SSW CS410 Progress Report: Design a Results Cache for Popular Queries (System Extension)

Github Link: https://github.com/waelmb/Results-Caching-System

1. **Progress Made Thus Far**
    a. Looked at health conferences, medical databases, and medical news sites to find best formatted sites with medical info - these mostly turned out to be news sites
    b. Scraped the sites for the latest articles and stored the articles as .txt files and to an Elasticsearch deployment for cloud usage in the future.
    c. Used topic modeling to see what the most popular topics were among the scraped articles
2. **Remaining Tasks**
    a. Fine tune the algorithm for identifying popular and relevant queries.
        i. Issue: results from topic modeling contains stop words which are irrelevant
        ii. Plan: combine topic modeling with keyword extraction (Yake)
    b. Create and maintain an Least Recently Used (LRU) cache index on Elasticsearch
    c. Put together the code within a web framework that can be easily deployed as a microservice
    d. Clean up, test, and fine tune the microservice
3. **Challenges/Issues Faced Thus Far**
    a. Integration with 1-Search to improve search performance
        i. We are still thinking about how to integrate them to work together in a loosely-coupled way as we aim to deploy this system as a microservice. We investigated potential frameworks that will make cloud deployment very easy, which include Flask and FastAPI, but the decision has not been made yet.
        ii. The original system uses REST APIs to deliver the search results to the client, but to leverage the microservice we are building in 1-Search, we might need to convert the REST API architecture to Web Sockets architecture. This will allow a stream of results to be delivered as they become available. As such, we can deliver the results from the cache first, then deliver the remaining results as they become available a few seconds later. With that said, this might be out of the scope of this project given the time, but we have not decided yet.
    b. Operating system variability: since the team members use different operating systems, we faced issues with code reliability, but were able to account for it at the end
    c. Testing for users with no access to 1-Search: since our microservice will closely run with 1-Search, we are not entirely sure how graders and other testers will be

able run our code. Our code not only requires access to 1-Search source code, but also access to an Elasticsearch deployment.

4. **Takeaways From Proposal Review**
    a. The comments form the reviewers advised us to reassess our tasks and goals for this project so we can deliver what we proposed.
        i. One of the suggestions was to look into existing extensions or services that tackle similar problems we are trying to solve. We were unable to find such a service with all the functionalities we desired and thus decided to stick to what we have proposed originally.
        ii. Another reviewer stated that we may have difficulty completing the tasks due to time constraints. We have decided to utilize an LRU cache instead of storing popular queries, which will thus cut down on the time required for the project.