**Website Crawler and Searcher**

This program is intended to provide a product search functionality for the user to search across multiple E-commerce websites. There are 2 projects which does the following things. The main search engine that we are using here is ***ElasticSearch*** to solve all of the problems given in the statement.

1. E-commerce website Crawler (manjeer.ecomm.crawler Gradle Project)
2. E-commerce website Search. (manjeer.ecomm.search Gradle Project)
3. **E-commerce website Crawler:** This is responsible for crawling the website starting from a given base url. It mainly uses ES to index the crawled data into index “ecomm” and uses the mapping “product”. The JSON to create index with mapping is “create-index.json” which is at src/main/resources.

Implementation:

1. It uses **Crawler4J** framework to crawl websites. The crawling happens in the backend to find all URLs in each page and stores in **Thread Safe** list named “LinksList.java”(***We should make use of some persistence layer here to scale crawling and indexing horizontally***) into which it inserts all new URLs found in previous URLs. In Parallel, the “ParseAndIndexer.java” will start parsing both the URL and Content for products and indexes it to ES.
2. The “product” ES mapping is the key to solve most of the problems given in the exercise. Below if the explanation on how I solved each of the problems.

1. **Write a program to crawl all the products in redmart.com and** [**www.groupon.sg**](http://www.groupon.sg) **(Solved and Working)**

This was done by creating 2 modules “Crawler” and “Search” as explained above. It is written in such a way that

* It runs both crawling and indexing by parsing URLs and Content in parallel.
* Adding a new website is very easy as we need to implement EcommWebsite.java interface and the parsing using regex, URL format can be defined within that implementation and it is highly independent and loosely coupled.
* Product format is very minimal currently as the time is very short to implement. But, this mapping can contain any type of product as it is JSON and its very flexible.
* We can specify number products or we can run it infinitely. We can periodically clear “LinksList.java” and insert base URL into the queue which will start fresh index again which is so easy to perform.

1. **Provide a schema to accommodate all the products available? (Solved and Working)**

Solution: Please see product mapping file for schema (create-index.json)

1. **And provide an “update” strategy to products that are available now but sold out later (Solved and Working)**

Solution: **UPSERT** operation of ES is the solution that I am using for this. This is designed not only for Sold Out items. But, also for all the changes happened on specific item. Whenever crawling happens it updates data. There is field in mapping “isAvailable” represents if it is available or not. We can filters also on top of it to filter out sold out items.

1. **How do you know that the search engine is improving every day? (Just throwing an idea)**

Thought: This is one area where I have some confusion. Are we talking about making search engine intelligent (Some machine learning algos) or how do we measure accuracy of results everyday?

If the second one, then we might need to rely on the scores of the results which we are getting for each query. We can run the crawler everyday or with some interval and it updates all the data that we currently have. We can do it only for specific websites also with current design.

1. **How will you design multi-language schema for all the products you have just crawled? (Solved and Working)**

Solution: This is solved with **Language Analyzers** (See mapping file **create-index.json** ) and **Multi Match Query** (QueryExecutor.java) of ES. Different language analyzers for different languages and the multi match query gives more weightage to different language analyzers.

1. **How will you handle text-typos and provide the right suggestions? (Solved and Working)**

Solution: This is solved by using **Suggestions Query** (QueryExecutor.java) of ES. Currently I am using term suggestion. But, we can also use Phrase Suggestions, Completion Suggestion etc. I implemented only term suggestions because of time constraints.

1. **When we display result to our users, we do not want to show too many similar/duplicate products. How do you identify duplicate or near-duplicate products? Example “iPhone 6S” vs “iPhone 6 S” vs “iPhone-6”? (Solved)**

Solution: This can be done using **Fuzzy Queries (**ParseAndIndexer.java**)** of ES in multiple ways based on the business requirements. Fuzzy queries get items which are nearer than specific distance provided in query. If we don’t need to index duplicate/similar products, we can make a Fuzzy query with “*name” field* and if its already exists, we can just update same document or just add the URL to same product as per other use cases.

1. **E-commerce website Search:** This is a responsible for taking query text from user and display results. This will make query to get and display near by results and also suggestions. Currently this is not so complicated as this just makes a single query and we handle every problem stated in crawler itself using ES and its queries, mappings and analyzers etc.

**Unit Testing:** I have written UNIT test cases for most of thekey classes. And they will run during Gradle build. We can cover every single line of code and use case with proper planning and execution od writing test cases. I may not have written test cases for every class due to time constraint. (Or I might complete before submitting and writing this doc before finishing all test cases)

**Execution and Demo:**

1. **Download and extract 2 projects EcomCrawler.zip and EcomSearch.zip (Both are Gradle projects)**
2. **Create ElasticSearch cluster locally.**
3. **Provide cluster name in ESClient.java in both projects.**
4. **Very Imp: Please use appropriate Elasticsearch package version in build.gradle in both projects. (I am using 2.3.3)It should be same as your ES version.**
5. **Create index with name “ecomm” using JSON in “create-index.json” file.**
6. **Open terminal and go to the “**manjeer.ecomm.crawler**”( EcomCrawler) project directory.**
7. **Run “gradle clean build run”. Allow sometime for it to run. (Approx. 30mins based on internet connection)**
8. **Open terminal and go to the “**manjeer.ecomm.search**” (EcomSearch)project directory.**
9. **Run “gradle clean build run”. It will ask you for key words. Enter few and see results as “product name and URL”.**
10. **Enter some near by words or spelling mistakes. It will give suggessions.**
11. **Enter some French word and it will still give you results using language analyzers. (Ex: hôtel )**

**Note:**

1. I focused mostly on solving main problem of creating search engine.
2. Focused on Good design for crawling engine with parallelism, Ease in scaling horizontally and Extendibility of code like adding another website and also solving given questions.
3. Product schema is very minimal which is enough to solve/test/demo given questions. It can be extendable to any extent based on actual requirement.
4. Not focused much on writing hard coded regex to actually parse each and every field on a product page.
5. Trying to write as many test cases as possible and all the test cases may not be finished. (because of time constraint). You can observe the way I am writing unit tests and we can cover every line of code and every use case almost if we have time.