Information Retrieval P05 - Software Architecture Plan

WS 2024-25

Sourya Shome, Sakshi Choudhary, Neha Mohan Khade, Sandipan Seal, Nilam Suresh Ghaware

High-Level Software Architecture Plan

Use Case: Search Result Clustering

Goal: Implement a clustering-based visualization of at least 10 retrieved search results for a given query, supporting exploration by illustrating the relationships between results. The backend search engine will be implemented using **Apache Lucene**, and **K-Means Clustering** will be used for clustering.

Note: This is a high-level plan sharing details about the tech-stack and methods that we suppose we will be using to implement the software product, though subject to change, given we find a better performing method within Apache Lucene or any method that is deprecated to use compatible with latest methods

1. Backend Components

1.1. Search Engine API : Apache Lucene as it provides excellent text indexing and retrieval capabilities with a robust API for customizing queries and results.

1.2. Key Lucene Features and methods planned to be used

1. **Indexing:**
   - org.apache.lucene.analysis.core.LowerCaseFilterFactory
   - org.apache.lucene.analysis.core.StopFilterFactory
   - org.apache.lucene.analysis.custom.CustomAnalyzer
   - org.apache.lucene.analysis.en.PorterStemFilterFactory
   - org.apache.lucene.analysis.standard.StandardFilterFactory
   - org.apache.lucene.analysis.standard.StandardTokenizerFactory
2. **Searching:**
   - org.apache.lucene.analysis.custom.CustomAnalyzer;
   - org.apache.lucene.document.Document;
   - org.apache.lucene.index.DirectoryReader;
   - org.apache.lucene.queryparser.classic.ParseException;
   - org.apache.lucene.queryparser.classic.QueryParser;
   - org.apache.lucene.search.IndexSearcher;
   - org.apache.lucene.search.Query;
   - org.apache.lucene.search.ScoreDoc;
   - org.apache.lucene.search.TopScoreDocCollector;
   - org.apache.lucene.search.similarities.ClassicSimilarity;
   - org.apache.lucene.store.Directory;
   - org.apache.lucene.store.FSDirectory;

3. **Clustering:**
   - weka.clusterers.SimpleKMeans;
   - weka.filters.unsupervised.attribute.StringToWordVector

2. Frontend Interface

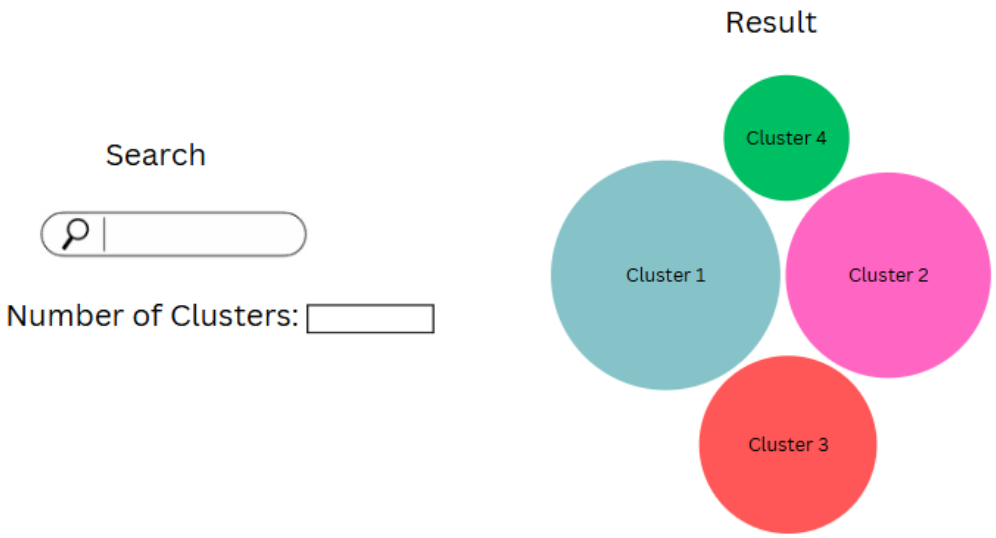
1.3. Wireframe

1. **Search Bar:** Allows users to input queries.
2. **Cluster Visualization Panel:**
    - Interactive visualization
    - A hovering over a cluster that shows summary details

2. Tech Stack and Libraries Used

1. **Java**: Backend
2. **Python**: Documents collection via web scraping
3. **React**: Frontend

Vision:



Steps to run the code:

Backend:

1. Go inside folder server
2. mvn spring-boot:run

Frontend :

1. Go inside folder client
2. npm install
3. npm start

For more details, please read the Readme.md file after unzipping the submission. Or you can refer from the cloud link provided for the same.

The application can be accessed on your local browser at **http://localhost:3000/** and their server is running at localhost, **port 8080**

OVGU Cloud link for codebase - https://cloud.ovgu.de/s/3DxZHQMEJKrDBjt