Search engine solution

# Internet information retrieval - project

Student/author: Marinel Amariei

The presented implementation is created around the concepts of information retrieval from a web page, boolean search, tf/idf search.

## Specifications and requirements

The application is based on:

-database server

-web services server

-front end server

Main implemented features:

-Information/text retrieval from a given path

-Data storing in database (direct/inverse form)

-Boolean search

-Tf/idf search

-Porter word stemming algorithm

-Frontend interface powered by reactJS

Hardware/software resources:

-Visual Studio Code, Sublime – text editor

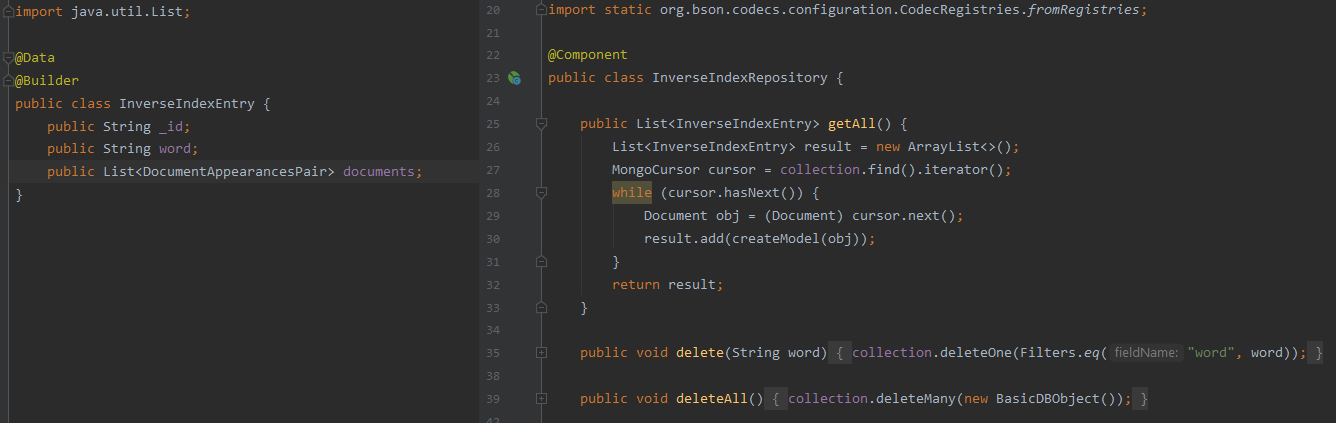
-IntelliJ – Java IDE

-MongoDB, Robo3T – Database server

-Github – Project management

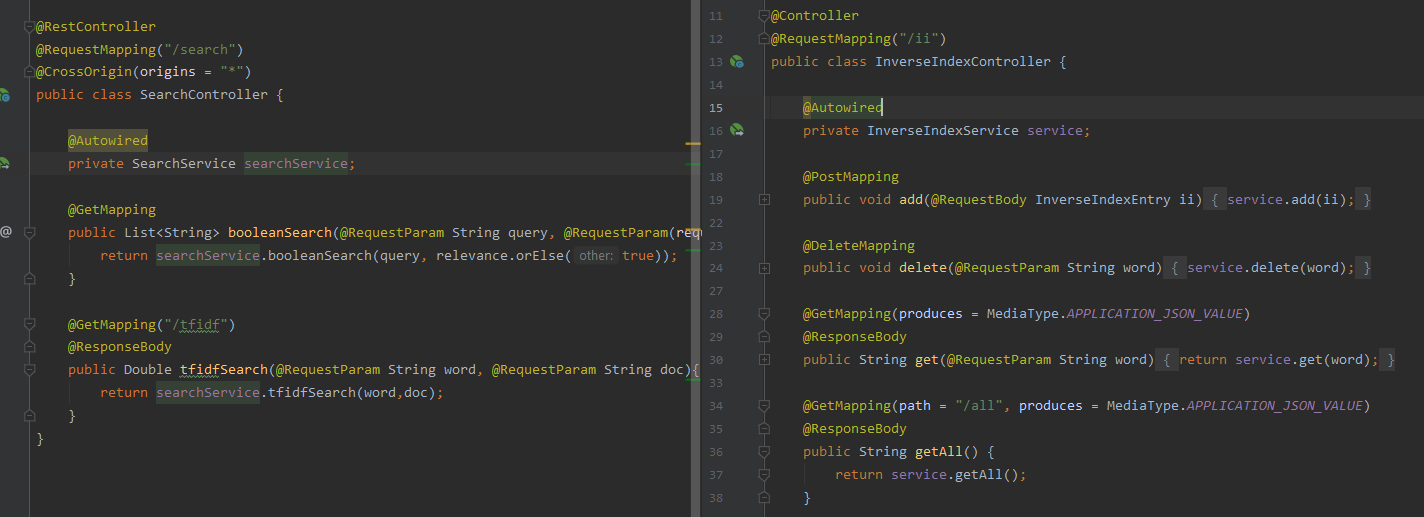
## DATABASE

The chosed database is MongoDB based on the similarity of storing data with the JSON format used before. CRUD (Create, Read, Update, Delete) method were implemented inside the repository module. The database consists of 2 collections: InverseIndex(with word, documents and counters), DirectIndex(with document, words and counters) .



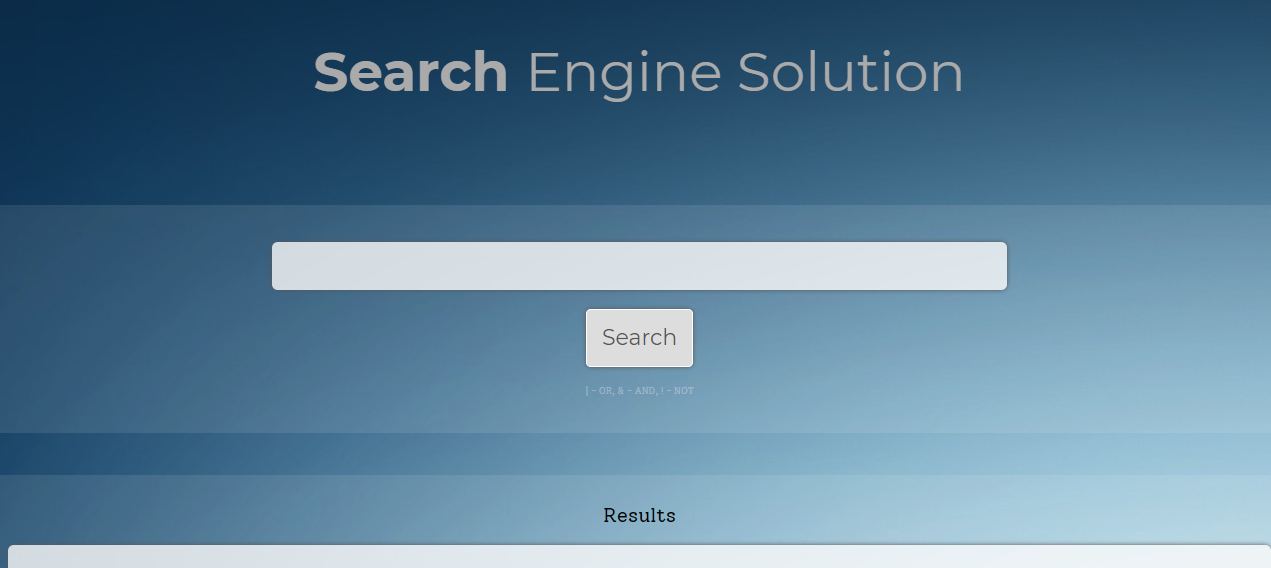
## Web Services

Using Spring Framework, 3 endpoints were created for interacting with the database - /search, /ii - inverse index, /di - direct index.



## Frontend

For the frontend interface, ReactJS framework was used. This is a single-page application focused on the search algorithm. In order to send requests to the services, axios library is used.



## Implemented features

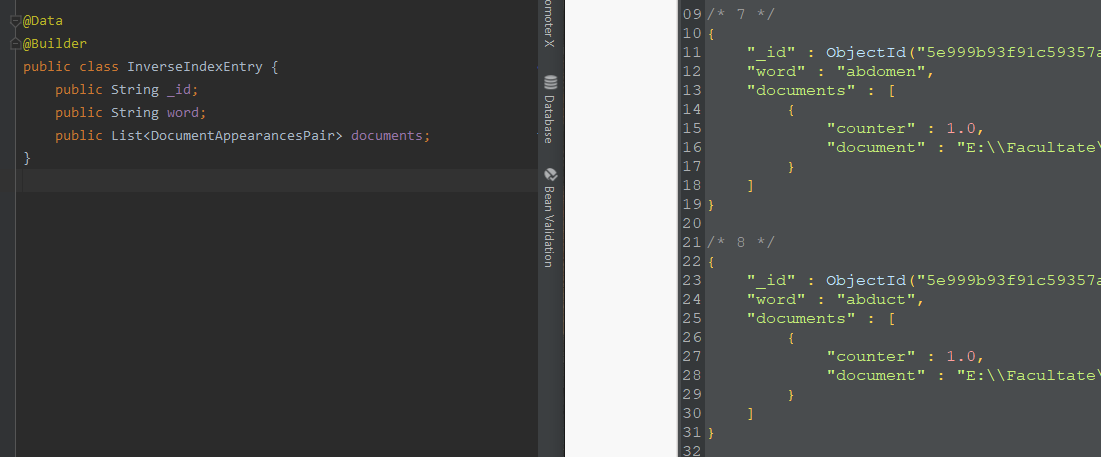
The main implemented features are:

**-Information/text retrieval from a given path:**

The input location is recursively parsed and all text files are retrieved and read character by character.

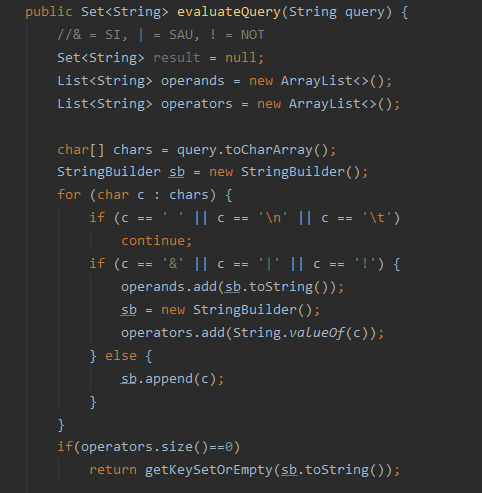
**-Data storing in database (direct/inverse form):**

Previously mentioned retrieved data is stored under direct index and inverse index format in the MongoDB database.



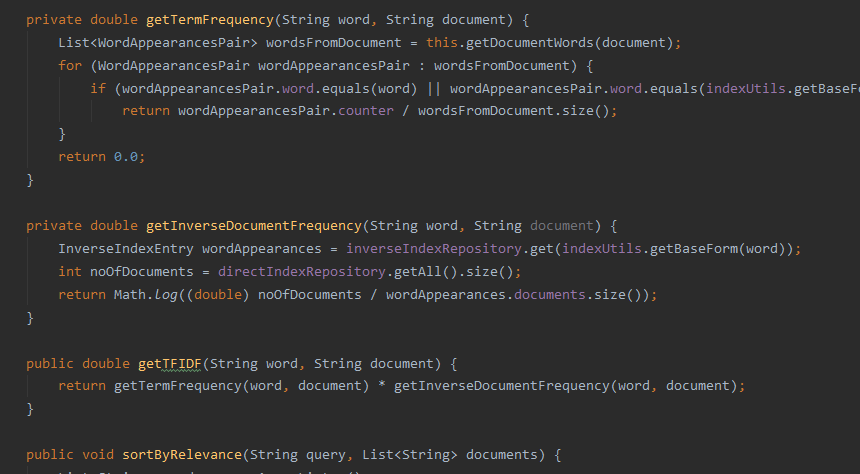
**-Boolean search:**

With the help of the frontend interface, a search can be launched throughout the whole retrieved data with following symbols/operations: | - OR, & - AND, ! - NOT.



**-Term frequency - inverse document frequency search:**

There is the option to sort results by relevance. This relevance coefficient is given by the TF/IDF algorithm.



**-Porter word stemming algorithm:**

The chosen stemming algorithm is Porter’s implementation. It is used to get the base form of any word, either from a search query or a data storing operation. It is a multi step algorithm which come down to:

1. cut plurals and participles

2.changing adverbs/adjectives to nouns

3.cut state/quality suffixes (-ness/-ful)

4.cut derivational suffixes (-ize/-al/-ly)

5.append (e/s) / double(l/t) (if needed)

Pros of porters stemming implementation: [4]

-easy to understand with 60 predetermined suffixes

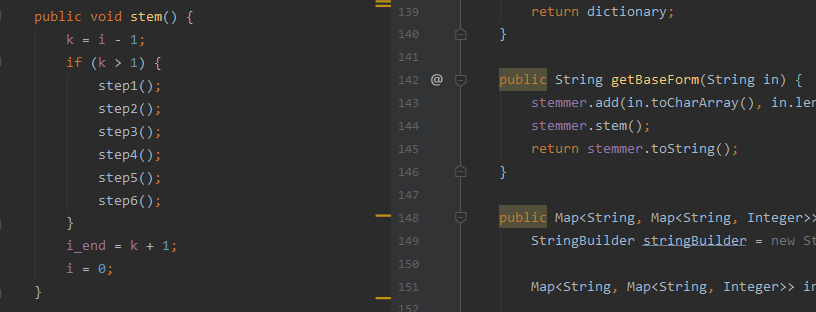
-well known with many improvements over the time

-more steps (5 compared to lovins 2/3), enough to tidy up the stems after cutting operations

-advantage in terms of speed over Lovins stemmer

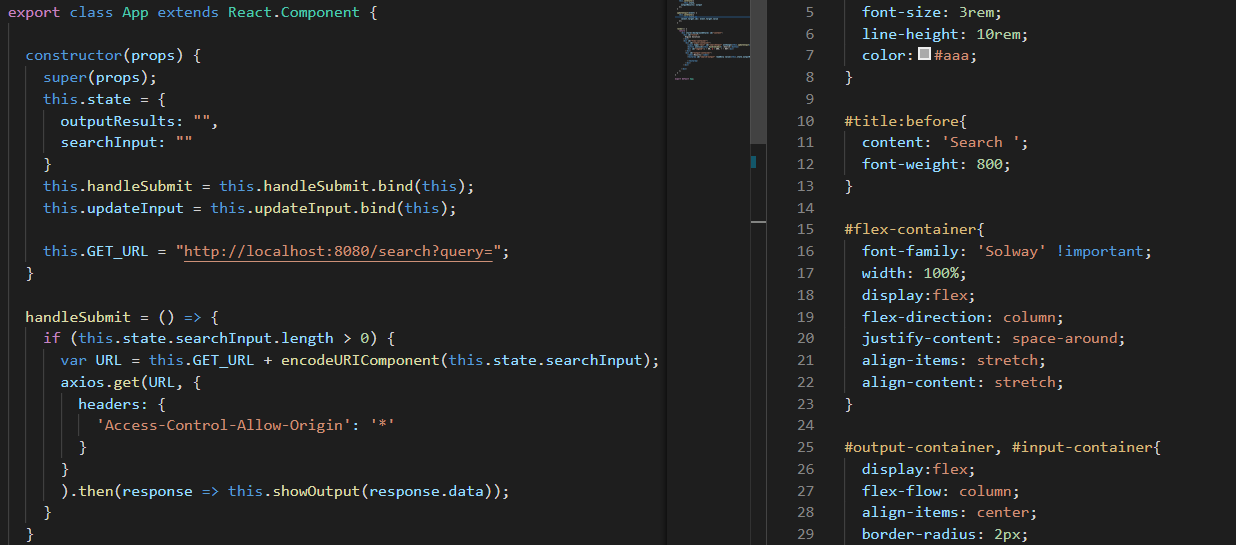
Cons:

-other stemmers (Lovins) produce a better data reduction



**-Frontend interface powered by reactJS:**

The interface consists of a SPA powered by ReactJS framework. Its layout is based around search feature.



## CONCLUSIONS

**Difficulties:** big and constant workload and requirements

**Ways to expand the app:**

In the future, the solution could benefit from a paralel/distributed implementation based on the MapReduce mechanism.

# Bibliography

[1]ReactJS Getting started - <https://reactjs.org/docs/getting-started.html>

[2]MongoDB Docs - <https://docs.mongodb.com/>

[3]Spring Framework Docs - https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html

[4]Strength and Accuracy Analysis of Affix Removal Stemming Algorithms - [<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.2855&rep=rep1&type=pdf>]