctuation, so email addresses are not preserved as single tokens.

StopFilterFactory: This filter discards, or *stops* analysis of, tokens that are on the given stop words list. A standard stop words list is included in the Solr config directory, named stopwords.txt, which is appropriate for typical English language text.

LowerCaseFilter: Converts any uppercase letters in a token to the equivalent lowercase token. All other characters are left unchanged.

PorterStemmerFilter: This filter applies the Porter Stemming Algorithm for English.

ShingleFilterFactory: This filter constructs shingles, which are token n-grams, from the token stream. It combines runs of tokens into a single token. The number of tokens per single is 2 by default and we retain this default. **Shingling is necessary to store indexes as Bi-grams which can later help in auto suggest feature**

SynonmynFilter: Two filters have been created containing some common spelling mistakes and handling some countries which have short forms of their names also the mapping of the country with respect to their names languages so if the use types Japanese it will fetch the documents with Japan.

Set the default Solr Query parser to "AND"

Set omitNorms=false for text and title fields to allow for Zone scoring.

Data files Indexing

a) Web data file Processing

Web index contains pages of different places around the world. It also contains pages about topics related to those places, i.e., the names of places occur in the context of the page. It may include pages of people, monuments, politics, architecture etc. significantly related to the corresponding place.

Web Data file enwiki-pages-meta-current.xml.bz2 whose size is 19.4 GB when it is still compressed was downloaded from http://data files.wikimedia.org/enwiki/

Since search engine is location centric, this theme which makes search results related, the data files which have been downloaded was very huge in size which can be parsed In local system to the implemented python code parse these data files based on the cities and countries into smaller size. **Pages which are redirects are omitted.**

A text file with major cities and countries around the globe is fetched from www.geonames.org. This text file is loaded to a hash table. Then parser have parsed each Wikipedia page at a time and check to see if the title of that page matches as a pattern with any entry in the hash table. If yes, then parse that document as a valid location related page and add it to Solr. Else it will move on with the next document. Finally Solr contains documents which contain pages of major places around the world.

The Wikipedia pages are then processed and are added to Solr by means of existing SOLR api using JAVA. The text and title get appropriately indexed as mentioned in the schema file. The parser have parsed each Wikipedia into fields declared in the schema and add them.

The **zone weights** have been set while adding them to SOLR. This zone weight is set for a match on the title field higher than the zone weight for a match on the text field. **Only location based pages have been extracted which are specified by the places.txt file.**

- **How Web data file has been parsed?**

Python Script **web.py** is used to first process this huge data file in chunks and extract only those documents whose title contained a location in the world. For this, places.txt file had been used which is a list of all countries and major cities in the world. This python scripts now results in a smaller data file which contains just pages related to locations.

This smaller data file is now read by the code written in java and is processed page by page by identifying the start and end page tags. Wikipedia markup, templates are parsed and removed then retain just the text associated with a page and index this text using Solr using the Tokenizers and filters specified in the schema for Wikipedia core.

Here, the parser have parsed –

- Page Title
- List Items
- Text Formatting
- Accent Characters
- Tag Formatting
- Internal Links
- External Links
- Templates

In addition to title and text also capture all the internal links referenced by the page and store in a field in Solr. This is not indexed. Similarly, external links referenced by the page are also stored in Solr but nor indexed

Each page as it is processed is added to Solr before proceeding with the next page. SOLRJ will add documents. When adding each field, it is important to provide an index time boost and give a large zone weight to the title weight. This requires the omitNorms to be set to False in the Wikipedia schema. The code is great as it avoids reading the huge data file which contains documents in millions but still be able to fetch all those locations from this huge data file to create a smaller data file which can then be parsed. Python code developed to read huge data file in chunks efficiently does this. After the final filtering of all documents, the total documents left with **2970** documents out of **115051** documents present in the data file initially.

b) Travel data file Processing

Travel data file index contains pages giving travel (voyage) information about the place. It may include hot destinations, popular sites, and places to be visited etc. at the place searched for.

- **How Travel data file has been parsed?**

As the Travel data file had over 100,000 documents and many of those documents are not relevant. Again *places.txt* have been used in process to reduce the size of the data file and only extract the documents pertaining to the locations. These locations include countries and major cities around the world. The result is an extracted XML file of 53 MB. All the documents in the XML file are related to locations of our interest. Now the Java code had fetched the documents from the XML data file and add to the index. Travel data file has documents which are incomplete and don't have a lot of text. So the keeping the blind believe that these documents are irrelevant so it will be necessary not to add them to Solr index. So after running the *avglength.py* script to find out the average length of the documents. This number to be found was 1600 for Travel data file. As to discard documents which are very small it was recommended to remove the documents which are smaller than the average length. Pages with redirects were also discarded. SolrJ will add these documents to index and did a commit.

Adding the title, URL and text to the index. This is as per the solr Schema. Internal links and external links also have added in hopes to implement PageRank on the documents but as the project implementation have ran out of time But the text was processed to remove the Travel markup. This markup was removed by *WikiNewsParser.java.*

Here, the parser have parsed –

- Section Headings
- Addresses
- Wikivoyage/Travel Links
- Internal Links other Wiki's
- Infoboxes with Addresses
- Whitespace Formatting
- Accent Characters

- Tag Formatting
- Templates

The aim was to normalize the text so that it can highlight appears text on the GUI which can be read by the user.

Text formatting removes

- Special Characters
- Accents
- Normalizes the whitespace

Addresses Infobox format removes the labels but keeps text:

- {{| name=<text> | add=<text>}}

Tag format contains

- *Comments(<<text>>)*
- *&mdash*
- *&ndash*
- *Quotation (")*
- *&amp*
- *
 and
 tags*
- *<hr /> tags*

Links refer to the links present in the text, internal as well as external. These appear in following formats –

[[<text>]]
[[link |<text>]]
[[File:<text> | text]]
[[Category: <text>]]
[http://link <text>]

After the final filtering of all documents, the total documents left with **2970** documents out of **115051** documents present in the data file initially.

Map UI- Search

Google maps are used for the purpose of displaying world map to the user. Few major cities and counties in the world are indicated on the map as per their latitude and longitude information. For this manually created list of places from **http://en.wikipedia.org/wiki/Latitude_and_longitude_of_cities** and parsed the latitude and longitude information which a google map query can understand.

- **How MAP user interface was created?**

a) Latitude and Longitude information was downloaded and saved as CSV file.
b) LAMP stack was setup. (Linux, Apache Httpd, MySql, PHP)
c) Data was loaded in table in MySql.
d) PHP is used to created XML file by accessing the MySql.
e) XML file is read by Javascript to plot the locations.
f) Google Maps API is used to display all the locations.

The list of places contained on the map is just a subset of the many famous locations that are indexed. To avoid crowding of points on map, it will be wise decision to choose a small subset. User can always make use of the search field to query on the places of interest that are not highlighted on the map.

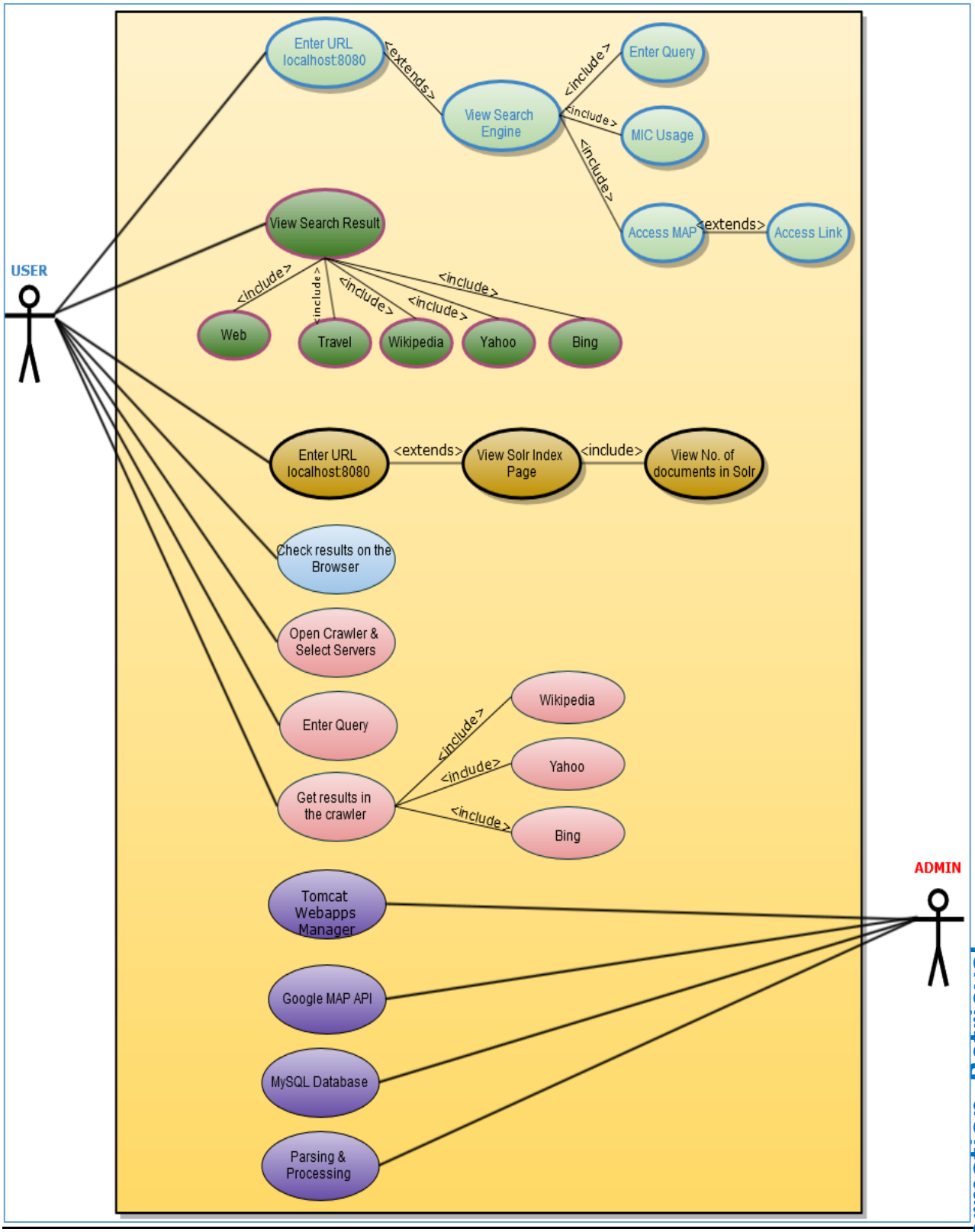
The RED location show major cities and the BLUE locations show countries. The coordinates for the countries are calculated by taking the median of the cities in that country. There are some exceptions to this rule. Like Chile, Libya and Russia which had been manually corrected the entries for these countries in the MySql database but usually the median method works well.

- **How MAP user interface does search?**

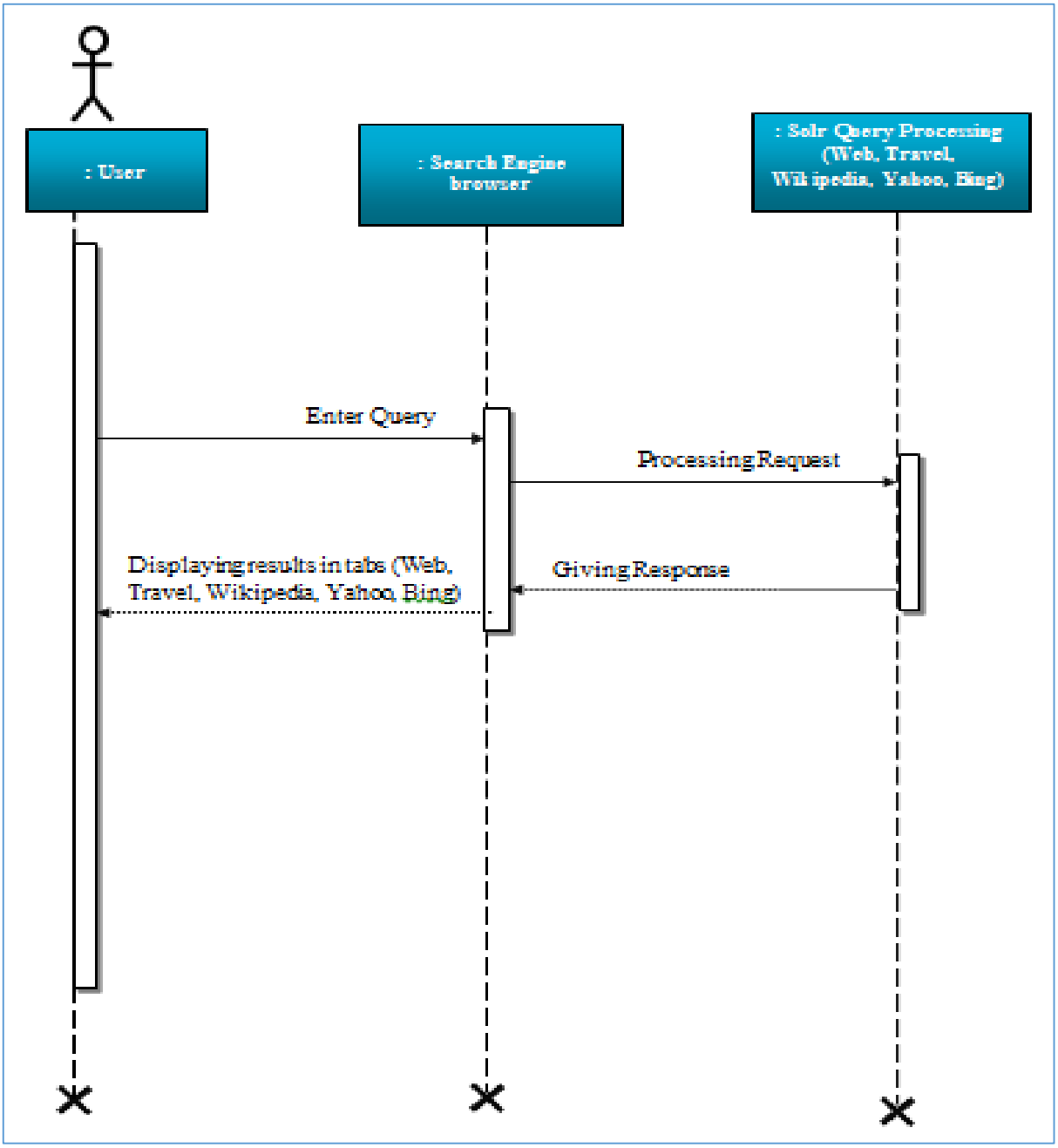
a) The MAP user interface opens a mini-html page on click of a location.
b) The location is labeled as country or city.
c) The mini-html pop-up provides and hidden form.
d) When clicked; it sends HTTP GET on the original user interface.
e) The default user interface then does the search and shows the results.

Chapter 5
System Design

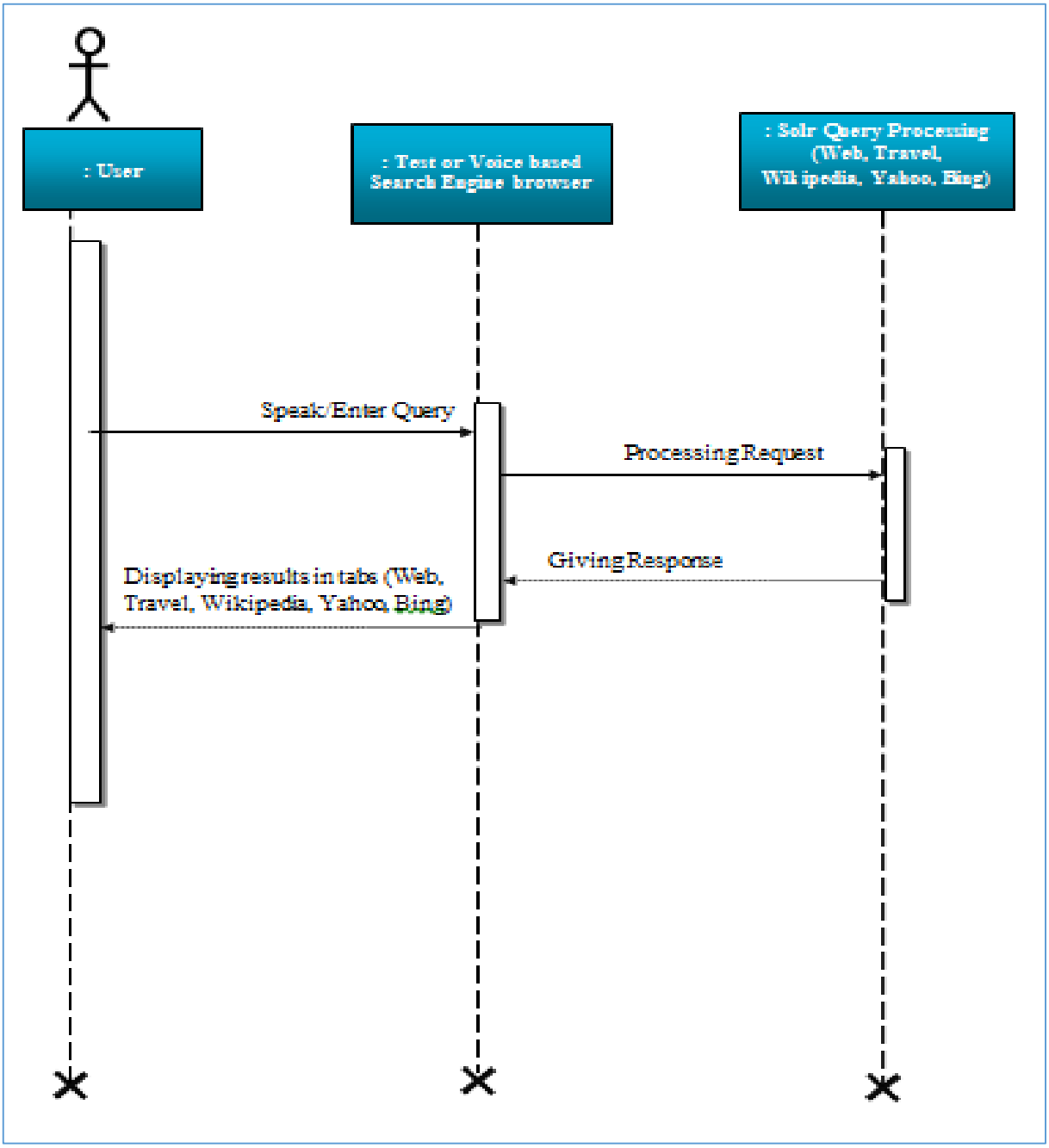
Use Case Diagram for ADMIN & USER



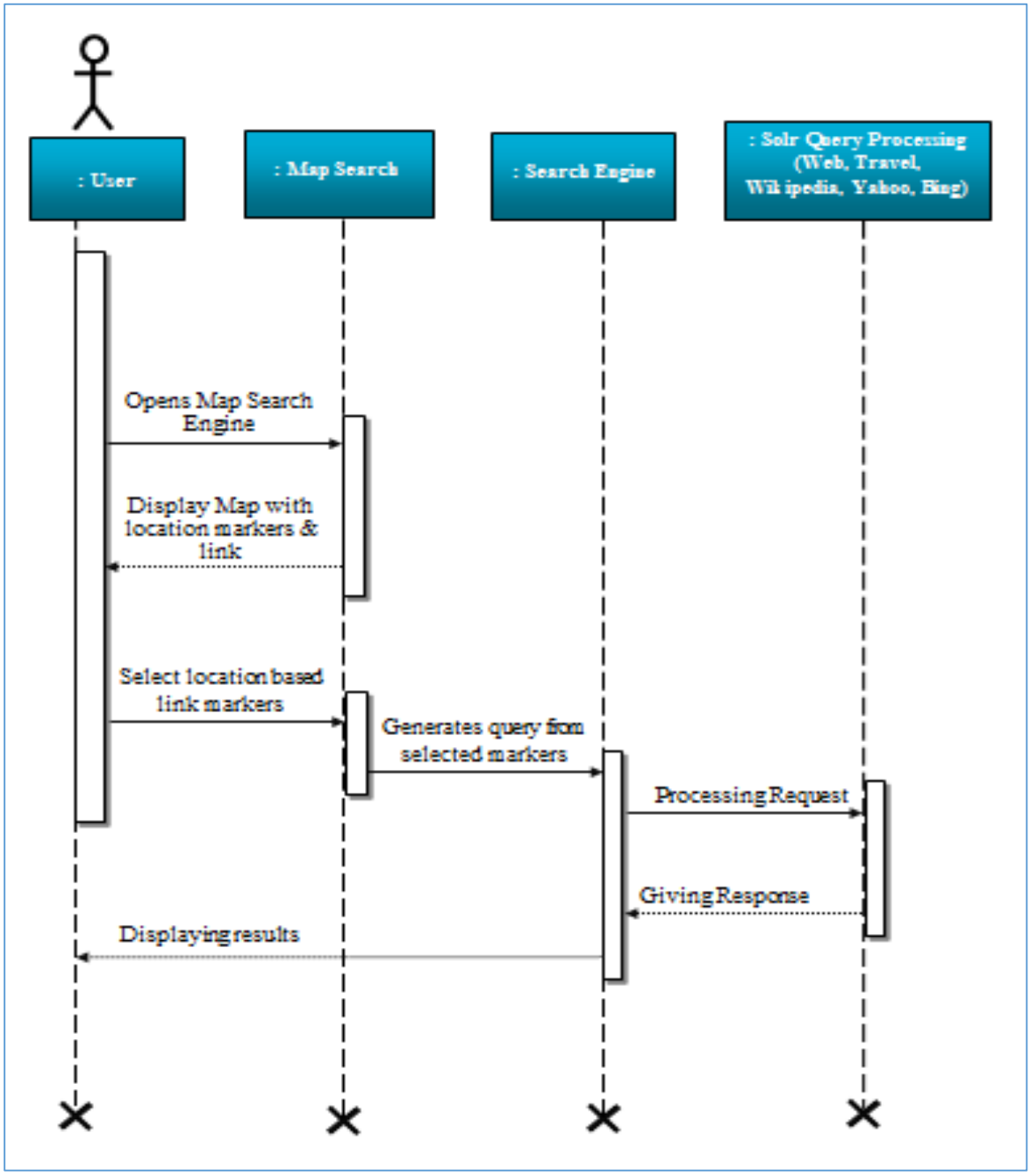
Sequence Diagram for Text Search



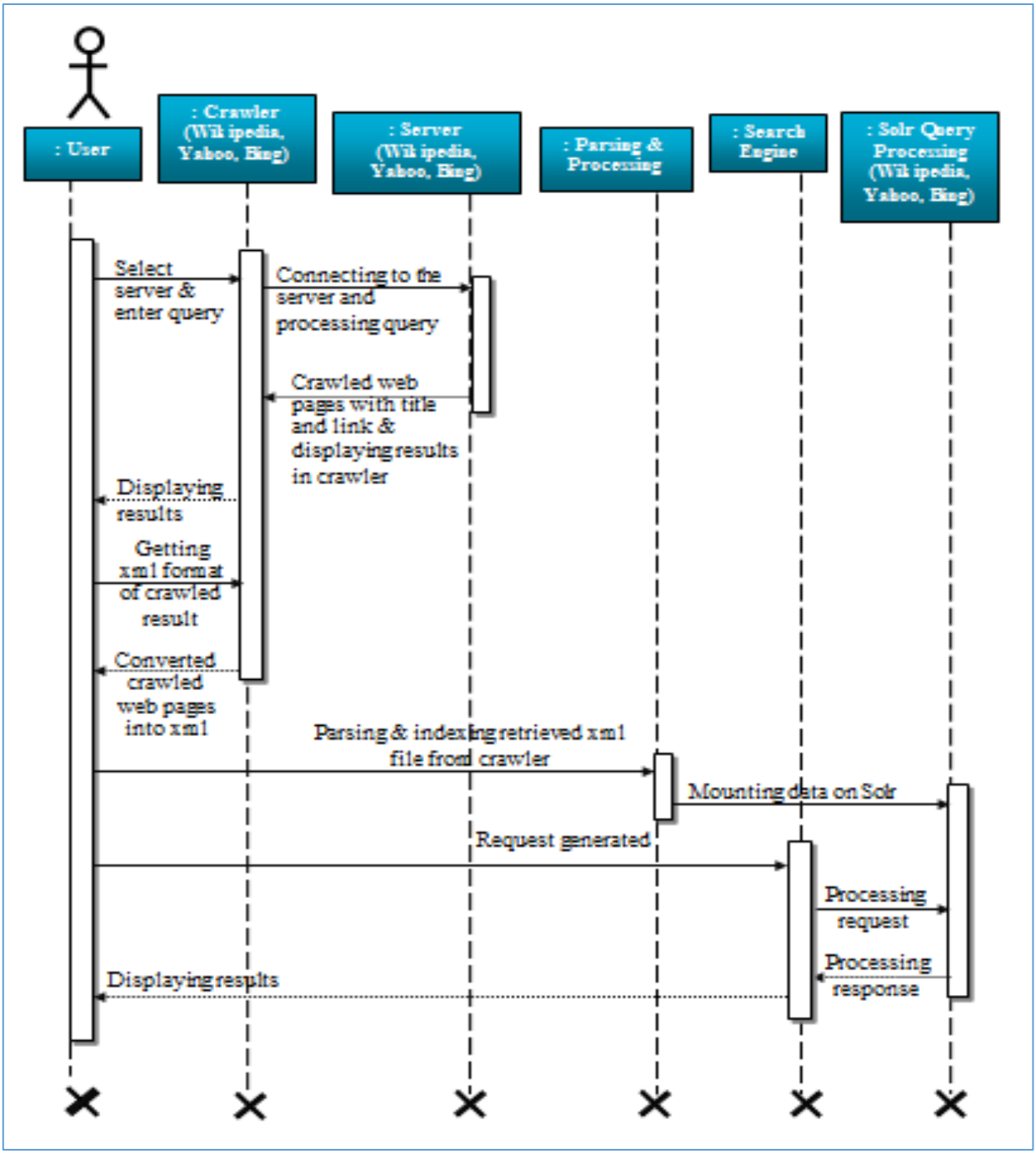
Sequence Diagram for Voice Search



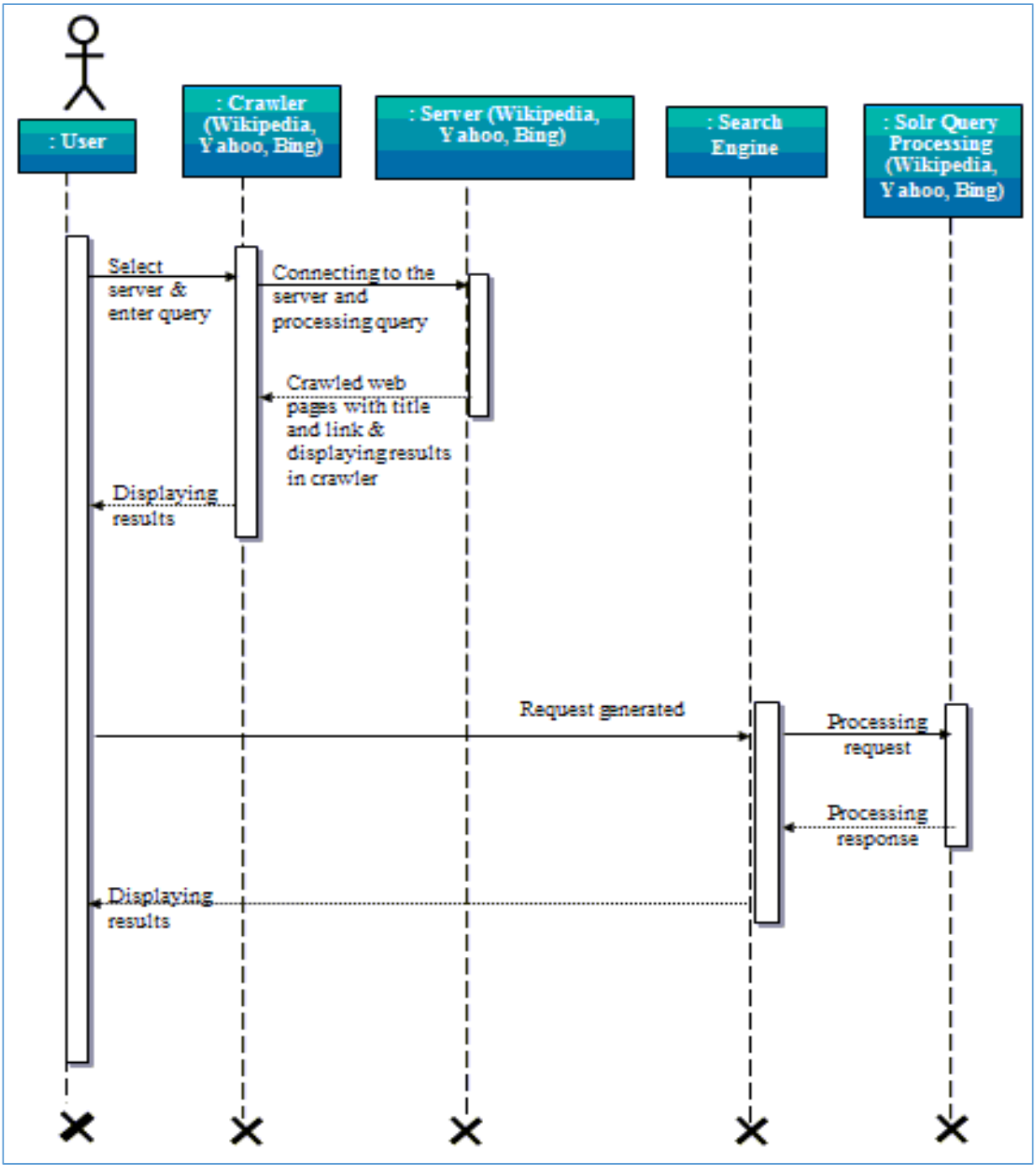
Sequence Diagram for Map Search



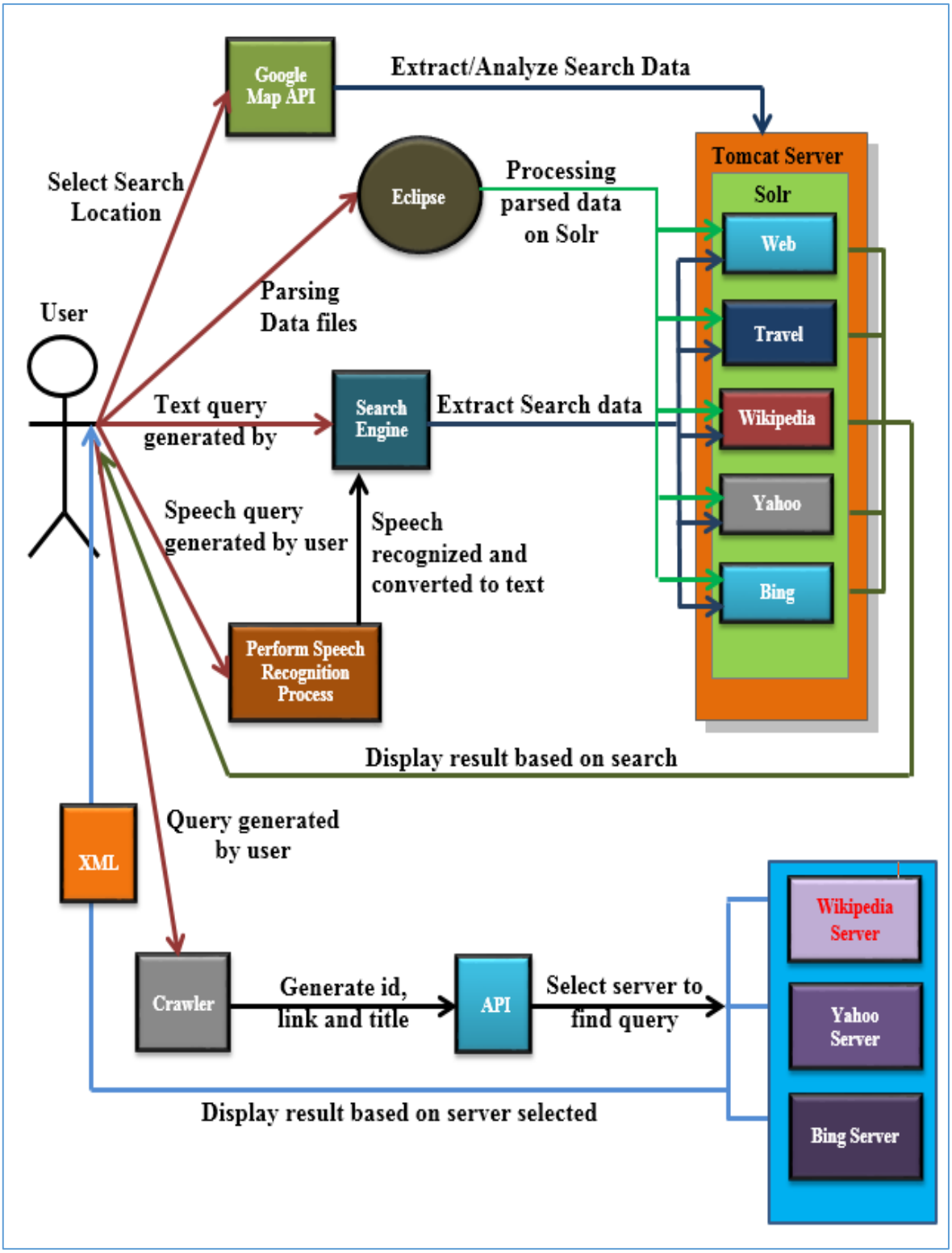
ADMIN Sequence Diagram for Crawler



USER Sequence Diagram for Crawler



Data Flow Design



Chapter 6
Codes
&
Screenshots

Project Codes & UI Screenshots

1. Eclipse Processing

IndexSolr.java

```
package launch;

import wikinews.WikiNewsRunner;
import wikipedia.WikipediaRunner;
import wikivoyage.WikiVoyageRunner;
import wikipediacrawler.wikipediacrawlerRunner;
import yahoo.yahooRunner;
import bing.bingRunner;

public class IndexSolr {

    public static void main(String[] args) {

        WikiNewsRunner.starter();
        System.out.println("Wikinews Indexed.\n");
        WikiVoyageRunner.starter();
        System.out.println("WikiVoyage Indexed.\n");
        WikipediaRunner.starter();
        System.out.println("Wikipedia Indexed.\n");

//============================ Crawler Processing ==========

        bingRunner.starter();
        System.out.println("Bing Indexed.\n");
        yahooRunner.starter();
        System.out.println("Yahoo! Indexed.\n");
        wikipediacrawlerRunner.starter();
        System.out.println("Wikipedia-Crawler Indexed.\n");


        System.out.println("Done!");
    }

}
```

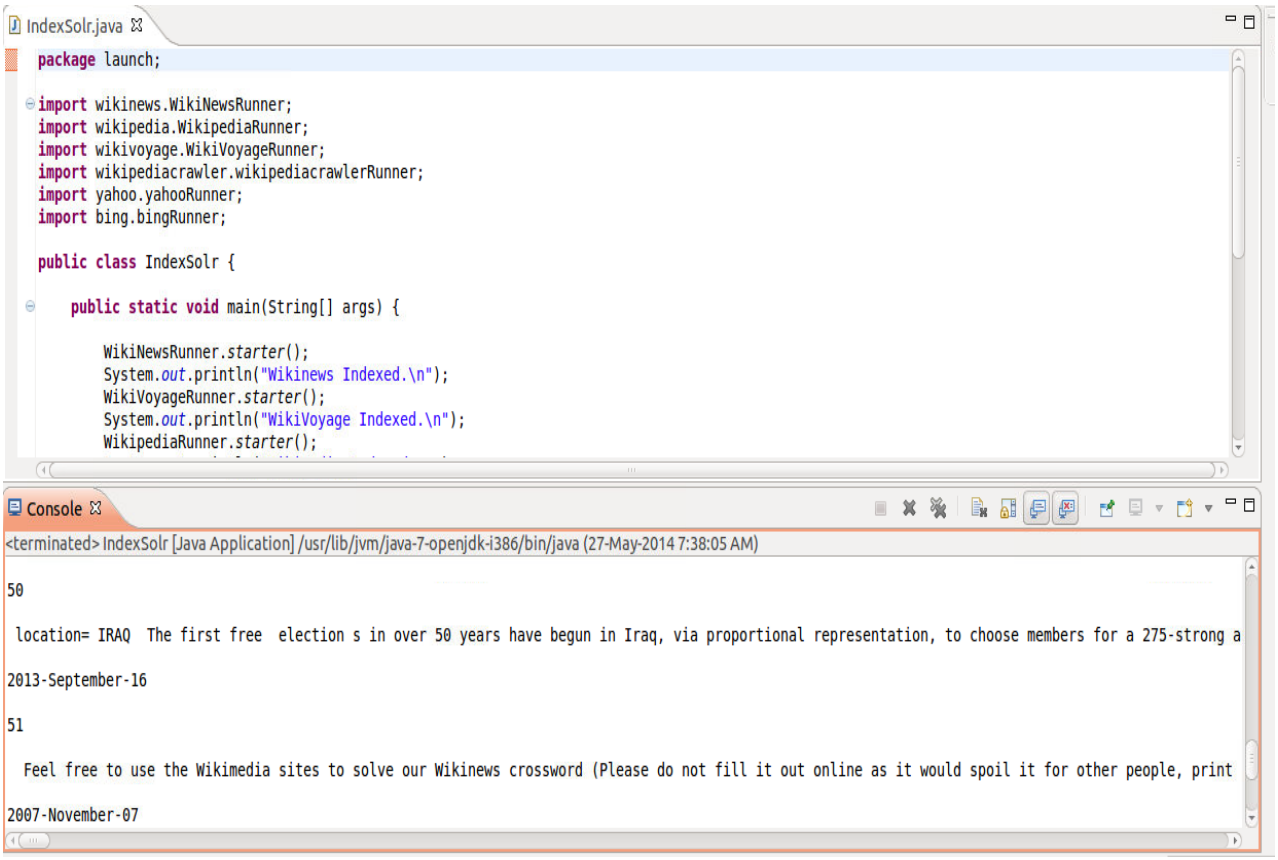
This code file runs all runner of the cores and mounts the data on the solr.

SolrAdd.java

```
package bing;


import java.io.IOException;


public class SolrAdd {

    public SolrAdd() {}

    public static void addition(HashMap<String,String> solrdoc){

        String urlString = "http://localhost:8080/solr/bing";
        SolrServer solr = new HttpSolrServer(urlString);
        SolrInputDocument document = new SolrInputDocument();
        document.addField("id", solrdoc.get("id"));
        document.addField("title", solrdoc.get("title"), 20.0f);
        document.addField("iLink", solrdoc.get("iLink"));
        try {
            solr.add(document);
            solr.commit();
        } catch (SolrServerException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

This is code file is responsible for adding all the parsed on the solr in the specific cores.

Eclipse Indexing Process



The parsing processing have started and the data is being mounted on the solr simultaneously.

Python Preprocessing Codes

1> Web.py

```
#######################################################################
#   Make a XML file out of Wikipedia Dump without extracting          #
#   Include only countries and major cities                           #
#   Very Strict Matching only the mentioned cities & countries        #
#######################################################################

import sys
import os
import re
import bz2

# Set of places to Match
places_set = set(place.strip().lower() for place in open('/home/d01//Desktop/places.txt'))

adder = 0

filename = 'wikipedia-051105-preprocessed.tar.bz2'
newFile = open('/home/d01/Desktop' + 'wikidump.xml',"a")
newFile.write('<?xml version="1.0" encoding="UTF-8"?>\n')
newFile.flush()
for line in bz2.BZ2File(filename, 'rb', 10000000):
    if ( line.find('<title>') != -1):
        title = re.sub(r'<[^>]+>', '', line)
        if ( title in places_set ):
            adder = 1
            newFile.write('<page>\n')
            newFile.flush()
    if ( adder == 1 and line.find('</page>') == -1):
        newFile.write(line)
        newFile.flush()
    if ( adder == 1 and line.find('</page>') != -1):
        newFile.write(line)
        newFile.flush()
        adder = 0
newFile.close()
print 'Successful'
```

This is Web python code responsible for decompressing compressed data and extracting the relevant required data base don the place.txt file.

2> Travel.py

```
#####################################################################
#   Make a XML file out of WikiVoyage data file without extracting  #
#   Include only countries and major cities                         #
#   Moderately strict matching only titles                          #
#####################################################################

import sys
import os
import re
import bz2

# Set of places to Match
places_set = set(place.rstrip().lower() for place in open('places.txt'))

adder = 0

filename = 'enwikivoyage-pages-meta-current.xml.bz2'
newFile = open('/home/d01/Desktop' + 'wikivoyage.xml',"a")
newFile.write('<?xml version="1.0" encoding="UTF-8"?>\n')
newFile.write('<wikimedia>\n')
newFile.flush()
for line in bz2.BZ2File(filename, 'rb', 10000000):
    if ( line.find('<title>') != -1):
        title = re.sub(r'<[^>]+>', '', line)
        title = title.lower()
        list_title = title.split()
        for word in list_title:
            if ( word in places_set ):
                adder = 1
                newFile.write('<page>\n')
                newFile.flush()
                break
    if ( adder == 1 and line.find('</page>') == -1):
        newFile.write(line)
        newFile.flush()
    if ( adder == 1 and line.find('</page>') != -1):
        newFile.write(line)
        newFile.flush()
        adder = 0
newFile.write('</wikimedia>\n')
newFile.flush()
newFile.close()
print 'Successful'

```

This is Travel python code responsible for decompressing compressed data and extracting the relevant required data base don the place.txt file.

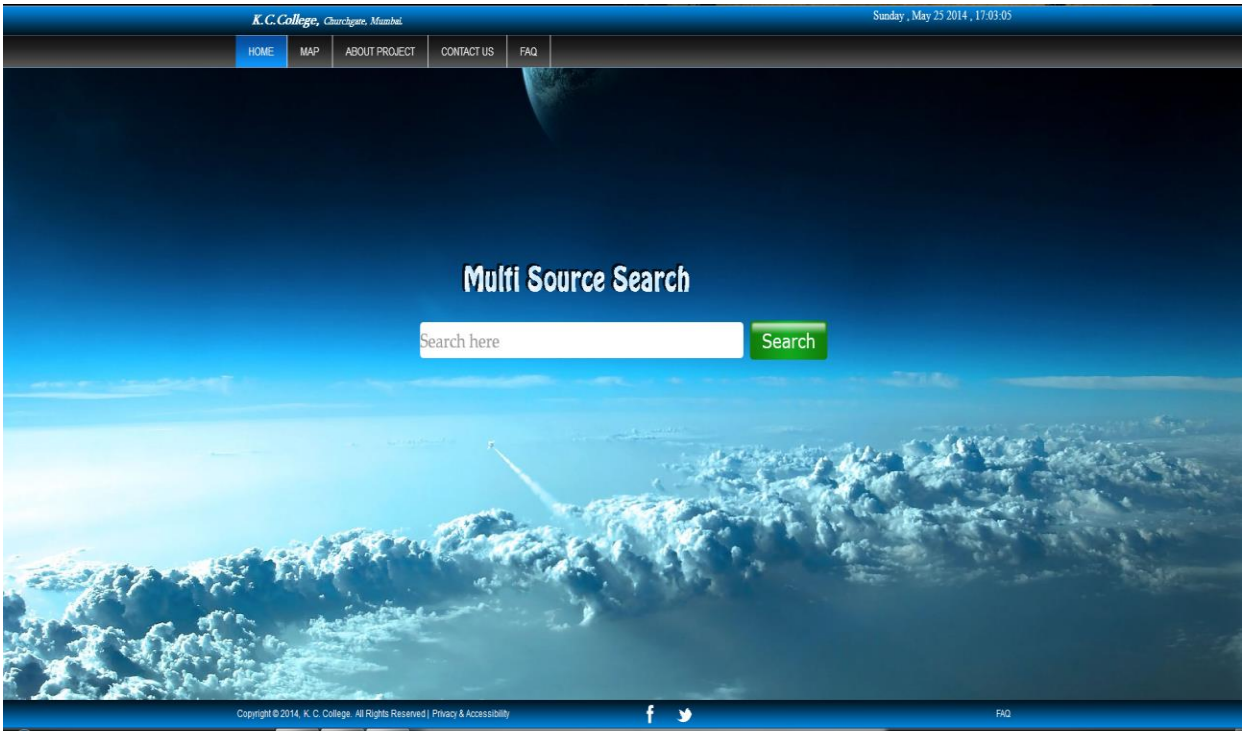
3> Avglength.py

```
#################################################################
#   Analyse Wiki XML data file and find out                     #
#   The average size of text                                    #
#   Median value of text size                                   #
#   These numbers help in getting rid of small documents        #
#################################################################

import sys
import os
import re

size = 0
count = 0
adder = 0
str = ''
median = []
redalert = 0
filename = 'wikinews.xml'

for line in open(filename, 'r'):
    if ( line.find('<redirect') != -1):
        redalert = 1
    if ( line.find('<text') != -1 and redalert == 0):
        adder = 1
        str = str + line
        count = count + 1
    if ( adder == 1 and line.find('</text>') == -1):
        str = str + line
    if ( adder == 1 and line.find('</text>') != -1):
        str = str + line
        adder = 0
        size = size + str.__len__()
        median.append(str.__len__())
        str = ''
        redalert = 0
    if ( line.find('</text>') != -1 ):
        redalert = 0


median.sort()
print 'Average doc length: ', size/count,'\n Median: ', median[count/2]
print 'Doc count', count
print 'Successful'
```

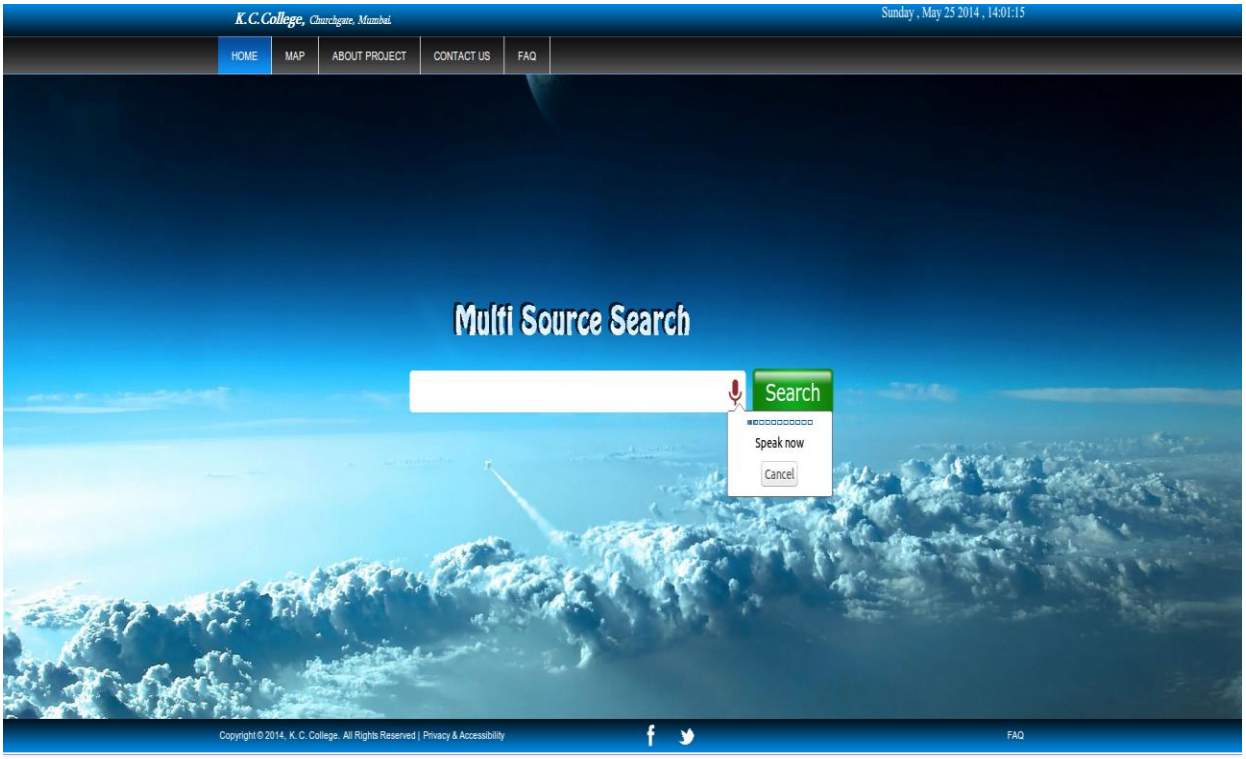
This is Average python code responsible for counting number of documents in the data source within its data sets also it calculate the average length of the documents containing text and text sizes.

UI Screenshots

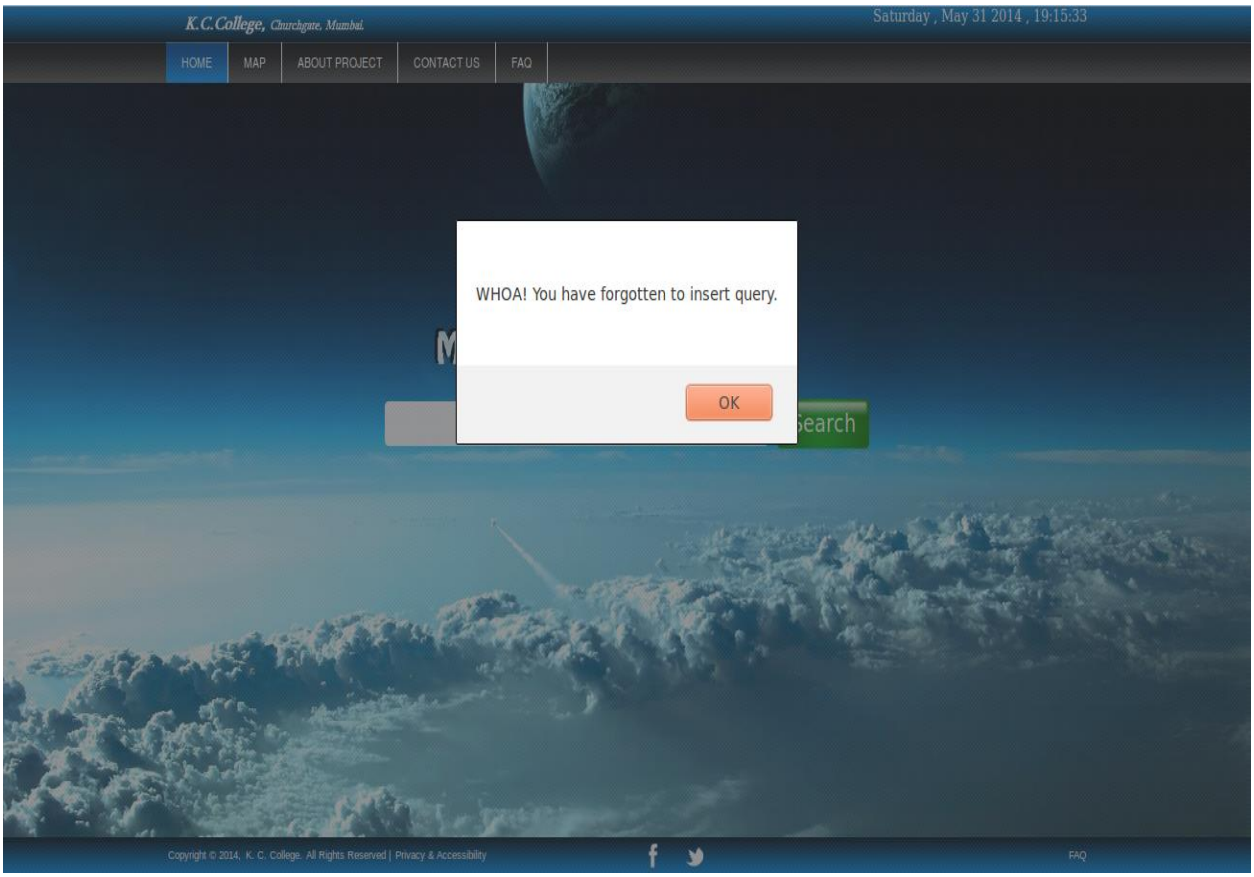
2. Index Page



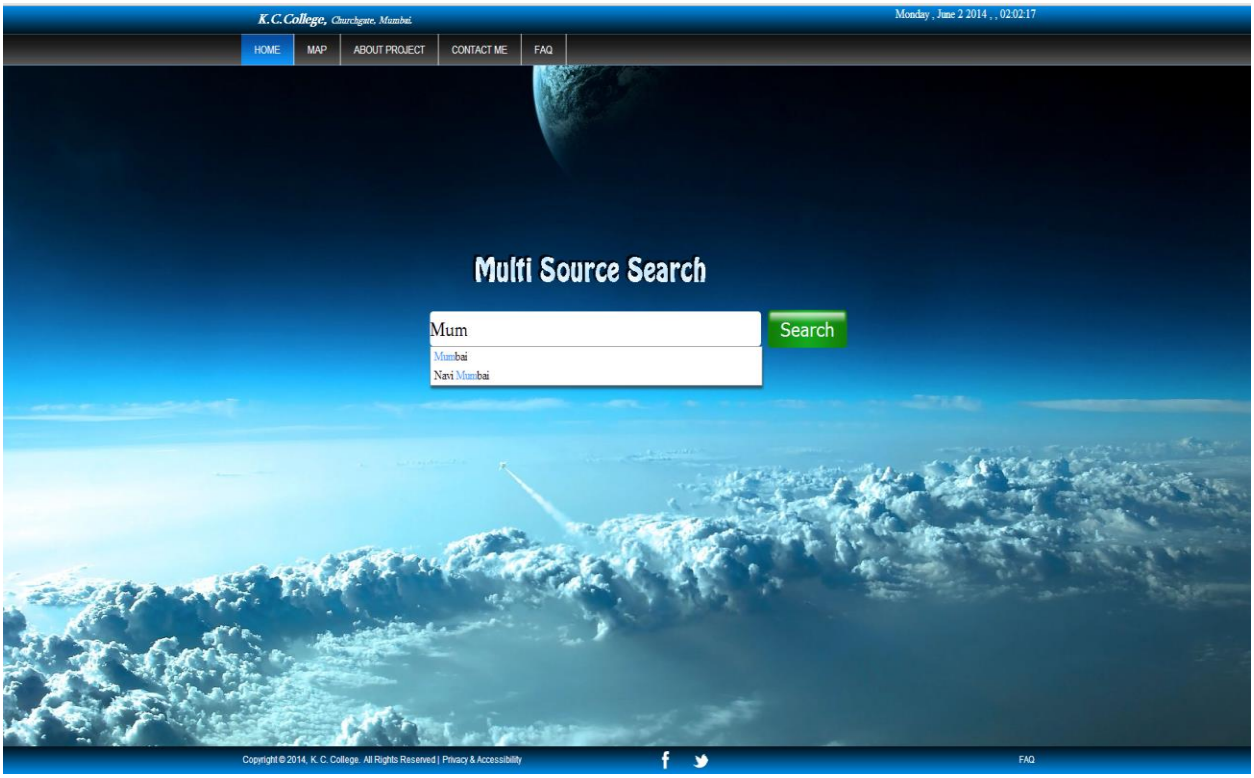
Voice to Text



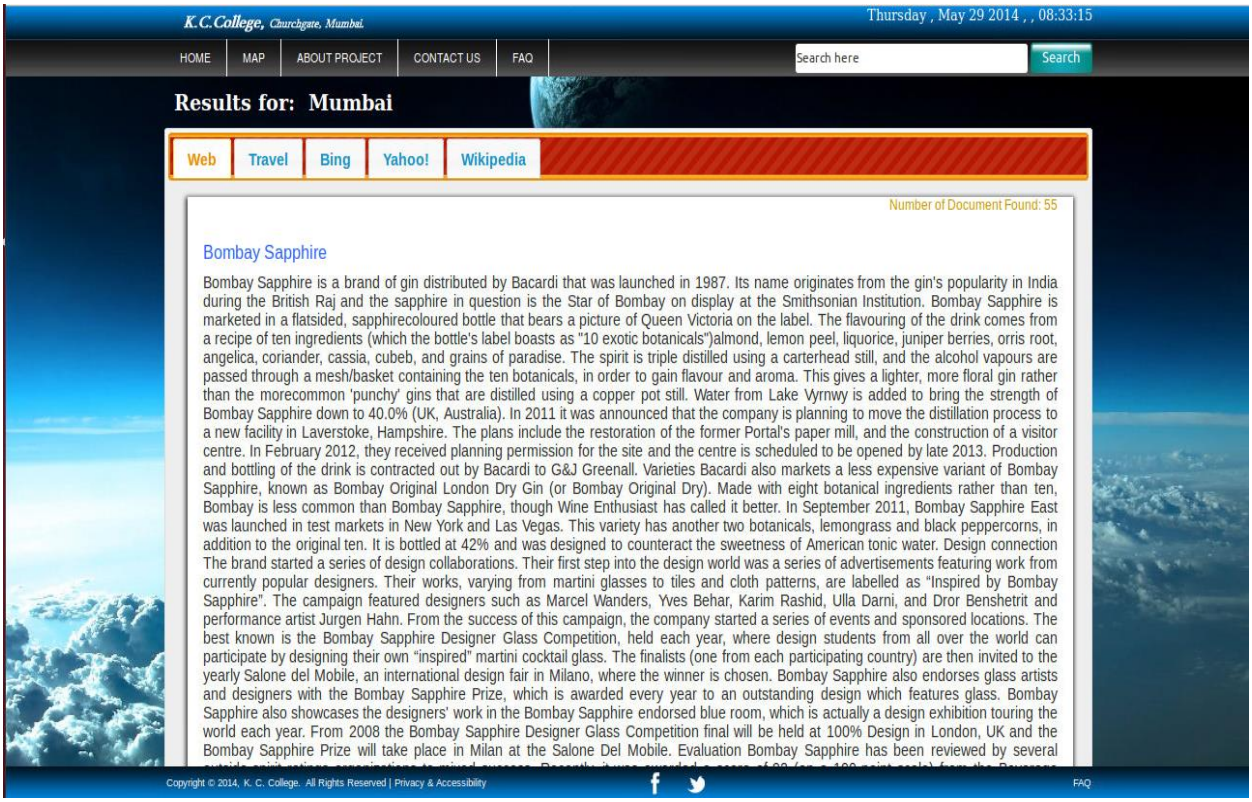
3. Validation of Textbox



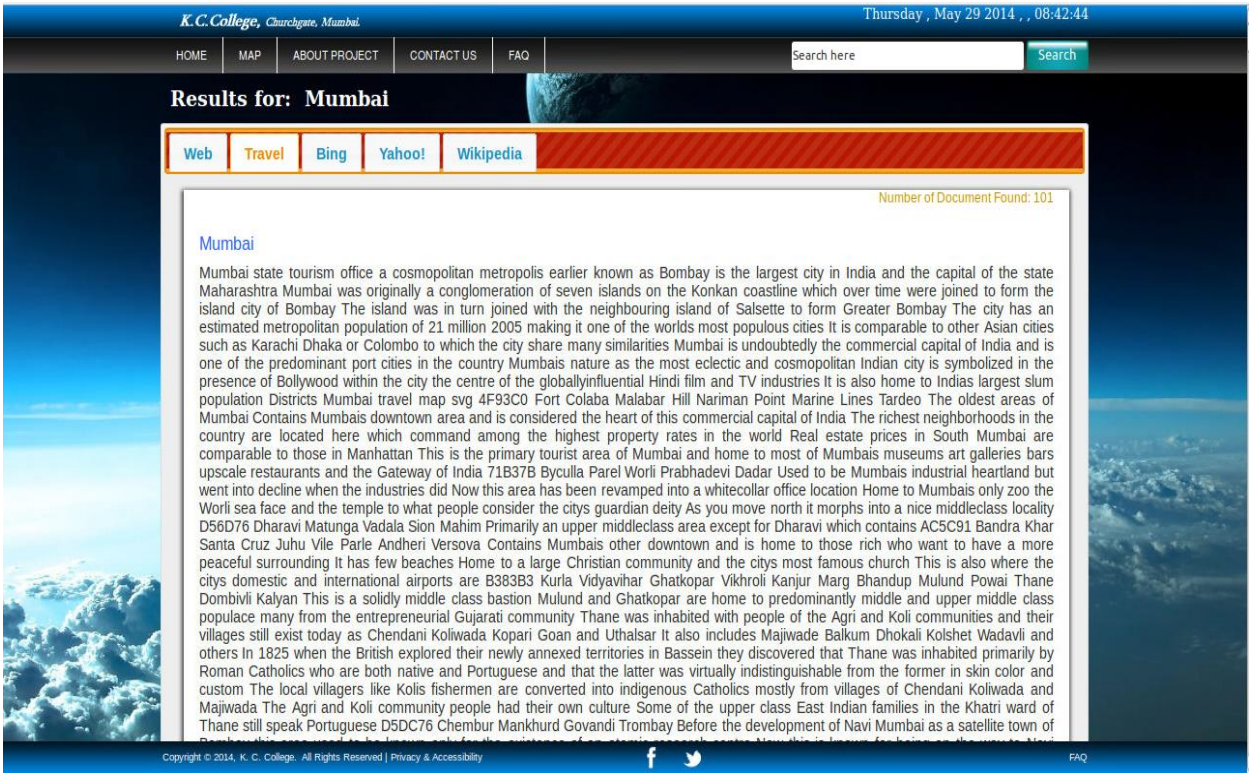
4. Auto-Suggestion Page



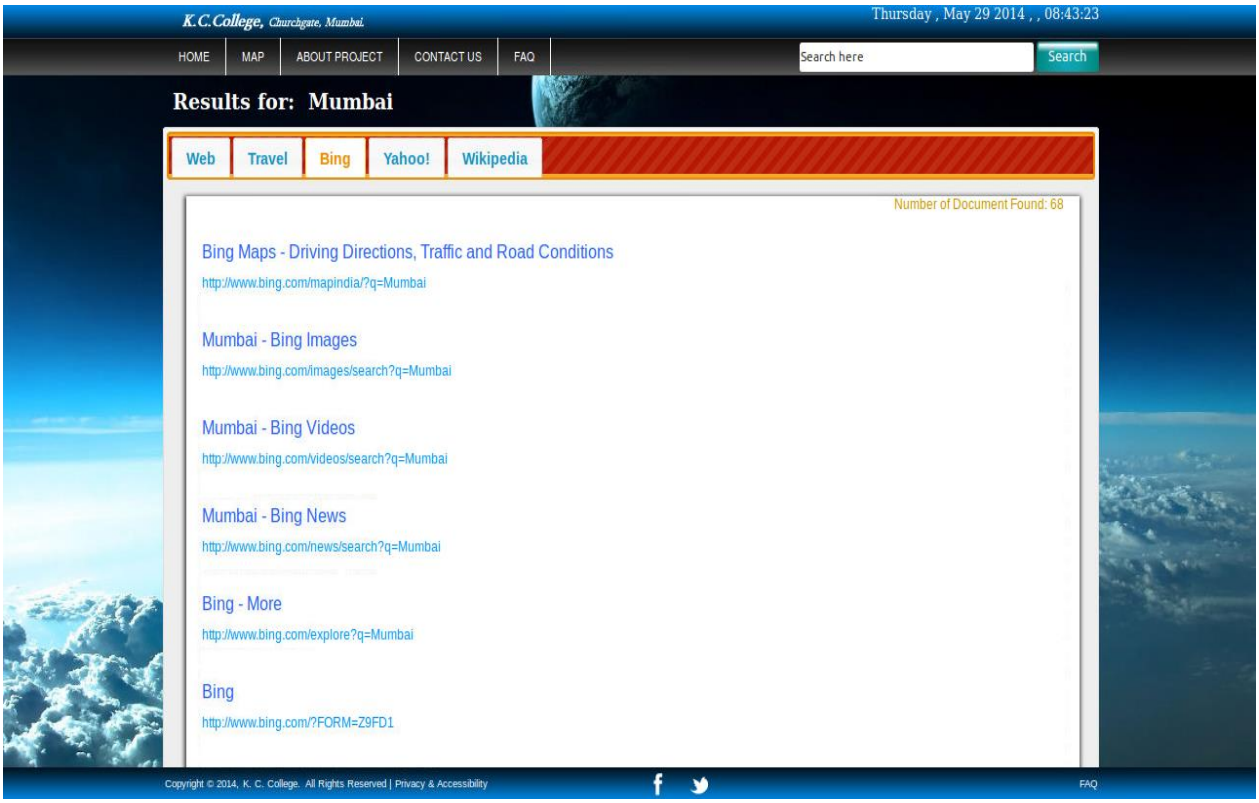
5. Result from WEB



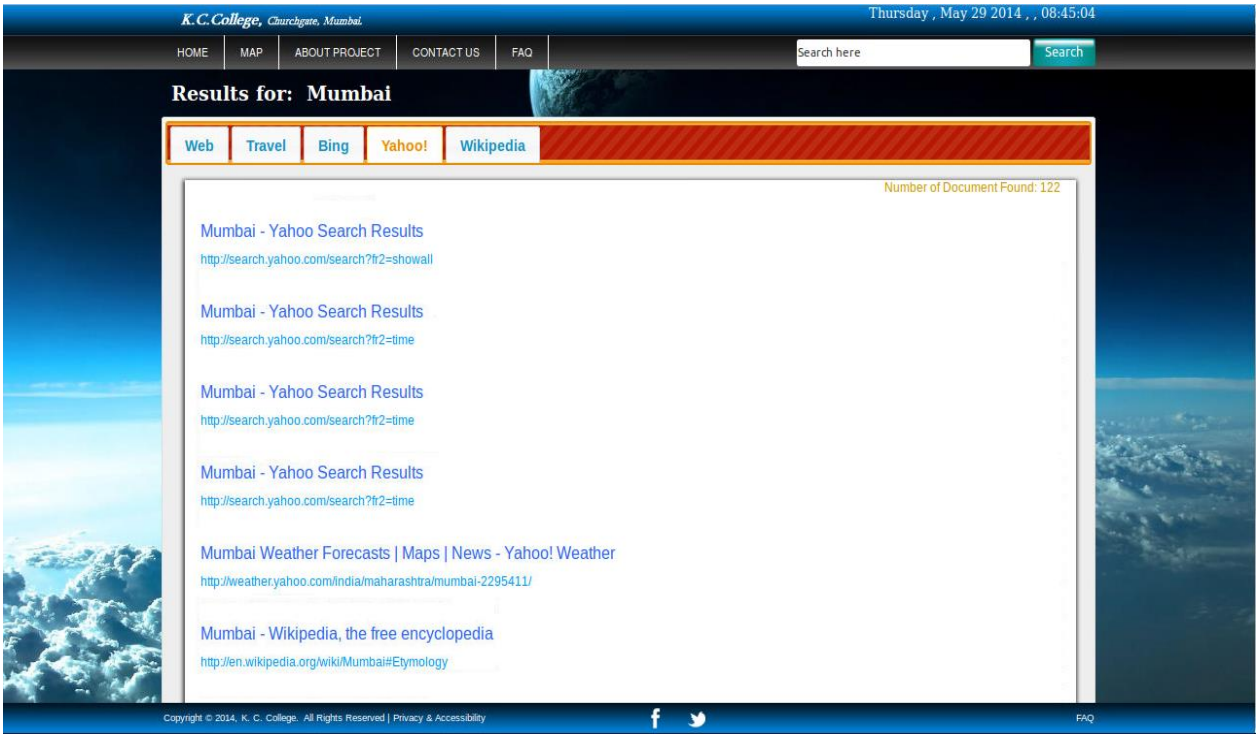
6. Result from Travel Tab



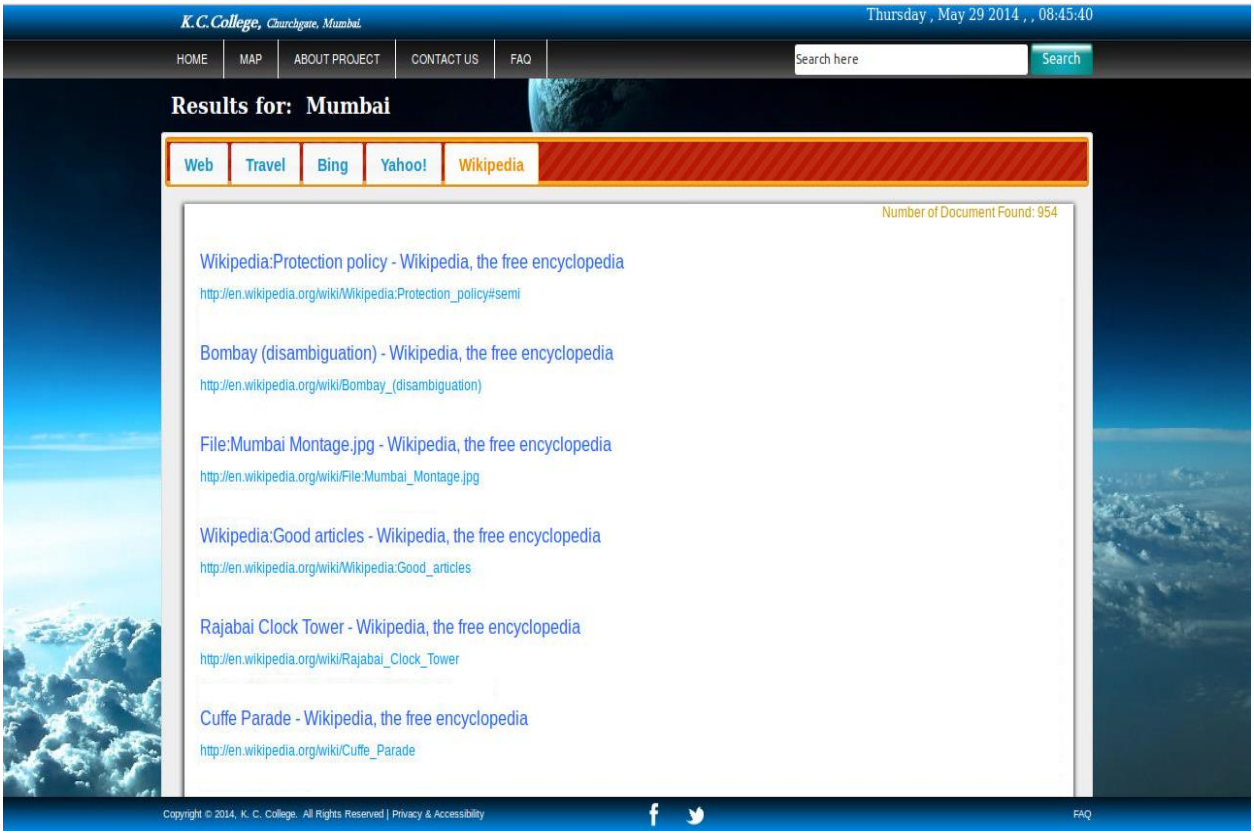
7. Result from Bing Tab



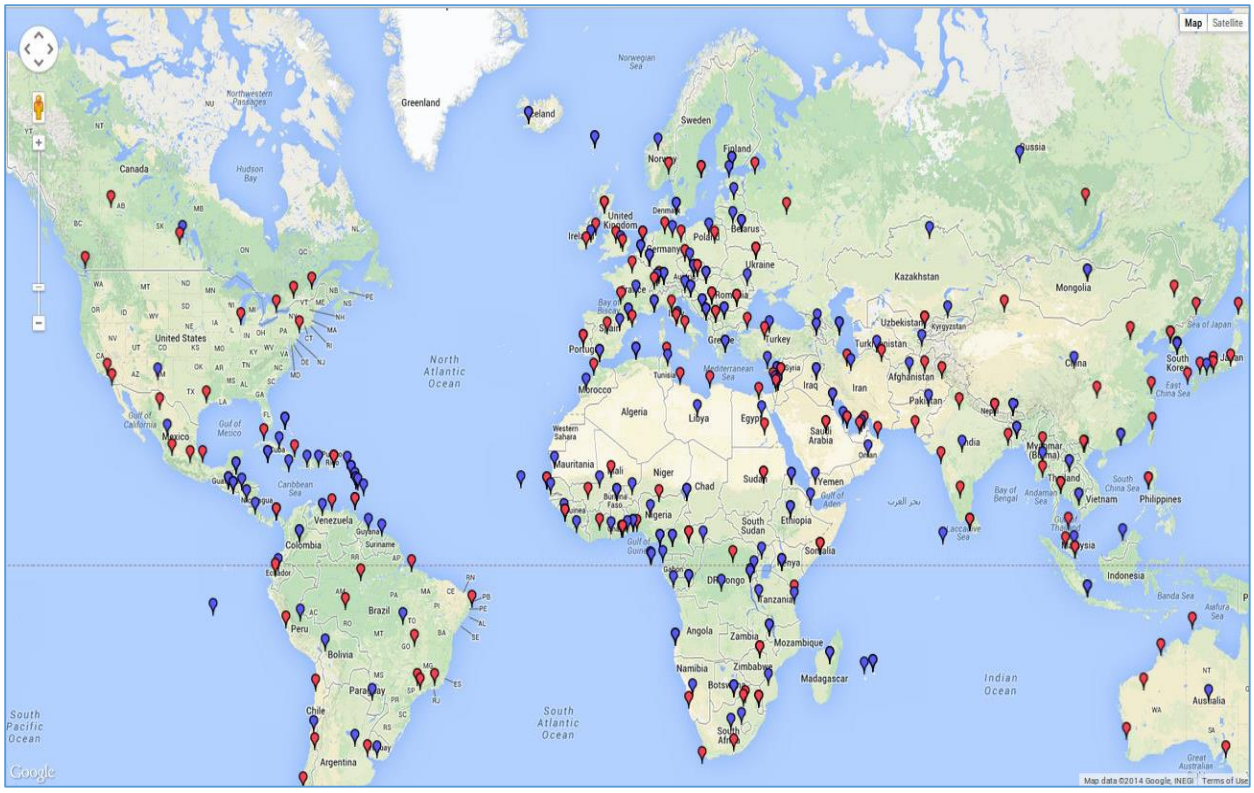
8. Result from Yahoo Tab



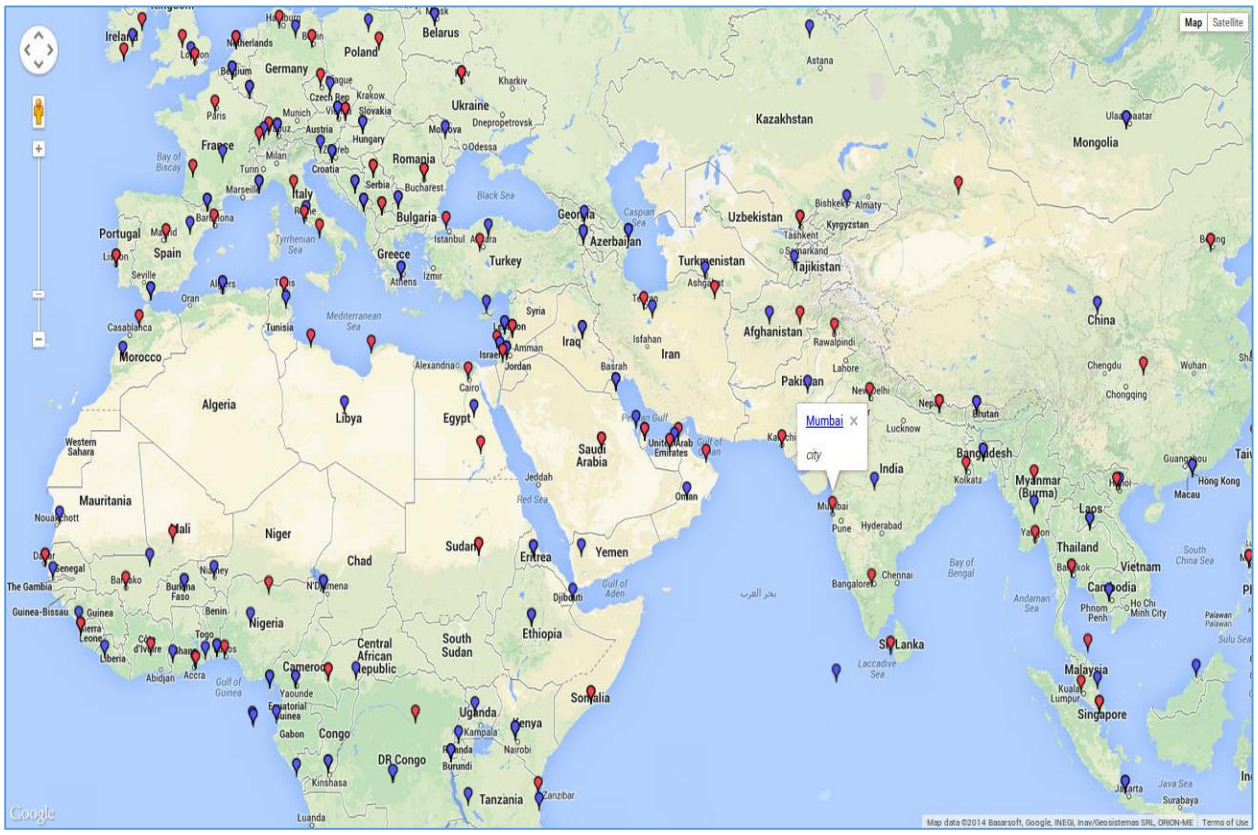
9. Result from Wikipedia



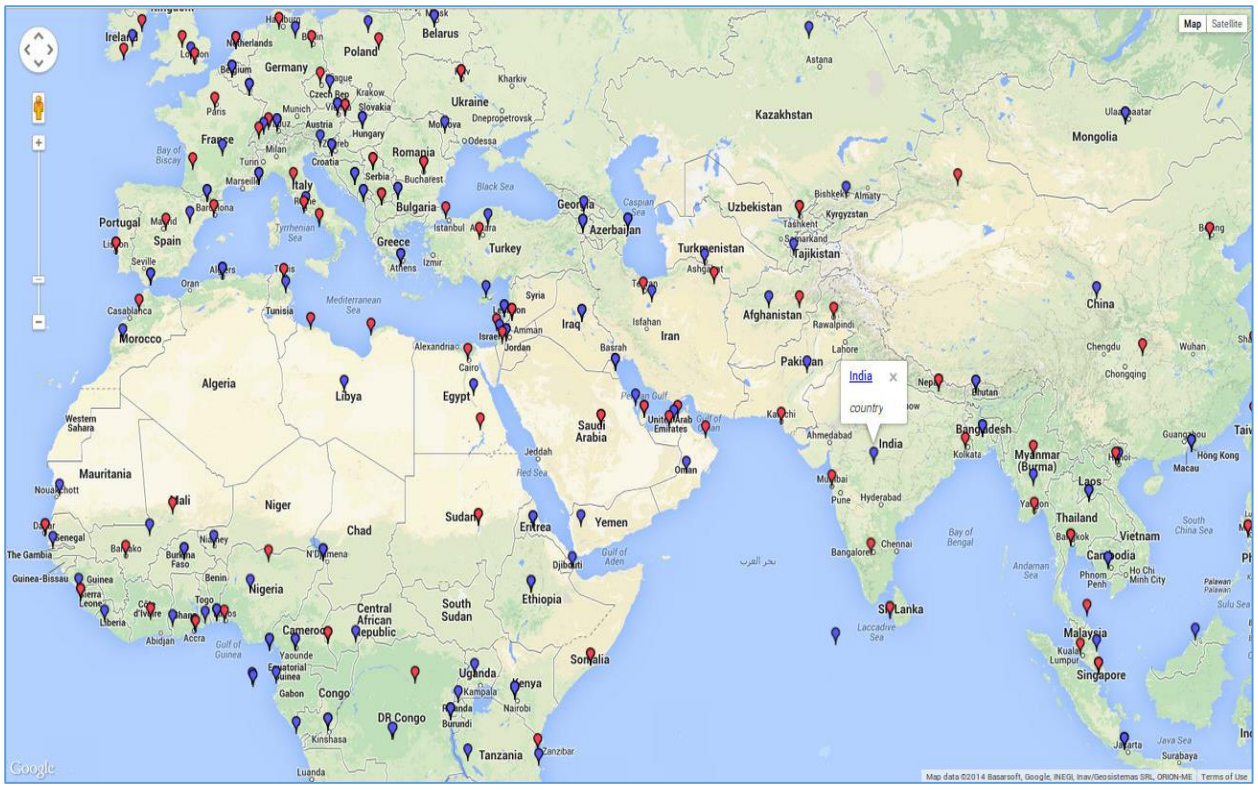
10. MAP Search UI with city and country Markers



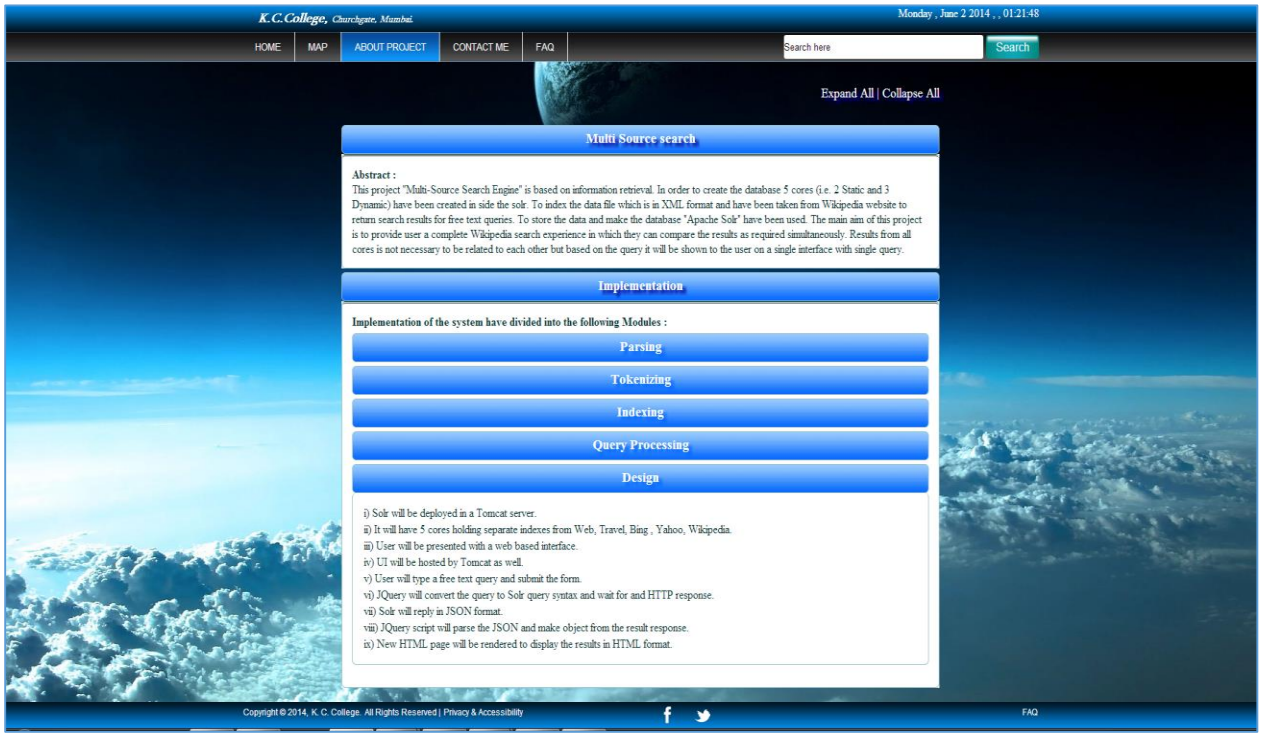
11. MAP Search UI with city Markers



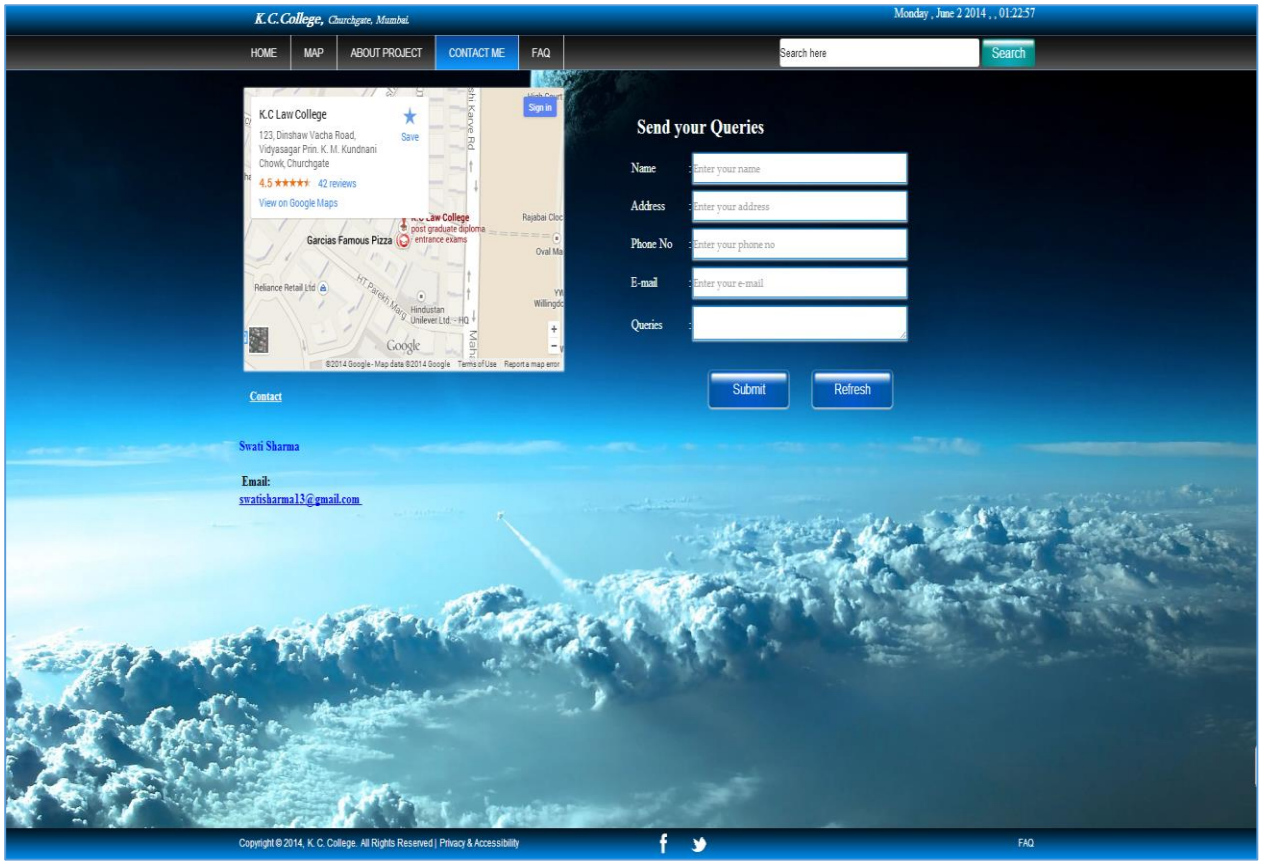
12. MAP Search UI with country Markers



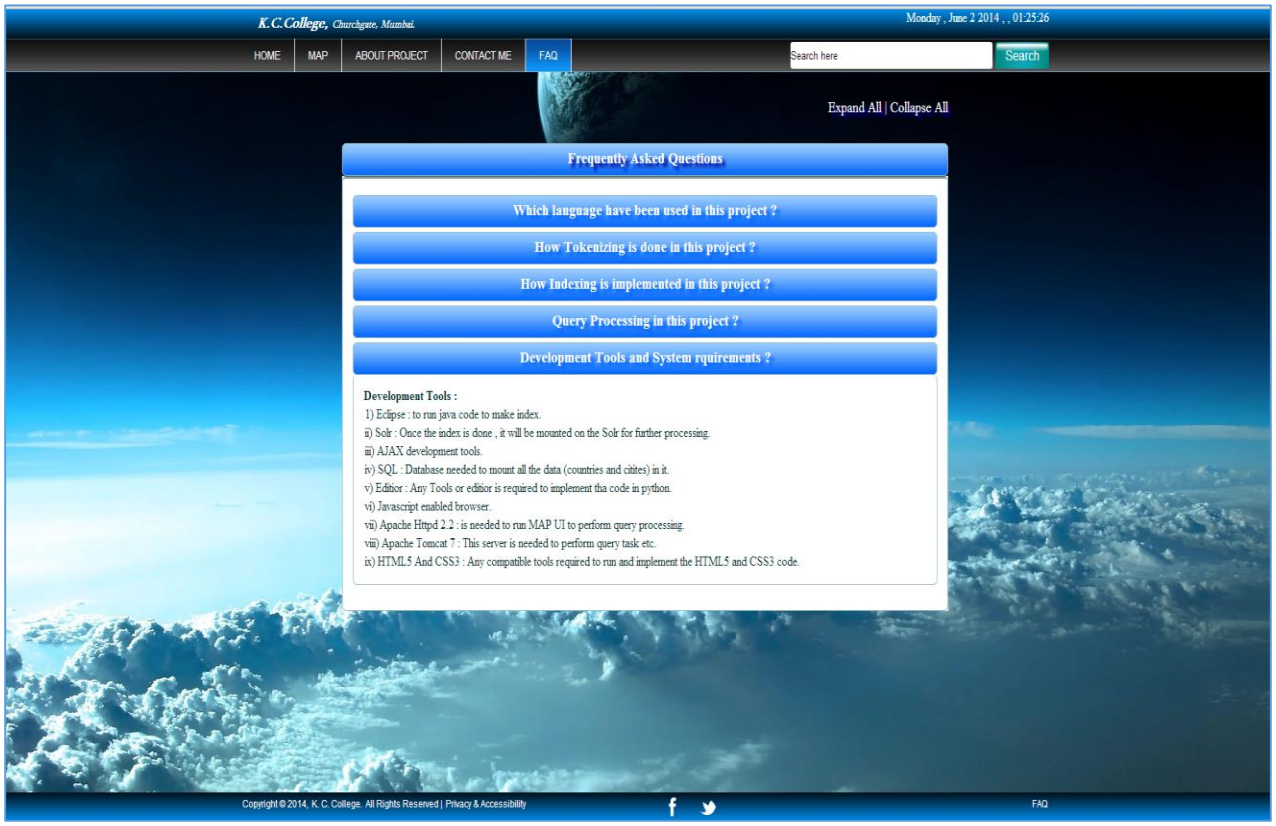
13. About the project



14. Contact Page

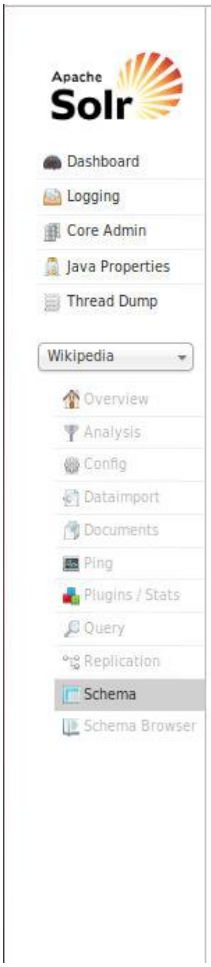


15. FAQ Page


K.C.College, Churchgate, Mumbai.
Monday , June 2 2014 , , 01:25:26
HOME
MAP
ABOUT PROJECT
CONTACT ME
FAQ
Search here
Search
Expand All | Collapse All
Frequently Asked Questions
Which language have been used in this project ?
How Tokenizing is done in this project ?
How Indexing is implemented in this project ?
Query Processing in this project ?
Development Tools and System rquirements ?
Development Tools :
1) Eclipse : to run java code to make index.
ii) Solr : Once the index is done , it will be mounted on the Solr for further processing.
iii) AJAX development tools.
iv) SQL : Database needed to mount all the data (countries and cities) in it.
v) Editor : Any Tools or editor is required to implement tha code in python.
vi) Javascript enabled browser.
vii) Apache Httpd 2.2 : is needed to run MAP UI to perform query processing.
viii) Apache Tomcat 7 : This server is needed to perform query task etc.
ix) HTML5 And CSS3 : Any compatible tools required to run and implement the HTML5 and CSS3 code.
Copyright © 2014, K.C. College. All Rights Reserved | Privacy & Accessibility
FAQ

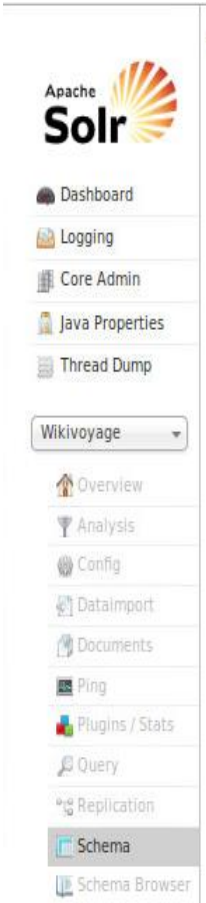
16. Solr Schema Index Page

i. Web Schema




```
 See the License for the specific language governing permissions and
 limitations under the License.
-->

<schema name="Wikipedia" version="1.1">
  <types>
    <fieldtype name="string"  class="solr.StrField" sortMissingLast="true" omitNorms="false"/>
    <fieldtype name="text"    class="solr.TextField" sortMissingLast="true" omitNorms="false">
        <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.PorterStemFilterFactory" />
                <filter class="solr.ShingleFilterFactory"/>
        </analyzer>
        <analyzer type="query">
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="querysynonyms.txt"/>
                <filter class="solr.PatternReplaceCharFilterFactory" pattern="-" replacement=""/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                <filter class="solr.LowerCaseFilterFactory" />
                <filter class="solr.PatternReplaceFilterFactory" pattern="^(\p{Punct}*)(.*?)(\p{Punct}*)$" replacement="$2"/>
                <filter class="solr.PorterStemFilterFactory" />
        </analyzer>
    </fieldtype>
    <fieldType name="spell" class="solr.TextField" positionIncrementGap="100">
    <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
                <filter class="solr.StandardFilterFactory"/>
                <filter class="solr.LowerCaseFilterFactory" />
                <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
    </analyzer>
    <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
                <filter class="solr.StandardFilterFactory"/>
                <filter class="solr.LowerCaseFilterFactory" />
                <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
  </analyzer>
  </fieldType>
   <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0"/>
 </types>

<fields>
 <!-- general -->
 <field name="id"        type="long"     indexed="true"  stored="true"  multiValued="false" required="true"/>
 <field name="url"       type="string"   indexed="false" stored="true"  multiValued="false" />
 <field name="title"      type="string"   indexed="true"  stored="true"  multiValued="false" omitNorms="false"/>
 <field name="text"      type="text"     indexed="true"  stored="true"  multiValued="false" omitNorms="false"/>
 <field name="iLink"      type="string"      indexed="false" stored="true"  multiValued="false" />
 <field name="eLink"      type="string"   indexed="false" stored="true"  multiValued="false"/>
 <field name="_version_" type="long"      indexed="true"  stored="true"/>
</fields>

<!-- field to use to determine and enforce document uniqueness. -->
<uniqueKey>id</uniqueKey>

<!--<copyField source="title" dest="/suggest"/>-->
<!-- field for the QueryParser to use when an explicit fieldname is absent -->
<defaultSearchField>text</defaultSearchField>

<!-- SolrQueryParser configuration: defaultOperator="AND|OR" -->
<solrQueryParser defaultOperator="AND"/>
</schema>
```

ii. Travel Schema

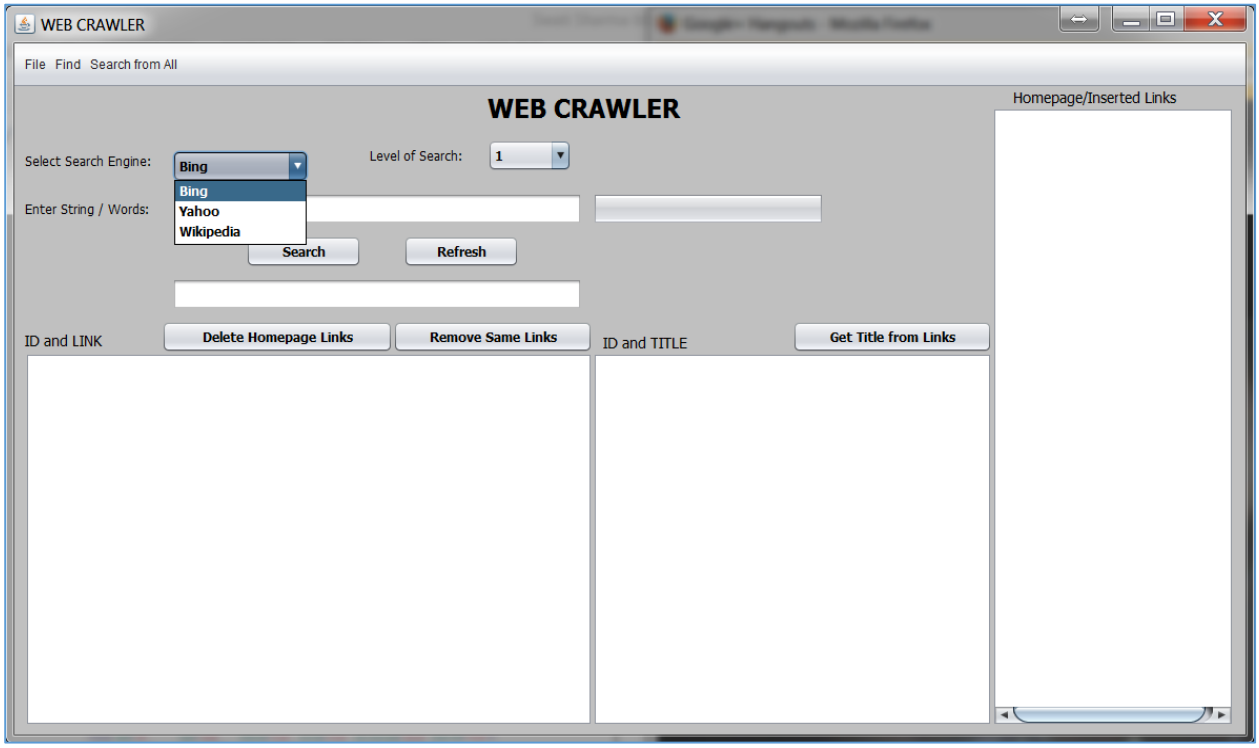


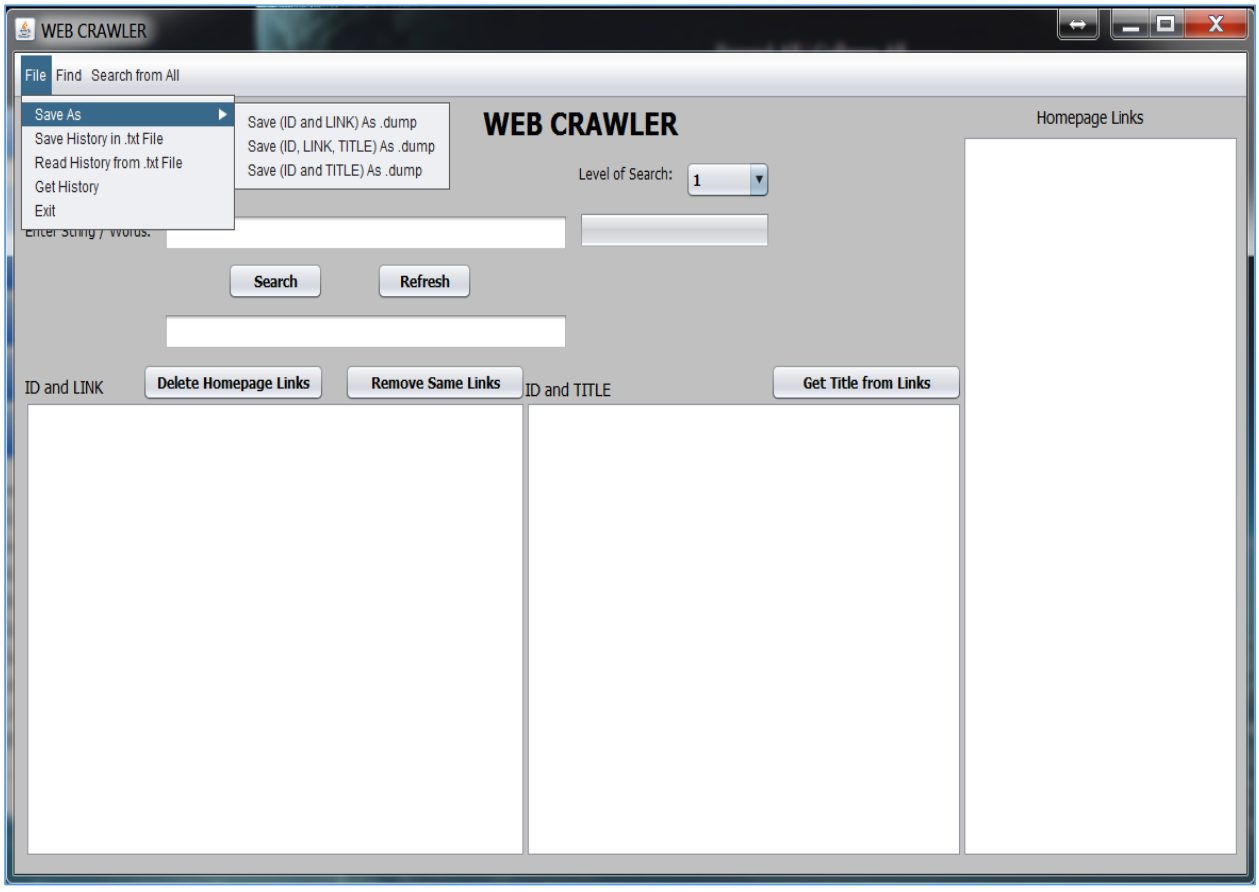

```
<schema name="Wikivoyage" version="1.1">
  <types>
   <fieldtype name="string"  class="solr.StrField" sortMissingLast="true" omitNorms="false"/>
   <fieldtype name="text"    class="solr.TextField" sortMissingLast="true" omitNorms="false">
        <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                <filter class="solr.LowerCaseFilterFactory"/>
                <filter class="solr.PorterStemFilterFactory" />
                <filter class="solr.ShingleFilterFactory"/>
        </analyzer>
        <analyzer type="query">
                <tokenizer class="solr.WhitespaceTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="querysynonyms.txt"/>
                <filter class="solr.PatternReplaceCharFilterFactory" pattern="-" replacement=""/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                <filter class="solr.LowerCaseFilterFactory" />
                <filter class="solr.PatternReplaceFilterFactory" pattern="^(\p{Punct}*)(.*?)(\p{Punct}*)$" replacement="$2"/>
                <filter class="solr.PorterStemFilterFactory" />
        </analyzer>
   </fieldtype>
   <fieldType name="spell" class="solr.TextField" positionIncrementGap="100">
   <analyzer type="index">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
                <filter class="solr.StandardFilterFactory"/>
                <filter class="solr.LowerCaseFilterFactory" />
                <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
   </analyzer>
   <analyzer type="query">
                <tokenizer class="solr.StandardTokenizerFactory"/>
                <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
                <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
                <filter class="solr.StandardFilterFactory"/>
                <filter class="solr.LowerCaseFilterFactory" />
                <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
   </analyzer>
   </fieldType>
    <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0"/>
   </types>
```

```
 <fields>
  <!-- general -->
  <field name="id"        type="long"     indexed="true"  stored="true"  multiValued="false" required="true"/>
  <field name="url"       type="string"   indexed="false" stored="true"  multiValued="false" />
  <field name="title"     type="string"   indexed="true"  stored="true"  multiValued="false" />
  <field name="text"      type="text"     indexed="true"  stored="true"  multiValued="false" />
  <field name="iLink"     type="string"   indexed="false" stored="true"  multiValued="false" />
  <field name="eLink"     type="string"   indexed="false" stored="true"  multiValued="false" />
  <field name="_version_" type="long"     indexed="true"  stored="true"/>
 </fields>

 <!-- field to use to determine and enforce document uniqueness. -->
 <uniqueKey>id</uniqueKey>

 <!--<copyField source="title" dest="/suggest"/>-->
 <!-- field for the QueryParser to use when an explicit fieldname is absent -->
 <defaultSearchField>text</defaultSearchField>

 <!-- SolrQueryParser configuration: defaultOperator="AND|OR" -->
 <solrQueryParser defaultOperator="AND"/>
</schema>
```

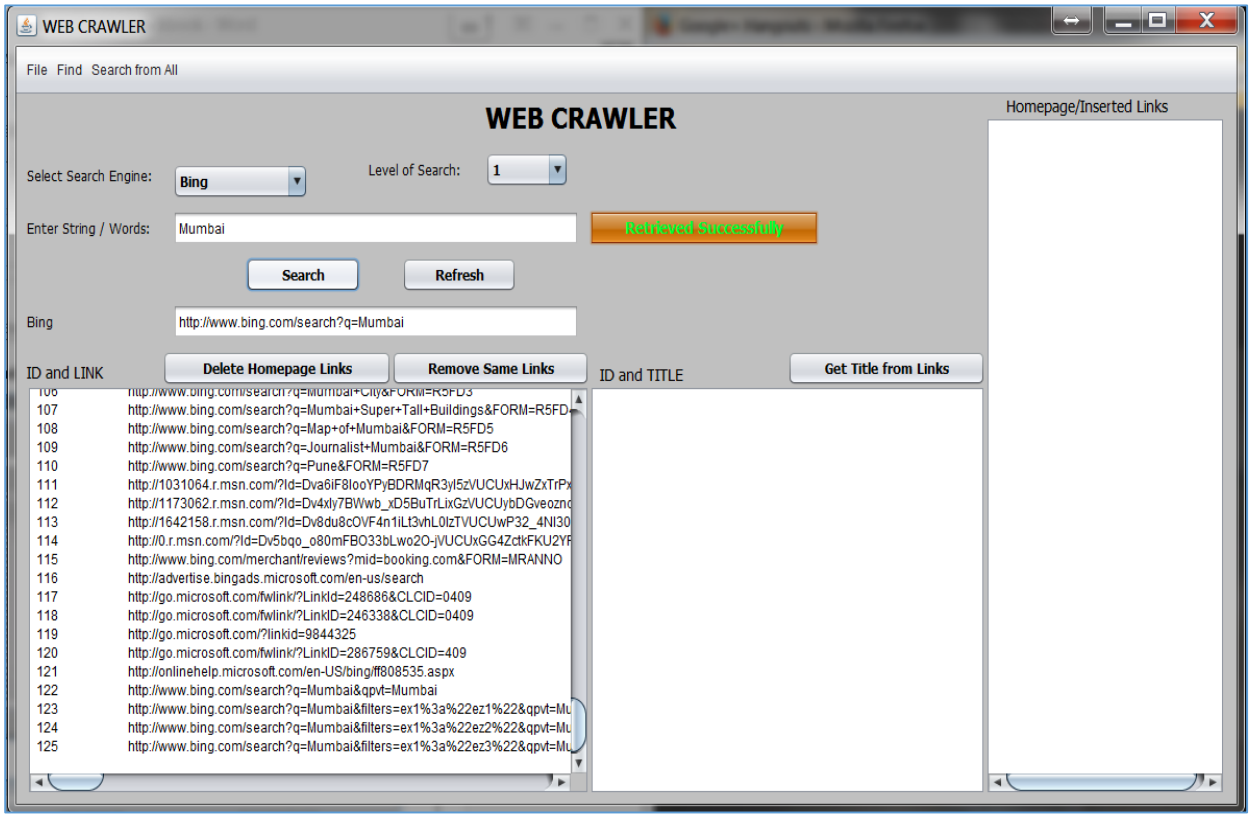
17. Crawler

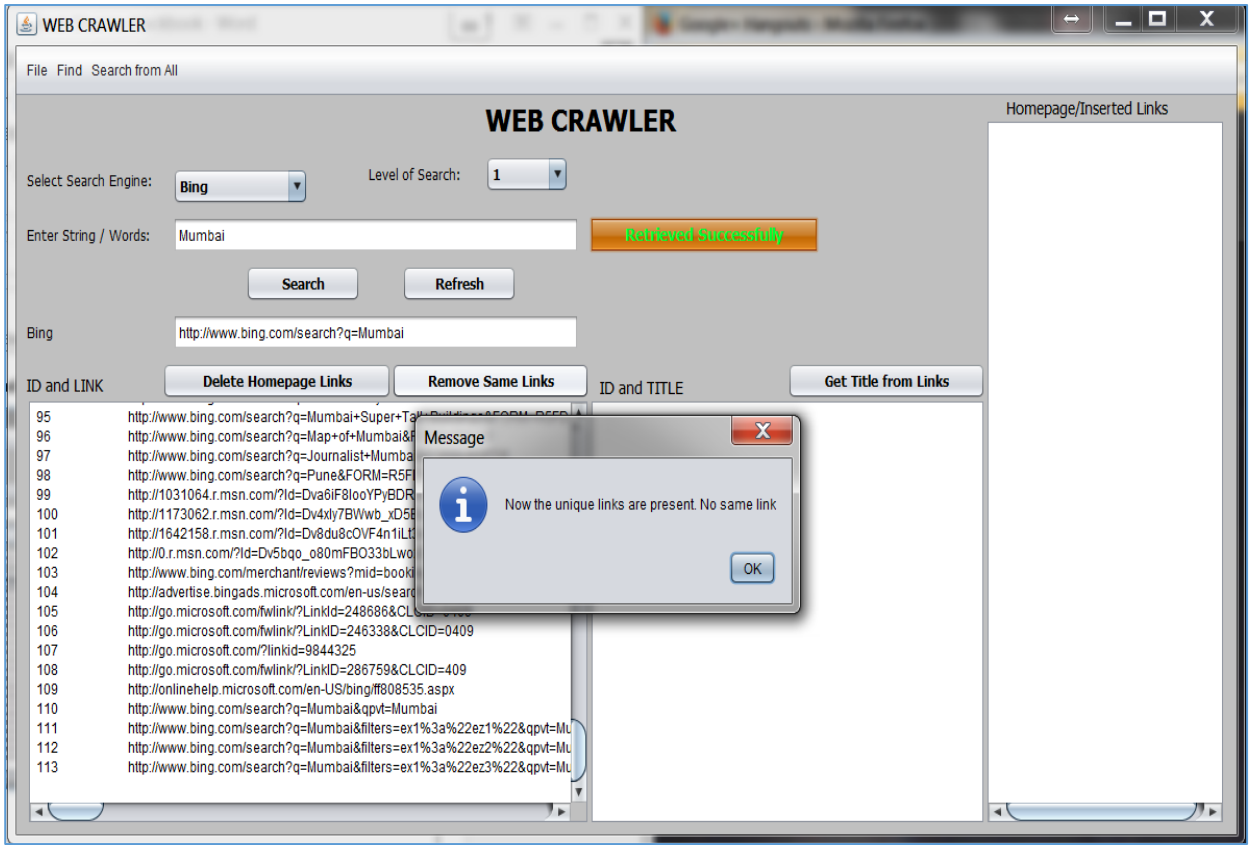
i. Main Page



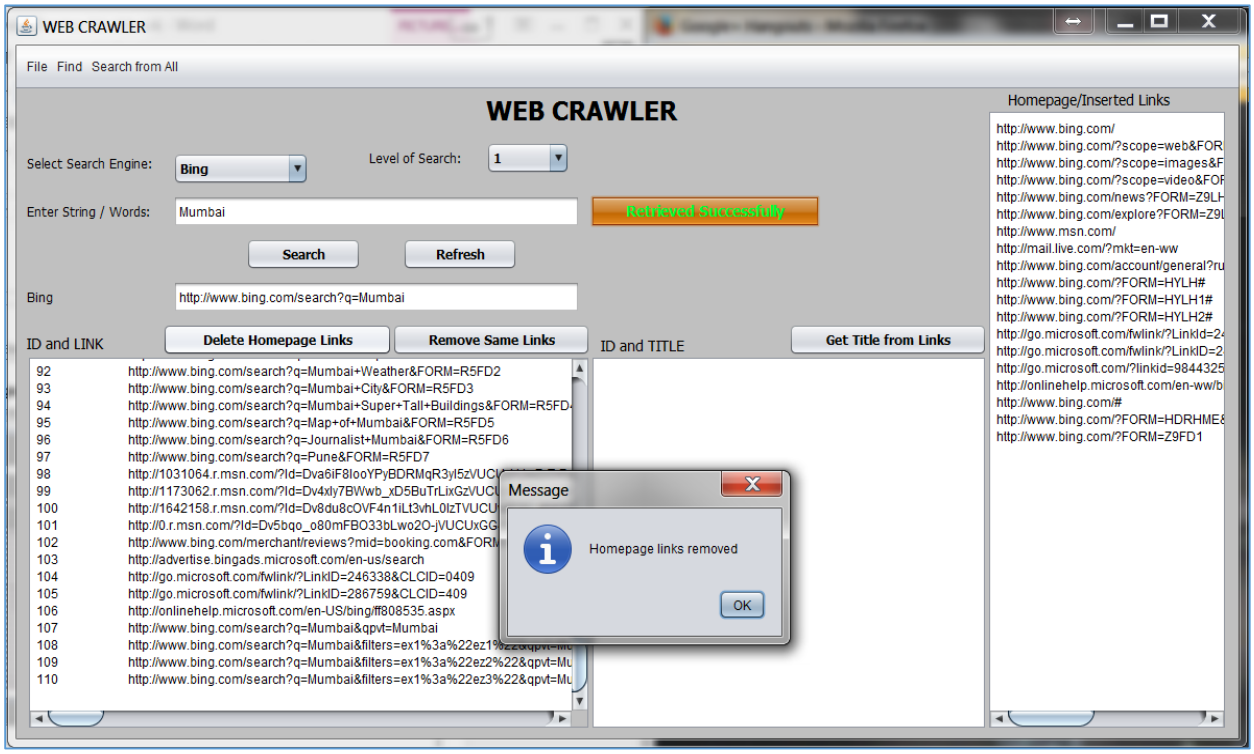


Crawled link and assigned corresponding ID

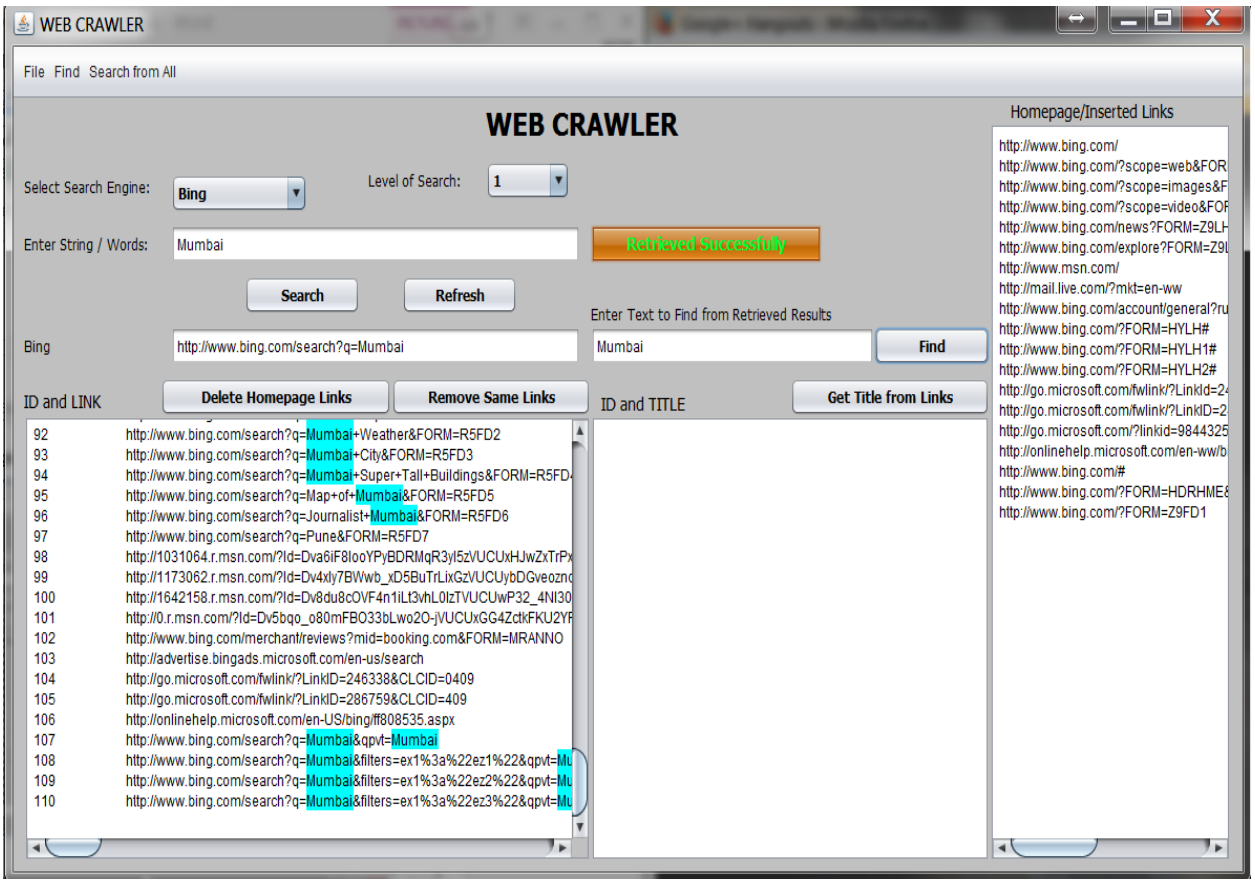


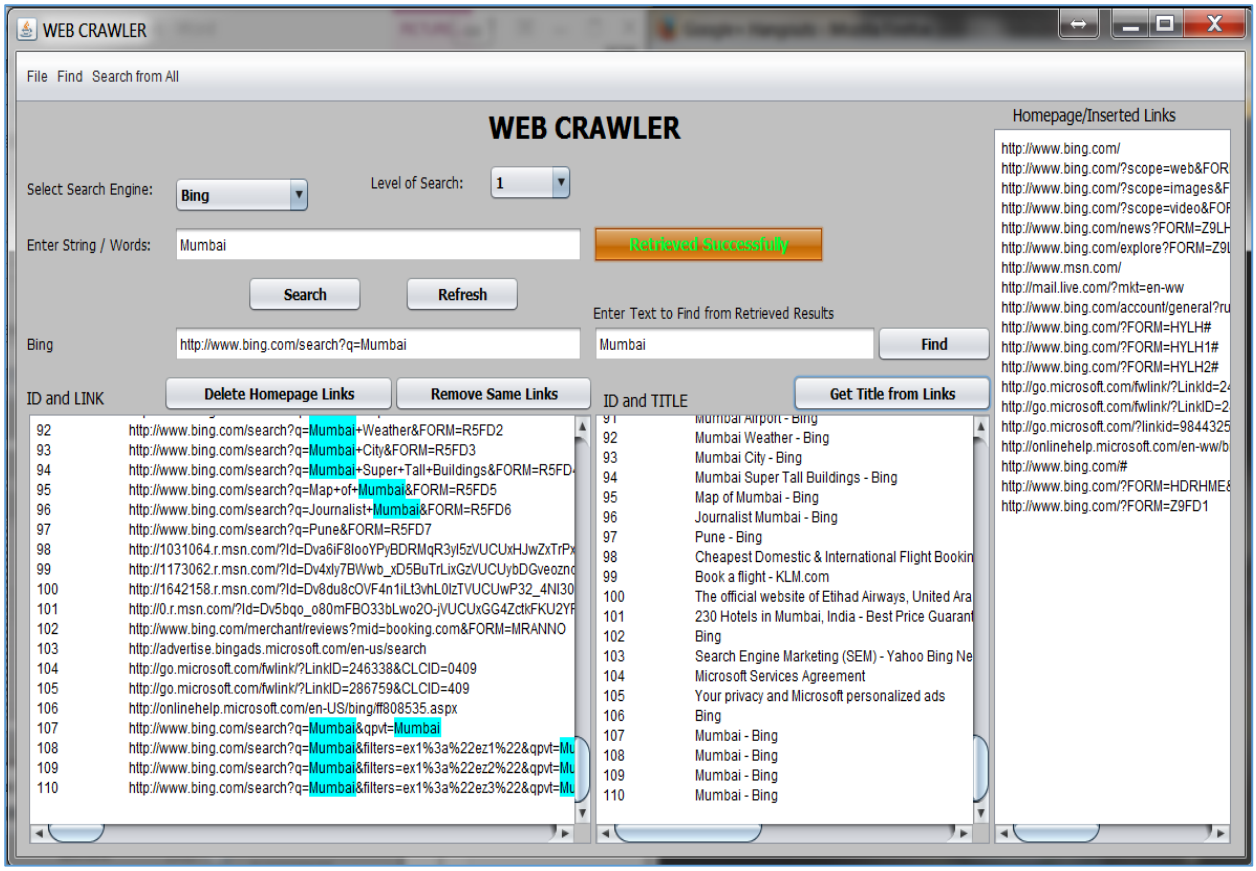


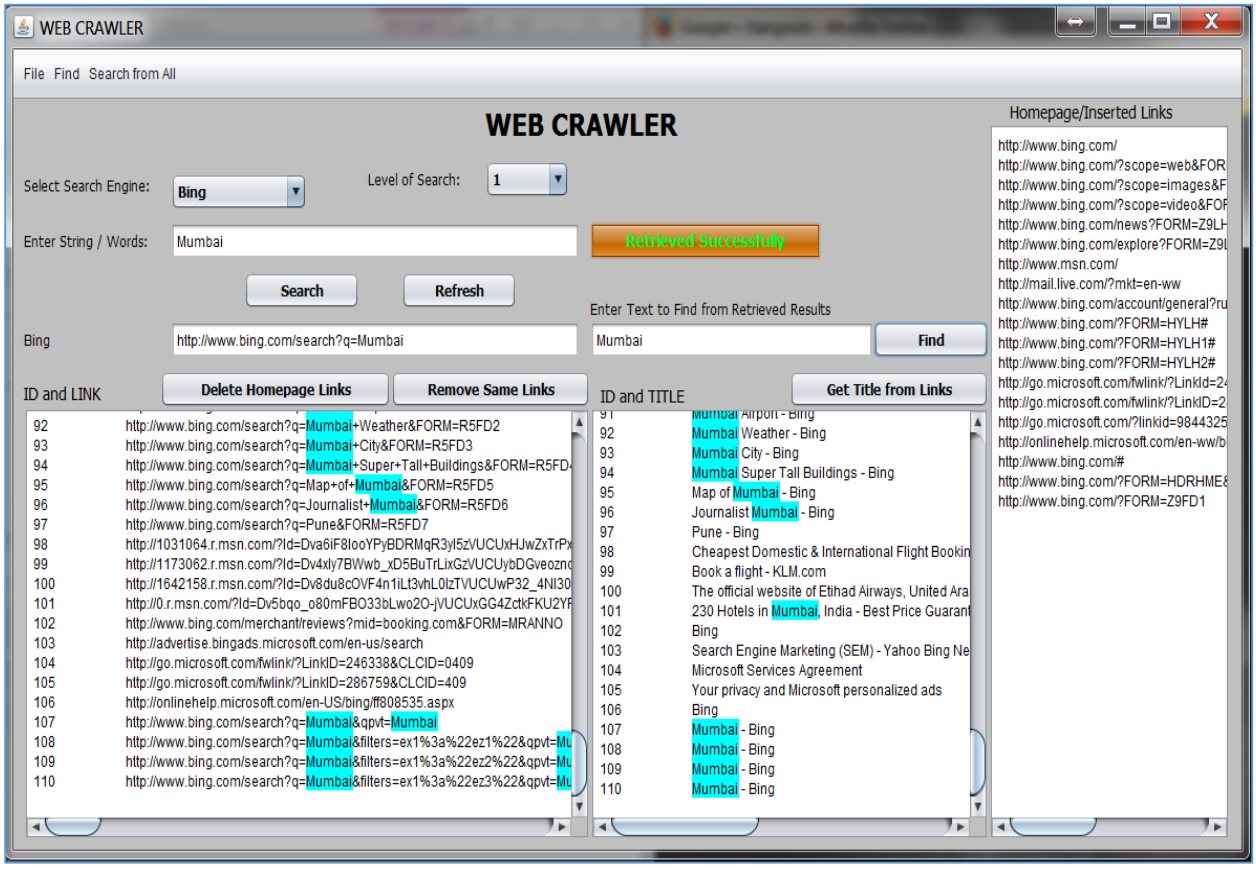
Deleting Homepage Links after removing duplicate links



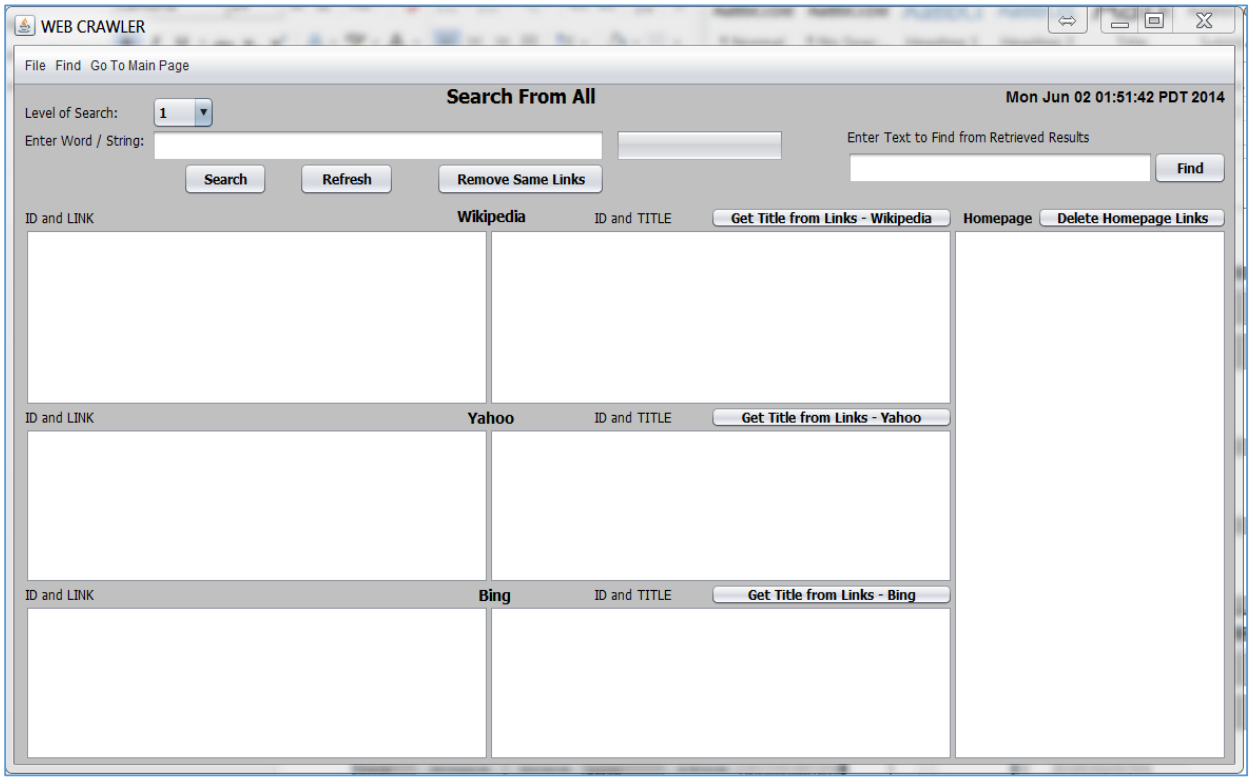
Find Option :

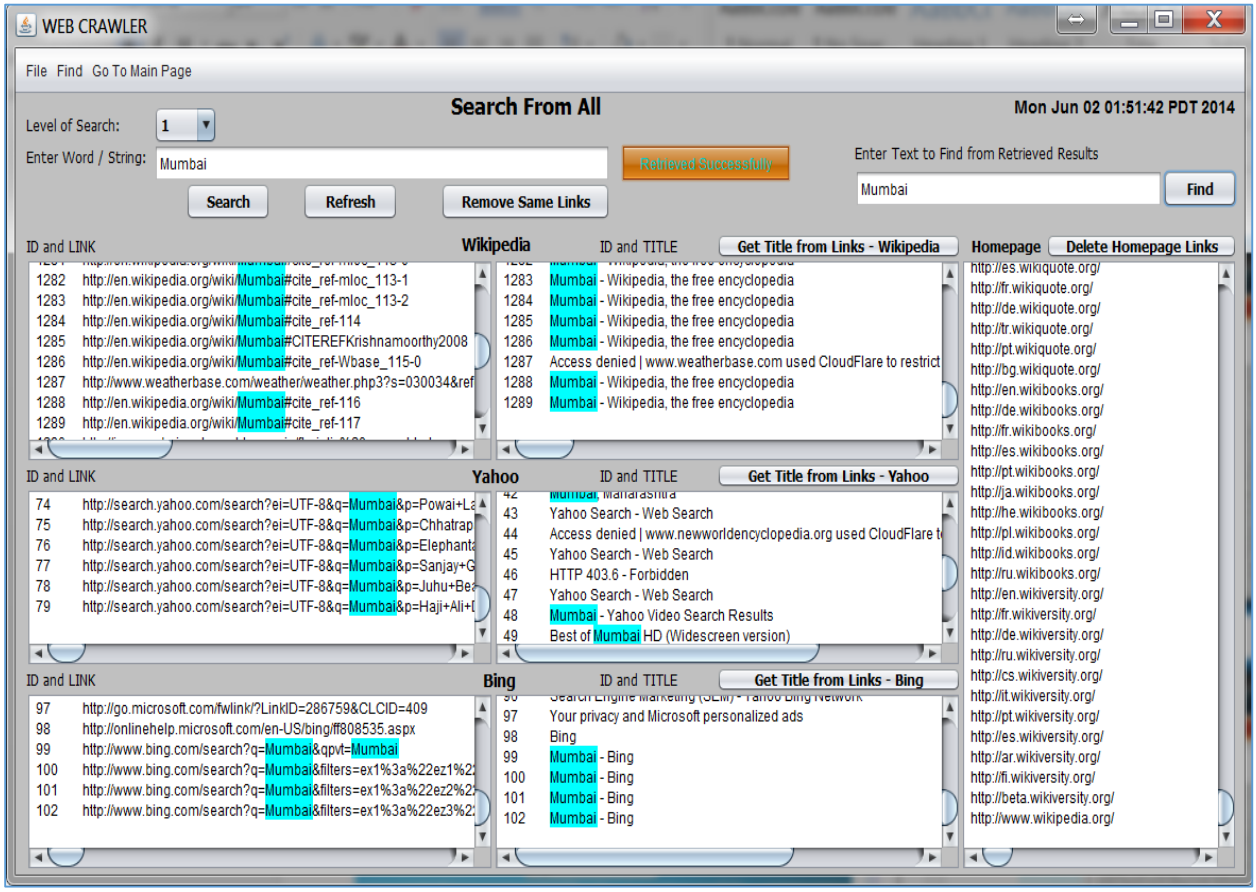



WEB CRAWLER
File Find Search from All
WEB CRAWLER
Select Search Engine:
Bing
Level of Search:
1
Enter String / Words:
Mumbai
Retrieved Successfully
Search
Refresh
Bing
http://www.bing.com/search?q=Mumbai
Enter Text to Find from Retrieved Results
Mumbai
Find
ID and LINK
Delete Homepage Links
Remove Same Links
ID and TITLE
Get Title from Links
Homepage/Inserted Links
92 http://www.bing.com/search?q=Mumbai+Weather&FORM=R5FD2
93 http://www.bing.com/search?q=Mumbai+City&FORM=R5FD3
94 http://www.bing.com/search?q=Mumbai+Super+Tall+Buildings&FORM=R5FD
95 http://www.bing.com/search?q=Map+of+Mumbai&FORM=R5FD5
96 http://www.bing.com/search?q=Journalist+Mumbai&FORM=R5FD6
97 http://www.bing.com/search?q=Pune&FORM=R5FD7
98 http://1031064.r.msn.com/?id=Dva6iF8looYPyBDRMqR3yl5zVUCUxHJwZxTrPx
99 http://1173062.r.msn.com/?id=Dv4xly7BWwb_xD5BuTrLixGzVUCUybDGveozn
100 http://1642158.r.msn.com/?id=Dv8du8cOVF4n1iLt3vhL0lzTVUCUwP32_4NI30
101 http://0.r.msn.com/?id=Dv5bqo_o80mFBO33bLwo2O-jVUCUxGG4ZctkFKU2YF
102 http://www.bing.com/merchant/reviews?mid=booking.com&FORM=MRANNO
103 http://advertise.bingads.microsoft.com/en-us/search
104 http://go.microsoft.com/fwlink/?LinkID=246338&CLCID=0409
105 http://go.microsoft.com/fwlink/?LinkID=286759&CLCID=409
106 http://onlinehelp.microsoft.com/en-US/bing/ff808535.aspx
107 http://www.bing.com/search?q=Mumbai&qpvt=Mumbai
108 http://www.bing.com/search?q=Mumbai&filters=ex1%3a%22ez1%22&qpvt=Mu
109 http://www.bing.com/search?q=Mumbai&filters=ex1%3a%22ez2%22&qpvt=Mu
110 http://www.bing.com/search?q=Mumbai&filters=ex1%3a%22ez3%22&qpvt=Mu
92 Mumbai Weather - Bing
93 Mumbai City - Bing
94 Mumbai Super Tall Buildings - Bing
95 Map of Mumbai - Bing
96 Journalist Mumbai - Bing
97 Pune - Bing
98 Cheapest Domestic & International Flight Bookin
99 Book a flight - KLM.com
100 The official website of Etihad Airways, United Ara
101 230 Hotels in Mumbai, India - Best Price Guarant
102 Bing
103 Search Engine Marketing (SEM) - Yahoo Bing Ne
104 Microsoft Services Agreement
105 Your privacy and Microsoft personalized ads
106 Bing
107 Mumbai - Bing
108 Mumbai - Bing
109 Mumbai - Bing
110 Mumbai - Bing
http://www.bing.com/
http://www.bing.com/?scope=web&FOR
http://www.bing.com/?scope=images&F
http://www.bing.com/?scope=video&FOR
http://www.bing.com/news?FORM=Z9LH
http://www.bing.com/explore?FORM=Z9L
http://www.msn.com/
http://mail.live.com/?mkt=en-ww
http://www.bing.com/account/general?ru
http://www.bing.com/?FORM=HYLH#
http://www.bing.com/?FORM=HYLH1#
http://www.bing.com/?FORM=HYLH2#
http://go.microsoft.com/fwlink/?LinkId=24
http://go.microsoft.com/fwlink/?LinkID=2
http://go.microsoft.com/?linkid=9844325
http://onlinehelp.microsoft.com/en-ww/b
http://www.bing.com/#
http://www.bing.com/?FORM=HDRHME8
http://www.bing.com/?FORM=Z9FD1


WEB CRAWLER
File Find Search from All
WEB CRAWLER
Select Search Engine:
Bing
Level of Search:
1
Enter String / Words:
Mumbai
Retrieved Successfully
Search
Refresh
Bing
http://www.bing.com/search?q=Mumbai
Enter Text to Find from Retrieved Results
Mumbai
Find
ID and LINK
Delete Homepage Links
Remove Same Links
ID and TITLE
Get Title from Links
Homepage/Inserted Links
92 http://www.bing.com/search?q=Mumbai+Weather&FORM=R5FD2
93 http://www.bing.com/search?q=Mumbai+City&FORM=R5FD3
94 http://www.bing.com/search?q=Mumbai+Super+Tall+Buildings&FORM=R5FD
95 http://www.bing.com/search?q=Map+of+Mumbai&FORM=R5FD5
96 http://www.bing.com/search?q=Journalist+Mumbai&FORM=R5FD6
97 http://www.bing.com/search?q=Pune&FORM=R5FD7
98 http://1031064.r.msn.com/?id=Dva6iF8looYPyBDRMqR3yl5zVUCUxHJwZxTrPx
99 http://1173062.r.msn.com/?id=Dv4xly7BWwb_xD5BuTrLixGzVUCUybDGveozn
100 http://1642158.r.msn.com/?id=Dv8du8cOVF4n1iLt3vhL0lzTVUCUwP32_4NI30
101 http://0.r.msn.com/?id=Dv5bqo_o80mFBO33bLwo2O-jVUCUxGG4ZctkFKU2YF
102 http://www.bing.com/merchant/reviews?mid=booking.com&FORM=MRANNO
103 http://advertise.bingads.microsoft.com/en-us/search
104 http://go.microsoft.com/fwlink/?LinkID=246338&CLCID=0409
105 http://go.microsoft.com/fwlink/?LinkID=286759&CLCID=409
106 http://onlinehelp.microsoft.com/en-US/bing/ff808535.aspx
107 http://www.bing.com/search?q=Mumbai&qpvt=Mumbai
108 http://www.bing.com/search?q=Mumbai&filters=ex1%3a%22ez1%22&qpvt=Mu
109 http://www.bing.com/search?q=Mumbai&filters=ex1%3a%22ez2%22&qpvt=Mu
110 http://www.bing.com/search?q=Mumbai&filters=ex1%3a%22ez3%22&qpvt=Mu
92 Mumbai Weather - Bing
93 Mumbai City - Bing
94 Mumbai Super Tall Buildings - Bing
95 Map of Mumbai - Bing
96 Journalist Mumbai - Bing
97 Pune - Bing
98 Cheapest Domestic & International Flight Bookin
99 Book a flight - KLM.com
100 The official website of Etihad Airways, United Ara
101 230 Hotels in Mumbai, India - Best Price Guarant
102 Bing
103 Search Engine Marketing (SEM) - Yahoo Bing Ne
104 Microsoft Services Agreement
105 Your privacy and Microsoft personalized ads
106 Bing
107 Mumbai - Bing
108 Mumbai - Bing
109 Mumbai - Bing
110 Mumbai - Bing
http://www.bing.com/
http://www.bing.com/?scope=web&FOR
http://www.bing.com/?scope=images&F
http://www.bing.com/?scope=video&FOR
http://www.bing.com/news?FORM=Z9LH
http://www.bing.com/explore?FORM=Z9L
http://www.msn.com/
http://mail.live.com/?mkt=en-ww
http://www.bing.com/account/general?ru
http://www.bing.com/?FORM=HYLH#
http://www.bing.com/?FORM=HYLH1#
http://www.bing.com/?FORM=HYLH2#
http://go.microsoft.com/fwlink/?LinkId=24
http://go.microsoft.com/fwlink/?LinkID=2
http://go.microsoft.com/?linkid=9844325
http://onlinehelp.microsoft.com/en-ww/b
http://www.bing.com/#
http://www.bing.com/?FORM=HDRHME8
http://www.bing.com/?FORM=Z9FD1

ii. Search From All Page

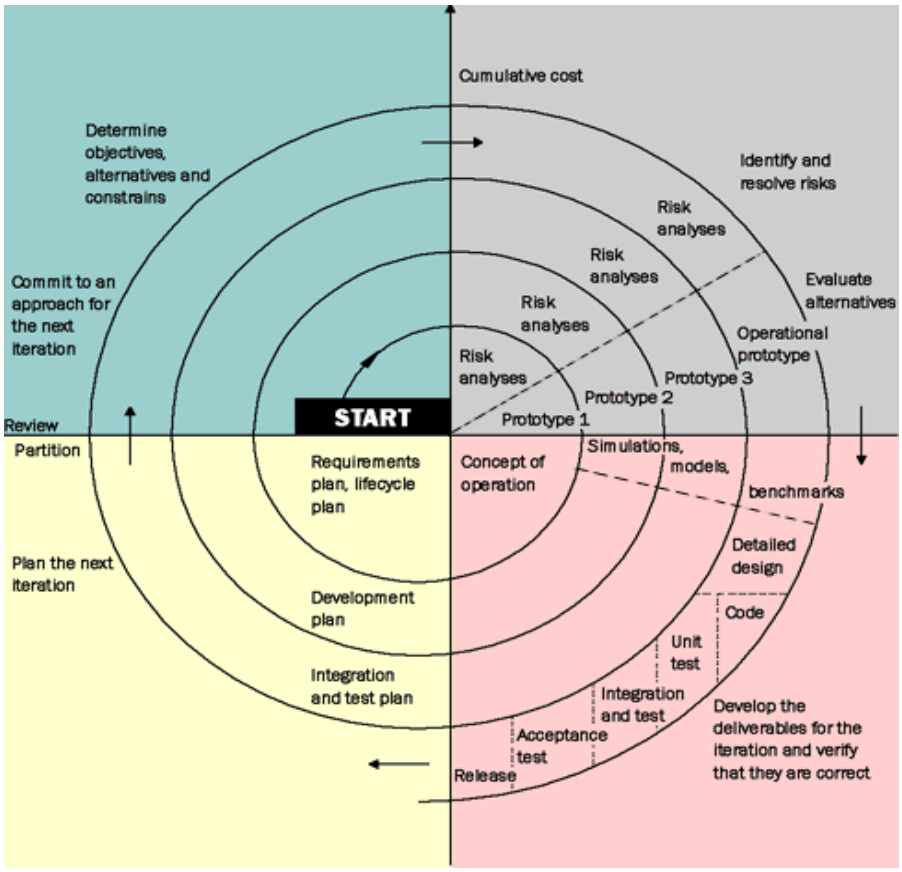




Chapter 7
System Implementation

Methodology Adopted

In this project Spiral model have been followed as “Waterfall Model”, which is one of the oldest and most simple model designed and followed during software development process. But “Waterfall Model” has its own disadvantages such as there is no fair division of phases in the life cycle, not all the errors /problems related to phase are resolved during the same phase, instead all those problems related to one phase are carried out in the next phase and are needed to be resolved in the next phase, this takes much of time of the next phase to solve them. The risk is the most important part, which affects the success rate of the software developed by following “The Waterfall Model”. In order to overcome the cons of “The Waterfall Model”, it was necessary to develop a new Software Development Model, which could help in ensuring the success of the software project. One such model was developed which incorporated the common methodologies followed in the “The Waterfall Model”, but it also eliminated almost every possible/known risk factors from it. This model is referred as “The Spiral Model” or “Boehm’s Model”. There are four



phases in the "Spiral Model" which are: Planning, Evaluation, Risk Analysis, and Engineering. These four phases are iteratively followed one after other in order to eliminate all the problems, which were faced in "The Waterfall Model". Iterating the phases helps in understanding the problems associated with a phase and dealing with those problems when the same phase is repeated next time, planning and developing strategies to be followed while iterating through the phases. The phases in the "Spiral Model" are as follows"

Plan: In this phase, the objectives, alternatives and constraints of the project are determined and are documented. The objectives and other specifications are fixed in order to decide which strategies/approaches to follow during the project life cycle.

Risk Analysis: This phase is the most important part of "Spiral Model". In this phase all possible (and available) alternatives, which can help in developing a cost effective project are analysed and strategies are decided to use them. This phase has been added specially in order to identify and resolve all the possible risks in the project development. If risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed with the available data and find out possible solution in order to deal with the potential changes in the requirements.

Engineering: In this phase, the actual development of the project is carried out. The output of this phase is passed through all the phases iteratively in order to obtain improvements in the same.

Customer Evaluation: In this phase, developed product is passed on to the customer in order to receive customer's comments and suggestions which can help in identifying and resolving potential problems/errors in the software developed. This phase is very much similar to TESTING phase. The process

progresses in spiral sense to indicate iterative path followed, progressively more complete software is built as we go on iterating through all four phases. The first iteration in this model is considered to be most important, as in the first iteration almost all possible risk factors, constraints, requirements are identified and in the next iterations all known strategies are used to bring up a complete software system. The radical dimensions indicate evolution of the product towards a complete system. However, as every system has its own pros and cons, "The Spiral Model" does have its pros and cons too. As this model is developed to overcome the disadvantages of the "Waterfall Model", to follow "Spiral Model", highly skilled people in the area of planning, risk analysis and mitigation, development, customer relation etc. are required. This along with the fact that the process needs to be iterated more than once demands more time and is somehow expensive task.

Chapter 8
System Testing

Methodology Adopted for Testing

Testing is a process of executing a program with the intent of finding errors. A good test case is one that has a high probability of finding an as yet undiscovered error. A successful test is one that uncovers an as yet undiscovered error.
If testing is conducted successfully it will uncover error in the software and testing demonstrates that software functions appears to be working according to specification, that behavior and performance requirements appear to have been met. In addition, data collected as testing provide a good indication of software reliability and some indication of software quality as a whole. But testing cannot show the absence of errors and defects, it can only show that errors and defects are present.

- **Testing Principles:**

All tests should be traceable to customer requirements.

- Test should be planned long before testing begins.
- Testing should begin in small scale and progress towards large scale.
- Exhaustive testing is not possible.
- To be most effective testing should be conducted by independent third party.

- **Testing Methods:**

Test must be designed with the highest likelihood to find possible errors in the system to avoid major problems before the system goes live. There are two methods to design the test cases.

White Box Testing:

Glass box testing is a test case design method that uses the control structure of the procedural design to drive test cases. Using white box testing the software engineer can derive test cases that

- Guarantee that all independent paths within the module have been exercised at least once.
- Exercise all logical decisions on their true and false side.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to ensure their validity.

Black Box Testing:

Black box tests are used to demonstrate that software function are operational, that input is properly accepted and output is correctly produced. It is also used to demonstrate that integrity of external information is maintained. Black box test examines some fundamental aspects of system with little regard for the internal logical structure of the software.

Verification and Validation:

Verification refers to the set of activities that ensure, software correctly implements a specific function. Validation refers to different set of activities that ensure, the software that has been built is traceable to customer requirements.

Types of tests

- **Alpha testing:**

It tests the software at the developers' site. Software testers conduct the tests using information about customer requirements. It is usually done in the presence of the developer.

- **Beta testing:**

It tests the software at the clients' site. End user performs testing in absence of the developer and list down all the errors and problem occurred during the testing.

- **System testing:**

Tests to examine compatibility of software with hardware such as CPU, RAM, and disk drives etc.

- **Recovery Testing:**

Uses tests cases to examine how easily and completely system recovers from disasters such as power failure, or disk crash or any natural disaster.

- **Performance Testing:**

Tests the performance level of the software when load is low and when load is heavy or regular and records the amount of resource that the software uses.

Software Testing Strategies Adopted

The following are the different perspectives tested:

- Internal program logic by "white box" test case design techniques.
- Software requirements using "black box" test case design techniques.

After testing individual components integration was done. After integration testing of the software was tested as a whole. Finally, a series of tests were executed once the full program was in operation.

Unit Testing*:*

All units testing will be done in White Box fashion. Testing will be conducted using Basis Path testing methods, because of its simplicity and high effectiveness. Loop testing will also be conducted to compliment the Basis Path testing. Individual components are tested separately. Due to the system's modular design, there is no need for test beds.

Integration Testing:

The interface of the modules is tested. Using black box techniques the interfaces and the linking of different modules is checked for flaws. Any corrections to faulty interface are made.

Functional Testing:

Testing will be done in a black box fashion to check the functionality of the system as a whole. Using different test cases the logic of the system is checked for flaws.

System Testing:

In System checking, the software as a whole is observed and checked to see if it meets the user requirements. We tested it again using actual real life scenarios and data.

Project Test Cases:

Experiment 1 : With Oversize of Data

- This experiment have been performed to check the physical memory of the solr in the local machine as it is running in the local system so it is important step to check the limitation of the storing data.
- As the number is documents of data increases in the solr the size of "JVM" and physical memory of solr also increases.
- This causes the "Leaking into the solr memory".
- This is very unique problem into the solr as it uses RAM is it's processing so when RAM gets full at that particular period of time then that problem arises again.
- As the size of data increases the processing speed of the solr also decreases for this it is required to have huge size of processing memory (RAM) and processor and storage memory.
- Uses on local system can be reduced to get quick response.

Experiment 2 : With Solr Query

- This is have been experienced that if the response handler of solr is not properly configured then big query could cause slow processing in the sense of query response.
- This also depends on the amount of the data stored on the solr.
- Before firing lengthy query it is important to configure schema so that the query syntax can match with the solr syntax.

Experiment 3 : With Query Processing time

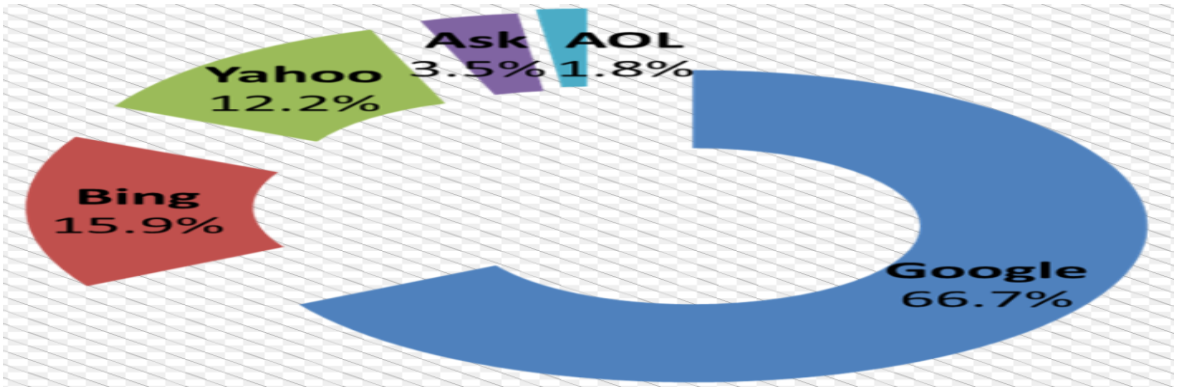
- It have been experienced that solr gives slow response on local machine due not having high performance system also in the situation when solr have to retrieved multiple results from the multiple resources or cores.

Experiment 4 : With the Calculation of tf & idf and Score Boosting

- As if the response handler of solr schema have not been configured then there might be a case arise when result will not in proper format or it will not be ranked. In order to give Rank the solution is to apply "Solr Boosting" in the query and this can also be done in the solr Schema.

Numerical comparisons in terms of speed and Popularity

- **Popularity with market Share :**



- **Speed Comparison :**

Sr No.	Search engines name	Avg. Execution Speed	Reason
1	Google	~4.24/relevancy query response.	Uses of high performance system, algorithms.
2	Bing	~4.30/ relevancy query response.	Uses of high performance system, algorithms.
3	Yahoo	~4.50/ relevancy query response.	Uses of high performance system, algorithms.
4	Multi-source Search	~5.60/ relevancy query response	Low performance system, low budget, poor scalability etc.

Chapter 9
System Maintenance

System Maintenance:

Maintenance of this project system will be done by removing any residual errors etc. system maintenance is concerned with making changes into the system after it has been delivered. These changes enhance the software functionality and increase the service that the product offers. Here in the context, two types of maintenance can be provided.

Corrective Maintenance:

Corrective maintenance is concerned with the removal of errors discovered after the delivery of the system for regular usage. The removal of such errors requires changes in the existing system.

Adaptive Maintenance:

Adaptive maintenance is concerned with the changes into the system due to change in the environment in which the system functions.

Chapter 10
Conclusion

Conclusion

After the successful completion of the project the surety that was given that the problems in the existing system would overcome using different algorithms and approach. Developing Multi Source Search- Information Retrieval was a learning experience as well as the finding out the new approach in order to get the familiar in this fields. As the development of this project helped to understand the complexities involved in this domain, more importantly it helped to learn and understand the importance of proper planning and the need to properly understand the user requirements while developing the system.

Since this project is based on the information retrieval multi-source search engine. Indexing of the three XML data files from Wikipedia website and return search results for free text queries. Uses of the Apache Solr to create search engine. The aim of the project is to provide user a complete Wikipedia search experience. Results from all three data files will be related to each other and will be shown to the user on a single interface.

Users do not have to check all the links to find out the appropriate result based on the search. Single link is provided to retrieve the best result required by the user. They can compare the result from the three sources i.e. from Wikipedia, Yahoo and Bing as per the requirements. Enable users to enter search criteria once and access several search engines simultaneously.

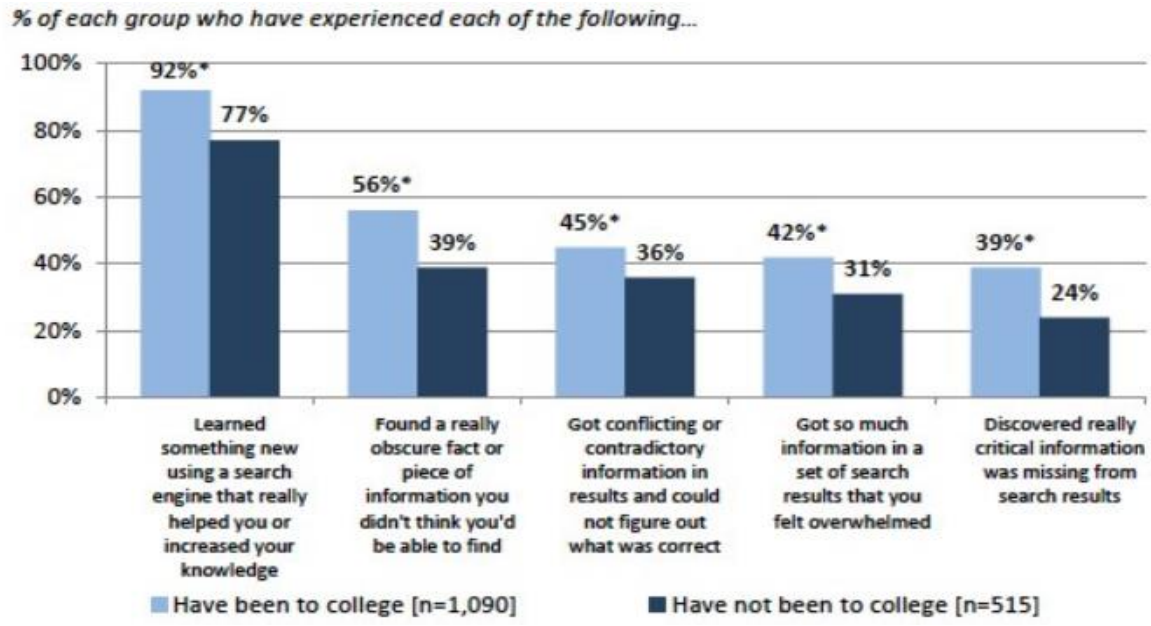
It is expected that this project will go a long way in satisfying user's requirement. It will also go a long way in helping the establishment section work more effectively, accurately and efficiently.

Limitations:

- As there can be possibilities that user can experience limited search in response. As this is initial phase project so if there will be any chances to enhance this in future then it can very easily.
- Since the indexing is the most important feature of any search engines which are not very easy to implement as indexing required to implement tokenizer rules there are nearly 20rules, Stammers, Wiki indexing, Maintaining dictionary of each terms and document based on the calculation of the term frequency (td) and inverse documents

frequency(idf). So implementing indexing is not at all easy task as there are many more fields which need to implement in order to get full- fledged indexer.

- Ranking of pages can be more improved by applying different algorithms. At this moment we are applying BM25 algorithm to get maintain the ranking of the pages and boosting of the scores are also playing an important to get correct ranking of the pages.
- Crawler performance can be improved in future using enhanced version of the algorithms.
- As the solr do not respond images henceforth no any images are shown on the UI so to get this facility different unique handler and database can be implemented which can then integrate with solr to give the images on the UI.



The above statistics show how much important is getting correct information about anything. In today's world this is very important to get the knowledge f everything by just type some keywords in one textbox. Today search engine fields is booming very drastically as from the above stats 92% of the college students uses search engines to get more knowledge and 77% of the people who are not in the college but are using search engines to enhance their knowledge. All this cases tends to make use and demand of knowledge in the

market. Henceforth today in the world company are competing each other to give best services to the users. As the demands increases it becomes bottleneck for the company to fulfill those demands of the users also they uses very high level of technology to provide quick response to the users. Some of the popular search engines Google, Bing, Wikipedia, Yahoo, Yandix , webopedia, Ask.com these search engines have their own specific way of presentation to provide services to the users.

Future Implementation

1. **Search as you type feature:**
   Basically, as you type in the search box, the result set is continually updated with better and better search results. The idea that we hit on to implement this is to use wildcards and user's current input into the search. If the user types "appl" then the query is appl + appl*. In the first case token "appl" returns nothing and the wildcard "appl*" at least returns reasonable results. If the user enters "apple", the query is (apple+apple*) and the search term "apple" matches and the results all come back correctly ordered.

2. **Relevance feedback:**
   We had the idea to implement some sort of feedback mechanism based on click through. A click on a document could have been used to boost the score for that document.

3. **Display pictorial data and essential info from info box:**
   Most Wikipedia pages have info boxes and sometimes a photograph of the location. We could have captured that information and displayed it to the user as essential information along with the search results.

4. **Auto suggestion of the related topics :**
   When the user type query he/she expects to get more suggestions about the good and relevant pages which they are looking for. In that case search engines should fulfill all this requirements of the user. This can be done using some machine learning techniques.

5. **Fully interactive Search engines :**
   Many times user do not wish to read whole paragraphs of the page in that there should be a feature which can read all the results page based on the user needs. This is not easy to implement but this can be taken into consideration according to user point of view.

Bibliography

I gathered the information required to understand the project concept and the ideas used, from various online resources such as Wikipedia, Lucene, and from various articles from web. URL to some of these articles are as follows :

Web References :

[1]http://cdn.oreillystatic.com/en/assets/1/event/61/Solr%20Application%20Development%20Tutorial%20Presentation.pdf

[2] http://people.apache.org/~hossman/apachecon2008us/ootb/apache-solr-out-of-the-box.pdf

[3]https://cwiki.apache.org/confluence/display/solr/Field+Types+Included+with+Solr

[4] Eric Prudhommeaux, Presentation of W3C and Semantic Web, 2001, http://www.w3.org/2001/Talks/0710-ep-grid

After reading these articles and finalizing the technologies to be used I had further referred to the following IEEE papers and Books for gaining in depth knowledge of the selected technologies. These titles are

IEEE Reference Papers :

[1] Apache Lucene 4 - Andrzej Białecki, Robert Muir, Grant Ingersoll Lucid Imagination

[2] NLP techniques for improving search of side effects in patient discussions - Kameswara Rao Bh, Venkatesham G, *iGATE Global Solutions,*

[3] A NEW APPROACH TOWARDS VERTICAL SEARCH ENGINES Intelligent Focused Crawling and Multilingual Semantic Techniques - Sybille Peters.