# GSOC 2021 Proposal

## Cuneiform Digital Library Initiative

***Discovery search and advanced search features***

| | | |
|---|---|---|
| **Name** | : | Yashraj Desai |
| **Username** | : | yashrajdesai30 |
| **Email** | : | yashrajdesai30@gmail.com |
| **Location** | : | Mumbai, India |
| **Time Zone** | : | UTC +5:30 |

## ***Project Description :***

The project mainly focuses on **enhancing** the **search and advanced search features** in the **CDLI framework** .

The **objectives** of the project are :

- Implementing **Fuzzy** queries.
- Add the **"IDs"** and **"Keywords"** search field to both **Simple** & **Advanced** Search.
- Enable search **inscription** with **sign value** permutation.
- Highlight **transliteration sign values** in **ATF** display.
- On **add** or **edit**, save a **sanitised** version of an **inscription** as a list of **sign names**.
- Making **search input** flexible by allowing users to search using both **UTF-8** and **ASCII** characters.
- Filter search result by **RTI**, **Image**, **Transliterations**, **3D data**.
- Port request to **ElasticSearch** from **cURL** to **HttpClient**.
- Port **elasticsearch** to the **cakephp** elasticsearch plugin.

## ***Implementation :***

## 1) Implementing Fuzzy queries

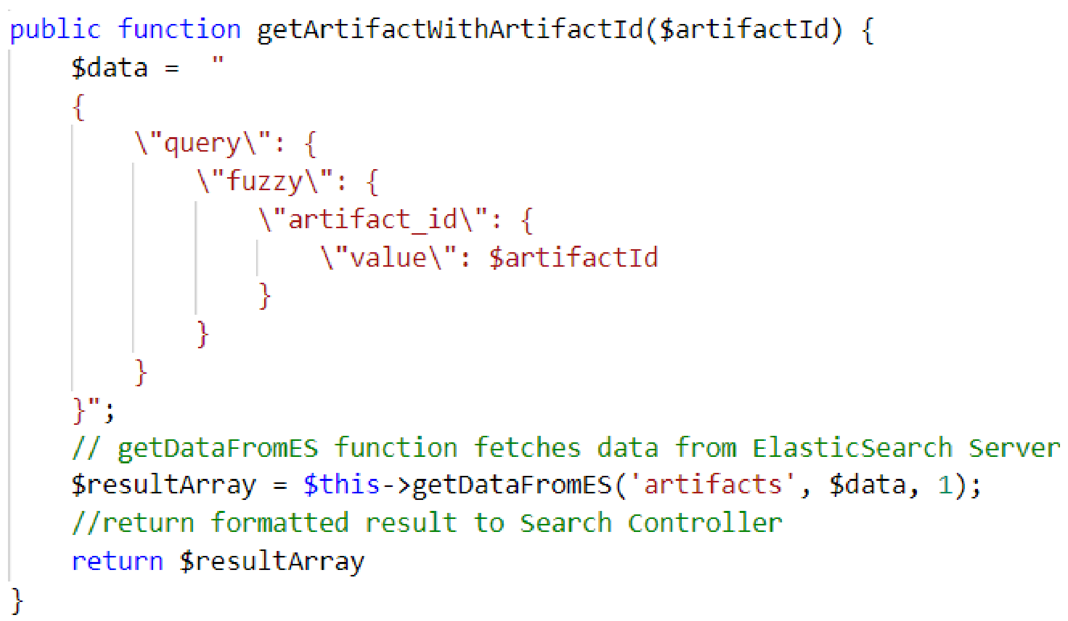
In this task, we will implement fuzzy queries for the entire search.
Example: If a user searches for **OIM 00042** or **OI 42** it must return search results for **OIM 00042**.
This can be implemented using the Elasticsearch **Fuzzy** query.

**Outcomes :**

Fuzziness will be implemented for every search result within a respectable response time.

**Approach :**

1. We'll need to modify the search queries in the **ElasticSearchComponent.php** file.



```
public function getArtifactWithArtifactId($artifactId) {
    $data =  "
    {
        \"query\": {
            \"fuzzy\": {
                \"artifact_id\": {
                    \"value\": $artifactId
                }
            }
        }
    }";
    // getDataFromES function fetches data from ElasticSearch Server
    $resultArray = $this->getDataFromES('artifacts', $data, 1);
    //return formatted result to Search Controller
    return $resultArray
}
```

2. We will design various **user cases** and **test** against various **fuzziness configurations** to make sure that the **response time** is respectable.
3. If some queries requires processing of input, it can be converted to Regex and search can be made with this input.

**References** : Elasticsearch Fuzzy query, Regex in ElasticSearch

## 2) Add the "IDs" and “keyword” search field to both Simple and Advanced Search

**Related Issues** : #298 (WIP: Search fields) , #314 (Simple Search for ID and Keywords)

In this task, we will be adding **IDs** and **keyword** search fields to both **simple** and **advanced search** with reference to **entries** mentioned in the outcomes given below .

**Outcomes** :

1. If a user searches for an **ID/ numbers artifacts** which matches the following entries :
    - publications.bibtexkey
    - publications.designation concatenated with artifacts_publications.exact_refrence
    - publications.designation
    - artifacts.designation
    - artifacts.excavation_no
    - artifacts.composite_no
    - artifacts.museum_no
    - artifacts.seal_no
    - artifacts.excavation_no
    - artifacts.id

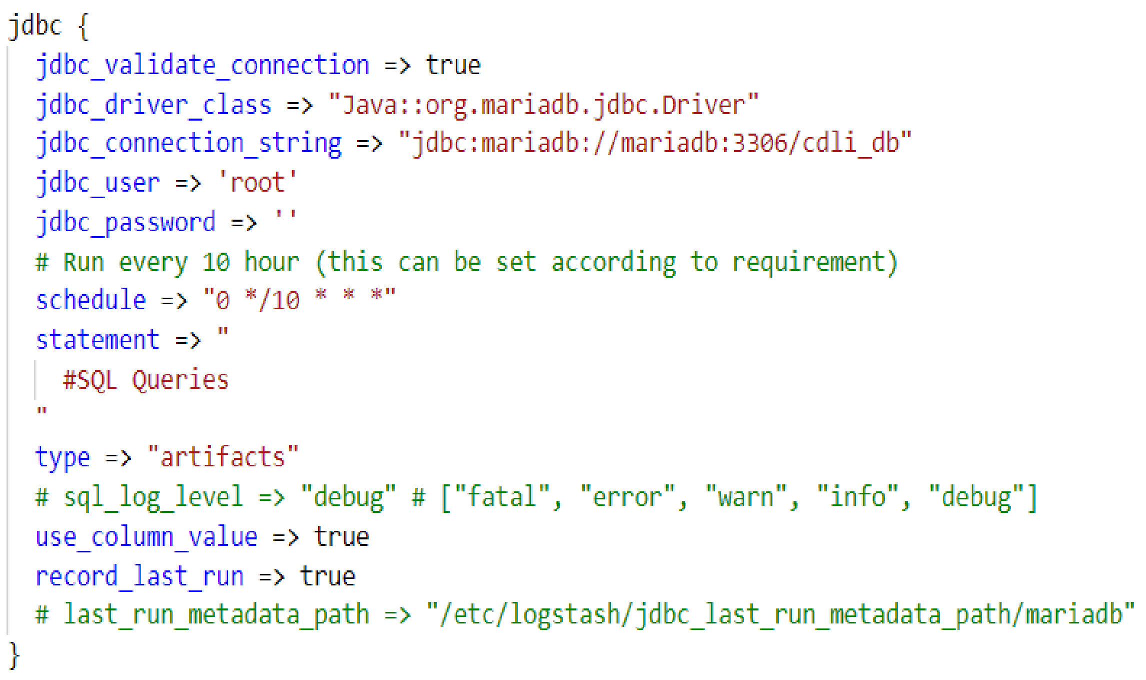
   it will return the relevant artifact entries.

2. If a user searches for specific **keywords** with reference to :
    - Publications
    - All main entities (Eg. Collections, Materials, Periods,etc.)
    - Id’s and numbers
    - All artifacts text & varchar fields

   It will return respective results .

**<u>Approach :</u>**

1. Updating current indices using logstash by adding optimised **SQL** queries in the **logstash.conf** file . The **new columns** which need to be added are from the **inscriptions table** such as **transliteration_sign_names** , **transliteration**, **transliteration_clean** ,etc. Also an entire **new table** of **artifacts_seals** need to be added.



```
jdbc {
  jdbc_validate_connection => true
  jdbc_driver_class => "Java::org.mariadb.jdbc.Driver"
  jdbc_connection_string => "jdbc:mariadb://mariadb:3306/cdli_db"
  jdbc_user => 'root'
  jdbc_password => ''
  # Run every 10 hour (this can be set according to requirement)
  schedule => "0 */10 * * *"
  statement => "
    #SQL Queries
  "
  type => "artifacts"
  # sql_log_level => "debug" # ["fatal", "error", "warn", "info", "debug"]
  use_column_value => true
  record_last_run => true
  # last_run_metadata_path => "/etc/logstash/jdbc_last_run_metadata_path/mariadb"
}
```

2. To search from elasticsearch **indices** we need to modify our current elasticsearch queries in the respective methods of the **ElasticSearchComponent.php** file .
3. Finally , we will test various **user cases** to improve the quality and performance of search results.

## 3) Enable search inscription with sign value permutation :

In this task, we need to **convert** the **search input sign values** into **sign names** using the **jtf-signlist**, search with these **sign names** and return Inscriptions with matching **sign values** from our database.

**Outcomes :**

When a user will **enable** the **search inscription** with **sign value permutation** feature and search for a **text/regex** in the Inscriptions then all inscriptions with all sign values matching the sign names of the sign values in the query must be returned as a search result.

**Approach :**

1. Users will be provided with a **switch** to enable this feature in search .
2. When a user will search with **sign reading permutation** on, the **sign values input** will be **converted** into **sign names** using **jtf-signlist**.
3. These **sign names** will be matched with the ones stored in the database's **transliteration_sign_names** field and all **matching** results will be returned .
4. **Indices** are already **created** for all these fields in the database .
5. Once required changes are made in the database the **logstash** container should be executed to **update** all indices .
6. We'll **modify search queries** in the ElasticSearchComponent.php file which will match the **sign_names** field and return Inscriptions with matching **sign values** .
7. We'll test the implementation against various **sign values** to make sure that the search yields relevant results.

## 4) Highlight transliteration sign values in ATF display

**Related issue:** #347

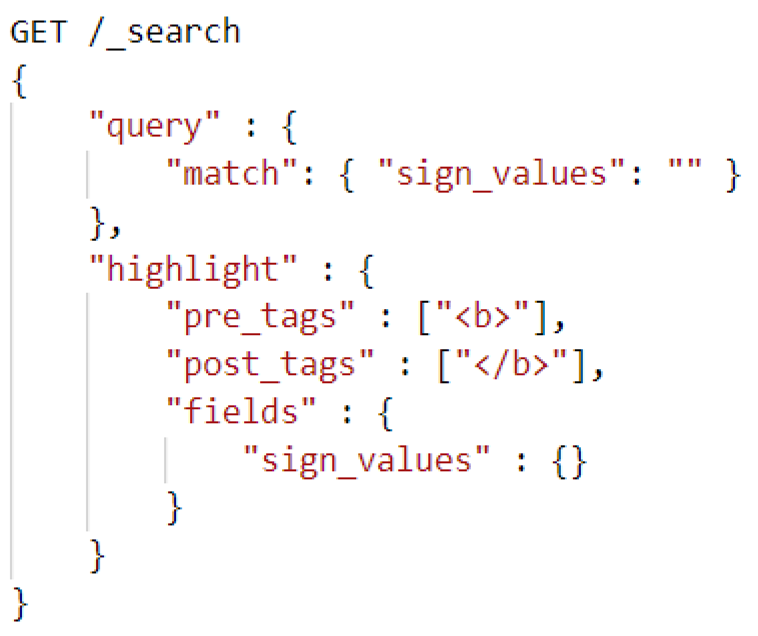
In this task, the search results generated by matching **transliteration sign values** with **sign names** will be **highlighted** in the compact search results page. This can be done by modifying our elasticsearch queries by adding the **highlight** field in it.

**Outcomes:**

When a user will search for sign values, the matching transliteration sign values will be highlighted in the compact search results page.

**Approach:**

The elasticsearch queries in **ElasticSearchComponent.php** file will be modified by adding the **highlight** field in it.



```
GET /_search
{
    "query" : {
        "match": { "sign_values": "" }
    },
    "highlight" : {
        "pre_tags" : ["<b>"],
        "post_tags" : ["</b>"],
        "fields" : {
            "sign_values" : {}
        }
    }
}
```

**References:** Highlight data

## 5) On add or edit, save a sanitized version of an inscription as a list of sign names

<u>**Related issue**</u> : [#120](#) (Add processing for separating comment, structure and translation from transliteration) , [#259](#) (Add and edit inscriptions)

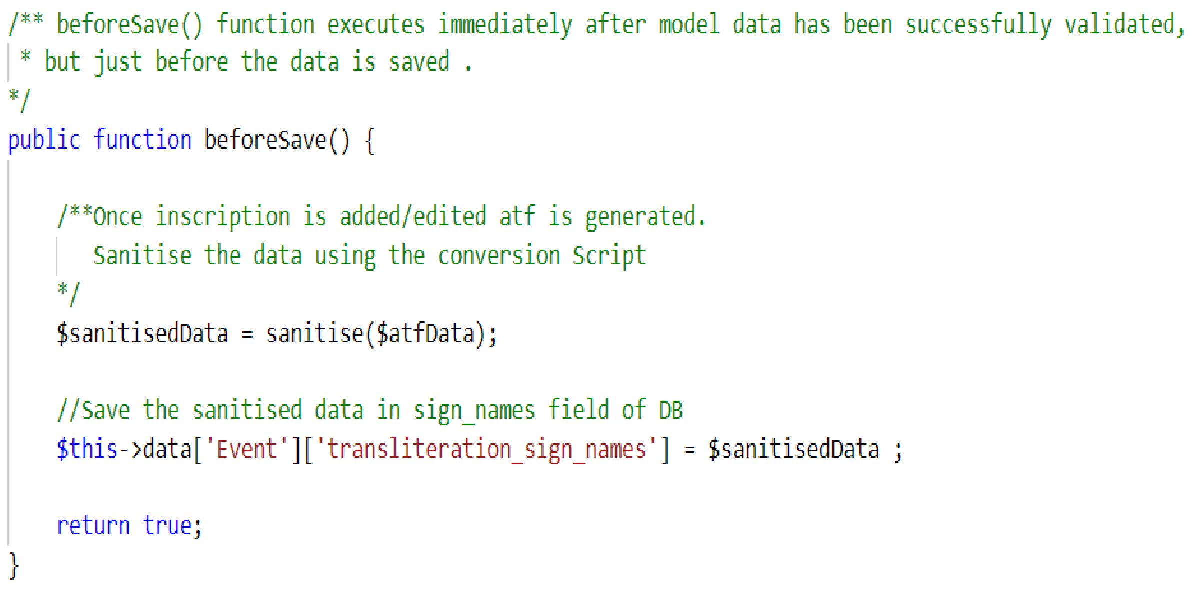
When an **inscription** is **added/edited** , corresponding **atf** is generated after processing it . In this objective , we will be **sanitising** this **atf** field with the help of a sanitisation **script** and **store** the **sign names** in the transliteration_sign_names field of our **database** .This will aid our search when someone searches for **sign values**.

<u>**Outcomes**</u> **:**

When an inscription is **added/edited,** the **sanitised** version of an inscription as a list of sign names **without word boundaries** will be saved.

<u>**Approach**</u> **:**

1. When an inscription is added or edited, the **atf field** is generated which contains all the data of **inscription** .
2. The **atf** field should be **sanitised** by **removing** all **word boundaries** .
3. **Sanitised data** should be saved in the **transliteration_sign_names** field of the database.In the **inscription model** file we will need to add the **callback function** which will be invoked once inscription is **added/edited** .



```
/** beforeSave() function executes immediately after model data has been successfully validated,
 * but just before the data is saved .
*/
public function beforeSave() {

    /**Once inscription is added/edited atf is generated.
       Sanitise the data using the conversion Script
    */
    $sanitisedData = sanitise($atfData);

    //Save the sanitised data in sign_names field of DB
    $this->data['Event']['transliteration_sign_names'] = $sanitisedData ;

    return true;
}
```

## 6) Input flexibility enhancement

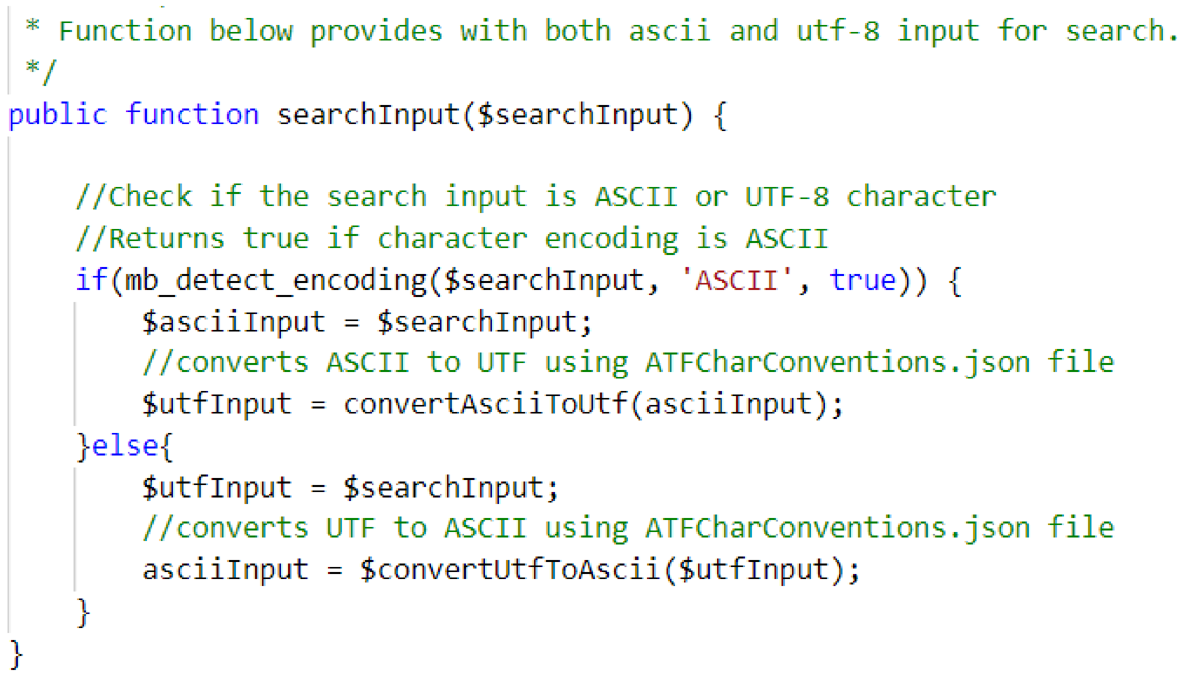
Users will be able to search with both **UTF-8** and **ASCII** characters which will improve the **flexibility** of our **search inputs** . We'll need to convert the input accordingly using our **conversion tables** and return search results.

**Outcomes :**

1. Users will be able to search using **UTF-8 characters** in the **entities** which store data **only** as **ASCII characters** .
2. Users can search using both **ASCII** and **UTF-8 characters** for **entities** which use **both UTF-8** and **ASCII** characters .

**Approach :**

1. List out the entities which store data **only** as **ASCII** characters and the ones which have **either ASCII** or **UTF-8** .
2. We will need to include the **searchInput** function in the **searchController** which will provide us with both **ASCII** and **UTF-8 input** for search .



```
 * Function below provides with both ascii and utf-8 input for search.
 */
public function searchInput($searchInput) {

    //Check if the search input is ASCII or UTF-8 character
    //Returns true if character encoding is ASCII
    if(mb_detect_encoding($searchInput, 'ASCII', true)) {
        $asciiInput = $searchInput;
        //converts ASCII to UTF using ATFCharConventions.json file
        $utfInput = convertAsciiToUtf(asciiInput);
    }else{
        $utfInput = $searchInput;
        //converts UTF to ASCII using ATFCharConventions.json file
        asciiInput = $convertUtfToAscii($utfInput);
    }
}
```

3. If the user searches in the tables which store values **only** in **ASCII** format then we will **search** with **ASCII** input, else we'll search with **both ASCII** and **UTF-8** for other tables.
4. Modify search queries in **ElasticSearchComponent.php** files according to the **condition** in point **3**.

## 7) Filter search results by RTI Image , Transliterations , 3D data

**Related issue** : #136 (RTI Filter)

If the user applies the filters such as **RTI Images,Transliterations,3D Data** , it will be queried in their respective tables and the results will be returned according to status fields for respective queries.

**Outcomes :**

1. When the user will apply **RTI Images** as a filter, search results which have associated **RTI Images** will be shown to the user.
2. When the user will apply **Transliterations** or **3D Data** as a filter, associated **Inscriptions** with matching **Transliterations** and **3D Data** will be shown to the user .

**Dependency**: Some of the above filter's data is yet to be processed . Hence, we can work with **test data** (except for **Images** Table) for implementing search functionality .

**Approach :**

1. Till the time data is processed and ready, we have to work with **test** data.
2. We'll have to create tables which have **status fields** the **same** as the **given filters (RTI Image , Transliterations ,3D data)**.
3. **Indexing** the new fields which we have used for search filters .
4. **Modifying** our **search queries** according to the new filters in the ElasticSearchComponent.php file .
5. Make changes in respective **view** files to display filter results .
6. Finally when the data is processed and ready , integrate it with our **database** by replacing the test data .

## 8) Port request to ElasticSearch from cURL to HttpClient

Current implementation of **elasticsearch** in the **framework** is using **cURL** . In this task we would be replacing it with HttpClient using **CakePHP HTTP Client**.
**CakePHP** includes a basic but powerful **HTTP client** which can be used for making requests to **ElasticSearch** . It is a great way to communicate with **remote APIs**.

**Outcomes :**

Current implementation of **cURL** for **elasticsearch** would be **replaced** with **CakePHP HTTP Client** .

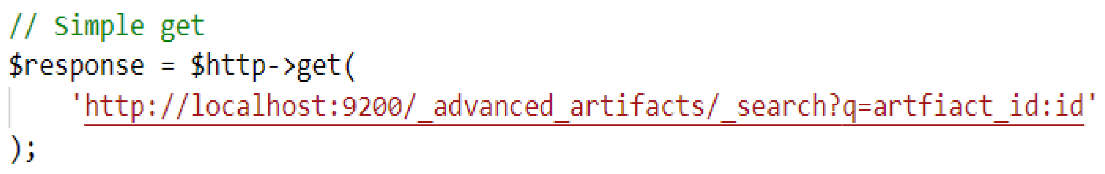
**Approach :**

1. Initialise the **http** object.

```php
use Cake\Http\Client;
$http = new Client();
```

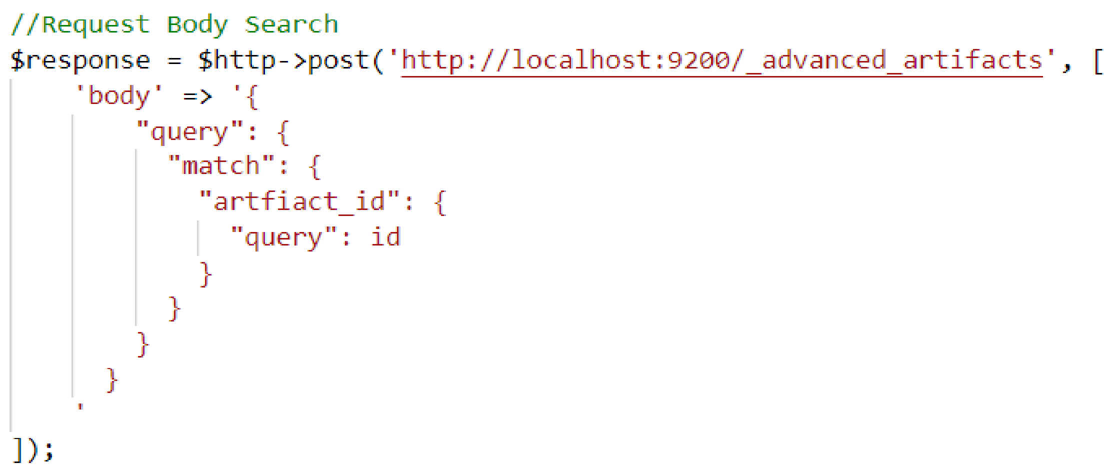
2. To **search indices** we can either perform **GET** or **POST** request :

- **GET** request :



```php
// Simple get
$response = $http->get(
    'http://localhost:9200/_advanced_artifacts/_search?q=artfiact_id:id'
);
```

- **POST** request using **request body search** :



```php
//Request Body Search
$response = $http->post('http://localhost:9200/_advanced_artifacts', [
    'body' => '{
        "query": {
          "match": {
            "artfiact_id": {
              "query": id
            }
          }
        }
      }
    '
]);
```

## 9) Port elasticsearch to the cakephp elasticsearch plugin :

**Related Issue** : #460 (Implementation of cakephp elasticsearch plugin)

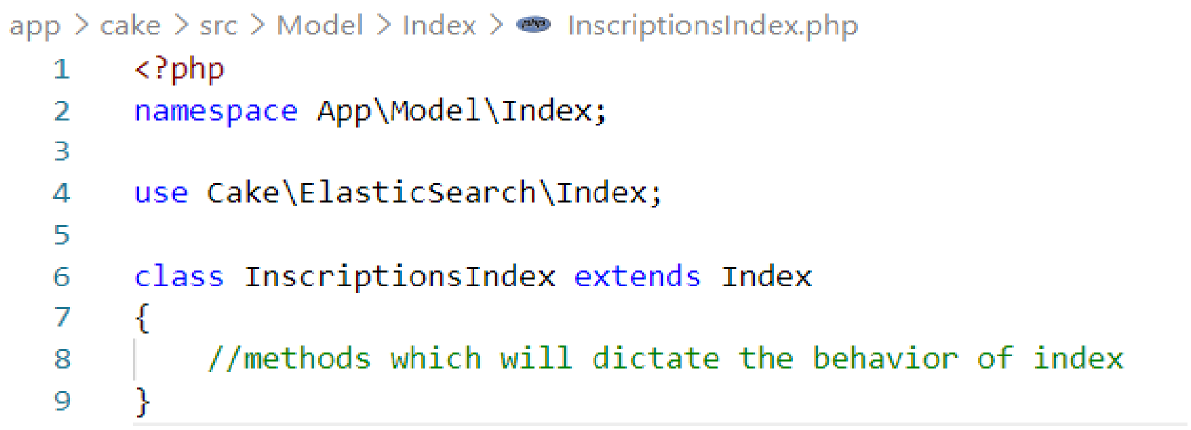
This task focuses on implementing **cakephp elasticsearch plugin** which would provide **indexing** , **searching** and performing **complex queries** functionalities . Also we need to update the current implementation of **pagination** using **\Cake\Datasource\Paginator.**

**Outcomes**:

1. **Indexing** and performing **queries** using **cakephp elasticsearch** plugin.
2. Updated pagination with **\Cake\Datasource\Paginator .**

**Approach:**

1. **Installing** and **configuring** cakephp elasticsearch plugin. (**Already completed**).
2. Create **index objects** which are similar to **table-like** classes in elasticsearch.The **ElasticSearch plugin** makes it easier to interact with an **elasticsearch index** and provides an interface similar to the **/orm**.



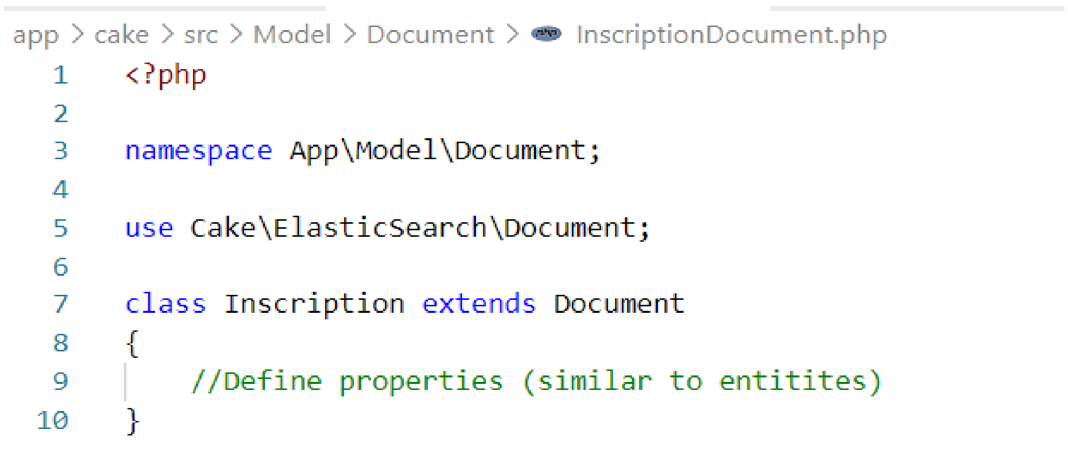
app > cake > src > Model > Index > InscriptionsIndex.php

```php
<?php
namespace App\Model\Index;

use Cake\ElasticSearch\Index;

class InscriptionsIndex extends Index
{
    //methods which will dictate the behavior of index
}
```

3. We then have to define **Document** which will then **index** Documents. The **interface** and **functionality** provided by Documents are the same as those in **Entities** .



app > cake > src > Model > Document > InscriptionDocument.php

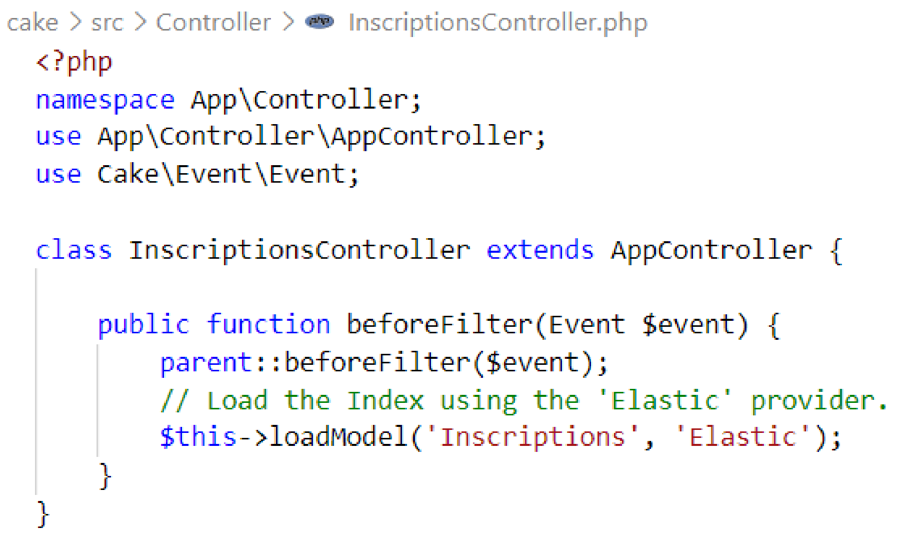
```php
<?php

namespace App\Model\Document;

use Cake\ElasticSearch\Document;

class Inscription extends Document
{
    //Define properties (similar to entitites)
}
```

4. Now, we can use the indexes in our **controllers** and write queries .

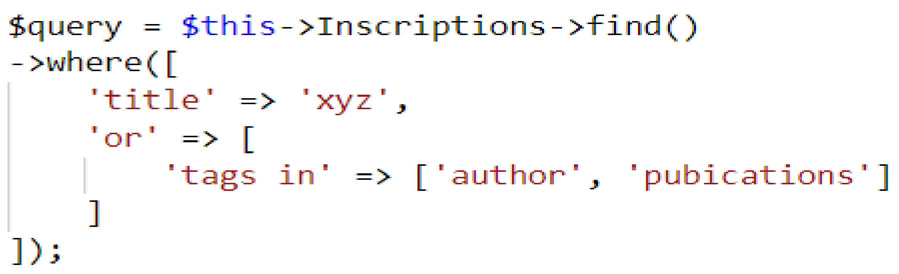

cake > src > Controller > InscriptionsController.php

```php
<?php
namespace App\Controller;
use App\Controller\AppController;
use Cake\Event\Event;

class InscriptionsController extends AppController {

    public function beforeFilter(Event $event) {
        parent::beforeFilter($event);
        // Load the Index using the 'Elastic' provider.
        $this->loadModel('Inscriptions', 'Elastic');
    }
}
```

5. The **ElasticSearch plugin** provides a **query builder** that allows us to build complex search queries .


```php
$query = $this->Inscriptions->find()
->where([
    'title' => 'xyz',
    'or' => [
        'tags in' => ['author', 'pubications']
    ]
]);
```

6. On Successful implementation of the plugin, pagination implementation can be updated by using **\Cake\Datasource\Paginator**. This class is used to handle **automatic model data pagination**.

**References :** [https://book.cakephp.org/elasticsearch/3/en/index.html](https://book.cakephp.org/elasticsearch/3/en/index.html)

## ***Timeline :***

### Phase 0: Pre-community bonding period (Present - 17th May)

- Work on current issues present in the CDLI framework .
- Research more about elasticsearch and cakephp .
- Explore the framework and post new issues encountered

### Phase 1: Community bonding period (17th May - 7th June)

- Discuss with the mentor about the implementation in detail.
- Document the workflow for better understanding of the implementation approach.
- Research about cakephp elasticsearch plugin and test basic queries .

### Phase 2: Coding Phase 1 (7th June - 16th July)

| Week # | Task / Objectives Covered | Deliverables |
|---|---|---|
| Week 1 and Week 2 (7th June - 20th June) | ● **Implementation of Fuzzy queries and Addition of the "IDs" and “Keywords” search field to both Simple & Advanced Search**. (Objectives **1** & **2**)<br>● Modify search queries which should include fuzziness in search .<br>● Updating current indices with the fields required for ID’s and keywords .<br>● Testing search results with different fuzzy queries .<br>● Documentation of implementation . | ● Fuzzy queries would also yield search results.<br>● Users will be able to search for specific keywords,Id/Numbers artifacts.<br>● Documentation of added search features . |
| Week 3 and Week 4 (21st June - 4th July) | ● **Enabling search Inscription with sign value permutation and highlighting the results.** (Objectives **3** & **4**)<br>● Modify elasticsearch queries which will match sign names and will | ● When a user will enable this search feature and search for sign values, all inscriptions with matching sign values will be returned.<br>● Search results will be highlighted with the |

| | | |
|---|---|---|
| | return inscriptions with matching sign values.<br>● Highlighting matching sign values in atf display.<br>● Testing and documentation of the implemented logic . | matching sign values in atf display.<br>● Documentation of the implementation . |
| Week 5<br>(5th July - 16th July) | ● Buffer week for completing all the remaining tasks in phase 1.<br>● Fix all the issues developed in phase 1.<br>● Blog post for phase 1. | ● Complete all the documentation of phase 1.<br>● Report of phase 1.<br>● Submit phase 1 evaluation . |

## Phase 3: Coding Phase 2 (17th July - 16th August)

| Week # | Task | Deliverables |
|---|---|---|
| Week 1<br>(17th July - 23rd July) | ● **On add or edit save a sanitized version of inscription as a list of sign names.** (Objective **5**)<br>● Prepare a sanitisation script which will sanitise atf field by removing all word boundaries.<br>● Adding callback functions in model classes which will invoke the sanitisation script and will save the sign_names in transilteration_sign_names field of the database every time the artifact is added/edited. | ● When an artifact is added/edited the sanitised atf data will be stored in the transilteration_sign_names field of the database .<br>● Documentation and thorough testing of the sanitisation script . |
| Week 2<br>(24th July - 30th July) | ● **Input flexibility enhancements** (Objective **6**).<br>● List out tables which use only ASCII data for storing .<br>● Write conversion functions in searchController and provide UTF-8 and ASCII inputs to ES queries .<br>● Modify ES queries to search for respective inputs.<br>● Testing the implementation with different search inputs . | ● Users will have the flexibility to search with both UTF-8 and ASCII characters .<br>● Documentation of the implementation. |

| | | |
|---|---|---|
| Week 3<br>(31st July - 6th August) | • **Filter search results by RTI Image, Transliterations , 3D Data** (Objective **7**).<br>• Index new fields which were added to the database for implementing search filters of RTI , Image, Transliteration.<br>• Modify our search queries to match our filters .<br>• Make changes in our view files to display filter results . | • Users can apply filters such as RTI Image, Transliterations , 3D Data .<br>• Documentation of implementation. |
| Week 4<br>(7th August - August 13th) | • **Port request to Elasticsearch from cURL to HttpClient and implementation of CakePHP elasticsearch plugin.** (Objectives **8** & **9**)<br>• Creating index objects similar to table-like classes in orm .<br>• Creating documents similar to entities in orm which can be used for performing elasticsearch queries.<br>• Replacing cURL based implementation with HTTP Client.<br>• Testing by performing various search queries on indices.<br>• Updating pagination on search results using \Cake\Datasource\Paginator. | • Replaced cURL implementation with HTTP Client .<br>• Updated Pagination on search results .<br>• Documenting cakephp elastic search plugin. |
| Week 5<br>(14th August - 23rd August) | • Buffer week for pending deliverables.<br>• Prepare the final project report.<br>• Submit the final code. | • Final submission of report, code and blog post. |

## ***Future plans (Post GSOC) :***

- Contribute to the CDLI framework and be an active member in the community .
- Add community requested features and fix bugs .
- Reach out to my college mates to introduce them to the CDLI community and encourage them to make contributions to it.

## ***Contribution to CDLI framework :***

### PR:

| Type | Link | Description |
|---|---|---|
| Enhancement PR | PR #225 | Added corpus search item . |

### Issue:

| Type | Link | Description |
|---|---|---|
| Plugin Implementation | Issue #460 | Installed and configured elasticsearch plugin. |