1. Technical Requirements analysis

Implementation of a search engine that contains a set of documents. Each document consists of a unique ID and a list of tokens. The search engine responds to queries by finding documents which contain certain tokens and returning their IDs. The program should accept a sequence of commands from standard input and respond on standard output. Commands are terminated by newlines. There are two types of commands: 1. The index command index doc-id token1 … tokenN The index command adds a document to the index. The doc-id is an integer. Tokens are arbitrary alphanumeric strings. A document can contain an arbitrary number of tokens greater than zero. The same token may occur more than once in a document. If the doc-id in an index command is the same as in a previously seen index command, the previously stored document should be completely replaced (i.e., only the tokens from the latest command should be associated with the doc-id). Examples:

index 1 soup tomato cream salt

index 2 cake sugar eggs flour sugar cocoa cream butter

index 1 bread butter salt

index 3 soup fish potato salt pepper

When the program successfully processes an index command, it should output index ok doc-id If the program sees an invalid index command (e.g, a token contains a non-alphanumeric character, or a doc-id is not an integer) it should report an error to standard output and continue processing commands. The error output should have the following form index error error message 2. The query command query expression Where expression is an arbitrary expression composed of alphanumeric tokens and the special symbols &, |, (, and ). The most simple expression is a single token, and the result of executing this query is a list of the IDs of the documents that contain the token. More complex expressions can be built built using the operations of set conjunction (denoted by &) and disjunction (denoted by |). The & and | operation have equal precedence and are commutative and associative. Parentheses have the standard meaning. Parentheses are mandatory: a | b | c is not valid, (a | b) | c must be used (this is to make parsing queries simpler). Logically, to execute the query the program looks at every document previously specified by the index command, checks if the document matches the query, and outputs the doc-id if it does. However this is suboptimal and much more efficient implementations exist. Upon reading the query command the program should execute the query and produce the following output query results doc-id1 doc-id2 … The doc-ids of the matching documents can be output in arbitrary order. If there is an error, the output should be query error error message Examples, given the index commands shown in the example above: in: query butter

out:query results 2 1

in: query sugar

out: query results 2

in: query soup

out: query results 3

in: query (butter | potato) & salt

out: query results 1 3

* Please implement a search engine in any language you like. You may use standard and open-source libraries and you may consult online materials or books. Please try to make the implementation reasonably fast, for a fairly large number of documents, say in the tens of thousands (you don’t need to test it on large sets of documents, just think about what this would be for your design). Query speed is more important than indexing speed.

1. Define the main points of the requirement

After taking note on the requirements document the entire process of this subject will rely on 2 main tasks. The first task will be the ‘index’ part where is required to store a list of terms as a sequence which will be defined by the **index** keyword followed by the document ID number (name of the document where the terms will be stored- integer value) followed by a single term in the most simple case or a list of terms (in our case alphanumeric values) separated by space.

The second task is the search phase. In this case the valid syntax for this command to be processed will be the ‘query’ keyword followed by a token in the most simple scenario or a group of tokens in the aggregated search method. More complex search methods of search will be implemented in a further evolving stage of the application.

1. Investigate different approaches in problem solving

The main approach in designing and building this application is by using a certain technology able to interact with the user (in this stage by me) via an interface or terminal in such way the commands that will be used (index or query) to either perform the creation of the document storing the terms and also the searching part to start in the same place. This kind of approach eases the part of thinking for multiple technologies for the moment and focus more on the back side logic and the process of data interaction. In this way than be sure this data will be handled in a simple but secure repository which can qualify as the most performant alternative.

In this case I am going for Java as the technology that provides both a way to make an simple interface to interact with the application, such as its terminal with the help of Scanner library for the Input commands. Java benefits also from the ability for platform independence execution that adds more points to this selection.

As for the data management Java allows quick interactions with the help of well organized data structure implementations that are part of the language. In terms of performance it guaranties a memory safe solution and state management control during application lifecycle.

Last but not least as the main purpose of this task is to provide a performant way to achieve both storing and searching though the data I have to give myself options for making this jobs.

After a deep research in this matter I found very interesting results and new techniques used to surf through the data in the most efficient ways.

I even got impressed from some of the technologies I had previously worked and their ability to perform in such tasks. The result from a long research point to the single word, ElasticSearch.

Elastic itself is a way of treating this task more than just the technology used. The way of converting the data we want to search in a super efficient datatype(indexing) and using this provided transformation to run a super performant algorithm that backs this strategy. Definitely a lot of languages offer easy ways to implement libraries that are designed using this algorithms and provide the power of elastic in their core. But for me switching to another language alternative was not an option in this case. Also the storage part were this libraries apply is on memory storages or documentet type of storages.

1. Select the ideal implementation that fulfill the requirements

Wanting to use Java as a layer of interaction between client and data because of the reasons mentioned before altogether with the fact that knowing that the job vacancy refer more to this technology, I stayed with my first choice.

I decided that the right approach will be to use Java as it offers a lot in interaction with data and modeling it in an object oriented style can create the perfect scalable application even in future improvements. This structure will mock the data in what I found to be the best of both worlds.

Finally my decision for data storage went on using a robust and light technology that would guaranty the safety of the data meanwhile taking advantage of one of the most performant techniques so far in data lookup processing such as elastic-search. Only this time this heavy job will be provided by the core engine of a relation database such as MySQL’s INNODB and it method, FULL TEXT INDEX.

The way it treats the data in our case will be same logic that elastic-searches use, inverted index. The formula in the algorithm performing in this case is one of the most performant so far.

1. Design the model in regard of the selected technologies

Making sure that the technology chosen will perform in the way I expect designing process should be as rigorous as it can be.

I have used the design pattern that in this case creates ability and enough abstraction for the application to handle code reuse and splitting logic without making it to complicated.

For the data access part I have used the DAO pattern , Data Access Object and keeping business logic out of dataset. Reading the command instructions I have build certain ParserClasses for each command as will work as a middle layer and will involve extra checks and validations. In this stage also the response for the user is composed and then sent to the interface. User will interact with the application for the moment only via terminal ( its system console).

1. Check /Inspect further integration details

After taking most of the row job done with creating the right configurations in the database part, creating the indexed column for the FULL TEXT INDEX to be able to run the querys, connecting the MySQL cluster with Java via it drivers and handle data in the business layer with the help of Java data structures, there will be needed one more step to complete the buildup. Having the needs for external libraries in order to perform further specific tasks is required a more fast and better way of dependency management. For this purpose I have used Maven and mostly keeping in mind 2 main reasons. Better integration in deployment environment. Till now the application was only about data and was running under the terminal. A good way to make this interface look more as an interface built to be used by non technical user such in real life usage was better for me to make this able to perform in a WEB fashion.

To create the Web Application I have build a REST service API that will deal with specific commands from the user. I used Jackarta implementation of JAX-RS and Jersey library.

Maven dependency helped for this and also making the run process simpler in an application server environment such as Glassfish Server. Now the application is turned into a web application ready to be consumed by multiple users.

1. Implement the solution

For the implementation of this solution in a real world scenario will be needed one or more server environments that can access each other. The backend part where the logic of the applications is developed and is served via the API services (REST architecture) and will be available for any FrontEnd technology used in order to consume this APIs. I have used a simple static client based interface for this part running on different server (port in my case) and using the asynchronous AJAX calls from jQuery.

This is a simple implementation of MVC an web application project.

1. Test the results achieved and add performance enhancement

Testing the application is used Junit5 library for the unit testing. This test are more concentrated in debugging the ability of certain methods in our project to run in an isolated mode.

As for the test results the data will be fetched manually since automated testing yet to be involved.

Integration or System Testing also non specified in this project purpose but can be applied in any case with the help of Assumption class and Junit library.

1. Further inspection/ Addons

Since this application is not designed specific to handle multiple session by multiple usage there is no implementation of other strategies to manage this kind of scenarios.

Ex. Handling multiple Db calls and interactions there would be a good alternative to create a pooling technique and store some of the DB connection process and not having to terminate and reopen connection in each call. Create another table in the DB with the help of features supported by MySQLs FULL TEXT INDEX Full Language Support and preserve the most frequent searched words to later refer before each search.

This and also more complex searching techniques can also backed by this technology.

Using JPA libraries that can automate some processes and allow more efficiency accessing DB connections and persistence.

1. Documentation