Search Engine

CS172

By:

Katherine Gallaher

and

Shilpa Chirackel

Partner 1: Katherine Gallaher kgall005@ucr.edu

Partner 2: Shilpa Chirackel schir001@ucr.edu

Collaboration Details:

We divided up the work evenly. When one partner got stuck debugging, the other partner would step in to try to fix the problem. We used github to host our project so each of us could have up-to-date code at any moment. We met up on a regular basis to work on the project side by side to help each other when any problems encountered. We both used a linux and Mac environment and vim to code.

The specifics are as follows:

Shilpa Chirackel: Shilpa mainly worked on Main.java, SearchQuery.java and the web application. She decided to base SearchQuery.java on the code that wasy given on ilearn. However, instead of returning a topDocs a String is returned in order to use for output onto the webpage. For the web application, she decided to base it off of the example given on ilearn. She decided to use the JSP example that was given. She uses our SearchQuery.java file in order to get the results and output of the user input query.

Katherine Gallaher: Katherine mainly worked on Parser.java and Index.java. Both of these files are based on the sample code that was given on ilearn. In Parser.java, she decided to use jsoup in order to parse the html files to get the wanted information, including title and body. In Index.java she decided to use Lucene version 34 in order to use the sample code given.

As stated before, the items mentioned above are not limited to the particular partner. Each individual worked on each aspect of the code one way or another not limited to implementation strategies, implementation, and debugging.

Overview of The System

Architecture

We wrote our program in Java, and we did not use an IDE until we created the web application, for that we used Eclipse version Luna. Instead, we used the Vim editor and the command line to compile and run our program. We were working on a Linux and Macbook air environment. The version of Java that we used is 8. We used Lucene version 3.4, Jsoup 1.8.1 and Tomcat version 7. Our program is divided into four classes, Main.java, Index.java, Parser.java and SearchQuery.java. We separated the assignment into two parts. The java program which encompses of Main.java, Index.java, and Parser.java, creates the index file. The second part consists of our web application which uses SearchQuery.java to create a website that reutrns appropriate results.

Main.java is where our main method is located. To run the program the user types ./run.sh <output-dir> where output-dir is the directory that contains the html pages that will be indexed. The html pages must have the ':' and '/' characters replaced with '_'. When running the script an index will be created from all of the html pages in the output folder. The output folder should be put in the current directory.

In Parser.java we have one method, public static Document loadDoc(File child), this method returns a lucene document that will be used for indexing. First, we use jsoup to parse child file, from there we retrieve strings for the title and body. Our body only contains text that belongs to paragraphs. We retrieve the url by getting the name of the file and using replace to

change the first '_' to a ':' and then use replace again to change the remaining '_' to '/'. From all of these strings we create a lucene document analyzing the title and body and return the document. We analyze the title and body only because that is what we want to search in order to find the query. We also detect for duplicated pages in here. We add the urls into a hashSet so that we know that urls are unique. If the url is in the hashSet, we don't index this particular page.

In Index.java we have one method, void createIndex(File dir), which creates the inverted index. In this method we loop through all of the html files in the output directory and create documents, using loadDoc, and add these documents to the index.

In SearchQuery.java we have one method, public static String searchQ(String queryString, int topk), this method searches through the index that was created in createIndex for the queryString. It returns a String that is output onto the webpage. The string outputs each result on it's own line, the text is the title of the page and when you click the text you are taken to the URL.

For the web page we used the JSP sample code given on ilearn for. In the jsp file we created an instance of our SearchQuery class and called the searchQ function. the searchQ function returns a String that has all of the result information and clickable links. We print out this information in our jsp file. In our actual web page we have a textbox when the user inputs a query in the textbox and presses the submit button the input is saved as the queryString and passed into searchQ.

Index Structures

For this project we used Lucene version 34. The index that is created from all of the html files is an inverted index created by lucene. This inverted index is created based on the frequency of the term in its specific document.

Search Algorithm

We used Lucene version 34 in order to search the index that was created. The BM25 ranking method is used in Lucene in order to rank documents. This method of ranking gives a higher score for the title than the body of the page. Lucene's search returns a TopDocs that is a list of documents in order of highest relevance to lowest relevance.

Limitations (if any) of System

You must delete the testIndex folder, if it exists, before each run of the script or else errors will occur.

A limitation is that we only display a maximum of 10 results in the web page. If the user wants more than 10 results they must manually go into SearchQuery.java and change the code. We are using a deprecated version of some Lucene functions, however this does not affect our project because it still works properly.

Instructions on how to deploy system

We created a shell script to create the index folder.

In the command prompt window locate the files and use the following command:

[user@server] ./run.sh <output-dir>

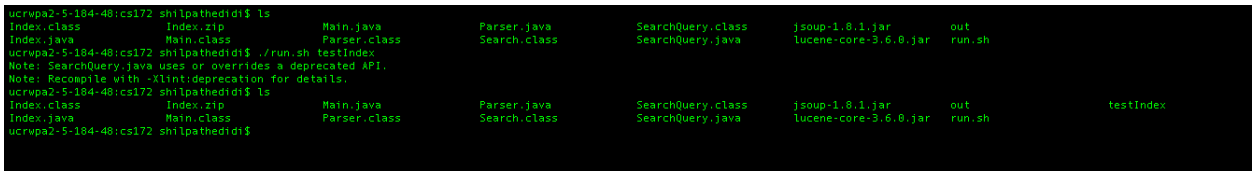
This is how we compiled and ran our code without the script:

Compilation: java -cp jsoup-1.8.1.jar:lucene-core-3.6.0.jar:. Index.java Parser.java Main.java

Run: java -cp jsoup-1.8.1.jar:lucene-core-3.6.0.jar:. Main <output-dir>

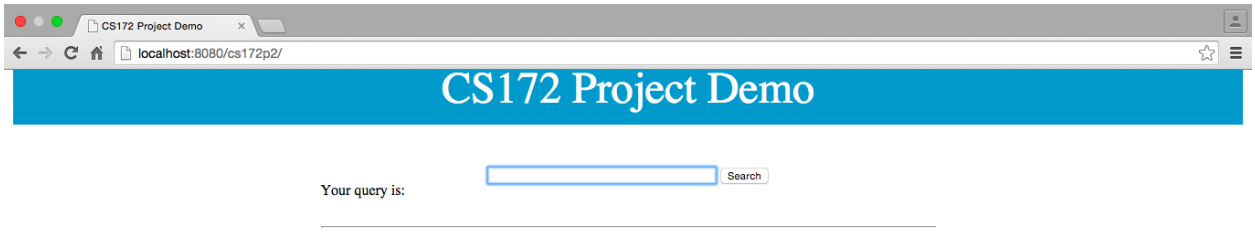
In order to use the web page for searching. The user has to manually place the index folder in the tomcat/bin directory. Also, we have exported our Eclipse project as a .war file, we then put this .war file in tomcat/webapps directory. In order to open the web page we have to start up tomcat by using ./startup.sh and then we have to deploy the war file by going into manager gui and give the path name for the .war file and click deploy. Then on google chromium input localhost:8080/cs172p2/ into the search bar which will take your to our website where you can input your search.

Screenshot of System at action



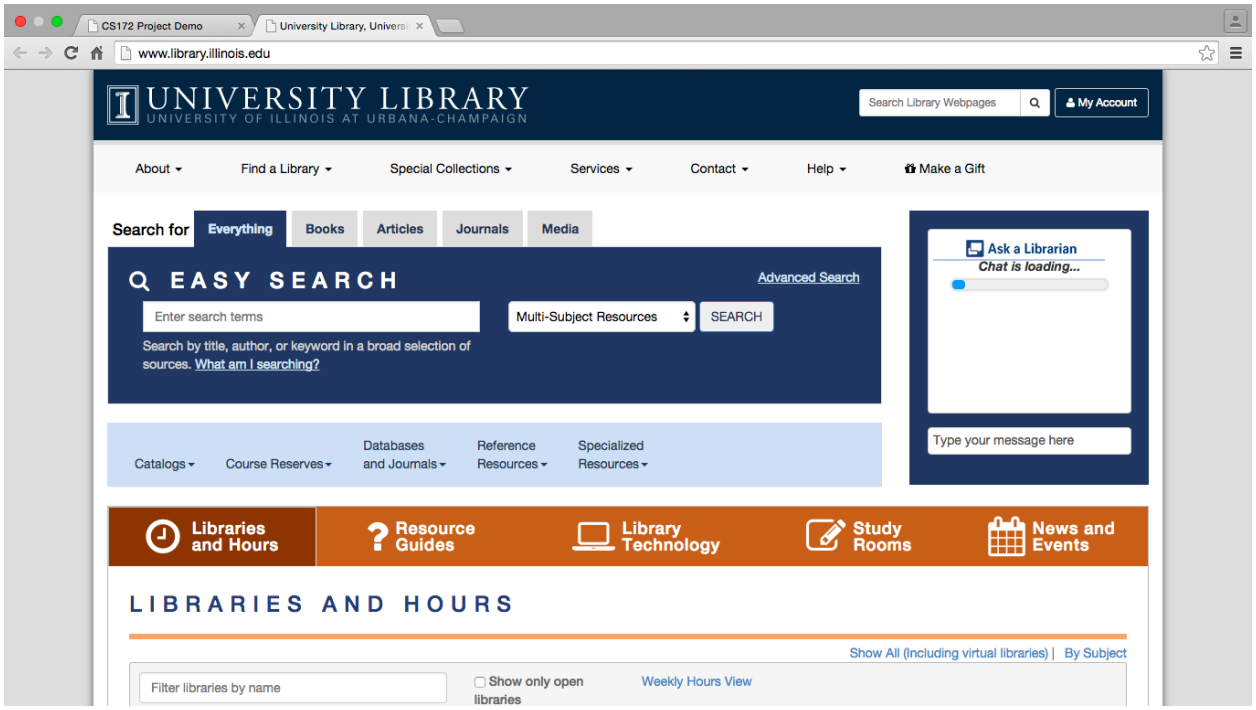
Above screenshot is creating the testIndex.

The following screenshots show the search engine in action:

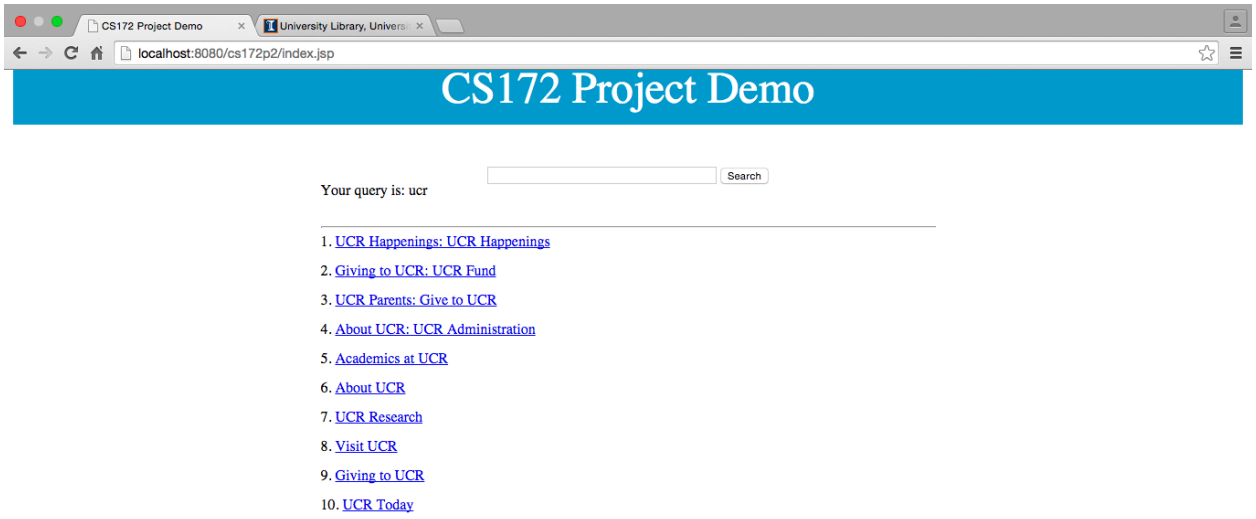


Before any query is input.

Using a query with no results.



Page we went to by clicking on a result.



Results of query: ucr.