COMP 4601

Assignment #1

Due Feb 24th 2019 at 11:55 pm

Last updated 02/06/2019 08:32:45

Document changed since last visit

Objective

The objective of this assignment is to have the student develop a distributed search engine using crawled data, document parsing using Tika and analysis by Lucene.

Submission

A README.TXT file must be provided that documents how your assignment is to be tested. It is your responsibility to ensure that the instructions are *accurate, tested, and sufficient to allow the TA to test your submission.*

Submission is through cuLearn. Submission is for groups of 2. One member of the group should submit the assignment archive and the other a README containing the names and student numbers of both students. Package your Eclipse project submission in a Zip file. This zip file must contain a web aarchive (WAR) file for the project indicated below. The WAR file will be used to demonstrate the required functionality indicated in the Requirements section below. The zip file must also include the source code for Eclipse project and a README file. **NOTE: Ensure that your upload is successful. A 10% penalty will be incurred for submissions which do not contain *all* files, or are empty. No late assignments are accepted.**

Testing

Testing will be via demonstration to the TA. Demonstrations will take place by appointment. The TA will post a schedule for time slots. You will have 30 minutes to demonstrate your assignment. The TA will download your WAR file and deploy it to a running Tomcat version 9.0.14 instance. Demonstration will then proceed. The requirements indicated below must be satisfied.

Assignment Details

The following naming conventions **must** be observed:

- The server project name within Eclipse must be COMP4601-SDA.
- The package edu.carleton.comp4601 must be used.
- The main application class implementing your basic RESTful web service must be edu.carleton.comp4601.resources.SearchableDocumentArchive.
- The name of your web service must be COMP4601-SDA. The path to the root web service should be /sda.
- The Document and DocumentCollection classes **must** be used. These classes are in the edu.carleton.comp4601.dao package. **These classes are included in the sda jar file (see below).**
- **You should use the SDA jar in development**. It provides much of the distributed search functionality.
- In order to understand the distributed search engine functionality, read these notes.
- Using "eclipse+apple" as a search query, an example of a search output page can be found here.

Clarifications

- In the discussion below, database refers to your mongoDB database. You must use version 3.69 in your development.
- The word *document* has multiple meanings here:
  - database document -- a document in the mongoDB.
  - Lucene document -- a document in the Lucene index.
  - search document -- a presentation object used to communicate with the user and represented in the Document class provided.

Requirements for assignment

1. You are to develop a client and a server.
   1. The client may be built using: Chrome (version 70 and above), iOS (version 11 or later) or Android (version 8 and above).
   2. The server must be built using Tomcat (version 9.0.14) and Jersey (version 2.27).
   3. The server must use MongoDB (version 3.6.9) and use the appropriate Java driver (version 3.6.1).
2. The SDA stores documents which are both tagged and linked together. Document and DocumentCollection classes are provided. So, a document contains:
   1. name: This is a user-friendly display string for the document.
   2. id: This is a unique number for a document.
   3. content: This is a string that contains the content of the document. It is not structured in any way (for now).
   4. tags: This is an array of strings representing the searchable content for the document. A document must be tagged with at least 1 keyword/phrase. These can be user-provided or inferred (e.g., metadata).

   5. links: This is an array of URLs representing links to other documents. You may assume that these links are relative to the URL of the web service itself. A document may have no links. Links are only present for user-generated documents
3. In the text below, the web service is indicated as: **sda/X, where X is the web service.** You must provide the following RESTful web services:
   1. Create a document and insert it into the SDA. The SDA should indicate success using an HTTP response code of 200 and 204 otherwise. **sda using POST**.
   2. Update an existing document with new tags or new links. The SDA should indicate success using an HTTP response code of 200. **sda/{id} using PUT or POST**.
   3. View a document. **sda/{id} using GET**. You must support XML and HTML representations.
   4. Delete a document, **sda/{id} using DELETE**. If the document exists, an HTTP response code of 200 is returned, otherwise 204.
   5. Delete an existing set of document(s) with specific tags. If one or more documents is deleted, an HTTP response code of 200 should be provided, otherwise 204. **sda/delete/{tags} using GET**.
   6. Search for documents with specific tags. **sda/search/{tags}** (see #21 below)
   7. When the results of a search are displayed, it should be possible to navigate to the linked documents.
   8. View the library: the names of all documents should be displayed. **sda/documents using GET**. You must support XML and HTML representations.
4. When running, directing a browser to http://localhost:8080/COMP4601-SDA/rest/sda should display the text "COMP4601 Searchable Document Archive: **X**" in a browser window. Here, **X** is the name of your group; e.g., "Bob White and Alice Green".
5. When started, the SDA connects to the DistributedSearch (DS) service.
6. When a search request is made the SDA uses the DS web service to search known search engines.
7. When terminated, the SDA disconnects from the DS web service.
8. The **SDA.jar** includes all necessary code for distributed functionality. You must augment your web.xml document in order to provide a general_error.html document that handles HTTP 500 response codes.
9. Assuming that you have understood the function of the SearchServiceManager as described in the slides, requirements 5,6 and 7 will not require substantial software development by you. **Information on the classes above can be found here. Be sure to read it!**.
10. The SDA stores documents which are both tagged and linked together. Document and DocumentCollection classes are provided and **MUST** be used. This is required in order to ensure that the search engines interoperate. The SDA.jar contains these files. You do not need to copy these source files into your project. So, a database document contains:
    1. name: This is a user-friendly display string for the document.
    2. id: This is a unique number for a document.

   3. score: This is the score computed by Lucene.
   4. text: This is a string that contains the content of the document. It is not structured in any way.
   5. tags: This is an array of strings representing the searchable content for the document. A document must be tagged with at least 1 keyword/phrase/field.
   6. links: This is an array of URLs representing links to other documents. You may assume that these links are relative to the URL of the web service itself. A document may have no links.

11. You must demonstrate that you can crawl the following URLs:
    1. https://sikaman.dyndns.org:8443/WebSite/rest/site/courses/4601/handouts/. Ensure that you prevent off site page visits. Ensure that you limit the number of pages indexed!
    2. https://sikaman.dyndns.org:8443/WebSite/rest/site/courses/4601/resources/. This directory contains 1000 hyperlinked files. You should begin your crawl with the N-0.html file. The graph has known structure and content. PageRank values are known for this graph.
    3. A URL of your choice.
12. At the end of the crawl, a single graph must be stored as a database document in serialized format.
13. When crawling, you must demonstrate that your crawl is adaptive; i.e., changes the frequency with which it visits pages depending of the time taken to crawl the last several pages.
14. When crawling, along with HTML documents, you must index binary data; i.e., images (jpeg, tiff, gif, png), pdf, doc, docx, xls, xlsx, ppt and pptx documents.
15. When crawling, image documents referenced within <img> tags with an alt attribute must have those images indexed with a field that includes the tag text.
16. When crawling, the docId determined by the crawler must be used as the document id of the content stored in the database.
17. You must demonstrate that you can parse the crawled pages using Tika. You must extract all metadata and use the AutoDetectParser to generate content.
18. You must index all of the pages stored in your database using Lucene using the StandardAnalyzer.
19. Using the graph from (12), compute the Page Rank for all crawled pages.
20. Your Lucene documents must contain fields for:
    1. URL -- the resolved location of the original document.
    2. IndexedBy -- the identity of the indexer (e.g., Tony). Field identifier is: **i**
    3. DocID -- the id of the document stored within your database (e.g., 101). Field identifier is: **docId**
    4. Date -- meaning the date and time when the document was crawled.
    5. Content -- this is the content returned by the content handler used in the Standard Analyzer. For HTML page this should contain information extracted using JSOUP; e.g., content of paragraph, heading and title tags. For images, this may not

contain much but contains the content of any associated **alt** attribute for the image when available. Field identifier is: **content**

6. Metadata fields -- a field should be created for each piece of meta data; e.g., for a file with a MIME type of image/jpeg the field name would be type and the value would be image/jpeg. Minimally, the MIME type field should be stored; e.g., application/xml. Field identifier is: **type**

21. In the text below, the web service is indicated as: **sda/X, where X is the web service.** You must provide the following RESTful web services:
    1. Reset the document archive. **sda/reset using GET**. This is a testing convenience only. HTML should be returned stating that the reset has occurred (or an error).
    2. List the discovered search services. **sda/list using GET**. This is a testing convenience only. HTML should be returned providing a list of names of the known services found.
    3. Show the Page Rank scores for all documents. **sda/pagerank using GET**. This is designed to test the analysis of the graph obtained from the crawl. A 2 column table of all documents should be returned with the document name and page rank information in the first and second column respectively.
    4. Boost document relevance using Page Rank scores. **sda/boost using GET**. This will re-index the database and apply a boost value to each document field that is equal to the Page Rank score of the document.
    5. No boost. **sda/noboost using GET**. This will re-index the database giving a boost value of 1 to all fields of all documents in the database.
    6. When creating your own documents, the created documents must be analyzed and indexed using Lucene. All tags must be stored in a separate searchable field. This field is given a default boost value of 2 (you may provide a field to specify this value in a form). The field id should be **tags**.
    7. When a document is updated, the index must also be updated; i.e., the old Lucene document must be deleted and a new Lucene document inserted.
    8. Query for documents with specific terms. **sda/search/{terms} using GET**. This provides for distributed search.
    9. You must also support **sda/query/{terms} using GET.** This provides a local document search only. Both RESTful APIs (for requiremens 8 and 9) must support TEXT_HTML and APPLICATION_XML requests.
        - Terms must conform to the format: +term1+term2+ ... meaning: retrieve ranked documents with term1 term2 ... Terms may also include the field names (see notes for format here).
        - Your **sda/search/{terms} using GET** web service must produce 2 forms of output: APPLICATION_XML and TEXT_HTML. The search web service must invoke SearchServiceManager.query(String tags) as this is used during a distributed search. This web service must return a DocumentCollection.
        - The documents found should be returned as a list. You must support XML and HTML representations. The XML must use the representation derived

from the Document and DocumentCollection classes. You will need to use the SearchResult class in order to coordinate and aggregate content retrieved during the distributed search. These classes are included in SDA jar. Slides 10, 11 and 12 in the associated assignment notes are crucial here.

- When searching, *all* known registered search web services must be queried. You may wait up to 10 seconds for other SWS to return content.
- The SearchServiceManager provides the distribution functionality. You must use it. It is included in SDA jar
- Clicking on a queried document displays its contents. You must support XML and HTML representations.
- If no documents are found, "No documents found." should be displayed. This will be returned in either plain text or HTML representations.

10. When the results of a query are displayed, it should be possible to navigate to the linked documents. An example of a search output page is here.

22. REST is unlike an RPC, so it is vital that you handle errors appropriately in your RESTful web service. Both XML and HTML representations are expected:
    1. No document(s) found.
    2. Link not found.
    3. Arguments not provided to web service; e.g., tags.

Functional Testing

Functional testing will be used to mark the assignment. In order to facilitate testing of the assignment, you must add the following fields to your indexed content: contents (**content**), indexed by (**i**), type (**type**) and document id (**docId**). The strings in brackets are the case-sensitive field names. Make all of the aforementioned fields of type TextField. If a field name is not provided in a query the default index field searched must be **content**. Note, this is the second argument in the 3 argument constructor for QueryParser. The value of the indexed by field should be the group names of the individuals who built the search engine.

Part of testing for your assignment will be to issue queries of the form:

- "eclipse"
- "i:tony"
- "docId:2"
- "content:eclipse+i:tony"

Combinations of the above will also be used during testing by the TA. As stated above, assignment marking will be undertaken by demonstration.
