GSOC 22’ Proposal

Cuneiform Digital Library Initiative

Search Enhancements(175h)

Name : Ajinkya Morankar

Username : ajinkyamorankar03

G-mail : ajinkyamorankar03@gmail.com

Location : Mumbai, India

Time Zone : UTC +5:30

About The Project (Medium)

This project mainly focuses on **updating and improving** the **current search features**. The search setup has been completed under past programs of GSOC and hence the **documentation** of it is needed which would also be covered under the scope of this project.

The objectives of the project are:

- Show **stats** for search results.
- Optimize search filters by implementing **Elasticsearch buckets.**
- **JSON Encode** Elasticsearch queries
- **Documentation** of search(675)
- Add the number of hits for filters(526)
- Fix search bugs for certain keywords(1067)
- extend free search (996)
- Search filter problem.(531)
- search children entities (1035)

Implementation:

1)JSON Encode Elasticsearch queries

Approach:

1) We need to use the **json_encode () function**. The Function returns the **JSON representation** of the value passed.
2) We will have to **create arrays** of the string in current queries present in **ElasticSearchComponent.php** then by passing the arrays to the function json_encode () we can get our required output.
3) For e.g:
   The following is a **concatenation of string** which we will need to convert into arrays so that they are accessible by their indexes.

```
\"_source\": {
    \"includes\": [
        \"id\", \"adesignation\", \"collection\", \"accession_no\", \"museum_no\", \"written_in\",
    ]
```

   we will have key value pairs like {0: id, 1: adesignation}.

OUTCOMES:

- Decrease in syntax errors and eventually bugs
- Readability of queries will be better
- Updating and adding features for elasticsearch will be easy

View of the current code:

```
\"sort\": [
    {
        \"_score\": {
            \"order\": \"desc\"
        }
    }
],
\"size\": 10000
```

View After completing objective:

```
"sort": [
    {
        "_score": {
            "order": "desc"
        }
    }
],
"size": 10000
```

REFERENCE: [Php JSON Encode]

2) Optimize search filters by implementing Elasticsearch buckets

Approach:

1) Currently the filtering in our framework is set up in such a way that every time you apply a filter on the search page the elasticsearch queries are run again with specific filters.

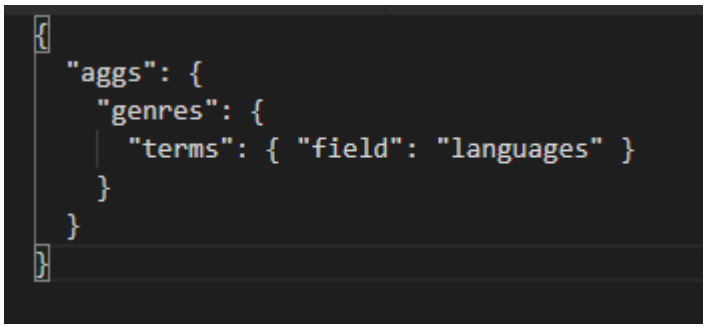
```
public function filterESQuery($filterData)
{
    /**
        $filterForSameField = '
            {
                \"match_phrase\":  {
                    \"materials\": \"limestone\"
                }
            }';

        $filterForCombinedInnerField = '        {
            "bool": {
                "should": [
                    {
                        "match_phrase":  {
                            "atype": "barrel"
                        }
                    },
                    {
                        "match_phrase":  {
                            "atype": "other (see object remarks)"
                        }
                    }
```

2) We should also use elasticsearch buckets because of Storing the filter categories as when you go to the second page of results, the filter categories would need to be calculated and re-calculating takes a long time

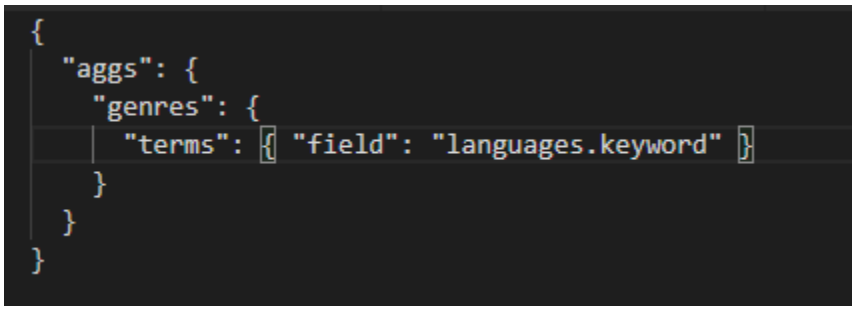
3) To resolve this we can use bucket aggregation provided by elasticsearch which helps by grouping a set of documents together and then searches them.

Implementation:

- We could write the following query for filtering all the documents corresponding to language



- It would return the documents bucketed separately for each language such as Sumerian, Akkadian.
- We would have to enhance the queries depending upon the indexing also as if languages also exist as an index then we would need to write the following query.



3)Add the number of hits for filters(526)

Approach:

- This is a relatively important feature from the user's point of view as the user would like to know how many results are present for each filter before applying them.
- It would give them a better understanding of the search result they are looking for.
- I have purposely kept this objective after the Buckets one since we would be changing the filtering system it would be better to solve the filter issues after implementing buckets first.

Implementation:

- After implementation of buckets We would get document_count for all the filters every time a keyword is searched.
- This count is basically the number of hits for every filter
- To display it we would simply have to pass it to the view file

4) Show stats for search results(911)

We want to give statistics all over the corpus on one of the pages and then on the other we want dynamic statistics over every search that is executed.

Approach:

- We will be using the d3.js library to create data visualizations.
- In d3 we have to provide data which will be then made as a document object. This document object can then be transformed into various data visualizations such as tables, bar graphs etc.
- We could also re-use the code which has been written previously for data visualization.

Implementation:

- We can re-use the previous individual functions for each chart to pass the data using a variable.
- Next what we can do is pass the data to the required functions so that it can be stored in respective variables
- for e.g -

  ```
  $barChartData = filteredSearchBarChart($searchResults)
  ```

- We will have to use functions for all types of charts for e.g- filteredSearchBarChart, filteredDonutChart

- We then need to convert the data using JSON-encode so that it can be passed onto the d3.js files.
- After transforming data into the format that is required for d3.js library, we would have to start working on using this data and designing various charts.
- Designing part includes putting in values for height,width etc.
- We would also be required to put in a color scheme which would best suit our requirements.
- The part regarding where we could place the button for the corresponding feature can be discussed with the design team later on.
- we could also re-use the code written by previous developers who implemented data visualizations.
- The code for this already exists as this feature has been previously added.
- We can try using the same code and then solve all the bugs that arise with it.
- We will have to then add the functionality which was missing last time such as the title issue for graphs.

5)Extend Free Search(996)

Approach:

- There exists comments in multiple entities hence if we want to include comments we would have to create joins using SQL.
- And then we can move onto **indexing** by using the **logstash.conf file.**
- And then we would modify elasticsearch queries to include comments field **in elasticsearchcomponent.php**
- For e.g

  we will have to add “comments” to the following list

```
    'seal_no',
    'museum_no',
    'id',
    'dates',
    'artifact_project',
    'inscription_project',
    'artifact_credit',
    'inscription_credit',
    'archive'
]
```

Outcomes:

- We will be able to search through the comments field using free search.

6) Fix search bugs for certain keywords (1067)

Approach:

- We will have to check if elasticsearch allows the given symbols to be searched.
- IF it does then try to fix the bug whatever it might be
- If it doesn't allow then figure out a way for it to be used in search.

Implementation:

- After debugging the search for "HS 145 (+)" we understand that taking the string is not an issue but elasticsearch cannot process any one of the symbols i.e "(", "+", ")".
- Debug the simple search query and see what is exactly happening to the string during search.
- While searching for the keyword " nam-sipa-zu " one of the cases is " - - ".
- Currently for composite numbers and for other numerical fields the queries are written in such a way that they will take integers and symbols(-) and ignore alphabet but for some reason it is also affecting other results.
- We need to figure out why elasticsearch queries are breaking as the input strings are taken properly.

Outcomes:

- Users will be able to search string which was previously giving errors.

7)search children entities (1035)

There exists a relation between various entities such as genre, language and materials is called parent and child. We want to update the current search queries such that whenever we search for an entity the result should also include the child, grand-child and also great-grand-child entities of the corresponding entity shall be included.

Implementation:

- Firstly, we would need to find a way to create a logical relation between the keyword that is searched and the parent id whose child should be shown in the results.
- We can access every child entity using it's parent id.
- To use the parent id field we will have to index it inside advanced_artifacts.
- Then we can proceed to write queries such that we could add the required entity to the result.
- There is a special query in elasticsearch which allows us to include the children on the basis of parent id.
- Reference: parent id query
- To implement this we will have to create joins between parent and children.

Outcomes:

- We would be able to get the children, grand-children and other related entities from parent id using both free search and advance search.

8) **Documentation of search(675)**

Approach:

- We need two types of documents i.e User and Developer. Both of these documents will contain contents respective of their audience.
- In user one we should describe how the user should use the search options in an effective way.
- Whereas, in the developer one we would need to tell them about the workflow of the search setup.
- We will have to mention about the various new functions which were created and also about the pattern of elasticsearch queries which are currently written so that it would be easy for newcomers to understand.
- We would also need to give a description of fuzzy search, highlight Inscriptions and search settings.

Outcomes:

- A proper document with all information regarding search would be established.
- It could be the go-to for everyone who wants to understand how the search setup works in the framework.
- Also a user version of document will be created for the users

9) Search Filter Problem(531)

Approach:

- Currently when two or more filters are being applied we get the result as an union of all of them.
- Check why union is being returned.
- then modify current elasticsearch queries to return the intersection.
- Test the queries for the cases which were giving an issue before.
- check if the changes made in the backend are being reflected or not on the search results page.

Outcomes:

- The filters would provide an intersection on the results page instead of an union.

Timeline:

Phase 0: Pre-community bonding period **(Present - 20th May)**

- Work on current issues present in the CDLI framework.
- Research more about functionalities which will be implemented during the coding period.
- Explore the framework and read documentation related to objectives

Phase 1: Community bonding period **(20th May - 13th June)**

- Discuss with the mentor about the implementation in detail.
- Document the workflow for better understanding of the implementation approach.
- Create issues for the objectives on gitlab.

Phase 2: Coding phase 1 **(13th June - 25th July)**

week	Tasks/Deliverables	Output
Week 1 and week 2 (13th June-26th June)	• **Optimize search filters by implementing Elasticsearch buckets** • Extensively Read through Documentation for terms aggregation Ref: Bucket elasticsearch • Initially try **terms aggregations queries** for few filters and run test cases on it such as for **languages** which are mentioned above. • Implement everything that is tested into the local framework and then add the bucket feature to the framework.	• After Implementation is complete We could also see the number of hits for all filter • Filtering would become faster as every time a search is done we would already have documents with filtered data ready • Another benefit is that creation of documents is done dynamically hence no need of querying it further.

Week 3 and week 4 (27th June - 10th July)	● **Show number of hits in the parenthesis.** ● we would have already implied buckets in the previous week so we should just have to return the doc_count for all the filters. ● **Search Filter Problem** ● check why union of all the applied filters is returned. ● Test out the cases mentioned in the issue. ● Modify elasticsearch queries to return an intersection.	● Users will be able see number of hits in the parenthesis before applying a filter ● The filters would now return the intersection of all the applied filters.

Week 5 (11th July- 17th July)	● **Search breaks when searching for a certain string** ● Understand how the keyword is being modified for search in numerical fields. ● **search children entities** ● Find a way to get parent id from the keyword that is searched. ● We would have to create joins while indexing. ● Modify queries to give results which include child, grandchild of an entity.	● child, grandchild would be included in the search result for a parent. ● Search would be fixed for the strings which were breaking earlier.
week 6 (18th July - 25th July)	● Buffer week ● Have a chat with the design team regarding the statistics feature as it would be our next task. ● To complete all the remaining tasks from the above objectives ● Fixing the bugs generated ● Prepare all necessary documentation for phase 1 evaluation.	● Report for phase-1 ● Code submission for phase-1

Phase-3: Work Period **(26th July - 4th September)**

Week 7 and week 8 (26th July- 7th August)	● **Show stats for search results** ● Change the title of the graphs in relation to the data they represent, not the type of chart. ● Re-use the code written by previous developers ● Resolve the bugs that arise after re-using. ● Add the changes which were not present in the last statistics feature. ● Test the changes and functionality ● Take a review from the team members regarding visuals and other designing parts. Implement the suggestions.	● Static Data visualizations to represent an overview of the corpus ● Dynamic data visualizations after filters have been applied so that the user gets a better understanding of how many different results are present inside each filter.

Week 9 and week 10 (8th August- 21st August)	● **Encode Elasticsearch queries using JSON by replacing the current string concatenation.** ● Test the function for a single elasticsearch query such as simple search or any filtering query for that instance. ● Create arrays for all the strings present in queries and pass them to the function to get desired output. ● **Extend free search.** ● Write SQL queries to create joins for the comments. ● Then index them and finally add it to the list as mentioned above.	● Developers will be able to write elasticsearch queries more systematically ● Cleaner code base for beginners to read and understand. ● Free search would now include Comments field.
Week 11 (22nd August - 28th August)	● Complete Documentation ● Take suggestions as much as possible to create a more effective document	● Two types of documents: User and Developer would be created with respect to search.

Week 12 (29th August - 4th September)	• Buffer week • Complete all the remaining objectives • Prepare the final project report • Solve any bugs which were generated along.	• Submit the code • Submit the final project report.

Post GSOC :

- Reach out to my college mates to introduce them to the CDLI community and encourage them to make contributions to it.
- Look out for various other updations or installations for the framework which could be implemented.

Contribution To CDLI:

Type	Link	Description
Improvement PR	PR-1046	Filtered free and advanced search to eliminate retired artifacts.
Improvement PR	PR-1069	Highlights inscription using free search