Text Retrieval System

by

Bill Qualls

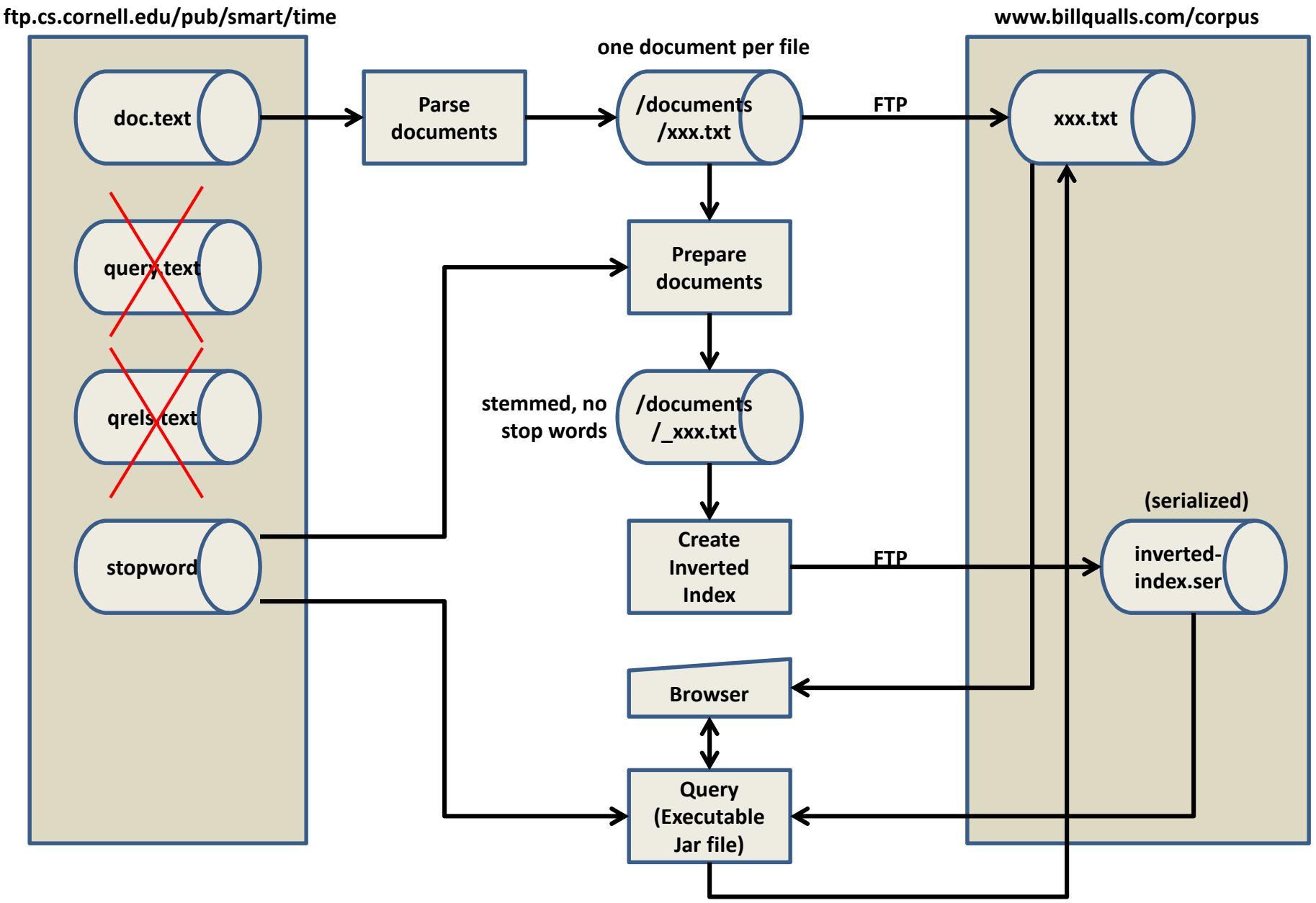
DePaul University

CSC575 – Intelligent Information Retrieval

Dr. Bamshad Mobasher, Professor

Winter Quarter 2012

Contents


ftp.cs.cornell.edu/pub/smart/time
www.billqualls.com/corpus
one document per file
doc.text
Parse documents
/documents /xxx.txt
FTP
xxx.txt
query.text
Prepare documents
qrels.text
stemmed, no stop words
/documents /_xxx.txt
(serialized)
stopword
Create Inverted Index
FTP
inverted-index.ser
Browser
Query (Executable Jar file)

Source of Data

- This project used the Time Magazine data found at ftp://ftp.cs.cornell.edu/pub/smart/time.
- All documents were contained in a single file, **doc.text**. A program was written to parse this file into one file per document.
- The site contained **query.text** with 83 sample queries.
- The site also contained a **qrels.text** which was supposed to indicate relevant documents for each query, but this file was clearly flawed.

Program 1: ParseDocuments

Program: ParseDocuments.java
Purpose: Create a separate file for each document
Input: doc.text (unzipped from doc.text.Z)
Output: project/documents/xxx.txt (where xxx = three digit number, such as 017).
First line each file is information about the source, as contained in the input file.
Output folder must exist before running program.

Program 1: ParseDocuments

Sample Input

*TEXT 017 01/04/63 PAGE 020
THE ALLIES AFTER NASSAU IN DECEMBER 1960, THE U.S . FIRST
PROPOSED TO HELP NATO DEVELOP ITS OWN NUCLEAR STRIKE FORCE . BUT
EUROPE
.
.
GENERAL ASSEMBLY, AND IS THUS EQUAL IN VOTING POWER WITH SUCH
NUCLEAR
GIANTS AS THE SOVIET UNION AND THE U.S .
*STOP

EOJ messages

23053 lines written to 425 files.
Normal end of program.

Program 2: Stemmer

Program: Stemmer.java
Purpose: Static method to stem a String
Input: aString
Output: aString, stemmed.
Notes:

I download a Java implementation of Porter's Stemming Algorithm from http://tartarus.org/martin/PorterStemmer/java.txt.

Its main method was written to take a list of files as arguments from the command line. I modified the main() method to take the files from a static array to simplify testing while in Eclipse. (cont.)

Program 2: Stemmer

I was surprised that there wasn't a method to stem a String, and I'm going to need one so I can stem the query. So I am added a static method stem(String s) which will return a String.

Also, as written, it does NOT remove punctuation. I would prefer to leave some punctuation in (example: I would want slashes in dates) but I am not going to spend too much time rewriting this program. Instead, I will simply remove all punctuation.

Sample usage

```
String stemmedString = Stemmer.stem(rawString);
```

Program 3: Stopword

Program: Stopword.java
Purpose: Static method to work with stopwords
Input: project/stopword
Output: depends on method called
Notes:

Constructor will read stopword file found at ftp://ftp.cs.cornell.edu/pub/smart/time/stopword.

Sample usage

```
boolean isStopword = Stopword.isStopword(word);
```

Sample usage

```
String noStopwords = Stopword.removeStopwords(line);
```

There is a simple main() method for testing purposes.

Program 4: PrepareDocuments

Program: PrepareDocuments.java
Purpose: For each .txt file in the indicated directory, remove stopwords and stem
Input: documents folder, populated by ParseDocuments.java (above)
Process each file, xxx.txt.
Output: xxx.txt becomes _xxx.txt

EOJ messages:

```
There are 423 input files.
22972 lines written to 423 files.
Normal end of program.
```

Token Class

Program: Token.java

Purpose: A very simple class I created to make use of HashMap more clear. All this class contains at this time is the token (term).

TokenInfo Class

Program: TokenInfo.java

Purpose: As presented in class. Documents where this token appears. Minor changes such as adding a toString method.

TokenOccurence Class

Program: TokenOccurence.java

Purpose: As presented in class. Information about an occurrence of a token (term) within a document. Minor changes such as adding a toString method.

DocumentReference Class

Program: DocumentReference.java

Purpose: As presented in class. Store a reference to a document. Minor changes such as adding toString() and hashCode() methods.

InvertedIndex Class

Program: InvertedIndex.java
Purpose: A class which holds the inverted index.

All classes used within InvertedIndex (Token, TokenInfo, TokenOccurrence, and DocumentReference) implement Serializable so the index itself can be serialized.

Index is created in program CreateInvertedIndex.java.

I have been doing a lot of work with JSON recently, so I added a method to write the index as a JSON object (mostly out of curiosity, but it came in handy for checking my index!)

Program 5: Create Inverted Index

Program: CreateInvertedIndex.java
Purpose: Create inverted index file from each _xxx.txt file in the indicated directory.
Input: documents folder, populated by PrepareDocuments.java (above)
Process each file, xxx.txt.
Output: InvertedIndex.txt (inverted index written to a file for checking purposes)
InvertedIndex.ser (serialized version, so it can be read into the search engine)

Developed by following the instructions contained within the "Implementation Notes" powerpoints provided by instructor.

Program 5: Create Inverted Index

EOJ messages

```
424 files indexed.
JSON version of index written to
C:/.../project/inverted_index.json
Serialized version of index written to
C:/.../project/inverted_index.ser
Normal end of program.
```

Program 6: QueryApp

Program: QueryApp.java
Purpose: The text retrieval system implemented as an executable Jar file.

Makes use of the files and classes discusses already.

Uses the **tf x idf** algorithm.

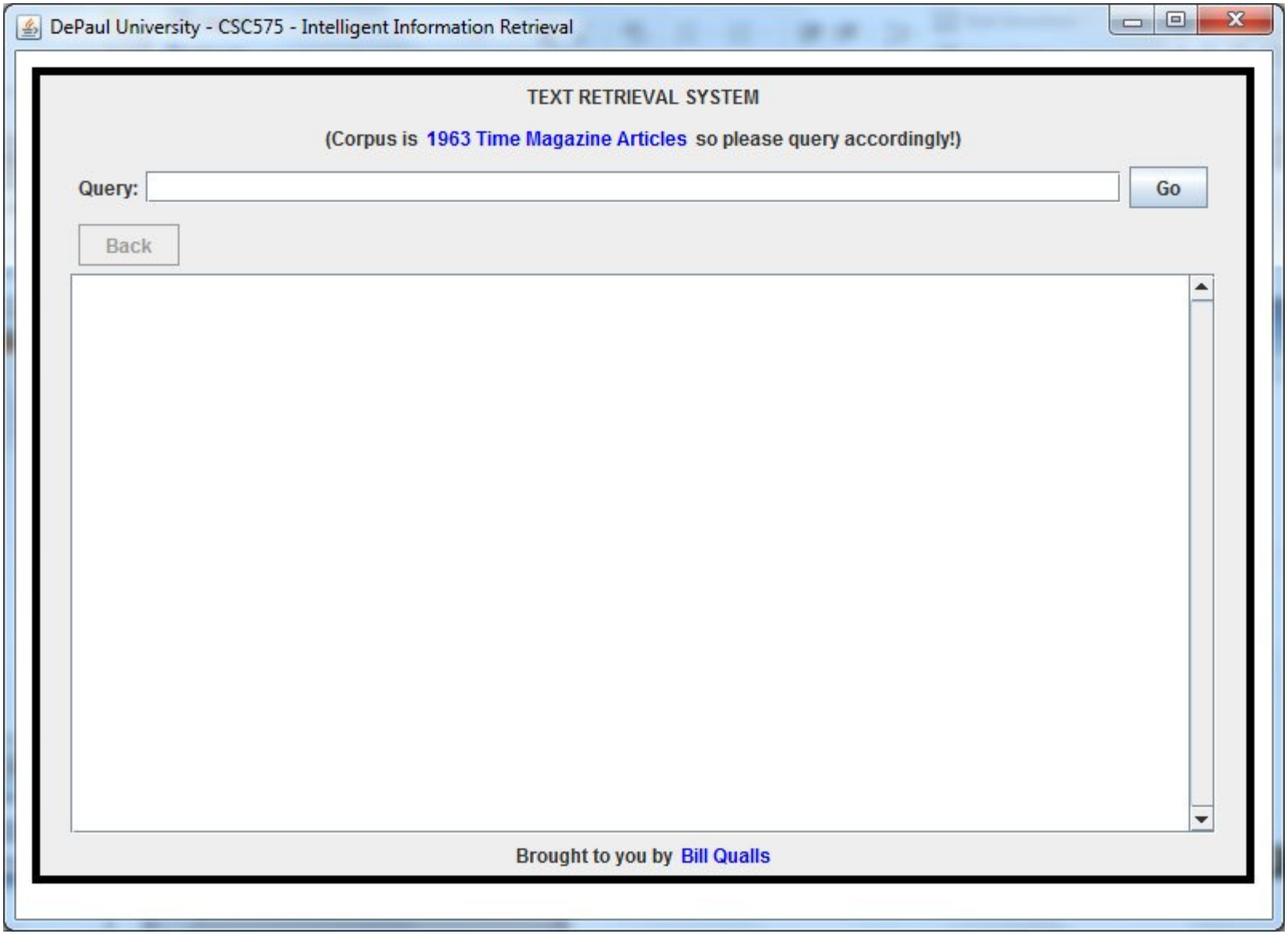
For each retrieval, shows the document number (as a link), the relevance score, and the first two lines of the document.

The actual logic is contained in QueryGUI.java.

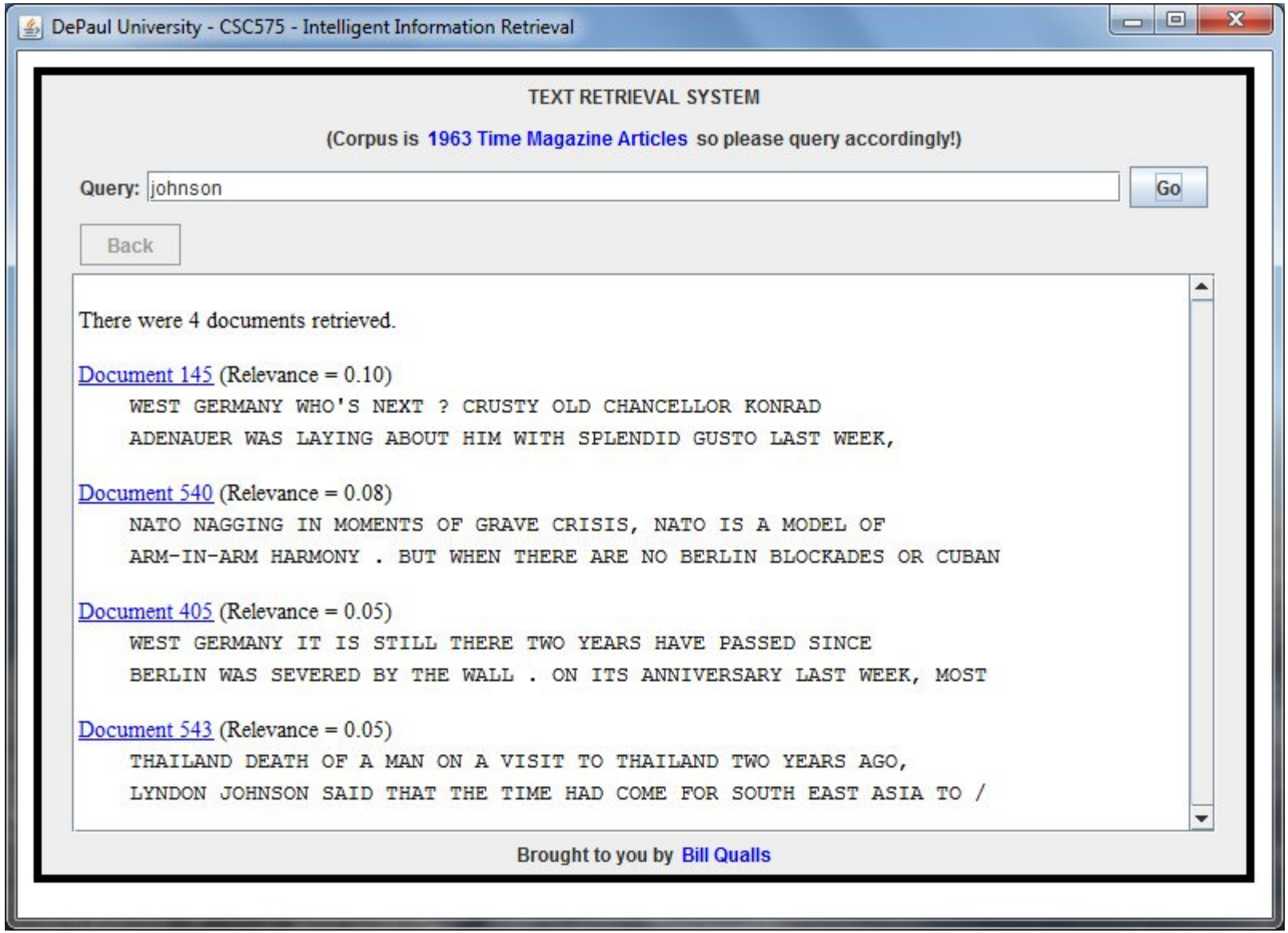
Access this application through the following URL:
http://www.billqualls.com/pa/qualls_query.jar

→ **I have also submitted a .exe version, created with Launch4j.**

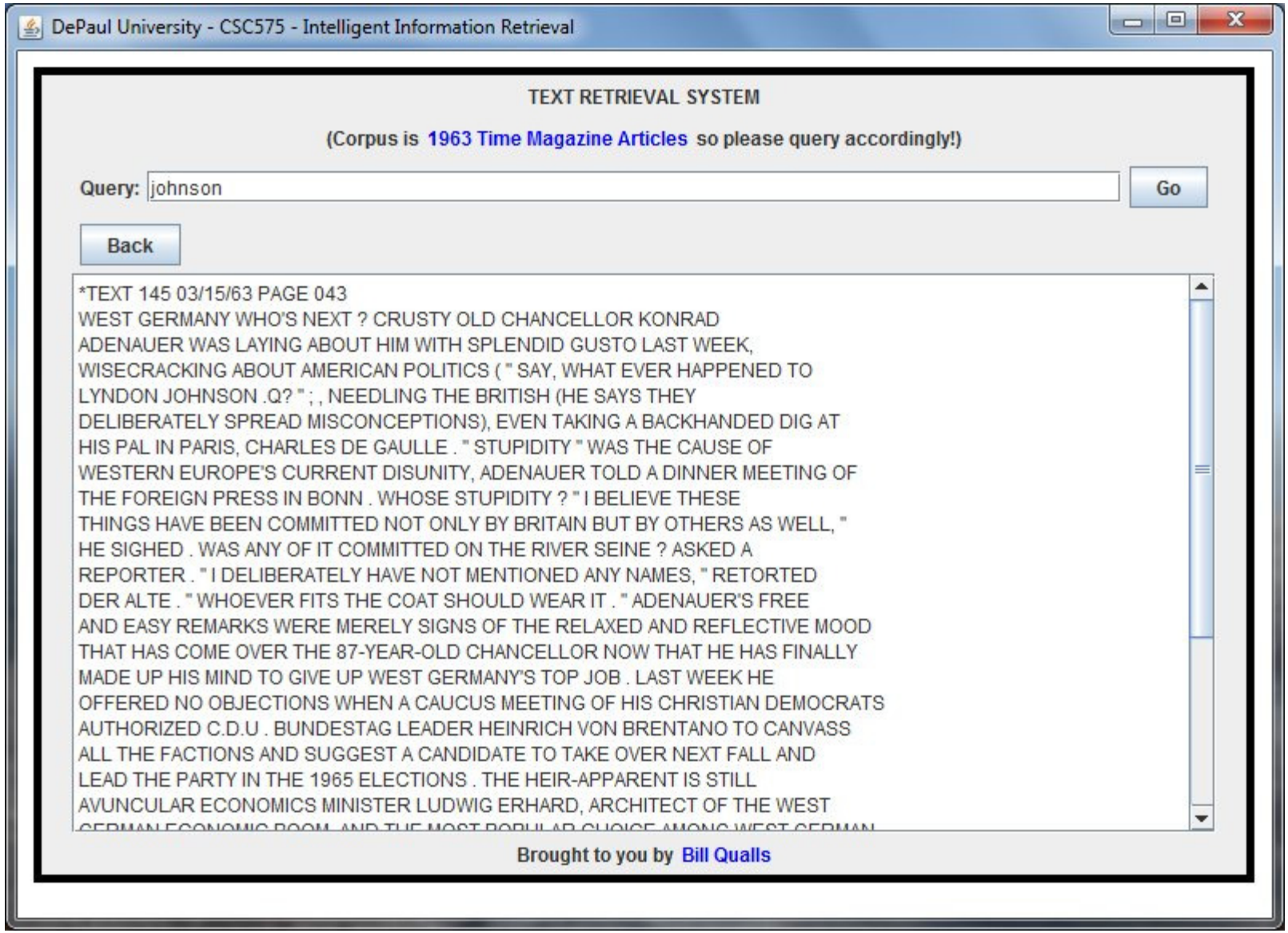
1 of 3: Opening screen



2 of 3: Query Results



3 of 3: Retrieved Document



Program 7: Query (Servlet)

Program: Query.java
Purpose: The text retrieval system implemented as a servlet.

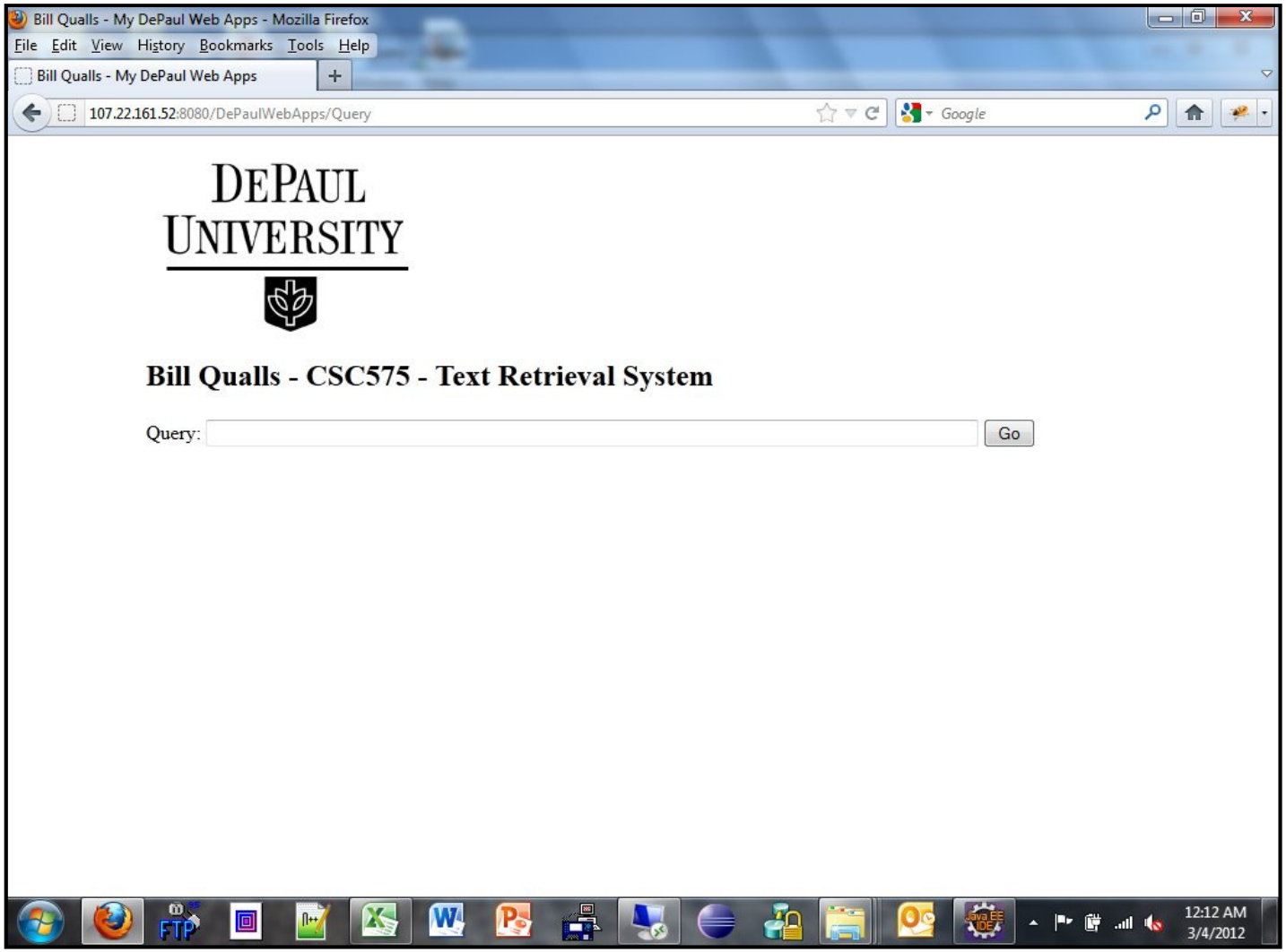
Same functionality, implemented as a servlet.

When the user clicks the document number link, the document itself appears in a separate window.

Access this servlet through the following URL (not available at night): http://107.22.161.52:8080/DePaulWebApps/Query

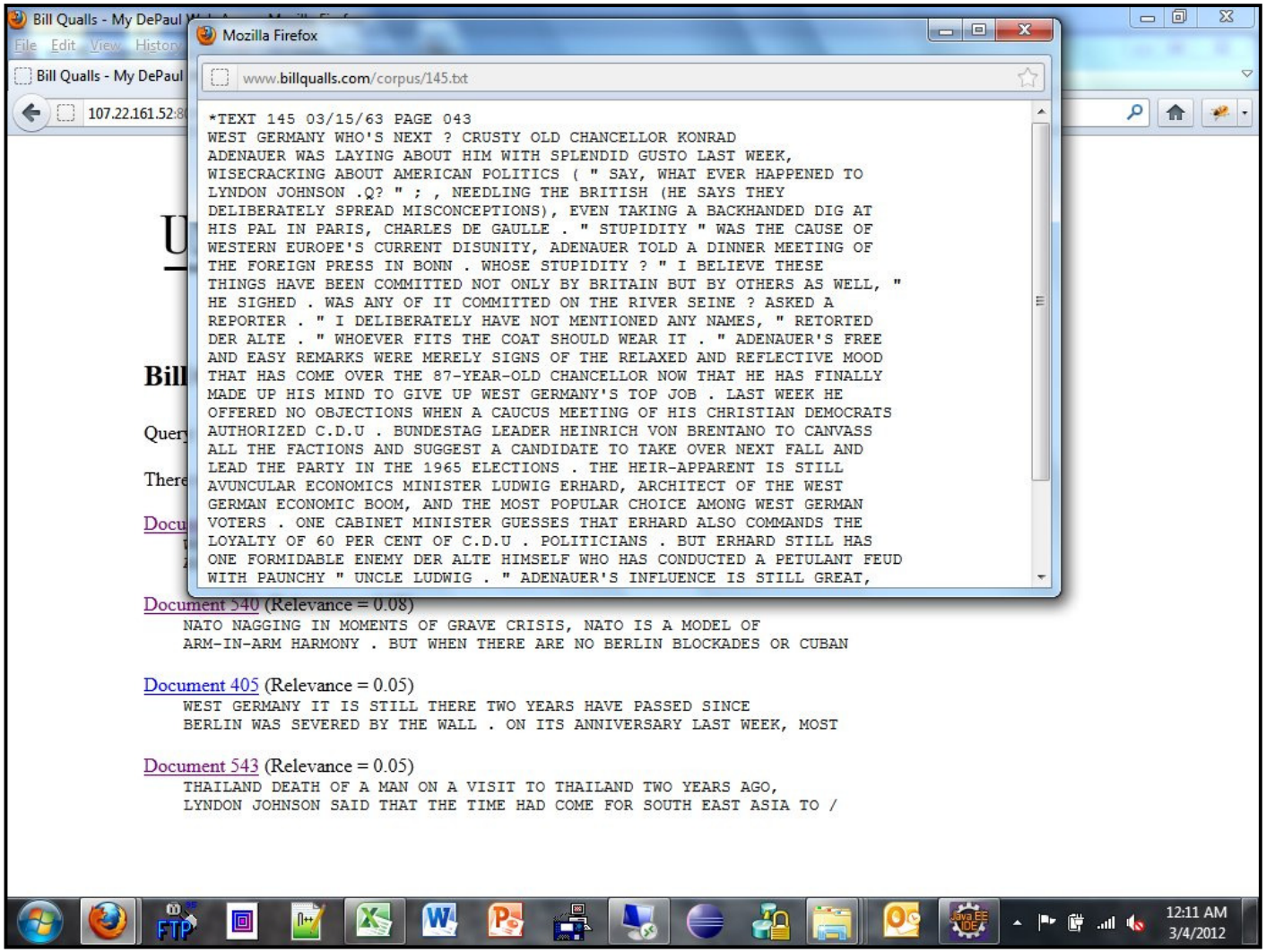
(I originally wrote this servlet, but since my current service provider does not support servlets, I cannot guarantee it will be available for viewing, so I created the Java app version as well.)

Opening screen



After searching for "Johnson"

After selecting first "Johnson" document

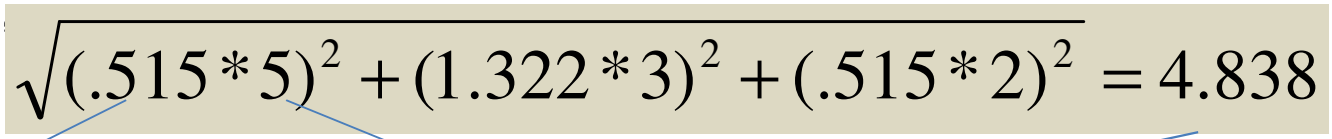


Testing

- I was unable to test with the given queries and exprected results because they were flawed. (Example: queries about Syria, with supposed relevant documents about Viet Nam.)
- So to test this system I used the data contained in homework assignment #2, question 4(a).
- I created ten documents, each with some number of terms "t1" through "t8".
- I was able to validate the index by viewing the JSON output.

- Following are:
  - screenshot of Excel spreadsheet from homework assignment
  - extract of the JSON version of the inverted index
  - example of text files used as input
  - screenshot of the results from retrieving documents with term "t5"

Inverted Index as JSON (partial)

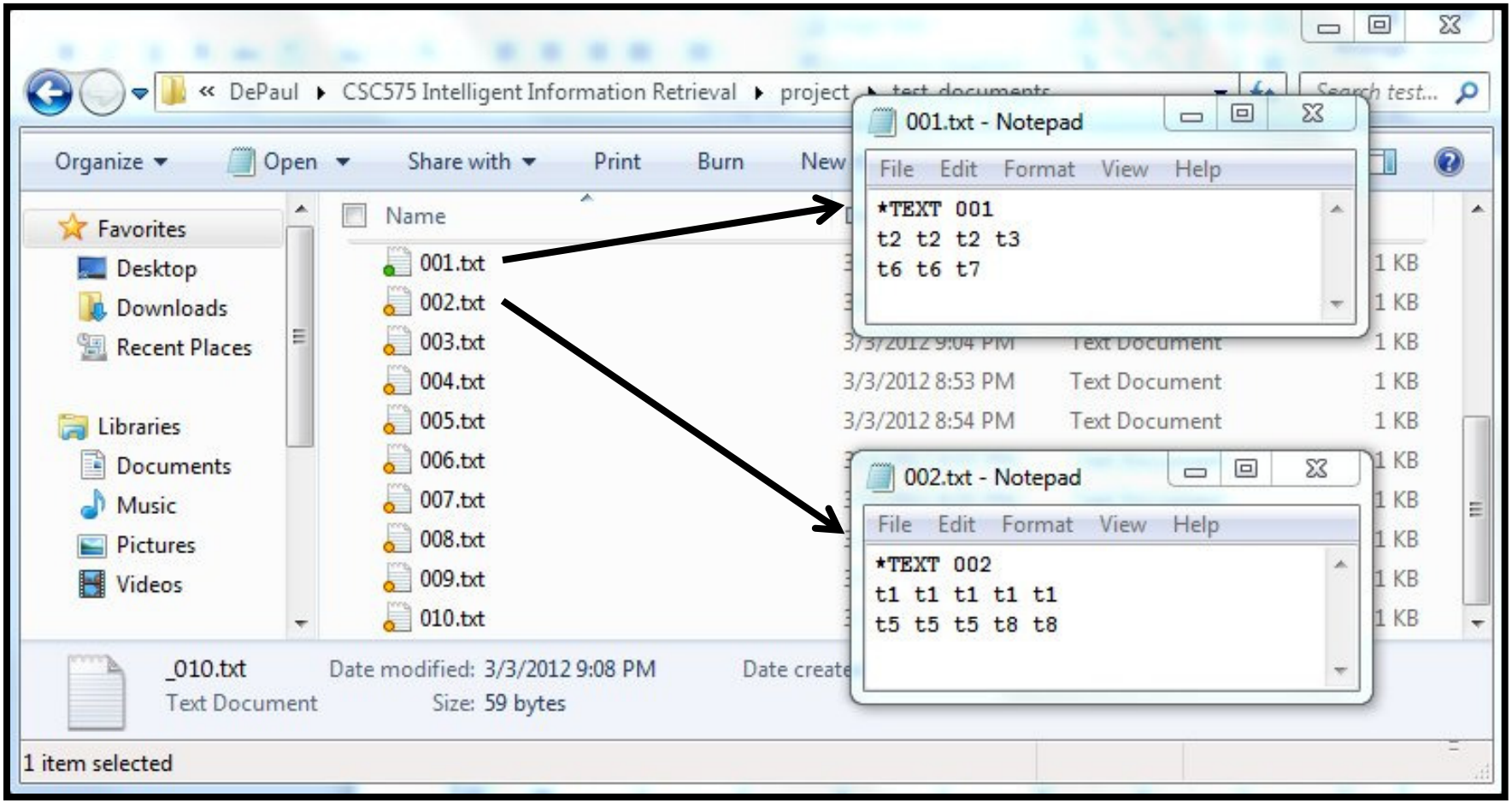



```
{
'documentCount':10,
'index':[
{
'token':'t1',
'occlist':[
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_002.txt','length':4.838},'count':5}},
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_003.txt','length':7.872},'count':3}},
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_004.txt','length':9.477},'count':1}},
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_006.txt','length':3.744},'count':2}},
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_007.txt','length':7.257},'count':2}},
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_008.txt','length':4.332},'count':3}},
{'idf':0.515,'tokenOccurrence':
  {'docRef':{'file':'_010.txt','length':6.481},'count':1}}]
},
etc.
```

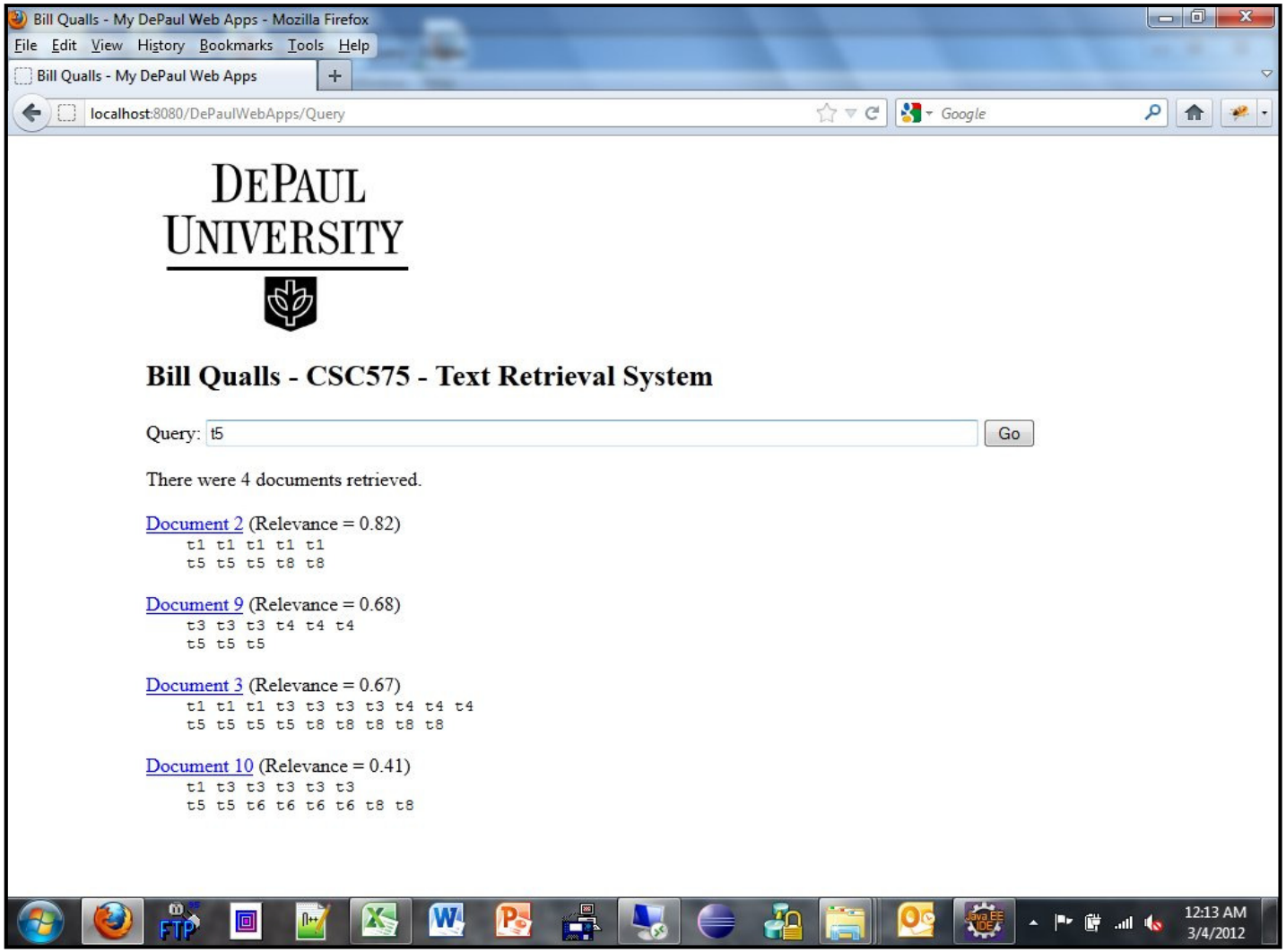
Test data taken from homework #2, part 4a

	A	B	C	D	E	F	G	H	I
1	**TF x IDF**								
2	CSC575 - Homework #2 - Question 4(a)								
3	by Bill Qualls								
4									
5			Number of documents:		10 <--ALL				
6									
7	**docs**	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**
8	D1	0	3	1	0	0	2	1	0
9	D2	5	0	0	0	3	0	0	2
10	D3	3	0	4	3	4	0	0	5
11	D4	1	8	0	3	0	1	4	0
12	D5	0	1	0	0	0	5	4	2
13	D6	2	0	2	0	0	4	0	1
14	D7	2	5	0	3	0	1	4	2
15	D8	3	3	0	2	0	0	1	3
16	D9	0	0	3	3	3	0	0	0
17	D10	1	0	5	0	2	4	0	2
18	10	7	5	5	5	4	6	5	7

Test data taken from homework #2, part 4a



Test data: searching for term "t5"



the end