RAWDATA Portfolio-project 3 requirements

(Please refer to the note: *RAWDATA Portfolio project* for a general introduction)

Your SOVA application is under development in a 3-layered architecture with a structured relational database in the data layer. However, a major part of the data content is textual and the key function of the application is retrieval of documents, that is, posts. Thus, an obvious extension is to introduce Information Retrieval (IR) functionality. The goal of this portfolio subproject 3 is to extend the functionality of the SOVA application (Stack Overflow Viewer Application).

This subproject 3 does not bring any changes to the overall architecture of the application to be developed. Thus, the sketch in the general *RAWDATA Portfolio project* description and the details in *Portfolio-subproject 2 requirements* should still hold.

A. Application design

Describe the changes you have done to the design of your SOVA application. You may refer to the A-sections of portfolio 1 and 2 or repeat fractions from these as you prefer. The important thing here is that it is possible for the reader to get an overview of your design and to understand your motivation and arguments for your recent decisions. The level of detail in the design description in portfolio 2 is not required for portfolio 3.

B. Extended IR functionality

To simplify the indexing for Information Retrieval on the textual columns in the stackoverflow data, a table **words** is provided. It can be used as source to build inverted indexes and, as it is, it also comprise a combined inverted index for posts and comments. The column id is the id of the comment or post where the word appear. The columns tablename and what are used to distinguish what an entry refers to (either the title or body in posts or the text column in comments). The position of the word, given as sentence and word number, is encoded in sen and idx. In addition, the table provides a lemmatization[1] of words with the column lemma and a specification of the word category (part-of-speech) with the pos column using the Penn Tree Bank tags[2].

id	tablename	what	sen	idx	word	pos	lemma
60496	posts	body	8	12	when	WRB	when
60496	posts	body	8	13	something	NN	something
60496	posts	body	8	14	goes	VBZ	go
60496	posts	body	8	15	wrong	JJ	wrong
60496	posts	body	8	16	.	.	.

Figure 1: Small excerpt from the table **words**

From figure 1 we see that sentence 8, in the body of the post with id 60496, includes the phrase "when something goes wrong" as the last 4 of 15 words in that sentence (must be the last 4 words due to the punctuation mark as word 16).

B.1. Introduce a querying procedure that takes one or more keywords as arguments and returns posts that match all of these (thus an AND query). Build and use an inverted index (rather than simply using the like-operator or "cheating" with MySQL's Full-text indexing). Thus, your querying should be processed (conceptually) by merging postings lists. You can derive your own inverted index from the provided table words.

B.2. Develop a refined procedure similar to B.1, but now with a "best-match" ranking and ordering of objects in the answer. A best-match ranking simply means: the more keywords that match, the higher rank.

[1] https://en.wikipedia.org/wiki/Lemmatisation
[2] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

B.3. Build a new inverted index for weighted indexing similar to your index from B.1, but now with added weights to the postings. Discuss and decide on a weighting strategy. A good choice would probably be a variant of TF-IDF, but apart from the details of this you should also consider other related issues for instance how to normalize terms and whether or not to use stopwords.

B.4. Develop a refined procedure similar to B.2, but now with a ranking based on a TF-IDF weighting of words provided by the indexing derived in B.3.

B.5. An alternative to search results as of ranked lists of posts is ranked lists of words, that is, "weighted keyword lists". Develop functionality to provide such lists as answer. One option to do this is the following: 1) Evaluate the keyword query and derive the set of all matching posts, 2) count word frequencies over all matching posts, 3) provide the most frequent words (in decreasing order) as an answer (the frequency is thus the weight here).

B.6. Develop a refined version of B.5 that ranks based on TF-IDF weighting and thus is based on sum of weights rather than frequencies.

B.7. Provide one or a few examples of visualizations of weighted keyword list answers, by copying the answer and pasting it into a word cloud visualization tool[3]. Notice that this is intended to be a subject for further elaboration in Portfolio 4.

B.8. Extend your data model and database by introducing a co-occurrence term network, that is, a table of two-word combinations that includes weights corresponding to how often the two words co-occur. The weight can be based on the count of co-occurences in all the posts in the database. To allow investigation of the database as a whole, develop a procedure that can take a word as input and return the closest terms in decreasing order.

B.9. Provide one or a few examples of visualizations of "force networks" based on the result of a word query as in B.8 by copy-pasting to a force network tool[4]. Use the retrieved words as nodes and include all edges connecting nodes in this set. Edges are the weighted co-occurrence in the term network. Notice that this is intended to be a subject for automation in Portfolio 4.

Consider, *if time allows*, one or two of the following issues.

B.10. [OPTIONAL] Introduce a similarity search, such that evaluation of keyword-match is independent of the word form (ending) used in the posts.

B.11. [OPTIONAL] Introduce a word proximity based search, such that a query for instance may be in the form "A B; C; D E" when looking for: A followed by B, C, and D followed by E, where the first and third criteria are better satisfied, the closer B (respectively E) comes after A (respectively D).

B.12. [OPTIONAL] Consider and suggest an approach to relevance feedback that allows the user to select most interesting documents in the first result set and the "system" to generate a second result set taking the users preference into consideration.

B.13. [OPTIONAL] Consider and suggest an approach to query expansion, where the query is modified before the answer is derived. The general principle is to add words to the query that are similar to words already there – e.g. by being synonyms, by being frequently co-occurring or by being inflections of the same word.

B.14. [OPTIONAL] If you have an idea of your own, you can plug it in here …

[3] One option is to use the word cloud visualization tool available on Moodle.
[4] One option is to use the force network tool available on Moodle

C. Improving performance by indexing

One of the advantages of using a relational database in the data layer is that query performance can be improved and tailored to the actual needs (most frequent and/or most important queries/query types) at any state of the development process (even after the development has finished). Without need to reconsider other parts of the system, performance can be improved by adding or modifying the **indexing** of tables in the database.

C.1. Review the indexing of your database as modeled in portfolio subproject 1. Consider improvements.

C.2. Consider the extension developed under **B** and discuss/explain what is needed to obtain an acceptable performance when dealing with inverted index, weighting and term networks.

D. Data Access Layer and Web Service Layer

As in the A section it is satisfactory just to focus on the changes compared to Portfolio 2. Describe, document and implement the changes that your additions in **B** calls for.
Follow the structure from Portfolio 2 and cover your model, access layer and interfaces. Be brief and assume that the reader has read your previous reports.

E. Testing

The changes you make to your implementation in this subproject should of course be tested. However, the detailed unit level and integration tests you did in Portfolio 2 are not required here. Rather you should perform a black box testing of your system in line with what you did in assignment 3. Write a test suite and in that connection, discuss and decide on a reasonable level of detail.

The project report

You are supposed to continue in the groups from portfolio 1 and 2 and submit your portfolio 3 result by implementing the revised database (including new functionality) on the course server wt-220.ruc.dk and committing the Visual Studio solution, including projects for each part(layer) to GitHub (or similar resource) with a Section3 tag. The URL of this resource together with the Portfolio project 3 report (in pdf format) must be uploaded to the course website on Moodle.

The report should in size be around 6-12 normal-pages[5] excluding appendices. The submission deadline for the report as well as the product is 25/4-2018.

Notice that the report you hand in for Portfolio project 3 is not supposed to be revised later. However, if you find good reasons for this, your design and implementation can be subject to revision later. Documentation for such changes can be included in the final combined Portfolio project 4 / Reflexive synopsis report.

[5] A normal-page corresponds to 2400 characters (including spaces). Images and figures are not counted.