Requirements from Client-

Features for search engine

1. When a user types in our search box, we should give suggestions for relevant terms which start with the letters a user is typing. Example, if a user types ammonium, we can give options – ammonium sulphate, ammonium carbonate, ammonium sulphamate, etc. This can be given from the product names loaded on the website.
2. If a user adds incorrect spelling in a search query, suggestions based on the best match should be provided.
3. When a user types with a different name i.e. synonym / CAS, search engines should recognize the right product based on synonyms.
4. After the user types the search query and clicks on enter, a list of products is shown. Here basic specifications, image of the product needs to be shown, and an action button which leads to the detailed product page. Also, we should have sorting and filters.
5. Sorting – by relevance, by name, by price – low to high, high to low
6. Filters – Product Category, Brand, Pack size, Price range, Products on Offer, User Ratings, Country of origin. Depending on the product, relevant attributes should be shown. Example – If a user searches for centrifuge, attributes like maximum speed, type of centrifuge, no of places in centrifuge, etc can be shown.
7. In search results, we should have sections – sponsored products if any on the top, then the most relevant results – based on percent match of product names with the search query. Example if the search query is Ammonium sulphate, we should first show product names which just have Ammonium sulphate, then Ammonium sulphate AR, Ammonium sulphate-PCT0003-5KG, etc and followed by Ammonium ferrous sulphate, Ammonium nicket sulpate etc.
8. In case a search query does not match with any product on the site, we can say ‘No results found’ and ask to post an enquiry. Also provide contact details if user likes to talk with us for their queries.
9. Integrate with google analytics to get data for the top search queries

Purpose of Project

Our main purpose of the project is to make a search engine which is more efficient and can replace the existing Biomall search engine . And in this project we have used elasticsearch as it is more robust and easy to use. Elasticsearch allows you to store, search, and analyze huge volumes of data quickly and in near real-time and give back answers in milliseconds. It's able to achieve fast search responses because instead of searching the text directly, it searches an index.
ElasticSearch is used for a lot of different use cases as well, for example, “classic” full-text search, auto-complete, spell checker, synonym, sorting, filter feature etc.

Prerequisites

Github Link : https://github.com/anujaagarwal/biomall_search_engine
Elasticsearch - 7.9.1: https://www.elastic.co/downloads/elasticsearch
Kibana - 7.9.1 : https://www.elastic.co/downloads/kibana
Python 3 : **https://www.python.org/downloads/**
Elasticsearch Library in Python : pip3 install elasticsearch
Flask Library in python : pip3 install flask
Postman/ResClient : For Testing APIs

Search Engine Features

1. Autocomplete Feature

Autocomplete, or word completion, is a **feature** in which an application predicts the rest of a word a user is typing. **Autocomplete** speeds up human-computer interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. For the test purpose we have integrated the autocomplete feature in the title **field** which is product name , **brand field** and **category field** .

Getting started with the code : - For the testing purpose our elasticsearch index name is biomall, so you will notice in every code snippet there will be a biomall written as index name.

To integrate autocomplete features you have to change the settings and mappings of the index. You have to include the mentioned code below:-

PUT biomall

```
{"settings":{
    "analysis" : {
      "analyzer" : {
       "autocomplete" : {
        "filter" : [
         "lowercase"
        ],
        "tokenizer" : "autocomplete"
       },
       "autocomplete_search" : {
        "tokenizer" : "lowercase"
       }
      },
      "tokenizer" : {
       "autocomplete" : {
        "token_chars" : [
         "letter"
        ],
        "min_gram" : "2",
        "type" : "edge_ngram",
        "max_gram" : "20"
       }
      }
    },
"mappings": {
  "properties" : {
     "brand" : {
      "type" : "text",
      "analyzer" : "autocomplete",
      "search_analyzer" : "autocomplete_search"
     }}}}
```

2. Sorting Feature:-

Sorting is a feature in which text fields are sorted alphabetically and numeric fields are sorted in descending order or ascending order. For our test purpose we have made variations of sort features like multiple sort variables or single variables. According to the requirement you can use the required API. To use the sorting feature you have to add something in the mapping section. Like for example if you have to add sorting feature in the title field you have to add the curly braces content under title field in mapping.

```
For Ex: -
"title": {
        "type" : "text",
        "fielddata" : true
      }
```

3. Filter Feature

The **Filter feature** allows you to filter a range of data based on criteria you define. Like in the biomall index there are various fields like brand, category, price range basis which we can filter our data. So, we have made various API for filter features having range fields like user ratings, list_price and filter fields like category, brand. API request body is defined below. Various Filter API of different scenarios have been implemented.

4. Spell Corrector Feature

Spell checker is a feature which checks for misspellings in a text.

Spell checker is implemented by using the fuzziness feature of elasticsearch. A **fuzziness** is done by **means** of a **fuzzy** matching query, which returns a list of results based on likely relevance even though search argument words and spellings may not exactly match. If the typed search string doesn't match the top result then output with the same fuzziness is suggested. Request body of the API is explained below.

Implementation

Step 1 : Make use of API - /search
Step 2 : Display with “Did You Mean?” message.

5. Facets Feature:-

You can use facets in your search engines to filter results by some fields. You can use /search and /autocomplete Api to use facets feature.

6. Sponsored Products:-

In search results, we should have sections – sponsored products if any on the top, then the most relevant results – based on percent match of product names with the search query. Example if the search query is Ammonium sulphate, we should first show product names which just have Ammonium sulphate, then Ammonium sulphate AR, Ammonium sulphate-PCT0003-5KG, etc and followed by Ammonium ferrous sulphate, Ammonium nickel sulphate etc. API request is explained below. Sponsored product feature was implemented using sort feature based on the weight of the sponsored value. Example if A product has greater value than B in a column field “sponsored” then in output A will be shown first . A

new column or field needs to be added for the sponsored feature to work. Developer needs to make a new column “sponsored” which contains weights according to which the sponsored product will be displayed. We have made use of sorting features.

7. Synonym Feature:-

The synonym token filter allows easy handle synonyms during the analysis process.

Procedure to implement Synonym Feature

Step 1: At the time of indexing, i.e. creating index in elasticsearch, set the setting by making use of below query in Kibana.

/indexname - is the database name eg. /biomall

```
PUT /indexname
{
  "settings": {
```

```
"index": {
  "analysis": {
    "analyzer": {
      "synonym_analyzer": {
        "tokenizer": "whitespace",
        "filter": ["lowercase","my_synonyms"]
      },

          "autocomplete": {
             "tokenizer": "autocomplete",
             "filter": ["lowercase"]
          }
    },
    "filter": {
      "my_synonyms": {
        "type": "synonym",
        "synonyms_path": "analysis/synonym.txt",
        "updateable": true
```

```
        }
      },
       "tokenizer": {
           "autocomplete": {
              "type": "edge_ngram",
                 "min_gram": 2,
                 "max_gram": 20,
                 "token_chars": [
                    "letter",
                    "digit"
                 ]
           }
        }
    }
  }
 }
}
```

Step 2. Set the mappings of index /indexname using below query . Autocomplete and Synonym Feature will work for fieldname1. Eg. fieldname1 can be "productname".

```
POST /synonym_test/_mapping
{
  "properties": {
    "fieldname1": {
      "type": "text",
      "analyzer": "autocomplete",
      "search_analyzer": "synonym_analyzer"
    }
  }
}
```

Step 3. Insert Data using CSV to the /indexname.

Kibana -> Insert CSV data -> Enter Index Name(eg. biomall) -> Advanced Settings -> Enter Mappings and Settings as mentioned above. -> Import Data.

Step 4. Insert synonym data by using POST method for API endpoint "/synonym_update".
This API updates the "synonym.txt" whose relative path with respect to config folder is /analysis/synonym.txt.

Step 5. Make use of the Search API endpoints mentioned.

8. Provide Contact details when no results found:-

In case a search query does not match with any product on the site, we can say 'No results found' and ask to post an enquiry. Also provide contact details if the user likes to talk with us for their queries. Make use of /search_by_string API, if the response contains no hits then a simple if-else logic can be implemented by using this information.

9. Google Analytics Feature

Google Analytics Features have to be implemented in the frontend. Procedure to implement Google Analytics Feature is given below

1. Login to Google Tag Manager https://tagmanager.google.com/
2. Create an Account - Enter container name as your website
3. Agree to Terms of Service Agreement
4. Setup analytics Tag with Google Tag Manager

Follow the steps

APIs Documentation

Method : POST
Endpoint : localhost:port_number/create_index

Request Body :

```
{
   "indexname":"biomall",
   "mapping_property":{
      "brand" : {
       "type": "text",
      "analyzer": "autocomplete",
      "search_analyzer": "synonym_analyzer"

      },
      "brand_id" : {
        "type" : "keyword"
      },
      "brand_status" : {
        "type" : "keyword"
      },
      "cas" : {
        "type" : "keyword"
      },
      "cat_id" : {
        "type" : "keyword"
      },
      "cat_status" : {
        "type" : "keyword"
      },
      "catalog_no" : {
```

```
  "type" : "keyword"
},
"category" : {
  "type": "text",
  "analyzer": "autocomplete",
"search_analyzer": "synonym_analyzer"
},
"description" : {
  "type" : "keyword"
},
"featured_product" : {
  "type" : "keyword"
},
"id" : {
  "type" : "keyword"
},
"list_price" : {
  "type" : "keyword"
},
"p_image" : {
  "type" : "keyword"
},
"prod_url" : {
  "type" : "keyword"
},
"product_status" : {
```

```
        "type" : "keyword"
      },
      "title" : {
      "type": "text",
      "analyzer": "autocomplete",
      "search_analyzer": "synonym_analyzer",
      "fielddata" : "true"
      },
      "quantity" : {
        "type" : "keyword"
      },
      "unit" : {
        "type" : "keyword"
      },
      "sponsored_value":{
        "type": "keyword"
      }
    }
}
```

Explanation :

“Indexname” : index name to be entered

“mapping_property” : is mapping

Response : Test on Postman

Method : POST
Endpoint : localhost:port_number/autocomplete
Request Body :

```
{
   “indexname”:”biomall”,
   "input":"MP"
}
```

Explanation :
“MP” is the input typed in search field
Response : Test on Postman

Method : POST
Endpoint : localhost:port_number/search
Request Body :

```
{
"indexname":"biomall",
"search_fields" : ["title","category","brand"],
"input" : "Lead",
"range_fields_value":{},
"filter_terms":{"brand_id":1,"cat_id":2},
"sortlist":[],
```

"size" : 3,
"From": 1
}
Explanation :

“Title”, “category” is the multiple search field or search column name
“Lead” is the input string entered by customer
“Filter_terms” is expandable dictionary, can be used for multiple columns to be filtered at once
“sortlist” is a list where field to be sorted and sort type are displayed in the form of dictionary.
“Size” is the number of result we want in the response
“From” is the pagination
Response : Test on Postman

Method : POST
Endpoint : localhost:port_number/update_field_by_query
Request Body :

```
{
   "indexname":"biomall9",
   "query":[{"id":"7"}],
   "update":[{"sponsored_value":"8"}]
}
```

Explanation :
“Indexname” is the index’s name used for biomall
“Query” denotes which item should be updates in the index.
“Sponsored_value” : is the weight value with which the sponsored products to be sorted.
Response : Test on Postman

Method : POST
Endpoint : localhost:port_number/bulk_update/<indexname>
Request Body :

```
<root>
  <products>
    <product>
        <brand>MP Biomedicals</brand>
        <brand_id>1</brand_id>
        <brand_status>1</brand_status>
        <cas>100-02-7</cas>
        <cat_id>2</cat_id>
        <cat_status>1</cat_status>
        <catalog_no>0210246125</catalog_no>
        <category>Pipette Tips</category>
        <description>(4-Nitrophenol)Spectro Grade. Purity: 99+%Pale yellow crystals</description>
        <featured_product>0</featured_product>
        <id>6</id>
```

```
    <list_price>10689.4</list_price>
    <p_image>product/1.jpg</p_image>
    <prod_url>p-nitrophenol-cas-100-02-7-25-gm-0210246125</prod_url>
    <product_status>1</product_status>
    <quantity>25 gm</quantity>
    <title>p-NITROPHENOL (CAS # 100-02-7) 25 gm</title>
    <unit>Bottle </unit>
    <sponsored_value>9</sponsored_value>
</product>
<product>
    <brand>MP Biomedicals</brand>
    <brand_id>1</brand_id>
    <brand_status>1</brand_status>
    <cas>100-02-7</cas>
    <cat_id>2</cat_id>
    <cat_status>1</cat_status>
    <catalog_no>0210246105</catalog_no>
    <category>Pipette Tips</category>
    <description>(4-Nitrophenol)Spectro Grade. Purity: 99+%Pale yellow crystals</description>
    <featured_product>0</featured_product>
    <id>7</id>
    <list_price>3632.65</list_price>
    <p_image>product/1.jpg</p_image>
    <prod_url>p-nitrophenol-cas-100-02-7-5-gm-0210246105</prod_url>
    <product_status>1</product_status>
    <quantity>5 gm</quantity>
```

```
        <title>p-NITROPHENOL (CAS # 100-02-7) 5 gm</title>
        <unit>Bottle </unit>
        <sponsored_value>7</sponsored_value>
    </product>
  </products>
</root>
```

Explanation :

XML format for data to be inserted

Response : Test on Postman

Method : GET
Endpoint : localhost:port_number/get_synonym
Request Body : No body required
Response : Test on Postman

Method : POST
Endpoint : localhost:port_number/delete_synonym
Request Body :

{
“synonyms” : “NH3, ammonia”
}
Explanation :
“Synonyms” is the synonym pair which needs to be deleted
Response : Test on Postman

Method : POST
Endpoint : localhost:port_number/<indexname>/delete_by_query

Request Body :
{
"indexname": "biomall",
"query": [{"id": 7}]
}
Explanation :
“Indexname” is the index’s name used for biomall.
“Query” is the query based on which a specified product description to be deleted.
Response : Test on Postman

Method : POST

Endpoint : localhost:port_number/delete_index
Request Body :

```
{
   "indexname":"biomall"
}
```

Explanation :

“Indexname” is the index’s name used for biomall.

Response : Test on Postman

Conclusion

E-commerce website can make use of above 33 APIs which contains features like autocomplete, spell-checker, highlight feature, filter feature, sort feature, facets feature along with developer has an option to make use of size of the response desired. The query has gone through a sanitization process which will eliminate special characters if required. Elasticsearch provides many features which an e-commerce website can

benefit from. The above APIs if properly used, will improvise performance of existing websites because of the reason being , elasticsearch is a NoSQL based database .